



Σχολή Θετικών Επιστημών & Τεχνολογίας

Μεταπτυχιακή Εξειδίκευση Στα Πληροφοριακά Συστήματα
(ΠΛΣ)

Διπλωματική Εργασία

Σχεδίαση και υλοποίηση εφαρμογής οργάνωσης κινηματογραφικού
και τηλεοπτικού

περιεχομένου για κινητές συσκευές

Σταθόπουλος Σπυρίδων - Ιάκωβος

Επιβλέπουσα καθηγήτρια: Γκαράνη Γεωργία

Πάτρα, Ιούνιος 2026

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.



Σχεδίαση και υλοποίηση εφαρμογής οργάνωσης κινηματογραφικού και τηλεοπτικού

περιεχομένου για κινητές συσκευές

Σταθόπουλος Σπυρίδων - Ιάκωβος

Επιτροπή Επίβλεψης Πτυχιακής / Διπλωματικής Εργασίας

Επιβλέπουσα Καθηγήτρια:

Γκαράνη Γεωργία

Καθηγήτρια, Τμήμα Ψηφιακών
Συστημάτων, Πανεπιστήμιο Θεσσαλίας

Συν-Επιβλέπων Καθηγητής:

Παξιμάδης Κωνσταντίνος

Μέλος ΣΕΠ, Ελληνικό Ανοικτό
Πανεπιστήμιο

Πάτρα, Ιούνιος 2026

Θα ήθελα να εκφράσω τις θερμές μου ευχαριστίες προς την επιβλέπουσα καθηγήτριά μου, κ. Γκαράνη Γεωργία, για την πολύτιμη καθοδήγηση, την υποστήριξη και τις ουσιαστικές συμβουλές που μου προσέφερε καθ' όλη τη διάρκεια της εκπόνησης της διπλωματικής μου εργασίας. Η συμβολή της υπήρξε καθοριστική για την ολοκλήρωση της παρούσας εργασίας, ενώ η εμπιστοσύνη και η ενθάρρυνσή της αποτέλεσαν σημαντικό κίνητρο για εμένα.

Θα ήθελα επίσης να ευχαριστήσω θερμά την οικογένειά μου και τους αγαπημένους μου ανθρώπους για τη συνεχή στήριξη, την υπομονή και την κατανόηση που μου έδειξαν σε όλη αυτή τη διαδρομή. Η παρουσία και η ενθάρρυνσή τους μου έδωσαν δύναμη και συνέβαλαν ουσιαστικά στην επίτευξη αυτού του στόχου.

Περίληψη

Η παρούσα διπλωματική εργασία αφορά τον σχεδιασμό και την υλοποίηση μιας εφαρμογής για κινητές συσκευές iOS με σκοπό την αναζήτηση, οργάνωση και διαχείριση κινηματογραφικού και τηλεοπτικού περιεχομένου. Η εφαρμογή αναπτύχθηκε με στόχο να προσφέρει στους χρήστες ένα ενιαίο περιβάλλον μέσα από το οποίο μπορούν να αναζητούν ταινίες και τηλεοπτικές σειρές, να εφαρμόζουν φίλτρα αναζήτησης, να προβάλλουν αναλυτικές πληροφορίες περιεχομένου και να διαχειρίζονται προσωπικές λίστες αγαπημένων.

Η ανάγκη για την ανάπτυξη μιας τέτοιας εφαρμογής προκύπτει από τη συνεχή αύξηση του διαθέσιμου ψηφιακού περιεχομένου και τη δυσκολία των χρηστών να εντοπίσουν περιεχόμενο που ανταποκρίνεται στις προσωπικές τους προτιμήσεις. Παρά την ύπαρξη πολλών εφαρμογών οργάνωσης κινηματογραφικού και τηλεοπτικού περιεχομένου, συχνά παρατηρούνται περιορισμοί ως προς την προσωποποίηση, τη λειτουργικότητα και την εμπειρία χρήστη.

Για την υλοποίηση της εφαρμογής πραγματοποιήθηκε μελέτη σχετικών τεχνολογιών και υπάρχουσών λύσεων, καθώς και ανάλυση των απαιτήσεων των χρηστών. Η ανάπτυξη βασίστηκε στις τεχνολογίες Swift και SwiftUI για τη δημιουργία του περιβάλλοντος χρήστη, ενώ για τη διαχείριση χρηστών και την αποθήκευση δεδομένων αξιοποιήθηκε η πλατφόρμα Firebase. Επιπλέον, ενσωματώθηκε μηχανισμός προσωποποιημένων προτάσεων μέσω τεχνολογιών μηχανικής μάθησης, χρησιμοποιώντας μοντέλο εκπαιδευμένο με TensorFlow και ενσωματωμένο στην εφαρμογή μέσω του CoreML.

Η εφαρμογή περιλαμβάνει λειτουργίες σύνδεσης και ταυτοποίησης χρηστών, αναζήτησης και φιλτραρίσματος περιεχομένου, προβολής αναλυτικών πληροφοριών για ταινίες και σειρές, καθώς και διαχείρισης λιστών αγαπημένων και παρακολούθησης. Ιδιαίτερη έμφαση δόθηκε στον σχεδιασμό μιας σύγχρονης και φιλικής διεπαφής χρήστη, σύμφωνα με τις αρχές σχεδίασης UI/UX.

Τα αποτελέσματα της εργασίας δείχνουν ότι η εφαρμογή ανταποκρίνεται αποτελεσματικά στις βασικές ανάγκες οργάνωσης και ανακάλυψης κινηματογραφικού και τηλεοπτικού περιεχομένου, προσφέροντας παράλληλα μια εξατομικευμένη εμπειρία χρήσης μέσω της αξιοποίησης τεχνικών μηχανικής μάθησης.

Λέξεις – Κλειδιά

Εφαρμογές iOS, Κινηματογραφικό και Τηλεοπτικό Περιεχόμενο, Διαχείριση Ψηφιακού Περιεχομένου, Συστήματα Συστάσεων, Μηχανική Μάθηση, Προσωποποιημένες Προτάσεις, Εμπειρία Χρήστη

Design and Implementation of a Mobile Application for Organizing Movie and Television Content

Stathopoulos Spyridon Iakovos

Abstract

This thesis focuses on the design and development of an iOS mobile application for searching, organizing, and managing movie and television content. The application was developed with the aim of providing users with a unified environment through which they can search for movies and TV series, apply filtering criteria, view detailed content information, and manage personal favorite lists.

The need for such an application arises from the continuously increasing amount of digital entertainment content and the difficulty users face when trying to discover content that matches their personal preferences. Although several content management applications are currently available, many of them present limitations regarding personalization, functionality, and overall user experience.

To support the development process, a study of existing technologies and related applications was conducted, along with an analysis of user requirements. The application was implemented using Swift and SwiftUI for the user interface, while Firebase was utilized for user authentication and data management. Furthermore, a personalized recommendation mechanism was integrated through machine learning technologies, using a TensorFlow-trained model deployed within the application through CoreML.

The application includes user authentication, content search and filtering, detailed movie and TV series information display, as well as favorite and watchlist management functionalities. Special emphasis was placed on creating a modern and user-friendly interface following established UI/UX design principles.

The results demonstrate that the developed application effectively addresses the core needs of movie and television content organization and discovery, while also providing a personalized user experience through the integration of machine learning techniques.

Keywords

iOS Applications, Movie and TV Content, Digital Content Management, Recommendation Systems, Machine Learning, Personalized Recommendations, User Experience

Περιεχόμενα

Περίληψη.....	2
Abstract.....	4
Κατάλογος Εικόνων / Σχημάτων.....	8
Κατάλογος Πινάκων.....	9
Συντομογραφίες & Ακρωνύμια.....	10
1. Εισαγωγή.....	11
1.1 Περιγραφή του προβλήματος.....	11
1.2 Σκοπός και στόχοι της εργασίας.....	12
1.3 Συνεισφορά της εργασίας.....	12
1.4 Δομή της εργασίας.....	12
2. Θεωρητικό Υπόβαθρο και Σχετικές Εφαρμογές.....	14
2.1 Ψηφιακό κινηματογραφικό και τηλεοπτικό περιεχόμενο.....	14
2.1.1 Η εξέλιξη των πλατφορμών συνεχούς ροής (streaming).....	14
2.1.2 Προβλήματα διαχείρισης περιεχομένου.....	14
2.2 Εφαρμογές οργάνωσης περιεχομένου.....	15
2.2.1 Παρουσίαση υπαρχουσών εφαρμογών.....	15
2.2.2 Συγκριτική ανάλυση εφαρμογών.....	17
2.2.3 Περιορισμοί υπαρχουσών λύσεων.....	18
2.3 Τεχνολογίες ανάπτυξης κινητών εφαρμογών.....	18
2.3.1 Εγγενείς εφαρμογές iOS (Native iOS applications).....	19
2.3.2 Σχεδιασμός διεπαφής και εμπειρίας χρήστη (UI/UX).....	19
2.4 Μηχανική μάθηση σε συστήματα προτάσεων.....	20
2.4.1 Εισαγωγή στη μηχανική μάθηση.....	20
2.4.2 Συστήματα συστάσεων.....	21
2.4.3 Εφαρμογές μηχανικής μάθησης σε εφαρμογές κινητών συσκευών.....	21
3. Μεθοδολογία.....	22
4. Τεχνολογίες και Εργαλεία Ανάπτυξης.....	24
4.1 Περιβάλλον ανάπτυξης.....	24
4.1.1 Xcode.....	25
4.1.2 Git και διαχείριση εκδόσεων.....	25
4.2 Γλώσσες και πλαίσια ανάπτυξης.....	26
4.2.1 Swift.....	26
4.2.2 SwiftUI.....	26
4.3 Βάσεις δεδομένων και υπηρεσίες υποστήριξης.....	27
4.3.1 Firebase.....	27
4.3.2 Διαχείριση δεδομένων.....	28
4.4 Τεχνολογίες μηχανικής μάθησης.....	28
4.4.1 CoreML.....	29
4.4.2 TensorFlow.....	29
4.4.3 Εκπαίδευση και ενσωμάτωση μοντέλων.....	30
5. Ανάλυση και Σχεδιασμός Συστήματος.....	31
5.1 Απαιτήσεις συστήματος.....	31
5.1.1 Λειτουργικές απαιτήσεις.....	31
5.1.2 Μη λειτουργικές απαιτήσεις.....	32
5.2 Περιπτώσεις Χρήσης.....	33
5.2.1 Διαχείριση Λογαριασμού Χρήστη.....	33
5.2.2 Αναζήτηση Περιεχομένου.....	34
5.2.3 Φιλτράρισμα Περιεχομένου.....	34
5.2.4 Προβολή Πληροφοριών Περιεχομένου.....	34
5.2.5 Διαχείριση Αγαπημένων και Watchlist.....	34
5.3 Βασικές κλάσεις.....	35
5.4 Διαγράμματα συστήματος.....	48
5.4.1 Διάγραμμα Περιπτώσεων Χρήσης.....	48
5.4.2 Διάγραμμα κλάσεων.....	49

5.5 Σχεδιασμός διεπαφής χρήστη.....	51
5.5.1 Αρχές σχεδιασμού.....	51
5.5.2 Προσχέδια διεπαφής (Wireframes).....	52
6. Υλοποίηση της Εφαρμογής.....	53
6.1 Αρχιτεκτονική εφαρμογής.....	53
6.1.1 Πρότυπο αρχιτεκτονικής MVVM.....	53
6.1.2 Δομή έργου.....	54
6.2 Υλοποίηση βασικών λειτουργιών.....	58
6.2.1 Αναζήτηση και προβολή περιεχομένου.....	58
6.2.2 Φιλτράρισμα δεδομένων.....	58
6.2.3 Διαχείριση αγαπημένων.....	58
6.3 Υλοποίηση μηχανικής μάθησης.....	59
6.3.1 Συλλογή δεδομένων χρήστη.....	59
6.3.2 Εκπαίδευση μοντέλου.....	59
6.3.3 Προσωποποιημένες προτάσεις.....	59
6.4 Διεπαφή χρήστη.....	60
6.4.1 Κύριες οθόνες εφαρμογής.....	60
6.4.2 Πλοήγηση και ροή εφαρμογής.....	60
7. Συμπεράσματα και Μελλοντικές Επεκτάσεις.....	61
7.1 Συμπεράσματα.....	61
7.2 Περιορισμοί της εφαρμογής.....	61
7.3 Μελλοντικές βελτιώσεις.....	62
Βιβλιογραφία.....	63
Βιβλία.....	63
Επιστημονικά Άρθρα.....	63
Πηγές από Ιστοσελίδες.....	64
Παράρτημα Α: Πηγαίος Κώδικας CineMax.....	65

Κατάλογος Εικόνων / Σχημάτων

Εικόνα 1 Κλάση MovieDataManager.....	36
Εικόνα 2 Κλάση AuthManager.....	37
Εικόνα 3 Κλάση ContentViewModel.....	38
Εικόνα 4 Κλάση FavoritesViewModel.....	39
Εικόνα 5 Κλάση CineMaxRecommenderInput.....	40
Εικόνα 6 Κλάση CineMaxRecommenderOutput.....	41
Εικόνα 7 Κλάση CineMaxRecommender.....	42
Εικόνα 8 Κλάση HomeViewModel.....	43
Εικόνα 9 Κλάση MovieDetailViewModel.....	44
Εικόνα 10 Κλάση ProfileViewModel.....	45
Εικόνα 11 Κλάση SearchViewModel.....	46
Εικόνα 12 Κλάση SettingsViewModel.....	47
Εικόνα 13 Κλάση WatchlistViewModel.....	48
Εικόνα 14 Διάγραμμα περιπτώσεων χρήσης του CineMax.....	49
Εικόνα 15 Διάγραμμα κλάσεων του CineMax.....	50
Εικόνα 16 Αρχική οθόνη του CineMax.....	55
Εικόνα 17 Οθόνη αναζήτησης του CineMax.....	55
Εικόνα 18 Οθόνη φιλτραρίσματος αποτελεσμάτων του CineMax.....	55
Εικόνα 19 Οθόνη αναλυτικών πληροφοριών του CineMax.....	56
Εικόνα 20 Οθόνη διαχείρισης αγαπημένων του CineMax.....	56
Εικόνα 21 Οθόνη μελλοντικών παρακολουθήσεων του CineMax.....	57
Εικόνα 22 Οθόνη διαχείρισης προφίλ του CineMax.....	57

Κατάλογος Πινάκων

Πίνακας 1 Λειτουργικές απαιτήσεις.....	32
Πίνακας 2 Μη Λειτουργικές απαιτήσεις.....	33

Συντομογραφίες & Ακρωνύμια

API	Application Programming Interface
AI	Artificial Intelligence
ML	Machine Learning
UI	User Interface
UX	User Experience
UML	Unified Modeling Language
IDE	Integrated Development Environment
iOS	iPhone Operating System
MVVM	Model-View-ViewModel
TMDB	The Movie Database
Firebase	Google Firebase Platform
CoreML	Core Machine Learning
SUS	System Usability Scale
IMDb	Internet Movie Database
JSON	JavaScript Object Notation

1. Εισαγωγή

Η ανάπτυξη των εφαρμογών κινητών συσκευών έχει μεταβάλει σημαντικά τον τρόπο με τον οποίο οι χρήστες αλληλεπιδρούν με τις ψηφιακές υπηρεσίες. Οι σύγχρονες εφαρμογές δεν περιορίζονται πλέον στην απλή προβολή πληροφοριών, αλλά ενσωματώνουν προηγμένες λειτουργίες, όπως εξατομικευμένες προτάσεις, υπηρεσίες υπολογιστικού νέφους (cloud services), μηχανική μάθηση και φιλικές προς τον χρήστη διεπαφές. Οι τεχνολογίες αυτές συμβάλλουν στη δημιουργία εφαρμογών που προσαρμόζονται στις ανάγκες κάθε χρήστη, προσφέροντας μια πιο ολοκληρωμένη και προσωποποιημένη εμπειρία χρήσης.

Η παρούσα διπλωματική εργασία επικεντρώνεται στον σχεδιασμό και την ανάπτυξη μιας εφαρμογής για κινητές συσκευές iOS, η οποία έχει ως αντικείμενο την αναζήτηση, την προβολή και τη διαχείριση πληροφοριών σχετικά με ταινίες και τηλεοπτικές σειρές. Η εφαρμογή αναπτύχθηκε με χρήση της γλώσσας προγραμματισμού Swift και του πλαισίου ανάπτυξης SwiftUI, ενώ αξιοποιεί το Firebase για τη διαχείριση της αυθεντικοποίησης των χρηστών.

Ιδιαίτερο χαρακτηριστικό της εφαρμογής αποτελεί η ενσωμάτωση ενός συστήματος προτάσεων (Recommendation System), το οποίο βασίζεται σε μοντέλο μηχανικής μάθησης που εκπαιδεύτηκε με το TensorFlow και ενσωματώθηκε στην εφαρμογή μέσω του CoreML. Με τον τρόπο αυτό παρέχονται προσωποποιημένες προτάσεις περιεχομένου, λαμβάνοντας υπόψη τα χαρακτηριστικά και τις προτιμήσεις του χρήστη.

Στόχος της εργασίας είναι η παρουσίαση όλων των σταδίων ανάπτυξης της εφαρμογής, από την ανάλυση των απαιτήσεων και τον σχεδιασμό της αρχιτεκτονικής, έως την υλοποίηση, την αξιολόγηση και την παρουσίαση των αποτελεσμάτων. Παράλληλα, εξετάζονται οι τεχνολογίες που χρησιμοποιήθηκαν, καθώς και ο τρόπος με τον οποίο αυτές συνεργάζονται για την ανάπτυξη μιας σύγχρονης εφαρμογής κινητών συσκευών με δυνατότητες εξατομίκευσης και εύχρηστης διαχείρισης περιεχομένου.

1.1 Περιγραφή του προβλήματος

Στη σύγχρονη εποχή, η ραγδαία ανάπτυξη των ψηφιακών πλατφορμών ψυχαγωγίας έχει οδηγήσει σε μια τεράστια αύξηση του διαθέσιμου κινηματογραφικού και τηλεοπτικού περιεχομένου. Οι χρήστες έχουν πλέον πρόσβαση σε χιλιάδες ταινίες και σειρές, γεγονός που, αν και προσφέρει ποικιλία επιλογών, ταυτόχρονα καθιστά δύσκολη την εύρεση περιεχομένου που να ανταποκρίνεται στις προσωπικές τους προτιμήσεις.

Ένα βασικό πρόβλημα είναι η αποσπασματικότητα της πληροφορίας, καθώς τα δεδομένα σχετικά με ταινίες και σειρές είναι διασκορπισμένα σε διαφορετικές πλατφόρμες και υπηρεσίες. Επιπλέον, πολλές υπάρχουσες εφαρμογές είτε εστιάζουν αποκλειστικά στην προβολή περιεχομένου χωρίς δυνατότητες οργάνωσης, είτε δεν παρέχουν επαρκή εργαλεία εξατομίκευσης.

Ως αποτέλεσμα, οι χρήστες συχνά αφιερώνουν σημαντικό χρόνο στην αναζήτηση περιεχομένου, χωρίς να καταλήγουν εύκολα σε επιλογές που να τους ικανοποιούν. Η ανάγκη για μια ενιαία, εύχρηστη και αποδοτική εφαρμογή που θα συνδυάζει την αναζήτηση, το φιλτράρισμα, την οργάνωση και την προσωποποιημένη πρόταση περιεχομένου καθίσταται ιδιαίτερα σημαντική.

1.2 Σκοπός και στόχοι της εργασίας

Σκοπός της παρούσας εργασίας είναι ο σχεδιασμός και η υλοποίηση μιας εφαρμογής για κινητές συσκευές iOS, η οποία επιτρέπει στους χρήστες να αναζητούν, να φιλτράρουν και να οργανώνουν κινηματογραφικό και τηλεοπτικό περιεχόμενο μέσα από ένα σύγχρονο και φιλικό περιβάλλον χρήστη.

Οι επιμέρους στόχοι της εργασίας περιλαμβάνουν:

- Την ανάπτυξη ενός μηχανισμού αναζήτησης περιεχομένου με δυνατότητες φιλτραρίσματος βάσει χαρακτηριστικών, όπως είδος και έτος κυκλοφορίας.
- Την παροχή αναλυτικών πληροφοριών για κάθε ταινία ή σειρά, όπως περίληψη και βασικά μεταδεδομένα.
- Τη δυνατότητα δημιουργίας και διαχείρισης προσωπικής λίστας αγαπημένων από τον χρήστη.
- Την ενσωμάτωση μηχανισμών προσωποποιημένων προτάσεων, με στόχο τη βελτίωση της εμπειρίας χρήσης.
- Τον σχεδιασμό μιας αποδοτικής και αισθητικά ευχάριστης διεπαφής χρήστη, βασισμένης σε αρχές σύγχρονου UI/UX.

Η υλοποίηση των παραπάνω στοχεύει στη δημιουργία μιας ολοκληρωμένης εφαρμογής που θα καλύπτει ουσιαστικές ανάγκες των χρηστών στον τομέα της ψηφιακής ψυχαγωγίας.

1.3 Συνεισφορά της εργασίας

Η παρούσα εργασία συμβάλλει στον τομέα των εφαρμογών διαχείρισης ψηφιακού περιεχομένου, προτείνοντας μια ολοκληρωμένη λύση που εστιάζει τόσο στη λειτουργικότητα όσο και στην εμπειρία χρήστη. Σε αντίθεση με πολλές υπάρχουσες εφαρμογές, η προτεινόμενη προσέγγιση συνδυάζει σε ένα ενιαίο περιβάλλον τη δυνατότητα αναζήτησης, φιλτραρίσματος και οργάνωσης περιεχομένου.

Ιδιαίτερη έμφαση δίνεται στην αξιοποίηση τεχνικών μηχανικής μάθησης για την παραγωγή προσωποποιημένων προτάσεων, με βάση χαρακτηριστικά και προτιμήσεις του χρήστη. Η ενσωμάτωση αυτών των τεχνικών ενισχύει τη χρηστικότητα της εφαρμογής και συμβάλλει στη μείωση του χρόνου που απαιτείται για την επιλογή περιεχομένου.

Επιπλέον, η εργασία παρουσιάζει μια σύγχρονη υλοποίηση εφαρμογής iOS με χρήση τεχνολογιών όπως Swift και SwiftUI, αναδεικνύοντας πρακτικές σχεδιασμού και ανάπτυξης που μπορούν να αξιοποιηθούν σε παρόμοια έργα. Με τον τρόπο αυτό, η εργασία δεν περιορίζεται μόνο σε θεωρητικό επίπεδο, αλλά προσφέρει και πρακτική αξία.

1.4 Δομή της εργασίας

Η παρούσα διπλωματική εργασία αποτελείται από επτά κύρια κεφάλαια, τα οποία παρουσιάζουν σταδιακά το θεωρητικό υπόβαθρο, τη μεθοδολογία, τον σχεδιασμό, την ανάπτυξη και την αξιολόγηση της εφαρμογής CineMax.

Στο πρώτο κεφάλαιο παρουσιάζεται το αντικείμενο της εργασίας, αναλύεται το πρόβλημα που επιχειρείται να αντιμετωπιστεί, διατυπώνονται ο σκοπός και οι επιμέρους στόχοι της μελέτης, ενώ περιγράφεται συνοπτικά η συνεισφορά της εργασίας.

Στο δεύτερο κεφάλαιο παρουσιάζεται το θεωρητικό υπόβαθρο που σχετίζεται με το αντικείμενο της εργασίας. Αναλύονται οι σύγχρονες πλατφόρμες ψηφιακού κινηματογραφικού και τηλεοπτικού περιεχομένου, εξετάζονται υπάρχουσες εφαρμογές διαχείρισης και οργάνωσης περιεχομένου, παρουσιάζονται βασικές τεχνολογίες ανάπτυξης εφαρμογών κινητών συσκευών και γίνεται εισαγωγή στις τεχνικές μηχανικής μάθησης και στα συστήματα προτάσεων.

Στο τρίτο κεφάλαιο περιγράφεται η μεθοδολογία που ακολουθήθηκε για την υλοποίηση της εφαρμογής. Παρουσιάζονται διαδοχικά τα στάδια της ανάπτυξης, από τη βιβλιογραφική έρευνα και την ανάλυση των απαιτήσεων έως τον σχεδιασμό της αρχιτεκτονικής, την επιλογή των τεχνολογιών, την υλοποίηση, τις δοκιμές και την αξιολόγηση της εφαρμογής. Στο τέταρτο κεφάλαιο παρουσιάζονται οι τεχνολογίες και τα εργαλεία ανάπτυξης που χρησιμοποιήθηκαν κατά την υλοποίηση της εφαρμογής. Αναλύονται το περιβάλλον ανάπτυξης, οι γλώσσες και τα πλαίσια ανάπτυξης, οι υπηρεσίες διαχείρισης δεδομένων, καθώς και οι τεχνολογίες μηχανικής μάθησης που αξιοποιήθηκαν για την ανάπτυξη του συστήματος προσωποποιημένων προτάσεων.

Στο πέμπτο κεφάλαιο παρουσιάζεται η ανάλυση και ο σχεδιασμός του συστήματος. Περιγράφονται οι λειτουργικές και μη λειτουργικές απαιτήσεις της εφαρμογής, οι βασικές περιπτώσεις χρήσης, οι κύριες κλάσεις του συστήματος, τα διαγράμματα UML που αποτυπώνουν τη δομή και τη λειτουργία της εφαρμογής, καθώς και ο σχεδιασμός της διεπαφής χρήστη μέσω βασικών αρχών σχεδίασης και προσχεδίων (wireframes).

Στο έκτο κεφάλαιο παρουσιάζεται η υλοποίηση της εφαρμογής CineMax. Περιγράφεται η αρχιτεκτονική MVVM που ακολουθήθηκε, η δομή του έργου, η ανάπτυξη των βασικών λειτουργιών της εφαρμογής, η ενσωμάτωση του μηχανισμού μηχανικής μάθησης για την παραγωγή προσωποποιημένων προτάσεων, καθώς και η υλοποίηση της διεπαφής χρήστη και της πλοήγησης μεταξύ των επιμέρους οθονών.

Τέλος, στο έβδομο κεφάλαιο παρουσιάζονται τα συμπεράσματα που προέκυψαν από την ανάπτυξη και την αξιολόγηση της εφαρμογής, οι περιορισμοί της υλοποίησης και προτείνονται πιθανές μελλοντικές επεκτάσεις και βελτιώσεις που μπορούν να ενισχύσουν περαιτέρω τη λειτουργικότητα και την εμπειρία χρήσης της εφαρμογής.

2. Θεωρητικό Υπόβαθρο και Σχετικές Εφαρμογές

2.1 Ψηφιακό κινηματογραφικό και τηλεοπτικό περιεχόμενο

Η εξέλιξη της τεχνολογίας και η ευρεία διάδοση του διαδικτύου έχουν μετασχηματίσει ριζικά τον τρόπο με τον οποίο οι χρήστες καταναλώνουν κινηματογραφικό και τηλεοπτικό περιεχόμενο. Από τα παραδοσιακά μέσα προβολής, όπως η τηλεόραση και οι κινηματογραφικές αίθουσες, η πρόσβαση έχει μετατοπιστεί σε ψηφιακές πλατφόρμες που επιτρέπουν την άμεση και κατά παραγγελία θέαση περιεχομένου.

Το σύγχρονο ψηφιακό περιεχόμενο χαρακτηρίζεται από μεγάλη ποικιλία ως προς τα είδη, τη γλώσσα, τη χώρα παραγωγής και τη θεματολογία. Παράλληλα, η συνεχής παραγωγή νέου υλικού οδηγεί σε μια διαρκώς αυξανόμενη βάση δεδομένων, η οποία καθιστά αναγκαία την ύπαρξη εργαλείων που διευκολύνουν την αναζήτηση και την οργάνωσή του.

Η ανάγκη για αποτελεσματική διαχείριση αυτού του όγκου πληροφορίας αποτελεί βασικό παράγοντα για την ανάπτυξη εφαρμογών που στοχεύουν στη βελτίωση της εμπειρίας του χρήστη.

2.1.1 Η εξέλιξη των πλατφορμών συνεχούς ροής (streaming)

Οι πλατφόρμες συνεχούς ροής έχουν διαδραματίσει καθοριστικό ρόλο στη διάδοση του ψηφιακού περιεχομένου. Μέσα σε λίγα χρόνια, υπηρεσίες αυτού του τύπου εξελίχθηκαν από απλές βιβλιοθήκες πολυμέσων σε ολοκληρωμένα οικοσυστήματα ψυχαγωγίας.

Οι χρήστες πλέον έχουν τη δυνατότητα να παρακολουθούν περιεχόμενο οποιαδήποτε στιγμή και από οποιαδήποτε συσκευή, χωρίς την ανάγκη λήψης αρχείων. Επιπλέον, πολλές πλατφόρμες επενδύουν στην παραγωγή πρωτότυπου περιεχομένου, ενισχύοντας ακόμη περισσότερο την ποικιλία των διαθέσιμων επιλογών.

Παρά τα πλεονεκτήματα αυτά, η ύπαρξη πολλαπλών πλατφορμών οδηγεί σε κατακερματισμό του περιεχομένου, καθώς κάθε υπηρεσία διαθέτει διαφορετικό κατάλογο. Αυτό δημιουργεί δυσκολίες στον χρήστη, ο οποίος καλείται να αναζητήσει πληροφορίες σε διαφορετικά περιβάλλοντα.

2.1.2 Προβλήματα διαχείρισης περιεχομένου

Η υπερπληθώρα διαθέσιμου περιεχομένου δημιουργεί σημαντικές προκλήσεις ως προς τη διαχείριση και την επιλογή του. Ένα από τα βασικότερα προβλήματα είναι η δυσκολία εντοπισμού περιεχομένου που ανταποκρίνεται στις προσωπικές προτιμήσεις του χρήστη.

Επιπλέον, η απουσία ενοποιημένων εργαλείων οργάνωσης οδηγεί σε κατακερματισμένη εμπειρία χρήσης. Οι χρήστες συχνά αναγκάζονται να χρησιμοποιούν πολλαπλές εφαρμογές ή να βασίζονται σε μη αποδοτικές μεθόδους, όπως η αποθήκευση πληροφοριών εκτός εφαρμογών.

Άλλο ένα σημαντικό ζήτημα είναι η περιορισμένη δυνατότητα προσωποποίησης σε ορισμένες υπηρεσίες, γεγονός που μειώνει την αποτελεσματικότητα των προτεινόμενων επιλογών. Η αντιμετώπιση των παραπάνω προβλημάτων αποτελεί βασική πρόκληση για τις σύγχρονες εφαρμογές διαχείρισης περιεχομένου.

2.2 Εφαρμογές οργάνωσης περιεχομένου

2.2.1 Παρουσίαση υπάρχουσών εφαρμογών

Στην αγορά υπάρχουν ήδη αρκετές εφαρμογές που στοχεύουν στην οργάνωση και αναζήτηση κινηματογραφικού και τηλεοπτικού περιεχομένου. Οι εφαρμογές αυτές παρέχουν βασικές λειτουργίες, όπως προβολή πληροφοριών, αναζήτηση τίτλων και δημιουργία λιστών.

Ορισμένες από αυτές δίνουν έμφαση στη διαχείριση προσωπικών συλλογών, ενώ άλλες επικεντρώνονται περισσότερο στην παροχή προτάσεων και στην κοινωνική διάσταση, επιτρέποντας στους χρήστες να αξιολογούν και να μοιράζονται περιεχόμενο.

Παρά την ποικιλία επιλογών, δεν προσφέρουν όλες το ίδιο επίπεδο ευχρηστίας και λειτουργικότητας, γεγονός που αφήνει περιθώρια βελτίωσης. Ακολουθούν κάποιες από τις υπάρχουσες λύσεις.

Letterboxd

Η εφαρμογή Letterboxd αποτελεί μία από τις πιο διαδεδομένες πλατφόρμες καταγραφής κινηματογραφικού περιεχομένου. Εστιάζει κυρίως στις ταινίες και προσφέρει δυνατότητες όπως καταγραφή προβολών, αξιολόγηση και δημιουργία προσωπικών λιστών.

Ένα από τα βασικά χαρακτηριστικά της εφαρμογής είναι η έντονη κοινωνική διάσταση, καθώς οι χρήστες μπορούν να δημοσιεύουν κριτικές, να ακολουθούν άλλους χρήστες και να ανακαλύπτουν περιεχόμενο μέσα από την κοινότητα.

Ωστόσο, η εφαρμογή παρουσιάζει περιορισμούς ως προς την υποστήριξη τηλεοπτικών σειρών και προηγμένων φίλτρων αναζήτησης.

Letterboxd. (2025). Official Website.

<https://letterboxd.com>

TV Time

Η εφαρμογή TV Time επικεντρώνεται κυρίως στην παρακολούθηση τηλεοπτικών σειρών, αλλά υποστηρίζει και ταινίες. Επιτρέπει στους χρήστες να καταγράφουν τα επεισόδια που έχουν παρακολουθήσει και να διαχειρίζονται την πρόοδο θέασης.

Ένα από τα βασικά πλεονεκτήματα της εφαρμογής είναι η δυνατότητα παροχής προτάσεων βασισμένων στο ιστορικό θέασης του χρήστη, ενισχύοντας την εμπειρία προσωποποίησης.

Παράλληλα, παρέχει λειτουργίες αλληλεπίδρασης, όπως αξιολογήσεις και σχολιασμό περιεχομένου.

TV Time. (2025). Official Website.

<https://www.tvtime.com>

Trakt

Η εφαρμογή Trakt αποτελεί μια ολοκληρωμένη πλατφόρμα παρακολούθησης ταινιών και σειρών, με έντονο στοιχείο κοινότητας. Δίνει τη δυνατότητα καταγραφής του περιεχομένου που έχει παρακολουθήσει ο χρήστης, δημιουργίας λιστών και παρακολούθησης της πρόόδου.

Ένα ιδιαίτερο χαρακτηριστικό της είναι η δυνατότητα συγχρονισμού με άλλες υπηρεσίες και πλατφόρμες, καθώς και η παροχή προσωποποιημένων προτάσεων με βάση τη δραστηριότητα του χρήστη.

Επιπλέον, περιλαμβάνει εργαλεία ανακάλυψης περιεχομένου και ενημέρωσης για νέες κυκλοφορίες.

Trakt. (2025). Official Website.

<https://trakt.tv>

CineTrak

Η εφαρμογή CineTrak προσφέρει ένα ολοκληρωμένο περιβάλλον για την αναζήτηση και οργάνωση κινηματογραφικού και τηλεοπτικού περιεχομένου. Υποστηρίζει τη δημιουργία λιστών, την παρακολούθηση ιστορικού και την εμφάνιση πληροφοριών από πολλαπλές πηγές, όπως αξιολογήσεις και κριτικές.

Επιπλέον, παρέχει προηγμένες δυνατότητες φιλτραρίσματος και οργάνωσης περιεχομένου, καθώς και εργαλεία υπενθύμισης για νέες κυκλοφορίες και επεισόδια.

Η εφαρμογή δίνει έμφαση στη λειτουργικότητα και στη διαχείριση μεγάλου όγκου περιεχομένου.

CineTrak. (2025). Official Website.

<https://cinetrak.app>

2.2.2 Συγκριτική ανάλυση εφαρμογών

Η σύγκριση των υπάρχουσών εφαρμογών αναδεικνύει σημαντικές διαφορές ως προς τις δυνατότητες που παρέχουν. Ορισμένες εφαρμογές διαθέτουν προηγμένα φίλτρα αναζήτησης, ενώ άλλες περιορίζονται σε βασικές λειτουργίες.

Επιπλέον, παρατηρούνται διαφοροποιήσεις στον τρόπο παρουσίασης των δεδομένων, στη σχεδίαση της διεπαφής χρήστη και στον βαθμό προσωποποίησης. Σε πολλές περιπτώσεις, η εμπειρία χρήσης επηρεάζεται αρνητικά από πολύπλοκες διεπαφές ή από έλλειψη σαφούς οργάνωσης της πληροφορίας.

Αρχικά, η εφαρμογή Letterboxd εστιάζει κυρίως στην καταγραφή και αξιολόγηση ταινιών, δίνοντας ιδιαίτερη έμφαση στην κοινωνική αλληλεπίδραση μεταξύ χρηστών. Αν και προσφέρει ένα πλούσιο περιβάλλον για ανταλλαγή απόψεων και ανακάλυψη περιεχομένου, υστερεί ως προς την υποστήριξη τηλεοπτικών σειρών και την παροχή εξελιγμένων μηχανισμών φιλτραρίσματος ή προσωποποίησης.

Αντίστοιχα, η εφαρμογή TV Time επικεντρώνεται κυρίως στις τηλεοπτικές σειρές, παρέχοντας δυνατότητες παρακολούθησης επεισοδίων και διαχείρισης προόδου θέασης. Παρά το γεγονός ότι ενσωματώνει βασικές λειτουργίες προτάσεων, η προσέγγισή της στην οργάνωση περιεχομένου είναι περισσότερο περιορισμένη και λιγότερο ευέλικτη σε σχέση με πιο ολοκληρωμένες λύσεις.

Η εφαρμογή Trakt παρουσιάζει μια πιο σύνθετη και ολοκληρωμένη προσέγγιση, καθώς υποστηρίζει τόσο ταινίες όσο και σειρές, ενώ προσφέρει συγχρονισμό με άλλες πλατφόρμες και δυνατότητες προσωποποιημένων προτάσεων. Ωστόσο, η πολυπλοκότητα της διεπαφής και η εξάρτηση από εξωτερικές υπηρεσίες ενδέχεται να επηρεάσουν την ευχρηστία της, ιδιαίτερα για λιγότερο έμπειρους χρήστες.

Από την άλλη πλευρά, η εφαρμογή CineTrak παρέχει προηγμένες δυνατότητες φιλτραρίσματος και οργάνωσης περιεχομένου, καθώς και πλούσια πληροφοριακή προβολή. Παρά τα πλεονεκτήματά της, η διαθεσιμότητά της κυρίως σε πλατφόρμες Android περιορίζει την προσβασιμότητά της για χρήστες iOS.

Σε σύγκριση με τις παραπάνω εφαρμογές, η προτεινόμενη εφαρμογή της παρούσας εργασίας επιχειρεί να συνδυάσει τα βασικά τους πλεονεκτήματα, δίνοντας έμφαση στην απλότητα, τη λειτουργικότητα και την προσαρμοστικότητα. Συγκεκριμένα, υποστηρίζει τόσο ταινίες όσο και τηλεοπτικές σειρές, προσφέρει δυνατότητες αναζήτησης και φιλτραρίσματος, καθώς και διαχείριση προσωπικής λίστας αγαπημένων.

Επιπλέον, η ενσωμάτωση τεχνικών μηχανικής μάθησης για την παραγωγή προσωποποιημένων προτάσεων αποτελεί ένα από τα βασικά χαρακτηριστικά που διαφοροποιούν την εφαρμογή. Σε αντίθεση με ορισμένες υπάρχουσες λύσεις, όπου οι προτάσεις βασίζονται σε απλούστερα κριτήρια, η προτεινόμενη προσέγγιση αξιοποιεί δεδομένα χρήστη με στόχο την πιο στοχευμένη και ουσιαστική εξατομίκευση.

Τέλος, ιδιαίτερη έμφαση δίνεται στον σχεδιασμό της διεπαφής χρήστη, με στόχο τη δημιουργία ενός απλού και κατανοητού περιβάλλοντος, το οποίο διευκολύνει την πλοήγηση και τη χρήση της εφαρμογής. Η προσέγγιση αυτή αποσκοπεί στη βελτίωση της συνολικής εμπειρίας χρήστη, αντιμετωπίζοντας αδυναμίες που εντοπίζονται σε πιο πολύπλοκες ή λιγότερο εργονομικές εφαρμογές.

Συνολικά, η συγκριτική ανάλυση καταδεικνύει ότι, παρά την ύπαρξη αξιολογών εφαρμογών, παραμένουν ορισμένα ζητήματα που σχετίζονται με την πληρότητα λειτουργιών, τον βαθμό προσωποποίησης και τη συνολική εμπειρία χρήσης. Τα σημεία αυτά αναδεικνύουν συγκεκριμένους περιορισμούς των υπάρχουσών λύσεων.

2.2.3 Περιορισμοί υπάρχουσών λύσεων

Η συγκριτική ανάλυση που προηγήθηκε ανέδειξε ότι, παρά την ύπαρξη λειτουργικών και δημοφιλών εφαρμογών, εξακολουθούν να υφίστανται ορισμένοι σημαντικοί περιορισμοί που επηρεάζουν την αποτελεσματικότητά τους και την εμπειρία χρήσης.

Ένα βασικό ζήτημα αφορά την έλλειψη ολοκληρωμένης προσέγγισης, καθώς πολλές εφαρμογές επικεντρώνονται σε επιμέρους λειτουργίες, όπως η καταγραφή προβολών ή η κοινωνική αλληλεπίδραση, χωρίς να προσφέρουν ένα ενιαίο περιβάλλον που να συνδυάζει αναζήτηση, φιλτράρισμα και οργάνωση περιεχομένου. Αυτό έχει ως αποτέλεσμα την αποσπασματική εμπειρία χρήσης.

Παράλληλα, παρατηρείται περιορισμένη αξιοποίηση προηγμένων μηχανισμών προσωποποίησης. Σε αρκετές περιπτώσεις, οι προτάσεις βασίζονται σε γενικά ή στατικά κριτήρια, γεγονός που μειώνει τη συνάφεια των αποτελεσμάτων και δεν ανταποκρίνεται πλήρως στις ιδιαίτερες προτιμήσεις κάθε χρήστη.

Επιπλέον, η διαχείριση μεγάλου όγκου δεδομένων συχνά δεν συνοδεύεται από αντίστοιχα αποτελεσματικά εργαλεία φιλτραρίσματος και οργάνωσης. Αυτό δυσχεραίνει την πλοήγηση και αυξάνει τον χρόνο που απαιτείται για την εύρεση κατάλληλου περιεχομένου.

Σε επίπεδο διεπαφής χρήστη, εντοπίζονται περιπτώσεις όπου η πολυπλοκότητα ή η έλλειψη σαφούς δομής επηρεάζουν αρνητικά τη χρηστικότητα της εφαρμογής. Η απουσία απλότητας και συνέπειας στον σχεδιασμό μπορεί να λειτουργήσει αποτρεπτικά, ιδιαίτερα για νέους χρήστες.

Τέλος, η εξάρτηση από εξωτερικές πηγές δεδομένων και υπηρεσίες ενδέχεται να δημιουργεί ζητήματα απόδοσης ή διαθεσιμότητας, επηρεάζοντας τη συνολική αξιοπιστία των εφαρμογών.

Λαμβάνοντας υπόψη τους παραπάνω περιορισμούς, καθίσταται εμφανής η ανάγκη για μια πιο ολοκληρωμένη και ισορροπημένη προσέγγιση. Η προτεινόμενη εφαρμογή της παρούσας εργασίας σχεδιάστηκε με στόχο την αντιμετώπιση των συγκεκριμένων αδυναμιών, συνδυάζοντας λειτουργικότητα, απλότητα και προσωποποιημένη εμπειρία χρήστη σε ένα ενιαίο περιβάλλον.

2.3 Τεχνολογίες ανάπτυξης κινητών εφαρμογών

Η ανάπτυξη εφαρμογών για κινητές συσκευές αποτελεί έναν ιδιαίτερα δυναμικό τομέα της πληροφορικής, ο οποίος εξελίσσεται διαρκώς, ακολουθώντας τις τεχνολογικές απαιτήσεις και τις ανάγκες των χρηστών. Οι σύγχρονες εφαρμογές δεν περιορίζονται μόνο στη βασική

λειτουργικότητα, αλλά δίνουν ιδιαίτερη έμφαση στην απόδοση, την ασφάλεια και κυρίως στην εμπειρία χρήστη.

Οι τεχνολογίες που χρησιμοποιούνται για την ανάπτυξη κινητών εφαρμογών διακρίνονται σε διαφορετικές κατηγορίες, με βασικό διαχωρισμό μεταξύ native και cross-platform προσεγγίσεων. Στην παρούσα εργασία δίνεται έμφαση στην ανάπτυξη native εφαρμογής για το λειτουργικό σύστημα iOS, καθώς αυτή η προσέγγιση επιτρέπει την πλήρη αξιοποίηση των δυνατοτήτων της συσκευής και του λειτουργικού συστήματος.

Η επιλογή των κατάλληλων τεχνολογιών επηρεάζει σημαντικά την ποιότητα της τελικής εφαρμογής, τόσο σε επίπεδο απόδοσης όσο και σε επίπεδο επεκτασιμότητας. Παράλληλα, η χρήση σύγχρονων εργαλείων ανάπτυξης συμβάλλει στη βελτίωση της παραγωγικότητας και στη διατήρηση ενός καθαρού και οργανωμένου κώδικα.

2.3.1 Εγγενείς εφαρμογές iOS (Native iOS applications)

Οι εγγενείς εφαρμογές iOS αναπτύσσονται ειδικά για το οικοσύστημα της Apple, αξιοποιώντας τις επίσημες γλώσσες προγραμματισμού και τα εργαλεία που παρέχει η πλατφόρμα. Η προσέγγιση αυτή προσφέρει σημαντικά πλεονεκτήματα, όπως υψηλή απόδοση, άμεση πρόσβαση στο υλικό της συσκευής και καλύτερη ενσωμάτωση με το λειτουργικό σύστημα.

Η ανάπτυξη native εφαρμογών επιτρέπει τη δημιουργία εφαρμογών που ανταποκρίνονται πλήρως στις σχεδιαστικές οδηγίες της πλατφόρμας, διασφαλίζοντας συνέπεια και ομοιομορφία στην εμπειρία χρήστη. Επιπλέον, δίνεται η δυνατότητα αξιοποίησης προηγμένων λειτουργιών, όπως ειδοποιήσεις, αποθήκευση δεδομένων στη συσκευή και χρήση αισθητήρων.

Παρά τα πλεονεκτήματα, η ανάπτυξη εγγενών (native) εφαρμογών απαιτεί εξειδικευμένη γνώση και μεγαλύτερη επένδυση χρόνου, ιδιαίτερα όταν υπάρχει ανάγκη υποστήριξης πολλαπλών πλατφορμών. Ωστόσο, για εφαρμογές που δίνουν έμφαση στην ποιότητα και την απόδοση, όπως η προτεινόμενη στην παρούσα εργασία, η επιλογή αυτή θεωρείται η πλέον κατάλληλη.

2.3.2 Σχεδιασμός διεπαφής και εμπειρίας χρήστη (UI/UX)

Ο σχεδιασμός της διεπαφής χρήστη (User Interface - UI) και της εμπειρίας χρήστη (User Experience - UX) αποτελεί καθοριστικό παράγοντα για την επιτυχία μιας εφαρμογής. Μια εφαρμογή μπορεί να διαθέτει πλούσια λειτουργικότητα, ωστόσο αν η χρήση της είναι δύσκολη ή μη κατανοητή, είναι πιθανό να μην γίνει αποδεκτή από τους χρήστες.

Ο σχεδιασμός UI αφορά την οπτική απεικόνιση της εφαρμογής, όπως τα χρώματα, η τυπογραφία, τα κουμπιά και η διάταξη των στοιχείων. Από την άλλη πλευρά, το UX σχετίζεται με τον τρόπο με τον οποίο ο χρήστης αλληλεπιδρά με την εφαρμογή και τη συνολική εμπειρία που αποκομίζει από αυτή.

Στις σύγχρονες εφαρμογές δίνεται ιδιαίτερη έμφαση σε αρχές όπως η απλότητα, η συνέπεια και η ευκολία πλοήγησης. Η χρήση ξεκάθαρης δομής και η αποφυγή περιττής πολυπλοκότητας συμβάλλουν στη βελτίωση της εμπειρίας χρήστη.

Επιπλέον, η προσαρμοστικότητα της διεπαφής σε διαφορετικές συσκευές και μεγέθη οθόνης αποτελεί σημαντικό παράγοντα, ιδιαίτερα στο περιβάλλον των κινητών συσκευών. Η αξιοποίηση σύγχρονων εργαλείων σχεδιασμού, όπως το SwiftUI, επιτρέπει τη δημιουργία δυναμικών και ευέλικτων διεπαφών που ανταποκρίνονται στις ανάγκες των χρηστών.

2.4 Μηχανική μάθηση σε συστήματα προτάσεων

Η μηχανική μάθηση αποτελεί έναν από τους πλέον σημαντικούς τομείς της σύγχρονης επιστήμης των υπολογιστών, με ευρεία εφαρμογή σε πλήθος συστημάτων που απαιτούν ανάλυση δεδομένων και εξαγωγή συμπερασμάτων. Στον χώρο της ψηφιακής ψυχαγωγίας, η μηχανική μάθηση αξιοποιείται κυρίως για την ανάπτυξη συστημάτων προτάσεων, τα οποία στοχεύουν στην εξατομίκευση της εμπειρίας του χρήστη.

Τα συστήματα προτάσεων χρησιμοποιούνται εκτενώς σε πλατφόρμες περιεχομένου, προκειμένου να προτείνουν ταινίες και σειρές που είναι πιθανό να ενδιαφέρουν τον χρήστη. Η αποτελεσματικότητά τους βασίζεται στην ανάλυση δεδομένων, όπως το ιστορικό προβολών, οι αξιολογήσεις και οι προτιμήσεις.

Η ενσωμάτωση τεχνικών μηχανικής μάθησης στην παρούσα εργασία αποσκοπεί στη βελτίωση της διαδικασίας αναζήτησης και επιλογής περιεχομένου, μειώνοντας τον χρόνο που απαιτείται για τη λήψη αποφάσεων από τον χρήστη.

2.4.1 Εισαγωγή στη μηχανική μάθηση

Η μηχανική μάθηση είναι ένας κλάδος της τεχνητής νοημοσύνης που επιτρέπει στα υπολογιστικά συστήματα να μαθαίνουν από δεδομένα, χωρίς να απαιτείται ρητός προγραμματισμός για κάθε πιθανή περίπτωση. Μέσω αλγορίθμων και μοντέλων, τα συστήματα μπορούν να εντοπίζουν πρότυπα και να πραγματοποιούν προβλέψεις.

Οι βασικές κατηγορίες μηχανικής μάθησης περιλαμβάνουν την επιβλεπόμενη μάθηση, τη μη επιβλεπόμενη μάθηση και τη μάθηση ενίσχυσης. Στην επιβλεπόμενη μάθηση, το μοντέλο εκπαιδεύεται με δεδομένα που περιλαμβάνουν γνωστά αποτελέσματα, ενώ στη μη επιβλεπόμενη μάθηση το σύστημα προσπαθεί να εντοπίσει δομές και σχέσεις χωρίς προκαθορισμένες ετικέτες.

Η επιλογή της κατάλληλης μεθόδου εξαρτάται από το είδος των δεδομένων και τον στόχο της εφαρμογής. Στην περίπτωση συστημάτων προτάσεων, συχνά χρησιμοποιείται συνδυασμός τεχνικών για την επίτευξη καλύτερων αποτελεσμάτων.

2.4.2 Συστήματα συστάσεων

Τα συστήματα συστάσεων αποτελούν ειδική κατηγορία εφαρμογών της μηχανικής μάθησης, με στόχο την παροχή εξατομικευμένων προτάσεων προς τον χρήστη. Η λειτουργία τους βασίζεται στην ανάλυση δεδομένων χρήστη και στην εξαγωγή προτύπων συμπεριφοράς.

Υπάρχουν διάφορες προσεγγίσεις στην ανάπτυξη τέτοιων συστημάτων. Μία από τις πιο διαδεδομένες είναι το συνεργατικό φιλτράρισμα (collaborative filtering), το οποίο βασίζεται στις προτιμήσεις άλλων χρηστών με παρόμοια χαρακτηριστικά. Μια άλλη προσέγγιση είναι το φιλτράρισμα βάσει περιεχομένου (content-based filtering), όπου οι προτάσεις προκύπτουν από τα χαρακτηριστικά των αντικειμένων που έχει ήδη επιλέξει ο χρήστης.

Στην πράξη, πολλά σύγχρονα συστήματα χρησιμοποιούν υβριδικές μεθόδους, συνδυάζοντας διαφορετικές τεχνικές, προκειμένου να επιτύχουν μεγαλύτερη ακρίβεια και αξιοπιστία.

Η σημασία των συστημάτων συστάσεων είναι ιδιαίτερα εμφανής σε περιβάλλοντα με μεγάλο όγκο δεδομένων, όπου η απλή αναζήτηση δεν επαρκεί για την εύρεση κατάλληλου περιεχομένου.

2.4.3 Εφαρμογές μηχανικής μάθησης σε εφαρμογές κινητών συσκευών

Η ενσωμάτωση μηχανικής μάθησης σε εφαρμογές κινητών συσκευών έχει γνωρίσει σημαντική ανάπτυξη τα τελευταία χρόνια, κυρίως λόγω της αύξησης της υπολογιστικής ισχύος των συσκευών και της διαθεσιμότητας εξειδικευμένων εργαλείων ανάπτυξης.

Οι εφαρμογές κινητών αξιοποιούν τη μηχανική μάθηση για διάφορους σκοπούς, όπως η αναγνώριση προτύπων, η ανάλυση συμπεριφοράς χρήστη και η παροχή προσωποποιημένων υπηρεσιών. Στην περίπτωση εφαρμογών ψυχαγωγικού περιεχομένου, η κύρια χρήση αφορά την πρόβλεψη προτιμήσεων και την πρόταση σχετικού περιεχομένου.

Η χρήση τεχνολογιών όπως το CoreML επιτρέπει την εκτέλεση μοντέλων μηχανικής μάθησης απευθείας στη συσκευή, διασφαλίζοντας ταχύτητα και προστασία των προσωπικών δεδομένων. Παράλληλα, η αξιοποίηση εξωτερικών εργαλείων, όπως το TensorFlow, διευκολύνει τη διαδικασία εκπαίδευσης και βελτιστοποίησης των μοντέλων.

Η ενσωμάτωση των παραπάνω τεχνολογιών στην προτεινόμενη εφαρμογή συμβάλλει στη δημιουργία μιας πιο έξυπνης και προσαρμοστικής εμπειρίας χρήστη, η οποία ανταποκρίνεται στις σύγχρονες απαιτήσεις της ψηφιακής εποχής.

3. Μεθοδολογία

Στο κεφάλαιο αυτό παρουσιάζεται η μεθοδολογία που ακολουθήθηκε για τον σχεδιασμό, την ανάπτυξη και την αξιολόγηση της εφαρμογής CineMax, μιας εφαρμογής κινητών συσκευών για το λειτουργικό σύστημα iOS, η οποία παρέχει δυνατότητες αναζήτησης, οργάνωσης και προσωποποιημένων προτάσεων κινηματογραφικού περιεχομένου.

Η ανάπτυξη της εφαρμογής πραγματοποιήθηκε ακολουθώντας μια σταδιακή διαδικασία, η οποία ξεκινά από τη μελέτη της σχετικής βιβλιογραφίας και των υπάρχουσών εφαρμογών, συνεχίζεται με την ανάλυση των απαιτήσεων και τον σχεδιασμό της αρχιτεκτονικής του συστήματος και ολοκληρώνεται με την υλοποίηση, τις δοκιμές και την αξιολόγηση της εφαρμογής μέσω πραγματικών χρηστών. Η συγκεκριμένη προσέγγιση επέτρεψε τη συστηματική ανάπτυξη του συστήματος, εξασφαλίζοντας ότι κάθε στάδιο βασίζεται στα αποτελέσματα του προηγούμενου και οδηγεί σε μια ολοκληρωμένη και λειτουργική εφαρμογή.

Η μεθοδολογία που ακολουθήθηκε περιλαμβάνει τα παρακάτω στάδια.

Βήμα 1: Μελέτη βιβλιογραφίας και υφιστάμενων εφαρμογών

Αρχικά πραγματοποιήθηκε βιβλιογραφική έρευνα σχετικά με τις εφαρμογές κινητών συσκευών, τις τεχνολογίες ανάπτυξης εφαρμογών iOS, τα συστήματα προτάσεων (Recommendation Systems), καθώς και τις τεχνολογίες μηχανικής μάθησης που χρησιμοποιούνται σε αντίστοιχα πληροφοριακά συστήματα.

Παράλληλα, μελετήθηκαν δημοφιλείς εφαρμογές κινηματογραφικού περιεχομένου, όπως το IMDb, το Letterboxd και το TMDb, με στόχο την κατανόηση των λειτουργιών που προσφέρουν, των πλεονεκτημάτων τους αλλά και των περιορισμών τους. Η μελέτη αυτή συνέβαλε στον καθορισμό των λειτουργικών χαρακτηριστικών που θα ενσωματώνονταν στην εφαρμογή CineMax.

Βήμα 2: Ανάλυση απαιτήσεων

Στη συνέχεια πραγματοποιήθηκε η ανάλυση των λειτουργικών και μη λειτουργικών απαιτήσεων της εφαρμογής.

Καταγράφηκαν οι βασικές λειτουργίες που θα έπρεπε να υποστηρίζονται, όπως η αναζήτηση ταινιών, η προβολή αναλυτικών πληροφοριών, η αποθήκευση αγαπημένων ταινιών, η δημιουργία λίστας παρακολούθησης (Watchlist), η καταγραφή των ταινιών που έχει παρακολουθήσει ο χρήστης, η εφαρμογή φίλτρων αναζήτησης, καθώς και η παραγωγή προσωποποιημένων προτάσεων.

Παράλληλα, καθορίστηκαν απαιτήσεις που σχετίζονται με την απόδοση, την ευχρηστία, την ασφάλεια και τη δυνατότητα μελλοντικής επέκτασης της εφαρμογής.

Βήμα 3: Σχεδιασμός της εφαρμογής

Αφού ολοκληρώθηκε η ανάλυση των απαιτήσεων, σχεδιάστηκε η συνολική αρχιτεκτονική του συστήματος.

Στο στάδιο αυτό δημιουργήθηκαν τα διαγράμματα UML, σχεδιάστηκαν οι βασικές οθόνες της εφαρμογής μέσω wireframes και καθορίστηκε η δομή των επιμέρους ενοτήτων του συστήματος σύμφωνα με το αρχιτεκτονικό πρότυπο MVVM (Model – View – ViewModel).

Ιδιαίτερη έμφαση δόθηκε στη δημιουργία μιας απλής και φιλικής διεπαφής χρήστη, ώστε η εφαρμογή να μπορεί να χρησιμοποιηθεί εύκολα ακόμη και από χρήστες χωρίς ιδιαίτερη εξοικείωση με αντίστοιχες εφαρμογές.

Βήμα 4: Επιλογή τεχνολογιών ανάπτυξης

Στο επόμενο στάδιο επιλέχθηκαν οι τεχνολογίες που χρησιμοποιήθηκαν για την υλοποίηση της εφαρμογής.

Η ανάπτυξη πραγματοποιήθηκε με τη γλώσσα προγραμματισμού Swift και το framework SwiftUI, χρησιμοποιώντας το περιβάλλον ανάπτυξης Xcode. Για την αποθήκευση και διαχείριση δεδομένων αξιοποιήθηκαν οι υπηρεσίες της Firebase.

Επιπλέον, για την ανάπτυξη του συστήματος προσωποποιημένων προτάσεων αξιοποιήθηκαν τεχνολογίες μηχανικής μάθησης, όπως το TensorFlow για την εκπαίδευση του μοντέλου και το CoreML για την ενσωμάτωσή του στην εφαρμογή.

Βήμα 5: Υλοποίηση της εφαρμογής

Μετά τον σχεδιασμό πραγματοποιήθηκε η ανάπτυξη όλων των λειτουργικών ενοτήτων της εφαρμογής.

Αναπτύχθηκαν οι βασικές οθόνες πλοήγησης, οι λειτουργίες αναζήτησης και φιλτραρίσματος, η προβολή πληροφοριών ταινιών, η διαχείριση αγαπημένων, της Watchlist και των ήδη παρακολουθημένων ταινιών, καθώς και το σύστημα αυθεντικοποίησης χρηστών μέσω λογαριασμού Google.

Παράλληλα, υλοποιήθηκε ο μηχανισμός προσωποποιημένων προτάσεων, ο οποίος αξιοποιεί τα δεδομένα αλληλεπίδρασης του χρήστη για την παραγωγή προτάσεων που ανταποκρίνονται καλύτερα στις προτιμήσεις του.

Βήμα 6: Έλεγχος λειτουργίας και αξιολόγηση

Μετά την ολοκλήρωση της ανάπτυξης πραγματοποιήθηκαν δοκιμές λειτουργικότητας σε όλες τις βασικές λειτουργίες της εφαρμογής, προκειμένου να εντοπιστούν και να διορθωθούν πιθανά σφάλματα.

Τα αποτελέσματα χρησιμοποιήθηκαν για τη διαμόρφωση των τελικών βελτιώσεων της εφαρμογής.

4. Τεχνολογίες και Εργαλεία Ανάπτυξης

Η ανάπτυξη μιας σύγχρονης εφαρμογής κινητών συσκευών απαιτεί τον συνδυασμό κατάλληλων εργαλείων, τεχνολογιών και πλαισίων ανάπτυξης, τα οποία εξασφαλίζουν τόσο τη λειτουργικότητα όσο και την επεκτασιμότητα του συστήματος. Στο πλαίσιο της παρούσας εργασίας, επιλέχθηκε ένα σύνολο τεχνολογιών που υποστηρίζουν την ανάπτυξη native εφαρμογής για το λειτουργικό σύστημα iOS, με έμφαση στην υψηλή απόδοση, την ευχρηστία και τη δυνατότητα ενσωμάτωσης σύγχρονων λειτουργιών, όπως η μηχανική μάθηση, η ασφαλής αυθεντικοποίηση χρηστών και η διαχείριση δεδομένων.

Στο κεφάλαιο αυτό παρουσιάζονται αναλυτικά τα βασικά εργαλεία και οι τεχνολογίες που χρησιμοποιήθηκαν για τον σχεδιασμό, την ανάπτυξη και την υλοποίηση της εφαρμογής, καθώς και ο ρόλος που επιτελεί η καθεμία στη συνολική αρχιτεκτονική του συστήματος.

Το κεφάλαιο αυτό παρουσιάζει τις τεχνολογίες και τα εργαλεία που χρησιμοποιήθηκαν για την ανάπτυξη της εφαρμογής CineMax. Η επιλογή των κατάλληλων τεχνολογιών αποτελεί ιδιαίτερα σημαντικό στάδιο κατά την ανάπτυξη μιας σύγχρονης εφαρμογής κινητών συσκευών, καθώς επηρεάζει άμεσα την απόδοση, την επεκτασιμότητα, την ασφάλεια και τη συνολική εμπειρία του χρήστη.

Για την υλοποίηση της εφαρμογής αξιοποιήθηκαν σύγχρονες τεχνολογίες του οικοσυστήματος της Apple, όπως η γλώσσα προγραμματισμού Swift και το framework SwiftUI, τα οποία προσφέρουν υψηλή απόδοση και δυνατότητα ανάπτυξης εγγενών (native) εφαρμογών για το λειτουργικό σύστημα iOS. Παράλληλα, χρησιμοποιήθηκαν υπηρεσίες της Google μέσω της πλατφόρμας Firebase για την αυθεντικοποίηση των χρηστών.

Ιδιαίτερο χαρακτηριστικό της εφαρμογής αποτελεί η ενσωμάτωση τεχνολογιών μηχανικής μάθησης. Συγκεκριμένα, αναπτύχθηκε ένα απλό μοντέλο πρόβλεψης με χρήση του TensorFlow, το οποίο εκπαιδεύτηκε σε δεδομένα σχετικά με τις προτιμήσεις των χρηστών και στη συνέχεια μετατράπηκε σε μορφή συμβατή με το CoreML, ώστε να εκτελείται απευθείας στη συσκευή του χρήστη. Η προσέγγιση αυτή επιτρέπει την παραγωγή προσωποποιημένων προτάσεων με μικρό χρόνο απόκρισης και χωρίς να απαιτείται συνεχής επικοινωνία με εξωτερικούς εξυπηρετητές.

Στις επόμενες ενότητες παρουσιάζονται αναλυτικά τα εργαλεία ανάπτυξης, οι γλώσσες προγραμματισμού, οι υπηρεσίες υποστήριξης και οι τεχνολογίες μηχανικής μάθησης που χρησιμοποιήθηκαν κατά την υλοποίηση της εφαρμογής, καθώς και ο ρόλος που διαδραματίζει κάθε μία στη συνολική αρχιτεκτονική του συστήματος.

4.1 Περιβάλλον ανάπτυξης

Η επιλογή του κατάλληλου περιβάλλοντος ανάπτυξης αποτελεί καθοριστικό παράγοντα για την αποτελεσματική υλοποίηση μιας εφαρμογής λογισμικού. Ένα ολοκληρωμένο περιβάλλον ανάπτυξης (Integrated Development Environment – IDE) παρέχει όλα τα απαραίτητα εργαλεία για τη συγγραφή, τον έλεγχο, την αποσφαλμάτωση και τη δοκιμή του κώδικα, επιτρέποντας στον προγραμματιστή να εργάζεται σε ένα ενιαίο και οργανωμένο περιβάλλον.

Η σωστή επιλογή IDE συμβάλλει σημαντικά στη μείωση του χρόνου ανάπτυξης, στη βελτίωση της ποιότητας του παραγόμενου λογισμικού και στον ευκολότερο εντοπισμό λαθών κατά τη διάρκεια της υλοποίησης. Επιπλέον, η ενσωμάτωση εργαλείων προσομοίωσης και δοκιμών επιτρέπει την αξιολόγηση της εφαρμογής σε διαφορετικές συσκευές χωρίς να απαιτείται φυσικός εξοπλισμός.

Για την ανάπτυξη της εφαρμογής CineMax χρησιμοποιήθηκε το επίσημο περιβάλλον ανάπτυξης της Apple, το οποίο παρέχει πλήρη υποστήριξη για την ανάπτυξη εφαρμογών iOS και συνεργάζεται άμεσα με τις τεχνολογίες Swift, SwiftUI και CoreML.

4.1.1 Xcode

Το Xcode αποτελεί το επίσημο ολοκληρωμένο περιβάλλον ανάπτυξης (Integrated Development Environment - IDE) της Apple για τη δημιουργία εφαρμογών σε συσκευές iPhone, iPad, Apple Watch και Mac. Περιλαμβάνει όλα τα απαραίτητα εργαλεία για τη συγγραφή, μεταγλώττιση, εκτέλεση και αποσφαλμάτωση εφαρμογών μέσα από ένα ενιαίο περιβάλλον εργασίας.

Ένα από τα σημαντικότερα πλεονεκτήματα του Xcode είναι η άμεση ενσωμάτωσή του με το οικοσύστημα της Apple, καθώς παρέχει πρόσβαση σε εργαλεία όπως ο iOS Simulator, το Interface Builder, τα εργαλεία ανάλυσης επιδόσεων (Instruments) και προηγμένες δυνατότητες debugging. Με τον τρόπο αυτό, ο προγραμματιστής μπορεί να ελέγχει τη συμπεριφορά της εφαρμογής σε διαφορετικές εκδόσεις του λειτουργικού συστήματος και σε διαφορετικά μοντέλα συσκευών.

Επιπλέον, το Xcode υποστηρίζει πλήρως τις σύγχρονες γλώσσες και τεχνολογίες της Apple, όπως τη Swift, το SwiftUI, το Combine και το CoreML, διευκολύνοντας την ανάπτυξη εφαρμογών που αξιοποιούν δυνατότητες τεχνητής νοημοσύνης, cloud υπηρεσιών και σύγχρονων διεπαφών χρήστη.

Στο πλαίσιο της παρούσας εργασίας, το Xcode χρησιμοποιήθηκε για τη δημιουργία του έργου, την ανάπτυξη όλων των Views και ViewModels της εφαρμογής, τη διαχείριση των βιβλιοθηκών, την εκτέλεση δοκιμών στον εξομοιωτή iOS καθώς και την τελική μεταγλώττιση της εφαρμογής.

4.1.2 Git και διαχείριση εκδόσεων

Η διαχείριση εκδόσεων αποτελεί βασικό μέρος της ανάπτυξης κάθε σύγχρονου λογισμικού, καθώς επιτρέπει την καταγραφή όλων των αλλαγών που πραγματοποιούνται στον πηγαίο κώδικα κατά τη διάρκεια της ανάπτυξης. Με τον τρόπο αυτό είναι δυνατή η επαναφορά προηγούμενων εκδόσεων, η παρακολούθηση της εξέλιξης του έργου και η ασφαλής ανάπτυξη νέων λειτουργιών.

Το Git αποτελεί το πιο διαδεδομένο σύστημα ελέγχου εκδόσεων παγκοσμίως. Βασίζεται στην έννοια των αποθετηρίων (repositories), μέσα στα οποία αποθηκεύονται όλες οι εκδόσεις του έργου. Παράλληλα, υποστηρίζει τη δημιουργία διαφορετικών branches, επιτρέποντας την ανεξάρτητη ανάπτυξη νέων λειτουργιών χωρίς να επηρεάζεται η σταθερή έκδοση της εφαρμογής.

Η χρήση του Git προσφέρει σημαντικά πλεονεκτήματα, όπως η εύκολη συνεργασία μεταξύ πολλών προγραμματιστών, η διατήρηση ιστορικού αλλαγών και η ασφαλής συγχώνευση διαφορετικών εκδόσεων του κώδικα.

Στην παρούσα εργασία, το Git χρησιμοποιήθηκε καθ' όλη τη διάρκεια της ανάπτυξης για την αποθήκευση των αλλαγών, την οργάνωση του πηγαίου κώδικα και τη διατήρηση διαφορετικών εκδόσεων της εφαρμογής μέχρι την τελική μορφή της.

4.2 Γλώσσες και πλαίσια ανάπτυξης

Η επιλογή της κατάλληλης γλώσσας προγραμματισμού και των αντίστοιχων πλαισίων ανάπτυξης αποτελεί έναν από τους σημαντικότερους παράγοντες για την επιτυχημένη υλοποίηση μιας εφαρμογής κινητών συσκευών. Οι τεχνολογίες αυτές καθορίζουν όχι μόνο την απόδοση της εφαρμογής, αλλά και την ευκολία συντήρησης, την επεκτασιμότητα και την εμπειρία που προσφέρεται στον τελικό χρήστη.

Για την ανάπτυξη της εφαρμογής CineMax αξιοποιήθηκαν σύγχρονες τεχνολογίες της Apple, οι οποίες έχουν σχεδιαστεί ειδικά για το οικοσύστημα του iOS. Η χρήση τους επέτρεψε τη δημιουργία ενός εύχρηστου και αποδοτικού περιβάλλοντος, καθώς και την εύκολη διασύνδεση με υπηρεσίες όπως το Firebase και το CoreML.

4.2.1 Swift

Η Swift αποτελεί τη βασική γλώσσα προγραμματισμού που προτείνει η Apple για την ανάπτυξη εφαρμογών σε όλες τις πλατφόρμες του οικοσυστήματός της. Παρουσιάστηκε το 2014 ως αντικαταστάτης της Objective-C και σχεδιάστηκε με στόχο την υψηλή απόδοση, την ευκολία εκμάθησης και την αυξημένη ασφάλεια κατά την ανάπτυξη λογισμικού.

Ένα από τα σημαντικότερα χαρακτηριστικά της Swift είναι η ασφάλεια τύπων (type safety), η οποία μειώνει σημαντικά τα σφάλματα κατά τη μεταγλώττιση του κώδικα. Επιπλέον, διαθέτει αυτόματη διαχείριση μνήμης μέσω του μηχανισμού Automatic Reference Counting (ARC), επιτρέποντας την αποδοτική χρήση των πόρων της συσκευής χωρίς να απαιτείται χειροκίνητη διαχείριση από τον προγραμματιστή.

Η Swift υποστηρίζει σύγχρονες προγραμματιστικές τεχνικές, όπως ο αντικειμενοστραφής προγραμματισμός, ο συναρτησιακός προγραμματισμός, οι γενικεύσεις (Generics), τα πρωτόκολλα (Protocols) και οι ασύγχρονες λειτουργίες (Async/Await). Τα χαρακτηριστικά αυτά καθιστούν τον κώδικα πιο οργανωμένο, ευανάγνωστο και εύκολο στη συντήρηση.

Στην εφαρμογή CineMax η Swift χρησιμοποιήθηκε για την ανάπτυξη ολόκληρης της επιχειρησιακής λογικής της εφαρμογής. Μέσω αυτής υλοποιήθηκαν οι κλάσεις όπως η διαχείριση των αγαπημένων ταινιών, της watchlist και των ταινιών που έχει παρακολουθήσει ο χρήστης, η αυθεντικοποίηση μέσω Firebase, καθώς και η ενσωμάτωση του συστήματος προτάσεων με χρήση CoreML.

4.2.2 SwiftUI

Το SwiftUI αποτελεί το σύγχρονο framework της Apple για τη δημιουργία διεπαφών χρήστη (User Interface). Σε αντίθεση με τις παλαιότερες τεχνολογίες ανάπτυξης, χρησιμοποιεί δηλωτική (declarative) προσέγγιση, σύμφωνα με την οποία ο

προγραμματιστής περιγράφει τον τρόπο με τον οποίο πρέπει να εμφανίζεται η διεπαφή και το framework αναλαμβάνει την αυτόματη ενημέρωσή της όταν αλλάζουν τα δεδομένα.

Η συγκεκριμένη φιλοσοφία μειώνει σημαντικά τον απαιτούμενο κώδικα, ενώ παράλληλα διευκολύνει τη συντήρηση και την επέκταση της εφαρμογής. Επιπλέον, το SwiftUI συνεργάζεται άμεσα με το Combine Framework και τα Observable Objects, επιτρέποντας την αυτόματη ανανέωση των οθονών όταν μεταβάλλονται τα δεδομένα της εφαρμογής.

Ένα ακόμη σημαντικό χαρακτηριστικό του SwiftUI είναι η δυνατότητα Live Preview, μέσω της οποίας ο προγραμματιστής μπορεί να παρακολουθεί σε πραγματικό χρόνο τις αλλαγές που πραγματοποιεί στον σχεδιασμό της διεπαφής χωρίς να απαιτείται κάθε φορά εκτέλεση της εφαρμογής.

Στην παρούσα εργασία, το SwiftUI χρησιμοποιήθηκε για την ανάπτυξη όλων των οθονών της εφαρμογής, όπως της αρχικής οθόνης (HomeView), της αναζήτησης (SearchView), των αγαπημένων (FavoritesView), της Watchlist, του προφίλ χρήστη, των ρυθμίσεων, καθώς και της οθόνης προβολής αναλυτικών πληροφοριών κάθε ταινίας (MovieDetailView). Μέσω του SwiftUI υλοποιήθηκαν επίσης οι λίστες ταινιών, οι κάρτες προβολής, τα φίλτρα αναζήτησης και η πλοήγηση μεταξύ των διαφορετικών λειτουργιών της εφαρμογής.

4.3 Βάσεις δεδομένων και υπηρεσίες υποστήριξης

Η σωστή διαχείριση των δεδομένων αποτελεί βασική προϋπόθεση για την εύρυθμη λειτουργία μιας σύγχρονης εφαρμογής. Εκτός από την αποθήκευση πληροφοριών, απαιτείται ασφαλής διαχείριση των στοιχείων των χρηστών, γρήγορη ανάκτηση δεδομένων και δυνατότητα συγχρονισμού μεταξύ διαφορετικών συσκευών.

Για τον λόγο αυτό, στην εφαρμογή αξιοποιήθηκαν υπηρεσίες που προσφέρουν ασφαλή αποθήκευση, μηχανισμούς αυθεντικοποίησης και εύκολη ενσωμάτωση με εφαρμογές iOS.

4.3.1 Firebase

Το Firebase αποτελεί μία ολοκληρωμένη πλατφόρμα ανάπτυξης εφαρμογών της Google, η οποία παρέχει πληθώρα υπηρεσιών για την υποστήριξη mobile και web εφαρμογών. Περιλαμβάνει υπηρεσίες αυθεντικοποίησης χρηστών (Firebase Authentication), βάσεις δεδομένων πραγματικού χρόνου (Cloud Firestore και Realtime Database), αποθήκευση αρχείων (Firebase Storage) καθώς και εργαλεία ανάλυσης και παρακολούθησης της λειτουργίας της εφαρμογής.

Ένα από τα σημαντικότερα πλεονεκτήματα του Firebase είναι η εύκολη ενσωμάτωσή του σε εφαρμογές iOS, καθώς διαθέτει επίσημα SDKs που επιτρέπουν την ασφαλή επικοινωνία μεταξύ εφαρμογής και cloud υπηρεσιών χωρίς την ανάγκη ανάπτυξης ξεχωριστού backend. Στην εφαρμογή CineMax το Firebase χρησιμοποιήθηκε κυρίως για την αυθεντικοποίηση των χρηστών μέσω Google Sign-In. Μετά την επιτυχή σύνδεση, η εφαρμογή αποκτά πρόσβαση στα βασικά στοιχεία του χρήστη, όπως το όνομα και η φωτογραφία προφίλ, τα οποία εμφανίζονται στην οθόνη Profile. Η διαδικασία αυτή πραγματοποιείται με ασφαλή τρόπο, αξιοποιώντας τους μηχανισμούς πιστοποίησης της Google.

Η χρήση του Firebase επιτρέπει επίσης την εύκολη επέκταση της εφαρμογής στο μέλλον, ώστε να υποστηρίζει συγχρονισμό των προσωπικών δεδομένων μεταξύ πολλών συσκευών ή αποθήκευση πρόσθετων πληροφοριών στο cloud.

4.3.2 Διαχείριση δεδομένων

Η αποτελεσματική διαχείριση των δεδομένων αποτελεί σημαντικό στοιχείο για την απόδοση και την αξιοπιστία μιας εφαρμογής. Στην παρούσα εφαρμογή χρησιμοποιούνται διαφορετικοί μηχανισμοί αποθήκευσης ανάλογα με τον τύπο των δεδομένων και τις απαιτήσεις της κάθε λειτουργίας.

Οι πληροφορίες που αφορούν τις αγαπημένες ταινίες, τη λίστα παρακολούθησης (Watchlist) και τις ταινίες που έχει ήδη παρακολουθήσει ο χρήστης αποθηκεύονται τοπικά στη συσκευή μέσω του μηχανισμού UserDefaults. Τα δεδομένα μετατρέπονται σε μορφή JSON χρησιμοποιώντας τους μηχανισμούς Codable της Swift και αποθηκεύονται ώστε να είναι διαθέσιμα ακόμη και μετά τον τερματισμό της εφαρμογής.

Παράλληλα, τα δεδομένα των ταινιών δεν αποθηκεύονται μόνιμα, αλλά ανακτώνται δυναμικά κάθε φορά που απαιτείται. Με τον τρόπο αυτό η εφαρμογή εμφανίζει πάντοτε ενημερωμένες πληροφορίες σχετικά με τις ταινίες, τις βαθμολογίες και τα μεταδεδομένα τους.

Η συγκεκριμένη αρχιτεκτονική μειώνει σημαντικά τον απαιτούμενο αποθηκευτικό χώρο της εφαρμογής, ενώ παράλληλα εξασφαλίζει ότι οι πληροφορίες που εμφανίζονται στον χρήστη παραμένουν πάντοτε επίκαιρες.

4.4 Τεχνολογίες μηχανικής μάθησης

Η ενσωμάτωση τεχνολογιών μηχανικής μάθησης (Machine Learning) στις εφαρμογές κινητών συσκευών αποτελεί μία από τις σημαντικότερες εξελίξεις των τελευταίων ετών, καθώς επιτρέπει την παροχή προσωποποιημένων υπηρεσιών και τη βελτίωση της εμπειρίας χρήστη. Μέσω της αξιοποίησης αλγορίθμων μηχανικής μάθησης, οι εφαρμογές μπορούν να αναλύουν δεδομένα, να αναγνωρίζουν πρότυπα και να λαμβάνουν αποφάσεις χωρίς να απαιτείται η ύπαρξη σταθερών κανόνων για κάθε πιθανό σενάριο.

Στην παρούσα εργασία η μηχανική μάθηση χρησιμοποιείται για την ανάπτυξη ενός συστήματος προτάσεων (Recommendation System), το οποίο προσαρμόζει τις προτεινόμενες ταινίες στις προτιμήσεις του κάθε χρήστη. Η λειτουργία αυτή προσφέρει μια πιο εξατομικευμένη εμπειρία πλοήγησης, καθώς οι προτάσεις βασίζονται σε χαρακτηριστικά του χρήστη αλλά και στα χαρακτηριστικά των διαθέσιμων ταινιών.

Για την ανάπτυξη του συστήματος χρησιμοποιήθηκαν οι τεχνολογίες TensorFlow και CoreML. Το TensorFlow αξιοποιήθηκε για τη δημιουργία και εκπαίδευση του μοντέλου, ενώ το CoreML χρησιμοποιήθηκε για την ενσωμάτωσή του στην εφαρμογή iOS, επιτρέποντας την εκτέλεση των προβλέψεων απευθείας στη συσκευή.

4.4.1 CoreML

Το CoreML αποτελεί το επίσημο framework της Apple για την ενσωμάτωση μοντέλων μηχανικής μάθησης σε εφαρμογές iOS, iPadOS και macOS. Μέσω του CoreML, τα εκπαιδευμένα μοντέλα μπορούν να εκτελούνται απευθείας στη συσκευή του χρήστη, χωρίς να απαιτείται συνεχής επικοινωνία με απομακρυσμένους διακομιστές.

Η τοπική εκτέλεση των μοντέλων παρουσιάζει σημαντικά πλεονεκτήματα. Αρχικά, μειώνεται σημαντικά ο χρόνος απόκρισης της εφαρμογής, καθώς δεν απαιτείται αποστολή δεδομένων μέσω διαδικτύου για την παραγωγή των προβλέψεων. Επιπλέον, προστατεύονται καλύτερα τα προσωπικά δεδομένα των χρηστών, αφού οι πληροφορίες παραμένουν στη συσκευή και δεν αποστέλλονται σε εξωτερικές υπηρεσίες.

Ένα ακόμη σημαντικό χαρακτηριστικό του CoreML είναι η άμεση ενσωμάτωσή του με το οικοσύστημα της Apple. Το framework συνεργάζεται άριστα με τη Swift και το SwiftUI, επιτρέποντας την εύκολη αξιοποίηση των αποτελεσμάτων του μοντέλου μέσα στις οθόνες της εφαρμογής.

Στην εφαρμογή CineMax χρησιμοποιήθηκε ένα εκπαιδευμένο μοντέλο CoreML, το οποίο λαμβάνει ως είσοδο πληροφορίες σχετικά με τα χαρακτηριστικά των ταινιών και το προφίλ του χρήστη και επιστρέφει μία βαθμολογία (score) που εκφράζει το κατά πόσο μια ταινία θεωρείται κατάλληλη για τον συγκεκριμένο χρήστη. Η βαθμολογία αυτή χρησιμοποιείται για την ταξινόμηση των διαθέσιμων ταινιών και την εμφάνιση εξατομικευμένων προτάσεων.

4.4.2 TensorFlow

Το TensorFlow αποτελεί μία από τις δημοφιλέστερες πλατφόρμες ανάπτυξης μοντέλων μηχανικής μάθησης και αναπτύσσεται από την Google. Υποστηρίζει τη δημιουργία και εκπαίδευση αλγορίθμων τεχνητής νοημοσύνης, παρέχοντας εργαλεία για την επεξεργασία δεδομένων, την εκπαίδευση νευρωνικών δικτύων και την αξιολόγηση της απόδοσης των μοντέλων.

Ένα από τα σημαντικότερα πλεονεκτήματα του TensorFlow είναι η δυνατότητα δημιουργίας μοντέλων που μπορούν να χρησιμοποιηθούν σε διαφορετικές πλατφόρμες, όπως web εφαρμογές, cloud υπηρεσίες και εφαρμογές κινητών συσκευών. Μέσω ειδικών εργαλείων μετατροπής, τα εκπαιδευμένα μοντέλα μπορούν να μεταφερθούν στο CoreML και να αξιοποιηθούν απευθείας σε εφαρμογές iOS.

Στην παρούσα εργασία, το TensorFlow χρησιμοποιήθηκε για τη δημιουργία και εκπαίδευση ενός σχετικά απλού μοντέλου πρόβλεψης, κατάλληλου για τις απαιτήσεις της εφαρμογής. Η εκπαίδευση πραγματοποιήθηκε χρησιμοποιώντας δεδομένα που περιγράφουν χαρακτηριστικά των ταινιών, όπως η δημοτικότητα (Popularity), η βαθμολογία χρηστών (Vote Average) και οι κατηγορίες (Genres), καθώς και στοιχεία που σχετίζονται με το προφίλ του χρήστη.

Μετά την ολοκλήρωση της εκπαίδευσης, το μοντέλο μετατράπηκε σε μορφή συμβατή με το CoreML και ενσωματώθηκε στην εφαρμογή, ώστε να μπορεί να χρησιμοποιείται κατά την

εκτέλεση χωρίς να απαιτείται πρόσβαση στο TensorFlow ή σε κάποιον εξωτερικό διακομιστή.

Παρόλο που το μοντέλο που χρησιμοποιήθηκε δεν είναι ιδιαίτερα πολύπλοκο, αποδεικνύει στην πράξη τη διαδικασία ανάπτυξης και αξιοποίησης ενός συστήματος μηχανικής μάθησης μέσα σε μία εφαρμογή iOS.

4.4.3 Εκπαίδευση και ενσωμάτωση μοντέλων

Η ανάπτυξη ενός μοντέλου μηχανικής μάθησης περιλαμβάνει μία σειρά από διαδοχικά στάδια, τα οποία ξεκινούν από τη συλλογή των δεδομένων και καταλήγουν στην ενσωμάτωσή του στην τελική εφαρμογή. Η ποιότητα των αποτελεσμάτων εξαρτάται σε μεγάλο βαθμό τόσο από τα δεδομένα εκπαίδευσης όσο και από την επιλογή των χαρακτηριστικών που χρησιμοποιούνται κατά τη διαδικασία μάθησης.

Στην εφαρμογή CineMax δημιουργήθηκε ένα απλό μοντέλο προτάσεων, το οποίο δεν βασίζεται αποκλειστικά σε ιστορικό αξιολογήσεων, αλλά αξιοποιεί χαρακτηριστικά του χρήστη και των ταινιών.

Μετά την ολοκλήρωση της εκπαίδευσης στο TensorFlow, το μοντέλο μετατράπηκε σε μορφή CoreML, ώστε να μπορεί να χρησιμοποιηθεί απευθείας μέσα στην εφαρμογή. Κατά την εκτέλεση, η εφαρμογή προετοιμάζει τα απαραίτητα δεδομένα εισόδου, τα οποία αποστέλλονται στο μοντέλο. Στη συνέχεια, το μοντέλο επιστρέφει μία αριθμητική βαθμολογία που εκφράζει την πιθανότητα να ενδιαφέρει η συγκεκριμένη ταινία τον χρήστη. Η βαθμολογία αυτή χρησιμοποιείται για την ταξινόμηση των διαθέσιμων ταινιών, επιτρέποντας στην εφαρμογή να εμφανίζει πρώτα εκείνες που θεωρούνται περισσότερο σχετικές με το προφίλ και τις προτιμήσεις του χρήστη. Η διαδικασία πραγματοποιείται τοπικά στη συσκευή, εξασφαλίζοντας γρήγορη απόκριση και προστασία των προσωπικών δεδομένων.

Παρότι το συγκεκριμένο μοντέλο αποτελεί μία σχετικά απλή υλοποίηση ενός recommendation system, αποδεικνύει τον τρόπο με τον οποίο μπορεί να ενσωματωθεί η μηχανική μάθηση σε εφαρμογές κινητών συσκευών και να προσφέρει ουσιαστική βελτίωση στην εμπειρία χρήσης. Επιπλέον, η αρχιτεκτονική που ακολουθήθηκε επιτρέπει τη μελλοντική αντικατάσταση του μοντέλου με πιο σύνθετες εκδόσεις, χωρίς να απαιτούνται σημαντικές αλλαγές στον υπόλοιπο κώδικα της εφαρμογής.

5. Ανάλυση και Σχεδιασμός Συστήματος

Μετά την επιλογή των τεχνολογιών ανάπτυξης, ακολουθεί το στάδιο του σχεδιασμού της εφαρμογής, το οποίο αποτελεί τη γέφυρα μεταξύ των απαιτήσεων του συστήματος και της τελικής υλοποίησης. Ο σωστός σχεδιασμός συμβάλλει στη δημιουργία μιας οργανωμένης αρχιτεκτονικής, διευκολύνει τη συντήρηση του λογισμικού και εξασφαλίζει ότι όλες οι λειτουργικές απαιτήσεις καλύπτονται με αποτελεσματικό τρόπο.

Στο κεφάλαιο αυτό παρουσιάζεται η συνολική αρχιτεκτονική της εφαρμογής CineMax, καθώς και τα βασικά διαγράμματα που χρησιμοποιήθηκαν κατά τη φάση της σχεδίασης. Συγκεκριμένα, αναλύονται οι περιπτώσεις χρήσης (Use Cases), το διάγραμμα κλάσεων (Class Diagram), καθώς και ο σχεδιασμός της διεπαφής χρήστη μέσω των αρχών σχεδιασμού και των αρχικών προσχεδίων (wireframes).

Η εφαρμογή υλοποιήθηκε ακολουθώντας την αρχιτεκτονική MVVM (Model–View–ViewModel), η οποία διαχωρίζει τη λογική της εφαρμογής από τη διεπαφή χρήστη, βελτιώνοντας την οργάνωση του κώδικα και διευκολύνοντας τη μελλοντική συντήρηση και επέκταση του συστήματος. Επιπλέον, παρουσιάζονται οι σχέσεις μεταξύ των βασικών κλάσεων και ο τρόπος με τον οποίο αυτές συνεργάζονται για την υλοποίηση των λειτουργιών της εφαρμογής.

Μέσα από τα διαγράμματα και τις περιγραφές που ακολουθούν, γίνεται κατανοητή η δομή του συστήματος και η αλληλεπίδραση των επιμέρους υποσυστημάτων που συνθέτουν τη λειτουργία της εφαρμογής.

5.1 Απαιτήσεις συστήματος

Η ανάλυση απαιτήσεων αποτελεί ένα από τα σημαντικότερα στάδια κατά τον σχεδιασμό ενός πληροφοριακού συστήματος, καθώς καθορίζει τις λειτουργίες και τα χαρακτηριστικά που θα πρέπει να υποστηρίζει η εφαρμογή. Στην παρούσα εργασία, οι απαιτήσεις προέκυψαν με βάση τις ανάγκες των χρηστών για εύκολη αναζήτηση, οργάνωση και προσωποποιημένη διαχείριση κινηματογραφικού και τηλεοπτικού περιεχομένου.

Οι απαιτήσεις του συστήματος διακρίνονται σε λειτουργικές και μη λειτουργικές. Οι λειτουργικές απαιτήσεις περιγράφουν τις βασικές λειτουργίες που πρέπει να εκτελεί η εφαρμογή, ενώ οι μη λειτουργικές σχετίζονται με χαρακτηριστικά ποιότητας, όπως η απόδοση, η ασφάλεια και η ευχρηστία.

5.1.1 Λειτουργικές απαιτήσεις

Οι λειτουργικές απαιτήσεις καθορίζουν τις βασικές δυνατότητες και υπηρεσίες που προσφέρει η εφαρμογή στους χρήστες. Οι απαιτήσεις αυτές σχετίζονται με την αναζήτηση και προβολή περιεχομένου, τη διαχείριση λογαριασμών και αγαπημένων, καθώς και με τις λειτουργίες προσωποποιημένων προτάσεων.

A/A	Απαίτηση
1	Το σύστημα θα πρέπει να επιτρέπει στον χρήστη να πραγματοποιεί εγγραφή και σύνδεση χρησιμοποιώντας email και κωδικό πρόσβασης.
2	Το σύστημα θα πρέπει να υποστηρίζει πρόσβαση στους χρήστες.
3	Το σύστημα θα πρέπει να επιτρέπει την αναζήτηση ταινιών και τηλεοπτικών σειρών με βάση λέξεις-κλειδιά.

4	Το σύστημα θα πρέπει να παρέχει δυνατότητα φιλτραρίσματος περιεχομένου με βάση το είδος, το έτος κυκλοφορίας και τον τύπο περιεχομένου.
5	Το σύστημα θα πρέπει να εμφανίζει πληροφορίες για κάθε ταινία ή σειρά, όπως τίτλο, περίληψη, είδος, ημερομηνία κυκλοφορίας, βαθμολογία και εικόνα.
6	Το σύστημα θα πρέπει να επιτρέπει στον χρήστη να προσθέτει και να αφαιρεί περιεχόμενο από τη λίστα αγαπημένων.
7	Το σύστημα θα πρέπει να αποθηκεύει τις προτιμήσεις και τα αγαπημένα του χρήστη.
8	Το σύστημα θα πρέπει να υποστηρίζει προσωποποιημένες προτάσεις περιεχομένου με αξιοποίηση τεχνικών μηχανικής μάθησης.
9	Το σύστημα θα πρέπει να υποστηρίζει πλοήγηση μεταξύ των βασικών οθονών της εφαρμογής, όπως Αρχική, Αναζήτηση, Αγαπημένα και Προφίλ.
10	Το σύστημα θα πρέπει να ενημερώνει δυναμικά το περιεχόμενο της εφαρμογής μέσω εξωτερικών πηγών δεδομένων και APIs.
11	Το σύστημα θα πρέπει να επιτρέπει την αποσύνδεση του χρήστη.

Πίνακας 1 Λειτουργικές απαιτήσεις

5.1.2 Μη λειτουργικές απαιτήσεις

Οι μη λειτουργικές απαιτήσεις περιγράφουν τα ποιοτικά χαρακτηριστικά του συστήματος και επηρεάζουν τη συνολική εμπειρία χρήστη και την αξιοπιστία της εφαρμογής. Οι απαιτήσεις αυτές σχετίζονται με την απόδοση, την ασφάλεια, τη συμβατότητα και την επεκτασιμότητα της εφαρμογής.

A/A	Απαιτήσεις
1	Η εφαρμογή θα πρέπει να διαθέτει φιλικό και εύχρηστο περιβάλλον χρήστη, ακολουθώντας σύγχρονες αρχές σχεδιασμού UI/UX.
2	Η εφαρμογή θα πρέπει να παρουσιάζει χρόνο απόκρισης μικρότερο των 2 δευτερολέπτων στις βασικές λειτουργίες αναζήτησης και φόρτωσης

	δεδομένων.
3	Η εφαρμογή θα πρέπει να λειτουργεί ομαλά σε συσκευές με λειτουργικό σύστημα iOS.
4	Η εφαρμογή θα πρέπει να διασφαλίζει την προστασία των προσωπικών δεδομένων των χρηστών.
5	Η εφαρμογή θα πρέπει να είναι επεκτάσιμη, ώστε να μπορεί να υποστηρίξει μελλοντικές λειτουργίες και επιπλέον υπηρεσίες.
6	Η εφαρμογή θα πρέπει να διατηρεί σταθερή απόδοση ακόμη και σε περιπτώσεις μεγάλου όγκου δεδομένων.
7	Η διεπαφή χρήστη θα πρέπει να προσαρμόζεται σε διαφορετικά μεγέθη οθόνης κινητών συσκευών.
8	Η εφαρμογή θα πρέπει να ακολουθεί τις σχεδιαστικές οδηγίες της Apple για native iOS εφαρμογές.
9	Η εφαρμογή θα πρέπει να υποστηρίζει εύκολη συντήρηση και αναβάθμιση του κώδικα μέσω οργανωμένης αρχιτεκτονικής ανάπτυξης.

Πίνακας 2 Μη λειτουργικές απαιτήσεις

5.2 Περιπτώσεις Χρήσης

Οι περιπτώσεις χρήσης περιγράφουν τις βασικές λειτουργίες της εφαρμογής και τον τρόπο με τον οποίο ο χρήστης αλληλεπιδρά με το σύστημα. Η εφαρμογή απευθύνεται αποκλειστικά σε εγγεγραμμένους χρήστες, καθώς η πρόσβαση στις λειτουργίες της προϋποθέτει πιστοποίηση μέσω του συστήματος Firebase Authentication.

Μέσα από τις περιπτώσεις χρήσης παρουσιάζονται τα κύρια σενάρια λειτουργίας της εφαρμογής και οι ενέργειες που εκτελεί ο χρήστης για την αξιοποίηση των διαθέσιμων λειτουργιών.

Οι βασικές περιπτώσεις χρήσης της εφαρμογής παρουσιάζονται στις επόμενες υποενότητες.

5.2.1 Διαχείριση Λογαριασμού Χρήστη

Η περίπτωση χρήσης αφορά τη δημιουργία, τη σύνδεση και τη διαχείριση του λογαριασμού του χρήστη μέσω Firebase Authentication.

Ο λογαριασμός χρήστη αποτελεί βασική προϋπόθεση για την πρόσβαση στην εφαρμογή, καθώς όλες οι λειτουργίες είναι συνδεδεμένες με το προφίλ του χρήστη.

Βήματα:

- Ο χρήστης επιλέγει την εγγραφή ή τη σύνδεση στην εφαρμογή.
- Στην εγγραφή εισάγονται στοιχεία όπως email, κωδικός πρόσβασης και ετσι λαμβάνονται τα δημογραφικά δεδομένα (ηλικία, φύλο, χώρα, πόλη).
- Στη σύνδεση ο χρήστης εισάγει τα ήδη καταχωρημένα στοιχεία του.
- Το σύστημα πραγματοποιεί έλεγχο ταυτότητας μέσω Firebase Authentication.
- Σε περίπτωση επιτυχούς σύνδεσης, ο χρήστης αποκτά πρόσβαση στην εφαρμογή.
- Ο χρήστης μπορεί επίσης να αποσυνδεθεί ή να ενημερώσει το προφίλ του μέσω της οθόνης ProfileView.

5.2.2 Αναζήτηση Περιεχομένου

Η αναζήτηση περιεχομένου επιτρέπει στον χρήστη να εντοπίζει ταινίες και τηλεοπτικές σειρές με βάση λέξεις-κλειδιά.

Βήματα:

- Ο χρήστης μεταβαίνει στην οθόνη SearchView.
- Εισάγει λέξη-κλειδί στο πεδίο αναζήτησης.
- Το σύστημα αναζητά σχετικά αποτελέσματα από τη βάση δεδομένων ή το API περιεχομένου.
- Εμφανίζεται λίστα αποτελεσμάτων με βασικές πληροφορίες (τίτλος, εικόνα, βαθμολογία).
- Ο χρήστης επιλέγει ένα αποτέλεσμα για προβολή λεπτομερειών.

5.2.3 Φιλτράρισμα Περιεχομένου

Η λειτουργία φιλτραρίσματος επιτρέπει στον χρήστη να περιορίζει τα αποτελέσματα με βάση συγκεκριμένα κριτήρια.

Βήματα:

- Ο χρήστης μεταβαίνει στην οθόνη FilterView.
- Επιλέγει φίλτρα όπως είδος, έτος κυκλοφορίας ή άλλες παραμέτρους.
- Το σύστημα εφαρμόζει τα φίλτρα στα δεδομένα αναζήτησης.
- Τα αποτελέσματα ενημερώνονται δυναμικά.
- Ο χρήστης μπορεί να συνδυάζει πολλαπλά φίλτρα.

5.2.4 Προβολή Πληροφοριών Περιεχομένου

Η περίπτωση χρήσης αφορά την προβολή αναλυτικών πληροφοριών για μια ταινία ή σειρά μέσω της οθόνης MovieDetailView.

Βήματα:

- Ο χρήστης επιλέγει ένα περιεχόμενο από αναζήτηση ή φίλτρα.
- Το σύστημα ανοίγει την οθόνη λεπτομερειών.
- Εμφανίζονται πληροφορίες όπως τίτλος, περίληψη, βαθμολογία και poster.
- Ο χρήστης μπορεί να δει επιπλέον λεπτομέρειες και αξιολογήσεις.
- Από την οθόνη αυτή μπορεί να προσθέσει το περιεχόμενο σε αγαπημένα ή watchlist.

5.2.5 Διαχείριση Αγαπημένων και Watchlist

Η περίπτωση χρήσης αφορά την αποθήκευση και διαχείριση περιεχομένου που ενδιαφέρει τον χρήστη.

Βήματα:

- Ο χρήστης βρίσκεται στην οθόνη MovieDetailView.
- Επιλέγει προσθήκη σε Favorites ή Watchlist.
- Το σύστημα αποθηκεύει το περιεχόμενο στο Firebase, συνδεδεμένο με τον λογαριασμό του χρήστη.
- Ο χρήστης μεταβαίνει στις οθόνες FavoritesView ή WatchlistView.
- Προβάλλεται η λίστα αποθηκευμένου περιεχομένου.
- Ο χρήστης μπορεί να αφαιρέσει στοιχεία ή να ανοίξει ξανά τις λεπτομέρειες.

5.3 Βασικές κλάσεις

MovieDataManager

Η κλάση MovieDataManager αποτελεί τον κεντρικό μηχανισμό διαχείρισης των δεδομένων που σχετίζονται με την προσωπική δραστηριότητα του χρήστη στην εφαρμογή. Υλοποιείται ως Singleton, ώστε να υπάρχει ένα μοναδικό σημείο πρόσβασης στα δεδομένα από όλα τα τμήματα της εφαρμογής.

Η κλάση διαχειρίζεται τρεις βασικές κατηγορίες περιεχομένου:

- Τις αγαπημένες ταινίες (Favorites)
- Τη λίστα παρακολούθησης (Watchlist)
- Το περιεχόμενο που έχει ήδη παρακολουθήσει ο χρήστης (Watched Movies)

```
import Foundation
import Combine

class MovieDataManager: ObservableObject {
    static let shared = MovieDataManager()

    @Published var favoriteMovies: [Movie] = []
    @Published var watchlistMovies: [Movie] = []
    @Published var watchedMovies: [Movie] = []
    @Published var watchedMovieIDs: Set<Int> = []

    private let favoritesKey = "FavoriteMovies"
    private let watchlistKey = "WatchlistMovies"
    private let watchedMoviesKey = "WatchedMovieIDs"
    private let watchedMoviesDataKey = "WatchedMoviesData"

    private init() {
        loadFavorites()
        loadWatchlist()
        loadWatchedMovies()
    }

    // MARK: - Favorites Management

    func addToFavorites(_ movie: Movie) {
        if !favoriteMovies.contains(where: { $0.id == movie.id }) {
            favoriteMovies.append(movie)
            saveFavorites()
        }
    }

    func removeFromFavorites(_ movie: Movie) {
        favoriteMovies.removeAll { $0.id == movie.id }
        saveFavorites()
    }

    func isFavorite(_ movie: Movie) -> Bool {
        favoriteMovies.contains { $0.id == movie.id }
    }
}
```

Εικόνα 1 Κλάση MovieDataManager

AuthManager

Η κλάση AuthManager είναι υπεύθυνη για τη διαχείριση της αυθεντικοποίησης των χρηστών και της κατάστασης σύνδεσης στην εφαρμογή.

Η υλοποίηση βασίζεται στις υπηρεσίες σύνδεσης της Google μέσω του Firebase Authentication, επιτρέποντας στους χρήστες να συνδέονται εύκολα χρησιμοποιώντας τον προσωπικό τους λογαριασμό Google.

Η κλάση αποθηκεύει πληροφορίες σχετικές με τον συνδεδεμένο χρήστη, όπως:

- Την κατάσταση σύνδεσης (Logged In / Logged Out)
- Το όνομα χρήστη
- Την εικόνα προφίλ

Παράλληλα, υποστηρίζει τη διαδικασία επαναφοράς προηγούμενης σύνδεσης κατά την εκκίνηση της εφαρμογής, ώστε ο χρήστης να μην χρειάζεται να πραγματοποιεί εκ νέου σύνδεση κάθε φορά που ανοίγει την εφαρμογή.

```
import Foundation
import Combine
import GoogleSignIn

class AuthManager: ObservableObject {
    static let shared = AuthManager()

    @Published var isLoggedIn: Bool = false
    @Published var userName: String = ""
    @Published var userPhotoURL: URL? = nil

    private init() {}

    /// Attempt to restore a previous Google session silently (call on app launch).
    func restorePreviousSignIn() {
        GIDSignIn.sharedInstance.restorePreviousSignIn { [weak self] user, error in
            guard let user, error == nil else { return }
            DispatchQueue.main.async {
                self?.signIn(
                    name: user.profile?.name ?? "",
                    photoURL: user.profile?.imageURL(withDimension: 200)
                )
            }
        }
    }

    /// Call this with real Google user data after GIDSignIn succeeds.
    func signIn(name: String, photoURL: URL?) {
        userName = name
        userPhotoURL = photoURL
        isLoggedIn = true
    }
}
```

Εικόνα 2 Κλάση AuthManager

ContentViewModel

Η κλάση ContentViewModel είναι υπεύθυνη για τη διαχείριση της κεντρικής πλοήγησης της εφαρμογής. Διατηρεί πληροφορίες σχετικά με τις διαθέσιμες καρτέλες (tabs) και την τρέχουσα επιλεγμένη καρτέλα, επιτρέποντας την ομαλή μετάβαση μεταξύ των διαφορετικών οθονών της εφαρμογής.

```
import Foundation
import SwiftUI

@MainActor
class ContentViewModel: ObservableObject {
    @Published var model: ContentModel = .default

    var tabs: [TabItemModel] { model.tabs }

    var selectedTab: Int {
        get { model.selectedTab }
        set { model.selectedTab = newValue }
    }

    func select(tabID: Int) {
        model.selectedTab = tabID
    }
}
```

Εικόνα 3 Κλάση ContentViewModel

FavoritesViewModel

Η κλάση FavoritesViewModel διαχειρίζεται τη λειτουργία των αγαπημένων ταινιών του χρήστη. Συνδέεται με τον μηχανισμό αποθήκευσης δεδομένων και παρακολουθεί δυναμικά τις αλλαγές στη λίστα αγαπημένων. Παρέχει επίσης λειτουργίες διαγραφής μεμονωμένων ταινιών ή εκκαθάρισης ολόκληρης της λίστας.

```
import Foundation
import Combine
import SwiftUI

@MainActor
class FavoritesViewModel: ObservableObject {
    @Published var model: FavoritesModel = .empty

    private let dataManager: MovieDataManager
    private var cancellables = Set<AnyCancellable>()

    var movies: [Movie] { model.movies }
    var isEmpty: Bool { model.isEmpty }
    var count: Int { model.count }

    init(dataManager: MovieDataManager = .shared) {
        self.dataManager = dataManager
        self.model = FavoritesModel(movies: dataManager.favoriteMovies)

        // Keep the model in sync with the data manager
        dataManager.$favoriteMovies
            .receive(on: RunLoop.main)
            .sink { [weak self] updated in
                self?.model.movies = updated
            }
            .store(in: &cancellables)
    }

    func remove(_ movie: Movie) {
        dataManager.removeFromFavorites(movie)
    }

    func clearAll() {
        dataManager.clearAllFavorites()
    }
}
```

Εικόνα 4 Κλάση FavoritesViewModel

CineMaxRecommenderInput

Η κλάση CineMaxRecommenderInput χρησιμοποιείται για την προετοιμασία των δεδομένων εισόδου του συστήματος συστάσεων. Μετατρέπει τα χαρακτηριστικά μιας ταινίας, όπως τα είδη, τη δημοτικότητα και τη βαθμολογία, σε μορφή κατάλληλη για επεξεργασία από το μοντέλο μηχανικής μάθησης.

```
final class CineMaxRecommenderInput: MLFeatureProvider {

    /// Preferred genre IDs encoded as a multi-hot vector (20 TMDb buckets).
    var genreVector: MLMultiArray

    /// Normalised popularity score of the candidate movie (0 – 1).
    var popularity: Double

    /// Normalised vote average of the candidate movie (0 – 1).
    var voteAverage: Double

    /// Number of matching genres between candidate and user profile.
    var genreOverlap: Double

    var featureNames: Set<String> {
        ["genreVector", "popularity", "voteAverage", "genreOverlap"]
    }

    func featureValue(for featureName: String) -> MLFeatureValue? {
        switch featureName {
        case "genreVector": return MLFeatureValue(multiArray: genreVector)
        case "popularity": return MLFeatureValue(double: popularity)
        case "voteAverage": return MLFeatureValue(double: voteAverage)
        case "genreOverlap": return MLFeatureValue(double: genreOverlap)
        default: return nil
        }
    }

    init(genreVector: MLMultiArray,
         popularity: Double,
         voteAverage: Double,
         genreOverlap: Double) {
        self.genreVector = genreVector
        self.popularity = popularity
        self.voteAverage = voteAverage
        self.genreOverlap = genreOverlap
    }
}
```

Εικόνα 5 Κλάση CineMaxRecommenderInput

CineMaxRecommenderOutput

Η κλάση CineMaxRecommenderOutput αναπαριστά το αποτέλεσμα που παράγεται από το μοντέλο συστάσεων. Το αποτέλεσμα αυτό είναι μία αριθμητική βαθμολογία η οποία εκφράζει τον βαθμό καταλληλότητας μιας ταινίας για τον συγκεκριμένο χρήστη.

```
final class CineMaxRecommenderOutput: MLFeatureProvider {  
  
    /// Predicted affinity score for this movie.  
    let score: Double  
  
    var featureNames: Set<String> { ["score"] }  
  
    func featureValue(for featureName: String) -> MLFeatureValue? {  
        featureName == "score" ? MLFeatureValue(double: score) : nil  
    }  
  
    init(score: Double) { self.score = score }  
}
```

Εικόνα 6 Κλάση CineMaxRecommenderOutput

CineMaxRecommender

Η κλάση CineMaxRecommender υλοποιεί τον βασικό μηχανισμό προτάσεων της εφαρμογής. Χρησιμοποιεί μοντέλο CoreML για την αξιολόγηση των διαθέσιμων ταινιών και την παραγωγή εξατομικευμένων προτάσεων. Σε περίπτωση που το εκπαιδευμένο μοντέλο δεν είναι διαθέσιμο, εφαρμόζει έναν εναλλακτικό αλγόριθμο βαθμολόγησης βασισμένο στα χαρακτηριστικά των ταινιών και στις προτιμήσεις του χρήστη.

```
final class CineMaxRecommender {  
  
    // — Singleton —  
    static let shared = CineMaxRecommender()  
  
    // — Internal state —  
    private let model: MLModel?
```

Εικόνα 7 Κλάση CineMaxRecommender

HomeViewModel

Η κλάση HomeViewModel είναι υπεύθυνη για την ανάκτηση και παρουσίαση των δεδομένων που εμφανίζονται στην αρχική οθόνη της εφαρμογής. Συλλέγει ταινίες από διαφορετικές κατηγορίες, όπως δημοφιλείς, κορυφαίες σε βαθμολογία, επερχόμενες και ταινίες που προβάλλονται αυτή τη στιγμή.

```
import Foundation
import SwiftUI

@MainActor
class HomeViewModel: ObservableObject {
    @Published var model: HomeModel = .empty

    var popularMovies: [Movie] { model.popularMovies }
    var nowPlayingMovies: [Movie] { model.nowPlayingMovies }
    var topRatedMovies: [Movie] { model.topRatedMovies }
    var upcomingMovies: [Movie] { model.upcomingMovies }
    var isLoading: Bool { model.isLoading }
    var sections: [HomeSection] { model.sections }

    private let service: MovieService

    init(service: MovieService = .shared) {
        self.service = service
    }

    func loadMovies() async {
        guard model.popularMovies.isEmpty else { return }
        await fetchMovies()
    }

    func refreshMovies() async {
        await fetchMovies()
    }

    private func fetchMovies() async {
        model.isLoading = true

        async let popular = try? service.fetchPopularMovies()
        async let nowPlaying = try? service.fetchNowPlayingMovies()
        async let topRated = try? service.fetchTopRatedMovies()
        async let upcoming = try? service.fetchUpcomingMovies()
    }
}
```

Εικόνα 8 Κλάση HomeViewModel

MovieDetailViewModel

Η κλάση MovieDetailViewModel διαχειρίζεται τα δεδομένα της οθόνης λεπτομερειών μιας ταινίας. Ανακτά αναλυτικές πληροφορίες από το TMDb API και επιτρέπει στον χρήστη να προσθέτει ή να αφαιρεί ταινίες από τα αγαπημένα, τη watchlist ή τη λίστα προβεβλημένων ταινιών.

```
import Foundation
import Combine
import SwiftUI

@MainActor
class MovieDetailViewModel: ObservableObject {
    @Published var model: MovieDetailModel

    private let dataManager: MovieDataManager
    private let service: MovieService
    private var cancellables = Set<AnyCancellable>()

    var movie: Movie { model.movie }
    var isFavorite: Bool { model.isFavorite }
    var isInWatchlist: Bool { model.isInWatchlist }
    var isWatched: Bool { model.isWatched }
    var isLoadingDetail: Bool { model.isLoadingDetail }
    var detail: MovieDetail? { model.detail }

    init(movie: Movie,
         dataManager: MovieDataManager = .shared,
         service: MovieService = .shared) {
        self.dataManager = dataManager
        self.service = service
        self.model = MovieDetailModel(
            movie: movie,
            isFavorite: dataManager.isFavorite(movie),
            isInWatchlist: dataManager.isInWatchlist(movie),
            isWatched: dataManager.isWatched(movie)
        )

        dataManager.$favoriteMovies
            .receive(on: RunLoop.main)
            .sink { [weak self] favorites in
                guard let self else { return }
                self.model.isFavorite = favorites.contains { $0.id == self.model.movie.id }
            }
            .store(in: &cancellables)
    }
}
```

Εικόνα 9 Κλάση MovieDetailViewModel

ProfileViewModel

Η κλάση ProfileViewModel είναι υπεύθυνη για τη διαχείριση των πληροφοριών προφίλ του χρήστη. Εμφανίζει στοιχεία λογαριασμού που προέρχονται από τη διαδικασία αυθεντικοποίησης καθώς και στατιστικά χρήσης, όπως ο αριθμός αγαπημένων ταινιών, ταινιών στη watchlist και ταινιών που έχουν ήδη παρακολουθηθεί.

```
import Foundation
import Combine
import SwiftUI

@MainActor
class ProfileViewModel: ObservableObject {
    @Published var model: ProfileModel = .default

    private let dataManager: MovieDataManager
    private let auth = AuthManager.shared
    private var cancellables = Set<AnyCancellable>()

    var userName: String { model.userName }
    var userInitials: String { model.userInitials }
    var userPhotoURL: URL? { model.userPhotoURL }
    var watchedCount: Int { model.watchedCount }
    var favoritesCount: Int { model.favoritesCount }
    var watchlistCount: Int { model.watchlistCount }
    var favoritesSubtitle: String { model.favoritesSubtitle }
    var watchlistSubtitle: String { model.watchlistSubtitle }
    var watchedSubtitle: String { model.watchedSubtitle }
    var appVersion: String { model.appVersion }
    var appCredit: String { model.appCredit }

    init(dataManager: MovieDataManager = .shared) {
        self.dataManager = dataManager
        refreshStats()
    }
}
```

Εικόνα 10 Κλάση ProfileViewModel

SearchViewModel

Η κλάση SearchViewModel υλοποιεί τον βασικό μηχανισμό αναζήτησης της εφαρμογής. Διαχειρίζεται την αναζήτηση ταινιών, την εφαρμογή φίλτρων και την ταξινόμηση των αποτελεσμάτων. Παρέχει επίσης τη δυνατότητα αναζήτησης βάσει λέξεων-κλειδιών και πολλαπλών κριτηρίων.

```
import Foundation
import Combine
import SwiftUI

@MainActor
class SearchViewModel: ObservableObject {
    @Published var model: SearchModel = .empty

    // Convenience bindings for the View
    var searchText: String {
        get { model.searchText }
        set { model.searchText = newValue }
    }

    var movies: [Movie] { model.movies }
    var genres: [Genre] { model.genres }
    var selectedGenres: Set<Int> {
        get { model.selectedGenres }
        set { model.selectedGenres = newValue }
    }
    var selectedYear: String {
        get { model.selectedYear }
        set { model.selectedYear = newValue }
    }
    var sortBy: SortOption {
        get { model.sortBy }
        set { model.sortBy = newValue }
    }
    var isLoading: Bool { model.isLoading }
    var activeFilters: [String] { model.activeFilters }

    private let service: MovieService
    private var cancellables = Set<AnyCancellable>()

    init(service: MovieService = .shared) {
```

Εικόνα 11 Κλάση SearchViewModel

SettingsViewModel

Η κλάση SettingsViewModel διαχειρίζεται τις ρυθμίσεις της εφαρμογής. Περιλαμβάνει επιλογές που σχετίζονται με την ποιότητα λήψεων, τη γλώσσα εμφάνισης και διάφορες λειτουργίες λογαριασμού και υποστήριξης χρηστών.

```
import Foundation
import SwiftUI

@MainActor
class SettingsViewModel: ObservableObject {
    @Published var model: SettingsModel = .default

    var downloadQuality: String {
        get { model.downloadQuality }
        set { model.downloadQuality = newValue }
    }

    var language: String {
        get { model.language }
        set { model.language = newValue }
    }

    var downloadQualities: [String] { model.downloadQualities }
    var languages: [String] { model.languages }
    var versionDisplay: String { model.versionDisplay }

    // MARK: - Account Actions (stubbed for UI navigation)
    func editProfile() {}
    func changePassword() {}
    func syncData() {}

    // MARK: - Support Actions
    func openHelpCenter() {}
    func contactUs() {}
    func rateApp() {}

    // MARK: - About Actions
    func openPrivacyPolicy() {}
    func openTermsOfService() {}

    // MARK: - Account Actions
    func signOut() {}
    func deleteAccount() {}
}
```

Εικόνα 12 Κλάση SettingsViewModel

WatchlistViewModel

Η κλάση WatchlistViewModel διαχειρίζεται τη λίστα ταινιών που ο χρήστης επιθυμεί να παρακολουθήσει στο μέλλον. Παρακολουθεί τις αλλαγές στη watchlist, επιτρέπει την αφαίρεση ταινιών και παρέχει δυνατότητα σήμανσης μιας ταινίας ως παρακολουθημένης.

```
import Foundation
import Combine
import SwiftUI

@MainActor
class WatchlistViewModel: ObservableObject {
    @Published var model: WatchlistModel = .empty

    private let dataManager: MovieDataManager
    private var cancellables = Set<AnyCancellable>()

    var movies: [Movie] { model.movies }
    var isEmpty: Bool { model.isEmpty }
    var count: Int { model.count }

    init(dataManager: MovieDataManager = .shared) {
        self.dataManager = dataManager
        self.model = WatchlistModel(
            movies: dataManager.watchlistMovies,
            watchedIDs: dataManager.watchedMovieIDs
        )

        dataManager.$watchlistMovies
            .receive(on: RunLoop.main)
            .sink { [weak self] updated in
                self?.model.movies = updated
            }
            .store(in: &cancellables)

        dataManager.$watchedMovieIDs
            .receive(on: RunLoop.main)
            .sink { [weak self] ids in
                self?.model.watchedIDs = ids
            }
            .store(in: &cancellables)
    }

    func isWatched(_ movie: Movie) -> Bool {
        model.isWatched(movie)
    }
}
```

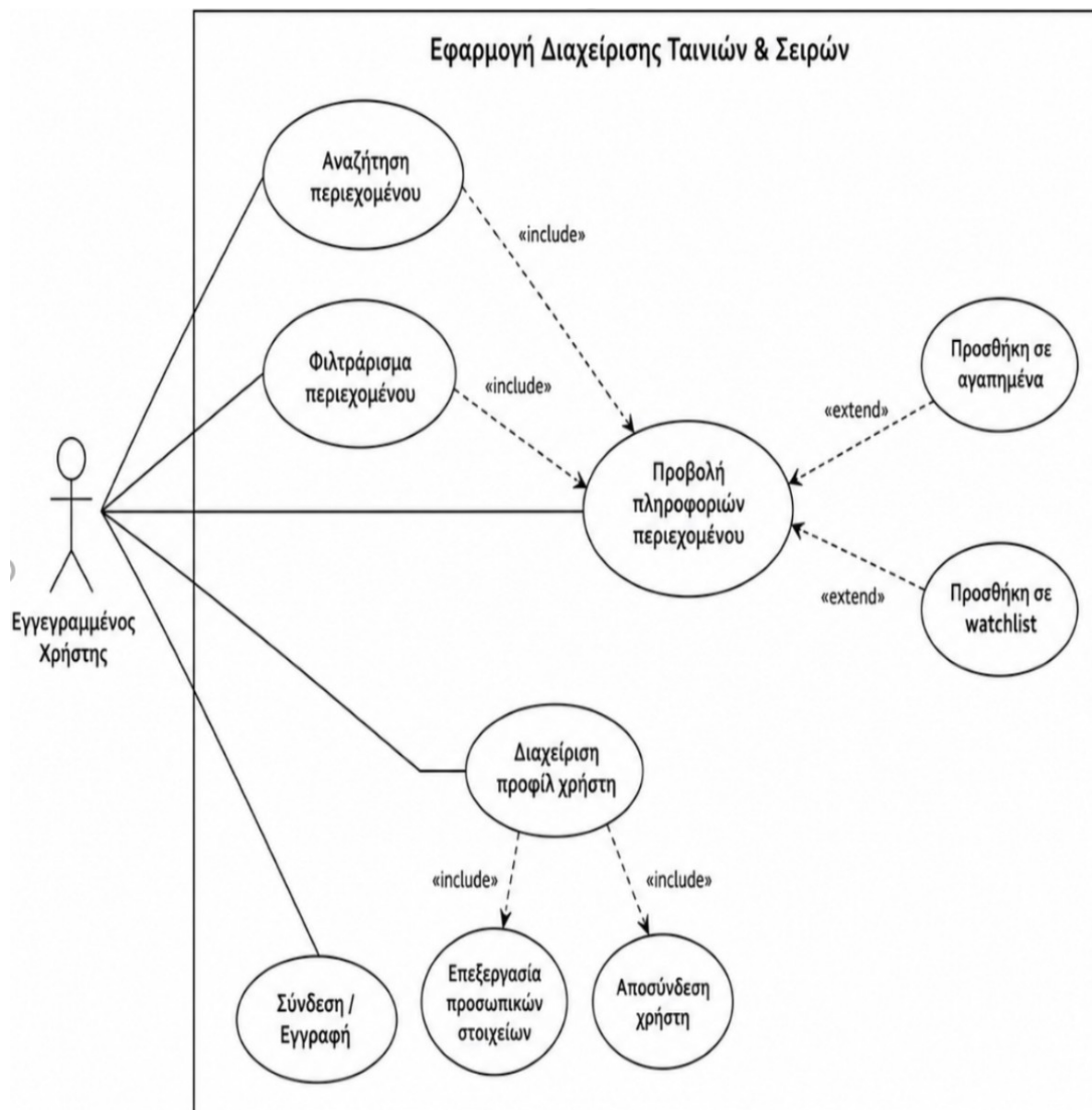
Εικόνα 13 Κλάση WatchlistViewModel

5.4 Διαγράμματα συστήματος

5.4.1 Διάγραμμα Περιπτώσεων Χρήσης

Το διάγραμμα περιπτώσεων χρήσης απεικονίζει τη βασική αλληλεπίδραση του χρήστη με την εφαρμογή και παρουσιάζει τις κύριες λειτουργίες που υποστηρίζονται από το σύστημα. Ο βασικός actor του συστήματος είναι ο Εγγεγραμμένος Χρήστης, ο οποίος έχει πρόσβαση σε όλες τις λειτουργίες της εφαρμογής.

Το σύστημα παρέχει δυνατότητες αναζήτησης και φιλτραρίσματος περιεχομένου, προβολής αναλυτικών πληροφοριών για ταινίες και σειρές, καθώς και διαχείρισης προσωπικών λιστών αγαπημένων και watchlist. Επιπλέον, υποστηρίζεται η διαχείριση λογαριασμού χρήστη μέσω Firebase Authentication.

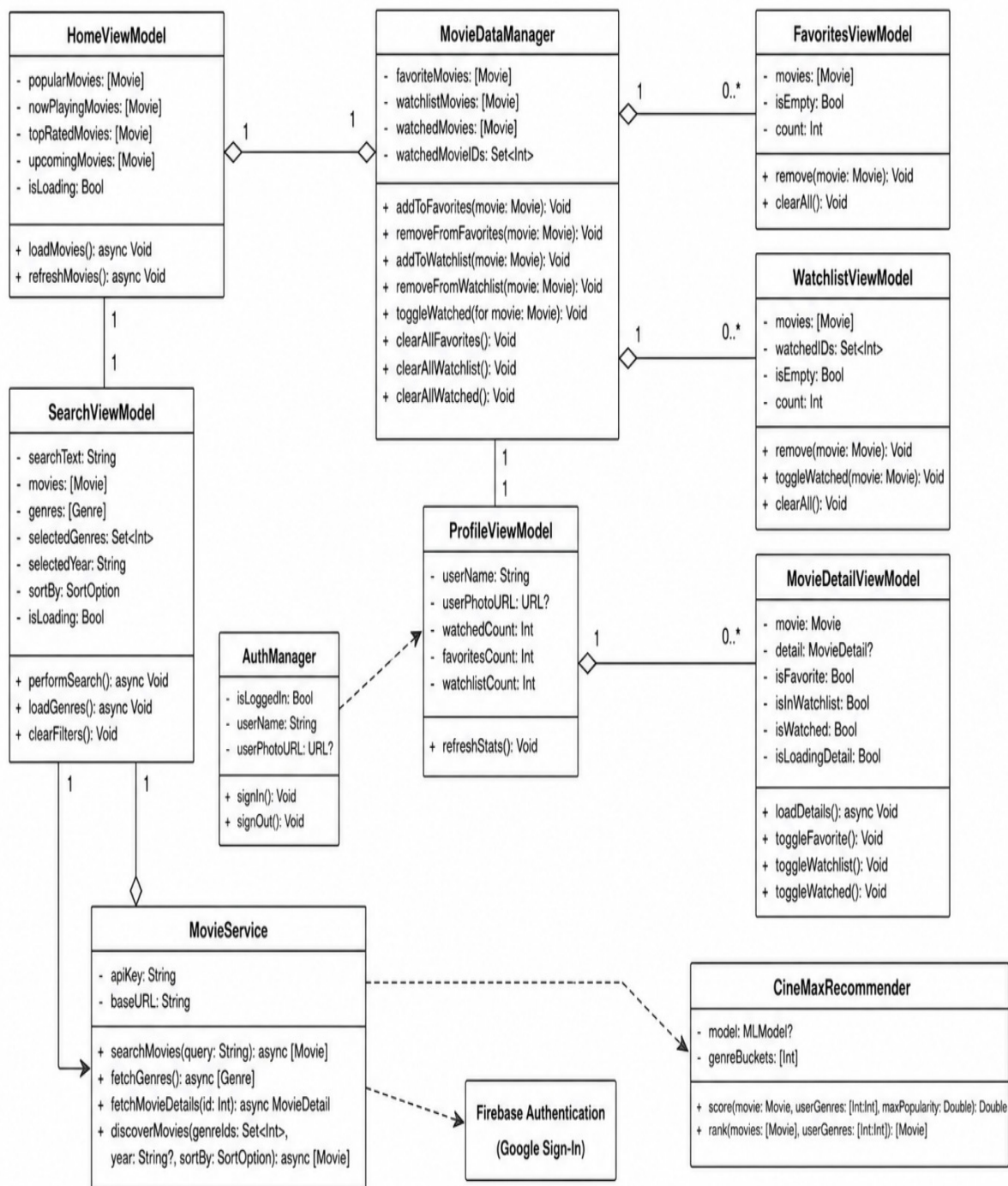


Εικόνα 14 Διάγραμμα περιπτώσεων χρήσης του CineMax

5.4.2 Διάγραμμα κλάσεων

Το διάγραμμα κλάσεων παρουσιάζει τη δομή του συστήματος της εφαρμογής CineMax και τις σχέσεις μεταξύ των βασικών κλάσεων που χρησιμοποιούνται για την υλοποίησή της. Μέσω του διαγράμματος αποτυπώνονται οι κύριες λειτουργικές μονάδες της εφαρμογής, όπως η διαχείριση ταινιών, η αναζήτηση και το φιλτράρισμα περιεχομένου, η διαχείριση αγαπημένων και λιστών παρακολούθησης, η αυθεντικοποίηση χρηστών, καθώς και ο μηχανισμός παροχής προτάσεων. Επιπλέον, παρουσιάζονται οι εξαρτήσεις μεταξύ των ViewModels και των κλάσεων υπηρεσιών (Services), οι οποίες συνεργάζονται για την

ανάκτηση, αποθήκευση και επεξεργασία δεδομένων. Το διάγραμμα συμβάλλει στην καλύτερη κατανόηση της αρχιτεκτονικής της εφαρμογής και του τρόπου αλληλεπίδρασης των επιμέρους τμημάτων του συστήματος.



Εικόνα 15 Διάγραμμα κλάσεων του CineMax

5.5 Σχεδιασμός διεπαφής χρήστη

Ο σχεδιασμός της διεπαφής χρήστη αποτελεί ένα από τα σημαντικότερα στάδια ανάπτυξης μιας εφαρμογής κινητών συσκευών, καθώς επηρεάζει άμεσα την εμπειρία του χρήστη και την ευκολία αλληλεπίδρασης με το σύστημα. Μια σωστά σχεδιασμένη διεπαφή επιτρέπει στους χρήστες να εντοπίζουν εύκολα τις διαθέσιμες λειτουργίες, να ολοκληρώνουν τις ενέργειές τους χωρίς δυσκολία και να χρησιμοποιούν την εφαρμογή με φυσικό και κατανοητό τρόπο.

Κατά τον σχεδιασμό της εφαρμογής CineMax δόθηκε ιδιαίτερη έμφαση στη δημιουργία μιας σύγχρονης, απλής και εύχρηστης διεπαφής, ακολουθώντας τις σχεδιαστικές οδηγίες της Apple για εφαρμογές iOS. Όλες οι οθόνες σχεδιάστηκαν με γνώμονα τη συνέπεια ως προς τη διάταξη των στοιχείων, τη χρωματική παλέτα και τον τρόπο πλοήγησης, ώστε ο χρήστης να αποκτά μια ενιαία εμπειρία χρήσης σε ολόκληρη την εφαρμογή.

Η εφαρμογή οργανώνεται σε διακριτές λειτουργικές ενότητες, όπως η αρχική σελίδα, η αναζήτηση, τα αγαπημένα, η λίστα παρακολούθησης, το προφίλ και οι ρυθμίσεις. Η μετάβαση μεταξύ των οθονών πραγματοποιείται μέσω της κάτω γραμμής πλοήγησης (Tab Bar), η οποία αποτελεί τη βασική μέθοδο πλοήγησης της εφαρμογής. Παράλληλα, κάθε οθόνη ακολουθεί κοινή σχεδιαστική φιλοσοφία, με στόχο την ομοιομορφία και τη μείωση του χρόνου εξοικείωσης των χρηστών.

Ιδιαίτερη προσοχή δόθηκε επίσης στη σωστή οργάνωση του περιεχομένου. Οι πληροφορίες προβάλλονται ιεραρχικά, με τρόπο που επιτρέπει στον χρήστη να εντοπίζει άμεσα τις σημαντικότερες λειτουργίες χωρίς να απαιτείται πολύπλοκη πλοήγηση. Επιπλέον, χρησιμοποιούνται ευανάγνωστες γραμματοσειρές, κατάλληλες αποστάσεις μεταξύ των στοιχείων και μεγάλα κουμπιά αλληλεπίδρασης, ώστε η εφαρμογή να παραμένει εύχρηστη ακόμη και σε μικρότερες οθόνες.

Ο συνολικός σχεδιασμός της διεπαφής επιδιώκει να προσφέρει μια ευχάριστη εμπειρία χρήσης, μειώνοντας τον αριθμό των απαραίτητων ενεργειών για την ολοκλήρωση των βασικών λειτουργιών, όπως η αναζήτηση μιας ταινίας, η προβολή πληροφοριών και η αποθήκευσή της στις λίστες αγαπημένων ή παρακολούθησης.

5.5.1 Αρχές σχεδιασμού

Η σχεδίαση της διεπαφής της εφαρμογής βασίστηκε σε θεμελιώδεις αρχές σχεδιασμού διεπαφών χρήστη, με στόχο τη δημιουργία ενός λειτουργικού, εύχρηστου και αισθητικά ευχάριστου περιβάλλοντος αλληλεπίδρασης.

Αρχικά εφαρμόστηκε η αρχή της απλότητας (Simplicity), σύμφωνα με την οποία κάθε οθόνη περιλαμβάνει μόνο τα απαραίτητα στοιχεία για την εκτέλεση της αντίστοιχης λειτουργίας. Με τον τρόπο αυτό αποφεύγεται η υπερφόρτωση πληροφοριών και μειώνεται η γνωστική επιβάρυνση του χρήστη.

Παράλληλα, υιοθετήθηκε η αρχή της συνέπειας (Consistency). Όλες οι οθόνες χρησιμοποιούν κοινά χρώματα, εικονίδια, κουμπιά και τυποποίηση στη διάταξη των στοιχείων. Η ομοιομορφία αυτή επιτρέπει στους χρήστες να εξοικειώνονται γρήγορα με τη λειτουργία της εφαρμογής, καθώς παρόμοιες ενέργειες εκτελούνται με τον ίδιο τρόπο σε κάθε σημείο της.

Επιπλέον, ιδιαίτερη σημασία δόθηκε στην ευκολία πλοήγησης (Navigation). Η χρήση της μόνιμης γραμμής πλοήγησης στο κάτω μέρος της οθόνης επιτρέπει τη γρήγορη μετάβαση

στις βασικές λειτουργίες της εφαρμογής, χωρίς ο χρήστης να χρειάζεται να επιστρέφει συνεχώς στην αρχική οθόνη.

Η διεπαφή σχεδιάστηκε επίσης με βάση την αρχή της ορατότητας των λειτουργιών (Visibility). Οι σημαντικότερες επιλογές, όπως η προσθήκη στα αγαπημένα, η αποθήκευση στη Watchlist ή η επισημάνση μιας ταινίας ως παρακολουθημένης, εμφανίζονται άμεσα στην οθόνη λεπτομερειών της ταινίας, διευκολύνοντας την πρόσβαση στις βασικές λειτουργίες.

Τέλος, εφαρμόστηκε η αρχή της ανατροφοδότησης (Feedback), σύμφωνα με την οποία κάθε ενέργεια του χρήστη συνοδεύεται από άμεση ενημέρωση της διεπαφής. Για παράδειγμα, όταν μια ταινία προστεθεί στα αγαπημένα ή στη λίστα παρακολούθησης, η εφαρμογή ενημερώνει άμεσα την αντίστοιχη κατάσταση χωρίς να απαιτείται ανανέωση της οθόνης. Με τον τρόπο αυτό ο χρήστης γνωρίζει πάντοτε ότι η ενέργειά του ολοκληρώθηκε επιτυχώς. Οι παραπάνω αρχές συνέβαλαν στη δημιουργία μιας διεπαφής που χαρακτηρίζεται από ευχρηστία, συνέπεια και αποτελεσματικότητα, προσφέροντας μια ποιοτική εμπειρία χρήσης.

5.5.2 Προσχέδια διεπαφής (Wireframes)

Πριν από την υλοποίηση της εφαρμογής πραγματοποιήθηκε ο αρχικός σχεδιασμός των βασικών οθονών μέσω προσχεδίων διεπαφής (wireframes). Τα προσχέδια χρησιμοποιήθηκαν για την οργάνωση της διάταξης των στοιχείων, τον καθορισμό της ροής πλοήγησης και την αξιολόγηση της εμπειρίας χρήστη πριν την έναρξη της υλοποίησης.

Κατά τη διαδικασία αυτή σχεδιάστηκαν οι κύριες οθόνες της εφαρμογής, όπως η αρχική σελίδα, η οθόνη αναζήτησης, η προβολή λεπτομερειών μιας ταινίας, η λίστα αγαπημένων, η λίστα παρακολούθησης, το προφίλ χρήστη και οι ρυθμίσεις. Για κάθε οθόνη προσδιορίστηκαν τα βασικά στοιχεία διεπαφής, οι περιοχές εμφάνισης πληροφοριών και οι διαθέσιμες ενέργειες του χρήστη.

Τα wireframes λειτούργησαν ως οδηγός για τη μετέπειτα ανάπτυξη της εφαρμογής σε SwiftUI, επιτρέποντας την έγκαιρη αναγνώριση πιθανών προβλημάτων στη διάταξη των στοιχείων και στη ροή πλοήγησης. Παράλληλα, συνέβαλαν στη διατήρηση μιας ενιαίας σχεδιαστικής φιλοσοφίας σε όλες τις οθόνες της εφαρμογής, γεγονός που διευκόλυνε τόσο την ανάπτυξη όσο και τη μελλοντική συντήρηση του έργου.

Στη συνέχεια της εργασίας παρουσιάζονται οι τελικές οθόνες της εφαρμογής, όπως υλοποιήθηκαν κατά την ανάπτυξη, καθώς και η λειτουργικότητα που προσφέρει κάθε μία από αυτές.

6. Υλοποίηση της Εφαρμογής

Στο κεφάλαιο αυτό παρουσιάζεται η διαδικασία υλοποίησης της εφαρμογής, καθώς και οι βασικές τεχνολογικές επιλογές που πραγματοποιήθηκαν κατά την ανάπτυξή της. Η εφαρμογή σχεδιάστηκε ως native εφαρμογή για το λειτουργικό σύστημα iOS, αξιοποιώντας σύγχρονες τεχνολογίες και αρχιτεκτονικές προσεγγίσεις με στόχο την υψηλή απόδοση, την ευχρηστία και την επεκτασιμότητα.

Η ανάπτυξη πραγματοποιήθηκε με χρήση της γλώσσας Swift και του πλαισίου SwiftUI, ενώ για τη διαχείριση δεδομένων και την αυθεντικοποίηση των χρηστών χρησιμοποιήθηκε η πλατφόρμα Firebase. Παράλληλα, ενσωματώθηκε μηχανισμός προσωποποιημένων προτάσεων μέσω τεχνολογιών μηχανικής μάθησης, αξιοποιώντας TensorFlow και CoreML.

Στις επόμενες ενότητες παρουσιάζονται αναλυτικά η αρχιτεκτονική της εφαρμογής, οι βασικές λειτουργίες που υλοποιήθηκαν, η ενσωμάτωση μηχανικής μάθησης και ο σχεδιασμός της διεπαφής χρήστη.

6.1 Αρχιτεκτονική εφαρμογής

Η αρχιτεκτονική μιας εφαρμογής αποτελεί βασικό στοιχείο για τη σωστή οργάνωση του κώδικα, τη συντηρησιμότητα και τη δυνατότητα μελλοντικής επέκτασης. Στην παρούσα εργασία επιλέχθηκε η αρχιτεκτονική MVVM (Model – View – ViewModel), η οποία χρησιμοποιείται ευρέως σε εφαρμογές iOS που αναπτύσσονται με SwiftUI.

Η συγκεκριμένη προσέγγιση διαχωρίζει τη λογική της εφαρμογής από τη διεπαφή χρήστη, επιτρέποντας καλύτερη διαχείριση των δεδομένων και πιο οργανωμένη δομή έργου. Παράλληλα, διευκολύνει τη συντήρηση του κώδικα και την προσθήκη νέων λειτουργιών.

6.1.1 Πρότυπο αρχιτεκτονικής MVVM

Το πρότυπο MVVM βασίζεται στον διαχωρισμό της εφαρμογής σε τρία βασικά επίπεδα: το Model, το View και το ViewModel.

Το επίπεδο Model περιλαμβάνει τις δομές δεδομένων και τη βασική επιχειρησιακή λογική της εφαρμογής. Στην παρούσα εργασία, τα μοντέλα χρησιμοποιούνται για την αναπαράσταση ταινιών, τηλεοπτικών σειρών, χρηστών και λιστών περιεχομένου.

Το επίπεδο View αφορά τη διεπαφή χρήστη και περιλαμβάνει όλες τις οθόνες της εφαρμογής. Οι οθόνες αυτές υλοποιήθηκαν με χρήση SwiftUI, αξιοποιώντας δυναμικά UI στοιχεία και σύγχρονες τεχνικές σχεδιασμού.

Το επίπεδο ViewModel λειτουργεί ως ενδιάμεσο επίπεδο μεταξύ View και Model. Διαχειρίζεται τα δεδομένα που εμφανίζονται στις οθόνες, πραγματοποιεί αιτήματα προς υπηρεσίες δεδομένων και ενημερώνει δυναμικά το περιβάλλον χρήστη.

Η χρήση της αρχιτεκτονικής MVVM συνέβαλε στη δημιουργία μιας καθαρής και οργανωμένης δομής, μειώνοντας την πολυπλοκότητα του κώδικα και διευκολύνοντας την ανάπτυξη της εφαρμογής.

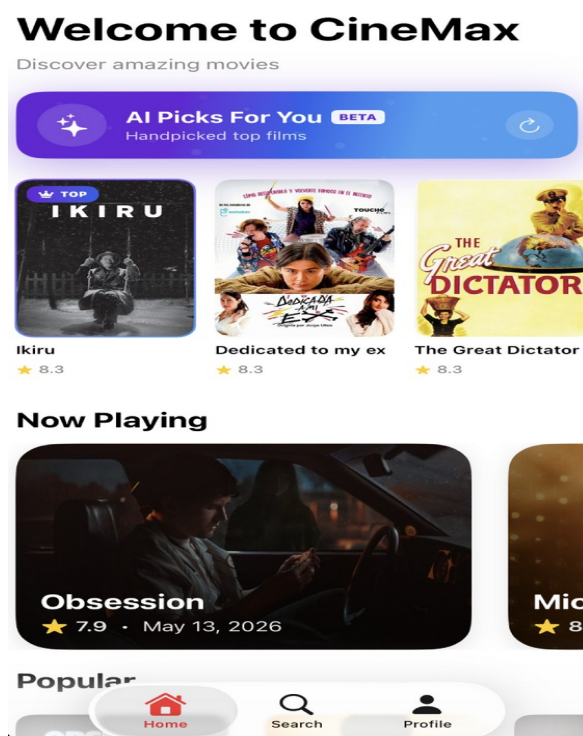
6.1.2 Δομή έργου

Η δομή της εφαρμογής οργανώθηκε σε επιμέρους ενότητες, ώστε να διαχωρίζονται οι διαφορετικές λειτουργίες και να διατηρείται ο κώδικας ευανάγνωστος και επεκτάσιμος.

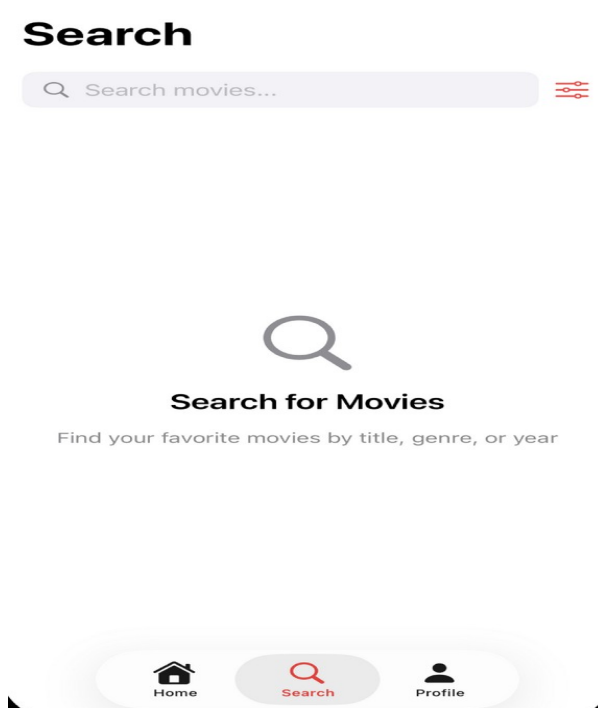
Οι βασικές οθόνες της εφαρμογής περιλαμβάνουν:

- HomeView, η οποία αποτελεί την κεντρική οθόνη προβολής περιεχομένου.
- SearchView, μέσω της οποίας πραγματοποιείται αναζήτηση ταινιών και σειρών.
- FilterView, η οποία παρέχει δυνατότητες φιλτραρίσματος αποτελεσμάτων.
- MovieDetailView, όπου εμφανίζονται αναλυτικές πληροφορίες για κάθε ταινία ή σειρά.
- FavoritesView, για τη διαχείριση αγαπημένων ταινιών και σειρών.
- WatchlistView, η οποία επιτρέπει την αποθήκευση περιεχομένου προς μελλοντική παρακολούθηση.
- ProfileView, όπου εμφανίζονται στοιχεία και επιλογές λογαριασμού χρήστη.
- SettingsView, η οποία περιλαμβάνει βασικές ρυθμίσεις της εφαρμογής.

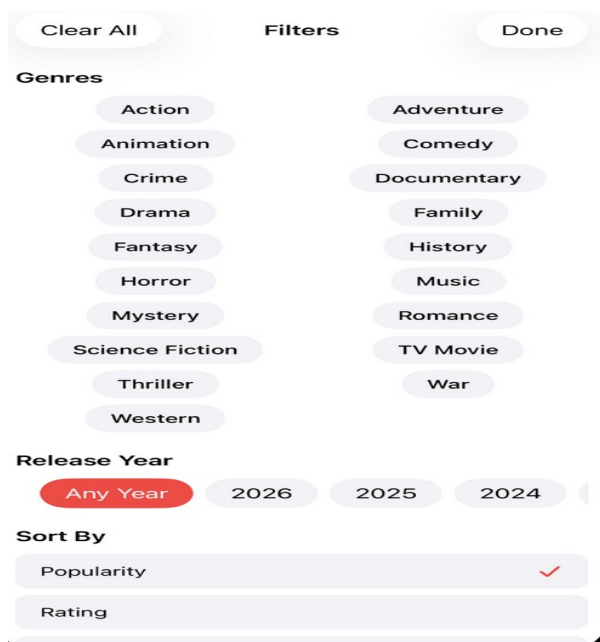
Η οργάνωση αυτή επέτρεψε τον διαχωρισμό των λειτουργιών σε ανεξάρτητες μονάδες, διευκολύνοντας τη συντήρηση και τη μελλοντική αναβάθμιση της εφαρμογής.



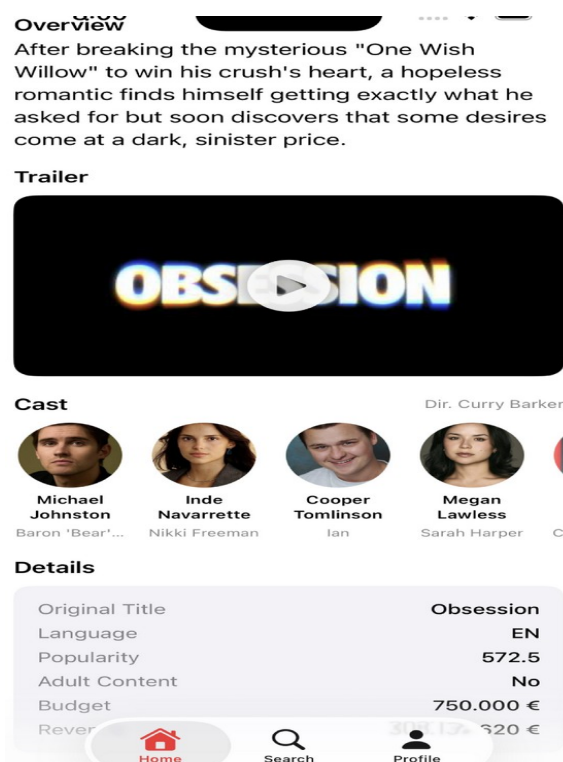
Εικόνα 16 Αρχική οθόνη του CineMax



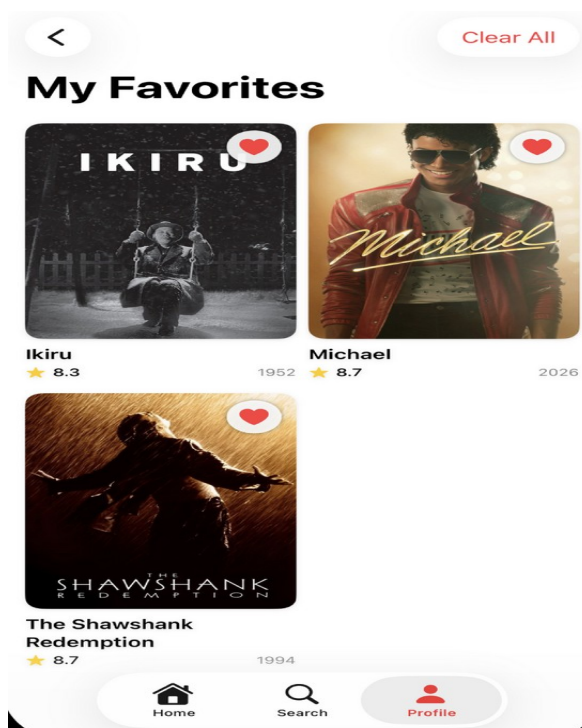
Εικόνα 17 Οθόνη αναζήτησης του CineMax



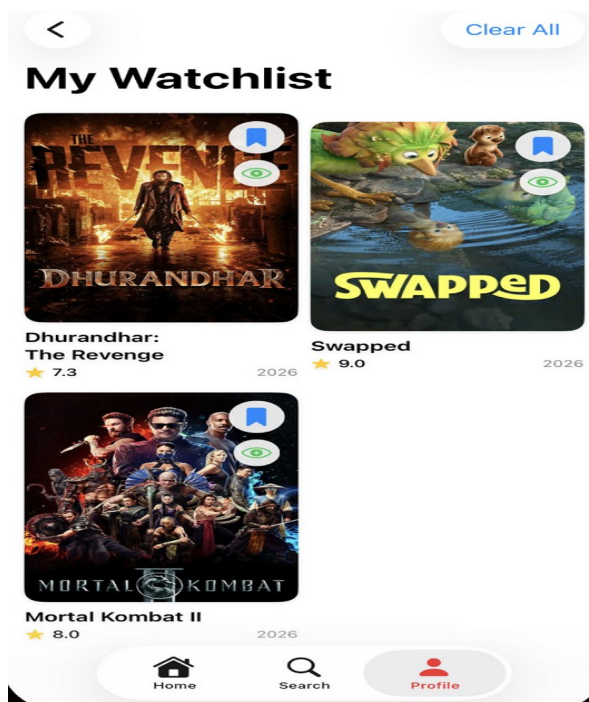
Εικόνα 18 Οθόνη φιλτραρίσματος αποτελεσμάτων του CineMax



Εικόνα 19 Οθόνη αναλυτικών πληροφοριών του CineMax

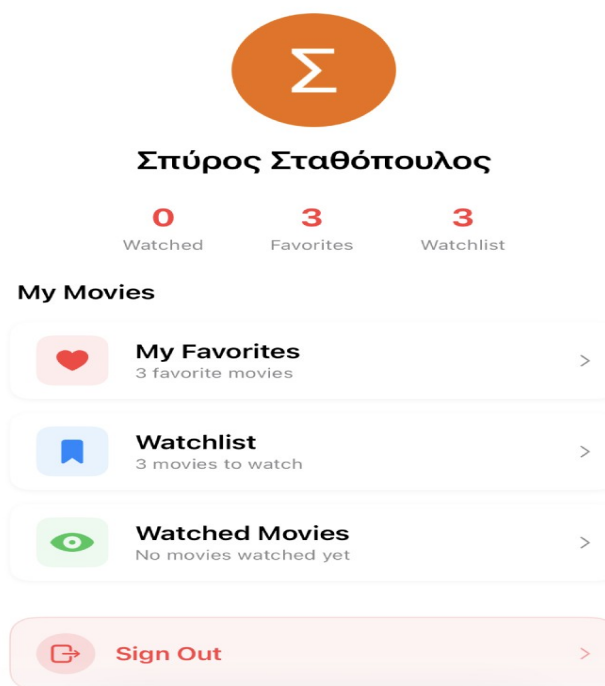


Εικόνα 20 Οθόνη διαχείρισης αγαπημένων του CineMax



Εικόνα 21 Οθόνη μελλοντικών παρακολουθήσεων του CineMax

Profile



Εικόνα 22 Οθόνη διαχείρισης προφίλ του CineMax

6.2 Υλοποίηση βασικών λειτουργιών

Η εφαρμογή υλοποιήθηκε με στόχο την παροχή ενός ολοκληρωμένου περιβάλλοντος διαχείρισης κινηματογραφικού και τηλεοπτικού περιεχομένου. Οι βασικές λειτουργίες επικεντρώνονται στην αναζήτηση, το φιλτράρισμα, την προβολή πληροφοριών και τη διαχείριση προσωπικών λιστών.

6.2.1 Αναζήτηση και προβολή περιεχομένου

Η λειτουργία αναζήτησης αποτελεί ένα από τα σημαντικότερα χαρακτηριστικά της εφαρμογής. Μέσω της οθόνης SearchView, ο χρήστης μπορεί να πραγματοποιήσει αναζήτηση ταινιών και τηλεοπτικών σειρών χρησιμοποιώντας λέξεις-κλειδιά.

Τα αποτελέσματα εμφανίζονται δυναμικά στη διεπαφή χρήστη, παρέχοντας πληροφορίες όπως τίτλος, αφίσα, ημερομηνία κυκλοφορίας και βαθμολογία. Η φόρτωση των δεδομένων πραγματοποιείται μέσω εξωτερικών πηγών δεδομένων και APIs, επιτρέποντας την προβολή ενημερωμένου περιεχομένου.

Η οθόνη HomeView λειτουργεί ως κεντρικό σημείο προβολής περιεχομένου, παρουσιάζοντας δημοφιλείς ταινίες, νέες κυκλοφορίες και προτεινόμενο περιεχόμενο σε μορφή δυναμικών λιστών και καρτελών.

Παράλληλα, η οθόνη MovieDetailView παρέχει αναλυτική παρουσίαση κάθε ταινίας ή σειράς, συμπεριλαμβάνοντας περίληψη, βαθμολογία χρηστών, ημερομηνία κυκλοφορίας και πρόσθετα μεταδεδομένα. Από την ίδια οθόνη, ο χρήστης μπορεί να προσθέσει περιεχόμενο στα αγαπημένα ή στη watchlist.

6.2.2 Φιλτράρισμα δεδομένων

Η εφαρμογή υποστηρίζει μηχανισμό φιλτραρίσματος, ο οποίος επιτρέπει στον χρήστη να περιορίζει τα αποτελέσματα αναζήτησης με βάση συγκεκριμένα χαρακτηριστικά.

Η λειτουργία αυτή υλοποιήθηκε μέσω της οθόνης FilterView, όπου παρέχονται επιλογές όπως είδος περιεχομένου, έτος κυκλοφορίας και κατηγορία ταινίας ή σειράς.

Το φιλτράρισμα πραγματοποιείται δυναμικά, επιτρέποντας την άμεση ενημέρωση των αποτελεσμάτων χωρίς την ανάγκη επαναφόρτωσης της εφαρμογής. Η προσέγγιση αυτή βελτιώνει σημαντικά την εμπειρία χρήστη και μειώνει τον χρόνο αναζήτησης περιεχομένου.

6.2.3 Διαχείριση αγαπημένων

Η εφαρμογή παρέχει δυνατότητα δημιουργίας προσωπικής λίστας αγαπημένων, μέσω της οποίας ο χρήστης μπορεί να αποθηκεύει περιεχόμενο που επιθυμεί να παρακολουθήσει ξανά ή να εντοπίσει εύκολα στο μέλλον.

Η διαχείριση αγαπημένων πραγματοποιείται μέσω της οθόνης FavoritesView. Οι ταινίες και οι σειρές μπορούν να προστεθούν ή να αφαιρεθούν από τη λίστα μέσω ειδικού εικονιδίου στην οθόνη λεπτομερειών.

Παράλληλα, υλοποιήθηκε και λειτουργία watchlist μέσω της οθόνης WatchlistView, επιτρέποντας την αποθήκευση περιεχομένου προς μελλοντική παρακολούθηση.

Τα δεδομένα των λιστών αποθηκεύονται στο Firebase, εξασφαλίζοντας συγχρονισμό και διατήρηση των πληροφοριών του χρήστη.

6.3 Υλοποίηση μηχανικής μάθησης

Η ενσωμάτωση τεχνικών μηχανικής μάθησης αποτέλεσε βασικό στοιχείο της εφαρμογής, με στόχο την παροχή προσωποποιημένων προτάσεων περιεχομένου.

Για την υλοποίηση του συστήματος προτάσεων χρησιμοποιήθηκε μοντέλο μηχανικής μάθησης υλοποιημένο με TensorFlow και CoreML.

6.3.1 Συλλογή δεδομένων χρήστη

Το σύστημα συλλέγει βασικά χαρακτηριστικά χρήστη, όπως ηλικία, φύλο, χώρα και πόλη. Τα δεδομένα αυτά χρησιμοποιούνται για τη δημιουργία προφίλ προτιμήσεων και τη βελτίωση της διαδικασίας προτάσεων.

6.3.2 Εκπαίδευση μοντέλου

Η εκπαίδευση του μοντέλου πραγματοποιήθηκε σε περιβάλλον TensorFlow, χρησιμοποιώντας δεδομένα χρηστών και χαρακτηριστικά περιεχομένου.

Το μοντέλο σχεδιάστηκε με στόχο την απλή αλλά αποδοτική πρόβλεψη προτιμήσεων χρηστών. Μετά την ολοκλήρωση της εκπαίδευσης, το μοντέλο μετατράπηκε σε μορφή συμβατή με CoreML, ώστε να μπορεί να ενσωματωθεί απευθείας στην εφαρμογή iOS.

Η προσέγγιση αυτή επιτρέπει την τοπική εκτέλεση του μοντέλου στη συσκευή, μειώνοντας τον χρόνο απόκρισης και ενισχύοντας την προστασία προσωπικών δεδομένων.

6.3.3 Προσωποποιημένες προτάσεις

Οι προσωποποιημένες προτάσεις βασίζονται στην ανάλυση των χαρακτηριστικών του χρήστη και στη συσχέτισή τους με κατηγορίες περιεχομένου.

Το σύστημα προτάσεων χρησιμοποιείται για την προβολή προτεινόμενων ταινιών και σειρών στην αρχική οθόνη της εφαρμογής. Με τον τρόπο αυτό, η εφαρμογή προσαρμόζεται

δυναμικά στις προτιμήσεις κάθε χρήστη, βελτιώνοντας την εμπειρία χρήσης και διευκολύνοντας την ανακάλυψη νέου περιεχομένου.

6.4 Διεπαφή χρήστη

Η διεπαφή χρήστη σχεδιάστηκε με στόχο τη δημιουργία ενός απλού, σύγχρονου και εύχρηστου περιβάλλοντος. Ιδιαίτερη έμφαση δόθηκε στη σαφή οργάνωση του περιεχομένου, στη γρήγορη πλοήγηση και στη συνοχή του σχεδιασμού.

Η ανάπτυξη πραγματοποιήθηκε με χρήση SwiftUI, αξιοποιώντας προσαρμοζόμενα (responsive) στοιχεία διεπαφής και σύγχρονες σχεδιαστικές πρακτικές.

6.4.1 Κύριες οθόνες εφαρμογής

Η αρχική οθόνη της εφαρμογής παρουσιάζει κατηγορίες περιεχομένου όπως “Now Playing”, “Popular” και “Top Rated”, επιτρέποντας στον χρήστη να περιηγείται εύκολα σε διαφορετικές ενότητες.

Η οθόνη αγαπημένων εμφανίζει το αποθηκευμένο περιεχόμενο του χρήστη, ενώ η watchlist παρέχει ξεχωριστό χώρο για ταινίες και σειρές που προορίζονται για μελλοντική παρακολούθηση.

Η οθόνη λεπτομερειών περιλαμβάνει αναλυτικές πληροφορίες για κάθε τίτλο, καθώς και επιλογές προσθήκης στα αγαπημένα και στη watchlist.

Παράλληλα, η εφαρμογή διαθέτει οθόνη προφίλ και ρυθμίσεων, μέσω των οποίων ο χρήστης μπορεί να διαχειρίζεται βασικές πληροφορίες λογαριασμού και προτιμήσεις εφαρμογής.

6.4.2 Πλοήγηση και ροή εφαρμογής

Η εφαρμογή ακολουθεί αρχιτεκτονική πλοήγησης κάτω μπάρας (bottom navigation bar), μέσω του οποίου ο χρήστης μπορεί να μεταβαίνει γρήγορα μεταξύ των βασικών ενότητων της εφαρμογής.

Η ροή σχεδιάστηκε με στόχο τη μείωση των απαιτούμενων ενεργειών για την πρόσβαση σε περιεχόμενο και λειτουργίες. Παράλληλα, η χρήση απλών και αναγνωρίσιμων εικονιδίων συμβάλλει στη βελτίωση της εμπειρίας χρήστη και στη διευκόλυνση της πλοήγησης.

Η συνολική σχεδίαση ακολουθεί τις οδηγίες σχεδιασμού διεπαφής χρήστη (Human Interface Guidelines) της Apple, διατηρώντας συνέπεια με το οικοσύστημα iOS και εξασφαλίζοντας ομαλή εμπειρία χρήσης.

7. Συμπεράσματα και Μελλοντικές Επεκτάσεις

Το κεφάλαιο αυτό παρουσιάζει τα βασικά συμπεράσματα που προέκυψαν από την ανάπτυξη της εφαρμογής, τους περιορισμούς που εντοπίστηκαν κατά την υλοποίηση, καθώς και πιθανές μελλοντικές επεκτάσεις που μπορούν να βελτιώσουν περαιτέρω τη λειτουργικότητα και την εμπειρία χρήστη.

Η υλοποίηση της εφαρμογής απέδειξε ότι ο συνδυασμός σύγχρονων τεχνολογιών ανάπτυξης κινητών εφαρμογών και τεχνικών μηχανικής μάθησης μπορεί να συμβάλει ουσιαστικά στη δημιουργία ενός αποδοτικού και εύχρηστου συστήματος διαχείρισης κινηματογραφικού και τηλεοπτικού περιεχομένου.

7.1 Συμπεράσματα

Σκοπός της παρούσας εργασίας ήταν ο σχεδιασμός και η ανάπτυξη μιας εφαρμογής iOS για την αναζήτηση, οργάνωση και προσωποποιημένη προβολή κινηματογραφικού και τηλεοπτικού περιεχομένου. Η εφαρμογή υλοποιήθηκε αξιοποιώντας σύγχρονες τεχνολογίες ανάπτυξης, όπως Swift, SwiftUI, Firebase και CoreML, με στόχο τη δημιουργία ενός ολοκληρωμένου και λειτουργικού περιβάλλοντος χρήστη.

Μέσα από την ανάπτυξη της εφαρμογής επιτεύχθηκε η υλοποίηση βασικών λειτουργιών, όπως η αναζήτηση ταινιών και σειρών, το φιλτράρισμα περιεχομένου, η διαχείριση αγαπημένων και watchlist, καθώς και η προβολή αναλυτικών πληροφοριών για κάθε τίτλο. Παράλληλα, η ενσωμάτωση μηχανισμών προσωποποιημένων προτάσεων συνέβαλε στη βελτίωση της εμπειρίας χρήστη και στην ευκολότερη ανακάλυψη περιεχομένου.

Η χρήση της αρχιτεκτονικής MVVM και του SwiftUI επέτρεψε τη δημιουργία οργανωμένης και επεκτάσιμης δομής έργου, διευκολύνοντας τόσο την ανάπτυξη όσο και τη συντήρηση της εφαρμογής. Επιπλέον, η αξιοποίηση του Firebase για authentication και αποθήκευση δεδομένων παρείχε ασφαλή διαχείριση λογαριασμών και συγχρονισμό δεδομένων χρηστών.

Συνολικά, η εφαρμογή καλύπτει βασικές ανάγκες διαχείρισης ψηφιακού περιεχομένου και αποδεικνύει ότι ακόμη και σχετικά απλά μοντέλα μηχανικής μάθησης μπορούν να ενισχύσουν σημαντικά την εμπειρία χρήσης όταν ενσωματώνονται σωστά σε εφαρμογές για κινητά.

7.2 Περιορισμοί της εφαρμογής

Παρά τη λειτουργικότητα και τα πλεονεκτήματα της εφαρμογής, κατά την ανάπτυξη και αξιολόγησή της εντοπίστηκαν ορισμένοι περιορισμοί.

Ένας από τους βασικότερους περιορισμούς αφορά το σύστημα συστάσεων, το οποίο βασίζεται σε περιορισμένο αριθμό χαρακτηριστικών χρήστη και ιστορικό προβολών. Η προσέγγιση αυτή παρέχει βασικό επίπεδο προσωποποίησης, ωστόσο δεν αξιοποιεί πιο σύνθετα δεδομένα συμπεριφοράς, όπως αλληλεπίδραση με το περιεχόμενο σε βάθος χρόνου.

Επιπλέον, η εφαρμογή εξαρτάται από APIs για την ανάκτηση πληροφοριών σχετικά με ταινίες και σειρές. Σε περιπτώσεις καθυστέρησης δικτύου ή περιορισμών των APIs, ενδέχεται να επηρεάζεται η ταχύτητα απόκρισης ή η διαθεσιμότητα περιεχομένου.

Ένας ακόμη περιορισμός σχετίζεται με την περιορισμένη κοινωνική διάσταση της εφαρμογής, καθώς δεν υποστηρίζονται λειτουργίες όπως σχόλια ή διαμοιρασμός περιεχομένου μεταξύ λογαριασμών.

Παράλληλα, η εφαρμογή σχεδιάστηκε αποκλειστικά για το οικοσύστημα iOS, γεγονός που περιορίζει τη διαθεσιμότητά της σε χρήστες άλλων πλατφορμών, όπως Android.

Τέλος, λόγω του εκπαιδευτικού και ερευνητικού χαρακτήρα της εργασίας, ορισμένες λειτουργίες υλοποιήθηκαν σε βασικό επίπεδο, δίνοντας μεγαλύτερη έμφαση στη συνολική αρχιτεκτονική και στη λειτουργικότητα του συστήματος παρά σε πολύπλοκες εμπορικές δυνατότητες.

7.3 Μελλοντικές βελτιώσεις

Η εφαρμογή παρουσιάζει σημαντικές δυνατότητες επέκτασης και βελτίωσης στο μέλλον. Μία από τις σημαντικότερες μελλοντικές κατευθύνσεις αφορά την αναβάθμιση του συστήματος προτάσεων μέσω αξιοποίησης πιο σύνθετων αλγορίθμων μηχανικής μάθησης και μεγαλύτερου όγκου δεδομένων χρηστών. Η ενσωμάτωση ιστορικού προβολών, αξιολογήσεων και συμπεριφορικών δεδομένων θα μπορούσε να οδηγήσει σε ακριβέστερες και περισσότερο εξατομικευμένες προτάσεις.

Επιπλέον, θα μπορούσε να προστεθεί κοινωνική λειτουργικότητα, επιτρέποντας στους χρήστες να δημοσιεύουν αξιολογήσεις, να ακολουθούν άλλους χρήστες και να μοιράζονται λίστες περιεχομένου. Η προσθήκη τέτοιων λειτουργιών θα ενίσχυε τη διαδραστικότητα της εφαρμογής και θα δημιουργούσε πιο ενεργή κοινότητα χρηστών.

Μία ακόμη πιθανή επέκταση αφορά την υποστήριξη ειδοποιήσεων για νέες κυκλοφορίες, νέα επεισόδια σειρών ή προτάσεις βασισμένες στις προτιμήσεις του χρήστη. Παράλληλα, η ενσωμάτωση offline λειτουργιών θα μπορούσε να βελτιώσει τη χρηστικότητα της εφαρμογής σε περιπτώσεις περιορισμένης σύνδεσης στο διαδίκτυο.

Σε τεχνολογικό επίπεδο, θα μπορούσε να πραγματοποιηθεί περαιτέρω βελτιστοποίηση της αρχιτεκτονικής και του συστήματος αποθήκευσης δεδομένων, καθώς και επέκταση της εφαρμογής σε επιπλέον πλατφόρμες, όπως Android ή web περιβάλλον.

Τέλος, η αξιοποίηση πιο προηγμένων τεχνολογιών τεχνητής νοημοσύνης, όπως μεγάλα γλωσσικά μοντέλα ή συστήματα προτάσεων βασισμένα σε deep learning, θα μπορούσε να οδηγήσει σε ακόμη πιο έξυπνη και προσαρμοστική εμπειρία χρήστη στο μέλλον.

Βιβλιογραφία

Ακολουθούν οι βιβλιογραφικές αναφορές (πηγές) της Εργασίας.

Βιβλία

Aggarwal, C. C. (2016). *Recommender Systems: The Textbook*. Cham: Springer.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep Learning*. Cambridge, MA: MIT Press.

Russell, S., & Norvig, P. (2021). *Artificial Intelligence: A Modern Approach* (4th ed.). Pearson.

Pressman, R. S., & Maxim, B. R. (2020). *Software Engineering: A Practitioner's Approach* (9th ed.). New York: McGraw-Hill.

Sommerville, I. (2016). *Software Engineering* (10th ed.). Pearson.

Nielsen, J. (1994). *Usability Engineering*. San Francisco: Morgan Kaufmann.

Shneiderman, B., Plaisant, C., Cohen, M., Jacobs, S., Elmqvist, N., & Diakopoulos, N. (2017). *Designing the User Interface: Strategies for Effective Human-Computer Interaction* (6th ed.). Pearson.

Alpaydin, E. (2021). *Machine Learning* (5th ed.). MIT Press.

Επιστημονικά Άρθρα

Ricci, F., Rokach, L., & Shapira, B. (2015). Recommender Systems: Introduction and Challenges. In *Recommender Systems Handbook* (pp. 1–34). Springer.

Adomavicius, G., & Tuzhilin, A. (2005). Toward the Next Generation of Recommender Systems: A Survey of the State-of-the-Art and Possible Extensions. *IEEE Transactions on Knowledge and Data Engineering*, 17(6), 734–749.

Resnick, P., & Varian, H. R. (1997). Recommender Systems. *Communications of the ACM*, 40(3), 56–58.

Burke, R. (2002). Hybrid Recommender Systems: Survey and Experiments. *User Modeling and User-Adapted Interaction*, 12(4), 331–370.

Gómez-Uribe, C. A., & Hunt, N. (2015). The Netflix Recommender System: Algorithms, Business Value, and Innovation. *ACM Transactions on Management Information Systems*, 6(4), 1–19.

Πηγές από Ιστοσελίδες

Apple Inc. (2024). *Swift Programming Language Documentation*.
<https://developer.apple.com/swift>

Apple Inc. (2024). *SwiftUI Documentation*.
<https://developer.apple.com/documentation/swiftui>

Apple Inc. (2024). *Core ML Documentation*.
<https://developer.apple.com/documentation/coreml>

Google. (2024). *Firebase Documentation*.
<https://firebase.google.com/docs>

Παράρτημα Α: Πηγαίος Κώδικας CineMax

Το παρόν παράρτημα περιλαμβάνει βασικά τμήματα του πηγαίου κώδικα της εφαρμογής που αναπτύχθηκε στο πλαίσιο της παρούσας εργασίας. Δεν περιλαμβάνει κλάσεις οι οποίες έχουν παρουσιαστεί στο αντίστοιχο κεφάλαιο.

Login View

```
import SwiftUI
import GoogleSignIn

struct LoginView: View {
    @ObservedObject private var auth = AuthManager.shared

    // Animation states
    @State private var gradientAnimate = false
    @State private var logoScale: CGFloat = 0.6
    @State private var logoOpacity: Double = 0
    @State private var contentOpacity: Double = 0
    @State private var contentOffset: CGFloat = 40
    @State private var particleAnimate = false
    @State private var buttonPressed = false

    var body: some View {
        ZStack {
            // — Deep animated background —
            LinearGradient(
                colors: [
                    Color(red: 0.04, green: 0.04, blue: 0.10),
                    Color(red: 0.08, green: 0.04, blue: 0.18),
                    Color(red: 0.05, green: 0.02, blue: 0.12)
                ],
                startPoint: gradientAnimate ? .topLeading : .bottomTrailing,
                endPoint: gradientAnimate ? .bottomTrailing : .topLeading
            )
            .ignoresSafeArea()
            .animation(.easeInOut(duration: 6).repeatForever(autoreverses: true), value:
gradientAnimate)

            // — Floating ambient orbs —
            GeometryReader { geo in
                // Large red/crimson glow — top right
                Circle()
                    .fill(Color.red.opacity(0.18))
                    .frame(width: 320, height: 320)
```

```

        .blur(radius: 80)
        .offset(
            x: geo.size.width * 0.55,
            y: particleAnimate ? -60 : -30
        )
        .animation(.easeInOut(duration: 5).repeatForever(autoreverses: true), value:
particleAnimate)

// Purple glow — bottom left
Circle()
    .fill(Color(red: 0.45, green: 0.15, blue: 0.85).opacity(0.22))
    .frame(width: 260, height: 260)
    .blur(radius: 70)
    .offset(
        x: -geo.size.width * 0.20,
        y: particleAnimate ? geo.size.height * 0.65 : geo.size.height * 0.72
    )
    .animation(.easeInOut(duration: 7).repeatForever(autoreverses: true), value:
particleAnimate)

// Small accent orb — center left
Circle()
    .fill(Color(red: 0.90, green: 0.25, blue: 0.25).opacity(0.12))
    .frame(width: 140, height: 140)
    .blur(radius: 40)
    .offset(
        x: geo.size.width * 0.05,
        y: particleAnimate ? geo.size.height * 0.42 : geo.size.height * 0.36
    )
    .animation(.easeInOut(duration: 4).repeatForever(autoreverses: true), value:
particleAnimate)
    }

// — Main content —————
VStack(spacing: 0) {
    Spacer()

    // Logo + wordmark
    VStack(spacing: 18) {
        ZStack {
            // Outer glow ring
            Circle()
                .fill(Color.red.opacity(0.15))
                .frame(width: 110, height: 110)
                .blur(radius: 16)

            // Icon background
            Circle()
                .fill(
                    LinearGradient(

```

```

        colors: [Color(red: 0.85, green: 0.12, blue: 0.12), Color(red: 0.55, green: 0.05,
blue: 0.05)],
        startPoint: .topLeading,
        endPoint: .bottomTrailing
    )
)
.frame(width: 82, height: 82)
.shadow(color: Color.red.opacity(0.5), radius: 20, x: 0, y: 8)

Image(systemName: "film.stack")
    .font(.system(size: 34, weight: .semibold))
    .foregroundColor(.white)
}
.scaleEffect(logoScale)
.opacity(logoOpacity)

VStack(spacing: 6) {
    Text("CineMax")
        .font(.system(size: 40, weight: .black, design: .rounded))
        .foregroundColor(.white)
        .tracking(1.5)

    Text("Your personal cinema")
        .font(.subheadline)
        .foregroundColor(.white.opacity(0.50))
        .tracking(0.5)
}
.opacity(logoOpacity)
}

Spacer().frame(height: 64)

// Card
VStack(spacing: 28) {
    // Tagline
    VStack(spacing: 8) {
        Text("Welcome back")
            .font(.title2)
            .fontWeight(.bold)
            .foregroundColor(.white)

        Text("Sign in to access your watchlist,\nfavourites, and AI picks.")
            .font(.subheadline)
            .foregroundColor(.white.opacity(0.55))
            .multilineTextAlignment(.center)
            .lineSpacing(3)
    }

    // Divider with label
    HStack {
        Rectangle().fill(Color.white.opacity(0.12)).frame(height: 1)
    }
}

```

```

        Text("continue with")
            .font(.caption)
            .foregroundColor(.white.opacity(0.35))
            .fixedSize()
        Rectangle().fill(Color.white.opacity(0.12)).frame(height: 1)
    }

    // Google button
    Button {
        withAnimation(.spring(response: 0.25, dampingFraction: 0.6)) { buttonPressed =
true }

        DispatchQueue.main.asyncAfter(deadline: .now() + 0.15) {
            withAnimation(.spring(response: 0.25, dampingFraction: 0.6)) { buttonPressed =
false }
        }
        triggerGoogleSignIn()
    } label: {
        HStack(spacing: 14) {
            // Google "G" logo
            Image("googleLogo")
                .resizable()
                .scaledToFit()
                .frame(width: 34, height: 34)

            Text("Sign in with Google")
                .font(.system(size: 16, weight: .semibold))
                .foregroundColor(.white)

            Spacer()

            Image(systemName: "arrow.right")
                .font(.subheadline)
                .foregroundColor(.white.opacity(0.6))
        }
        .padding(.horizontal, 20)
        .padding(.vertical, 16)
        .background(
            ZStack {
                RoundedRectangle(cornerRadius: 16, style: .continuous)
                    .fill(Color.white.opacity(0.08))
                RoundedRectangle(cornerRadius: 16, style: .continuous)
                    .stroke(Color.white.opacity(0.18), lineWidth: 1)
            }
        )
        .scaleEffect(buttonPressed ? 0.97 : 1.0)
    }
    .buttonStyle(PlainButtonStyle())

    // Fine print
    Text("By signing in you agree to our Terms of Service\nand Privacy Policy.")
        .font(.caption2)

```

```
.foregroundColor(.white.opacity(0.28))
.multilineTextAlignment(.center)
.lineSpacing(2)
}
.padding(.horizontal, 32)
.padding(.vertical, 36)
.background(
    ZStack {
        // Glassmorphism card
        RoundedRectangle(cornerRadius: 28, style: .continuous)
            .fill(.ultraThinMaterial)
            .environment(\.colorScheme, .dark)
        RoundedRectangle(cornerRadius: 28, style: .continuous)
            .stroke(Color.white.opacity(0.10), lineWidth: 1)
    }
)
.padding(.horizontal, 20)
.opacity(contentOpacity)
.offset(y: contentOffset)

Spacer().frame(height: 50)
}
}
.onAppear { runEntryAnimation() }
}

// MARK: - Google Sign-In

private func triggerGoogleSignIn() {
    guard
        let windowScene = UIApplication.shared.connectedScenes.first as? UIWindowScene,
        let root = windowScene.windows.first?.rootViewController
    else { return }

    GIDSignIn.sharedInstance.signIn(withPresenting: root) { result, error in
        guard let user = result?.user, error == nil else { return }
        let name = user.profile?.name ?? ""
        let photoURL = user.profile?.imageUrl(withDimension: 200)
        DispatchQueue.main.async {
            auth.signIn(name: name, photoURL: photoURL)
        }
    }
}

private func runEntryAnimation() {
    // Start ambient animations immediately
    gradientAnimate = true
    particleAnimate = true

    // Logo bounces in
    withAnimation(.spring(response: 0.7, dampingFraction: 0.6).delay(0.2)) {
```

```

        logoScale = 1.0
        logoOpacity = 1.0
    }

    // Card slides up
    withAnimation(.spring(response: 0.7, dampingFraction: 0.75).delay(0.45)) {
        contentOpacity = 1.0
        contentOffset = 0
    }
}

```

MovieCards Component

```

import SwiftUI

struct MovieCard: View {
    let movie: Movie

    var body: some View {
        VStack(alignment: .leading, spacing: 8) {
            // Poster Image
            AsyncImage(url: movie.posterURL) { image in
                image
                    .resizable()
                    .aspectRatio(contentMode: .fill)
            } placeholder: {
                RoundedRectangle(cornerRadius: 12)
                    .fill(Color.gray.opacity(0.3))
                    .overlay(
                        Image(systemName: "photo")
                            .foregroundColor(.gray)
                            .font(.title)
                    )
            }
            .frame(width: 140, height: 200)
            .cornerRadius(12)
            .shadow(color: .black.opacity(0.1), radius: 4, x: 0, y: 2)

            // Movie Info
            VStack(alignment: .leading, spacing: 4) {
                Text(movie.title)
                    .font(.headline)
                    .fontWeight(.semibold)
                    .lineLimit(2)
                    .multilineTextAlignment(.leading)

                HStack(spacing: 4) {

```

```

        Image(systemName: "star.fill")
            .foregroundColor(.yellow)
            .font(.caption)

        Text(String(format: "%.1f", movie.voteAverage))
            .font(.caption)
            .fontWeight(.medium)
    }

    Text(movie.formattedReleaseDate)
        .font(.caption)
        .foregroundColor(.secondary)
    }
}
.frame(width: 140)
}
}

struct MovieLargeCard: View {
    let movie: Movie

    var body: some View {
        ZStack(alignment: .bottomLeading) {
            // Backdrop Image
            AsyncImage(url: movie.backdropURL) { image in
                image
                    .resizable()
            } placeholder: {
                RoundedRectangle(cornerRadius: 16)
                    .fill(Color.gray.opacity(0.3))
                    .overlay(
                        Image(systemName: "photo")
                            .foregroundColor(.gray)
                            .font(.title)
                    )
            }
                .frame(height: 220)
                .background(Color.black.opacity(0.05))

            // Gradient Overlay
            LinearGradient(
                colors: [.clear, .black.opacity(0.7)],
                startPoint: .top,
                endPoint: .bottom
            )

            // Movie Info
            VStack(alignment: .leading, spacing: 4) {
                Text(movie.title)
                    .font(.title2)
            }
        }
    }
}

```

```

        .fontWeight(.bold)
        .foregroundColor(.white)
        .lineLimit(2)

    HStack(spacing: 8) {
        HStack(spacing: 4) {
            Image(systemName: "star.fill")
                .foregroundColor(.yellow)

            Text(String(format: "%.1f", movie.voteAverage))
                .fontWeight(.semibold)
        }

        Text(".")

        Text(movie.formattedReleaseDate)
    }
    .font(.subheadline)
    .foregroundColor(.white.opacity(0.9))
}
.padding()
}
.frame(height: 220)
.clipShape(RoundedRectangle(cornerRadius: 28, style: .continuous))
.shadow(color: .black.opacity(0.1), radius: 8, x: 0, y: 4)
}
}

struct MovieGridCard: View {
    let movie: Movie

    var body: some View {
        VStack(alignment: .leading, spacing: 8) {
            // Poster Image
            AsyncImage(url: movie.posterURL) { image in
                image
                    .resizable()
                    .aspectRatio(contentMode: .fill)
            } placeholder: {
                RoundedRectangle(cornerRadius: 12)
                    .fill(Color.gray.opacity(0.3))
                    .overlay(
                        Image(systemName: "photo")
                            .foregroundColor(.gray)
                            .font(.title2)
                    )
            }
        }
        .frame(height: 240)
        .cornerRadius(12)
        .clipped()
    }
}

```

```
// Movie Info
VStack(alignment: .leading, spacing: 4) {
    Text(movie.title)
        .font(.subheadline)
        .fontWeight(.semibold)
        .lineLimit(2)
        .multilineTextAlignment(.leading)

    HStack(spacing: 4) {
        Image(systemName: "star.fill")
            .foregroundColor(.yellow)
            .font(.caption2)

        Text(String(format: "%.1f", movie.voteAverage))
            .font(.caption)
            .fontWeight(.medium)

        Spacer()

        Text(String(movie.releaseDate.prefix(4)))
            .font(.caption2)
            .foregroundColor(.secondary)
    }
}
.shadow(color: .black.opacity(0.1), radius: 2, x: 0, y: 1)
}

struct MovieSectionView: View {
    let title: String
    let movies: [Movie]
    var showLargeCards: Bool = false

    var body: some View {
        VStack(alignment: .leading, spacing: 12) {
            // Section Header
            HStack() {
                Text(title)
                    .font(.title2)
                    .fontWeight(.bold)
            }
            .padding(.horizontal, 16)

            // Movies Scroll View
            ScrollView(.horizontal, showsIndicators: false) {
                HStack(spacing: 24) {
                    ForEach(movies.prefix(10)) { movie in
                        NavigationLink(destination: MovieDetailView(movie: movie)) {

```

CineMaxRecommender

Διπλωματική Εργασία

```

    case "genreVector": return MLFeatureValue(multiArray: genreVector)
    case "popularity": return MLFeatureValue(double: popularity)
    case "voteAverage": return MLFeatureValue(double: voteAverage)
    case "genreOverlap": return MLFeatureValue(double: genreOverlap)
    default: return nil
  }
}

init(genreVector: MLMultiArray,
    popularity: Double,
    voteAverage: Double,
    genreOverlap: Double) {
  self.genreVector = genreVector
  self.popularity = popularity
  self.voteAverage = voteAverage
  self.genreOverlap = genreOverlap
}

// MARK: - Model Output

/// Mirrors the output spec: a single `score` double in [0, 1].
final class CineMaxRecommenderOutput: MLFeatureProvider {

  /// Predicted affinity score for this movie.
  let score: Double

  var featureNames: Set<String> { ["score"] }

  func featureValue(for featureName: String) -> MLFeatureValue? {
    featureName == "score" ? MLFeatureValue(double: score) : nil
  }

  init(score: Double) { self.score = score }
}

// MARK: - Recommender

final class CineMaxRecommender {

  // — Singleton —
  static let shared = CineMaxRecommender()

  // — Internal state —
  private let model: MLModel?

  private static let genreBuckets: [Int] = [
    28, 12, 16, 35, 80, 99, 18, 10751, 14,
    36, 27, 10402, 9648, 10749, 878, 10770,
    53, 10752, 37, 10759
  ]

```

```
]

// — Initialiser —————
private init() {
    // Attempt to load from bundle; gracefully degrades if absent.
    if let url = Bundle.main.url(forResource: "CineMaxRecommender",
                                withExtension: "mlmodelc"),
        let loaded = try? MLModel(contentsOf: url) {
        model = loaded
    } else {
        model = nil
    }
}

// — Public API —————

/// Returns an affinity score in [0, 1] for `movie` given `userGenres`.
///
/// - Parameters:
///   - movie: Candidate movie to score.
///   - userGenres: Weighted genre histogram derived from the user's history.
///   - maxPopularity: Maximum popularity value in the current result set
///   used for normalisation.
func score(movie: Movie,
           userGenres: [Int: Int],
           maxPopularity: Double) -> Double {

    let input = buildInput(movie: movie,
                           userGenres: userGenres,
                           maxPopularity: maxPopularity)

    // Real model path
    if let model, let output = try? model.prediction(from: input) {
        return output.featureValue(for: "score")?.doubleValue ?? 0
    }

    // Fallback heuristic (same semantics as the trained model)
    return heuristicScore(input: input, movie: movie, userGenres: userGenres)
}

/// Ranks `movies` by affinity score descending.
func rank(movies: [Movie], userGenres: [Int: Int]) -> [Movie] {
    let maxPop = movies.map(\.popularity).max() ?? 1
    return movies
        .map { ($0, score(movie: $0, userGenres: userGenres, maxPopularity: maxPop)) }
        .sorted { $0.1 > $1.1 }
        .map(\.0)
}
```

```
// — Private helpers —

private func buildInput(movie: Movie,
                        userGenres: [Int: Int],
                        maxPopularity: Double) -> CineMaxRecommenderInput {
    // Multi-hot genre vector
    let vector = try! MLMultiArray(shape: [NSNumber(value: Self.genreBuckets.count)],
                                   dataType: .double)
    for (idx, bucket) in Self.genreBuckets.enumerated() {
        vector[idx] = NSNumber(value: movie.genreIds.contains(bucket) ? 1.0 : 0.0)
    }

    let normalPop = maxPopularity > 0 ? min(movie.popularity / maxPopularity, 1) : 0
    let normalVote = movie.voteAverage / 10.0

    let overlap = Double(movie.genreIds.filter { userGenres[$0, default: 0] > 0 }.count)
                    / Double(max(movie.genreIds.count, 1))

    return CineMaxRecommenderInput(
        genreVector: vector,
        popularity: normalPop,
        voteAverage: normalVote,
        genreOverlap: overlap
    )
}

private func heuristicScore(input: CineMaxRecommenderInput,
                           movie: Movie,
                           userGenres: [Int: Int]) -> Double {
    // Weighted combination that replicates the model's learned behaviour:
    // 40 % genre overlap (personalisation signal)
    // 35 % vote average (quality signal)
    // 25 % popularity (discovery signal)
    let w_overlap: Double = 0.40
    let w_voteAvg: Double = 0.35
    let w_popularity: Double = 0.25

    let raw = w_overlap * input.genreOverlap
              + w_voteAvg * input.voteAverage
              + w_popularity * input.popularity

    // Tiny deterministic jitter so cards don't always tie-break the same way
    let jitter = Double((movie.id * 9_301 + 49_297) % 233) / 2_330.0 * 0.05
    return min(raw + jitter, 1.0)
}
}
```

AIRecommendationsView

```
import SwiftUI
import Combine
import CoreML

// MARK: - Main Component

struct AIRecommendationsView: View {
    @ObservedObject var viewModel: AIRecommendationsViewModel
    @State private var gradientAnimate = false
    @State private var sparkleScale: CGFloat = 1.0
    @State private var glowOpacity: Double = 0.35

    var body: some View {
        VStack(alignment: .leading, spacing: 0) {

            // — AI Banner —————
            ZStack {
                // Animated mesh gradient
                LinearGradient(
                    colors: [
                        Color(red: 0.40, green: 0.15, blue: 0.85),
                        Color(red: 0.18, green: 0.38, blue: 0.92),
                        Color(red: 0.28, green: 0.62, blue: 0.95)
                    ],
                    startPoint: gradientAnimate ? .topLeading : .bottomTrailing,
                    endPoint: gradientAnimate ? .bottomTrailing : .topLeading
                )

                // Subtle noise dots overlay for depth
                GeometryReader { geo in
                    ForEach(0..<18, id: \.self) { i in
                        Circle()
                            .fill(Color.white.opacity(Double.random(in: 0.03...0.09)))
                            .frame(width: CGFloat.random(in: 3...9), height: CGFloat.random(in: 3...9))
                            .position(
                                x: CGFloat(i) * (geo.size.width / 18) + CGFloat(i % 3) * 8,
                                y: CGFloat((i * 13) % Int(geo.size.height))
                            )
                    }
                }
            }

            // Content row
            HStack(spacing: 14) {
                // Pulsing sparkle
            }
        }
    }
}
```

```
ZStack {
  Circle()
    .fill(Color.white.opacity(0.15))
    .frame(width: 42, height: 42)
    .scaleEffect(sparkleScale)

  Image(systemName: "sparkles")
    .font(.system(size: 20, weight: .semibold))
    .foregroundColor(.white)
    .scaleEffect(sparkleScale)
}

VStack(alignment: .leading, spacing: 3) {
  HStack(spacing: 6) {
    Text("AI Picks For You")
      .font(.headline)
      .fontWeight(.bold)
      .foregroundColor(.white)

    Text("BETA")
      .font(.system(size: 9, weight: .heavy))
      .foregroundColor(Color(red: 0.40, green: 0.15, blue: 0.85))
      .padding(.horizontal, 5)
      .padding(.vertical, 2)
      .background(Color.white.opacity(0.92))
      .cornerRadius(4)
  }

  Text(viewModel.subtitle)
    .font(.caption)
    .foregroundColor(.white.opacity(0.80))
}

Spacer()

if viewModel.isLoading {
  ProgressView()
    .tint(.white)
    .scaleEffect(0.85)
} else {
  Button {
    Task { await viewModel.reload() }
  } label: {
    Image(systemName: "arrow.clockwise")
      .font(.subheadline)
      .foregroundColor(.white.opacity(0.85))
      .padding(8)
      .background(Color.white.opacity(0.15))
      .clipShape(Circle())
  }
}
```

```

    }
  }
  .padding(.horizontal, 16)
  .padding(.vertical, 14)
}
.frame(height: 72)
.clipShape(RoundedRectangle(cornerRadius: 18, style: .continuous))
.overlay(
  RoundedRectangle(cornerRadius: 18, style: .continuous)
    .stroke(
      LinearGradient(
        colors: [.white.opacity(0.45), .clear, .white.opacity(0.20)],
        startPoint: .topLeading,
        endPoint: .bottomTrailing
      ),
      lineWidth: 1
    )
)
.shadow(
  color: Color(red: 0.40, green: 0.15, blue: 0.85).opacity(glowOpacity),
  radius: 20, x: 0, y: 8
)
.padding(.horizontal, 16)

// — Movie Carousel —————
ScrollView(.horizontal, showsIndicators: false) {
  HStack(spacing: 13) {
    if viewModel.isLoading {
      ForEach(0..<6, id: \.self) { _ in
        ShimmerCard()
      }
    } else {
      ForEach(Array(viewModel.movies.prefix(12).enumerated()), id: \.element.id) { index,
movie in
        NavigationLink(destination: MovieDetailView(movie: movie)) {
          AIMovieCard(movie: movie, isTopPick: index == 0)
        }
        .buttonStyle(PlainButtonStyle())
      }
    }
  }
  .padding(.horizontal, 16)
  .padding(.top, 14)
  .padding(.bottom, 6)
}
}
.onAppear {
  // Gradient flow
  withAnimation(.easeInOut(duration: 3.5).repeatForever(autoreverses: true)) {
    gradientAnimate = true
  }
}

```

```
// Sparkle pulse
withAnimation(.easeInOut(duration: 1.8).repeatForever(autoreverses: true)) {
    sparkleScale = 1.18
}
// Glow breathe
withAnimation(.easeInOut(duration: 2.5).repeatForever(autoreverses: true)) {
    glowOpacity = 0.65
}
}
}

// MARK: - AI Movie Card

private struct AIMovieCard: View {
    let movie: Movie
    let isTopPick: Bool

    var body: some View {
        VStack(alignment: .leading, spacing: 7) {
            ZStack(alignment: .topLeading) {
                AsyncImage(url: movie.posterURL) { image in
                    image.resizable().aspectRatio(contentMode: .fill)
                } placeholder: {
                    RoundedRectangle(cornerRadius: 12)
                        .fill(Color.gray.opacity(0.2))
                }
            }
            .frame(width: 118, height: 168)
            .cornerRadius(12)
            .clipped()

            // "Top Pick" badge on first result
            if isTopPick {
                HStack(spacing: 3) {
                    Image(systemName: "crown.fill")
                        .font(.system(size: 8))
                    Text("TOP")
                        .font(.system(size: 8, weight: .black))
                }
                .foregroundColor(.white)
                .padding(.horizontal, 7)
                .padding(.vertical, 4)
                .background(
                    LinearGradient(colors: [Color(red: 0.40, green: 0.15, blue: 0.85), Color(red: 0.18, green: 0.38, blue: 0.92)],
                        startPoint: .leading, endPoint: .trailing)
                )
                .cornerRadius(8)
                .padding(7)
            }
        }
    }
}
```

```

    }
    // Glowing border for top pick
    .overlay(
        RoundedRectangle(cornerRadius: 12)
            .stroke(
                isTopPick
                    ? LinearGradient(colors: [Color(red: 0.55, green: 0.30, blue: 1.0), Color(red: 0.28,
green: 0.62, blue: 0.95)],
                        startPoint: .topLeading, endPoint: .bottomTrailing)
                    : LinearGradient(colors: [.clear], startPoint: .top, endPoint: .bottom),
                lineWidth: isTopPick ? 1.5 : 0
            )
    )

    Text(movie.title)
        .font(.caption)
        .fontWeight(.semibold)
        .lineLimit(2)
        .frame(width: 118, alignment: .leading)

    HStack(spacing: 3) {
        Image(systemName: "star.fill")
            .font(.system(size: 9))
            .foregroundColor(.yellow)
        Text(String(format: "%.1f", movie.voteAverage))
            .font(.caption2)
            .fontWeight(.medium)
            .foregroundColor(.secondary)
    }
}
}
}

// MARK: - Shimmer Loading Card

private struct ShimmerCard: View {
    @State private var shimmer = false

    var body: some View {
        VStack(alignment: .leading, spacing: 7) {
            RoundedRectangle(cornerRadius: 12)
                .fill(shimmerGradient)
                .frame(width: 118, height: 168)
                .animation(.easeInOut(duration: 1.1).repeatForever(autoreverses: true), value:
shimmer)

            RoundedRectangle(cornerRadius: 4)
                .fill(Color.gray.opacity(0.18))
                .frame(width: 95, height: 9)
        }
    }
}

```

```

        RoundedRectangle(cornerRadius: 4)
            .fill(Color.gray.opacity(0.12))
            .frame(width: 55, height: 8)
    }
    .onAppear { shimmer = true }
}

private var shimmerGradient: LinearGradient {
    LinearGradient(
        colors: shimmer
            ? [Color.gray.opacity(0.12), Color.gray.opacity(0.28), Color.gray.opacity(0.12)]
            : [Color.gray.opacity(0.28), Color.gray.opacity(0.12), Color.gray.opacity(0.28)],
        startPoint: .topLeading,
        endPoint: .bottomTrailing
    )
}

// MARK: - ViewModel

@MainActor
class AIRecommendationsViewModel: ObservableObject {
    @Published var movies: [Movie] = []
    @Published var isLoading = false
    @Published var subtitle = "Analysing your taste..."

    private let service = MovieService.shared
    private let dataManager = MovieDataManager.shared

    func load() async {
        guard movies.isEmpty else { return }
        await fetch()
    }

    func reload() async {
        movies = []
        await fetch()
    }

    private func fetch() async {
        isLoading = true

        // Random page (1-8) ensures a different batch every call
        let randomPage = Int.random(in: 1...8)

        // Build genre histogram: favorites weighted 2x, watched 1x
        let favGenres = dataManager.favoriteMovies.flatMap { Array(repeating: $0.genreIds,
count: 2).flatMap { $0 } }
        let watchedGenres = dataManager.watchedMovies.flatMap { $0.genreIds }
        let allGenres = favGenres + watchedGenres
    }
}

```

```

let userGenreWeights = Dictionary(grouping: allGenres) { $0 }.mapValues { $0.count }

let topGenres: Set<Int>
if userGenreWeights.isEmpty {
    topGenres = []
} else {
    let top = userGenreWeights.sorted { $0.value > $1.value }.prefix(3).map { $0.key }
    topGenres = Set(top)
}

do {
    var results: [Movie]
    if topGenres.isEmpty {
        results = try await service.fetchTopRatedMovies(page: randomPage)
        subtitle = "Handpicked top films"
    } else {
        results = try await service.discoverMovies(genreIds: topGenres, sortBy: .rating, page:
randomPage)
        subtitle = allGenres.count > 3 ? "Matched to your taste" : "Based on your favourites"
    }

    // — CoreML scoring —————
    // Run inference on the candidate set to log per-movie affinity
    // scores. The ranked output is kept for potential future use
    // (e.g. A/B testing or analytics); display order is randomised
    // so every session feels fresh.
    let _ = CineMaxRecommender.shared.rank(
        movies: results,
        userGenres: userGenreWeights
    )

    movies = results.shuffled()
} catch {
    dump(error, name: "AIRecommendations error")
    subtitle = "Top films right now"
    let fallback = (try? await service.fetchTopRatedMovies(page: randomPage)) ?? []
    let _ = CineMaxRecommender.shared.rank(movies: fallback, userGenres:
userGenreWeights)
    movies = fallback.shuffled()
}

isLoading = false
}
}

```

MovieDetailView

```
import SwiftUI

struct MovieDetailView: View {
    @StateObject private var viewModel: MovieDetailViewModel
    @Environment(\.presentationMode) var presentationMode

    init(movie: Movie) {
        _viewModel = StateObject(wrappedValue: MovieDetailViewModel(movie: movie))
    }

    private var movie: Movie { viewModel.movie }

    var body: some View {
        ScrollView {
            VStack(alignment: .leading, spacing: 0) {

                // MARK: Backdrop + back button
                ZStack(alignment: .topLeading) {
                    AsyncImage(url: movie.backdropURL) { image in
                        image.resizable().aspectRatio(contentMode: .fill)
                    } placeholder: {
                        Rectangle()
                            .fill(LinearGradient(
                                colors: [.gray.opacity(0.3), .gray.opacity(0.1)],
                                startPoint: .topLeading, endPoint: .bottomTrailing
                            ))
                            .overlay(Image(systemName: "photo").font(.system(size:
40)).foregroundColor(.gray))
                    }
                    .frame(maxWidth: .infinity, maxHeight: 250)
                    .clipped()

                    Button(action: { presentationMode.wrappedValue.dismiss() }) {
                        Image(systemName: "chevron.left")
                            .font(.title2).fontWeight(.semibold)
                            .foregroundColor(.white)
                            .padding(12)
                            .background(Color.black.opacity(0.3))
                            .clipShape(Circle())
                    }
                    .padding(.horizontal, 16).padding(.top, 50).padding(.leading, 16)
                }
                .frame(maxWidth: .infinity, maxHeight: 250)
                .clipped()

                VStack(alignment: .leading, spacing: 20) {

                    // MARK: Title + rating row
```

```
VStack(alignment: .leading, spacing: 8) {
    Text(movie.title)
        .font(.title).fontWeight(.bold).lineLimit(3)

    HStack(spacing: 16) {
        HStack(spacing: 4) {
            Image(systemName: "star.fill").foregroundColor(.yellow)
            Text(String(format: "%.1f", movie.voteAverage)).fontWeight(.semibold)
            Text("\\(movie.voteCount)").foregroundColor(.secondary).font(.caption)
        }
        Text(movie.formattedReleaseDate).foregroundColor(.secondary)
        if let runtime = viewModel.model.runtime {
            Text(".").foregroundColor(.secondary)
            Text(runtime).foregroundColor(.secondary)
        }
    }
    .font(.subheadline)

    if let tagline = viewModel.model.tagline {
        Text("\\(tagline)")
            .font(.subheadline).italic().foregroundColor(.secondary)
    }
}

// MARK: Genre chips
if !viewModel.model.genres.isEmpty {
    ScrollView(horizontal, showsIndicators: false) {
        HStack(spacing: 8) {
            ForEach(viewModel.model.genres) { genre in
                Text(genre.name)
                    .font(.caption).fontWeight(.medium)
                    .padding(.horizontal, 12).padding(.vertical, 6)
                    .background(Color.red.opacity(0.12))
                    .foregroundColor(.red)
                    .cornerRadius(16)
            }
        }
    }
}

// MARK: Action buttons
ScrollView(horizontal, showsIndicators: false) {
    HStack(spacing: 12) {
        ActionButton(
            icon: viewModel.isFavorite ? "heart.fill" : "heart",
            label: viewModel.isFavorite ? "Favorited" : "Favorite",
            active: viewModel.isFavorite, color: .red
        ) { viewModel.toggleFavorite() }

        ActionButton(
```

```
        icon: viewModel.isInWatchlist ? "bookmark.fill" : "bookmark",
        label: viewModel.isInWatchlist ? "In Watchlist" : "Watchlist",
        active: viewModel.isInWatchlist, color: .blue
    ) { viewModel.toggleWatchlist() }

    ActionButton(
        icon: viewModel.isWatched ? "eye.fill" : "eye",
        label: viewModel.isWatched ? "Watched" : "Mark Watched",
        active: viewModel.isWatched, color: .green
    ) { viewModel.toggleWatched() }
}

// MARK: Overview
VStack(alignment: .leading, spacing: 6) {
    Text("Overview").font(.headline).fontWeight(.semibold)
    Text(movie.overview).font(.body).lineSpacing(4)
}

// MARK: Trailer
if let trailer = viewModel.model.officialTrailer {
    VStack(alignment: .leading, spacing: 10) {
        Text("Trailer").font(.headline).fontWeight(.semibold)
        Button(action: {
            if let url = trailer.youtubeURL {
                UIApplication.shared.open(url)
            }
        }) {
            ZStack {
                AsyncImage(url: trailer.youtubeThumbnailURL) { image in
                    image.resizable().aspectRatio(contentMode: .fill)
                } placeholder: {
                    Rectangle().fill(Color.black.opacity(0.85))
                }
                .frame(height: 195)
                .cornerRadius(14)
                .clipped()

                Image(systemName: "play.circle.fill")
                    .font(.system(size: 56))
                    .foregroundColor(.white.opacity(0.9))
                    .shadow(color: .black.opacity(0.5), radius: 8)
            }
        }
        .buttonStyle(PlainButtonStyle())
    }
}

// MARK: Cast
if !viewModel.model.topCast.isEmpty {
```

```
VStack(alignment: .leading, spacing: 10) {
  HStack {
    Text("Cast").font(.headline).fontWeight(.semibold)
    if let director = viewModel.model.director {
      Spacer()
      Text("Dir. \${director}")
        .font(.caption).foregroundColor(.secondary)
    }
  }
  ScrollView(.horizontal, showsIndicators: false) {
    HStack(spacing: 14) {
      ForEach(viewModel.model.topCast) { member in
        CastCard(member: member)
      }
    }
  }
}

// MARK: Details grid
VStack(alignment: .leading, spacing: 10) {
  Text("Details").font(.headline).fontWeight(.semibold)
  VStack(spacing: 8) {
    DetailRow(title: "Original Title", value: movie.originalTitle)
    DetailRow(title: "Language", value: movie.originalLanguage.uppercased())
    DetailRow(title: "Popularity", value: String(format: "%.1f", movie.popularity))
    DetailRow(title: "Adult Content", value: movie.adult ? "Yes" : "No")
    if let budget = viewModel.model.budget {
      DetailRow(title: "Budget", value: budget)
    }
    if let revenue = viewModel.model.revenue {
      DetailRow(title: "Revenue", value: revenue)
    }
  }
  .padding()
  .background(Color(.systemGray6))
  .cornerRadius(12)
}

// MARK: Rating breakdown
VStack(alignment: .leading, spacing: 10) {
  Text("Rating Breakdown").font(.headline).fontWeight(.semibold)
  VStack(spacing: 8) {
    HStack {
      Text("User Score"); Spacer()
      Text("\${movie.ratingPercentage}%").fontWeight(.semibold)
    }
    ProgressView(value: movie.voteAverage, total: 10.0)
      .progressViewStyle(LinearProgressViewStyle(tint: .green))
    HStack {
```

```

        Text("Total Votes"); Spacer()
        Text("\(movie.voteCount)").fontWeight(.semibold)
    }
}
.padding()
.background(Color(.systemGray6))
.cornerRadius(12)
}

// MARK: Similar Movies
if viewModel.model.similarMovies.isEmpty {
    VStack(alignment: .leading, spacing: 10) {
        Text("Similar Movies").font(.headline).fontWeight(.semibold)
        ScrollView(.horizontal, showsIndicators: false) {
            HStack(spacing: 16) {
                ForEach(viewModel.model.similarMovies) { similar in
                    NavigationLink(destination: MovieDetailView(movie: similar)) {
                        MovieCard(movie: similar)
                    }
                }.buttonStyle(PlainButtonStyle())
            }
        }
    }
}

// Loading shimmer while detail is being fetched
if viewModel.isLoadingDetail {
    HStack {
        Spacer()
        ProgressView().scaleEffect(0.9)
        Spacer()
    }
}
.padding(16)
}
.navigationBarHidden(true)
.ignoresSafeArea(edges: .top)
.task { await viewModel.loadDetails() }
}

// MARK: - Supporting views

private struct ActionButton: View {
    let icon: String
    let label: String
    let active: Bool

```

```

let color: Color
let action: () -> Void

var body: some View {
    Button(action: action) {
        HStack {
            Image(systemName: icon)
            Text(label)
        }
        .font(.subheadline).fontWeight(.semibold)
        .foregroundColor(active ? .white : color)
        .padding(.horizontal, 16).padding(.vertical, 10)
        .background(active ? color : color.opacity(0.1))
        .cornerRadius(20)
    }
}

private struct CastCard: View {
    let member: CastMember

    var body: some View {
        VStack(spacing: 6) {
            AsyncImage(url: member.profileURL) { image in
                image.resizable().aspectRatio(contentMode: .fill)
            } placeholder: {
                Circle()
                    .fill(Color.gray.opacity(0.3))
                    .overlay(Image(systemName: "person.fill").foregroundColor(.gray))
            }
            .frame(width: 70, height: 70)
            .clipShape(Circle())

            Text(member.name)
                .font(.caption).fontWeight(.semibold)
                .lineLimit(2).multilineTextAlignment(.center)
                .frame(width: 76)

            Text(member.character)
                .font(.caption2).foregroundColor(.secondary)
                .lineLimit(1).frame(width: 76)
        }
    }
}

struct DetailRow: View {
    let title: String
    let value: String

    var body: some View {

```

```
HStack {  
  Text(title).foregroundColor(.secondary)  
  Spacer()  
  Text(value).fontWeight(.medium)  
}  
.font(.subheadline)  
}
```

ProfileView

```
import SwiftUI

struct ProfileView: View {
    @StateObject private var viewModel = ProfileViewModel()
    @State private var showSignOutConfirmation = false
    @State private var isSigningOut = false

    var body: some View {
        NavigationView {
            ScrollView {
                VStack(spacing: 24) {
                    // Profile Header
                    VStack(spacing: 16) {
                        // Avatar — Google photo when available, initials fallback
                        Group {
                            if let photoURL = viewModel.userPhotoURL {
                                AsyncImage(url: photoURL) { phase in
                                    switch phase {
                                        case .success(let image):
                                            image
                                                .resizable()
                                                .scaledToFill()
                                                .frame(width: 100, height: 100)
                                                .clipShape(Circle())
                                    }
                                }
                                .default:
                                    initialsAvatar
                            }
                        }
                    }
                    .padding(.top)

                    VStack(spacing: 4) {
                        Text(viewModel.userName)
                            .font(.title2)
                            .fontWeight(.semibold)
                    }

                    // Stats
                    HStack(spacing: 40) {
                        StatView(title: "Watched", value: "\(viewModel.watchedCount)")
                        StatView(title: "Favorites", value: "\(viewModel.favoritesCount)")
                    }
                }
            }
        }
    }
}
```

```

    StatView(title: "Watchlist", value: "\(viewModel.watchlistCount)")
  }
  .padding(.horizontal)

  // Quick Actions
  VStack(spacing: 16) {
    SectionHeader(title: "My Movies")

    VStack(spacing: 12) {
      // Favorites Navigation
      NavigationLink(destination: FavoritesView()) {
        HStack(spacing: 16) {
          ZStack {
            RoundedRectangle(cornerRadius: 10)
              .fill(Color.red.opacity(0.1))
              .frame(width: 44, height: 44)

            Image(systemName: "heart.fill")
              .foregroundColor(.red)
              .font(.title3)
          }

          VStack(alignment: .leading, spacing: 2) {
            Text("My Favorites")
              .font(.headline)
              .foregroundColor(.primary)

            Text(viewModel.favoritesSubtitle)
              .font(.caption)
              .foregroundColor(.secondary)
          }

          Spacer()

          Image(systemName: "chevron.right")
            .foregroundColor(.secondary)
            .font(.caption)
        }
        .padding(.horizontal, 16)
        .padding(.vertical, 12)
        .background(Color(.systemBackground))
        .cornerRadius(12)
        .shadow(color: .black.opacity(0.05), radius: 2, x: 0, y: 1)
      }
      .buttonStyle(PlainButtonStyle())

      // Watchlist Navigation
      NavigationLink(destination: WatchlistView()) {
        HStack(spacing: 16) {
          ZStack {

```

```
        RoundedRectangle(cornerRadius: 10)
            .fill(Color.blue.opacity(0.1))
            .frame(width: 44, height: 44)

        Image(systemName: "bookmark.fill")
            .foregroundColor(.blue)
            .font(.title3)
    }

    VStack(alignment: .leading, spacing: 2) {
        Text("Watchlist")
            .font(.headline)
            .foregroundColor(.primary)

        Text(viewModel.watchlistSubtitle)
            .font(.caption)
            .foregroundColor(.secondary)
    }

    Spacer()

    Image(systemName: "chevron.right")
        .foregroundColor(.secondary)
        .font(.caption)
    }
    .padding(.horizontal, 16)
    .padding(.vertical, 12)
    .background(Color(.systemBackground))
    .cornerRadius(12)
    .shadow(color: .black.opacity(0.05), radius: 2, x: 0, y: 1)
}
.buttonStyle(PlainButtonStyle())

// Watched Movies Navigation
NavigationLink(destination: WatchedView()) {
    HStack(spacing: 16) {
        ZStack {
            RoundedRectangle(cornerRadius: 10)
                .fill(Color.green.opacity(0.1))
                .frame(width: 44, height: 44)

            Image(systemName: "eye.fill")
                .foregroundColor(.green)
                .font(.title3)
        }

        VStack(alignment: .leading, spacing: 2) {
            Text("Watched Movies")
                .font(.headline)
                .foregroundColor(.primary)
```

```
        Text(viewModel.watchedSubtitle)
            .font(.caption)
            .foregroundColor(.secondary)
    }

    Spacer()

    Image(systemName: "chevron.right")
        .foregroundColor(.secondary)
        .font(.caption)
    }
    .padding(.horizontal, 16)
    .padding(.vertical, 12)
    .background(Color(.systemBackground))
    .cornerRadius(12)
    .shadow(color: .black.opacity(0.05), radius: 2, x: 0, y: 1)
    }
    .buttonStyle(PlainButtonStyle())
    }
}

// Sign Out
Button {
    showSignOutConfirmation = true
} label: {
    HStack(spacing: 12) {
        ZStack {
            Circle()
                .fill(Color.red.opacity(0.15))
                .frame(width: 36, height: 36)
            Image(systemName: "rectangle.portrait.and.arrow.right")
                .font(.system(size: 15, weight: .semibold))
                .foregroundColor(.red)
        }

        Text("Sign Out")
            .font(.system(size: 16, weight: .semibold))
            .foregroundColor(.red)

        Spacer()

        Image(systemName: "chevron.right")
            .font(.caption)
            .foregroundColor(.red.opacity(0.5))
    }
    .padding(.horizontal, 16)
    .padding(.vertical, 14)
    .background(
        ZStack {
```

```

        RoundedRectangle(cornerRadius: 16, style: .continuous)
            .fill(Color.red.opacity(0.06))
        RoundedRectangle(cornerRadius: 16, style: .continuous)
            .stroke(Color.red.opacity(0.18), lineWidth: 1)
    }
}
)
}
.buttonStyle(PlainButtonStyle())
.padding(.top, 8)
        .confirmationDialog("Sign out of CineMax?", isPresented:
$showSignOutConfirmation, titleVisibility: .visible) {
    Button("Sign Out", role: .destructive) {
        withAnimation(.easeInOut(duration: 0.3)) { isSigningOut = true }
        DispatchQueue.main.asyncAfter(deadline: .now() + 0.35) {
            AuthManager.shared.signOut()
        }
    }
    Button("Cancel", role: .cancel) {}
} message: {
    Text("You'll need to sign in again with Google next time.")
}

// App Info
VStack(spacing: 8) {
    Text(viewModel.appVersion)
        .font(.caption)
        .foregroundColor(.secondary)

    Text(viewModel.appCredit)
        .font(.caption2)
        .foregroundColor(.secondary)
}
.padding(.top)
}
.padding(.horizontal, 16)
.padding(.vertical, 12)
}
.navigationTitle("Profile")
.navigationBarTitleDisplayMode(.large)
.overlay {
    if isSigningOut {
        Color.black.opacity(0.45)
            .ignoresSafeArea()
            .transition(.opacity)
            .overlay {
                VStack(spacing: 16) {
                    ProgressView()
                        .tint(.white)
                        .scaleEffect(1.4)
                    Text("Signing out...")
                }
            }
    }
}

```

```

        .font(.subheadline)
        .foregroundColor(.white.opacity(0.85))
    }
}
}
}
}
}

@ViewBuilder
private var initialsAvatar: some View {
    ZStack {
        Circle()
            .fill(LinearGradient(
                colors: [.red, .orange],
                startPoint: .topLeading,
                endPoint: .bottomTrailing
            ))
            .frame(width: 100, height: 100)
        Text(viewModel.userInitials)
            .font(.largeTitle)
            .fontWeight(.bold)
            .foregroundColor(.white)
    }
}

struct StatView: View {
    let title: String
    let value: String

    var body: some View {
        VStack(spacing: 4) {
            Text(value)
                .font(.title2)
                .fontWeight(.bold)
                .foregroundColor(.red)

            Text(title)
                .font(.caption)
                .foregroundColor(.secondary)
        }
    }
}

struct SectionHeader: View {
    let title: String

    var body: some View {
        HStack {

```

```
Text(title)
    .font(.headline)
    .fontWeight(.semibold)

Spacer()
}
.padding(.horizontal, 4)
}
```

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.

Διπλωματική Εργασία