ΠΛΗΡΟΦΟΡΙΚΗ

ΠΛΗ40 – ΠΡΑΚΤΙΚΗ ΕΞΑΣΚΗΣΗ ΣΕ ΘΕΜΑΤΑ ΛΟΓΙΣΜΙΚΟΥ

Πτυχιακή Εργασία

Ανάπτυξη πλήρους εφαρμογής για την αποδοτική επίλυση του προβλήματος Nurse rostering (εύρεση βέλτιστου ωρολογίου προγράμματος εργασίας σε νοσοκομειακά ιδρύματα)

Αριστείδης Παυλίδης

Επιβλέπων καθηγητής: Δρ. Γρηγόριος Μπεληγιάννης

Η παρούσα εργασία αποτελεί πνευματική ιδιοκτησία του φοιτητή («συγγραφέας/δημιουργός») που την εκπόνησε. Στο πλαίσιο της πολιτικής ανοικτής πρόσβασης ο συγγραφέας/δημιουργός εκχωρεί στο ΕΑΠ, μη αποκλειστική άδεια χρήσης του δικαιώματος αναπαραγωγής, προσαρμογής, δημόσιου δανεισμού, παρουσίασης στο κοινό και ψηφιακής διάχυσής τους διεθνώς, σε ηλεκτρονική μορφή και σε οποιοδήποτε μέσο, για διδακτικούς και ερευνητικούς σκοπούς, άνευ ανταλλάγματος και για όλο το χρόνο διάρκειας των δικαιωμάτων πνευματικής ιδιοκτησίας. Η ανοικτή πρόσβαση στο πλήρες κείμενο για μελέτη και ανάγνωση δεν σημαίνει καθ' οιονδήποτε τρόπο παραχώρηση δικαιωμάτων διανοητικής ιδιοκτησίας του συγγραφέα/δημιουργού ούτε επιτρέπει την αναπαραγωγή, αναδημοσίευση, αντιγραφή, αποθήκευση, πώληση, εμπορική χρήση, μετάδοση, διανομή, έκδοση, εκτέλεση, «μεταφόρτωση» (downloading), «ανάρτηση» (uploading), μετάφραση, τροποποίηση με οποιονδήποτε τρόπο, τμηματικά ή περιληπτικά της εργασίας, χωρίς τη ρητή προηγούμενη έγγραφη συναίνεση του συγγραφέα/δημιουργού. Ο συγγραφέας/δημιουργός διατηρεί το σύνολο των ηθικών και περιουσιακών του δικαιωμάτων.

ΕΛΛΗΝΙΚΟ ΑΝΟΙΚΤΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

Ανάπτυξη πλήρους εφαρμογής για την αποδοτική επίλυση του προβλήματος Nurse rostering (εύρεση βέλτιστου ωρολογίου προγράμματος εργασίας σε νοσοκομειακά ιδρύματα)

Αριστείδης Παυλίδης

Επιτροπή Επίβλεψης Πτυχιακής / Διπλωματικής Εργασίας

Επιβλέπων Καθηγητής:

Δρ. Γρηγόριος Μπεληγιάννης

Αν. Καθηγητής

Πρόεδρος Τμήματος Διοίκησης Επιχειρήσεων Αγροτικών

Προϊόντων και Τροφίμων

Πανεπιστήμιο Πατρών

Συν-Επιβλέπων Καθηγητής:

Δρ. Ευστράτιος Γεωργόπουλος

Τακτικός Καθηγητής Πρώτης Βαθμίδας

Αντιπρύτανης Οικονομικών, Προγραμματισμού και Ανάπτυξης

Διευθυντής του Εργαστηρίου Ανάλυσης Δεδομένων και Μοντελοποίησης

Συν-Επιβλέπων Καθηγητής:

Σπυρίδων Λυκοθανάσης

Διευθυντής Εργαστηρίου Αναγνώρισης Προτύπων

Πάτρα, Ιούνιος 2019

ΕΥΧΑΡΙΣΤΙΕΣ

Θα ήθελα να ευχαριστήσω τους γονείς μου, Παυλίδη Γεώργιο και Παυλίδου Ελευθερία, για τη υποστήριξη που μου παρείχαν στη διάρκεια της ακαδημαϊκής μου πορείας. Επίσης, θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου κύριο Γρηγόριο Μπεληγιάννη για την ανάθεση του θέματος και την εμπιστοσύνη που έδειξε στο πρόσωπό μου, καθώς επίσης και για την ελευθερία και υποστήριξη που μου παρείχε κατά την εκπόνηση της πτυχιακής εργασίας.

Αφιέρωση

Αφιερώνεται στη σύζυγό μου Δήμητρα
και στον υιό μου Γεώργιο – Χρήστο.

ΠΕΡΙΛΗΨΗ

Η παρούσα πτυχιακή εργασία, πραγματεύεται τη δημιουργία μιας πλήρους εφαρμογής σε γραφικό περιβάλλον, η οποία θα επιλύει το πρόβλημα του Nurse Rostering, δηλαδή αυτό της κατάρτισης ωρολογίου προγράμματος εργασίας νοσοκομειακών ιδρυμάτων. Το πρόβλημα του Nurse Rostering χαρακτηρίζεται ως πρόβλημα βελτιστοποίησης, αφού αναζητείται η βέλτιστη ή η σχεδόν βέλτιστη λύση μέσα από μια πλειάδα άλλων λύσεων ικανοποιώντας παράλληλα τους όποιους περιορισμούς έχουν τεθεί και εντάσσεται στην κατηγορία των NP – Πλήρων προβλημάτων. Για την επίλυση των προβλημάτων αυτών, χρησιμοποιούνται ευρετικοί και μεταευρετικοί αλγόριθμοι. Οι αλγόριθμοι αυτοί εφαρμόζονται κυρίως σε πολύπλοκα προβλήματα βελτιστοποίησης και σε περιπτώσεις όπου άλλες μέθοδοι βελτιστοποίησης είτε έχουν αποτύχει τελείως είτε έχουν αποτύχει να είναι αποτελεσματικές. Σε αυτή την εφαρμογή, χρησιμοποιείται ο αλγόριθμος αναζήτησης μεταβλητής γειτονιάς που αναπτύχθηκε από τους κ.κ. Μπεληγιάννη Γρηγόριο, Τασόπουλο Ιωάννη και Σώλο Ιωάννη, στην εκτελέσιμη μορφή του.

Η εφαρμογή, κατά την υλοποίησή της, πέρασε όλα τα στάδια της σχεδίασης από τα διαγράμματα μέχρι τη συγγραφή του κώδικα και την αποσφαλμάτωσή της. Για τη σχεδίαση των διαγραμμάτων περιπτώσεων χρήσης και των διαγραμμάτων ακολουθίας χρησιμοποιήθηκε το λογισμικό σχεδίασης Visual Paradigm. Για τη σχεδίαση του εννοιολογικού μοντέλου, των κλάσεων και για τη συγγραφή του κώδικα χρησιμοποιήθηκε το Microsoft Visual Studio 2017, ενώ η διαχείριση της βάσης δεδομένων πραγματοποιείται με το Entity Framework 6.2 της Microsoft. Η βάση δεδομένων υλοποιείται στο σύστημα Microsoft SQL Server 2017 με το οποίο πραγματοποιήθηκε και η δημιουργία του διαγράμματος οντοτήτων – συσχετίσεων της βάσης δεδομένων.

Η εφαρμογή είναι σχεδιασμένη έτσι ώστε να είναι εύχρηστη και ευέλικτη, λαμβάνοντας υπόψη τις απαιτήσεις και τα πιθανά προβλήματα που μπορεί ο χρήστης να συναντήσει κατά τη χρήση της. Έτσι σχεδιάστηκε δίνοντας στον χρήστη τη δυνατότητα να μεταβαίνει από μια φόρμα σε κάποια άλλη, προς συμπλήρωση κάποιων δεδομένων, χωρίς να του κοστίζει σε χρόνο, πόρους συστήματος και επαναλαμβανόμενη εργασία.

Λέξεις-κλειδιά: Πρόγραμμα εργασίας νοσηλευτικών ιδρυμάτων, αναζήτηση μεταβλητής γειτονιάς, ευρετικές και μεταευρετικές αλγόριθμοι, χρονοπρογραμματισμός

Περιεχόμενο: Κείμενο, βάσεις δεδομένων, πρόγραμμα σε γλώσσα C#, Entity Framework, MSSQL, πρωτότυπο εφαρμογής.

ABSTRACT

DEVELOP AN APPLICATION TO EFFECTIVELY SOLVING THE NURSE ROSTERING PROBLEM

Aristidis G. Pavlidis

Dr. Gregory Beligiannis Likothanasis Spiridon Dr. Georgopoulos Efstratios

This diploma thesis deals with the develop of a full graphical application that will solve the problem of Nurse Rostering, that is to say, the training of a workplace schedule of hospital institutions. Nurse Rostering's problem is characterized as an optimization problem, as the optimal or near-optimal solution is sought through a variety of other solutions while meeting any limitations that have been put into the NP - Complete Problems category. To solve these problems, heuristic and transformational algorithms are used. These algorithms are mainly applied to complicated optimization problems and in cases where other optimization methods either have failed altogether or have failed to be effective. In this application, the Neighborhood Search algorithm developed by Messrs. Beligiannis Gregorios, Tasopoulos Ioannis and John Ioannis, in his executable form.

This application, during its implementation, passed all stages of the design from the charts to the writing of the code and debugging it. For designing case usage charts and sequence diagrams, Visual Paradigm design software was used. Microsoft Visual Studio 2017 was used to design the conceptual model, the classes, and to write the code, while database management is performed with Microsoft Entity Framework 6.2. The database is implemented in the Microsoft SQL Server 2017 system, which also created the Entity Relationship Diagram of database.

This application is designed to be easy to use and flexible, taking into account the requirements and potential problems the user may encounter when using it. This was designed to give the user the ability to switch from one form to another, to fill in some data without costing him time, system resources, and repetitive work.

Keywords: Nursing Institutions Work Program, Neighborhood Quest, Heuristic and Post-Algorithm Algorithms, Scheduling

Content: Text, databases, program in C # language, Entity Framework, MSSQL, application prototype.

Περιεχόμενα

ΕΥΧΑΡΙΣΤΙΕΣ 5

ΠΕΡΙΛΗΨΗ 6

ABSTRACT 7

Περιεχόμενα 8

Συντομογραφίες & Ακρωνύμια 13

1. Αλγόριθμοι 14
   1.1 Αντικείμενο της πτυχιακής εργασίας 14
   1.2 Πεδίο εφαρμογής 15

2. Το πρόβλημα του χρονοπρογραμματισμού 20
   2.1 Χρονοπρογραμματισμός και NP-Πληρότητα 20
   2.2 Γενετικοί Αλγόριθμοι 21
   2.3 Βελτιστοποίηση σμήνους σωματιδίων 23
   2.4 Αναζήτηση Ταμπού 24
   2.5 Βελτιστοποίηση αποικιών μυρμηγκιών 24

3. Ο Αλγόριθμος Variable Neighborhood Search – VNS 26
   3.1 Η φιλοσοφία του αλγορίθμου VNS 26
   3.2 Παρουσίαση του αλγορίθμου VNS για επίλυση του προβλήματος Nurse Rostering 27

4. Ωρολόγιο πρόγραμμα εργασίας σε νοσοκομειακά ιδρύματα 40
   4.1 Η κατασκευή ωρολογίων προγραμμάτων στα νοσοκομειακά ιδρύματα 40
   4.2 Ο ορισμός του προβλήματος 41
   4.3 Το μαθηματικό μοντέλο του προβλήματος 44
      4.3.1. Ανελαστικοί περιορισμοί 44

5. Σχεδίαση 50
   5.1 Διάγραμμα Περιπτώσεων Χρήσης 50
   5.2 Εννοιολογικό μοντέλο 52
   5.3 Λεκτική περιγραφή περιπτώσεων χρήσης και σχετικά διαγράμματα 57
   5.4 Διαγράμματα κλάσεων 83
      5.4.1. Πακέτο InputData 85
      5.4.2. Πακέτο DataBase 89
      5.4.3. Πακέτο OutputData 90
      5.4.4. Πακέτο GUI 91
      5.4.5. Πακέτο Resourses 97

6. Βάση δεδομένων 98

7. Περιγραφή – επίδειξη λειτουργίας της εφαρμογής 102

8. Εργαλεία που χρησιμοποιήθηκαν 120

8.1 Visual Studio 2017 120

8.2 Entity Framework 121

8.2.1. Πρώτα το μοντέλο 122

8.2.2. Πρώτα η βάση δεδομένων 124

8.2.3. Πρώτα ο κώδικας 125

8.3 Visual Paradigm 127

9. Συμπεράσματα – Προτάσεις 128

Βιβλιογραφία και αναφορές 134

Παράρτημα Α: Κώδικας του προγράμματος NurseRostering 136

Πίνακας εικόνων

Εικόνα 1: Επισκόπηση του αλγορίθμου. 27
Εικόνα 2: Η διαδικασία Swap_Random_Rosters() 28
Εικόνα 3: Ψευδοκώδικας της 1ης φάσης 29
Εικόνα 4: Δομή και ψευδοκώδικας της Swap_Random_Rosters() 30
Εικόνα 5: Ο ψευδοκώδικας της διαδικασίας Groups_Swaps(R1, R2) 31
Εικόνα 6: Η δομή της διαδικασίας Groups_Swaps(R1, R2) 31
Εικόνα 7: Η δομή της 2ης Φάσης 33
Εικόνα 8: Δομή και ψευδοκώδικας της διαδικασίας Swap_Ordered_Roster() 35
Εικόνα 9: Ο ψευδοκώδικας της Swap_1() 36
Εικόνα 10: Ο ψευδοκώδικας της Swap_2() 36
Εικόνα 11: Ο ψευδοκώδικας της Swap_3() 37
Εικόνα 12: Ο ψευδοκώδικας της Swap_4() 38
Εικόνα 13: Ο ψευδοκώδικας της Swap_5() 38
Εικόνα 14: Ο ψευδοκώδικας της Swap_6() 39
Εικόνα 15: Διάγραμμα περιπτώσεων χρήσης του προγράμματος NurseRostering 51
Εικόνα 16: Εννοιολογικό μοντέλο των κλάσεων του προγράμματος εργασίας 53
Εικόνα 17: Διασύνδεση φορμών γραφικού περιβάλλοντος αλληλεπίδρασης με το NurseRostering 56
Εικόνα 18: Διάγραμμα ευρωστίας της ΠΧ "Διαχείριση υπαλλήλων" 63
Εικόνα 19: Διάγραμμα ακολουθίας της ΠΧ "Διαχείριση προσωπικού" 64
Εικόνα 20: Διάγραμμα ευρωστίας της ΠΧ "Διαχείριση βαρδιών" 69
Εικόνα 21: Διάγραμμα ακολουθίας της ΠΧ "Διαχείριση βαρδιών" 70
Εικόνα 22Διάγραμμα ευρωστίας της ΠΧ "Συμβόλαια προγράμματος εργασίας" 73
Εικόνα 23: Διάγραμμα ακολουθίας της ΠΧ "Συμβόλαια προγράμματος εργασίας" 74
Εικόνα 24: Διάγραμμα ευρωστίας της ΠΧ "Σχεδίαση προγράμματος εργασίας" 78
Εικόνα 25: Διάγραμμα ακολουθίας της ΠΧ "Σχεδίαση προγράμματος εργασίας" 79
Εικόνα 26: Διάγραμμα ευρωστίας της ΠΧ "Διαχείριση προγράμματος εργασίας" 81
Εικόνα 27:Διάγραμμα ακολουθίας της ΠΧ "Διαχείριση προγράμματος εργασίας" 83
Εικόνα 28: Διασύνδεση πακέτων μέσα στην εφαρμογή NurseRostering 84
Εικόνα 29: Κλάση Enotita 85
Εικόνα 30: Κλάση Skill 85
Εικόνα 31: Κλάση Day_Of_Week_Cover 85
Εικόνα 32: Κλάση Available_Pattern 85
Εικόνα 33: Κλάση Date_Specific_Cover 85

Εικόνα 34: Κλάση Available_Skill 85
Εικόνα 35: Κλάση Shift_On_Request 86
Εικόνα 36: Κλάση Day_Off_Request 86
Εικόνα 37: Κλάση Day_On_Request 86
Εικόνα 38: Κλάση Shift_Off_Request 86
Εικόνα 39: Κλάση Pattern 86
Εικόνα 40: Κλάση Scheduling_Period 86
Εικόνα 41: Κλάση Shift_Type 87
Εικόνα 42: Κλάση Employee 87
Εικόνα 43: Κλάση Contracts 88
Εικόνα 44: Κλάση NurseRosteringDB 89
Εικόνα 45: Κλάση MyDbConfiguration 89
Εικόνα 46: Κλάση Instance 90
Εικόνα 47: Κλάση Assignment 90
Εικόνα 48: Κλάση φόρμας frmUnderCon 91
Εικόνα 49: Κλάση φόρμας frmSplash 91
Εικόνα 50: Κλάση φόρμας frmEmployee 92
Εικόνα 51: Κλάση φόρμας frmAddSkill 92
Εικόνα 52: Κλάση φόρμας frmContracts 92
Εικόνα 53: Κλάση φόρμας frmChoice 93
Εικόνα 54: Κλάση φόρμας frmEdit 93
Εικόνα 55: Κλάση φόρμας frmNewPattern 93
Εικόνα 56: Κλάση φόρμας frmPreview 93
Εικόνα 57: Κλάση φόρμας frmEmpWork 94
Εικόνα 58: Κλάση φόρμας frmMain 94
Εικόνα 59: Κλάση φόρμας frmNewShift 95
Εικόνα 60: Κλάση φόρμας frmProgram 95
Εικόνα 61: Κλάση φόρμας frmScheduling_Period 96
Εικόνα 62: Κλάση φόρμας frmUnwantedPatterns 96
Εικόνα 63: Κλάση Modules 97
Εικόνα 64: Κλάση ClsPrint 97
Εικόνα 65: Διάγραμμα οντοτήτων - συσχετίσεων της βάσης δεδομένων του NurseRostering 100
Εικόνα 66: Εισαγωγική - ενημερωτική φόρμα 103
Εικόνα 67: Κύριο μενού επιλογών 104

Εικόνα 68: Μενού διαχείρισης προσωπικού 105
Εικόνα 69: Κλήση της φόρμας συμβολαίων ή της φόρμας καταχώρησης νέας ειδικότητας από τη φόρμα καταχώρησης νέου υπαλλήλου 106
Εικόνα 70: Φόρμα διαχείρισης ειδικοτήτων 106
Εικόνα 71: Επιθυμίες υπαλλήλου για ημέρες εργασίας και ανάπαυσης 107
Εικόνα 72: Επιθυμίες υπαλλήλου για ημέρες εργασίας και ανάπαυσης σε συγκεκριμένη βάρδια 107
Εικόνα 73: Μενού επιλογών διαχείρισης βαρδιών 108
Εικόνα 74: Πίνακας διαχείρισης δεδομένων 108
Εικόνα 75: Μενού διαχείρισης βαρδιών 109
Εικόνα 76: Καταχώρηση νέας βάρδιας 110
Εικόνα 77: Ανεπιθύμητα στοιχειώδη πρότυπα εργασίας 110
Εικόνα 78: Σύνθεση ανεπιθύμητων προτύπων εργασίας 111
Εικόνα 79: Διαχείριση συμβολαίων εργασίας 112
Εικόνα 80: Σχεδίαση και δημιουργία προγράμματος εργασίας 113
Εικόνα 81: Πίνακας προεπισκόπησης δεδομένων και αλλαγής αυτών 114
Εικόνα 82: Φόρμα επιλογής είδους προγράμματος εργασίας για εμφάνιση 115
Εικόνα 83: Πρόγραμμα εργασίας προσωπικού 116
Εικόνα 84: Προεπισκόπηση εκτύπωσης προγράμματος εργασίας 117
Εικόνα 85: Παράθυρο αποθήκευσης του προγράμματος εργασίας σε μορφή αρχείου *.xlsx 118
Εικόνα 86: Το πρόγραμμα εργασίας μέσα σε αρχείο μορφής *.xlsx 118
Εικόνα 87: Τοποθεσία του Entity Framework στην εφαρμογή. 122
Εικόνα 88: Λειτουργία προσέγγισης Πρώτα το μοντέλο 123
Εικόνα 89: Λειτουργία προσέγγισης Πρώτα η βάση δεδομένων 125
Εικόνα 90: Λειτουργία προσέγγισης Πρώτα ο κώδικας 126
Εικόνα 91: Visual Paradigm 127
Εικόνα 92: Απασχόληση κεντρικού επεξεργαστή και επεξεργαστή κάρτας γραφικών 128
Εικόνα 93: Απαιτήσεις μνήμης RAM 128
Εικόνα 94: Συγκεντρωτικά αποτελέσματα μετρήσεων κώδικα ανά πακέτο κλάσεων 129
Εικόνα 95: Αναλυτικά αποτελέσματα μετρήσεων κώδικα ανά κλάση 129
Εικόνα 96: Αποτελέσματα χρήσης CPU από την εκδοχή sprint του αλγορίθμου 130
Εικόνα 97: Αποτελέσματα χρήσης CPU από την εκδοχή medium του αλγορίθμου 131
Εικόνα 98: Αποτελέσματα χρήσης CPU από την εκδοχή long του αλγορίθμου 131

Συντομογραφίες & Ακρωνύμια

VNS	Variable Neighborhood Search
ABC	Artificial Bee Colony
ACO	Ant Colony Optimization
CSO	Cat Swarm Otpimization
AFS	Artificial Fish Swarm
PSO	Particle Swarm Otpimization
SVNS	Stochastic Variable Neighborhood Search
NR	Nurse Rostering
INRC-I	International Nurse Rostering Competition I
AI	Artificial Intelligence
TN	Τεχνητή Νοημοσύνη
ΠΧ	Περίπτωση Χρήσης
GUI	Graphic User Interface
IDE	Integrated Development Environment
API	Application Programming Interface
O/RM	Object-Relational Mapping
XML	eXtensible Markup Language
UML	Unified Modeling Language
CPU	Central Processor Unit
GPU	Graphics Processor Unit
RAM	Random Access Memory
OS	Operating System

1. Αλγόριθμοι

Στο αυτό το κεφάλαιο, πραγματοποιείται μια ανάλυση των προδιαγραφών που απαιτούνται για την ανάπτυξη πλήρους εφαρμογής η οποία θα επιλύει αποδοτικά το πρόβλημα του ωρολογίου προγράμματος εργασίας νοσοκομειακών ιδρυμάτων, με τέτοιο τρόπο ώστε να ικανοποιούνται κάποιοι περιορισμοί που τίθενται από το νοσοκομειακό ίδρυμα ή/και από τις ιδιαιτερότητες του προσωπικού. Επίσης, θα πραγματοποιηθεί και μια παρουσίαση και ανάλυση του στοχαστικού αλγορίθμου Variable Neighborhood Search (VNS στο εξής) – Αναζήτηση μεταβλητής γειτονιάς.

1.1 Αντικείμενο της πτυχιακής εργασίας

Η παρούσα πτυχιακή εργασία, έχει ως αντικείμενο τη ανάπτυξη μιας εφαρμογής η οποία θα αυτοματοποιεί τη δημιουργία ενός ωρολογίου προγράμματος εργασίας των υπαλλήλων νοσοκομειακών ιδρυμάτων, η διάρκεια του οποίου θα καθορίζεται από τον χρήστη. Η εφαρμογή αυτή, θα βασίζεται στον αλγόριθμο VNS ο οποίος σχεδιάστηκε και παρουσιάστηκε από τους κκ. Μπεληγιάννη Γρηγόρη, Τασσόπουλο Ιωάννη και Σώλο Ιωάννη (Beligiannis, Tasopoulos, & Solos, 2013). Παρουσίαση και ανάλυση αυτού του αλγορίθμου θα πραγματοποιηθεί σε παρακάτω κεφάλαιο.

Το πρόβλημα χρονοπρογραμματισμού προσωπικού των νοσοκομείων είναι από τα δυσκολότερα προβλήματα που πρέπει να αντιμετωπίσουν οι διοικήσεις των νοσοκομειακών ιδρυμάτων. Κατά τη διαδικασία κατασκευής του προγράμματος θα πρέπει να ληφθούν υπόψιν πολλές οντότητες και παράμετροι οι οποίοι καθορίζουν και την πολυπλοκότητα της λύσης. Μερικές από τις οντότητες, ενδεικτικά αναφέρεται ότι, μεταξύ άλλων, είναι οι νοσοκόμοι και οι βάρδιες στις οποίες καλούνται να εργαστούν. Ως παράμετροι, μπορούν να χαρακτηριστούν οι ιδιαιτερότητες του προσωπικού ως προς το χρόνο εργασίας, η κατάρτιση του προσωπικού, το πλήθος των νοσοκόμων που θα βρίσκονται στο χώρο εργασίας σε κάθε βάρδια, καθώς και πολλοί άλλοι περιορισμοί καθοριζόμενοι είτε από τη διοίκηση του νοσοκομειακού ιδρύματος, είτε από το ίδιο το προσωπικό. Θα πρέπει όλα τα ανωτέρω, να ληφθούν υπόψιν κατά τη σχεδίαση της εφαρμογής, έτσι ώστε να κατασκευαστεί ένα σωστά δομημένο και αποδοτικό πρόγραμμα το οποίο θα επιλύει το πρόβλημα του χρονοπρογραμματισμού προσωπικού.

Σήμερα, παρόλο που η εξέλιξη των μέσων υψηλής τεχνολογίας, σε σχέση με το πρόσφατο παρελθόν (20 με 30 χρόνια, το πολύ), «τρέχει» με πολύ γρηγορότερους ρυθμούς, δεν είναι ακόμα δυνατόν να εφαρμοστεί η εξαντλητική αναζήτηση σε πλήθος προβλημάτων, λόγω του ότι ο απαιτούμενος χρόνος επίλυσής τους παραμένει ακόμα πολύ μεγάλος και σε πολλές

περιπτώσεις όχι πραγματικός σε σχέση με τη διάρκεια της ανθρώπινης ζωής. Παρόλα αυτά όμως, μπορεί να δοθεί μια αποδοτική και εφικτή λύση η οποία αν δεν είναι η βέλτιστη, τείνει προς σ' αυτήν, με τη δημιουργία κάποιων αλγορίθμων οι οποίοι δεν αναζητούν τη βέλτιστη λύση ανάμεσα σε όλες τις δυνατές λύσεις ενός προβλήματος, αλλά με κάποια κριτήρια και τεχνικές περιορίζονται σε ένα υποσύνολο λύσεων το οποίο είναι πιθανό να περιέχει και τη βέλτιστη λύση, μειώνοντας κατά πολύ τον χρόνο ανεύρεσης λύσης ενός προβλήματος, εντός αποδεκτών χρονικών ορίων. Τέτοιοι αλγόριθμοι επίλυσης προβλημάτων είναι ο Artificial Bee Colony (ABC), ο Cat Swarm Otpimization (CSO), ο Artificial Fish Swarm (AFS), ο Particle Swarm Otpimization (PSO) και ο Variable Neighborhood Search (VNS).

Στους αλγόριθμους Variable Neighborhood Search – VNS ανήκει και ο αλγόριθμος ο οποίος παρουσιάστηκε για πρώτη φορά στον από τους κκ. Μπεληγιάννη Γρηγόρη, Τασσόπουλο Ιωάννη και Σώλο Ιωάννη στις 21 Μαΐου 2013 στο περιοδικό Algorithms (Beligiannis, Tasopoulos, & Solos, 2013, σσ. 278 - 308). Το 2010 διεξήχθη ο 1ος Διεθνής Διαγωνισμός (NR Competition) σχετικά με την αποδοτική επίλυση του προβλήματος Nurse Rostering. Συγκριτικά με τις απαιτήσεις του διαγωνισμού, ο ανωτέρω αλγόριθμος κατάφερε να ισοφαρίσει τα καλύτερα γνωστά αποτελέσματα σχεδόν για όλα τα αρχεία του διαγωνισμού και σε ορισμένες περιπτώσεις ακόμα και να τα ξεπεράσει. Στη συνέχεια σχεδιάστηκε και υλοποιήθηκε δεύτερος αλγόριθμος στοχαστικής μεταβλητής γειτονιάς (SVNS – Stochastic Variable Neighborhood Search) ο οποίος είναι και ο καλύτερος στην αντίστοιχη βιβλιογραφία, σχετικά με την απόδοσή του στα δεδομένα εισόδου του διαγωνισμού INRC-I (Σώλος, 2016, σ. 16). Ο αλγόριθμος αυτός, θα υλοποιηθεί μέσω εφαρμογής που θα αναπτυχθεί για την επίλυση του προβλήματος προγραμματισμού βαρδιών του νοσηλευτικού προσωπικού ενός νοσοκομειακού ιδρύματος. Θα αντληθούν δεδομένα και πληροφορίες από το διαδίκτυο, από βιβλιοθήκες αλλά και από τον διαγωνισμό αυτό.

1.2 Πεδίο εφαρμογής

Για την εύρυθμη και απροβλημάτιστη λειτουργία κάθε επιχείρησης, ιδρύματος, οργανισμού και υπηρεσίας οι οποίες λειτουργούν υπό το καθεστώς των εναλλασσόμενων βαρδιών πρέπει να δημιουργεί κάποιο πρόγραμμα, περιορισμένης χρονικής διάρκειας, για την κατανομή του προσωπικού σε βάρδιες, σύμφωνα με τις απαιτήσεις και τις ανάγκες του εργοδότη αλλά και σε συνδυασμό με τις ανάγκες και ιδιαιτερότητες του προσωπικού. Το πρόγραμμα έχει διάρκεια συνήθως μια εβδομάδα ή ένα μήνα, αλλά πάντα είναι ορισμένο για κάθε ημέρα ξεχωριστά. Η δημιουργία αυτού του προγράμματος, καλείται ως χρονοπρογραμματισμός προσωπικού. Ο χρονοπρογραμματισμός προσωπικού είναι μια

δύσκολη διαδικασία διότι για την κατασκευή ενός πλάνου εργασίας όλων των υπαλλήλων για κάθε ημέρα της εβδομάδας, για κάθε εβδομάδα του μήνα πρέπει να λαμβάνονται υπόψιν όλες τις απαιτήσεις και ιδιαιτερότητες αλλά και άλλες παραμέτρους πχ νομικοί περιορισμοί αλλά και σύγκρουση κάποιων ιδιαιτεροτήτων.

Ένα καλό παράδειγμα μπορεί να αποτελέσει ένα γηροκομείο. Σε ένα γηροκομείο, οι τρόφιμοι δεν είναι πάντα με υγεία σε άριστη κατάσταση. Κάποιοι μπορεί να είναι άρρωστοι, κάποιοι να χρήζουν βοήθειας για τις βασικές τους ανθρώπινες ανάγκες, ενώ μπορεί να έχουν και κινητικά προβλήματα. Γι' αυτό, σε ένα γηροκομείο πρέπει να υπάρχει προσωπικό διαφόρων ειδικοτήτων 24 ώρες το 24ωρο. Σε κάθε βάρδια, δύναται να απασχολείται διαφορετικό πλήθος υπαλλήλων κάθε μέρα, ανάλογα με τις άδειες, τις ημέρες ανάπαυσης αλλά και διάφορους άλλους μη προβλέψιμους λόγους. Επίσης, ανάλογα με τις απαιτήσεις του γηροκομείου, το ωράριο του καθενός εργαζομένου μπορεί να είναι διαφορετικής χρονικής διάρκειας, καθώς και η προσέλευσή τους στο ίδρυμα αλλά και αποχώρησή τους από αυτό, να γίνεται σε διαφορετικές ώρες. Κάποιες άλλες παράμετροι που μπορεί να επηρεάζουν τη λειτουργία του γηροκομείου, είναι κάποιες εργασίες που δεν γίνονται καθημερινά αλλά σε κάποια τακτά χρονικά διαστήματα κατά τα οποία απαιτείται αυξημένο προσωπικό. Με βάση όλα τα παραπάνω, γίνεται εύκολα αντιληπτό του γεγονός ότι ο χρονοπρογραμματισμός προσωπικού υπό περιορισμούς είναι δύσκολη και χρονοβόρα διαδικασία. Η δυσκολία της και οι χρονικές απαιτήσεις του χρονοπρογραμματισμού, αυξάνονται εκθετικά ή ακόμα και παραγοντικά σε σχέση με το πλήθος του προσωπικού. Μέσα σε όλα αυτά, μπορούμε να προσθέσουμε και την σύγκρουση ή ανταγωνιστικότητα κάποιων παραμέτρων οι οποίες μπορούν να μην οδηγήσουν σε βέλτιστη λύση του προβλήματος αλλά ακόμα και να καταστήσουν το πρόβλημα άλυτο.

Ένα άλλο αντιπροσωπευτικό παράδειγμα όπου απαιτείται χρονοπρογραμματισμός προσωπικού, είναι αυτό ενός νοσοκομειακού ιδρύματος. Σε ένα νοσοκομειακό ίδρυμα υπάρχουν διάφορες κλινικές και κάθε κλινική έχει τους νοσηλευτές της. Κάθε κλινική καταρτίζει το δικό της χρονοπρογραμματισμό προσωπικού για τους νοσηλευτές της, λαμβάνοντας υπόψιν διάφορες παραμέτρους και περιορισμούς που θέτουν το νοσοκομειακό ίδρυμα και οι νοσηλευτές.

Μέχρι μερικά χρόνια πριν, η κατασκευή του χρονοπρογραμματισμού προσωπικού γινόταν «με το χέρι» από κάποιους υπαλλήλους. Είναι μια διαδικασία πάρα πολύ χρονοβόρα, με όχι πάντα ικανοποιητικά αποτελέσματα για όλους τους υπαλλήλους, αφού δεν είναι δυνατόν να ληφθούν υπόψιν και να συνδυαστούν μεταξύ τους όλες οι απαιτήσεις ή όλοι οι περιορισμοί που επιβαλλόντουσαν εκατέρωθεν των πλευρών (ίδρυμα – υπάλληλοι). Την λύση στο πρόβλημα αυτό όμως, φαίνεται να τη βρίσκουμε με τη χρήση των Ηλεκτρονικών

Υπολογιστών. Η ολοένα αυξανόμενη πρόοδος στον τομέα των εξαρτημάτων των ηλεκτρονικών υπολογιστών (Hardware) σε σχέση με το παρελθόν, έχει οδηγήσει στη σημαντική αύξηση της υπολογιστικής ισχύος. Αυτό, σε συνδυασμό με την ανάπτυξη και υλοποίηση, σχετικών με συγκεκριμένα προβλήματα, αλγορίθμων, έχουν καταστήσει δυνατή την επίλυση πάρα πολλών προβλημάτων του είδους τους, σε αποδεκτό χρονικό διάστημα, ανάλογο με τις απαιτήσεις και τους περιορισμούς που τίθενται σε κάθε πρόβλημα.

Ο χρονοπρογραμματισμός προσωπικού ανήκει στα προβλήματα βελτιστοποίησης διότι πρέπει να βρούμε τη βέλτιστη – ανάμεσα σε άλλες πολλές – λύση, σύμφωνα με κάποιους περιορισμούς που επιβάλλονται. Η επιστήμη που ασχολείται με τα προβλήματα αυτού του είδους, ονομάζεται Επιχειρησιακή Έρευνα. Επιχειρησιακή Έρευνα, είναι μια διεπιστημονική μαθηματική επιστήμη που επικεντρώνεται στην αποτελεσματική χρήση της τεχνολογίας από τις οργανώσεις. Βέβαια ο όρος «Επιχειρησιακή» (Operations) έχει την έννοια της διαδικασίας ή λειτουργίας και όχι της εταιρίας ή επιχείρησης. Αντίθετα, πολλοί άλλοι κλάδοι της Τεχνολογίας και της Μηχανικής, έχουν ως εστίαση την ανάπτυξη αυτής της επιστήμης για την παραγωγή μοντέλων λύσεων. (Wikipedia - Επιχειρησιακή έρευνα). Άλλος ορισμός που έχει δοθεί σχετικά με την Επιχειρησιακή Έρευνα είναι πως «Η Επιχειρησιακή Έρευνα είναι η επιστήμη που ασχολείται με την διαδικασία λήψης αποφάσεων μέσω διάφορων επιστημονικών μεθόδων και μαθηματικών προτύπων». Σχετικοί επιστημονικοί κλάδοι με την Επιχειρησιακή Έρευνα, είναι η Υπολογιστική Νοημοσύνη και η Τεχνητή Νοημοσύνη.

Η *κλασσική* ή *συμβολική Τεχνητή Νοημοσύνη* (symbolic AI) που βασίζεται στην κατανόηση των νοητικών διεργασιών και ασχολείται με την προσομοίωση της ανθρώπινης νοημοσύνης προσεγγίζοντάς τη με αλγορίθμους και συστήματα που βασίζονται στη γνώση, χρησιμοποιώντας ως δομικές μονάδες τα σύμβολα. Ένα σύμβολο μπορεί να αναπαριστά μια έννοια ή μια σχέση ανάμεσα σε έννοιες. Παραδείγματα αυτής της κατηγορίας είναι οι εφαρμογές της ΤΝ που χρησιμοποιούν αναπαράσταση γνώσης όπως λογική, κανόνες, πλαίσια, κτλ. (Βλαχάβας, Βασιλειάδης, Κόκκορας, Σακελλαρίου, & Κεφάλας, 2011)

Η Υπολογιστική Νοημοσύνη (computational intelligence) ή συνδετική (connectionist) ή μη συμβολική Τεχνητή Νοημοσύνη (non symbolic AI) που βασίζεται στη μίμηση βιολογικών διεργασιών, όπως η διαδικασία της εξέλιξης των ειδών ή η λειτουργία του εγκεφάλου. Παραδείγματα τέτοιων τεχνικών αποτελούν τα Νευρωνικά Δίκτυα (neural networks) και οι γενετικοί αλγόριθμοι (genetic algorithms). (Βλαχάβας, Βασιλειάδης, Κόκκορας, Σακελλαρίου, & Κεφάλας, 2011)

Εμβαθύνοντας λίγο στην Επιχειρησιακή Έρευνα θα συναντήσουμε τις παρακάτω έννοιες: δυναμικός προγραμματισμός, ακέραιος προγραμματισμός, γραμμικός και μη γραμμικός προγραμματισμός, θεωρία παιγνίων, προγραμματισμός με περιορισμούς καθώς και

διάφορες άλλες οι οποίες δεν αναφέρονται διότι δεν αποτελούν αντικείμενο της παρούσας πτυχιακής εργασίας. Μια ευρύτερη κατηγοριοποίηση της Επιχειρησιακής Έρευνας μπορεί να θεωρηθεί ότι καλύπτει το μεγαλύτερο υποσύνολο προβλημάτων. Κύριες κατηγορίες μεθόδων επίλυσης προβλημάτων μπορούμε να θεωρήσουμε τις παρακάτω:

- Μέθοδοι προσομοίωσης – μελέτη της δυνατότητας βελτιώσεων και στη συνέχεια έλεγχος της απόδοσης των βελτιώσεων αυτών.
- Μέθοδοι βελτιστοποίησης – επιλογή βέλτιστης ή αποδεκτής λύσης με τρόπο αποτελεσματικό και αποδοτικό, μέσα από ένα πλήθος λύσεων με τη χρήση κάποιων περιορισμών ή κριτηρίων.
- Μέθοδοι ανάλυσης δεδομένων – αναγνώριση μοτίβων και σύνδεση αυτών με τα δεδομένα που έχουμε στη διάθεσή μας. Αυτή η μέθοδος είναι χρήσιμη στον τομέα των προβλέψεων αλλά και σε αυτόν της απόκτησης γνώσης, γνώσης που δεν κατείχαμε πρότερα, από τα δεδομένα που έχουμε στη διάθεσή μας.

Από όλες τις προαναφερθείσες έννοιες, θα αναφερθούμε μόνο στον Ακέραιο Προγραμματισμό και ειδικότερα στον Προγραμματισμό με Περιορισμούς και στις Μεταευρετικές Τεχνικές.

Στον Ακέραιο Προγραμματισμό συμπεριλαμβάνονται όλα τα προβλήματα των οποίων οι μεταβλητές τους μπορούν να πάρουν μόνο ακέραιες τιμές. Σε κάποιες περιπτώσεις που οι τιμές κάποιων μεταβλητών μπορεί να μην είναι ακέραιος αριθμός και κάποιων άλλων να είναι ακέραιος, τότε έχουμε πρόβλημα Μεικτού Ακέραιου Προγραμματισμού. Σε κάθε περίπτωση όμως, ένα πρόβλημα Ακέραιου Προγραμματισμού θα είναι γραμμικό ή μη γραμμικό. Δύσκολα προβλήματα συνδυαστικής βελτίωσης λύνονται αποδοτικά με χρήση του Ακέραιου Προγραμματισμού. Μέσα στο πεδίο του Ακέραιου Προγραμματισμού, συναντάμε διάφορα μοντέλα αυτού. Παρόλο που ένα πρόβλημα μπορεί να εφαρμοστεί σε παραπάνω από ένα μοντέλα, μπορεί και σε όλα, αυτό δεν θα λυθεί το ίδιο αποδοτικά από όλα τα μοντέλα ή σε αποδεκτό χρόνο από όλα τα μοντέλα. Άρα, κάποιες φορές η εύρεση της βέλτιστης λύσης ή ακόμα και απλώς μιας λύσης κάποιου προβλήματος, εξαρτάται κατά πολύ από το μοντέλο το οποίο θα επιλέξουμε για να λύσουμε το πρόβλημα αυτό.

Οι ευρετικές και οι μεταευρετικές μέθοδοι χρησιμοποιούνται ευρέως για την προβλημάτων χρονοπρογραμματισμού προσωπικού. Η ευρεία αποδοχή και δημοτικότητα αυτών, οφείλεται στο ότι:

- Είναι σχετικά ισχυροί. Δεν εγγυώνται την εύρεση βέλτιστης λύσης. Συνήθως παράγουν αποδεκτές λύσεις εντός αποδεκτού χρονικού διαστήματος, με ένα ευρύ φάσμα δεδομένων εισόδου. Σε αντίθεση με άλλες τεχνικές του Ακέραιου

Προγραμματισμού, υπάρχει περίπτωση να μην μπορέσουν να βρουν κάποια αποδεκτή λύση για μεγάλο χρονικό διάστημα, ακόμα κι αν αυτή υπάρχει.

- Οι ευρετικές συναρτήσεις ασχολούνται με σύνθετους στόχους
- Οι περισσότερες μεταευρετικές είναι απλές στην εφαρμογή τους με αποτέλεσμα να επιτρέπουν ενσωμάτωση και αξιοποίηση ειδικών πληροφοριών προς επίλυση του προβλήματος

Στη βιβλιογραφία υπάρχει ένα πλήθος μεταευρετικών αλγορίθμων που έχουν αναπτυχθεί σχετικά με το πρόβλημα του χρονοπρογραμματισμού προσωπικού νοσοκομειακών ιδρυμάτων. Το πρόβλημα αυτό, αντιμετωπίζεται με: Γενετικούς αλγόριθμους, Αναζήτηση Ταμπού, Αναζήτηση Μεταβλητής Γειτονιάς, Προσομοιωμένη Ανόπτηση, Επαναλαμβανόμενη Τοπική Αναζήτηση, Particle Swarm Optimization (PSO), Ant Colony Optimization (ACO) κλπ. (Beligiannis, Tasopoulos, & Solos, 2013)

2. Το πρόβλημα του χρονοπρογραμματισμού

Στο κεφάλαιο αυτό, θα γίνει μια σύντομη παρουσίαση του προβλήματος του χρονοπρογραμματισμού. Ειδικότερα δε, θα μελετηθεί το πρόβλημα του χρονοπρογραμματισμού των βαρδιών των υπαλλήλων ενός νοσοκομειακού ιδρύματος. Επίσης δίνεται ο ορισμός της κλάσεως των NP-πλήρων (NP-Complete) προβλημάτων στην οποία εντάσσεται και το πρόβλημα του χρονοπρογραμματισμού. Στη συνέχεια, θα παρουσιαστούν 4 μεταευρετικές μέθοδοι που χρησιμοποιούνται στον χρονοπρογραμματισμό προσωπικού.

2.1 Χρονοπρογραμματισμός και NP-Πληρότητα

Σε όλους τους οργανισμούς, όλα τα ιδρύματα, υπηρεσίες και επιχειρήσεις που λειτουργούν με σύστημα εργασίας βασισμένο σε βάρδιες προσωπικού, υπάρχει ανάγκη χρονοπρογραμματισμού του προσωπικού αυτού και κατανομή τους στις βάρδιες αυτές για την απρόσκοπτη και ομαλή λειτουργία τους. Ο χρονοπρογραμματισμός προσωπικού και γενικότερο ο χρονοπρογραμματισμός, είναι ένα δύσκολο πρόβλημα, λόγω των πολλών οντοτήτων και παραμέτρων που περιλαμβάνει. Στα μαθηματικά εντάσσονται στην κατηγορία των NP – Πλήρων (NPC – NP-Complete) προβλημάτων. Ένα πρόβλημα χαρακτηρίζεται ως NP – Πλήρες πρόβλημα, όταν για την επίλυσή του δεν υπάρχει κάποιος ντετερμινιστικός αλγόριθμος πολυωνυμικού χρόνου εκτέλεσής του που να είμαστε σε θέση να γνωρίζουμε, αλλά για την επαλήθευσή του υπάρχει κάποιος ντετερμινιστικός αλγόριθμος πολυωνυμικού χρόνου. Συνεπώς, για την επίλυση NP – πλήρων προβλημάτων σε αποδεκτό χρόνο, καταφεύγουμε σε εναλλακτικούς αλγόριθμους οι οποίοι μας παρέχουν μια σχεδόν βέλτιστη λύση. Έχει αποδειχθεί (Cooper & Jeffrey) ότι το πρόβλημα του χρονοπρογραμματισμού στη γενική του μορφή, είναι NP – Πλήρες. Οι αλγόριθμοι που βοηθούν στην επίλυση τέτοιου είδους προβλημάτων είναι οι ευρετικοί και οι μεταευρετικοί. Οι ευρετικοί και οι μεταευρετικοί αλγόριθμοι περιλαμβάνουν μια μεγάλη κατηγορία μεθόδων για την επίλυση του προβλήματος χρονοπρογραμματισμού νοσοκομείων. Εφαρμόζονται κυρίως σε πολύπλοκα προβλήματα βελτιστοποίησης και σε περιπτώσεις όπου άλλες μέθοδοι βελτιστοποίησης έχουν αποτύχει να είναι είτε αποτελεσματικές. Οι ευρετικές και οι μεταευρετικές μέθοδοι επίλυσης του προβλήματος χρονοπρογραμματισμού νοσοκομείων περιλαμβάνουν (Σώλος, 2016, σσ. 104 - 105):

- εξελικτικούς αλγόριθμους
- μιμητικούς αλγόριθμους
- αναζήτηση μεταβλητής γειτονιάς
- αναζήτηση σκέδασης
- αναζήτηση ταμπού
- επαναλαμβανόμενη τοπική αναζήτηση
- προσομοιωμένη ανόπτηση
- βελτιστοποίηση αποικίας μυρμηγκιών
- βελτιστοποίηση σμήνους σωματιδίων

Στις επόμενες 4 ενότητες γίνεται μια συνοπτική παρουσίαση μερικών αλγορίθμων από τους ανωτέρω. Η Αναζήτηση Μεταβλητής Γειτονιάς θα παρουσιαστεί σε παρακάτω κεφάλαιο λεπτομερώς.

2.2 Γενετικοί Αλγόριθμοι

Οι Γενετικοί αλγόριθμοι ανήκουν στο κλάδο της επιστήμης υπολογιστών και αποτελούν μια μέθοδο αναζήτησης βέλτιστων λύσεων σε συστήματα που μπορούν να περιγραφούν ως μαθηματικό πρόβλημα. Είναι χρήσιμοι σε προβλήματα που περιέχουν πολλές παραμέτρους/διαστάσεις και δεν υπάρχει αναλυτική μέθοδος που να μπορεί να βρει το βέλτιστο συνδυασμό τιμών για τις μεταβλητές ώστε το υπό εξέταση σύστημα να αντιδρά με όσο το δυνατόν με το επιθυμητό τρόπο.

Ο τρόπος λειτουργίας των Γενετικών Αλγορίθμων είναι εμπνευσμένος από τη βιολογία. Χρησιμοποιεί την ιδέα της εξέλιξης μέσω γενετικής μετάλλαξης, φυσικής επιλογής και διασταύρωσης. Οι Γενετικοί Αλγόριθμοι είναι αρκετά απλοί στην υλοποίησή τους. Οι τιμές για τις παραμέτρους του συστήματος πρέπει να κωδικοποιούνται με τρόπο ώστε να αναπαρασταθούν από μια μεταβλητή που περιέχει σειρά χαρακτήρων ή δυαδικών ψηφίων (0/1). Αυτή η μεταβλητή μιμείται το γενετικό κώδικα που υπάρχει στους ζωντανούς οργανισμούς. Αρχικά, ο Γενετικός Αλγόριθμος παράγει πολλαπλά αντίγραφα της μεταβλητής/γεννητικού κώδικα, συνήθως με τυχαίες τιμές, δημιουργώντας ένα πληθυσμό λύσεων. Κάθε λύση (τιμές για τις παραμέτρους του συστήματος) δοκιμάζεται για το πόσο κοντά φέρνει την αντίδραση του συστήματος στην επιθυμητή, μέσω μιας συνάρτησης που δίνει το μέτρο ικανότητας της λύσης και η οποία ονομάζεται συνάρτηση ικανότητας (Σ.Ι).

Οι λύσεις που βρίσκονται πιο κοντά στην επιθυμητή, σε σχέση με τις άλλες, σύμφωνα με το μέτρο που μας δίνει η Σ.Ι, αναπαράγονται στην επόμενη γενιά λύσεων και λαμβάνουν μια τυχαία μετάλλαξη. Επαναλαμβάνοντας αυτή τη διαδικασία για αρκετές γενιές, οι τυχαίες μεταλλάξεις σε συνδυασμό με την επιβίωση και αναπαραγωγή των γονιδίων/λύσεων που πλησιάζουν καλύτερα το επιθυμητό αποτέλεσμα θα παράγουν ένα γονίδιο/λύση που θα περιέχει τις τιμές για τις παραμέτρους που ικανοποιούν όσο καλύτερα γίνεται την Σ.Ι.

Υπάρχουν διάφορες εκδοχές της παραπάνω διαδικασίας για τους Γ.Α από τις οποίες κάποιες περιλαμβάνουν και τη διασταύρωση (ζευγάρωμα) γονιδίων/λύσεων ώστε ο αλγόριθμος να φτάσει στο αποτέλεσμα πιο γρήγορα. Καθώς υπάρχει το στοχαστικό (τυχαίο) συστατικό της μετάλλαξης και ζευγαρώματος, κάθε εκτέλεση του Γ.Α μπορεί να συγκλίνει σε διαφορετική λύση και σε διαφορετικό χρόνο. Η απόδοση του Γ.Α εξαρτάται επί το πλείστον από την συνάρτηση ικανότητας και συγκεκριμένα από το κατά πόσο το μέτρο της περιγράφει την βέλτιστη λύση. Οι γενετικοί αλγόριθμοι είναι ένα πεπερασμένο σύνολο οδηγιών για την εκπλήρωση ενός έργου, το οποίο δεδομένης μιας αρχικής κατάστασης θα οδηγήσει σε μια αναγνωρίσιμη τελική κατάσταση, και το οποίο προσπαθεί να μιμηθεί την διαδικασία της βιολογικής εξέλιξης. Οι γενετικοί αλγόριθμοι προσπαθούν να βρουν τη λύση ενός προβλήματος με το να προσομοιώνουν την εξέλιξη ενός πληθυσμού «λύσεων» του προβλήματος.

Είναι μια τεχνική προγραμματισμού που εισήγαγε στα τέλη της δεκαετίας του 1960 ο Τζον Χόλαντ, ερευνητής του Ινστιτούτου της Σάντα Φε (ΗΠΑ).

Ο τρόπος λειτουργίας των Γενετικών Αλγορίθμων είναι εμπνευσμένος από την βιολογία. Χρησιμοποιεί την ιδέα της εξέλιξης μέσω γενετικής μετάλλαξης, φυσικής επιλογής και διασταύρωσης.

Οι Γενετικοί Αλγόριθμοι διατηρούν έναν πληθυσμό πιθανών λύσεων, του προβλήματος που μας ενδιαφέρει, πάνω στον οποίο δουλεύουν, σε αντίθεση με άλλες μεθόδους αναζήτησης που επεξεργάζονται ένα μόνο σημείο του διαστήματος αναζήτησης. Έτσι ένας Γενετικός Αλγόριθμος πραγματοποιεί αναζήτηση σε πολλές κατευθύνσεις και υποστηρίζει καταγραφή και ανταλλαγή πληροφοριών μεταξύ αυτών των κατευθύνσεων. Ο πληθυσμός υφίσταται μια προσομοιωμένη γενετική εξέλιξη χρησιμοποιώντας διάφορους γενετικούς τελεστές όπως η επιλογή, η διασταύρωση και η μετάλλαξη.

Στην πράξη ο αλγόριθμος ξεκινά μ' ένα σύνολο λύσεων - ονομάζονται γονιδιώματα, δανειζόμενες το όνομά τους από τη βιολογία- οι οποίες συνιστούν τον «πληθυσμό». Κατόπιν ζητείται από τον υπολογιστή να δημιουργήσει μια σειρά τυχαίων ανασυνδυασμών και μεταλλάξεων των «γονιδιωμάτων».

Οι πιο ικανές λύσεις για ένα συγκεκριμένο πρόβλημα συνεχίζουν να εξελίσσονται και ανασυνδυάζονται τυχαία, μέχρις ότου "επιβιώσουν" οι καλύτερες. Συνήθως, όσο περισσότερες γενιές περνούν τόσο καλύτερες λύσεις βρίσκονται, μπορεί όμως ο αλγόριθμος να βρεθεί σε σημείο του πεδίου των λύσεων από όπου και δεν μπορεί να προχωρήσει λόγο του ότι βρίσκεται σε τοπικό μέγιστο. Για το λόγο αυτό έχουν υπάρχουν διαφορετικές εκδοχές του αλγόριθμου ανάλογα με τη μορφή του προβλήματος (Wikipedia - Γενετικοί αλγόριθμοι).

2.3 Βελτιστοποίηση σμήνους σωματιδίων

Στην υπολογιστική επιστήμη, η βελτιστοποίηση σμήνους σωματιδίων (PSO) είναι μια υπολογιστική μέθοδος που βελτιστοποιεί ένα πρόβλημα με την επανειλημμένη προσπάθεια βελτίωσης μιας υποψήφιας λύσης σε σχέση με ένα δεδομένο μέτρο ποιότητας. Επιλύει ένα πρόβλημα με το να έχει ένα πληθυσμό, ο οποίος καλείται «σμήνος», των υποψήφιων λύσεων, οι οποίες καλούνται «σωματίδια» και μετακινεί αυτά τα σωματίδια γύρω από τον χώρο αναζήτησης σύμφωνα με απλούς μαθηματικούς τύπους πάνω από τη θέση και την ταχύτητα του σωματιδίου. Η κίνηση κάθε σωματιδίου επηρεάζεται από την τοπικά καλύτερη γνωστή θέση του, αλλά καθοδηγείται επίσης προς τις πιο καλές γνωστές θέσεις στον χώρο αναζήτησης, οι οποίες ενημερώνονται καθώς οι καλύτερες θέσεις εντοπίζονται από άλλα σωματίδια. Αυτό αναμένεται να μετακινήσει το σμήνος προς τις καλύτερες λύσεις.

Το PSO αποδίδεται αρχικά στους Kennedy, Eberhart και Shi και αρχικά προορίστηκε για την προσομοίωση της κοινωνικής συμπεριφοράς ως σχηματοποιημένη αναπαράσταση της κίνησης των ατόμων σε ένα σμήνος πουλιών ή σε ένα κοπάδι ψαριών. Ο αλγόριθμος απλοποιήθηκε και παρατηρήθηκε ότι είναι κατάλληλος για βελτιστοποίηση. Το βιβλίο του Kennedy και του Eberhart (Swarm Intelligence, 2001) περιγράφει πολλές φιλοσοφικές πτυχές της PSO και της ευφυΐας.

Το PSO είναι μεταευρετικό, καθώς κάνει λίγες ή καθόλου υποθέσεις σχετικά με το πρόβλημα που έχει βελτιστοποιηθεί και μπορεί να αναζητήσει πολύ μεγάλους χώρους υποψήφιων λύσεων. Ωστόσο, δεν υπάρχουν ποτέ μεταευρετικά, όπως το PSO που εγγυώνται την βέλτιστη λύση (Wikipedia - PSO).

2.4 Αναζήτηση Ταμπού

Η αναζήτηση Tabu , που δημιουργήθηκε από τον Fred W. Glover το 1986 και επισημοποιήθηκε το 1989. Είναι μια μεταευρετική μέθοδος αναζήτησης που χρησιμοποιεί τοπικές μεθόδους αναζήτησης που χρησιμοποιούνται για τη μαθηματική βελτιστοποίηση.

Οι τοπικές αναζητήσεις γειτονιάς λαμβάνουν μια πιθανή λύση σε ένα πρόβλημα και ελέγχουν τους άμεσους γείτονές του (δηλαδή λύσεις που είναι παρόμοιες, εκτός από πολύ λίγες λεπτομέρειες) με την ελπίδα να βρεθεί μια βελτιωμένη λύση. Οι τοπικές μέθοδοι αναζήτησης έχουν την τάση να κολλάνε σε υποβέλτιστες περιοχές ή σε οροπέδια όπου πολλές λύσεις είναι εξίσου κατάλληλες.

Η αναζήτηση Tabu ενισχύει την απόδοση της τοπικής αναζήτησης χαλαρώνοντας τον βασικό της κανόνα. Κατ' αρχάς, σε κάθε βήμα μπορούν να γίνουν αποδεκτές κινήσεις επιδείνωσης εάν δεν υπάρχει διαθέσιμη βελτίωση (όπως όταν η αναζήτηση είναι κολλημένη σε αυστηρό τοπικό ελάχιστο). Επιπλέον, οι απαγορεύσεις (εφεξής ο όρος «tabu») εισάγονται για να αποθαρρύνουν την αναζήτηση από την επιστροφή στις λύσεις που είχαν προηγουμένως επισκεφθεί.

Η εφαρμογή της αναζήτησης tabu χρησιμοποιεί δομές μνήμης που περιγράφουν τις λύσεις επισκέψεων ή σύνολα κανόνων που παρέχονται από το χρήστη. Εάν μια πιθανή λύση έχει επισκεφθεί προηγουμένως μέσα σε μια συγκεκριμένη βραχυπρόθεσμη περίοδο ή εάν έχει παραβιάσει έναν κανόνα, επισημαίνεται ως «tabu» (απαγορευμένη), έτσι ώστε ο αλγόριθμος να μην εξετάζει επανειλημμένα αυτή την πιθανότητα (Wikipedia - Tabu search).

2.5 Βελτιστοποίηση αποικιών μυρμηγκιών

Στην επιστήμη των υπολογιστών και την έρευνα των λειτουργιών , ο αλγόριθμος βελτιστοποίησης αποικιών μυρμηγκιών (Ant Colony Optimization – ACO) είναι μια πιθανοτική τεχνική για την επίλυση υπολογιστικών προβλημάτων που μπορούν να περιοριστούν στην εύρεση καλών διαδρομών μέσω γραφημάτων. Τα τεχνητά μυρμήγκια αντιπροσωπεύουν μεθόδους πολλών παραγόντων εμπνευσμένες από τη συμπεριφορά των πραγματικών μυρμηγκιών. Η επικοινωνία των βιολογικών μυρμηγκιών με φερομόνες είναι συχνά το κυρίαρχο πρότυπο που χρησιμοποιείται. Οι συνδυασμοί τεχνητών μυρμηγκιών και αλγόριθμοι τοπικής αναζήτησης έχουν γίνει μια μέθοδος επιλογής για πολλές εργασίες βελτιστοποίησης που περιλαμβάνουν κάποιο είδος γραφήματος , π.χ. δρομολόγηση οχημάτων

και δρομολόγηση μέσω διαδικτύου. Η αυξανόμενη δραστηριότητα σε αυτόν τον τομέα έχει οδηγήσει σε συνέδρια αφιερωμένα αποκλειστικά σε τεχνητά μυρμήγκια και σε πολλές εμπορικές εφαρμογές από εξειδικευμένες εταιρείες όπως η (AntOptima).

Για παράδειγμα, η βελτιστοποίηση αποικιών των μυρμηγκιών είναι μια κλάση αλγορίθμων βελτιστοποίησης που σχεδιάστηκε με βάση τις δράσεις μιας αποικίας μυρμηγκιών . Τα τεχνητά «μυρμήγκια» (π.χ. παράγοντες προσομοίωσης) εντοπίζουν τις βέλτιστες λύσεις μετακινώντας ένα χώρο παραμέτρων που αντιπροσωπεύει όλες τις πιθανές λύσεις. Τα πραγματικά μυρμήγκια καθορίζουν τις φερομόνες που κατευθύνονται ο ένας στον άλλο σε πόρους ενώ εξερευνά το περιβάλλον τους. Τα προσομοιωμένα «μυρμήγκια» καταγράφουν ομοίως τις θέσεις τους και την ποιότητα των λύσεών τους, έτσι ώστε σε μεταγενέστερες προσομοιώσεις επαναλήψεις περισσότερα μυρμήγκια να βρουν καλύτερες λύσεις. Μια παραλλαγή αυτής της προσέγγισης είναι ο αλγόριθμος αποικίας των μελισσών (Artificial Bee Colony – ABC), ο οποίος είναι περισσότερο ανάλογος με τα μοτίβα τροφής της μέλισσας, ενός άλλου κοινωνικού εντόμου.

Αυτός ο αλγόριθμος είναι μέλος της οικογένειας αλγορίθμων αποικίας μυρμηγκιών, σε μεθόδους πληροφοριών για το σμήνος και αποτελεί μερικές μεταυρετικές βελτιστοποιήσεις. Αρχικά προτάθηκε από τον Marco Dorigo το 1992 με τη διδακτορική διατριβή του, ο πρώτος αλγόριθμος στοχεύει στην αναζήτηση μιας βέλτιστης διαδρομής σε ένα γράφημα, με βάση τη συμπεριφορά των μυρμηγκιών που αναζητούν μια διαδρομή μεταξύ της αποικίας τους και μιας πηγής τροφής . Η αρχική ιδέα έχει διαφοροποιηθεί από τότε για να επιλύσει μια ευρύτερη τάξη αριθμητικών προβλημάτων και ως εκ τούτου προέκυψαν διάφορα προβλήματα, βασιζόμενα σε διάφορες πτυχές της συμπεριφοράς των μυρμηγκιών. Από μια ευρύτερη προοπτική, η ACO εκτελεί μια αναζήτηση βασισμένη στο μοντέλο και μοιράζεται κάποιες ομοιότητες με την εκτίμηση των αλγορίθμων διανομής (Wikipedia - ACO).

3. Ο Αλγόριθμος Variable Neighborhood Search – VNS

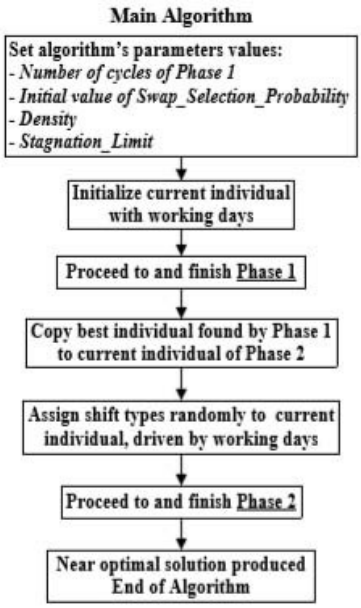
Σε αυτό το κεφάλαιο θα αναφερθούμε στον Αλγόριθμο Variable Neighborhood Search – VNS. Στο κεφάλαιο αυτό θα αναλυθεί η λειτουργία του καθώς και οι αρχές οι οποίες τον διέπουν. Στο τέλος, θα πραγματοποιηθεί σε μορφή ψευδοκώδικα παρουσίαση του αλγορίθμου και επίσης, θα γίνει αναφορά στις παραμέτρους του και στη ρύθμισή τους.

3.1 Η φιλοσοφία του αλγορίθμου VNS

Ο Αλγόριθμος Αναζήτησης Μεταβλητής Γειτονιάς (Variable Neighborhood Search) είναι ένας μεταευρετικός αλγόριθμος ο οποίος προτάθηκε από τους Nenad Mladenovic και Pierre Hansen τον Νοέμβριο του 1997 (Mladenovic & Hansen, 1997). Οι αλγόριθμοι αυτού του είδους είναι ικανοί για την επίλυση μιας σειράς συνδυαστικών και καθολικών προβλημάτων βελτιστοποίησης. Εξετάζει τις μακρινές γειτονιές της τρέχουσας ισχύουσας λύσης και μετακινείται από εκεί σε μια νέα, αν και μόνο εάν η νέα γειτονιά στην οποία θα μετακινηθεί έχει καλύτερα αποτελέσματα. Η τοπική μέθοδος αναζήτησης εφαρμόζεται επανειλημμένα για να πάρει από τις λύσεις στη γειτονιά την τοπική βέλτιστη. Ο αλγόριθμος Αναζήτησης Μεταβλητής Γειτονιάς (Variable Neighborhood Search – VNS) έχει σχεδιαστεί για την προσέγγιση λύσεων διακριτών και συνεχών προβλημάτων βελτιστοποίησης και σύμφωνα με αυτά, στοχεύει στην επίλυση προβλημάτων γραμμικού προγραμματισμού, προβλήματα ακέραιου προγραμματισμού, προβλήματα μικτού προγραμματισμούς αλλά και προβλήματα μη γραμμικού προγραμματισμού καθώς και πλήθος άλλων (Wikipedia - VNS). Οι μεταευρετικοί αλγόριθμοι αναζητούν τη βέλτιστη λύση ή μια λύση που να προσεγγίζει αρκετά ικανοποιητικά τη βέλτιστη, μέσα σε ένα μεγάλο χώρο υποψήφιων λύσεων. Συνήθως για την εύρεση της λύσης, κάνουν και κάποιες υποθέσεις. Η εύρεση της βέλτιστης λύσης από τους μεταυρετικούς αλγορίθμους δεν είναι εγγυημένη, αλλά η λύση η οποία θα βρεθεί, από τους περισσότερους από αυτούς, τείνει να είναι κοντά στη βέλτιστη και μάλιστα σε ελάχιστο χρόνο. Οι μεταευρετικοί αλγόριθμοι για την εύρεση της βέλτιστης λύσης κάνουν χρήση κάποιου είδους στοχαστικής (πιθανοτικής) βελτιστοποίησης, διαδικασία που μπορούμε να πούμε ότι βασίζεται και λίγο στην τύχη.

3.2 Παρουσίαση του αλγορίθμου VNS για επίλυση του προβλήματος Nurse Rostering

Ο αλγόριθμος «Στοχαστική Αναζήτηση Μεταβλητής Γειτονιάς Δύο Φάσεων για την αποδοτική επίλυση του προβλήματος του Nurse Rostering» που αφορά στην επίλυση του προβλήματος του Nurse Rostering, ο οποίος παρουσιάζεται στην παρούσα πτυχιακή εργασία ως μηχανή για την κατασκευή σχετικού λογισμικού για αλληλεπίδραση του χρήστη με τον αλγόριθμο, σχεδιάστηκε και παρουσιάστηκε από τους από τους κκ. Μπεληγιάννη Γρηγόρη, Τασσόπουλο Ιωάννη και Σώλο Ιωάννη (Beligiannis, Tasopoulos, & Solos, 2013). Πρόκειται για έναν στοχαστικό αλγόριθμο ο οποίος ολοκληρώνεται σε δύο φάσεις. Στην πρώτη γίνεται η αρχικοποίηση της υποψήφιας λύσης με τυχαίο τρόπο και με ικανοποίηση των ενεργών βαρδιών ανά ημέρα. Επίσης, γίνεται η αρχικοποίηση των διάφορων μεταβλητών του αλγορίθμου. Τέλος, με χρήση της συνάρτησης Swap_Random_Rosters() προσδιορίζεται το ποιοι νοσοκόμοι θα εργαστούν ανά ημέρα, ανεξαρτήτως shift type. Ο ορισμός και η λειτουργικότητα της Swap_Random_Rosters() περιγράφεται παρακάτω. Στην δεύτερη φάση, ο αλγόριθμος «βασίζεται» στις αναθέσεις εργασίας των νοσοκόμων στις διάφορες ημέρες και απλώς αναθέτει το είδος της βάρδιας σε κάθε νοσοκόμο. (Πανεπιστήμιο Πάτρας, σ. 16). Στην παρακάτω εικόνα φαίνεται σε επισκόπηση η λειτουργία του αλγορίθμου.



Εικόνα 1: Επισκόπηση του αλγορίθμου.

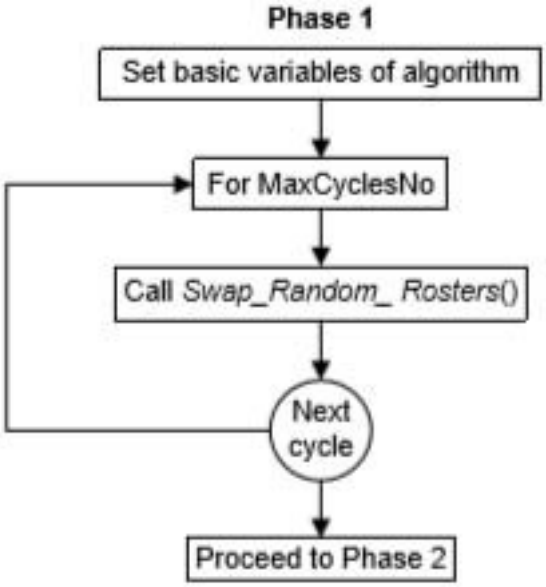
Στη συνέχεια, αναλύονται οι δύο φάσεις καθώς και η ρύθμιση των παραμέτρων του αλγορίθμου.

3.2.1. 1η Φάση

Κατά την έναρξη του αλγορίθμου πραγματοποιείται αρχικοποίηση των βασικών παραμέτρων του αλγορίθμου. Οι παράμετροι που αρχικοποιούνται είναι:

- Number of cycles of Phase 1 – Ορίζει τον αριθμό εκτελέσεων της 1ης φάσης.
- Initial value of Swap_Selection_Probability – Επιλογή στρατηγικής αναζήτησης.
- Density – Δυσκολία διαφορετικής περίπτωσης εισόδου.
- Stagnation_Limit – Ανοχή του προτεινόμενου αλγορίθμου στην παγίδευση σε τοπικά βέλτιστα.

Η υποψήφια λύση σε αυτό το σημείο είναι ακόμα κενή. Επόμενο βήμα είναι η τυχαία ανάθεση των νοσηλευτών σε κάθε εργάσιμη ημέρα σύμφωνα με τις απαιτήσεις σε πλήθος νοσηλευτών για κάθε εργάσιμη ημέρα. Σε όλους αυτούς εκχωρείται το σύμβολο «W» ενώ στους υπόλοιπους εκχωρείται ο αριθμός «-1» ο οποίος δηλώνει τη μη εργασία του νοσηλευτή την αντίστοιχη ημέρα.



Εικόνα 2: Η διαδικασία Swap_Random_Rosters()

1. Set values to the basic parameters of the algorithm, i.e.

$$Number\ of\ cycles\ of\ Phase\ 1 = 1$$

$$Initial\ value\ of\ Swap_Selection_Probability = 0.3$$

$$Density = \frac{\sum_{d=1}^{D}\sum_{s=1}^{S} nurses_required(d,s)}{Number_of_nurses * Number_of_days}$$

$$Stagnation_Limit = \frac{Number_of_nurses * Number_of_shift_types}{5}$$

2. Execute procedure *Swap_Random_Rosters*()

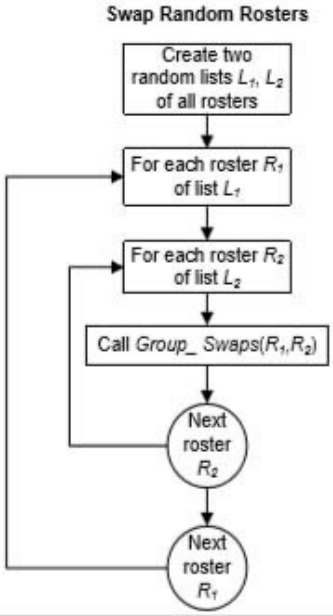
3. Construct first candidate solution (assign nurses to work)

Εικόνα 3: Ψευδοκώδικας της 1ης φάσης

Έπειτα αρχίζει να εκτελείται η διαδικασία Swap_Random_Rosters() τόσες φορές όσες ορίστηκε στη σχετική παράμετρο. Η λειτουργία της Swap_Random_Rosters() περιγράφεται με τα επόμενα βήματα.

1. Δημιουργούνται τυχαία δύο διαφορετικές λίστες L1 και L2 που αποτελούνται από όλους τους νοσοκόμους της υποψήφιας λύσης.
2. Για κάθε νοσοκόμο R1 στο L1 και για κάθε νοσοκόμο R2 στον L2, εκτελείται η διαδικασία Group_Swaps(R1,R2).

Ο ψευδοκώδικας και η δομή της Swap_Random_Rosters() φαίνεται στην παρακάτω εικόνα.



1. Create two random lists L_1 and L_2 of all rosters
2. For each roster R_1 of list L_1 do
    2.1 For each roster R_2 of list L_2 do
        2.1.1 Execute *Group_Swaps*(R_1, $R2$)
    2.2 End do loop
3. End do loop

Εικόνα 4: Δομή και ψευδοκώδικας της Swap_Random_Rosters()

Η διαδικασία Groups_Swaps(R_1, R_2) δέχεται σαν ορίσματα δύο ρόστερ που δημιουργήθηκαν από την διαδικασία Swap_Random_Rosters() και πραγματοποιεί δοκιμές βελτίωσής τους όπως περιγράφεται στα επόμενα βήματα.

1. Δημιουργείται μια τυχαία λίστα L1 που αποτελείται από όλες τις ημέρες της περιόδου που διαρκεί το πρόγραμμα εργασίας.
2. Για κάθε ημέρα D1 της L1
    2.1. Για κάθε μέρα D2 της L1 όπου D2 ≥ D1, δημιουργείται ένα πλαίσιο [D1, D2]
    2.2. Έχοντας την ημέρα D1 αριστερά και την ημέρα D2, από τα δεξιά.
    2.3. Στη συνέχεια, για κάθε ημέρα $D \in [D1, D2]$
    2.4. Γίνεται εναλλαγή μεταξύ των κελιών [R1][D] και [R2][D] με την πιθανότητα που ορίστηκε στην αντίστοιχη παράμετρο.

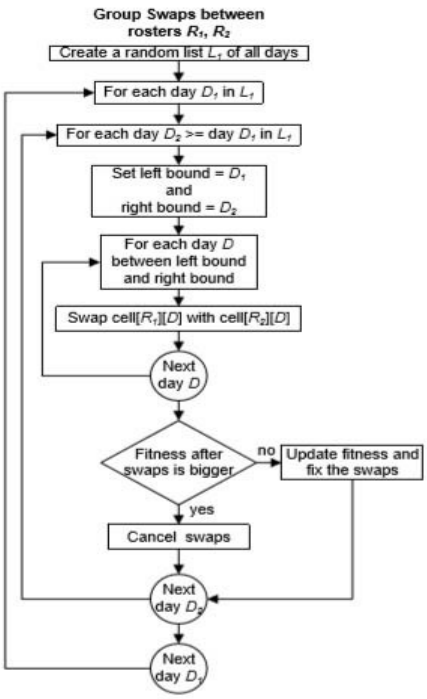
2.5. Υπολογίζεται η αξιολόγηση του χρονοδιαγράμματος των δύο γραμμών. Αν η τιμή είναι μεγαλύτερη σε σχέση με εκείνη πριν από τις αλλαγές, τότε όλες οι αλλαγές που πραγματοποιήθηκαν απορρίπτονται.

2.6. Αν η τιμή είναι μικρότερη ή ίση με την τιμή πριν από τις αλλαγές, τότε αυτές γίνονται αποδεκτές και ενημερώνεται η τιμή καταλληλότητας του προγράμματος.

3. Οι ενέργειες αυτές, πραγματοποιούνται μεταξύ της ημέρας D1 και όλων των άλλων ημερών D2 που είναι μέσα στη λίστα L1, όπου D2 ≥ D1, μέχρι να εξαντληθούν όλες οι ημέρες D2 στη λίστα L1, και ξεκινάει από την αρχή για κάθε διαφορετική ημέρα D1 της λίστας L1.

1. Create a random list *Days* of all days of the planning horizon
2. For each day D_1 in *Days* do
    2.1 For each day D_2 in *Days* with $D_2 >= D_1$ do
        2.1.1 Set left bound = D_1 and set right bound = D_2
            2.1.1.1 For each day *D* of *Days* that is between left and right bounds
                2.1.1.1.1 Swap the contents of cells [R_1, *D*] and [R_2, *D*] with a predefined probability
            2.1.1.2 End for
        2.1.2 If fitness after swaps is worse than before then cancel all swaps
        2.1.3 Else update fitness and fix the swaps
    2.2. End do loop
3. End do loop

Εικόνα 5: Ο ψευδοκώδικας της διαδικασίας Groups_Swaps(R1, R2)

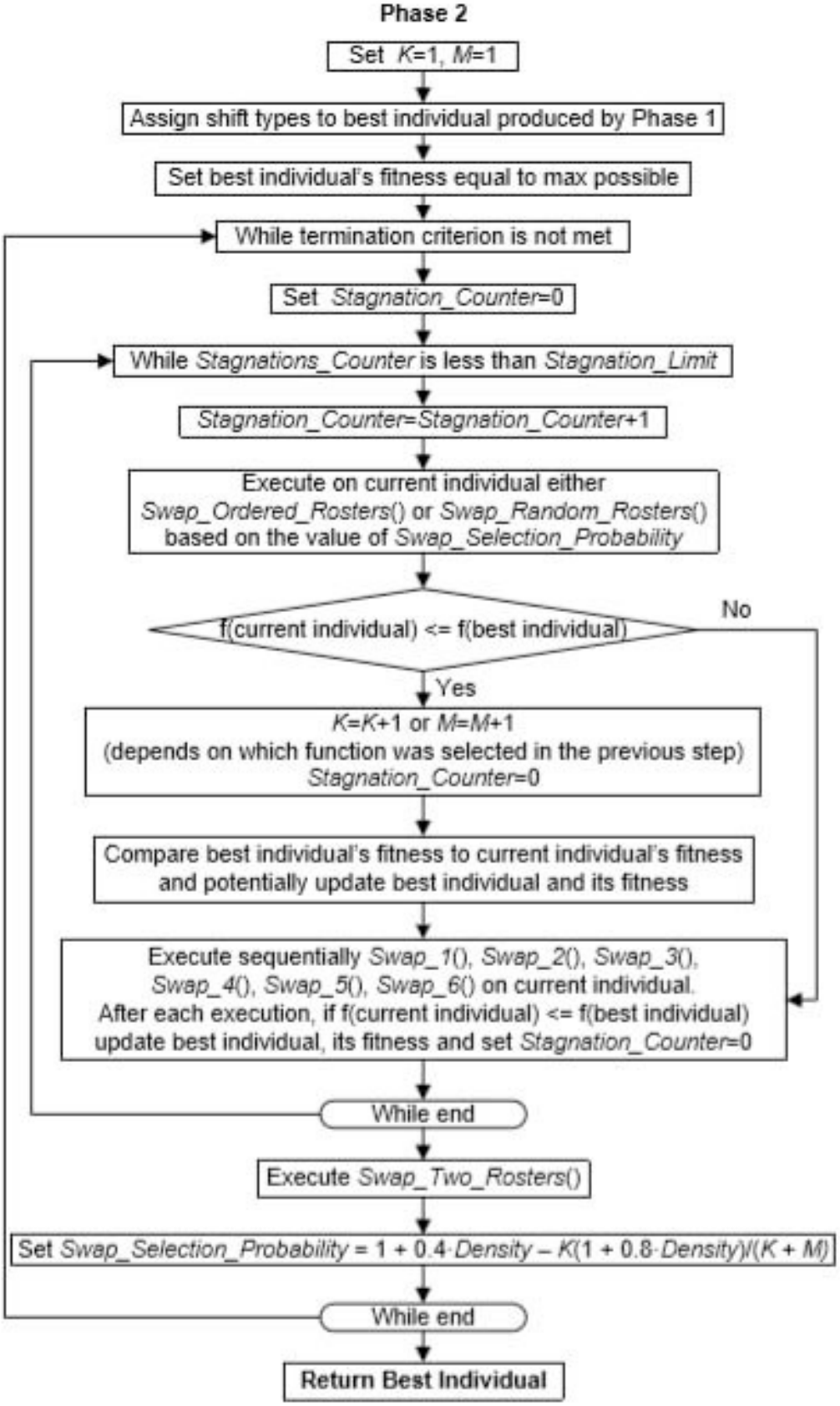


Εικόνα 6: Η δομή της διαδικασίας Groups_Swaps(R1, R2)

3.2.2. 2η Φάση

Στη 2η Φάση του αλγορίθμου πραγματοποιείται η ανάθεση των βαρδιών στους νοσηλευτές. Κατά το τέλος της 1ης φάσης, το καλύτερο πρόγραμμα που βρέθηκε, παραδίδεται στην 2η φάση για περεταίρω επεξεργασία. Στην αρχή της εκτέλεσης της 2ης φάσης, όπως ισχύει και στην 1η φάση, πραγματοποιούνται αρχικοποιήσεις κάποιων παραμέτρων του αλγορίθμου. Αυτές οι παράμετροι είναι:

- K – Μετρητής βελτιώσεων αλγορίθμου από τη διαδικασία Swap_Ordered_Roster()
- M – Μετρητής βελτιώσεων αλγορίθμου από τη διαδικασία Swap_Random_Rosters()
- Stagnations_Counter – Μέτρηση διαδοχικών εφαρμογών των διαδικασιών Swap_Ordered_Rosters(), Swap_Random_Rosters(), Swap_1(), Swap_2(), Swap_3(), Swap_4(), Swap_5() και Swap_6().



Εικόνα 7: Η δομή της 2ης Φάσης

Παρατηρώντας τη δομή της 2ης φάσης θα δούμε μια μεταβλητή με το όνομα Stagnation_Limit. Για όσο η τιμή της μεταβλητής Stagnations_Counter είναι μικρότερη από την τιμή της Stagnation_Limit εκτελείται είτε η διαδικασία Swap_Random_Rosters() είτε η διαδικασία Swap_Ordered_Roster(). Για το ποια από τις δύο διαδικασίες θα εκτελεστεί, αποφασίζεται από την πιθανοτική τιμή που θα εισάγουμε στον αλγόριθμο. Το εύρος αυτής της τιμής είναι [0, 1]. Αν η τιμή είναι μικρότερη ή ίση με την τιμή που υπολογίζεται κατά τη διάρκεια εκτέλεσης του αλγορίθμου για την μεταβλητή Swap_Selection_Probability τότε θα εκτελεστεί η διαδικασία Swap_Ordered_Roster() ενώ αν η τιμή είναι μεγαλύτερη από την τιμή της μεταβλητής Swap_Selection_Probability θα εκτελεστεί η διαδικασία Swap_Random_Rosters(). Η εκτέλεση των δύο αυτών διαδικασιών, οδηγεί τον αλγόριθμο σε διαφορετικές γειτονιές του χώρου αναζήτησης.

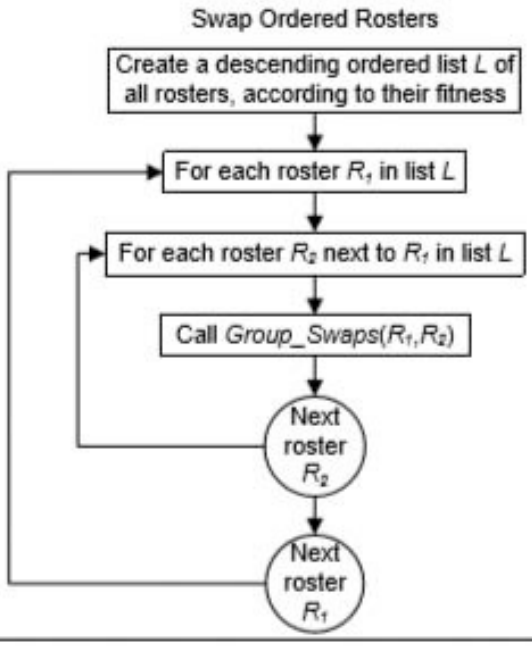
Η τιμή της Swap_Selection_Probability υπολογίζεται σε κάθε επανάληψη του αλγορίθμου με βάση τον τύπο:

$$Swap_{Selection_{Propability}} = 1 + 0.4 * Density - K * \frac{1 + 0.8 * Density}{K + M} \qquad (1)$$

Στην περίπτωση που η τιμή της Stagnations_Counter γίνει ίση με την τιμή της Stagnation_Limit και δεν έχει επιτευχθεί καμία βελτίωση του αποτελέσματος, τότε θεωρείται ότι ο αλγόριθμός έχει παγιδευτεί σε τοπικό βέλτιστο και καλεί για εκτέλεση την διαδικασία Swap_Two_Rosters(). Η διαδικασία αυτή βοηθάει τον αλγόριθμο να ξεφύγει από το τοπικό βέλτιστο στο οποίο εγκλωβίστηκε ο αλγόριθμος, οδηγώντας τον για αναζήτηση σε άλλη γειτονιά. Το πόσο έντονη θα είναι η ανατάραξη που θα προκαλέσει η διαδικασία Swap_Two_Rosters() είναι ένα θέμα προς συζήτηση διότι μια μεγάλη ανατάραξη μπορεί να θεωρηθεί επανεκκίνηση του αλγορίθμου ενώ μια μικρή μπορεί να μη βοηθήσει σημαντικά τον αλγόριθμο οδηγώντας τον στο ίδιο τοπικό βέλτιστο. Η λειτουργία της διαδικασίας Swap_Two_Rosters() περιγράφεται παρακάτω (Σώλος, σσ. 184 - 185):

1. Επιλέγει δύο νοσοκόμους (δύο γραμμές) R1 και R2 και μια μέρα (στήλη) D από το τρέχον πρόγραμμα τυχαία.
2. Εναλλάσσει τα περιεχόμενα του κελιού [R1] [D] με τα περιεχόμενα του κελιού [R2] [D].

Στη συνέχεια παρουσιάζεται η δομή και ο ψευδοκώδικας της διαδικασίας Swap_Ordered_Roster().



1. Create a descending ordered list L of all rosters, according to their fitness, i.e. rosters with smaller fitness follow rosters with bigger ones, from left to right.
2. for each roster R_1 of L do
    2.1 for each roster R_2 of L, to the right of R_1, do
        2.1.1 Execute *Group_Swaps*(R_1, R_2)
    2.2 End do
3. End do

Εικόνα 8: Δομή και ψευδοκώδικας της διαδικασίας Swap_Ordered_Roster()

Με την εκτέλεση των διαδικασιών Swap_1(), Swap_2(), Swap_3(), Swap_4(), Swap_5() και Swap_6() ο αλγόριθμος αναζητεί λύση στο πρόβλημα σε διαφορετικές γειτονιές του χώρου αναζήτησης, αν μετά την εκτέλεση των Swap_Ordered_Rosters() και Swap_Random_Rosters() η απόδοση του αλγορίθμου δεν βελτιώθηκε.

Τέλος, μετά από την εκτέλεση όλων των παραπάνω, ελέγχεται η τιμή της συνάρτησης αξιολόγησης της τρέχουσας λύσης. Αν αυτή είναι μικρότερη από την χαμηλότερη τιμή που βρέθηκε μέχρι εκείνη τη στιγμή, αποδεχόμαστε τη νέα καλύτερη λύση και θέτουμε τον μετρητή Stagnation_Counter = 0.

Οι λειτουργία των διαδικασιών Swap_1(), Swap_2(), Swap_3(), Swap_4(), Swap_5() και Swap_6() περιγράφεται παρακάτω:

- Swap_1(): Σε κάθε στήλη δοκιμάζει τις εναλλαγές μεταξύ κενών και μη κενών κελιών. Για κάθε στήλη αποδέχεται την εναλλαγή που οδηγεί σε χαμηλότερη τιμή της αντικειμενικής συνάρτησης. Για κάθε μέρα (στήλη) D του προγράμματος, πραγματοποιεί εναλλαγές ανάμεσα σε κελιά με κενό περιεχόμενο (που έχει τιμή «-

1») και κελιά με μη κενό περιεχόμενο που ανήκουν στη D για όλους τους διαφορετικούς συνδυασμούς τους και υπολογίζει την αντίστοιχη τιμή της αντικειμενικής συνάρτησης. Στη συνέχεια, επιλέγει να εκτελέσει την καλύτερη εναλλαγή για κάθε ημέρα (στήλη), δηλαδή, την εναλλαγή που έχει ως αποτέλεσμα τη χαμηλότερη τιμή της αντικειμενικής συνάρτησης για κάθε ημέρα (Σώλος, σσ. 180 - 181). Ο ψευδοκώδικας της Swap_1() έχει ως εξής:

```
1. For each day (column) D of current individual do
        1.1 For each empty cell EC of day D do
                1.1.1 For each non empty cell nEC of day D do
                        1.1.1.1 Swap the contents of cells EC and nEC
                        1.1.1.2 Record the swap and the fitness of current individual after the swap
                        1.1.1.3 Restore the swap
                1.1.2 End do
        1.2 End do
        1.3 Select the swap that corresponds to the best fitness among all above recorded swaps
                (step1.1.1.2) and apply it to current individual
2. End do
```

Εικόνα 9: Ο ψευδοκώδικας της Swap_1()

- Swap_2(): Σε κάθε στήλη δοκιμάζει τις εναλλαγές μεταξύ όλων των μη κενών κελιών της και καταγράφει τη βελτίωση που τυχόν επιτυγχάνεται στην τιμή της αντικειμενικής συνάρτησης. Τελικά, αποδέχεται για κάθε στήλη την καλύτερη εναλλαγή. Η συνάρτηση αυτή λειτουργεί παρόμοια με τη Swap_1(). Ωστόσο, όλες οι εναλλαγές που πραγματοποιούνται είναι αυστηρά μεταξύ κελιών που είναι μη κενά (Σώλος, σ. 181). Ο ψευδοκώδικας της Swap_2() έχει ως εξής:

```
1. For each day (column) D of current individual do
        1.1 For each non empty cell nEC1 of day D do
                1.1.1 For each non empty cell nEC2 of day D do
                        1.1.1.1 Swap the contents of cells nEC1 and nEC2
                        1.1.1.2 Record the swap and the fitness of current individual after the swap
                        1.1.1.3 Restore the swap
                1.1.2 End do
        1.2 End do
        1.3 Select the swap that corresponds to the best fitness among all above recorded swaps
                (step1.1.1.2) and apply it to current individual
2. End do
```

Εικόνα 10: Ο ψευδοκώδικας της Swap_2()

- Swap_3(): Σε κάθε στήλη δοκιμάζει τις εναλλαγές μεταξύ όλων των κελιών της, ανεξαρτήτως περιεχομένου και καταγράφει την πιθανή βελτίωση στην τιμή της αντικειμενικής συνάρτησης. Τελικά, αποδέχεται για κάθε στήλη την καλύτερη

εναλλαγή. Η συνάρτηση αυτή λειτουργεί παρόμοια με τη Swap_1(). Ωστόσο, όλες οι εναλλαγές πραγματοποιούνται τυχαία, δηλαδή, χωρίς να ελέγχεται εάν τα κελιά που εμπλέκονται είναι κενά ή όχι. Η συνάρτηση αναζήτησης γειτονιάς Swap_3() αποτελεί ένα υπερσύνολο των συναρτήσεων αναζήτησης γειτονιάς. Swap_2() και Swap_1() (Σώλος, σσ. 181 - 182). Ο ψευδοκώδικας της Swap_3() έχει ως εξής:

```
1. For each day (column) D of current individual do
        1.1 For each cell C1 of day D do
                1.1.1 For each cell C2 of day D do
                        1.1.1.1 Swap the contents of cells C1 and C2
                        1.1.1.2 Record the swap and the fitness of current individual after the swap
                        1.1.1.3 Restore the swap
                1.1.2 End do
        1.2 End do
        1.3 Select the swap that corresponds to the best fitness among all above recorded swaps
        (step1.1.1.2) and apply it to current individual
2. End do
```

Εικόνα 11: Ο ψευδοκώδικας της Swap_3()

- Swap_4(): Δοκιμάζει σε όλες τις στήλες τις εναλλαγές μεταξύ των κελιών τους, ανεξαρτήτως περιεχομένου και κρατάει τη μοναδική καλύτερη εναλλαγή από όλες τις στήλες. Για κάθε μέρα (στήλη) D του προγράμματος, εκτελεί όλες τις πιθανές εναλλαγές μεταξύ των κελιών που ανήκουν στη D χωρίς να ελέγχει, εάν τα κελιά που εμπλέκονται είναι κενά ή όχι και υπολογίζει τις αντίστοιχες τιμές της αντικειμενικής συνάρτησης. Στη συνέχεια, επιλέγει να εκτελέσει την καλύτερη εναλλαγή για όλες τις ημέρες, δηλαδή, την εναλλαγή εκείνη που πετυχαίνει τη χαμηλότερη τιμή της αντικειμενικής συνάρτησης σε όλες τις ημέρες του τρέχοντος προγράμματος. Αυτή είναι η διαφορά μεταξύ των συναρτήσεων Swap_4() και Swap_3(). Στη συνάρτηση Swap_3() επιλέγεται η εναλλαγή που αντιστοιχεί στην καλύτερη τιμής της αντικειμενικής συνάρτησης μεταξύ όλων των εναλλαγών που εκτελούνται στην τρέχουσα στήλη (ημέρα), ενώ στη συνάρτηση Swap_4() επιλέγεται η εναλλαγή που αντιστοιχεί στην καλύτερη τιμή της αντικειμενικής συνάρτησης μεταξύ όλων των εναλλαγών που εκτελούνται στην τρέχουσα λύση (σε όλες τις ημέρες) (Σώλος, σ. 182). Ο ψευδοκώδικας της συνάρτησης Swap_4() έχει ως εξής:

```
1. For each day (column) D of current individual do
        1.1 For each cell C1 of day D do
                1.1.1 For each cell C2 of day D do
                        1.1.1.1 Swap the contents of cells C1 and C2
                        1.1.1.2 Record the swap and the fitness of current individual after the swap
                        1.1.1.3 Restore the swap
                1.1.2 End do
        1.2 End do
2. End do
3. Select the swap that corresponds to the best fitness among all above recorded swaps (step1.1.1.2)
and apply it to current individual
```

Εικόνα 12: Ο ψευδοκώδικας της Swap_4()

- Swap_5(): Δοκιμάζει σε όλες τις στήλες τις εναλλαγές μεταξύ των κελιών τους, όπου το ένα είναι απαραίτητα κενό, και κρατάει τη μοναδική καλύτερη εναλλαγή από όλες τις στήλες. Η συνάρτηση αυτή λειτουργεί παρόμοια με τη συνάρτηση Swap_4(). Ωστόσο, όλες οι εναλλαγές πραγματοποιούνται αυστηρά μεταξύ κελιών με κενό περιεχόμενο και κελιών με μη κενό περιεχόμενο, για όλους τους διαφορετικούς συνδυασμούς των γραμμών για κάθε μέρα (στήλη) D. Στη συνέχεια, επιλέγει να εκτελέσει την καλύτερη εναλλαγή για όλες τις ημέρες, δηλαδή, την εναλλαγή που επιτυγχάνει την εναλλαγή με τη χαμηλότερη τιμή της αντικειμενικής συνάρτησης σε όλες τις ημέρες του τρέχοντος προγράμματος. Η διαφορά μεταξύ της συνάρτησης Swap_5() και της συνάρτησης Swap_1() είναι η εξής: η Swap_1() επιλέγει την εναλλαγή που αντιστοιχεί στην καλύτερη τιμή της αντικειμενικής συνάρτησης μεταξύ όλων των εναλλαγών που εκτελούνται στην τρέχουσα στήλη (ημέρα), ενώ η Swap_5() επιλέγει την εναλλαγή που αντιστοιχεί στην καλύτερη τιμή της αντικειμενικής συνάρτησης μεταξύ όλων των εναλλαγών που πραγματοποιούνται στην τρέχουσα λύση (όλες τις ημέρες) (Σώλος, σ. 183). Ο ψευδοκώδικας της συνάρτησης Swap_5() έχει ως εξής:

```
1. For each day (column) D of current individual do
        1.1 For each empty cell EC of day D do
                1.1.1 For each non empty cell nEC of day D do
                        1.1.1.1 Swap the contents of cells EC and nEC
                        1.1.1.2 Record the swap and the fitness of current individual after the swap
                        1.1.1.3 Restore the swap
                1.1.2 End do
        1.2 End do
2. End do
3. Select the swap that corresponds to the best fitness among all above recorded swaps (step1.1.1.2)
and apply it to current individual
```

Εικόνα 13: Ο ψευδοκώδικας της Swap_5()

- Swap_6(): Σε κάθε στήλη δοκιμάζει τις εναλλαγές μεταξύ όλων των κελιών της, όπου κανένα κελί δεν είναι κενό και καταγράφει τη βελτίωση στην τιμή της αντικειμενικής συνάρτησης. Τελικά, αποδέχεται τη μοναδική καλύτερη εναλλαγή μεταξύ όλων των στηλών. Σε κάθε μια στήλη συνδυάζει κάθε γραμμή με όλες τις άλλες αλλάζοντας τα περιεχόμενα των κελιών που έχουν περιεχόμενο μη κενό με κελιά που έχουν επίσης περιεχόμενο μη κενό. Η συνάρτηση αυτή λειτουργεί παρόμοια με τη συνάρτηση Swap_4(). Ωστόσο, οι αλλαγές γίνονται αυστηρά μεταξύ κελιών με περιεχόμενο μη κενό και κελιών με περιεχόμενο επίσης μη κενό, για κάθε ημέρα (στήλη) D. Η διαφορά μεταξύ των συναρτήσεων Swap_6() και Swap_2() είναι η εξής: η συνάρτηση Swap_2() επιλέγει την εναλλαγή που επιτυγχάνει την καλύτερη τιμή της αντικειμενικής συνάρτησης μεταξύ όλων των εναλλαγών που εκτελούνται στην τρέχουσα στήλη (ημέρα), ενώ η συνάρτηση Swap_6() επιλέγει την εναλλαγή που επιτυγχάνει την καλύτερη τιμή της αντικειμενικής συνάρτησης μεταξύ όλων των συνδυασμών εναλλαγών που πραγματοποιούνται στην τρέχουσα λύση (σε όλες τις ημέρες) (Σώλος, σσ. 183 - 184). Ο ψευδοκώδικας της συνάρτησης Swap_6() έχει ως εξής:

```
1. For each day (column) D of current individual do
        1.1 For each non empty cell nEC1 of day D do
                1.1.1 For each non empty cell nEC2 of day D do
                        1.1.1.1 Swap the contents of cells nEC1 and nEC2
                        1.1.1.2 Record the swap and the fitness of current individual after the swap
                        1.1.1.3 Restore the swap
                1.1.2 End do
        1.2 End do
2. End do
3 Select the swap that corresponds to the best fitness among all above recorded swaps (step1.1.1.2) and
apply it to current individual
```

Εικόνα 14: Ο ψευδοκώδικας της Swap_6()

4. Ωρολόγιο πρόγραμμα εργασίας σε νοσοκομειακά ιδρύματα

Στο παρακάτω κεφάλαιο θα παρουσιαστεί η διαδικασία κατασκευής ωρολογίων προγραμμάτων εργασίας του προσωπικού των νοσοκομειακών ιδρυμάτων. Παράλληλα θα οριστεί αυστηρά το πρόβλημα της εύρεσης του βέλτιστου ωρολογίου προγράμματος εργασίας σε νοσοκομειακά ιδρύματα, καθώς και το μαθηματικό μοντέλο, βάση του οποίου θα προκύψει η επίλυση του προβλήματος.

4.1 Η κατασκευή ωρολογίων προγραμμάτων στα νοσοκομειακά ιδρύματα

Σε κάθε ίδρυμα, οργανισμό, εταιρεία ή υπηρεσία (στο εξής «εργοδότης») που λειτουργεί με το σύστημα εργασίας των βαρδιών, το προσωπικό που εργάζεται, κατανέμεται στις αντίστοιχες βάρδιες που έχουν οριστεί από τον εργοδότη, σύμφωνα με τις ανάγκες του. Καμία βάρδια δεν είναι υποχρεωτικό να έχει το ίδιο πλήθος υπαλλήλων με όλες τις άλλες. Αφού ολοκληρωθεί η δημιουργία του πλάνου εργασίας από τον εργοδότη σύμφωνα με τις απαιτήσεις του αλλά και μερικούς νομικούς περιορισμούς, το επόμενο στάδιο είναι η ανάθεση του διαθέσιμου προσωπικού στις ημέρες και ώρες που έχουν οριστεί. Από αυτό το σημείο και έπειτα, αυξάνεται κατακόρυφα η δυσκολία κατασκευής του προγράμματος γιατί πρέπει να ληφθούν υπόψιν πάρα πολλές παράμετροι. Υπάρχουν νομικοί περιορισμοί, όπως για παράδειγμα, δεν μπορεί κάποιος υπάλληλος να δουλεύει παραπάνω από 8 ώρες ανά ημέρα. Επίσης δεν μπορεί να δουλεύει παραπάνω από 4 συνεχόμενες ημέρες χωρίς να έχει μια ημέρα ανάπαυσης. Άλλος περιορισμός είναι ότι κάποιος υπάλληλος που δουλεύει σήμερα βραδινή βάρδια, δεν επιτρέπεται να δουλέψει αύριο στην πρωινή βάρδια. Στη δημιουργία του προγράμματος θα πρέπει να ληφθούν υπόψιν και οι ιδιαιτερότητες του προσωπικού, καθώς θα υπάρχουν υπάλληλοι που δε θα θέλουν ή δε θα μπορούν να εργαστούν συγκεκριμένες ώρες ή ημέρες της εβδομάδας. Αφού τελικώς καταρτιστεί ένα πλάνο εργασίας (εβδομαδιαίο, δεκαπενθήμερο, μηνιαίο κτλ), σε καμία περίπτωση δεν θεωρείται ότι είναι σταθερό και αμετάβλητο. Αντιθέτως, θα βρίσκεται σε μια κατάσταση ενημερώσεων και αλλαγών διότι μπορεί να υπάρξουν προσλήψεις νέου προσωπικού, απολύσεις, ασθένειες και πολλά άλλα περιστατικά τα οποία μπορούν θα θέσουν εύκολα το πρόγραμμα εκτός πραγματικότητας.

Η κατασκευή του προγράμματος πραγματοποιείται είτε από τον διευθυντή κάποιας κλινικής ή τμήματος, είτε από τον προϊστάμενο. Σε μερικές περιπτώσεις, συνήθως όταν το προσωπικό είναι μικρός αριθμός, το πλάνο εργασίας μπορεί να δημιουργηθεί από τους ίδιους τους υπαλλήλους. Σε κάθε περίπτωση όμως, πρέπει να ληφθούν υπόψιν ορισμένοι σκληροί ή ανελαστικοί περιορισμοί οι οποίοι είναι περιορισμοί που δεν είναι δυνατόν να παραβιαστούν,

όπως για παράδειγμα ότι ένας υπάλληλος δεν μπορεί να δουλεύει σε δύο βάρδιες την ίδια μέρα ή ότι δεν μπορεί σε κάποιο τύπο βάρδιας να μην οριστεί προσωπικό. Η ικανοποίηση των ανελαστικών ή σκληρών περιορισμών είναι αυτή που καθιστά την υποψήφια λύση εφικτή ή όχι. Υπόψιν, όμως, πρέπει να λαμβάνονται και οι μαλακοί ή ελαστικοί περιορισμοί όπως για παράδειγμα ο μέγιστος και ο ελάχιστος αριθμός βαρδιών που πρέπει να εκτελέσουν στην χρονική περίοδο που διαρκεί το πρόγραμμα, τις συνεχόμενες ημέρες ανάπαυσης και αρκετούς ακόμα. Όταν αναφερόμαστε στους «ελαστικούς περιορισμούς» εννοούμε αυτούς τους περιορισμούς τους οποίους πρέπει να ικανοποιήσουμε κατά το δυνατόν περισσότερους, αν όχι όλους, κατά τη δημιουργία του προγράμματος για την εύρυθμη και ικανοποιητική λειτουργία του ιδρύματος ή της υπηρεσίας. Αυτό όμως δε σημαίνει ότι δεν μπορούν να παραβιαστούν κάποιοι, αν παραστεί κάποια ανάγκη ή πρόβλημα. Από το πόσοι ελαστικοί ή μαλακοί περιορισμοί θα ικανοποιηθούν, εξαρτάται και η ποιότητα της λύσης που θα δοθεί. Συνήθως, ο υπάλληλος ο οποίος είναι επιφορτισμένος με τη δημιουργία του προγράμματος, έχει στη διάθεσή του κάποιον ηλεκτρονικό υπολογιστή και πιθανότατα κάποιο σχετικό πρόγραμμα γιατί η πολυπλοκότητα του προβλήματος είναι πολύ μεγάλη, όπως παραπάνω αναφέρθηκε. Παρόλα αυτά, η δημιουργία του προγράμματος βαρδιών παραμένει μια δύσκολη και επίπονη διαδικασία καθώς υπάρχουν πάρα πολλές παράμετροι που πρέπει να ληφθούν υπόψιν. Και το πρόβλημα δυσκολεύει ακόμα πιο πολύ για τον αρμόδιο υπάλληλο, γιατί δεν απαλλάσσεται των καθηκόντων του με τη δημιουργία ενός αποδεκτού προγράμματος, καθώς θα πρέπει να το παρακολουθεί και να το ενημερώνει σύμφωνα με τις αλλαγές που συντελούνται κατά τη διάρκεια εκτέλεσής του. Παρακάτω, θα γίνει μια συνοπτική παρουσίαση μερικών προγραμμάτων που κυκλοφορούν στο εμπόριο, σχετικά με τον χρονοπρογραμματισμό προσωπικού σε νοσηλευτικά ιδρύματα.

4.2 Ο ορισμός του προβλήματος

Το πρόβλημα κατασκευής ωρολογίου προγράμματος εργασίας σε βάρδιες για τα νοσοκομειακά ιδρύματα περιλαμβάνει πολλές παραμέτρους και ιδιαιτερότητες και πρέπει να ικανοποιεί ένα αρκετά μεγάλο πλήθος από περιορισμούς που επιβάλλονται είτε από τον εργοδότη, όπως ορίστηκε παραπάνω, είτε από τους νοσοκόμους. Για την κατασκευή ενός αποδοτικού και αποδεκτού προγράμματος εργασίας σε βάρδιες δεν εμπλέκονται πολλές οντότητες. Οι εμπλεκόμενες οντότητες είναι οι νοσοκόμοι και οι βάρδιες. Ειδικότερα, οι νοσοκόμοι πρέπει να δουλεύουν σε κάποιες βάρδιες. Με μια πρώτη ανάγνωση του προβλήματος γίνεται αντιληπτό ότι δεν είναι δύσκολη η δημιουργία του προγράμματος εργασίας. Λαμβάνοντας μέρος στη διαδικασία και οι περιορισμοί οι οποίοι πρέπει να

ικανοποιηθούν, αν όχι όλοι, όσο περισσότεροι μπορούν, η πολυπλοκότητα του προβλήματος αυξάνεται εκθετικά ή ακόμα και παραγοντικά.

Για να ξεκινήσουμε τη δημιουργία του προγράμματος εργασίας για τους νοσοκόμους του νοσοκομειακού ιδρύματος, πρέπει να έχουμε στη διάθεσή μας όλα τα δεδομένα εισόδου. Με τον όρο «δεδομένα εισόδου» εννοούμε όλους τους ανελαστικούς και ελαστικούς περιορισμούς οι οποίοι μας παρέχουν όλες τις απαιτούμενες πληροφορίες για τη σχέση που διέπει τους νοσοκόμους με τη βάρδια που πρέπει να εκτελέσουν.

Παρακάτω γίνεται παρουσίαση όλων των ανελαστικών αλλά και ελαστικών περιορισμών που διέπουν το πρόβλημα του χρονοπρογραμματισμού εργασίας σε βάρδιες στα νοσοκομειακά ιδρύματα, όπως διατυπώθηκαν και χρησιμοποιήθηκαν στον διαγωνισμό INRC-2010 (NR Competition) σύμφωνα με το Πανεπιστήμιο του Νόττινγχαμ. Αυτοί είναι (Σώλος, 2016) (Πανεπιστήμιο Πάτρας):

4.2.1. Ανελαστικοί περιορισμοί (Hard Constraints)

H1: Κάλυψη ζήτησης τύπων βαρδιών – σε κάθε ημέρα του ορίζοντα προγραμματισμού πρέπει να παρέχεται ο απαραίτητος αριθμός βαρδιών από κάθε είδος βάρδιας.

H2: Μοναδικότητα βάρδιας-κάθε νοσοκόμος εργάζεται μια το πολύ βάρδια σε κάθε ημέρα.

4.2.2. Ελαστικοί περιορισμοί (Soft Constraints)

S1: Μέγιστο πλήθος αναθέσεων νοσοκόμων-σε κάθε νοσοκόμο ανατίθεται ένας μέγιστος αριθμός βαρδιών, σύμφωνα με την σύμβασή του.

S2: Ελάχιστο πλήθος αναθέσεων βαρδιών νοσοκόμων-σε κάθε νοσοκόμο ανατίθεται ένας μέγιστος αριθμός βαρδιών, σύμφωνα με την σύμβασή του.

S3: Μέγιστο πλήθος συνεχόμενων ημερών εργασίας νοσοκόμου.

S4: Ελάχιστος πλήθος συνεχόμενων ημερών εργασίας νοσοκόμου.

S5: Μέγιστο πλήθος συνεχόμενων ημερών ανάπαυσης νοσοκόμου.

S6: Ελάχιστο πλήθος συνεχόμενων ημερών ανάπαυσης νοσοκόμου.

S7: Μέγιστο πλήθος συνεχόμενων Σαββατοκύριακων με τουλάχιστον μια βάρδια σε αυτά.

S8: Μέγιστο πλήθος Σαββατοκύριακων με τουλάχιστον μια βάρδια σε αυτά.

S9: Δύο ημέρες ανάπαυσης μετά από κάθε νυχτερινή βάρδια.

S10: Πλήρη Σαββατοκύριακο – αν ένας νοσοκόμος εργάζεται τουλάχιστον μια βάρδια σε ένα Σαββατοκύριακο τότε πρέπει να έχει βάρδια σε όλες τις ημέρες του Σαββατοκύριακου.

S11: Πανομοιότυπες βάρδιες Σαββατοκύριακου – κάθε νοσοκόμος που εργάζεται σε ένα Σαββατοκύριακο πρέπει να έχει το ίδιο είδος βάρδιας σε όλες τις ημέρες του Σαββατοκύριακου.

S12: Ανεπιθύμητα πρότυπα πρέπει να αποφεύγονται. ο νοσοκόμος δε θα πρέπει να εργαστεί για ένα συγκεκριμένο ανεπιθύμητο πρότυπο (ένα ανεπιθύμητο πρότυπο είναι μια σειρά από βάρδιες που δεν είναι στις προτιμήσεις του νοσοκόμου). Σύμφωνα με τη σύμβασή του κάθε νοσοκόμου υπάρχουν ανεπιθύμητα πρότυπα που αφορούν συγκεκριμένους τύπους βαρδιών.

S13: Προτίμηση για ημέρα εργασίας. Αιτήσεις από τους νοσηλευτές να εργαστούν σε συγκεκριμένες ημέρες της εβδομάδας θα πρέπει να τηρούνται, αλλιώς εμφανίζεται μια ποινή.

S14: Προτίμηση για ημέρα ανάπαυσης. Πρέπει να τηρούνται αιτήσεις από τους νοσηλευτές να μην εργάζονται σε συγκεκριμένες ημέρες της εβδομάδας, αλλιώς εμφανίζεται μια ποινή.

S15: Προτίμηση για εργασία συγκεκριμένου τύπου βάρδιας. Θα πρέπει να τηρούνται αιτήσεις από τους νοσηλευτές να εργάζονται συγκεκριμένες βάρδιες σε ορισμένες ημέρες της εβδομάδας, διαφορετικά εμφανίζεται μια ποινή.

S16: Προτίμηση για αποφυγή τύπου βάρδιας σε ημέρα εργασίας. Αιτήσεις από τους νοσηλευτές να μην εργάζονται συγκεκριμένες βάρδιες σε ορισμένες ημέρες της εβδομάδας πρέπει να γίνεται σεβαστή, διαφορετικά η λύση τιμωρείται αναλόγως.

S17: Βάρδια με εναλλακτικά προσόντα. εάν ο νοσοκόμος έχει τοποθετηθεί να εργαστεί σε έναν συγκεκριμένο τύπο που απαιτεί μια ιδιαίτερη ικανότητα που αυτός δεν έχει, τότε εμφανίζεται μια ποινή.

S18: Όχι νυχτερινή βάρδια πριν από ένα Σαββατοκύριακο με ανάπαυση. Νυχτερινή βάρδια την Παρασκευή δεν επιτρέπεται πριν από ένα ελεύθερο Σαββατοκύριακο. Να σημειωθεί ότι αν και ο περιορισμός αυτός δεν αναφέρεται στο κείμενο του Διαγωνισμού, λαμβάνεται υπόψιν του αλγορίθμου διότι περιλαμβάνεται σε πολλές συμβάσεις.

4.3 Το μαθηματικό μοντέλο του προβλήματος

Για τους ανωτέρω ανελαστικούς και ελαστικούς περιορισμούς καθώς και όλα τα απαραίτητα δεδομένα για την επίλυση του προβλήματος, το μαθηματικό μοντέλο είναι (Σώλος, σσ. 164-170):

4.3.1. Ανελαστικοί περιορισμοί

- **N** είναι το σύνολο των νοσηλευτών στους οποίους γίνεται η ανάθεση βαρδιών – κάθε νοσοκόμος έχει κάποιες συγκεκριμένες δεξιότητες και λειτουργεί κάτω από ένα συγκεκριμένο συμβόλαιο.
- **D** είναι το σύνολο των ημερών κατά τις οποίες οι νοσηλευτές πρέπει να χρονοπρογραμματιστούν.
- **S** είναι το σύνολο των διαφορετικών τύπων βάρδιας – κάθε τύπος βάρδιας χαρακτηρίζεται από ένα σύνολο απαιτούμενων δεξιοτήτων.
- **C** είναι το σύνολο των συμβάσεων – κάθε σύμβαση περιγράφει μια σειρά από ρυθμίσεις που θα πρέπει να ακολουθούνται από όλους τους νοσηλευτές που εργάζονται στο πλαίσιο της παρούσας σύμβασης.
- **P** είναι το σύνολο των προτύπων – Κάθε πρότυπο περιλαμβάνει μια σειρά από βάρδιες που κάποιος νοσοκόμος δε θα ήθελε να εργαστεί συνεχόμενα.

Η μαθηματική διατύπωση του πρώτου ανελαστικού περιορισμού H1 είναι η παρακάτω:

$$\sum_{n=1}^{N} \chi(r_{n,d} = s) = nurses_required(d,s), \forall d \in D, s \in S \qquad (2)$$

Όπου:

- nurses_required(d,s): Το πλήθος των νοσηλευτών που απαιτούνται σε μια ημέρα d και τον τύπο της βάρδιας s
- χ(): Συνάρτηση η οποία εξάγει το αποτέλεσμα περί ικανοποίησης του ανελαστικού περιορισμού H1. Στην περίπτωση που ικανοποιείται ο ανελαστικός περιορισμός H1 τότε η επιστρεφόμενη τιμή της συνάρτησης είναι 1 ενώ στην αντίθετη περίπτωση είναι 0. Στον αλγόριθμο τον οποίο θα χρησιμοποιήσουμε στην παρούσα πτυχιακή εργασία, ο ανελαστικός περιορισμός H1 πραγματοποιείται κατά την αρχικοποίηση των ατόμων σαν προεπιλογή, οπότε ο έλεγχος περί ικανοποιησιμότητάς του είναι περιττός.

Η κάθε υποψήφια λύση αναπαρίσταται από έναν $|N|\ x\ |D|$ πίνακα R όπου κάθε κελί r_{ij} αναπαριστά τον τύπο της βάρδιας που θα εργαστεί ο νοσοκόμος n_i κατά την ημέρα d_j. Ο πίνακας R της κάθε υποψήφιας λύσης δομείται με βάση τη συμπλήρωση της κάθε ημέρας εργασίας από την αρχή προς το τέλος της περιόδου προγραμματισμού που έχει οριστεί και όχι με βάση τη σειρά των νοσοκόμων. Έτσι εξασφαλίζεται και η ικανοποίηση του ανελαστικού περιορισμού Η2, δηλαδή ότι δεν θα εργαστεί κάποιος νοσοκόμος σε δύο βάρδιες την ίδια ημέρα. Η μορφή του πίνακα των υποψηφίων λύσεων φαίνεται παρακάτω:

$$\begin{array}{cc} & \overbrace{\begin{array}{ccccc} \text{Ημέρες προγράμματος} & d1 & d\ldots & dj & d\ldots & dD \\ \text{Νοσοκόμοι ημέρας} & 6 & \ldots & 4 & \ldots & 4 \end{array}}^{D\ days} \\ N\ \text{νοσοκόμοι} & \begin{cases} \text{νοσοκόμος } 1 \\ \text{νοσοκόμος } 2 \\ \text{νοσοκόμος } 3 \\ \text{νοσοκόμος } 4 \\ \text{νοσοκόμος } 5 \end{cases} \begin{bmatrix} r_{11} & \cdots & r_{1d_j} & \cdots & r_{1D} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ r_{n_i1} & \cdots & r_{n_id_j} & \cdots & r_{n_iD} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ r_{N1} & \cdots & r_{Nd_j} & \cdots & r_{ND} \end{bmatrix} \end{array}$$

Πίνακας 1: Μορφή πίνακα R που αναπαριστά τις υποψήφιες λύσεις.

4.3.2. Ελαστικοί περιορισμοί

Η μαθηματική έκφραση του κόστους καθενός εκ των 18 ελαστικών περιορισμών είναι:

$$\max\left(\sum_{d\in D}\chi(r_{n,d}\neq -1)-\max_shifts(n),0\right),\forall n\in N,(\boldsymbol{S1}) \quad (3)$$

Όπου

- max_shifts(n) το μέγιστο πλήθος βαρδιών που μπορεί να πραγματοποιήσει ο νοσοκόμος n

$$\max\left(min_shifts(n)-\sum_{d\in D}\chi(r_{n,d}\neq -1),0\right),\forall n\in N,(\boldsymbol{S2}) \quad (4)$$

Όπου

- min_shifts(n) το ελάχιστο πλήθος βαρδιών που μπορεί να πραγματοποιήσει ο νοσοκόμος n

$$\sum_{i\in I_n}\max(working_section_lenght(n,i)-\text{max_working_days(n)},0),\forall \text{n}\in \text{N},(\boldsymbol{S3}) \quad (5)$$

Όπου

- I_n είναι το σύνολο των τμημάτων της εργασίας του νοσοκόμου n (ένα τμήμα εργασίας είναι μια σειρά διαδοχικών εργάσιμων ημερών),
- working_section_length(n,i) είναι το μήκος της i-οστής ενότητας του νοσοκόμου n

- max_working_days(n) είναι ο μέγιστος αριθμός διαδοχικών ημερών κατά τις οποίες ο νοσοκόμος επιτρέπεται να έχει μια βάρδια (να εργάζεται).

$$\sum_{i \in I_n} \max(\mathrm{m}in_\mathrm{working_days}(\mathrm{n}) - working_section_lenght(n,i), 0), \forall \mathrm{n} \in \mathrm{N}, (\mathbf{S4}) \quad (6)$$

Όπου

- min_working_days(n) είναι ο ελάχιστος αριθμός διαδοχικών ημερών κατά τις οποίες ο νοσοκόμος επιτρέπεται να έχει μια βάρδια (να εργάζεται).

$$\sum_{j \in J_n} \max(free_section_lenght(n,j) - \mathrm{max_}free_\mathrm{days}(\mathrm{n}), 0), \forall \mathrm{n} \in \mathrm{N}, (\mathbf{S5}) \quad (7)$$

Όπου

- J_n είναι το σύνολο των ελεύθερων τμημάτων του νοσοκόμου n (ένα ελεύθερο τμήμα είναι μια σειρά διαδοχικών ελεύθερων ημερών),
- free_section_length(n,j) είναι το μήκος του j-οστού ελεύθερου τμήματος του νοσοκόμου n
- max_free_days(n) είναι ο μέγιστος αριθμός διαδοχικών ημερών κατά τις οποίες ο νοσοκόμος n επιτρέπεται να μην έχει βάρδια.

$$\sum_{j \in J_n} \max(\mathrm{min_free_days}(n) - free_section_\mathrm{lenght}(\mathrm{n,j}), 0), \forall \mathrm{n} \in \mathrm{N}, (\mathbf{S6}) \quad (8)$$

Όπου

- min_free_days(n) είναι ο ελάχιστος αριθμός διαδοχικών ημερών κατά τις οποίες ο νοσοκόμος n επιτρέπεται να μην έχει βάρδια

$$\sum_{i \in W} \max(weekend_section_lenght(n,i) - consecutive_working_weekends(n), 0),$$
$$\forall n \in N, (\boldsymbol{S7}) \quad (9)$$

Όπου

- W είναι το σύνολο των Σαββατοκύριακων κατά την περίοδο χρονοπρογραμματισμού.
- working_section_length(n,i) είναι το μήκος του Σαββατοκύριακου i του νοσοκόμου n.

- consecutive_working_weekends(n) είναι ο μέγιστος αριθμός διαδοχικών Σαββατοκύριακων που επιτρέπεται να εργάζεται ο νοσοκόμος n.

$$\max\left(\sum_{i\in W}\chi(weekend_working_days(n,i)>0)-working_weekends(n),0\right),\forall n\in N,(\boldsymbol{S8})\quad(10)$$

Όπου

- weekend_working_days(n,i) είναι ο αριθμός των εργάσιμων ημερών από τον νοσοκόμο.
- n το Σαββατοκύριακο.
- working_weekends(n) είναι ο μέγιστος αριθμός των Σαββατοκύριακων του νοσοκόμου n που επιτρέπεται να εργαστεί.

$$\sum_{i=1}^{D-2}\chi\Big(r_{n,i}=night\wedge\big((r_{n,i+1}\neq night\wedge r_{n,i+1}\neq -1)\vee(r_{n,i+2}\neq night\wedge r_{n,i+2}\neq -1)\big)\Big)+$$
$$+\chi(r_{n,D-1}=night\wedge r_{n,D}\neq\ night\wedge r_{n,D}\neq -1),\qquad(\boldsymbol{S9})\quad(11)$$

$$\sum_{i\in W}weekend_days(n)-weekend_working_days(n,i),\forall n\in N\ \kappa\alpha\iota$$
$$0<weekend_workind_days(n,i)\leq weekend_days(n),(\boldsymbol{S10})\quad(12)$$

Όπου

- weekend_days(n) είναι ο συνολικός αριθμός των ημερών για κάθε Σαββατοκύριακο του νοσοκόμου n.

$$\sum_{i\in W}Identical_Weekend_Cost(n,i),\forall n\in N\quad(13)$$

Όπου

$$Identical_Weekend_Cost(n,i)=\sum_{s\in S}(weekend_days(n)-weekend_shift_types(n,i,s))$$

Εάν $weekend_working_days(n,i)=weekend_days(n)\ \kappa\alpha\iota\ 0\ \delta\iota\alpha\varphi o\rho\varepsilon\tau\iota\kappa\acute{\alpha},\forall n\in N,s\in S,i\in W\ (\boldsymbol{S11})\kappa\alpha\iota\ weekend_shift_types(n,i,s)>0,\quad(14)$

$\acute{o}\pi o\upsilon\ weekend_shift_types(n,i,s)\ \varepsilon\acute{\iota}\nu\alpha\iota\ o\ \sigma\upsilon\nu o\lambda\iota\kappa\acute{o}\varsigma\ \alpha\rho\iota\theta\mu\acute{o}\varsigma\ \tau\omega\nu\ \beta\alpha\rho\delta\iota\acute{\omega}\nu\ \tau o\upsilon\ \tau\acute{\upsilon}\pi o\upsilon\ S\ \tau o\upsilon\ \nu o\sigma\eta\lambda\varepsilon\upsilon\tau\acute{\eta}\ n\ \pi o\upsilon\ \varepsilon\rho\gamma\acute{\alpha}\zeta\varepsilon\tau\alpha\iota\ \sigma\tau o\iota-o\sigma\tau\acute{o}\ \Sigma\alpha\beta\beta\alpha\tau o\kappa\acute{\upsilon}\rho\iota\alpha\kappa o.$

$$\sum_{p\in U_n}number_of_patterns(n,p),\forall n\in N,(\boldsymbol{S12})\quad(15)$$

Όπου

- Un είναι το σύνολο των ανεπιθύμητων προτύπων του νοσοκόμου n
- number_of_patterns(n,p) είναι ο αριθμός των φορών που κάθε ανεπιθύμητο πρότυπο p συμβαίνει για το νοσοκόμο n.

$$\sum_{d \in D} \chi\left(working_day_request(n,d) = yes \wedge r_{n,d} = -1\right), \forall n \in N, (\boldsymbol{S13}) \quad (16)$$

Όπου

- working_day_request(n,d) ισούται με «yes» αν ο νοσοκόμος n έχει ζητήσει να εργαστεί την ημέρα d.

$$\sum_{d \in D} \chi\left(working_day_request(n,d) = no \wedge r_{n,d} \neq -1\right), \forall n \in N, (\boldsymbol{S14}) \quad (17)$$

Όπου

- working_day_request(n,d) ισούται με «no» αν ο νοσοκόμος n έχει ζητήσει να μην εργαστεί την ημέρα d.

$$\sum_{d \in D} \sum_{s \in S} \chi\left(working_shift_day_request(n,d,s) = yes \wedge r_{n,d} \neq s\right), \forall n \in N, (\boldsymbol{S15}) \quad (18)$$

Όπου

- working_shift_day_request(n,d,s) ισούται με «yes» αν ο νοσοκόμος n ζήτησε να εργαστεί τη βάρδια s την ημέρα d.

$$\sum_{d \in D} \sum_{s \in S} \chi\left(working_shift_day_request(n,d,s) = no \wedge r_{n,d} \neq s\right), \forall n \in N, (\boldsymbol{S16}) \quad (19)$$

Όπου

- working_shift_day_request(n,d,s) ισούται με «no» αν ο νοσοκόμος n ζήτησε να μην εργαστεί τη βάρδια s την ημέρα d.

$$\sum_{d \in D} \chi\left(r_{n,d} \neq -1 \wedge skills_{required(n,r_{n,d})} = false\right), \forall n \in N, (\boldsymbol{S17}) \quad (20)$$

Όπου

- skills_required(n,rn,d) είναι αληθής αν ο νοσοκόμος n έχει όλες τις δεξιότητες που απαιτούνται για τη βάρδια, $r_{n,d}$.

$$\sum_{i \in W} no_night_before_weekend_cost(n,i), \forall n \in N, (\boldsymbol{S18}) \quad (21)$$

Όπου

$$no_night_before_weekend_cost(n,i) = \\ = \begin{cases} \sum_{\forall d=Friday} \chi(r_{n,d} = night \wedge r_{n,d+1} = -1 \wedge r_{n,d+2} = -1) & if\ weekend_days(n) = 2 \\ \sum_{\forall d=Thursday} \chi(r_{n,d} = night \wedge r_{n,d+1} = -1 \wedge r_{n,d+2} = -1 \wedge r_{n,d+3} = -1) & if\ weekend_days(n) = 3 \end{cases} \quad (22)$$

Έτσι, με βάση όλους τους παραπάνω μοντελοποιημένους ανελαστικούς αλλά και ελαστικούς περιορισμούς, το μοντέλο του προβλήματός μας, μπορεί να εκφραστεί ως εξής:

$$\min(w_{H1} * cost_H1 + \sum_{n=1}^{N} \sum_{i=1}^{18} w_{n,i} * cost_S_{\iota}) \quad (23)$$

Όπου:

- w_{H1} είναι το βάρος που σχετίζεται με το ανελαστικό περιορισμό H1 και $w_{n,i}$ είναι το βάρος που σχετίζεται με τους ελαστικούς περιορισμούς S_i για κάθε νοσοκόμο n. Επίσης, το w_{H1} και τα $w_{n,i}$, θα μπορούσαν να ήταν ίσα με 0, εάν οι αντίστοιχοι περιορισμοί δε χρησιμοποιούνται.

5. Σχεδίαση

5.1 Διάγραμμα Περιπτώσεων Χρήσης

Στο σύστημα διαχείρισης βαρδιών νοσηλευτικού προσωπικού υπάρχει ένας χειριστής, έστω κάποιος υπάλληλος της γραμματείας (στο εξής «Γραμματεία»), ο οποίος είναι επιφορτισμένος με τη δημιουργία του προγράμματος εργασίας των νοσηλευτών. Κατά την παραπάνω ανάπτυξη της λειτουργίας του αλγορίθμου, είδαμε ότι υπάρχουν απαιτήσεις για τις βάρδιες από τη διοίκηση αλλά και από το προσωπικό του νοσοκομειακού ιδρύματος, σχετικά με την κατάρτιση του προγράμματος. Τις απαιτήσεις αυτές, αλλά και πολλές άλλες ενέργειες (διαχείριση προσωπικού, βαρδιών κλπ), καλείται να τις διαχειριστεί η Γραμματεία του ιδρύματος, μέσα από μια εφαρμογή, ως διάφορες περιπτώσεις χρήσης αυτής της εφαρμογής.
Οι στόχοι της Γραμματείας είναι:

ΠΧ1. Να διαχειριστεί το προσωπικό. Να μπορεί δηλαδή να εισάγει νέο προσωπικό στο δυναμολόγιο του ιδρύματος, να τροποποιεί τα στοιχεία του και να εισάγει τις προτιμήσεις εργασίας του στο σύστημα.

ΠΧ2. Να διαχειριστεί τις βάρδιες. Να μπορεί δηλαδή να προσθέσει ή να αφαιρέσει κάποια βάρδια ή να την τροποποιήσει.

ΠΧ3. Να δημιουργήσει συμβόλαια εργασίας τα οποία θα περιγράφουν τις σχετικές απαιτήσεις του ιδρύματος αλλά και του νοσοκόμου ή των νοσοκόμων που θα σχετίζονται με το συμβόλαιο αυτό.

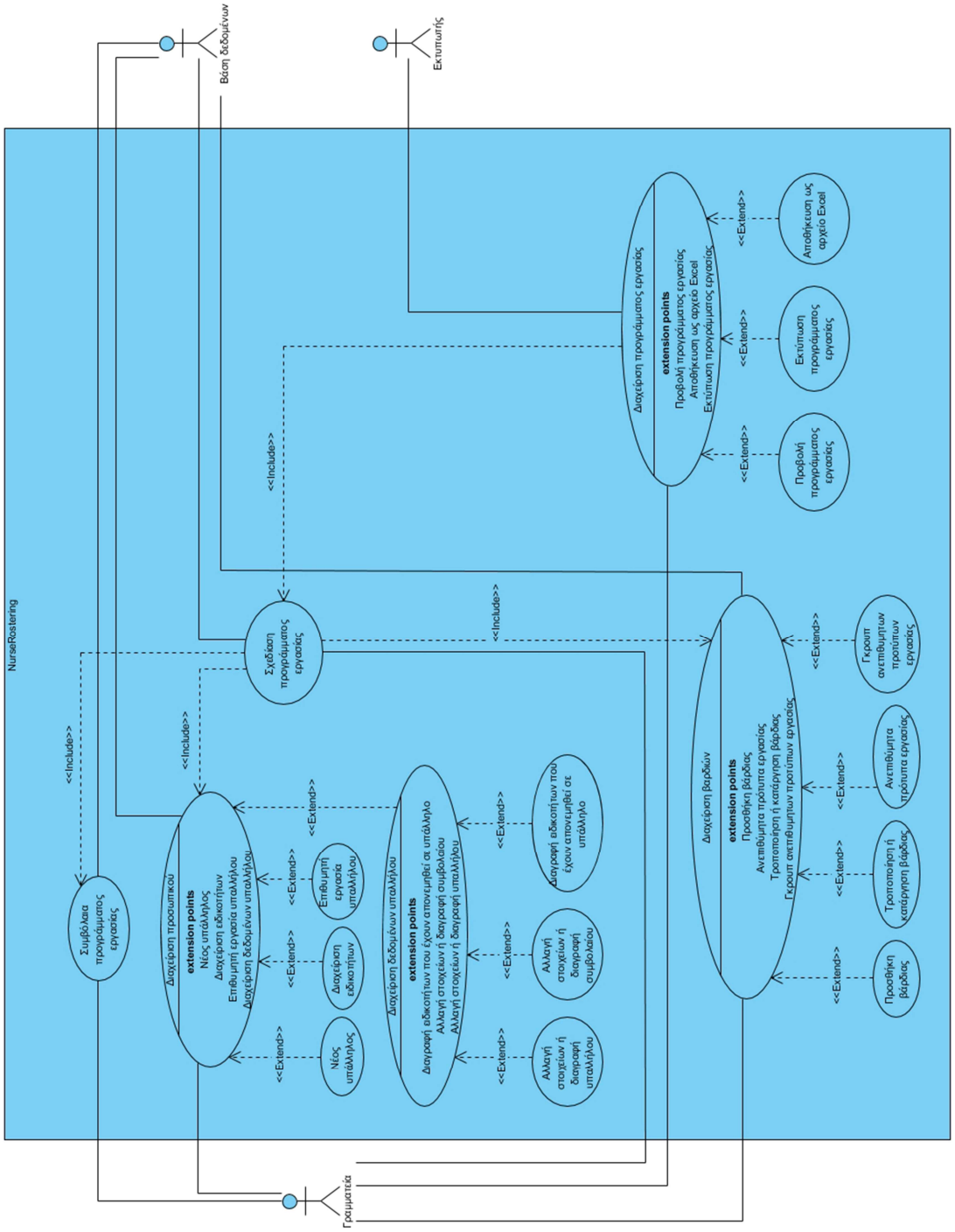
ΠΧ4. Να εκκινήσει τη διαδικασία δημιουργίας ενός προγράμματος εργασίας.

ΠΧ5. Να μπορεί να διαχειριστεί περεταίρω το πρόγραμμα εργασίας που δημιουργήθηκε. Να μπορεί, για παράδειγμα να εξαγάγει το πρόγραμμα εργασίας σε μορφή αρχείου Excel (.xls), να το εκτυπώσει αλλά και να το απεικονίσει γραφικά.

Στους δευτερεύοντες χειριστές, βλέπουμε ότι υπάρχει μια βάση δεδομένων και ένας εκτυπωτής.

Στη βάση δεδομένων αποθηκεύονται όλα τα απαιτούμενα δεδομένα των υπαλλήλων, τα συμβόλαια, τα σχήματα των βαρδιών, οι επιθυμίες εργασίας των υπαλλήλων, οι περιορισμοί που επιβάλλονται γενικώς αλλά και τα προγράμματα εργασίας που δημιουργήθηκαν.

Ο εκτυπωτής αναλαμβάνει να δημιουργήσει ένα έντυπο του προγράμματος εργασίας, για οποιαδήποτε επιθυμητή χρήση του.



Εικόνα 15: Διάγραμμα περιπτώσεων χρήσης του προγράμματος NurseRostering

Σχολιασμός

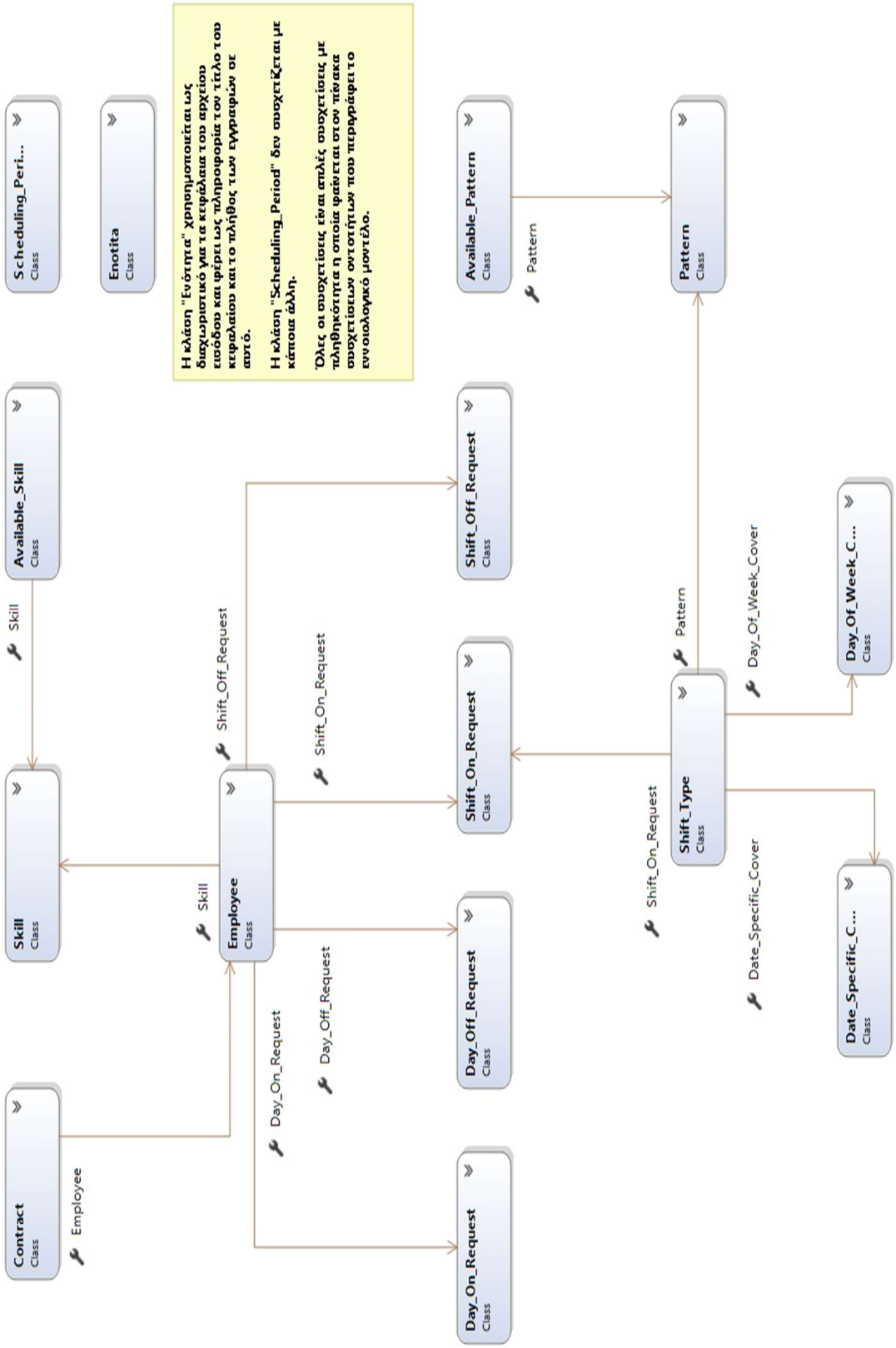
- Το ανωτέρω σύστημα, έχει ένα κύριο χειριστή, τη *Γραμματεία* και δύο δευτερεύοντες οι οποίοι είναι ένα σύστημα *Βάσης Δεδομένων* στην οποία αποθηκεύονται όλα τα δεδομένα και ρυθμίσεις του προγράμματος εργασίας και

ένας *Εκτυπωτής* στον οποίο τυπώνει η *Γραμματεία* το δημιουργημένο πρόγραμμα εργασίας του προσωπικού.

- Η *Γραμματεία* καταχωρεί για κάθε υπάλληλο, τα προσωπικά του στοιχεία, την ειδικότητά του, τις επιθυμίες του σχετικά με την εργασία ή τις αναπαύσεις του, τα συμβόλαια εργασίας, τη χρονική περίοδο ισχύος του προγράμματος και όποιες άλλες πληροφορίες είναι απαραίτητες για τη δημιουργία του προγράμματος εργασίας. Όλα τα ανωτέρω στοιχεία καταχωρούνται στο δευτερεύοντα χρήστη *Βάση Δεδομένων*.
- Εφόσον κρίνει η *Γραμματεία* ότι έχει όλες τις πληροφορίες για τη δημιουργία του προγράμματος εργασίας, τότε εκκινεί τη διαδικασία δημιουργίας του. Αν κατά τον έλεγχο πληρότητας των δεδομένων βρεθεί ότι κάποιες καταχωρήσεις δεν υπάρχουν καθόλου, τότε σταματάει η διαδικασία δημιουργίας αυτόματα και το σύστημα ζητάει από τη *Γραμματεία* τα απαιτούμενα δεδομένα.
- Όταν δημιουργηθεί επιτυχώς το πρόγραμμα, η *Γραμματεία* έχει τη δυνατότητα να το αποθηκεύσει σε αρχείο μορφής Excel ή να το τυπώσει στον δευτερεύοντα χρήστη *Εκτυπωτή*.

5.2 Εννοιολογικό μοντέλο

Το εννοιολογικό μοντέλο των κλάσεων, οι οποίες αποτελούν τα κεφάλαια πληροφοριών που περιέχονται στο αρχείο εισόδου του αλγορίθμου, που προκύπτει από το διάγραμμα Περιπτώσεων Χρήσης φαίνεται στην παρακάτω εικόνα. Έχει σχεδιαστεί με το εργαλείο Class Diagram του Microsoft Visual Studio 2017, το οποίο δεν φέρει όλα τα σχεδιαστικά εργαλεία που απαιτούνται για το εννοιολογικό μοντέλο. Γι' αυτό το λόγο το εννοιολογικό μοντέλο συνοδεύεται και από τον πίνακα συσχέτισης οντοτήτων μέσα στον οποίο αναγράφεται το είδος της συσχέτισης, η πληθηκότητα αλλά και η αιτιολογία της συσχέτισης.



Εικόνα 16: Εννοιολογικό μοντέλο των κλάσεων του προγράμματος εργασίας

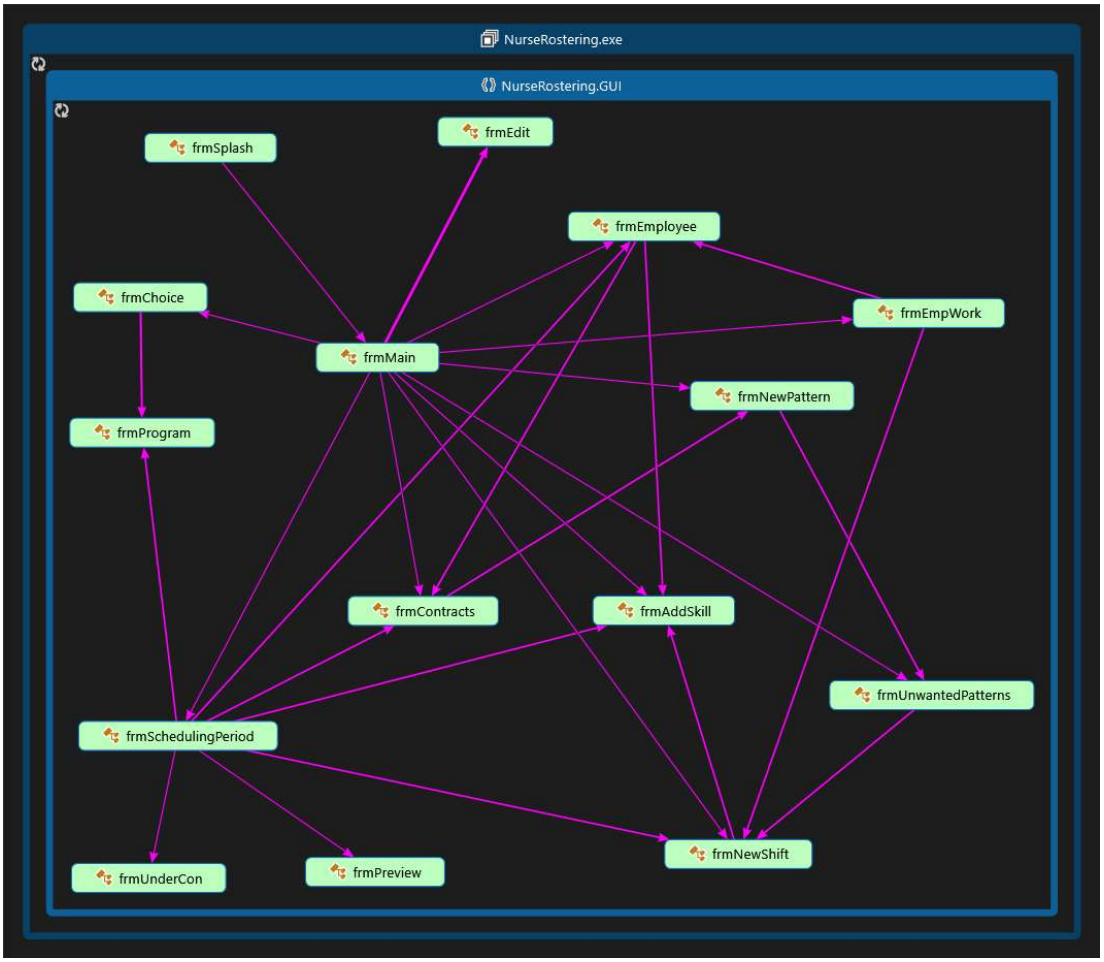
Πίνακας 1: Πίνακας συσχετίσεων μεταξύ των κλάσεων

Εννοιολογική κλάση	Σχετίζεται με	Περιγραφή / Αιτιολόγηση
Available_Skill	**Skill** Απλή συσχέτιση, 1:1…*	Η κλάση Available_Skill δημιουργήθηκε για να αναπαριστά τη λίστα των διαθέσιμων ειδικοτήτων από τις οποίες θα επιλέξει η Γραμματεία να αποδώσει στο Νοσοκόμο.
Employee	**Skill**, Απλή συσχέτιση, 1:1…*	Σε κάθε Νοσοκόμο, αντιστοιχίζεται τουλάχιστον μια ειδικότητα.
Employee	**Day_On_Request**, Απλή συσχέτιση, 1:0…*	Κάθε Νοσοκόμος μπορεί να δηλώσει ή όχι κάποια συγκεκριμένη ημέρα κατά την οποία θέλει να εργαστεί.
Employee	**Day_Off_Request**, Απλή συσχέτιση, 1:0…*	Κάθε Νοσοκόμος μπορεί να δηλώσει ή όχι κάποια συγκεκριμένη ημέρα κατά την οποία **ΔΕΝ** θέλει να εργαστεί.
Employee	**Shift_On_Request**, Απλή συσχέτιση, 1:0…*	Κάθε Νοσοκόμος μπορεί να δηλώσει ή όχι κάποια συγκεκριμένη βάρδια σε συγκεκριμένη ημέρα κατά την οποία θέλει να εργαστεί.
Employee	**Shift_Off_Request**, Απλή συσχέτιση, 1:0…*	Κάθε Νοσοκόμος μπορεί να δηλώσει ή όχι κάποια συγκεκριμένη βάρδια σε συγκεκριμένη ημέρα κατά την οποία **ΔΕΝ** θέλει να εργαστεί.
Contract	**Employee**, Απλή συσχέτιση 1:1…*	Κάθε Νοσοκόμος δεσμεύεται από ένα συμβόλαιο εργασίας με το νοσηλευτικό ίδρυμα, αλλά ένα συμβόλαιο μπορεί να υπογραφεί από πολλούς Νοσοκόμους.

Shift_Type	**Shift_On_Request**, Απλή συσχέτιση, 1:1…*	Ένας τύπος βάρδιας, μπορεί να καταχωρηθεί σε πολλές εγγραφές αιτημάτων εργασίας ενός νοσοκόμου.
Shift_Type	**Shift_Off_Request**, Απλή συσχέτιση, 1:1…*	Ένας τύπος βάρδιας, μπορεί να καταχωρηθεί σε πολλές εγγραφές αιτημάτων ανάπαυσης ενός νοσοκόμου.
Shift_Type	**Date_Specific_Cover**, Απλή συσχέτιση, 1:1…*	Ένας τύπος βάρδιας, μπορεί να καταχωρηθεί σε πολλές εγγραφές απαιτήσεων συγκεκριμένου πλήθους νοσοκόμων, από τη διοίκηση του ιδρύματος, για εργασία σε συγκεκριμένη ημέρα.
Shift_Type	**Day_Of_Week_Cover**, Απλή συσχέτιση, 1:1…*	Ένας τύπος βάρδιας, μπορεί να καταχωρηθεί σε πολλές εγγραφές απαιτήσεων συγκεκριμένου πλήθους νοσοκόμων, για κάθε ημέρα της εβδομάδος, από τη διοίκηση του ιδρύματος, για εργασία.
Shift_Type	**Pattern**, Απλή συσχέτιση, 0…*:0…*	Κανένας ή περισσότεροι τύποι βάρδιας, μπορούν να καταχωρηθούν σε κανένα έως πολλά πρότυπα ανεπιθύμητης εργασίας σε μια εγγραφή. Υπάρχει η επιλογή **Any**, η οποία καλύπτει όλες τις βάρδιες και λογίζεται σαν βάρδια αλλά και η επιλογή **None** η οποία δεν καλύπτει καμία βάρδια.
Available_Patterns	**Pattern**, Απλή συσχέτιση, 1…*:1	Η κλάση Available_Patterns δημιουργήθηκε για να αναπαριστά τη λίστα των διαθέσιμων στοιχείων τα οποία με ομαδοποίηση ή από

	μόνα τους θα συνθέσουν τα πρότυπα ανεπιθύμητης εργασίας των Νοσοκόμων.

Λόγω του ότι η εφαρμογή που θα δημιουργηθεί, θα εκτελείται σε γραφικό περιβάλλον, πρέπει να υπάρχει και μια γραφική διεπαφή αλληλεπίδρασης με τη Γραμματεία. Η διεπαφή αυτή, αποτελείται από φόρμες οι οποίες είναι ουσιαστικά κλάσεις, οι οποίες κληρονομούν την κλάση Form του λειτουργικού των Windows. Έτσι, μπορούμε να κατασκευάσουμε και γι' αυτές ένα εννοιολογικό μοντέλο. Στο μοντέλο αυτό όμως, δεν είναι δυνατός ο ορισμός πληθηκότητας. Μπορούν να αποτυπωθούν μόνο οι συνδέσεις των φορμών που υπάρχουν αναμεταξύ τους, μέσα στην εφαρμογή. Το εννοιολογικό μοντέλο των κλάσεων, οι οποίες αποτελούν τη γραφική διεπαφή αλληλεπίδρασης της Γραμματείας με το σύστημα, όπως επίσης προκύπτει από το διάγραμμα Περιπτώσεων Χρήσης, φαίνεται στην παρακάτω εικόνα. Έχει σχεδιαστεί και αυτό το εννοιολογικό μοντέλο, με το εργαλείο *Code Map* του *Microsoft Visual Studio 2017*.



Εικόνα 17: Διασύνδεση φορμών γραφικού περιβάλλοντος αλληλεπίδρασης με το NurseRostering

5.3 Λεκτική περιγραφή περιπτώσεων χρήσης και σχετικά διαγράμματα

Περίπτωση χρήσης: ΠΧ «Διαχείριση προσωπικού»

Κύριος χειριστής: Γραμματεία

Δευτερεύοντες χειριστές: Βάση δεδομένων

Προϋποθέσεις: Η Γραμματεία θα πρέπει να εκτελέσει την εφαρμογή.

Μετασυνθήκες (Κατάσταση εξόδου): Θα γίνεται εισαγωγή των δεδομένων των υπαλλήλων καθώς και των προτιμήσεων εργασίας τους και θα αποθηκεύονται σε μια βάση δεδομένων με μελλοντική χρήση.

Σύντομη περιγραφή: Η Γραμματεία δημιουργεί μια καρτέλα νέου υπαλλήλου ή ενημερώνει κάποια με στοιχεία που αφορούν σε κάποιον υπάλληλο είτε είναι προσωπικά δεδομένα είτε είναι προτιμήσεις εργασίας.

Βασική ροή:

1. Το σύστημα δίνει λίστα με τις διαθέσιμες ενέργειες σχετικά με τους υπαλλήλους, που μπορεί να πραγματοποιήσει η Γραμματεία.
2. Η Γραμματεία επιλέγει μια ενέργεια [Εναλλακτική ροή Α: «Νέος υπάλληλος»].
3. Η Γραμματεία επιλέγει μια ενέργεια [Εναλλακτική ροή Β: «Διαχείριση ειδικοτήτων»].
4. Η Γραμματεία επιλέγει μια ενέργεια [Εναλλακτική ροή Γ: «Επιθυμητή εργασία υπαλλήλου»].
5. Η Γραμματεία επιλέγει μια ενέργεια [Εναλλακτική ροή Δ: «Διαχείριση δεδομένων υπαλλήλου].
6. Η Γραμματεία επιστρέφει στο κεντρικό μενού, πατώντας το κουμπί «Επιστροφή στο κεντρικό μενού».
7. Η ΠΧ τελειώνει.

<u>Εναλλακτική ροή Α:</u> «Νέος υπάλληλος»

Α.1. Το σύστημα εμφανίζει ένα πίνακα με πεδία εισαγωγής στοιχείων του υπαλλήλου.

Α.2. Η Γραμματεία επιλέγει κάποιο συμβόλαιο για τον υπό καταχώρηση υπάλληλο ή δημιουργεί κάποιο νέο [Εναλλακτική ροή η ΠΧ «Συμβόλαια προγράμματος εργασίας»].

Α.3. Η Γραμματεία εισάγει τα υπόλοιπα δεδομένα του υπαλλήλου.

Α.4. Η Γραμματεία επιλέγει μια ενέργεια [Εναλλακτική ροή Β της βασικής ροής: «Διαχείριση ειδικοτήτων»].

Α.5. Η Γραμματεία πατάει το κουμπί «Καταχώρηση».

Α.6. Το σύστημα αποθηκεύει την καρτέλα του υπαλλήλου στη βάση δεδομένων.

Τα βήματα Α.2 έως Α.6 επαναλαμβάνονται μέχρι η Γραμματεία να πατήσει το κουμπί «Επιστροφή στο μενού διαχείρισης προσωπικού».

Α.7. Η Γραμματεία επιστρέφει στο μενού της περίπτωσης χρήσης «Διαχείριση προσωπικού» πατώντας το κουμπί «Επιστροφή στο μενού διαχείρισης προσωπικού».

Ο έλεγχος επιστρέφει στο βήμα 2 της βασικής ροής

Αυτή η εναλλακτική ροή δύναται να κληθεί και από την εναλλακτική ροή Γ της ΠΧ «Διαχείρισης προσωπικού». Σε αυτή την περίπτωση, οι ενέργειες που πραγματοποιούνται είναι οι ίδιες ακριβώς με τη διαφορά ότι όταν η Γραμματεία τελειώσει την εργασία της και τερματίσει τη λειτουργία της φόρμας, τότε ο έλεγχος επιστρέφει στο βήμα Γ.2 της εναλλακτικής ροής Γ της ΠΧ «Διαχείρισης προσωπικού».

Εναλλακτική ροή Β: «Διαχείριση ειδικοτήτων»

Β.1. Το σύστημα εμφανίζει ένα πίνακα με μια λίστα όλων των καταχωρημένων ειδικοτήτων.

Β.2. Η Γραμματεία εισάγει μια νέα ειδικότητα στο κατάλληλο πλαίσιο και πατάει το κουμπί «Εισαγωγή».

Β.3. Το σύστημα ελέγχει για την ύπαρξη αυτής της ειδικότητας και αν υπάρχει ενημερώνει σχετικά τη Γραμματεία, διαφορετικά την εισάγει στη λίστα των ειδικοτήτων προς εισαγωγή.

Β.4. Η Γραμματεία επιλέγει μια ειδικότητα και πατάει το κουμπί «Διαγραφή».

Β.5. Το σύστημα διαγράφει την επιλεγμένη ειδικότητα από τη λίστα των προς καταχώρηση στη βάση δεδομένων ειδικοτήτων.

Β.6. Η Γραμματεία πατάει το κουμπί «Καθαρισμός λίστας».

Β.7. Το σύστημα απαλείφει όλες τις ειδικότητες από τη λίστα των προς καταχώρηση στη βάση ειδικοτήτων.

Β.8. Η Γραμματεία επιλέγει μια καταχωρημένη ειδικότητα και πατάει το κουμπί «Διαγραφή επιλεγμένης εγγεγραμμένης ειδικότητας».

Β.9. Η Γραμματεία πατάει το κουμπί «Καταχώρηση λίστας».

Β.10. Το σύστημα αποθηκεύει τη λίστα ειδικοτήτων που δημιουργήθηκε από τη Γραμματεία στη βάση δεδομένων.

Β.11. Τα βήματα Β.2 έως Β.10 επαναλαμβάνονται μέχρι η Γραμματεία να πατήσει το κουμπί «Επιστροφή στο μενού διαχείρισης προσωπικού».

Ο έλεγχος επιστρέφει στο βήμα 3 της βασικής ροής

Αυτή η εναλλακτική ροή δύναται να κληθεί και από την εναλλακτική ροή Α της ΠΧ «Διαχείρισης προσωπικού». Σε αυτή την περίπτωση, οι ενέργειες που πραγματοποιούνται είναι οι ίδιες ακριβώς με τη διαφορά ότι όταν η Γραμματεία τελειώσει την εργασία της και τερματίσει τη λειτουργία της φόρμας, τότε ο έλεγχος επιστρέφει στο βήμα Α.4 της εναλλακτικής ροής Α της ΠΧ «Διαχείρισης προσωπικού». Επίσης, αυτή η εναλλακτική ροή, μπορεί να κληθεί και από την εναλλακτική ροή Α «Προσθήκη βάρδιας» της ΠΧ «Διαχείρισης βαρδιών». Και σε αυτή την περίπτωση, οι ενέργειες που πραγματοποιούνται είναι οι ίδιες ακριβώς με τη διαφορά ότι όταν η Γραμματεία τελειώσει την εργασία της και τερματίσει τη λειτουργία της φόρμας, τότε ο έλεγχος επιστρέφει στο βήμα Α.2 της εναλλακτικής ροής Α «Προσθήκη βάρδιας» της ΠΧ «Διαχείρισης βαρδιών» όταν η Γραμματεία πατήσει το κουμπί «Επιστροφή στο μενού διαχείρισης βαρδιών».

Εναλλακτική ροή Γ: «Επιθυμητή εργασία υπαλλήλου»

Γ.1. Το σύστημα εμφανίζει ένα πίνακα με μια λίστα όλων των καταχωρημένων νοσοκόμων.

Γ.2. Η Γραμματεία επιλέγει μια ενέργεια [Εναλλακτική ροή Α της βασικής ροής: «Νέος υπάλληλος»]

Γ.3. Η Γραμματεία επιλέγει μια ενέργεια [Εναλλακτική ροή Α της βασικής ροής της ΠΧ «Διαχείριση βαρδιών»: «Προσθήκη βάρδιας»]

Γ.4. Η Γραμματεία επιλέγει ένα νοσοκόμο.

Γ.5. Η Γραμματεία επιλέγει μια από τις καρτέλες σχετικές με τις επιθυμίες εργασίας ή ανάπαυσης του νοσοκόμου.

Γ.6. Το σύστημα απενεργοποιεί τη λίστα των καταχωρημένων νοσοκόμων, αποτρέποντας τη Γραμματεία να αλλάξει νοσοκόμο κατά τη διάρκεια δήλωσης των ημερών εργασίας και ανάπαυσης του νοσοκόμου.

Γ.7. Η Γραμματεία εισάγει τα δεδομένα επιθυμητής εργασίας του νοσοκόμου.

Γ.8. Η Γραμματεία πατάει το κουμπί «Προσθήκη».

Γ.9. Το σύστημα εισάγει τα δεδομένα στη λίστα προς καταχώρηση στη βάση δεδομένων αφού έχει πραγματοποιήσει έλεγχο για την ύπαρξη αυτής στη βάση δεδομένων. Αν υπάρχει, ενημερώνει με σχετικό μήνυμα τη Γραμματεία.

Γ.10. Η Γραμματεία επιλέγει μια επιθυμία του νοσοκόμου και πατάει το κουμπί «Αφαίρεση».

Γ.11. Το σύστημα διαγράφει την επιθυμία του νοσοκόμου από τη λίστα προς καταχώρηση στη βάση δεδομένων.

Γ.12. Η Γραμματεία πατάει το κουμπί «Καθαρισμός λίστας».

Γ.13. Το σύστημα απαλείφει όλα τα δεδομένα από τις σχετικές λίστες επιθυμιών που είναι προς καταχώρηση στη βάση δεδομένων.

Γ.14. Η Γραμματεία πατάει το κουμπί «Καταχώριση επιλογών».

Γ.15. Το σύστημα εγγράφει όλες τις καταχωρημένες επιλογές της Γραμματείας, στη βάση δεδομένων και απελευθερώνει τη λίστα των καταχωρημένων νοσοκόμων.

Γ.16. Τα βήματα Γ.2 έως Γ.15 επαναλαμβάνονται μέχρι η Γραμματεία να πατήσει το κουμπί «Επιστροφή στο μενού διαχείρισης υπαλλήλων»

Ο έλεγχος επιστρέφει στο βήμα 4 της βασικής ροής.

Εναλλακτική ροή Δ: «Διαχείριση δεδομένων υπαλλήλων»

Δ.1. Το σύστημα εμφανίζει ένα πίνακα με επιλογές που σχετίζονται με τον υπάλληλο.

Δ.2. Η Γραμματεία επιλέγει μια ενέργεια [Εναλλακτική ροή ΔΔ1: «Αλλαγή στοιχείων ή διαγραφή υπαλλήλου»].

Δ.3. Η Γραμματεία επιλέγει μια ενέργεια [Εναλλακτική ροή ΔΔ2: «Αλλαγή στοιχείων ή διαγραφή συμβολαίου»].

Δ.4. Η Γραμματεία επιλέγει μια ενέργεια [Εναλλακτική ροή ΔΔ3: «Διαγραφή ειδικοτήτων που έχουν απονεμηθεί σε υπάλληλο»].

Δ.5. Η Γραμματεία πατάει το κουμπί «Επιστροφή στο μενού διαχείρισης υπαλλήλων».

Ο έλεγχος επιστρέφει στο βήμα 5 της βασικής ροής.

Εναλλακτική ροή ΔΔ1: «Αλλαγή στοιχείων ή διαγραφή υπαλλήλου»

ΔΔ1.1 Το σύστημα εμφανίζει ένα πίνακα με τα όλα τα στοιχεία, προσωπικά και εργασίας, όλων των καταχωρημένων νοσοκόμων.

ΔΔ1.2 Η Γραμματεία επιλέγει ένα νοσοκόμο.

ΔΔ1.3 Η Γραμματεία επιλέγει μια εγγραφή και πατάει το κουμπί «Διαγραφή επιλεγμένης εγγραφής»

ΔΔ1.4 Το σύστημα ζητάει επιβεβαίωση για να διαγράψει την επιλεγμένη εγγραφή από τη βάση δεδομένων.

ΔΔ1.5 Η Γραμματεία απαντάει καταφατικά.

ΔΔ1.6 Το σύστημα διαγράφει την επιλεγμένη εγγραφή από τη βάση δεδομένων.

ΔΔ1.7 Η Γραμματεία επιλέγει ένα από τα πλαίσια που επιθυμεί να αλλάξει τα δεδομένα του και το ενεργοποιεί με διπλό κλικ επάνω του.

ΔΔ1.8 Η Γραμματεία εισάγει τα νέα δεδομένα και πατάει το κουμπί «Αποθήκευση αλλαγών».

ΔΔ1.9 Το σύστημα αποθηκεύει τις αλλαγές στη βάση δεδομένων.

ΔΔ1.10 Τα βήματα ΔΔ1.2 έως ΔΔ1.9 επαναλαμβάνονται μέχρι η Γραμματεία να πατήσει το κουμπί «Κλείσιμο προβολής».

Ο έλεγχος επιστρέφει στο βήμα Δ.2 της εναλλακτικής ροής Δ: «Διαχείριση δεδομένων υπαλλήλων»

Εναλλακτική ροή ΔΔ2: «Αλλαγή στοιχείων ή διαγραφή συμβολαίου»

ΔΔ2.1 Το σύστημα εμφανίζει ένα πίνακα με τα όλα τα στοιχεία, προσωπικά και εργασίας, όλων των καταχωρημένων νοσοκόμων.

ΔΔ2.2 Η Γραμματεία επιλέγει ένα νοσοκόμο.

ΔΔ2.3 Η Γραμματεία πατάει το κουμπί «Διαγραφή επιλεγμένης εγγραφής»

ΔΔ2.4 Το σύστημα εμφανίζει ένα μήνυμα ερώτησης για διαγραφή της επιλεγμένης εγγραφής

ΔΔ2.5 Η Γραμματεία απαντάει καταφατικά.

ΔΔ2.6 Το σύστημα διαγράφει την σχετική εγγραφή από τη βάση δεδομένων.

ΔΔ2.7 Η Γραμματεία επιλέγει ένα από τα πλαίσια που επιθυμεί να αλλάξει τα δεδομένα του και το ενεργοποιεί με διπλό κλικ επάνω του.

ΔΔ2.8 Η Γραμματεία εισάγει τα νέα δεδομένα.

ΔΔ2.9 Η Γραμματεία πατάει το κουμπί «Αποθήκευση αλλαγών».

ΔΔ2.10 Το σύστημα αποθηκεύει τις αλλαγές στη βάση δεδομένων.

ΔΔ2.11 Τα βήματα ΔΔ2.2 έως ΔΔ2.11 επαναλαμβάνονται μέχρι η Γραμματεία να πατήσει το κουμπί «Κλείσιμο προβολής».

Ο έλεγχος επιστρέφει στο βήμα Δ.3 της εναλλακτικής ροής Δ: «Διαχείριση δεδομένων υπαλλήλων».

Εναλλακτική ροή ΔΔ3: «Διαγραφή ειδικοτήτων που έχουν απονεμηθεί σε υπάλληλο»

ΔΔ3.1 Το σύστημα εμφανίζει ένα πίνακα με τα όλα τα στοιχεία, προσωπικά και εργασίας, όλων των καταχωρημένων νοσοκόμων.

ΔΔ3.2 Η Γραμματεία επιλέγει ένα νοσοκόμο και πατάει το κουμπί «Διαγραφή επιλεγμένης εγγραφής».

ΔΔ3.3 Το σύστημα εμφανίζει ένα μήνυμα ερώτησης για διαγραφή της επιλεγμένης εγγραφής

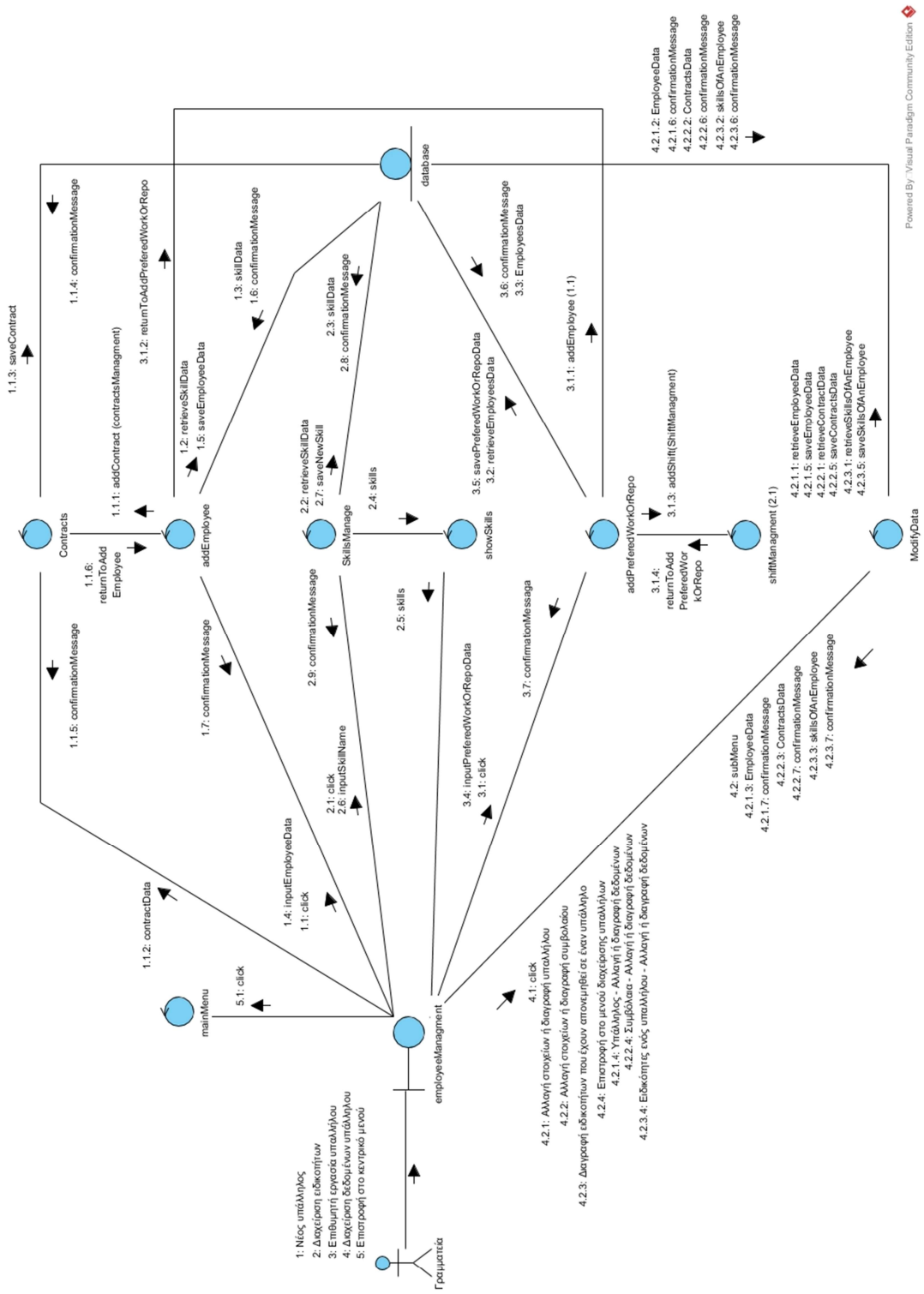
ΔΔ3.4 Η Γραμματεία απαντάει καταφατικά.

ΔΔ3.5 Το σύστημα διαγράφει την σχετική εγγραφή από τη βάση δεδομένων.

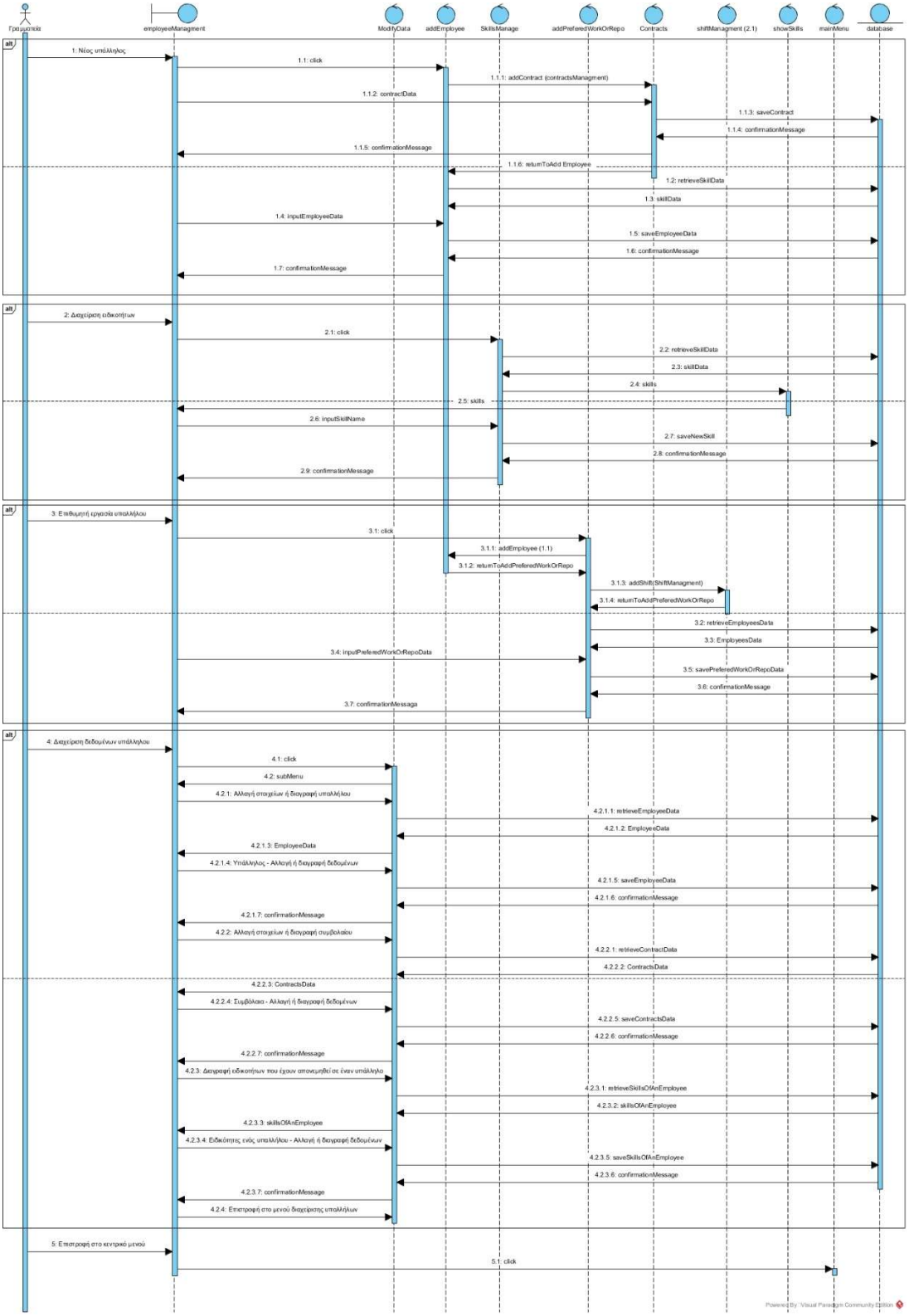
ΔΔ3.6 Τα βήματα ΔΔ3.2 έως ΔΔ3.5 επαναλαμβάνονται μέχρι η Γραμματεία να πατήσει το κουμπί «Κλείσιμο προβολής».

Ο έλεγχος επιστρέφει στο βήμα Δ.4 της εναλλακτικής ροής Δ: «Διαχείριση δεδομένων υπαλλήλων».

Στις επόμενες δύο εικόνες, φαίνονται τα διαγράμματα ευρωστίας και ακολουθίας της ΠΧ «Διαχείριση προσωπικού»:



Εικόνα 18: Διάγραμμα ευρωστίας της ΠΧ "Διαχείριση υπαλλήλων"



Εικόνα 19: Διάγραμμα ακολουθίας της ΠΧ "Διαχείριση προσωπικού"

Περίπτωση χρήσης: ΠΧ «Διαχείριση βαρδιών»

Κύριος χειριστής: Γραμματεία

Δευτερεύοντες χειριστές: Βάση δεδομένων

Προϋποθέσεις: Η Γραμματεία θα πρέπει να εκτελέσει την εφαρμογή.

Μετασυνθήκες (Κατάσταση εξόδου): Θα γίνεται επεξεργασία δεδομένων των βαρδιών και θα αποθηκεύονται σε μια βάση δεδομένων με μελλοντική χρήση.

Σύντομη περιγραφή: Η Γραμματεία δημιουργεί μια νέα καρτέλα με τα χαρακτηριστικά μιας βάρδιας (ώρα έναρξης – λήξης, όνομα βάρδιας κλπ) ή ενημερώνει κάποια άλλη, υπάρχουσα, με τυχόν αλλαγές στοιχείων. Επίσης, μπορεί να διαγραφεί όποια βάρδια δεν είναι επιθυμητή πλέον. Επίσης, μπορεί να δημιουργεί και τα πρότυπα ανεπιθύμητης εργασίας των υπαλλήλων.

Βασική ροή:

1. Το σύστημα δίνει λίστα με τις διαθέσιμες ενέργειες σχετικά με τις βάρδιες, που μπορεί να πραγματοποιήσει η Γραμματεία.
2. Η Γραμματεία επιλέγει μια ενέργεια [Εναλλακτική ροή Α: «Προσθήκη βάρδιας»].
3. Η Γραμματεία επιλέγει μια ενέργεια [Εναλλακτική ροή Β: «Τροποποίηση ή κατάργηση βάρδιας»].
4. Η Γραμματεία επιλέγει μια ενέργεια [Εναλλακτική ροή Γ: «Ανεπιθύμητα πρότυπα εργασίας»].
5. Η Γραμματεία επιλέγει μια ενέργεια [Εναλλακτική ροή Δ: «Γκρουπ ανεπιθύμητων προτύπων εργασίας»].
6. Η Γραμματεία επιστρέφει στο κεντρικό μενού, πατώντας το κουμπί “Επιστροφή στο κεντρικό μενού”.
7. Η ΠΧ τελειώνει.

<u>Εναλλακτική ροή Α:</u> «Προσθήκη βάρδιας»

Α.1. Το σύστημα εμφανίζει ένα πίνακα με πεδία εισαγωγής στοιχείων της βάρδιας.

Α.2. Η Γραμματεία επιλέγει μια ενέργεια [Εναλλακτική ροή Β «Διαχείριση ειδικοτήτων» της βασικής ροής της ΠΧ «Διαχείριση προσωπικού»].

Α.3. Η Γραμματεία εισάγει όλα τα απαραίτητα δεδομένα.

Α.4. Η Γραμματεία πατάει το κουμπί «Καταχώρηση βάρδιας».

Α.5. Το σύστημα καταχωρεί στη βάση δεδομένων τα στοιχεία της βάρδιας, αφού πρώτα ελέγξει την ύπαρξη σχετικών δεδομένων στη βάση δεδομένων. Αν υπάρχει σχετική εγγραφή, ενημερώνει τη Γραμματεία με σχετικό μήνυμα, αλλιώς καταχωρεί.

Α.6. Η Γραμματεία πατάει το κουμπί «Καθαρισμός πεδίων»

Α.7. Το σύστημα απαλείφει όλα τα δεδομένα από τα σχετικά πεδία δημιουργίας βάρδιας.

Α.8. Τα βήματα Α.2 έως Α.7 επαναλαμβάνονται μέχρι η Γραμματεία να πατήσει το κουμπί «Επιστροφή στο μενού διαχείρισης βαρδιών.

Ο έλεγχος μεταφέρεται στο βήμα 2 της βασικής ροής

Εναλλακτική ροή Β: «Τροποποίηση ή κατάργηση βάρδιας».

Β.1 Το σύστημα εμφανίζει ένα πίνακα με τα όλα τα στοιχεία των καταχωρημένων βαρδιών.

Β.2 Η Γραμματεία επιλέγει μια βάρδια και πατάει το κουμπί «Διαγραφή επιλεγμένης εγγραφής»

Β.3. Το σύστημα εμφανίζει ένα μήνυμα ερώτησης για διαγραφή της επιλεγμένης εγγραφής

Β.4 Η Γραμματεία απαντάει καταφατικά.

Β.5 Το σύστημα διαγράφει την σχετική εγγραφή από τη βάση δεδομένων.

Β.6. Τα βήματα Β.2 έως Β.5 επαναλαμβάνονται όσες φορές το επιθυμεί η Γραμματεία.

Β.7 Η Γραμματεία επιλέγει ένα από τα πλαίσια που επιθυμεί να αλλάξει τα δεδομένα του και το ενεργοποιεί με διπλό κλικ επάνω του.

Β.8 Η Γραμματεία εισάγει τα νέα δεδομένα.

Β.9 Η Γραμματεία πατάει το κουμπί «Αποθήκευση αλλαγών».

Β.10 Το σύστημα αποθηκεύει τις αλλαγές στη βάση δεδομένων.

Β.11 Τα βήματα Β.2 έως Β.10 επαναλαμβάνονται μέχρι η Γραμματεία να πατήσει το κουμπί «Κλείσιμο προβολής».

Ο έλεγχος επιστρέφει στο βήμα 3 της βασικής ροής

Εναλλακτική ροή Γ: «Ανεπιθύμητα πρότυπα εργασίας»

Γ.1. Το σύστημα εμφανίζει ένα πίνακα με τις διαθέσιμες βάρδιες και τα καταχωρημένα πρότυπα ανεπιθύμητης εργασίας.

Γ.2. Η Γραμματεία επιλέγει μια ενέργεια [Εναλλακτική ροή Α της βασικής ροής «Προσθήκη βάρδιας»].

Γ.3. Η Γραμματεία ένα καταχωρημένο πρότυπο και πατάει το κουμπί «Διαγραφή επιλεγμένου στοιχείου».

Γ.4. Το σύστημα ρωτάει τη Γραμματεία για επιβεβαίωση της διαγραφής με σχετικό μήνυμα.

Γ.5. Η Γραμματεία απαντά καταφατικά.

Γ.6. Το σύστημα αφαιρεί την επιλεγμένη ειδικότητα από τη βάση δεδομένων.

Γ.7. Τα βήματα Γ.3 έως Γ.6 επαναλαμβάνονται όσες φορές το επιθυμεί η Γραμματεία.

Γ.8. Η Γραμματεία εισάγει τα απαραίτητα δεδομένα.

Γ.9. Η Γραμματεία πατάει το κουμπί «Προσθήκη».

Γ.10. Το σύστημα εισάγει το δημιουργημένο πρότυπο σε μια λίστα ανεπιθύμητων προτύπων εργασίας η οποία θα καταχωρηθεί αργότερα στη βάση δεδομένων.

Γ.11. Η Γραμματεία επιλέγει ένα στοιχείο από τη λίστα των προτύπων ανεπιθύμητης εργασίας και πατάει το κουμπί «Αφαίρεση».

Γ.12. Το σύστημα αφαιρεί το επιλεγμένο πρότυπο από τη λίστα των προς καταχώρηση στη βάση δεδομένων προτύπων ανεπιθύμητης εργασίας.

Γ.13. Τα βήματα Γ.11 έως Γ.12 επαναλαμβάνονται όσες φορές το επιθυμεί η Γραμματεία.

Γ.14. Η Γραμματεία πατάει το κουμπί «Δημιουργία προτύπων».

Γ.15. Το σύστημα αποθηκεύει τη λίστα ανεπιθύμητων προτύπων εργασίας στη βάση δεδομένων.

Γ.16. Η Γραμματεία πατάει το κουμπί «Καθαρισμός λίστας».

Γ.17. Το σύστημα απαλείφει όλα τα πρότυπα ανεπιθύμητης εργασίας από την προς καταχώρηση στη βάση δεδομένων λίστα.

Γ.18. Τα βήματα Γ.2 έως Γ.17 επαναλαμβάνονται μέχρι η Γραμματεία να πατήσει το κουμπί «Επιστροφή στο μενού διαχείρισης βαρδιών».

Ο έλεγχος επιστρέφει στο βήμα 4 της βασικής ροής.

Εναλλακτική ροή Δ: «Γκρουπ ανεπιθύμητων προτύπων εργασίας»

Δ.1. Το σύστημα εμφανίζει ένα πίνακα με τα καταχωρημένα πρότυπα ανεπιθύμητης εργασίας.

Δ.2. Η Γραμματεία επιλέγει μια ενέργεια [Εναλλακτική ροή Γ της βασικής ροής «Δημιουργία νέου προτύπου»].

Δ.3. Η Γραμματεία εισάγει τα απαραίτητα δεδομένα.

Δ.4. Η Γραμματεία πατάει το κουμπί «Προσθήκη».

Δ.5. Το σύστημα εισάγει το επιλεγμένο στοιχειώδες πρότυπο σε μια λίστα ανεπιθύμητων στοιχειωδών προτύπων εργασίας τα οποία θα συνθέσουν το ανεπιθύμητο πρότυπο εργασίας και το οποίο θα καταχωρηθεί αργότερα στη βάση δεδομένων.

Δ.6. Η Γραμματεία επιλέγει ένα στοιχείο από τη λίστα των στοιχείων του υπό σύνθεση πρότυπου ανεπιθύμητης εργασίας και πατάει το κουμπί «Αφαίρεση».

Δ.7. Το σύστημα αφαιρεί το επιλεγμένο πρότυπο από τη λίστα των στοιχείων του υπό σύνθεση πρότυπου ανεπιθύμητης εργασίας.

Δ.8. Τα βήματα Δ6 έως Δ7 επαναλαμβάνονται όσες φορές το επιθυμεί η Γραμματεία.

Δ.9. Η Γραμματεία πατάει το κουμπί «Δημιουργία πρότυπου».

Δ.10. Το σύστημα αποθηκεύει το ανεπιθύμητο πρότυπο εργασίας στη βάση δεδομένων.

Δ.11. Η Γραμματεία πατάει το κουμπί «Καθαρισμός λίστας».

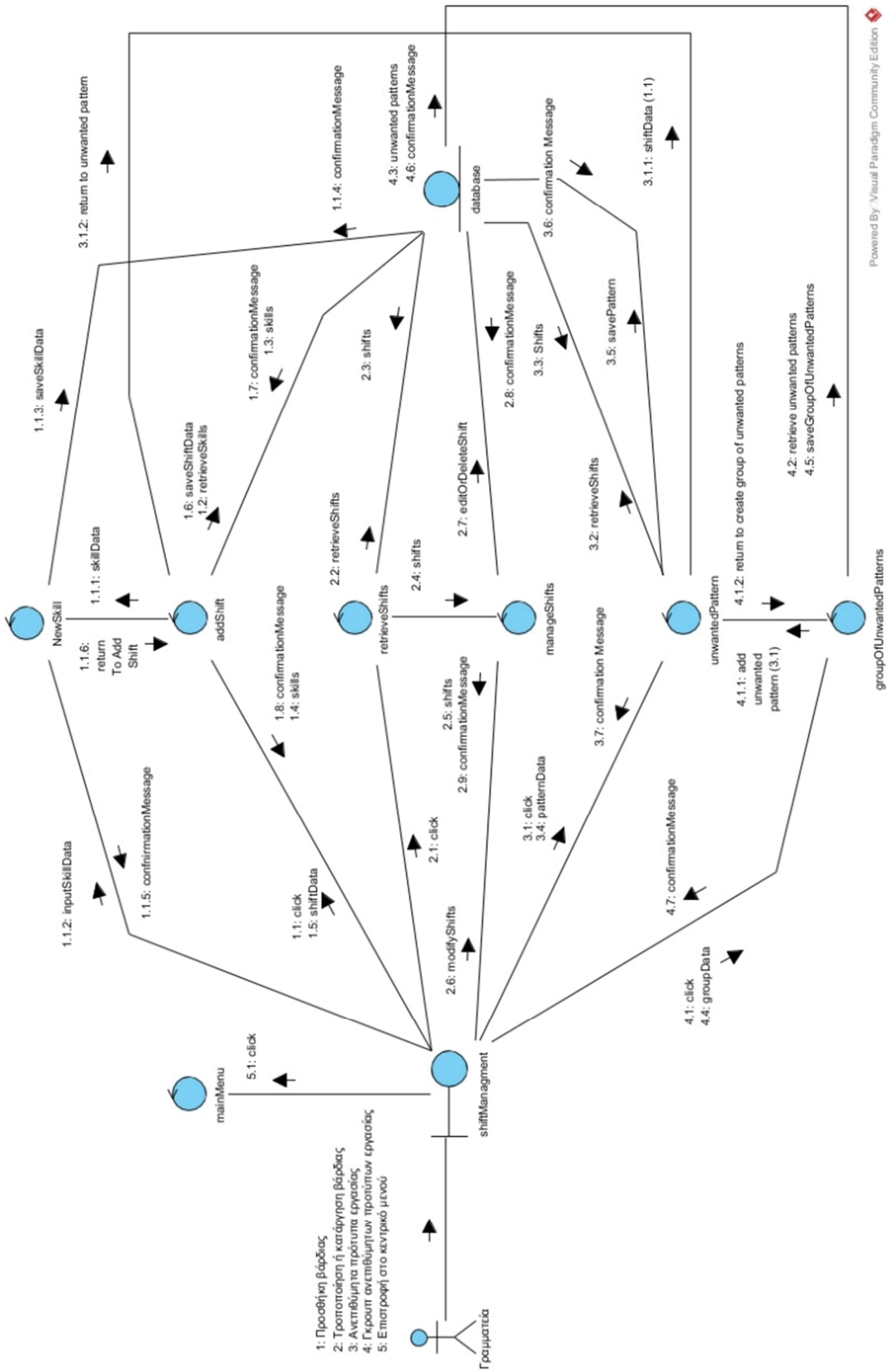
Δ.12. Το σύστημα απαλείφει όλα τα στοιχεία από τη λίστα σύνθεσης του πρότυπου ανεπιθύμητης εργασίας.

Δ.13. Τα βήματα Δ.2 έως Δ.12 επαναλαμβάνονται μέχρι η Γραμματεία να πατήσει το κουμπί «Επιστροφή στο μενού διαχείρισης βαρδιών».

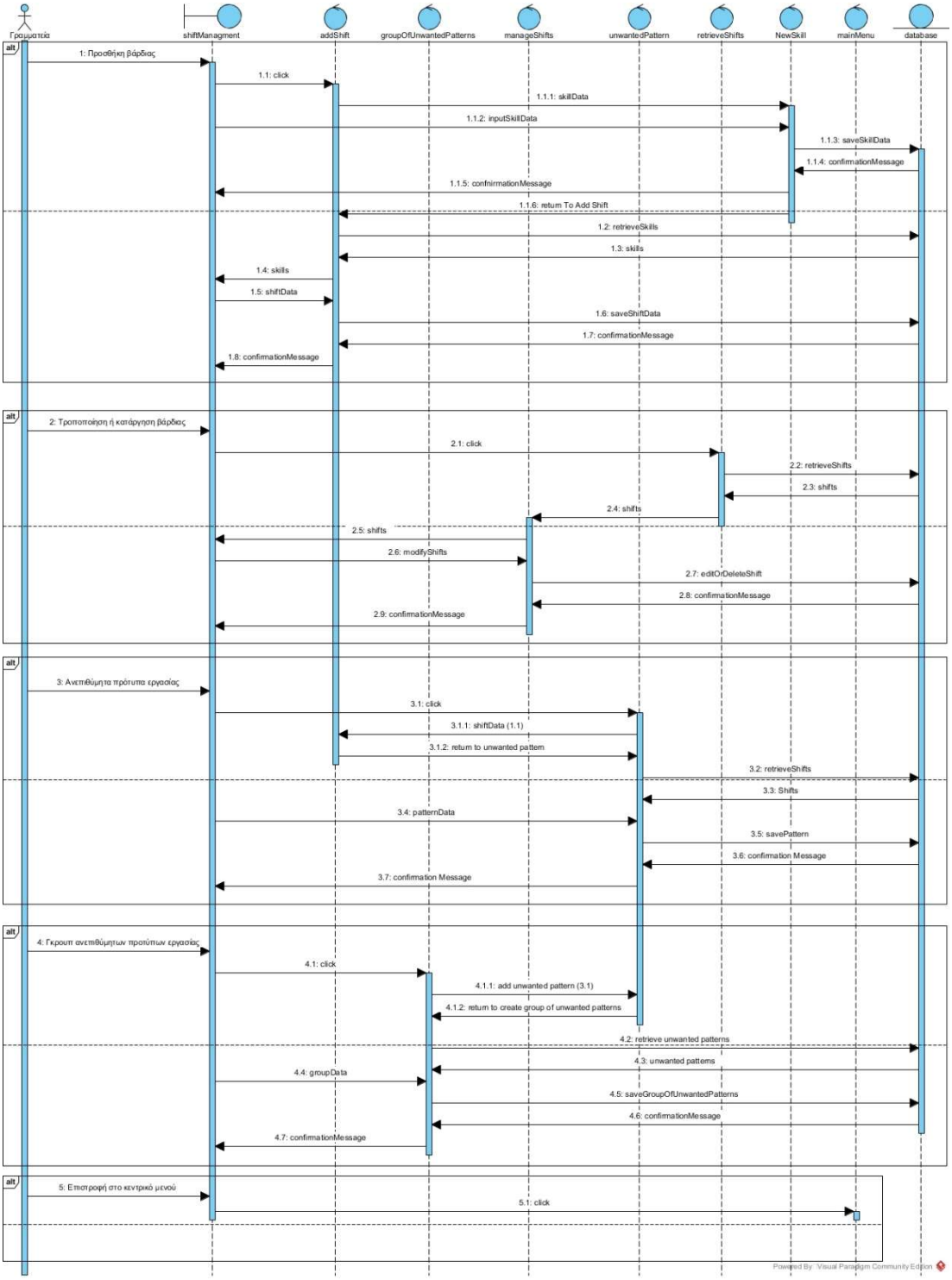
Ο έλεγχος επιστρέφει στο βήμα 5 της βασικής ροής.

Αυτή η εναλλακτική ροή δύναται να κληθεί και από το βήμα 4 της βασικής ροής της ΠΧ «Συμβόλαια προγράμματος εργασίας». Σε αυτή την περίπτωση, οι ενέργειες που πραγματοποιούνται είναι οι ίδιες ακριβώς με τη διαφορά ότι όταν η Γραμματεία τελειώσει την εργασία της και τερματίσει τη λειτουργία της φόρμας, τότε ο έλεγχος επιστρέφει στο βήμα 4 της βασικής ροής της ΠΧ «Συμβόλαια προγράμματος εργασίας».

Κι εδώ, όπως και στην προηγούμενη ΠΧ, στις επόμενες δύο εικόνες φαίνονται τα διαγράμματα ευρωστίας και ακολουθίας της ΠΧ «Διαχείριση βαρδιών»:



Εικόνα 20: Διάγραμμα ευρωστίας της ΠΧ "Διαχείριση βαρδιών"



Εικόνα 21: Διάγραμμα ακολουθίας της ΠΧ "Διαχείριση βαρδιών"

Περίπτωση χρήσης: ΠΧ «Συμβόλαια προγράμματος εργασίας»

Κύριος χειριστής: Γραμματεία

Δευτερεύοντες χειριστές: Βάση δεδομένων

Προϋποθέσεις: Η Γραμματεία θα πρέπει να εκτελέσει την εφαρμογή.

Μετασυνθήκες (Κατάσταση εξόδου): Θα έχουν δημιουργηθεί μερικά συμβόλαια με κατ' επιλογή όρους και θα αποθηκευτούν στη βάση δεδομένων για μελλοντική χρήση. Επίσης, η Γραμματεία θα μπορεί να διαγράφει τα συμβόλαια της επιλογής της.

Σύντομη περιγραφή: Η Γραμματεία εισάγει σε ένα πίνακα όλα τα απαραίτητα δεδομένα και περιορισμούς που θέτουν η διοίκηση και το προσωπικό του νοσηλευτικού ιδρύματος για την κατάρτιση του προγράμματος εργασίας.

Βασική ροή:

1. Το σύστημα εμφανίζει ένα πίνακα με διάφορες ρυθμίσεις, πεδία κειμένου και επιλογών τα οποία περιγράφουν τις απαιτήσεις-περιορισμούς για το πρόγραμμα εργασίας καθώς και τα αποθηκευμένα συμβόλαια.
2. Η Γραμματεία επιλέγει μια ενέργεια [Εναλλακτική ροή ΝΣ: «Νέο συμβόλαιο»].
3. Η Γραμματεία επιλέγει μια ενέργεια [Εναλλακτική ροή ΔΣ: «Διαγραφή συμβολαίου»].
4. Η Γραμματεία επιλέγει μια ενέργεια [Εναλλακτική ροή Δ «Γκρουπ ανεπιθύμητων προτύπων εργασίας» της ΠΧ «Διαχείριση βαρδιών»].
5. Η Γραμματεία επιστρέφει στο κεντρικό μενού, πατώντας το κουμπί «Επιστροφή στο κύριο μενού».
6. Η ΠΧ τελειώνει.

   Η ΠΧ αυτή, μπορεί να κληθεί και από την εναλλακτική ροή Α «Νέος υπάλληλος» της ΠΧ «Διαχείριση προσωπικού». Σε αυτή την περίπτωση, η Γραμματεία εκτελεί ακριβώς τις ίδιες ενέργειες, με τη διαφορά ότι όταν τερματιστεί η λειτουργία αυτής της φόρμας πατώντας το κουμπί «Επιστροφή στην καταχώρηση υπαλλήλου», ο έλεγχος θα επιστρέψει στο βήμα Α.2 της εναλλακτικής ροής Α «Νέος υπάλληλος» της ΠΧ «Διαχείριση προσωπικού».

<u>Εναλλακτική ροή ΝΣ:</u> «Νέο συμβόλαιο»

ΝΣ.1. Η Γραμματεία επιλέγει από το μενού επιλογών το κουμπί Συμβόλαιο → «Νέο»

ΝΣ.2. Το σύστημα «ξεκλειδώνει» το πλαίσιο κειμένου ονόματος του συμβολαίου και ενεργοποιεί το κουμπί Συμβόλαιο → «Αποθήκευση».

ΝΣ.3. Η Γραμματεία εισάγει όλα τα απαραίτητα δεδομένα.

ΝΣ.4. Η Γραμματεία επιλέγει ένα ανεπιθύμητο πρότυπο και πατάει το κουμπί «Προσθήκη»

ΝΣ.5. Το σύστημα εισάγει το επιλεγμένο ανεπιθύμητο πρότυπο στη σχετική λίστα του υπό δημιουργία συμβολαίου.

ΝΣ.6. Τα βήματα ΝΣ.4 έως ΝΣ.5 επαναλαμβάνονται όσες φορές επιθυμεί η Γραμματεία.

ΝΣ.7. Η Γραμματεία επιλέγει ένα ανεπιθύμητο πρότυπο εργασίας από τη λίστα των επιλεγμένων προτύπων ανεπιθύμητης εργασίας του υπό σύνθεση συμβολαίου και πατάει το κουμπί «Αφαίρεση».

ΝΣ.8. Το σύστημα αφαιρεί το επιλεγμένο πρότυπο από τη λίστα των επιλεγμένων προτύπων ανεπιθύμητης εργασίας του υπό σύνθεση συμβολαίου.

ΝΣ.9. Τα βήματα ΝΣ.7 έως ΝΣ.8 επαναλαμβάνονται όσες φορές το επιθυμεί η Γραμματεία.

ΝΣ.10. Η Γραμματεία πατάει το κουμπί «Καθαρισμός λίστας».

ΝΣ.11. Το σύστημα απαλείφει όλα τα στοιχεία από τη λίστα σύνθεσης του πρότυπου ανεπιθύμητης εργασίας.

ΝΣ.12. Η Γραμματεία πατάει το κουμπί Συμβόλαιο → «Αποθήκευση».

ΝΣ.13. Το σύστημα αποθηκεύει το συμβόλαιο στη βάση δεδομένων.

ΝΣ.14. Τα βήματα ΝΣ.1 έως ΝΣ.13 επαναλαμβάνονται όσες φορές επιθυμεί η Γραμματεία

Ο έλεγχος επιστρέφει στο βήμα 2 της βασικής ροής

<u>Εναλλακτική ροή ΔΣ:</u> «Διαγραφή συμβολαίου»

ΔΣ.1. Η Γραμματεία επιλέγει ένα συμβόλαιο από τη σχετική λίστα και πατάει το κουμπί Συμβόλαιο → «Διαγραφή».

ΔΣ.2. Το σύστημα ζητάει επιβεβαίωση διαγραφής από τη Γραμματεία με σχετικό μήνυμα.

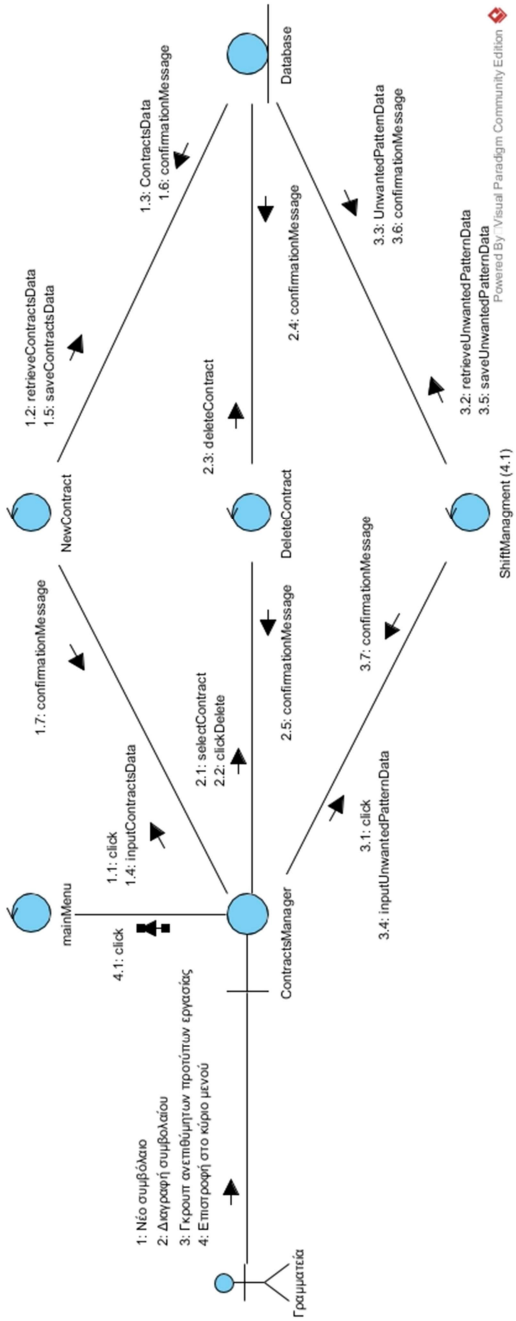
ΔΣ.3. Η Γραμματεία απαντάει καταφατικά.

ΔΣ.4. Το σύστημα ελέγχει αν το υπό διαγραφή συμβόλαιο αντιστοιχεί σε κάποιο νοσοκόμο. Αν βρει αντιστοίχιση, με τουλάχιστον ένα νοσοκόμο, ενημερώνει τη Γραμματεία για τα αποτελέσματα με σχετικό μήνυμα και σταματάει τη διαγραφή. Σε διαφορετική περίπτωση, διαγράφει το συμβόλαιο από τη βάση δεδομένων

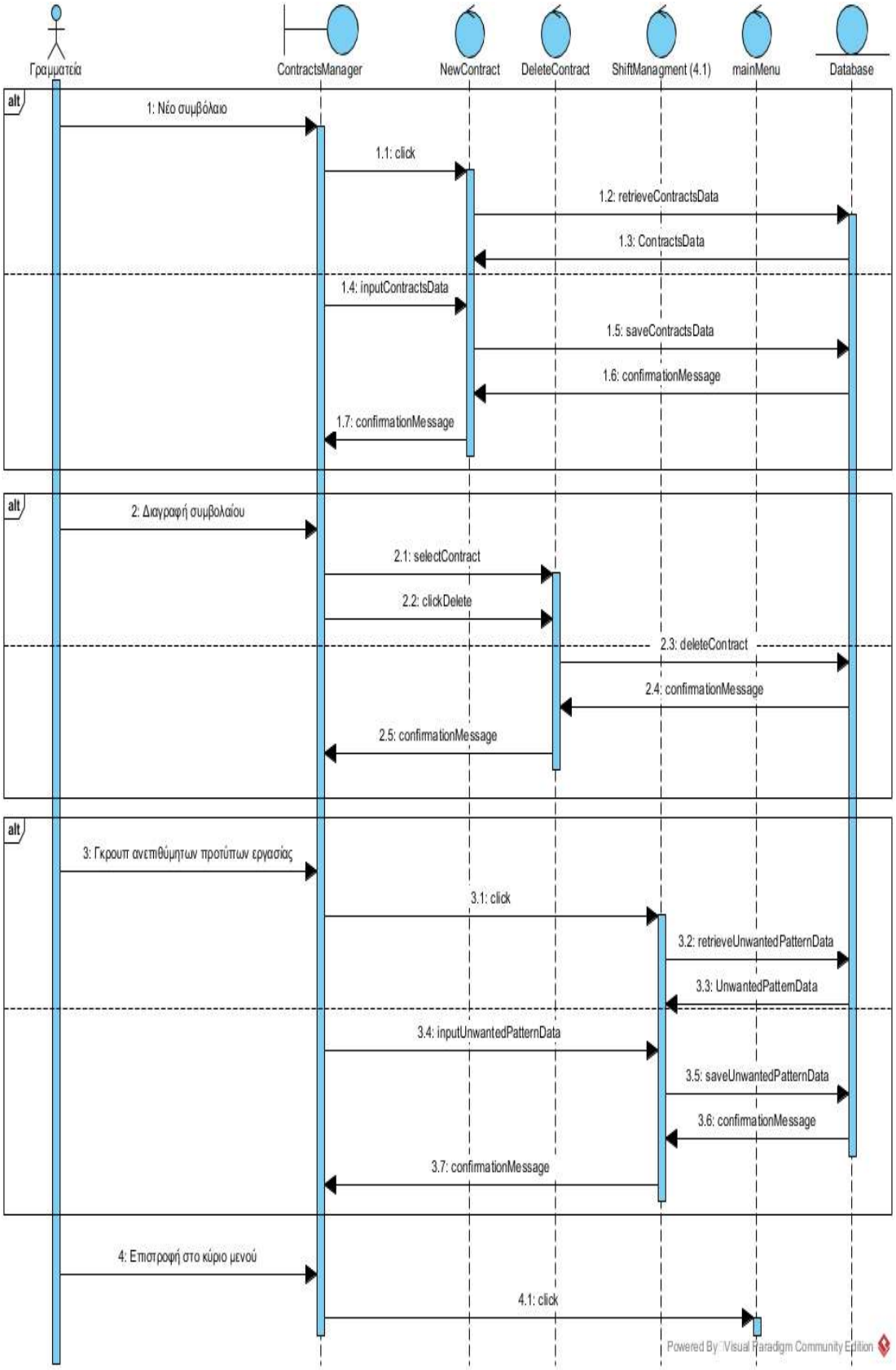
ΔΣ.5. Τα βήματα ΔΣ.1 έως ΔΣ.4 επαναλαμβάνονται όσες φορές επιθυμεί η Γραμματεία

Ο έλεγχος επιστρέφει στο βήμα 3 της βασικής ροής

Ακολούθως κι εδώ, παρουσιάζονται στη συνέχεια τα δύο διαγράμματα, ευρωστίας και ακολουθίας αντίστοιχα, της ΠΧ «Συμβόλαια προγράμματος εργασίας»



Εικόνα 22Διάγραμμα ευρωστίας της ΠΧ "Συμβόλαια προγράμματος εργασίας"



Εικόνα 23: Διάγραμμα ακολουθίας της ΠΧ "Συμβόλαια προγράμματος εργασίας"

Περίπτωση χρήσης: ΠΧ «Σχεδίαση προγράμματος εργασίας»

Κύριος χειριστής: Γραμματεία

Δευτερεύοντες χειριστές: Βάση δεδομένων

Προϋποθέσεις: Η Γραμματεία θα πρέπει να εκτελέσει την εφαρμογή, να έχει εκτελέσει τις ΠΧ «Διαχείριση προσωπικού», «Διαχείριση βαρδιών» και «Συμβόλαια προγράμματος εργασίας» τουλάχιστον μια φορά από την εγκατάσταση του προγράμματος και να πραγματοποιηθεί η εισαγωγή των απαραίτητων δεδομένων από αυτή.

Μετασυνθήκες (Κατάσταση εξόδου): Θα δημιουργηθεί ένα αρχείο κειμένου μορφής .txt το οποίο θα περιλαμβάνει τα δεδομένα που είναι απαραίτητα για τη δημιουργία του προγράμματος εργασίας.

Σύντομη περιγραφή: Η Γραμματεία αφού εκκινήσει τη λειτουργία του αλγορίθμου, με το πέρας της λειτουργίας αυτού, θα έχει στη διάθεσή της ένα αρχείο απλού κειμένου (.txt) το οποίο θα περιλαμβάνει τα δεδομένα εργασίας και ανάπαυσης και των δύο πλευρών της εργασιακής σχέσης (νοσοκόμοι και νοσηλευτικό ίδρυμα) αλλά και πλήθος άλλων δεδομένων (συμβόλαια, ανεπιθύμητες βάρδιες, τύποι βαρδιών κλπ). Το αρχείο αυτό θα χρησιμοποιηθεί ως είσοδος για το εκτελέσιμο αρχείο του VNS αλγορίθμου δημιουργίας προγράμματος εργασίας του προσωπικού του νοσηλευτικού ιδρύματος.

Βασική ροή:

1. Το σύστημα εμφανίζει ένα πίνακα με τα ζητούμενα δεδομένα για τη δημιουργία του προγράμματος.
2. Η Γραμματεία εισάγει όλα τα απαραίτητα δεδομένα στη λίστα ιδιαίτερων απαιτήσεων σε προσωπικό σε συγκεκριμένη ημέρα και συγκεκριμένη βάρδια και πατάει το κουμπί «Προσθήκη στη λίστα»
3. Το σύστημα εισάγει τα δεδομένα στη λίστα προς καταχώρηση στη βάση δεδομένων, αφού πρώτα έχει ελέγξει την ύπαρξή τους σε αυτή.
4. Η Γραμματεία επιλέγει μια ιδιαίτερη απαίτηση εργασίας προσωπικού σε συγκεκριμένη ημέρα και βάρδια από τη σχετική λίστα και πατάει το κουμπί «Αφαίρεση από τη λίστα».
5. Το σύστημα αφαιρεί την επιλεγμένη απαίτηση από τη λίστα των ιδιαίτερων απαιτήσεων εργασίας προσωπικού σε συγκεκριμένη ημέρα και βάρδια.
6. Τα βήματα 4 έως 6 επαναλαμβάνονται όσες φορές το επιθυμεί η Γραμματεία.
7. Η Γραμματεία πατάει το κουμπί «Καθαρισμός λίστας».
8. Το σύστημα απαλείφει όλα τα στοιχεία από τη λίστα των ιδιαίτερων απαιτήσεων εργασίας προσωπικού σε συγκεκριμένη ημέρα και βάρδια.

9. Τα βήματα 2 έως 8 επαναλαμβάνονται μέχρι η Γραμματεία να πατήσει το κουμπί «Αποθήκευση απαιτήσεων».

10. Το σύστημα αποθηκεύει όλες τις επιλογές σχετικά με τις ιδιαίτερες απαιτήσεις εργασίας προσωπικού σε συγκεκριμένη ημέρα και βάρδια στη βάση δεδομένων.

11. Η Γραμματεία εισάγει όλα τα απαραίτητα δεδομένα στη λίστα απαιτήσεων σε προσωπικό για κάθε ημέρα της εβδομάδας σε κάθε βάρδια και πατάει το κουμπί «Προσθήκη στη λίστα».

12. Το σύστημα εισάγει τα δεδομένα στη λίστα προς καταχώρηση στη βάση δεδομένων, αφού πρώτα έχει ελέγξει την ύπαρξή τους σε αυτή.

13. Η Γραμματεία επιλέγει μια απαίτηση σε προσωπικό για εργασία κάθε ημέρας της εβδομάδας και κάθε βάρδιας από τη σχετική λίστα και πατάει το κουμπί «Αφαίρεση από τη λίστα».

14. Το σύστημα αφαιρεί την επιλεγμένη απαίτηση από τη σχετική λίστα

15. Τα βήματα 13 έως 14 επαναλαμβάνονται όσες φορές το επιθυμεί η Γραμματεία.

16. Η Γραμματεία πατάει το κουμπί «Καθαρισμός λίστας».

17. Το σύστημα απαλείφει όλα τα στοιχεία από τη λίστα απαιτήσεων σε προσωπικό για εργασία σε συγκεκριμένη ημέρα και βάρδια.

18. Τα βήματα 11 έως 17 επαναλαμβάνονται μέχρι η Γραμματεία να πατήσει το κουμπί «Καταχώρηση χρονικού διαστήματος προγράμματος».

19. Το σύστημα αποθηκεύει όλες τις επιλογές σχετικά με την κάλυψη σε προσωπικό κάθε ημέρας της εβδομάδας σε κάθε βάρδια στη βάση δεδομένων

20. Η Γραμματεία εισάγει όλα τα δεδομένα σχετικά με τη χρονική διάρκεια του προγράμματος εργασίας και πατάει το κουμπί «Καταχώρηση χρονικού διαστήματος προγράμματος.»

21. Το σύστημα αποθηκεύει όλα τα δεδομένα της χρονικής διάρκειας του προγράμματος στη βάση δεδομένων, αφού έχει ελεγχθεί η ορθότητά τους και η τυχόν ύπαρξή τους σε αυτήν.

22. Η Γραμματεία επιλέγει μια ενέργεια [Εναλλακτική ροή ΠΡ: «Προεπισκόπηση και αλλαγές ρυθμίσεων προγράμματος»].

23. Η Γραμματεία πατάει το κουμπί «Δημιουργία προγράμματος εργασίας».

24. Το σύστημα ζητά επιβεβαίωση από τη Γραμματεία για την ορθότητα του χρονικού διαστήματος του προγράμματος εργασίας και για την εκκίνηση της διαδικασίας δημιουργίας.

25. Το σύστημα εκκινεί τον αλγόριθμο δημιουργίας προγράμματος εργασίας.

26. Το σύστημα μετά το πέρας των διαδικασιών δημιουργίας του προγράμματος εργασίας, εμφανίζει το δημιουργημένο πρόγραμμα εργασίας στη Γραμματεία [Εναλλακτική ροή η ΠΧ «Διαχείριση προγράμματος εργασίας»].

27. Η Γραμματεία επιστρέφει στο κεντρικό μενού, πατώντας το κουμπί «Επιστροφή στο κεντρικό μενού».

28. Η ΠΧ τελειώνει.

Εναλλακτική ροή ΠΡ: «Αφαίρεση από τη λίστα»

ΠΡ.1. Το σύστημα εμφανίζει ένα πίνακα με όλες τις ρυθμίσεις που έχει κάνει η Γραμματεία για τη διαχείριση του προσωπικού, τις απαιτήσεις του, τη διαχείριση βαρδιών και συμβολαίων αλλά και τη διάρκεια των προγραμμάτων που έχουν αποθηκευτεί στη βάση δεδομένων.

ΠΡ.2. Η Γραμματεία κάνει μια επιλογή από τις εμφανιζόμενες περιόδους προγραμματισμού.

ΠΡ.3. Το σύστημα εμφανίζει τις αιτήσεις εργασίας και ανάπαυσης του προσωπικού για το χρονικό διάστημα που ορίζεται από την προηγούμενη επιλογή της Γραμματείας.

ΠΡ.4. Τα βήματα ΠΡ.2 έως ΠΡ.3 επαναλαμβάνονται όσες φορές το επιθυμεί η Γραμματεία.

ΠΡ.5. Η Γραμματεία κάνει μια επιλογή από την εμφανιζόμενη λίστα των νοσοκόμων.

ΠΡ.6. Το σύστημα εμφανίζει όλες τις αιτήσεις εργασίας και ανάπαυσης του προσωπικού καθώς και τις ειδικότητες τις οποίες κατέχει.

ΠΡ.7. Τα βήματα ΠΡ.5 έως ΠΡ.6 επαναλαμβάνονται όσες φορές το επιθυμεί η Γραμματεία.

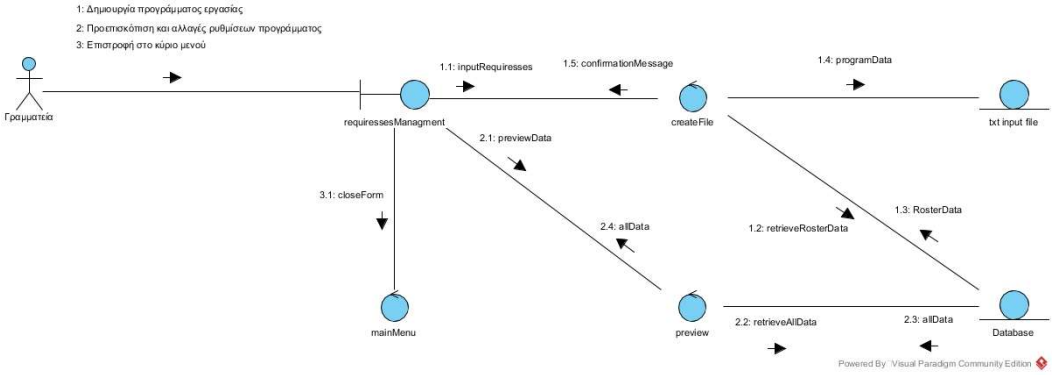
ΠΡ.8. Η Γραμματεία αλλάζει τα δεδομένα σε συγκεκριμένα πεδία και πατάει το κουμπί «Αποθήκευση αλλαγών».

ΠΡ.9. Το σύστημα αποθηκεύει τις αλλαγές που έκανε η Γραμματεία στη βάση δεδομένων.

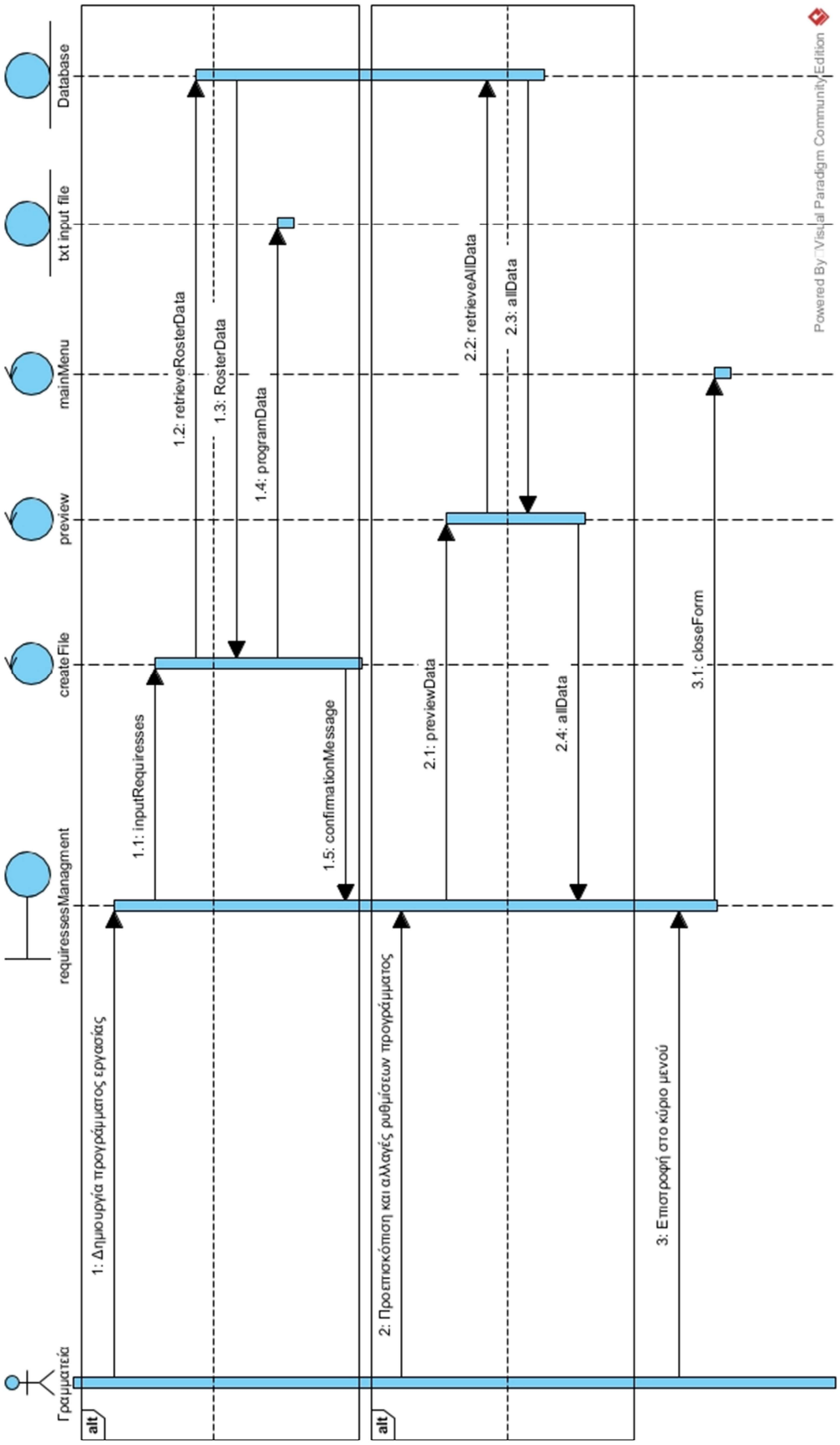
ΠΡ.10. Τα βήματα ΠΡ.2 έως ΠΡ.9 επαναλαμβάνονται μέχρι η Γραμματεία να πατήσει το κουμπί «Επιστροφή στη δημιουργία προγράμματος».

Ο έλεγχος επιστρέφει στο βήμα 16 της βασική ροής

Στις επόμενες δύο εικόνες, παρουσιάζονται τα δύο διαγράμματα, ευρωστίας και ακολουθίας της ΠΧ «Σχεδίαση προγράμματος εργασίας»



Εικόνα 24: Διάγραμμα ευρωστίας της ΠΧ "Σχεδίαση προγράμματος εργασίας"



Εικόνα 25: Διάγραμμα ακολουθίας της ΠΧ "Σχεδίαση προγράμματος εργασίας"

Περίπτωση χρήσης: ΠΧ «Διαχείριση προγράμματος εργασίας»

Κύριος χειριστής: Γραμματεία

Δευτερεύοντες χειριστές: Βάση δεδομένων, Εκτυπωτής.

Προϋποθέσεις: Η Γραμματεία θα πρέπει να εκτελέσει την εφαρμογή και την ΠΧ «Δημιουργία προγράμματος εργασίας» και να έχει δημιουργηθεί το αρχείο με το πρόγραμμα εργασίας επιτυχώς.

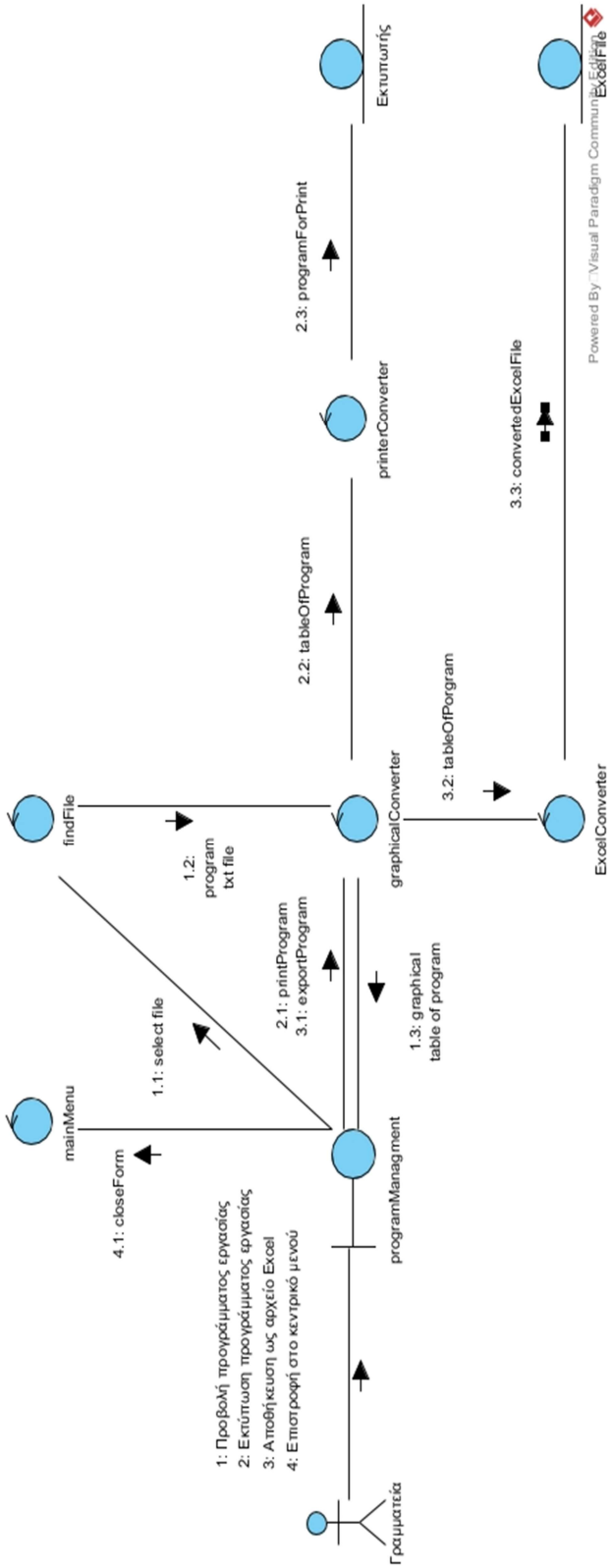
Μετασυνθήκες (Κατάσταση εξόδου): Θα γίνεται γραφική απεικόνιση του προγράμματος εργασίας. Θα μπορεί να εξαχθεί σε αρχείο μορφής Excel (.xlsx) αλλά και να εκτυπώνεται.

Σύντομη περιγραφή: Η Γραμματεία μετά την δημιουργία του αρχείου του προγράμματος εργασίας, θα μπορεί να επιλέξει από ένα μενού, την εκτύπωση του προγράμματος από το αρχείο κειμένου, την μετατροπή και αποθήκευσή του σε αρχείο μορφής Excel αλλά και τη γραφική απεικόνισή του.

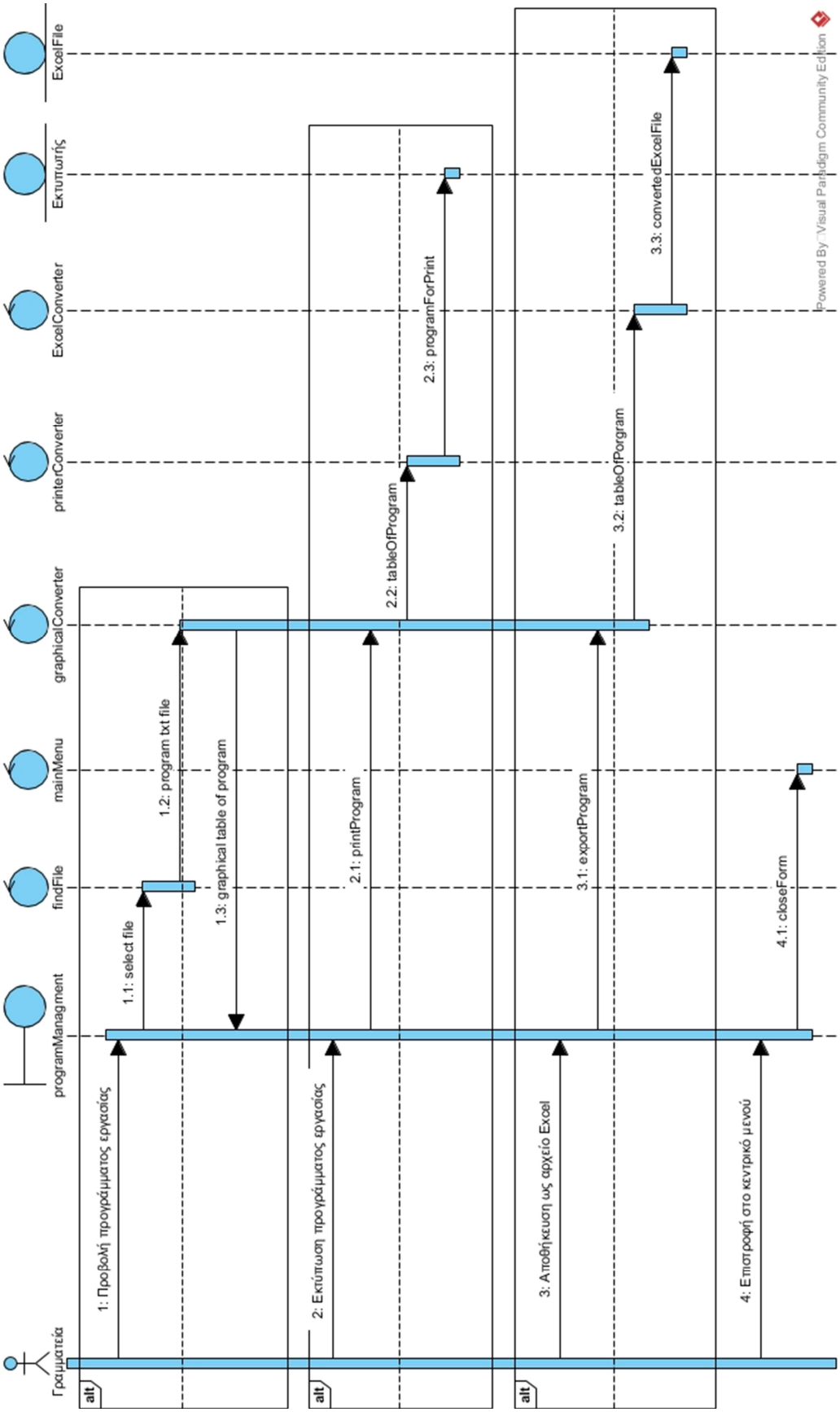
Βασική ροή:

1. Το σύστημα εμφανίζει ένα πίνακα για να επιλέξει η Γραμματεία το είδος του προγράμματος που αναζητεί να εμφανίσει και πατάει το κουμπί «Προβολή».
2. Το σύστημα εμφανίζει το τελευταίο δημιουργημένο πρόγραμμα εργασίας των νοσοκόμων. Αν αυτό δεν υπάρχει, τότε το σύστημα ενημερώνει σχετικά τη Γραμματεία.
3. Η Γραμματεία πατάει το κουμπί «Εκτύπωση του προγράμματος εργασίας».
4. Το σύστημα εμφανίζει ένα πίνακα ο οποίος διαχειρίζεται την εκτύπωση.
5. Η Γραμματεία αφού πραγματοποιήσει τις επιθυμητές επιλογές, πατάει το κουμπί «Εκτύπωση».
6. Το σύστημα εκτυπώνει σε χαρτί το δημιουργημένο πρόγραμμα εργασίας και το επιστρέφει στη Γραμματεία.
7. Η Γραμματεία πατάει το κουμπί «Εξαγωγή προγράμματος σε αρχείο Excel».
8. Το σύστημα ζητάει από τη Γραμματεία το όνομα και την τοποθεσία αποθήκευσης του προγράμματος εργασίας σε αρχείο Excel μορφής .xlsx.
9. Η Γραμματεία εισάγει τα ζητούμενα δεδομένα και πατάει το κουμπί «Αποθήκευση».
10. Το σύστημα αποθηκεύει το πρόγραμμα εργασίας με το επιθυμητό όνομα και στην επιθυμητή τοποθεσία της Γραμματείας.
11. Το σύστημα παρουσιάζει τα περιεχόμενα του αρχείου Excel στη Γραμματεία.
12. Η Γραμματεία επιστρέφει στο κεντρικό μενού, πατώντας το κουμπί «Επιστροφή στο κεντρικό μενού».
13. Η ΠΧ τελειώνει.

Τέλος, τα αντίστοιχα διαγράμματα ευρωστίας και ακολουθίας για την ΠΧ «Διαχείριση προγράμματος εργασίας», φαίνονται στις παρακάτω δύο εικόνες:



Εικόνα 26: Διάγραμμα ευρωστίας της ΠΧ "Διαχείριση προγράμματος εργασίας"


Γραμματεία
programManagment
findFile
mainMenu
graphicalConverter
printerConverter
ExcelConverter
Εκτυπωτής
ExcelFile
alt
1: Προβολή προγράμματος εργασίας
1.1: select file
1.2: program txt file
1.3: graphical table of program
alt
2: Εκτύπωση προγράμματος εργασίας
2.1: printProgram
2.2: tableOfProgram
2.3: programForPrint
alt
3: Αποθήκευση ως αρχείο Excel
3.1: exportProgram
3.2: tableOfPorgram
3.3: convertedExcelFile
4: Επιστροφή στο κεντρικό μενού
4.1: closeForm
Powered By Visual Paradigm Community Edition

Εικόνα 27:Διάγραμμα ακολουθίας της ΠΧ "Διαχείριση προγράμματος εργασίας"

5.4 Διαγράμματα κλάσεων

Εφόσον έχουν ολοκληρωθεί και τα διαγράμματα ακολουθίας, είναι δυνατή πλέον η δημιουργία των κλάσεων του προγράμματος. Τις κλάσεις θα τις τοποθετήσουμε σε 5 πακέτα τα οποία θα χαρακτηρίζουν και ένα σχετικό αντικείμενο. Τα πακέτα αυτά είναι:

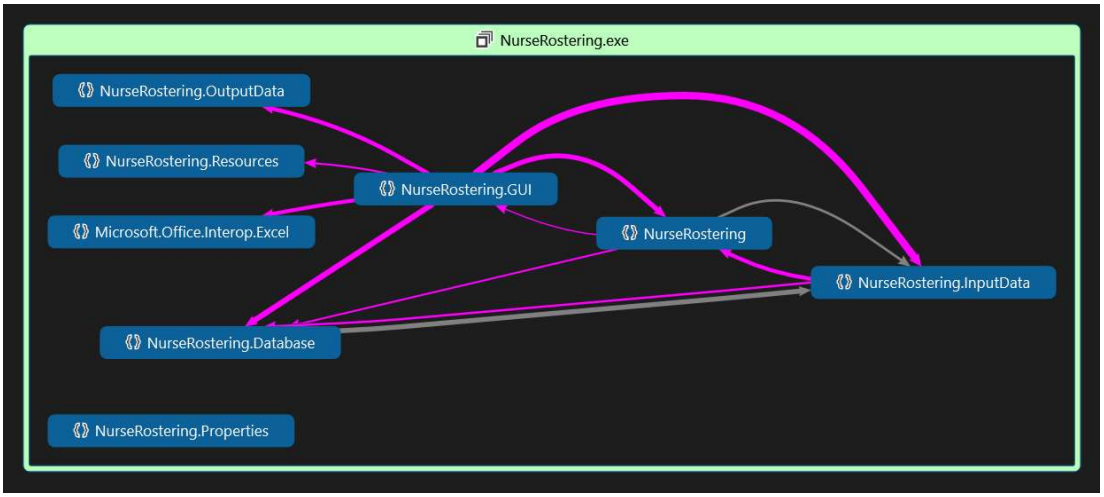
- **InputData**, το οποίο περιέχει τις κλάσεις οι οποίες αναπαριστούν τα κεφάλαια που περιέχουν τα δεδομένα εισόδου στο αρχείο εισόδου του αλγορίθμου.
- **Database**, το οποίο αναπαριστά τη βάση δεδομένων. Περιέχει δύο κλάσεις από τις οποίες η μια είναι η βάση δεδομένων (NurseRosteringDB) και η άλλη είναι ένας χάρτης της βάσης δεδομένων (MyDbConfiguration) για να εκκινεί ταχύτερα η λειτουργία της βάσης δεδομένων μέσα από την εφαρμογή. Κατά την πρώτη σύνδεση με τη βάση δεδομένων, η εφαρμογή δεν δημιουργεί νέο χάρτη της βάσης στη μνήμη, αλλά φορτώνει τον ήδη υπάρχον, γλυτώνοντας το χρήστη μερικά δευτερόλεπτα. Στις επόμενες επικοινωνίες με τη βάση δεδομένων, απαιτούμενος χρόνος είναι ελάχιστα δυνατός.
- **OutputData**, το οποίο περιέχει τις κλάσεις οι οποίες αναπαριστούν τα κεφάλαια που περιέχουν τα δεδομένα εξόδου στο αρχείο εξόδου που δημιουργήθηκε από τον αλγόριθμο.
- **GUI**, το οποίο περιέχει τις κλάσεις των φορμών της γραφικής διεπαφής αλληλεπίδρασης της Γραμματείας με την εφαρμογή.
- **Resources**, το οποίο περιέχει είτε τις κλάσεις που δεν μπορούν να τοποθετηθούν σε κάποιο άλλο πακέτο είτε τις κλάσεις οι οποίες περιέχουν μεθόδους οι οποίες είναι αξιοποιήσιμες από όλες ή τις περισσότερες άλλες κλάσεις.

Τα πακέτα συνδέονται μεταξύ τους μέσω των κλάσεων. Τα πακέτα δεν είναι κάποια οντότητα όπως οι κλάσεις, αλλά μπορούμε να πούμε ότι είναι απλώς μια έννοια με την οποία κατηγοριοποιούμε κλάσεις σύμφωνα με το περιεχόμενό τους ή τη λειτουργία τους. Στην παρακάτω εικόνα βλέπουμε τη σύνδεση των πακέτων που αναπαριστά ουσιαστικά, όλη την κίνηση των δεδομένων και όλες τις ενέργειες που γίνονται στην εφαρμογή κατά τη λειτουργία της. Υπάρχουν και μερικά αντικείμενα, τα οποία είτε δεν είναι πακέτα είτε είναι κλάσεις και δεν έχουν ενταχθεί σε κάποιο πακέτο. Αυτά είναι:

- **Microsoft.Office.Interop.Excel**, με το οποίο εισάγεται στην εφαρμογή NurseRostering η μηχανή του Excel η οποία είναι απαραίτητη για τη δημιουργία και προβολή του αρχείου Excel που θα δημιουργηθεί από τη Γραμματεία. Γι' αυτή την

ενέργεια, είναι απαραίτητο να βρίσκεται εγκατεστημένο το Microsoft Office Excel στον υπολογιστή που εκτελείται η εφαρμογή NurseRostering.

- **NurseRostering.Properties**, το οποίο περιέχει πληροφορίες σχετικές με την εφαρμογή.



Εικόνα 28: Διασύνδεση πακέτων μέσα στην εφαρμογή NurseRostering

Στη συνέχεια, θα δούμε αναλυτικά τι περιέχει το κάθε πακέτο, σε επίπεδο κλάσεων. Για λόγους ευκρίνειας των περιεχομένων των κλάσεων, ως διάγραμμα κλάσεων θεωρείται το διάγραμμα του εννοιολογικού μοντέλου διότι είναι ακριβώς το ίδιο. Παρακάτω θα παρουσιαστούν τα περιεχόμενα των κλάσεων αναλυτικά. Πριν από αυτό, παρατίθεται ένα μικρό υπόμνημα, στο οποίο ερμηνεύονται τα σύμβολα που θα παρουσιαστούν μέσα στις κλάσεις.

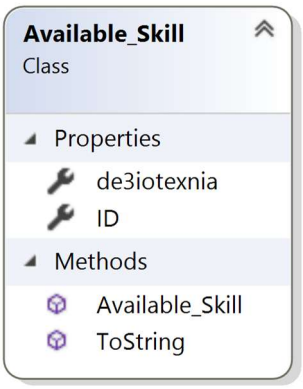
Επεξήγηση συμβόλων των παρακάτω κλάσεων

- Ιδιωτική μεταβλητή (Private Field)
- Προστατευμένη μεταβλητή (Protected Field)
- Δημόσια μεταβλητή (Public Field)
- Ιδιότητα (Property)
- Ιδιωτική μέθοδος (Private Method)
- Προστατευμένη μέθοδος (Protected Method)
- Δημόσια μέθοδος (Public Method)
- Κληρονομικότητα (Inheritance)

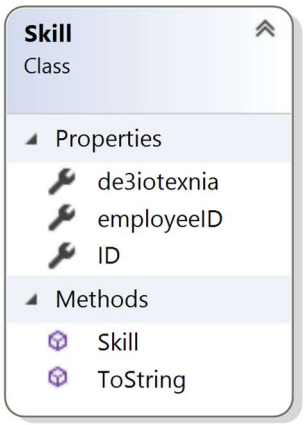
- Στο πακέτο GUI, τα πεδία και οι μέθοδοι τα οποία είναι αντικείμενα της φόρμας ή μέθοδοι αυτόματα δημιουργημένες χρωματίζονται με γκρι χρώμα ενώ αυτά που δημιουργήθηκαν από τον προγραμματιστή χρωματίζονται με μαύρο χρώμα.

5.4.1. Πακέτο InputData

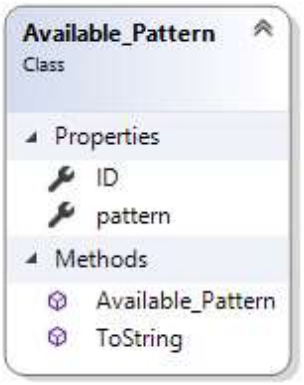
Οι κλάσεις που έχουν δημιουργηθεί σε αυτό το πακέτο, αναπαριστούν τα κεφάλαια του αρχείου εισόδου του αλγορίθμου. Κάθε μια φέρει τα σχετικά δεδομένα τα οποία είτε εισάγει η Γραμματεία, είτε αντλεί από τη βάση δεδομένων. Εξαίρεση από το αρχείο εισόδου, αποτελούν οι δύο κλάσεις *Available_Pattern* και *Available_Skill*, οι οποίες είναι βοηθητικές για τα αντίστοιχα δεδομένα. Αυτές, χρησιμοποιούνται ως κατάλογοι με τα διαθέσιμα στοιχειώδη πρότυπα ανεπιθύμητης εργασίας και τις διαθέσιμες ειδικότητες για να μπορέσει η Γραμματεία να πραγματοποιήσει τις επιλογές της μέσα από αυτούς. Παρακάτω φαίνονται οι κλάσεις, με τα περιεχόμενά τους, αυτού του πακέτου:



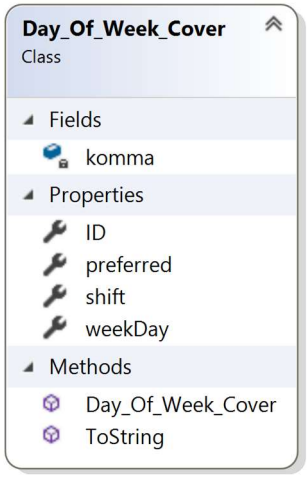
Εικόνα 34: Κλάση Available_Skill



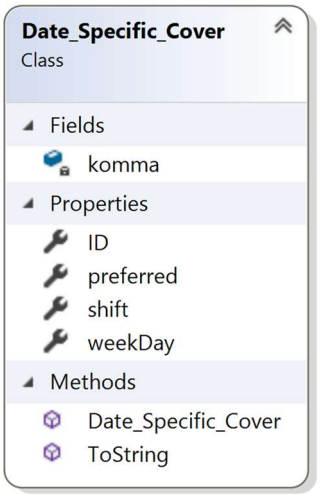
Εικόνα 30: Κλάση Skill



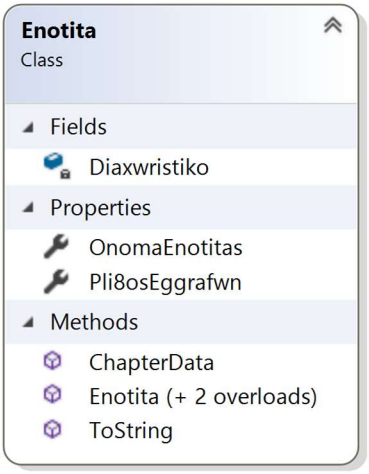
Εικόνα 32: Κλάση Available_Pattern



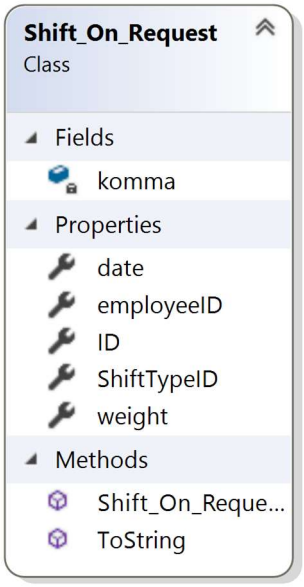
Εικόνα 31: Κλάση Day_Of_Week_Cover



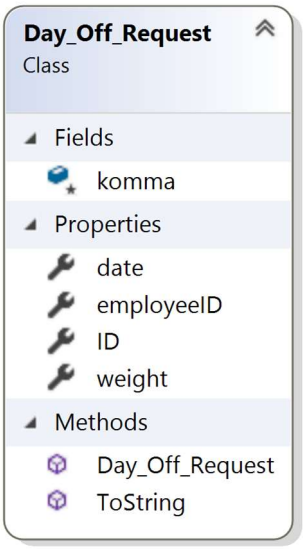
Εικόνα 33: Κλάση Date_Specific_Cover



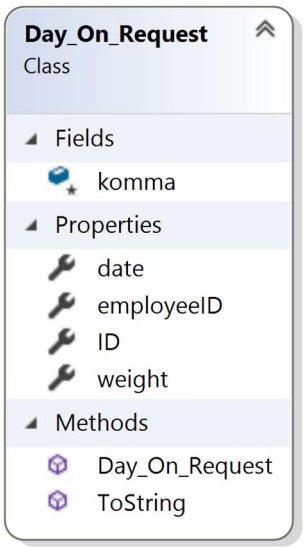
Εικόνα 29: Κλάση Enotita



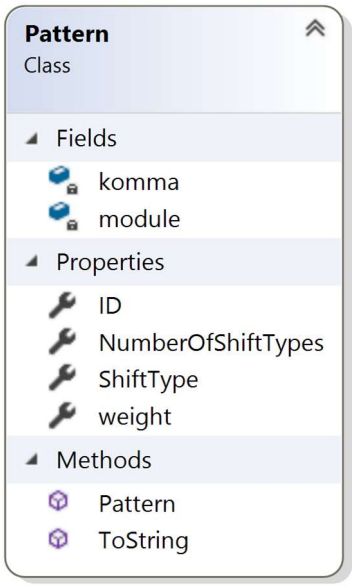
Εικόνα 35: Κλάση Shift_On_Request



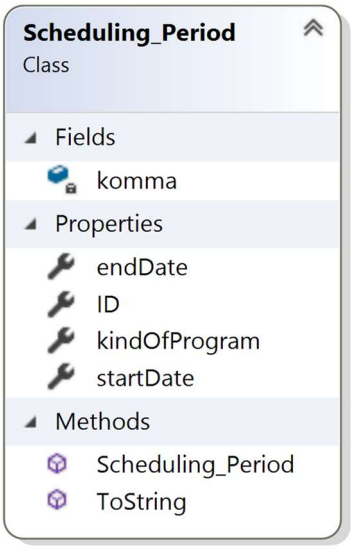
Εικόνα 36: Κλάση Day_Off_Request



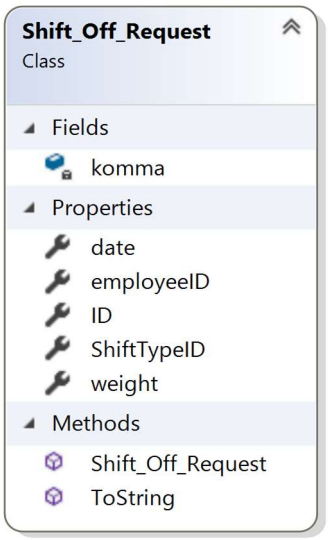
Εικόνα 37: Κλάση Day_On_Request



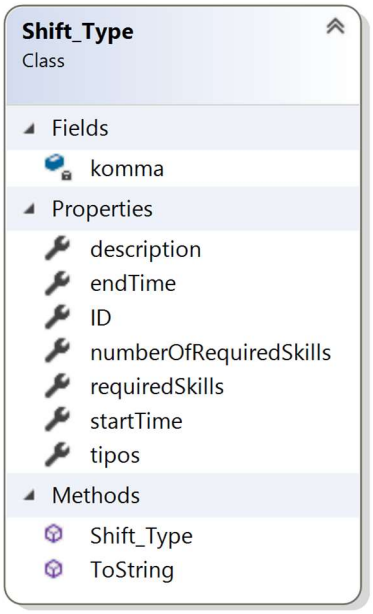
Εικόνα 39: Κλάση Pattern



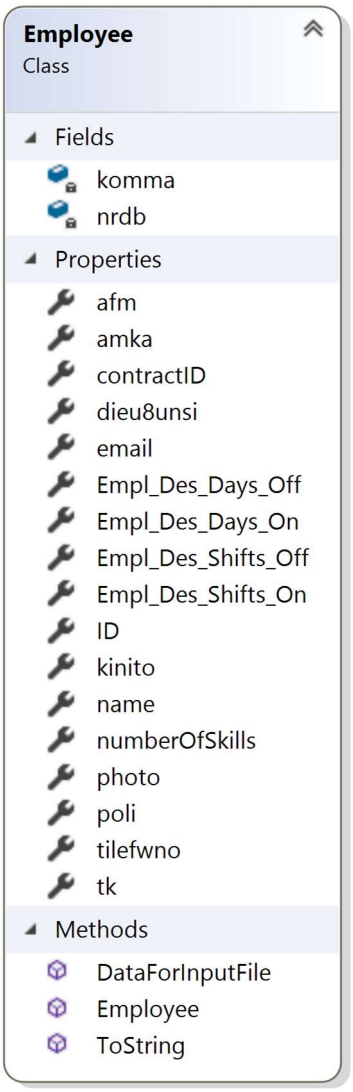
Εικόνα 40: Κλάση Scheduling_Period



Εικόνα 38: Κλάση Shift_Off_Request



Εικόνα 41: Κλάση Shift_Type

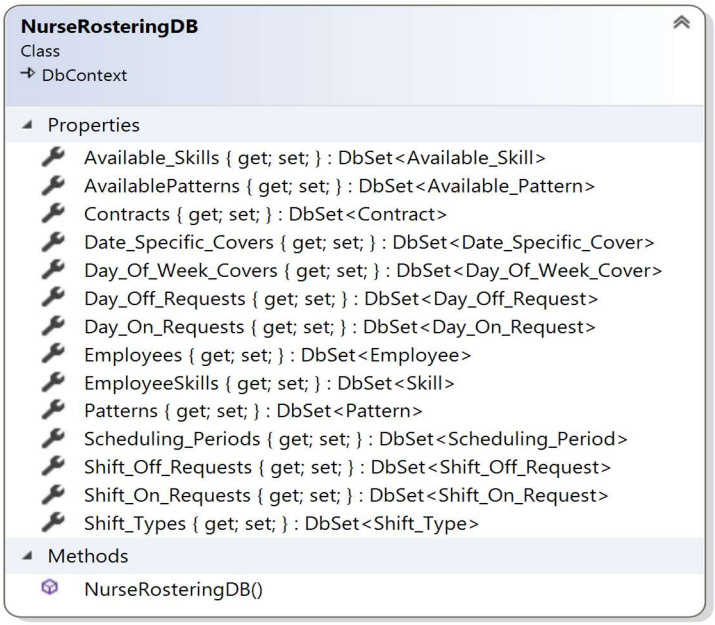


Εικόνα 42: Κλάση Employee

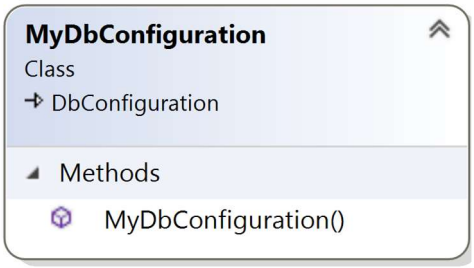
Εικόνα 43: Κλάση Contracts

5.4.2. Πακέτο DataBase

Το πακέτο αυτό περιλαμβάνει δύο κλάσεις, την *NurseRosteringDB* η οποία είναι και η βάση δεδομένων που χρησιμοποιείται μέσα από το Entity Framework και την *MyDbConfiguration* η οποία δημιουργεί ένα μοντέλο – χάρτη της υπάρχουσας βάσης δεδομένων έτσι ώστε κατά την πρώτη επικοινωνία της εφαρμογής με τη βάση δεδομένων να μη δημιουργείται κάθε φορά το μοντέλο της βάσης δεδομένων στη μνήμη του συστήματος, αλλά να αναγιγνώσκεται από το σχετικό αρχείο για ταχύτερη ανταπόκριση. Οι δύο κλάσεις με τα περιεχόμενά τους, φαίνονται παρακάτω:



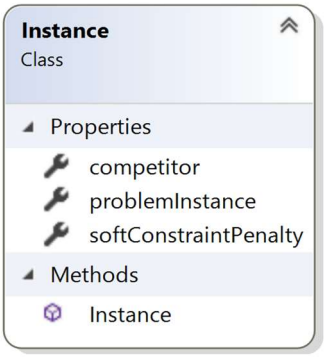
Εικόνα 44: Κλάση NurseRosteringDB



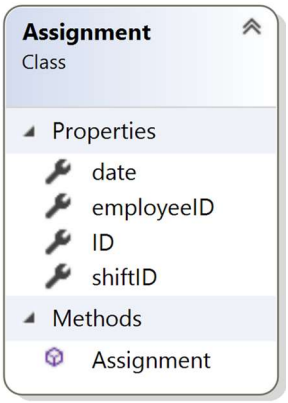
Εικόνα 45: Κλάση MyDbConfiguration

5.4.3. Πακέτο OutputData

Οι δύο κλάσεις αυτού του πακέτου είναι η *Instance* και η *Assignments*. Στην μέχρι τώρα υλοποίηση της εφαρμογής, έχει χρησιμοποιηθεί μόνο η κλάση Assignments, η οποία αναπαριστά την εργασία ενός συγκεκριμένου νοσοκόμου σε συγκεκριμένη ημέρα και σε συγκεκριμένη βάρδια. Πολλά στιγμιότυπα (αντικείμενα) της κλάσης αυτής, αναπαριστούν το πρόγραμμα εργασίας όλων των νοσοκόμων στο προκαθορισμένο χρονικό διάστημα ισχύος του προγράμματος εργασίας τους. Η κλάση Instance περιέχει πληροφορίες σχετικές με τη λύση του προβλήματος από τον αλγόριθμο. Επί του παρόντος, δεν κρίθηκε απαραίτητη η παροχή αυτών των πληροφοριών στη Γραμματεία. Οι δύο, αυτές, κλάσεις με τα περιεχόμενά τους, είναι:



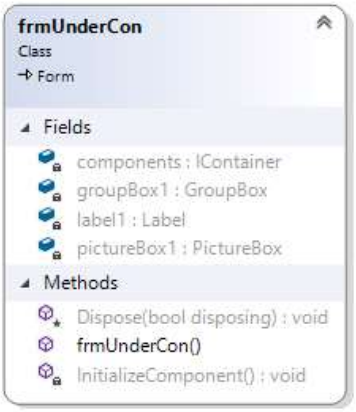
Εικόνα 46: Κλάση Instance



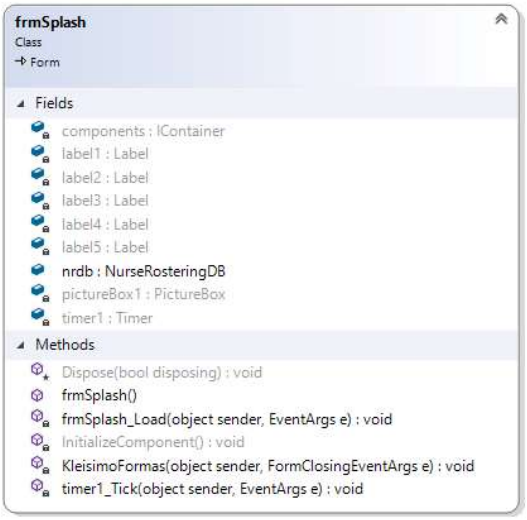
Εικόνα 47: Κλάση Assignment

5.4.4. Πακέτο GUI

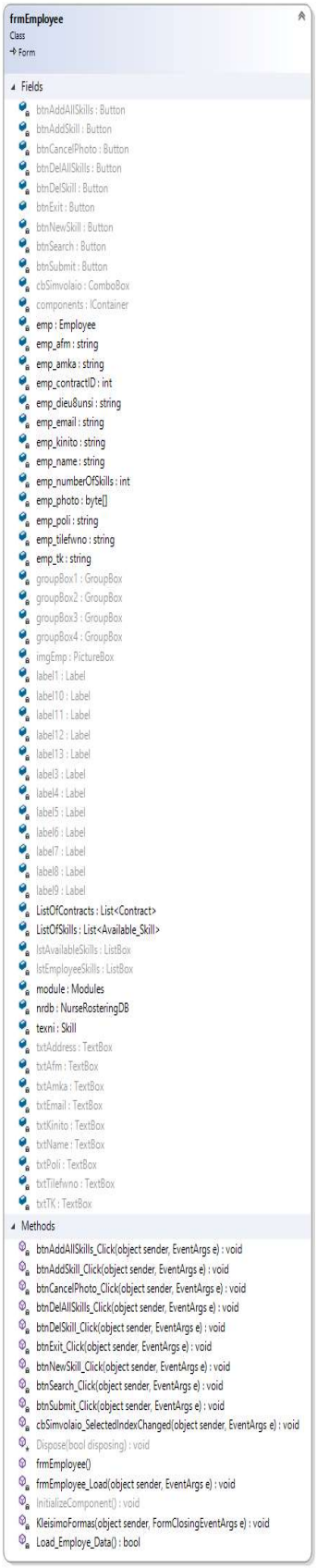
Οι κλάσεις που περιέχονται σε αυτό το πακέτο, είναι οι κλάσεις της γραφικής διεπαφής αλληλεπίδρασης της εφαρμογής με το χρήστη. Μέσα από αυτές, η Γραμματεία εισάγει όλα τα απαραίτητα δεδομένα για τη δημιουργία του προγράμματος, διατηρεί τα στοιχεία των νοσοκόμων συγκροτημένα σε ατομικές καρτέλες και γενικότερα φροντίζει για την ορθή δημιουργία του προγράμματος εργασίας των νοσοκόμων. Οι φόρμες με τα περιεχόμενά τους είναι οι παρακάτω:



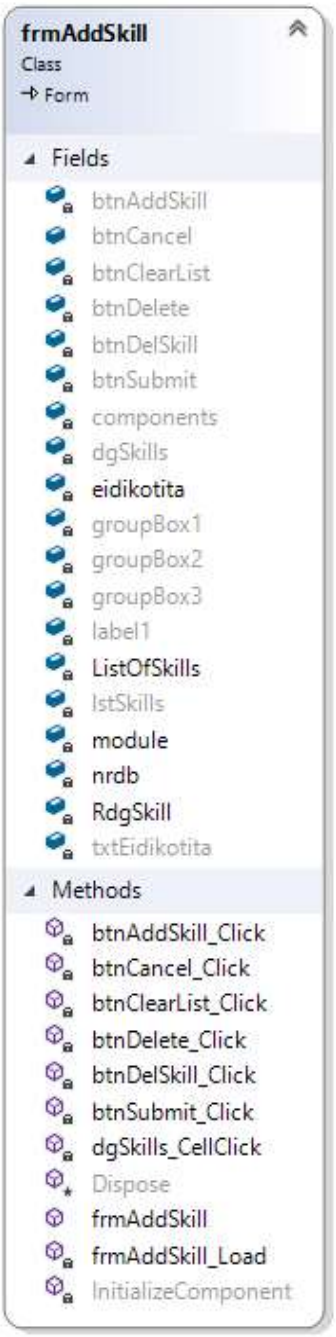
Εικόνα 48: Κλάση φόρμας frmUnderCon



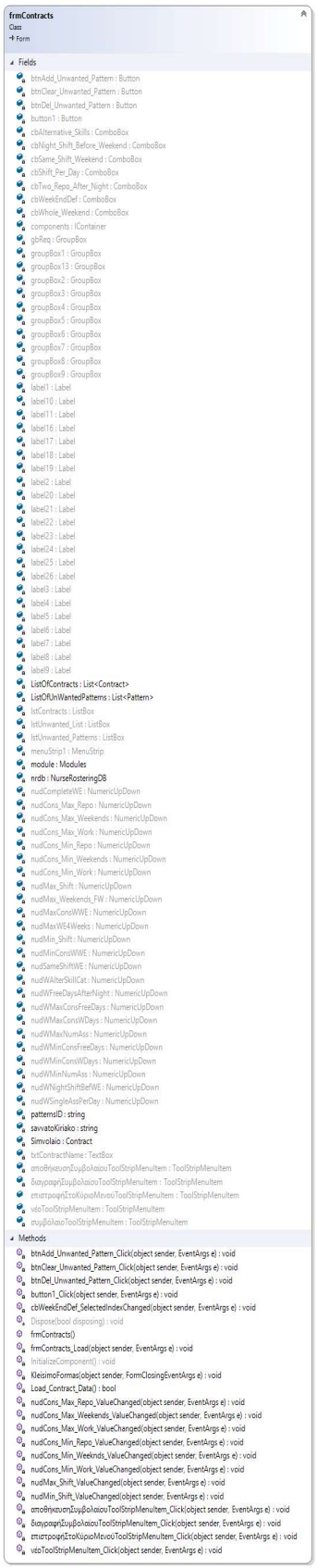
Εικόνα 49: Κλάση φόρμας frmSplash



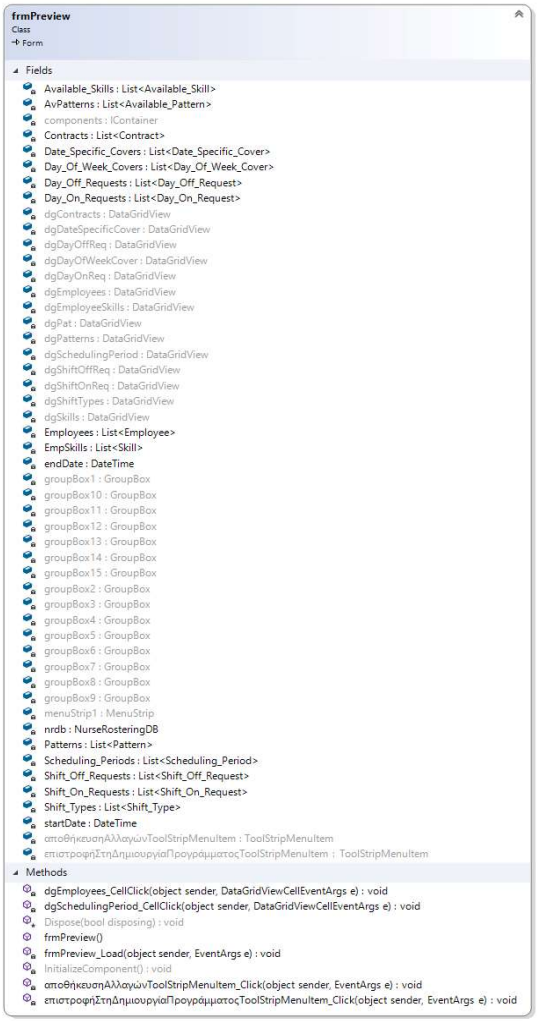
Εικόνα 50: Κλάση φόρμας frmEmployee



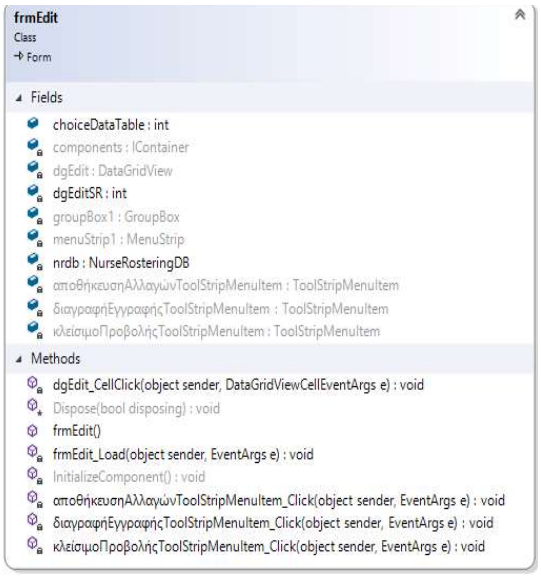
Εικόνα 51: Κλάση φόρμας frmAddSkill



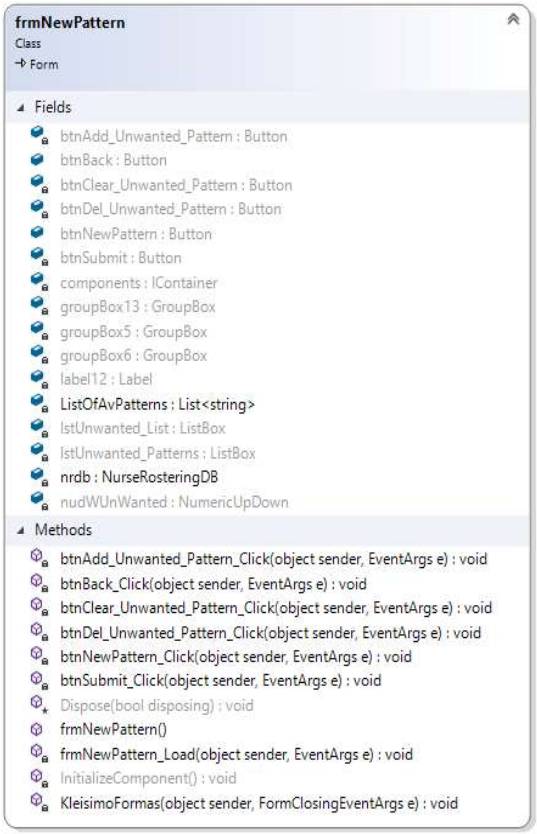
Εικόνα 52: Κλάση φόρμας frmContracts



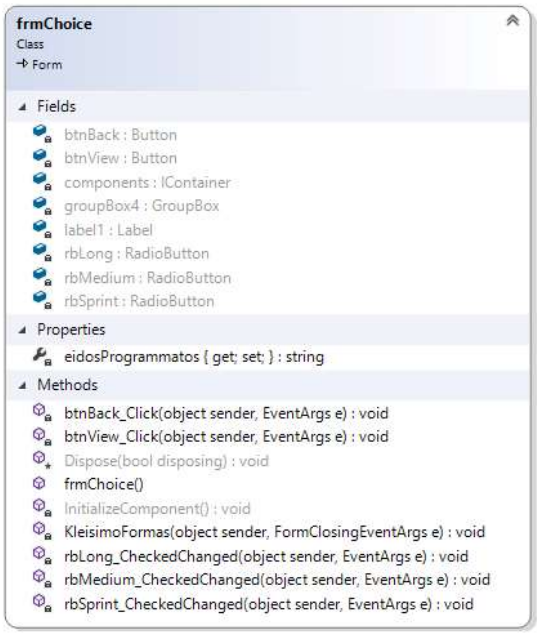
Εικόνα 56: Κλάση φόρμας frmPreview



Εικόνα 54: Κλάση φόρμας frmEdit



Εικόνα 55: Κλάση φόρμας frmNewPattern



Εικόνα 53: Κλάση φόρμας frmChoice

frmEmpWork
Class
→ Form

Fields
- aitima_ergasias : string
- aitima_repo : string
- aitima_vardias_ergasias : string
- aitima_vardias_repo : string
- btnAdd_Repo_Day : Button
- btnAdd_Repo_Shift : Button
- btnAdd_Work_Day : Button
- btnAdd_Work_Shift : Button
- btnBack : Button
- btnCancel : Button
- btnClear_Repo_Days : Button
- btnClear_Repo_Shift : Button
- btnClear_Work_Days : Button
- btnClear_Work_Shift : Button
- btnCorrect : Button
- btnDel_Repo_Day : Button
- btnDel_Repo_Shift : Button
- btnDel_Work_Day : Button
- btnDel_Work_Shift : Button
- btnNewEmp : Button
- btnNewShift : Button
- cbShiftOff : ComboBox
- cbShiftOn : ComboBox
- components : IContainer
- DOffReq : Day_Off_Request
- DOnReq : Day_On_Request
- empl : Employee
- groupBox1 : GroupBox
- groupBox10 : GroupBox
- groupBox11 : GroupBox
- groupBox12 : GroupBox
- groupBox2 : GroupBox
- groupBox5 : GroupBox
- groupBox6 : GroupBox
- idNosokomou : string
- komma : string
- label1 : Label
- label12 : Label
- label13 : Label
- label14 : Label
- label15 : Label
- label2 : Label
- lst_Repo_Shift : ListBox
- lst_Req_Shift : ListBox
- lstNosokomoi : ListBox
- lstRepo_Days : ListBox
- lstWorking_Days : ListBox
- mc_Repo_Day : MonthCalendar
- mc_Repo_Shift : MonthCalendar
- mc_Requested_Shift : MonthCalendar
- mc_Work_Day : MonthCalendar
- module : Modules
- nrdb : NurseRosteringDB
- nudDayOffReq : NumericUpDown
- nudDayOnReq : NumericUpDown
- nudDesWorkShifts : NumericUpDown
- nudUnwantWorkShifts : NumericUpDown
- SOffReq : Shift_Off_Request
- SOnReq : Shift_On_Request
- tabRepo : TabPage
- tabReqDays : TabControl
- tabShiftRepo : TabPage
- tabShiftWork : TabPage
- tabWork : TabPage

Properties
- EmployeeList { get; set; } : List<Employee>

Methods
- btnAdd_Repo_Day_Click(object sender, EventArgs e) : void
- btnAdd_Repo_Shift_Click(object sender, EventArgs e) : void
- btnAdd_Work_Day_Click(object sender, EventArgs e) : void
- btnAdd_Work_Shift_Click(object sender, EventArgs e) : void
- btnClear_Repo_Days_Click(object sender, EventArgs e) : void
- btnClear_Repo_Shift_Click(object sender, EventArgs e) : void
- btnClear_Work_Days_Click(object sender, EventArgs e) : void
- btnClear_Work_Shift_Click(object sender, EventArgs e) : void
- btnCorrect_Click(object sender, EventArgs e) : void
- btnDel_Repo_Day_Click(object sender, EventArgs e) : void
- btnDel_Repo_Shift_Click(object sender, EventArgs e) : void
- btnDel_Work_Day_Click(object sender, EventArgs e) : void
- btnDel_Work_Shift_Click(object sender, EventArgs e) : void
- btnNewEmp_Click(object sender, EventArgs e) : void
- btnNewShift_Click(object sender, EventArgs e) : void
- btnSubmit_Click(object sender, EventArgs e) : void
- button1_Click(object sender, EventArgs e) : void
- Dispose(bool disposing) : void
- EmfanisiNosokomwn() : void
- frmEmpWork()
- frmEmpWork_Load(object sender, EventArgs e) : void
- InitializeComponent() : void
- KleisimoFormas(object sender, FormClosingEventArgs e) : void
- lst_Repo_Shift_SelectedIndexChanged(object sender, EventArgs e) : void
- lst_Req_Shift_SelectedIndexChanged(object sender, EventArgs e) : void
- lstNosokomoi_SelectedIndexChanged(object sender, EventArgs e) : void
- lstRepo_Days_SelectedIndexChanged(object sender, EventArgs e) : void
- lstWorking_Days_SelectedIndexChanged(object sender, EventArgs e) : void

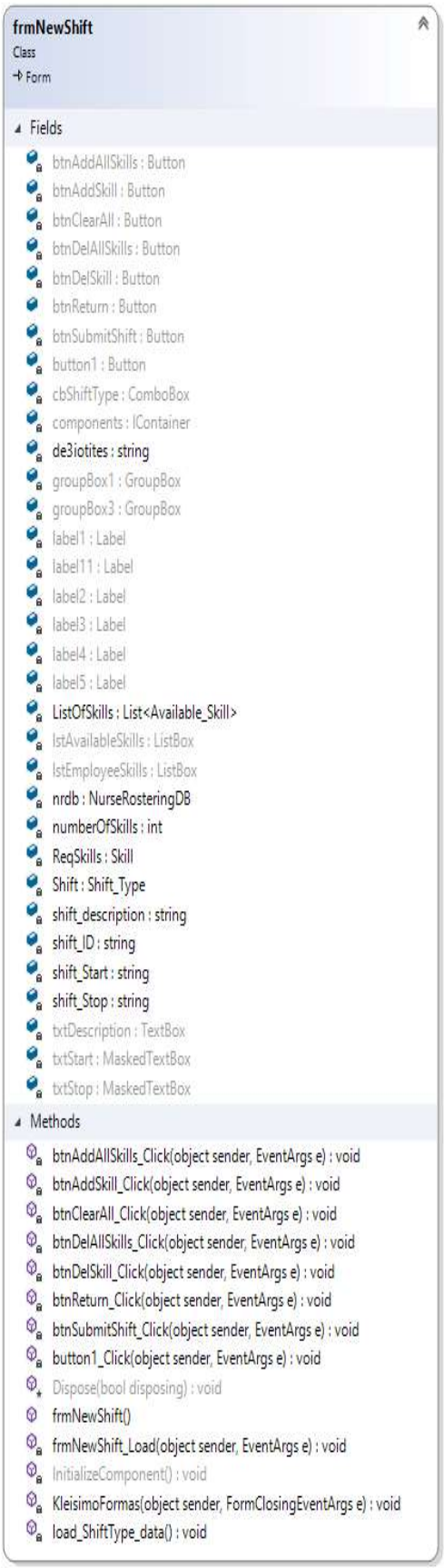
Εικόνα 57: Κλάση φόρμας frmEmpWork

frmMain
Class
→ Form

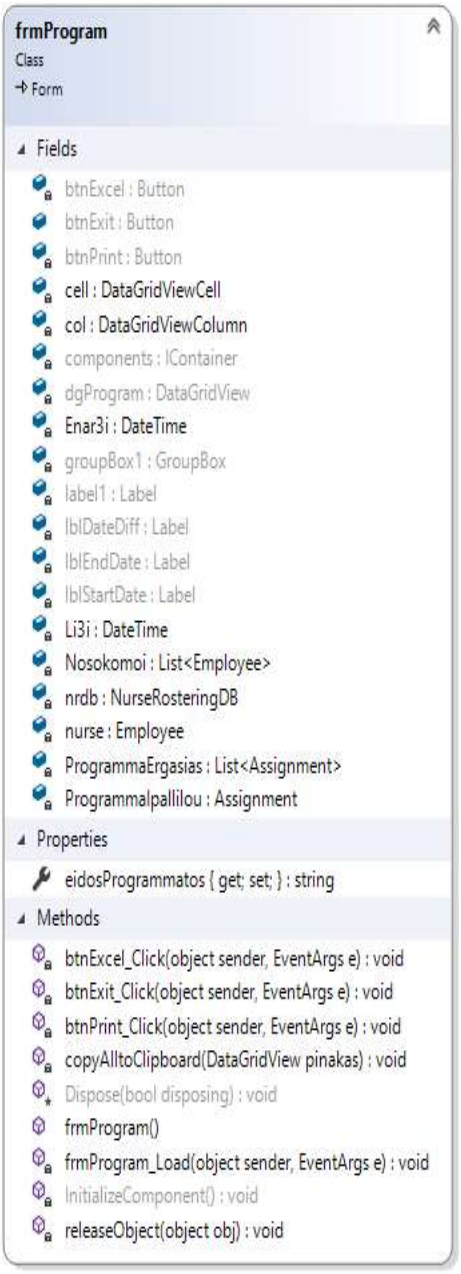
Fields
- btnAddShift : Button
- btnCreateRoster : Button
- btnDesiresEmp : Button
- btnEditDelete : Button
- btnEditEmp : Button
- btnEditShift : Button
- btnEmp : Button
- btnExit : Button
- btnGroupUnwanted : Button
- btnManageRoster : Button
- btnNewEmp : Button
- btnReqRoster : Button
- btnReturnEmp : Button
- btnReturnRecords : Button
- btnReturnShift : Button
- btnShift : Button
- btnSkills : Button
- btnUnwantedPatterns : Button
- button1 : Button
- button2 : Button
- components : IContainer
- gbEmpMenu : GroupBox
- gbMainMenu : GroupBox
- gbManageRoster : GroupBox
- gbShiftMenu : GroupBox

Methods
- btnAddShift_Click(object sender, EventArgs e) : void
- btnCreateRoster_Click(object sender, EventArgs e) : void
- btnDesiresEmp_Click(object sender, EventArgs e) : void
- btnEditDelete_Click(object sender, EventArgs e) : void
- btnEditEmp_Click(object sender, EventArgs e) : void
- btnEditShift_Click(object sender, EventArgs e) : void
- btnEmp_Click(object sender, EventArgs e) : void
- btnExit_Click(object sender, EventArgs e) : void
- btnGroupUnwanted_Click(object sender, EventArgs e) : void
- btnManageRoster_Click(object sender, EventArgs e) : void
- btnNewEmp_Click(object sender, EventArgs e) : void
- btnReqRoster_Click(object sender, EventArgs e) : void
- btnReturnEmp_Click(object sender, EventArgs e) : void
- btnReturnRecords_Click(object sender, EventArgs e) : void
- btnReturnShift_Click_1(object sender, EventArgs e) : void
- btnShift_Click(object sender, EventArgs e) : void
- btnSkills_Click(object sender, EventArgs e) : void
- btnUnwantedPatterns_Click(object sender, EventArgs e) : void
- button1_Click(object sender, EventArgs e) : void
- button2_Click(object sender, EventArgs e) : void
- Dispose(bool disposing) : void
- frmMain()
- InitializeComponent() : void
- KleisimoFormas(object sender, FormClosingEventArgs e) : void

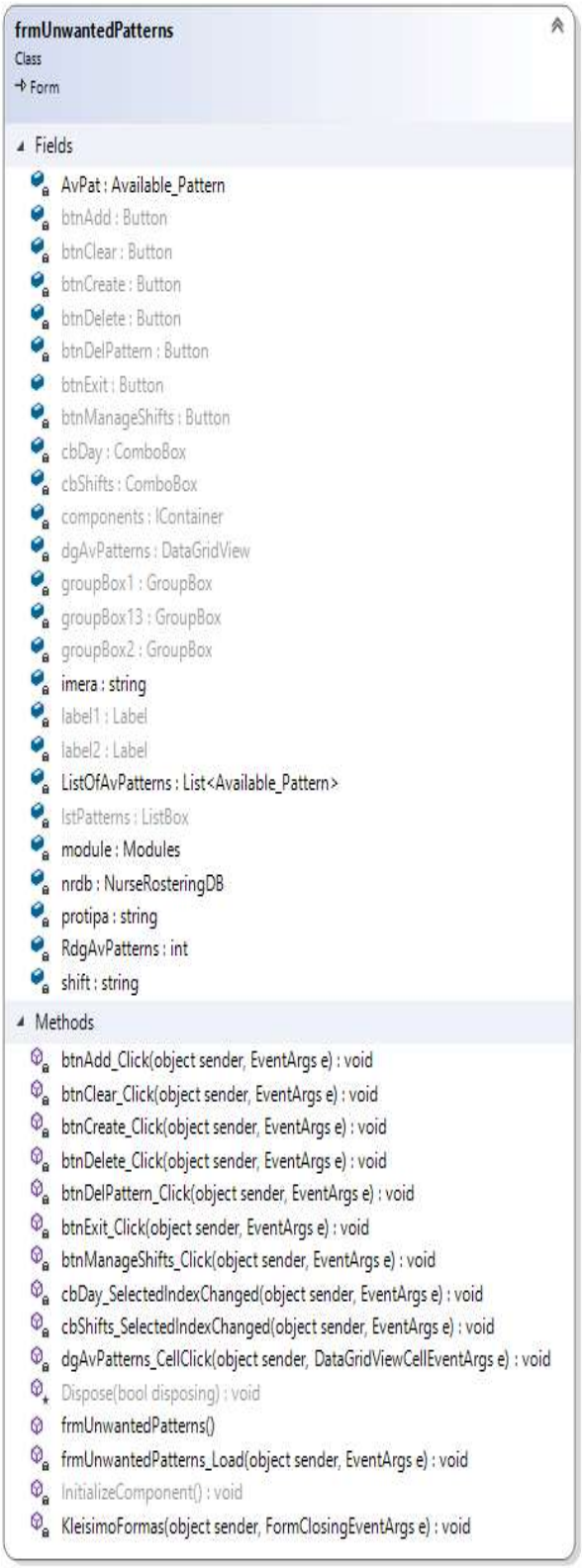
Εικόνα 58: Κλάση φόρμας frmMain



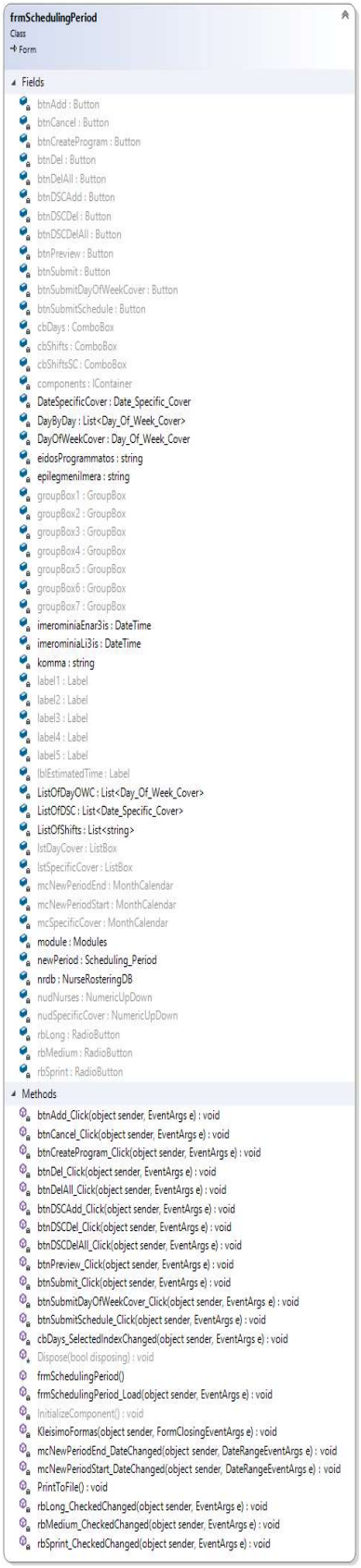
Εικόνα 59: Κλάση φόρμας frmNewShift



Εικόνα 60: Κλάση φόρμας frmProgram



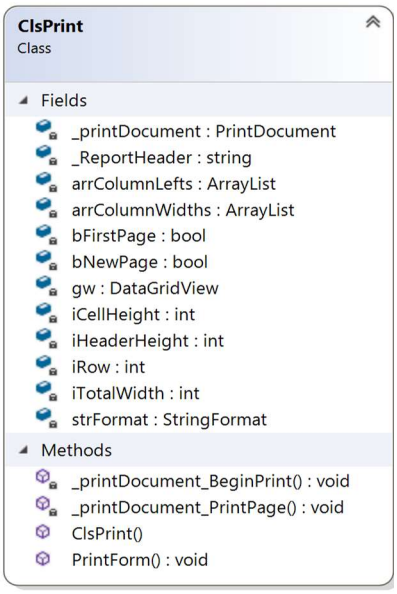
Εικόνα 62: Κλάση φόρμας frmUnwantedPatterns



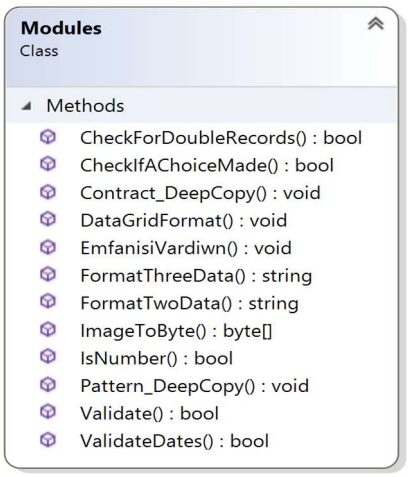
Εικόνα 61: Κλάση φόρμας frmScheduling_Period

5.4.5. Πακέτο Resourses

Τέλος, οι παρακάτω δύο κλάσεις οι οποίες περιλαμβάνονται στο πακέτο Resources είναι η κλάση Modules η οποία περιέχει τις περισσότερες μεθόδους που έχει δημιουργήσει ο προγραμματιστής για χρήση μέσα στην εφαρμογή και η κλάση ClsPrint η οποία είναι επιφορτισμένη με την εκτύπωση του δημιουργημένου προγράμματος εργασίας στον εκτυπωτή του συστήματος, αφού έχουν προηγηθεί οι σχετικές ρυθμίσεις από τη Γραμματεία μέσω της σχετικής διεπαφής που προβάλλεται όταν ζητηθεί η εκτύπωση. Η ύπαρξη της κλάσης Modules αποσκοπεί στο να είναι όλες οι δημιουργημένες μέθοδοι από τον προγραμματιστή, συγκεντρωμένες σε ένα αρχείο για ευκολότερη διαχείριση και πρόσβαση. Οι κλάσεις αυτές είναι:



Εικόνα 64: Κλάση ClsPrint



Εικόνα 63: Κλάση Modules

6. Βάση δεδομένων

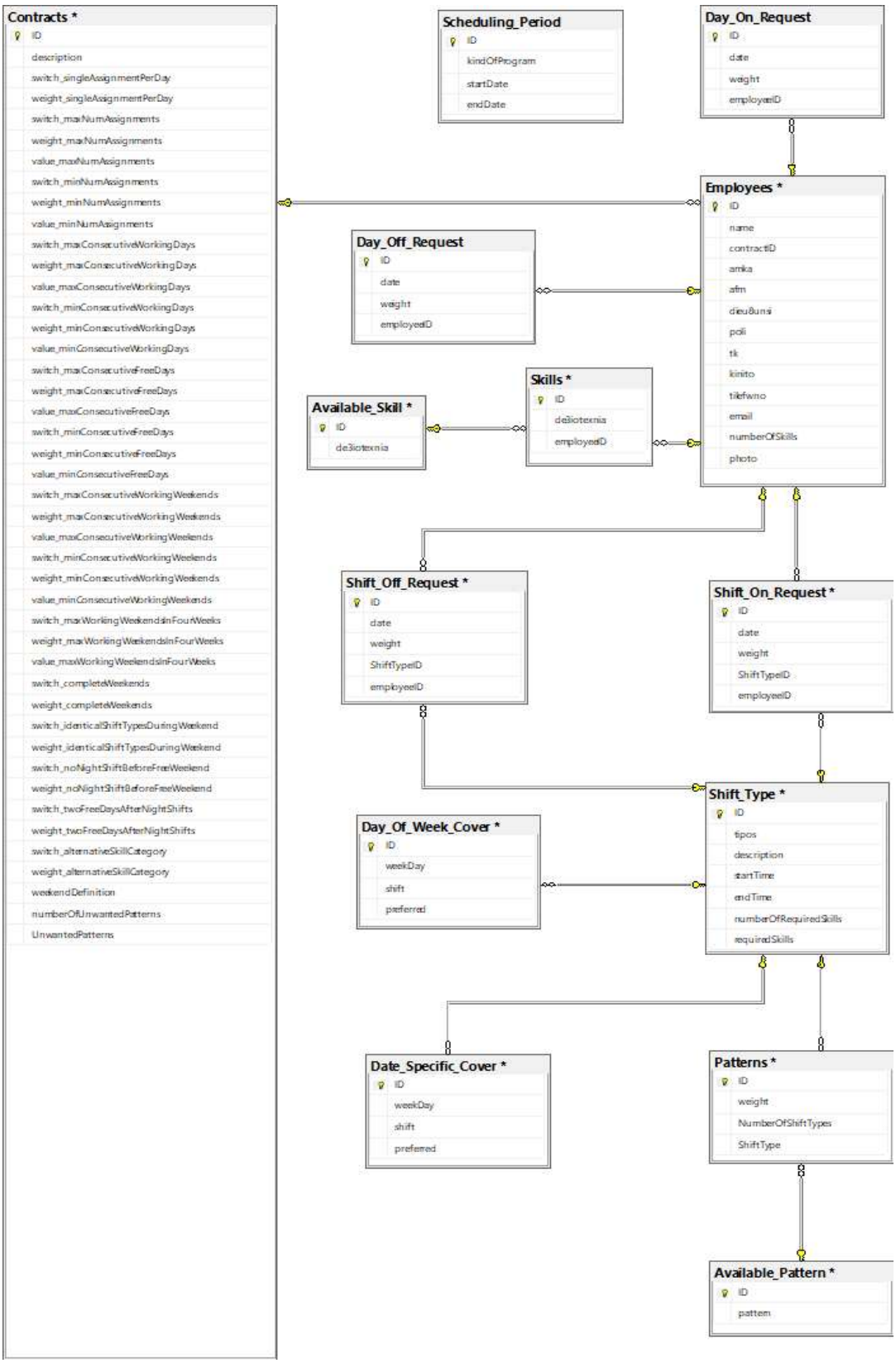
Για τη λειτουργία της εφαρμογής και για την αποφυγή της επαναλαμβανόμενης εισαγωγής των δεδομένων των συμβολαίων, των υπαλλήλων και άλλων περιπτώσεων από τη Γραμματεία, κρίθηκε απαραίτητο, τα δεδομένα αυτά να αποθηκεύονται σε μια βάση δεδομένων. Στη βάση αυτή κρατούνται τα εξής δεδομένα:

- **Συμβολαίων (Contracts)** – Σ' αυτόν τον πίνακα περιέχονται όλα τα δεδομένα των συμβολαίων, κοινώς οι όροι εργασίας των νοσοκόμων που δεσμεύονται από αυτό. Οι όροι εργασίας έχουν περιγραφεί νωρίτερα, στο κεφάλαιο 4.2. Κάθε γραμμή στον πίνακα των συμβολαίων στη βάση δεδομένων, αναπαριστά και ένα συμβόλαιο.
- **Νοσοκόμων (Employees)** – Στον πίνακα των νοσοκόμων περιέχονται όλα τα προσωπικά στοιχεία των νοσοκόμων, όπως ονοματεπώνυμο, τόπος διαμονής, στοιχεία επικοινωνίας και άλλα. Όπως και στον πίνακα των συμβολαίων, κάθε εγγραφή (γραμμή) στον πίνακα αυτόν αναπαριστά και ένα νοσοκόμο. Ο πίνακας των νοσοκόμων συνδέεται με τον πίνακα των συμβολαίων μέσω του αναγνωριστικού του συμβολαίου.
- **Αιτούμενων ημέρων εργασίας νοσοκόμων (Day_On_Request)** – Ο πίνακας αυτός περιέχει όλες τις ημέρες που έχει επιλέξει ο νοσοκόμος για να εργαστεί. Συνδέεται με τον πίνακα των νοσοκόμων μέσω του αναγνωριστικού του νοσοκόμου.
- **Αιτούμενων ημέρων ανάπαυσης νοσοκόμων (Day_Off_Request)** – Ο πίνακας αυτός περιέχει όλες τις ημέρες που έχει επιλέξει ο νοσοκόμος για να αναπαυθεί. Συνδέεται με τον πίνακα των νοσοκόμων μέσω του αναγνωριστικού του νοσοκόμου.
- **Αιτούμενων ημέρων εργασίας νοσοκόμων σε συγκεκριμένη ημέρα και βάρδια (Shift_On_Request)** – Ο πίνακας αυτός περιέχει όλες τις ημέρες που έχει επιλέξει ο νοσοκόμος για να εργαστεί σε συγκεκριμένη βάρδια της επιλεγμένης ημέρας. Συνδέεται με τον πίνακα των νοσοκόμων μέσω του αναγνωριστικού του νοσοκόμου και με τον πίνακα των βαρδιών μέσω του αναγνωριστικού της βάρδιας.
- **Αιτούμενων ημέρων μη εργασίας νοσοκόμων σε συγκεκριμένη ημέρα και βάρδια (Shift_Off_Request)** – Ο πίνακας αυτός περιέχει όλες τις ημέρες που έχει επιλέξει ο νοσοκόμος για να μην εργαστεί σε συγκεκριμένη βάρδια της επιλεγμένης ημέρας. Συνδέεται με τον πίνακα των νοσοκόμων μέσω του αναγνωριστικού του νοσοκόμου και με τον πίνακα των βαρδιών μέσω του αναγνωριστικού της βάρδιας.
- **Ειδικοτήτων (Skills)** – Στον πίνακα ειδικοτήτων εγγράφονται οι ειδικότητες που έχουν συνδεθεί με κάποιο νοσοκόμο. Στη συγκεκριμένη εφαρμογή έχει επιλεγεί να εισάγει η Γραμματεία τις ειδικότητες και όχι να επιλέγονται από έναν έτοιμο πίνακα.

Η Γραμματεία εισάγοντας τις ειδικότητες, δημιουργεί ένα δικό της πίνακα με διαθέσιμες ειδικότητες **(Available_Skill)** και από αυτόν αντλεί όποιες χρειάζεται και τις συνδέει με το νοσοκόμο στον πίνακα ειδικοτήτων **(Skills)**. Συνδέεται και με τον πίνακα των νοσοκόμων μέσω του αναγνωριστικού του νοσοκόμου.

- **Βαρδιών (Shift_Type)** – Ο πίνακας βαρδιών, περιέχει τα χαρακτηριστικά στοιχεία της κάθε βάρδιας όπως, ώρα έναρξης και λήξης, περιγραφή βάρδιας, τύπος κλπ. Κάθε εγγραφή στον πίνακα αυτόν αναπαριστά και μια βάρδια. Ο τύπος της βάρδιας ορίζεται από τον αλγόριθμο δημιουργίας του προγράμματος και δεν είναι δυνατόν να αλλάξει μέσα από την εφαρμογή. Έτσι, η Γραμματεία επιλέγει τον προκαθορισμένο τύπο και προσθέτει όλα τα υπόλοιπα χαρακτηριστικά που συνθέτουν τη βάρδια.
- **Κάλυψη βαρδιών σε προσωπικό για κάθε ημέρα της εβδομάδας (Day_Of_Week_Cover)** – Στον πίνακα αυτόν, αποθηκεύονται όλα τα δεδομένα τα οποία χρειάζονται για την κάλυψη όλων των βαρδιών σε πλήθος προσωπικού για κάθε ημέρα της εβδομάδας. Αυτός ο πίνακας συνδέεται με τον πίνακα των βαρδιών μέσω του αναγνωριστικού των βαρδιών.
- **Κάλυψη σε προσωπικό σε συγκεκριμένη ημέρα και βάρδια (Date_Specific_Cover)** – Ο πίνακας αυτός περιέχει δεδομένα τα οποία χρησιμοποιούνται για την κάλυψη από συγκεκριμένο πλήθος προσωπικού σε συγκεκριμένη βάρδια και ημέρα πχ λόγω κάποιας έκτακτης ανάγκης.
- **Ανεπιθύμητα πρότυπα εργασίας (Patterns)** – Σε αυτόν τον πίνακα ισχύει ότι ισχύει για τον πίνακα των ειδικοτήτων. Είναι, δηλαδή, συνδετικός πίνακας μεταξύ του πίνακα των βαρδιών και του πίνακα των στοιχειωδών ανεπιθύμητων προτύπων. Όταν αναφερόμαστε στα *στοιχειώδη ανεπιθύμητα πρότυπα εργασίας*, εννοούμε τα στοιχεία τα οποία από μόνα τους είναι ένα πρότυπο ανεπιθύμητης εργασίας, αλλά συνδυαζόμενα με 2 ή περισσότερα του είδους, προκύπτουν άλλα πρότυπα ανεπιθύμητης εργασίας. Η Γραμματεία εισάγοντας τα στοιχειώδη ανεπιθύμητα πρότυπα εργασίας, δημιουργεί ένα δικό της πίνακα με διαθέσιμα πρότυπα **(Available_Pattern)** και από αυτόν αντλεί όποια χρειάζεται και τα συνδέει με τη βάρδια στον πίνακα ειδικοτήτων **(Patterns)**. Συνδέεται και με τον πίνακα των βαρδιών μέσω του τύπου της βάρδιας.
- **Περίοδοι προγραμματισμού (Scheduling_Period)** – Ο πίνακας αυτός περιέχει τα χρονολογικά δεδομένα του προγράμματος εργασίας και τον τύπο με τον οποίο δημιουργήθηκε το πρόγραμμα αυτό (sprint, medium, long). Αυτός ο πίνακας είναι ανεξάρτητος και δε συνδέεται με κανέναν άλλο.

Παρακάτω, φαίνεται το διάγραμμα οντοτήτων – συσχετίσεων στο οποίο συνοψίζονται όλα τα προαναφερθέντα.



Εικόνα 65: Διάγραμμα οντοτήτων - συσχετίσεων της βάσης δεδομένων του NurseRostering

Η δημιουργία της βάσης δεδομένων έχει γίνει με χρήση του Entity Framework 6.2 της Microsoft. Έγινε χρήση της τεχνικής σχολίων δεδομένων (Data Annotation) στις κλάσεις του πακέτου InputData οι οποίες αποτελούν και τους πίνακες της βάσης δεδομένων. Οι συσχετίσεις των πινάκων της βάσης δεδομένων γίνονται είτε με χρήση ξένων κλειδιών είτε με χρήση προγραμματισμού. Η χρήση του προγραμματισμού έχει επιλεχθεί με γνώμονα την σύνδεση διαφορετικού τύπου δεδομένων (πχ μια τιμή integer να αποθηκευτεί σε μια θέση τύπου varchar). Επίσης, δεν έχει γίνει σύνδεση των πλαισίων κειμένου και άλλων αντικειμένων εισαγωγής δεδομένων από τις φόρμες προς τη βάση δεδομένων, προς αποφυγή άμεσης έκθεσης των δεδομένων στο χρήστη με κίνδυνο ακούσια αλλαγή ή εισαγωγή μη έγκυρων δεδομένων. Η επικοινωνία των δεδομένων των φορμών με τη βάση δεδομένων γίνεται μέσω κώδικα για να ελέγχεται όσο το δυνατόν καλύτερα η ορθότητα των εισαγόμενων δεδομένων στη βάση. Διαφορετικά, θα προκύψει πρόβλημα στη λειτουργία του αλγορίθμου.

7. Περιγραφή – επίδειξη λειτουργίας της εφαρμογής

Η εφαρμογή, αφού λάβει τα όλα τα δεδομένα από τη Γραμματεία (στοιχεία υπαλλήλων, απαιτήσεις αυτών και της διοίκησης σχετικά με τη δύναμη σε προσωπικό για κάθε ημέρα και κάθε βάρδια, χρονική περίοδος προγραμματισμού και άλλες ιδιαιτερότητες) και τα αποθηκεύσει στη βάση δεδομένων, για μελλοντική χρήση, δημιουργεί το αρχείο εισόδου του αλγορίθμου. Τότε, καλείται ο αλγόριθμος, ως εξωτερική εφαρμογή, σε νέο νήμα, για να μπορεί να εποπτεύεται από την εφαρμογή σχετικά με το πότε θα ολοκληρώσει την εργασία της. Ο αλγόριθμος δημιουργίας του προγράμματος εργασίας, τροποποιήθηκε ελαφρώς, ως προς τα δεδομένα εισόδου, τα οποία ζητάει από τον χρήστη. Οι τροποποιήσεις που έγιναν είναι οι παρακάτω:

- Αφαιρέθηκε το μενού επιλογής αρχείου και τροποποιήθηκε έτσι ώστε να δέχεται αρχείο με φέρον όνομα, την κατάσταση λειτουργίας του προγράμματος που επιλέγεται από τον χρήστη (sprint.txt, medium.txt και long.txt). Αναφορικά, αν ο χρήστης επιλέξει την δημιουργία προγράμματος εργασίας με επιλεγμένη την κατάσταση sprint, τότε ο αλγόριθμος θα ψάξει να βρει το αρχείο sprint.txt, αν επιλεγεί η κατάσταση medium θα αναζητηθεί το αρχείο medium.txt ενώ στην κατάσταση long, ο αλγόριθμος θα ψάξει να βρει το αρχείο long.txt. Αντίστοιχα, δημιουργήθηκαν τρία διαφορετικά εκτελέσιμα στιγμιότυπα του αλγορίθμου, ένα για κάθε κατάσταση λειτουργίας.
- Αφαιρέθηκε η δυνατότητα εισόδου τιμής για τη μεταβλητή seed από τον χρήστη και επιλέχθηκε η τιμή ενεργοποίησης της αυτόματης επιλογής από το σύστημα (-1).
- Αφαιρέθηκε η παύση η οποία κρατούσε ενεργή την κονσόλα μετά την ολοκλήρωση της δημιουργίας του προγράμματος εργασίας, προτρέποντας τον χρήστη να πιέσει ένα κουμπί για τον τερματισμό αυτής. Έτσι, τερματίζει αυτόματα τη λειτουργία του.

Στη συνέχεια, αφού δημιουργηθεί το αρχείο του προγράμματος και τερματιστεί ο αλγόριθμος, τότε η Γραμματεία μπορεί πατώντας το κατάλληλο κουμπί στη φόρμα, να δει το δημιουργημένο πρόγραμμα σε μια άλλη φόρμα. Αφού φορτωθεί το πρόγραμμα στη φόρμα, αυτή εμφανίζεται στη Γραμματεία, παρουσιάζοντας τα ονόματα των νοσοκόμων κατά σειρά του πίνακα προγράμματος, τις ημερομηνίες για τις οποίες σχεδιάστηκε το πρόγραμμα εργασίας, κατά στήλη του πίνακα προγράμματος εργασίας και στην τομή των στηλών και των γραμμών, δηλαδή της ημερομηνίας και του ονόματος του νοσοκόμου, η Γραμματεία πληροφορείται για τη βάρδια στην οποία εργάζεται ο συγκεκριμένος νοσοκόμος κατά τη συγκεκριμένη ημερομηνία. Επίσης, η φόρμα αυτή, δίνει στον χρήστη επιπλέον των πληροφοριών, τρεις επιλογές επιπλέον. Αυτές είναι:

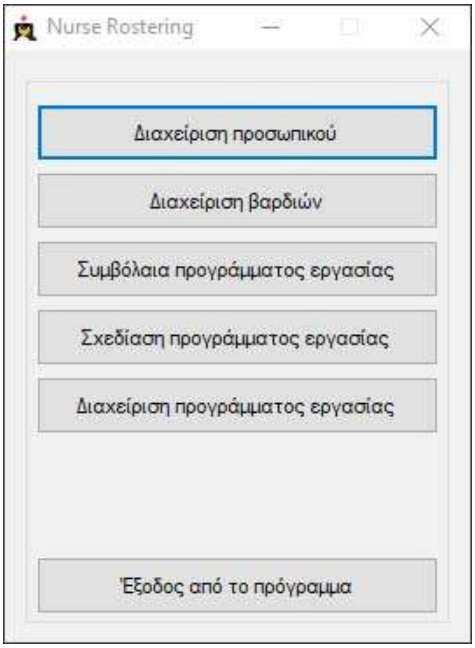
- Παρέμβαση στο πρόγραμμα που δημιουργήθηκε από τον αλγόριθμο, για αντιμετώπιση αμοιβαίων αλλαγών της τελευταίας στιγμής ή μετά από αυτήν. Η παρέμβαση αυτή αποτυπώνεται μόνο στο γραφικό περιβάλλον και όχι στη λύση που έδωσε ο αλγόριθμος. Μπορεί να αποθηκευτεί στο αρχείο excel αλλά και να εκτυπωθεί.
- Δυνατότητα εκτύπωσης του προγράμματος εργασίας, όπως δημιουργήθηκε ή όπως τροποποιήθηκε από τον χρήστη.
- Δυνατότητα αποθήκευσης του προγράμματος σε αρχείο μορφής Excel. Γι' αυτή τη δυνατότητα, πρέπει να είναι εγκατεστημένη η εφαρμογή Microsoft Excel.

Στη συνέχεια, παρουσιάζεται μια περιγραφή – επίδειξη της λειτουργίας της εφαρμογής όπως σχεδιάστηκε και λειτουργεί ως αυτό το σημείο. Αφού ο χρήστης «τρέξει» την εφαρμογή, αυτή εκκινεί εμφανίζοντας την παρακάτω εισαγωγική – ενημερωτική φόρμα.

Εικόνα 66: Εισαγωγική - ενημερωτική φόρμα

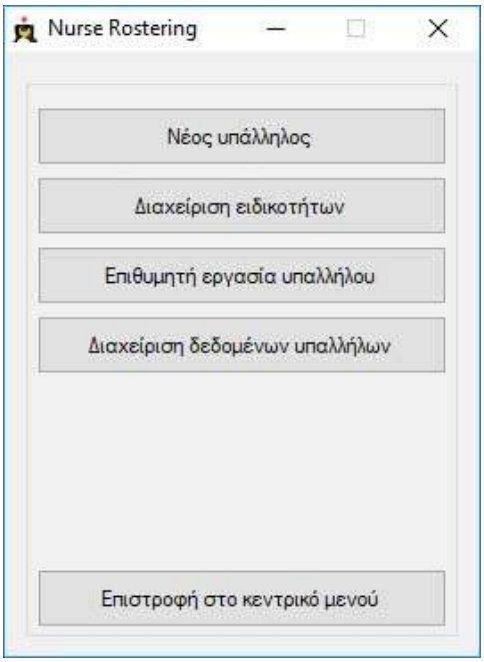
Κατά τη διάρκεια εμφάνισης της ανωτέρω φόρμας, εκτελείται η δημιουργία της βάσης δεδομένων αφού έχει προηγηθεί έλεγχος για την ύπαρξη αυτής. Έπειτα, ορίζεται ένα χρονικό διάστημα τριών δευτερολέπτων, παραμονής της φόρμας. Αφού περάσει το ορισμένο χρονικό διάστημα αυτό, η φόρμα κρύβεται και δεν τερματίζεται η λειτουργία της, γιατί σε αυτήν την περίπτωση θα τερματιστεί και η λειτουργία της εφαρμογής. Στη συνέχεια, εμφανίζεται η παρακάτω φόρμα του κύριου μενού επιλογών του χρήστη η οποία δίνει στη Γραμματεία ένα

πλήθος επιλογών σχετικά με τη διαχείριση του προσωπικού, των βαρδιών αλλά και του προγράμματος εργασίας.



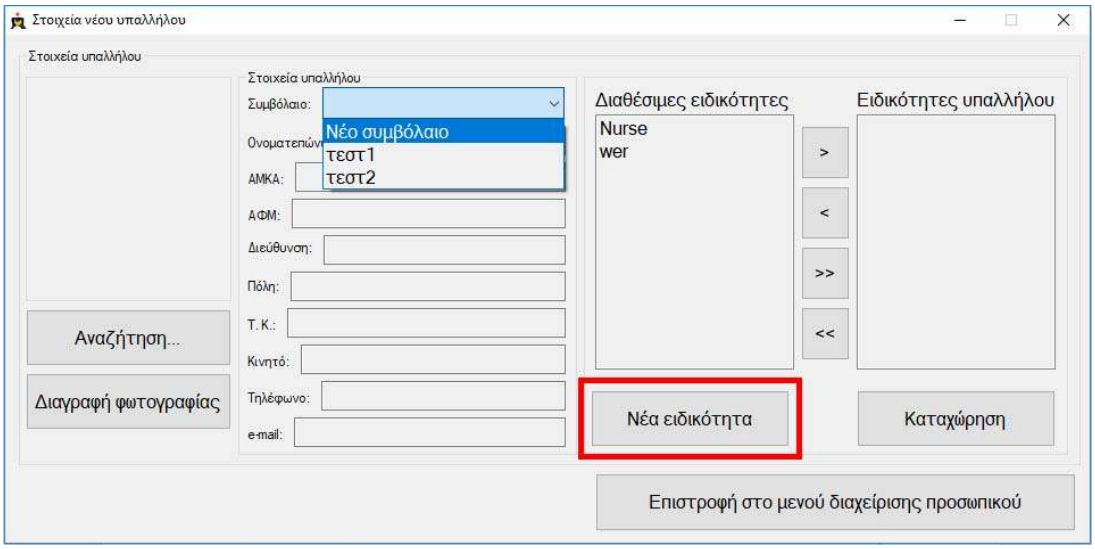
Εικόνα 67: Κύριο μενού επιλογών

Όταν πατηθεί το κουμπί «Διαχείριση προσωπικού» του κύριου μενού, εμφανίζεται το παρακάτω μενού με το οποίο είναι εφικτή η προσθήκη υπαλλήλων στο δυναμολόγιο του ιδρύματος, η διαχείριση των ειδικοτήτων του προσωπικού και η διαχείριση των αιτούμενων ημερών εργασίας και ανάπαυσης του προσωπικού ή την επιθυμία για εργασία σε συγκεκριμένη βάρδια κάποιας συγκεκριμένης ημέρας. Τέλος, η Γραμματεία μπορεί να επεξεργαστεί άμεσα όλα τα δεδομένα που αφορούν σε ένα νοσοκόμο.



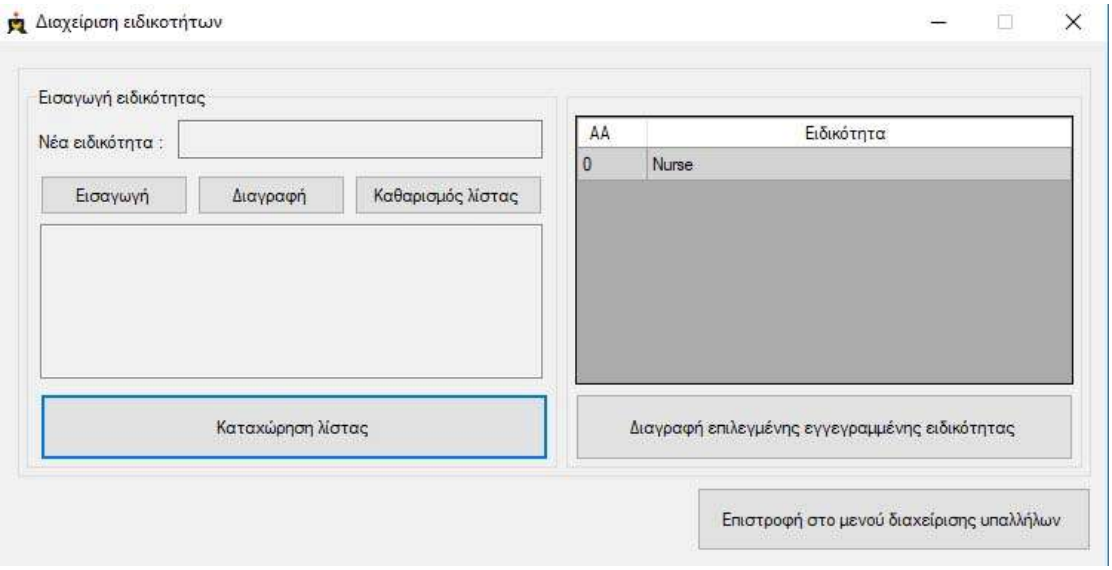
Εικόνα 68: Μενού διαχείρισης προσωπικού

Όταν η Γραμματεία πατήσει το κουμπί «Νέος υπάλληλος», τότε εμφανίζεται η παρακάτω φόρμα εισαγωγής προσωπικών δεδομένων του νοσοκόμου. Όλα τα πεδία είναι υποχρεωτικά πλην αυτό της φωτογραφίας του υπαλλήλου. Μέσα από αυτή τη φόρμα η Γραμματεία, μπορεί να μεταβεί σε άλλες δύο φόρμες, αυτή της καταχώρησης νέου συμβολαίου και της καταχώρησης νέας ειδικότητας, όπως φαίνεται στην παρακάτω εικόνα στα σχετικά σημεία. Ένα παράδειγμα της ύπαρξης αυτών των μεταβάσεων είναι ότι κατά τη διάρκεια καταχώρησης των στοιχείων του νέου υπαλλήλου μπορεί να γίνει αντιληπτή η έλλειψη των σχετικών δεδομένων. Ενδεχόμενο κλείσιμο της φόρμας και μετάβαση μέσα από το αρχικό μενού, σημαίνει ότι όλα τα δεδομένα που έχουν εισαχθεί μέχρι τη δεδομένη στιγμή, χάνονται, μαζί και χρόνος εργασίας. Με αυτόν τον τρόπο, όμως, γίνεται η καταχώρηση των δεδομένων στη βάση και είναι άμεσα για χρήση στη φόρμα καταχώρησης νέου υπαλλήλου από το σημείο που σταμάτησε η προηγούμενη εργασία της Γραμματείας. Η περιγραφή των φορμών συμβολαίων και βαρδιών θα γίνει παρακάτω.



Εικόνα 69: Κλήση της φόρμας συμβολαίων ή της φόρμας καταχώρησης νέας ειδικότητας από τη φόρμα καταχώρησης νέου υπαλλήλου

Η φόρμα που φαίνεται παρακάτω, εμφανίζεται όταν η Γραμματεία πατήσει το κουμπί «Διαχείριση ειδικοτήτων». Μέσα από αυτή τη φόρμα, η Γραμματεία έχει τη δυνατότητα να εισάγει στη βάση δεδομένων τις ειδικότητες που απονέμονται στους υπαλλήλους μία-μία ή πολλές ταυτόχρονα είτε να διαγράψει κάποια από τη βάση δεδομένων.

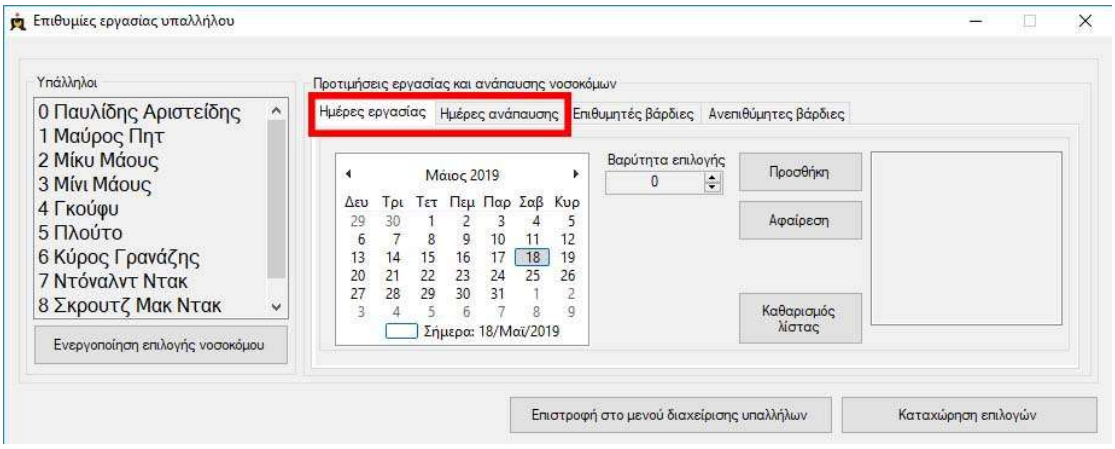


Εικόνα 70: Φόρμα διαχείρισης ειδικοτήτων

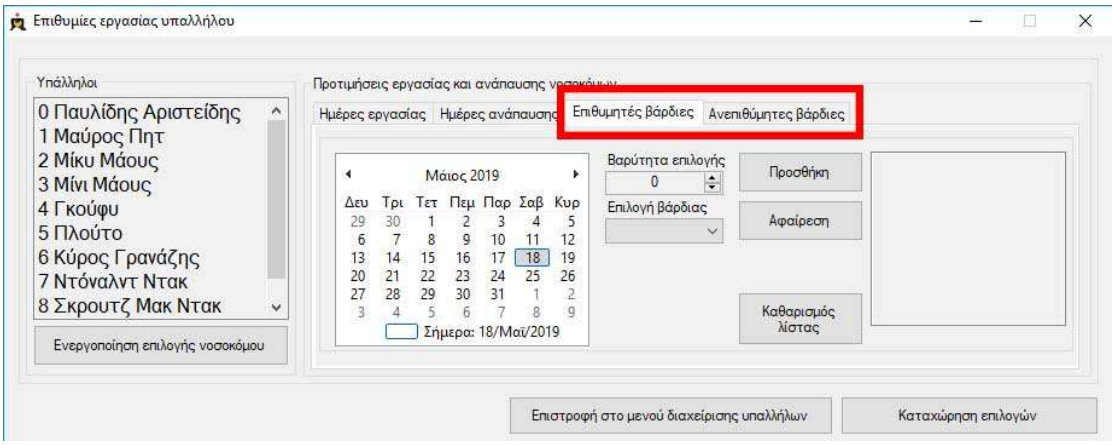
Στο τρίτο κουμπί της λειτουργίας διαχείρισης προσωπικού «Επιθυμητή εργασία υπαλλήλου» βρίσκουμε την παρακάτω φόρμα η οποία περιέχει τέσσερις καρτέλες. Η πρώτη και η δεύτερη καρτέλα αναφέρονται στις επιθυμητές ημέρες μόνο, για εργασία και ανάπαυση αντίστοιχα

ενώ η τρίτη και η τέταρτη αναφέρονται στις επιθυμητές βάρδιες σε συγκεκριμένη ημερομηνία για εργασία ή ανάπαυση αντίστοιχα.

Η Γραμματεία επιλέγει έναν υπάλληλο και τότε η λίστα των υπαλλήλων «κλειδώνει» μέχρι να ολοκληρωθεί η διαδικασία ή μέχρι η Γραμματεία να πατήσει το κουμπί «Ενεργοποίηση επιλογής νοσοκόμου» για να αλλάξει την επιλογή της. Αν έχει πραγματοποιηθεί έστω και μια επιλογή για κάποιο νοσοκόμο, η αλλαγή του νοσοκόμου από τη λίστα δεν είναι δυνατή μέχρι να ολοκληρωθεί η διαδικασία ή να απαλειφθούν όλες οι επιλογές που έγιναν.



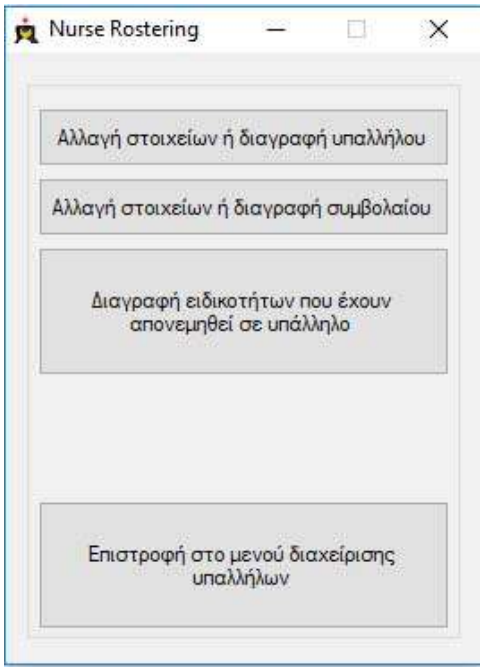
Εικόνα 71: Επιθυμίες υπαλλήλου για ημέρες εργασίας και ανάπαυσης



Εικόνα 72: Επιθυμίες υπαλλήλου για ημέρες εργασίας και ανάπαυσης σε συγκεκριμένη βάρδια

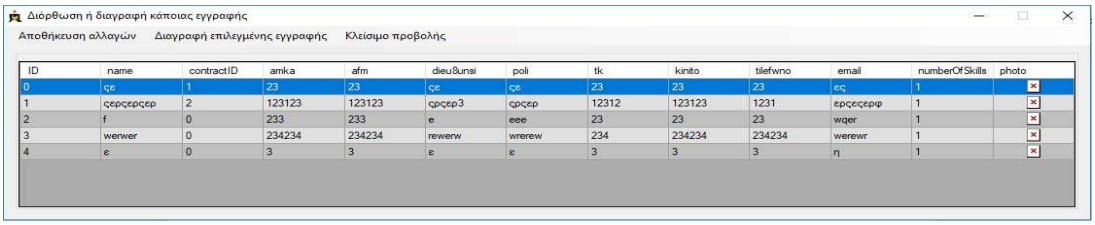
Το τέταρτο κουμπί «Διαχείριση δεδομένων υπαλλήλων» μας μεταφέρει σε ένα άλλο μενού επιλογών σχετικό με τη διαχείριση δεδομένων των υπαλλήλων. Μέσα από αυτές τις επιλογές η Γραμματεία μπορεί να πραγματοποιήσει αλλαγή η διαγραφή των δεδομένων που επιθυμεί

σε ότι αφορά στα συμβόλαια, στους υπαλλήλους και στις ειδικότητες που έχουν απονεμηθεί στους νοσοκόμους, μέσα από τον σχετικό πίνακα που φαίνεται στην Εικόνα 74



Εικόνα 73: Μενού επιλογών διαχείρισης βαρδιών

Όταν ο χρήστης πατήσει κάποιο από τα κουμπιά της παραπάνω φόρμας, τότε εμφανίζεται ο παρακάτω πίνακας εγγραφών (ενδεικτικά εδώ, φαίνεται ο πίνακας με τους νοσοκόμους) προσφέροντας στο χρήστη τη δυνατότητα να τροποποιήσει ή να διαγράψει εγγραφές. Η τροποποίηση δεν επιτρέπεται για όλες τις στήλες μιας εγγραφής. Έχει επιλεγεί αυτός ο τρόπος εμφάνισης και πρόσβασης στα δεδομένα (των υπαλλήλων, στο παράδειγμά μας) διότι προσφέρει γρήγορη πρόσβαση σε πολλές εγγραφές.

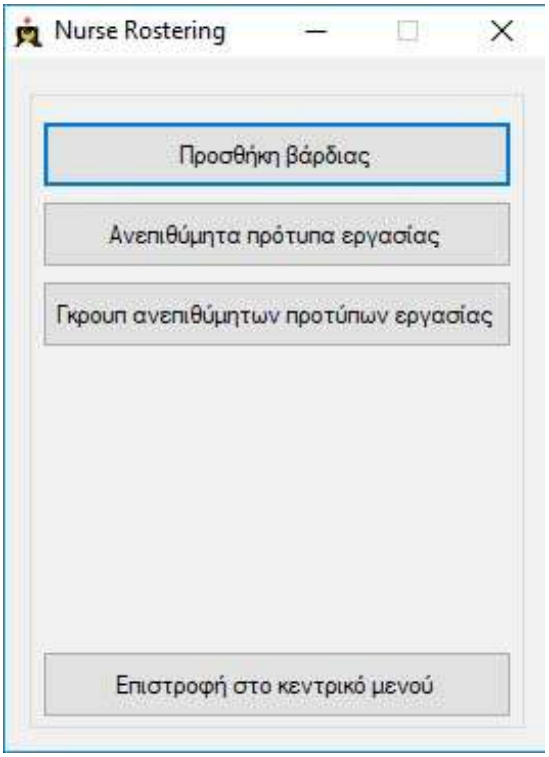


ID	name	contractID	amka	afm	dieu8unai	poli	tk	kinito	tilefwno	email	numberOfSkills	photo
0	ςε	1	23	23	ςε	ςε	23	23	23	ες	1	
1	ςερςερςερ	2	123123	123123	ςρςερ3	ςρςερ	12312	123123	1231	ερςεςερφ	1	
2	f	0	233	233	e	eee	23	23	23	wqer	1	
3	werwer	0	234234	234234	rewerw	wrerew	234	234234	234234	werewr	1	
4	ε	0	3	3	ε	ε	3	3	3	η	1	

Εικόνα 74: Πίνακας διαχείρισης δεδομένων

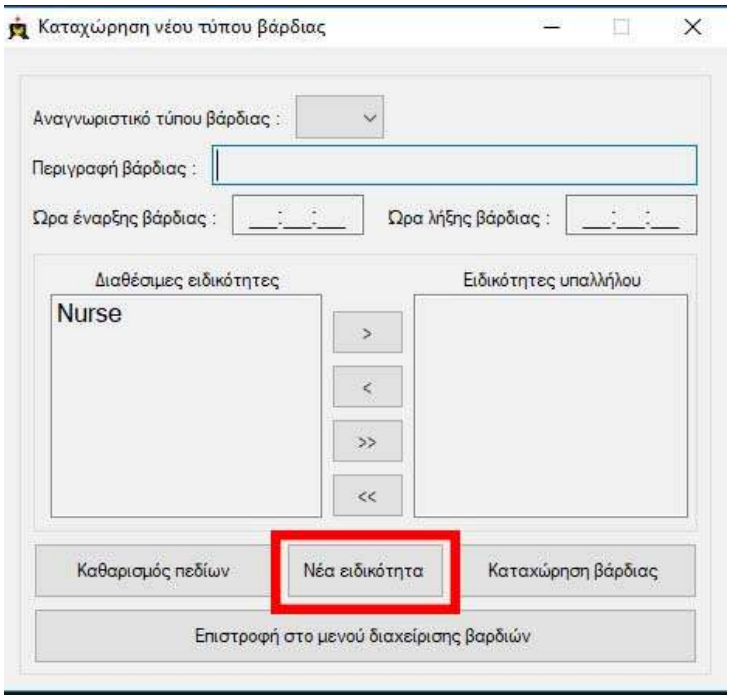
Σε αυτό το σημείο τελειώνουν οι επιλογές του χρήστη σχετικά με τη διαχείριση των υπαλλήλων. Στη συνέχεια, το μενού των επιλογών σχετικών με τις βάρδιες το οποίο φαίνεται παρακάτω, επιτρέπει τη δημιουργία τύπου βάρδιας, τη δημιουργία στοιχειωδών προτύπων

ανεπιθύμητης εργασίας και τη συγκρότηση μεγαλύτερων προτύπων ανεπιθύμητης εργασίας με την ομαδοποίηση των στοιχειωδών προτύπων, τα οποία εισάγονται στα συμβόλαια των υπαλλήλων.



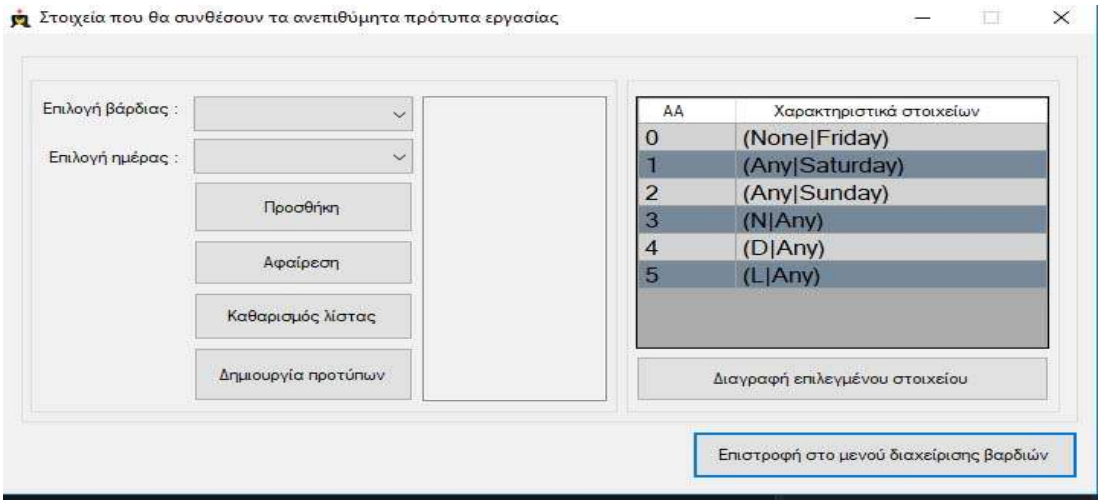
Εικόνα 75: Μενού διαχείρισης βαρδιών

Το κουμπί «Προσθήκη βάρδιας» φανερώνει την παρακάτω φόρμα η οποία έχει κι αυτή ένα στοιχείο πλοήγησης μέσα στην εφαρμογή, σχετικά με τη δημιουργία νέας ειδικότητας. Πέρα από αυτό, η εφαρμογή αυτή συμβάλει στην καταχώρηση δεδομένων σχετικών με τις βάρδιες. Έχει προκαθορισμένα και μη επεξεργάσιμα τα αναγνωριστικά τύπων βαρδιών, λόγω απαιτήσεων από τον αλγόριθμο δημιουργίας του προγράμματος εργασίας. Η Γραμματεία μπορεί να ορίσει την έναρξη και λήξη της κάθε βάρδιας καθώς και την περιγραφή της και τις ειδικότητες που απαιτείται να έχουν όσοι εργάζονται σε αυτήν.



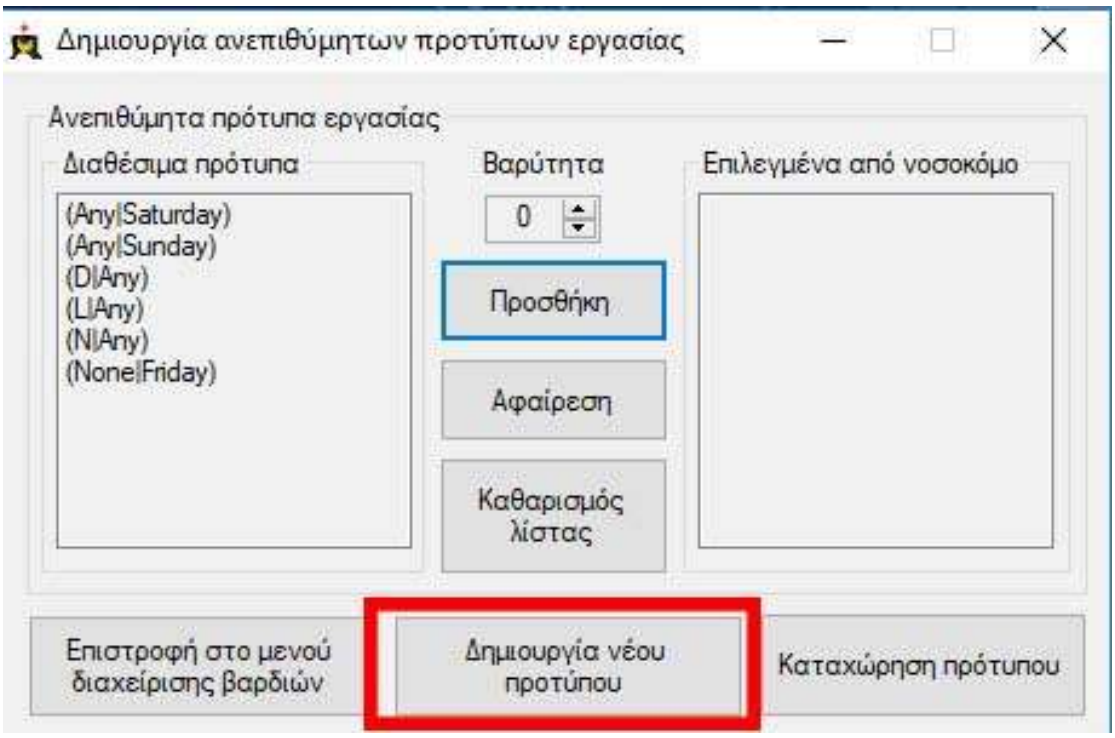
Εικόνα 76: Καταχώρηση νέας βάρδιας

Τα στοιχειώδη ανεπιθύμητα πρότυπα εργασίας, όπως αναφέρθηκε σε προηγούμενο κεφάλαιο, είναι το ελάχιστο στοιχείο το οποίο μπορεί από μόνο του να αποτελέσει ένα πρότυπο, αλλά και σε συνδυασμό με άλλα στοιχειώδη πρότυπα, μπορούν να αποτελέσουν ένα νέο πρότυπο μεγαλύτερης πολυπλοκότητας. Για τη σύνθεση ενός προτύπου, απαιτούνται οι βάρδιες, είτε με καθολική παρουσία (any) ή απουσία (none), είτε μεμονωμένες αλλά και οι ημέρες της εβδομάδας όλες (για όλες τις ημέρες της εβδομάδας – any) ή κάθε μια ξεχωριστά. Επίσης, μέσα από αυτή τη φόρμα, μπορεί να γίνει η διαγραφή από τη βάση δεδομένων, ενός προτύπου που δεν είναι πλέον επιθυμητό ή χρήσιμο.



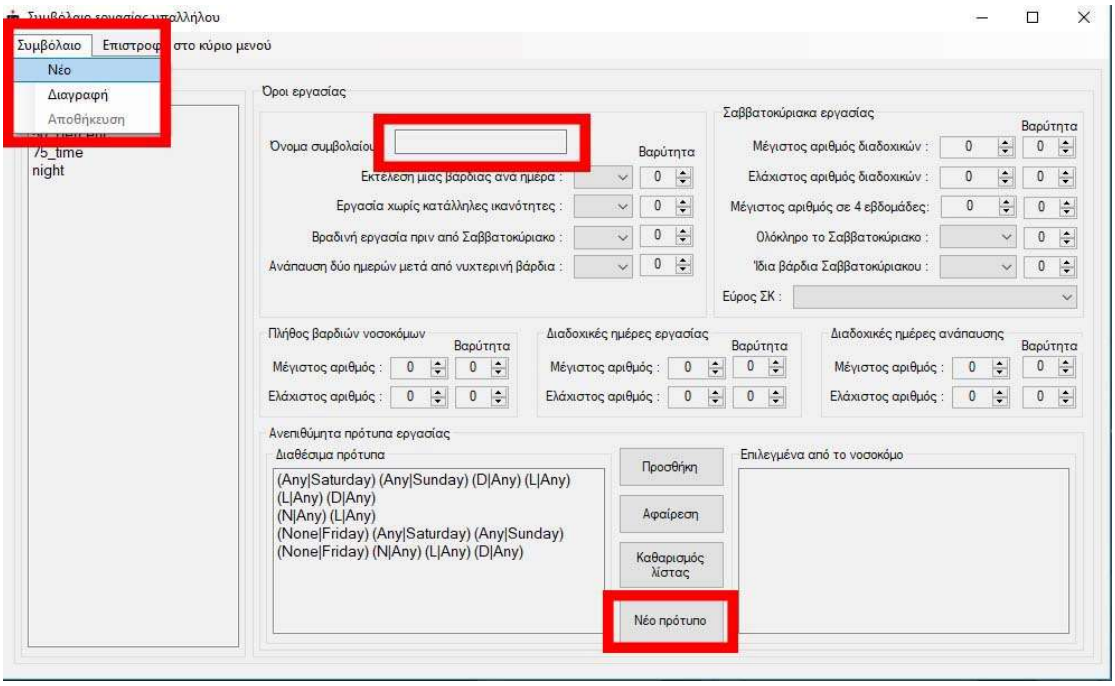
Εικόνα 77: Ανεπιθύμητα στοιχειώδη πρότυπα εργασίας

Στην παρακάτω εικόνα φαίνεται η φόρμα μέσα από την οποία συνθέτουμε τα γκρουπ των ανεπιθύμητων προτύπων εργασίας. Η φόρμα αυτή, προσφέρει δυνατότητα πλοήγησης καλώντας τη φόρμα δημιουργίας νέου προτύπου, όπως περιεγράφηκε παραπάνω.



Εικόνα 78: Σύνθεση ανεπιθύμητων προτύπων εργασίας

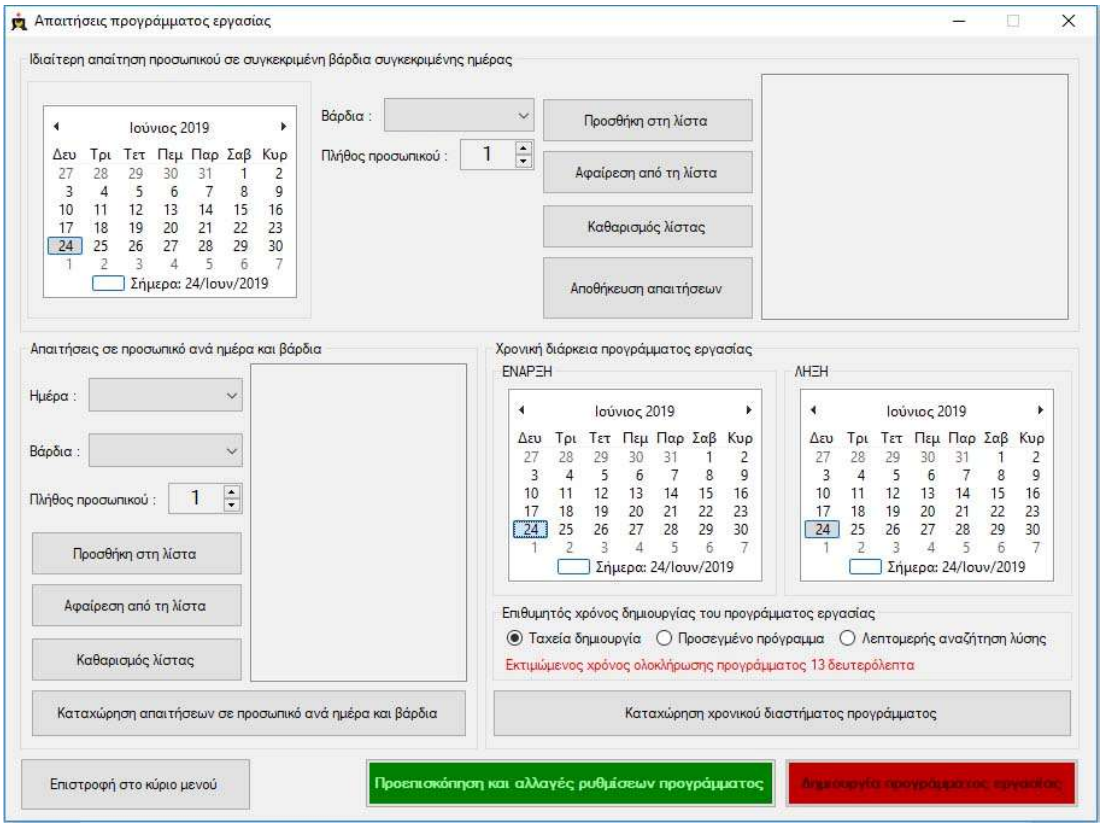
Αφού τελειώσει η Γραμματεία και με το μενού διαχείρισης βαρδιών, μπορεί να επιλέξει το μενού διαχείρισης «Συμβόλαια προγράμματος εργασίας» για να εμφανίσει την παρακάτω φόρμα διαχείρισης των συμβολαίων εργασίας με τα οποία συνδέονται οι νοσοκόμοι και περιγράφουν τους όρους εργασίας. Και αυτή η φόρμα, προσφέρει δυνατότητα πλοήγησης μέσα στην εφαρμογή NurseRostering.



Εικόνα 79: Διαχείριση συμβολαίων εργασίας

Αφού η Γραμματεία εμφανίσει τη φόρμα, έχει τη δυνατότητα να καταχωρήσει ένα νέο συμβόλαιο ενεργοποιώντας την κατάλληλη επιλογή από το μενού επιλογών για να «ξεκλειδώσει» το αντίστοιχο πλαίσιο κειμένου και να μπορέσει να εισάγει το όνομα του συμβολαίου, όπως φαίνεται στην παραπάνω εικόνα, αφού έχει πραγματοποιήσει τις όποιες επιλογές της από τις διαθέσιμες, αλλά και να ενεργοποιήσει το κουμπί «Αποθήκευση». Στη συνέχεια, η Γραμματεία πατάει το κουμπί «Αποθήκευση» και το συμβόλαιο όπως δημιουργήθηκε, αποθηκεύεται στη βάση δεδομένων στον αντίστοιχο πίνακα. Η Γραμματεία μπορεί επίσης, να διαγράψει όποιο συμβόλαιο επιθυμεί από τη βάση δεδομένων μέσω αυτής της φόρμας και πατώντας του κουμπί «Διαγραφή» αφού έχει επιλέξει πρώτα το συμβόλαιο που την ενδιαφέρει.

Το επόμενο κουμπί στο κύριο μενού της εφαρμογής, «Σχεδίαση προγράμματος εργασίας» αφήνει εύκολα να γίνει αντιληπτό την εργασία που επιτελεί. Όταν πατηθεί, εμφανίζει την παρακάτω φόρμα στην οποία γίνεται εισαγωγή όλων των απαραίτητων δεδομένων και αφού η Γραμματεία ορίσει τη χρονική διάρκεια του προγράμματος εργασίας, τότε και μόνο τότε, ενεργοποιείται το κουμπί «Δημιουργία προγράμματος εργασίας».



Εικόνα 80: Σχεδίαση και δημιουργία προγράμματος εργασίας

Πριν από αυτό όμως, η Γραμματεία πρέπει να κάνει μια σειρά από ενέργειες και μετά να πατήσει το κουμπί της σχεδίασης του προγράμματος εργασίας, οι οποίες ενέργειες είναι να εισάγει:

- Το χρονικό διάστημα σχεδίασης του προγράμματος
- Την κατάσταση λειτουργίας, όσο αφορά τον χρόνο παραγωγής αυτού
- Τις απαιτήσεις σε προσωπικό κάθε ημέρα για κάθε βάρδια
- Τις ιδιαίτερες απαιτήσεις σε προσωπικό για κάποια συγκεκριμένη ημέρα σε κάποια συγκεκριμένη βάρδια.

Στη συνέχεια, έχει τις παρακάτω επιλογές

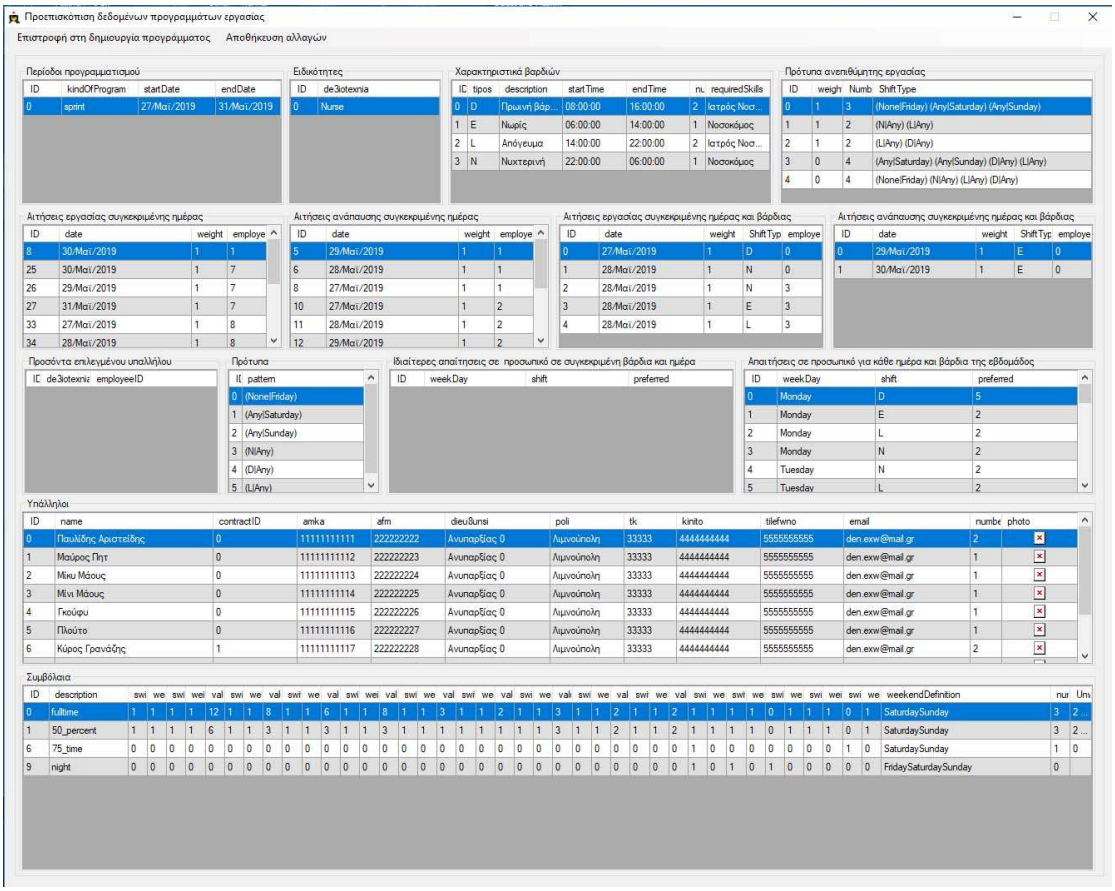
- Δημιουργία προγράμματος εργασίας
- Προεπισκόπηση όλων των δεδομένων που βρίσκονται στη βάση δεδομένων

Πατώντας το κουμπί της δημιουργίας προγράμματος καλείται, σε ξεχωριστό νήμα ο αλγόριθμος δημιουργίας του προγράμματος εργασίας με τη μορφή της ανεξάρτητης εφαρμογής. Επιλέχθηκε η λειτουργία του αλγορίθμου σε ξεχωριστό νήμα, για να μπορεί ο χρήστης να έχει τον έλεγχο της εφαρμογής διεπαφής. Αν εκτελούνταν ο αλγόριθμος στο ίδιο νήμα με την διεπαφή, τότε αυτή κατά τον χρόνο εκτέλεσης του αλγορίθμου, θα έμοιαζε

«παγωμένη» και «κολλημένη». Με τον τρόπο αυτό, ο χρήστης έχει τον έλεγχο της εφαρμογής διεπαφής. Δεύτερο όφελος της εκτέλεσης σε άλλο νήμα είναι ότι ο χρήστης μπορεί να «τρέξει» πέραν του ενός, στιγμιότυπα του αλγορίθμου, για διαφορετικού χρόνου επίλυσης προγράμματα εργασίας.

Στη φόρμα υπάρχει και ένα κουμπί που φέρει το όνομα «Προεπισκόπηση και αλλαγές ρυθμίσεων προγράμματος». Όταν πατηθεί, εμφανίζει την παρακάτω φόρμα μέσα από την οποία η Γραμματεία έχει μια πανοραμική άποψη της βάσης δεδομένων.

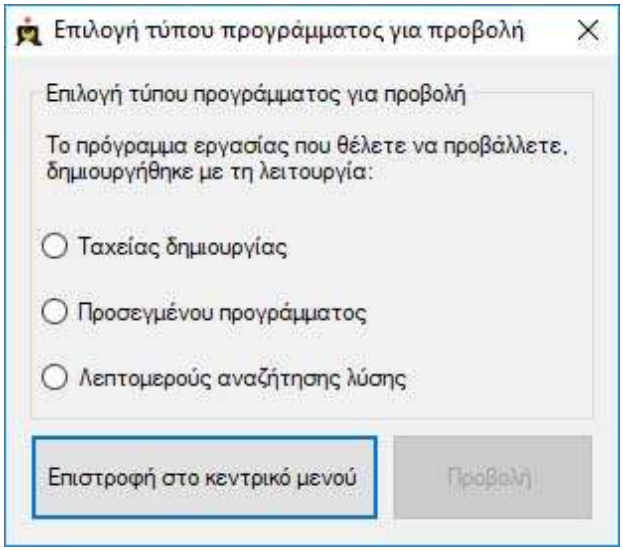
Μέσα από αυτή τη φόρμα, μπορεί να δει όλα τα δεδομένα ανεξάρτητα αν σχετίζονται ή όχι με το πρόγραμμα εργασίας. Μπορεί, για παράδειγμα, να δει τί περιλαμβάνει όσο αφορά τις απαιτήσεις του προσωπικού για εργασία ή ανάπαυση, ένα αποθηκευμένο χρονικό διάστημα, κατά το οποίο δημιουργήθηκε ένα πρόγραμμα, ή ποιες ημέρες ή/και βάρδιες έχει δηλώσει ένας νοσοκόμος να εργαστεί ή να αναπαυθεί, ανεξαρτήτως χρονικού διαστήματος.



Εικόνα 81: Πίνακας προεπισκόπησης δεδομένων και αλλαγής αυτών

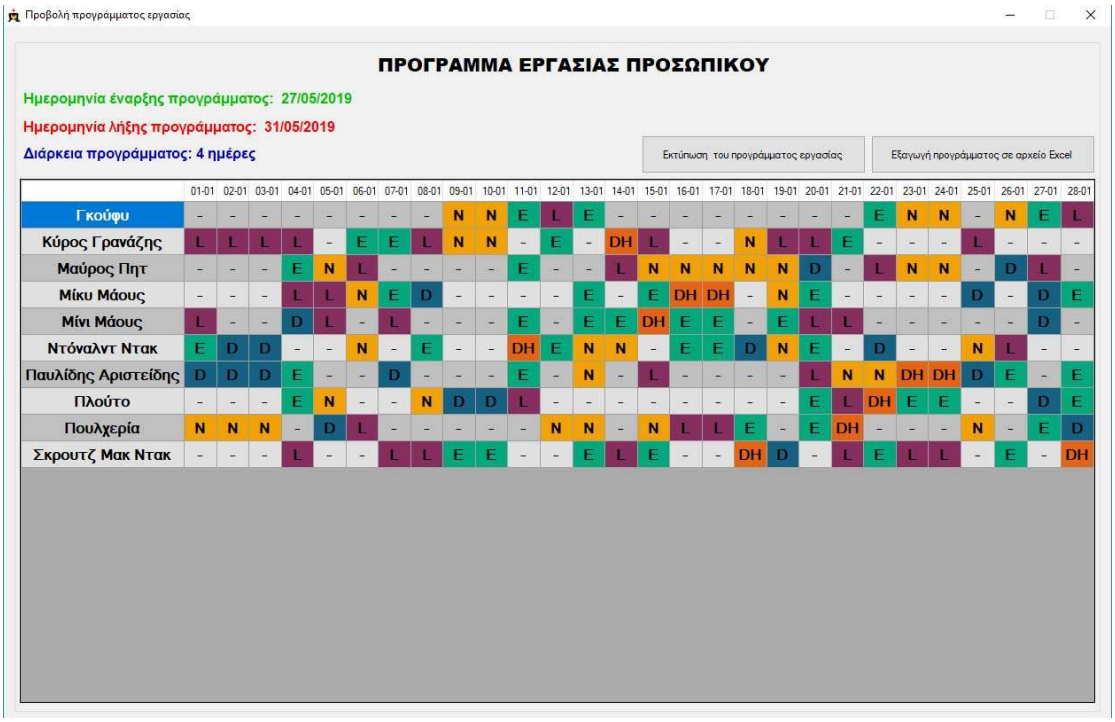
Τέλος, αφού έχει δημιουργηθεί το πρόγραμμα εργασίας και έχει εξαχθεί από τον αλγόριθμο σε ένα αρχείο κειμένου, αυτό παραλαμβάνεται από την εφαρμογή όταν η Γραμματεία μέσα

από την παρακάτω φόρμα, επιλέξει τον τύπο του προγράμματος που δημιούργησε και πατήσει το κουμπί «Προβολή» της φόρμας.



Εικόνα 82: Φόρμα επιλογής είδους προγράμματος εργασίας για εμφάνιση

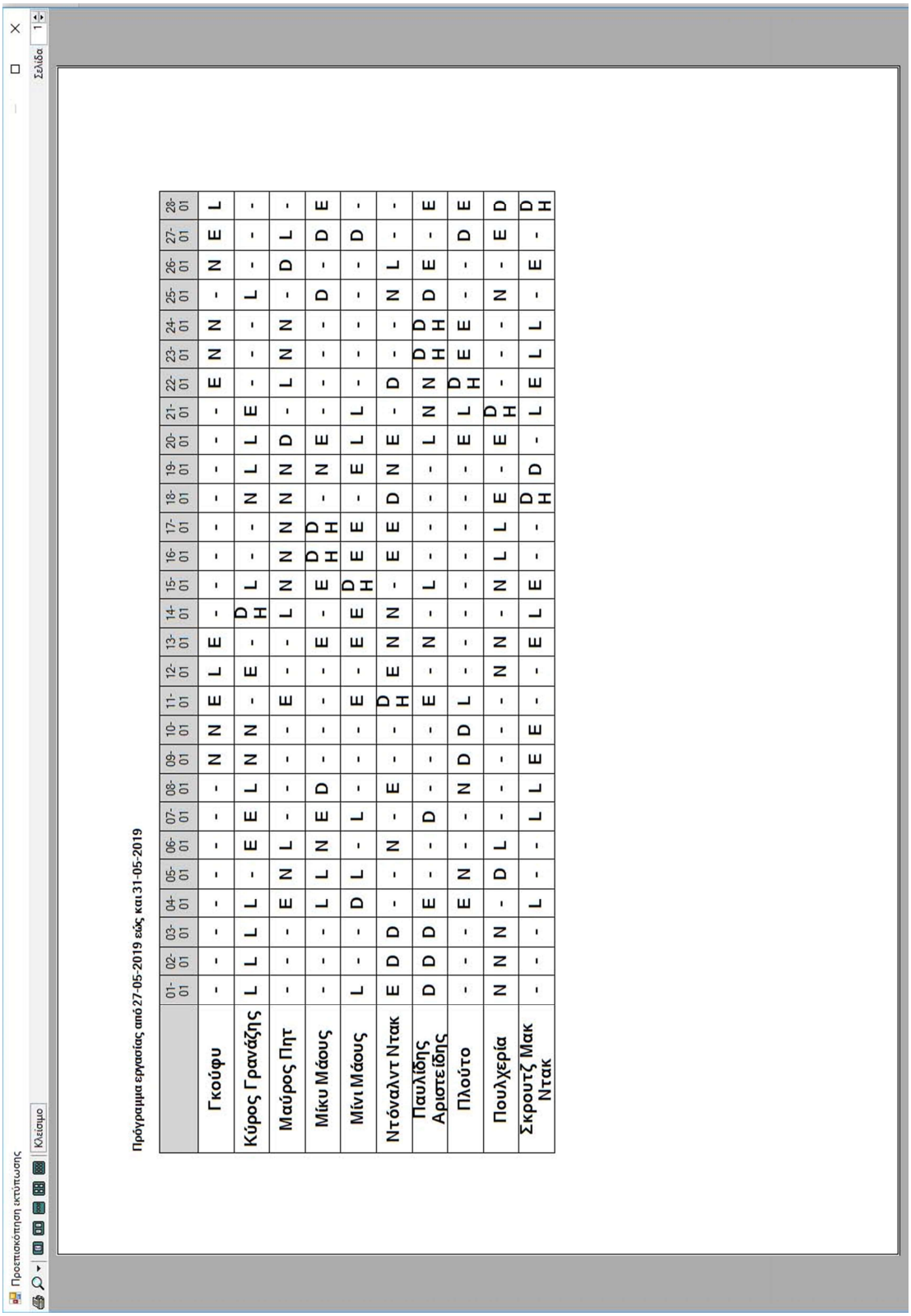
Κατόπιν πραγματοποιείται λήψη του προγράμματος εργασίας από την εφαρμογή NurseRostering και στη συνέχεια αναλύεται και τοποθετείται σε έναν πίνακα – πρόγραμμα εργασίας. Προς διευκόλυνση της Γραμματείας και για εύκολη ανάγνωση, οι βάρδιες κάθε υπαλλήλου για κάθε ημέρα είναι ξεχωριστά χρωματισμένες, ανάλογα με το είδος της βάρδιας. Μέσα από τη φόρμα αυτή, η Γραμματεία μπορεί να πραγματοποιήσει αλλαγές στο δημιουργημένο πρόγραμμα εργασίας, αν το κρίνει απαραίτητο ή αν ζητηθεί από κάποιους νοσοκόμους, για παράδειγμα, στα πλαίσια αμοιβαίας αλλαγής υπηρεσίας. Σε αυτή την περίπτωση, τα χρώματα των βαρδιών παραμένουν όπως δημιουργήθηκαν αρχικά, για να είναι σε θέση η Γραμματεία να έχει την αρχική εικόνα του προγράμματος εργασίας. Η επόμενη εικόνα αποτυπώνει την κατάσταση του προγράμματος όταν έχει εμφανίσει το πρόγραμμα εργασίας που δημιούργησε ο αλγόριθμος.



	01-01	02-01	03-01	04-01	05-01	06-01	07-01	08-01	09-01	10-01	11-01	12-01	13-01	14-01	15-01	16-01	17-01	18-01	19-01	20-01	21-01	22-01	23-01	24-01	25-01	26-01	27-01	28-01
Γκούφυ	-	-	-	-	-	-	-	-	N	N	E	L	E	-	-	-	-	-	-	-	-	E	N	N	-	N	E	L
Κύρος Γρανάζης	L	L	L	L	-	E	E	L	N	N	-	E	-	DH	L	-	-	N	L	L	E	-	-	-	L	-	-	-
Μαύρος Πητ	-	-	-	E	N	L	-	-	-	-	E	-	-	L	N	N	N	N	N	D	-	L	N	N	-	D	L	-
Μίκυ Μάους	-	-	-	L	L	N	E	D	-	-	-	-	E	-	E	DH	DH	-	N	E	-	-	-	-	D	-	D	E
Μίνι Μάους	L	-	-	D	L	-	L	-	-	-	E	-	E	E	DH	E	E	-	E	L	L	-	-	-	-	-	D	-
Ντόναλντ Ντακ	E	D	D	-	-	N	-	E	-	-	DH	E	N	N	-	E	E	D	N	E	-	D	-	-	N	L	-	-
Παυλίδης Αριστείδης	D	D	D	E	-	-	D	-	-	-	E	-	N	-	L	-	-	-	-	L	N	N	DH	DH	D	E	-	E
Πλούτο	-	-	-	E	N	-	-	N	D	D	L	-	-	-	-	-	-	-	-	E	L	DH	E	E	-	-	D	E
Πουλχερία	N	N	N	-	D	L	-	-	-	-	-	N	N	-	N	L	L	E	-	E	DH	-	-	-	N	-	E	D
Σκρουτζ Μακ Ντακ	-	-	-	L	-	-	L	L	E	E	-	-	E	L	E	-	-	DH	D	-	L	E	L	L	-	E	-	DH

Εικόνα 83: Πρόγραμμα εργασίας προσωπικού

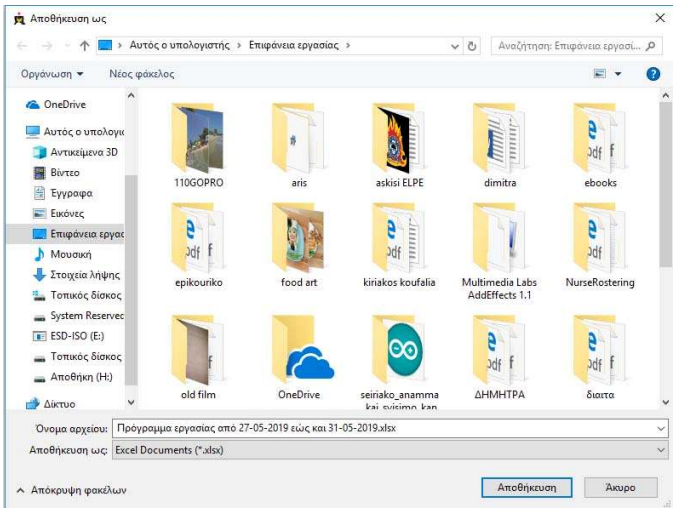
Στην παραπάνω φόρμα, διακρίνονται δύο κουμπιά. Το ένα αφορά στην εκτύπωση του προγράμματος εργασίας, όπως αποτυπώνεται, χωρίς να εμφανίζεται το χρώμα των κελιών στην εκτύπωση, όπως φαίνεται στην εικόνα της επόμενης σελίδας.



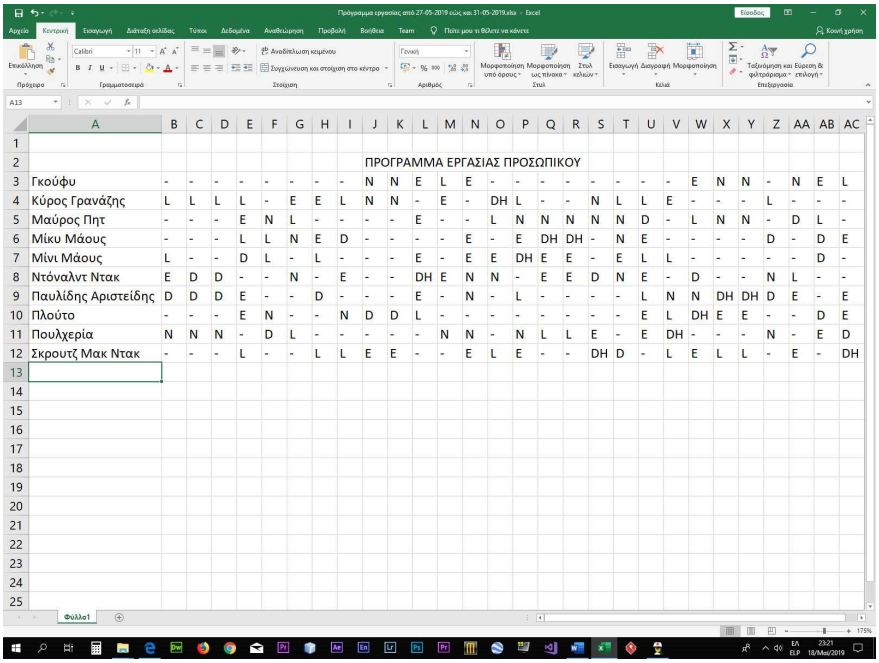
	01-01	02-01	03-01	04-01	05-01	06-01	07-01	08-01	09-01	10-01	11-01	12-01	13-01	14-01	15-01	16-01	17-01	18-01	19-01	20-01	21-01	22-01	23-01	24-01	25-01	26-01	27-01	28-01
Γκούφυ	-	-	-	-	-	-	-	-	N	N	E	L	E	-	-	-	-	-	-	-	-	E	N	N	-	N	E	L
Κύρος Γρανάζης	L	L	L	L	-	E	E	L	N	N	-	E	-	D H	L	-	-	N	L	L	E	-	-	-	L	-	-	-
Μαύρος Πητ	-	-	-	E	N	L	-	-	-	-	E	-	-	L	N	N	N	N	N	D	-	L	N	N	-	D	L	-
Μίκυ Μάους	-	-	-	L	L	N	E	D	-	-	-	-	E	-	E	D H	D H	-	N	E	-	-	-	-	D	-	D	E
Μίνι Μάους	L	-	-	D	L	-	L	-	-	-	E	-	E	E	D H	E	E	-	E	L	L	-	-	-	-	-	D	-
Ντόναλντ Ντακ	E	D	D	-	-	N	-	E	-	-	D H	E	N	N	-	E	E	D	N	E	-	D	-	-	N	L	-	-
Παυλίδης Αριστείδης	D	D	D	E	-	-	D	-	-	-	E	-	N	-	L	-	-	-	-	L	N	N	D H	D H	D	E	-	E
Πλούτο	-	-	-	E	N	-	-	N	D	D	L	-	-	-	-	-	-	-	-	E	L	D H	E	E	-	-	D	E
Πουλχερία	N	N	N	-	D	L	-	-	-	-	-	N	N	-	N	L	L	E	-	E	D H	-	-	-	N	-	E	D
Σκρουτζ Μακ Ντακ	-	-	-	L	-	-	L	L	E	E	-	-	E	L	E	-	-	D H	D	-	L	E	L	L	-	E	-	D H

Εικόνα 84: Προεπισκόπηση εκτύπωσης προγράμματος εργασίας

Η δεύτερη ενέργεια που έχει στη διάθεσή του ο χρήστης, είναι να αποθηκεύσει το πρόγραμμα σε ένα αρχείο του προγράμματος Microsoft Excel σε μορφή αρχείου *.xlsx για περεταίρω επεξεργασία, αποστολή με e-mail κλπ. Όταν πατήσει το κουμπί για αποθήκευση του προγράμματος σε αρχείο Excel, τότε ερωτάται που θέλει να το αποθηκεύσει και του προτείνεται ένα όνομα το οποίο περιλαμβάνει και τις ημερομηνίες έναρξης και λήξης του προγράμματος εργασίας, διασφαλίζοντας έτσι μια σειρά για τυχόν αρχειοθέτησή τους από τη Γραμματεία.



Εικόνα 85: Παράθυρο αποθήκευσης του προγράμματος εργασίας σε μορφή αρχείου *.xlsx



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W	X	Y	Z	AA	AB	AC
1																													
2										ΠΡΟΓΡΑΜΜΑ ΕΡΓΑΣΙΑΣ ΠΡΟΣΩΠΙΚΟΥ																			
3	Γκούφυ	-	-	-	-	-	-	-	-	N	N	E	L	E	-	-	-	-	-	-	-	-	E	N	N	-	N	E	L
4	Κύρος Γρανάζης	L	L	L	L	-	E	E	L	N	N	-	E	-	DH	L	-	-	N	L	L	E	-	-	-	L	-	-	-
5	Μαύρος Πητ	-	-	-	E	N	L	-	-	-	-	E	-	-	L	N	N	N	N	N	D	-	L	N	N	-	D	L	-
6	Μίκυ Μάους	-	-	-	L	L	N	E	D	-	-	-	-	E	-	E	DH	DH	-	N	E	-	-	-	-	D	-	D	E
7	Μίνι Μάους	L	-	-	D	L	-	L	-	-	-	E	-	E	E	DH	E	E	-	E	L	L	-	-	-	-	-	D	-
8	Ντόναλντ Ντακ	E	D	D	-	-	N	-	E	-	-	DH	E	N	N	-	E	E	D	N	E	-	D	-	-	N	L	-	-
9	Παυλίδης Αριστείδης	D	D	D	E	-	-	D	-	-	-	E	-	N	-	L	-	-	-	-	L	N	N	DH	DH	D	E	-	E
10	Πλούτο	-	-	-	E	N	-	-	N	D	D	L	-	-	-	-	-	-	-	-	E	L	DH	E	E	-	-	D	E
11	Πουλχερία	N	N	N	-	D	L	-	-	-	-	-	N	N	-	N	L	L	E	-	E	DH	-	-	-	N	-	E	D
12	Σκρουτζ Μακ Ντακ	-	-	-	L	-	-	L	L	E	E	-	-	E	L	E	-	-	DH	D	-	L	E	L	L	-	E	-	DH
13																													

Εικόνα 86: Το πρόγραμμα εργασίας μέσα σε αρχείο μορφής *.xlsx

Κλείνοντας την παρουσίαση – επίδειξη της εφαρμογής, πρέπει να σημειωθεί η ευελιξία στη χρήση της εφαρμογής και η δυνατότητα πλοήγησης μέσα σε αυτή. Αυτό σημαίνει ότι αν ο χρήστης στην απόπειρά του να καταχωρήσει ένα αντικείμενο (υπάλληλο, συμβόλαιο, βάρδια κλπ), αποφασίσει ότι θέλει να καταχωρήσει ένα αντικείμενο άλλης κλάσης, δεν είναι απαραίτητο να εγκαταλείψει τη φόρμα στην οποία εργάζεται αλλά μπορεί να μετακινείται από φόρμα σε φόρμα χωρίς να χάνει τα δεδομένα του.

8. Εργαλεία που χρησιμοποιήθηκαν

8.1 Visual Studio 2017

Το Microsoft Visual Studio (Microsoft Visual Studio) είναι ένα ολοκληρωμένο περιβάλλον ανάπτυξης (IDE) από τη Microsoft. Χρησιμοποιείται για την ανάπτυξη προγραμμάτων ηλεκτρονικών υπολογιστών, καθώς και ιστοσελίδων, εφαρμογών ιστού, υπηρεσιών ιστού και εφαρμογών για κινητά. Το Visual Studio χρησιμοποιεί πλατφόρμες ανάπτυξης λογισμικού της Microsoft, όπως τα Windows API, τα Windows Forms, το Windows Presentation Foundation, το Windows Store και το Microsoft Silverlight. Μπορεί να παράγει τόσο εγγενή κώδικα όσο και διαχειριζόμενο κώδικα.

Το Visual Studio περιλαμβάνει ένα πρόγραμμα επεξεργασίας κώδικα το οποίο υποστηρίζει το IntelliSense (το στοιχείο ολοκλήρωσης κώδικα) καθώς και το refactoring κώδικα. Το ενσωματωμένο πρόγραμμα εντοπισμού σφαλμάτων λειτουργεί τόσο ως εντοπιστής σφαλμάτων σε επίπεδο πηγής όσο και ως εργαλείο εντοπισμού σφαλμάτων σε επίπεδο μηχανής. Άλλα ενσωματωμένα εργαλεία περιλαμβάνουν έναν βοηθό κώδικα, σχεδιαστή μορφών για την κατασκευή εφαρμογών γραφικών διεπαφών αλληλεπίδρασης με το χρήστη, σχεδιαστής ιστοσελίδων, σχεδιαστή κλάσεων και σχεδιαστής σχήματος βάσης δεδομένων. Δέχεται πρόσθετα (plug-ins) που βελτιώνουν τη λειτουργικότητα σχεδόν σε όλα τα επίπεδα - συμπεριλαμβανομένης της προσθήκης υποστήριξης για συστήματα ελέγχου εκδόσεων (όπως το Subversion και το Git) και την προσθήκη νέων εργαλείων όπως οι διορθωτές και οι γραφικούς σχεδιαστές για συγκεκριμένες γλώσσες ή ομάδες εργαλείων για άλλες πτυχές της ανάπτυξης λογισμικού (όπως ο πελάτης Team Foundation Server: Team Explorer).

Το Visual Studio υποστηρίζει 36 διαφορετικές γλώσσες προγραμματισμού και επιτρέπει στον επεξεργαστή κώδικα και στον εντοπιστή σφαλμάτων να υποστηρίζει (σε διαφορετικούς βαθμούς) σχεδόν οποιαδήποτε γλώσσα προγραμματισμού, υπό την προϋπόθεση ότι υπάρχει μια ειδική υπηρεσία γλώσσας. Οι ενσωματωμένες γλώσσες περιλαμβάνουν τις C, C++, C++/CLI, Visual Basic .NET, C#, F#, JavaScript, TypeScript, XML, XSLT, HTML και CSS. Υποστήριξη για άλλες γλώσσες όπως Python, Ruby, Node.js και M μεταξύ άλλων είναι διαθέσιμη μέσω πρόσθετων (plug-ins). Τα Java (και J #) υποστηρίχθηκαν στο παρελθόν.

Η βασική έκδοση του Visual Studio Community Edition, διατίθεται δωρεάν. Το σύνθημα για την έκδοση του Visual Studio Community είναι «Δωρεάν, πλήρως εξοπλισμένο IDE για φοιτητές, ανοιχτού κώδικα και μεμονωμένους προγραμματιστές».

Η έκδοση Visual Studio που υποστηρίζεται αυτήν τη στιγμή είναι η 2019.

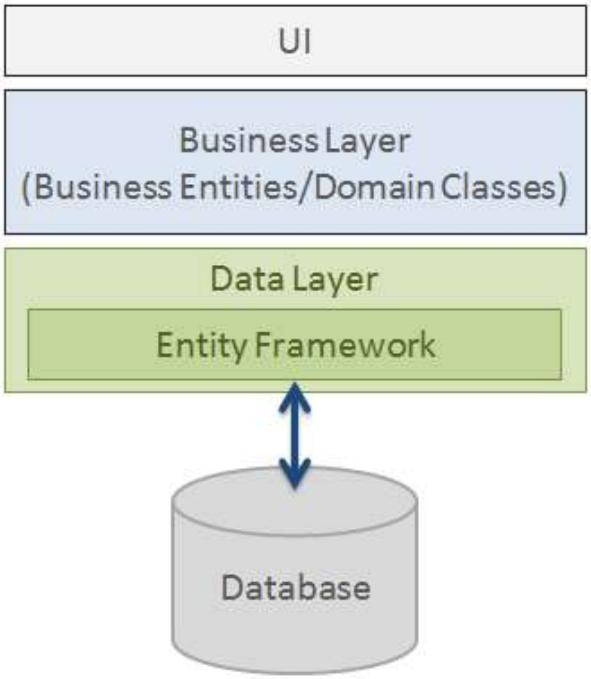
8.2 Entity Framework

Στο παρελθόν, πριν από το .NET 3.5, ήταν συχνά χρησιμοποιούμενη η συγγραφή κώδικα για το ADO.NET ή το Enterprise Data Access Block για να αποθηκεύσουμε ή να ανακτήσουμε δεδομένα εφαρμογών από την μια βάση δεδομένων. Έπρεπε να δημιουργηθεί μια σύνδεση με τη βάση δεδομένων, να δημιουργηθεί ένα σετ δεδομένων (Data Set) για να φέρουμε ή να υποβάλλουμε τα δεδομένα στη βάση δεδομένων και να μετατρέπουμε δεδομένα από το DataSet σε αντικείμενα .NET. Πρόκειται για μια όχι και τόσο ευέλικτη πρακτική αλλά και επιρρεπή σε σφάλματα διαδικασία. Η Microsoft έχει δημιουργήσει ένα πλαίσιο που ονομάζεται «Entity Framework» (Entity Framework) για να αυτοματοποιήσει όλες αυτές τις δραστηριότητες που σχετίζονται με τη βάση δεδομένων για την εφαρμογή σας.

Το Entity Framework είναι ένα πλαίσιο ORM (**O**bject – **R**elational **M**apping) ανοικτού κώδικα για εφαρμογές .NET που υποστηρίζονται από τη Microsoft. Δίνει τη δυνατότητα στους προγραμματιστές να δουλεύουν με δεδομένα χρησιμοποιώντας αντικείμενα κατηγοριών συγκεκριμένων κλάσεων χωρίς να εστιάζουν στους πίνακες βάσεων δεδομένων και στις στήλες όπου αποθηκεύονται αυτά τα δεδομένα. Με το Entity Framework, οι προγραμματιστές μπορούν να δουλέψουν σε υψηλότερο επίπεδο αφαίρεσης όταν ασχολούνται με δεδομένα και μπορούν να δημιουργήσουν και να διατηρήσουν εφαρμογές με λιγότερο κώδικα σε σχέση με τις παραδοσιακές εφαρμογές. Το Entity Framework εκτελεί λειτουργίες **CRUD** (**C**reate – **R**ead – **U**pdate – **D**elete δηλαδή Δημιουργία – Ανάγνωση – Ενημέρωση – Διαγραφή) επάνω στα δεδομένα της βάσης χωρίς τη συγγραφή SQL ερωτημάτων.

Επίσημος ορισμός: «Το Entity Framework είναι ένας αντικειμενικός σχεσιακός χαρτογράφος (O/RM) ο οποίος επιτρέπει στους προγραμματιστές .NET να συνεργάζονται με μια βάση δεδομένων χρησιμοποιώντας αντικείμενα .NET. Εξαλείφει την ανάγκη για το μεγαλύτερο μέρος του κώδικα πρόσβασης δεδομένων που συνήθως πρέπει να γράψουν οι προγραμματιστές.»

Η παρακάτω εικόνα, δείχνει πού ακριβώς τοποθετείται το Entity Framework μέσα στην εφαρμογή μας.



Εικόνα 87: Τοποθεσία του Entity Framework στην εφαρμογή.

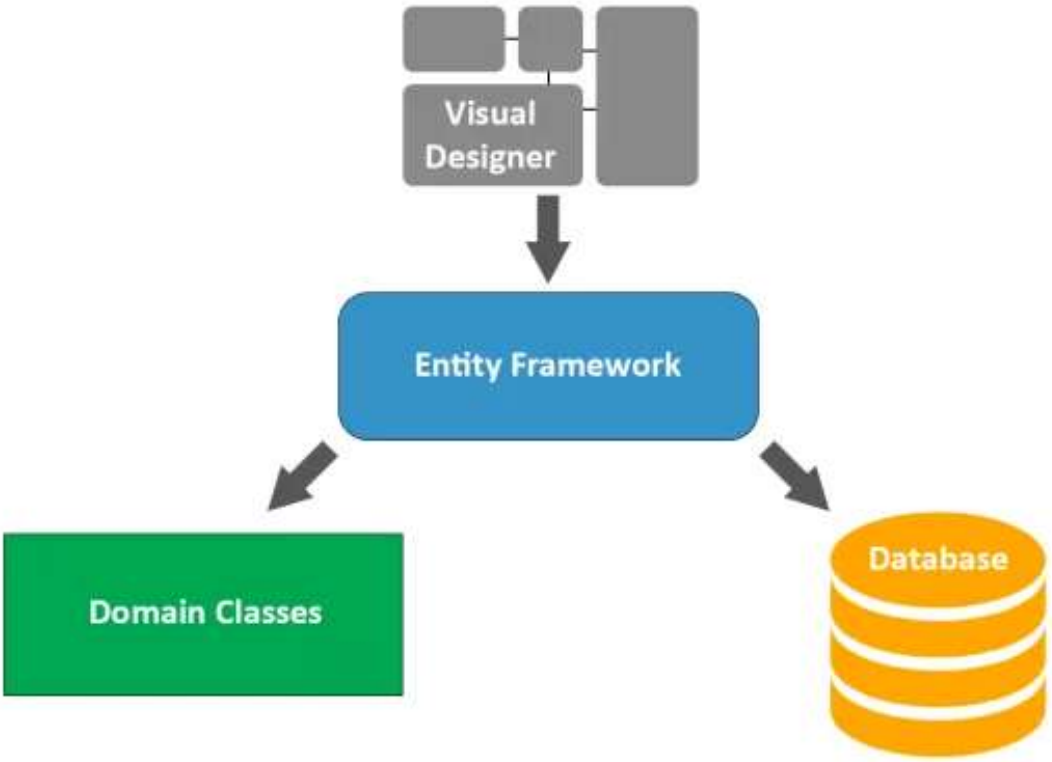
Το Entity Framework προσφέρει στον προγραμματιστή τρεις πολύ σημαντικές προσεγγίσεις, όσο αφορά στη χρήση του μέσα στην εφαρμογή. Το ποια από τις τρεις θα επιλεχθεί από τον προγραμματιστή, αφήνεται ουσιαστικά στο στυλ εργασίας του καθενός αλλά και στις απαιτήσεις που μπορεί να έχει μια εφαρμογή. Οι προσεγγίσεις αυτές είναι:

8.2.1. Πρώτα το μοντέλο

Αν δεν γνωρίζουμε τα περισσότερα εργαλεία σχεδιασμού IDE, όπως το XML Scheme DataSet (XSD) και το Visual Interface Designer Model XML (Microsoft Dynamics NAV) που βασίζονται σε XML (EDMX), η προσέγγιση *Πρώτα το μοντέλο* (Πρώτα το μοντέλο) μπορεί μάλλον να μας προκαλέσει σύγχυση. Το κλειδί για την κατανόηση αυτής της προσέγγισης είναι να αναγνωριστεί το γεγονός ότι η λέξη *Μοντέλο* εδώ έχει σκοπό να καθορίσει ένα διάγραμμα που δημιουργήθηκε με τα εργαλεία σχεδιασμού. Αυτό το διάγραμμα θα χρησιμοποιηθεί στη συνέχεια από το Entity Framework για να δημιουργήσει αυτόματα τη δέσμη ενεργειών βάσης δεδομένων SQL και τα αρχεία πηγαίου κώδικα του μοντέλου δεδομένων.

Για να το συνοψίσουμε, μπορούμε να πούμε ότι η μετάβαση στην προσέγγιση *Πρώτα το μοντέλο* ουσιαστικά σημαίνει "να δημιουργήσουμε ένα διάγραμμα και να αφήσουμε το Entity Framework να δημιουργήσει / να ενημερώσει το υπόλοιπο ανάλογα". Ο

τρόπος λειτουργίας της προσέγγισης *Πρώτα το μοντέλο* δεδομένων φαίνεται στην επόμενη εικόνα:



Εικόνα 88: Λειτουργία προσέγγισης Πρώτα το μοντέλο

Η προσέγγιση αυτή έχει τα ακόλουθα πλεονεκτήματα:

- Θα μπορέσουμε να δημιουργήσουμε το σχήμα βάσης δεδομένων και το διάγραμμα κλάσης στο σύνολό του χρησιμοποιώντας ένα εργαλείο οπτικής σχεδίασης, το οποίο μπορεί να είναι εξαιρετικό όταν η βάση δεδομένων είναι αρκετά μεγάλη.
- Κάθε φορά που αλλάζει η βάση δεδομένων, το μοντέλο μπορεί να ενημερωθεί αναλόγως, χωρίς απώλεια δεδομένων

Αλλά και τα παρακάτω μειονεκτήματα:

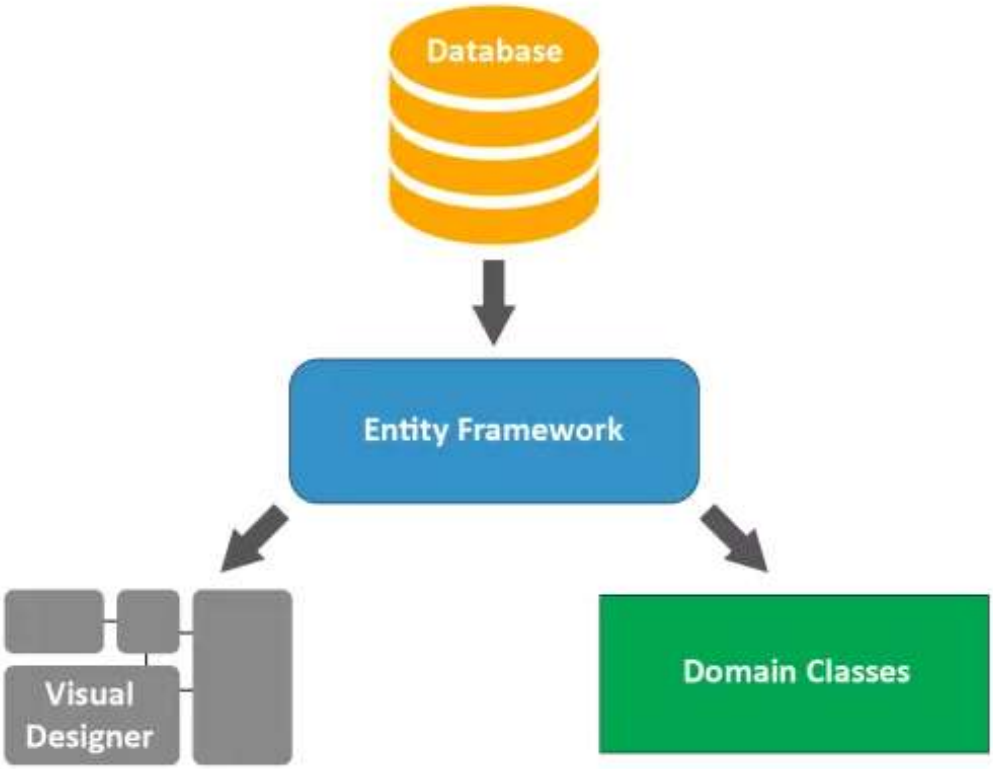
- Τα σενάρια SQL που δημιουργούνται αυτόματα από το διάγραμμα – οδηγό μπορούν να οδηγήσουν σε απώλεια δεδομένων σε περίπτωση ενημερώσεων. Μια εύκολη λύση γι αυτό, είναι να δημιουργηθούν τα σενάρια σε αρχείο/α στο δίσκο και να τροποποιηθούν κατάλληλα με το χέρι, κάτι που θα απαιτήσει ικανοποιητική γνώση γλώσσας SQL.

- Η ενασχόληση με το διάγραμμα μπορεί να είναι δύσκολη, ειδικά εάν θέλουμε να έχουμε ακριβή έλεγχο των κλάσεων του μοντέλου μας. δεν θα είμαστε πάντα σε θέση να πάρουμε αυτό που θέλουμε, καθώς ο πραγματικός κώδικας πηγής θα δημιουργηθεί αυτόματα από ένα εργαλείο.

8.2.2. Πρώτα η βάση δεδομένων

Δεδομένων των μειονεκτημάτων της προσέγγισης Πρώτα το μοντέλο, μπορούμε να σκεφτούμε ότι η προσέγγιση *Πρώτα η βάση δεδομένων* (Πρώτα η βάση δεδομένων) μπορεί να είναι καλύτερη και ευκολότερη προσέγγιση. Αυτό ισχύει εάν έχουμε ήδη μια βάση δεδομένων. Τούτο συμβαίνει, γιατί η προσέγγιση *Πρώτα η βάση δεδομένων* είναι παρόμοια με την προσέγγιση *Πρώτα το μοντέλο.* Το σημείο διαφοροποίησης των δύο μοντέλων είναι πως αντί να σχεδιάσουμε το EDMX με το χέρι και να δημιουργήσουμε τη δέσμη ενεργειών SQL για να δημιουργήσουμε τη βάση δεδομένων, δημιουργούμε τη βάση δεδομένων και στη συνέχεια δημιουργούμε το EDMX χρησιμοποιώντας το εργαλείο Designer Framework Entity.

Συνοψίζοντας, μπορούμε να πούμε ότι η μετάβαση στην προσέγγιση *Πρώτα η βάση δεδομένων* σημαίνει "δημιουργία της βάσης δεδομένων και αφήνουμε το Entity Framework να δημιουργήσει / ενημερώσει τα υπόλοιπα ανάλογα". Ο τρόπος λειτουργίας της προσέγγισης *Πρώτα η βάση δεδομένων* φαίνεται στην επόμενη εικόνα:



Εικόνα 89: Λειτουργία προσέγγισης Πρώτα η βάση δεδομένων

Παρακάτω παρατίθενται τα πλεονεκτήματα αυτής της προσέγγισης:

- Εάν διαθέτουμε ήδη υπάρχουσα βάση δεδομένων, αυτή η προσέγγιση είναι η καταλληλότερη, καθώς θα μας απαλλάξει από την ανάγκη να την δημιουργήσουμε.
- Ο κίνδυνος απώλειας δεδομένων θα περιοριστεί στο ελάχιστο, διότι οποιαδήποτε αλλαγή ή ενημέρωση θα πραγματοποιηθεί πάντα στην υπάρχουσα βάση δεδομένων.

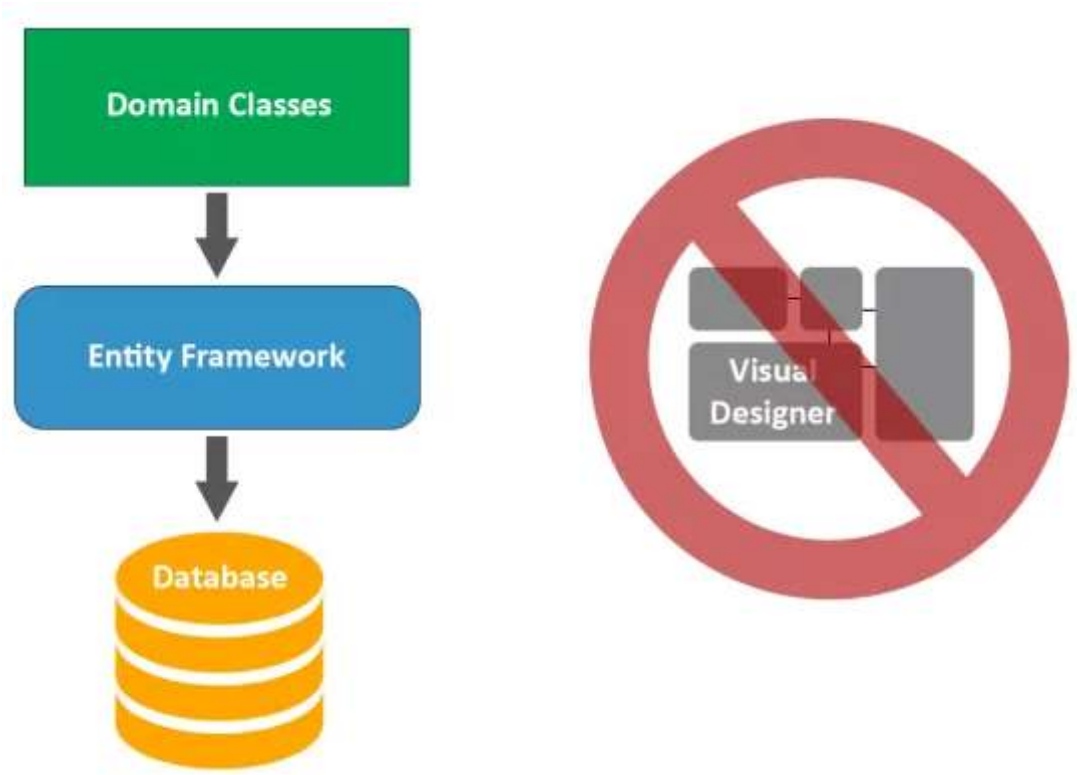
Και εδώ είναι τα μειονεκτήματα:

- Η μη αυτόματη ενημέρωση της Βάσης Δεδομένων μπορεί να είναι δύσκολη εάν ασχοληθούμε με συστοιχίες, πολλαπλά στιγμιότυπα ή με ένα περιβάλλον ανάπτυξης / δοκιμών / παραγωγής, καθώς θα πρέπει να τις διατηρήσουμε με το χέρι, αντί να βασιστούμε σε ενημερώσεις / μετακινήσεις που βασίζονται στον κώδικα ή τα σενάρια SQL που δημιουργήθηκαν αυτόματα.
- Θα έχουμε ακόμη μικρότερο έλεγχο πάνω στις κλάσεις μοντέλου (και τον πηγαίο κώδικα) που δημιουργήθηκαν αυτόματα από ό, τι χρησιμοποιώντας την προσέγγιση *Πρώτα το μοντέλο*. Απαιτείται εκτεταμένη γνώση του Entity Framework, διαφορετικά, για την ολοκλήρωση του έργου θα χρειαστεί μεγάλη δαπάνη χρόνου.

8.2.3. Πρώτα ο κώδικας

Τελευταία, αλλά εξίσου σημαντική, είναι η εμβληματική προσέγγιση του Entity Framework από την έκδοση 4 και μετά, η οποία επιτρέπει μια κομψή και εξαιρετικά αποδοτική ροή εργασιών για το μοντέλο δεδομένων. Η εφαρμογή που πραγματεύεται η παρούσα πτυχιακή εργασία είναι γραμμένη με αυτή την προσέγγιση. Ο λόγος της προσφυγής στην προσέγγιση αυτή, μπορεί εύκολα να γίνει αντιληπτός από το όνομά της. Η προσέγγιση *Πρώτα ο κώδικας (Code–First)* (Πρώτα ο κώδικας) επιτρέπει στον προγραμματιστή να δημιουργεί αντικείμενα από τις κλάσεις που έχει δημιουργήσει για την εφαρμογή του, χωρίς την ανάγκη οποιουδήποτε εργαλείου σχεδίασης, αρχείων χαρτογράφησης XML ή δύσχρηστων εργαλείων αυτοματοποιημένου κώδικα.

Για να το συνοψίσουμε, μπορούμε να πούμε ότι η προσέγγιση *Πρώτα ο κώδικας* σημαίνει να γράψουμε τις κλάσεις οντοτήτων Μοντέλου Δεδομένων που θα χρησιμοποιήσουμε στο έργο μας και να αφήσουμε το Entity Framework να δημιουργήσει την βάση δεδομένων, όπως φαίνεται και στην επόμενη εικόνα:



Εικόνα 90: Λειτουργία προσέγγισης Πρώτα ο κώδικας

Η προσέγγιση *Πρώτα ο κώδικας* έχει τα ακόλουθα πλεονεκτήματα:

- Δεν χρειάζονται διαγράμματα και σχεδιαστικά εργαλεία, το οποίο σημαίνει ότι τουλάχιστον για εφαρμογές μικρού έως μεσαίου μεγέθους μας εξοικονομεί πολύ χρόνο.
- Επιτρέπει στον προγραμματιστή να ακολουθεί μια προσέγγιση *διαμόρφωσης κατά την ανάπτυξη*, για διαχείριση των πιο συνηθισμένων καταστάσεων και ταυτόχρονα να του δίνει την ευκαιρία να προσαρμόζει την υπό σχεδίαση εφαρμογή τροποποιώντας τις ιδιότητές της χωρίς να χρειάζεται να προσαρμόσει τη χαρτογράφηση της βάσης δεδομένων.

Ωστόσο, έχει και αυτή η προσέγγιση μερικά μειονεκτήματα:

- Απαιτείται καλή γνώση της γλώσσας προγραμματισμού ORM και των συμβάσεων που ακολουθεί (C# για το Entity Framework).
- Η διατήρηση της βάσης δεδομένων, μπορεί να είναι δύσκολη μερικές φορές, καθώς και οι ενημερώσεις αυτής, γιατί μπορεί να προκληθεί ακούσια απώλεια δεδομένων. Αυτά τα προβλήματα του Entity Framework άρχισαν να επιλύονται και να

ξεπερνιόνται από την έκδοση EF4.3 και έκτοτε, μετριάζει σε μεγάλο βαθμό το πρόβλημα, αν και επηρέασε αρνητικά την ευκολία μάθησης.

8.3 Visual Paradigm

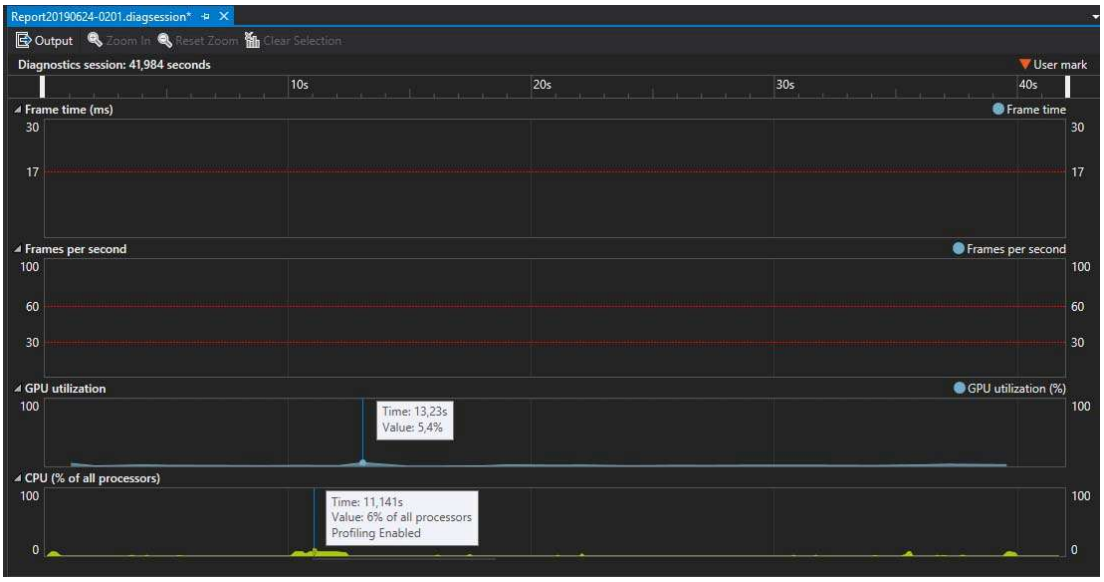
Το πρόγραμμα Visual Paradigm είναι μια εφαρμογή με την οποία μπορούμε να δημιουργήσουμε και να διαχειριστούμε τα διάφορα στάδια κατά τη διάρκεια δημιουργίας εφαρμογών. Μπορεί ο χρήστης να σχεδιάσει σε UML διάφορα διαγράμματα (Περιπτώσεων χρήσης, ευρωστίας ακολουθίας, κλάσεων κλπ) για αρκετές γλώσσες και στη συνέχεια το Visual Paradigm να δημιουργήσει τον κώδικα για τα διαγράμματα κλάσεων, μπορεί να σχεδιάσει σε SCRUM διαγράμματα για την αποδοτικότερη διαχείριση μιας ομάδας, τήρηση χρονοδιαγραμμάτων και παρακολούθηση της πορείας της κατασκευής μιας εφαρμογής. Μια άλλη δυνατότητα του Visual Paradigm είναι να συνεργάζεται με διάφορα μεγάλα και γνωστά clouds όπως τα Azure, Google, Tencent, Alibaba, IBM και Oracle. Περισσότερες πληροφορίες μπορούμε να βρούμε στο διαδικτυακό ιστότοπο https://www.visual-paradigm.com/.

Εικόνα 91: Visual Paradigm

Στη συγκεκριμένη εφαρμογή, χρησιμοποιήθηκε η δωρεάν έκδοση Visual Paradigm v.15.2 Community Edition η οποία είναι για ακαδημαϊκή χρήση, για την παραγωγή των διαγραμμάτων Περιπτώσεων Χρήσης, ευρωστίας και ακολουθίας.

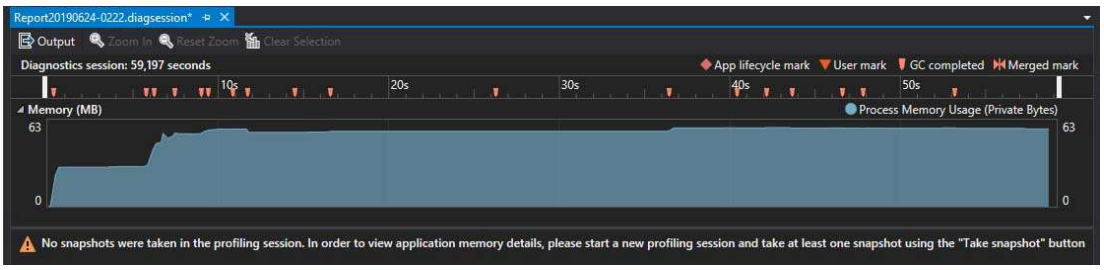
9. Συμπεράσματα – Προτάσεις

Μετά και την ολοκλήρωση της εφαρμογής, πραγματοποιήθηκαν κάποιες μετρήσεις σχετικά με τη χρήση του κεντρικού επεξεργαστή του συστήματος (CPU) και του επεξεργαστή της κάρτας γραφικών (GPU). Η χρήση των δύο επεξεργαστών βρέθηκε να είναι σε πάρα πολύ χαμηλά επίπεδα. Τα ανώτερα επίπεδα που σημειώθηκαν, ήταν κατά τη διαδικασία λειτουργίας του Microsoft Entity Framework. Τα αποτελέσματα φαίνονται στην παρακάτω εικόνα.



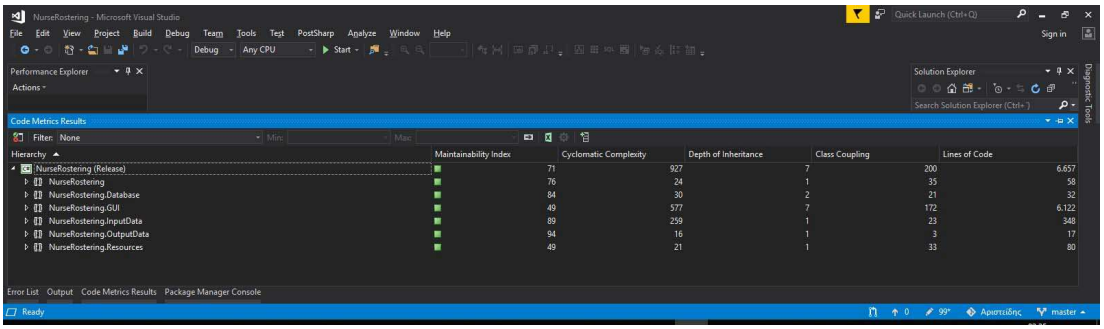
Εικόνα 92: Απασχόληση κεντρικού επεξεργαστή και επεξεργαστή κάρτας γραφικών

Σχετικά με τις απαιτήσεις της μνήμης RAM, μπορούμε να πούμε ότι και αυτές δεν είναι μεγάλες. Η εφαρμογή απαιτεί μόνο 63MB μνήμης RAM, όπως φαίνεται στην επόμενη εικόνα. Το γραφικό περιβάλλον της εφαρμογής κατά την εκκίνησή του απαιτεί μόλις 31MB και τα υπόλοιπα τα απαιτεί το Microsoft Entity Framework για τη λειτουργία του.

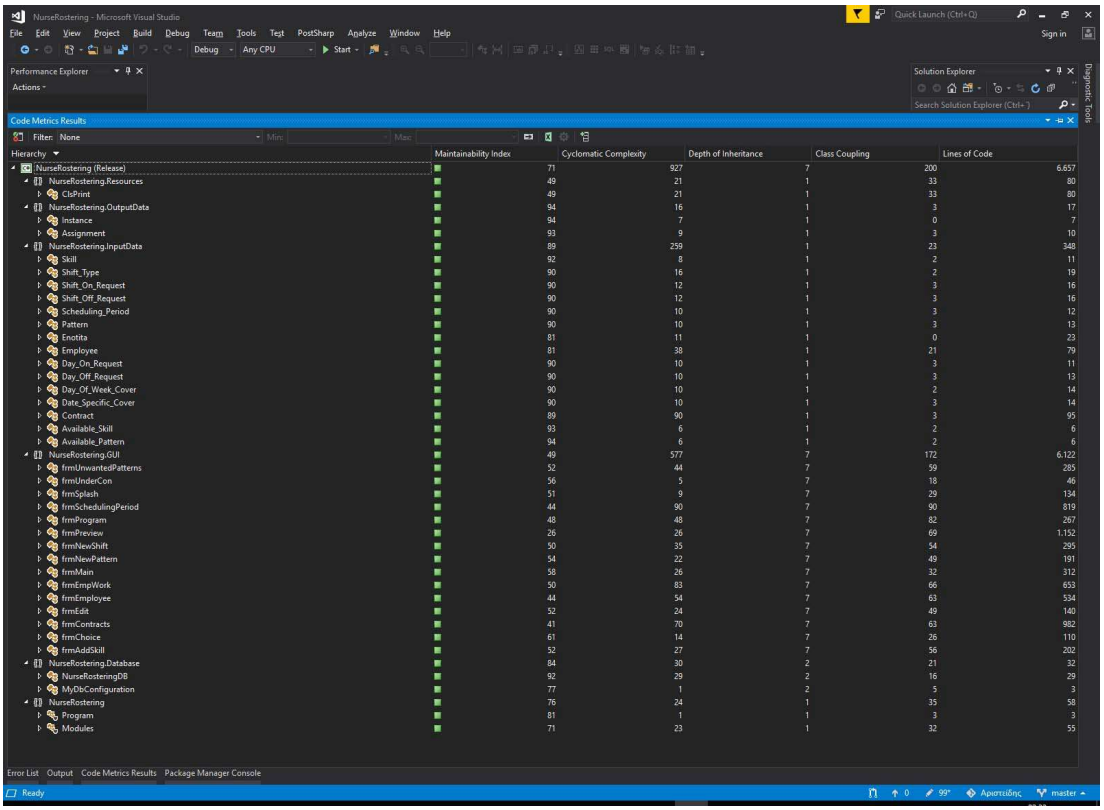


Εικόνα 93: Απαιτήσεις μνήμης RAM

Στη συνέχεια παρατίθενται κάποιες μετρήσεις για τον κώδικα της εφαρμογής αναφορικά με την κυκλωματική πολυπλοκότητά του, το βάθος κληρονομικότητας, τις γραμμές του κώδικα κλπ. Οι παρακάτω δύο εικόνες δείχνουν τα αποτελέσματα συγκεντρωτικά ανά πακέτο κλάσεων αλλά και αναλυτικά, ανά κλάση.

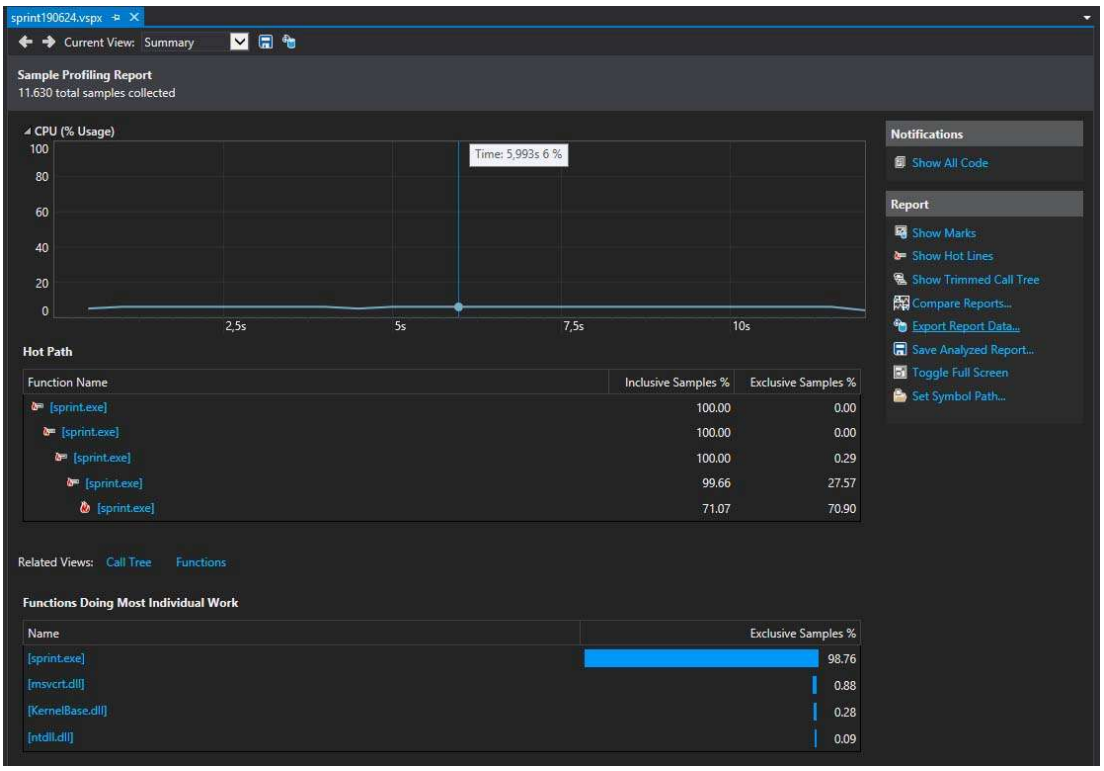


Εικόνα 94: Συγκεντρωτικά αποτελέσματα μετρήσεων κώδικα ανά πακέτο κλάσεων

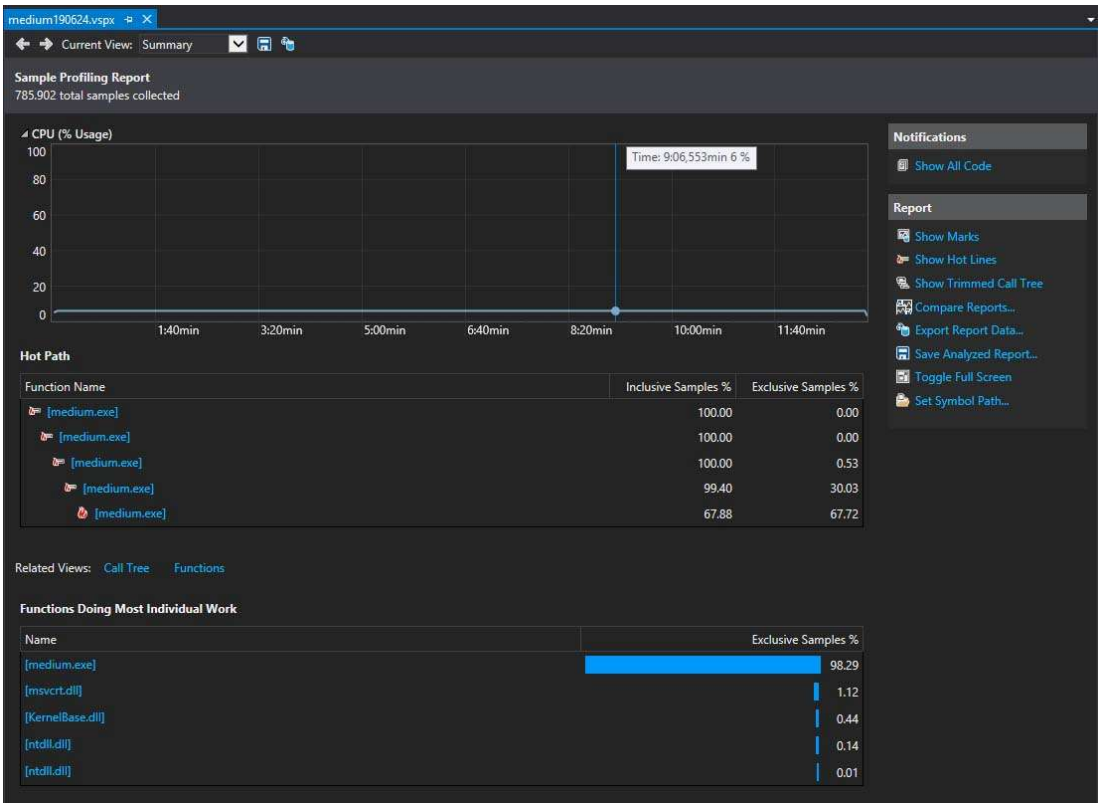


Εικόνα 95: Αναλυτικά αποτελέσματα μετρήσεων κώδικα ανά κλάση

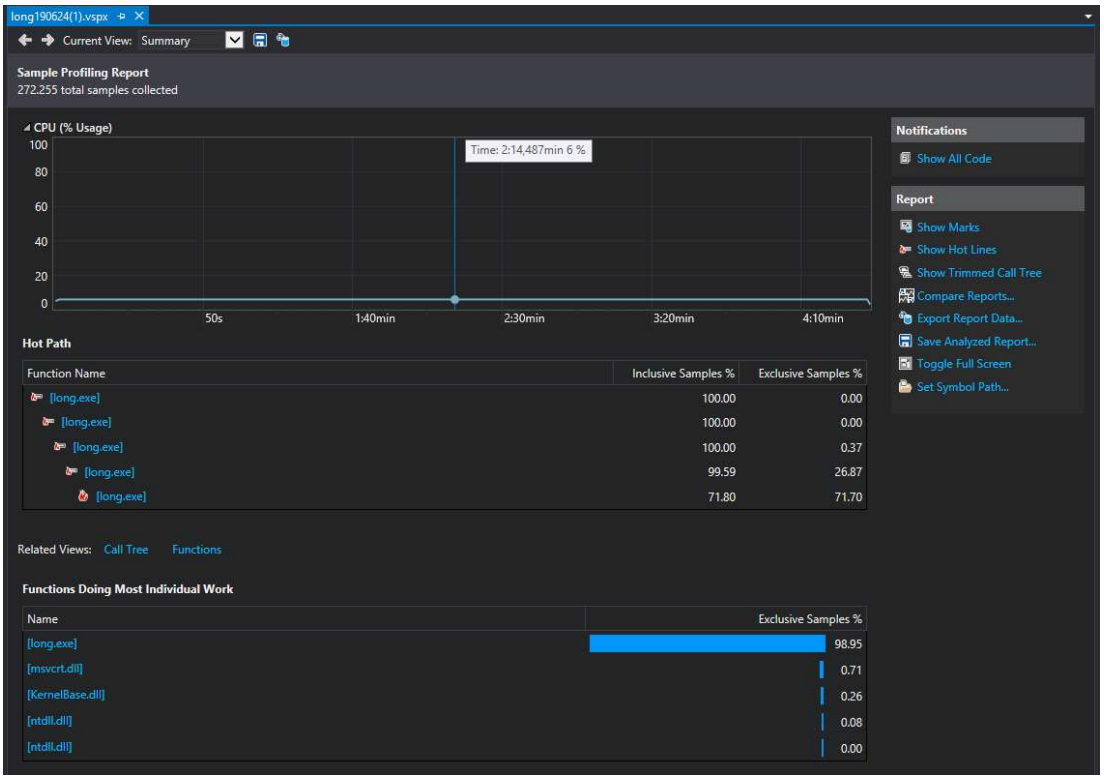
Το εκτελέσιμο πρόγραμμα του αλγορίθμου μετρήθηκε ξεχωριστά από την εφαρμογή και στις τρεις διαφορετικές εκδοχές του, την sprint, την medium και την long. Τα αποτελέσματα εμφανίζονται κατά σειρά στις παρακάτω εικόνες:



Εικόνα 96: Αποτελέσματα χρήσης CPU από την εκδοχή sprint του αλγορίθμου



Εικόνα 97: Αποτελέσματα χρήσης CPU από την εκδοχή medium του αλγορίθμου



Εικόνα 98: Αποτελέσματα χρήσης CPU από την εκδοχή long του αλγορίθμου

Στην Εικόνα 98, βλέπουμε τα αποτελέσματα του αλγορίθμου κατά την εκτέλεση της εκδοχής long. Λόγω του απαιτούμενου χρόνου (13 ώρες), μόλις ο αλγόριθμος ισοφάρισε την απόδοσή του σε αυτή την εκδοχή, σύμφωνα με τα δημοσιευμένα αποτελέσματα του 1ου διεθνούς διαγωνισμού για την αποδοτική επίλυση του προβλήματος του nurse rostering, η λειτουργία του διακόπηκε. Παρόλα αυτά, βλέπουμε ότι η χρήση του επεξεργαστή είναι σταθερή στο 6% όπως και στις άλλες δύο εκδοχές του αλγορίθμου. Υποθέτοντας ότι επιδεικνύει αυτή τη συμπεριφορά σε όλο το διάστημα της λειτουργίας του, μπορούμε με σαφήνεια να πούμε ότι δεν έχει μεγάλες απαιτήσεις σε υπολογιστική ισχύ ο αλγόριθμος επίλυσης του προβλήματος Nurse Rostering, ανεξαρτήτως της εκδοχής του την οποία θα κάνουμε χρήση. Η εκδοχή του αλγορίθμου, καθορίζει μόνο το χρόνο στον οποίο θα δημιουργηθεί το πρόγραμμα εργασίας.

Υπό την προϋπόθεση ύπαρξης περισσότερου χρόνου για την υλοποίηση της εφαρμογής ή σε μια μελλοντική επανέκδοσή της, μπορούν να γίνουν μερικές προτάσεις βελτίωσής της, είτε σε επίπεδο γραφικών είτε σε επίπεδο λειτουργιών.

- Η πρώτη πρόταση βελτίωσης που θα μπορούσε να γίνει, αφορά σε μερικές φόρμες, όπως αυτή των συμβολαίων ή των υπαλλήλων, έτσι ούτως ώστε να μπορεί η εφαρμογή να προβάλλει περισσότερο καλαίσθητα τα δεδομένα από τη βάση δεδομένων στη Γραμματεία για επεξεργασία.
- Η δεύτερη πρόταση βελτίωσης αφορά στη βάση δεδομένων και στο σχεδιασμό της. Θα μπορούσε να βελτιωθεί καλύτερα έτσι ώστε οι συσχετίσεις των πινάκων να υλοποιούνται από τη βάση απ' ευθείας και όχι από τμήματα κώδικα. Αυτό θα συμβάλλει στην απλούστευση του κώδικα αλλά και στην ευκολότερη αλλαγή της δομής – σχήματος της βάσης δεδομένων.
- Τρίτη πρόταση βελτίωσης είναι η μετάφραση των κώδικα του αλγορίθμου σε γλώσσα προγραμματισμού C# και πλήρη ενσωμάτωσή του στην εφαρμογή και όχι ως εξωτερική εφαρμογή που έχει επιλεγεί στην παρούσα Πτυχιακή Εργασία. Αυτό θα μπορούσε να επηρεάσει την απόδοση του αλγορίθμου, με χρήση πολυνηματικού προγραμματισμού και με παράλληλη χρήση των επεξεργαστών της κάρτας γραφικών να γίνει ταχύτερος, ειδικά στην κατάσταση λειτουργίας προγράμματος δημιουργίας long. Επιπλέον, ο αλγόριθμος θα είναι ενσωματωμένος και αναπόσπαστο κομμάτι της εφαρμογής, χωρίς να καθιστά αδύνατη τη λειτουργίας της από την απουσία του, όπως συμβαίνει με την επιλογή του ως εξωτερικό εκτελέσιμο.

- Τέταρτη και τελευταία πρόταση βελτίωσης είναι να δημιουργηθούν επιπλέον χειριστήρια για όλες τις παραμέτρους του αλγορίθμου, όπως για παράδειγμα της ευαισθησίας του ή της ανοχής του στην παγίδευση σε τοπικό βέλτιστο.

Η σχεδίαση, υλοποίηση και μέτρηση των επιδόσεων της εφαρμογής, έγινε σε υπολογιστή που αποτελείται από:

- CPU – AMD Ryzen 7 2700X Eight Core Processor 3,7GHz
- RAM – 16GB 3GHz
- Video Card – AMD Radeon R5 200 Series
- Storage device – Nvidia NVM 500GB
- OS – Windows 10 Pro x64

Βιβλιογραφία και αναφορές

AntOptima. (2018, 11 24). Ανάκτηση από AntOptima - We speed up your business: https://www.antoptima.com/site/en/index.php

Beligiannis, G., Tasopoulos, I., & Solos, I. (2013, 5 21). A Generic Two-Phase Stochastic Variable Neighborhood Approach for Effectively Solving the Nurse Rostering Problem. *Algorithms*, σσ. 278-308. Ανάκτηση από https://www.mdpi.com: https://www.mdpi.com/1999-4893/6/2/278/htm

Cooper, B. T., & Jeffrey, K. H. (2018, 11 23). *CiteSeerX.* Ανάκτηση από http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.84.222&rep=rep1&type=pdf: http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.84.222&rep=rep1&type=pdf

Entity Framework. (χ.χ.). Ανάκτηση από http://www.entityframeworktutorial.net

Flexi Roster. (χ.χ.). Ανάκτηση από Flexi Roster: http://www.flexiroster.com.au/

Kennedy, J., & Eberhart, R. (2001). *Swarm Intelligence.* Morgan Kaufmann.

Microsoft Visual Studio. (χ.χ.). Ανάκτηση από Microsoft Visual Studio: https://en.wikipedia.org/wiki/Microsoft_Visual_Studio

Mladenovic, N., & Hansen, P. (1997, 11). Variable Neighborhood Search. *Computers & Operations Research*, σσ. 1097-1100.

NR Competition. (2018, 11 20). *KU LEUVEN KULAK*. Ανάκτηση από https://www.kuleuven-kulak.be: https://www.kuleuven-kulak.be/nrpcompetition

Wikipedia - ACO. (2018, 11 24). Ανάκτηση από Wikipedia - The free encyclopedia: https://translate.google.gr/translate?sl=en&tl=el&js=y&prev=_t&hl=el&ie=UTF-8&u=https%3A%2F%2Fen.wikipedia.org%2Fwiki%2FAnt_colony_optimization_algorithms&edit-text=&act=url

Wikipedia - PSO. (2018, 11 24). Ανάκτηση από Wikipedia - The free encyclopedia: https://translate.google.gr/translate?sl=en&tl=el&js=y&prev=_t&hl=el&ie=UTF-8&u=https%3A%2F%2Fen.wikipedia.org%2Fwiki%2FParticle_swarm_optimization&edit-text=&act=url

Wikipedia - Tabu search. (2018, 11 24). Ανάκτηση από Wikipedia - The free encyclopedia: https://translate.google.gr/translate?sl=en&tl=el&js=y&prev=_t&hl=el&ie=UTF-8&u=https%3A%2F%2Fen.wikipedia.org%2Fwiki%2FTabu_search&edit-text=&act=url

Wikipedia - VNS. (2018, 11 28). Ανάκτηση από Wikipedia - The free encyclopedia: https://translate.google.com/translate?hl=el&sl=en&tl=el&u=https%3A%2F%2Fen.wikipedia.org%2Fwiki%2FVariable_neighborhood_search

Wikipedia - Γενετικοί αλγόριθμοι. (2018, 11 24). Ανάκτηση από Βικιπαίδεια - Η ελεύθερη εγκυκλοπαίδεια: https://el.wikipedia.org/wiki/Γενετικοί_Αλγόριθμοι

Wikipedia - Επιχειρησιακή έρευνα. (2018, 22 22). Ανάκτηση από Βικιπαίδεια - Η ελεύθερη εγκυκλοπαίδεια: https://el.wikipedia.org/wiki/%CE%95%CF%80%CE%B9%CF%87%CE%B5%CE%B9%CF%81%CE%B7%CF%83%CE%B9%CE%B1%CE%BA%CE%AE_%CE%AD%CF%81%CE%B5%CF%85%CE%BD%CE%B1

Βλαχάβας, Ι., Βασιλειάδης, Ν., Κόκκορας, Φ., Σακελλαρίου, Η., & Κεφάλας, Π. (2011). *Τεχνητή Νοημοσύνη.* Πανεπιστήμιο Μακεδονίας.

Πανεπιστήμιο Πάτρας. (2018, 10 8). Ανάκτηση από Πλατφόρμα Τηλεκπαίδευσης: https://eclass.upatras.gr/modules/document/file.php/DEAPT171/2016-2017/7_ΣΥΣΤΗΜΑΤΑ%20ΕΠΙΧΕΙΡΗΜΑΤΙΚΗΣ%20ΕΥΦΥΙΑΣ.pdf

Πρώτα η βάση δεδομένων. (χ.χ.). Ανάκτηση από https://www.ryadel.com/en/code-first-model-first-database-first-vs-comparison-orm-asp-net-core-entity-framework-ef-data/

Πρώτα ο κώδικας. (χ.χ.). Ανάκτηση από https://www.ryadel.com/en/code-first-model-first-database-first-vs-comparison-orm-asp-net-core-entity-framework-ef-data/

Πρώτα το μοντέλο. (χ.χ.). Ανάκτηση από https://www.ryadel.com/en/code-first-model-first-database-first-vs-comparison-orm-asp-net-core-entity-framework-ef-data/

Σώλος, Π. Ι. (2016). *Εθνικό Αρχείο Διδακτορικών Διατριβών.* Ανάκτηση από http://thesis.ekt.gr: http://thesis.ekt.gr/thesisBookReader/id/43894#page/16/mode/2up

Σώτος, Χ. (2015 - 2016). *Τ.Ε.Ι. Ηπείρου.* Ανάκτηση από Τ.Ε.Ι. Ηπείρου: http://apothetirio.teiep.gr/xmlui/bitstream/handle/123456789/5430/1423.pdf?sequence=1

Παράρτημα Α: Κώδικας του προγράμματος NurseRostering

ΠΑΚΕΤΟ IputData

ΚΛΑΣΗ Available_Pattern

```
using System;
using System.ComponentModel.DataAnnotations;

namespace NurseRostering.InputData
{
   [Serializable]
   public class Available_Pattern
   {
      [Key]
      public int ID { get; set; } // Αναγνωριστικό εγγραφής
      [Required]
      public string pattern { get; set; } // Πρότυπο

      // Κατασκευαστής
      public Available_Pattern() { }

      public override string ToString()
      {
         return pattern;
      }
   }
}
```

ΚΛΑΣΗ Available_Skill

```
using System;
using System.ComponentModel.DataAnnotations;

namespace NurseRostering.InputData
{
   [Serializable]
   public class Available_Skill
   {
      #region ΠΑΡΑΜΕΤΡΟΙ ΔΕΞΙΟΤΗΤΩΝ ΝΟΣΟΚΟΜΩΝ

      [Key]
      public int ID { get; set; } // Αναγνωριστικό εγγραφής
      [Required]
      public string de3iotexnia { get; set; } // Δεξιότητα υπαλλήλου

      #endregion
```

```
        // Κατασκευαστής
        public Available_Skill() { }

        public override string ToString()
        {
            return de3iotexnia + ";";
        }
    }
}
```

ΚΛΑΣΗ Contract

```
using System;
using System.ComponentModel.DataAnnotations;

namespace NurseRostering.InputData
{
    [Serializable]
    public class Contract
    {
        Modules module = new Modules();
        private string komma = ", ";

        public Contract() { }

        #region ΠΑΡΑΜΕΤΡΟΙ ΣΥΜΒΟΛΑΙΟΥ ΝΟΣΟΚΟΜΩΝ

        /* Οι παρακάτω παράμετροι αποτελούν τους όρους εργασίας ενός νοσοκόμου οι
οποίοι περιλαμβάνονται
         * στο συμβόλαιο που συνάπτει με το νοσοκομειακό ίδρυμα.
         * Θέτοντας σε ένα διακόπτη (μεταβλητή με πρόθεμα onOff) την τιμή 0, τότε η
σχετική μεταβλητή
         * δεν λαμβάνεται υπόψη από τον αλγόριθμο, κατά τη δημιουργία του
προγράμματος, ανεξαρτήτως την τιμή ή
         * το βάρος της μεταβλητής. Αν θέσουμε την τιμή 1, τότε η μεταβλητή
λαμβάνεται υπόψη από τον
         * αλγόριθμό κατά τα οριζόμενα από την άλλη παράμετρο ή τις άλλες
παραμέτρους της μεταβλητής. */
        [Key]
        public int ID { get; set; } = -1; // Μοναδικό αναγνωριστικό συμβολαίου
        [Required]
        public string description { get; set; } = ""; // Περιγραφή είδους συμβολαίου
        [Required]
        public int switch_singleAssignmentPerDay { get; set; } = 0; // Μια βάρδια ανά
ημέρα - Διακόπτης On-Off - 1=>On 0=>Off
        [Required]
        public int weight_singleAssignmentPerDay { get; set; } = 0; // Μια βάρδια ανά
ημέρα - Βαρύτητα μεταβλητής - Τιμές 0 - 20
        [Required]
```

```
        public int switch_maxNumAssignments { get; set; } = 0; // Μέγιστο πλήθος
βαρδιών ενός νοσοκόμου - Διακόπτης On-Off - 1=>On 0=>Off
        [Required]
        public int weight_maxNumAssignments { get; set; } = 0; // Μέγιστο πλήθος
βαρδιών ενός νοσοκόμου - Βαρύτητα μεταβλητής - Τιμές 0 - 20
        [Required]
        public int value_maxNumAssignments { get; set; } = 0; // Μέγιστο πλήθος
βαρδιών ενός νοσοκόμου - Τιμή μεταβλητής
        [Required]
        public int switch_minNumAssignments { get; set; } = 0; // Ελάχιστο πλήθος
βαρδιών ενός νοσοκόμου - Διακόπτης On-Off - 1=>On 0=>Off
        [Required]
        public int weight_minNumAssignments { get; set; } = 0; // Ελάχιστο πλήθος
βαρδιών ενός νοσοκόμου - Βαρύτητα μεταβλητής - Τιμές 0 - 20
        [Required]
        public int value_minNumAssignments { get; set; } = 0; // Ελάχιστο πλήθος
βαρδιών ενός νοσοκόμου - Τιμή μεταβλητής
        [Required]
        public int switch_maxConsecutiveWorkingDays { get; set; } = 0; // Μέγιστο
πλήθος συνεχόμενων ημερών εργασίας - Διακόπτης On-Off - 1=>On 0=>Off
        [Required]
        public int weight_maxConsecutiveWorkingDays { get; set; } = 0; // Μέγιστο
πλήθος συνεχόμενων ημερών εργασίας - Βαρύτητα μεταβλητής - Τιμές 0 - 20
        [Required]
        public int value_maxConsecutiveWorkingDays { get; set; } = 0; // Μέγιστο
πλήθος συνεχόμενων ημερών εργασίας - Τιμή μεταβλητής
        [Required]
        public int switch_minConsecutiveWorkingDays { get; set; } = 0; // Ελάχιστο
πλήθος συνεχόμενων ημερών εργασίας - Διακόπτης On-Off - 1=>On 0=>Off
        [Required]
        public int weight_minConsecutiveWorkingDays { get; set; } = 0; // Ελάχιστο
πλήθος συνεχόμενων ημερών εργασίας - Βαρύτητα μεταβλητής - Τιμές 0 - 20
        [Required]
        public int value_minConsecutiveWorkingDays { get; set; } = 0; // Ελάχιστο
πλήθος συνεχόμενων ημερών εργασίας - Τιμή μεταβλητής
        [Required]
        public int switch_maxConsecutiveFreeDays { get; set; } = 0; // Μέγιστο πλήθος
συνεχόμενων ημερών ανάπαυσης - Διακόπτης On-Off - 1=>On 0=>Off
        [Required]
        public int weight_maxConsecutiveFreeDays { get; set; } = 0; // Μέγιστο πλήθος
συνεχόμενων ημερών ανάπαυσης - Βαρύτητα μεταβλητής - Τιμές 0 - 20
        [Required]
        public int value_maxConsecutiveFreeDays { get; set; } = 0; // Μέγιστο πλήθος
συνεχόμενων ημερών ανάπαυσης - Τιμή μεταβλητής
        [Required]
        public int switch_minConsecutiveFreeDays { get; set; } = 0; // Ελάχιστο πλήθος
συνεχόμενων ημερών ανάπαυσης - Διακόπτης On-Off - 1=>On 0=>Off
        [Required]
        public int weight_minConsecutiveFreeDays { get; set; } = 0; // Ελάχιστο πλήθος
συνεχόμενων ημερών ανάπαυσης - Βαρύτητα μεταβλητής - Τιμές 0 - 20
        [Required]
        public int value_minConsecutiveFreeDays { get; set; } = 0; // Ελάχιστο πλήθος
συνεχόμενων ημερών ανάπαυσης - Τιμή μεταβλητής
```

```
        [Required]
        public int switch_maxConsecutiveWorkingWeekends { get; set; } = 0; // Μέγιστο πλήθος συνεχόμενων Σαββατοκύριακων εργασίας - Διακόπτης On-Off - 1=>On 0=>Off
        [Required]
        public int weight_maxConsecutiveWorkingWeekends { get; set; } = 0; // Μέγιστο πλήθος συνεχόμενων Σαββατοκύριακων εργασίας - Βαρύτητα μεταβλητής - Τιμές 0 - 20
        [Required]
        public int value_maxConsecutiveWorkingWeekends { get; set; } = 0; // Μέγιστο πλήθος συνεχόμενων Σαββατοκύριακων εργασίας - Τιμή μεταβλητής
        [Required]
        public int switch_minConsecutiveWorkingWeekends { get; set; } = 0; // Ελάχιστο πλήθος συνεχόμενων Σαββατοκύριακων εργασίας - Διακόπτης On-Off - 1=>On 0=>Off
        [Required]
        public int weight_minConsecutiveWorkingWeekends { get; set; } = 0; // Ελάχιστο πλήθος συνεχόμενων Σαββατοκύριακων εργασίας - Βαρύτητα μεταβλητής - Τιμές 0 - 20
        [Required]
        public int value_minConsecutiveWorkingWeekends { get; set; } = 0; // Ελάχιστο πλήθος συνεχόμενων Σαββατοκύριακων εργασίας - Τιμή μεταβλητής
        [Required]
        public int switch_maxWorkingWeekendsInFourWeeks { get; set; } = 0; // Μέγιστο πλήθος Σαββατοκύριακων εργασίας σε 4 εβδομάδες - Διακόπτης On-Off - 1=>On 0=>Off
        [Required]
        public int weight_maxWorkingWeekendsInFourWeeks { get; set; } = 0; // Μέγιστο πλήθος Σαββατοκύριακων εργασίας σε 4 εβδομάδες - Βαρύτητα μεταβλητής - Τιμές 0 - 20
        [Required]
        public int value_maxWorkingWeekendsInFourWeeks { get; set; } = 0; // Μέγιστο πλήθος Σαββατοκύριακων εργασίας σε 4 εβδομάδες - Τιμή μεταβλητής
        [Required]
        public int switch_completeWeekends { get; set; } = 0; // Ολοκληρωμένα Σαββατοκύριακα - Διακόπτης On-Off - 1=>On 0=>Off
        [Required]
        public int weight_completeWeekends { get; set; } = 0; // Ολοκληρωμένα Σαββατοκύριακα - Βαρύτητα μεταβλητής - Τιμές 0 - 20
        [Required]
        public int switch_identicalShiftTypesDuringWeekend { get; set; } = 0; // Σταθερό είδος βάρδιας όλο το Σαββατοκύριακο - Διακόπτης On-Off - 1=>On 0=>Off
        [Required]
        public int weight_identicalShiftTypesDuringWeekend { get; set; } = 0; // Σταθερό είδος βάρδιας όλο το Σαββατοκύριακο - Βαρύτητα μεταβλητής - Τιμές 0 - 20
        [Required]
        public int switch_noNightShiftBeforeFreeWeekend { get; set; } = 0; // Όχι νυχτερινή βάρδια πριν από Σαββατοκύριακο ανάπαυσης - Διακόπτης On-Off - 1=>On 0=>Off
        [Required]
        public int weight_noNightShiftBeforeFreeWeekend { get; set; } = 0; // Όχι νυχτερινή βάρδια πριν από Σαββατοκύριακο ανάπαυσης - Βαρύτητα μεταβλητής - Τιμές 0 - 20
```

```
        [Required]
        public int switch_twoFreeDaysAfterNightShifts { get; set; } = 0; // Δύο ημέρες
ανάπαυσης έπειτα από νυχτερινή βάρδια - Διακόπτης On-Off - 1=>On 0=>Off
        [Required]
        public int weight_twoFreeDaysAfterNightShifts { get; set; } = 0; // Δύο ημέρες
ανάπαυσης έπειτα από νυχτερινή βάρδια - Βαρύτητα μεταβλητής - Τιμές 0 - 20
        [Required]
        public int switch_alternativeSkillCategory { get; set; } = 0; // Ελάχιστες
απαιτούμενες δεξιότητες νοσοκόμου - Διακόπτης On-Off - 1=>On 0=>Off
        [Required]
        public int weight_alternativeSkillCategory { get; set; } = 0; // Ελάχιστες
απαιτούμενες δεξιότητες νοσοκόμου - Βαρύτητα μεταβλητής - Τιμές 0 - 20
        [Required]
        public string weekendDefinition { get; set; } = "SaturdaySunday"; //
Σαββατοκύριακα
        [Required]
        public int numberOfUnwantedPatterns { get; set; } = 0; // Πλήθος ανεπιθύμητων
συνδυασμών
        public string UnwantedPatterns { get; set; } // Λίστα ανεπιθύμητων προτύπων
εργασίας

        #endregion

        /* Υπερκάλυψη της μεθόδου ToString() για κατάλληλα διαμορφωμένη έξοδο
         * για εγγραφή των περιεχομένων του συμβολαίου στο αρχείο εισόδου του
αλγορίθμου */
        public override string ToString()
        {
            return ID
                + komma + description
                + komma + module.FormatTwoData(switch_singleAssignmentPerDay,
weight_singleAssignmentPerDay)
                + komma + module.FormatThreeData(switch_maxNumAssignments,
weight_maxNumAssignments, value_maxNumAssignments)
                + komma + module.FormatThreeData(switch_minNumAssignments,
weight_minNumAssignments, value_minNumAssignments)
                + komma +
module.FormatThreeData(switch_maxConsecutiveWorkingDays,
weight_maxConsecutiveWorkingDays, value_maxConsecutiveWorkingDays)
                + komma +
module.FormatThreeData(switch_minConsecutiveWorkingDays,
weight_minConsecutiveWorkingDays, value_minConsecutiveWorkingDays)
                + komma + module.FormatThreeData(switch_maxConsecutiveFreeDays,
weight_maxConsecutiveFreeDays, value_maxConsecutiveFreeDays)
                + komma + module.FormatThreeData(switch_minConsecutiveFreeDays,
weight_minConsecutiveFreeDays, value_minConsecutiveFreeDays)
                + komma +
module.FormatThreeData(switch_maxConsecutiveWorkingWeekends,
weight_maxConsecutiveWorkingWeekends,
value_maxConsecutiveWorkingWeekends)
                + komma +
module.FormatThreeData(switch_minConsecutiveWorkingWeekends,
```

```
weight_minConsecutiveWorkingWeekends,
value_minConsecutiveWorkingWeekends)
            + komma +
module.FormatThreeData(switch_maxWorkingWeekendsInFourWeeks,
weight_maxWorkingWeekendsInFourWeeks,
value_maxWorkingWeekendsInFourWeeks)
            + komma + weekendDefinition
            + komma + module.FormatTwoData(switch_completeWeekends,
weight_completeWeekends)
            + komma +
module.FormatTwoData(switch_identicalShiftTypesDuringWeekend,
weight_identicalShiftTypesDuringWeekend)
            + komma +
module.FormatTwoData(switch_noNightShiftBeforeFreeWeekend,
weight_noNightShiftBeforeFreeWeekend)
            + komma + module.FormatTwoData(switch_twoFreeDaysAfterNightShifts,
weight_twoFreeDaysAfterNightShifts)
            + komma + module.FormatTwoData(switch_alternativeSkillCategory,
weight_alternativeSkillCategory)
            + komma + numberOfUnwantedPatterns
            + komma + UnwantedPatterns + ";";
        }
    }
}
```

ΚΛΑΣΗ Date_Specific_Cover

```
using System;
using System.ComponentModel.DataAnnotations;

namespace NurseRostering.InputData
{
    [Serializable]
    public class Date_Specific_Cover
    {
        private string komma = ", ";

        #region ΠΑΡΑΜΕΤΡΟΙ ΕΡΓΑΣΙΑΣ ΓΙΑ ΣΥΓΚΕΚΡΙΜΜΕΝΗ ΜΕΡΑ ΤΗΣ
ΕΒΔΟΜΑΔΑΣ

        [Key]
        public int ID { get; set; } = -1; // Αναγνωριστικό εγγραφής
        [Required]
        public DateTime weekDay { get; set; } // Ημερομηνία ειδικής κάλυψης
        [Required]
        public string shift { get; set; } = ""; // Βάρδια
        [Required]
        public int preferred { get; set; } = 1; // Προτίμηση

        #endregion
```

```
        // Κατασκευαστής
        public Date_Specific_Cover()
        {
        }
        public override string ToString()
        {
            return weekDay.ToString("yyyy-MM-dd") + komma + shift + komma +
preferred + ";";
        }
    }
}
```

ΚΛΑΣΗ Day_Of_Week_Cover

```
using System;
using System.ComponentModel.DataAnnotations;

namespace NurseRostering.InputData
{
    [Serializable]
    public class Day_Of_Week_Cover
    {
        private string komma = ", ";

        #region ΠΑΡΑΜΕΤΡΟΙ ΕΡΓΑΣΙΑΣ ΓΙΑ ΚΑΘΕ ΜΕΡΑ ΤΗΣ ΕΒΔΟΜΑΔΑΣ

        [Key]
        public int ID { get; set; } // Αναγνωριστικό εγγραφής
        [Required]
        public string weekDay { get; set; } = ""; // Ημέρα της εβδομάδας
        [Required]
        public string shift { get; set; } = ""; // Βάρδια
        [Required]
        public int preferred { get; set; } = 1; // Προτίμηση

        #endregion

        // Κατασκευαστής
        public Day_Of_Week_Cover()
        {
        }

        public override string ToString()
        {
            return weekDay + komma + shift + komma + preferred + ";";
        }
    }
}
```

ΚΛΑΣΗ Day_Off_Request

```
using System;
using System.ComponentModel.DataAnnotations;

namespace NurseRostering.InputData
{
   [Serializable]
   public class Day_Off_Request
   {
      protected string komma = ", ";

      #region ΠΑΡΑΜΕΤΡΟΙ ΑΙΤΟΥΜΕΝΩΝ ΗΜΕΡΩΝ ΑΠΟ ΤΟ ΝΟΣΟΚΟΜΟ
ΓΙΑ ΑΝΑΠΑΥΣΗ

      [Key]
      public int ID { get; set; } = -1; // Ανανγωριστικό εγγραφής
      [Required]
      public DateTime date { get; set; }  // Επιλεγμένη ημερομηνία από το νοσοκόμο
για ανάπαυση
      [Required]
      public string weight { get; set; } = ""; /* Βαρύτητα του κριτηρίου κατά τη
δημιουργία του προγράμματος εργασίας.
                        * Αυτό σημαίνει ότι πιο μεγάλος είναι ο αριθμός που δίδεται στην
μεταβλητή
                        * τόσο πιο πολύ θα λαμβάνεται υπόψη αυτή η παράμετρος από τον
αλγόριθμο
                        * κατά τη δημιουργία του προγράμματος εργασίας. Η τιμή αυτή δεν
μπορεί
                        * να λαμβάνει αρνητικές τιμές */
      [Required]
      public int employeeID { get; set; }

      #endregion


      // Κατασκευαστής
      public Day_Off_Request()
      {
      }

      public override string ToString()
      {
         return employeeID + komma + date.ToString("yyyy-MM-dd") + komma +
weight + ";";
      }
   }
}
```

ΚΛΑΣΗ Day_On_Request

```
using System;
using System.ComponentModel.DataAnnotations;

namespace NurseRostering.InputData
{
   [Serializable]
   public class Day_On_Request
   {
      protected string komma = ", ";

      #region ΠΑΡΑΜΕΤΡΟΙ ΑΙΤΟΥΜΕΝΩΝ ΗΜΕΡΩΝ ΑΠΟ ΤΟ ΝΟΣΟΚΟΜΟ
ΓΙΑ ΕΡΓΑΣΙΑ

      [Key]
      public int ID { get; set; } // Ανανγωριστικό εγγραφής
      [Required]
      public DateTime date { get; set; } // Επιλεγμένη ημερομηνία από το νοσοκόμο
για εργασία
      [Required]
      public string weight { get; set; } /* Βαρύτητα του κριτηρίου κατά τη δημιουργία
του προγράμματος εργασίας.
                        * Αυτό σημαίνει ότι πιο μεγάλος είναι ο αριθμός που δίδεται στην
μεταβλητή
                        * τόσο πιο πολύ θα λαμβάνεται υπόψη αυτή η παράμετρος από τον
αλγόριθμο
                        * κατά τη δημιουργία του προγράμματος εργασίας. Η τιμή αυτή δεν
μπορεί
                        * να λαμβάνει αρνητικές τιμές */
      [Required]
      public int employeeID { get; set; }

      #endregion


      // Κατασκευαστής
      public Day_On_Request()
      {
      }

      public override string ToString()
      {
         return employeeID + komma + date.ToString("yyyy-MM-dd") + komma +
weight + ";";
      }
   }
}
```

ΚΛΑΣΗ Employee

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.ComponentModel.DataAnnotations;
using NurseRostering.Database;

namespace NurseRostering.InputData
{
   [Serializable]
   public class Employee
   {
      NurseRosteringDB nrdb = new NurseRosteringDB();

      private string komma = ", ";

      #region ΠΑΡΑΜΕΤΡΟΙ ΣΧΕΤΙΚΑ ΜΕ ΤΑ ΣΤΟΙΧΕΙΑ ΤΟΥ ΝΟΣΟΚΟΜΟΥ

      [Key]
      public int ID { get; set; } = -1; // Αναγνωριστικό νοσοκόμου
      [Required]
      public string name { get; set; } = ""; // Όνοματεπώνυμο νοσοκόμου
      [Required]
      public int contractID { get; set; } // Αναγνωριστικό συμβολαίου
      [Required]
      public string amka { get; set; } = ""; // ΑΜΚΑ νοσοκόμου
      [Required]
      public string afm { get; set; } = ""; // ΑΦΜ νοσοκόμου
      [Required]
      public string dieu8unsi { get; set; } = ""; // Διεύθυνση οικίας νοσοκόμου
      [Required]
      public string poli { get; set; } = ""; // Πόλη διαμονής νοσοκόμου
      [Required]
      public string tk { get; set; } = ""; // Ταχυδρομικός κωδικός
      [Required]
      public string kinito { get; set; } = ""; // Αριθμός κινητού τηλεφώνου νοσοκόμου
      public string tilefwno { get; set; } = ""; // Αριθμός τηλεφώνου οικίας νοσοκόμου
      [Required]
      public string email { get; set; } = ""; // Διεύθυνση e-mail νοσοκόμου
      [Required]
      public int numberOfSkills { get; set; } = 0; // Πλήθος δεξιοτεχνιών νοσοκόμου
      public byte[] photo { get; set; } // Φωτογραφία νοσοκόμου

      // Επιθυμητές ημέρες εργασίας του νοσοκόμου
      public List<Day_On_Request> Empl_Des_Days_On { get; set; }

      // Ανεπιθύμητες ημέρες εργασίας του νοσοκόμου
      public List<Day_Off_Request> Empl_Des_Days_Off { get; set; }

      // Εργασία του νοσοκόμου σε συγκεκριμένη ημέρα και βάρδια
```

```
        public List<Shift_On_Request> Empl_Des_Shifts_On { get; set; }

        // Συγκεκριμένη ημέρα και βάρδια που ο νοσοκόμος ΔΕΝ επιθυμεί να εργαστεί
        public List<Shift_Off_Request> Empl_Des_Shifts_Off { get; set; }


        #endregion

        // Κατασκευαστής
        public Employee()
        {
            // Δημιουργία λιστών αιτημάτων εργασίας και ανάπαυσης του νοσοκόμου
            Empl_Des_Days_On = new List<Day_On_Request>();
            Empl_Des_Days_Off = new List<Day_Off_Request>();
            Empl_Des_Shifts_On = new List<Shift_On_Request>();
            Empl_Des_Shifts_Off = new List<Shift_Off_Request>();
        }

        public string DataForInputFile()
        {
            // Λίστα στην οποία θα φορτωθούν οι ειδικότητες κάθε νοσοκόμου
            List<Skill> EmployeeSkill = new List<Skill>();

            string EmpSkills = ""; // ΜΕταβλητή εξόδου ειδικοτήτων
            int empAfm = int.Parse(this.afm); // Μετατροπή κειμένου σε αριθμό (ΑΦΜ ως
αναγνωριστικό υπαλλήλου)
            EmployeeSkill.Clear(); // Καθαρισμός λίστας για αποφυγή τυχόν διπλών
εγγραφών

            // Λήψη ειδικοτήτων κάθε νοσοκόμου από τη βάση δεδομένων
            EmployeeSkill = (from x in nrdb.EmployeeSkills select x).Where(x =>
x.employeeID == empAfm).ToList();

            // Για κάθε εγγραφή που βρέθηκε
            foreach(var item in EmployeeSkill)
            {
                // διαμόρφωση σύμφωνα με το πρότυπο που απαιτείται στο αρχείο εισόδου
του αλγορίθμου
                EmpSkills += item.de3iotexnia + " ";
            }

            // Έξοδος εγγραφής νοσοκόμου με όλες τις ειδικότητες κατάλληλα
διαμορφωμένη για εγγραφή στο αρχείο εισόδου
            return ID.ToString() + komma + ID.ToString() + komma + contractID +
komma + numberOfSkills + komma + EmpSkills.Trim() + ";";
        }
        public override string ToString()
        {
            return ID.ToString() + komma + name + komma + contractID + komma +
amka
                + komma + afm + komma + dieu8unsi + komma + poli + komma + tk
                + komma + kinito + komma + tilefwno + komma + email + komma +
numberOfSkills;
```

```
        }
    }
}
```

ΚΛΑΣΗ Enotita

```
using System;

namespace NurseRostering.InputData
{
    /* Με την παρούσα κλάση το μόνο που υλοποιείται είναι ο καθορισμός του
ονόματος των κεφαλαίων
     * στο αρχείο εισόδου. Το κάθε κεφάλαιο αφορά σε ένα αντικείμενο με τις ιδιότητές
ου και το
     * πως εγγράφονται μέσα στο αρχείο εισόδου. Το κάθε αντικείμενο καταχωρεί τις
δικές του εγγραφές
     * ενώ η παρούσα κλάση καταχωρεί το όνομα του κεφαλαίου μαζί με τα
απαραίτητα διαχωριστικά. */
    [Serializable]
    public class Enotita
    {
        #region ΠΑΡΑΜΕΤΡΟΙ ΕΝΟΤΗΤΑΣ

        public string OnomaEnotitas { get; set; } = ""; // Όνομα κεφαλαίου που θα
εγγραφεί στο αρχείο
        public int Pli8osEggrafwn { get; set; } = 0; // Πλήθος εγγραφών που περιέχει το
κεφάλαιο

        // Οριοθέτης πινακίδας τίτλου κεφαλαίου
        private string Diaxwristiko = "////////////////////////////////////////////////////////////////////////";

        #endregion

        // Κατασκευαστές
        public Enotita() { }

        public Enotita(string OnomaEnotitas)
        {
            this.OnomaEnotitas = OnomaEnotitas.ToUpper();
        }

        public Enotita(string OnomaEnotitas, int Pli8osEggrafwn)
        {
            this.OnomaEnotitas = OnomaEnotitas.ToUpper();
            this.Pli8osEggrafwn = Pli8osEggrafwn;
        }

        public void ChapterData(string OnomaEnotitas, int Pli8osEggrafwn)
        {
```

```
            this.OnomaEnotitas = OnomaEnotitas.ToUpper();
            this.Pli8osEggrafwn = Pli8osEggrafwn;
        }

        // Επιστροφή στον χρήστη, ολοκληρωμένου του τίτλου κεφαλαίου
        public override string ToString()
        {
            // Εάν ο τίτλος κεφαλαίου αφορά στη χρονική περίοδο προγραμματισμού
            if (OnomaEnotitas == "SCHEDULING_PERIOD")
            {
                return this.Diaxwristiko + "\r\n" + "SCHEDULING_PERIOD;\r\n" +
this.Diaxwristiko + "\r\n";
            }

            // Εάν ο τίτλος κεφαλαίου αφορά στους υπάλληλους
            if (OnomaEnotitas == "EMPLOYEES")
            {
                return this.Diaxwristiko + "\r\n" + "EMPLOYEES " + this.Pli8osEggrafwn
+ ";\r\n" + this.Diaxwristiko + "\r\n";
            }

            /* Εάν ο τίτλος κεφαλαίου αφορά σε οποιοδήποτε άλλο κεφάλαιο του αρχείου
εισόδου, τότε
             * εγγράφεται το κεφάλαιο με το σχετικό πλήθος εγγραφών μέσα στο κεφάλαιο
αυτό, οπωσδήποτε
             * με κεφαλαία γράμματα. */
            return this.Diaxwristiko + "\r\n" + this.OnomaEnotitas.ToUpper() + " = " +
Pli8osEggrafwn + ";\r\n" + this.Diaxwristiko + "\r\n";

        }
    }
}
```

ΚΛΑΣΗ Pattern

```
using System;
using System.ComponentModel.DataAnnotations;

namespace NurseRostering.InputData
{
    [Serializable]
    public class Pattern
    {
        // Αντικείμενα
        Modules module = new Modules();

        //Μεταβλητές
        private string komma = ", ";
```

```
        #region ΠΑΡΑΜΕΤΡΟΙ ΠΡΟΤΥΠΩΝ (ΣΥΝΔΥΑΣΜΟΙ ΗΜΕΡΩΝ) ΕΡΓΑΣΙΑΣ

        [Key]
        public int ID { get; set; } // Αναγνωριστικό πρότυπου
        [Required]
        public int weight { get; set; } = 1; /* Βαρύτητα του κριτηρίου κατά τη δημιουργία
του προγράμματος εργασίας.
                        * Αυτό σημαίνει ότι πιο μεγάλος είναι ο αριθμός που δίδεται στην
μεταβλητή
                        * τόσο πιο πολύ θα λαμβάνεται υπόψη αυτή η παράμετρος από τον
αλγόριθμο
                        * κατά τη δημιουργία του προγράμματος εργασίας. Η τιμή αυτή δεν
μπορεί
                        * να λαμβάνει αρνητικές τιμές */
        [Required]
        public int NumberOfShiftTypes { get; set; } = 0;// Πλήθος εγγραφών των
ανεπιθύμητων προτύπων εργασίας
        [Required]
        public string ShiftType { get; set; } // Περιγραφή ανεπιθύμητου προτύπου
εργασίας

        #endregion

        // Κατασκευαστής
        public Pattern() { }

        public override string ToString()
        {
            return ID + komma + weight + komma + NumberOfShiftTypes + " " +
ShiftType + ";";
        }
    }
}
```

ΚΛΑΣΗ Scheduling_Period

```
using System;
using System.ComponentModel.DataAnnotations;

namespace NurseRostering.InputData
{
    [Serializable]
    public class Scheduling_Period
    {
        private string komma = ", ";

        #region ΠΑΡΑΜΕΤΡΟΙ ΤΗΣ ΠΕΡΙΟΔΟΥ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ ΕΡΓΑΣΙΑΣ

        [Key]
```

```
        public int ID { get; set; } // Αναγνωριστικό προγράμματος εργασίας
συγκεκριμένης περιόδου
        [Required]
        public string kindOfProgram { get; set; } = ""; // Είδος προγράμματος
        [Required]
        public DateTime startDate { get; set; } // Ημερομηνία έναρξης προγράμματος
        [Required]
        public DateTime endDate { get; set; } // Ημερομηνία λήξης προγράμματος

        #endregion


        // Κατασκευαστής
        public Scheduling_Period()
        {
        }

        public override string ToString()
        {
            return kindOfProgram + komma + startDate.ToString("yyyy-MM-dd") +
komma + endDate.ToString("yyyy-MM-dd") + ";";
        }
    }
}
```

ΚΛΑΣΗ Shift_Off_Request

```
using System;
using System.ComponentModel.DataAnnotations;

namespace NurseRostering.InputData
{
    [Serializable]
    public class Shift_Off_Request
    {
        private string komma = ", ";

        #region ΠΑΡΑΜΕΤΡΟΙ ΑΙΤΟΥΜΕΝΩΝ ΗΜΕΡΩΝ ΚΑΙ ΒΑΡΔΙΩΝ ΑΠΟ ΤΟ
ΝΟΣΟΚΟΜΟ ΓΙΑ ΑΝΑΠΑΥΣΗ

        [Key]
        public int ID { get; set; } = -1; // Ανανγωριστικό εγγραφής
        [Required]
        public DateTime date { get; set; } // Επιλεγμένη ημερομηνία από το νοσοκόμο
για ανάπαυση
        [Required]
        public string weight { get; set; } = ""; /* Βαρύτητα του κριτηρίου κατά τη
δημιουργία του προγράμματος εργασίας.
```

```
                    * Αυτό σημαίνει ότι πιο μεγάλος είναι ο αριθμός που δίδεται στην
μεταβλητή
                    * τόσο πιο πολύ θα λαμβάνεται υπόψη αυτή η παράμετρος από τον
αλγόριθμο
                    * κατά τη δημιουργία του προγράμματος εργασίας. Η τιμή αυτή δεν
μπορεί
                    * να λαμβάνει αρνητικές τιμές */

        [Required]
        public string ShiftTypeID { get; set; } = ""; // Αναγνωριστικό τύπου βάρδιας
        [Required]
        public int employeeID { get; set; }

        #endregion


        // Κατασκευαστής
        public Shift_Off_Request()
        {
        }

        public override string ToString()
        {
            return employeeID + komma + date.ToString("yyyy-MM-dd") + komma +
ShiftTypeID + komma + weight + ";";
        }
    }
}
```

ΚΛΑΣΗ Shift_On_Request

```
using System;
using System.ComponentModel.DataAnnotations;

namespace NurseRostering.InputData
{
    [Serializable]
    public class Shift_On_Request
    {
        private string komma = ", ";

        #region ΠΑΡΑΜΕΤΡΟΙ ΑΙΤΟΥΜΕΝΩΝ ΗΜΕΡΩΝ ΚΑΙ ΒΑΡΔΙΩΝ ΑΠΟ ΤΟ
ΝΟΣΟΚΟΜΟ ΓΙΑ ΕΡΓΑΣΙΑ

        [Key]
        public int ID { get; set; } = -1; // Ανανγωριστικό εγγραφής
        [Required]
        public DateTime date { get; set; } // Επιλεγμένη ημερομηνία από το νοσοκόμο
για εργασία
```

```
        [Required]
        public string weight { get; set; } = ""; /* Βαρύτητα του κριτηρίου κατά τη
δημιουργία του προγράμματος εργασίας.
                        * Αυτό σημαίνει ότι πιο μεγάλος είναι ο αριθμός που δίδεται στην
μεταβλητή
                        * τόσο πιο πολύ θα λαμβάνεται υπόψη αυτή η παράμετρος από τον
αλγόριθμο
                        * κατά τη δημιουργία του προγράμματος εργασίας. Η τιμή αυτή δεν
μπορεί
                        * να λαμβάνει αρνητικές τιμές */

        [Required]
        public string ShiftTypeID { get; set; } = ""; // Αναγνωριστικό τύπου βάρδιας
        [Required]
        public int employeeID { get; set; }

        #endregion


        // Κατασκευαστής
        public Shift_On_Request()
        {
        }

        public override string ToString()
        {
            return employeeID + komma + date.ToString("yyyy-MM-dd") + komma +
ShiftTypeID + komma + weight + ";";
        }
    }
}
```

ΚΛΑΣΗ Shift_Type

```
using System;
using System.ComponentModel.DataAnnotations;

namespace NurseRostering.InputData
{
    [Serializable]
    public class Shift_Type
    {
        private string komma = ", ";

        #region ΠΑΡΑΜΕΤΡΟΙ ΤΩΝ ΤΥΠΩΝ ΤΩΝ ΒΑΡΔΙΩΝ

        [Key]
        public int ID { get; set; } // Αναγνωριστικό εγγραφής
        [Required]
```

```
        public string tipos { get; set; } = ""; // Αναγνωριστικό τύπου βάρδιας
        [Required]
        public string description { get; set; } = ""; // Περιγραφή τύπου βάρδιας
        [Required]
        public string startTime { get; set; } // Ώρα έναρξης της βάρδιας
        [Required]
        public string endTime { get; set; } // Ώρα λήξης της βάρδιας
        [Required]
        public int numberOfRequiredSkills { get; set; } = 0; // Απαιτούμενες ικανότητες
για τη βάρδια
        public string requiredSkills { get; set; } // Λίστα ικανοτήτων για κάθε τύπο
βάρδιας.

        #endregion

        // Κατασκευαστής
        public Shift_Type()
        {
        }

        public override string ToString()
        {
            return tipos + komma + description + komma + startTime + komma + endTime
+ komma + numberOfRequiredSkills + komma + requiredSkills + ";";
        }
    }
}
```

ΚΛΑΣΗ Skill

```
using System;
using System.ComponentModel.DataAnnotations;

namespace NurseRostering.InputData
{
    [Serializable]
    public class Skill
    {
        #region ΠΑΡΑΜΕΤΡΟΙ ΔΕΞΙΟΤΗΤΩΝ ΝΟΣΟΚΟΜΩΝ

        [Key]
        public int ID { get; set; } = -1; // Αναγνωριστικό εγγραφής
        [Required]
        public string de3iotexnia { get; set; } = ""; // Δεξιότητα υπαλλήλου
        [Required]
        public int employeeID { get; set; } = -1; // Αναγνωριστικό υπαλλήλου

        #endregion

        // Κατασκευαστής
```

```
        public Skill() { }

        public override string ToString()
        {
            return de3iotexnia;
        }
    }
}
```

ΠΑΚΕΤΟ OutputData

ΚΛΑΣΗ MyDbConfiguration

```
using System;
using System.ComponentModel.DataAnnotations;

namespace NurseRostering.OutputData
{
    public class Assignment
    {
        #region ΠΑΡΑΜΕΤΡΟΙ ΛΥΜΜΕΝΟΥ ΠΡΟΒΛΗΜΑΤΟΣ ΒΑΡΔΙΟΛΟΓΙΟΥ
ΝΟΣΟΚΟΜΩΝ

        [Key]
        public int ID { get; set; }
        [Required]
        public DateTime date { get; set; }
        [Required]
        public int employeeID { get; set; }
        [Required]
        public string shiftID { get; set; } = "R";

        #endregion

        //Κατασκευαστής
        public Assignment()
        {
        }
    }
}
```

ΚΛΑΣΗ MyDbConfiguration

```
namespace NurseRostering.OutputData
{
   public class Instance
   {
      #region ΠΑΡΑΜΕΤΡΟΙ ΣΤΙΓΜΙΟΤΥΠΟΥ ΛΥΣΗΣ

      public string problemInstance { get; set; } // Αριθμός λύσης του προβλήματος
      public string competitor { get; set; }
      public int softConstraintPenalty { get; set; } // Ποινές παραβίασης ελαστικών
περιορισμών

      #endregion

      // Κατασκευαστής
      public Instance()
      {
      }
   }
}
```

ΠΑΚΕΤΟ Database

ΚΛΑΣΗ NurseRosteringDB

```
using System.Data.Entity;
using NurseRostering.InputData;

namespace NurseRostering.Database
{
   public class NurseRosteringDB : DbContext
   {
      public NurseRosteringDB() { }

      public DbSet<Available_Pattern> AvailablePatterns { get; set; }
      public DbSet<Contract> Contracts { get; set; }
      public DbSet<Date_Specific_Cover> Date_Specific_Covers { get; set; }
      public DbSet<Day_Of_Week_Cover> Day_Of_Week_Covers { get; set; }
      public DbSet<Day_On_Request> Day_On_Requests { get; set; }
      public DbSet<Day_Off_Request> Day_Off_Requests { get; set; }
      public DbSet<Employee> Employees { get; set; }
      public DbSet<Available_Skill> Available_Skills { get; set; }
      public DbSet<Pattern> Patterns { get; set; }
      public DbSet<Scheduling_Period> Scheduling_Periods { get; set; }
      public DbSet<Shift_On_Request> Shift_On_Requests { get; set; }
      public DbSet<Shift_Off_Request> Shift_Off_Requests { get; set; }
```

```
        public DbSet<Shift_Type> Shift_Types { get; set; }
        public DbSet<Skill> EmployeeSkills { get; set; }

    }
}
```

ΚΛΑΣΗ MyDbConfiguration

```
using System.Data.Entity;
using System.Data.Entity.Infrastructure;
using System.IO;

namespace NurseRostering.Database
{
    /* Με την παρακάτω κλάση, δίδεται η δυνατότητα στο Entity Framework
     * να εκκινεί ταχύτερα γιατί δημιουργεί ένα cache αρχείο με το μοντέλο
     * της βάσης δεδομένων και το χρησιμοποιεί κάθε φορά που εκκινεί, αντί
     * να το δημιουργεί κάθε φορά κατά την εκτέλεση του πρώτου ερωτήματος
     * προς τη βάση δεδομένων σε κάθε νέα εκκίνηση της εφαρμογής. */
    public class MyDbConfiguration : DbConfiguration
    {
        public MyDbConfiguration() : base()
        {
            var path = Path.GetDirectoryName(this.GetType().Assembly.Location);
            SetModelStore(new DefaultDbModelStore(path));
        }
    }
}
```

ΠΑΚΕΤΟ Resources

ΚΛΑΣΗ ClsPrint

```
using System;
using System.Collections;
using System.Drawing;
using System.Drawing.Printing;
using System.Linq;
using System.Windows.Forms;

namespace NurseRostering.Resources
{
    public class ClsPrint
    {
        #region Variables
```

```
        int iCellHeight = 0; //Used to get/set the datagridview cell height
        int iTotalWidth = 0; //
        int iRow = 0;//Used as counter
        bool bFirstPage = false; //Used to check whether we are printing first page
        bool bNewPage = false;// Used to check whether we are printing a new page
        int iHeaderHeight = 0; //Used for the header height
        StringFormat strFormat; //Used to format the grid rows.
        ArrayList arrColumnLefts = new ArrayList();//Used to save left coordinates of
columns
        ArrayList arrColumnWidths = new ArrayList();//Used to save column widths
        private PrintDocument _printDocument = new PrintDocument();
        private DataGridView gw = new DataGridView();
        private string _ReportHeader;

        #endregion

        public ClsPrint(DataGridView gridview, string ReportHeader)
        {
            _printDocument.PrintPage += new
PrintPageEventHandler(_printDocument_PrintPage);
            _printDocument.BeginPrint += new
PrintEventHandler(_printDocument_BeginPrint);
            gw = gridview;
            _ReportHeader = ReportHeader;
        }

        public void PrintForm()
        {
            //Open the print preview dialog
            PrintPreviewDialog objPPdialog = new PrintPreviewDialog();
            objPPdialog.Document = _printDocument;
            objPPdialog.Document.DefaultPageSettings.Landscape = true;
            objPPdialog.ShowDialog();
        }

        private void _printDocument_PrintPage(object sender,
System.Drawing.Printing.PrintPageEventArgs e)
        {
            //try
            //{
            //Set the left margin
            int iLeftMargin = e.MarginBounds.Left;
            //Set the top margin
            int iTopMargin = e.MarginBounds.Top;
            //Whether more pages have to print or not
            bool bMorePagesToPrint = false;
            int iTmpWidth = 0;

            //For the first page to print set the cell width and header height
            if (bFirstPage)
            {
                foreach (DataGridViewColumn GridCol in gw.Columns)
```

```
            {
                iTmpWidth = (int)(Math.Floor((double)((double)GridCol.Width /
                    (double)iTotalWidth * (double)iTotalWidth *
                    ((double)e.MarginBounds.Width / (double)iTotalWidth))));

                iHeaderHeight = (int)(e.Graphics.MeasureString(GridCol.HeaderText,
                    GridCol.InheritedStyle.Font, iTmpWidth).Height) + 11;

                // Save width and height of headers
                arrColumnLefts.Add(iLeftMargin);
                arrColumnWidths.Add(iTmpWidth);
                iLeftMargin += iTmpWidth;
            }
        }
        //Loop till all the grid rows not get printed
        while (iRow <= gw.Rows.Count - 1)
        {
            DataGridViewRow GridRow = gw.Rows[iRow];
            //Set the cell height
            iCellHeight = GridRow.Height + 5;
            int iCount = 0;
            //Check whether the current page settings allows more rows to print
            if (iTopMargin + iCellHeight >= e.MarginBounds.Height +
e.MarginBounds.Top)
            {
                bNewPage = true;
                bFirstPage = false;
                bMorePagesToPrint = true;
                break;
            }
            else
            {

                if (bNewPage)
                {
                    //Draw Header
                    e.Graphics.DrawString(_ReportHeader,
                        new Font(gw.Font, FontStyle.Bold),
                        Brushes.Black, e.MarginBounds.Left,
                        e.MarginBounds.Top - e.Graphics.MeasureString(_ReportHeader,
                        new Font(gw.Font, FontStyle.Bold),
                        e.MarginBounds.Width).Height - 13);

                    String strDate = "";
                    //Draw Date
                    e.Graphics.DrawString(strDate,
                        new Font(gw.Font, FontStyle.Bold), Brushes.Black,
                        e.MarginBounds.Left +
                        (e.MarginBounds.Width - e.Graphics.MeasureString(strDate,
                        new Font(gw.Font, FontStyle.Bold),
                        e.MarginBounds.Width).Width),
                        e.MarginBounds.Top - e.Graphics.MeasureString(_ReportHeader,
                        new Font(new Font(gw.Font, FontStyle.Bold),
```

```
                    FontStyle.Bold), e.MarginBounds.Width).Height - 13);

                //Draw Columns
                iTopMargin = e.MarginBounds.Top;
                DataGridViewColumn[] _GridCol = new
DataGridViewColumn[gw.Columns.Count];
                int colcount = 0;
                //Convert ltr to rtl
                foreach (DataGridViewColumn GridCol in gw.Columns)
                {
                    _GridCol[colcount++] = GridCol;
                }
                for (int i = 0; i<_GridCol.Count(); i++)
                {
                    e.Graphics.FillRectangle(new SolidBrush(Color.LightGray),
                        new Rectangle((int)arrColumnLefts[iCount], iTopMargin,
                        (int)arrColumnWidths[iCount], iHeaderHeight));

                    e.Graphics.DrawRectangle(Pens.Black,
                        new Rectangle((int)arrColumnLefts[iCount], iTopMargin,
                        (int)arrColumnWidths[iCount], iHeaderHeight));

                    e.Graphics.DrawString(_GridCol[i].HeaderText,
                        _GridCol[i].InheritedStyle.Font,
                        new SolidBrush(_GridCol[i].InheritedStyle.ForeColor),
                        new RectangleF((int)arrColumnLefts[iCount], iTopMargin,
                        (int)arrColumnWidths[iCount], iHeaderHeight), strFormat);
                    iCount++;
                }
                bNewPage = false;
                iTopMargin += iHeaderHeight;
            }
            iCount = 0;
            DataGridViewCell[] _GridCell = new
DataGridViewCell[GridRow.Cells.Count];
            int cellcount = 0;
            //Convert ltr to rtl
            foreach (DataGridViewCell Cel in GridRow.Cells)
            {
                _GridCell[cellcount++] = Cel;
            }
            //Draw Columns Contents
            for (int i = 0; i <_GridCell.Count(); i++)
            {
                if (_GridCell[i].Value != null)
                {
                    e.Graphics.DrawString(_GridCell[i].FormattedValue.ToString(),
                        _GridCell[i].InheritedStyle.Font,
                        new SolidBrush(_GridCell[i].InheritedStyle.ForeColor),
                        new RectangleF((int)arrColumnLefts[iCount],
                        (float)iTopMargin,
                        (int)arrColumnWidths[iCount], (float)iCellHeight),
                        strFormat);
```

```
                }
                //Drawing Cells Borders
                e.Graphics.DrawRectangle(Pens.Black,
                    new Rectangle((int)arrColumnLefts[iCount], iTopMargin,
                    (int)arrColumnWidths[iCount], iCellHeight));
                iCount++;
            }
        }
        iRow++;
        iTopMargin += iCellHeight;
    }
    //If more lines exist, print another page.
    if (bMorePagesToPrint)
        e.HasMorePages = true;
    else
        e.HasMorePages = false;
}

private void _printDocument_BeginPrint(object sender, PrintEventArgs e)
{
    try
    {
        strFormat = new StringFormat();
        strFormat.Alignment = StringAlignment.Center;
        strFormat.LineAlignment = StringAlignment.Center;
        strFormat.Trimming = StringTrimming.EllipsisCharacter;

        arrColumnLefts.Clear();
        arrColumnWidths.Clear();
        iCellHeight = 0;
        iRow = 0;
        bFirstPage = true;
        bNewPage = true;

        // Calculating Total Widths
        iTotalWidth = 0;
        foreach (DataGridViewColumn dgvGridCol in gw.Columns)
        {
            iTotalWidth += dgvGridCol.Width;
        }
    }
    catch (Exception ex)
    {
        MessageBox.Show(ex.Message, "Error", MessageBoxButtons.OK,
MessageBoxIcon.Error);
    }
}

  }
}
```

ΚΛΑΣΗ Modules

```
using System;
using System.Collections.Generic;
using System.Drawing;
using System.IO;
using System.Linq;
using System.Runtime.Serialization.Formatters.Binary;
using System.Windows.Forms;
using NurseRostering.Database;
using NurseRostering.InputData;

namespace NurseRostering.Resources
{
   [Serializable]
   class Modules
   {
      // Έλεγχος για το αν μια τιμή πεδίου MIN είναι μεγαλύτερη από μια τιμή πεδίου
MAX
      public bool Validate(int min, int max)
      {
         if (min > max)
         {
            return false;
         }
         return true;
      }

      // Μορφοποίηση δύο δεδομένων
      public string FormatTwoData(int parameter1, int parameter2)
      {
         // Παράδοση δεδομένων στον χρήστη
         return "(" + parameter1 + "|" + parameter2 + ")";
      }

      // Μορφοποίηση δύο δεδομένων
      public string FormatThreeData(int parameter1, int parameter2, int parameter3)
      {
         // Παράδοση δεδομένων στον χρήστη
         return "(" + parameter1 + "|" + parameter2 + "|" + parameter3 +")";
      }

      /* Προστασία από διπλή καταχώρηση ημερομηνίας, είτε εργασίας είτε
ανάπαυσης.
       * Εάν η μέθοδος επιστρέψει τιμή false σημαίνει ότι βρέθηκαν τα ζητούμενα
δεδομένα
       * μέσα στο οριζόμενο Listbox και ενημερώνεται σχετικά ο χρήστης.
       * Εάν η μέθοδος επιστρέψει τιμή true σημαίνει ότι τα προς αναζήτηση δεδομένα
       * δεν υπάρχουν μέσα στο Listbox στο οποίο διενεργείται η αναζήτηση */
      public bool CheckForDoubleRecords(string searchString, ListBox
plaisioAnazitisis)
```

```
        {
            foreach (var data in plaisioAnazitisis.Items)
            {
                if (searchString.TrimEnd(',', ' ') == Convert.ToString(data).TrimEnd(',', ' '))
                {
                    MessageBox.Show("Η εγγραφή αυτή έχει ήδη καταχωρηθεί.",
"NurseRostering",MessageBoxButtons.OK,MessageBoxIcon.Error);
                    return false;
                }
            }
            return true;
        }

        // Έλεγχος για το αν έχει πραγματοποιηθεί μια επιλογή στο ελεγχόμενο
αντικείμενο combobox
        public bool CheckIfAChoiceMade(ComboBox stoixeioComboBox)
        {
            // Εάν έχει πραγματοποιηθεί επιλογή μιας τιμής
            if(stoixeioComboBox.SelectedIndex != -1)
            {
                return true; // Επιτρέπεται να εκτελεστεί η ενέργεια του χρήστη
            } else // Εάν δεν έχει πραγματοποιηθεί καμιά επιλογή στο ελεγχόμενο
combobox
            {
                return false; // Δεν επιτρέπεται να εκτελεστεί η ενέργεια του χρήστη
            }
        }

        /* Με την παρακάτω μέθοδο, πραγματοποιείται βαθιά αντιγραφή ενός
συμβολαίου σε ένα άλλο.
         * Το προς αντιγραφή συμβόλαιο δηλώνεται μέσω της μεταβλητής
SimvolaioProsAntigrafi */
        public void Contract_DeepCopy(Contract SimvolaioProsAntigrafi)
        {
            MemoryStream ms = new MemoryStream();
            BinaryFormatter formatter = new BinaryFormatter();
            formatter.Serialize(ms, SimvolaioProsAntigrafi);
            MemoryStream ms2 = new MemoryStream(ms.ToArray());
            Contract NeoAntikeimeno = (Contract)formatter.Deserialize(ms2);
        }

        /* Με την παρακάτω μέθοδο, πραγματοποιείται βαθιά αντιγραφή ενός προτύπου
σε ένα άλλο.
         * Το προς αντιγραφή πρότυπο δηλώνεται μέσω της μεταβλητής
ProtipoProsAntigrafi */
        public void Pattern_DeepCopy(Pattern ProtipoProsAntigrafi)
        {
            MemoryStream ms = new MemoryStream();
            BinaryFormatter formatter = new BinaryFormatter();
            formatter.Serialize(ms, ProtipoProsAntigrafi);
            MemoryStream ms2 = new MemoryStream(ms.ToArray());
            Contract NeoAntikeimeno = (Contract)formatter.Deserialize(ms2);
        }
```

```
        // Μετατροπή μιας φωτογραφίας σε bytes
        public byte[] ImageToByte(Image img)
        {
            byte[] byteArray = new byte[0];
            using (MemoryStream stream = new MemoryStream())
            {
                img.Save(stream, System.Drawing.Imaging.ImageFormat.Jpeg);
                stream.Close();

                byteArray = stream.ToArray();
            }
            return byteArray;
        }
        // Λήψη δεδομένων βαρδιών από τη βάση δεδομένων και τοποθέτησή τους στη
λίστα βαρδιών της διεπαφής
        public void EmfanisiVardiwn(ComboBox cbShiftViewer)
        {
            // Λίστα στην οποία θα τοποθετηθούν οι τύποι των βαρδιών που υπάρχουν στη
βάση δεδομένων
            List<string> ShiftList = new List<string>();

            // Βάση δεδομένων
            NurseRosteringDB nrdb = new NurseRosteringDB();
            // Λήψη όλων των βαρδιών και εμφάνιση στη λίστα του αναγνωριστικού τους
            ShiftList = (from shift in nrdb.Shift_Types select shift.tipos).ToList();
            foreach (var item in ShiftList)
            {
                // Προβολή μόνο του αναγνωριστικού
                cbShiftViewer.Items.Add(item.ToString());
            }
            // Ανανέωση των αντικειμένων
            cbShiftViewer.Refresh();
        }
        public void DataGridFormat(DataGridView AntikeimenoDataGrid)
        {
            // Εναλλαγή χρώματος γραμμών
            AntikeimenoDataGrid.RowsDefaultCellStyle.BackColor = Color.LightGray;
            AntikeimenoDataGrid.AlternatingRowsDefaultCellStyle.BackColor =
Color.LightSlateGray;
            // Χρωματισμός επιλεγμένης γραμμής
            AntikeimenoDataGrid.DefaultCellStyle.SelectionBackColor = Color.Black;
            AntikeimenoDataGrid.DefaultCellStyle.SelectionForeColor = Color.White;

            // Κρύψιμο του κελιού ελέγχου γραμμής
            AntikeimenoDataGrid.RowHeadersVisible = false;

            // Επιλογή όλης της γραμμής ανεξαρτήτως του ποιο κελί θα επιλέξει ο χρήστης
            AntikeimenoDataGrid.SelectionMode =
DataGridViewSelectionMode.FullRowSelect;
        }
```

```
        // Έλεγχος για το αν΄το εισαγώμενο string ειναι αριθμητική τιμή
        public bool IsNumber(string inputString)
        {
            long n;
            if (long.TryParse(inputString, out n) == true) // Εάν είναι αριθμητική τιμή
            {
                return true; // Επέστρεψε true
            }
            else
            {
                MessageBox.Show("Η τιμή που προσπαθείτε να εισάγετε δεν είναι
αριθμητική. Παρακαλώ, διορθώστε την.", "NurseRostering", MessageBoxButtons.OK,
MessageBoxIcon.Error);
                return false; // Επέστρεψε false
            }
        }
        /* Σύγκριση των δύο ημερομηνιών που σχετίζονται με την έναρξη και λήξη του
υπο δημιουργία προγράμματος εργασίας.
         * Παρακάτω, πραγματοποιείται έλεγχος για την εγκυρότητα των ημερομηνιών.
Για παράδειγμα, δεν επιτρέπεται η ημερομηνία
         * έναρξης του προγράμματος να είναι η σημερινή ή προγενέστερη αυτής ή η
ημερομηνία λήξης να είναι προγενέστερη
         * της ημερομηνίας έναρξης του προγράμματος εργασίας */
        public bool ValidateDates(DateTime startDate, DateTime endDate)
        {
            if(startDate > endDate)
            {
                MessageBox.Show("Η ημερομηνία έναρξης δεν μπορεί να είναι
μεταγενέστερη από την ημερομηνία λήξης του προγράμματος εργασίας.", "
NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Error);
                return false;
            }
            else
            {
                if(startDate <= DateTime.Now)
                {
                    MessageBox.Show("Η ημερομηνία έναρξης δεν επιτρέπεται να είναι η
σημερινή ή προγενέστερη αυτής.","NurseRostering",
MessageBoxButtons.OK,MessageBoxIcon.Error); ;
                    return false;
                }
                else
                {
                    return true;
                }
            }
        }
    }
}
```

ΚΛΑΣΗ Program

```
using System;
using System.Windows.Forms;
using NurseRostering.GUI;

namespace NurseRostering
{
    /* Η παρακάτω κλάση χρησιμοποιείται για την εκκίνηση του προγράμματος
     * και διαβάζεται πρώτη */
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new frmSplash());
        }
    }
}
```

ΠΑΚΕΤΟ OutputData

ΦΟΡΜΑ frmAddSkill

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Linq;
using System.Windows.Forms;
using NurseRostering.Database;
using NurseRostering.InputData;

namespace NurseRostering.GUI
{
    public partial class frmAddSkill : Form
    {
        Available_Skill eidikotita; // Ειδικότητα του νοσοκόμου
        Modules module = new Modules(); // Διάφορες μέθοδοι
        private List<Available_Skill> ListOfSkills = new List<Available_Skill>(); //
Λίστα όλων των ειδικοτήτων
        private int RdgSkill = 0; // Επιλεγμένη σειρά του datagrid
```

```
        NurseRosteringDB nrdb = new NurseRosteringDB(); // Βάση δεδομένων

        public frmAddSkill()
        {
            InitializeComponent();
        }

        #region ΕΝΕΡΓΕΙΕΣ ΑΝΤΙΚΕΙΜΕΝΩΝ ΤΗΣ ΦΟΡΜΑΣ

        // Κλείσιμο της φόρμας και επιστροφή σε αυτήν που την κάλεσε
        private void btnCancel_Click(object sender, EventArgs e)
        {
            this.Close();
            GC.Collect(); // Συλλογή "σκουπιδιών"
        }

        private void btnSubmit_Click(object sender, EventArgs e)
        {
            // Σημαία που υποδεικνύει την ύπαρξη διπλών εγγραφών
            bool doubleRecord = false;

            // Εάν υπάρχει τουλάχιστον μια εισηγμένη ειδικότητα
            if (lstSkills.Items.Count > 0)
            {
                // Για κάθε ειδικότητα που βρίσκεται στη λίστα ειδικοτήτων
                foreach (var item in lstSkills.Items)
                {
                    doubleRecord = false; // Κατέβασμα της σημαίας διπλών εγγραφών
                    eidikotita = new Available_Skill(); // Δημιουργία νέου αντικειμένου
ειδικότητας
                    eidikotita.de3iotexnia = item.ToString(); // Εισαγωγή ειδικότητας στο
αντικείμενο
                    try // Απόπειρα εγγραφής δεδομένων
                    {
                        /* Για κάθε εγγραφή που αποπειράται να εγγραφεί στη βάση δεδομένων,
πραγματοποιείται
                         * σύγκριση με τις ήδη υπάρχουσες εγγραφές, προς αποφυγή
διπλότυπων. Ο έλεγχος γίνεται
                         * μέσα από το πρόγραμμα και όχι από το μηχανισμό της βάσης
δεδομένων για να μην υπάρχει
                         * άσκοπη κίνηση στη σύνδεση με τη βάση δεδομένων. Αν βρεθεί
κάποια ίδια εγγραφή, τότε
                         * η προς έλεγχο καταχώρησης εγγραφή δεν πραγματοποιείται. */
                        for(int i = 0; i < ListOfSkills.Count; i++)
                        {
                            // Εάν βρεθεί κάποια ίδια εγγραφή
                            if(item.ToString() == ListOfSkills[i].de3iotexnia.ToString())
                            {
                                // Σήκωμα σχετικής σημαίας
                                doubleRecord = true;
                            }
                        }
                        if (!doubleRecord)
```

```
                    {
                        // προσθήκη του στη βάση δεδομένων
                        nrdb.Available_Skills.Add(eidikotita);
                    }

                }
                catch (Exception ex) // Σε περίπτωση αποτυχίας, λήψη του σφάλματος
                {
                    // και εμφάνισή του με σχετικό μήνυμα στον χρήστη.
                    MessageBox.Show("Προέκυψε κάποιο απροσδόκητο σφάλμα. Σφάλμα:
" + ex.ToString(), "NurseRostering", MessageBoxButtons.OK,
MessageBoxIcon.Error);
                    return;
                }

            }
            nrdb.SaveChanges(); // Αποθήκευση στη βάση δεδομένων
            // Ενημέρωση του χρήστη για την επιτυχημένη καταχώρηση των δεδομένων
            MessageBox.Show("Η καταχώρηση ολοκληρώθηκε επιτυχώς.",
"NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Information);
            ListOfSkills.Clear(); // Καθαρισμός της λίστας των ειδικοτήτων που
λαμβάνεται από τη βάση δεδομένων
            frmAddSkill_Load(this, e); // Επαναφόρτωση όλων των δεδομένων
        }
    }

    // Κουμπί καταχώρησης ειδικότητας
    private void btnAddSkill_Click(object sender, EventArgs e)
    {
        /* Στην περίπτωση που δεν είναι κενό το πλαίσιο κειμένου ειδικότητας
         * και δεν υπάρχει ήδη η ειδικότητα προς καταχώρηση */
        if (module.CheckForDoubleRecords(txtEidikotita.Text.Trim(), lstSkills))
        {
            int check = nrdb.Available_Skills.Count(x => x.de3iotexnia ==
txtEidikotita.Text);

            if(check > 0)
            {
                // Ενημέρωση του χρήστη με σχετικό μήνυμα
                MessageBox.Show("Υπάρχει ήδη μια εγγραφή στη βάση δεδομένων με
αυτή την ειδικότητα.", "NurseRostering", MessageBoxButtons.OK,
MessageBoxIcon.Error);
                return; // Τερματισμός διαδικασίας
            }

            if (txtEidikotita.Text.Trim() != "")
            {
                // Εισαγωγή στη λίστα
                lstSkills.Items.Add(txtEidikotita.Text.Replace(" ", string.Empty));
                txtEidikotita.Text = ""; // Καθαρισμός πλαισίου κειμένου ειδικότητας
                txtEidikotita.Focus(); // Τοποθέτηση του κέρσορα στο πλαίσιο κειμένου
της ειδικότητας
            }
```

```
            // Σε διαφορετική περίπτωση ενημέρωση του χρήστη με σχετικό μήνυμα σφάλματος
            else
            {
                MessageBox.Show("Δεν έχει εισαχθεί κάποιο κείμενο.", "NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }
    }

    // Κουμπί διαγραφής επιλεγμένης ειδικότητας από τη λίστα ειδικοτήτων
    private void btnDelSkill_Click(object sender, EventArgs e)
    {
        // Έλεγχος για το εάν έχει πραγματοποιηθεί κάποια επιλογή ειδικότητας
        if(lstSkills.SelectedIndex > -1)
        {
            // Διαγραφή ειδικότητας
            lstSkills.Items.RemoveAt(lstSkills.SelectedIndex);
        }
    }

    // Κουμπί απαλοιφής όλων των ειδικοτήτων από τη λίστα
    private void btnClearList_Click(object sender, EventArgs e)
    {
        lstSkills.Items.Clear();
    }

    private void frmAddSkill_Load(object sender, EventArgs e)
    {
        // Λήψη όλων των ειδικοτήτων και τοποθέτησή τους στις σχετικές λίστες
        ListOfSkills = (from skl in nrdb.Available_Skills select skl).Distinct().ToList();

        /*********************************
         * ΑΡΧΗ ΔΙΑΜΟΡΦΩΣΗΣ ΤΟΥ DATAGRID *
         *********************************/

        // Τοποθέτηση των ειδικοτήτων στο DataGrid με αύξουσα σειρά τακτοποίησης στο αναγνωριστικό
        dgSkills.DataSource = ListOfSkills.OrderBy(x=>x.ID).ToList();

        // Χρωματισμός του datagrid
        module.DataGridFormat(dgSkills);

        // Καθορισμός πλάτους στηλών
        dgSkills.Columns[0].Width = (int)(dgSkills.Width * 0.15);
        dgSkills.Columns[1].Width = dgSkills.Width - (int)(dgSkills.Width * 0.15);

        // Καθορισμός τίτλου στηλών
        dgSkills.Columns[0].HeaderText = "ΑΑ";
        dgSkills.Columns[1].HeaderText = "Ειδικότητα";

        /*********************************
         * ΤΕΛΟΣ ΔΙΑΜΟΡΦΩΣΗΣ ΤΟΥ DATAGRID *
```

```
            ***********************************/

            // Ανανέωση των περιεχομένων των αντικειμένων προβολής δεδομένων
            lstSkills.Refresh();
            dgSkills.Refresh();
        }

        private void btnDelete_Click(object sender, EventArgs e)
        {
            // Ερώτηση προς στον χρήστη σχετικά με τη διαγραφή της επιλεγμένης
εγγραφής
            if(MessageBox.Show("Πρόκειται να διαγράψετε μια εγγραφή. Η διαγραφή θα
είναι μόνιμη και μη αναστρέψιμη. Θέλετε να συνεχίσετε;", "NurseRostering",
MessageBoxButtons.YesNo, MessageBoxIcon.Question) == DialogResult.Yes)
            {
                // Στην περίπτωση που απαντήσει καταφατικά
                var skillToDelete =
nrdb.Available_Skills.Find(int.Parse(dgSkills.Rows[RdgSkill].Cells[0].Value.ToString(
))); // Εύρεση εγγραφής προς διαγραφή
                nrdb.Available_Skills.Remove(skillToDelete); // Διαγραφή εγγραφής
                nrdb.SaveChanges();
                ListOfSkills.Clear(); // Καθαρισμός της λίστας των ειδικοτήτων που
λαμβάνεται από τη βάση δεδομένων
                frmAddSkill_Load(this, e); // Επαναφόρτωση όλων των δεδομένων
            }
            else // ενώ στην περίπτωση αρνητικής απάντησης
            {
                // Ενημέρωση του χρήστη με σχετικό μήνυμα
                MessageBox.Show("Η διαδικασία διαγραφής ακυρώθηκε από τον χρήστη.",
"NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Information);
            }
        }

        // Κατά την επιλογή ενός κελιού από το datagrid
        private void dgSkills_CellClick(object sender, DataGridViewCellEventArgs e)
        {
            // Δημοσίευση του αριθμού της γραμμής στην οποία βρίσκεται το κελί αυτό.
            RdgSkill = e.RowIndex;
        }

        #endregion
    }
}
```

ΦΟΡΜΑ frmChoice

```
using System;
using System.IO;
using System.Windows.Forms;

namespace NurseRostering.GUI
{
   public partial class frmChoice : Form
   {
      private string eidosProgrammatos { get; set; } = "sprint";

      public frmChoice()
      {
         InitializeComponent();
      }

      #region ΕΝΕΡΓΕΙΕΣ ΠΟΥ ΑΦΟΡΟΥΝ ΓΕΝΙΚΑ ΣΤΗΝ ΚΥΡΙΑ ΦΟΡΜΑ ΚΑΙ
ΟΧΙ ΣΕ ΚΑΠΟΙΑ ΣΥΓΚΕΚΡΙΜΜΕΝΗ

      private void KleisimoFormas(object sender, FormClosingEventArgs e)
      {
         this.Show(); // Εμφάνιση κύριας φόρμας, όταν κλείσει η φόρμα απαιτήσεων
προγράμματος εργασίας
      }

      #endregion

      private void btnView_Click(object sender, EventArgs e)
      {
         /* Έλεγχος για το εάν υπάρχει κάποιο αρχείο δημιουργημένου προγράμματος
          * από το οποίο θα ληφθούν οι ημερομηνίες έναρξης και λήξης αυτού. */
         if (!File.Exists(eidosProgrammatos + "_sol.txt"))
         {
            MessageBox.Show("Δεν υπάρχει κάποιο σχέδιο του προγράμματος
εργασίας.", "NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Error);
            return;
         }
         else
         {
            /* Δημιουργία και φόρτωση νέας φόρμας σχετική με την καταχώρηση στη
βάση δεδομένων
            * όλων των απαιτήσεων για τη δημιουργία του προγράμματος εργασίας. */
            frmProgram Program = new frmProgram();

            Program.eidosProgrammatos = eidosProgrammatos;// Ανακοίνωση στη
φόρμα του είδους του προγράμματος που δημιουργήθηκε
            Program.btnExit.Text = "Επιστροφή στην επιλογή τύπου προγράμματος για
προβολή";

            Program.Show(); // Εμφάνιση της φόρμας
```

```
            this.Hide(); // Απόκρυψη της φόρμας του κύριου μενού

            // Δημιουργία χειριστηρίου για τον έλεγχο κλεισίματος της φόρμας
            Program.FormClosing += KleisimoFormas;
        }
    }

    private void btnBack_Click(object sender, EventArgs e)
    {
        this.Close(); // Τερματισμός λειτουργίας της παρούσας φόρμας
    }

    private void rbSprint_CheckedChanged(object sender, EventArgs e)
    {
        eidosProgrammatos = "sprint";
        btnView.Enabled = true;
    }

    private void rbMedium_CheckedChanged(object sender, EventArgs e)
    {
        eidosProgrammatos = "medium";
        btnView.Enabled = true;
    }

    private void rbLong_CheckedChanged(object sender, EventArgs e)
    {
        eidosProgrammatos = "long";
        btnView.Enabled = true;
    }
  }
}
```

ΦΟΡΜΑ frmContracts

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Linq;
using System.Windows.Forms;
using NurseRostering.InputData;
using NurseRostering.GUI;
using NurseRostering.Database;

namespace NurseRostering.GUI
{
  public partial class frmContracts : Form
  {
    // Αντικείμενα
    Contract Simvolaio; // Ορισμός νέου αντικειμένου "Συμβόλαιο"
    Modules module = new Modules(); // Συλλογή μεθόδων ασφαλείας
```

```
        NurseRosteringDB nrdb = new NurseRosteringDB();

        /* Συλλογή αντικειμένων ανεπιθύμητων προτύπων που έχει δημιουργήσει ο
χρήστης. Αφορά σε όλα τα
         * διαθέσιμα πρότυπα ανεπιθύμητης εργασίας από ένα νοσοκόμο, μέσα από τα
οποία πρέπει να πραγματοποιήσει
         * τις επιλογές του. Μπορεί και να μην κάνει καμιά επιλογή. */
        private List<Pattern> ListOfUnWantedPatterns;

        // Συλλογή όλων των συμβολαίων
        private List<Contract> ListOfContracts = new List<Contract>();

        // Μεταβλητές
        // Εύρος Σαββατοκύριακου. Αρχικοποιείται για λόγους ασφαλείας με την default
τιμή SaturdaySunday
        private string savvatoKiriako = "SaturdaySunday"; // Προεπιλεγμένη τιμή εύρους
Σαββατοκύριακων
        private string patternsID = ""; // Λίστα με τα αναγνωριστικά των επιλεγμένων
ανεπιθύμητων προτύπων
        public frmContracts()
        {
            InitializeComponent();
            ListOfUnWantedPatterns = new List<Pattern>(); // Δημιουργία συλλογής
αντικειμένων ανεπιθύμητων προτύπων που έχει δημιουργήσει ο χρήστης
        }

        #region ΠΡΟΣΤΑΣΙΑ ΤΙΜΩΝ - ΟΙ ΕΛΑΧΙΣΤΕΣ ΔΕΝ ΥΠΕΡΒΑΙΝΟΥΝ ΤΙΣ
ΜΕΓΙΣΤΕΣ - ΟΧΙ ΔΙΠΛΕΣ ΗΜΕΡΟΜΗΝΙΕΣ

        /* Με τις παρακάτω συναρτήσεις πραγματοποιείται, για όλα τα αντικείμενα τα
οποία
         * είναι numericUpDown, έλεγχος για το αν το ελάχιστο πλήθος απαιτούμενων
βαρδιών είναι
         * μεγαλύτερο από το μέγιστο. Σε αυτή την περίπτωση, επείδη δεν είναι
         * επιτρεπτό κάτι τέτοιο, ο χρήστης θα ειδοποιείται με κατάλληλο μήνυμα
         * και η ελάχιστη τιμή δεν θα επιτρέπεται να υπερβεί τη μέγιστη */
        private void nudMin_Shift_ValueChanged(object sender, EventArgs e)
        {
            if (module.Validate((int)nudMin_Shift.Value, (int)nudMax_Shift.Value) ==
false)
            {
                // Μήνυμα προς τον χρήστη
                MessageBox.Show("Η τιμή δεν επιτρέπεται να είναι μεγαλύτερη από " +
nudMax_Shift.Value, "Nurse Rostering", MessageBoxButtons.OK,
MessageBoxIcon.Error);

                // Επανόρθωση τιμής
                nudMin_Shift.Value -= 1;
            }
        }

        private void nudCons_Min_Repo_ValueChanged(object sender, EventArgs e)
        {
```

```
            if (module.Validate((int)nudCons_Min_Repo.Value,
(int)nudCons_Max_Repo.Value) == false)
            {
                // Μήνυμα προς τον χρήστη
                MessageBox.Show("Η τιμή δεν επιτρέπεται να είναι μεγαλύτερη από " +
nudCons_Max_Repo.Value, "Nurse Rostering", MessageBoxButtons.OK,
MessageBoxIcon.Error);

                // Επανόρθωση τιμής
                nudCons_Min_Repo.Value -= 1;
            }
        }

        private void nudCons_Min_Work_ValueChanged(object sender, EventArgs e)
        {
            if (module.Validate((int)nudCons_Min_Work.Value,
(int)nudCons_Max_Work.Value) == false)
            {
                // Μήνυμα προς τον χρήστη
                MessageBox.Show("Η τιμή δεν επιτρέπεται να είναι μεγαλύτερη από " +
nudCons_Max_Work.Value, "Nurse Rostering", MessageBoxButtons.OK,
MessageBoxIcon.Error);

                // Επανόρθωση τιμής
                nudCons_Min_Work.Value -= 1;
            }
        }

        private void nudCons_Min_Weeknds_ValueChanged(object sender, EventArgs
e)
        {
            if (module.Validate((int)nudCons_Min_Weekends.Value,
(int)nudCons_Max_Weekends.Value) == false)
            {
                // Μήνυμα προς τον χρήστη
                MessageBox.Show("Η τιμή δεν επιτρέπεται να είναι μεγαλύτερη από " +
nudCons_Max_Weekends.Value, "Nurse Rostering", MessageBoxButtons.OK,
MessageBoxIcon.Error);

                // Επανόρθωση τιμής
                nudCons_Min_Weekends.Value -= 1;
            }
        }

        private void nudCons_Max_Weekends_ValueChanged(object sender, EventArgs
e)
        {
            if (module.Validate((int)nudCons_Min_Weekends.Value,
(int)nudCons_Max_Weekends.Value) == false)
            {
                // Μήνυμα προς τον χρήστη
```

```
            MessageBox.Show("Η τιμή δεν επιτρέπεται να είναι μικρότερη από " +
nudCons_Min_Weekends.Value, "Nurse Rostering", MessageBoxButtons.OK,
MessageBoxIcon.Error);

            // Επανόρθωση τιμής
            nudCons_Max_Weekends.Value += 1;
        }
    }

    private void nudMax_Shift_ValueChanged(object sender, EventArgs e)
    {
        if (module.Validate((int)nudMin_Shift.Value, (int)nudMax_Shift.Value) ==
false)
        {
            // Μήνυμα προς τον χρήστη
            MessageBox.Show("Η τιμή δεν επιτρέπεται να είναι μικρότερη από " +
nudMin_Shift.Value, "Nurse Rostering", MessageBoxButtons.OK,
MessageBoxIcon.Error);

            // Επανόρθωση τιμής
            nudMax_Shift.Value += 1;
        }
    }

    private void nudCons_Max_Repo_ValueChanged(object sender, EventArgs e)
    {
        if (module.Validate((int)nudCons_Min_Repo.Value,
(int)nudCons_Max_Repo.Value) == false)
        {
            // Μήνυμα προς τον χρήστη
            MessageBox.Show("Η τιμή δεν επιτρέπεται να είναι μικρότερη από " +
nudCons_Min_Repo.Value, "Nurse Rostering", MessageBoxButtons.OK,
MessageBoxIcon.Error);

            // Επανόρθωση τιμής
            nudCons_Max_Repo.Value += 1;
        }

    }

    private void nudCons_Max_Work_ValueChanged(object sender, EventArgs e)
    {
        if (module.Validate((int)nudCons_Min_Work.Value,
(int)nudCons_Max_Work.Value) == false)
        {
            // Μήνυμα προς τον χρήστη
            MessageBox.Show("Η τιμή δεν επιτρέπεται να είναι μικρότερη από " +
nudCons_Min_Work.Value, "Nurse Rostering", MessageBoxButtons.OK,
MessageBoxIcon.Error);

            // Επανόρθωση τιμής
            nudCons_Max_Work.Value += 1;
        }
```

```
        }

        #endregion

        /* Καθορισμός εύρους Σαββατοκύριακου με βάση την επιλογή του χρήστη. Η
τιμή αυτή θα
         * μεταφερθεί στη μεταβλητή savvatoKiriako για να μπορέσει να αποδοθεί στο
αντικείμενο του
         * συμβολαίου. Η τιμή της θα αλλάζει κάθε φορά που ο χρήστης αλλάζει την
εμφανιζόμενη τιμή
         * του σχετικού combobox και θα παίρνει μια από τις παρακάτω τιμές:
         * 1. SaturdaySunday
         * 2. FridaySaturdaySunday
         * 3. SaturdaySundayMonday
         * 4. FridaySaturdaySundayMonday */
        private void cbWeekEndDef_SelectedIndexChanged(object sender, EventArgs e)
        {
            switch (cbWeekEndDef.SelectedIndex)
            {
                case 0:
                    savvatoKiriako = "SaturdaySunday";
                    break;
                case 1:
                    savvatoKiriako = "FridaySaturdaySunday";
                    break;
                case 2:
                    savvatoKiriako = "SaturdaySundayMonday";
                    break;
                case 3:
                    savvatoKiriako = "FridaySaturdaySundayMonday";
                    break;
                default:
                    savvatoKiriako = "SaturdaySunday";
                    break;
            }
        }

        #region ΔΙΑΧΕΙΡΙΣΗ ΑΝΕΠΙΘΥΜΗΤΩΝ ΠΡΟΤΥΠΩΝ ΕΡΓΑΣΙΑΣ ΕΝΟΣ
ΝΟΣΟΚΟΜΟΥ

        // Εισαγωγή μιας επιλογής από τις διαθέσιμες, στη λίστα επιλεγμένων
ανεπιθύμητων προτύπων εργασίας
        private void btnAdd_Unwanted_Pattern_Click(object sender, EventArgs e)
        {
            // Εάν έχει πραγματοποιηθεί μια επιλογή από τη λίστα των διαθέσιμων
επιλογών
            if (lstUnwanted_List.SelectedIndex > -1)
            {
                /* Μεταφορά της επιλογής στη λίστα επιλεγμένων και διαγραφή της από τις
διαθέσιμες.
                 * Με αυτόν τον τρόπο, αποφεύγονται οι διπλές εγγραφές στη λίστα
επιλεγμένων */
```

```
lstUnwanted_Patterns.Items.Add(lstUnwanted_List.SelectedItem.ToString());
            lstUnwanted_List.Items.RemoveAt(lstUnwanted_List.SelectedIndex);
        }
    }

    /* Διαγραφή ενός επιλεγμένου ανεπιθύμητου προτύπου εργασίας από τη λίστα με
τις επιλογές
     * που πραγματοποίησε ο νοσοκόμος και εγγραφή της στη λίστα των διαθέσιμων
επιλογών του χρήστη */
    private void btnDel_Unwanted_Pattern_Click(object sender, EventArgs e)
    {
        // Εάν έχει πραγματοποιηθεί μια επιλογή από τον χρήστη
        if (lstUnwanted_Patterns.SelectedIndex > -1)
        {
            /* Μεταφορά της επιλογής στη λίστα διαθέσιμων επιλογών και διαγραφή
της από τις επιλεγμένες.
             * Με αυτόν τον τρόπο, αποφεύγονται οι διπλές εγγραφές στη λίστα των
διαθέσιμων επιλογών */

lstUnwanted_List.Items.Add(lstUnwanted_Patterns.SelectedItem.ToString());

lstUnwanted_Patterns.Items.RemoveAt(lstUnwanted_Patterns.SelectedIndex);
        }
    }

    // Καθαρισμός ολόκληρης της λίστας των επιλογών ενός νοσοκόμου και
επαναφορά τους στη λίστα των διαθέσιμων επιλογών
    private void btnClear_Unwanted_Pattern_Click(object sender, EventArgs e)
    {
        foreach (var item in lstUnwanted_Patterns.Items)
        {
            /* Μεταφορά της επιλογής στη λίστα διαθέσιμων επιλογών και έπειτα
διαγραφή όλων
             * από τις επιλεγμένες, από τη διεπαφή της εφαρμογής */
            lstUnwanted_List.Items.Add(item.ToString());
        }
        lstUnwanted_Patterns.Items.Clear(); // Καθαρισμός λίστας
    }

    #endregion

    #region ΜΕΝΟΥ ΕΠΙΛΟΓΩΝ

    // Διαδικασίες που αφορούν στη δημιουργία νέου κενού συμβολαίου
    private void νέοToolStripMenuItem_Click(object sender, EventArgs e)
    {
        Simvolaio = new Contract(); // Δημιουργία συμβολαίου
        // Απελευθέρωση του πλαισίου κειμένου για το όνομα του συμβολαίου
        txtContractName.ReadOnly = false;
        // Ενεργοποίηση κουμπιού αποθήκευσης συμβολαίου
        αποθήκευσηΣυμβολαίουToolStripMenuItem.Enabled = true;
    }
```

```
        // Μέταφορά όλων των δεδομένων του συμβολαίου, στο αντικείμενο της σχετικής κλάσης
        private void αποθήκευσηΣυμβολαίουToolStripMenuItem_Click(object sender, EventArgs e)
        {
            int contractFounded = 0; // Μετρητής διπλότυπων εγγραφών. Πρέπει να παραμείνει 0 μετά από έλεγχο για να συνεχιστεί η καταχώρηση

            // Έλεγχος στη βάση δεδομένων για το προς καταχώρηση συμβόλαιο με βάση το όνομά του, αφού δεν υπάρχει ο κωδικός συμβολαίου ακόμα.
            contractFounded = nrdb.Contracts.Count(x=>x.description == txtContractName.Text);

            if (contractFounded > 0) // Στην περίπτωση που βρεθεί κάποια εγγραφή με τα ζητούμενα κριτήρια
            {
                // Ενημέρωση του χρήστη με σχετικό μήνυμα
                MessageBox.Show("Αυτό το όνομα συμβολαίου υπάρχει ήδη. Η καταχώρηση αυτού του συμβολαίου δεν είναι δυνατή.", "NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Error);
                return; // Τερματισμός διαδικασίας εγγραφής.
            }

            // Εάν φορτωθούν όλα τα δεδομένα επιτυχώς στο συμβόλαιο
            if (Load_Contract_Data())
            {
                try // Απόπειρα αποθήκευσης προτύπου
                {
                    nrdb.Contracts.Add(Simvolaio); // Προσθήκη του συμβολαίου στη λίστα για καταχώρηση στη βάση δεδομένων
                    nrdb.SaveChanges(); // Αποθήκευση συμβολαίου
                    lstContracts.Items.Add(Simvolaio.description); // Προσθήκη του ονόματος του συμβολαίου, στη λίστα αυτών

                    // Ενημέρωση του χρήστη για την επιτυχή καταχώρηση των δεδομένων
                    MessageBox.Show("Το συμβόλαιο καταχωρήθηκε επιτυχώς.", "NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Information);
                }
                catch (Exception ex) // Στην περίπτωση που προκύψει κάποιο σφάλμα
                {
                    // Ενημέρωση του χρήστη σχετικά με το σφάλμα που προέκυψε
                    MessageBox.Show("Προέκυψε το παρακάτω σφάλμα " + ex + ".", "NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Error);
                }
            }
        }

        // Κουμπί επιστροφής στην κύρια φόρμα
        private void επιστροφήΣτοΚύριοΜενούToolStripMenuItem_Click(object sender, EventArgs e)
        {
            this.Close(); // Τερματισμός αυτής της φόρμας
```

```
            GC.Collect(); // Συλλογή "σκουπιδιών"
        }

        #endregion

        #region ΕΙΣΑΓΩΓΗ ΔΕΔΟΜΕΝΩΝ ΣΤΟ ΑΝΤΙΚΕΙΜΕΝΟ ΤΗΣ ΚΛΑΣΗΣ
CONTRACT

        private bool Load_Contract_Data()
        {
            // Όνομα συμβολαίου
            // Εάν μετά την αφαίρεση των λευκών χαρακτήρων δεν υπάρχει κάποιος άλλος
χαρακτήρας
            if (txtContractName.Text.Trim() == "")
            {
                // Εμφάνιση σχετικού μηνύματος σφάλματος
                MessageBox.Show("Δεν έχει καθοριστεί το όνομα του συμβολαίου. Δεν
είναι δυνατή η συνέχεια.", "NurseRostering", MessageBoxButtons.OK,
MessageBoxIcon.Error);
                this.txtContractName.Focus(); // Εστίαση στο αντικείμενο
                return false;
            }
            else
            {
                // Καταχώρηση του ονόματος συμβολαίου
                Simvolaio.description = txtContractName.Text.Trim();
            }

            if (module.CheckIfAChoiceMade(cbShift_Per_Day))
            {
                // Μια βάρδια ανά ημέρα
                Simvolaio.switch_singleAssignmentPerDay =
cbShift_Per_Day.SelectedIndex;
                Simvolaio.weight_singleAssignmentPerDay =
(int)nudWSingleAssPerDay.Value;
            }
            else
            {
                // Εμφάνιση σχετικού μηνύματος σφάλματος
                MessageBox.Show("Δεν έχει επιλεγεί η εργασία μιας βάρδιας ανά ημέρα.
Δεν είναι δυνατή η συνέχεια.", "NurseRostering", MessageBoxButtons.OK,
MessageBoxIcon.Error);
                this.cbShift_Per_Day.Focus(); // Εστίαση στο αντικείμενο
                return false;
            }

            /* Μέγιστο πλήθος βαρδιών ενός νοσοκόμου. Για να ενεργοποιηθεί η ρύθμιση,
πρέπει
             * η τιμή του nudMax_Shift να είναι μεγαλύτερη από το 0 (ON). Σε άλλη
περίπτωση, θεωρείται
             * ότι δεν είναι ενεργοποιημένη η ρύθμιση και ο διακόπτης παίρνει την τιμή 0
(OFF) */
            if (nudMax_Shift.Value == 0)
```

```
            {
                Simvolaio.switch_maxNumAssignments = 0;
            }
            else
            {
                Simvolaio.switch_maxNumAssignments = 1;
            }
            Simvolaio.weight_maxNumAssignments = (int)nudWMaxNumAss.Value;
            Simvolaio.value_maxNumAssignments = (int)nudMax_Shift.Value;

            /* Ελάχιστο πλήθος βαρδιών ενός νοσοκόμου. Για να ενεργοποιηθεί η
ρύθμιση, πρέπει
             * η τιμή του nudMin_Shift να είναι μεγαλύτερη από το 0 (ON). Σε άλλη
περίπτωση, θεωρείται
             * ότι δεν είναι ενεργοποιημένη η ρύθμιση και ο διακόπτης παίρνει την τιμή 0
(OFF) */
            if (nudMin_Shift.Value == 0)
            {
                Simvolaio.switch_minNumAssignments = 0;
            }
            else
            {
                Simvolaio.switch_minNumAssignments = 1;
            }
            Simvolaio.weight_minNumAssignments = (int)nudWMinNumAss.Value;
            Simvolaio.value_minNumAssignments = (int)nudMin_Shift.Value;

            /* Μέγιστο πλήθος συνεχόμενων ημερών εργασίας ενός νοσοκόμου. Για να
ενεργοποιηθεί η ρύθμιση, πρέπει
             * η τιμή του nudCons_Max_Work να είναι μεγαλύτερη από το 0 (ON). Σε
άλλη περίπτωση, θεωρείται
             * ότι δεν είναι ενεργοποιημένη η ρύθμιση και ο διακόπτης παίρνει την τιμή 0
(OFF) */
            if (nudCons_Max_Work.Value == 0)
            {
                Simvolaio.switch_maxConsecutiveWorkingDays = 0;
            }
            else
            {
                Simvolaio.switch_maxConsecutiveWorkingDays = 1;
            }
            Simvolaio.weight_maxConsecutiveWorkingDays =
(int)nudWMaxConsWDays.Value;
            Simvolaio.value_maxConsecutiveWorkingDays =
(int)nudCons_Max_Work.Value;

            /* Ελάχιστο πλήθος συνεχόμενων ημερών εργασίας ενός νοσοκόμου. Για να
ενεργοποιηθεί η ρύθμιση, πρέπει
             * η τιμή του nudCons_Min_Work να είναι μεγαλύτερη από το 0 (ON). Σε
άλλη περίπτωση, θεωρείται
             * ότι δεν είναι ενεργοποιημένη η ρύθμιση και ο διακόπτης παίρνει την τιμή 0
(OFF) */
            if (nudMin_Shift.Value == 0)
```

```
            {
                Simvolaio.switch_minConsecutiveWorkingDays = 0;
            }
            else
            {
                Simvolaio.switch_minConsecutiveWorkingDays = 1;
            }
            Simvolaio.weight_minConsecutiveWorkingDays =
(int)nudWMinNumAss.Value;
            Simvolaio.value_minConsecutiveWorkingDays = (int)nudMin_Shift.Value;

            /* Μέγιστο πλήθος συνεχόμενων ημερών ανάπαυσης ενός νοσοκόμου. Για να
ενεργοποιηθεί η ρύθμιση, πρέπει
             * η τιμή του nudCons_Max_Repo να είναι μεγαλύτερη από το 0 (ON). Σε
άλλη περίπτωση, θεωρείται
             * ότι δεν είναι ενεργοποιημένη η ρύθμιση και ο διακόπτης παίρνει την τιμή 0
(OFF) */
            if (nudCons_Max_Repo.Value == 0)
            {
                Simvolaio.switch_maxConsecutiveFreeDays = 0;
            }
            else
            {
                Simvolaio.switch_maxConsecutiveFreeDays = 1;
            }
            Simvolaio.weight_maxConsecutiveFreeDays =
(int)nudWMaxConsFreeDays.Value;
            Simvolaio.value_maxConsecutiveFreeDays = (int)nudCons_Max_Repo.Value;

            /* Ελάχιστο πλήθος συνεχόμενων ημερών ανάπαυσης ενός νοσοκόμου. Για να
ενεργοποιηθεί η ρύθμιση, πρέπει
             * η τιμή του nudCons_Min_Repo να είναι μεγαλύτερη από το 0 (ON). Σε
άλλη περίπτωση, θεωρείται
             * ότι δεν είναι ενεργοποιημένη η ρύθμιση και ο διακόπτης παίρνει την τιμή 0
(OFF) */
            if (nudCons_Min_Repo.Value == 0)
            {
                Simvolaio.switch_minConsecutiveFreeDays = 0;
            }
            else
            {
                Simvolaio.switch_minConsecutiveFreeDays = 1;
            }
            Simvolaio.weight_minConsecutiveFreeDays =
(int)nudWMaxConsFreeDays.Value;
            Simvolaio.value_minConsecutiveFreeDays = (int)nudCons_Min_Repo.Value;

            /* Μέγιστο πλήθος συνεχόμενων Σαββατοκύριακων εργασίας ενός νοσοκόμου.
Για να ενεργοποιηθεί η ρύθμιση, πρέπει
             * η τιμή του nudCons_Max_Weekends να είναι μεγαλύτερη από το 0 (ON).
Σε άλλη περίπτωση, θεωρείται
             * ότι δεν είναι ενεργοποιημένη η ρύθμιση και ο διακόπτης παίρνει την τιμή 0
(OFF) */
```

```
        if (nudCons_Max_Weekends.Value == 0)
        {
            Simvolaio.switch_maxConsecutiveWorkingWeekends = 0;
        }
        else
        {
            Simvolaio.switch_maxConsecutiveWorkingWeekends = 1;
        }
        Simvolaio.weight_maxConsecutiveWorkingWeekends =
(int)nudMaxConsWWE.Value;
        Simvolaio.value_maxConsecutiveWorkingWeekends =
(int)nudCons_Max_Weekends.Value;

        /* Ελάχιστο πλήθος συνεχόμενων Σαββατοκύριακων εργασίας ενός
νοσοκόμου. Για να ενεργοποιηθεί η ρύθμιση, πρέπει
         * η τιμή του nudCons_Min_Weeknds να είναι μεγαλύτερη από το 0 (ON). Σε
άλλη περίπτωση, θεωρείται
         * ότι δεν είναι ενεργοποιημένη η ρύθμιση και ο διακόπτης παίρνει την τιμή 0
(OFF) */
        if (nudCons_Min_Weekends.Value == 0)
        {
            Simvolaio.switch_minConsecutiveWorkingWeekends = 0;
        }
        else
        {
            Simvolaio.switch_minConsecutiveWorkingWeekends = 1;
        }
        Simvolaio.weight_minConsecutiveWorkingWeekends =
(int)nudMinConsWWE.Value;
        Simvolaio.value_minConsecutiveWorkingWeekends =
(int)nudCons_Min_Weekends.Value;

        /* Μέγιστο πλήθος Σαββατοκύριακων εργασίας ενός νοσοκόμου σε χρονικό
διάστημα προγράμματος 4 εβδομάδων.
         * Για να ενεργοποιηθεί η ρύθμιση, πρέπει η τιμή του nudMax_Weekends_FW
να είναι μεγαλύτερη
         * από το 0 (ON). Σε άλλη περίπτωση, θεωρείται ότι δεν είναι ενεργοποιημένη
η ρύθμιση και ο
         * διακόπτης παίρνει την τιμή 0 (OFF) */
        if (nudMax_Weekends_FW.Value == 0)
        {
            Simvolaio.switch_maxWorkingWeekendsInFourWeeks = 0;
        }
        else
        {
            Simvolaio.switch_maxWorkingWeekendsInFourWeeks = 1;
        }
        Simvolaio.weight_maxWorkingWeekendsInFourWeeks =
(int)nudMaxWE4Weeks.Value;
        Simvolaio.value_maxWorkingWeekendsInFourWeeks =
(int)nudMax_Weekends_FW.Value;
```

```
        // Ο νοσοκόμος θα εργάζεται ολόκληρο το Σαββατοκύριακο αν εργαστεί την
πρώτη ημέρα σε αυτό.
        if (module.CheckIfAChoiceMade(cbWhole_Weekend))
        {
            // Ολοκληρωμένα Σαββατοκύριακα
            Simvolaio.switch_completeWeekends = cbWhole_Weekend.SelectedIndex;
            Simvolaio.weight_completeWeekends = (int)nudCompleteWE.Value;
        }
        else
        {
            // Εμφάνιση σχετικού μηνύματος σφάλματος
            MessageBox.Show("Δεν έχει επιλεγεί η εργασία σε ίδια βάρδια για όλο το
Σαββατοκύριακο. Δεν είναι δυνατή η συνέχεια.", "NurseRostering",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            this.cbWhole_Weekend.Focus(); // Εστίαση στο αντικείμενο
            return false;
        }

        /* Η εργασία τα Σαββατοκύριακα θα παραμένει ίδια καθ' όλη τη διάρκειά του.
         * Δηλαδή, ο νοσοκόμος θα δουλεύει στην ίδια βάρδια ολόκληρο το
Σαββατοκύριακο */
        if (module.CheckIfAChoiceMade(cbSame_Shift_Weekend))
        {
            // Σταθερό είδος βάρδιας όλο το Σαββατοκύριακο
            Simvolaio.switch_identicalShiftTypesDuringWeekend =
cbSame_Shift_Weekend.SelectedIndex;
            Simvolaio.weight_identicalShiftTypesDuringWeekend =
(int)nudSameShiftWE.Value;
        }
        else
        {
            // Εμφάνιση σχετικού μηνύματος σφάλματος
            MessageBox.Show("Δεν έχει επιλεγεί το εύρος των Σαββατοκύριακων. Δεν
είναι δυνατή η συνέχεια.", "NurseRostering", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            this.cbSame_Shift_Weekend.Focus(); // Εστίαση στο αντικείμενο
            return false;
        }

        if (module.CheckIfAChoiceMade(cbNight_Shift_Before_Weekend))
        {
            // Όχι νυχτερινή βάρδια πριν από Σαββατοκύριακο ανάπαυσης
            Simvolaio.switch_noNightShiftBeforeFreeWeekend =
cbNight_Shift_Before_Weekend.SelectedIndex;
            Simvolaio.weight_noNightShiftBeforeFreeWeekend =
(int)nudWNightShiftBefWE.Value;
        }
        else
        {
            // Εμφάνιση σχετικού μηνύματος σφάλματος
            MessageBox.Show("Δεν έχει επιλεγεί η νυχτερινή εργασία πριν από ένα
Σαββατοκύριακο. Δεν είναι δυνατή η συνέχεια.", "NurseRostering",
MessageBoxButtons.OK, MessageBoxIcon.Error);
```

```
            this.cbNight_Shift_Before_Weekend.Focus(); // Εστίαση στο αντικείμενο
            return false;
        }

        if (module.CheckIfAChoiceMade(cbTwo_Repo_After_Night))
        {
            // Δύο ημέρες ανάπαυσης έπειτα από νυχτερινή βάρδια
            Simvolaio.switch_twoFreeDaysAfterNightShifts =
cbTwo_Repo_After_Night.SelectedIndex;
            Simvolaio.weight_twoFreeDaysAfterNightShifts =
(int)nudWFreeDaysAfterNight.Value;
        }
        else
        {
            // Εμφάνιση σχετικού μηνύματος σφάλματος
            MessageBox.Show("Δεν έχει επιλεγεί η δυνατότητα ανάπαυσης δύο ημερών
μετά άπό νυχτερινή βάρδια. Δεν είναι δυνατή η συνέχεια.", "NurseRostering",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            this.cbTwo_Repo_After_Night.Focus(); // Εστίαση στο αντικείμενο
            return false;
        }

        if (module.CheckIfAChoiceMade(cbAlternative_Skills))
        {
            // Ελάχιστες απαιτούμενες δεξιότητες νοσοκόμου
            Simvolaio.switch_alternativeSkillCategory =
cbAlternative_Skills.SelectedIndex;
            Simvolaio.weight_alternativeSkillCategory = (int)nudWAlterSkillCat.Value;
        }
        else
        {
            // Εμφάνιση σχετικού μηνύματος σφάλματος
            MessageBox.Show("Δεν έχει επιλεγεί η δυνατότητα εργασίας χωρίς
κατάλληλα προσόντα. Δεν είναι δυνατή η συνέχεια.", "NurseRostering",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            this.cbAlternative_Skills.Focus(); // Εστίαση στο αντικείμενο
            return false;
        }

        if (module.CheckIfAChoiceMade(cbWeekEndDef))
        {
            // Ορισμός εύρους του Σαββατοκύριακου
            Simvolaio.weekendDefinition = savvatoKiriako;
        }
        else
        {
            // Εμφάνιση σχετικού μηνύματος σφάλματος
            MessageBox.Show("Δεν έχει επιλεγεί το εύρος των Σαββατοκύριακων. Δεν
είναι δυνατή η συνέχεια.", "NurseRostering", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            this.cbWeekEndDef.Focus(); // Εστίαση στο αντικείμενο
            return false;
        }
```

```
            // Αν το επιλεγμένο, από το νοσοκόμο, πρότυπο ανεπιθύμητης εργασίας
            foreach(var record in ListOfUnWantedPatterns)
            {
                // με κάποιο από τα πρότυπα που βρίσκονται στη βάση δεδομένων
                foreach(var item in lstUnwanted_Patterns.Items)
                {
                    // ταιριάζουν μεταξύ τους
                    if(item.ToString() == record.ShiftType)
                    {
                        // τότε τοποθέτησε το αναγνωριστικό της εγγραφής στη σχετική
μεταβλητή
                        patternsID += record.ID.ToString() + " ";
                    }
                }
            }

            Simvolaio.UnwantedPatterns = patternsID.Trim();

            /* Καθαρισμός του περιεχόμενου αυτής της μεταβλητής γιατί προσθέτει τις
τιμές της
             * σε κάθε επόμενο συμβόλαιο που καταχωρείται, αν ο χρήστης επιλέξει να
καταχωρήσει
             * περισσότερα του ενός με μια εκτέλεση της φόρμας. */
            patternsID = "";
            // Πλήθος ανεπιθύμητων προτύπων εργασίας που επιλέχθηκαν από το χρήστη.
            Simvolaio.numberOfUnwantedPatterns = lstUnwanted_Patterns.Items.Count;
            return true;

        }

        #endregion

        private void frmContracts_Load(object sender, EventArgs e)
        {
            // Καθαρισμός όλων των λιστών για παν ενδεχόμενο
            lstUnwanted_List.Items.Clear(); // Πρότυπα διαθέσιμα στον χρήστη για
επιλογή
            lstContracts.Items.Clear(); // Συμβόλαια
            ListOfUnWantedPatterns.Clear(); // Λίστα ανεπιθύμητων προτύπων που
φορτώνεται από τη βάση δεδομένων
            ListOfContracts.Clear(); // Λίστα συμβολαίων που φορτώνεται από τη βάση
δεδομένων

            // Λήψη όλων των ανεπιθύμητων προτύπων εργασίας
            ListOfUnWantedPatterns = (from UnWP in nrdb.Patterns select
UnWP).Distinct().OrderBy(x=>x.ID).ToList();

            /* Προσθήκη του ονόματος κάθε προτύπου ανεπιθύμητης εργασίας στη
σχετική λίστα.
             * Αυτός ο τρόπος επιλέγεται γιατί δεν δημιουργεί κάποιο σφάλμα κατά τη
διαγραφή μιας επιλογής
             * από τη λίστα, κατά την εκτέλεση του προγράμματος. */
```

```
            foreach (var item in ListOfUnWantedPatterns)
            {
                lstUnwanted_List.Items.Add(item.ShiftType);
            }
            lstUnwanted_List.Refresh(); // Ανανέωση των περιεχομένων της λίστας
διαθέσιμων ανεπιθύμητων προτύπων εργασίας

            // Λήψη όλων των συμβολαίων που έχουν δημιουργηθεί
            ListOfContracts = (from c in nrdb.Contracts select c).OrderBy(x =>
x.ID).ToList();

            // Τοποθέτηση όλων των συμβολαίων που αντλήθηκαν από τη βάση, στο
πλαίσιο λίστας της διεπαφής
            foreach(var item in ListOfContracts)
            {
                lstContracts.Items.Add(item.description);
            }
            lstContracts.Refresh(); // Ανανέωση των περιεχομένων
        }

        private void διαγραφήΣυμβολαίουToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            if(lstContracts.SelectedIndex > -1)
            {
                // Ερώτηση προς στον χρήστη σχετικά με τη διαγραφή της επιλεγμένης
εγγραφής
                if (MessageBox.Show("Πρόκειται να διαγράψετε μια εγγραφή. Η διαγραφή
θα είναι μόνιμη και μη αναστρέψιμη. Θέλετε να συνεχίσετε;", "NurseRostering",
MessageBoxButtons.YesNo, MessageBoxIcon.Question) == DialogResult.Yes)
                {
                    // Στην περίπτωση που απαντήσει καταφατικά, εύρεση εγγραφής προς
διαγραφή
                    var recordToDelete = (from x in nrdb.Contracts select x).Where(x =>
x.description == lstContracts.SelectedItem.ToString()).FirstOrDefault();

                    // Έλεγχος για το αν σχετίζεται με κάποιους νοσοκόμους το υπό διαγραφή
συμβόλαιο
                    int numberOfNurses = (from z in nrdb.Employees select z).Where(z =>
z.contractID == recordToDelete.ID).Count();

                    // Αν σχετίζεται τουλάχιστον με ένα νοσοκόμο το υπό διαγραφή
συμβόλαιο
                    if (numberOfNurses > 0)
                    {
                        // Ενημέρωση του χρήστη ότι δεν είναι δυνατή η διαγραφή του
                        MessageBox.Show("Το συμβόλαιο με αναγνωριστικό όνομα \"" +
recordToDelete.description + "\" σχετίζεται με " + numberOfNurses + " νοσοκόμους.
Η διαγραφή του δεν είναι δυνατή.", "NurseRostering", MessageBoxButtons.OK,
MessageBoxIcon.Error);
                        return; // Τερματισμός διαδικασίας
                    }
                    else // αλλιώς
```

```
                {
                    // Διαγραφή του συμβολαίου από τη βάση δεδομένων και σχετική
ενημέρωση του χρήστη.
                    nrdb.Contracts.Remove(recordToDelete); // Διαγραφή εγγραφής
                    nrdb.SaveChanges();
                    lstContracts.Items.RemoveAt(lstContracts.SelectedIndex); // Αφαίρεση
συμβολαίου από τη λίστα αυτών, αφού διαγραφεί από τη βάση δεδομένων
                    MessageBox.Show("Η διαγραφή ολοκληρώθηκε επιτυχώς",
"NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Information);
                }
            }
            else // ενώ στην περίπτωση αρνητικής απάντησης
            {
                // Ενημέρωση του χρήστη με σχετικό μήνυμα
                MessageBox.Show("Η διαδικασία διαγραφής ακυρώθηκε από τον
χρήστη.", "NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Information);
            }
        }
        else
        {
            // Ενημέρωση του χρήστη με σχετικό μήνυμα
            MessageBox.Show("Δεν πραγματοποιήθηκε καμιά επιλογή συμβολαίου για
διαγραφή.", "NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }

    }

    private void button1_Click(object sender, EventArgs e)
    {
        // Δημιουργία νέας φόρμας καταχώρησης νέου γκρουπ πρότυπων
ανεπιθύμητης εργασίας
        frmNewPattern newUnwanted = new frmNewPattern();
        newUnwanted.btnBack.Text = "Επιστροφή στη δημιουργία συμβολαίου";
        newUnwanted.Show(); // Εμφάνιση φόρμας καταχώρησης νέου γκρουπ
πρότυπων ανεπιθύμητης εργασίας
        this.Hide(); // Κρύψιμο της παρούσας φόρμας

        // Δημιουργία χειριστηρίου για τον έλεγχο κλεισίματος της φόρμας
        newUnwanted.FormClosing += KleisimoFormas;
    }

    private void KleisimoFormas(object sender, FormClosingEventArgs e)
    {
        this.Show(); // Εμφάνιση κύριας φόρμας, όταν κλείσει η φόρμα απαιτήσεων
προγράμματος εργασίας
        frmContracts_Load(this, e);
    }
  }
}
```

ΦΟΡΜΑ frmEdit

```
using NurseRostering.Database;
using System;
using System.Data;
using System.Linq;
using System.Windows.Forms;

namespace NurseRostering.GUI
{

   public partial class frmEdit : Form
   {
      public frmEdit()
      {
         InitializeComponent();
      }

      NurseRosteringDB nrdb = new NurseRosteringDB();

      public int choiceDataTable = 0;
      private int dgEditSR = -1;

      private void frmEdit_Load(object sender, EventArgs e)
      {
         switch (choiceDataTable)
         {
            case 1:
               dgEdit.DataSource = (from x in nrdb.Shift_Types select x).OrderBy(x =>
x.ID).ToList();
               dgEdit.Columns[0].ReadOnly = true;
               dgEdit.Columns[1].ReadOnly = true;
               dgEdit.Columns[5].ReadOnly = true;
               dgEdit.Columns[6].ReadOnly = true;
               break;
            case 2:
               dgEdit.DataSource = (from x in nrdb.Contracts select x).OrderBy(x =>
x.ID).ToList();
               dgEdit.Columns[0].ReadOnly = true;
               dgEdit.Columns[1].ReadOnly = true;
               break;
            case 3:
               dgEdit.DataSource = (from x in nrdb.Employees select x).OrderBy(x =>
x.ID).ToList();
               dgEdit.Columns[0].ReadOnly = true;
               dgEdit.Columns[1].ReadOnly = true;
               dgEdit.Columns[3].ReadOnly = true;
               dgEdit.Columns[4].ReadOnly = true;
               dgEdit.Columns[11].ReadOnly = true;
               dgEdit.Columns[12].ReadOnly = true;
               break;
```

```
            case 4:
                dgEdit.DataSource = (from x in nrdb.EmployeeSkills select x).OrderBy(x
=> x.ID).ToList();
                dgEdit.Columns[0].ReadOnly = true;
                dgEdit.Columns[1].ReadOnly = true;
                dgEdit.Columns[2].ReadOnly = true;
                break;
            default:
                MessageBox.Show("Δεν έχει επιλεγεί κάποιος πίνακας για προβολή",
"NurseRostering");
                break;
        }
    }

    private void διαγραφήΕγγραφήςToolStripMenuItem_Click(object sender,
EventArgs e)
    {
        if(dgEditSR < 0)
        {
            // Ενημέρωση του χρήστη με σχετικό μήνυμα
            MessageBox.Show("Δεν πραγματοποιήθηκε καμιά επιλογή συμβολαίου για
διαγραφή.", "NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Error);
            return;
        }

        switch (choiceDataTable)
        {
            case 1:
                // Ερώτηση προς στον χρήστη σχετικά με τη διαγραφή της επιλεγμένης
εγγραφής
                if (MessageBox.Show("Πρόκειται να διαγράψετε μια εγγραφή. Η
διαγραφή θα είναι μόνιμη και μη αναστρέψιμη. Θέλετε να συνεχίσετε;",
"NurseRostering", MessageBoxButtons.YesNo, MessageBoxIcon.Question) ==
DialogResult.Yes)
                {
                    // Στην περίπτωση που απαντήσει καταφατικά
                    var recordToDelete =
nrdb.Shift_Types.Find(int.Parse(dgEdit.Rows[dgEditSR].Cells[0].Value.ToString())); //
Εύρεση εγγραφής προς διαγραφή
                    nrdb.Shift_Types.Remove(recordToDelete); // Διαγραφή εγγραφής
                    nrdb.SaveChanges();
                    dgEdit.DataSource = (from x in nrdb.Shift_Types select x).OrderBy(x
=> x.ID).ToList();
                    MessageBox.Show("Η διαγραφή ολοκληρώθηκε επιτυχώς",
"NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Information);
                }
                else // ενώ στην περίπτωση αρνητικής απάντησης
                {
                    // Ενημέρωση του χρήστη με σχετικό μήνυμα
                    MessageBox.Show("Η διαδικασία διαγραφής ακυρώθηκε από τον
χρήστη.", "NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Information);
                }
                break;
```

```
            case 2:
                // Ερώτηση προς στον χρήστη σχετικά με τη διαγραφή της επιλεγμένης
εγγραφής
                if (MessageBox.Show("Πρόκειται να διαγράψετε μια εγγραφή. Η
διαγραφή θα είναι μόνιμη και μη αναστρέψιμη. Θέλετε να συνεχίσετε;",
"NurseRostering", MessageBoxButtons.YesNo, MessageBoxIcon.Question) ==
DialogResult.Yes)
                {
                    // Στην περίπτωση που απαντήσει καταφατικά
                    var recordToDelete =
nrdb.Contracts.Find(int.Parse(dgEdit.Rows[dgEditSR].Cells[0].Value.ToString())); //
Εύρεση εγγραφής προς διαγραφή
                    nrdb.Contracts.Remove(recordToDelete); // Διαγραφή εγγραφής
                    nrdb.SaveChanges();
                    dgEdit.DataSource = (from x in nrdb.Contracts select x).OrderBy(x =>
x.ID).ToList();
                    MessageBox.Show("Η διαγραφή ολοκληρώθηκε επιτυχώς",
"NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Information);
                }
                else // ενώ στην περίπτωση αρνητικής απάντησης
                {
                    // Ενημέρωση του χρήστη με σχετικό μήνυμα
                    MessageBox.Show("Η διαδικασία διαγραφής ακυρώθηκε από τον
χρήστη.", "NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Information);
                }
                break;
            case 3:
                // Ερώτηση προς στον χρήστη σχετικά με τη διαγραφή της επιλεγμένης
εγγραφής
                if (MessageBox.Show("Πρόκειται να διαγράψετε μια εγγραφή. Η
διαγραφή θα είναι μόνιμη και μη αναστρέψιμη. Θέλετε να συνεχίσετε;",
"NurseRostering", MessageBoxButtons.YesNo, MessageBoxIcon.Question) ==
DialogResult.Yes)
                {
                    // Στην περίπτωση που απαντήσει καταφατικά
                    var recordToDelete =
nrdb.Employees.Find(int.Parse(dgEdit.Rows[dgEditSR].Cells[0].Value.ToString())); //
Εύρεση εγγραφής προς διαγραφή
                    nrdb.Employees.Remove(recordToDelete); // Διαγραφή εγγραφής
                    nrdb.SaveChanges();
                    dgEdit.DataSource = (from x in nrdb.Employees select x).OrderBy(x =>
x.ID).ToList();
                    MessageBox.Show("Η διαγραφή ολοκληρώθηκε επιτυχώς",
"NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Information);
                }
                else // ενώ στην περίπτωση αρνητικής απάντησης
                {
                    // Ενημέρωση του χρήστη με σχετικό μήνυμα
                    MessageBox.Show("Η διαδικασία διαγραφής ακυρώθηκε από τον
χρήστη.", "NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Information);
                }
                break;
            case 4:
```

```
                // Ερώτηση προς στον χρήστη σχετικά με τη διαγραφή της επιλεγμένης
εγγραφής
                if (MessageBox.Show("Πρόκειται να διαγράψετε μια εγγραφή. Η
διαγραφή θα είναι μόνιμη και μη αναστρέψιμη. Θέλετε να συνεχίσετε;",
"NurseRostering", MessageBoxButtons.YesNo, MessageBoxIcon.Question) ==
DialogResult.Yes)
                {
                    // Στην περίπτωση που απαντήσει καταφατικά
                    var recordToDelete =
nrdb.EmployeeSkills.Find(int.Parse(dgEdit.Rows[dgEditSR].Cells[0].Value.ToString())
); // Εύρεση εγγραφής προς διαγραφή
                    nrdb.EmployeeSkills.Remove(recordToDelete); // Διαγραφή εγγραφής
                    nrdb.SaveChanges();
                    dgEdit.DataSource = (from x in nrdb.EmployeeSkills select
x).OrderBy(x => x.ID).ToList();
                    MessageBox.Show("Η διαγραφή ολοκληρώθηκε επιτυχώς",
"NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Information);
                }
                else // ενώ στην περίπτωση αρνητικής απάντησης
                {
                    // Ενημέρωση του χρήστη με σχετικό μήνυμα
                    MessageBox.Show("Η διαδικασία διαγραφής ακυρώθηκε από τον
χρήστη.", "NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Information);
                }
                break;
            default:
                MessageBox.Show("Δεν έχει επιλεγεί κάποιος πίνακας για προβολή",
"NurseRostering");
                break;
            }

        }

        // Κατά την επιλογή ενός κελιού από το datagrid
        private void dgEdit_CellClick(object sender, DataGridViewCellEventArgs e)
        {
            // Δημοσίευση του αριθμού της γραμμής στην οποία βρίσκεται το κελί αυτό.
            dgEditSR = e.RowIndex;
        }

        private void αποθήκευσηΑλλαγώνToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            /* Απώλεια εστίασης από το κελί του πλέγματος δεδομένων. Αυτή η ενέργεια
             * πραγματοποιείται για να μπορέσει να καταχωρηθεί στη βάση δεδομένων η
αλλαγή
             * που πραγματοποιήθηκε από τον χρήστη */
            groupBox1.Focus();
            try // Απόπειρα αποθήκευσης
            {
                nrdb.SaveChanges(); // Αποθήκευση και ενημέρωση του χρήστη
                MessageBox.Show("Η εγγραφή ενημερώθηκε επιτυχώς.", "NurseRostering",
MessageBoxButtons.OK, MessageBoxIcon.Information);
```

```
            }
            catch(Exception ex) // Στην περίπτωση που συμβεί κάποιο σφάλμα, σύλληψη
αυτού.
            {
                // Και ενημέρωση του χρήστη
                MessageBox.Show("Συνέβει κάποιο απροσδόκητο σφάλμα. Σφάλμα: " +
ex.ToString(), "NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }

        private void κλείσιμοΠροβολήςToolStripMenuItem_Click(object sender,
EventArgs e)
        {
            // Κλείσιμο της παρούσας φόρμας
            this.Close();
        }
    }

}
```

ΦΟΡΜΑ frmEmployee

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Windows.Forms;
using NurseRostering.Database;
using NurseRostering.InputData;

namespace NurseRostering.GUI
{
    public partial class frmEmployee : Form
    {
        Modules module = new Modules();
        Employee emp; // Δήλωση αντικειμένου νέου υπαλλήλου
        Skill texni; // Ειδικότητα του νοσοκόμου
        NurseRosteringDB nrdb = new NurseRosteringDB(); // Βάση δεδομένων

        // Λίστα συμβολαίων από τα οποία θα γίνει επιλογή ενός για τον υπάλληλο
        List<Contract> ListOfContracts = new List<Contract>();

        // Λίστα με όλες τις διαθέσιμες ειδικότητες για επιλογή
        List<Available_Skill> ListOfSkills = new List<Available_Skill>();

        private string emp_name = ""; // Ονοματεπώνυμο νοσοκόμου
        private int emp_contractID = -1; // Αναγνωριστικό συμβολαίου
        private string emp_amka = ""; // ΑΜΚΑ νοσοκόμου
```

```
        private string emp_afm = ""; // ΑΦΜ νοσοκόμου
        private string emp_dieu8unsi = ""; // Διεύθυνση οικίας νοσοκόμου
        private string emp_poli = ""; // Πόλη διαμονής νοσοκόμου
        private string emp_tk = ""; // Ταχυδρομικός κωδικός
        private string emp_kinito = ""; // Αριθμός κινητού τηλεφώνου νοσοκόμου
        private string emp_tilefwno = ""; // Αριθμός τηλεφώνου οικίας νοσοκόμου
        private string emp_email = ""; // Διεύθυνση e-mail νοσοκόμου
        private int emp_numberOfSkills = 0; // Πλήθος δεξιοτεχνιών νοσοκόμου
        private byte[] emp_photo;  // Φωτογραφία υπαλλήλου

        public frmEmployee()
        {
            InitializeComponent();
        }

        #region ΕΝΕΡΓΕΙΕΣ ΠΟΥ ΠΡΑΓΜΑΤΟΠΟΙΟΥΝΤΑΙ ΑΦΟΥ Ο ΧΡΗΣΤΗΣ
ΠΑΤΗΣΕΙ ΤΟ ΚΟΥΜΠΙ ΤΗΣ ΚΑΤΑΧΩΡΗΣΗΣ - ΕΙΣΑΓΩΓΗ ΔΕΔΟΜΕΝΩΝ ΣΤΟ
ΑΝΤΙΚΕΙΜΕΝΟ ΤΟΥ ΥΠΑΛΛΗΛΟΥ

        // Κουμπί καταχώρησης
        private void btnSubmit_Click(object sender, EventArgs e)
        {
            int employeFoundedAmka = 0; // Μετρητής διπλότυπων εγγραφών σχετικά με
τον ΑΜΚΑ. Πρέπει να παραμείνει 0 μετά από έλεγχο για να συνεχιστεί η
καταχώρηση
            int employeFoundedAfm = 0; // Μετρητής διπλότυπων εγγραφών σχετικά με
το ΑΦΜ. Πρέπει να παραμείνει 0 μετά από έλεγχο για να συνεχιστεί η καταχώρηση

            // Έλεγχος στη βάση δεδομένων για το προς καταχώρηση ΑΜΚΑ και ΑΦΜ,
αφού δεν υπάρχει ο κωδικός πελάτη ακόμα.
            employeFoundedAmka = nrdb.Employees.Count(x=>x.amka ==
txtAmka.Text);
            employeFoundedAfm =  nrdb.Employees.Count(x => x.afm == txtAfm.Text);

            if (employeFoundedAmka > 0 || employeFoundedAfm > 0) // Στην περίπτωση
που βρεθεί κάποια εγγραφή με τα ζητούμενα κριτήρια
            {
                // Καθορισμός εστίασης δείκτη στο κατάλληλο πλαίσιο κειμένου και
ενημέρωση του χρήστη με σχετικό μήνυμα.
                if(employeFoundedAmka > 0)
                {
                    // Ενημέρωση του χρήστη με σχετικό μήνυμα
                    MessageBox.Show("Υπάρχει ήδη κάποιος υπάλληλος με τον Α.Μ.Κ.Α.
που εισάγατε. Η καταχώρηση αυτού του υπαλλήλου δεν είναι δυνατή.",
"NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Error);

                    txtAmka.Focus();
                }

                if (employeFoundedAfm > 0)
                {
                    // Ενημέρωση του χρήστη με σχετικό μήνυμα
```

```
                MessageBox.Show("Υπάρχει ήδη κάποιος υπάλληλος με τον Α.Φ.Μ. που εισάγατε. Η καταχώρηση αυτού του υπαλλήλου δεν είναι δυνατή.", "NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Error);

                txtAfm.Focus();
            }

            return; // Τερματισμός διαδικασίας εγγραφής.
        }

        // Εισαγωγή των δεδομένων στο νέο αντικείμενο του υπαλλήλου
        if(Load_Employe_Data() == true)
        {
            // Ενημέρωση του χρήστη για την επιτυχή καταχώρηση του νέου υπαλλήλου
            MessageBox.Show("Ο νέος υπάλληλος καταχωρήθηκε με επιτυχία.", "NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Information);
        }

    }

    #endregion

    #region ΜΕΤΑΚΙΝΗΣΗ ΕΙΔΙΚΟΤΗΤΩΝ ΜΕΤΑΞΥ ΛΙΣΤΩΝ ΔΙΑΘΕΣΙΜΩΝ ΚΑΙ ΕΙΔΙΚΟΤΗΤΩΝ ΤΟΥ ΝΟΣΟΚΟΜΟΥ

    /* Παρακάτω πραγματοποιείται διαγραφή μιας ειδικότητας από αυτές που διατίθενται
     * για κάποιο νέο υπάλληλο και προστίθεται στη λίστα με τις ειδικότητες που έχει ο υπάλληλος.
     * Με αυτόν τον τρόπο, αποφεύγεται η διπλή καταχώρηση ειδικοτήτων στον ίδιο υπάλληλο.*/
    private void btnAddSkill_Click(object sender, EventArgs e)
    {
        if (lstEmployeeSkills.SelectedIndex > -1) // Εάν έχει επιλεγεί κάποια ειδικότητα
        {
            // Εισαγωγή αυτής της ειδικότητας στη λίστα των διαθέσιμων ειδικοτήτων
            lstAvailableSkills.Items.Add(lstEmployeeSkills.SelectedItem.ToString());

            // Διαγραφή από τη λίστα των ειδικοτήτων που αφορούν στον συγκεκριμμένο υπάλληλο.
            lstEmployeeSkills.Items.RemoveAt(lstEmployeeSkills.SelectedIndex);
        }
    }

    /* Παρακάτω πραγματοποιείται διαγραφή μιας ειδικότητας από αυτές που έχουν
     * απονεμηθεί στο νέο υπάλληλο και προστίθεται στη λίστα με τις διαθέσιμες ειδικότητες.
     * Με αυτόν τον τρόπο, αποφεύγεται η διπλή καταχώρηση ειδικοτήτων στον ίδιο υπάλληλο.*/
    private void btnDelSkill_Click(object sender, EventArgs e)
    {
        if(lstAvailableSkills.SelectedIndex > -1) // Εάν έχει επιλεγεί κάποια ειδικότητα
```

```
            {
                // Εισαγωγή αυτής της ειδικότητας στη λίστα του υπαλλήλου
                lstEmployeeSkills.Items.Add(lstAvailableSkills.SelectedItem.ToString());

                // Διαγραφή από τη λίστα των διαθέσιμων ειδικοτήτων.
                lstAvailableSkills.Items.RemoveAt(lstAvailableSkills.SelectedIndex);
            }
        }

        // Εκχώρηση όλων των διαθέσιμων ειδικοτήτων σε έναν υπάλληλο
        private void btnAddAllSkills_Click(object sender, EventArgs e)
        {
            // Για κάθε διαθέσιμη ειδικότητα
            foreach(var item in lstAvailableSkills.Items)
            {
                // Εκχώρησή της στη λίστα ειδικοτήτων του υπαλλήλου
                lstEmployeeSkills.Items.Add(item.ToString());
            }
            lstAvailableSkills.Items.Clear(); // Διαγραφή όλων των ειδικοτήτων από τη
λίστα των διαθέσιμων
        }

        // Αφαίρεση όλων των ειδικοτήτων από έναν υπάλληλο
        private void btnDelAllSkills_Click(object sender, EventArgs e)
        {
            // Για κάθε διαθέσιμη ειδικότητα
            foreach (var item in lstEmployeeSkills.Items)
            {
                // Εκχώρησή της στη λίστα ειδικοτήτων του υπαλλήλου
                lstAvailableSkills.Items.Add(item.ToString());
            }
            lstEmployeeSkills.Items.Clear(); // Διαγραφή όλων των ειδικοτήτων από τη
λίστα των διαθέσιμων
        }

        #endregion

        // Κουμπί αναζήτησης φωτογραφίας υπαλλήλου
        private void btnSearch_Click(object sender, EventArgs e)
        {
            // Εμφανίζει ένα OpenFileDialog πλαίσιο για την επιλογή της κατάλληλης
φωτογραφίας.
            OpenFileDialog imageSearch = new OpenFileDialog();
            imageSearch.Filter = "Αρχεία φωτογραφιών|*.jpg";
            imageSearch.Title = "Επιλέξτε μια φωτογραφία";

            // Εμφάνιση του παραθύρου.
            /* Όταν πατηθεί το OK και έχει επιλεγεί κάποια φωτογραφία, τότε αυτή
             * θα εμφανιστεί μέσα στο σχετικό πλαίσιο */
            if (imageSearch.ShowDialog() == DialogResult.OK)
            {
                imgEmp.Image = Image.FromFile(imageSearch.FileName);
                imgEmp.SizeMode = PictureBoxSizeMode.Zoom;
```

```
            }
        }

        private void btnExit_Click(object sender, EventArgs e)
        {
            nrdb = null;
            // Τερματισμός αυτής της φόρμας και εμφάνιση αυτής που την κάλεσε.
            this.Close();
            GC.Collect(); // Συλλογή "σκουπιδιών"
        }

        private void btnCancelPhoto_Click(object sender, EventArgs e)
        {
            // Διαγραφή φωτογραφίας υπαλλήλου
            imgEmp.Image = null;
        }

        private void frmEmployee_Load(object sender, EventArgs e)
        {
            // Καθαρισμός του ComboBox συμβολαίων από τυχόν κατάλοιπα
            cbSimvolaio.Items.Clear();

            // Εισαγωγή επιλογής δημιουργίας νέου συμβολαίου
            cbSimvolaio.Items.Add("Νέο συμβόλαιο");

            // Λήψη όλων των διαθέσιμων συμβολαίων για επιλογή ενός
            ListOfContracts = (from c in nrdb.Contracts select c).OrderBy(x =>
x.description).ToList();

            foreach (var item in ListOfContracts)
            {
                // Προσθήκη του ονόματος του συμβολαίου στη λίστα για επιλογή
                cbSimvolaio.Items.Add(item.description);
            }

            // Λήψη όλων των ειδικοτήτων και τοποθέτησή τους στις σχετικές λίστες
            ListOfSkills = (from skl in nrdb.Available_Skills select skl).OrderBy(x =>
x.de3iotexnia).ToList();

            // Τοποθέτηση των μοναδικών ειδικοτήτων στη λίστα
            var SkillList = ListOfSkills.Select(x => x.de3iotexnia).Distinct().ToList();
            foreach (var item in SkillList)
            {
                // Προβολή της ειδικότητας στη λίστα ειδικοτήτων
                lstAvailableSkills.Items.Add(item.ToString());
            }
        }

        private bool Load_Employe_Data()
        {
            // Εισαγωγή δεδομένων στο αντικείμενο που αφορά στο νοσοκόμο
            if (module.CheckIfAChoiceMade(cbSimvolaio))
```

```
            {
                Contract conID = (from c in nrdb.Contracts select c).Where(x =>
x.description == cbSimvolaio.SelectedItem.ToString()).FirstOrDefault();
                emp_contractID = int.Parse(conID.ID.ToString());
            }
            else
            {
                MessageBox.Show("Δεν έχει πραγματοποιηθεί επιλογή συμβολαίου γι
αυτόν τον νοσοκόμο. Η καταχώρηση στοιχείων δεν είναι δυνατόν να συνεχιστεί.",
"NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Error);
                return false;
            }

            if (txtName.Text.Trim() != "")
            {
                emp_name = txtName.Text;
            }
            else
            {
                MessageBox.Show("Δεν έχει πραγματοποιηθεί εισαγωγή ονοματεπώνυμου
του νοσοκόμου. Η καταχώρηση στοιχείων δεν είναι δυνατόν να συνεχιστεί.",
"NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Error);
                txtName.Focus();
                return false;
            }

            if (txtAmka.Text.Trim() != "")
            {
                if(module.IsNumber(txtAmka.Text) == true)
                {
                    emp_amka = txtAmka.Text;
                }
                else
                {
                    return false;
                }

            }
            else
            {
                MessageBox.Show("Δεν έχει πραγματοποιηθεί εισαγωγή ΑΜΚΑ του
νοσοκόμου. Η καταχώρηση στοιχείων δεν είναι δυνατόν να συνεχιστεί.",
"NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Error);
                txtAmka.Focus();
                return false;
            }

            if (txtAfm.Text.Trim() != "")
            {
                if (module.IsNumber(txtAfm.Text) == true)
                {
                    emp_afm = txtAfm.Text;
                }
```

```
            else
            {
                return false;
            }
        }
        else
        {
            MessageBox.Show("Δεν έχει πραγματοποιηθεί εισαγωγή ΑΦΜ του
νοσοκόμου. Η καταχώρηση στοιχείων δεν είναι δυνατόν να συνεχιστεί.",
"NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Error);
            txtAfm.Focus();
            return false;
        }

        if (txtAddress.Text.Trim() != "")
        {
            emp_dieu8unsi = txtAddress.Text;
        }
        else
        {
            MessageBox.Show("Δεν έχει πραγματοποιηθεί εισαγωγή διεύθυνσης
κατοικίας του νοσοκόμου. Η καταχώρηση στοιχείων δεν είναι δυνατόν να
συνεχιστεί.", "NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Error);
            txtAddress.Focus();
            return false;
        }

        if (txtPoli.Text.Trim() != "")
        {
            emp_poli = txtPoli.Text;
        }
        else
        {
            MessageBox.Show("Δεν έχει πραγματοποιηθεί εισαγωγή πόλης διαμονής
του νοσοκόμου. Η καταχώρηση στοιχείων δεν είναι δυνατόν να συνεχιστεί.",
"NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Error);
            txtPoli.Focus();
            return false;
        }

        if (txtTK.Text.Trim() != "")
        {
            if (module.IsNumber(txtTK.Text) == true)
            {
                emp_tk = txtTK.Text;
            }
            else
            {
                return false;
            }
        }
        else
        {
```

```
            MessageBox.Show("Δεν έχει πραγματοποιηθεί εισαγωγή ταχυδρομικού
κώδικα του νοσοκόμου. Η καταχώρηση στοιχείων δεν είναι δυνατόν να συνεχιστεί.",
"NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Error);
            txtTK.Focus();
            return false;
        }

        if (txtKinito.Text.Trim() != "")
        {
            if (module.IsNumber(txtKinito.Text) == true)
            {
                emp_kinito = txtKinito.Text;
            }
            else
            {
                return false;
            }
        }
        else
        {
            MessageBox.Show("Δεν έχει πραγματοποιηθεί εισαγωγή αριθμού κινητού
τηλεφώνου του νοσοκόμου. Η καταχώρηση στοιχείων δεν είναι δυνατόν να
συνεχιστεί.", "NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Error);
            txtKinito.Focus();
            return false;
        }

        if (txtTilefwno.Text.Trim() != "")
        {
            if (module.IsNumber(txtTilefwno.Text) == true)
            {
                emp_tilefwno = txtTilefwno.Text;
            }
            else
            {
                return false;
            }
        }
        else
        {
            MessageBox.Show("Δεν έχει πραγματοποιηθεί εισαγωγή αριθμού σταθερού
τηλεφώνου του νοσοκόμου. Η καταχώρηση στοιχείων δεν είναι δυνατόν να
συνεχιστεί.", "NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Error);
            txtTilefwno.Focus();
            return false;
        }

        if (txtEmail.Text.Trim() != "")
        {
            emp_email = txtEmail.Text;
        }
        else
        {
```

```
            MessageBox.Show("Δεν έχει πραγματοποιηθεί εισαγωγή διεύθυνσης e-mail
του νοσοκόμου. Η καταχώρηση στοιχείων δεν είναι δυνατόν να συνεχιστεί.",
"NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Error);
            txtEmail.Focus();
            return false;
        }

        foreach (var item in lstEmployeeSkills.Items)
        {
            texni = new Skill();
            texni.de3iotexnia = item.ToString();
            texni.employeeID = int.Parse(emp_afm);
            emp_numberOfSkills += 1;
            nrdb.EmployeeSkills.Add(texni);
        }

        // Πλήθος δεξιοτήτων του υπαλλήλου
        emp_numberOfSkills = lstEmployeeSkills.Items.Count;

        if (imgEmp.Image != null)
        {
            // Αποθήκευση της φωτογραφίας του υπάλλήλου
            emp_photo = module.ImageToByte(imgEmp.Image);
        }

        emp = new Employee // Δημιουργία αντικειμένου νέου υπαλλήλου
        {
            name = emp_name,
            contractID = emp_contractID,
            amka = emp_amka,
            afm = emp_afm,
            dieu8unsi = emp_dieu8unsi,
            poli = emp_poli,
            tk = emp_tk,
            kinito = emp_kinito,
            tilefwno = emp_tilefwno,
            email = emp_email,
            numberOfSkills = emp_numberOfSkills,
            photo = emp_photo,
        };

        try // Απόπειρα καταχώρησης νέου υπαλλήλου.
        {
            nrdb.Employees.Add(emp); // Προσθήκη του νοσοκόμου στη λίστα προς
καταχώρηση στη βάση δεδομένων
            nrdb.SaveChanges(); // Καταχώρηση στη βάση δεδομένων
            /* Καθαρισμός της λίστας δεξιοτήτων. Αυτή η ενέργεια είναι απαραίτητη,
διότι
             * αν δεν καθαριστεί, τα περιεχόμενά της ενδέχεται να περάσουν στα
στοιχεία του
             * επόμενου υπαλληλου, στην περίπτωση καταχώρησης περισσότερων του
ενός
             * νοσοκόμων τη φορά */
```

```
            return true;
        }
        catch (Exception ex) // Εάν προκύψει κάποιο σφάλμα
        {
            // Ενημέρωση του χρήστη σχετικά με το σφάλμα που προέκυψε
            MessageBox.Show("Προέκυψε το παρακάτω σφάλμα. Η καταχώρηση του
υπαλλήλου δεν πραγματοποιήθηκε. Σφαλμα: " + ex, "NurseRostering",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            return false; // Διακοπή της διαδικασίας.
        }
    }

    private void btnNewSkill_Click(object sender, EventArgs e)
    {
        // Δημιουργία νέας φόρμας καταχώρησης νέου υπαλλήλου
        frmAddSkill newEmp = new frmAddSkill();

        newEmp.btnCancel.Text = "Επιστροφή στην καταχώρηση υπαλλήλου";
        newEmp.Show(); // Εμφάνιση φόρμας καταχώρησης νέου υπαλλήλου
        this.Hide(); // Κρύψιμο της παρούσας φόρμας

        // Δημιουργία χειριστηρίου για τον έλεγχο κλεισίματος της φόρμας
        newEmp.FormClosing += KleisimoFormas;
    }

    private void KleisimoFormas(object sender, FormClosingEventArgs e)
    {
        // Καθαρισμός λιστών
        ListOfContracts.Clear();
        ListOfSkills.Clear();

        // Καθαρισμός λιστών διεπαφής
        lstAvailableSkills.Items.Clear();
        lstEmployeeSkills.Items.Clear();

        frmEmployee_Load(this, e);
        this.Show(); // Εμφάνιση κύριας φόρμας, όταν κλείσει η φόρμα απαιτήσεων
προγράμματος εργασίας
    }

    private void cbSimvolaio_SelectedIndexChanged(object sender, EventArgs e)
    {
        if(cbSimvolaio.SelectedItem.ToString() == "Νέο συμβόλαιο")
        {
            /* Δημιουργία και φόρτωση νέας φόρμας σχετική με την καταχώρηση στη
βάση δεδομένων
             * όλων των απαιτήσεων για τη δημιουργία του προγράμματος εργασίας. */
            frmContracts Requirements = new frmContracts();
            Requirements.επιστροφήΣτοΚύριοΜενούToolStripMenuItem.Text =
"Επιστροφή στην καταχώρηση υπαλλήλου";
            Requirements.Show(); // Εμφάνιση της φόρμας
            this.Hide(); // Απόκρυψη της φόρμας του κύριου μενού
```

```
            // Δημιουργία χειριστηρίου για τον έλεγχο κλεισίματος της φόρμας
            Requirements.FormClosing += KleisimoFormas;
        }
    }
  }
}
```

ΦΟΡΜΑ frmEmpWork

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Data.Entity;
using System.Linq;
using System.Windows.Forms;
using NurseRostering.Database;
using NurseRostering.InputData;


namespace NurseRostering.GUI
{
  public partial class frmEmpWork : Form
  {
      Modules module = new Modules();
      Employee empl = new Employee();
      NurseRosteringDB nrdb = new NurseRosteringDB();

      Day_On_Request DOnReq;
      Day_Off_Request DOffReq;
      Shift_On_Request SOnReq;
      Shift_Off_Request SOffReq;

      private string komma = ", ";
      private string aitima_ergasias;
      private string aitima_repo;
      private string aitima_vardias_ergasias = "";
      private string aitima_vardias_repo = "";

      // Λίστα στην οποία θα τοποθετηθούν όλοι οι υπάλληλοι που θα "έρχονται" από
τη βάση δεδομένων
      private List<Employee> EmployeeList { get; set; } = new List<Employee>();

      /* Μεταβλητή η οποία θα φέρει το ID του νοσοκόμου και θα ενημερώνεται κάθε
φορά που
       * ο χρήστης θα επιλέγει ένα νοσοκόμο από αυτούς που εμφανίζονται στη
σχετική λίστα */
      private string idNosokomou = "-1";
```

```
        public frmEmpWork()
        {
            InitializeComponent();
        }

        #region ΔΙΑΧΕΙΡΙΣΗ ΤΩΝ ΗΜΕΡΟΜΗΝΙΩΝ ΠΟΥ ΕΠΙΛΕΓΕΙ Ο
ΝΟΣΟΚΟΜΟΣ ΝΑ ΔΟΥΛΕΨΕΙ

        // Επιλογή αιτήματος από τη λίστα και προσθήκη του χαρακτήρα ; για αναζήτησή
του μέσα στη λίστα των σχετικών αντικειμένων
        private void lstWorking_Days_SelectedIndexChanged(object sender, EventArgs
e)
        {
            if(lstWorking_Days.SelectedIndex > -1)
            {
                aitima_ergasias = lstWorking_Days.SelectedItem + ";";
            }

        }

        // Προσθήκη της επιλεγμένης ημερομηνίας στη λίστα εργασίας
        private void btnAdd_Work_Day_Click(object sender, EventArgs e)
        {
            /* Στοιχεία αιτήματος του νοσοκόμου για επιθυμητή ημέρα εργασίας, τα οποια
             * θα χρησημοποιηθούν για την αναζήτηση ήδη υπάρχοντος αιτήματος και
προς
             * αποφυγή διπλών εγγραφών. */
            string stoixeiaAitimatos = idNosokomou + komma
                + mc_Work_Day.SelectionRange.Start.ToString("yyyy-MM-dd") + komma
                + nudDayOnReq.Value.ToString();

            if(int.Parse(idNosokomou.Trim()) != -1) // Εάν έχει επιλεγεί κάποιος
νοσοκόμος από τη σχετική λίστα
            {
                // Και εάν το προς εισαγωγή αίτημα, δεν έχει καταχωρηθεί ήδη
                if (module.CheckForDoubleRecords(stoixeiaAitimatos, lstWorking_Days))
                {
                    int nurseID = int.Parse(idNosokomou);
                    if((from x in nrdb.Day_On_Requests
                            where (x.date == mc_Work_Day.SelectionRange.Start.Date &&
x.employeeID == nurseID)
                            select x).Count() > 0)
                    {
                        MessageBox.Show("Υπάρχει ήδη μια εγγραφή αυτού του νοσοκόμου
στη συγκεκιμένη ημερομηνία καταχωρημένη στη βάση δεδομένων.",
"NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Error);
                        return;
                    }

                    DOnReq = new Day_On_Request();// Δημιουργία αιτήματος ημέρας
επιθυμητής εργασίας

                    DOnReq.date = mc_Work_Day.SelectionRange.Start.Date;
```

```
                DOnReq.weight = nudDayOnReq.Value.ToString();
                DOnReq.employeeID = int.Parse(idNosokomou);

                // Προσθήκη του αιτήματος ημέρας εργασίας στη λίστα σχετικών
αντικειμένων
                empl.Empl_Des_Days_On.Add(DOnReq);

                // Προσθήκη του αιτήματος ημέρας εργασίας στη λίστα αιτημάτων που
βλέπει ο χρήστης.
                lstWorking_Days.Items.Add(DOnReq.ToString().TrimEnd(';'));
            }
        }
        else
        {
            // Ενημέρωση του χρήστη σχετικά με το σφάλμα που προέκυψε.
            MessageBox.Show("Δεν έχει επιλεχθεί κάποιος νοσοκόμος για να
υποβάλλει το αίτημα.", "NurseRostering", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            return; // Τερματιμός εκτέλεσης μεθόδου
        }

    }

    /* Αφαίρεση της επιλεγμένης ημερομηνίας από τη λίστα εργασίας
     * αλλά και από τη σχετική λίστα αντικειμένων */
    private void btnDel_Work_Day_Click(object sender, EventArgs e)
    {
        if(lstWorking_Days.SelectedIndex > -1)
        {
            if (aitima_ergasias.Trim() != "") // Εάν έχει επιλεγεί κάποιο αίτημα από τη
σχετική λίστα
            {
                // Αναζήτηση του αιτήματος με τα συγκεκριμένα στοιχεία
                for(int i = 0; i < empl.Empl_Des_Days_On.Count; i++)
                {
                    // Εάν βρεθεί ένα αίτημα με τα συγκεκριμένα στοιχεία
                    if (empl.Empl_Des_Days_On[i].ToString() == aitima_ergasias)
                    {
                        // Διαγραφή αυτού του αιτήματος
                        empl.Empl_Des_Days_On.RemoveAt(i);
                    }
                }
            }
            lstWorking_Days.Items.Remove(lstWorking_Days.SelectedItem);
        }
        else
        {
            // Ενημέρωση του χρήστη σχετικά με το σφάλμα που προέκυψε.
            MessageBox.Show("Δεν έχει επιλεχθεί κάποιο αίτημα για διαγραφή.",
"NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Error);
            return; // Τερματιμός εκτέλεσης μεθόδου
        }
```

```
        }

        // Καθαρισμός της λίστας εργασίας και της σχετικής λίστα αντικειμένων
        private void btnClear_Work_Days_Click(object sender, EventArgs e)
        {
            lstWorking_Days.Items.Clear();
            empl.Empl_Des_Days_On.Clear();
        }

        #endregion

        #region ΔΙΑΧΕΙΡΙΣΗ ΤΩΝ ΗΜΕΡΟΜΗΝΙΩΝ ΑΝΑΠΑΥΣΗΣ ΠΟΥ ΕΠΙΛΕΓΕΙ
Ο ΝΟΣΟΚΟΜΟΣ

        // Επιλογή αιτήματος από τη λίστα και προσθήκη του χαρακτήρα ; για αναζήτησή
του μέσα στη λίστα των σχετικών αντικειμένων
        private void lstRepo_Days_SelectedIndexChanged(object sender, EventArgs e)
        {
            if(lstRepo_Days.SelectedIndex > -1)
            {
                aitima_repo = lstRepo_Days.SelectedItem + ";";
            }
        }

        // Προσθήκη της επιλεγμένης ημερομηνίας ανάπαυσης στη λίστα
        private void btnAdd_Repo_Day_Click(object sender, EventArgs e)
        {
            /* Στοιχεία αιτήματος του νοσοκόμου για επιθυμητή ημέρα ανάπαυσης, τα
οποια
             * θα χρησημοποιηθούν για την αναζήτηση ήδη υπάρχοντος αιτήματος και
προς
             * αποφυγή διπλών εγγραφών. */
            string stoixeiaAitimatos = idNosokomou + komma
                + mc_Repo_Day.SelectionRange.Start.ToString("yyyy-MM-dd") + komma
                + nudDayOffReq.Value.ToString();

            if (int.Parse(idNosokomou.Trim()) != -1) // Εάν έχει επιλεγεί κάποιος
νοσοκόμος από τη σχετική λίστα
            {
                // Και εάν το προς εισαγωγή αίτημα, δεν έχει καταχωρηθεί ήδη
                if (module.CheckForDoubleRecords(stoixeiaAitimatos, lstRepo_Days))
                {
                    int nurseID = int.Parse(idNosokomou);
                    if ((from x in nrdb.Day_Off_Requests
                        where (x.date == mc_Repo_Day.SelectionRange.Start.Date &&
x.employeeID == nurseID)
                        select x).Count() > 0)
                    {
                        MessageBox.Show("Υπάρχει ήδη μια εγγραφή αυτού του νοσοκόμου
στη συγκεκιμένη ημερομηνία καταχωρημένη στη βάση δεδομένων.",
"NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Error);
                        return;
```

```
                }
                DOffReq = new Day_Off_Request(); // Δημιουργία αιτήματος ημέρας
επιθυμητής ανάπαυσης

                // Στοιχεία αιτήματος
                DOffReq.date = mc_Repo_Day.SelectionRange.Start.Date;
                DOffReq.weight = nudDayOffReq.Value.ToString();
                DOffReq.employeeID = int.Parse(idNosokomou);

                // Προσθήκη του αιτήματος ημέρας ανάπαυσης στη λίστα σχετικών
αντικειμένων.
                empl.Empl_Des_Days_Off.Add(DOffReq);

                // Προσθήκη του αιτήματος ημέρας ανάπαυσης στη λίστα αιτημάτων που
βλέπει ο χρήστης.
                lstRepo_Days.Items.Add(DOffReq.ToString().TrimEnd(';'));
            }
        }
        else
        {
            // Ενημέρωση του χρήστη σχετικά με το σφάλμα που προέκυψε.
            MessageBox.Show("Δεν έχει επιλεχθεί κάποιος νοσοκόμος για να
υποβάλλει το αίτημα.", "NurseRostering", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            return; // Τερματιμός εκτέλεσης μεθόδου
        }
    }

    // Αφαίρεση της επιλεγμένης ημερομηνίας ανάπαυσης από τη λίστα
    private void btnDel_Repo_Day_Click(object sender, EventArgs e)
    {
        if(lstRepo_Days.SelectedIndex > -1)
        {
            if (aitima_repo.Trim() != "") // Εάν έχει επιλεγεί κάποιο αίτημα από τη
σχετική λίστα
            {
                // Αναζήτηση του αιτήματος με τα συγκεκριμένα στοιχεία
                for (int i = 0; i < empl.Empl_Des_Days_Off.Count; i++)
                {
                    // Εάν βρεθεί ένα αίτημα με τα συγκεκριμένα στοιχεία
                    if (empl.Empl_Des_Days_Off[i].ToString() == aitima_repo)
                    {
                        // Διαγραφή αυτού του αιτήματος
                        empl.Empl_Des_Days_Off.RemoveAt(i);
                    }
                }
            }
            lstRepo_Days.Items.Remove(lstRepo_Days.SelectedItem);
        }
        else
        {
            // Ενημέρωση του χρήστη σχετικά με το σφάλμα που προέκυψε.
```

```
            MessageBox.Show("Δεν έχει επιλεχθεί κάποιο αίτημα για διαγραφή.",
"NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Error);
            return; // Τερματιμός εκτέλεσης μεθόδου
        }

    }

    // Καθαρισμός της λίστας ανάπαυσης
    private void btnClear_Repo_Days_Click(object sender, EventArgs e)
    {
        lstRepo_Days.Items.Clear();
        empl.Empl_Des_Days_Off.Clear();
    }

    #endregion

    #region ΔΙΑΧΕΙΡΙΣΗ ΤΩΝ ΒΑΡΔΙΩΝ ΠΟΥ ΕΠΙΛΕΓΕΙ ΝΑ ΕΡΓΑΣΤΕΙ Ο
ΝΟΣΟΚΟΜΟΣ

    // Επιλογή αιτήματος από τη λίστα και προσθήκη του χαρακτήρα ; για αναζήτησή
του μέσα στη λίστα των σχετικών αντικειμένων
    private void lst_Req_Shift_SelectedIndexChanged(object sender, EventArgs e)
    {
        if(lst_Req_Shift.SelectedIndex > -1)
        {
            aitima_vardias_ergasias = lst_Req_Shift.SelectedItem + ";";
        }
    }

    // Προσθήκη της επιλεγμένης ημέρας εργασίας σε συγκεκριμένη βάρδια στη
λίστα
    private void btnAdd_Work_Shift_Click(object sender, EventArgs e)
    {
        // Εάν έχει πραγματοποιηθεί επιλογή κάποιας βάρδιας
        if (cbShiftOn.SelectedIndex > -1)
        {
            /* Στοιχεία αιτήματος του νοσοκόμου για επιθυμητή ημέρα εργασίας σε
συγκεκριμένη βάρδια, τα οποια
             * θα χρησημοποιηθούν για την αναζήτηση ήδη υπάρχοντος αιτήματος και
προς
             * αποφυγή διπλών εγγραφών. */
            string stoixeiaAitimatos = idNosokomou + komma
                + mc_Requested_Shift.SelectionRange.Start.ToString("yyyy-MM-dd") +
komma
                + cbShiftOn.SelectedItem.ToString() + komma
                + nudDesWorkShifts.Value.ToString();

            if (int.Parse(idNosokomou.Trim()) != -1) // Εάν έχει επιλεγεί κάποιος
νοσοκόμος από τη σχετική λίστα
            {
                // Και εάν το προς εισαγωγή αίτημα, δεν έχει καταχωρηθεί ήδη
                if (module.CheckForDoubleRecords(stoixeiaAitimatos, lst_Req_Shift))
                {
```

```
                    SOnReq = new Shift_On_Request(); // Δημιουργία αιτήματος ημέρας
επιθυμητής εργασίας σε συγκεκριμένη βάρδια

                    // Στοιχεία αιτήματος
                    SOnReq.date = mc_Requested_Shift.SelectionRange.Start.Date;
                    SOnReq.ShiftTypeID = cbShiftOn.SelectedItem.ToString();
                    SOnReq.weight = nudDesWorkShifts.Value.ToString();
                    SOnReq.employeeID = int.Parse(idNosokomou);

                    // Προσθήκη του αιτήματος ημέρας εργασίας σε συγκεκριμένη βάρδια
στη λίστα σχετικών αντικειμένων.
                    empl.Empl_Des_Shifts_On.Add(SOnReq);

                    // Προσθήκη του αιτήματος ημέρας εργασίας σε συγκεκριμένη βάρδια
στη λίστα αιτημάτων που βλέπει ο χρήστης.
                    lst_Req_Shift.Items.Add(SOnReq.ToString().TrimEnd(';'));
                }
            }
            else // εάν δεν έχει επιλεχθεί κάποιος νοσοκόμος
            {
                // Ενημέρωση του χρήστη σχετικά με το σφάλμα που προέκυψε.
                MessageBox.Show("Δεν έχει επιλεχθεί κάποιος νοσοκόμος για να
υποβάλλει το αίτημα.", "NurseRostering", MessageBoxButtons.OK,
MessageBoxIcon.Error);
                return; // Τερματιμός εκτέλεσης μεθόδου
            }
        }
        else // εάν δεν έχει επιλεχθεί κάποια βάρδια
        {
            // Ενημέρωση του χρήστη με σχετικό μήνυμα
            MessageBox.Show("Δεν έχει επιλεχθεί μια βάρδια.", "NurseRostering",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            cbShiftOn.Focus(); // Μεταφορά του ελέγχου στο αντικειμενο επιλογής της
βάρδιας
        }
    }

    // Αφαίρεση του επιλεγμένου αιτήματος εργασίας σε συγκεκριμένη ημέρα και
βάρδια από τη λίστα αιτημάτων
    private void btnDel_Work_Shift_Click(object sender, EventArgs e)
    {
        if (aitima_vardias_ergasias.Trim() != "") // Εάν έχει επιλεγεί κάποιο αίτημα
από τη σχετική λίστα
        {
            // Αναζήτηση του αιτήματος με τα συγκεκριμένα στοιχεία
            for (int i = 0; i < empl.Empl_Des_Shifts_On.Count; i++)
            {
                // Εάν βρεθεί ένα αίτημα με τα συγκεκριμένα στοιχεία
                if (empl.Empl_Des_Shifts_On[i].ToString() == aitima_vardias_ergasias)
                {
                    // Διαγραφή αυτού του αιτήματος
                    empl.Empl_Des_Shifts_On.RemoveAt(i);
                }
```

```
            }
        }
        else
        {
            // Ενημέρωση του χρήστη σχετικά με το σφάλμα που προέκυψε.
            MessageBox.Show("Δεν έχει επιλεχθεί κάποιο αίτημα για διαγραφή.",
"NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Error);
            return; // Τερματιμός εκτέλεσης μεθόδου
        }
        // Διαγραφή του αιτήματος από τη λίστα
        lst_Req_Shift.Items.Remove(lst_Req_Shift.SelectedItem);
    }

    // Καθαρισμός της λίστας αιτημάτων εργασίας σε συγκεκριμένη ημέρα και
βάρδια
    private void btnClear_Work_Shift_Click(object sender, EventArgs e)
    {
        lst_Req_Shift.Items.Clear();
        empl.Empl_Des_Shifts_On.Clear();
    }

    #endregion

    #region ΔΙΑΧΕΙΡΙΣΗ ΤΩΝ ΒΑΡΔΙΩΝ ΠΟΥ ΕΠΙΛΕΓΕΙ ΝΑ ΜΗΝ ΕΡΓΑΣΤΕΙ
Ο ΝΟΣΟΚΟΜΟΣ

    // Επιλογή αιτήματος από τη λίστα και προσθήκη του χαρακτήρα ; για αναζήτησή
του μέσα στη λίστα των σχετικών αντικειμένων
    private void lst_Repo_Shift_SelectedIndexChanged(object sender, EventArgs e)
    {
        if(lst_Repo_Shift.SelectedIndex > -1)
        {
            aitima_vardias_repo = lst_Repo_Shift.SelectedItem + ";";
        }
    }

    // Προσθήκη της επιλεγμένης ημέρας εργασίας σε συγκεκριμένη βάρδια στη
λίστα
    private void btnAdd_Repo_Shift_Click(object sender, EventArgs e)
    {
        // Εάν έχει επιλεχθεί κάποια βάρδια
        if (cbShiftOff.SelectedIndex > -1)
        {
            /* Στοιχεία αιτήματος του νοσοκόμου για επιθυμητή ημέρα ανάπαυσης σε
συγκεκριμένη βάρδια, τα οποια
             * θα χρησημοποιηθούν για την αναζήτηση ήδη υπάρχοντος αιτήματος και
προς
             * αποφυγή διπλών εγγραφών. */
            string stoixeiaAitimatos = idNosokomou + komma
                + mc_Repo_Shift.SelectionRange.Start.ToString("yyyy-MM-dd") +
komma
                + cbShiftOff.SelectedItem.ToString() + komma
                + nudUnwantWorkShifts.Value.ToString();
```

```
            if (int.Parse(idNosokomou.Trim()) != -1) // Εάν έχει επιλεγεί κάποιος
νοσοκόμος από τη σχετική λίστα
            {
                // Και εάν το προς εισαγωγή αίτημα, δεν έχει καταχωρηθεί ήδη
                if (module.CheckForDoubleRecords(stoixeiaAitimatos, lst_Repo_Shift))
                {
                    SOffReq = new Shift_Off_Request(); // Δημιουργία αιτήματος ημέρας
επιθυμητής εργασίας σε συγκεκριμένη βάρδια

                    // Στοιχεία αιτήματος
                    SOffReq.date = mc_Repo_Shift.SelectionRange.Start.Date;
                    SOffReq.ShiftTypeID = cbShiftOff.SelectedItem.ToString();
                    SOffReq.weight = nudUnwantWorkShifts.Value.ToString();
                    SOffReq.employeeID = int.Parse(idNosokomou);

                    // Προσθήκη του αιτήματος ημέρας εργασίας σε συγκεκριμένη βάρδια
στη λίστα σχετικών αντικειμένων.
                    empl.Empl_Des_Shifts_Off.Add(SOffReq);

                    // Προσθήκη του αιτήματος ημέρας εργασίας σε συγκεκριμένη βάρδια
στη λίστα αιτημάτων που βλέπει ο χρήστης.
                    lst_Repo_Shift.Items.Add(SOffReq.ToString().TrimEnd(';'));
                }
            }
            else
            {
                // Ενημέρωση του χρήστη σχετικά με το σφάλμα που προέκυψε.
                MessageBox.Show("Δεν έχει επιλεχθεί κάποιος νοσοκόμος για να
υποβάλλει το αίτημα.", "NurseRostering", MessageBoxButtons.OK,
MessageBoxIcon.Error);
                return; // Τερματιμός εκτέλεσης μεθόδου
            }
        }
        else // εάν δεν έχει επιλεχθεί κάποια βάρδια
        {
            // Ενημέρωση του χρήστη με σχετικό μήνυμα
            MessageBox.Show("Δεν έχει επιλεχθεί μια βάρδια.", "NurseRostering",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            cbShiftOff.Focus(); // Μεταφορά του ελέγχου στο αντικειμενο επιλογής της
βάρδιας
        }
    }

    // Αφαίρεση του επιλεγμένου αιτήματος εργασίας σε συγκεκριμένη ημέρα και
βάρδια από τη λίστα αιτημάτων
    private void btnDel_Repo_Shift_Click(object sender, EventArgs e)
    {
        if (aitima_vardias_repo.Trim() != "") // Εάν έχει επιλεγεί κάποιο αίτημα από τη
σχετική λίστα
        {
            // Αναζήτηση του αιτήματος με τα συγκεκριμένα στοιχεία
            for (int i = 0; i < empl.Empl_Des_Shifts_Off.Count; i++)
```

```
                {
                    // Εάν βρεθεί ένα αίτημα με τα συγκεκριμένα στοιχεία
                    if (empl.Empl_Des_Shifts_Off[i].ToString() == aitima_vardias_repo)
                    {
                        // Διαγραφή αυτού του αιτήματος
                        empl.Empl_Des_Shifts_Off.RemoveAt(i);
                    }
                }
            }
            else
            {
                // Ενημέρωση του χρήστη σχετικά με το σφάλμα που προέκυψε.
                MessageBox.Show("Δεν έχει επιλεχθεί κάποιο αίτημα για διαγραφή.",
"NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Error);
                return; // Τερματιμός εκτέλεσης μεθόδου
            }
            // Διαγραφή του αιτήματος από τη λίστα
            lst_Repo_Shift.Items.Remove(lst_Repo_Shift.SelectedItem);
        }

        // Καθαρισμός της λίστας αιτημάτων εργασίας σε συγκεκριμένη ημέρα και
βάρδια
        private void btnClear_Repo_Shift_Click(object sender, EventArgs e)
        {
            lst_Repo_Shift.Items.Clear();
            empl.Empl_Des_Shifts_Off.Clear();
        }

        #endregion

        #region ΒΑΣΗ ΔΕΔΟΜΕΝΩΝ

        // Κουμπί καταχώρησης δεδομένων και επιστροφής στο μενού διαχείρισης
προσωπικού.
        private void btnSubmit_Click(object sender, EventArgs e)
        {
            if(!(lstWorking_Days.Items.Count > 0 ||
                lstRepo_Days.Items.Count > 0 ||
                lst_Repo_Shift.Items.Count > 0 ||
                lst_Req_Shift.Items.Count > 0))
            {
                MessageBox.Show("Δεν έχει εισαχθεί καμία επιθυμία κάποιου νοσοκόμου
για καταχώρηση στη βάση δεδομένων. Η ενέργεια δεν μπορεί να συνεχιστεί",
                    "NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Error);
                return;
            }
            try
            {
                // Επιλεγμένες ημέρες εργασίας
                if (empl.Empl_Des_Days_On.Count > 0)
                {
                    // Προετοιμασία αιτήσεων εργασίας σε επιθυμητή ημέρα, για εγγραφή
στη βάση δεδομένων
```

```
                foreach (var item in empl.Empl_Des_Days_On)
                {
                    nrdb.Entry(item).State = EntityState.Added;
                }
            }

            // Επιλεγμένες ημέρες ανάπαυσης
            if (empl.Empl_Des_Days_Off.Count > 0)
            {
                // Προετοιμασία αιτήσεων ανάπαυσης σε επιθυμητή ημέρα, για εγγραφή
στη βάση δεδομένων
                foreach (var item in empl.Empl_Des_Days_Off)
                {
                    nrdb.Entry(item).State = EntityState.Added;
                }
            }

            // Επιλεγμένες βάρδιες εργασίας
            if (empl.Empl_Des_Shifts_On.Count > 0)
            {
                // Προετοιμασία αιτήσεων εργασίας σε επιθυμητή ημέρα και βάρδια, για
εγγραφή στη βάση δεδομένων
                foreach (var item in empl.Empl_Des_Shifts_On)
                {
                    nrdb.Entry(item).State = EntityState.Added;
                }
            }

            // Επιλεγμένες βάρδιες ανάπαυσης
            if (empl.Empl_Des_Shifts_Off.Count > 0)
            {
                // Προετοιμασία αιτήσεων ανάπαυσης σε επιθυμητή ημέρα και βάρδια,
για εγγραφή στη βάση δεδομένων
                foreach (var item in empl.Empl_Des_Shifts_Off)
                {
                    nrdb.Entry(item).State = EntityState.Added;
                }
            }
            nrdb.Entry(empl).State = EntityState.Modified;
            nrdb.SaveChanges(); // Εγγραφή στη βάση δεδομένων
            lstNosokomoi.Enabled = true;  // Επανενεργοποίηση πλαισίου επιλογής
νοσοκόμων
            // Ενεργοποίηση εκ νέου του στοιχείου αυτού για αποφυγή πολλαπλών
νοσοκόμων σε κάθε καταχώρηση
            lstNosokomoi.Enabled = true;
            MessageBox.Show("Η εγγραφή καταχωρήθηκε επιτυχώς.",
"NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
        catch(Exception ex) // Σύλληψη σφάλματος που προέκυψε
        {
            // και ενημέρωση του χρήστη σχετικά
```

```
            MessageBox.Show("Προέκυψε κάποιο απροσδόκητο σφάλμα. Η εγγραφή
δεν καταχωρήθηκε. Σφάλμα: " + ex.ToString(), "NurseRostering",
MessageBoxButtons.OK, MessageBoxIcon.Error);
        }

    }

    // Λήψη δεδομένων νοσοκόμων από τη βάση δεδομένων και τοποθέτησή τους
στη λίστα νοσοκόμων της διεπαφής
    private void EmfanisiNosokomwn ()
    {
        NurseRosteringDB nrdb = new NurseRosteringDB();
        // Λήψη όλων των υπαλλήλων και εμφάνιση στη λίστα νοσοκόμων, του
αναγνωριστικού και του ονοματεπώνυμού τους
        EmployeeList = (from emp in nrdb.Employees select emp).OrderBy(x =>
x.ID).ToList();
        foreach (var item in EmployeeList)
        {
            // Προβολή μόνο του αναγνωριστικού και του ονοματεπώνυμου του
νοσοκόμου στη λίστα νοσοκόμων της διεπαφής
            lstNosokomoi.Items.Add(item.ID.ToString() + " " + item.name.ToString());
        }
    }

    #endregion

    // Ενέργειες που πραγματοποιούνται κατά την επιλογή ενός νοσοκόμου από τη
σχετική λίστα
    private void lstNosokomoi_SelectedIndexChanged(object sender, EventArgs e)
    {
        if(lstNosokomoi.SelectedIndex > -1)
        {
            // Μεταβλητές
            int search = -1; // Θέση αναζητούμενου χαρακτήρα
            string name = ""; // Ονοματεπώνυμο υπαλλήλου. Εξάγεται σε τύπο string
από τη λίστα νοσοκόμων

            /* Εύρεση της θέσης πρώτου κενού διαστήματος για διαχωρισμό του
             * αναγνωριστικού νοσοκόμου από το ονοματεπώνυμο αυτού */
            search = lstNosokomoi.SelectedItem.ToString().IndexOf(" ");

            // Εξαγωγή του αναγνωριστικού του νοσοκόμου από τη λίστα
            idNosokomou = lstNosokomoi.SelectedItem.ToString().Substring(0,
search);

            // Εξαγωγή του ονοματεπώνυμου του νοσοκόμου από τη λίστα
            name = lstNosokomoi.SelectedItem.ToString().Substring(search + 1);

            // Δημιουργία αντικειμένου του νοσοκόμου
            empl = EmployeeList.Find(x => x.ID == int.Parse(idNosokomou) &&
x.name == name);
```

```
            // Απενεργοποίηση του στοιχείου αυτού για αποφυγή πολλαπλών
νοσοκόμων σε κάθε καταχώρηση
            lstNosokomoi.Enabled = false;
        }

    }


    private void button1_Click(object sender, EventArgs e)
    {
        this.Close(); // Κλείσιμο της παρούσας φόρμας και εμφάνιση αυτής που την
κάλεσε.
        GC.Collect(); // Συλλογή "σκουπιδιών"
    }

    // Κατα την εκκίνηση της φόρμας, τροφοδοσία της λίστας των νοσοκόμων με
μερικά στοιχεία τους (ID και ονοματεπώνυμο)
    private void frmEmpWork_Load(object sender, EventArgs e)
    {
        EmfanisiNosokomwn(); // Νοσοκόμοι
        lstNosokomoi.Refresh();

        // Βάρδιες που επιθυμεί ο νοσοκόμος να εργαστεί
        module.EmfanisiVardiwn(cbShiftOn);
        cbShiftOn.Refresh();

        // Βάρδιες που ΔΕΝ επιθυμεί ο νοσοκόμος να εργαστεί
        module.EmfanisiVardiwn(cbShiftOff);
        cbShiftOff.Refresh();
    }

    private void btnCorrect_Click(object sender, EventArgs e)
    {
        // Ενημέρωση του χρήστη για την ύπαρξη κάποιων εγγραφών στις λίστες
        if(lstWorking_Days.Items.Count > 0 || lstRepo_Days.Items.Count > 0 ||
lst_Req_Shift.Items.Count > 0 || lst_Repo_Shift.Items.Count > 0)
        {
            MessageBox.Show("Υπάρχουν εγγραφές στις λίστες που αναμένουν για
καταχώρηση. Δεν είναι δυνατή η αλλαγή του επιλεγμένου νοσοκόμου.",
"NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
        else
        {
            // Ενεργοποίηση εκ νέου του στοιχείου αυτού για αποφυγή πολλαπλών
νοσοκόμων σε κάθε καταχώρηση
            lstNosokomoi.Enabled = true;
        }
    }

    private void btnNewShift_Click(object sender, EventArgs e)
    {
        // Δημιουργία νέας φόρμας καταχώρησης νέας βάρδιας
        frmNewShift newShift = new frmNewShift();
```

```
            newShift.Show(); // Εμφάνιση φόρμας καταχώρησης νέας βάρδιας
            newShift.btnReturn.Text = "Επιστροφή στις επιθυμίες εργασίας νοσοκόμου";
            this.Hide(); // Κρύψιμο της παρούσας φόρμας

            // Δημιουργία χειριστηρίου για τον έλεγχο κλεισίματος της φόρμας
            newShift.FormClosing += KleisimoFormas;
        }

        private void KleisimoFormas(object sender, FormClosingEventArgs e)
        {
            lstNosokomoi.Items.Clear();
            cbShiftOn.Items.Clear();
            cbShiftOff.Items.Clear();
            frmEmpWork_Load(this, e);

            this.Show(); // Εμφάνιση κύριας φόρμας, όταν κλείσει η φόρμα απαιτήσεων
προγράμματος εργασίας
        }

        private void btnNewEmp_Click(object sender, EventArgs e)
        {
            // Δημιουργία νέας φόρμας καταχώρησης νέου υπαλλήλου
            frmEmployee newEmp = new frmEmployee();

            newEmp.btnExit.Text = "Επιστροφή στις επιθυμίες εργασίας υπαλλήλου";
            newEmp.Show(); // Εμφάνιση φόρμας καταχώρησης νέου υπαλλήλου
            this.Hide(); // Κρύψιμο της παρούσας φόρμας

            // Δημιουργία χειριστηρίου για τον έλεγχο κλεισίματος της φόρμας
            newEmp.FormClosing += KleisimoFormas;
        }
    }
}
```

ΦΟΡΜΑ frmMain

```
using System;
using System.Drawing;
using System.Windows.Forms;

namespace NurseRostering.GUI
{
    public partial class frmMain : Form
    {
        public frmMain()
        {
            InitializeComponent();
            gbEmpMenu.Location = new Point(12, 12); // Τοποθέτηση πλαισίου μενού
υπαλλήλων στην αριστερή επάνω γωνία της φόρμας
```

```
            gbShiftMenu.Location = new Point(12, 12); // Τοποθέτηση πλαισίου μενού
βαρδιών στην αριστερή επάνω γωνία της φόρμας
            gbManageRoster.Location = new Point(12, 12); // Τοποθέτηση πλαισίου μενού
διαχείρισης προγράμματος στην αριστερή επάνω γωνία της φόρμας
            gbMainMenu.Location = new Point(12, 12); // Τοποθέτηση πλαισίου κύριου
μενού στην αριστερή επάνω γωνία της φόρμας
        }

        #region ΕΝΕΡΓΕΙΕΣ ΠΟΥ ΠΡΑΓΜΑΤΟΠΟΙΟΥΝΤΑΙ ΣΤΟ ΚΥΡΙΟ ΜΕΝΟΥ

        /**************************************************
         **************************************************
         **                                              **
         ** ΕΝΕΡΓΕΙΕΣ ΠΟΥ ΠΡΑΓΜΑΤΟΠΟΙΟΥΝΤΑΙ ΣΤΟ ΚΥΡΙΟ ΜΕΝΟΥ **
         **                                              **
         **************************************************
         **************************************************/

        // Κουμπί διαχείρισης υπαλλήλων
        private void btnEmp_Click(object sender, EventArgs e)
        {
            gbMainMenu.Visible = false; // Απόκρυψη του γκρουπ του κύριου μενού
            gbEmpMenu.Visible = true; // Εμφάνιση του μενού επιλογών σχετικές με τον
υπάλληλο

        }

            // Κουμπί διαχείρισης βαρδιών
        private void btnShift_Click(object sender, EventArgs e)
        {
            gbMainMenu.Visible = false; // Απόκρυψη του γκρουπ του κύριου μενού
            gbShiftMenu.Visible = true; // Εμφάνιση του μενού επιλογών σχετικές με τις
βάρδιες
        }

            // Κουμπί τερματισμού του προγράμματος
        private void btnExit_Click(object sender, EventArgs e)
        {
            GC.Collect(); // Συλλογή "σκουπιδιών"
            Application.Exit(); // Τερματισμός της λειτουργίας του προγράμματος
        }

        #endregion

        #region ΕΝΕΡΓΕΙΕΣ ΠΟΥ ΠΡΑΓΜΑΤΟΠΟΙΟΥΝΤΑΙ ΣΤΟ ΜΕΝΟΥ ΣΧΕΤΙΚΑ
ΜΕ ΤΟ ΠΡΟΣΩΠΙΚΟ


/*********************************************************************
*

*********************************************************************
*
```

```
        **                                                    **
        ** ΕΝΕΡΓΕΙΕΣ ΠΟΥ ΠΡΑΓΜΑΤΟΠΟΙΟΥΝΤΑΙ ΣΤΟ ΜΕΝΟΥ ΣΧΕΤΙΚΑ ΜΕ
ΤΟ ΠΡΟΣΩΠΙΚΟ **
        **                                                    **

*****************************************************************
*

*****************************************************************
*/


      // Κουμπί επιστροφής στο κύριο μενού
      private void btnReturnEmp_Click(object sender, EventArgs e)
      {
         gbEmpMenu.Visible = false; // Απόκρυψη του μενού επιλογών σχετικές με τον
υπάλληλο
         gbMainMenu.Visible = true; // Εμφάνιση του γκρουπ του κύριου μενού
      }

      // Κουμπί εισαγωγής νέου υπαλλήλου
      private void btnNewEmp_Click(object sender, EventArgs e)
      {
         // Δημιουργία νέας φόρμας καταχώρησης νέου υπαλλήλου
         frmEmployee newEmp = new frmEmployee();

         newEmp.Show(); // Εμφάνιση φόρμας καταχώρησης νέου υπαλλήλου
         this.Hide(); // Κρύψιμο της παρούσας φόρμας

         // Δημιουργία χειριστηρίου για τον έλεγχο κλεισίματος της φόρμας
         newEmp.FormClosing += KleisimoFormas;
      }

      // Κουμπί επιθυμητής εργασίας και ανάπαυσης του υπαλλήλου
      private void btnDesiresEmp_Click(object sender, EventArgs e)
      {
         // Δημιουργία νέας φόρμας καταχώρησης επιθυμητής εργασίας και ανάπαυσης
του υπαλλήλου
         frmEmpWork desiredWork = new frmEmpWork();

         desiredWork.Show(); // Εμφάνιση φόρμας καταχώρησης επιθυμητής εργασίας ή
ανάπαυσης του υπαλλήλου
         this.Hide(); // Κρύψιμο της παρούσας φόρμας

         // Δημιουργία χειριστηρίου για τον έλεγχο κλεισίματος της φόρμας
         desiredWork.FormClosing += KleisimoFormas;
      }

      // Κουμπί εισαγωγής νέου υπαλλήλου
       private void btnSkills_Click(object sender, EventArgs e)
      {
         // Δημιουργία νέας φόρμας καταχώρησης νέου υπαλλήλου
         frmAddSkill newSkill = new frmAddSkill();
```

```
            newSkill.Show(); // Εμφάνιση φόρμας καταχώρησης νέου υπαλλήλου
            this.Hide(); // Κρύψιμο της παρούσας φόρμας

            // Δημιουργία χειριστηρίου για τον έλεγχο κλεισίματος της φόρμας
            newSkill.FormClosing += KleisimoFormas;
        }
        #endregion

        #region ΕΝΕΡΓΕΙΕΣ ΠΟΥ ΠΡΑΓΜΑΤΟΠΟΙΟΥΝΤΑΙ ΣΤΟ ΜΕΝΟΥ ΣΧΕΤΙΚΑ
ΜΕ ΤΙΣ ΒΑΡΔΙΕΣ



/*****************************************************************
*

*****************************************************************
*
         **                                          **
         ** ΕΝΕΡΓΕΙΕΣ ΠΟΥ ΠΡΑΓΜΑΤΟΠΟΙΟΥΝΤΑΙ ΣΤΟ ΜΕΝΟΥ ΣΧΕΤΙΚΑ ΜΕ
ΤΙΣ ΒΑΡΔΙΕΣ  **
         **                                          **

*****************************************************************
*

*****************************************************************
*/


        // Κουμπί επιστροφής στο κύριο μενού
        private void btnReturnShift_Click_1(object sender, EventArgs e)
        {
            gbShiftMenu.Visible = false; // Απόκρυψη του μενού επιλογών σχετικές με τις
βάρδιες
            gbMainMenu.Visible = true; // Εμφάνιση του γκρουπ του κύριου μενού
        }

        private void btnUnwantedPatterns_Click(object sender, EventArgs e)
        {
            // Δημιουργία νέας φόρμας καταχώρησης νέου προτύπου ανεπιθύμητης
εργασίας
            frmUnwantedPatterns newUnwanted = new frmUnwantedPatterns();

            newUnwanted.Show(); // Εμφάνιση φόρμας καταχώρησης νέου προτύπου
ανεπιθύμητης εργασίας
            this.Hide(); // Κρύψιμο της παρούσας φόρμας

            // Δημιουργία χειριστηρίου για τον έλεγχο κλεισίματος της φόρμας
            newUnwanted.FormClosing += KleisimoFormas;
        }
```

```
        private void btnAddShift_Click(object sender, EventArgs e)
        {
            // Δημιουργία νέας φόρμας καταχώρησης νέας βάρδιας
            frmNewShift newShift = new frmNewShift();

            newShift.Show(); // Εμφάνιση φόρμας καταχώρησης νέας βάρδιας
            this.Hide(); // Κρύψιμο της παρούσας φόρμας

            // Δημιουργία χειριστηρίου για τον έλεγχο κλεισίματος της φόρμας
            newShift.FormClosing += KleisimoFormas;
        }

        private void btnGroupUnwanted_Click(object sender, EventArgs e)
        {
            // Δημιουργία νέας φόρμας καταχώρησης νέου γκρουπ πρότυπων ανεπιθύμητης
εργασίας
            frmNewPattern newUnwanted = new frmNewPattern();

            newUnwanted.Show(); // Εμφάνιση φόρμας καταχώρησης νέου γκρουπ
πρότυπων ανεπιθύμητης εργασίας
            this.Hide(); // Κρύψιμο της παρούσας φόρμας

            // Δημιουργία χειριστηρίου για τον έλεγχο κλεισίματος της φόρμας
            newUnwanted.FormClosing += KleisimoFormas;
        }

        #endregion

        #region ΕΝΕΡΓΕΙΕΣ ΠΟΥ ΠΡΑΓΜΑΤΟΠΟΙΟΥΝΤΑΙ ΣΤΟ ΜΕΝΟΥ ΣΧΕΤΙΚΑ
ΜΕ ΤΙΣ ΑΠΑΙΤΗΣΕΙΣ ΤΟΥ ROSTER


/*****************************************************************
***************

*****************************************************************
***************
         **                                                  **
         ** ΕΝΕΡΓΕΙΕΣ ΠΟΥ ΠΡΑΓΜΑΤΟΠΟΙΟΥΝΤΑΙ ΣΤΟ ΜΕΝΟΥ ΣΧΕΤΙΚΑ ΜΕ
ΤΙΣ ΑΠΑΙΤΗΣΕΙΣ ΤΟΥ ROSTER  **
         **                                                  **

*****************************************************************
***************

*****************************************************************
***************/

        // Κουμπί απαιτήσεων του ρόστερ
        private void btnReqRoster_Click(object sender, EventArgs e)
        {
            /* Δημιουργία και φόρτωση νέας φόρμας σχετική με την καταχώρηση στη βάση
δεδομένων
```

```
         * όλων των απαιτήσεων για τη δημιουργία του προγράμματος εργασίας. */
         frmContracts Requirements = new frmContracts();

         Requirements.Show(); // Εμφάνιση της φόρμας
         this.Hide(); // Απόκρυψη της φόρμας του κύριου μενού

         // Δημιουργία χειριστηρίου για τον έλεγχο κλεισίματος της φόρμας
         Requirements.FormClosing += KleisimoFormas;
      }


      #endregion

      #region ΕΝΕΡΓΕΙΕΣ ΠΟΥ ΠΡΑΓΜΑΤΟΠΟΙΟΥΝΤΑΙ ΣΤΟ ΜΕΝΟΥ ΣΧΕΤΙΚΑ
ΜΕ ΤΗ ΔΗΜΙΟΥΡΓΙΑ ΤΟΥ ROSTER


/*****************************************************************
**************

*****************************************************************
**************
      **                                                  **
      ** ΕΝΕΡΓΕΙΕΣ ΠΟΥ ΠΡΑΓΜΑΤΟΠΟΙΟΥΝΤΑΙ ΣΤΟ ΜΕΝΟΥ ΣΧΕΤΙΚΑ ΜΕ
ΤΗ ΔΗΜΙΟΥΡΓΙΑ ΤΟΥ ROSTER  **
      **                                                  **

*****************************************************************
**************

*****************************************************************
**************/

         // Κουμπί δημιουργίας προγράμματος εργασίας
      private void btnCreateRoster_Click(object sender, EventArgs e)
      {
         // Δημιουργία νέας φόρμας για την κατασκευή του προγράμματος εργασίας
         frmSchedulingPeriod newPeriod = new frmSchedulingPeriod();

         newPeriod.Show(); // Εμφάνιση φόρμας δημιουργίας προγράμματος εργασίας
         this.Hide(); // Κρύψιμο της παρούσας φόρμας

         // Δημιουργία χειριστηρίου για τον έλεγχο κλεισίματος της φόρμας
         newPeriod.FormClosing += KleisimoFormas;
      }



      #endregion

      #region ΕΝΕΡΓΕΙΕΣ ΠΟΥ ΠΡΑΓΜΑΤΟΠΟΙΟΥΝΤΑΙ ΣΤΟ ΜΕΝΟΥ ΣΧΕΤΙΚΑ
ΜΕ ΤΗ ΔΙΑΧΕΙΡΙΣΗ ΤΟΥ ROSTER
```

```
/*****************************************************************
*************

*****************************************************************
*************
      **                                                  **
      ** ΕΝΕΡΓΕΙΕΣ ΠΟΥ ΠΡΑΓΜΑΤΟΠΟΙΟΥΝΤΑΙ ΣΤΟ ΜΕΝΟΥ ΣΧΕΤΙΚΑ ΜΕ
ΤΗ ΔΙΑΧΕΙΡΙΣΗ ΤΟΥ ROSTER  **
      **                                                  **

*****************************************************************
*************

*****************************************************************
**************/

      // Κουμπί διαχείρισης προγράμματος εργασίας
      private void btnManageRoster_Click(object sender, EventArgs e)
      {
         /* Δημιουργία και φόρτωση νέας φόρμας σχετική με την καταχώρηση στη βάση
δεδομένων
          * όλων των απαιτήσεων για τη δημιουργία του προγράμματος εργασίας. */
         frmChoice Choice = new frmChoice();

         Choice.Show(); // Εμφάνιση της φόρμας
         this.Hide(); // Απόκρυψη της φόρμας του κύριου μενού

         // Δημιουργία χειριστηρίου για τον έλεγχο κλεισίματος της φόρμας
         Choice.FormClosing += KleisimoFormas;
      }

      #endregion

      #region ΕΝΕΡΓΕΙΕΣ ΠΟΥ ΑΦΟΡΟΥΝ ΓΕΝΙΚΑ ΣΤΗΝ ΚΥΡΙΑ ΦΟΡΜΑ ΚΑΙ
ΟΧΙ ΣΕ ΚΑΠΟΙΑ ΣΥΓΚΕΚΡΙΜΜΕΝΗ

      private void KleisimoFormas(object sender, FormClosingEventArgs e)
      {
         this.Show(); // Εμφάνιση κύριας φόρμας, όταν κλείσει η φόρμα απαιτήσεων
προγράμματος εργασίας
      }



      #endregion

      private void btnReturnRecords_Click(object sender, EventArgs e)
      {
         gbManageRoster.Visible = false; // Απόκρυψη του μενού επιλογών σχετικά με
τις εγγραφές
         gbEmpMenu.Visible = true; // Εμφάνιση του μενού διαχείρισης υπαλλήλων
      }
```

```
        private void btnEditShift_Click(object sender, EventArgs e)
        {
            /* Δημιουργία και φόρτωση νέας φόρμας σχετική με την καταχώρηση στη βάση
δεδομένων
            * όλων των απαιτήσεων για τη δημιουργία του προγράμματος εργασίας. */
            frmEdit EditDelete = new frmEdit();

            EditDelete.choiceDataTable = 1;
            EditDelete.Show(); // Εμφάνιση της φόρμας
            this.Hide(); // Απόκρυψη της φόρμας του κύριου μενού

            // Δημιουργία χειριστηρίου για τον έλεγχο κλεισίματος της φόρμας
            EditDelete.FormClosing += KleisimoFormas;
        }

        private void btnEditDelete_Click(object sender, EventArgs e)
        {
            gbEmpMenu.Visible = false; // Απόκρυψη του γκρουπ του κύριου μενού
            gbManageRoster.Visible = true; // Εμφάνιση του μενού επιλογών σχετικές με
τον υπάλληλο

        }

        private void button1_Click(object sender, EventArgs e)
        {
            /* Δημιουργία και φόρτωση νέας φόρμας σχετική με την καταχώρηση στη βάση
δεδομένων
            * όλων των απαιτήσεων για τη δημιουργία του προγράμματος εργασίας. */
            frmEdit EditDelete = new frmEdit();

            EditDelete.choiceDataTable = 2;
            EditDelete.Show(); // Εμφάνιση της φόρμας
            this.Hide(); // Απόκρυψη της φόρμας του κύριου μενού

            // Δημιουργία χειριστηρίου για τον έλεγχο κλεισίματος της φόρμας
            EditDelete.FormClosing += KleisimoFormas;
        }

        private void btnEditEmp_Click(object sender, EventArgs e)
        {
            /* Δημιουργία και φόρτωση νέας φόρμας σχετική με την καταχώρηση στη βάση
δεδομένων
            * όλων των απαιτήσεων για τη δημιουργία του προγράμματος εργασίας. */
            frmEdit EditDelete = new frmEdit();

            EditDelete.choiceDataTable = 3;
            EditDelete.Show(); // Εμφάνιση της φόρμας
            this.Hide(); // Απόκρυψη της φόρμας του κύριου μενού

            // Δημιουργία χειριστηρίου για τον έλεγχο κλεισίματος της φόρμας
            EditDelete.FormClosing += KleisimoFormas;
        }
```

```
        private void button2_Click(object sender, EventArgs e)
        {
            /* Δημιουργία και φόρτωση νέας φόρμας σχετική με την καταχώρηση στη βάση
δεδομένων
            * όλων των απαιτήσεων για τη δημιουργία του προγράμματος εργασίας. */
            frmEdit EditDelete = new frmEdit();

            EditDelete.choiceDataTable = 4;
            EditDelete.Show(); // Εμφάνιση της φόρμας
            this.Hide(); // Απόκρυψη της φόρμας του κύριου μενού

            // Δημιουργία χειριστηρίου για τον έλεγχο κλεισίματος της φόρμας
            EditDelete.FormClosing += KleisimoFormas;
        }
    }
}
```

ΦΟΡΜΑ frmNewPattern

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Linq;
using System.Windows.Forms;
using NurseRostering.Database;
using NurseRostering.InputData;

namespace NurseRostering.GUI
{
    public partial class frmNewPattern : Form
    {
        // Βάση δεδομένων
        NurseRosteringDB nrdb = new NurseRosteringDB();
        List<string> ListOfAvPatterns = new List<string>();

        public frmNewPattern()
        {
            InitializeComponent();
        }

        #region ΔΙΑΧΕΙΡΙΣΗ ΑΝΕΠΙΘΥΜΗΤΩΝ ΠΡΟΤΥΠΩΝ ΕΡΓΑΣΙΑΣ ΕΝΟΣ
ΝΟΣΟΚΟΜΟΥ

        // Εισαγωγή μιας επιλογής από τις διαθέσιμες, στη λίστα επιλεγμένων
ανεπιθύμητων προτύπων εργασίας
        private void btnAdd_Unwanted_Pattern_Click(object sender, EventArgs e)
        {
```

```
            // Εάν έχει πραγματοποιηθεί μια επιλογή από τη λίστα των διαθέσιμων
επιλογών
            if (lstUnwanted_List.SelectedIndex > -1)
            {
                // Μεταφορά της επιλογής στη λίστα επιλεγμένων.

lstUnwanted_Patterns.Items.Add(lstUnwanted_List.SelectedItem.ToString());
            }
        }

        /* Διαγραφή ενός επιλεγμένου ανεπιθύμητου προτύπου εργασίας από τη λίστα με
τις επιλογές
         * που πραγματοποίησε ο νοσοκόμος και εγγραφή της στη λίστα των διαθέσιμων
επιλογών του χρήστη */
        private void btnDel_Unwanted_Pattern_Click(object sender, EventArgs e)
        {
            // Εάν έχει πραγματοποιηθεί μια επιλογή από τον χρήστη
            if (lstUnwanted_Patterns.SelectedIndex > -1)
            {
                // Μεταφορά της επιλογής στη λίστα διαθέσιμων επιλογών.

lstUnwanted_Patterns.Items.RemoveAt(lstUnwanted_Patterns.SelectedIndex);
            }
        }

        // Καθαρισμός ολόκληρης της λίστας των επιλογών ενός νοσοκόμου.
        private void btnClear_Unwanted_Pattern_Click(object sender, EventArgs e)
        {
            lstUnwanted_Patterns.Items.Clear(); // Καθαρισμός λίστας
        }

        #endregion

        private void btnSubmit_Click(object sender, EventArgs e)
        {
            string types = "";
            foreach(var item in lstUnwanted_Patterns.Items)
            {
                types += item.ToString() + " ";
            }

            if(lstUnwanted_Patterns.Items.Count > 0)
            {
                // Έλεγχος διπλοεγγραφών στη βάση δεδομένων
                int check = nrdb.Patterns.Count(x => x.ShiftType == types.Trim());

                if (check > 0)
                {
                    if (check > 0) // Εάν βρεθεί μια εγγραφή έστω
                    {
                        // Ενημέρωση του χρήστη με σχετικό μήνυμα
```

```
                    MessageBox.Show("Υπάρχει ήδη μια εγγραφή στη βάση δεδομένων με
αυτό το πρότυπο.", "NurseRostering", MessageBoxButtons.OK,
MessageBoxIcon.Error);
                    return; // Τερματισμός διαδικασίας
                }
            }

            try
            {
                Pattern NewPattern = new Pattern();

                NewPattern.NumberOfShiftTypes = lstUnwanted_Patterns.Items.Count;
                NewPattern.weight = (int)nudWUnWanted.Value;
                NewPattern.ShiftType = types.Trim();
                nrdb.Patterns.Add(NewPattern);
                nrdb.SaveChanges();
                MessageBox.Show("Το πρότυπο ανεπιθύμητης εργασίας καταχωρήθηκε
επιτυχώς.", "NurseRostering", MessageBoxButtons.OK,
MessageBoxIcon.Information);

            }
            catch (Exception ex)
            {
                MessageBox.Show("Προέκυψε κάποιο απροσδόκητο σφάλμα. Η εγγραφή
δεν πραγματοποιήθηκε. Σφάλμα: " + ex.ToString(), "NurseRostering",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }
        else
        {
            // Ενημέρωση του χρήστη με σχετικό μήνυμα
            MessageBox.Show("Δεν υπάρχει κάποια εγγραφή προς καταχώρηση.",
"NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Error);
            return; // Τερματισμός διαδικασίας
        }

    }

    private void btnBack_Click(object sender, EventArgs e)
    {
        this.Close(); // Κλείσιμο της παρούσας φόρμας και εμφάνιση αυτής που την
κάλεσε
        GC.Collect(); // Συλλογή "σκουπιδιών"
    }

    private void frmNewPattern_Load(object sender, EventArgs e)
    {
        // Καθαρισμός λιστών
        ListOfAvPatterns.Clear();
        lstUnwanted_List.Items.Clear();

        // Άντληση δεδομένων από τη βάση και εμφάνισή τους στο σχετικό
αντικείμενο
```

```
            ListOfAvPatterns = (from pt in nrdb.AvailablePatterns select
pt.pattern).ToList();
            foreach (var item in ListOfAvPatterns)
            {
                lstUnwanted_List.Items.Add(item);
            }
            lstUnwanted_List.Refresh();
        }

        private void btnNewPattern_Click(object sender, EventArgs e)
        {
            // Δημιουργία νέας φόρμας καταχώρησης νέου προτύπου ανεπιθύμητης
εργασίας
            frmUnwantedPatterns newUnwanted = new frmUnwantedPatterns();
            newUnwanted.btnExit.Text = " Επιστροφή στη διαχείριση προτύπων
ανεπιθύμητης εργασίας";
            newUnwanted.Show(); // Εμφάνιση φόρμας καταχώρησης νέου προτύπου
ανεπιθύμητης εργασίας
            this.Hide(); // Κρύψιμο της παρούσας φόρμας

            // Δημιουργία χειριστηρίου για τον έλεγχο κλεισίματος της φόρμας
            newUnwanted.FormClosing += KleisimoFormas;
        }

        private void KleisimoFormas(object sender, FormClosingEventArgs e)
        {
            frmNewPattern_Load(this, e);
            this.Show(); // Εμφάνιση κύριας φόρμας, όταν κλείσει η φόρμα απαιτήσεων
προγράμματος εργασίας
        }
    }
}
```

ΦΟΡΜΑ frmNewShift

```
using NurseRostering.Database;
using NurseRostering.InputData;
using System;
using System.Collections.Generic;
using System.Data;
using System.Linq;
using System.Windows.Forms;

namespace NurseRostering.GUI
{
    public partial class frmNewShift : Form
    {
        Shift_Type Shift; // Νέα βάρδια
        Skill ReqSkills; // Δεξιότητες υπαλλήλου
```

```
        NurseRosteringDB nrdb = new NurseRosteringDB(); // Βάση δεδομένων
        List<Available_Skill> ListOfSkills = new List<Available_Skill>(); // Λίστα με
όλες τις διαθέσιμες ειδικότητες για επιλογή

        private string de3iotites = ""; // Λίστα δεξιοτήτων που απαιτούνται για τη βάρδια
        private string shift_ID = ""; // Αναγνωριστικό βάρδιας
        private string shift_description = ""; // Περιγραφή βάρδιας
        private string shift_Start; // Ώρα έναρξης βάρδιας
        private string shift_Stop; // Ώρα λήξης βάρδιας
        private int numberOfSkills = 0; // Πλήθος δεξιοτήτων που απαιτούνται για τη
βάρδια

        public frmNewShift()
        {
            InitializeComponent();
        }

        #region ΕΝΕΡΓΕΙΕΣ ΠΟΥ ΓΙΝΟΝΤΑΙ ΜΕ ΤΑ ΚΟΥΜΠΙΑ ΤΗΣ ΦΟΡΜΑΣ

        // Κουμπί επιστροφής στο μενού διαχείρισης βαρδιών
        private void btnReturn_Click(object sender, EventArgs e)
        {
            // επιστροφή στην προηγούμενη φόρμα
            this.Close();
            GC.Collect(); // Συλλογή "σκουπιδιών"
        }

        private void btnClearAll_Click(object sender, EventArgs e)
        {
            // Καθαρισμός όλων των πεδίων με παραμονή σε αυτή τη φόρμα
            txtDescription.Text = "";
            txtStart.Text = "";
            txtStop.Text = "";
            if(lstEmployeeSkills.Items.Count > 0) // Εάν έχει επιλεχθεί έστω και μια
ειδικότητα
            {
                // κάθε επιλεγμένη ειδικότητα
                foreach(var item in lstEmployeeSkills.Items)
                {
                    // μεταφορά της στη λίστα των διαθέσιμων ειδικοτήτων
                    lstAvailableSkills.Items.Add(item.ToString());
                }
                // και καθαρισμός της λίστας των ειδικοτήτων του υπαλλήλου
                lstEmployeeSkills.Items.Clear();
            }
        }

        /* Μεταφορά μιας ειδικότητας από τη λίστα των διαθέσιμων ειδικοτήτων
         * στη λίστα ειδικοτήτων του υπαλλήλου */
        private void btnDelSkill_Click(object sender, EventArgs e)
        {
            if(lstEmployeeSkills.SelectedIndex > -1)
            {
```

```
            lstAvailableSkills.Items.Add(lstEmployeeSkills.SelectedItem.ToString());
            lstEmployeeSkills.Items.RemoveAt(lstEmployeeSkills.SelectedIndex);
        }
    }

    /* Μεταφορά μιας ειδικότητας από τη λίστα των διαθέσιμων ειδικοτήτων
     * στη λίστα ειδικοτήτων του υπαλλήλου */
    private void btnAddSkill_Click(object sender, EventArgs e)
    {
        if (lstAvailableSkills.SelectedIndex > -1)
        {
            lstEmployeeSkills.Items.Add(lstAvailableSkills.SelectedItem.ToString());
            lstAvailableSkills.Items.RemoveAt(lstAvailableSkills.SelectedIndex);
        }
    }

    /* Μεταφορά όλων των ειδικοτήτων από τη λίστα των διαθέσιμων ειδικοτήτων
     * στη λίστα ειδικοτήτων του υπαλλήλου */
    private void btnAddAllSkills_Click(object sender, EventArgs e)
    {
        foreach(var item in lstAvailableSkills.Items)
        {
            lstEmployeeSkills.Items.Add(item.ToString());
        }
        lstAvailableSkills.Items.Clear();
    }

    /* Μεταφορά όλων των ειδικοτήτων από τη λίστα των ειδικοτήτων του
υπαλλήλου
     * στη λίστα διαθέσιμων ειδικοτήτων */
    private void btnDelAllSkills_Click(object sender, EventArgs e)
    {
        foreach (var item in lstEmployeeSkills.Items)
        {
            lstAvailableSkills.Items.Add(item.ToString());
        }
        lstEmployeeSkills.Items.Clear();
    }

    private void btnSubmitShift_Click(object sender, EventArgs e)
    {
        load_ShiftType_data(); // Καταχώρηση των δεδομένων στη βάση δεδομένων
    }

    #endregion

    private void load_ShiftType_data()
    {
        Shift = new Shift_Type(); // Αντικείμενο νέας βάρδιας.

        // Αν το αναγνωριστικό δεν είναι κενό χαρακτήρων
        if (cbShiftType.SelectedIndex > -1)
        {
```

```
            shift_ID = cbShiftType.SelectedItem.ToString();
        }
        else
        {
            MessageBox.Show("Το αναγνωριστικό βάρδιας, δεν μπορεί να είναι κενό. Η
εγγραφή δεν πραγματοποιείται.", "NurseRostering", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            return;
        }


        // Αν η περιγραφή δεν είναι κενή χαρακτήρων
        if (txtDescription.Text.Trim() != "")
        {
            shift_description = txtDescription.Text;
        }
        else
        {
            MessageBox.Show("Η περιγραφή της βάρδιας, δεν μπορεί να είναι κενή. Η
εγγραφή δεν πραγματοποιείται.", "NurseRostering", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            return;
        }

        // Έλεγχος διπλοεγγραφών στη βάση δεδομένων
        int check = nrdb.Shift_Types.Count(x => x.tipos ==
cbShiftType.SelectedItem.ToString() || x.description == txtDescription.Text.Trim());

        if (check > 0)
        {
            if (check > 0) // Εάν βρεθεί μια εγγραφή έστω
            {
                // Ενημέρωση του χρήστη με σχετικό μήνυμα
                MessageBox.Show("Υπάρχει ήδη μια εγγραφή στη βάση δεδομένων με
αυτό το αναγνωριστικό ή την περιγραφή.", "NurseRostering",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                return; // Τερματισμός διαδικασίας
            }
        }

        // Μετατροπή των ωρών έναρξης και λήξης της βάρδιας από String σε τύπο
DateTime
        shift_Start = txtStart.Text;
        shift_Stop = txtStop.Text;

        // Για κάθε εγκεγραμμένη ειδικότητα του νοσηλευτή
        de3iotites = "";
        foreach (var item in lstEmployeeSkills.Items)
        {
            ReqSkills = new Skill(); // Δημιουργία αντικειμένου δεξιοτεχνίας
            ReqSkills.de3iotexnia = item.ToString(); // Απόδοση τιμής
            nrdb.EmployeeSkills.Add(ReqSkills); // Αποστολή για εγγραφή στη βάση
δεδομένων
```

```
            de3iotites += item.ToString() + " "; // Προσθήκη στη λίστα δεξιοτεχνιών
που απαιτούνται για τη βάρδια
            numberOfSkills += 1; // Επαύξηση του μετρητή ειδικοτήτων κατά μια
μονάδα
        }

        Shift = new Shift_Type // Δημιουργία αντικειμένου βάρδιας
        {
            tipos = shift_ID,
            description = shift_description,
            startTime = shift_Start,
            endTime = shift_Stop,
            numberOfRequiredSkills = numberOfSkills,
            requiredSkills = de3iotites.Trim(),
        };

        try // Απόπειρα καταχώρησης νέας βάρδιας
        {
            nrdb.Shift_Types.Add(Shift); // Προσθήκη της βάρδιας στη λίστα προς
καταχώρηση στη βάση δεδομένων
            nrdb.SaveChanges(); // Καταχώρηση στη βάση δεδομένων
            /* Καθαρισμός της λίστας δεξιοτήτων και του μετρητή. Αυτή η ενέργεια
είναι απαραίτητη, διότι
             * αν δεν καθαριστεί, τα περιεχόμενά της ενδέχεται να περάσουν στα
στοιχεία της
             * επόμενης βάρδιας, στην περίπτωση καταχώρησης περισσότερων της μίας
βάρδιας τη φορά */
            de3iotites = "";
            numberOfSkills = 0;

            // Ενημέρωση του χρήστη για την επιτυχή καταχώρηση της νέας βάρδιας
            MessageBox.Show("Η νέα βάρδια καταχωρήθηκε με επιτυχία.",
"NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
        catch (Exception ex) // Εάν προκύψει κάποιο σφάλμα
        {
            // Ενημέρωση του χρήστη σχετικά με το σφάλμα που προέκυψε
            MessageBox.Show("Προέκυψε το παρακάτω σφάλμα. Η καταχώρηση της
βάρδιας δεν πραγματοποιήθηκε. Σφαλμα: " + ex, "NurseRostering",
MessageBoxButtons.OK, MessageBoxIcon.Error);
            return; // Διακοπή της διαδικασίας.
        }
    }

    private void frmNewShift_Load(object sender, EventArgs e)
    {
        lstAvailableSkills.Items.Clear();
        ListOfSkills.Clear();

        // Λήψη όλων των ειδικοτήτων και τοποθέτησή τους στις σχετικές λίστες
        ListOfSkills = (from skl in nrdb.Available_Skills select skl).OrderBy(x =>
x.de3iotexnia).ToList();
```

```
            // Τοποθέτηση των μοναδικών ειδικοτήτων στη λίστα
            var SkillList = ListOfSkills.Select(x => x.de3iotexnia).Distinct().ToList();
            foreach (var item in SkillList)
            {
                // Προβολή της ειδικότητας στη λίστα ειδικοτήτων
                lstAvailableSkills.Items.Add(item.ToString());
            }
        }

        private void button1_Click(object sender, EventArgs e)
        {
            // Δημιουργία νέας φόρμας καταχώρησης νέου υπαλλήλου
            frmAddSkill newEmp = new frmAddSkill();
            newEmp.btnCancel.Text = "Επιστροφή στη δημιουργία βαρδιών";
            newEmp.Show(); // Εμφάνιση φόρμας καταχώρησης νέου υπαλλήλου
            this.Hide(); // Κρύψιμο της παρούσας φόρμας

            // Δημιουργία χειριστηρίου για τον έλεγχο κλεισίματος της φόρμας
            newEmp.FormClosing += KleisimoFormas;
        }

        private void KleisimoFormas(object sender, FormClosingEventArgs e)
        {
            frmNewShift_Load(this, e);
            this.Show(); // Εμφάνιση κύριας φόρμας, όταν κλείσει η φόρμα απαιτήσεων
προγράμματος εργασίας
        }

    }
}
```

ΦΟΡΜΑ frmPreview

```
using NurseRostering.Database;
using NurseRostering.InputData;
using System;
using System.Collections.Generic;
using System.Data;
using System.Linq;
using System.Windows.Forms;

namespace NurseRostering.GUI
{
    public partial class frmPreview : Form
    {
        NurseRosteringDB nrdb = new NurseRosteringDB();

        /* Λίστες στις οποίες θα φορτωθούν όλα τα δεδομένα από τη βάση, για να
προβληθούν
```

```
        * στα σχετικά datagridviews της διεπαφής, προς ενημέρωση και έλεγχο από τον χρήστη */
        private List<Contract> Contracts = new List<Contract>();
        private List<Date_Specific_Cover> Date_Specific_Covers = new List<Date_Specific_Cover>();
        private List<Day_Of_Week_Cover> Day_Of_Week_Covers = new List<Day_Of_Week_Cover>();
        private List<Day_On_Request> Day_On_Requests = new List<Day_On_Request>();
        private List<Day_Off_Request> Day_Off_Requests = new List<Day_Off_Request>();
        private List<Employee> Employees = new List<Employee>();
        private List<Available_Skill> Available_Skills = new List<Available_Skill>();
        private List<Pattern> Patterns = new List<Pattern>();
        private List<Scheduling_Period> Scheduling_Periods = new List<Scheduling_Period>();
        private List<Shift_On_Request> Shift_On_Requests = new List<Shift_On_Request>();
        private List<Shift_Off_Request> Shift_Off_Requests = new List<Shift_Off_Request>();
        private List<Shift_Type> Shift_Types = new List<Shift_Type>();
        private List<Available_Pattern> AvPatterns = new List<Available_Pattern>();
        private List<Skill> EmpSkills = new List<Skill>();

        private DateTime startDate;
        private DateTime endDate;

        public frmPreview()
        {
            InitializeComponent();
        }

        private void επιστροφήΣτηΔημιουργίαΠρογράμματοςToolStripMenuItem_Click(object sender, EventArgs e)
        {
            this.Close(); // Κλείσιμο της φόρμας προεπισκόπισης και εμφάνιση αυτής που την κάλεσε
            GC.Collect(); // Συλλογή "σκουπιδιών"
        }

        // Ενέργειες που πραγματοποιούνται κατά την φόρτωση της φόρμας
        private void frmPreview_Load(object sender, EventArgs e)
        {

            #region ΑΝΤΛΗΣΗ ΔΕΔΟΜΕΝΩΝ ΑΠΟ ΤΗ ΒΑΣΗ ΚΑΙ ΠΡΟΒΟΛΗ ΤΟΥΣ ΣΤΟ ΧΡΗΣΤΗ

            // Άντληση δεδομένων από τη βάση
            Scheduling_Periods = (from r in nrdb.Scheduling_Periods select r).OrderByDescending(x => x.startDate).ToList();

            DateTime schStartDate;
```

```
        DateTime schStopDate;

        // Έλεγχος για το αν υπάρχει καταχωρημένο κάποιο πρόγραμμα εργασίας
        if (Scheduling_Periods.Count > 0)
        {
          // Δημοσίευση σχετικών ημερομηνιών αν υπάρχει κάποιο πρόγραμμα
εργασίας
          schStartDate = Scheduling_Periods[0].startDate.Date;
          schStopDate = Scheduling_Periods[0].endDate.Date;
        }
        else // Άν δεν υπάρχει κάποιο πρόγραμμα εργασίας
        {
          // Δημοσίευση της σημερινής ημερομηνίας
          schStartDate = DateTime.Now.Date;
          schStopDate = DateTime.Now.Date;
        }


        Contracts = (from r in nrdb.Contracts select r).ToList();
        Date_Specific_Covers = (from r in nrdb.Date_Specific_Covers select r)
          .Where(r => r.weekDay >= schStartDate && r.weekDay <= schStopDate)
          .ToList();
        Day_Of_Week_Covers = (from r in nrdb.Day_Of_Week_Covers select
r).ToList();
        Day_On_Requests = (from r in nrdb.Day_On_Requests select r)
          .Where(r => r.date >= schStartDate && r.date <= schStopDate)
          .ToList();
        Day_Off_Requests = (from r in nrdb.Day_Off_Requests select r)
          .Where(r => r.date >= schStartDate && r.date <= schStopDate)
          .ToList();
        Employees = (from r in nrdb.Employees select r).ToList();
        Available_Skills = (from r in nrdb.Available_Skills select r).ToList();
        Patterns = (from r in nrdb.Patterns select r).ToList();
        Shift_On_Requests = (from r in nrdb.Shift_On_Requests select r)
          .Where(r => r.date >= schStartDate && r.date <= schStopDate)
          .ToList();
        Shift_Off_Requests = (from r in nrdb.Shift_Off_Requests select r)
          .Where(r => r.date >= schStartDate && r.date <= schStopDate)
          .ToList();
        Shift_Types = (from r in nrdb.Shift_Types select r).ToList();
        AvPatterns = (from r in nrdb.AvailablePatterns select r).ToList();

        // Εμφάνιση δεδομένων στο σχετικό  datagridview
        dgContracts.DataSource = Contracts;
        dgDateSpecificCover.DataSource = Date_Specific_Covers;
        dgDayOfWeekCover.DataSource = Day_Of_Week_Covers;
        dgDayOnReq.DataSource = Day_On_Requests;
        dgDayOffReq.DataSource = Day_Off_Requests;
        dgEmployees.DataSource = Employees;
        dgSkills.DataSource = Available_Skills;
        dgPatterns.DataSource = Patterns;
        dgSchedulingPeriod.DataSource = Scheduling_Periods;
        dgShiftOnReq.DataSource = Shift_On_Requests;
```

```
        dgShiftOffReq.DataSource = Shift_Off_Requests;
        dgShiftTypes.DataSource = Shift_Types;
        dgPat.DataSource = AvPatterns;
        dgEmployeeSkills.DataSource = EmpSkills;

        #endregion

        #region ΔΙΑΜΟΡΦΩΣΗ ΤΟΥ ΠΛΑΤΟΥΣ ΤΩΝ ΣΤΗΛΩΝ ΟΛΩΝ ΤΩΝ
DATAGRIDVIEWS

        // Σχεδίαση προγράμματος
        dgSchedulingPeriod.Columns[0].Width = (int)(dgSchedulingPeriod.Width *
0.15);
        dgSchedulingPeriod.Columns[1].Width = (int)(dgSchedulingPeriod.Width *
0.3);
        dgSchedulingPeriod.Columns[2].Width = (int)(dgSchedulingPeriod.Width *
0.3);
        dgSchedulingPeriod.Columns[3].Width = dgSchedulingPeriod.Width -
(int)(dgSchedulingPeriod.Width * 0.75);

        // Ειδικότητες
        dgSkills.Columns[0].Width = (int)(dgSkills.Width * 0.2);
        dgSkills.Columns[1].Width = dgSkills.Width - (int)(dgSkills.Width * 0.2);

        // Βάρδιες
        dgShiftTypes.Columns[0].Width = (int)(dgShiftTypes.Width * 0.1);
        dgShiftTypes.Columns[1].Width = (int)(dgShiftTypes.Width * 0.1);
        dgShiftTypes.Columns[2].Width = (int)(dgShiftTypes.Width * 0.2);
        dgShiftTypes.Columns[3].Width = (int)(dgShiftTypes.Width * 0.2);
        dgShiftTypes.Columns[4].Width = (int)(dgShiftTypes.Width * 0.2);
        dgShiftTypes.Columns[5].Width = (int)(dgShiftTypes.Width * 0.1);
        dgShiftTypes.Columns[6].Width = dgShiftTypes.Width -
(int)(dgShiftTypes.Width * 0.7);

        // Πρότυπα ανεπιθύμητης εργασίας
        dgPatterns.Columns[0].Width = (int)(dgPatterns.Width * 0.1);
        dgPatterns.Columns[1].Width = (int)(dgPatterns.Width * 0.1);
        dgPatterns.Columns[2].Width = (int)(dgPatterns.Width * 0.1);
        dgPatterns.Columns[3].Width = dgPatterns.Width - (int)(dgPatterns.Width *
0.3);

        // Ιδιαίτερες απαταιτήσεις σε προσωπικό σε συγκεκριμένη ημέρα και βάρδια
        dgDateSpecificCover.Columns[0].Width = (int)(dgDateSpecificCover.Width *
0.1);
        dgDateSpecificCover.Columns[1].Width = (int)(dgDateSpecificCover.Width *
0.3);
        dgDateSpecificCover.Columns[2].Width = (int)(dgDateSpecificCover.Width *
0.3);
        dgDateSpecificCover.Columns[3].Width = dgDateSpecificCover.Width -
(int)(dgDateSpecificCover.Width * 0.7);

        // Απαιτήσεις σε προσωπικό κάθε ημέρα της εβδομάδας
```

```
        dgDayOfWeekCover.Columns[0].Width = (int)(dgDayOfWeekCover.Width *
0.1);
        dgDayOfWeekCover.Columns[1].Width = (int)(dgDayOfWeekCover.Width *
0.3);
        dgDayOfWeekCover.Columns[2].Width = (int)(dgDayOfWeekCover.Width *
0.3);
        dgDayOfWeekCover.Columns[3].Width = dgDayOfWeekCover.Width -
(int)(dgDayOfWeekCover.Width * 0.7);

        // Αιτήσεις προσωπικού για εργασία σε συγκεκριμένη ημέρα
        dgDayOnReq.Columns[0].Width = (int)(dgDayOnReq.Width * 0.15);
        dgDayOnReq.Columns[1].Width = (int)(dgDayOnReq.Width * 0.55);
        dgDayOnReq.Columns[2].Width = (int)(dgDayOnReq.Width * 0.15);
        dgDayOnReq.Columns[3].Width = dgDayOnReq.Width -
(int)(dgDayOnReq.Width * 0.85);

        // Αιτήσεις προσωπικού για ανάπαυση σε συγκεκριμένη ημέρα
        dgDayOffReq.Columns[0].Width = (int)(dgDayOffReq.Width * 0.15);
        dgDayOffReq.Columns[1].Width = (int)(dgDayOffReq.Width * 0.55);
        dgDayOffReq.Columns[2].Width = (int)(dgDayOffReq.Width * 0.15);
        dgDayOffReq.Columns[3].Width = dgDayOffReq.Width -
(int)(dgDayOffReq.Width * 0.85);

        // Αιτήσεις προσωπικού για εργασία σε συγκεκριμένη ημέρα και βάρδια
        dgShiftOnReq.Columns[0].Width = (int)(dgShiftOnReq.Width * 0.15);
        dgShiftOnReq.Columns[1].Width = (int)(dgShiftOnReq.Width * 0.4);
        dgShiftOnReq.Columns[2].Width = (int)(dgShiftOnReq.Width * 0.15);
        dgShiftOnReq.Columns[3].Width = (int)(dgShiftOnReq.Width * 0.15);
        dgShiftOnReq.Columns[4].Width = dgShiftOnReq.Width -
(int)(dgShiftOnReq.Width * 0.85);

        // Αιτήσεις προσωπικού για ανάπαυση σε συγκεκριμένη ημέρα και βάρδια
        dgShiftOffReq.Columns[0].Width = (int)(dgShiftOffReq.Width * 0.15);
        dgShiftOffReq.Columns[1].Width = (int)(dgShiftOffReq.Width * 0.4);
        dgShiftOffReq.Columns[2].Width = (int)(dgShiftOffReq.Width * 0.15);
        dgShiftOffReq.Columns[3].Width = (int)(dgShiftOffReq.Width * 0.15);
        dgShiftOffReq.Columns[4].Width = dgShiftOffReq.Width -
(int)(dgShiftOffReq.Width * 0.85);

        // Προσωπικό
        dgEmployees.Columns[0].Width = (int)(dgEmployees.Width * 0.03);
        dgEmployees.Columns[1].Width = (int)(dgEmployees.Width * 0.15);
        dgEmployees.Columns[2].Width = (int)(dgEmployees.Width * 0.08);
        dgEmployees.Columns[3].Width = (int)(dgEmployees.Width * 0.07);
        dgEmployees.Columns[4].Width = (int)(dgEmployees.Width * 0.07);
        dgEmployees.Columns[5].Width = (int)(dgEmployees.Width * 0.10);
        dgEmployees.Columns[6].Width = (int)(dgEmployees.Width * 0.07);
        dgEmployees.Columns[7].Width = (int)(dgEmployees.Width * 0.05);
        dgEmployees.Columns[8].Width = (int)(dgEmployees.Width * 0.08);
        dgEmployees.Columns[9].Width = (int)(dgEmployees.Width * 0.08);
        dgEmployees.Columns[10].Width = (int)(dgEmployees.Width * 0.12);
        dgEmployees.Columns[11].Width = (int)(dgEmployees.Width * 0.03);
```

```
        dgEmployees.Columns[12].Width = dgEmployees.Width -
(int)(dgEmployees.Width * 0.93);

        // Συμβόλαια
        dgContracts.Columns[1].Width = (int)(dgEmployees.Width * 0.07);
        for (int i = 2; i < 41; i++)
        {
           dgContracts.Columns[i].Width = (int)(dgEmployees.Width * 0.0175);
        }
        dgContracts.Columns[41].Width = (int)(dgEmployees.Width * 0.15);
        dgContracts.Columns[42].Width = (int)(dgEmployees.Width * 0.0175);
        dgContracts.Columns[43].Width = dgContracts.Width -
(int)(dgEmployees.Width * 0.98);

        // Διαθέσιμα βασικά πρότυπα για σύνθεση ανεπιθύμητων προτύπων εργασίας
        dgPat.Columns[0].Width = (int)(dgPat.Width * 0.2);
        dgPat.Columns[1].Width = dgPat.Width - (int)(dgPat.Width * 0.2);

        // Διαθέσιμα βασικά πρότυπα για σύνθεση ανεπιθύμητων προτύπων εργασίας
        dgEmployeeSkills.Columns[0].Width = (int)(dgEmployeeSkills.Width * 0.2);
        dgEmployeeSkills.Columns[1].Width = dgEmployeeSkills.Width -
(int)(dgEmployeeSkills.Width * 0.8);
        dgEmployeeSkills.Columns[2].Visible = false;
        #endregion

        #region ΟΝΟΜΑΣΙΑ ΚΑΙ ΚΛΕΙΔΩΜΑ ΣΤΗΛΩΝ ΓΙΑ ΑΠΟΤΡΟΠΗ
ΑΛΛΑΓΩΝ ΤΟΥΣ ΑΠΟ ΤΟΝ ΧΡΗΣΤΗ

        dgSchedulingPeriod.Columns[1].HeaderText = "Πρόγραμμα";
        dgSchedulingPeriod.Columns[2].HeaderText = "Έναρξη";
        dgSchedulingPeriod.Columns[3].HeaderText = "Λήξη";

        dgSkills.Columns[1].HeaderText = "Ειδικότητα";

        dgShiftTypes.Columns[1].HeaderText = "Τύπος";
        dgShiftTypes.Columns[2].HeaderText = "Περιγραφή";
        dgShiftTypes.Columns[3].HeaderText = "Έναρξη";
        dgShiftTypes.Columns[4].HeaderText = "Λήξη";
        dgShiftTypes.Columns[5].HeaderText = "Απαιτούμενος αριθμός ειδικοτήτων";
        dgShiftTypes.Columns[6].HeaderText = "Ειδικότητες";

        dgShiftTypes.Columns[0].ReadOnly = true;
        dgShiftTypes.Columns[1].ReadOnly = true;
        dgShiftTypes.Columns[2].ReadOnly = true;
        dgShiftTypes.Columns[5].ReadOnly = true;
        dgShiftTypes.Columns[6].ReadOnly = true;


        dgPatterns.Columns[1].HeaderText = "Προτεραιότητα";
        dgPatterns.Columns[2].HeaderText = "Αριθμός στοιχειωδών προτύπων";
        dgPatterns.Columns[3].HeaderText = "Πρότυπο ανεπιθύμητης εργασίας";

        dgDayOnReq.Columns[1].HeaderText = "Ημερομηνία";
```

```
dgDayOnReq.Columns[2].HeaderText = "Προτεραιότητα";
dgDayOnReq.Columns[3].HeaderText = "Υπάλληλος";

dgDayOffReq.Columns[1].HeaderText = "Ημερομηνία";
dgDayOffReq.Columns[2].HeaderText = "Προτεραιότητα";
dgDayOffReq.Columns[3].HeaderText = "Υπάλληλος";

dgShiftOnReq.Columns[1].HeaderText = "Ημερομηνία";
dgShiftOnReq.Columns[2].HeaderText = "Προτεραιότητα";
dgShiftOnReq.Columns[3].HeaderText = "Βάρδια";
dgShiftOnReq.Columns[4].HeaderText = "Υπάλληλος";

dgShiftOffReq.Columns[1].HeaderText = "Ημερομηνία";
dgShiftOffReq.Columns[2].HeaderText = "Προτεραιότητα";
dgShiftOffReq.Columns[3].HeaderText = "Βάρδια";
dgShiftOffReq.Columns[4].HeaderText = "Υπάλληλος";

dgEmployeeSkills.Columns[1].HeaderText = "Ειδικότητα";
dgEmployeeSkills.Columns[2].HeaderText = "Υπάλληλος";

dgPat.Columns[1].HeaderText = "Πρότυπα";

dgDateSpecificCover.Columns[1].HeaderText = "Ημέρα";
dgDateSpecificCover.Columns[2].HeaderText = "Βάρδια";
dgDateSpecificCover.Columns[3].HeaderText = "Πλήθος προσωπικού";

dgDateSpecificCover.Columns[0].ReadOnly = true;
dgDateSpecificCover.Columns[1].ReadOnly = true;
dgDateSpecificCover.Columns[2].ReadOnly = true;

dgDayOfWeekCover.Columns[1].HeaderText = "Ημέρα";
dgDayOfWeekCover.Columns[2].HeaderText = "Βάρδια";
dgDayOfWeekCover.Columns[3].HeaderText = "Πλήθος προσωπικού";

dgDayOfWeekCover.Columns[0].ReadOnly = true;
dgDayOfWeekCover.Columns[1].ReadOnly = true;
dgDayOfWeekCover.Columns[2].ReadOnly = true;

dgEmployees.Columns[1].HeaderText = "Ονοματεπώνυμο";
dgEmployees.Columns[2].HeaderText = "Αριθμός συμβολαίου";
dgEmployees.Columns[3].HeaderText = "Α.Μ.Κ.Α.";
dgEmployees.Columns[4].HeaderText = "Α.Φ.Μ.";
dgEmployees.Columns[5].HeaderText = "Διεύθυνση";
dgEmployees.Columns[6].HeaderText = "Πόλη";
dgEmployees.Columns[7].HeaderText = "ΤΚ";
dgEmployees.Columns[8].HeaderText = "Κινητό";
dgEmployees.Columns[9].HeaderText = "Τηλέφωνο";
dgEmployees.Columns[10].HeaderText = "Διεύθυνση e-mail";
dgEmployees.Columns[11].HeaderText = "Πλήθος ειδικοτήτων";
dgEmployees.Columns[12].HeaderText = "Φωτογραφία";

dgEmployees.Columns[0].ReadOnly = true;
dgEmployees.Columns[1].ReadOnly = true;
```

```
            dgEmployees.Columns[2].ReadOnly = true;
            dgEmployees.Columns[3].ReadOnly = true;
            dgEmployees.Columns[4].ReadOnly = true;
            dgEmployees.Columns[11].ReadOnly = true;
            dgEmployees.Columns[12].ReadOnly = true;

            dgContracts.Columns[1].HeaderText = "Περιγραφή";
            dgContracts.Columns[2].HeaderText = "Μια βάρδια ανά ημέρα";
            dgContracts.Columns[3].HeaderText = "Προτεραιότητα";

            /* Επειδή τα επόμενα κελιά ακολουθούν κάποιο πρότυπο επανάληψης,
             * η ονομασία τους τυποποιείται σύμφωνα με το πρώτο κελί τις ομάδας τους.
             * Για κάθε τριάδα μέσα στον παρακάτω βρόγχο, επαναλαμβάνεται το πρότυπο
(On/Off|Προτεραιότητα|Πλήθος)*/

            // Ονομασία πρώτου στοιχείου (On/Off) της μεταβλητής
            dgContracts.Columns[4].HeaderText = "Μέγιστο πλήθος βαρδιών";
            dgContracts.Columns[7].HeaderText = "Ελάχιστο πλήθος βαρδιών";
            dgContracts.Columns[10].HeaderText = "Μέγιστο πλήθος συνεχόμενων
ημερών εργασίας";
            dgContracts.Columns[13].HeaderText = "Ελάχιστο πλήθος συνεχόμενων
ημερών εργασίας";
            dgContracts.Columns[16].HeaderText = "Μέγιστο πλήθος συνεχόμενων
ημερών αναπαύσεων";
            dgContracts.Columns[19].HeaderText = "Ελάχιστο πλήθος συνεχόμενων
ημερών αναπαύσεων";
            dgContracts.Columns[22].HeaderText = "Μέγιστο πλήθος συνεχόμενων Σ/Κ
εργασίας";
            dgContracts.Columns[25].HeaderText = "Ελάχιστο πλήθος συνεχόμενων Σ/Κ
εργασίας";
            dgContracts.Columns[28].HeaderText = "Μέγιστος αριθμός Σ/Κ εργασίας σε 4
εβδομάδες";
            dgContracts.Columns[31].HeaderText = "Ολόκληρο το Σ/Κ";
            dgContracts.Columns[33].HeaderText = "Ίδια βάρδια το Σ/Κ";
            dgContracts.Columns[35].HeaderText = "Όχι νυχτερινή βάρδια πριν από το
Σ/Κ";
            dgContracts.Columns[37].HeaderText = "Δύο αναπαύσεις μετά από νυχτερινή
βάρδια";
            dgContracts.Columns[39].HeaderText = "Εργασία χωρίς κατάλληλη ειδίκευση";
            dgContracts.Columns[41].HeaderText = "Πλήθος ημερών Σ/Κ";
            dgContracts.Columns[42].HeaderText = "Πλήθος ανεπιθύμητων προτύπων
εργασίας";
            dgContracts.Columns[43].HeaderText = "Ανεπιθύμητα πρότυπα εργασίας";

            for (int i = 5; i < 41; i++)
            {
              if (i < 31)
              {
                if (i % 3 == 2)
                {
                  dgContracts.Columns[i].HeaderText = dgContracts.Columns[i -
1].HeaderText + " - προτεραιότητα";
                }
```

```
                if (i % 3 == 0)
                {
                    dgContracts.Columns[i].HeaderText = dgContracts.Columns[i -
2].HeaderText + " - πλήθος";
                }
            }
            if (i > 31)
            {
                if (i % 2 == 0)
                {
                    dgContracts.Columns[i].HeaderText = dgContracts.Columns[i -
1].HeaderText + " - προτεραιότητα";
                }
            }
        }

        // Κλείδωμα συγκεκριμένων κελιών
        dgContracts.Columns[0].ReadOnly = true;
        dgContracts.Columns[1].ReadOnly = true;
        dgContracts.Columns[2].ReadOnly = true;
        dgContracts.Columns[41].ReadOnly = true;
        dgContracts.Columns[42].ReadOnly = true;
        dgContracts.Columns[43].ReadOnly = true;

        #endregion

    }

    // Ενέργειες που πραγματοποιούνται όταν ο χρήστης επιλέξει μια γραμμή από το
DataGridView των υπαλλήλων
    private void dgEmployees_CellClick(object sender, DataGridViewCellEventArgs
e)
    {
        if (e.RowIndex != -1)
        {
            /* Μεταβλητή η οποία φέρει την τιμή του ΑΦΜ του υπαλλήλου, η οποία
αντλείται
             * από το σχετικό DataGridView, όταν ο χρήστης επιλέγει μια γραμμή
εγγραφής */
            int empCoordAFM =
int.Parse(dgEmployees.Rows[e.RowIndex].Cells[4].Value.ToString());
            int empID =
int.Parse(dgEmployees.Rows[e.RowIndex].Cells[0].Value.ToString());

            /* Έυρεση όλων των δεδομένων του επιλεγμένου υπαλλήλου από τη βάση
δεδομένων
             * και φόρτωσή τους στη σχετική λίστα για εμφάνιση στο DataGridView της
διεπαφής */

            // Καθαρισμός των λιστών δεδομένων του επιλεγμένου υπαλλήλου
            EmpSkills.Clear();
            Day_On_Requests.Clear();
```

```
            Day_Off_Requests.Clear();
            Shift_On_Requests.Clear();
            Shift_Off_Requests.Clear();

            /* Άντληση δεδομένων από τη βάση δεδομένων σύμφωνα με το ΑΦΜ του
υπαλλήλου. Δεν
             * χρησιμοποιείται ο κωδικός υπαλλήλου διότι δεν υπήρχε κατά την στιγμή
της
             * εγγραφής του και δήλωσης των ειδικοτήτων του */
            EmpSkills = (from x in nrdb.EmployeeSkills
                         select x)
                         .Where(x => x.employeeID == empCoordAFM)
                         .ToList();

            // Άντληση δεδομένων από τη βάση δεδομένων σύμφωνα με τον κωδικό του
υπαλλήλου.
            Day_On_Requests = (from r in nrdb.Day_On_Requests
                               select r)
                               .Where(r => r.employeeID == empID)
                               .ToList();
            Day_Off_Requests = (from r in nrdb.Day_Off_Requests
                                select r)
                                .Where(r => r.employeeID == empID)
                                .ToList();
            Shift_On_Requests = (from r in nrdb.Shift_On_Requests
                                 select r)
                                 .Where(r => r.employeeID == empID)
                                 .ToList();
            Shift_Off_Requests = (from r in nrdb.Shift_Off_Requests
                                  select r)
                                  .Where(r => r.employeeID == empID)
                                  .ToList();

            // Εμφάνιση των δεδομένων στα σχετικά DataGridView
            dgEmployeeSkills.DataSource = EmpSkills;
            dgDayOnReq.DataSource = Day_On_Requests;
            dgDayOffReq.DataSource = Day_Off_Requests;
            dgShiftOnReq.DataSource = Shift_On_Requests;
            dgShiftOffReq.DataSource = Shift_Off_Requests;

            dgDayOnReq.Columns[1].HeaderText = "Ημερομηνία";
            dgDayOnReq.Columns[2].HeaderText = "Προτεραιότητα";
            dgDayOnReq.Columns[3].HeaderText = "Υπάλληλος";

            dgDayOffReq.Columns[1].HeaderText = "Ημερομηνία";
            dgDayOffReq.Columns[2].HeaderText = "Προτεραιότητα";
            dgDayOffReq.Columns[3].HeaderText = "Υπάλληλος";

            dgShiftOnReq.Columns[1].HeaderText = "Ημερομηνία";
            dgShiftOnReq.Columns[2].HeaderText = "Προτεραιότητα";
            dgShiftOnReq.Columns[3].HeaderText = "Βάρδια";
            dgShiftOnReq.Columns[4].HeaderText = "Υπάλληλος";
```

```
            dgShiftOffReq.Columns[1].HeaderText = "Ημερομηνία";
            dgShiftOffReq.Columns[2].HeaderText = "Προτεραιότητα";
            dgShiftOffReq.Columns[3].HeaderText = "Βάρδια";
            dgShiftOffReq.Columns[4].HeaderText = "Υπάλληλος";
        }
    }

    private void αποθήκευσηΑλλαγώνToolStripMenuItem_Click(object sender,
EventArgs e)
    {
        try
        {
            // Αποθήκευση αλλαγών στη βάση δεδομένων
            nrdb.SaveChanges();

            // Ενημέρωση του χρήστη σχετικά με την επιτυψημένη καταχώρηση
            MessageBox.Show("Οι αλλαγές καταχωρήθηκαν επιτυχώς.",
"NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
        catch (Exception ex) // Σύλληψη σφάλματος
        {
            // Ενημέρωση του χρήστη σχετικά με το σφάλμα που προέκυψε
            MessageBox.Show("Προέκυψε κάποιο απροσδόκητο σφάλμα. Σφάλμα: " +
ex.ToString(),"NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
        finally
        {
            GC.Collect(); // Συλλογή "σκουπιδιών"
        }

    }

    private void dgSchedulingPeriod_CellClick(object sender,
DataGridViewCellEventArgs e)
    {
        if (e.RowIndex != -1)
        {

            startDate =
Convert.ToDateTime(dgSchedulingPeriod.Rows[e.RowIndex].Cells[2].Value.ToString
());
            endDate =
Convert.ToDateTime(dgSchedulingPeriod.Rows[e.RowIndex].Cells[3].Value.ToString
());

            List<Scheduling_Period> currentProgram = new List<Scheduling_Period>();

            // Καθαρισμός των λιστών δεδομένων του επιλεγμένου υπαλλήλου
            Day_On_Requests.Clear();
            Day_Off_Requests.Clear();
            Shift_On_Requests.Clear();
            Shift_Off_Requests.Clear();
```

```
            currentProgram = (from x in nrdb.Scheduling_Periods
                              where (x.startDate >= startDate && x.endDate <= endDate)
                              select x).ToList();

            foreach (var item in currentProgram)
            {
              // Άντληση δεδομένων από τη βάση δεδομένων σύμφωνα με τον κωδικό
του υπαλλήλου.
              Day_On_Requests = (from r in nrdb.Day_On_Requests
                                 select r)
                                 .Where(r => r.date >= startDate && r.date <= endDate)
                                 .ToList();
              Day_Off_Requests = (from r in nrdb.Day_Off_Requests
                                  select r)
                                  .Where(r => r.date >= startDate && r.date <= endDate)
                                  .ToList();
              Shift_On_Requests = (from r in nrdb.Shift_On_Requests
                                   select r)
                                   .Where(r => r.date >= startDate && r.date <= endDate)
                                   .ToList();
              Shift_Off_Requests = (from r in nrdb.Shift_Off_Requests
                                    select r)
                                    .Where(r => r.date >= startDate && r.date <= endDate)
                                    .ToList();
            }

            // Εμφάνιση των δεδομένων στα σχετικά DataGridView
            if (Day_On_Requests.Count > 0)
            {
              dgDayOnReq.DataSource = Day_On_Requests;
              dgDayOnReq.Columns[1].HeaderText = "Ημερομηνία";
              dgDayOnReq.Columns[2].HeaderText = "Προτεραιότητα";
              dgDayOnReq.Columns[3].HeaderText = "Υπάλληλος";
            }
            else
            {
              dgDayOnReq.DataSource = null;
              dgDayOnReq.Rows.Clear();
            }

            if (Day_On_Requests.Count > 0)
            {
              dgDayOffReq.DataSource = Day_Off_Requests;
              dgDayOffReq.Columns[1].HeaderText = "Ημερομηνία";
              dgDayOffReq.Columns[2].HeaderText = "Προτεραιότητα";
              dgDayOffReq.Columns[3].HeaderText = "Υπάλληλος";
            }
            else
            {
              dgDayOffReq.DataSource = null;
              dgDayOffReq.Rows.Clear();
            }
```

```
            if (Day_On_Requests.Count > 0)
            {
                dgShiftOnReq.DataSource = Shift_On_Requests;
                dgShiftOnReq.Columns[1].HeaderText = "Ημερομηνία";
                dgShiftOnReq.Columns[2].HeaderText = "Προτεραιότητα";
                dgShiftOnReq.Columns[3].HeaderText = "Βάρδια";
                dgShiftOnReq.Columns[4].HeaderText = "Υπάλληλος";
            }
            else
            {
                dgShiftOnReq.DataSource = null;
                dgShiftOnReq.Rows.Clear();
            }


            if (Day_On_Requests.Count > 0)
            {
                dgShiftOffReq.DataSource = Shift_Off_Requests;
                dgShiftOffReq.Columns[1].HeaderText = "Ημερομηνία";
                dgShiftOffReq.Columns[2].HeaderText = "Προτεραιότητα";
                dgShiftOffReq.Columns[3].HeaderText = "Βάρδια";
                dgShiftOffReq.Columns[4].HeaderText = "Υπάλληλος";
            }
            else
            {
                dgShiftOffReq.DataSource = null;
                dgShiftOffReq.Rows.Clear();
            }
        }
    }
  }
}
```

ΦΟΡΜΑ frmProgram

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Windows.Forms;
using NurseRostering.OutputData;
using NurseRostering.InputData;
using Excel = Microsoft.Office.Interop.Excel;
using System.IO;
using Microsoft.Office.Interop.Excel;
using NurseRostering.Resources;
using NurseRostering.Database;

namespace NurseRostering.GUI
```

```
{
   public partial class frmProgram : Form
   {
      NurseRosteringDB nrdb = new NurseRosteringDB();
      Assignment ProgrammaIpallilou;
      List<Assignment> ProgrammaErgasias = new List<Assignment>();
      List<Employee> Nosokomoi = new List<Employee>();
      Employee nurse = new Employee();

      DataGridViewColumn col;
      DataGridViewCell cell;

      private DateTime Enar3i;
      private DateTime Li3i;
      public string eidosProgrammatos { get; set; }

      public frmProgram()
      {
         InitializeComponent();
      }


      private void frmProgram_Load(object sender, EventArgs e)
      {
         /* Έλεγχος για το εάν υπάρχει κάποιο αρχείο δημιουργημένου προγράμματος
          * από το οποίο θα ληφθούν οι ημερομηνίες έναρξης και λήξης αυτού. */
         if (!File.Exists(eidosProgrammatos + "_sol.txt"))
         {
            MessageBox.Show("Δεν υπάρχει κάποιο σχέδιο του προγράμματος
εργασίας.", "NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Error);
            btnExcel.Enabled = false;
            btnPrint.Enabled = false;
            return;
         }

         string line;
         string inputFileLine;
         DateTime result;

         // Ανάγνωση από το αρχείο εισόδου του προγράμματος και πέρασμα των τιμών
γραμμή-γραμμή
         StreamReader inputFile = new StreamReader(eidosProgrammatos + ".txt");

         // Ανάγνωση όλων των γραμμών του αρχείου
         while ((inputFileLine = inputFile.ReadLine()) != null)
         {
            // μέχρι να βρεθεί η κατάλληλη που φέρει το εύρος του προγράμματος
εργασίας. Όταν βρεθεί, λαμβάνεται αυτομάτως η ημερομηνία λήξης του προγράματος
            if (inputFileLine.Length > 9 &&
DateTime.TryParse(inputFileLine.Substring(inputFileLine.TrimEnd(';').Length - 10,
10), out Li3i))
            {
```

```
                // απομάκρυνση περιττών χαρακτήρων και "σπάσιμο" της λέξης σε
κουπόνια
                string[] dates = inputFileLine.TrimEnd(';').Split(',');
                DateTime.TryParse(dates[1], out Enar3i); // Ημερομηνία έναρξης του
προγράμματος εργασίας
                eidosProgrammatos = dates[0]; // Επιλογή χρόνου εκτέλεσης του
προγράμματος
                break;
            }
        }

        inputFile.Close(); // Κλείσιμο του αρχείου εισόδου από το οποίο έγινε λήψη
της ημερομηνίας προγράμματος

        if (!File.Exists(eidosProgrammatos + "_sol.txt")) // Έλεγχος για το εάν υπάρχει
αρχείο δημιουργημένου προγράμματος
        {
            MessageBox.Show("Δεν έχει δημιουργηθεί κάποιο πρόγραμμα του είδους "
+ eidosProgrammatos, "NurseRostering", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            btnExcel.Enabled = false;
            btnPrint.Enabled = false;
            return;
        }

        // Προσθήκη ημερομηνιών έναρξης, λήξης καθώς και της διάρκειας του
προγράμματος στις σχετικές ετικέτες.
        lblStartDate.Text += " " + Enar3i.Date.ToString("dd/MM/yyyy");
        lblEndDate.Text += " " + Li3i.Date.ToString("dd/MM/yyyy");
        lblDateDiff.Text += " " + Li3i.Subtract(Enar3i.AddDays(-1)).Days.ToString()
+ " ημέρες";

        // Ανάγνωση από το αρχείο του προγράμματος και πέρασμα των τιμών
γραμμή-γραμμή
        StreamReader file = new StreamReader(eidosProgrammatos + "_sol.txt");

        col = new DataGridViewColumn();
        cell = new DataGridViewTextBoxCell();
        col.CellTemplate = cell;
        col.AutoSizeMode =
DataGridViewAutoSizeColumnMode.AllCellsExceptHeader;
        col.ReadOnly = true;
        col.Visible = true;

        dgProgram.Columns.Add(col);

        while ((line = file.ReadLine()) != null)
        {
            if (line.Length > 9 && DateTime.TryParse(line.Substring(0, 10).ToString(),
out result))
            {
                bool flag = false;
```

```
                string[] assignment = line.TrimEnd(';').Split(',');

                col = new DataGridViewColumn();
                cell = new DataGridViewTextBoxCell();

                ProgrammaIpallilou = new Assignment();
                ProgrammaIpallilou.date = result;
                ProgrammaIpallilou.employeeID = int.Parse(assignment[1]);
                ProgrammaIpallilou.shiftID = assignment[2];

                ProgrammaErgasias.Add(ProgrammaIpallilou);

                for (int i = 0; i < dgProgram.ColumnCount; i++)
                {
                    if (dgProgram.Columns[i].HeaderText == result.Date.ToString("dd-
MM"))
                    {
                        flag = true;
                    }
                }

                if (!flag)
                {
                    col.CellTemplate = cell;
                    col.HeaderText = result.Date.ToString("dd-MM");
                    col.Name = "Ημερομηνία";
                    col.Width = 40;
                    col.Visible = true;

                    dgProgram.Columns.Add(col);
                }

                assignment = null;
            }

        }

        // Λήψη όλων των στοιχείων προσωπικού
        Nosokomoi = (from n in nrdb.Employees select n).OrderBy(n =>
n.name).ToList();

        // Γραμμή δεδομένων στο DataGridView
        DataGridViewRow dgRow;
        DataGridViewCellStyle style = new DataGridViewCellStyle();

        /* Εμφάνιση των ονομάτων των νοσοκόμων στο πρώτο κελί της κάθε γραμμής
του
         * DataGridView για προβολή του προγράμματος εργασίας του καθενός
νοσοκόμου */
        for (int i = 0; i < Nosokomoi.Count; i++)
        {
            dgRow = new DataGridViewRow(); // Δημιουργία νέας γραμμής
            dgRow.Height = 30; // Ύψος γραμμής
```

```
            dgRow.Visible = true; // Ορατή γραμμή

            // Προσθήκη της γραμμής στο DataGridView
            dgProgram.Rows.Add(dgRow);

            // Εισαγωγή του ονόματος του νοσοκόμου στην κεφαλίδα της γραμμής
            dgProgram.Rows[i].Cells[0].Value = Nosokomoi[i].name;


        }

        /* Παρακάτω, παργματοποιείται η εισαγωγή της βάρδιας εργασίας σε
συγκεκριμένη ημέρα */
        foreach (DataGridViewRow rowItem in dgProgram.Rows)
        {
            // Αναζήτηση του προγράμματος εργασίας κάθε υπαλλήλου, με βάση το
αναγνωριστικό του
            List<Assignment> PE = (from na in ProgrammaErgasias
                                   where na.employeeID == (from x in Nosokomoi // από τον
πίνακα των νοσοκόμων
                                                           where x.name ==
dgProgram.Rows[rowItem.Index].Cells[0].Value.ToString()
                                                           select x.ID).FirstOrDefault()
                                   select na).OrderBy(na => na.date).ToList();

            // Τοποθέτηση των βαρδιών, στον πίνακα του προγράμματος εργασίας.
            foreach (DataGridViewColumn colItem in dgProgram.Columns)
            {
                foreach (var item in PE)
                {
                    if (dgProgram.Columns[colItem.Index].HeaderText.ToString() ==
item.date.Date.ToString("dd-MM"))
                    {
                        dgProgram[colItem.Index, rowItem.Index].Value =
item.shiftID.Trim();

                        if (dgProgram[colItem.Index, rowItem.Index].Value.ToString() ==
"D")
                        {
                            dgProgram[colItem.Index, rowItem.Index].Style = new
DataGridViewCellStyle { BackColor = Color.FromArgb(23, 98, 131) };
                        }
                        else if (dgProgram[colItem.Index, rowItem.Index].Value.ToString()
== "E")
                        {
                            dgProgram[colItem.Index, rowItem.Index].Style = new
DataGridViewCellStyle { BackColor = Color.FromArgb(6, 168, 125) };
                        }
                        else if(dgProgram[colItem.Index, rowItem.Index].Value.ToString()
== "L")
                        {
                            dgProgram[colItem.Index, rowItem.Index].Style = new
DataGridViewCellStyle { BackColor = Color.FromArgb(135, 47, 95) };
```

```
                    }
                    else if (dgProgram[colItem.Index, rowItem.Index].Value.ToString()
== "N")
                    {
                        dgProgram[colItem.Index, rowItem.Index].Style = new
DataGridViewCellStyle { BackColor = Color.FromArgb(241, 162, 8) };
                    }
                    else if (dgProgram[colItem.Index, rowItem.Index].Value.ToString()
== "DH")
                    {
                        dgProgram[colItem.Index, rowItem.Index].Style = new
DataGridViewCellStyle { BackColor = Color.FromArgb(230, 100, 22) };
                    }
                    else
                    {
                        dgProgram[colItem.Index, rowItem.Index].Style = new
DataGridViewCellStyle { BackColor = Color.Red };
                    }
                }
                else if (dgProgram[colItem.Index, rowItem.Index].Value == null)
                {
                    dgProgram[colItem.Index, rowItem.Index].Value = "-";
                }
            }
        }

        PE.Clear();
    }

    ProgrammaErgasias = null;
    ProgrammaIpallilou = null;
    Nosokomoi = null;

    GC.Collect(); // Συλλογή "σκουπιδιών"
}

/*****************************************************
 *****************************************************
 **                                                **
 ** ΜΕΤΑΦΟΡΑ ΤΟΥ ΠΡΟΓΡΑΜΜΑΤΟΣ ΕΡΓΑΣΙΑΣ ΣΕ ΑΡΧΕΙΟ EXCEL
**
 **                                                **
 *****************************************************
 *****************************************************/
private void btnExcel_Click(object sender, EventArgs e)
{
    SaveFileDialog sfd = new SaveFileDialog();
    sfd.Filter = "Excel Documents (*.xlsx)|*.xlsx";
    sfd.FileName = "Πρόγραμμα εργασίας από " + Enar3i.Date.ToString("dd-MM-
yyyy") + " εώς και " + Li3i.Date.ToString("dd-MM-yyyy") + ".xlsx";
    if (sfd.ShowDialog() == DialogResult.OK)
    {
        copyAlltoClipboard(dgProgram);
```

```
            Excel.Application xlexcel;
            Workbook xlWorkBook;
            Worksheet xlWorkSheet;
            object misValue = System.Reflection.Missing.Value;
            xlexcel = new Excel.Application();
            xlexcel.Visible = true;
            xlWorkBook = xlexcel.Workbooks.Add(misValue);
            xlWorkSheet = (Worksheet)xlWorkBook.Worksheets.get_Item(1);
            xlWorkSheet.Cells[2, 10] = label1.Text;
            Range CR = (Range)xlWorkSheet.Cells[3, 1];
            CR.Select();
            xlWorkSheet.PasteSpecial(CR, Type.Missing, Type.Missing, Type.Missing,
Type.Missing, Type.Missing, true);
            xlWorkBook.SaveAs(sfd.FileName);

            releaseObject(xlWorkSheet);
            releaseObject(xlWorkBook);
            releaseObject(xlexcel);

            // Clear Clipboard and DataGridView selection
            Clipboard.Clear();
            dgProgram.ClearSelection();

        }
    }

    private void copyAlltoClipboard(DataGridView pinakas)
    {
        pinakas.SelectAll();
        DataObject dataObj = pinakas.GetClipboardContent();
        if (dataObj != null)
            Clipboard.SetDataObject(dataObj);
    }

    private void releaseObject(object obj)
    {
        try
        {
            System.Runtime.InteropServices.Marshal.ReleaseComObject(obj);
            obj = null;
        }
        catch (Exception ex)
        {
            obj = null;
            MessageBox.Show("Προέκυψε πρόβλημα κατά την αποδέσμευση του
στοιχείου " + ex.ToString(), "NurserRostering", MessageBoxButtons.OK,
MessageBoxIcon.Error);
        }
        finally
        {
            GC.Collect(); // Συλλογή "σκουπιδιών"
        }
    }
```

```
        private void btnPrint_Click(object sender, EventArgs e)
        {
            ClsPrint _ClsPrint = new ClsPrint(dgProgram, "Πρόγραμμα εργασίας από " +
Enar3i.Date.ToString("dd-MM-yyyy") + " εώς και " + Li3i.Date.ToString("dd-MM-
yyyy"));
            _ClsPrint.PrintForm();
        }

        private void btnExit_Click(object sender, EventArgs e)
        {
            // Τερματισμός λειτουργίας αυτής της φόρμας
            this.Close();
        }
    }
}
```

ΦΟΡΜΑ frmSchedulingPeriod

```
using NurseRostering.InputData;
using System;
using System.Collections.Generic;
using System.Data;
using System.Diagnostics;
using System.IO;
using System.Linq;
using System.Threading;
using System.Windows.Forms;
using System.Text;
using NurseRostering.Database;

namespace NurseRostering.GUI
{
    public partial class frmSchedulingPeriod : Form
    {
        //[DllImport("user32.dll")]
        //static extern IntPtr SetParent(IntPtr hWndChild, IntPtr hWndNewParent);
        //[DllImport("user32.dll")]
        //static extern bool ShowWindow(IntPtr hWnd, int nCmdShow);
        //const int SW_HIDE = 0;

        public frmSchedulingPeriod()
        {
            InitializeComponent();
        }

        Modules module = new Modules();
        NurseRosteringDB nrdb = new NurseRosteringDB();
        Scheduling_Period newPeriod;
```

```
        Day_Of_Week_Cover DayOfWeekCover;
        Date_Specific_Cover DateSpecificCover;
        List<string> ListOfShifts = new List<string>();
        List<Day_Of_Week_Cover> ListOfDayOWC = new
List<Day_Of_Week_Cover>();
        List<Date_Specific_Cover> ListOfDSC = new List<Date_Specific_Cover>();
        List<Day_Of_Week_Cover> DayByDay = new List<Day_Of_Week_Cover>();

        private string eidosProgrammatos = "";
        private DateTime imerominiaEnar3is = DateTime.Now.Date;
        private DateTime imerominiaLi3is = DateTime.Now.Date;
        private string komma = ", ";

        // Επιλεγμένη ημέρα της εβδομάδας για απαίτηση κάλυψης με κάποιο πλήθος
προσωπικού
        private string epilegmeniImera = "";

        private void btnCancel_Click(object sender, EventArgs e)
        {
            // Κλείσιμο της παρούσας φόρμας και εμφάνιση αυτής που την κάλεσε
            this.Close();
            GC.Collect(); // Συλλογή "σκουπιδιών"
        }

        #region ΕΝΕΡΓΕΙΕΣ ΜΕ ΒΑΣΗ ΤΙΣ ΕΠΙΛΟΓΕΣ ΓΙΑ ΤΟ ΑΝΤΙΚΕΙΜΕΝΟ
SCHEDULING PERIOD

        /* Ενημέρωση του χρήστη για την εκτιμώμενη διάρκεια δημιουργίας του
προγράμματος εργασίας ανάλογα με την επιλογή του,
         * σύμφωνα με τους χρόνους που έχουν ανακοινωθεί για τον αλγόριθμο. Τυχόν
αποκλίσεις οφείλονται στη χρήση διαφορετικών
         * υπολογιστικών συστημάτων από αυτά στα οποία μετρήθηκε ο αλγόριθμος */

        // Radio button ταχείας δημιουργίας
        private void rbSprint_CheckedChanged(object sender, EventArgs e)
        {
            if (rbSprint.Checked)
            {
                lblEstimatedTime.Text = "Εκτιμώμενος χρόνος ολοκλήρωσης
προγράμματος 13 δευτερόλεπτα";

                // Καθορισμός του αρχείου εισόδου
                eidosProgrammatos = "sprint";
            }
        }

        // Radio button προσεγμένου προγράμματος
        private void rbMedium_CheckedChanged(object sender, EventArgs e)
        {
            if (rbMedium.Checked)
            {
                lblEstimatedTime.Text = "Εκτιμώμενος χρόνος ολοκλήρωσης
προγράμματος 13 λεπτά";
```

```
            // Καθορισμός του αρχείου εισόδου
            eidosProgrammatos = "medium";
        }
    }


    // Radio button λεπτομερούς αναζήτησης
    private void rbLong_CheckedChanged(object sender, EventArgs e)
    {
        if (rbLong.Checked)
        {
            lblEstimatedTime.Text = "Εκτιμώμενος χρόνος ολοκλήρωσης
προγράμματος 13 ώρες";

            // Καθορισμός του αρχείου εισόδου
            eidosProgrammatos = "long";
        }
    }

    #endregion

    #region ΕΝΕΡΓΕΙΕΣ ΜΕ ΒΑΣΗ ΤΙΣ ΕΠΙΛΟΓΕΣ ΓΙΑ ΤΟ ΑΝΤΙΚΕΙΜΕΝΟ
DAY OF WEEK COVER

    /* Δημιουργία νέας καταχώρησης, εμφάνισή της στη λίστα διεπαφής
     * και προσθήκη της στη σχετική λίστα αντικειμένων */
    private void btnAdd_Click(object sender, EventArgs e)
    {
        // Έλεγχοι για το αν έχουν γίνει οι απαραίτητες επιλογές
        if (module.CheckIfAChoiceMade(cbDays))
        {
            if (module.CheckIfAChoiceMade(cbShifts))
            {
                string check = epilegmeniImera + komma
                    + cbShifts.SelectedItem.ToString() + komma
                    + nudNurses.Value;
                if (module.CheckForDoubleRecords(check, lstDayCover))
                {
                    int DR = 0; // Μετρητής διπλών εγγραφών στη βάση δεδομένων
                                // Αναζήτηση για τυχόν ύπαρξη αυτής της εγγραφής και
αποφυγή διπλών εγγραφών στη βάση δεδομένων
                    DR = nrdb.Day_Of_Week_Covers.Count(x => x.weekDay ==
epilegmeniImera && x.shift == cbShifts.SelectedItem.ToString());

                    if (DR > 0) // Εάν βρεθεί μια εγγραφή έστω
                    {
                        // Ενημέρωση του χρήστη με σχετικό μήνυμα
                        MessageBox.Show("Υπάρχει ήδη μια εγγραφή στη βάση δεδομένων
με αυτή την ημερομηνία και αυτή τη βάρδια", "NurseRostering",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                        return; // Τερματισμός διαδικασίας
                    }
```

```
                // Αφού ολοκληρώθηκαν επιτυχώς οι έλεγχοι
                DayOfWeekCover = new Day_Of_Week_Cover(); // Δημιουργία νέου
αντικειμένου

                // Τοποθέτηση τιμών στις ιδιότητες του αντικειμένου
                DayOfWeekCover.weekDay = epilegmeniImera;
                DayOfWeekCover.shift = cbShifts.SelectedItem.ToString();
                DayOfWeekCover.preferred = (int)nudNurses.Value;

                // Εμφάνιση δεδομένων στη λίστα
                lstDayCover.Items.Add(DayOfWeekCover.ToString().TrimEnd(';'));

                // Τοποθέτηση του αντικειμένου στη λίστα προετοιμάζοντάς το για
καταχώρηση στη βάση δεδομένων
                ListOfDayOWC.Add(DayOfWeekCover);
            }
        }
    }
}

// Διαγραφή μιας επιλεγμένης εγγραφής
private void btnDel_Click(object sender, EventArgs e)
{
    if (lstDayCover.SelectedIndex > -1)
    {
        // Εύρεση της κατάλληλης εγγραφής από τη λίστα αντικειμένων, για
διαγραφή
        var record = ListOfDayOWC.Find(x => x.ToString().TrimEnd(';') ==
lstDayCover.SelectedItem.ToString());

        // Διαγραφή εγγραφής από τη λίστα αντικειμένων
        ListOfDayOWC.Remove(record);

        // αλλά και από τη λίστα της διεπαφής
        lstDayCover.Items.RemoveAt(lstDayCover.SelectedIndex);
    }
}

// Καθαρισμός όλων των σχετικών λιστών
private void btnDelAll_Click(object sender, EventArgs e)
{
    // Καθαρισμός της λίστας αντικειμένων
    ListOfDayOWC.Clear();

    // αλλά και της λίστας της διεπαφής
    lstDayCover.Items.Clear();
}

#endregion

#region ΕΝΕΡΓΕΙΕΣ ΜΕ ΒΑΣΗ ΤΙΣ ΕΠΙΛΟΓΕΣ ΓΙΑ ΤΟ ΑΝΤΙΚΕΙΜΕΝΟ
DATE SPECIFIC COVER
```

```
        private void btnDSCAdd_Click(object sender, EventArgs e)
        {
            // Έλεγχοι για το αν έχουν γίνει οι απαραίτητες επιλογές
            if (module.CheckIfAChoiceMade(cbShiftsSC))
            {

                string check = mcSpecificCover.SelectionRange.Start.ToString("yyyy-MM-
dd") + komma +
                    cbShiftsSC.SelectedItem.ToString() + komma + nudSpecificCover.Value;

                // Έλεγχος για διπλές εγγραφές
                if (module.CheckForDoubleRecords(check, lstSpecificCover))
                {
                    int DR = 0; // Μετρητής διπλών εγγραφών στη βάση δεδομένων
                    // Αναζήτηση για τυχόν ύπαρξη αυτής της εγγραφής και αποφυγή διπλών
εγγραφών στη βάση δεδομένων
                    DateTime imerominia = mcSpecificCover.SelectionRange.Start.Date;
                    DR = nrdb.Date_Specific_Covers.Count(x => x.weekDay == imerominia
&& x.shift == cbShiftsSC.SelectedItem.ToString());

                    if (DR > 0) // Εάν βρεθεί μια εγγραφή έστω
                    {
                        // Ενημέρωση του χρήστη με σχετικό μήνυμα
                        MessageBox.Show("Υπάρχει ήδη μια εγγραφή στη βάση δεδομένων με
αυτή την ημερομηνία και αυτή τη βάρδια", "NurseRostering",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                        return; // Τερματισμός διαδικασίας
                    }

                    // Αφού ολοκληρώθηκαν επιτυχώς οι έλεγχοι
                    DateSpecificCover = new Date_Specific_Cover(); // Δημιουργία νέου
αντικειμένου

                    // Τοποθέτηση τιμών στις ιδιότητες του αντικειμένου
                    DateSpecificCover.weekDay =
mcSpecificCover.SelectionRange.Start.Date;
                    DateSpecificCover.shift = cbShiftsSC.SelectedItem.ToString();
                    DateSpecificCover.preferred = (int)nudSpecificCover.Value;

                    // Εμφάνιση δεδομένων στη λίστα
                    lstSpecificCover.Items.Add(DateSpecificCover.ToString().TrimEnd(';'));

                    // Τοποθέτηση του αντικειμένου στη λίστα προετοιμάζοντάς το για
καταχώρηση στη βάση δεδομένων
                    ListOfDSC.Add(DateSpecificCover);
                }

            }
        }

        // Διαγραφή μιας επιλεγμένης εγγραφής
        private void btnDSCDel_Click(object sender, EventArgs e)
```

```
        {
            if (lstSpecificCover.SelectedIndex > -1)
            {
                // Εύρεση της κατάλληλης εγγραφής από τη λίστα αντικειμένων, για
διαγραφή
                var record = ListOfDayOWC.Find(x => x.ToString().TrimEnd(';') ==
lstSpecificCover.SelectedItem.ToString());

                // Διαγραφή εγγραφής από τη λίστα αντικειμένων
                ListOfDayOWC.Remove(record);

                // αλλά και από τη λίστα της διεπαφής
                lstSpecificCover.Items.RemoveAt(lstSpecificCover.SelectedIndex);
            }
        }

        // Καθαρισμός όλων των σχετικών λιστών
        private void btnDSCDelAll_Click(object sender, EventArgs e)
        {
            // Καθαρισμός της λίστας αντικειμένων
            ListOfDayOWC.Clear();

            // αλλά και της λίστας της διεπαφής
            lstSpecificCover.Items.Clear();
        }

        #endregion

        // Προετοιμασία αντικειμένων και αποθήκευση στη βάση δεδομένων
        private void btnSubmit_Click(object sender, EventArgs e)
        {
            // Εάν υπάρχει έστω και ένα αντικείμενο
            if (ListOfDSC.Count > 0)
            {
                foreach (var item in ListOfDSC)
                {
                    try
                    {
                        // Μετακίνησέ το στη λίστα εγγραφής στη βάση δεδομένων
                        nrdb.Date_Specific_Covers.Add(item);
                    }
                    catch (Exception ex) // Σύλληψη σφάλματος
                    {
                        // Ενημέρωση του χρήστη σχετικά με το σφάλμα που προέκυψε
                        MessageBox.Show("Προέκυψε κάποιο απροσδόκητο σφάλμα. Σφάλμα:
" + ex.ToString(), "NurseRostering", MessageBoxButtons.OK,
MessageBoxIcon.Error);
                    }
                }
                ListOfDSC.Clear(); // Καθαρισμός λίστας
            }

            try
```

```
            {
                // Αποθήκευση αλλαγών στη βάση δεδομένων
                nrdb.SaveChanges();

                // Ενημέρωση του χρήστη σχετικά με την επιτυψημένη καταχώρηση
                MessageBox.Show("Οι αλλαγές καταχωρήθηκαν επιτυχώς.",
"NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Information);
            }
            catch (Exception ex) // Σύλληψη σφάλματος
            {
                // Ενημέρωση του χρήστη σχετικά με το σφάλμα που προέκυψε
                MessageBox.Show("Προέκυψε κάποιο απροσδόκητο σφάλμα. Σφάλμα: " +
ex.ToString(), "NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }

        #region ΕΝΕΡΓΕΙΕΣ ΠΟΥ ΑΦΟΡΟΥΝ ΓΕΝΙΚΑ ΣΤΗΝ ΠΑΡΟΥΣΑ ΦΟΡΜΑ
ΚΑΙ ΟΧΙ ΣΕ ΚΑΠΟΙΑ ΣΥΓΚΕΚΡΙΜΜΕΝΗ
        private void btnPreview_Click(object sender, EventArgs e)
        {
            // Δημιουργία νέας φόρμας προεπισκόπισης δεδομένων εισόδου προγράμματος
            frmPreview preview = new frmPreview();

            // Εμφάνιση της φόρμας προεπισκόπισης και απόκρυψη της παρούσας
            preview.Show();
            this.Hide();

            // Δημιουργία χειριστηρίου για τον έλεγχο κλεισίματος της φόρμας
            preview.FormClosing += KleisimoFormas;
        }

        private void frmSchedulingPeriod_Load(object sender, EventArgs e)
        {
            // Λήψη των τύπων βαρδιών από τη βάση δεδομένων
            ListOfShifts = (from s in nrdb.Shift_Types select s.tipos).Distinct().ToList();

            // και τοποθέτησή τους στα σχετικά αντικείμενα της διεπαφής
            foreach (var item in ListOfShifts)
            {
                cbShifts.Items.Add(item); // Λίστα βαρδιών ημερήσιας κάλυψης
                cbShiftsSC.Items.Add(item); // Λίστα βαρδιών ιδιαίτερης ημερήσιας
κάλυψης
            }

            // Ανανέωση περιεχομένων των αντικειμένων.
            cbShifts.Refresh();
            cbShiftsSC.Refresh();

            // Προεπιλογή τύπου προγράμματος
            rbSprint.Checked = true;
        }

        private void KleisimoFormas(object sender, FormClosingEventArgs e)
```

```
        {
            this.Show(); // Εμφάνιση κύριας φόρμας, όταν κλείσει η φόρμα απαιτήσεων
προγράμματος εργασίας
        }
        #endregion

        #region ΕΓΓΡΑΦΗ ΣΤΟ ΑΡΧΕΙΟ ΕΙΣΟΔΟΥ ΚΑΙ ΔΗΜΙΟΥΡΓΙΑ ΝΕΟΥ
ΝΗΜΑΤΟΣ ΜΕ ΝΕΑ ΔΙΕΡΓΑΣΙΑ

        // Δημιουργία αρχείου εισόδου του αλγορίθμου
        private void PrintToFile()
        {
            Enotita Chapter;

            /* Λίστες στις οποίες θα φορτωθούν όλα τα δεδομένα από τη βάση, για να
προβληθούν
             * στα σχετικά datagridviews της διεπαφής, προς ενημέρωση και έλεγχο από
τον χρήστη. */
            List<Contract> Contracts = new List<Contract>();
            List<Date_Specific_Cover> Date_Specific_Covers = new
List<Date_Specific_Cover>();
            List<Day_Of_Week_Cover> Day_Of_Week_Covers = new
List<Day_Of_Week_Cover>();
            List<Day_On_Request> Day_On_Requests = new List<Day_On_Request>();
            List<Day_Off_Request> Day_Off_Requests = new List<Day_Off_Request>();
            List<Employee> Employees = new List<Employee>();
            List<Available_Skill> Available_Skills = new List<Available_Skill>();
            List<Pattern> Patterns = new List<Pattern>();
            List<Scheduling_Period> Scheduling_Periods = new
List<Scheduling_Period>();
            List<Shift_On_Request> Shift_On_Requests = new
List<Shift_On_Request>();
            List<Shift_Off_Request> Shift_Off_Requests = new
List<Shift_Off_Request>();
            List<Shift_Type> Shift_Types = new List<Shift_Type>();
            List<Available_Pattern> AvPatterns = new List<Available_Pattern>();
            List<Skill> EmpSkills = new List<Skill>();

            /* Άντληση δεδομένων από τη βάση. Η συγκεκριμένη ενέργεια,
πραγματοποιείται εκ νέου,
             * για να φορτωθεί το τελευταίο αποθηκευμένο στιγμιότυπο της βάσης
δεδομένων έχοντας
             * όλα τα δεδομένα αυτής σύμφωνα με μερικά απαιτούμενα κριτήρια. */
            DateTime d1 = mcNewPeriodStart.SelectionRange.Start.Date;
            DateTime d2 = mcNewPeriodEnd.SelectionRange.Start.Date;

            Scheduling_Periods = (from r in nrdb.Scheduling_Periods
                                  where (r.startDate == d1 &&
                                  r.endDate == d2)
                                  select r).ToList();


            /* Οι μεταβλητές αυτές χρησιμοποιούνται για αναζήτηση αιτήσεων
```

```
         * των υπαλλήλων από τη βάση δεδομένων με βάση το εύρος του
προγράμμαος */
        DateTime schStartDate;
        DateTime schStopDate;

        // Έλεγχος για το αν υπάρχει κάποιο καταχωρημένο πρόγραμμα εργασίας
        if (Scheduling_Periods.Count > 0) // Εάν υπάρχει
        {
           // Λήψη των αποθηκευμένων ημερομηνιών
           schStartDate = Scheduling_Periods[0].startDate.Date;
           schStopDate = Scheduling_Periods[0].endDate.Date;
        }
        else // Στην περίπτωση που δεν υπάρχει
        {
           // Ενημέρωση του χρήστη ότι δεν είναι δυνατη η συνέχεια και τερματισμός
της διαδικασίας
           MessageBox.Show("Δεν βρέθηκε κάποιο χρονικό διάστημα που να ορίζει
ένα πρόγραμμα εργασίας. Η συνέχεια δεν είναι δυνατή.",
              "NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Error);
           return;
        }

        Contracts = (from r in nrdb.Contracts select
r).OrderByDescending(r=>r.ID).ToList();
        Date_Specific_Covers = (from r in nrdb.Date_Specific_Covers select r)
           .Where(r => r.weekDay >= schStartDate && r.weekDay <= schStopDate)
           .ToList();
        Day_Of_Week_Covers = (from r in nrdb.Day_Of_Week_Covers select
r).OrderBy(r=>r.ID).ToList();
        Day_On_Requests = (from r in nrdb.Day_On_Requests select r)
           .Where(r => r.date >= schStartDate && r.date <= schStopDate)
           .ToList();
        Day_Off_Requests = (from r in nrdb.Day_Off_Requests select r)
           .Where(r=>r.date >= schStartDate && r.date <= schStopDate)
           .ToList();
        Employees = (from r in nrdb.Employees select
r).OrderByDescending(r=>r.ID).ToList();
        Available_Skills = (from r in nrdb.Available_Skills select r).ToList();
        Patterns = (from r in nrdb.Patterns select
r).OrderByDescending(r=>r.ID).ToList();
        Shift_On_Requests = (from r in nrdb.Shift_On_Requests select r)
           .Where(r => r.date >= schStartDate && r.date <= schStopDate)
           .ToList();
        Shift_Off_Requests = (from r in nrdb.Shift_Off_Requests select r)
           .Where(r => r.date >= schStartDate && r.date <= schStopDate)
           .ToList();
        Shift_Types = (from r in nrdb.Shift_Types select r).ToList();
        AvPatterns = (from r in nrdb.AvailablePatterns select r).ToList();


        // Προετοιμασία δεδομένων για εγγραφή στο αρχείο
        string input = ""; // Μεταβλητή δεδομένων εγγραφής
```

```
        // Διάρκεια προγράμματος εργασίας
        Chapter = new Enotita("SCHEDULING_PERIOD"); // Τίτλος ενότητας
        input = Chapter.ToString();
        input += Scheduling_Periods[0].ToString() + "\r\n\r\n\r\n"; // Δεδομένα
ενότητας

        // Ειδικότητες και πλήθος αυτών
        Chapter.ChapterData("SKILLS", Available_Skills.Count); // Τίτλος ενότητας
και πλήθος αντικειμένων
        input += Chapter.ToString();
        foreach(var item in Available_Skills)
        {
          input += item.ToString() + "\r\n"; // Δεδομένα ενότητας
        }
        input += "\r\n\r\n"; // Προσθήκη κενών γραμμών

        // Βάρδιες και πλήθος αυτών
        Chapter.ChapterData("SHIFT_TYPES", Shift_Types.Count); // Τίτλος
ενότητας και πλήθος αντικειμένων
        input += Chapter.ToString();
        foreach(var item in Shift_Types)
        {
          input += item.ToString() + "\r\n"; // Δεδομένα ενότητας
        }
        input += "\r\n\r\n"; // Προσθήκη κενών γραμμών

        // Συμβόλαια και πλήθος αυτών
        Chapter.ChapterData("CONTRACTS", Contracts.Count); // Τίτλος ενότητας
και πλήθος αντικειμένων
        input += Chapter.ToString();
        foreach (var item in Contracts)
        {
          input += item.ToString() + "\r\n"; // Δεδομένα ενότητας
        }
        input += "\r\n\r\n"; // Προσθήκη κενών γραμμών

        // Πρότυπα ανεπιθύμητης εργασίας και πλήθος αυτών
        Chapter.ChapterData("PATTERNS", Patterns.Count); // Τίτλος ενότητας και
πλήθος αντικειμένων
        input += Chapter.ToString();
        foreach (var item in Patterns)
        {
          input += item.ToString() + "\r\n"; // Δεδομένα ενότητας
        }
        input += "\r\n\r\n"; // Προσθήκη κενών γραμμών

        // Υπάλληλοι και πλήθος αυτών
        Chapter.ChapterData("EMPLOYEES", Employees.Count); // Τίτλος ενότητας
και πλήθος αντικειμένων
        input += Chapter.ToString();
        foreach (var item in Employees)
        {
          input += item.DataForInputFile() + "\r\n"; // Δεδομένα ενότητας
```

```
            }
            input += "\r\n\r\n"; // Προσθήκη κενών γραμμών

            // Απαιτήσεις σε προσωπικό για κάθε ημέρα της εβδομάδας και πλήθος αυτών
            Chapter.ChapterData("DAY_OF_WEEK_COVER",
Day_Of_Week_Covers.Count); // Τίτλος ενότητας και πλήθος αντικειμένων
            input += Chapter.ToString();
            foreach (var item in Day_Of_Week_Covers)
            {
                input += item.ToString() + "\r\n"; // Δεδομένα ενότητας
            }
            input += "\r\n\r\n"; // Προσθήκη κενών γραμμών

            // Απαιτήσεις σε προσωπικό για κάθε ημέρα της εβδομάδας και πλήθος αυτών
            Chapter.ChapterData("DATE_SPECIFIC_COVER",
Date_Specific_Covers.Count); // Τίτλος ενότητας και πλήθος αντικειμένων
            input += Chapter.ToString();
            foreach (var item in Date_Specific_Covers)
            {
                input += item.ToString() + "\r\n"; // Δεδομένα ενότητας
            }
            input += "\r\n\r\n"; // Προσθήκη κενών γραμμών

            // Αιτήσεις προσωπικού για ανάπαυση σε συγκεκριμένη ημέρα και πλήθος
αυτών
            Chapter.ChapterData("DAY_OFF_REQUESTS", Day_Off_Requests.Count);
// Τίτλος ενότητας και πλήθος αντικειμένων
            input += Chapter.ToString();
            foreach (var item in Day_Off_Requests)
            {
                input += item.ToString() + "\r\n"; // Δεδομένα ενότητας
            }
            input += "\r\n\r\n"; // Προσθήκη κενών γραμμών

            // Αιτήσεις προσωπικού για εργασία σε συγκεκριμένη ημέρα και πλήθος αυτών
            Chapter.ChapterData("DAY_ON_REQUESTS", Day_On_Requests.Count); //
Τίτλος ενότητας και πλήθος αντικειμένων
            input += Chapter.ToString();
            foreach (var item in Day_On_Requests)
            {
                input += item.ToString() + "\r\n"; // Δεδομένα ενότητας
            }
            input += "\r\n\r\n"; // Προσθήκη κενών γραμμών

            // Αιτήσεις προσωπικού για ανάπαυση σε συγκεκριμένη ημέρα και βάρδια και
πλήθος αυτών
            Chapter.ChapterData("SHIFT_OFF_REQUESTS",
Shift_Off_Requests.Count); // Τίτλος ενότητας και πλήθος αντικειμένων
            input += Chapter.ToString();
            foreach (var item in Shift_Off_Requests)
            {
                input += item.ToString() + "\r\n"; // Δεδομένα ενότητας
            }
```

```
            input += "\r\n\r\n"; // Προσθήκη κενών γραμμών

            // Αιτήσεις προσωπικού για εργασία σε συγκεκριμένη ημέρα και βάρδια και
πλήθος αυτών
            Chapter.ChapterData("SHIFT_ON_REQUESTS", Shift_On_Requests.Count);
// Τίτλος ενότητας και πλήθος αντικειμένων
            input += Chapter.ToString();
            foreach (var item in Shift_On_Requests)
            {
                input += item.ToString() + "\r\n"; // Δεδομένα ενότητας
            }
            input += "\r\n\r\n"; // Προσθήκη κενών γραμμών

            File.WriteAllText(eidosProgrammatos + ".txt", input, Encoding.Default);
        }

        private void btnCreateProgram_Click(object sender, EventArgs e)
        {
            /* Ασφαλιστικά μέτρα για το ότι ο αλγόριθμος δεν θα εκκινήσει τη λειτουργία
του
             * χωρίς να είναι καταχωρημένα τα απαραίτητα δεδομένα στη βάση δεδομένων
*/

            // ΓΙΑ ΤΙΣ ΕΙΔΙΚΟΤΗΤΕΣ
            if(nrdb.Available_Skills.Count() == 0)
            {
                MessageBox.Show("Δεν υπάρχει καταχωρημένη καμιά ειδικότητα. Η
συνέχεια δεν είναι δυνατή.", "NurseRostering", MessageBoxButtons.OK,
MessageBoxIcon.Error);

                // Δημιουργία νέας φόρμας καταχώρησης νέου υπαλλήλου
                frmAddSkill newSkill = new frmAddSkill();

                // Αλλαγή κειμένου κουμπιού επιστροφής
                newSkill.btnCancel.Text = "Επιστροφή στη δημιουργία προγράμματος
εργασίας";
                newSkill.Show(); // Εμφάνιση φόρμας καταχώρησης νέου υπαλλήλου
                this.Hide(); // Κρύψιμο της παρούσας φόρμας

                // Δημιουργία χειριστηρίου για τον έλεγχο κλεισίματος της φόρμας
                newSkill.FormClosing += KleisimoFormas;

                // Τερματισμός διαδικασίας
                return;
            }

            // ΓΙΑ ΤΙΣ ΒΑΡΔΙΕΣ
            if (nrdb.Shift_Types.Count() == 0)
            {
                MessageBox.Show("Δεν υπάρχει καταχωρημένη καμιά βάρδια. Η συνέχεια
δεν είναι δυνατή.", "NurseRostering", MessageBoxButtons.OK,
MessageBoxIcon.Error);
```

```
            // Δημιουργία νέας φόρμας καταχώρησης νέας βάρδιας
            frmNewShift newShift = new frmNewShift();

            // Αλλαγή κειμένου κουμπιού επιστροφής
            newShift.btnReturn.Text = "Επιστροφή στη δημιουργία προγράμματος
εργασίας";
            newShift.Show(); // Εμφάνιση φόρμας καταχώρησης νέας βάρδιας
            this.Hide(); // Κρύψιμο της παρούσας φόρμας

            // Δημιουργία χειριστηρίου για τον έλεγχο κλεισίματος της φόρμας
            newShift.FormClosing += KleisimoFormas;

            // Τερματισμός διαδικασίας
            return;
        }

        // ΓΙΑ ΤΑ ΣΥΜΒΟΛΑΙΑ
        if (nrdb.Contracts.Count() == 0)
        {
            MessageBox.Show("Δεν υπάρχει καταχωρημένο κανένα συμβόλαιο. Η
συνέχεια δεν είναι δυνατή.", "NurseRostering", MessageBoxButtons.OK,
MessageBoxIcon.Error);

            /* Δημιουργία και φόρτωση νέας φόρμας σχετική με την καταχώρηση στη
βάση δεδομένων
            * όλων των απαιτήσεων για τη δημιουργία του προγράμματος εργασίας. */
            frmContracts Requirements = new frmContracts();

            Requirements.Show(); // Εμφάνιση της φόρμας
            this.Hide(); // Απόκρυψη της φόρμας του κύριου μενού

            // Δημιουργία χειριστηρίου για τον έλεγχο κλεισίματος της φόρμας
            Requirements.FormClosing += KleisimoFormas;

            // Τερματισμός διαδικασίας
            return;
        }

        /* ΓΙΑ ΤΑ ΠΡΟΤΥΠΑ ΑΝΕΠΙΘΥΜΗΤΗΣ ΕΡΓΑΣΙΑΣ ΣΕ ΣΥΝΔΥΑΣΜΟ
ΜΕ ΤΟ ΠΛΗΘΟΣ ΤΩΝ
         * ΠΡΟΤΥΠΩΝ ΑΝΕΠΙΘΥΜΗΤΗΣ ΕΡΓΑΣΙΑΣ ΠΟΥ ΕΙΝΑΙ
ΚΑΤΑΧΩΡΗΜΕΝΟ ΣΕ ΚΑΘΕ ΣΥΜΒΟΛΑΙΟ */

        /* Επιλέγεται ο έλεγχος του πλήθους των προτύπων ανεπιθύμητης εργασίας σε
συνδυασμό με το
         * πλήθος των προτύπων ανεπιθύμητης εργασίας που είναι καταχωρημένο σε
κάθε συμβόλαιο γιατί
         * είναι επιτρεπτό από τον αλγόριθμο να μην υπάρχει κανένα πρότυπο
ανεπιθύμητης εργασίας
         * όταν και μόνο όταν δεν υπάρχει καταχωρημένο κάποιο πρότυπο σε κάποιο
συμβόλαιο. Η ύπαρξη
         * κάποιου προτύπου ανεπιθύμητης εργασίας σε συμβόλαιο και όχι, πλέον, στη
βάση δεδομένων
```

```
         * θα μπορούσε να προκύψει από τη δημιουργία του προτύπου, την
καταχώρησή του σε κάποιο συμβόλαιο
         * και στη συνέχεια την απώλειά του από τη βάση δεδομένων. */
        if (nrdb.Patterns.Count() == 0 &&
(nrdb.Contracts.Select(x=>x.numberOfUnwantedPatterns > 0).Count() > 0))
        {
            MessageBox.Show("Δεν υπάρχει καταχωρημένο κανένα συμβόλαιο. Η
συνέχεια δεν είναι δυνατή.", "NurseRostering", MessageBoxButtons.OK,
MessageBoxIcon.Error);

            /* Δημιουργία και φόρτωση νέας φόρμας σχετική με την καταχώρηση στη
βάση δεδομένων
            * όλων των απαιτήσεων για τη δημιουργία του προγράμματος εργασίας. */
            frmContracts Requirements = new frmContracts();

            Requirements.Show(); // Εμφάνιση της φόρμας
            this.Hide(); // Απόκρυψη της φόρμας του κύριου μενού

            // Δημιουργία χειριστηρίου για τον έλεγχο κλεισίματος της φόρμας
            Requirements.FormClosing += KleisimoFormas;

            // Τερματισμός διαδικασίας
            return;
        }

        // ΓΙΑ ΤΟΥΣ ΥΠΑΛΛΗΛΟΥΣ
        if (nrdb.Employees.Count() == 0)
        {
            MessageBox.Show("Δεν υπάρχει καταχωρημένο κανένας υπάλληλος. Η
συνέχεια δεν είναι δυνατή.", "NurseRostering", MessageBoxButtons.OK,
MessageBoxIcon.Error);

            // Δημιουργία νέας φόρμας καταχώρησης νέου υπαλλήλου
            frmEmployee newEmp = new frmEmployee();

            // Αλλαγή κειμένου κουμπιού επιστροφής
            newEmp.btnExit.Text = "Επιστροφή στη δημιουργία προγράμματος
εργασίας";
            newEmp.Show(); // Εμφάνιση φόρμας καταχώρησης νέου υπαλλήλου
            this.Hide(); // Κρύψιμο της παρούσας φόρμας

            // Δημιουργία χειριστηρίου για τον έλεγχο κλεισίματος της φόρμας
            newEmp.FormClosing += KleisimoFormas;

            // Τερματισμός διαδικασίας
            return;
        }

        // ΓΙΑ ΤΗΝ ΚΑΛΥΨΗ ΣΕ ΠΡΟΣΩΠΙΚΟ ΚΑΘΕ ΒΑΡΔΙΑΣ ΓΙΑ ΚΑΘΕ
ΗΜΕΡΑ ΤΗΣ ΕΒΔΟΜΑΔΑΣ
        if (nrdb.Day_Of_Week_Covers.Count() == 0)
        {
```

```
            MessageBox.Show("Δεν υπάρχει καταχωρημένη καμία απαίτηση ημερήσιας
σε προσωπικό για κάθε βάρδια της κάθε ημέρας της εβδομάδας. Η συνέχεια δεν είναι
δυνατή.", "NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Error);

            // Τερματισμός διαδικασίας
            return;
        }
        nrdb.SaveChanges(); // Αποθήκευση αλλαγών που μπορεί να
πραγματοποιήθηκαν από τον χρήστη την τελευταία στιγμή
        // Διαπίστωση αν το χρονικό διάστημα τον υπο δημιουργία προγράμματος
είναι τουλάχιστον αυριανό.
        if(module.ValidateDates(mcNewPeriodStart.SelectionRange.Start.Date,
mcNewPeriodEnd.SelectionRange.Start.Date))
        {
            imerominiaEnar3is = mcNewPeriodStart.SelectionRange.Start.Date;
            imerominiaLi3is = mcNewPeriodEnd.SelectionRange.Start.Date;

            DialogResult apantisi = MessageBox.Show("Πρόκειται να δημιουργήσετε
πρόγραμμα εργασίας για την περίοδο: \n Ημερομηνία έναρξης: "
            + imerominiaEnar3is.ToString("dd/MM/yyyy")
            + "\n" + " Ημερομηνία λήξης :" +
imerominiaLi3is.ToString("dd/MM/yyyy")
            + "\n Θέλετε να συνεχίσετε;", "NurseRostering",
MessageBoxButtons.YesNo, MessageBoxIcon.Question);

            PrintToFile(); // Δημιουργία αρχείου εισόδου

            // Δημιουργία και φόρτωση νέας φόρμας για να αντιλαμβάνεται ο χρήστης
ότι δεν έχει "παγώσει" η εφαρμογή
            frmUnderCon Under = new frmUnderCon();

            Under.Show(); // Εμφάνιση της φόρμας

            // Δημιουργία χειριστηρίου για τον έλεγχο κλεισίματος της φόρμας
            Under.FormClosing += KleisimoFormas;

            this.Hide(); // Απόκρυψη της φόρμας του κύριου μενού

            Thread th = new Thread(() =>
            {
                Thread.CurrentThread.IsBackground = true;

                //Δημιουργία νέας διεργασίας
                Process proc = new Process();

                // Καθορισμός προγράμματος που θα εκτελεστεί από τη νέα διεργασία
                proc.StartInfo.FileName = Path.Combine(eidosProgrammatos + ".exe");

                // Κατάσταση παραθύρου
                proc.StartInfo.WindowStyle = ProcessWindowStyle.Hidden;
                proc.StartInfo.CreateNoWindow = true;

                proc.StartInfo.RedirectStandardOutput = false;
```

```
                proc.StartInfo.UseShellExecute = true;
                proc.Start();// Έναρξη εκτέλεσης διεργασίας
                proc.WaitForExit(); // Αναμονή για την ολοκλήρωση της διεργασίας

            });

            th.Start(); // Εκκίνηση νήματος που εκτελεί τον αλγόριθμο δημιουργίας
προγράμματος

            // "Παγίδα" για να παραμένει ενεργή η εφαρμογή όσο εκτελείται ο
αλγόριθμος
            while (th.IsAlive)
            {
                Application.DoEvents();
            }
            Under.Close();

            // Εμφάνιση του προγράμματος στον χρήστη
            frmProgram Programma = new frmProgram();

            Programma.eidosProgrammatos = eidosProgrammatos; // Ανακοίνωση στη
φόρμα του είδους του προγράμματος που δημιουργήθηκε
            Programma.Show(); // Εμφάνιση της φόρμας

            // Δημιουργία χειριστηρίου για τον έλεγχο κλεισίματος της φόρμας
            Programma.FormClosing += KleisimoFormas;

            this.Hide(); // Απόκρυψη της φόρμας του κύριου μενού
        }
        else
        {
            MessageBox.Show("Δεν είναι δυνατή η δημιουργία του προγράμματος με
τις επιλεγμένες ημερομηνίες.", "NurseRostering", MessageBoxButtons.OK,
MessageBoxIcon.Error);
            return;
        }

    }


    #endregion

    private void cbDays_SelectedIndexChanged(object sender, EventArgs e)
    {
        switch (cbDays.SelectedIndex)
        {
            case 0:
                epilegmeniImera = "Monday";
                break;
            case 1:
                epilegmeniImera = "Tuesday";
                break;
```

```
            case 2:
                epilegmeniImera = "Wednesday";
                break;
            case 3:
                epilegmeniImera = "Thursday";
                break;
            case 4:
                epilegmeniImera = "Friday";
                break;
            case 5:
                epilegmeniImera = "Saturday";
                break;
            case 6:
                epilegmeniImera = "Sunday";
                break;
            default:
                epilegmeniImera = "Monday";
                break;
        }
    }

    private void btnSubmitSchedule_Click(object sender, EventArgs e)
    {
        /* Έλεγχος για το εάν έχει πραγματοποιηθεί κάποια επιλογή
         * σχετικά με τη χρονική διάρκεια δημιουργίας του προγράμματος */
        if (rbSprint.Checked == false)
        {
            if (rbMedium.Checked == false)
            {
                if (rbLong.Checked == false)
                {
                    MessageBox.Show("Δεν έχει πραγματοποιηθεί κάποια επιλογή για τον
επιθυμητό χρόνο δημιουργίας του προγράμματος.", "NurseRostering",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                    return;
                }
            }
        }

        // Διαπίστωση αν το χρονικό διάστημα τον υπο δημιουργία προγράμματος
είναι τουλάχιστον αυριανό.
        if (module.ValidateDates(mcNewPeriodStart.SelectionRange.Start.Date,
mcNewPeriodEnd.SelectionRange.Start.Date))
        {
            imerominiaEnar3is = mcNewPeriodStart.SelectionRange.Start.Date;
            imerominiaLi3is = mcNewPeriodEnd.SelectionRange.Start.Date;

            int dateRange = (from pr in nrdb.Scheduling_Periods
                             where (pr.startDate == imerominiaEnar3is && pr.endDate ==
imerominiaLi3is)
                             select pr).Count();

            if (dateRange > 0)
```

```
            {
                DialogResult res = MessageBox.Show("Υπάρχει ήδη αυτό το χρονικό
διάστημα, καταχωρημένο στη βάση δεδομένων. Θέλετε να συνεχίσετε;",
"NurseRostering", MessageBoxButtons.YesNo, MessageBoxIcon.Question);
                if(res == DialogResult.No)
                {
                    return;
                }            }
            // Δεδομένα αντικειμένου Schedulig_Period
            newPeriod = new Scheduling_Period();
            newPeriod.kindOfProgram = eidosProgrammatos;
            newPeriod.startDate = imerominiaEnar3is;
            newPeriod.endDate = imerominiaLi3is;
        }
        else
        {
            return;
        }

        try
        {
            // Προσθήκη στη βάση δεδομένων
            nrdb.Scheduling_Periods.Add(newPeriod);

            // Αποθήκευση αλλαγών στη βάση δεδομένων
            nrdb.SaveChanges();

            btnCreateProgram.Enabled = true;

            // Ενημέρωση του χρήστη σχετικά με την επιτυψημένη καταχώρηση
            MessageBox.Show("Οι αλλαγές καταχωρήθηκαν επιτυχώς.",
"NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
        catch (Exception ex) // Σύλληψη σφάλματος
        {
            // Ενημέρωση του χρήστη σχετικά με το σφάλμα που προέκυψε
            MessageBox.Show("Προέκυψε κάποιο απροσδόκητο σφάλμα. Σφάλμα: " +
ex.ToString(), "NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Error);
        }
    }

    private void btnSubmitDayOfWeekCover_Click(object sender, EventArgs e)
    {
        string[] weekDays = { "Monday", "Tuesday", "Wednesday", "Thursday",
"Friday", "Saturday", "Sunday" };
        int c = 0;

        // Εάν υπάρχει έστω και ένα αντικείμενο
        if (ListOfDayOWC.Count > 0)
        {
            for (c = 0; c < 7; c++)
            {
                //foreach (var item in ListOfDayOWC)
```

```
                //{
                try
                {
                    DayByDay = (from d in ListOfDayOWC
                                where d.weekDay == weekDays[c]
                                select d).OrderBy(x => x.shift).ToList();
                    if(DayByDay.Count > 0)
                    {
                        foreach (var item in DayByDay)
                        {
                            // Μετακίνησέ το στη λίστα εγγραφής στη βάση δεδομένων
                            nrdb.Day_Of_Week_Covers.Add(item);
                        }
                    }
                }
                catch (Exception ex) // Σύλληψη σφάλματος
                {
                    // Ενημέρωση του χρήστη σχετικά με το σφάλμα που προέκυψε
                    MessageBox.Show("Προέκυψε κάποιο απροσδόκητο σφάλμα. Σφάλμα:
" + ex.ToString(), "NurseRostering", MessageBoxButtons.OK,
MessageBoxIcon.Error);
                }

                //}
            }

            ListOfDayOWC.Clear(); // Καθαρισμός λίστας
            try
            {
                // Αποθήκευση αλλαγών στη βάση δεδομένων
                nrdb.SaveChanges();

                // Ενημέρωση του χρήστη σχετικά με την επιτυψημένη καταχώρηση
                MessageBox.Show("Οι αλλαγές καταχωρήθηκαν επιτυχώς.",
"NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Information);
            }
            catch (Exception ex) // Σύλληψη σφάλματος
            {
                // Ενημέρωση του χρήστη σχετικά με το σφάλμα που προέκυψε
                MessageBox.Show("Προέκυψε κάποιο απροσδόκητο σφάλμα. Σφάλμα: "
+ ex.ToString(), "NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }

    }

    private void mcNewPeriodStart_DateChanged(object sender,
DateRangeEventArgs e)
    {
        // Εισαγωγή ημερομηνίας έναρξης προγράμματος
        imerominiaEnar3is = mcNewPeriodStart.SelectionRange.Start.Date;
    }
```

```
        private void mcNewPeriodEnd_DateChanged(object sender,
DateRangeEventArgs e)
        {
            // Εισαγωγή ημερομηνίας λήξης προγράμματος
            imerominiaLi3is = mcNewPeriodEnd.SelectionRange.Start.Date;
        }
    }
}
```

ΦΟΡΜΑ frmSplash

```
using NurseRostering.Database;
using System;
using System.Windows.Forms;

namespace NurseRostering.GUI
{
    public partial class frmSplash : Form
    {
        public NurseRosteringDB nrdb = new NurseRosteringDB();

        public frmSplash()
        {
            InitializeComponent();
        }

        private void frmSplash_Load(object sender, EventArgs e)
        {
            // Έλεγχος για το εάν υπάρχει η βάση δεδομένων
            if (!nrdb.Database.Exists())
            {
                // και δημιουργία αυτής αν δεν υπάρχει
                nrdb.Database.Create();

                /* Ταυτόχρονα όμως, ορίζεται και ο αριθμός εκκίνησης αυτόματης
μέτρησης
                 * αναγνωριστικού εγγραφής στο 0. Αυτή η ενέργεια είναι απαραίτητη γιατί
                 * ο αλγόριθμος δημιουργίας προγράμματος εργασίας, χρησιμοποιεί
αρίθμηση
                 * από το 0. Αν αρχίζει από οτιδήποτε άλλο, ο αλγόριθμος δεν εκτελείται.
                 * Η αρίθμηση θα τροποποιηθεί ακόμα και στους πίνακες που δεν είναι
απαραίτητη
                 * για να υπάρχει εννιαίος τρόπος εργασίας */
                nrdb.Database.ExecuteSqlCommand("DBCC
CHECKIDENT('Available_Skill', RESEED, 0)"); // Πίνακας διαθέσιμων ειδικοτήτων
                nrdb.Database.ExecuteSqlCommand("DBCC
CHECKIDENT('Available_Pattern', RESEED, 0)"); // Πίνακας στοιχείων
ανεπιθύμητων προτύπων
                nrdb.Database.ExecuteSqlCommand("DBCC CHECKIDENT('Contracts',
RESEED, 0)"); // Πίνακας συμβολαίων
```

```
            nrdb.Database.ExecuteSqlCommand("DBCC
CHECKIDENT('Date_Specific_Cover', RESEED, 0)"); // Πίνακας ειδικής κάλυψης
σε προσωπικό
            nrdb.Database.ExecuteSqlCommand("DBCC
CHECKIDENT('Day_Of_Week_Cover', RESEED, 0)"); // Πίνακας διαθέσιμων
ειδικοτήτων
            nrdb.Database.ExecuteSqlCommand("DBCC
CHECKIDENT('Day_Off_Request', RESEED, 0)"); // Πίνακας αιτούμενων ημερών
ανάπαυσης
            nrdb.Database.ExecuteSqlCommand("DBCC
CHECKIDENT('Day_On_Request', RESEED, 0)"); // Πίνακας αιτούμενων ημερών
εργασίας
            nrdb.Database.ExecuteSqlCommand("DBCC CHECKIDENT('Employees',
RESEED, 0)"); // Πίνακας υπαλλήλων
            nrdb.Database.ExecuteSqlCommand("DBCC CHECKIDENT('Patterns',
RESEED, 0)"); // Πίνακας προτύπων ανεπιθύμητης εργασίας
            nrdb.Database.ExecuteSqlCommand("DBCC
CHECKIDENT('Scheduling_Period', RESEED, 0)"); // Πίνακας ρυθμίσεων
προγράμματος εργασίας
            nrdb.Database.ExecuteSqlCommand("DBCC
CHECKIDENT('Shift_Off_Request', RESEED, 0)"); // Πίνακας αιτούμενων βαρδιών
ανάπαυσης
            nrdb.Database.ExecuteSqlCommand("DBCC
CHECKIDENT('Shift_On_Request', RESEED, 0)"); // Πίνακας αιτούμενων βαρδιών
εργασίας
            nrdb.Database.ExecuteSqlCommand("DBCC CHECKIDENT('Shift_Type',
RESEED, 0)"); // Πίνακας βαρδιών
            nrdb.Database.ExecuteSqlCommand("DBCC CHECKIDENT('Skills',
RESEED, 0)"); // Πίνακας ειδικοτήτων που απονεμήθηκαν στον υπάλληλο
        }

        timer1.Enabled = true;
    }
    private void KleisimoFormas(object sender, FormClosingEventArgs e)
    {
        this.Show(); // Εμφάνιση κύριας φόρμας, όταν κλείσει η φόρμα απαιτήσεων
προγράμματος εργασίας
    }

    private void timer1_Tick(object sender, EventArgs e)
    {
        frmMain MainForm = new frmMain();
        MainForm.Show();

        // Δημιουργία χειριστηρίου για τον έλεγχο κλεισίματος της φόρμας
        MainForm.FormClosing += KleisimoFormas;

        timer1.Enabled = false;
        this.Hide();
    }
  }
}
```

ΦΟΡΜΑ frmUnderCon

```
using NurseRostering.Database;
using System;
using System.Windows.Forms;

namespace NurseRostering.GUI
{
   public partial class frmSplash : Form
   {
      public NurseRosteringDB nrdb = new NurseRosteringDB();

      public frmSplash()
      {
         InitializeComponent();
      }

      private void frmSplash_Load(object sender, EventArgs e)
      {
         // Έλεγχος για το εάν υπάρχει η βάση δεδομένων
         if (!nrdb.Database.Exists())
         {
            // και δημιουργία αυτής αν δεν υπάρχει
            nrdb.Database.Create();

            /* Ταυτόχρονα όμως, ορίζεται και ο αριθμός εκκίνησης αυτόματης
μέτρησης
             * αναγνωριστικού εγγραφής στο 0. Αυτή η ενέργεια είναι απαραίτητη γιατί
             * ο αλγόριθμος δημιουργίας προγράμματος εργασίας, χρησιμοποιεί
αρίθμηση
             * από το 0. Αν αρχίζει από οτιδήποτε άλλο, ο αλγόριθμος δεν εκτελείται.
             * Η αρίθμηση θα τροποποιηθεί ακόμα και στους πίνακες που δεν είναι
απαραίτητη
             * για να υπάρχει εννιαίος τρόπος εργασίας */
            nrdb.Database.ExecuteSqlCommand("DBCC
CHECKIDENT('Available_Skill', RESEED, 0)"); // Πίνακας διαθέσιμων ειδικοτήτων
            nrdb.Database.ExecuteSqlCommand("DBCC
CHECKIDENT('Available_Pattern', RESEED, 0)"); // Πίνακας στοιχείων
ανεπιθύμητων προτύπων
            nrdb.Database.ExecuteSqlCommand("DBCC CHECKIDENT('Contracts',
RESEED, 0)"); // Πίνακας συμβολαίων
            nrdb.Database.ExecuteSqlCommand("DBCC
CHECKIDENT('Date_Specific_Cover', RESEED, 0)"); // Πίνακας ειδικής κάλυψης
σε προσωπικό
            nrdb.Database.ExecuteSqlCommand("DBCC
CHECKIDENT('Day_Of_Week_Cover', RESEED, 0)"); // Πίνακας διαθέσιμων
ειδικοτήτων
```

```
            nrdb.Database.ExecuteSqlCommand("DBCC
CHECKIDENT('Day_Off_Request', RESEED, 0)"); // Πίνακας αιτούμενων ημερών
ανάπαυσης
            nrdb.Database.ExecuteSqlCommand("DBCC
CHECKIDENT('Day_On_Request', RESEED, 0)"); // Πίνακας αιτούμενων ημερών
εργασίας
            nrdb.Database.ExecuteSqlCommand("DBCC CHECKIDENT('Employees',
RESEED, 0)"); // Πίνακας υπαλλήλων
            nrdb.Database.ExecuteSqlCommand("DBCC CHECKIDENT('Patterns',
RESEED, 0)"); // Πίνακας προτύπων ανεπιθύμητης εργασίας
            nrdb.Database.ExecuteSqlCommand("DBCC
CHECKIDENT('Scheduling_Period', RESEED, 0)"); // Πίνακας ρυθμίσεων
προγράμματος εργασίας
            nrdb.Database.ExecuteSqlCommand("DBCC
CHECKIDENT('Shift_Off_Request', RESEED, 0)"); // Πίνακας αιτούμενων βαρδιών
ανάπαυσης
            nrdb.Database.ExecuteSqlCommand("DBCC
CHECKIDENT('Shift_On_Request', RESEED, 0)"); // Πίνακας αιτούμενων βαρδιών
εργασίας
            nrdb.Database.ExecuteSqlCommand("DBCC CHECKIDENT('Shift_Type',
RESEED, 0)"); // Πίνακας βαρδιών
            nrdb.Database.ExecuteSqlCommand("DBCC CHECKIDENT('Skills',
RESEED, 0)"); // Πίνακας ειδικοτήτων που απονεμήθηκαν στον υπάλληλο
        }

        timer1.Enabled = true;

    }


    private void KleisimoFormas(object sender, FormClosingEventArgs e)
    {
        this.Show(); // Εμφάνιση κύριας φόρμας, όταν κλείσει η φόρμα απαιτήσεων
προγράμματος εργασίας
    }

    private void timer1_Tick(object sender, EventArgs e)
    {
        frmMain MainForm = new frmMain();
        MainForm.Show();

        // Δημιουργία χειριστηρίου για τον έλεγχο κλεισίματος της φόρμας
        MainForm.FormClosing += KleisimoFormas;

        timer1.Enabled = false;
        this.Hide();
    }
  }
}
```

ΦΟΡΜΑ frmUnwantedPattern

```
using System;
using System.Collections.Generic;
using System.Data;
using System.Linq;
using System.Windows.Forms;
using NurseRostering.Database;
using NurseRostering.InputData;

namespace NurseRostering.GUI
{
   public partial class frmUnwantedPatterns : Form
   {
      Modules module = new Modules();
      Available_Pattern AvPat;
      NurseRosteringDB nrdb = new NurseRosteringDB();
      List<Available_Pattern> ListOfAvPatterns = new List<Available_Pattern>();

      private string shift = ""; // Αναγνωριστικό επιλεγμένης βάρδιας
      private string imera; // Επιλεγμένη ημέρα για το πρότυπο ή όλες
      private string protipa  = ""; // Πρότυπα που δημιουργήθηκαν
      private int RdgAvPatterns = 0; // Επιλεγμένη γραμμή από τον χρήστη

      public frmUnwantedPatterns()
      {
         InitializeComponent();
      }

      // Κουμπί κλεισίματος της παρούσας φόρμας και εμφάνισης αυτής που την
κάλεσε.
      private void btnExit_Click(object sender, EventArgs e)
      {
         this.Close();
         GC.Collect(); // Συλλογή "σκουπιδιών"
      }

      /* Με το πάτημα του κουμπιού καταχώρησης ανεπιθύμητου προτύπου εργασίας
       * ο χρήστης θα ερωτηθεί για την βαρύτητα αυτού του προτύπου. Αυτό γίνεται για
να
       * προστατευτεί το πρότυπο από λανθασμένη χρήση, του χρήστη, εισαγωγής της
βαρύτητας
       * για το πρότυπο */
      private void btnCreate_Click(object sender, EventArgs e)
      {
         // Σημαία που υποδεικνύει την ύπαρξη διπλών εγγραφών
         bool doubleRecord = false;

         if (lstPatterns.Items.Count > 0) // Εάν υπάρχει τουλάχιστον μια εγγραφή
         {
```

```
            foreach(var item in lstPatterns.Items) // Για κάθε εγγραφή που βρέθηκε
            {
               AvPat = new Available_Pattern(); // Δημιούργησε ένα αντικείμνο γι αυτήν
               AvPat.pattern = item.ToString(); // Εισαγωγή στοιχείων
               try // Απόπειρα αποθήκευσης προτύπου
               {
                  /* Για κάθε εγγραφή που αποπειράται να εγγραφεί στη βάση δεδομένων,
πραγματοποιείται
                   * σύγκριση με τις ήδη υπάρχουσες εγγραφές, προς αποφυγή
διπλότυπων. Ο έλεγχος γίνεται
                   * μέσα από το πρόγραμμα και όχι από το μηχανισμό της βάσης
δεδομένων για να μην υπάρχει
                   * άσκοπη κίνηση στη σύνδεση με τη βάση δεδομένων. Αν βρεθεί
κάποια ίδια εγγραφή, τότε
                   * η προς έλεγχο καταχώρησης εγγραφή δεν πραγματοποιείται. */
                  for (int i = 0; i < ListOfAvPatterns.Count; i++)
                  {
                     // Εάν βρεθεί κάποια ίδια εγγραφή
                     if (item.ToString() == ListOfAvPatterns[i].pattern.ToString())
                     {
                        // Σήκωμα σχετικής σημαίας
                        doubleRecord = true;
                     }
                  }
                  if (!doubleRecord)
                  {
                     // προσθήκη του στη βάση δεδομένων
                     nrdb.AvailablePatterns.Add(AvPat);
                  }
               }
               catch(Exception ex) // Στην περίπτωση που προκύψει κάποιο σφάλμα
               {
                  // Ενημέρωση του χρήστη σχετικά με το σφάλμα που προέκυψε
                  MessageBox.Show("Προέκυψε το παρακάτω σφάλμα " + ex + ".",
"NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Error);
                  return; // Διακοπή της διαδικασίας
               }
            }

            // Εισαγωγή του προτύπου στη βάση δεδομένων
            nrdb.SaveChanges(); // Αποθήκευση προτύπου ανεπιθύμητης εργασίας

            // Ενημέρωση του χρήστη για την επιτυχή καταχώρηση των δεδομένων
            MessageBox.Show("Το πρότυπο καταχωρήθηκε επιτυχώς.",
"NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Information);
            frmUnwantedPatterns_Load(this,e);
         }
         else
         {
            // Ενημέρωση του χρήστη με σχετικό μήνυμα
            MessageBox.Show("Δεν έχει δημιουργηθεί κάποιο πρότυπο.",
"NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Error);
         }
```

```
        }

        // Προσθήκη της επιλογής του χρήστη στο υπό δημιουργία πρότυπο ανεπιθύμητης
εργασίας
        private void btnAdd_Click(object sender, EventArgs e)
        {
            // Δημιουργία προτύπου για εμφάνιση
            protipa = "(" + shift + "|" + imera + ")";

            // Έλεγχος διπλοεγγραφών στη βάση δεδομένων
            int check = nrdb.AvailablePatterns.Count(x => x.pattern == protipa);

            if (check > 0)
            {
                if (check > 0) // Εάν βρεθεί μια εγγραφή έστω
                {
                    // Ενημέρωση του χρήστη με σχετικό μήνυμα
                    MessageBox.Show("Υπάρχει ήδη μια εγγραφή στη βάση δεδομένων με
αυτό το αναγνωριστικό ή την περιγραφή.", "NurseRostering", MessageBoxButtons.OK,
MessageBoxIcon.Error);
                    return; // Τερματισμός διαδικασίας
                }
            }

            foreach(var item in lstPatterns.Items) // Έλεγχος για το εάν έχει καταχωρηθεί
ήδη το πρότυπο
            {
                // Εάν έχει ήδη καταχωρηθεί
                if(item.ToString() == protipa)
                {
                    lstPatterns.SelectedItem = item; // Ένδειξη προς τον χρήστη ότι υπάρχει
ήδη η εγγραφή
                    protipa = ""; // Καθαρισμός της σχετικής μεταβλητής.
                    return; // Τερματισμός της διαδικασίας εισαγωγής του προτύπου
                }
            }
            lstPatterns.Items.Add(protipa); // Εισαγωγή του προτύπου εφόσον δεν βρέθηκε
κάποιο άλλο όμοιο
            protipa = ""; // Καθαρισμός της σχετικής μεταβλητής
        }

        // Επιλογή βάρδιας
        private void cbShifts_SelectedIndexChanged(object sender, EventArgs e)
        {
            if (module.CheckIfAChoiceMade(cbShifts)) // Εάν έχει πραγματοποιηθεί
κάποια επιλογή
            {
                switch (cbShifts.SelectedIndex)
                {
                    case 0:
                        shift = "None";
                        break;
```

```
                case 1:
                    shift = "Any";
                    break;
                default:
                    shift = cbShifts.SelectedItem.ToString();
                    break;
            }
        }
        else
        {
            // Εμφάνιση σχετικού μηνύματος σφάλματος
            MessageBox.Show("Δεν έχει επιλεγεί κάποια βάρδια. Δεν είναι δυνατή η
συνέχεια.", "NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Error);
            this.cbShifts.Focus(); // Εστίαση στο αντικείμενο
            return;
        }

    }

    // Επιλογή ημέρας
    private void cbDay_SelectedIndexChanged(object sender, EventArgs e)
    {
        if (module.CheckIfAChoiceMade(cbShifts)) // Εάν έχει πραγματοποιηθεί
κάποια επιλογή
        {
            switch (cbDay.SelectedIndex)
            {
                case 0:
                    imera = "Any";
                    break;
                case 1:
                    imera = "Monday";
                    break;
                case 2:
                    imera = "Tuesday";
                    break;
                case 3:
                    imera = "Wednesday";
                    break;
                case 4:
                    imera = "Thursday";
                    break;
                case 5:
                    imera = "Friday";
                    break;
                case 6:
                    imera = "Saturday";
                    break;
                case 7:
                    imera = "Sunday";
                    break;
                default:
                    imera = "Any";
```

```
                break;
            }
        }
        else
        {
            // Εμφάνιση σχετικού μηνύματος σφάλματος
            MessageBox.Show("Δεν έχει επιλεγεί κάποια ημέρα. Δεν είναι δυνατή η
συνέχεια.", "NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Error);
            this.cbDay.Focus(); // Εστίαση στο αντικείμενο
            return;
        }

    }

    // Καθαρισμός λίστας
    private void btnClear_Click(object sender, EventArgs e)
    {
        lstPatterns.Items.Clear();
    }

    // Αφαίρεση κάποιας επιλογής από το πρότυπο ανεπιθύμητης εργασίας
    private void btnDelete_Click(object sender, EventArgs e)
    {
        if (lstPatterns.SelectedIndex > -1) // Εάν έχει πραγματοποιηθεί κάποια επιλογή
        {
            lstPatterns.Items.RemoveAt(lstPatterns.SelectedIndex);
        }
        else
        {
            // Εμφάνιση σχετικού μηνύματος σφάλματος
            MessageBox.Show("Δεν έχετε πραγματοποιήσει καμία επιλογή.",
"NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Error);
            this.lstPatterns.Focus(); // Εστίαση στο αντικείμενο
            return;
        }
    }

    private void frmUnwantedPatterns_Load(object sender, EventArgs e)
    {
        // Καθαρισμός του Combobox βαρδιών και του DataGridView από πιθανή
ύπαρξη τιμών
        cbShifts.Items.Clear();
        dgAvPatterns.Rows.Clear();
        dgAvPatterns.Columns.Clear();

        // Προσθήκη δύο καθολικών τιμών που πρέπει να βρίσκονται πάντα στο
ComboBox των βαρδιών
        cbShifts.Items.Add("Καμία");
        cbShifts.Items.Add("Όλες");

        // Προσθήκη στηλών στο datagridview
        dgAvPatterns.Columns.Add("", "");
        dgAvPatterns.Columns.Add("", "");
```

```
            // Άντληση δεδομένων από τη βάση και εμφάνισή τους στο σχετικό αντικείμενο
            ListOfAvPatterns = (from pt in nrdb.AvailablePatterns select
pt).OrderBy(x=>x.ID).ToList();
            foreach (var item in ListOfAvPatterns)
            {
                dgAvPatterns.Rows.Add(item.ID, item.pattern);
            }

            /*********************************
             * ΑΡΧΗ ΔΙΑΜΟΡΦΩΣΗΣ ΤΟΥ DATAGRID *
             *********************************/

            // Χρωματισμός του datagrid
            module.DataGridFormat(dgAvPatterns);

            // Καθορισμός πλάτους στηλών
            dgAvPatterns.Columns[0].Width = (int)(dgAvPatterns.Width * 0.25);
            dgAvPatterns.Columns[1].Width = dgAvPatterns.Width -
(int)(dgAvPatterns.Width * 0.25);

            // Καθορισμός τίτλου στηλών
            dgAvPatterns.Columns[0].HeaderText = "ΑΑ";
            dgAvPatterns.Columns[1].HeaderText = "Χαρακτηριστικά στοιχείων";

            /**********************************
             * ΤΕΛΟΣ ΔΙΑΜΟΡΦΩΣΗΣ ΤΟΥ DATAGRID *
             **********************************/

            // Άντληση τύπων βαρδιών από τη βάση και εμφάνισή τους στο σχετικό
αντικείμενο
            module.EmfanisiVardiwn(cbShifts);
            cbShifts.Refresh();
            dgAvPatterns.Refresh();
        }

        // Κατά την επιλογή ενός κελιού από το daragrid
        private void dgAvPatterns_CellClick(object sender, DataGridViewCellEventArgs
e)
        {
            // Δημοσίευση του αριθμού της γραμμής στην οποία βρίσκεται το κελί αυτό.
            RdgAvPatterns = e.RowIndex;
        }

        private void btnDelPattern_Click(object sender, EventArgs e)
        {
            // Ερώτηση προς στον χρήστη σχετικά με τη διαγραφή της επιλεγμένης
εγγραφής
            if (MessageBox.Show("Πρόκειται να διαγράψετε μια εγγραφή. Η διαγραφή θα
είναι μόνιμη και μη αναστρέψιμη. Θέλετε να συνεχίσετε;", "NurseRostering",
MessageBoxButtons.YesNo, MessageBoxIcon.Question) == DialogResult.Yes)
            {
                // Σε περίπτωση που ο χρήστης απαντήσει καταφατικά
```

```
            // Εύρεση εγγραφής προς διαγραφή
            var patternToDelete =
nrdb.AvailablePatterns.Find(int.Parse(dgAvPatterns.Rows[RdgAvPatterns].Cells[0].Va
lue.ToString()));
            nrdb.AvailablePatterns.Remove(patternToDelete); // Διαγραφή εγγραφής
            nrdb.SaveChanges();

            // Καθαρισμός όλων των αντικειμένων από τα περιεχόμενά τους
            ListOfAvPatterns.Clear();
            dgAvPatterns.Rows.Clear();
            dgAvPatterns.Columns.Clear();
            cbShifts.Items.Clear();

            // Επαναφόρτωση όλων των δεδομένων
            frmUnwantedPatterns_Load(this, e);
        }
        else // ενώ στην περίπτωση αρνητικής απάντησης
        {
            // Ενημέρωση του χρήστη με σχετικό μήνυμα
            MessageBox.Show("Η διαδικασία διαγραφής ακυρώθηκε από τον χρήστη.",
"NurseRostering", MessageBoxButtons.OK, MessageBoxIcon.Information);
        }
    }

    private void btnManageShifts_Click(object sender, EventArgs e)
    {
        // Δημιουργία νέας φόρμας καταχώρησης νέας βάρδιας
        frmNewShift newShift = new frmNewShift();

        newShift.Show(); // Εμφάνιση φόρμας καταχώρησης νέας βάρδιας
        newShift.btnReturn.Text = "Επιστροφή στη δημιουργία ανεπιθύμητων
πρότυπων";
        this.Hide(); // Κρύψιμο της παρούσας φόρμας

        // Δημιουργία χειριστηρίου για τον έλεγχο κλεισίματος της φόρμας
        newShift.FormClosing += KleisimoFormas;
    }

    private void KleisimoFormas(object sender, FormClosingEventArgs e)
    {
        this.Show(); // Εμφάνιση κύριας φόρμας, όταν κλείσει η φόρμα απαιτήσεων
προγράμματος εργασίας
    }
  }
}
```

Υπεύθυνη Δήλωση Συγγραφέα:

Δηλώνω ρητά ότι, σύμφωνα με το άρθρο 8 του Ν.1599/1986, η παρούσα εργασία αποτελεί αποκλειστικά προϊόν προσωπικής μου εργασίας, δεν προσβάλλει κάθε μορφής δικαιώματα διανοητικής ιδιοκτησίας, προσωπικότητας και προσωπικών δεδομένων τρίτων, δεν περιέχει έργα/εισφορές τρίτων για τα οποία απαιτείται άδεια των δημιουργών/δικαιούχων και δεν είναι προϊόν μερικής ή ολικής αντιγραφής, οι πηγές δε που χρησιμοποιήθηκαν περιορίζονται στις βιβλιογραφικές αναφορές και μόνον και πληρούν τους κανόνες της επιστημονικής παράθεσης.