



Σχολή Θετικών Επιστημών & Τεχνολογίας (ΣΘΕΤ)
Μεταπτυχιακή Εξειδίκευση στα Πληροφοριακά Συστήματα

Διπλωματική Εργασία

Έλεγχος Λογισμικού σε Web Εφαρμογές: Θεωρητική Προσέγγιση,
Manual Testing και Εισαγωγή στην Αυτοματοποίηση

Μαρία Μαρκάτου

Επιβλέπων καθηγητής: κος Γεώργιος Ρηγόπουλος

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.



Έλεγχος Λογισμικού σε Web Εφαρμογές: Θεωρητική Προσέγγιση, Χειροκίνητος Έλεγχος και Εισαγωγή στην Αυτοματοποίηση

Μαρία Μαρκάτου

Επιτροπή Επίβλεψης Διπλωματικής Εργασίας

Επιβλέπων Καθηγητής:

κος Γεώργιος Ρηγόπουλος

Μέλος ΣΕΠ, Ελληνικό Ανοικτό
Πανεπιστήμιο

Συν-Επιβλέπων Καθηγητής:

κος Ευστράτιος Γεωργόπουλος

Μέλος ΣΕΠ, Ελληνικό Ανοικτό
Πανεπιστήμιο

Πάτρα, Ιούνιος 2026

Με την ολοκλήρωση της παρούσας διπλωματικής εργασίας, αισθάνομαι την ανάγκη να εκφράσω την ειλικρινή μου ευγνωμοσύνη σε όλους όσους στάθηκαν δίπλα μου και συνέβαλαν, ο καθένας με τον δικό του τρόπο, στην επίτευξη αυτού του σημαντικού στόχου.

Πρωτίστως, θα ήθελα να εκφράσω τις θερμές μου ευχαριστίες στον επιβλέποντα καθηγητή μου για την εμπιστοσύνη που μου έδειξε αναλαμβάνοντας την επίβλεψη της εργασίας μου, καθώς και για την πολύτιμη καθοδήγηση, τις εύστοχες παρατηρήσεις, τις γνώσεις και τη συνεχή υποστήριξή του καθ' όλη τη διάρκεια της εκπόνησής της. Η συμβολή του υπήρξε καθοριστική για την ολοκλήρωση της παρούσας εργασίας.

Από καρδιάς ευχαριστώ την οικογένειά μου – τους γονείς μου, τις αδελφές μου και τον σύζυγό μου – για την αστείρευτη αγάπη, την κατανόηση, την ενθάρρυνση και την αδιάκοπη στήριξή τους. Ήταν πάντα δίπλα μου, δίνοντάς μου δύναμη να συνεχίζω, να ξεπερνώ τις δυσκολίες και να πιστεύω στις δυνατότητές μου. Επίσης, θα ήθελα να ευχαριστήσω και τα αγαπημένα μου σκυλάκια, που με τον δικό τους μοναδικό τρόπο γέμισαν και συνεχίζουν να γεμίζουν τη ζωή μου με αγάπη, ζεστασιά και αμέτρητες όμορφες στιγμές.

Η ολοκλήρωση αυτής της διπλωματικής εργασίας αποτελεί για εμένα ένα σημαντικό προσωπικό και ακαδημαϊκό ορόσημο. Δεν θα είχε, όμως, την ίδια αξία χωρίς την αγάπη, την ενθάρρυνση και την παρουσία όλων όσων κάνουν τη ζωή μου κάθε μέρα πιο όμορφη και πιο γεμάτη. Νιώθω βαθιά ευγνωμοσύνη και πραγματικά τυχερή για όλους όσους έχουν αφήσει και συνεχίζουν να αφήνουν το αποτύπωμά τους στη ζωή μου.

Σας ευχαριστώ όλους από καρδιάς.

Στην οικογένειά μου, με όλη μου την αγάπη.

Περίληψη

Ο έλεγχος λογισμικού αποτελεί κρίσιμο στάδιο του κύκλου ζωής ανάπτυξης εφαρμογών, καθώς διασφαλίζει την ποιότητα, την αξιοπιστία και τη λειτουργική ορθότητα των πληροφοριακών συστημάτων. Η συνεχής ανάπτυξη και η αυξανόμενη πολυπλοκότητα των εφαρμογών ιστού καθιστούν αναγκαία την αξιοποίηση αποτελεσματικών τεχνικών ελέγχου, τόσο χειροκίνητων (manual) όσο και αυτοματοποιημένων (automated).

Αντικείμενο της παρούσας διπλωματικής εργασίας είναι η μελέτη και η συγκριτική αξιολόγηση του χειροκίνητου και του αυτοματοποιημένου ελέγχου λογισμικού σε εφαρμογές ιστού. Παρουσιάζονται οι βασικές αρχές, οι κατηγορίες και οι τεχνικές δοκιμών, καθώς και τα χαρακτηριστικά, τα πλεονεκτήματα και οι περιορισμοί του manual και του automated testing.

Για την πρακτική αξιολόγηση χρησιμοποιήθηκε η εφαρμογή SauceDemo ως μελέτη περίπτωσης. Σχεδιάστηκαν και εκτελέστηκαν σενάρια δοκιμών για βασικές λειτουργίες της εφαρμογής, όπως η σύνδεση χρήστη, η διαχείριση του καλαθιού αγορών και η ολοκλήρωση αγοράς. Οι χειροκίνητες δοκιμές πραγματοποιήθηκαν με προκαθορισμένα test cases, ενώ οι αυτοματοποιημένες υλοποιήθηκαν με τη γλώσσα προγραμματισμού Java και το εργαλείο Selenium WebDriver.

Τα αποτελέσματα έδειξαν ότι το automated testing υπερτερεί ως προς την ταχύτητα, τη συνέπεια και τη δυνατότητα επαναληψιμότητας των δοκιμών, ενώ το manual testing αποδείχθηκε ιδιαίτερα αποτελεσματικό στον εντοπισμό μη αναμενόμενων σφαλμάτων και προβλημάτων χρηστικότητας. Συμπερασματικά, η αποτελεσματικότερη στρατηγική ελέγχου λογισμικού προκύπτει από τον συνδυασμό των δύο προσεγγίσεων, ανάλογα με τις απαιτήσεις του εκάστοτε έργου.

Λέξεις – Κλειδιά

Έλεγχος λογισμικού, Χειροκίνητος έλεγχος λογισμικού, Αυτοματοποιημένος έλεγχος λογισμικού, Εφαρμογές ιστού, Selenium WebDriver, Διασφάλιση ποιότητας.

Software Testing in Web Applications: A Theoretical Approach, Manual Testing and Introduction to Test Automation

Maria Markatou

Abstract

Software testing is a critical phase of the software development life cycle, as it ensures the quality, reliability, and functional correctness of information systems. The continuous growth and increasing complexity of web applications have made the use of effective testing techniques, both manual and automated, essential.

The purpose of this thesis is to study and comparatively evaluate manual and automated software testing in web applications. It presents the fundamental principles, categories, and testing techniques, as well as the characteristics, advantages, and limitations of both manual and automated testing.

For the practical evaluation, the SauceDemo web application was selected as a case study. Test scenarios were designed and executed to evaluate core application functionalities, including user authentication, shopping cart management, and the checkout process. Manual testing was performed using predefined test cases, while automated testing was implemented using the Java programming language and the Selenium WebDriver.

The results demonstrated that automated testing outperforms manual testing in terms of execution speed, consistency, and test repeatability. In contrast, manual testing proved to be particularly effective in identifying unexpected defects and usability issues. In conclusion, the most effective software testing strategy is achieved by combining both approaches, leveraging the strengths of each according to the specific requirements of the project.

Keywords

Software Testing, Manual Software Testing, Automated Software Testing, Web Applications, Selenium WebDriver, Quality Assurance.

Περιεχόμενα

Περίληψη.....	v
Abstract.....	vi
Περιεχόμενα.....	vii
Κατάλογος Εικόνων / Σχημάτων.....	x
Κατάλογος Πινάκων.....	xi
Συντομογραφίες & Ακρωνύμια.....	xii
1 Εισαγωγή.....	1
1.1 Σκοπός της Διπλωματικής Εργασίας.....	1
1.2 Αντικείμενο και Στόχοι της Εργασίας.....	1
1.3 Μεθοδολογία της Εργασίας.....	2
1.4 Δομή της Διπλωματικής Εργασίας.....	3
2 Θεμελιώδεις Έννοιες Ελέγχου Λογισμικού.....	4
2.1 Εισαγωγή στον Έλεγχο Λογισμικού.....	4
2.2 Ο Ρόλος του Ελέγχου Λογισμικού στη Διασφάλιση Ποιότητας.....	5
2.3 Έννοιες Error, Defect και Failure στον Έλεγχο Λογισμικού.....	6
2.4 Δραστηριότητες και Διαδικασία Ελέγχου Λογισμικού.....	7
3 Στατικός και Δυναμικός Έλεγχος Λογισμικού.....	9
3.1 Εισαγωγή στον Στατικό και Δυναμικό Έλεγχο Λογισμικού.....	9
3.2 Στατικός Έλεγχος.....	10
3.3 Δυναμικός Έλεγχος.....	11
3.4 Σύγκριση Στατικού και Δυναμικού Ελέγχου.....	13
3.5 Η Προσέγγιση Shift-Left Testing.....	14
4 Ποιότητα Λογισμικού και Διασφάλιση Ποιότητας σε Web Εφαρμογές.....	16
4.1 Η Έννοια της Ποιότητας Λογισμικού.....	16
4.2 Το Πρότυπο ISO/IEC 25010 και η Αξιολόγηση Ποιότητας.....	17
4.3 Λειτουργική Καταλληλότητα και Αξιοπιστία στις Web Εφαρμογές.....	18
4.4 Μη Λειτουργικά Χαρακτηριστικά Ποιότητας στις Web Εφαρμογές.....	20
4.5 Η Σχέση του ISO/IEC 25010 με τον Έλεγχο Λογισμικού.....	22
5 Ο Έλεγχος Λογισμικού στον Κύκλο Ζωής Ανάπτυξης Λογισμικού.....	24
5.1 Ο Κύκλος Ζωής Ανάπτυξης Λογισμικού (SDLC).....	24
5.2 Παραδοσιακά Μοντέλα Ανάπτυξης και Έλεγχος Λογισμικού.....	25
5.3 Agile Μεθοδολογίες και Εξέλιξη του Ελέγχου Λογισμικού.....	26
5.4 DevOps και Continuous Integration στις Διαδικασίες Testing.....	28
5.5 Ο Ρόλος του Testing στις Σύγχρονες Διαδικασίες Ανάπτυξης.....	29
6 Τεχνικές και Στρατηγικές Ελέγχου Λογισμικού.....	31
6.1 Εισαγωγή στις Τεχνικές Ελέγχου Λογισμικού.....	31
6.2 Black-Box Testing Techniques.....	32
6.3 White-Box Testing Techniques.....	34
6.4 Experience-Based Testing Techniques.....	35
6.5 Η Σημασία των Τεχνικών Ελέγχου στις Σύγχρονες Web Εφαρμογές.....	37
7 Διαδικασίες και Οργάνωση Ελέγχου Λογισμικού.....	39
7.1 Ο Σχεδιασμός Διαδικασιών Testing.....	39
7.2 Test Scenarios και Test Cases.....	40
7.3 Διαχείριση Defects και Αναφορά Αποτελεσμάτων Testing.....	42
7.4 Manual Testing Διαδικασίες.....	43
8 Automated Testing στις Web Εφαρμογές.....	46

8.1	Εισαγωγή στο Automated Testing.....	46
8.2	Πλεονεκτήματα και Περιορισμοί του Automated Testing.....	47
8.3	Frameworks και Εργαλεία Automated Testing.....	48
8.4	Selenium WebDriver.....	49
8.5	Ο Ρόλος του Automated Testing στις Σύγχρονες Web Εφαρμογές.....	50
9	Περιγραφή της Εφαρμογής (Case Study).....	52
9.1	Εισαγωγή.....	52
9.2	Γενική Περιγραφή της Εφαρμογής.....	52
9.3	Αρχιτεκτονική και Δομή της Εφαρμογής.....	53
9.4	Βασικές Λειτουργίες της Εφαρμογής.....	54
	9.4.1 Σύστημα Σύνδεσης Χρηστών (Login System).....	54
	9.4.2 Προβολή Προϊόντων (Product Inventory).....	54
	9.4.3 Καλάθι Αγορών (Shopping Cart).....	54
	9.4.4 Διαδικασία Ολοκλήρωσης Αγοράς (Checkout).....	55
9.5	Καταλληλότητα της Εφαρμογής για Testing.....	55
9.6	Συμπεράσματα.....	56
10	Περιβάλλον Δοκιμών και Μεθοδολογία Testing.....	57
10.1	Εισαγωγή.....	57
10.2	Περιβάλλον Δοκιμών (Testing Environment).....	57
10.3	Τύπος Δοκιμών.....	58
	10.3.1 Σύγκριση Manual και Automated Testing.....	58
10.4	Προσέγγιση Black Box Testing.....	59
10.5	Διαδικασία Σχεδιασμού Test Cases.....	59
10.6	Διαδικασία Εκτέλεσης Δοκιμών.....	60
10.7	Καταγραφή Σφαλμάτων (Defect Reporting).....	60
10.8	Εργαλεία Υποστήριξης Testing.....	61
10.9	Συμπεράσματα.....	61
11	Σχεδιασμός Test Scenarios και Test Cases.....	63
11.1	Εισαγωγή.....	63
	11.1.1 Μεθοδολογική Προσέγγιση Σχεδιασμού Δοκιμών.....	63
11.2	Test Scenarios.....	64
11.3	Συνοπτικός Πίνακας Test Cases.....	65
11.4	Αναλυτικά Test Cases.....	67
	11.4.1 Login Functionality Test Cases.....	67
	11.4.2 Product Inventory Test Cases.....	70
	11.4.3 Shopping Cart Functionality Test Cases.....	73
	11.4.4 Checkout Process Test Cases.....	75
	11.4.5 Navigation and Session Test Cases.....	79
	11.4.6 Input Field Validation Test Cases.....	81
11.5	Συμπεράσματα.....	82
12	Εκτέλεση Manual Testing και Αποτελέσματα.....	84
12.1	Εισαγωγή.....	84
12.2	Αποτελέσματα Εκτέλεσης Δοκιμών.....	84
12.3	Στιγμιότυπα Εκτέλεσης Δοκιμών.....	86
12.4	Αναφορά Σφαλμάτων (Defect Reports).....	92
	12.4.1 Defect 01 – Δυσλειτουργία εικονιδίου διαγραφής (Clear Icon).....	92
	12.4.2 Defect 02 – Ολοκλήρωση παραγγελίας με κενό καλάθι.....	93
12.5	Exploratory Testing.....	95
	12.5.1 Exploratory Test 01 – Ανανέωση σελίδας κατά τη διαδικασία checkout.....	95

12.5.2	Exploratory Test 02 – Χρήση του κουμπιού επιστροφής (Back) του περιηγητή.....	96
12.5.3	Συμπεράσματα Exploratory Testing.....	96
12.6	Μετρικές Δοκιμών.....	97
12.7	Οπτικοποίηση Αποτελεσμάτων Δοκιμών.....	98
12.8	Ανάλυση Αποτελεσμάτων.....	100
12.9	Συμβολή της Παρούσας Εργασίας.....	101
12.10	Συμπεράσματα.....	102
13	Αυτοματοποιημένες Δοκιμές και Συγκριτική Αξιολόγηση.....	103
13.1	Εισαγωγή.....	103
13.2	Μεθοδολογική Προσέγγιση.....	104
13.3	Περιβάλλον Υλοποίησης και Εργαλεία.....	105
13.3.1	Τεχνολογική Υποδομή.....	105
13.3.2	Ζητήματα Υλοποίησης και Εκτέλεσης.....	106
13.3.3	Συντήρηση και Επεκτασιμότητα.....	107
13.4	Υλοποίηση Αυτοματοποιημένων Δοκιμών.....	108
13.4.1	Αναλυτική Εκτέλεση Test Scenario: Επιτυχής Σύνδεση Χρήστη.....	109
13.4.2	Αναλυτική Εκτέλεση Test Scenario: Αποτυχημένη Σύνδεση.....	111
13.4.3	Αναλυτική Εκτέλεση Test Scenario: Προσθήκη Προϊόντος στο Καλάθι...113	
13.4.4	Αναλυτική Εκτέλεση Test Scenario: Αφαίρεση Προϊόντος από το Καλάθι 115	
13.4.5	Αναλυτική Εκτέλεση Σεναρίου Δοκιμής: Διαδικασία Checkout.....	117
13.5	Ανάλυση Εντοπισμένων Defects.....	119
13.6	Συγκριτική Ανάλυση και Συζήτηση.....	122
13.6.1	Ο Ρόλος του Automated Testing σε Σύγχρονα Περιβάλλοντα DevOps.....	124
13.7	Συμπεράσματα.....	125
13.8	Συνολική Αξιολόγηση των Δύο Προσεγγίσεων.....	126
13.9	Ποσοτική Αξιολόγηση Αποτελεσμάτων.....	127
13.10	Περιορισμοί της Μελέτης.....	128
13.11	Τελική Αξιολόγηση.....	130
13.12	Μελλοντική Εργασία.....	131
	Βιβλιογραφία.....	133
	Παράρτημα Α: Πλήρης Κώδικας Αυτοματοποιημένων Δοκιμών.....	136

Κατάλογος Εικόνων / Σχημάτων

Εικόνα 12.1.Σελίδα σύνδεσης χρήστη.....	87
Εικόνα 12.2.Προσθήκη προϊόντος στο καλάθι αγορών.....	88
Εικόνα 12.3.Προβολή καλαθιού αγορών.....	88
Εικόνα 12.4.Διαδικασία checkout με κενό καλάθι αγορών.....	89
Εικόνα 12.5.Επιτυχής ολοκλήρωση checkout με κενό καλάθι.....	89
Εικόνα 12.6.Σελίδα επισκόπησης παραγγελίας.....	90
Εικόνα 12.7.Επιτυχής ολοκλήρωση παραγγελίας.....	90
Εικόνα 12.8.Εντοπισμός σφάλματος στη φόρμα σύνδεσης.....	91
Εικόνα 12.9.Εντοπισμός σφάλματος στη φόρμα checkout.....	91
Εικόνα 13.1: Αυτόματη συμπλήρωση στοιχείων σύνδεσης μέσω Selenium WebDriver. 110	
Εικόνα 13.2: Επιτυχής εκτέλεση του αυτοματοποιημένου σεναρίου στην κονσόλα του IntelliJ (TEST PASSED).....	111
Εικόνα 13.3: Αποτέλεσμα εκτέλεσης του αυτοματοποιημένου σεναρίου στην κονσόλα του IntelliJ (TEST PASSED).....	113
Εικόνα 13.4: Αποτέλεσμα εκτέλεσης του αυτοματοποιημένου σεναρίου στην κονσόλα του IntelliJ (TEST PASSED).....	115
Εικόνα 13.5: Αποτέλεσμα εκτέλεσης του αυτοματοποιημένου σεναρίου στην κονσόλα του IntelliJ (TEST PASSED).....	117
Εικόνα 13.6: Αποτέλεσμα εκτέλεσης του αυτοματοποιημένου σεναρίου στην κονσόλα του IntelliJ (TEST PASSED).....	119
Σχήμα 1: Κατανομή αποτελεσμάτων δοκιμών (Pass / Fail).....	98
Σχήμα 2: Κατανομή test cases ανά λειτουργική ενότητα.....	99
Σχήμα 3: Κατανομή των εντοπισμένων σφαλμάτων ανά λειτουργική ενότητα της εφαρμογής.....	100

Κατάλογος Πινάκων

Πίνακας 8.1: Ενδεικτικά Frameworks Automated Testing.....	48
Πίνακας 11.1: Κατηγορίες Test Scenarios.....	65
Πίνακας 11.2: Test Cases της Εφαρμογής και Αντιστοίχισή τους με τα Test Scenarios....	67
Πίνακας 11.3: Test Case TC01.....	68
Πίνακας 11.4: Test Case TC02.....	68
Πίνακας 11.5: Test Case TC03.....	68
Πίνακας 11.6: Test Case TC04.....	69
Πίνακας 11.7: Test Case TC05.....	69
Πίνακας 11.8: Test Case TC06.....	69
Πίνακας 11.9: Test Case TC07.....	70
Πίνακας 11.10: Test Case TC08.....	71
Πίνακας 11.11: Test Case TC09.....	71
Πίνακας 11.12: Test Case TC10.....	71
Πίνακας 11.13: Test Case TC11.....	72
Πίνακας 11.14: Test Case TC12.....	72
Πίνακας 11.15: Test Case TC13.....	72
Πίνακας 11.16: Test Case TC14.....	73
Πίνακας 11.17: Test Case TC15.....	73
Πίνακας 11.18: Test Case TC16.....	74
Πίνακας 11.19: Test Case TC17.....	74
Πίνακας 11.20: Test Case TC18.....	74
Πίνακας 11.21: Test Case TC19.....	75
Πίνακας 11.22: Test Case TC20.....	76
Πίνακας 11.23: Test Case TC21.....	76
Πίνακας 11.24: Test Case TC22.....	76
Πίνακας 11.25: Test Case TC23.....	77
Πίνακας 11.26: Test Case TC24.....	77
Πίνακας 11.27: Test Case TC25.....	77
Πίνακας 11.28: Test Case TC26.....	78
Πίνακας 11.29: Test Case TC27.....	78
Πίνακας 11.30: Test Case TC28.....	78
Πίνακας 11.31: Test Case TC29.....	79
Πίνακας 11.32: Test Case TC30.....	79
Πίνακας 11.33: Test Case TC31.....	80
Πίνακας 11.34: Test Case TC32.....	80
Πίνακας 11.35: Test Case TC33.....	80
Πίνακας 11.36: Test Case TC34.....	81
Πίνακας 11.37: Test Case TC35.....	82
Πίνακας 12.1: Αποτελέσματα Εκτέλεσης Test Cases.....	86
Πίνακας 12.2: Συνοπτικές Μετρικές Δοκιμών.....	98
Πίνακας 13.1: Εργαλεία και Τεχνολογίες Υλοποίησης.....	106
Πίνακας 13.2: Σύγκριση Χειροκίνητου και Αυτοματοποιημένου Ελέγχου.....	123
Πίνακας 13.3: Εντοπισμός Defects ανά Μέθοδο Ελέγχου.....	127

Συντομογραφίες & Ακρωνύμια

ΔΕ	Διπλωματική Εργασία
ΕΑΠ	Ελληνικό Ανοικτό Πανεπιστήμιο
API	Application Programming Interface (Διεπαφή Προγραμματισμού Εφαρμογών)
CI	Continuous Integration
CI/CD	Continuous Integration / Continuous Delivery
DOM	Document Object Model
ID	Identifier
IEC	International Electrotechnical Commission
IEEE	Institute of Electrical and Electronics Engineers
ISO	International Organization for Standardization
ISTQB	International Software Testing Qualifications Board
JUnit	Java Unit
POM	Page Object Model
QA	Quality Assurance
SDLC	Software Development Life Cycle
SQA	Software Quality Assurance
SQL	Structured Query Language
SquaRE	Software Quality Requirements and Evaluation
TC	Test Case
TestNG	Testing Next Generation
TS	Test Scenario
UI	User Interface
URL	Uniform Resource Locator
XSS	Cross-Site Scripting

1 Εισαγωγή

1.1 Σκοπός της Διπλωματικής Εργασίας

Ο έλεγχος λογισμικού (Software Testing) αποτελεί βασική διαδικασία διασφάλισης ποιότητας κατά την ανάπτυξη σύγχρονων πληροφοριακών συστημάτων (Beizer, 1990). Η συνεχής εξέλιξη των web εφαρμογών (web applications), σε συνδυασμό με τις αυξημένες απαιτήσεις αξιοπιστίας, ασφάλειας και διαθεσιμότητας, έχει ενισχύσει σημαντικά τον ρόλο του testing στις διαδικασίες ανάπτυξης λογισμικού (Pressman & Maxim, 2020). Παράλληλα, η ευρεία χρήση web τεχνολογιών σε επιχειρησιακά και οργανωτικά περιβάλλοντα καθιστά απαραίτητη την εφαρμογή αποτελεσματικών διαδικασιών ελέγχου για τη διατήρηση της σταθερότητας και της λειτουργικότητας των εφαρμογών (Kaner, Bach, & Pettichord, 2002).

Οι σύγχρονες web εφαρμογές λειτουργούν σε ιδιαίτερα δυναμικά περιβάλλοντα, όπου διαφορετικοί browsers, λειτουργικά συστήματα, συσκευές και συνθήκες δικτύου επηρεάζουν άμεσα τη συμπεριφορά και την εμπειρία χρήστη (Nielsen, 1994). Επιπλέον, η υιοθέτηση Agile και DevOps πρακτικών έχει μεταβάλει ουσιαστικά τον τρόπο ανάπτυξης λογισμικού, αυξάνοντας την ανάγκη για συνεχές και συστηματικό testing σε όλη τη διάρκεια του κύκλου ζωής ανάπτυξης ενός συστήματος (Crispin & Gregory, 2014).

Στο πλαίσιο αυτό, η παρούσα διπλωματική εργασία εξετάζει τις διαδικασίες ελέγχου λογισμικού σε web εφαρμογές, με έμφαση στις προσεγγίσεις manual testing (χειροκίνητος έλεγχος) και automated testing (αυτοματοποιημένος έλεγχος) (ISO/IEC/IEEE 29119-1:2022). Βασικός σκοπός της εργασίας είναι η διερεύνηση της συμβολής των διαδικασιών testing στη βελτίωση της ποιότητας και αξιοπιστίας των web εφαρμογών, καθώς και η ανάδειξη του ρόλου τους στις σύγχρονες διαδικασίες ανάπτυξης λογισμικού (Sommerville, 2016).

1.2 Αντικείμενο και Στόχοι της Εργασίας

Το αντικείμενο της παρούσας διπλωματικής εργασίας αφορά τη μελέτη διαδικασιών testing σε web εφαρμογές, με έμφαση στις διαδικασίες manual testing και στις βασικές πρακτικές αυτοματοποίησης δοκιμών. Η εργασία εξετάζει τόσο το θεωρητικό πλαίσιο του ελέγχου

λογισμικού όσο και την πρακτική εφαρμογή διαδικασιών ελέγχου σε πραγματικό περιβάλλον web εφαρμογής (Pressman & Maxim, 2020).

Σε θεωρητικό επίπεδο αναλύονται οι βασικές έννοιες και τεχνικές testing, ο ρόλος του testing στον κύκλο ζωής ανάπτυξης λογισμικού και οι σύγχρονες προσεγγίσεις testing που εφαρμόζονται σε web εφαρμογές. Παράλληλα, εξετάζονται οι ιδιαιτερότητες που παρουσιάζουν οι web εφαρμογές ως προς τις απαιτήσεις ελέγχου και αξιολόγησης ποιότητας (ISTQB, 2023).

Το πρακτικό μέρος της εργασίας βασίζεται στον σχεδιασμό και την εκτέλεση περιπτώσεων δοκιμής (test cases) και σεναρίων δοκιμής (test scenarios) μέσω διαδικασιών manual testing και automated testing. Στο πλαίσιο αυτό πραγματοποιείται εφαρμογή automated tests μέσω Selenium WebDriver και καταγραφή των αποτελεσμάτων που προκύπτουν από την εκτέλεση των δοκιμών (Myers et al., 2011).

Βασικός στόχος της εργασίας είναι η συγκριτική αξιολόγηση των δύο προσεγγίσεων testing ως προς την αποτελεσματικότητα, την επαναληψιμότητα, τον χρόνο εκτέλεσης και τη συνολική υποστήριξη των διαδικασιών ελέγχου λογισμικού σε web εφαρμογές (Sommerville, 2016).

1.3 Μεθοδολογία της Εργασίας

Η μεθοδολογία της παρούσας διπλωματικής εργασίας βασίζεται σε συνδυασμό θεωρητικής μελέτης και πρακτικής εφαρμογής διαδικασιών testing σε web εφαρμογή. Η προσέγγιση αυτή επιτρέπει τη διερεύνηση τόσο των βασικών αρχών ελέγχου λογισμικού όσο και της πρακτικής εφαρμογής διαδικασιών manual και automated testing σε πραγματικό περιβάλλον λειτουργίας (Pressman & Maxim, 2020).

Σε αρχικό στάδιο πραγματοποιείται βιβλιογραφική ανασκόπηση σχετική με τον έλεγχο λογισμικού, τις τεχνικές ελέγχου λογισμικού και τις σύγχρονες πρακτικές testing που εφαρμόζονται στις web εφαρμογές. Η θεωρητική μελέτη βασίζεται σε διεθνή πρότυπα, επιστημονικά συγγράμματα και ακαδημαϊκές πηγές σχετικές με τη μηχανική λογισμικού και τη διασφάλιση ποιότητας (Sommerville, 2016).

Στη συνέχεια πραγματοποιείται σχεδιασμός test cases και test scenarios για την αξιολόγηση λειτουργιών της web εφαρμογής που χρησιμοποιείται ως μελέτη περίπτωσης. Ακολουθεί η

εκτέλεση manual testing, καθώς και η υλοποίηση automated tests μέσω Selenium WebDriver για την αυτοματοποίηση επιλεγμένων test scenarios (ISTQB, 2023).

Κατά τη διάρκεια της πρακτικής διαδικασίας πραγματοποιείται καταγραφή των αποτελεσμάτων testing και των σφαλμάτων (defects) που προκύπτουν από την εκτέλεση των δοκιμών. Τέλος, πραγματοποιείται συγκριτική αξιολόγηση των δύο προσεγγίσεων testing ως προς την αποτελεσματικότητα, την επαναληψιμότητα και τον χρόνο εκτέλεσης (Myers et al., 2011).

1.4 Δομή της Διπλωματικής Εργασίας

Η παρούσα διπλωματική εργασία οργανώνεται σε δύο βασικά μέρη, το θεωρητικό και το πρακτικό, ώστε να επιτυγχάνεται η σύνδεση της θεωρητικής ανάλυσης με την πρακτική εφαρμογή διαδικασιών testing σε web εφαρμογές.

Στο θεωρητικό μέρος παρουσιάζονται οι βασικές έννοιες και τεχνικές του ελέγχου λογισμικού, οι διαδικασίες διασφάλισης ποιότητας (quality assurance) λογισμικού και οι σύγχρονες προσεγγίσεις manual και automated testing που εφαρμόζονται στις web εφαρμογές (Pressman & Maxim, 2020).

Το πρακτικό μέρος επικεντρώνεται στην εφαρμογή διαδικασιών testing σε web εφαρμογή μέσω σχεδιασμού και εκτέλεσης test cases και test scenarios. Παράλληλα, παρουσιάζονται τα αποτελέσματα που προκύπτουν από τις διαδικασίες manual και automated testing και πραγματοποιείται συγκριτική αξιολόγηση των δύο προσεγγίσεων (ISTQB, 2023).

Η εργασία ολοκληρώνεται με την παρουσίαση των βασικών συμπερασμάτων και τη διατύπωση προτάσεων για πιθανή μελλοντική επέκταση και βελτίωση των διαδικασιών testing σε σύγχρονες web εφαρμογές (Sommerville, 2016).

2 Θεμελιώδεις Έννοιες Ελέγχου Λογισμικού

2.1 Εισαγωγή στον Έλεγχο Λογισμικού

Ο έλεγχος λογισμικού (Software Testing) αποτελεί θεμελιώδη δραστηριότητα της μηχανικής λογισμικού και εντάσσεται στις διαδικασίες διασφάλισης ποιότητας ενός πληροφοριακού συστήματος (Beizer, 1990). Η συνεχής εξέλιξη των τεχνολογιών λογισμικού, η αυξανόμενη πολυπλοκότητα των εφαρμογών και η εκτεταμένη χρήση web τεχνολογιών έχουν αναδείξει το testing σε αναπόσπαστο στοιχείο του κύκλου ζωής ανάπτυξης λογισμικού, (Pressman & Maxim, 2020).

Ιδιαίτερα στις σύγχρονες web εφαρμογές, η ανάγκη για αποτελεσματικές διαδικασίες testing είναι αυξημένη, καθώς οι εφαρμογές λειτουργούν σε δυναμικά και ετερογενή περιβάλλοντα. Η υποστήριξη πολλαπλών περιηγητών ιστού (browsers), λειτουργικών συστημάτων και συσκευών, σε συνδυασμό με τις αυξημένες απαιτήσεις αξιοπιστίας, διαθεσιμότητας και εμπειρίας χρήστη, καθιστούν απαραίτητη τη συστηματική αξιολόγηση της λειτουργικότητας και της σταθερότητας των εφαρμογών καθ' όλη τη διάρκεια ανάπτυξης και συντήρησής τους (Sommerville, 2016).

Σύμφωνα με τον Sommerville (2016), ο έλεγχος λογισμικού αποτελεί διαδικασία αξιολόγησης και επαλήθευσης της συμπεριφοράς ενός συστήματος με στόχο τον εντοπισμό αποκλίσεων από τις απαιτήσεις και τις τεχνικές προδιαγραφές του λογισμικού. Παράλληλα, το ISTQB (2023) προσεγγίζει το testing όχι μόνο ως διαδικασία εντοπισμού defects, αλλά και ως μηχανισμό παροχής πληροφόρησης σχετικά με την ποιότητα, την αξιοπιστία και τη συνολική συμπεριφορά του συστήματος κατά την εκτέλεσή του.

Η σημασία του testing δεν περιορίζεται αποκλειστικά στον εντοπισμό σφαλμάτων (Kaner, Bach, & Pettichord, 2002). Οι διαδικασίες testing λειτουργούν ως μηχανισμός αξιολόγησης της συνολικής ποιότητας του λογισμικού και συμβάλλουν στη μείωση τεχνικών και επιχειρησιακών κινδύνων πριν από την παραγωγική λειτουργία ενός συστήματος. Μέσω της συστηματικής εφαρμογής διαδικασιών ελέγχου λογισμικού, οι ομάδες ανάπτυξης αποκτούν τη δυνατότητα να αξιολογούν τη σταθερότητα, τη λειτουργικότητα και την αξιοπιστία μιας εφαρμογής, βελτιώνοντας παράλληλα την ποιότητα της εμπειρίας χρήστη και τη συνολική αποτελεσματικότητα του συστήματος (Pressman & Maxim, 2020).

2.2 Ο Ρόλος του Ελέγχου Λογισμικού στη Διασφάλιση Ποιότητας

Η αναγκαιότητα του ελέγχου λογισμικού συνδέεται άμεσα με την αυξανόμενη πολυπλοκότητα των σύγχρονων πληροφοριακών συστημάτων και τις επιπτώσεις που μπορεί να προκαλέσει μια αποτυχία λογισμικού. Οι σύγχρονες εφαρμογές καλούνται να λειτουργούν σε απαιτητικά και διαρκώς μεταβαλλόμενα περιβάλλοντα, υποστηρίζοντας μεγάλο αριθμό χρηστών, διαφορετικές τεχνολογικές πλατφόρμες και σύνθετες επιχειρησιακές λειτουργίες. Στο πλαίσιο αυτό, ακόμη και περιορισμένα defects μπορούν να οδηγήσουν σε σημαντικές δυσλειτουργίες, οικονομικές απώλειες ή προβλήματα αξιοπιστίας ενός οργανισμού (Sommerville, 2016).

Ένας από τους βασικότερους στόχους του testing είναι ο έγκαιρος εντοπισμός προβλημάτων πριν από την παραγωγική λειτουργία του συστήματος. Σύμφωνα με τους Myers, Sandler και Badgett (2011), σκοπός του testing δεν είναι η απόδειξη ότι το λογισμικό είναι πλήρως απαλλαγμένο από σφάλματα, αλλά η αποκάλυψη όσο το δυνατόν περισσότερων defects ώστε να μειωθεί η πιθανότητα αποτυχίας κατά τη χρήση του συστήματος. Πέρα από τον εντοπισμό defects, ο έλεγχος λογισμικού υποστηρίζει και ευρύτερες διαδικασίες αξιολόγησης και διασφάλισης ποιότητας λογισμικού.

Παράλληλα, το testing υποστηρίζει τις διαδικασίες επαλήθευσης (verification) και επικύρωσης (validation) του λογισμικού (ISO/IEC/IEEE 29119-1, 2022). Η επαλήθευση αφορά τον έλεγχο ότι το σύστημα αναπτύσσεται σύμφωνα με τις τεχνικές προδιαγραφές και τις απαιτήσεις σχεδίασης, ενώ η επικύρωση εξετάζει κατά πόσο το τελικό προϊόν ανταποκρίνεται στις πραγματικές ανάγκες και προσδοκίες των χρηστών (Pressman & Maxim, 2020).

Η σημασία του ελέγχου λογισμικού γίνεται ακόμη πιο εμφανής στις web εφαρμογές, όπου η λειτουργία του συστήματος επηρεάζεται από διαφορετικούς περιηγητές ιστού, λειτουργικά συστήματα, συσκευές και δικτυακές συνθήκες (ISO/IEC/IEEE 29119-1, 2022). Επιπλέον, οι συνεχείς ενημερώσεις λογισμικού και οι αυξημένες απαιτήσεις για γρήγορη παράδοση νέων λειτουργιών ενισχύουν την ανάγκη για συστηματικές και επαναλαμβανόμενες διαδικασίες ελέγχου.

Στις σύγχρονες Agile και DevOps προσεγγίσεις ανάπτυξης λογισμικού, το testing ενσωματώνεται σε όλα τα στάδια ανάπτυξης και δεν αντιμετωπίζεται ως απομονωμένη τελική διαδικασία αξιολόγησης (Humble & Farley, 2011). Η συνεχής εφαρμογή testing

συμβάλλει στη μείωση τεχνικών και επιχειρησιακών κινδύνων, στη βελτίωση της ποιότητας λογισμικού και στη διατήρηση της αξιοπιστίας των εφαρμογών σε περιβάλλοντα συνεχών αλλαγών και συχνών εκδόσεων (Crispin & Gregory, 2014).

2.3 Έννοιες Error, Defect και Failure στον Έλεγχο Λογισμικού

Η αποτελεσματική διαχείριση προβλημάτων λογισμικού προϋποθέτει τη σαφή κατανόηση της βασικής ορολογίας που χρησιμοποιείται στο πλαίσιο του ελέγχου λογισμικού και της διασφάλισης ποιότητας. Στο πλαίσιο του ελέγχου λογισμικού χρησιμοποιούνται διαφορετικοί όροι για την περιγραφή προβλημάτων που εμφανίζονται κατά την ανάπτυξη και λειτουργία ενός συστήματος λογισμικού. Η κατανόηση των εννοιών error, defect και failure θεωρείται σημαντική, καθώς επιτρέπει την αποτελεσματικότερη διαχείριση προβλημάτων και τη βελτίωση των διαδικασιών διασφάλισης ποιότητας (ISTQB, 2023).

Ο όρος error (σφάλμα) αναφέρεται σε ανθρώπινο λάθος που πραγματοποιείται κατά τη διάρκεια ανάλυσης απαιτήσεων, σχεδίασης, ανάπτυξης ή συντήρησης του λογισμικού (IEEE Std 610.12-1990). Ένα σφάλμα μπορεί να προκύψει από λανθασμένη κατανόηση απαιτήσεων, εσφαλμένες αποφάσεις σχεδίασης ή προγραμματιστικά λάθη.

Ως αποτέλεσμα ενός σφάλματος δημιουργείται ένα defect, γνωστό και ως bug, το οποίο αποτελεί ατέλεια ή πρόβλημα στο λογισμικό. Ένα defect μπορεί να επηρεάζει διαφορετικές πτυχές του συστήματος, όπως τη λειτουργικότητα, την απόδοση, την ασφάλεια ή τη σταθερότητα της εφαρμογής. Ωστόσο, η ύπαρξη ενός defect δεν συνεπάγεται απαραίτητα ότι το πρόβλημα είναι άμεσα εμφανές στον τελικό χρήστη (Pressman & Maxim, 2020).

Η έννοια του failure (αστοχία) αναφέρεται στην παρατηρήσιμη αποτυχία λειτουργίας του συστήματος κατά την εκτέλεσή του. Μία αστοχία εμφανίζεται όταν ένα defect επηρεάζει τη συμπεριφορά της εφαρμογής και οδηγεί σε λανθασμένα αποτελέσματα ή μη αναμενόμενη λειτουργία σε πραγματικές συνθήκες χρήσης. Η σχέση μεταξύ των παραπάνω εννοιών είναι άμεση, καθώς ένα ανθρώπινο λάθος μπορεί να οδηγήσει στη δημιουργία defect, το οποίο υπό συγκεκριμένες συνθήκες ενδέχεται να οδηγήσει σε αποτυχία λειτουργίας του συστήματος κατά την εκτέλεσή του (ISTQB, 2023).

Η διάκριση μεταξύ των παραπάνω εννοιών είναι ιδιαίτερα σημαντική στις διαδικασίες testing και διαχείρισης defects (defect management), καθώς διευκολύνει την ανάλυση της

προέλευσης και των επιπτώσεων ενός προβλήματος λογισμικού (IEEE Std 610.12-1990). Παράλληλα, επιτρέπει την αποτελεσματικότερη αξιολόγηση της σοβαρότητας ενός defect και την προτεραιοποίηση ενεργειών διόρθωσης κατά τη διαδικασία ανάπτυξης και συντήρησης του συστήματος.

Ο έλεγχος λογισμικού λειτουργεί συμπληρωματικά με τις διαδικασίες debugging (αποσφαλμάτωσης) στο πλαίσιο της διασφάλισης ποιότητας λογισμικού. Ο έλεγχος λογισμικού επικεντρώνεται στον εντοπισμό αποκλίσεων και αστοχιών μέσω της εκτέλεσης δοκιμών, ενώ το debugging αφορά τον εντοπισμό της αιτίας ενός προβλήματος και τη διόρθωσή του από τους προγραμματιστές (Myers et al., 2011). Η αποτελεσματικότητα του debugging εξαρτάται σε σημαντικό βαθμό από την ποιότητα των test cases και την ακρίβεια με την οποία έχουν καταγραφεί τα προβλήματα που εντοπίστηκαν κατά τη διαδικασία testing.

2.4 Δραστηριότητες και Διαδικασία Ελέγχου Λογισμικού

Η διαδικασία του ελέγχου λογισμικού περιλαμβάνει ένα σύνολο οργανωμένων δραστηριοτήτων που υποστηρίζουν τον σχεδιασμό, την εκτέλεση και την αξιολόγηση των δοκιμών λογισμικού (Black, 2002). Οι δραστηριότητες αυτές εφαρμόζονται τόσο σε manual όσο και σε automated testing διαδικασίες και αποτελούν βασικό μέρος των πρακτικών διασφάλισης ποιότητας λογισμικού (ISTQB, 2023).

Σύμφωνα με το πρότυπο ISO/IEC/IEEE 29119 και το syllabus του ISTQB, η διαδικασία testing ξεκινά με τον καθορισμό της στρατηγικής ελέγχου και των στόχων της διαδικασίας ελέγχου (ISO/IEC/IEEE, 2013; ISTQB, 2023). Στο στάδιο αυτό προσδιορίζονται τα κριτήρια αξιολόγησης, οι απαιτούμενοι πόροι, τα περιβάλλοντα δοκιμών και οι βασικές απαιτήσεις που θα πρέπει να καλύπτουν οι δοκιμές.

Στη συνέχεια πραγματοποιείται η ανάλυση απαιτήσεων και ο σχεδιασμός test cases και test scenarios που θα χρησιμοποιηθούν για την αξιολόγηση της εφαρμογής. Η ποιότητα του σχεδιασμού των test cases επηρεάζει άμεσα την αποτελεσματικότητα της διαδικασίας testing, καθώς καθορίζει τον βαθμό κάλυψης των λειτουργικών και μη λειτουργικών απαιτήσεων του συστήματος. Στο πλαίσιο αυτό, η διαδικασία ελέγχου λογισμικού περιλαμβάνει δραστηριότητες όπως ο σχεδιασμός δοκιμών (test planning), η εκτέλεση δοκιμών (test execution) και η αναφορά αποτελεσμάτων δοκιμών (test reporting), οι οποίες

συμβάλλουν στη συστηματική οργάνωση, εκτέλεση και αξιολόγηση των δοκιμών λογισμικού. Οι δραστηριότητες αυτές υποστηρίζουν τη διατήρηση της ποιότητας και της αξιοπιστίας του συστήματος σε όλα τα στάδια του κύκλου ζωής ανάπτυξης λογισμικού (Pressman & Maxim, 2020).

Ιδιαίτερα στις web εφαρμογές, η διαδικασία ελέγχου λογισμικού απαιτεί την αξιολόγηση διαφορετικών ροών χρηστών (user flows), αλληλεπιδράσεων με περιηγητές ιστού και δυναμικών στοιχείων διεπαφής χρήστη. Για τον λόγο αυτό, η οργάνωση των test scenarios και η σωστή διαχείριση των περιβαλλόντων δοκιμών αποτελούν κρίσιμους παράγοντες για την αξιοπιστία των αποτελεσμάτων του ελέγχου λογισμικού (Sommerville, 2016).

Κατά τη φάση εκτέλεσης πραγματοποιείται η εφαρμογή των test cases και η καταγραφή των αποτελεσμάτων των δοκιμών (IEEE Std 829-2008). Στο πλαίσιο αυτό εντοπίζονται defects, αξιολογούνται αποκλίσεις από την αναμενόμενη συμπεριφορά και πραγματοποιείται τεκμηρίωση των προβλημάτων που εμφανίζονται στο σύστημα.

Η διαδικασία ελέγχου λογισμικού ολοκληρώνεται με την αξιολόγηση των αποτελεσμάτων, τη σύνταξη αναφορών δοκιμών και την ανάλυση μετρικών ποιότητας που σχετίζονται με τη σταθερότητα και την αξιοπιστία του λογισμικού (ISO/IEC 25023, 2016). Οι δραστηριότητες αυτές υποστηρίζουν τη λήψη αποφάσεων σχετικά με την ποιότητα της εφαρμογής και συμβάλλουν στη βελτίωση των διαδικασιών ανάπτυξης και συντήρησης λογισμικού.

Στις σύγχρονες προσεγγίσεις ανάπτυξης λογισμικού, οι διαδικασίες ελέγχου λογισμικού συνδέονται όλο και περισσότερο με πρακτικές αυτοματοποίησης, συνεχούς ενοποίησης (continuous integration) και συνεχούς παράδοσης (continuous delivery) (Forsgren, Humble, & Kim, 2018). Η δυνατότητα επαναλαμβανόμενης και αξιόπιστης εκτέλεσης test cases αποτελεί πλέον βασικό παράγοντα υποστήριξης της ποιότητας στις σύγχρονες web εφαρμογές και συνδέεται άμεσα με την αποτελεσματικότητα των διαδικασιών automated testing που εξετάζονται στα επόμενα κεφάλαια (Crispin & Gregory, 2014).

3 Στατικός και Δυναμικός Έλεγχος Λογισμικού

3.1 Εισαγωγή στον Στατικό και Δυναμικό Έλεγχο Λογισμικού

Οι διαδικασίες ελέγχου λογισμικού διακρίνονται σε δύο βασικές κατηγορίες, τον στατικό έλεγχο (static testing) και τον δυναμικό έλεγχο (dynamic testing), οι οποίες αποτελούν θεμελιώδεις προσεγγίσεις διασφάλισης ποιότητας κατά την ανάπτυξη πληροφοριακών συστημάτων. Οι δύο μορφές testing διαφοροποιούνται ως προς τον τρόπο εφαρμογής τους, ωστόσο λειτουργούν συμπληρωματικά στο πλαίσιο αξιολόγησης της ποιότητας και της αξιοπιστίας του λογισμικού (Pressman & Maxim, 2020).

Ο στατικός έλεγχος επικεντρώνεται στην αξιολόγηση τεχνουργημάτων λογισμικού χωρίς την εκτέλεση του κώδικα. Η προσέγγιση αυτή εφαρμόζεται σε απαιτήσεις, σχεδιαστικά έγγραφα, αρχιτεκτονικά μοντέλα, test cases και πηγαίο κώδικα, με στόχο τον έγκαιρο εντοπισμό προβλημάτων και ασυνεπειών πριν από τη λειτουργία του συστήματος. Αντίθετα, ο δυναμικός έλεγχος βασίζεται στην εκτέλεση του λογισμικού και στην παρατήρηση της συμπεριφοράς του κατά τη λειτουργία του σε πραγματικές ή προσομοιωμένες συνθήκες χρήσης (ISTQB, 2023).

Η διάκριση μεταξύ στατικού και δυναμικού ελέγχου λογισμικού είναι ιδιαίτερα σημαντική στις σύγχρονες web εφαρμογές, όπου η πολυπλοκότητα των τεχνολογιών, η ανάγκη υποστήριξης πολλαπλών περιηγητών ιστού και συσκευών, καθώς και οι αυξημένες απαιτήσεις αξιοπιστίας και διαθεσιμότητας καθιστούν απαραίτητη την εφαρμογή πολυεπίπεδων διαδικασιών testing (Beizer, 1990). Η συνδυαστική χρήση των δύο προσεγγίσεων επιτρέπει τόσο τον έγκαιρο εντοπισμό defects όσο και την αξιολόγηση της πραγματικής συμπεριφοράς του συστήματος κατά την εκτέλεση.

Παράλληλα, οι σύγχρονες πρακτικές ανάπτυξης λογισμικού, όπως τα πλαίσια εργασίας Agile και οι πρακτικές DevOps, έχουν ενισχύσει σημαντικά τον ρόλο του στατικού και δυναμικού ελέγχου λογισμικού στον κύκλο ζωής ανάπτυξης λογισμικού (Forsgren, Humble, & Kim, 2018). Η συνεχής ενσωμάτωση διαδικασιών testing σε όλα τα στάδια ανάπτυξης συμβάλλει στη βελτίωση της ποιότητας, στη μείωση του τεχνικού χρέους και στην ταχύτερη αναγνώριση προβλημάτων πριν από την παραγωγική λειτουργία των εφαρμογών (Crispin & Gregory, 2014).

3.2 Στατικός Έλεγχος

Ο στατικός έλεγχος αποτελεί βασική προσέγγιση ελέγχου λογισμικού που πραγματοποιείται χωρίς την εκτέλεση του συστήματος. Η διαδικασία αυτή επικεντρώνεται στην αξιολόγηση τεχνουργημάτων λογισμικού, όπως απαιτήσεις, σχεδιαστικά έγγραφα, αρχιτεκτονικά μοντέλα, test cases και πηγαίος κώδικας, με στόχο τον έγκαιρο εντοπισμό προβλημάτων και ασυνεπειών πριν από τη φάση εκτέλεσης του λογισμικού (Sommerville, 2016).

Η βασική φιλοσοφία του στατικού ελέγχου στηρίζεται στην πρόληψη προβλημάτων σε πρώιμα στάδια ανάπτυξης. Η έγκαιρη αναγνώριση defects επιτρέπει τη μείωση του κόστους διόρθωσης και περιορίζει την πιθανότητα εμφάνισης σοβαρών αστοχιών σε μεταγενέστερα στάδια του κύκλου ζωής ανάπτυξης λογισμικού. Για τον λόγο αυτό, ο στατικός έλεγχος θεωρείται σημαντικός μηχανισμός υποστήριξης της ποιότητας και της αξιοπιστίας ενός πληροφοριακού συστήματος (Pressman & Maxim, 2020).

Μία από τις πιο διαδεδομένες μορφές στατικού ελέγχου είναι οι διαδικασίες ανασκόπησης (reviews) (Fagan, 1976). Κατά τη διάρκεια των ανασκοπήσεων, μέλη της ομάδας ανάπτυξης εξετάζουν τεχνουργήματα λογισμικού με στόχο τον εντοπισμό λαθών, παραλείψεων και ασυνεπειών. Οι διαδικασίες αυτές μπορούν να εφαρμοστούν σε απαιτήσεις, σχεδιαστικά έγγραφα, test plans, test cases και πηγαίο κώδικα. Σύμφωνα με το ISTQB (2023), οι ανασκοπήσεις συμβάλλουν όχι μόνο στη μείωση defects αλλά και στη βελτίωση της επικοινωνίας και της ανταλλαγής γνώσης μεταξύ των μελών της ομάδας ανάπτυξης.

Στο πλαίσιο του στατικού ελέγχου χρησιμοποιούνται επίσης τεχνικές ανασκόπησης, όπως οι καθοδηγούμενες ανασκοπήσεις (walkthroughs) και οι επιθεωρήσεις (inspections). Τα walkthroughs αποτελούν περισσότερο ανεπίσημες μορφές ανασκόπησης, όπου ο δημιουργός ενός τεχνουργήματος παρουσιάζει το περιεχόμενό του στην υπόλοιπη ομάδα προκειμένου να εντοπιστούν πιθανά προβλήματα μέσω συζήτησης και ανατροφοδότησης. Αντίθετα, τα inspections αποτελούν πιο δομημένες και επίσημες διαδικασίες αξιολόγησης, οι οποίες περιλαμβάνουν προκαθορισμένους ρόλους, συγκεκριμένα στάδια ελέγχου και συστηματική καταγραφή ευρημάτων (Myers et al., 2011).

Ιδιαίτερα σημαντικό ρόλο στις σύγχρονες διαδικασίες στατικού ελέγχου κατέχει και η στατική ανάλυση κώδικα (static code analysis). Η διαδικασία αυτή πραγματοποιείται μέσω

εξειδικευμένων εργαλείων που αναλύουν τον πηγαίο κώδικα χωρίς την εκτέλεσή του, με στόχο τον εντοπισμό προβλημάτων ποιότητας και πιθανών αδυναμιών του λογισμικού. Εργαλεία στατικής ανάλυσης κώδικα χρησιμοποιούνται ευρέως για τον εντοπισμό παραβιάσεων των coding standards, code smells, dead code και πιθανών προβλημάτων συντηρησιμότητας και ασφάλειας (Sommerville, 2016).

Η σημασία του στατικού ελέγχου έχει ενισχυθεί σημαντικά στα σύγχρονα Agile και DevOps περιβάλλοντα ανάπτυξης λογισμικού, όπου η συνεχής αξιολόγηση ποιότητας αποτελεί βασικό στοιχείο της διαδικασίας ανάπτυξης (Humble & Farley, 2011). Η εφαρμογή στατικού ελέγχου σε πρώιμα στάδια επιτρέπει τη γρηγορότερη αναγνώριση προβλημάτων, τη μείωση του τεχνικού χρέους και τη βελτίωση της συνολικής σταθερότητας του συστήματος.

Παρά τα σημαντικά πλεονεκτήματα που παρουσιάζει, ο στατικός έλεγχος δεν μπορεί να αξιολογήσει την πραγματική συμπεριφορά του λογισμικού κατά την εκτέλεσή του, ούτε να εντοπίσει αστοχίες κατά τον χρόνο εκτέλεσης ή προβλήματα που εκδηλώνονται αποκλειστικά κατά τη λειτουργία της εφαρμογής (IEEE Std 1028-2008). Για τον λόγο αυτό, ο στατικός έλεγχος δεν λειτουργεί ως υποκατάστατο του δυναμικού ελέγχου, αλλά ως συμπληρωματική διαδικασία στο πλαίσιο της συνολικής στρατηγικής διασφάλισης ποιότητας λογισμικού (ISTQB, 2023).

3.3 Δυναμικός Έλεγχος

Ο δυναμικός έλεγχος αποτελεί διαδικασία ελέγχου λογισμικού που βασίζεται στην εκτέλεση του συστήματος και στην αξιολόγηση της συμπεριφοράς του κατά τη λειτουργία του. Σε αντίθεση με τον στατικό έλεγχο, το οποίο πραγματοποιείται χωρίς εκτέλεση του κώδικα, ο δυναμικός έλεγχος επιτρέπει την παρατήρηση της πραγματικής λειτουργίας της εφαρμογής και τον εντοπισμό αστοχιών που εμφανίζονται κατά την εκτέλεση του λογισμικού (Kaner et al., 2002).

Βασικός στόχος του δυναμικού ελέγχου είναι η επαλήθευση της λειτουργικότητας και της συμπεριφοράς του συστήματος σε πραγματικές ή προσομοιωμένες συνθήκες χρήσης. Μέσω των διαδικασιών δυναμικού ελέγχου αξιολογείται κατά πόσο το λογισμικό ανταποκρίνεται στις λειτουργικές και μη λειτουργικές απαιτήσεις, ενώ παράλληλα εξετάζεται η σταθερότητα, η αξιοπιστία και η συνολική απόδοση της εφαρμογής (ISTQB, 2023).

Η σημασία του δυναμικού ελέγχου είναι ιδιαίτερα αυξημένη στις σύγχρονες web εφαρμογές, όπου η λειτουργία του συστήματος επηρεάζεται από μεγάλο αριθμό παραγόντων, όπως οι διαφορετικοί περιηγητές ιστού, οι συσκευές πρόσβασης, οι δικτυακές συνθήκες και η αλληλεπίδραση πολλών χρηστών με την εφαρμογή (Black, 2002). Στο πλαίσιο αυτό, ο δυναμικός έλεγχος χρησιμοποιείται για την αξιολόγηση αλληλεπιδράσεων χρηστών (user interactions), διαδικασιών αυθεντικοποίησης, της συμβατότητας με διαφορετικούς περιηγητές ιστού, της προσαρμοστικής συμπεριφοράς της εφαρμογής και της λειτουργίας των APIs (Διεπαφές Προγραμματισμού Εφαρμογών) (Pressman & Maxim, 2020).

Σύμφωνα με τον Sommerville (2016), ο δυναμικός έλεγχος εφαρμόζεται σε διαφορετικά επίπεδα ελέγχου, ανάλογα με το στάδιο ανάπτυξης και το εύρος αξιολόγησης του συστήματος. Το unit testing (έλεγχος μονάδων) επικεντρώνεται στον έλεγχο μεμονωμένων μονάδων κώδικα, όπως συναρτήσεων ή μεθόδων, με στόχο τον εντοπισμό προβλημάτων σε χαμηλό επίπεδο ανάπτυξης. Στη συνέχεια, το integration testing (έλεγχος ολοκλήρωσης) αξιολογεί τη συνεργασία μεταξύ διαφορετικών υποσυστημάτων ή μονάδων λογισμικού, όπως η επικοινωνία μεταξύ του frontend (διεπαφή χρήστη) και του backend (υποσύστημα εξυπηρέτησης), καθώς και η αλληλεπίδραση με βάσεις δεδομένων και APIs. Τα επίπεδα αυτά λειτουργούν συμπληρωματικά και επιτρέπουν τη σταδιακή αξιολόγηση της λειτουργικότητας και της σταθερότητας του συστήματος σε διαφορετικά στάδια ανάπτυξης.

Ακολούθως, το system testing (έλεγχος συστήματος) εξετάζει το λογισμικό ως ολοκληρωμένο σύστημα, αξιολογώντας κατά πόσο ικανοποιούνται οι συνολικές λειτουργικές και μη λειτουργικές απαιτήσεις της εφαρμογής. Τέλος, το acceptance testing (έλεγχος αποδοχής) πραγματοποιείται με στόχο την επιβεβαίωση ότι το σύστημα ανταποκρίνεται στις επιχειρησιακές ανάγκες και μπορεί να χρησιμοποιηθεί αποτελεσματικά από τους τελικούς χρήστες ή τον πελάτη (ISO/IEC/IEEE 29119-2, 2021).

Ο δυναμικός έλεγχος προσφέρει σημαντικά πλεονεκτήματα στις διαδικασίες διασφάλισης ποιότητας λογισμικού, καθώς επιτρέπει τον εντοπισμό πραγματικών αποτυχιών, την αξιολόγηση της λειτουργικότητας του συστήματος και την προσομοίωση πραγματικών συνθηκών χρήσης. Παράλληλα, συμβάλλει ουσιαστικά στην αξιολόγηση της εμπειρίας χρήστη και της συνολικής συμπεριφοράς της εφαρμογής σε περιβάλλον παραγωγικής λειτουργίας (Myers et al., 2011).

Ωστόσο, η εφαρμογή δυναμικού ελέγχου συνοδεύεται και από ορισμένους περιορισμούς (Ammann & Offutt, 2016). Η διαδικασία απαιτεί κατάλληλα περιβάλλοντα δοκιμών, επαρκή δεδομένα δοκιμών (test data) και αυξημένο χρόνο εκτέλεσης, ενώ δεν είναι εφικτή η πλήρης κάλυψη όλων των πιθανών περιπτώσεων χρήσης και συνθηκών λειτουργίας. Παρά τους περιορισμούς αυτούς, ο δυναμικός έλεγχος αποτελεί βασικό πυλώνα των σύγχρονων διαδικασιών testing και εφαρμόζεται εκτεταμένα στις web εφαρμογές για την αξιολόγηση της ποιότητας και της αξιοπιστίας του λογισμικού.

3.4 Σύγκριση Στατικού και Δυναμικού Ελέγχου

Ο στατικός έλεγχος και ο δυναμικός έλεγχος αποτελούν δύο διαφορετικές αλλά συμπληρωματικές προσεγγίσεις ελέγχου λογισμικού, οι οποίες εφαρμόζονται με στόχο τη βελτίωση της ποιότητας και της αξιοπιστίας ενός πληροφοριακού συστήματος. Παρότι οι δύο μορφές testing διαφοροποιούνται ως προς τον τρόπο εφαρμογής και το είδος προβλημάτων που μπορούν να εντοπίσουν, η συνδυαστική χρήση τους θεωρείται ιδιαίτερα σημαντική στις σύγχρονες διαδικασίες ανάπτυξης λογισμικού (Pressman & Maxim, 2020).

Ο στατικός έλεγχος πραγματοποιείται χωρίς την εκτέλεση του λογισμικού και επικεντρώνεται στην αξιολόγηση τεχνουργημάτων όπως απαιτήσεις, σχεδιαστικά έγγραφα και πηγαίος κώδικας. Η βασική του επιδίωξη είναι ο έγκαιρος εντοπισμός προβλημάτων σε πρώιμα στάδια ανάπτυξης, πριν τα defects επηρεάσουν τη λειτουργία του συστήματος. Μέσω διαδικασιών όπως οι ανασκοπήσεις και οι inspections, καθώς και μέσω της στατικής ανάλυσης κώδικα, ο στατικός έλεγχος συμβάλλει στη βελτίωση της ποιότητας τεκμηρίωσης και σχεδίασης, καθώς και στη μείωση του κόστους διόρθωσης προβλημάτων (ISTQB, 2023).

Αντίθετα, ο δυναμικός έλεγχος βασίζεται στην εκτέλεση του λογισμικού και στην αξιολόγηση της πραγματικής συμπεριφοράς του συστήματος κατά τη λειτουργία του (Beizer, 1990). Η προσέγγιση αυτή επιτρέπει τον εντοπισμό αστοχιών και προβλημάτων που εκδηλώνονται κατά τον χρόνο εκτέλεσης και δεν μπορούν να αναγνωριστούν μέσω διαδικασιών στατικού ελέγχου. Παράλληλα, ο δυναμικός έλεγχος παρέχει τη δυνατότητα αξιολόγησης της λειτουργικότητας, της απόδοσης και της εμπειρίας χρήστη σε πραγματικές συνθήκες λειτουργίας.

Μία ακόμη βασική διαφοροποίηση των δύο προσεγγίσεων αφορά το χρονικό σημείο εφαρμογής τους στον κύκλο ζωής ανάπτυξης λογισμικού (IEEE Std 1012-2016). Ο στατικός έλεγχος εφαρμόζεται κυρίως σε πρώιμα στάδια ανάπτυξης, ενώ ο δυναμικός έλεγχος πραγματοποιείται όταν το σύστημα ή μέρος αυτού μπορεί να εκτελεστεί. Για τον λόγο αυτό, ο στατικός έλεγχος λειτουργεί περισσότερο προληπτικά, ενώ ο δυναμικός έλεγχος επικεντρώνεται κυρίως στον εντοπισμό προβλημάτων κατά τη λειτουργία της εφαρμογής.

Στις σύγχρονες web εφαρμογές, όπου η πολυπλοκότητα των τεχνολογιών και οι απαιτήσεις αξιοπιστίας είναι ιδιαίτερα αυξημένες, η συνδυαστική αξιοποίηση στατικού και δυναμικού ελέγχου θεωρείται απαραίτητη (ISO/IEC/IEEE 29119-1, 2022). Η εφαρμογή του στατικού ελέγχου συμβάλλει στην έγκαιρη βελτίωση της ποιότητας του κώδικα και του σχεδιασμού, ενώ ο δυναμικός έλεγχος επιτρέπει την αξιολόγηση της πραγματικής συμπεριφοράς του συστήματος υπό διαφορετικές συνθήκες λειτουργίας και χρήσης.

Συνεπώς, οι δύο μορφές testing δεν λειτουργούν ανταγωνιστικά αλλά συμπληρωματικά στο πλαίσιο της συνολικής στρατηγικής διασφάλισης ποιότητας λογισμικού, υποστηρίζοντας τη δημιουργία πιο αξιόπιστων, σταθερών και λειτουργικών web εφαρμογών (Sommerville, 2016).

3.5 Η Προσέγγιση Shift-Left Testing

Το Shift-Left Testing αποτελεί σύγχρονη προσέγγιση ελέγχου λογισμικού που βασίζεται στη μεταφορά των δραστηριοτήτων ελέγχου σε πρώιμα στάδια του κύκλου ζωής ανάπτυξης λογισμικού. Η φιλοσοφία της προσέγγισης αυτής στηρίζεται στην έγκαιρη ανίχνευση προβλημάτων και στη συνεχή ενσωμάτωση διαδικασιών διασφάλισης ποιότητας καθ' όλη τη διάρκεια ανάπτυξης του συστήματος (Crispin & Gregory, 2014).

Στα παραδοσιακά μοντέλα ανάπτυξης λογισμικού, οι διαδικασίες testing πραγματοποιούνταν κυρίως μετά την ολοκλήρωση της υλοποίησης του συστήματος. Αντίθετα, η προσέγγιση shift-left επιδιώκει την ενεργή συμμετοχή των διαδικασιών testing ήδη από τα αρχικά στάδια ανάλυσης απαιτήσεων, σχεδίασης και ανάπτυξης του λογισμικού. Με τον τρόπο αυτό καθίσταται δυνατός ο εντοπισμός defects πριν αυτά επηρεάσουν

σημαντικά τη λειτουργικότητα και τη σταθερότητα της εφαρμογής (Pressman & Maxim, 2020).

Η εφαρμογή shift-left πρακτικών παρουσιάζει σημαντικά πλεονεκτήματα στις σύγχρονες διαδικασίες ανάπτυξης λογισμικού. Η έγκαιρη ανίχνευση προβλημάτων συμβάλλει στη μείωση του κόστους διόρθωσης defects, στη βελτίωση της ποιότητας κώδικα και στη μείωση του τεχνικού χρέους του συστήματος. Παράλληλα, ενισχύεται η συνεργασία μεταξύ ομάδων ανάπτυξης, testing και λειτουργίας συστημάτων, γεγονός που υποστηρίζει αποτελεσματικότερα τις σύγχρονες Agile και DevOps πρακτικές ανάπτυξης (Crispin & Gregory, 2014).

Στο πλαίσιο του Shift-Left Testing εφαρμόζονται τεχνικές όπως οι ανασκοπήσεις κώδικα, η στατική ανάλυση κώδικα, το unit testing, η συνεχής ενοποίηση και οι διαδικασίες automated testing (Martin Fowler, 2006). Η αξιοποίηση των τεχνικών αυτών επιτρέπει τη συνεχή αξιολόγηση της ποιότητας λογισμικού και την ταχύτερη παροχή ανατροφοδότησης προς τις ομάδες ανάπτυξης.

Η σημασία του shift-left testing είναι ιδιαίτερα αυξημένη στις web εφαρμογές, όπου οι απαιτήσεις για συχνές ενημερώσεις, συνεχείς αλλαγές λειτουργικότητας και γρήγορους κύκλους παράδοσης λογισμικού καθιστούν απαραίτητη τη διατήρηση υψηλού επιπέδου ποιότητας σε όλα τα στάδια ανάπτυξης. Μέσα από τη συστηματική εφαρμογή shift-left πρακτικών, οι οργανισμοί μπορούν να αναπτύσσουν πιο σταθερές, αξιόπιστες και ποιοτικές web εφαρμογές, μειώνοντας παράλληλα τον κίνδυνο εμφάνισης προβλημάτων σε περιβάλλον παραγωγικής λειτουργίας (ISTQB, 2023).

4 Ποιότητα Λογισμικού και Διασφάλιση Ποιότητας σε Web Εφαρμογές

4.1 Η Έννοια της Ποιότητας Λογισμικού

Η ποιότητα λογισμικού αποτελεί θεμελιώδη παράγοντα επιτυχίας ενός πληροφοριακού συστήματος και συνδέεται άμεσα με την ικανότητα του λογισμικού να ικανοποιεί λειτουργικές, τεχνικές και επιχειρησιακές απαιτήσεις (Kitchenham & Pfleeger, 1996). Στο σύγχρονο περιβάλλον ανάπτυξης web εφαρμογών, όπου η αξιοπιστία, η διαθεσιμότητα, η ασφάλεια και η εμπειρία χρήστη επηρεάζουν καθοριστικά τη χρήση και την αποδοχή ενός συστήματος, η διασφάλιση ποιότητας αποκτά ιδιαίτερη σημασία (Pressman & Maxim, 2020).

Η έννοια της ποιότητας στο λογισμικό δεν περιορίζεται αποκλειστικά στη σωστή λειτουργία μιας εφαρμογής. Αντίθετα, περιλαμβάνει ένα ευρύτερο σύνολο χαρακτηριστικών που σχετίζονται με τη σταθερότητα, τη χρηστικότητα, τη συμβατότητα, την αποδοτικότητα και τη συνολική αξιοπιστία του συστήματος. Σύμφωνα με τον Sommerville (2016), ένα ποιοτικό λογισμικό οφείλει όχι μόνο να ανταποκρίνεται στις τεχνικές προδιαγραφές ανάπτυξης, αλλά και να καλύπτει αποτελεσματικά τις πραγματικές ανάγκες και προσδοκίες των χρηστών.

Η διασφάλιση ποιότητας λογισμικού (Software Quality Assurance – SQA) περιλαμβάνει ένα σύνολο οργανωμένων διαδικασιών και δραστηριοτήτων που εφαρμόζονται με στόχο την πρόληψη προβλημάτων και τη συνεχή αξιολόγηση της ποιότητας κατά τη διάρκεια ανάπτυξης του συστήματος (IEEE Std 730-2014). Στο πλαίσιο αυτό, ο έλεγχος λογισμικού αποτελεί έναν από τους σημαντικότερους μηχανισμούς αξιολόγησης ποιότητας, καθώς επιτρέπει τον εντοπισμό defects, την επαλήθευση λειτουργικών απαιτήσεων και την αξιολόγηση της συμπεριφοράς της εφαρμογής σε διαφορετικές συνθήκες λειτουργίας (ISTQB, 2023).

Η σημασία της ποιότητας είναι ιδιαίτερα αυξημένη στις web εφαρμογές, οι οποίες λειτουργούν σε δυναμικά και ετερογενή περιβάλλοντα. Η υποστήριξη διαφορετικών περιηγητών ιστού, λειτουργικών συστημάτων και κινητών συσκευών, σε συνδυασμό με τις απαιτήσεις συνεχούς διαθεσιμότητας και αυξημένης ασφάλειας, καθιστούν απαραίτητη την εφαρμογή συστηματικών διαδικασιών διασφάλισης ποιότητας και testing σε όλα τα στάδια ανάπτυξης (Sommerville, 2016).

Παράλληλα, η εξάπλωση Agile και DevOps πρακτικών έχει μεταβάλει σημαντικά τον τρόπο με τον οποίο προσεγγίζεται η ποιότητα λογισμικού. Η ποιότητα δεν αντιμετωπίζεται πλέον ως τελικός έλεγχος μετά την ολοκλήρωση της ανάπτυξης, αλλά ως συνεχής διαδικασία που ενσωματώνεται σε ολόκληρο τον κύκλο ζωής ανάπτυξης λογισμικού. Για τον λόγο αυτό, οι διαδικασίες manual και automated testing διαδραματίζουν καθοριστικό ρόλο στη διατήρηση υψηλού επιπέδου ποιότητας και αξιοπιστίας στις σύγχρονες web εφαρμογές (Crispin & Gregory, 2014).

Στο πλαίσιο της αξιολόγησης ποιότητας έχουν αναπτυχθεί διεθνή πρότυπα και μοντέλα ποιότητας, τα οποία παρέχουν οργανωμένα κριτήρια αξιολόγησης λογισμικού και υποστηρίζουν τις διαδικασίες testing και διασφάλισης ποιότητας. Ένα από τα σημαντικότερα πρότυπα στον τομέα αυτό είναι το ISO/IEC 25010, το οποίο χρησιμοποιείται ευρέως για την αξιολόγηση των χαρακτηριστικών ποιότητας του λογισμικού και παρουσιάζεται αναλυτικά στην επόμενη ενότητα (ISO/IEC, 2011).

4.2 Το Πρότυπο ISO/IEC 25010 και η Αξιολόγηση Ποιότητας

Η αξιολόγηση της ποιότητας λογισμικού αποτελεί σύνθετη διαδικασία, καθώς η ποιότητα ενός πληροφοριακού συστήματος δεν εξαρτάται αποκλειστικά από τη σωστή λειτουργικότητά του, αλλά και από ένα ευρύτερο σύνολο τεχνικών και λειτουργικών χαρακτηριστικών. Για τον λόγο αυτό, έχουν αναπτυχθεί διεθνή πρότυπα τα οποία παρέχουν οργανωμένα μοντέλα αξιολόγησης ποιότητας και υποστηρίζουν τις διαδικασίες διασφάλισης ποιότητας λογισμικού (Pressman & Maxim, 2020).

Ένα από τα σημαντικότερα πρότυπα στον τομέα αυτό είναι το ISO/IEC 25010, το οποίο αποτελεί μέρος της σειράς προτύπων SQuaRE (Software Quality Requirements and Evaluation) και χρησιμοποιείται ευρέως στη μηχανική λογισμικού για τον καθορισμό και την αξιολόγηση χαρακτηριστικών ποιότητας λογισμικού (ISO/IEC, 2011). Το πρότυπο παρέχει ένα οργανωμένο μοντέλο ποιότητας που επιτρέπει την καλύτερη κατανόηση των απαιτήσεων ποιότητας ενός συστήματος και συμβάλλει στη συστηματική εφαρμογή διαδικασιών testing και αξιολόγησης.

Σύμφωνα με το ISO/IEC 25010, η ποιότητα λογισμικού αξιολογείται μέσω ενός συνόλου χαρακτηριστικών που σχετίζονται με τη λειτουργικότητα, την αξιοπιστία, τη χρηστικότητα,

την ασφάλεια, τη συμβατότητα και άλλες ιδιότητες του συστήματος (ISO/IEC 25023, 2016). Τα χαρακτηριστικά αυτά χρησιμοποιούνται τόσο για τον καθορισμό απαιτήσεων ποιότητας όσο και για τον σχεδιασμό διαδικασιών testing και την αξιολόγηση αποτελεσμάτων ελέγχου λογισμικού.

Η σημασία του προτύπου είναι ιδιαίτερα αυξημένη στις web εφαρμογές, όπου η ποιότητα επηρεάζεται από πολλούς τεχνικούς και λειτουργικούς παράγοντες (Kitchenham & Pfleeger, 1996). Η ανάγκη υποστήριξης διαφορετικών browsers και συσκευών, οι απαιτήσεις συνεχούς διαθεσιμότητας και οι αυξημένες απαιτήσεις ασφάλειας καθιστούν απαραίτητη τη χρήση οργανωμένων μοντέλων αξιολόγησης ποιότητας. Μέσω του ISO/IEC 25010 διευκολύνεται ο καθορισμός απαιτήσεων ποιότητας και η συστηματική αξιολόγηση των χαρακτηριστικών ενός συστήματος (ISO/IEC, 2011).

4.3 Λειτουργική Καταλληλότητα και Αξιοπιστία στις Web Εφαρμογές

Τα λειτουργικά χαρακτηριστικά ποιότητας διαδραματίζουν καθοριστικό ρόλο στην αξιολόγηση web εφαρμογών, καθώς επηρεάζουν άμεσα τόσο τη σωστή λειτουργία του συστήματος όσο και τη συνολική εμπειρία χρήστη. Στο πλαίσιο του ISO/IEC 25010, ιδιαίτερη σημασία αποδίδεται στη λειτουργική καταλληλότητα (functional suitability) και στην αξιοπιστία (reliability), χαρακτηριστικά που συνδέονται στενά με τις διαδικασίες ελέγχου λογισμικού και τη διασφάλιση ποιότητας λογισμικού.

Η λειτουργική καταλληλότητα αφορά τον βαθμό στον οποίο μια εφαρμογή παρέχει λειτουργίες που ανταποκρίνονται στις απαιτήσεις και στις ανάγκες των χρηστών υπό συγκεκριμένες συνθήκες λειτουργίας (ISO/IEC, 2011). Η αξιολόγηση της λειτουργικής καταλληλότητας σχετίζεται άμεσα με τη δυνατότητα του συστήματος να εκτελεί σωστά τις απαιτούμενες λειτουργίες, να διαχειρίζεται δεδομένα με ορθό τρόπο και να υποστηρίζει αποτελεσματικά τις επιχειρησιακές διαδικασίες για τις οποίες σχεδιάστηκε.

Στις web εφαρμογές, η λειτουργική καταλληλότητα επηρεάζεται από πολλούς παράγοντες, όπως η ορθή λειτουργία μηχανισμών αυθεντικοποίησης (authentication), η διαχείριση δεδομένων χρηστών, η λειτουργία φορμών και λειτουργικών ροών εργασίας, καθώς και η συνολική συνέπεια της επιχειρησιακής λογικής της εφαρμογής (ISO/IEC/IEEE 29119-1, 2022). Για τον λόγο αυτό, οι διαδικασίες λειτουργικού ελέγχου (functional testing)

αποτελούν βασικό τμήμα της αξιολόγησης ποιότητας και εφαρμόζονται τόσο σε manual όσο και σε automated testing περιβάλλοντα.

Η αξιολόγηση της λειτουργικής καταλληλότητας πραγματοποιείται κυρίως μέσω test cases που βασίζονται σε λειτουργικές απαιτήσεις και επιχειρησιακά σενάρια χρήσης. Μέσω των διαδικασιών αυτών εξετάζεται κατά πόσο τα αποτελέσματα της εφαρμογής συμφωνούν με τις τεχνικές προδιαγραφές και τις προσδοκίες των χρηστών. Η χρήση οργανωμένων τεχνικών ελέγχου, όπως το black-box testing (έλεγχος μαύρου κουτιού) και το acceptance testing, συμβάλλει σημαντικά στην αποτελεσματική αξιολόγηση της λειτουργικότητας του συστήματος (ISTQB, 2023).

Παράλληλα με τη λειτουργική καταλληλότητα, ιδιαίτερα σημαντικό χαρακτηριστικό ποιότητας αποτελεί η αξιοπιστία του λογισμικού. Η αξιοπιστία αναφέρεται στην ικανότητα ενός συστήματος να λειτουργεί σταθερά και χωρίς αποτυχίες υπό καθορισμένες συνθήκες και για συγκεκριμένο χρονικό διάστημα (Sommerville, 2016). Η αξιοπιστία είναι κρίσιμη στις web εφαρμογές, ιδιαίτερα όταν αυτές υποστηρίζουν επιχειρησιακές λειτουργίες ή εξυπηρετούν μεγάλο αριθμό χρηστών σε περιβάλλοντα συνεχούς λειτουργίας.

Προβλήματα αξιοπιστίας μπορούν να οδηγήσουν σε διακοπή υπηρεσιών, απώλεια δεδομένων, μειωμένη εμπιστοσύνη χρηστών και σημαντικές επιχειρησιακές επιπτώσεις (Laprie, 1992). Για τον λόγο αυτό, οι διαδικασίες testing που σχετίζονται με την αξιοπιστία επικεντρώνονται στην αξιολόγηση της σταθερότητας και της συμπεριφοράς της εφαρμογής υπό διαφορετικές συνθήκες λειτουργίας. Στις διαδικασίες αυτές περιλαμβάνονται το regression testing, το system testing και διάφορες μορφές performance evaluation.

Η αυτοματοποίηση δοκιμών διαδραματίζει ιδιαίτερα σημαντικό ρόλο στη διατήρηση της αξιοπιστίας web εφαρμογών, καθώς επιτρέπει την επαναλαμβανόμενη εκτέλεση tests μετά από αλλαγές ή ενημερώσεις του συστήματος. Η εφαρμογή διαδικασιών automated regression testing καθιστά δυνατή την έγκαιρη ανίχνευση προβλημάτων που ενδέχεται να επηρεάσουν τη σταθερότητα και τη συνολική λειτουργία της εφαρμογής (Myers et al., 2011).

Συνεπώς, η λειτουργική καταλληλότητα και η αξιοπιστία αποτελούν βασικά χαρακτηριστικά ποιότητας του λογισμικού, συμβάλλοντας καθοριστικά στην αποτελεσματική λειτουργία και την ικανοποίηση των απαιτήσεων των χρηστών. Η

συστηματική αξιολόγηση των χαρακτηριστικών αυτών συμβάλλει στη δημιουργία περισσότερο αξιόπιστων, σταθερών και λειτουργικών πληροφοριακών συστημάτων.

4.4 Μη Λειτουργικά Χαρακτηριστικά Ποιότητας στις Web Εφαρμογές

Πέρα από τη σωστή λειτουργικότητα ενός συστήματος, ιδιαίτερα σημαντικό ρόλο στην αξιολόγηση ποιότητας διαδραματίζουν και τα μη λειτουργικά χαρακτηριστικά (non-functional characteristics) των web εφαρμογών (ISO/IEC 25022, 2016). Τα χαρακτηριστικά αυτά δεν σχετίζονται άμεσα με το ποιες λειτουργίες παρέχει ένα σύστημα, αλλά με τον τρόπο με τον οποίο αυτές οι λειτουργίες εκτελούνται και γίνονται αντιληπτές από τους χρήστες. Στο πλαίσιο του ISO/IEC 25010, χαρακτηριστικά όπως η χρηστικότητα (usability), η συμβατότητα (compatibility) και η ασφάλεια (security) αποτελούν βασικούς παράγοντες αξιολόγησης της συνολικής ποιότητας λογισμικού.

Η χρηστικότητα αφορά τον βαθμό στον οποίο μια εφαρμογή μπορεί να χρησιμοποιηθεί αποτελεσματικά, αποδοτικά και με ικανοποίηση από τους τελικούς χρήστες (ISO/IEC, 2011). Στις web εφαρμογές, η χρηστικότητα επηρεάζει άμεσα την εμπειρία χρήστη και τη συνολική αποδοχή του συστήματος. Ακόμη και όταν μια εφαρμογή λειτουργεί ορθά από τεχνική άποψη, προβλήματα που σχετίζονται με την πλοήγηση, τις διεπαφές χρήστη ή την οργάνωση του περιεχομένου ενδέχεται να επηρεάσουν αρνητικά την αποτελεσματικότητα και τη χρηστικότητα του συστήματος.

Η αξιολόγηση της χρηστικότητας πραγματοποιείται κυρίως μέσω διαδικασιών manual testing και βασίζεται σε μεγάλο βαθμό στην παρατήρηση της αλληλεπίδρασης των χρηστών με το σύστημα και στην καταγραφή της εμπειρίας χρήσης (Nielsen, 1994). Οι διαδικασίες αυτές εξετάζουν στοιχεία όπως η ευκολία πλοήγησης, η σαφήνεια του περιβάλλοντος διεπαφής, η συνέπεια σχεδίασης και η συνολική αλληλεπίδραση μεταξύ χρήστη και εφαρμογής. Για τον λόγο αυτό, το manual testing διατηρεί ιδιαίτερα σημαντικό ρόλο στις διαδικασίες αξιολόγησης της χρηστικότητας, ακόμη και σε περιβάλλοντα όπου εφαρμόζονται εκτεταμένες πρακτικές automated testing (Crispin & Gregory, 2014).

Εξίσου σημαντικό χαρακτηριστικό ποιότητας αποτελεί η συμβατότητα, δηλαδή η ικανότητα της εφαρμογής να λειτουργεί σωστά σε διαφορετικά περιβάλλοντα εκτέλεσης και σε συνδυασμό με διαφορετικές τεχνολογίες και συσκευές. Στις σύγχρονες web

εφαρμογές, η συμβατότητα σχετίζεται κυρίως με τη λειτουργία σε διαφορετικούς περιηγητές ιστού, λειτουργικά συστήματα και κινητές συσκευές, καθώς και με την προσαρμογή της εφαρμογής σε διαφορετικές αναλύσεις οθόνης και προσαρμοστικά περιβάλλοντα χρήσης (Pressman & Maxim, 2020).

Η αξιολόγηση της συμβατότητας πραγματοποιείται μέσω browser testing και δοκιμών σε πολλαπλά περιβάλλοντα λειτουργίας, με στόχο τη διασφάλιση της ορθής λειτουργίας της εφαρμογής σε διαφορετικούς περιηγητές ιστού, συσκευές και λειτουργικά συστήματα (W3C, 2023). Η σημασία της συγκεκριμένης διαδικασίας έχει αυξηθεί σημαντικά λόγω της ευρείας χρήσης κινητών συσκευών και της μεγάλης ποικιλίας web τεχνολογιών που χρησιμοποιούνται στις σύγχρονες εφαρμογές. Παράλληλα, οι διαδικασίες automated testing επιτρέπουν την ταχεία και επαναλαμβανόμενη αξιολόγηση των απαιτήσεων συμβατότητας σε διαφορετικά περιβάλλοντα εκτέλεσης, συμβάλλοντας στη διασφάλιση της ορθής λειτουργίας της εφαρμογής σε ετερογενείς πλατφόρμες και συσκευές.

Παράλληλα με τη χρηστικότητα και τη συμβατότητα, ιδιαίτερα κρίσιμο χαρακτηριστικό ποιότητας αποτελεί και η ασφάλεια των web εφαρμογών. Η ασφάλεια αφορά την προστασία δεδομένων και πληροφοριών από μη εξουσιοδοτημένη πρόσβαση, αλλοίωση ή καταστροφή και συνδέεται άμεσα με τη συνολική αξιοπιστία ενός πληροφοριακού συστήματος (ISO/IEC, 2011; Pressman & Maxim, 2020).

Οι διαδικασίες security testing στοχεύουν στον εντοπισμό πιθανών ευπαθειών και αδυναμιών που ενδέχεται να επηρεάσουν τη λειτουργία ή την ασφάλεια της εφαρμογής. Παραδείγματα τέτοιων προβλημάτων περιλαμβάνουν αδύναμους μηχανισμούς αυθεντικοποίησης, προβλήματα ελέγχου πρόσβασης, καθώς και ευπάθειες όπως οι επιθέσεις SQL injection και cross-site scripting (XSS), οι οποίες μπορούν να οδηγήσουν σε μη εξουσιοδοτημένη πρόσβαση, παραβίαση δεδομένων ή διακοπή της ορθής λειτουργίας του συστήματος (OWASP Foundation, 2023). Παρότι η παρούσα εργασία δεν επικεντρώνεται σε εξειδικευμένες διαδικασίες security testing, η ασφάλεια αποτελεί βασικό χαρακτηριστικό ποιότητας στις web εφαρμογές και επηρεάζει σημαντικά τη συνολική αξιολόγηση του συστήματος.

Συνεπώς, τα μη λειτουργικά χαρακτηριστικά ποιότητας αποτελούν αναπόσπαστο μέρος της διαδικασίας διασφάλισης της ποιότητας του λογισμικού και λειτουργούν συμπληρωματικά προς τον λειτουργικό έλεγχο. Η αξιολόγηση της χρηστικότητας, της συμβατότητας και της

ασφάλειας συμβάλλει ουσιαστικά στη δημιουργία περισσότερων σταθερών, αξιόπιστων και αποδοτικών web εφαρμογών, ικανών να ανταποκριθούν στις απαιτήσεις των σύγχρονων ψηφιακών περιβαλλόντων (Sommerville, 2016; ISO/IEC, 2011).

4.5 Η Σχέση του ISO/IEC 25010 με τον Έλεγχο Λογισμικού

Το πρότυπο ISO/IEC 25010 συνδέεται άμεσα με τις διαδικασίες ελέγχου λογισμικού, καθώς παρέχει οργανωμένο πλαίσιο αξιολόγησης χαρακτηριστικών ποιότητας λογισμικού και υποστηρίζει τον σχεδιασμό και την εκτέλεση δοκιμών σε διαφορετικά επίπεδα ανάπτυξης. Μέσω του προτύπου καθίσταται δυνατός ο προσδιορισμός ποιοτικών απαιτήσεων και η αντιστοίχισή τους με συγκεκριμένες διαδικασίες testing και κριτήρια αξιολόγησης (ISO/IEC, 2011).

Στο πλαίσιο των διαδικασιών testing, τα χαρακτηριστικά ποιότητας του ISO/IEC 25010 χρησιμοποιούνται για τον καθορισμό των στόχων ελέγχου (test objectives), τον σχεδιασμό των test cases και την επιλογή κατάλληλων testing strategies. Η σύνδεση αυτή επιτρέπει στις ομάδες ανάπτυξης και testing να αξιολογούν όχι μόνο τη λειτουργικότητα της εφαρμογής αλλά και τη συνολική ποιότητα και αξιοπιστία του συστήματος (ISTQB, 2023).

Η λειτουργική καταλληλότητα (functional suitability), για παράδειγμα, αξιολογείται κυρίως μέσω διαδικασιών functional testing, acceptance testing και δοκιμών που βασίζονται στις λειτουργικές απαιτήσεις του συστήματος. Αντίστοιχα, χαρακτηριστικά όπως η αξιοπιστία και η συμβατότητα συνδέονται με system testing, regression testing και browser compatibility evaluation, ενώ η χρηστικότητα (usability) αξιολογείται κυρίως μέσω διαδικασιών manual testing και usability evaluation.

Παράλληλα, το ISO/IEC 25010 συμβάλλει στη βελτίωση της οργάνωσης και της πληρότητας των διαδικασιών testing, καθώς παρέχει περισσότερο σαφή και μετρήσιμα κριτήρια αξιολόγησης ποιότητας (Black, 2002). Μέσω του προτύπου διευκολύνεται η δημιουργία acceptance criteria και η καλύτερη σύνδεση μεταξύ επιχειρησιακών απαιτήσεων, test scenarios και αποτελεσμάτων testing (Pressman & Maxim, 2020).

Η σημασία του προτύπου είναι ιδιαίτερα αυξημένη στις web εφαρμογές, όπου η ποιότητα επηρεάζεται από πολλούς τεχνικούς και λειτουργικούς παράγοντες (ISO/IEC 25023, 2016). Η λειτουργία σε διαφορετικά περιβάλλοντα, οι απαιτήσεις ασφάλειας, η ανάγκη συνεχούς

διαθεσιμότητας και οι αυξημένες απαιτήσεις εμπειρίας χρήστη καθιστούν απαραίτητη την εφαρμογή οργανωμένων μοντέλων αξιολόγησης ποιότητας σε συνδυασμό με αποτελεσματικές testing διαδικασίες.

Επιπλέον, η σύνδεση του ISO/IEC 25010 με τις σύγχρονες πρακτικές ανάπτυξης Agile και DevOps ενισχύει τη σημασία του continuous testing και των πρακτικών automated testing (Forsgren, Humble, & Kim, 2018). Η συστηματική αξιολόγηση ποιοτικών χαρακτηριστικών μέσω automated και manual testing επιτρέπει τη διατήρηση υψηλού επιπέδου ποιότητας ακόμη και σε περιβάλλοντα συχνών αλλαγών και συνεχούς παράδοσης λογισμικού (Crispin & Gregory, 2014).

Συνεπώς, το ISO/IEC 25010 δεν αποτελεί απλώς θεωρητικό μοντέλο ποιότητας, αλλά λειτουργεί ως σημαντικό εργαλείο υποστήριξης των διαδικασιών ελέγχου λογισμικού και διασφάλισης ποιότητας λογισμικού. Η αξιοποίηση των χαρακτηριστικών του προτύπου συμβάλλει στη δημιουργία περισσότερο αξιόπιστων, ασφαλών και λειτουργικών web εφαρμογών, ικανών να ανταποκριθούν στις απαιτήσεις σύγχρονων ψηφιακών περιβαλλόντων (Sommerville, 2016).

5 Ο Έλεγχος Λογισμικού στον Κύκλο Ζωής Ανάπτυξης Λογισμικού

5.1 Ο Κύκλος Ζωής Ανάπτυξης Λογισμικού (SDLC)

Ο κύκλος ζωής ανάπτυξης λογισμικού (Software Development Life Cycle – SDLC) αποτελεί ένα οργανωμένο πλαίσιο διαδικασιών που χρησιμοποιείται για τον σχεδιασμό, την ανάπτυξη, την υλοποίηση, τη λειτουργία και τη συντήρηση πληροφοριακών συστημάτων. Ο SDLC περιγράφει τα βασικά στάδια που ακολουθούνται κατά την ανάπτυξη λογισμικού, με στόχο τη δημιουργία αξιόπιστων και ποιοτικών εφαρμογών που ανταποκρίνονται στις λειτουργικές και επιχειρησιακές απαιτήσεις των χρηστών (Sommerville, 2016).

Παρότι υπάρχουν διαφορετικές προσεγγίσεις ανάπτυξης λογισμικού, ο κύκλος ζωής ανάπτυξης περιλαμβάνει συνήθως στάδια όπως η ανάλυση απαιτήσεων, ο σχεδιασμός συστήματος, η υλοποίηση, το testing, η εγκατάσταση και η συντήρηση του λογισμικού (Royce, 1970). Τα στάδια αυτά λειτουργούν συνδυαστικά και επηρεάζουν άμεσα τη συνολική ποιότητα και αξιοπιστία του συστήματος.

Στο πλαίσιο του SDLC, ο έλεγχος λογισμικού δεν αντιμετωπίζεται ως απομονωμένη δραστηριότητα που πραγματοποιείται αποκλειστικά μετά την ολοκλήρωση της ανάπτυξης, αλλά ως συνεχής διαδικασία αξιολόγησης ποιότητας που ενσωματώνεται σε διαφορετικά στάδια ανάπτυξης λογισμικού. Η έγκαιρη εφαρμογή διαδικασιών testing συμβάλλει στον εντοπισμό defects, στη μείωση του κόστους διόρθωσης προβλημάτων και στη βελτίωση της σταθερότητας του συστήματος (Pressman & Maxim, 2020).

Η σημασία του testing στον κύκλο ζωής ανάπτυξης είναι ιδιαίτερα αυξημένη στις web εφαρμογές, όπου οι απαιτήσεις συνεχούς διαθεσιμότητας, συχνών ενημερώσεων και υποστήριξης πολλαπλών περιβαλλόντων λειτουργίας καθιστούν απαραίτητη τη συστηματική αξιολόγηση της ποιότητας λογισμικού. Για τον λόγο αυτό, οι διαδικασίες manual και automated testing εφαρμόζονται σε διαφορετικά στάδια ανάπτυξης, με στόχο τη διατήρηση υψηλού επιπέδου αξιοπιστίας και λειτουργικότητας.

Παράλληλα, η εξέλιξη των σύγχρονων πρακτικών ανάπτυξης λογισμικού έχει μεταβάλει σημαντικά τον ρόλο του testing στον SDLC. Οι προσεγγίσεις Agile και DevOps ενσωματώνουν διαδικασίες συνεχούς αξιολόγησης της ποιότητας και ενισχύουν τη συνεργασία μεταξύ των ομάδων ανάπτυξης, testing και λειτουργίας συστημάτων. Η

μετάβαση αυτή έχει ενισχύσει σημαντικά τη σημασία της αυτοματοποίησης δοκιμών και του continuous testing στις σύγχρονες web εφαρμογές (Crispin & Gregory, 2014).

Συνεπώς, ο SDLC αποτελεί βασικό πλαίσιο οργάνωσης των διαδικασιών ανάπτυξης και διασφάλισης ποιότητας λογισμικού. Η αποτελεσματική ενσωμάτωση διαδικασιών testing σε όλα τα στάδια του κύκλου ζωής ανάπτυξης συμβάλλει ουσιαστικά στη δημιουργία περισσότερο σταθερών, ασφαλών και αξιόπιστων πληροφοριακών συστημάτων.

5.2 Παραδοσιακά Μοντέλα Ανάπτυξης και Έλεγχος Λογισμικού

Οι πρώτες προσεγγίσεις ανάπτυξης λογισμικού βασίστηκαν σε γραμμικά και αυστηρά δομημένα μοντέλα ανάπτυξης, στα οποία κάθε στάδιο του κύκλου ζωής ολοκληρωνόταν πριν την έναρξη του επόμενου. Τα μοντέλα αυτά, γνωστά ως παραδοσιακά μοντέλα ανάπτυξης λογισμικού, επηρέασαν σημαντικά τον τρόπο με τον οποίο οργανώθηκαν οι διαδικασίες ελέγχου λογισμικού και διασφάλισης ποιότητας.

Ένα από τα πλέον διαδεδομένα παραδοσιακά μοντέλα ανάπτυξης λογισμικού είναι το Waterfall Model (μοντέλο καταρράκτη), το οποίο βασίζεται σε διαδοχικά στάδια ανάπτυξης, όπως η ανάλυση απαιτήσεων, ο σχεδιασμός, η υλοποίηση, το testing και η συντήρηση του συστήματος (Pressman & Maxim, 2020). Στο συγκεκριμένο μοντέλο, οι διαδικασίες testing πραγματοποιούνται κυρίως μετά την ολοκλήρωση της φάσης ανάπτυξης, γεγονός που περιορίζει τη δυνατότητα έγκαιρου εντοπισμού προβλημάτων και αυξάνει το κόστος διόρθωσης των defects σε μεταγενέστερα στάδια του έργου.

Παρότι το Waterfall Model συνέβαλε σημαντικά στην οργάνωση της ανάπτυξης λογισμικού, η αυστηρή διαδοχή των σταδίων ανάπτυξης παρουσίασε περιορισμούς σε έργα που απαιτούσαν συχνές αλλαγές ή συνεχή προσαρμογή απαιτήσεων (Boehm, 1988). Η περιορισμένη ενσωμάτωση διαδικασιών testing κατά τη διάρκεια ανάπτυξης δυσχέραινε τη γρήγορη αξιολόγηση της ποιότητας και μείωνε την ευελιξία των ομάδων ανάπτυξης και testing.

Στο πλαίσιο αντιμετώπισης των περιορισμών αυτών αναπτύχθηκε το V-Model (μοντέλο V) το οποίο αποτελεί εξέλιξη του Waterfall Model και δίνει μεγαλύτερη έμφαση στη σύνδεση μεταξύ σταδίων ανάπτυξης και διαδικασιών testing. Στο V-Model, κάθε στάδιο ανάπτυξης αντιστοιχίζεται με συγκεκριμένες δραστηριότητες ελέγχου λογισμικού, επιτρέποντας

καλύτερο σχεδιασμό test cases και περισσότερο οργανωμένη διαδικασία αξιολόγησης ποιότητας (ISTQB, 2023).

Η βασική φιλοσοφία του V-Model στηρίζεται στην έγκαιρη ενσωμάτωση διαδικασιών testing στον κύκλο ζωής ανάπτυξης λογισμικού. Για παράδειγμα, οι λειτουργικές απαιτήσεις συνδέονται με acceptance testing διαδικασίες, ο σχεδιασμός συστήματος με system testing και ο αναλυτικός σχεδιασμός με integration testing δραστηριότητες. Η προσέγγιση αυτή επιτρέπει περισσότερο συστηματική αξιολόγηση της ποιότητας και βελτιώνει την ιχνηλασιμότητα μεταξύ απαιτήσεων και διαδικασιών testing.

Παρά τα πλεονεκτήματα που προσφέρει το V-Model στη διασφάλιση ποιότητας, εξακολουθεί να βασίζεται σε σχετικά αυστηρή και δομημένη προσέγγιση ανάπτυξης. Η περιορισμένη ευελιξία του μοντέλου καθιστά δυσκολότερη την προσαρμογή σε περιβάλλοντα συνεχών αλλαγών και γρήγορων αναβαθμίσεων, χαρακτηριστικά που εμφανίζονται συχνά στις σύγχρονες web εφαρμογές.

Η εξέλιξη των απαιτήσεων ανάπτυξης λογισμικού και η ανάγκη για ταχύτερη παράδοση εφαρμογών οδήγησαν σταδιακά στην ανάπτυξη περισσότερο ευέλικτων προσεγγίσεων ανάπτυξης και testing, όπως οι Agile και DevOps πρακτικές, οι οποίες εξετάζονται στις επόμενες ενότητες (Pressman & Maxim, 2020).

5.3 Agile Μεθοδολογίες και Εξέλιξη του Ελέγχου Λογισμικού

Η αυξανόμενη πολυπλοκότητα των πληροφοριακών συστημάτων και η ανάγκη ταχύτερης ανάπτυξης λογισμικού οδήγησαν σταδιακά στην υιοθέτηση περισσότερο ευέλικτων προσεγγίσεων ανάπτυξης. Οι παραδοσιακές μέθοδοι ανάπτυξης, όπως το Waterfall Model και το V-Model, παρουσίαζαν περιορισμένη προσαρμοστικότητα σε περιβάλλοντα όπου οι απαιτήσεις μεταβάλλονταν συχνά και η συνεχής παράδοση νέων λειτουργιών αποτελούσε βασική επιχειρησιακή ανάγκη. Στο πλαίσιο αυτό αναπτύχθηκαν οι Agile (ευέλικτες) μεθοδολογίες, οι οποίες διαφοροποίησαν σημαντικά τον τρόπο οργάνωσης της ανάπτυξης λογισμικού και επηρέασαν ουσιαστικά τις διαδικασίες ελέγχου λογισμικού.

Οι Agile προσεγγίσεις βασίζονται σε επαναληπτική και σταδιακή ανάπτυξη λογισμικού, όπου η υλοποίηση πραγματοποιείται μέσω μικρών κύκλων ανάπτυξης (iterations) με συνεχή αξιολόγηση και ανατροφοδότηση (Beck et al., 2001). Η φιλοσοφία αυτή επιτρέπει

μεγαλύτερη ευελιξία στην προσαρμογή απαιτήσεων και διευκολύνει την ταχύτερη παράδοση λειτουργικών εκδόσεων λογισμικού στους τελικούς χρήστες (Crispin & Gregory, 2014).

Η μετάβαση στις Agile πρακτικές επηρέασε σημαντικά και τον ρόλο του ελέγχου λογισμικού στον κύκλο ζωής ανάπτυξης. Σε αντίθεση με τα παραδοσιακά μοντέλα, όπου το testing πραγματοποιούνταν κυρίως μετά την ολοκλήρωση της ανάπτυξης, στις Agile μεθοδολογίες οι διαδικασίες testing ενσωματώνονται συνεχώς σε όλα τα στάδια ανάπτυξης. Οι ομάδες ανάπτυξης και testing συνεργάζονται στενότερα, με στόχο τη συνεχή αξιολόγηση της ποιότητας και τον έγκαιρο εντοπισμό προβλημάτων.

Η συνεχής ενσωμάτωση testing διαδικασιών στις Agile προσεγγίσεις οδήγησε στην αυξημένη χρήση automated testing πρακτικών και εργαλείων αυτοματοποίησης. Η ανάγκη συχνών releases (εκδόσεων λογισμικού) και συνεχών αλλαγών στις web εφαρμογές καθιστά απαραίτητη την επαναλαμβανόμενη εκτέλεση tests και την ταχεία αξιολόγηση της σταθερότητας του συστήματος. Για τον λόγο αυτό, οι διαδικασίες automated testing χρησιμοποιούνται εκτεταμένα για την υποστήριξη του regression testing και της συνεχούς αξιολόγησης της ποιότητας του λογισμικού (ISTQB, 2023).

Παράλληλα, οι Agile μεθοδολογίες ενίσχυσαν τη σημασία της συνεχούς επικοινωνίας μεταξύ ομάδων ανάπτυξης, testing και επιχειρησιακών φορέων. Η ποιότητα λογισμικού αντιμετωπίζεται πλέον ως συλλογική ευθύνη της ομάδας ανάπτυξης και όχι αποκλειστικά ως αρμοδιότητα ξεχωριστού τμήματος ελέγχου ποιότητας. Η προσέγγιση αυτή συνέβαλε σημαντικά στη βελτίωση της συνεργασίας και στην ταχύτερη αντιμετώπιση προβλημάτων κατά τη διάρκεια ανάπτυξης (Crispin & Gregory, 2014).

Η εφαρμογή Agile πρακτικών είναι ιδιαίτερα διαδεδομένη στις σύγχρονες web εφαρμογές, όπου οι απαιτήσεις μεταβάλλονται διαρκώς και η ανάγκη συνεχούς αναβάθμισης λειτουργιών είναι ιδιαίτερα αυξημένη. Η δυνατότητα γρήγορης προσαρμογής αλλαγών, σε συνδυασμό με τη συνεχή αξιολόγηση ποιότητας μέσω testing διαδικασιών, καθιστούν τις Agile μεθοδολογίες ιδιαίτερα κατάλληλες για δυναμικά περιβάλλοντα ανάπτυξης λογισμικού.

Συνεπώς, οι Agile μεθοδολογίες διαφοροποίησαν ουσιαστικά τον τρόπο με τον οποίο προσεγγίζεται ο έλεγχος λογισμικού στις σύγχρονες διαδικασίες ανάπτυξης λογισμικού. Η ενσωμάτωση συνεχών testing δραστηριοτήτων και η αυξημένη χρήση automation

πρακτικών συνέβαλαν στη δημιουργία περισσότερο ευέλικτων, αξιόπιστων και ποιοτικών web εφαρμογών.

5.4 DevOps και Continuous Integration στις Διαδικασίες Testing

Η συνεχής εξέλιξη των πρακτικών ανάπτυξης λογισμικού και η αυξημένη ανάγκη ταχείας παράδοσης εφαρμογών οδήγησαν στην εμφάνιση του DevOps, μιας προσέγγισης που επιδιώκει τη στενότερη συνεργασία μεταξύ ομάδων ανάπτυξης λογισμικού και λειτουργίας συστημάτων. Η φιλοσοφία του DevOps βασίζεται στην ενοποίηση διαδικασιών ανάπτυξης, testing και deployment, με στόχο τη συνεχή παράδοση ποιοτικού λογισμικού και τη γρηγορότερη αντιμετώπιση προβλημάτων κατά τη διάρκεια ανάπτυξης και λειτουργίας των εφαρμογών (Pressman & Maxim, 2020).

Στο πλαίσιο των DevOps πρακτικών, ο έλεγχος λογισμικού αποκτά περισσότερο δυναμικό και συνεχές χαρακτήρα. Οι διαδικασίες testing δεν πραγματοποιούνται αποκλειστικά σε τελικά στάδια ανάπτυξης, αλλά ενσωματώνονται σε ολόκληρο τον κύκλο ζωής ανάπτυξης λογισμικού μέσω αυτοματοποιημένων μηχανισμών αξιολόγησης ποιότητας. Η προσέγγιση αυτή επιτρέπει τον ταχύτερο εντοπισμό defects και συμβάλλει στη διατήρηση υψηλού επιπέδου αξιοπιστίας και σταθερότητας των εφαρμογών.

Ιδιαίτερα σημαντικό ρόλο στις DevOps διαδικασίες διαδραματίζει το Continuous Integration (CI), το οποίο αναφέρεται στη συνεχή ενσωμάτωση αλλαγών κώδικα σε κοινό περιβάλλον ανάπτυξης μέσω αυτοματοποιημένων διαδικασιών build και testing. Κάθε νέα αλλαγή στον κώδικα αξιολογείται αυτόματα μέσω αυτοματοποιημένων διαδικασιών testing και διαδικασιών επαλήθευσης λειτουργικότητας, επιτρέποντας τον έγκαιρο εντοπισμό προβλημάτων πριν αυτά επηρεάσουν τη συνολική λειτουργία του συστήματος (ISTQB, 2023).

Η εφαρμογή Continuous Integration έχει αυξήσει σημαντικά τη σημασία της αυτοματοποίησης δοκιμών στις σύγχρονες web εφαρμογές. Οι automated testing διαδικασίες χρησιμοποιούνται εκτεταμένα για regression testing, system testing και επαλήθευση βασικών λειτουργιών του συστήματος μετά από κάθε αλλαγή ή ενημέρωση λογισμικού. Μέσω της αυτοματοποίησης επιτυγχάνεται ταχύτερη αξιολόγηση της

ποιότητας και καλύτερη υποστήριξη διαδικασιών συνεχούς ανάπτυξης και παράδοσης λογισμικού.

Παράλληλα, οι DevOps πρακτικές ενισχύουν τη φιλοσοφία του continuous testing, σύμφωνα με την οποία οι διαδικασίες αξιολόγησης ποιότητας εφαρμόζονται συνεχώς κατά τη διάρκεια ανάπτυξης και λειτουργίας του συστήματος. Η προσέγγιση αυτή επιτρέπει τη διαρκή παρακολούθηση της ποιότητας λογισμικού και συμβάλλει στη μείωση προβλημάτων που ενδέχεται να εμφανιστούν σε περιβάλλοντα παραγωγής (Crispin & Gregory, 2014).

Η σημασία των DevOps και Continuous Integration πρακτικών είναι ιδιαίτερα αυξημένη στις web εφαρμογές, όπου οι συχνές αλλαγές, οι συνεχείς αναβαθμίσεις και οι αυξημένες απαιτήσεις διαθεσιμότητας καθιστούν απαραίτητη την εφαρμογή αποτελεσματικών μηχανισμών testing και αυτοματοποίησης. Για τον λόγο αυτό, σύγχρονα εργαλεία automation και continuous integration χρησιμοποιούνται ευρέως για την υποστήριξη διαδικασιών testing σε δυναμικά περιβάλλοντα ανάπτυξης λογισμικού.

Συνεπώς, οι DevOps πρακτικές και το Continuous Integration έχουν μεταβάλει σημαντικά τον τρόπο με τον οποίο οργανώνονται οι διαδικασίες ελέγχου λογισμικού στις σύγχρονες web εφαρμογές (Humble & Farley, 2011). Η ενσωμάτωση αυτοματοποιημένων testing διαδικασιών στον κύκλο ανάπτυξης συμβάλλει στη δημιουργία περισσότερο σταθερών, αξιόπιστων και ποιοτικών πληροφοριακών συστημάτων.

5.5 Ο Ρόλος του Testing στις Σύγχρονες Διαδικασίες Ανάπτυξης

Στο σύγχρονο περιβάλλον ανάπτυξης λογισμικού, ο έλεγχος λογισμικού αποτελεί αναπόσπαστο στοιχείο των διαδικασιών διασφάλισης ποιότητας και δεν αντιμετωπίζεται πλέον ως απομονωμένη δραστηριότητα που πραγματοποιείται αποκλειστικά μετά την ολοκλήρωση της ανάπτυξης. Η αυξανόμενη πολυπλοκότητα των πληροφοριακών συστημάτων, οι απαιτήσεις συνεχούς διαθεσιμότητας και η ανάγκη ταχείας παράδοσης εφαρμογών έχουν ενισχύσει σημαντικά τον ρόλο του testing σε όλα τα στάδια του κύκλου ζωής ανάπτυξης λογισμικού (Pressman & Maxim, 2020).

Οι σύγχρονες web εφαρμογές λειτουργούν σε δυναμικά περιβάλλοντα όπου οι αλλαγές πραγματοποιούνται συχνά και οι απαιτήσεις των χρηστών μεταβάλλονται διαρκώς. Στο

πλαίσιο αυτό, η διατήρηση υψηλού επιπέδου ποιότητας προϋποθέτει τη συνεχή εφαρμογή διαδικασιών testing και την αποτελεσματική συνεργασία μεταξύ ομάδων ανάπτυξης, testing και λειτουργίας συστημάτων. Η ποιότητα λογισμικού αντιμετωπίζεται πλέον ως συλλογική ευθύνη, η οποία ενσωματώνεται σε ολόκληρη τη διαδικασία ανάπτυξης (Forsgren, Humble, & Kim, 2018).

Η μετάβαση σε Agile και DevOps πρακτικές ανάπτυξης έχει αυξήσει σημαντικά τη σημασία του continuous testing και της αυτοματοποίησης δοκιμών. Οι automated testing διαδικασίες επιτρέπουν την ταχεία αξιολόγηση της λειτουργικότητας και της σταθερότητας του συστήματος μετά από κάθε αλλαγή ή αναβάθμιση λογισμικού, συμβάλλοντας στον έγκαιρο εντοπισμό προβλημάτων και στη μείωση του κινδύνου εμφάνισης defects σε περιβάλλοντα παραγωγής (ISTQB, 2023).

Παράλληλα, το manual testing εξακολουθεί να διατηρεί ιδιαίτερα σημαντικό ρόλο στις διαδικασίες αξιολόγησης ποιότητας, κυρίως σε τομείς που σχετίζονται με τη χρηστικότητα, την εμπειρία χρήστη και την αξιολόγηση σύνθετων λειτουργικών σεναρίων. Η συνδυαστική εφαρμογή manual και automated testing επιτρέπει περισσότερο ολοκληρωμένη προσέγγιση διασφάλισης ποιότητας και υποστηρίζει αποτελεσματικότερα τις ανάγκες σύγχρονων web εφαρμογών.

Η ενσωμάτωση testing διαδικασιών στον κύκλο ζωής ανάπτυξης λογισμικού έχει συμβάλει ουσιαστικά στη βελτίωση της αξιοπιστίας, της σταθερότητας και της συνολικής ποιότητας των πληροφοριακών συστημάτων. Παράλληλα, η χρήση αυτοματοποιημένων εργαλείων testing και continuous integration πρακτικών επιτρέπει τη συστηματικότερη αξιολόγηση της ποιότητας και διευκολύνει τη διαχείριση σύνθετων περιβαλλόντων ανάπτυξης.

Συνεπώς, ο έλεγχος λογισμικού αποτελεί βασικό μηχανισμό υποστήριξης των σύγχρονων διαδικασιών ανάπτυξης λογισμικού και διαδραματίζει καθοριστικό ρόλο στη δημιουργία ποιοτικών και αξιόπιστων web εφαρμογών. Η συνεχής εξέλιξη των πρακτικών testing και αυτοματοποίησης αντανακλά τη διαρκώς αυξανόμενη σημασία της διασφάλισης ποιότητας στο πλαίσιο της σύγχρονης μηχανικής λογισμικού (Sommerville, 2016).

6 Τεχνικές και Στρατηγικές Ελέγχου Λογισμικού

6.1 Εισαγωγή στις Τεχνικές Ελέγχου Λογισμικού

Οι τεχνικές ελέγχου λογισμικού (software testing techniques) αποτελούν βασικό στοιχείο των διαδικασιών διασφάλισης ποιότητας, καθώς υποστηρίζουν τον συστηματικό σχεδιασμό και την αποτελεσματική εκτέλεση δοκιμών λογισμικού. Η χρήση κατάλληλων τεχνικών έλεγχου επιτρέπει την οργανωμένη αξιολόγηση της λειτουργικότητας, της αξιοπιστίας και της συνολικής συμπεριφοράς ενός πληροφοριακού συστήματος υπό διαφορετικές συνθήκες λειτουργίας (ISTQB, 2023).

Στο πλαίσιο του ελέγχου λογισμικού, οι τεχνικές έλεγχου χρησιμοποιούνται για τον σχεδιασμό test cases, την επιλογή κατάλληλων test δεδομένων και την αξιολόγηση πιθανών προβλημάτων ή defects πριν από την παράδοση του λογισμικού σε περιβάλλον παραγωγής. Η εφαρμογή οργανωμένων τεχνικών έλεγχου συμβάλλει στη βελτίωση της κάλυψης δοκιμών και στη μείωση του κινδύνου εμφάνισης σφαλμάτων κατά τη λειτουργία του συστήματος (Pressman & Maxim, 2020).

Οι τεχνικές έλεγχου διαφοροποιούνται ανάλογα με τον τρόπο αξιολόγησης του λογισμικού και το επίπεδο πρόσβασης που διαθέτει ο δοκιμαστής λογισμικού (tester) στην εσωτερική δομή του συστήματος. Ορισμένες τεχνικές επικεντρώνονται κυρίως στην αξιολόγηση της εξωτερικής λειτουργικότητας της εφαρμογής, ενώ άλλες βασίζονται στην ανάλυση της εσωτερικής δομής και του πηγαίου κώδικα. Παράλληλα, υπάρχουν τεχνικές που αξιοποιούν την εμπειρία και τη γνώση του tester για τον εντοπισμό πιθανών προβλημάτων σε πραγματικά σενάρια χρήσης.

Στις σύγχρονες web εφαρμογές, η σημασία των τεχνικών έλεγχου είναι ιδιαίτερα αυξημένη λόγω της πολυπλοκότητας των πληροφοριακών συστημάτων και της ανάγκης υποστήριξης πολλαπλών περιβαλλόντων λειτουργίας. Η σωστή επιλογή των τεχνικών έλεγχου συμβάλλει στην αποτελεσματικότερη αξιολόγηση των λειτουργικών και μη λειτουργικών απαιτήσεων, καθώς και στη βελτίωση της συνολικής ποιότητας και αξιοπιστίας των εφαρμογών.

Παράλληλα, οι τεχνικές testing συνδέονται άμεσα με τις διαδικασίες manual και automated testing. Πολλές τεχνικές μπορούν να εφαρμοστούν τόσο μέσω χειροκίνητων διαδικασιών έλεγχου όσο και μέσω εργαλείων αυτοματοποίησης δοκιμών, ανάλογα με τις απαιτήσεις

του συστήματος και τους στόχους της διαδικασίας testing. Η συνδυαστική αξιοποίηση διαφορετικών τεχνικών ελέγχου επιτρέπει περισσότερο ολοκληρωμένη προσέγγιση διασφάλισης ποιότητας και υποστηρίζει αποτελεσματικότερα τις ανάγκες σύγχρονων web εφαρμογών.

Στη βιβλιογραφία, οι τεχνικές ελέγχου κατηγοριοποιούνται συνήθως σε τεχνικές ελέγχου μαύρου κουτιού (black-box testing techniques), τεχνικές ελέγχου λευκού κουτιού (white-box testing techniques) και τεχνικές ελέγχου που βασίζονται στην εμπειρία (experience-based testing techniques), ανάλογα με το επίπεδο γνώσης και πρόσβασης του δοκιμαστή λογισμικού στην εσωτερική δομή του συστήματος (ISTQB, 2023). Οι βασικές κατηγορίες τεχνικών ελέγχου παρουσιάζονται στις επόμενες ενότητες.

6.2 Black-Box Testing Techniques

Οι τεχνικές Black-Box Testing αποτελούν μία από τις σημαντικότερες κατηγορίες τεχνικών ελέγχου λογισμικού και χρησιμοποιούνται ευρέως για την αξιολόγηση της λειτουργικότητας πληροφοριακών συστημάτων. Η βασική φιλοσοφία των τεχνικών αυτών στηρίζεται στην αξιολόγηση της συμπεριφοράς του λογισμικού χωρίς πρόσβαση στην εσωτερική δομή ή στον πηγαίο κώδικα της εφαρμογής. Η διαδικασία testing επικεντρώνεται κυρίως στις εισόδους, στις εξόδους και στην επαλήθευση της σωστής λειτουργίας του συστήματος σύμφωνα με τις λειτουργικές απαιτήσεις και τα αναμενόμενα αποτελέσματα (ISTQB, 2023; Ammann & Offutt, 2016).

Οι Black-Box Testing techniques χρησιμοποιούνται εκτεταμένα στις web εφαρμογές, καθώς επιτρέπουν την αξιολόγηση λειτουργικών σεναρίων χρήσης από την οπτική του τελικού χρήστη. Μέσω των τεχνικών αυτών εξετάζεται κατά πόσο η εφαρμογή ανταποκρίνεται σωστά σε διαφορετικές ενέργειες χρηστών, δεδομένα εισόδου και επιχειρησιακές απαιτήσεις, ανεξάρτητα από τον τρόπο υλοποίησης του εσωτερικού κώδικα.

Μία από τις πιο διαδεδομένες τεχνικές Black-Box Testing είναι το Equivalence Partitioning (διαμέριση ισοδυναμίας), κατά το οποίο τα δεδομένα εισόδου χωρίζονται σε ομάδες ισοδυναμίας με κοινά χαρακτηριστικά συμπεριφοράς. Η προσέγγιση αυτή επιτρέπει τη μείωση του αριθμού test cases, καθώς θεωρείται ότι όλα τα δεδομένα μιας ομάδας παράγουν παρόμοια αποτελέσματα κατά την εκτέλεση της εφαρμογής. Η τεχνική χρησιμοποιείται

συχνά σε φόρμες εισαγωγής δεδομένων και διαδικασίες επικύρωσης δεδομένων σε web εφαρμογές (Myers et al., 2011; Ammann & Offutt, 2016).

Εξίσου σημαντική τεχνική αποτελεί το Boundary Value Analysis (ανάλυση οριακών τιμών), το οποίο επικεντρώνεται στον έλεγχο οριακών τιμών εισόδου. Η συγκεκριμένη προσέγγιση βασίζεται στην παρατήρηση ότι πολλά defects εμφανίζονται σε ακραίες ή οριακές τιμές δεδομένων και χρησιμοποιείται ευρέως για την αξιολόγηση περιορισμών εισόδου και μηχανισμών επικύρωσης δεδομένων (ISTQB, 2023; Ammann & Offutt, 2016).

Παράλληλα, οι Decision Table Testing techniques (έλεγχος πινάκων αποφάσεων) χρησιμοποιούνται για την αξιολόγηση σύνθετων επιχειρησιακών κανόνων και λογικών συνθηκών. Μέσω decision tables καθίσταται δυνατή η συστηματική αναπαράσταση διαφορετικών συνδυασμών εισόδων και αναμενόμενων αποτελεσμάτων, γεγονός που διευκολύνει τον σχεδιασμό ολοκληρωμένων test scenarios για σύνθετες λειτουργίες web εφαρμογών (Pressman & Maxim, 2020; Ammann & Offutt, 2016).

Μία ακόμη σημαντική κατηγορία Black-Box Testing techniques είναι το State Transition Testing (έλεγχος μεταβάσεων καταστάσεων), το οποίο χρησιμοποιείται για την αξιολόγηση συστημάτων των οποίων η συμπεριφορά εξαρτάται από συγκεκριμένες καταστάσεις λειτουργίας και μεταβάσεις μεταξύ αυτών. Η τεχνική εφαρμόζεται συχνά σε διαδικασίες αυθεντικοποίησης, διαχείρισης συνεδριών (session management) και ροών εργασίας (workflows) σε web εφαρμογές, όπου διαφορετικές ενέργειες χρηστών μπορούν να επηρεάσουν τη συνολική κατάσταση του συστήματος (Ammann & Offutt, 2016).

Οι τεχνικές Black-Box Testing συνδέονται άμεσα τόσο με manual όσο και με automated testing διαδικασίες. Πολλές από τις τεχνικές αυτές χρησιμοποιούνται για τον σχεδιασμό test cases που εκτελούνται χειροκίνητα από testers, ενώ παράλληλα αποτελούν βασική βάση σχεδιασμού automated test scenarios σε εργαλεία web automation.

Συνεπώς, οι Black-Box Testing techniques αποτελούν βασικό μηχανισμό αξιολόγησης λειτουργικών απαιτήσεων και συμπεριφοράς web εφαρμογών. Η συστηματική εφαρμογή τους συμβάλλει στον αποτελεσματικότερο εντοπισμό defects και στη βελτίωση της συνολικής ποιότητας και αξιοπιστίας του λογισμικού.

6.3 White-Box Testing Techniques

Οι τεχνικές White-Box Testing αποτελούν σημαντική κατηγορία τεχνικών ελέγχου λογισμικού, οι οποίες βασίζονται στην ανάλυση της εσωτερικής δομής και του πηγαίου κώδικα της εφαρμογής. Σε αντίθεση με τις Black-Box Testing techniques, όπου η αξιολόγηση πραγματοποιείται χωρίς γνώση της εσωτερικής υλοποίησης του συστήματος, οι τεχνικές White-Box Testing προϋποθέτουν πρόσβαση στη λογική λειτουργίας και στη δομή του κώδικα του λογισμικού (Pressman & Maxim, 2020; Ammann & Offutt, 2016).

Ο βασικός στόχος των White-Box Testing techniques είναι η αξιολόγηση της ορθότητας της εσωτερικής λειτουργίας του συστήματος και ο εντοπισμός πιθανών προβλημάτων στη ροή εκτέλεσης του κώδικα. Οι τεχνικές αυτές χρησιμοποιούνται κυρίως σε επίπεδο unit testing και integration testing, όπου απαιτείται λεπτομερής έλεγχος της συμπεριφοράς επιμέρους μονάδων λογισμικού και της αλληλεπίδρασής τους.

Μία από τις πιο διαδεδομένες τεχνικές white-box testing είναι το Statement Coverage (κάλυψη εντολών), το οποίο αξιολογεί τον βαθμό στον οποίο οι εντολές του πηγαίου κώδικα εκτελούνται κατά τη διάρκεια των δοκιμών (ISTQB, 2023). Η προσέγγιση αυτή συμβάλλει στον εντοπισμό μη εκτελούμενων τμημάτων κώδικα και υποστηρίζει τη βελτίωση της κάλυψης δοκιμών (Myers et al., 2011; Ammann & Offutt, 2016).

Εξίσου σημαντική τεχνική αποτελεί το Branch Coverage (κάλυψη κλάδων), το οποίο επικεντρώνεται στην αξιολόγηση των διαφορετικών διαδρομών λογικής εκτέλεσης του προγράμματος. Η τεχνική εξετάζει κατά πόσο οι εναλλακτικές συνθήκες και αποφάσεις του κώδικα έχουν ελεγχθεί επαρκώς μέσω test cases, συμβάλλοντας στον αποτελεσματικότερο εντοπισμό λογικών σφαλμάτων (ISTQB, 2023; Ammann & Offutt, 2016).

Παράλληλα, το Path Coverage (κάλυψη διαδρομών) στοχεύει στην αξιολόγηση διαφορετικών πιθανών διαδρομών εκτέλεσης του συστήματος. Παρότι η πλήρης κάλυψη όλων των πιθανών διαδρομών είναι συχνά ιδιαίτερα δύσκολη σε σύνθετες εφαρμογές, η τεχνική αυτή επιτρέπει περισσότερο αναλυτική αξιολόγηση της συμπεριφοράς του λογισμικού και χρησιμοποιείται κυρίως σε κρίσιμα πληροφοριακά συστήματα υψηλών απαιτήσεων αξιοπιστίας (Ammann & Offutt, 2016).

Οι τεχνικές white-box testing συνδέονται στενά με τις διαδικασίες αυτοματοποίησης δοκιμών, καθώς η αξιολόγηση της κάλυψης κώδικα και η εκτέλεση επαναλαμβανόμενων δοκιμών πραγματοποιούνται συνήθως μέσω εξειδικευμένων εργαλείων testing και

πλαίσια αυτοματοποίησης (frameworks) (ISTQB, 2023). Η χρήση automated testing εργαλείων διευκολύνει τη συστηματική εκτέλεση unit tests και τη συνεχή αξιολόγηση της ποιότητας του πηγαίου κώδικα σε περιβάλλοντα Agile και DevOps ανάπτυξης (Pressman & Maxim, 2020).

Στις σύγχρονες web εφαρμογές, οι White-Box Testing techniques συμβάλλουν σημαντικά στη βελτίωση της αξιοπιστίας και της σταθερότητας του λογισμικού, ιδιαίτερα σε περιπτώσεις σύνθετης επιχειρησιακής λογικής και δυναμικής διαχείρισης δεδομένων. Η εφαρμογή τους επιτρέπει την αποτελεσματικότερη αξιολόγηση εσωτερικών μηχανισμών λειτουργίας και υποστηρίζει τη δημιουργία περισσότερο ασφαλών και αξιόπιστων πληροφοριακών συστημάτων.

Συνεπώς, οι White-Box Testing techniques αποτελούν σημαντικό μηχανισμό αξιολόγησης της εσωτερικής δομής λογισμικού και λειτουργούν συμπληρωματικά προς τις τεχνικές Black-Box Testing. Η συνδυαστική αξιοποίηση διαφορετικών testing techniques συμβάλλει στη βελτίωση της ποιότητας και της αποτελεσματικότητας των διαδικασιών ελέγχου λογισμικού στις σύγχρονες web εφαρμογές.

6.4 Experience-Based Testing Techniques

Οι Experience-Based Testing techniques (τεχνικές ελέγχου που βασίζονται στην εμπειρία) αποτελούν κατηγορία τεχνικών ελέγχου λογισμικού που βασίζονται κυρίως στη γνώση, την εμπειρία και την κριτική ικανότητα του tester. Σε αντίθεση με τις Black-Box και White-Box Testing techniques, οι οποίες στηρίζονται περισσότερο σε οργανωμένες διαδικασίες σχεδιασμού test cases και σε προκαθορισμένα κριτήρια αξιολόγησης, οι experience-based τεχνικές αξιοποιούν την πρακτική εμπειρία και τη διαισθητική προσέγγιση του tester για τον εντοπισμό πιθανών defects και προβλημάτων λειτουργίας (ISTQB, 2023).

Οι τεχνικές αυτές χρησιμοποιούνται ιδιαίτερα συχνά σε περιβάλλοντα όπου απαιτείται μεγαλύτερη ευελιξία και δυναμική αξιολόγηση της συμπεριφοράς της εφαρμογής. Στις σύγχρονες web εφαρμογές, όπου η αλληλεπίδραση χρηστών και οι λειτουργικές ροές μπορεί να παρουσιάζουν σημαντική πολυπλοκότητα, οι Experience-Based Testing techniques συμβάλλουν ουσιαστικά στον εντοπισμό προβλημάτων που ενδέχεται να μην αναδειχθούν μέσω αυστηρά δομημένων testing διαδικασιών.

Μία από τις πιο γνωστές τεχνικές της κατηγορίας αυτής είναι το Exploratory Testing (διερευνητικός έλεγχος), κατά το οποίο ο tester σχεδιάζει και εκτελεί δοκιμές ταυτόχρονα, χωρίς να περιορίζεται αποκλειστικά σε προκαθορισμένα test cases. Η διαδικασία βασίζεται στη συνεχή αλληλεπίδραση με την εφαρμογή και επιτρέπει την προσαρμογή της testing στρατηγικής ανάλογα με τη συμπεριφορά του συστήματος και τα προβλήματα που εντοπίζονται κατά τη διάρκεια της αξιολόγησης (Crispin & Gregory, 2014).

Το Exploratory Testing χρησιμοποιείται εκτεταμένα στις web εφαρμογές για την αξιολόγηση σύνθετων ροών εργασίας, μη αναμενόμενων ενεργειών χρηστών και ζητημάτων χρηστικότητας (ISTQB, 2023). Η τεχνική αυτή είναι ιδιαίτερα χρήσιμη σε περιπτώσεις όπου οι λειτουργικές απαιτήσεις μεταβάλλονται συχνά ή όπου η συμπεριφορά της εφαρμογής επηρεάζεται από δυναμικές αλληλεπιδράσεις χρηστών και δεδομένων.

Μία ακόμη σημαντική Experience-Based Testing technique είναι το Error Guessing (πρόβλεψη σφαλμάτων), το οποίο βασίζεται στην εμπειρία του tester σχετικά με συνηθισμένα σφάλματα και προβληματικά σημεία ενός συστήματος. Μέσω της τεχνικής αυτής, οι testers σχεδιάζουν δοκιμές με βάση προηγούμενη εμπειρία από defects, προβλήματα επικύρωσης, λανθασμένη διαχείριση δεδομένων ή αδυναμίες που εμφανίζονται συχνά σε αντίστοιχες εφαρμογές (Myers et al., 2011).

Οι Experience-Based Testing techniques συνδέονται κυρίως με manual testing διαδικασίες, καθώς απαιτούν ενεργή συμμετοχή και αξιολόγηση από τον tester κατά τη διάρκεια εκτέλεσης των δοκιμών. Παρότι πολλές σύγχρονες διαδικασίες testing βασίζονται σε πρακτικές αυτοματοποίησης, οι τεχνικές experience-based testing εξακολουθούν να διατηρούν ιδιαίτερα σημαντικό ρόλο στην αξιολόγηση της χρηστικότητας, της αλληλεπίδρασης των χρηστών και σύνθετων λειτουργικών σεναρίων (ISTQB, 2023).

Παράλληλα, οι τεχνικές αυτές λειτουργούν συμπληρωματικά προς περισσότερο δομημένες προσεγγίσεις ελέγχου λογισμικού και συμβάλλουν στη δημιουργία περισσότερο ολοκληρωμένης στρατηγικής διασφάλισης ποιότητας. Η συνδυαστική αξιοποίηση οργανωμένων τεχνικών ελέγχου και εμπειρικών προσεγγίσεων επιτρέπει αποτελεσματικότερη αξιολόγηση web εφαρμογών και βελτιώνει τη δυνατότητα εντοπισμού σύνθετων defects.

Συνεπώς, οι Experience-Based Testing techniques αποτελούν σημαντικό τμήμα των σύγχρονων διαδικασιών ελέγχου λογισμικού, ιδιαίτερα σε δυναμικά περιβάλλοντα web

εφαρμογών όπου η ανθρώπινη εμπειρία και η ευελιξία αξιολόγησης παραμένουν ιδιαίτερα σημαντικές για τη διασφάλιση ποιότητας λογισμικού.

6.5 Η Σημασία των Τεχνικών Ελέγχου στις Σύγχρονες Web Εφαρμογές

Οι τεχνικές ελέγχου αποτελούν βασικό στοιχείο των διαδικασιών διασφάλισης ποιότητας στις σύγχρονες web εφαρμογές, καθώς υποστηρίζουν τον οργανωμένο σχεδιασμό και την αποτελεσματική εκτέλεση δοκιμών λογισμικού. Η αυξανόμενη πολυπλοκότητα των πληροφοριακών συστημάτων, η ανάγκη συνεχών αναβαθμίσεων και οι υψηλές απαιτήσεις αξιοπιστίας και ασφάλειας καθιστούν απαραίτητη τη χρήση κατάλληλων τεχνικών testing σε όλα τα στάδια ανάπτυξης λογισμικού.

Οι σύγχρονες web εφαρμογές λειτουργούν σε δυναμικά περιβάλλοντα που περιλαμβάνουν διαφορετικούς περιηγητές ιστού, λειτουργικά συστήματα, κινητές συσκευές και σύνθετες αλληλεπιδράσεις χρηστών (Black, 2002). Στο πλαίσιο αυτό, η αξιολόγηση της λειτουργικότητας και της συμπεριφοράς του λογισμικού απαιτεί συνδυαστική εφαρμογή διαφορετικών τεχνικών ελέγχου, οι οποίες καλύπτουν τόσο λειτουργικές όσο και μη λειτουργικές απαιτήσεις ποιότητας.

Οι Black-Box Testing techniques συμβάλλουν κυρίως στην αξιολόγηση της λειτουργικότητας και της συμπεριφοράς της εφαρμογής από την οπτική του τελικού χρήστη, ενώ οι White-Box Testing techniques υποστηρίζουν την ανάλυση της εσωτερικής δομής και της λογικής λειτουργίας του συστήματος. Παράλληλα, οι Experience-Based Testing techniques επιτρέπουν περισσότερο ευέλικτη και δυναμική αξιολόγηση σύνθετων σεναρίων χρήσης, αξιοποιώντας την εμπειρία και τη γνώση των testers.

Η αποτελεσματικότητα των τεχνικών ελέγχου ενισχύεται σημαντικά μέσω της συνδυαστικής εφαρμογής manual και automated testing διαδικασιών. Οι automated testing πρακτικές επιτρέπουν την ταχεία και επαναλαμβανόμενη εκτέλεση test cases, ιδιαίτερα σε διαδικασίες regression testing και continuous integration, ενώ το manual testing παραμένει ιδιαίτερα σημαντικό για την αξιολόγηση της χρηστικότητας, της εμπειρίας χρήστη και σύνθετων λειτουργικών σεναρίων (ISTQB, 2023).

Παράλληλα, οι τεχνικές ελέγχου διαδραματίζουν σημαντικό ρόλο στις σύγχρονες Agile και DevOps διαδικασίες ανάπτυξης λογισμικού (Humble & Farley, 2011). Η συνεχής

αξιολόγηση ποιότητας και η ανάγκη ταχείας παράδοσης εφαρμογών απαιτούν αποτελεσματικές testing στρατηγικές που υποστηρίζουν continuous testing και αυτοματοποιημένη αξιολόγηση λειτουργικότητας και αξιοπιστίας (Crispin & Gregory, 2014).

Η σωστή επιλογή και εφαρμογή των τεχνικών ελέγχου συμβάλλει ουσιαστικά στη βελτίωση της ποιότητας λογισμικού, στη μείωση του κινδύνου εμφάνισης defects και στη δημιουργία περισσότερο σταθερών και αξιόπιστων web εφαρμογών. Παράλληλα, η αξιοποίηση κατάλληλων προσεγγίσεων ελέγχου λογισμικού διευκολύνει την καλύτερη οργάνωση των διαδικασιών testing και βελτιώνει την αποτελεσματικότητα των μηχανισμών διασφάλισης ποιότητας.

Συνεπώς, οι τεχνικές ελέγχου αποτελούν αναπόσπαστο μέρος των σύγχρονων διαδικασιών ελέγχου λογισμικού και υποστηρίζουν ουσιαστικά την ανάπτυξη ποιοτικών πληροφοριακών συστημάτων. Η συνδυαστική αξιοποίηση διαφορετικών τεχνικών ελέγχου και στρατηγικών testing επιτρέπει εκτενέστερη αξιολόγηση web εφαρμογών και συμβάλλει στη διατήρηση υψηλού επιπέδου ποιότητας και αξιοπιστίας λογισμικού.

7 Διαδικασίες και Οργάνωση Ελέγχου Λογισμικού

7.1 Ο Σχεδιασμός Διαδικασιών Testing

Ο σχεδιασμός διαδικασιών testing αποτελεί βασικό τμήμα της διασφάλισης ποιότητας λογισμικού και επηρεάζει άμεσα την αποτελεσματικότητα των δραστηριοτήτων ελέγχου σε ένα πληροφοριακό σύστημα. Η οργανωμένη προετοιμασία των testing διαδικασιών συμβάλλει στον συστηματικό εντοπισμό defects, στη βελτίωση της κάλυψης δοκιμών και στη διασφάλιση ότι οι λειτουργικές και μη λειτουργικές απαιτήσεις του συστήματος αξιολογούνται με επαρκή και αξιόπιστο τρόπο (ISTQB, 2023).

Στο πλαίσιο του ελέγχου λογισμικού, ο σχεδιασμός δοκιμών περιλαμβάνει τον καθορισμό των στόχων ελέγχου, την επιλογή κατάλληλων στρατηγικών ελέγχου, την οργάνωση των δραστηριοτήτων ελέγχου και τον προσδιορισμό των απαιτούμενων πόρων και διαδικασιών αξιολόγησης. Η διαδικασία αυτή επιτρέπει την καλύτερη οργάνωση των testing δραστηριοτήτων και συμβάλλει στη δημιουργία περισσότερο αποτελεσματικών μηχανισμών διασφάλισης ποιότητας (Pressman & Maxim, 2020; Graham et al., 2012)

Η ανάπτυξη της στρατηγικής ελέγχου (testing strategy) εξαρτάται από πολλούς παράγοντες, όπως η πολυπλοκότητα του συστήματος, οι λειτουργικές απαιτήσεις, οι απαιτήσεις αξιοπιστίας και ασφάλειας, καθώς και το περιβάλλον λειτουργίας της εφαρμογής (Graham et al., 2012). Στις σύγχρονες web εφαρμογές, όπου οι αλλαγές πραγματοποιούνται συχνά και η ανάγκη συνεχούς αξιολόγησης ποιότητας είναι ιδιαίτερα αυξημένη, ο σχεδιασμός testing διαδικασιών αποκτά ιδιαίτερη σημασία.

Σημαντικό στοιχείο του testing planning αποτελεί και ο καθορισμός των επιπέδων και τύπων testing που θα εφαρμοστούν κατά τη διάρκεια ανάπτυξης λογισμικού. Οι διαδικασίες αυτές μπορεί να περιλαμβάνουν δραστηριότητες functional testing, regression testing, αξιολόγησης της χρηστικότητας (usability evaluation), compatibility testing και automated testing, ανάλογα με τις απαιτήσεις του έργου και τα χαρακτηριστικά της εφαρμογής.

Παράλληλα, ο σχεδιασμός των διαδικασιών testing συνδέεται άμεσα με τη διαχείριση κινδύνων και την προτεραιοποίηση των δραστηριοτήτων ελέγχου (testing activities) (ISTQB, 2023; Graham et al., 2012). Η αξιολόγηση κρίσιμων λειτουργιών και πιθανών σημείων αστοχίας επιτρέπει την αποτελεσματικότερη κατανομή testing πόρων και

συμβάλλει στη μείωση του κινδύνου εμφάνισης σοβαρών defects σε περιβάλλον παραγωγής.

Στις Agile και DevOps προσεγγίσεις ανάπτυξης, ο σχεδιασμός testing αποκτά περισσότερο δυναμικό χαρακτήρα και ενσωματώνεται συνεχώς στον κύκλο ζωής ανάπτυξης λογισμικού (Humble & Farley, 2011). Οι testing διαδικασίες προσαρμόζονται διαρκώς στις αλλαγές απαιτήσεων και υποστηρίζουν continuous testing και automation πρακτικές, με στόχο τη διατήρηση υψηλού επιπέδου ποιότητας σε περιβάλλοντα συχνών αναβαθμίσεων και συνεχούς παράδοσης λογισμικού (Crispin & Gregory, 2014).

Συνεπώς, ο αποτελεσματικός σχεδιασμός διαδικασιών testing αποτελεί βασική προϋπόθεση για τη συστηματική αξιολόγηση της ποιότητας λογισμικού και τη δημιουργία αξιόπιστων web εφαρμογών. Η σωστή οργάνωση των διαδικασιών και των στρατηγικών ελέγχου συμβάλλει ουσιαστικά στη βελτίωση της αποτελεσματικότητας των διαδικασιών διασφάλισης ποιότητας και υποστηρίζει τη συνολική αξιοπιστία του πληροφοριακού συστήματος.

7.2 Test Scenarios και Test Cases

Τα test scenarios και τα test cases αποτελούν βασικά στοιχεία των διαδικασιών ελέγχου λογισμικού και χρησιμοποιούνται για την οργανωμένη αξιολόγηση της λειτουργικότητας και της συμπεριφοράς ενός πληροφοριακού συστήματος. Μέσω αυτών καθίσταται δυνατός ο συστηματικός σχεδιασμός δοκιμών, η καλύτερη κάλυψη λειτουργικών απαιτήσεων και η αποτελεσματικότερη διαχείριση των testing διαδικασιών κατά τη διάρκεια ανάπτυξης λογισμικού (ISTQB, 2023; Graham et al., 2012).

Τα test scenarios περιγράφουν γενικές λειτουργικές καταστάσεις ή επιχειρησιακές ροές που πρέπει να αξιολογηθούν κατά τη διαδικασία testing. Συνήθως βασίζονται στις απαιτήσεις του συστήματος και αποτυπώνουν βασικές ενέργειες ή λειτουργίες που εκτελούνται από τον τελικό χρήστη. Η χρήση test scenarios επιτρέπει την οργανωμένη προσέγγιση της διαδικασίας testing και διευκολύνει την κατανόηση των βασικών λειτουργικών απαιτήσεων της εφαρμογής.

Αντίστοιχα, τα test cases αποτελούν περισσότερο αναλυτικές περιγραφές δοκιμών και περιλαμβάνουν συγκεκριμένα βήματα εκτέλεσης, δεδομένα εισόδου και αναμενόμενα

αποτελέσματα. Μέσω των test cases αξιολογείται κατά πόσο η εφαρμογή ανταποκρίνεται σωστά στις απαιτήσεις λειτουργίας και στις προβλεπόμενες συνθήκες χρήσης. Η δημιουργία οργανωμένων test cases συμβάλλει στη βελτίωση της επαναληψιμότητας των διαδικασιών testing και υποστηρίζει τη συστηματική αξιολόγηση της ποιότητας λογισμικού (Graham et al., 2012).

Η σχέση μεταξύ test scenarios και test cases είναι ιδιαίτερα σημαντική στις web εφαρμογές, όπου σύνθετες λειτουργικές ροές και αλληλεπιδράσεις χρηστών απαιτούν περισσότερο οργανωμένο σχεδιασμό testing διαδικασιών. Ένα test scenario μπορεί να περιλαμβάνει πολλαπλά test cases που αξιολογούν διαφορετικές παραλλαγές λειτουργίας, πιθανά δεδομένα εισόδου ή εναλλακτικές συνθήκες εκτέλεσης της εφαρμογής.

Παράλληλα, τα test cases συνδέονται άμεσα με τις διαδικασίες ιχνηλασιμότητας απαιτήσεων (requirements traceability), καθώς επιτρέπουν τη συσχέτιση λειτουργικών απαιτήσεων με συγκεκριμένες δραστηριότητες testing. Η προσέγγιση αυτή διευκολύνει την αξιολόγηση της κάλυψης δοκιμών και υποστηρίζει την καλύτερη οργάνωση των διαδικασιών διασφάλισης ποιότητας (Pressman & Maxim, 2020).

Στις σύγχρονες Agile και DevOps διαδικασίες ανάπτυξης, η δημιουργία test scenarios και test cases αποκτά περισσότερο δυναμικό χαρακτήρα λόγω των συχνών αλλαγών απαιτήσεων και της συνεχούς εξέλιξης των εφαρμογών. Οι testing δραστηριότητες προσαρμόζονται διαρκώς στις αλλαγές του συστήματος και υποστηρίζουν τόσο manual όσο και automated testing διαδικασίες.

Ιδιαίτερα στις πρακτικές automated testing, τα test cases αποτελούν βασικό στοιχείο του σχεδιασμού αυτοματοποιημένων σεναρίων δοκιμών (test scripts) και διαδικασιών regression testing. Η ύπαρξη οργανωμένων και επαναχρησιμοποιήσιμων test cases διευκολύνει τη συστηματική εκτέλεση tests και συμβάλλει στη διατήρηση σταθερότητας και αξιοπιστίας στις web εφαρμογές (Myers et al., 2011).

Συνεπώς, τα test scenarios και τα test cases αποτελούν θεμελιώδη στοιχεία των διαδικασιών ελέγχου λογισμικού και διαδραματίζουν σημαντικό ρόλο στην οργάνωση και αξιολόγηση της ποιότητας λογισμικού. Η σωστή σχεδίαση και διαχείρισή τους συμβάλλει ουσιαστικά στη βελτίωση της αποτελεσματικότητας των testing διαδικασιών και στην ανάπτυξη περισσότερο αξιόπιστων πληροφοριακών συστημάτων.

7.3 Διαχείριση Defects και Αναφορά Αποτελεσμάτων Testing

Η διαχείριση defects αποτελεί βασικό τμήμα των διαδικασιών ελέγχου λογισμικού και συμβάλλει ουσιαστικά στη διασφάλιση ποιότητας λογισμικού. Κατά τη διάρκεια των testing διαδικασιών εντοπίζονται προβλήματα λειτουργίας, αποκλίσεις από τις απαιτήσεις και πιθανά σφάλματα που επηρεάζουν τη σταθερότητα, την αξιοπιστία και τη συνολική λειτουργικότητα του πληροφοριακού συστήματος. Η οργανωμένη καταγραφή και αξιολόγηση των προβλημάτων αυτών επιτρέπει την αποτελεσματικότερη διαχείριση της ποιότητας και υποστηρίζει τη διαδικασία βελτίωσης του λογισμικού.

Στο πλαίσιο του ελέγχου λογισμικού, ένα defect αναφέρεται σε ατέλεια ή πρόβλημα του συστήματος που μπορεί να οδηγήσει σε μη αναμενόμενη ή λανθασμένη συμπεριφορά της εφαρμογής. Η διαδικασία διαχείρισης defect περιλαμβάνει τον εντοπισμό, την καταγραφή, την αξιολόγηση, την προτεραιοποίηση και την παρακολούθηση των προβλημάτων μέχρι την τελική διόρθωσή τους (ISTQB, 2023).

Η οργανωμένη καταγραφή defects επιτρέπει τη σαφέστερη επικοινωνία μεταξύ ομάδων ανάπτυξης και testing και διευκολύνει τη διαδικασία ανάλυσης και επίλυσης προβλημάτων (IEEE Std 1044-2009). Συνήθως, οι αναφορές ελαττωμάτων (defect reports) περιλαμβάνουν πληροφορίες σχετικά με τη συμπεριφορά της εφαρμογής, τα βήματα αναπαραγωγής του προβλήματος, τα δεδομένα εισόδου, το αναμενόμενο αποτέλεσμα και τη σοβαρότητα του defect. Η πληρότητα και η σαφήνεια των αναφορών συμβάλλουν στη γρηγορότερη και αποτελεσματικότερη αντιμετώπιση προβλημάτων από τις ομάδες ανάπτυξης.

Ιδιαίτερα σημαντικό στοιχείο της διαδικασίας διαχείρισης defect αποτελεί η αξιολόγηση της σοβαρότητας (severity) και της προτεραιότητας (priority) των προβλημάτων. Η σοβαρότητα σχετίζεται με τον βαθμό επίδρασης ενός defect στη λειτουργία του συστήματος, ενώ η προτεραιότητα αφορά τον βαθμό προτεραιότητας που δίνεται στη διόρθωσή του με βάση επιχειρησιακές ή τεχνικές ανάγκες. Η ορθή αξιολόγηση των χαρακτηριστικών αυτών συμβάλλει στην αποτελεσματικότερη διαχείριση των πόρων των ομάδων testing και ανάπτυξης λογισμικού (Myers et al., 2011; ISTQB, 2023).

Παράλληλα, η αναφορά αποτελεσμάτων testing αποτελεί σημαντικό μηχανισμό αξιολόγησης της συνολικής ποιότητας λογισμικού. Μέσω των αναφορών ελέγχου παρουσιάζονται πληροφορίες σχετικά με την πρόοδο των testing δραστηριοτήτων, την

κάλυψη δοκιμών, τα defects που εντοπίστηκαν και τη γενικότερη κατάσταση ποιότητας του συστήματος. Οι αναφορές αυτές υποστηρίζουν τη διαδικασία λήψης αποφάσεων σχετικά με την ετοιμότητα της εφαρμογής για deployment ή περαιτέρω testing δραστηριότητες (Pressman & Maxim, 2020).

Στις σύγχρονες Agile και DevOps διαδικασίες ανάπτυξης, η διαχείριση defect και η αναφορά αποτελεσμάτων testing πραγματοποιούνται με περισσότερο δυναμικό και συνεχές τρόπο. Οι ομάδες ανάπτυξης και testing συνεργάζονται στενότερα και αξιοποιούν εργαλεία παρακολούθησης defect (defect tracking) και συνεχούς ενσωμάτωσης (continuous integration) για την ταχύτερη αναγνώριση και αντιμετώπιση προβλημάτων κατά τη διάρκεια της ανάπτυξης λογισμικού (ISTQB, 2023; Pressman & Maxim, 2020).

Στις web εφαρμογές, όπου οι αλλαγές και οι αναβαθμίσεις πραγματοποιούνται συχνά, η αποτελεσματική διαδικασία διαχείρισης defect είναι ιδιαίτερα σημαντική για τη διατήρηση υψηλού επιπέδου αξιοπιστίας και σταθερότητας. Η έγκαιρη αντιμετώπιση προβλημάτων συμβάλλει στη μείωση του κινδύνου εμφάνισης αστοχιών σε περιβάλλον παραγωγής και βελτιώνει τη συνολική εμπειρία χρήστη.

Συνεπώς, η διαχείριση defect και η οργανωμένη αναφορά αποτελεσμάτων testing αποτελούν αναπόσπαστο μέρος των διαδικασιών διασφάλισης ποιότητας λογισμικού. Η αποτελεσματική εφαρμογή τους υποστηρίζει τη δημιουργία περισσότερο αξιόπιστων και ποιοτικών web εφαρμογών και συμβάλλει στη βελτίωση της συνολικής αποτελεσματικότητας των testing διαδικασιών.

7.4 Manual Testing Διαδικασίες

Το manual testing αποτελεί μία από τις βασικότερες διαδικασίες ελέγχου λογισμικού και χρησιμοποιείται εκτεταμένα για την αξιολόγηση της λειτουργικότητας, της χρηστικότητας και της συνολικής συμπεριφοράς web εφαρμογών. Στις διαδικασίες manual testing, οι δοκιμές εκτελούνται άμεσα από testers χωρίς τη χρήση αυτοματοποιημένων scripts, επιτρέποντας περισσότερο ευέλικτη και δυναμική αξιολόγηση της εφαρμογής υπό πραγματικές συνθήκες χρήσης (ISTQB, 2023).

Η σημασία του manual testing παραμένει ιδιαίτερα αυξημένη ακόμη και σε περιβάλλοντα όπου εφαρμόζονται εκτεταμένες πρακτικές αυτοματοποίησης δοκιμών. Παρότι τα

automated testing εργαλεία επιτρέπουν την ταχεία και επαναλαμβανόμενη εκτέλεση tests, πολλές πτυχές της εμπειρίας χρήστη και της λειτουργικής συμπεριφοράς μιας εφαρμογής εξακολουθούν να απαιτούν ανθρώπινη αξιολόγηση και παρατήρηση.

Στο πλαίσιο των manual testing διαδικασιών, οι testers εκτελούν test cases και test scenarios με στόχο την αξιολόγηση της συμπεριφοράς του συστήματος, τον εντοπισμό defects και την επιβεβαίωση ότι η εφαρμογή ανταποκρίνεται στις λειτουργικές απαιτήσεις και στις προσδοκίες των χρηστών. Η διαδικασία αυτή περιλαμβάνει την αλληλεπίδραση με την εφαρμογή, την καταγραφή αποτελεσμάτων testing και την αξιολόγηση πιθανών προβλημάτων λειτουργίας ή χρηστικότητας.

Ιδιαίτερα σημαντικό ρόλο στο manual testing διαδραματίζει το Exploratory Testing, όπου ο tester αλληλεπιδρά δυναμικά με την εφαρμογή χωρίς να περιορίζεται αποκλειστικά σε προκαθορισμένα βήματα δοκιμών. Η προσέγγιση αυτή επιτρέπει την αποτελεσματικότερη αξιολόγηση σύνθετων workflows, μη αναμενόμενων συμπεριφορών και πιθανών προβλημάτων που ενδέχεται να μην εντοπιστούν μέσω αυστηρά δομημένων testing διαδικασιών (Crispin & Gregory, 2014).

Παράλληλα, το manual testing χρησιμοποιείται εκτεταμένα για την αξιολόγηση της χρηστικότητας και της εμπειρίας χρήστη στις web εφαρμογές. Η ανθρώπινη παρατήρηση είναι ιδιαίτερα σημαντική για την αναγνώριση προβλημάτων πλοήγησης, δυσνόητων διεπαφών χρήστη (interfaces) και λειτουργικών δυσκολιών που επηρεάζουν τη συνολική αλληλεπίδραση του χρήστη με το σύστημα (Crispin & Gregory, 2014).

Στις σύγχρονες Agile και DevOps διαδικασίες ανάπτυξης, το manual testing λειτουργεί συμπληρωματικά προς τις automated testing πρακτικές και ενσωματώνεται συνεχώς στις διαδικασίες αξιολόγησης ποιότητας. Οι ομάδες testing συνεργάζονται στενά με τις ομάδες ανάπτυξης και τους εμπλεκόμενους φορείς (stakeholders) για τη συνεχή αξιολόγηση των λειτουργικών απαιτήσεων και την έγκαιρη αντιμετώπιση προβλημάτων κατά τη διάρκεια της ανάπτυξης λογισμικού.

Παρά τα πλεονεκτήματα που προσφέρει, το manual testing παρουσιάζει και ορισμένους περιορισμούς, κυρίως ως προς τον χρόνο εκτέλεσης, την επαναληψιμότητα και τη δυσκολία εφαρμογής σε μεγάλης κλίμακας regression testing διαδικασίες. Για τον λόγο αυτό, στις σύγχρονες web εφαρμογές εφαρμόζεται συχνά συνδυασμός manual και automated testing,

με στόχο περισσότερο ολοκληρωμένη και αποτελεσματική προσέγγιση διασφάλισης ποιότητας (Myers et al., 2011).

Συνεπώς, το manual testing εξακολουθεί να αποτελεί ιδιαίτερα σημαντικό τμήμα των διαδικασιών ελέγχου λογισμικού και διαδραματίζει καθοριστικό ρόλο στην αξιολόγηση της λειτουργικότητας, της χρηστικότητας και της συνολικής εμπειρίας χρήστη στις σύγχρονες web εφαρμογές (Crispin & Gregory, 2014; ISTQB, 2023).

8 Automated Testing στις Web Εφαρμογές

8.1 Εισαγωγή στο Automated Testing

Το Automated Testing αποτελεί μία από τις σημαντικότερες σύγχρονες προσεγγίσεις ελέγχου λογισμικού και χρησιμοποιείται εκτεταμένα στις διαδικασίες ανάπτυξης web εφαρμογών. Η συνεχής αύξηση της πολυπλοκότητας των πληροφοριακών συστημάτων, καθώς και οι αυξημένες απαιτήσεις για ταχύτερες εκδόσεις λογισμικού και συχνές αναβαθμίσεις, έχουν ενισχύσει σημαντικά τον ρόλο της αυτοματοποίησης δοκιμών στις διαδικασίες διασφάλισης ποιότητας λογισμικού (ISTQB, 2023).

Στο πλαίσιο του ελέγχου λογισμικού, το automated testing αναφέρεται στη χρήση εργαλείων και αυτοματοποιημένων scripts για την εκτέλεση testing διαδικασιών χωρίς τη συνεχή ανθρώπινη παρέμβαση. Οι διαδικασίες αυτές επιτρέπουν την αυτοματοποιημένη εκτέλεση test cases, την επαναληπτική αξιολόγηση λειτουργιών του συστήματος και τη συστηματική επιβεβαίωση της ορθής λειτουργίας της εφαρμογής υπό διαφορετικές συνθήκες χρήσης (Pressman & Maxim, 2020).

Η σημασία του automated testing είναι ιδιαίτερα αυξημένη στις σύγχρονες web εφαρμογές, όπου οι συχνές αλλαγές κώδικα και οι συνεχείς διαδικασίες deployment απαιτούν ταχύτερη και περισσότερο επαναλαμβανόμενη αξιολόγηση της λειτουργικότητας του συστήματος. Μέσω της αυτοματοποίησης καθίσταται δυνατή η συχνή εκτέλεση regression testing διαδικασιών, η μείωση ανθρώπινων σφαλμάτων και η βελτίωση της συνολικής αποτελεσματικότητας των testing διαδικασιών (Myers et al., 2011).

Παράλληλα, το automated testing ενσωματώνεται στενά στις Agile και DevOps προσεγγίσεις ανάπτυξης λογισμικού, όπου οι testing διαδικασίες πραγματοποιούνται συνεχώς κατά τη διάρκεια του κύκλου ζωής ανάπτυξης. Η υιοθέτηση πρακτικών continuous integration και continuous delivery έχει ενισχύσει περαιτέρω τη σημασία της αυτοματοποίησης δοκιμών στις σύγχρονες διαδικασίες ανάπτυξης web εφαρμογών (Crispin & Gregory, 2014).

8.2 Πλεονεκτήματα και Περιορισμοί του Automated Testing

Η εφαρμογή automated testing προσφέρει σημαντικά πλεονεκτήματα στις διαδικασίες διασφάλισης ποιότητας λογισμικού και συμβάλλει ουσιαστικά στη βελτίωση της αποτελεσματικότητας των testing δραστηριοτήτων. Ένα από τα σημαντικότερα πλεονεκτήματα της αυτοματοποίησης είναι η δυνατότητα επαναλαμβανόμενης και ταχείας εκτέλεσης test cases χωρίς την ανάγκη συνεχούς ανθρώπινης παρέμβασης (ISTQB, 2023).

Μέσω της αυτοματοποίησης καθίσταται δυνατή η εκτέλεση μεγάλου αριθμού δοκιμών σε μικρό χρονικό διάστημα, γεγονός που βελτιώνει σημαντικά την κάλυψη testing και επιτρέπει τη συχνότερη αξιολόγηση της λειτουργικότητας των web εφαρμογών (Fewster & Graham, 1999). Επιπλέον, η επαναληψιμότητα των automated tests συμβάλλει στη μείωση ασυνεπειών και πιθανών ανθρώπινων λαθών που ενδέχεται να εμφανιστούν κατά τις διαδικασίες manual testing (Myers et al., 2011).

Ιδιαίτερα σημαντικός είναι και ο ρόλος του automated testing στις regression testing διαδικασίες. Σε περιβάλλοντα συνεχών αλλαγών και συχνών αναβαθμίσεων λογισμικού, η αυτοματοποίηση επιτρέπει την επαναλαμβανόμενη αξιολόγηση κρίσιμων λειτουργιών του συστήματος και τη γρηγορότερη αναγνώριση προβλημάτων που ενδέχεται να προκύψουν μετά από αλλαγές στον κώδικα (Pressman & Maxim, 2020).

Παρά τα σημαντικά πλεονεκτήματα, το automated testing παρουσιάζει και ορισμένους περιορισμούς. Η ανάπτυξη και συντήρηση automated test scripts απαιτεί χρόνο, τεχνική εξειδίκευση και αυξημένο αρχικό κόστος υλοποίησης. Επιπλέον, οι αυτοματοποιημένες διαδικασίες testing δεν μπορούν να αντικαταστήσουν πλήρως την ανθρώπινη αξιολόγηση σε θέματα χρηστικότητα, εμπειρίας χρήστη και δραστηριοτήτων exploratory testing (Crispin & Gregory, 2014).

Επιπλέον, οι αλλαγές στη διεπαφή χρήστη (user interface – UI) ή στη δομή των web εφαρμογών ενδέχεται να επηρεάσουν τη σταθερότητα των automated tests και να απαιτήσουν συχνές τροποποιήσεις στα test scripts. Για τον λόγο αυτό, στις σύγχρονες διαδικασίες ανάπτυξης εφαρμόζεται συνήθως συνδυασμός manual και automated testing, με στόχο περισσότερο ολοκληρωμένη και αποτελεσματική προσέγγιση διασφάλισης ποιότητας.

8.3 Frameworks και Εργαλεία Automated Testing

Η ανάπτυξη automated testing διαδικασιών βασίζεται στη χρήση εξειδικευμένων frameworks και εργαλείων αυτοματοποίησης, τα οποία υποστηρίζουν τη δημιουργία, εκτέλεση και διαχείριση automated test scripts. Τα εργαλεία αυτά επιτρέπουν τη συστηματική αυτοματοποίηση testing διαδικασιών και συμβάλλουν στη βελτίωση της αποδοτικότητας των διαδικασιών διασφάλισης ποιότητας λογισμικού.

Στις σύγχρονες web εφαρμογές χρησιμοποιούνται ευρέως frameworks όπως το Selenium, το Cypress και το Playwright για την αυτοματοποίηση των διαδικασιών ελέγχου λειτουργικότητας και αλληλεπίδρασης των χρηστών με διεπαφές web εφαρμογών. Τα frameworks αυτά υποστηρίζουν τη δημιουργία automated tests για διαφορετικούς περιηγητές ιστού και περιβάλλοντα λειτουργίας, επιτρέποντας περισσότερο ολοκληρωμένη αξιολόγηση της συμπεριφοράς των εφαρμογών (Sommerville, 2016).

Στον Πίνακα 8.1 παρουσιάζονται ενδεικτικά ορισμένα από τα πιο διαδεδομένα frameworks automated testing που χρησιμοποιούνται στις σύγχρονες web εφαρμογές.

Εργαλείο	Σκοπός Χρήσης	Βασικά Χαρακτηριστικά
Selenium	Browser automation testing	Υποστήριξη πολλών περιηγητών ιστού και γλωσσών προγραμματισμού
Cypress	Front-end και end-to-end testing	Γρήγορη εκτέλεση και εύκολη εγκατάσταση
Playwright	Automated testing σύγχρονων web εφαρμογών	Υποστήριξη πολλαπλών περιηγητών ιστού (cross-browser) και παράλληλη εκτέλεση δοκιμών (parallel execution)
JUnit	Unit testing σε Java εφαρμογές	Οργάνωση test suites και δημιουργία αναφορών (reporting)
TestNG	Advanced testing framework για Java	Παράλληλη εκτέλεση δοκιμών και αυξημένες δυνατότητες παραμετροποίησης

Πίνακας 8.1: Ενδεικτικά Frameworks Automated Testing

Όπως παρατηρείται στον Πίνακα 8.1, κάθε framework εξυπηρετεί διαφορετικές testing ανάγκες και χρησιμοποιείται ανάλογα με τις απαιτήσεις της εφαρμογής.

Παράλληλα, frameworks όπως τα JUnit και TestNG χρησιμοποιούνται εκτεταμένα για την οργάνωση, διαχείριση και εκτέλεση αυτοματοποιημένων συνόλων δοκιμών (automated test suites). Τα εργαλεία αυτά επιτρέπουν την ομαδοποίηση tests, τη δημιουργία αναφορών ελέγχου και την καλύτερη διαχείριση automated testing διαδικασιών σε περιβάλλοντα ανάπτυξης λογισμικού (Pressman & Maxim, 2020).

Η επιλογή κατάλληλου automation framework εξαρτάται από πολλούς παράγοντες, όπως οι τεχνολογίες της εφαρμογής, οι απαιτήσεις testing, το επίπεδο πολυπλοκότητας του συστήματος και οι ανάγκες συντήρησης των automated tests. Στις σύγχρονες διαδικασίες ανάπτυξης Agile και DevOps, τα frameworks αυτοματοποίησης ενσωματώνονται συχνά σε αγωγούς αυτοματοποίησης (pipelines) συνεχούς ενσωμάτωσης (continuous integration) και συνεχούς παράδοσης (continuous delivery), υποστηρίζοντας πρακτικές continuous testing και τη συστηματική αξιολόγηση της ποιότητας λογισμικού (Meszaros, 2007).

8.4 Selenium WebDriver

Το **Selenium WebDriver** αποτελεί ένα από τα πλέον διαδεδομένα εργαλεία automated testing για web εφαρμογές και χρησιμοποιείται εκτεταμένα για την αυτοματοποίηση διαδικασιών αλληλεπίδρασης με περιηγητές ιστού και λειτουργικού ελέγχου. Το εργαλείο επιτρέπει την προσομοίωση ενεργειών χρηστών σε διαφορετικούς περιηγητές ιστού και υποστηρίζει την εκτέλεση automated tests σε πολλαπλά περιβάλλοντα λειτουργίας (Selenium, 2024).

Η λειτουργία του Selenium WebDriver βασίζεται στην επικοινωνία μεταξύ των automated test scripts και του περιηγητή ιστού μέσω ειδικών προγραμμάτων οδήγησης περιηγητών (browser drivers). Μέσω αυτής της αρχιτεκτονικής καθίσταται δυνατή η αλληλεπίδραση με στοιχεία του DOM (Document Object Model), η εκτέλεση ενεργειών πλοήγησης και η αξιολόγηση της λειτουργικότητας της εφαρμογής υπό πραγματικές συνθήκες χρήσης. Ένα automated Selenium test μπορεί να ελέγχει αυτοματοποιημένα τη διαδικασία σύνδεσης (login), εισάγοντας τα στοιχεία πρόσβασης (credentials), επιλέγοντας το κουμπί σύνδεσης και επιβεβαιώνοντας την επιτυχή μετάβαση του χρήστη στο επόμενο στάδιο της εφαρμογής. Η δυνατότητα αυτή επιτρέπει τη συστηματική επανάληψη των διαδικασιών testing σε διαφορετικά περιβάλλοντα λειτουργίας και περιηγητές ιστού χωρίς ανθρώπινη παρέμβαση (Selenium, 2024).

Το Selenium υποστηρίζει πολλαπλές γλώσσες προγραμματισμού, όπως Java, Python και C#, επιτρέποντας την ανάπτυξη automated tests σε διαφορετικά περιβάλλοντα ανάπτυξης. Παράλληλα, προσφέρει δυνατότητες ενσωμάτωσης (integration) με testing frameworks και εργαλεία continuous integration, διευκολύνοντας την ενσωμάτωση διαδικασιών automated testing στις σύγχρονες πρακτικές ανάπτυξης λογισμικού (Pressman & Maxim, 2020).

Ιδιαίτερα σημαντικός είναι ο ρόλος του Selenium στις regression testing διαδικασίες, καθώς επιτρέπει την επαναλαμβανόμενη εκτέλεση test scenarios και τη συστηματική αξιολόγηση κρίσιμων λειτουργιών web εφαρμογών μετά από αλλαγές στον κώδικα. Για τον λόγο αυτό, το Selenium χρησιμοποιείται ευρέως στις σύγχρονες web εφαρμογές και αποτελεί βασικό εργαλείο αυτοματοποίησης testing διαδικασιών.

8.5 Ο Ρόλος του Automated Testing στις Σύγχρονες Web Εφαρμογές

Το automated testing διαδραματίζει καθοριστικό ρόλο στις σύγχρονες web εφαρμογές και αποτελεί βασικό στοιχείο των διαδικασιών διασφάλισης ποιότητας λογισμικού. Η συνεχής εξέλιξη των web τεχνολογιών και οι αυξημένες απαιτήσεις για γρήγορες και αξιόπιστες αναβαθμίσεις έχουν ενισχύσει σημαντικά τη σημασία της αυτοματοποίησης testing διαδικασιών στις σύγχρονες διαδικασίες ανάπτυξης λογισμικού (ISTQB, 2023).

Στα Agile και DevOps περιβάλλοντα ανάπτυξης, το automated testing ενσωματώνεται συνεχώς στις διαδικασίες continuous integration και continuous delivery, επιτρέποντας τη συστηματική αξιολόγηση της λειτουργικότητας και της σταθερότητας των εφαρμογών μετά από κάθε αλλαγή κώδικα (Humble & Farley, 2011). Η προσέγγιση αυτή συμβάλλει στη γρηγορότερη αναγνώριση defects και στη μείωση του κινδύνου εμφάνισης αστοχιών σε περιβάλλον παραγωγής (Crispin & Gregory, 2014).

Παράλληλα, η αυτοματοποίηση testing διαδικασιών επιτρέπει την καλύτερη διαχείριση σύνθετων απαιτήσεων ελέγχου (testing requirements) και την αποτελεσματικότερη υποστήριξη regression testing δραστηριοτήτων σε μεγάλης κλίμακας web εφαρμογές. Μέσω της συνεχούς εκτέλεσης automated tests καθίσταται δυνατή η διατήρηση υψηλού επιπέδου αξιοπιστίας και ποιότητας λογισμικού σε περιβάλλοντα συχνών αναβαθμίσεων και συνεχών αλλαγών.

Συνεπώς, το automated testing αποτελεί αναπόσπαστο μέρος των σύγχρονων διαδικασιών ανάπτυξης web εφαρμογών και αποτελεί πλέον βασική προϋπόθεση για την ανάπτυξη αξιόπιστων, επεκτάσιμων και ποιοτικών web εφαρμογών.

9 Περιγραφή της Εφαρμογής (Case Study)

9.1 Εισαγωγή

Στο πλαίσιο της παρούσας διπλωματικής εργασίας επιλέχθηκε ως μελέτη περίπτωσης η διαδικτυακή εφαρμογή SauceDemo, η οποία χρησιμοποιείται ευρέως για εκπαιδευτικούς και ερευνητικούς σκοπούς στον τομέα του ελέγχου λογισμικού. Η εφαρμογή προσομοιώνει ένα περιβάλλον ηλεκτρονικού εμπορίου και παρέχει ένα ελεγχόμενο πλαίσιο εκτέλεσης δοκιμών σε web εφαρμογές.

Η επιλογή κατάλληλης εφαρμογής για τη διεξαγωγή διαδικασιών testing αποτελεί σημαντικό στάδιο σε μια πειραματική μελέτη, καθώς η λειτουργικότητα, η δομή και η πολυπλοκότητα της εφαρμογής επηρεάζουν άμεσα το εύρος και την ποιότητα των δοκιμών που μπορούν να πραγματοποιηθούν. Στην παρούσα εργασία, η εφαρμογή SauceDemo αξιοποιείται ως πειραματικό περιβάλλον για την εφαρμογή τεχνικών χειροκίνητου και αυτοματοποιημένου ελέγχου λογισμικού, με στόχο την αξιολόγηση βασικών λειτουργιών μιας σύγχρονης web εφαρμογής.

Η εφαρμογή είναι δημόσια διαθέσιμη στο διαδίκτυο και χρησιμοποιείται εκτενώς σε εκπαιδευτικά προγράμματα ελέγχου λογισμικού, καθώς επιτρέπει την εκτέλεση δοκιμών χωρίς τον κίνδυνο επηρεασμού πραγματικών δεδομένων χρηστών ή παραγωγικών πληροφοριακών συστημάτων.

9.2 Γενική Περιγραφή της Εφαρμογής

Η εφαρμογή SauceDemo αποτελεί μια web εφαρμογή ηλεκτρονικού εμπορίου, η οποία προσομοιώνει τη βασική λειτουργία ενός διαδικτυακού καταστήματος. Μέσω της εφαρμογής, οι χρήστες μπορούν να εκτελούν βασικές λειτουργίες που συναντώνται σε σύγχρονα συστήματα ηλεκτρονικού εμπορίου (e-commerce), όπως η σύνδεση στο σύστημα, η προβολή προϊόντων, η διαχείριση του καλαθιού αγορών και η ολοκλήρωση της παραγγελίας.

Ειδικότερα, η εφαρμογή υποστηρίζει:

- διαδικασία σύνδεσης χρηστών (login),
- προβολή λίστας προϊόντων,

- προσθήκη και αφαίρεση προϊόντων από το καλάθι αγορών,
- ταξινόμηση προϊόντων,
- διαδικασία ολοκλήρωσης αγοράς (checkout).

Η λειτουργικότητα αυτή προσομοιώνει μια τυπική ροή χρήσης (user flow) συστήματος ηλεκτρονικού εμπορίου και καθιστά την εφαρμογή κατάλληλη για την ανάπτυξη και εκτέλεση σεναρίων δοκιμών λειτουργικότητας.

Παράλληλα, η εφαρμογή παρέχει χαρακτηριστικά που διευκολύνουν την εφαρμογή τεχνικών testing, όπως μηχανισμούς ελέγχου εγκυρότητας των δεδομένων εισόδου, διαχείριση σφαλμάτων, αλληλεπίδραση με στοιχεία της διεπαφής χρήστη (UI elements) και λειτουργίες πλοήγησης μεταξύ διαφορετικών σελίδων της εφαρμογής. Τα χαρακτηριστικά αυτά επιτρέπουν την αξιολόγηση τόσο των λειτουργικών όσο και των διεπαφικών πτυχών της εφαρμογής.

9.3 Αρχιτεκτονική και Δομή της Εφαρμογής

Η εφαρμογή SauceDemo ακολουθεί τη λογική μιας τυπικής αρχιτεκτονικής τριών επιπέδων (three-tier web architecture), η οποία χρησιμοποιείται ευρέως στις σύγχρονες διαδικτυακές εφαρμογές.

Το πρώτο επίπεδο αφορά το επίπεδο παρουσίασης (Presentation Layer), το οποίο περιλαμβάνει τη διεπαφή χρήστη μέσω της οποίας πραγματοποιείται η αλληλεπίδραση με το σύστημα. Το επίπεδο αυτό υλοποιείται μέσω web τεχνολογιών και προβάλλεται στον φυλλομετρητή του χρήστη.

Το δεύτερο επίπεδο αποτελεί το επίπεδο εφαρμογής (Application Layer), στο οποίο υλοποιείται η επιχειρησιακή λογική της εφαρμογής. Στο επίπεδο αυτό διαχειρίζονται λειτουργίες όπως η διαδικασία σύνδεσης χρήστη, η διαχείριση προϊόντων και η ολοκλήρωση αγοράς.

Το τρίτο επίπεδο αφορά το επίπεδο δεδομένων (Data Layer), το οποίο σχετίζεται με τη διαχείριση των δεδομένων που χρησιμοποιούνται από την εφαρμογή, όπως στοιχεία προϊόντων και πληροφορίες χρηστών.

Η συγκεκριμένη αρχιτεκτονική διευκολύνει την οργάνωση των λειτουργιών της εφαρμογής και υποστηρίζει αποτελεσματικά τη διαδικασία testing, καθώς επιτρέπει την αξιολόγηση της συμπεριφοράς του συστήματος σε διαφορετικά επίπεδα λειτουργικότητας.

9.4 Βασικές Λειτουργίες της Εφαρμογής

Οι βασικές λειτουργικές ενότητες της εφαρμογής που αξιοποιήθηκαν κατά τη διαδικασία testing παρουσιάζονται στις ακόλουθες υποενότητες.

9.4.1 Σύστημα Σύνδεσης Χρηστών (Login System)

Η διαδικασία σύνδεσης χρήστη αποτελεί το αρχικό στάδιο αλληλεπίδρασης με την εφαρμογή. Ο χρήστης καλείται να εισαγάγει όνομα χρήστη και κωδικό πρόσβασης προκειμένου να αποκτήσει πρόσβαση στις διαθέσιμες λειτουργίες του συστήματος.

Η λειτουργία αυτή είναι ιδιαίτερα σημαντική σε web εφαρμογές, καθώς σχετίζεται με τον έλεγχο πρόσβασης και την ταυτοποίηση χρηστών. Για τον λόγο αυτό, η διαδικασία login αποτελεί βασικό αντικείμενο λειτουργικών δοκιμών και δοκιμών επικύρωσης δεδομένων εισόδου.

9.4.2 Προβολή Προϊόντων (Product Inventory)

Μετά την επιτυχή σύνδεση, ο χρήστης μεταφέρεται στη σελίδα προβολής προϊόντων, όπου εμφανίζεται λίστα διαθέσιμων προϊόντων συνοδευόμενη από βασικές πληροφορίες, όπως όνομα, περιγραφή, τιμή και εικόνα προϊόντος.

Η συγκεκριμένη λειτουργία σχετίζεται άμεσα με την εμπειρία χρήστη και αποτελεί σημαντικό πεδίο αξιολόγησης στο πλαίσιο λειτουργικών και διεπαφικών δοκιμών.

9.4.3 Καλάθι Αγορών (Shopping Cart)

Η εφαρμογή παρέχει δυνατότητα προσθήκης προϊόντων στο καλάθι αγορών μέσω της επιλογής “Add to Cart”. Το καλάθι λειτουργεί ως ένας προσωρινός μηχανισμός αποθήκευσης των προϊόντων που επιλέγει ο χρήστης πριν την ολοκλήρωση της αγοράς.

Παράλληλα, παρέχεται δυνατότητα αφαίρεσης προϊόντων, ενημέρωσης του καλαθιού και μετάβασης στη διαδικασία checkout.

9.4.4 Διαδικασία Ολοκλήρωσης Αγοράς (Checkout)

Η διαδικασία checkout αποτελεί το τελικό στάδιο της ροής αγοράς. Σε αυτό το στάδιο ο χρήστης εισάγει βασικά στοιχεία, όπως όνομα, επώνυμο και ταχυδρομικό κώδικα, προκειμένου να ολοκληρωθεί η διαδικασία παραγγελίας.

Η λειτουργία αυτή θεωρείται ιδιαίτερα κρίσιμη σε εφαρμογές ηλεκτρονικού εμπορίου, καθώς σχετίζεται άμεσα με την ορθότητα της επιχειρησιακής λογικής και την αξιοπιστία της συνολικής διαδικασίας αγοράς.

9.5 Καταλληλότητα της Εφαρμογής για Testing

Η επιλογή της εφαρμογής SauceDemo πραγματοποιήθηκε με βάση τη δυνατότητα υποστήριξης διαφορετικών τεχνικών ελέγχου λογισμικού σε ένα σταθερό και επαναλήψιμο περιβάλλον δοκιμών.

Η εφαρμογή παρέχει ένα ολοκληρωμένο σύνολο βασικών λειτουργιών που συναντώνται σε πραγματικές web εφαρμογές ηλεκτρονικού εμπορίου, επιτρέποντας την ανάπτυξη τόσο θετικών όσο και αρνητικών σεναρίων δοκιμών. Παράλληλα, η δομή της εφαρμογής διευκολύνει την εκτέλεση τόσο χειροκίνητων όσο και αυτοματοποιημένων δοκιμών, γεγονός που την καθιστά κατάλληλη για συγκριτική αξιολόγηση διαφορετικών προσεγγίσεων testing.

Σύμφωνα με καθιερωμένες πρακτικές ελέγχου λογισμικού, η χρήση ελεγχόμενων περιβαλλόντων δοκιμών συμβάλλει στη βελτίωση της αξιοπιστίας των αποτελεσμάτων και στην επαναληψιμότητα της διαδικασίας testing. Η εφαρμογή SauceDemo χρησιμοποιείται εκτενώς σε εκπαιδευτικά και ερευνητικά περιβάλλοντα ακριβώς λόγω της δυνατότητας προσομοίωσης ρεαλιστικών σεναρίων χρήσης χωρίς περιορισμούς πρόσβασης ή εξάρτησης από πραγματικά δεδομένα χρηστών.

Επιπλέον, η εφαρμογή επιτρέπει τη μετατροπή χειροκίνητων test cases σε αυτοματοποιημένα σενάρια, γεγονός που υποστηρίζει τον βασικό στόχο της παρούσας

εργασίας, δηλαδή τη συγκριτική προσέγγιση μεταξύ manual testing και automated testing σε web εφαρμογές.

9.6 Συμπεράσματα

Η εφαρμογή SauceDemo αποτελεί ένα κατάλληλο και ελεγχόμενο περιβάλλον για την εφαρμογή τεχνικών ελέγχου λογισμικού σε web εφαρμογές. Η ύπαρξη βασικών λειτουργιών ηλεκτρονικού εμπορίου, σε συνδυασμό με τη δυνατότητα εκτέλεσης πολλαπλών σεναρίων δοκιμών, επιτρέπει την αξιολόγηση διαφορετικών προσεγγίσεων testing σε κοινό περιβάλλον αναφοράς.

Η χρήση ενός κοινού και ελεγχόμενου περιβάλλοντος δοκιμών επιτρέπει την αντικειμενική σύγκριση των αποτελεσμάτων μεταξύ manual testing και automated testing. Μέσω της εφαρμογής SauceDemo καθίσταται δυνατή η ανάπτυξη επαναλήψιμων σεναρίων δοκιμών, τα οποία μπορούν να αξιοποιηθούν τόσο στο πλαίσιο manual testing όσο και κατά τη διαδικασία αυτοματοποίησης των δοκιμών. Με τον τρόπο αυτό, η εφαρμογή λειτουργεί ως ενιαίο πειραματικό περιβάλλον αναφοράς για τη συγκριτική μελέτη των δύο προσεγγίσεων testing που εξετάζονται στην παρούσα εργασία.

Στα επόμενα κεφάλαια παρουσιάζονται το περιβάλλον δοκιμών, η μεθοδολογία που ακολουθήθηκε, καθώς και η διαδικασία σχεδιασμού και εκτέλεσης των test cases που χρησιμοποιήθηκαν στο πλαίσιο της παρούσας μελέτης.

10 Περιβάλλον Δοκιμών και Μεθοδολογία Testing

10.1 Εισαγωγή

Η διαδικασία δοκιμών λογισμικού αποτελεί βασικό στάδιο στον κύκλο ζωής ανάπτυξης λογισμικού (Software Development Life Cycle – SDLC), καθώς συμβάλλει στον εντοπισμό σφαλμάτων και στη διασφάλιση της ποιότητας του τελικού προϊόντος. Στο πλαίσιο της παρούσας διπλωματικής εργασίας πραγματοποιήθηκε μια συστηματική διαδικασία δοκιμών στη web εφαρμογή SauceDemo, με στόχο την αξιολόγηση της λειτουργικότητας και της συμπεριφοράς της εφαρμογής σε διαφορετικά σενάρια χρήσης.

Το παρόν κεφάλαιο παρουσιάζει το περιβάλλον μέσα στο οποίο πραγματοποιήθηκαν οι δοκιμές, καθώς και τη μεθοδολογία που ακολουθήθηκε για τον σχεδιασμό και την εκτέλεση των test cases. Η αναλυτική περιγραφή των συνθηκών δοκιμών είναι ιδιαίτερα σημαντική, καθώς συμβάλλει στην επαναληψιμότητα της διαδικασίας testing και στην αξιοπιστία των αποτελεσμάτων που προέκυψαν κατά την πειραματική διαδικασία.

10.2 Περιβάλλον Δοκιμών (Testing Environment)

Το περιβάλλον δοκιμών περιλαμβάνει το σύνολο των τεχνολογικών παραμέτρων και εργαλείων που χρησιμοποιούνται για την εκτέλεση των δοκιμών λογισμικού. Η ορθή διαμόρφωση του περιβάλλοντος δοκιμών είναι απαραίτητη για την αξιόπιστη αξιολόγηση της λειτουργικότητας και της συμπεριφοράς μιας web εφαρμογής.

Στην παρούσα εργασία οι δοκιμές πραγματοποιήθηκαν σε περιβάλλον προσωπικού υπολογιστή με λειτουργικό σύστημα Windows 11 και χρήση του φυλλομετρητή Google Chrome. Η προσέγγιση που ακολουθήθηκε βασίστηκε σε manual functional testing με εφαρμογή τεχνικών Black Box Testing. Η εφαρμογή SauceDemo χρησιμοποιήθηκε μέσω της δημόσια διαθέσιμης διαδικτυακής διεύθυνσης <https://www.saucedemo.com>.

Η χρήση σύγχρονου περιηγητή ιστού κρίθηκε απαραίτητη για τη σωστή απόδοση των στοιχείων της διεπαφής χρήστη και την ομαλή λειτουργία της εφαρμογής. Παράλληλα, οι δοκιμές πραγματοποιήθηκαν σε σταθερό περιβάλλον σύνδεσης στο διαδίκτυο, ώστε να περιοριστούν εξωτερικοί παράγοντες που θα μπορούσαν να επηρεάσουν τα αποτελέσματα της διαδικασίας testing.

10.3 Τύπος Δοκιμών

Στο πλαίσιο της παρούσας μελέτης εφαρμόστηκε η μέθοδος Manual Functional Testing. Η συγκεκριμένη προσέγγιση βασίζεται στη χειροκίνητη εκτέλεση δοκιμών από τον tester χωρίς τη χρήση αυτοματοποιημένων εργαλείων.

Κατά τη διαδικασία αυτή, ο tester αλληλεπιδρά με την εφαρμογή με τρόπο αντίστοιχο ενός πραγματικού χρήστη, εκτελώντας προκαθορισμένα σενάρια χρήσης και αξιολογώντας κατά πόσο η συμπεριφορά του συστήματος ανταποκρίνεται στα αναμενόμενα αποτελέσματα.

Το manual testing χρησιμοποιείται ευρέως κατά τα αρχικά στάδια αξιολόγησης μιας εφαρμογής, καθώς επιτρέπει την αναλυτική παρατήρηση της διεπαφής χρήστη, την αξιολόγηση της εμπειρίας χρήστη και τον εντοπισμό λειτουργικών ή διεπαφικών προβλημάτων που είναι πιθανό να μην εντοπιστούν εύκολα μέσω αυτοματοποιημένων διαδικασιών.

10.3.1 Σύγκριση Manual και Automated Testing

Η επιλογή του manual testing στην παρούσα εργασία βασίστηκε στα χαρακτηριστικά της συγκεκριμένης μεθόδου και στην ανάγκη αξιολόγησης της εφαρμογής από την οπτική του τελικού χρήστη. Ωστόσο, η διαδικασία testing σε σύγχρονες web εφαρμογές περιλαμβάνει συχνά συνδυασμό χειροκίνητων και αυτοματοποιημένων δοκιμών.

Το manual testing παρέχει αυξημένη ευελιξία κατά την εκτέλεση των δοκιμών και επιτρέπει την παρατήρηση μη αναμενόμενων συμπεριφορών της εφαρμογής, ιδιαίτερα σε περιπτώσεις exploratory testing και αξιολόγησης της εμπειρίας χρήστη. Αντίθετα, το automated testing χαρακτηρίζεται από υψηλή ταχύτητα εκτέλεσης, δυνατότητα επαναληψιμότητας και αποτελεσματική υποστήριξη επαναλαμβανόμενων δοκιμών, όπως τα regression tests.

Σύμφωνα με τις κατευθυντήριες πρακτικές του International Software Testing Qualifications Board (ISTQB), οι δύο προσεγγίσεις λειτουργούν συμπληρωματικά και όχι ανταγωνιστικά. Το manual testing είναι περισσότερο κατάλληλο για διερευνητικές και διεπαφικές δοκιμές, ενώ το automated testing προσφέρει σημαντικά πλεονεκτήματα σε περιβάλλοντα συνεχών επαναλαμβανόμενων δοκιμών και μεγάλης κλίμακας εφαρμογών.

Στο πλαίσιο της παρούσας εργασίας, το manual testing αποτέλεσε το αρχικό στάδιο της διαδικασίας αξιολόγησης της εφαρμογής. Στη συνέχεια, επιλεγμένα test cases χρησιμοποιήθηκαν ως βάση για την ανάπτυξη αυτοματοποιημένων σεναρίων δοκιμών, επιτρέποντας τη συγκριτική αξιολόγηση των δύο προσεγγίσεων testing σε κοινό περιβάλλον εφαρμογής.

10.4 Προσέγγιση Black Box Testing

Η προσέγγιση δοκιμών που χρησιμοποιήθηκε στην παρούσα εργασία βασίζεται στη μεθοδολογία Black Box Testing. Στην προσέγγιση αυτή, ο tester δεν έχει πρόσβαση στην εσωτερική δομή ή στον πηγαίο κώδικα της εφαρμογής, αλλά αξιολογεί αποκλειστικά τη συμπεριφορά του συστήματος μέσω των δεδομένων εισόδου και των αποτελεσμάτων που παράγονται.

Η διαδικασία αξιολόγησης επικεντρώνεται στη λειτουργικότητα της εφαρμογής, στην ορθότητα των αποτελεσμάτων και στη συμπεριφορά της διεπαφής χρήστη κατά την εκτέλεση συγκεκριμένων ενεργειών. Με τον τρόπο αυτό, η προσέγγιση Black Box Testing προσομοιώνει την εμπειρία ενός πραγματικού χρήστη που αλληλεπιδρά με την εφαρμογή χωρίς γνώση της εσωτερικής υλοποίησης του συστήματος.

Η συγκεκριμένη μέθοδος θεωρείται ιδιαίτερα κατάλληλη για web εφαρμογές, καθώς επιτρέπει την αξιολόγηση κρίσιμων λειτουργιών, όπως η διαδικασία σύνδεσης χρηστών, η διαχείριση δεδομένων εισόδου και η πλοήγηση μεταξύ διαφορετικών λειτουργικών ενότητων της εφαρμογής.

10.5 Διαδικασία Σχεδιασμού Test Cases

Ο σχεδιασμός των test cases αποτελεί ένα από τα σημαντικότερα στάδια της διαδικασίας testing, καθώς καθορίζει τη δομή και την οργάνωση των σεναρίων δοκιμών που θα εκτελεστούν κατά την αξιολόγηση της εφαρμογής.

Στην παρούσα εργασία, τα test cases σχεδιάστηκαν με στόχο την κάλυψη βασικών λειτουργιών της εφαρμογής SauceDemo, όπως η διαδικασία σύνδεσης χρηστών, η διαχείριση προϊόντων, η λειτουργία του καλαθιού αγορών και η ολοκλήρωση αγοράς.

Κάθε test case περιλάμβανε μοναδικό αναγνωριστικό αριθμό (Test Case ID), περιγραφή της δοκιμής, βήματα εκτέλεσης και αναμενόμενο αποτέλεσμα. Η συγκεκριμένη δομή συνέβαλε στην οργανωμένη εκτέλεση των δοκιμών και διευκόλυνε τη διαδικασία καταγραφής και αξιολόγησης των αποτελεσμάτων.

Η ανάπτυξη δομημένων test cases είναι ιδιαίτερα σημαντική σε διαδικασίες ελέγχου λογισμικού καθώς επιτρέπει τη συστηματική παρακολούθηση της διαδικασίας δοκιμών και ενισχύει την επαναληψιμότητα της μεθοδολογίας που ακολουθείται.

10.6 Διαδικασία Εκτέλεσης Δοκιμών

Η διαδικασία εκτέλεσης των δοκιμών πραγματοποιήθηκε σε διαδοχικά στάδια, σύμφωνα με τα σενάρια δοκιμών που είχαν σχεδιαστεί προηγουμένως. Αρχικά πραγματοποιήθηκε πρόσβαση στην εφαρμογή και στη συνέχεια εκτελέστηκαν οι προβλεπόμενες ενέργειες που αντιστοιχούσαν σε κάθε test case.

Κατά τη διάρκεια της διαδικασίας testing καταγράφηκαν τα αποτελέσματα κάθε δοκιμής, οι αποκλίσεις από τα αναμενόμενα αποτελέσματα, καθώς και παρατηρήσεις που σχετίζονταν με τη συμπεριφορά της εφαρμογής και της διεπαφής χρήστη.

Η συστηματική καταγραφή των αποτελεσμάτων συνέβαλε στην οργανωμένη αξιολόγηση της λειτουργικότητας της εφαρμογής και διευκόλυνε τη μετέπειτα ανάλυση των ευρημάτων που προέκυψαν από τη διαδικασία testing.

10.7 Καταγραφή Σφαλμάτων (Defect Reporting)

Η καταγραφή σφαλμάτων αποτελεί βασικό μέρος της διαδικασίας ελέγχου λογισμικού, καθώς επιτρέπει τη συστηματική αναφορά προβλημάτων που εντοπίζονται κατά την εκτέλεση των δοκιμών.

Στην παρούσα εργασία, για κάθε σφάλμα που εντοπίστηκε καταγράφηκαν πληροφορίες όπως η περιγραφή του προβλήματος, τα βήματα αναπαραγωγής, η αναμενόμενη και η πραγματική συμπεριφορά της εφαρμογής, καθώς και το επίπεδο σοβαρότητας (severity) του προβλήματος.

Η διαδικασία αυτή διευκολύνει την αξιολόγηση της επίδρασης κάθε σφάλματος στη λειτουργικότητα της εφαρμογής και υποστηρίζει τη συστηματική ανάλυση των αποτελεσμάτων testing. Παράλληλα, η κατηγοριοποίηση των σφαλμάτων συμβάλλει στην αποτελεσματικότερη διαχείριση και προτεραιοποίηση των προβλημάτων κατά τη διαδικασία ανάπτυξης λογισμικού.

10.8 Εργαλεία Υποστήριξης Testing

Κατά τη διαδικασία σχεδιασμού και εκτέλεσης των δοκιμών χρησιμοποιούνται συχνά εργαλεία που υποστηρίζουν την οργάνωση των test cases, την καταγραφή αποτελεσμάτων και τη διαχείριση σφαλμάτων.

Στην παρούσα εργασία, η καταγραφή των test cases και των αποτελεσμάτων πραγματοποιήθηκε μέσω δομημένων πινάκων καταγραφής. Ωστόσο, στη βιομηχανία λογισμικού χρησιμοποιούνται εξειδικευμένα εργαλεία testing, όπως το TestRail και το JIRA, τα οποία παρέχουν δυνατότητες οργάνωσης δοκιμών, παρακολούθησης σφαλμάτων και διαχείρισης ροών εργασίας testing (testing workflows).

Επιπλέον, για την υλοποίηση αυτοματοποιημένων δοκιμών χρησιμοποιούνται εργαλεία όπως το Selenium, το οποίο επιτρέπει την ανάπτυξη και εκτέλεση αυτοματοποιημένων σεναρίων σε web εφαρμογές.

Η επιλογή κατάλληλων εργαλείων testing εξαρτάται από τις απαιτήσεις του έργου, το είδος των δοκιμών και τον βαθμό αυτοματοποίησης που επιδιώκεται σε κάθε διαδικασία ελέγχου λογισμικού.

10.9 Συμπεράσματα

Στο κεφάλαιο αυτό παρουσιάστηκαν το περιβάλλον δοκιμών και η μεθοδολογία που χρησιμοποιήθηκε για την αξιολόγηση της εφαρμογής SauceDemo. Η εφαρμογή τεχνικών Manual Functional Testing σε συνδυασμό με την προσέγγιση Black Box Testing επέτρεψε την αξιολόγηση της λειτουργικότητας της εφαρμογής από την οπτική ενός τελικού χρήστη.

Η συστηματική οργάνωση των test cases και η καταγραφή των αποτελεσμάτων συνέβαλαν στη δημιουργία ενός ελεγχόμενου πλαισίου δοκιμών, το οποίο υποστηρίζει τόσο την

αξιολόγηση της λειτουργικότητας της εφαρμογής όσο και τη μετέπειτα συγκριτική ανάλυση μεταξύ manual και automated testing.

Στο επόμενο κεφάλαιο παρουσιάζονται τα test scenarios και τα test cases που σχεδιάστηκαν για την αξιολόγηση των βασικών λειτουργιών της εφαρμογής, καθώς και η διαδικασία συστηματικής καταγραφής των αποτελεσμάτων testing.

11 Σχεδιασμός Test Scenarios και Test Cases

11.1 Εισαγωγή

Ο σχεδιασμός των test cases αποτελεί ένα από τα σημαντικότερα στάδια της διαδικασίας ελέγχου λογισμικού, καθώς καθορίζει τον τρόπο με τον οποίο θα αξιολογηθεί η λειτουργικότητα μιας εφαρμογής. Μέσω της ανάπτυξης οργανωμένων σεναρίων δοκιμών καθίσταται δυνατή η συστηματική επαλήθευση της συμπεριφοράς του συστήματος υπό διαφορετικές συνθήκες χρήσης.

Στο πλαίσιο της παρούσας εργασίας σχεδιάστηκε ένα σύνολο test cases με στόχο την αξιολόγηση βασικών λειτουργιών της εφαρμογής SauceDemo. Τα σενάρια δοκιμών καλύπτουν κρίσιμες λειτουργικές ενότητες της εφαρμογής, όπως η διαδικασία σύνδεσης χρηστών, η διαχείριση προϊόντων, η λειτουργία του καλαθιού αγορών, η ολοκλήρωση αγοράς και η πλοήγηση μεταξύ των διαφορετικών σελίδων της εφαρμογής.

Η διαδικασία σχεδιασμού των δοκιμών βασίστηκε σε αρχές functional και Black Box Testing, δίνοντας έμφαση στην αξιολόγηση της συμπεριφοράς της εφαρμογής από την οπτική του τελικού χρήστη. Για τον λόγο αυτό, τα test cases αναπτύχθηκαν με τρόπο που να προσομοιώνει ρεαλιστικά σενάρια αλληλεπίδρασης με τη web εφαρμογή.

Παράλληλα, κατά τον σχεδιασμό των δοκιμών λήφθηκαν υπόψη τόσο θετικά όσο και αρνητικά σενάρια χρήσης, ώστε να καταστεί δυνατή η αξιολόγηση της συμπεριφοράς της εφαρμογής σε περιπτώσεις έγκυρων αλλά και μη έγκυρων δεδομένων εισόδου. Η προσέγγιση αυτή επιτρέπει την πληρέστερη αξιολόγηση της αξιοπιστίας και της λειτουργικότητας του συστήματος.

Στις επόμενες ενότητες παρουσιάζονται η μεθοδολογική προσέγγιση που ακολουθήθηκε για τον σχεδιασμό των δοκιμών, τα βασικά test scenarios, καθώς και τα αναλυτικά test cases που χρησιμοποιήθηκαν κατά τη διαδικασία manual testing της εφαρμογής.

11.1.1 Μεθοδολογική Προσέγγιση Σχεδιασμού Δοκιμών

Ο σχεδιασμός των test cases πραγματοποιήθηκε με βάση τις βασικές αρχές του functional και Black Box Testing, με στόχο την αξιολόγηση της συμπεριφοράς της εφαρμογής χωρίς πρόσβαση στον εσωτερικό κώδικα ή στην αρχιτεκτονική υλοποίησης του συστήματος. Η

διαδικασία σχεδιασμού επικεντρώθηκε στην αλληλεπίδραση του χρήστη με την εφαρμογή και στην επαλήθευση της ορθής λειτουργίας των βασικών λειτουργικών ενοτήτων.

Για την ανάπτυξη των σεναρίων δοκιμών λήφθηκαν υπόψη οι κύριες λειτουργίες της εφαρμογής SauceDemo, όπως η διαδικασία σύνδεσης χρηστών, η προβολή προϊόντων, η διαχείριση καλαθιού αγορών, η διαδικασία ολοκλήρωσης αγοράς και οι λειτουργίες πλοήγησης μεταξύ διαφορετικών σελίδων της εφαρμογής.

Η διαδικασία σχεδιασμού των δοκιμών βασίστηκε στη δημιουργία τόσο θετικών όσο και αρνητικών σεναρίων χρήσης. Τα θετικά test cases σχεδιάστηκαν με στόχο την επιβεβαίωση της ορθής λειτουργίας της εφαρμογής υπό κανονικές συνθήκες χρήσης, ενώ τα αρνητικά test cases χρησιμοποιήθηκαν για την αξιολόγηση της συμπεριφοράς του συστήματος σε περιπτώσεις μη έγκυρων ή ελλιπών δεδομένων εισόδου.

Παράλληλα, δόθηκε έμφαση στην κάλυψη διαφορετικών κατηγοριών λειτουργικότητας της εφαρμογής, ώστε να επιτευχθεί μεγαλύτερο εύρος αξιολόγησης κατά τη διαδικασία testing. Τα test cases οργανώθηκαν σε επιμέρους ομάδες λειτουργιών, επιτρέποντας τη συστηματική εκτέλεση και ανάλυση των αποτελεσμάτων.

Η μεθοδολογική αυτή προσέγγιση συνέβαλε στη δημιουργία ενός οργανωμένου και επαναλήψιμου πλαισίου δοκιμών, το οποίο μπορεί να αξιοποιηθεί τόσο για χειροκίνητη εκτέλεση δοκιμών όσο και ως βάση για την ανάπτυξη αυτοματοποιημένων test scripts στο πλαίσιο της συγκριτικής αξιολόγησης manual και automated testing που πραγματοποιείται στην παρούσα εργασία.

11.2 Test Scenarios

Τα test scenarios που σχεδιάστηκαν στο πλαίσιο της παρούσας εργασίας καλύπτουν τις βασικές λειτουργικές ενότητες της εφαρμογής SauceDemo και οργανώθηκαν με βάση τις κύριες λειτουργίες της εφαρμογής. Η κατηγοριοποίηση των σεναρίων δοκιμών πραγματοποιήθηκε με στόχο τη συστηματική αξιολόγηση της λειτουργικότητας της εφαρμογής και τη διευκόλυνση της διαδικασίας εκτέλεσης και ανάλυσης των αποτελεσμάτων.

Τα σενάρια δοκιμών περιλαμβάνουν τόσο θετικές όσο και αρνητικές περιπτώσεις χρήσης, επιτρέποντας την αξιολόγηση της συμπεριφοράς του συστήματος υπό διαφορετικές συνθήκες αλληλεπίδρασης με τον χρήστη.

Ο Πίνακας 11.1 παρουσιάζει τις βασικές κατηγορίες test scenarios που χρησιμοποιήθηκαν κατά τη διαδικασία manual testing της εφαρμογής

Scenario ID	Κατηγορία Δοκιμών	Περιγραφή
TS01	Login Functionality	Έλεγχος σύνδεσης χρηστών με έγκυρα και μη έγκυρα στοιχεία
TS02	Product Inventory	Έλεγχος προβολής και διαχείρισης προϊόντων
TS03	Shopping Cart Functionality	Έλεγχος λειτουργιών καλαθιού αγορών
TS04	Checkout Process	Έλεγχος διαδικασίας ολοκλήρωσης αγοράς
TS05	Navigation Testing	Έλεγχος πλοήγησης μεταξύ σελίδων
TS06	Input Field Validation	Έλεγχος λειτουργιών εκκαθάρισης και διαχείρισης πεδίων εισαγωγής δεδομένων

Πίνακας 11.1: Κατηγορίες Test Scenarios

Η κατηγοριοποίηση αυτή συνέβαλε στην καλύτερη οργάνωση της διαδικασίας testing και επέτρεψε την ανάπτυξη ενός πιο συστηματικού πλαισίου αξιολόγησης της εφαρμογής. Παράλληλα, η δομή αυτή διευκόλυνε τη μεταγενέστερη μετατροπή επιλεγμένων test cases σε αυτοματοποιημένα σενάρια δοκιμών στο πλαίσιο του automated testing.

11.3 Συνοπτικός Πίνακας Test Cases

Στην παρούσα ενότητα παρουσιάζεται συνοπτικά το σύνολο των test cases που σχεδιάστηκαν για την αξιολόγηση της εφαρμογής SauceDemo. Τα test cases οργανώθηκαν με βάση τα αντίστοιχα test scenarios και καλύπτουν βασικές λειτουργικές περιοχές της εφαρμογής.

Η οργανωμένη καταγραφή των test cases συμβάλλει στη συστηματική εκτέλεση των δοκιμών και διευκολύνει τη διαδικασία παρακολούθησης των αποτελεσμάτων testing.

Ο Πίνακας 11.2 παρουσιάζει συνοπτικά το σύνολο των test cases που σχεδιάστηκαν για την αξιολόγηση της εφαρμογής SauceDemo. Τα test cases οργανώθηκαν με βάση τα αντίστοιχα test scenarios και καλύπτουν βασικές λειτουργικές περιοχές της εφαρμογής.

Test Case ID	Scenario ID	Περιγραφή Δοκιμής
TC01	TS01	Επιτυχής σύνδεση χρήστη (Successful Login)
TC02	TS01	Λανθασμένος κωδικός πρόσβασης (Invalid Password)
TC03	TS01	Λανθασμένο όνομα χρήστη (Invalid Username)
TC04	TS01	Κενό όνομα χρήστη (Empty Username)
TC05	TS01	Κενός κωδικός πρόσβασης (Empty Password)
TC06	TS01	Κενά πεδία σύνδεσης (Empty Fields)
TC07	TS02	Εμφάνιση προϊόντων (Display Products)
TC08	TS02	Εμφάνιση εικόνων προϊόντων (Product Images)
TC09	TS02	Προβολή περιγραφής προϊόντων (Product Description)
TC10	TS02	Ταξινόμηση προϊόντων Α-Ζ (Sort Products A-Z)
TC11	TS02	Ταξινόμηση προϊόντων Ζ-Α (Sort Products Z-A)
TC12	TS02	Ταξινόμηση προϊόντων κατά αύξουσα τιμή (Low to High)
TC13	TS02	Ταξινόμηση προϊόντων κατά φθίνουσα τιμή (High to Low)
TC14	TS03	Προσθήκη προϊόντος στο καλάθι (Add Product to Cart)
TC15	TS03	Προσθήκη πολλαπλών προϊόντων (Add Multiple Products)
TC16	TS03	Αφαίρεση προϊόντος από τη σελίδα προϊόντων (Remove Product from Products Page)
TC17	TS03	Αφαίρεση προϊόντος από το καλάθι (Remove Product from Cart)
TC18	TS03	Ενημέρωση μετρητή καλαθιού (Cart Badge Update)
TC19	TS03	Προβολή καλαθιού αγορών (View Shopping Cart)
TC20	TS05	Πλοήγηση «Continue Shopping» (Continue Shopping)
TC21	TS04	Έναρξη διαδικασίας checkout (Start Checkout Process)
TC22	TS04	Έγκυρα στοιχεία checkout (Valid Checkout Information)
TC23	TS04	Checkout με κενό καλάθι (Checkout with Empty Cart)
TC24	TS04	Κενό όνομα χρήστη στο checkout (Empty First Name)
TC25	TS04	Κενό επώνυμο στο checkout (Empty Last Name)
TC26	TS04	Κενός ταχυδρομικός κώδικας (Empty Postal Code)
TC27	TS04	Λειτουργία «Cancel» στο checkout (Cancel Checkout)
TC28	TS04	Ολοκλήρωση παραγγελίας (Complete Checkout)
TC29	TS04	Ακύρωση διαδικασίας checkout (Cancel Checkout Process)
TC30	TS05	Λειτουργία «Back Home» (Back Home Functionality)
TC31	TS05	Λειτουργία «All Items» (All Items Functionality)
TC32	TS05	Λειτουργία αποσύνδεσης χρήστη (Logout Functionality)
TC33	TS05	Λειτουργία «Reset App State» (Reset App State Functionality)

TC34	TS06	Εκκαθάριση πεδίων σύνδεσης (Clear Login Fields)
TC35	TS06	Εκκαθάριση πεδίων checkout (Clear Checkout Fiel)

Πίνακας 11.2: Test Cases της Εφαρμογής και Αντιστοίχισή τους με τα Test Scenarios

11.4 Αναλυτικά Test Cases

Στην παρούσα ενότητα παρουσιάζονται αναλυτικά τα test cases που σχεδιάστηκαν για την αξιολόγηση της εφαρμογής SauceDemo. Κάθε test case περιλαμβάνει μοναδικό αναγνωριστικό αριθμό, περιγραφή της δοκιμής, βήματα εκτέλεσης και αναμενόμενο αποτέλεσμα.

Η οργανωμένη καταγραφή των test cases συμβάλλει στη συστηματική εκτέλεση των δοκιμών και διευκολύνει τη διαδικασία αξιολόγησης της λειτουργικότητας της εφαρμογής. Παράλληλα, η δομή αυτή επιτρέπει τη μελλοντική αξιοποίηση των test cases για την ανάπτυξη αυτοματοποιημένων σεναρίων testing.

11.4.1 Login Functionality Test Cases

Η κατηγορία Login Functionality περιλαμβάνει test cases που σχεδιάστηκαν για την αξιολόγηση της διαδικασίας σύνδεσης χρηστών στην εφαρμογή SauceDemo. Οι δοκιμές αυτές καλύπτουν τόσο θετικά όσο και αρνητικά σενάρια χρήσης, με στόχο την επαλήθευση της ορθής λειτουργίας του μηχανισμού ταυτοποίησης χρηστών και του ελέγχου εγκυρότητας δεδομένων εισόδου.

Τα συγκεκριμένα test cases αξιολογούν τη συμπεριφορά της εφαρμογής σε περιπτώσεις επιτυχούς σύνδεσης, λανθασμένων στοιχείων εισόδου και κενών πεδίων φόρμας login. Η διαδικασία αυτή είναι ιδιαίτερα σημαντική, καθώς η λειτουργία σύνδεσης αποτελεί κρίσιμο σημείο πρόσβασης στο σύστημα και βασικό στοιχείο λειτουργικότητας της εφαρμογής.

Οι Πίνακες 11.3 έως 11.8 παρουσιάζουν αναλυτικά τα test cases που χρησιμοποιήθηκαν για την αξιολόγηση της λειτουργικότητας σύνδεσης χρηστών της εφαρμογής.

TC01 – Επιτυχής σύνδεση χρήστη (Successful Login)

Πεδίο	Περιγραφή
Test Case ID	TC01
Scenario ID	TS01
Περιγραφή Δοκιμής	Έλεγχος επιτυχούς σύνδεσης χρήστη με έγκυρα στοιχεία εισόδου
Προϋποθέσεις	Ο χρήστης βρίσκεται στη σελίδα σύνδεσης της εφαρμογής
Βήματα Εκτέλεσης	1. Εισαγωγή έγκυρου username 2. Εισαγωγή έγκυρου password 3. Επιλογή του κουμπιού Login
Αναμενόμενο Αποτέλεσμα	Το σύστημα επιτρέπει επιτυχώς τη σύνδεση και μεταφέρει τον χρήστη στη σελίδα προϊόντων

Πίνακας 11.3: Test Case TC01

TC02 – Λανθασμένος κωδικός πρόσβασης (Invalid Password)

Πεδίο	Περιγραφή
Test Case ID	TC02
Scenario ID	TS01
Περιγραφή Δοκιμής	Σύνδεση χρήστη με λανθασμένο κωδικό πρόσβασης. Έλεγχος συμπεριφοράς του συστήματος κατά την εισαγωγή λανθασμένου κωδικού πρόσβασης
Προϋποθέσεις	Ο χρήστης βρίσκεται στη σελίδα σύνδεσης της εφαρμογής
Βήματα Εκτέλεσης	1. Εισαγωγή έγκυρου username 2. Εισαγωγή λανθασμένου password 3. Επιλογή του κουμπιού Login
Αναμενόμενο Αποτέλεσμα	Το σύστημα εμφανίζει μήνυμα σφάλματος « <i>Epic sadface: Username and password do not match any user in this service</i> » και δεν επιτρέπει τη σύνδεση του χρήστη

Πίνακας 11.4: Test Case TC02

TC03 – Λανθασμένο όνομα χρήστη (Invalid Username)

Πεδίο	Περιγραφή
Test Case ID	TC03
Scenario ID	TS01
Περιγραφή Δοκιμής	Έλεγχος συμπεριφοράς του συστήματος κατά την εισαγωγή λανθασμένου ονόματος χρήστη
Προϋποθέσεις	Ο χρήστης βρίσκεται στη σελίδα σύνδεσης της εφαρμογής
Βήματα Εκτέλεσης	1. Εισαγωγή λανθασμένου username 2. Εισαγωγή έγκυρου password 3. Επιλογή του κουμπιού Login
Αναμενόμενο Αποτέλεσμα	Το σύστημα εμφανίζει μήνυμα σφάλματος « <i>Epic sadface: Username and password do not match any user in this service</i> » και δεν επιτρέπει τη σύνδεση του χρήστη

Πίνακας 11.5: Test Case TC03

TC04 – Κενό όνομα χρήστη (Empty Username)

Πεδίο	Περιγραφή
Test Case ID	TC04
Scenario ID	TS01
Περιγραφή Δοκιμής	Έλεγχος συμπεριφοράς του συστήματος όταν δεν εισάγεται όνομα χρήστη
Προϋποθέσεις	Ο χρήστης βρίσκεται στη σελίδα σύνδεσης της εφαρμογής
Βήματα Εκτέλεσης	1. Αφήνεται κενό το πεδίο username 2. Εισαγωγή έγκυρου password 3. Επιλογή του κουμπιού Login
Αναμενόμενο Αποτέλεσμα	Το σύστημα εμφανίζει μήνυμα σφάλματος για το κενό πεδίο username « <i>Epic sadface: Username is required</i> ».

Πίνακας 11.6: Test Case TC04

TC05 – Κενός κωδικός πρόσβασης (Empty Password)

Πεδίο	Περιγραφή
Test Case ID	TC05
Scenario ID	TS01
Περιγραφή Δοκιμής	Έλεγχος συμπεριφοράς του συστήματος όταν δεν εισάγεται κωδικός πρόσβασης
Προϋποθέσεις	Ο χρήστης βρίσκεται στη σελίδα σύνδεσης της εφαρμογής
Βήματα Εκτέλεσης	1. Εισαγωγή έγκυρου username 2. Αφήνεται κενό το πεδίο password 3. Επιλογή του κουμπιού Login
Αναμενόμενο Αποτέλεσμα	Το σύστημα εμφανίζει μήνυμα σφάλματος για το κενό πεδίο password « <i>Epic sadface: Password is required</i> ».

Πίνακας 11.7: Test Case TC05

TC06 – Κενά πεδία σύνδεσης (Empty Fields)

Πεδίο	Περιγραφή
Test Case ID	TC06
Scenario ID	TS01
Περιγραφή Δοκιμής	Έλεγχος συμπεριφοράς του συστήματος όταν δεν εισάγονται στοιχεία σύνδεσης
Προϋποθέσεις	Ο χρήστης βρίσκεται στη σελίδα σύνδεσης της εφαρμογής
Βήματα Εκτέλεσης	1. Αφήνονται κενά τα πεδία username και password 2. Επιλογή του κουμπιού Login
Αναμενόμενο Αποτέλεσμα	Το σύστημα εμφανίζει μήνυμα σφάλματος « <i>Epic sadface: Username is required</i> » και δεν επιτρέπει τη σύνδεση του χρήστη

Πίνακας 11.8: Test Case TC06

Η συγκεκριμένη ομάδα test cases επέτρεψε την αξιολόγηση της αξιοπιστίας της διαδικασίας σύνδεσης χρηστών και του μηχανισμού επικύρωσης δεδομένων εισόδου της εφαρμογής. Μέσω των δοκιμών αυτών κατέστη δυνατός ο έλεγχος της συμπεριφοράς της εφαρμογής τόσο σε συνθήκες ορθής χρήσης όσο και σε περιπτώσεις μη έγκυρων ή ελλιπών δεδομένων εισόδου.

11.4.2 Product Inventory Test Cases

Η κατηγορία Product Inventory περιλαμβάνει test cases που σχεδιάστηκαν για την αξιολόγηση της λειτουργικότητας προβολής και διαχείρισης προϊόντων στην εφαρμογή SauceDemo. Οι δοκιμές αυτές επικεντρώνονται στην ορθή εμφάνιση των διαθέσιμων προϊόντων, στη λειτουργία ταξινόμησης και στη συνολική συμπεριφορά της σελίδας προϊόντων.

Οι συγκεκριμένες δοκιμές είναι ιδιαίτερα σημαντικές, καθώς η σελίδα προϊόντων αποτελεί το βασικό σημείο αλληλεπίδρασης του χρήστη με την εφαρμογή μετά τη διαδικασία σύνδεσης. Η αξιολόγηση της σωστής εμφάνισης και διαχείρισης των προϊόντων συμβάλλει στην επαλήθευση της λειτουργικότητας και της εμπειρίας χρήστη της εφαρμογής.

Οι Πίνακες 11.9 έως 11.15 παρουσιάζουν αναλυτικά τα test cases που χρησιμοποιήθηκαν για την αξιολόγηση της λειτουργικότητας της σελίδας προϊόντων.

TC07 – Εμφάνιση προϊόντων (Display Products)

Πεδίο	Περιγραφή
Test Case ID	TC07
Scenario ID	TS02
Περιγραφή Δοκιμής	Έλεγχος ορθής εμφάνισης της λίστας προϊόντων μετά την επιτυχή σύνδεση χρήστη
Προϋποθέσεις	Ο χρήστης έχει συνδεθεί επιτυχώς στο σύστημα
Βήματα Εκτέλεσης	1. Εκτέλεση επιτυχούς σύνδεσης με έγκυρα στοιχεία 2. Παρατήρηση της σελίδας προϊόντων
Αναμενόμενο Αποτέλεσμα	Το σύστημα εμφανίζει τη λίστα όλων των διαθέσιμων προϊόντων

Πίνακας 11.9: Test Case TC07

TC08 – Εμφάνιση εικόνων προϊόντων (Product Images)

Πεδίο	Περιγραφή
Test Case ID	TC08

Scenario ID	TS02
Περιγραφή Δοκιμής	Έλεγχος ορθής εμφάνισης των εικόνων προϊόντων στη σελίδα προϊόντων
Προϋποθέσεις	Ο χρήστης βρίσκεται στη σελίδα προϊόντων
Βήματα Εκτέλεσης	1. Μετάβαση στη σελίδα προϊόντων 2. Παρατήρηση των προϊόντων που εμφανίζονται στη λίστα
Αναμενόμενο Αποτέλεσμα	Το σύστημα εμφανίζει σωστά τις εικόνες όλων των προϊόντων

Πίνακας 11.10: Test Case TC08

TC09 – Προβολή περιγραφής προϊόντων (Product Description)

Πεδίο	Περιγραφή
Test Case ID	TC09
Scenario ID	TS02
Περιγραφή Δοκιμής	Έλεγχος ορθής εμφάνισης των περιγραφών των προϊόντων στη σελίδα προϊόντων
Προϋποθέσεις	Ο χρήστης βρίσκεται στη σελίδα προϊόντων
Βήματα Εκτέλεσης	1. Μετάβαση στη σελίδα προϊόντων 2. Παρατήρηση των περιγραφών των προϊόντων που εμφανίζονται στη λίστα
Αναμενόμενο Αποτέλεσμα	Το σύστημα εμφανίζει σωστά τις περιγραφές όλων των προϊόντων

Πίνακας 11.11: Test Case TC09

TC10 – Ταξινόμηση προϊόντων A-Z (Sort Products A-Z)

Πεδίο	Περιγραφή
Test Case ID	TC10
Scenario ID	TS02
Περιγραφή Δοκιμής	Έλεγχος λειτουργίας ταξινόμησης προϊόντων με αλφαβητική σειρά από A έως Z
Προϋποθέσεις	Ο χρήστης βρίσκεται στη σελίδα προϊόντων
Βήματα Εκτέλεσης	1. Επιλογή του φίλτρου ταξινόμησης προϊόντων 2. Επιλογή της λειτουργίας “Name (A to Z)”
Αναμενόμενο Αποτέλεσμα	Το σύστημα εμφανίζει τα προϊόντα ταξινομημένα αλφαβητικά από A έως Z

Πίνακας 11.12: Test Case TC10

TC11 – Ταξινόμηση προϊόντων Z-A (Sort Products Z-A)

Πεδίο	Περιγραφή
Test Case ID	TC11
Scenario ID	TS02
Περιγραφή Δοκιμής	Έλεγχος λειτουργίας ταξινόμησης προϊόντων με αλφαβητική σειρά από Z έως A
Προϋποθέσεις	Ο χρήστης βρίσκεται στη σελίδα προϊόντων

Βήματα Εκτέλεσης	1. Επιλογή του φίλτρου ταξινόμησης προϊόντων 2. Επιλογή της λειτουργίας “Name (Z to A)”
Αναμενόμενο Αποτέλεσμα	Το σύστημα εμφανίζει τα προϊόντα ταξινομημένα αλφαβητικά από Z έως A

Πίνακας 11.13: Test Case TC11

TC12 – Ταξινόμηση προϊόντων κατά αύξουσα τιμή (Low to High)

Πεδίο	Περιγραφή
Test Case ID	TC12
Scenario ID	TS02
Περιγραφή Δοκιμής	Έλεγχος λειτουργίας ταξινόμησης προϊόντων κατά αύξουσα σειρά τιμής
Προϋποθέσεις	Ο χρήστης βρίσκεται στη σελίδα προϊόντων
Βήματα Εκτέλεσης	1. Επιλογή του φίλτρου ταξινόμησης προϊόντων 2. Επιλογή της λειτουργίας “Price (low to high)”
Αναμενόμενο Αποτέλεσμα	Το σύστημα εμφανίζει τα προϊόντα ταξινομημένα από τη χαμηλότερη προς την υψηλότερη τιμή

Πίνακας 11.14: Test Case TC12

TC13 – Ταξινόμηση προϊόντων κατά φθίνουσα τιμή (High to Low)

Πεδίο	Περιγραφή
Test Case ID	TC13
Scenario ID	TS02
Περιγραφή Δοκιμής	Έλεγχος λειτουργίας ταξινόμησης προϊόντων κατά φθίνουσα σειρά τιμής
Προϋποθέσεις	Ο χρήστης βρίσκεται στη σελίδα προϊόντων
Βήματα Εκτέλεσης	1. Επιλογή του φίλτρου ταξινόμησης προϊόντων 2. Επιλογή της λειτουργίας “Price (high to low)”
Αναμενόμενο Αποτέλεσμα	Το σύστημα εμφανίζει τα προϊόντα ταξινομημένα από την υψηλότερη προς τη χαμηλότερη τιμή

Πίνακας 11.15: Test Case TC13

Όπως παρουσιάζεται στους Πίνακες 11.9 έως 11.15, τα test cases της κατηγορίας Product Inventory καλύπτουν βασικές λειτουργίες προβολής και διαχείρισης προϊόντων της εφαρμογής. Η διαδικασία αυτή συνέβαλε στην αξιολόγηση της ορθής λειτουργίας της σελίδας προϊόντων και της συνολικής εμπειρίας αλληλεπίδρασης του χρήστη με την εφαρμογή.

11.4.3 Shopping Cart Functionality Test Cases

Η κατηγορία Shopping Cart Functionality περιλαμβάνει test cases που σχεδιάστηκαν για την αξιολόγηση της λειτουργίας του καλαθιού αγορών στην εφαρμογή SauceDemo. Οι δοκιμές αυτές επικεντρώνονται στη διαδικασία προσθήκης και αφαίρεσης προϊόντων, στην ενημέρωση του καλαθιού αγορών και στη συνολική διαχείριση των επιλεγμένων προϊόντων από τον χρήστη.

Η λειτουργία του καλαθιού αγορών αποτελεί βασικό στοιχείο μιας εφαρμογής ηλεκτρονικού εμπορίου, καθώς συνδέεται άμεσα με τη διαδικασία ολοκλήρωσης αγοράς και τη συνολική εμπειρία χρήστη κατά την αλληλεπίδραση με το σύστημα.

Οι Πίνακες 11.16 έως 11.21 παρουσιάζουν αναλυτικά τα test cases που χρησιμοποιήθηκαν για την αξιολόγηση της λειτουργικότητας του καλαθιού αγορών.

TC14 – Προσθήκη προϊόντος στο καλάθι (Add Product to Cart)

Πεδίο	Περιγραφή
Test Case ID	TC14
Scenario ID	TS03
Περιγραφή Δοκιμής	Έλεγχος επιτυχούς προσθήκης προϊόντος στο καλάθι αγορών
Προϋποθέσεις	Ο χρήστης βρίσκεται στη σελίδα προϊόντων
Βήματα Εκτέλεσης	1. Επιλογή προϊόντος από τη λίστα προϊόντων 2. Επιλογή του κουμπιού “Add to cart”
Αναμενόμενο Αποτέλεσμα	Το προϊόν προστίθεται επιτυχώς στο καλάθι αγορών

Πίνακας 11.16: Test Case TC14

TC15 – Προσθήκη πολλαπλών προϊόντων στο καλάθι (Add Multiple Products)

Πεδίο	Περιγραφή
Test Case ID	TC15
Scenario ID	TS03
Περιγραφή Δοκιμής	Έλεγχος επιτυχούς προσθήκης πολλαπλών προϊόντων στο καλάθι αγορών
Προϋποθέσεις	Ο χρήστης βρίσκεται στη σελίδα προϊόντων
Βήματα Εκτέλεσης	1. Επιλογή περισσότερων από ενός προϊόντων 2. Επιλογή του κουμπιού “Add to cart” για κάθε προϊόν
Αναμενόμενο Αποτέλεσμα	Τα επιλεγμένα προϊόντα προστίθενται επιτυχώς στο καλάθι αγορών

Πίνακας 11.17: Test Case TC15

TC16 – Αφαίρεση προϊόντος από τη σελίδα προϊόντων (Remove Product from Products Page)

Πεδίο	Περιγραφή
Test Case ID	TC16
Scenario ID	TS03
Περιγραφή Δοκιμής	Έλεγχος επιτυχούς αφαίρεσης προϊόντος από το καλάθι μέσω της σελίδας προϊόντων
Προϋποθέσεις	Υπάρχει ήδη προϊόν προστιθέμενο στο καλάθι αγορών
Βήματα Εκτέλεσης	1. Μετάβαση στη σελίδα προϊόντων 2. Επιλογή του κουμπιού “Remove” σε προϊόν που έχει προστεθεί στο καλάθι
Αναμενόμενο Αποτέλεσμα	Το προϊόν αφαιρείται επιτυχώς από το καλάθι αγορών

Πίνακας 11.18: Test Case TC16

TC17 – Αφαίρεση προϊόντος από το καλάθι (Remove Product from Cart)

Πεδίο	Περιγραφή
Test Case ID	TC17
Scenario ID	TS03
Περιγραφή Δοκιμής	Έλεγχος επιτυχούς αφαίρεσης προϊόντος από το καλάθι αγορών
Προϋποθέσεις	Υπάρχει ήδη προϊόν προστιθέμενο στο καλάθι αγορών
Βήματα Εκτέλεσης	1. Μετάβαση στο καλάθι αγορών 2. Επιλογή του κουμπιού “Remove” στο επιλεγμένο προϊόν
Αναμενόμενο Αποτέλεσμα	Το προϊόν αφαιρείται επιτυχώς από το καλάθι αγορών

Πίνακας 11.19: Test Case TC17

TC18 – Ενημέρωση μετρητή καλαθιού (Cart Badge Update)

Πεδίο	Περιγραφή
Test Case ID	TC18
Scenario ID	TS03
Περιγραφή Δοκιμής	Έλεγχος ορθής ενημέρωσης του μετρητή προϊόντων στο καλάθι αγορών
Προϋποθέσεις	Ο χρήστης βρίσκεται στη σελίδα προϊόντων
Βήματα Εκτέλεσης	1. Προσθήκη προϊόντων στο καλάθι αγορών 2. Παρατήρηση του μετρητή καλαθιού
Αναμενόμενο Αποτέλεσμα	Ο μετρητής καλαθιού ενημερώνεται σωστά σύμφωνα με τον αριθμό των προϊόντων που έχουν προστεθεί

Πίνακας 11.20: Test Case TC18

TC19 – Προβολή καλαθιού αγορών (View Shopping Cart)

Πεδίο	Περιγραφή
Test Case ID	TC19
Scenario ID	TS03
Περιγραφή Δοκιμής	Έλεγχος ορθής προβολής του καλαθιού αγορών
Προϋποθέσεις	Υπάρχει τουλάχιστον ένα προϊόν στο καλάθι αγορών
Βήματα Εκτέλεσης	1. Επιλογή του εικονιδίου καλαθιού αγορών 2. Παρατήρηση της σελίδας καλαθιού
Αναμενόμενο Αποτέλεσμα	Το σύστημα εμφανίζει σωστά τα προϊόντα που έχουν προστεθεί στο καλάθι αγορών

Πίνακας 11.21: Test Case TC19

Όπως παρουσιάζεται στους Πίνακες 11.16 έως 11.21, τα test cases της κατηγορίας Shopping Cart Functionality καλύπτουν βασικές λειτουργίες διαχείρισης προϊόντων στο καλάθι αγορών. Η διαδικασία αυτή συνέβαλε στην αξιολόγηση της αξιοπιστίας και της ορθής λειτουργίας της εφαρμογής κατά τη διαχείριση προϊόντων από τον χρήστη.

11.4.4 Checkout Process Test Cases

Η κατηγορία Checkout Process περιλαμβάνει test cases που σχεδιάστηκαν για την αξιολόγηση της διαδικασίας ολοκλήρωσης αγοράς στην εφαρμογή SauceDemo. Οι δοκιμές αυτές επικεντρώνονται στη σωστή λειτουργία της φόρμας checkout, στον έλεγχο εγκυρότητας των δεδομένων εισόδου και στην επιτυχή ολοκλήρωση της διαδικασίας αγοράς.

Η διαδικασία checkout αποτελεί ένα από τα σημαντικότερα λειτουργικά τμήματα μιας εφαρμογής ηλεκτρονικού εμπορίου, καθώς σχετίζεται άμεσα με την ολοκλήρωση της αλληλεπίδρασης του χρήστη με το σύστημα και την επιτυχή εκτέλεση της διαδικασίας αγοράς.

Οι Πίνακες 11.22 έως 11.31 παρουσιάζουν αναλυτικά τα test cases που χρησιμοποιήθηκαν για την αξιολόγηση της λειτουργικότητας της διαδικασίας checkout.

TC20 – Πλοήγηση «Continue Shopping» (Continue Shopping Navigation)

Πεδίο	Περιγραφή
Test Case ID	TC20
Scenario ID	TS05
Περιγραφή Δοκιμής	Έλεγχος λειτουργίας του κουμπιού “Continue Shopping” από το καλάθι αγορών
Προϋποθέσεις	Ο χρήστης βρίσκεται στη σελίδα καλαθιού αγορών
Βήματα Εκτέλεσης	1. Επιλογή του κουμπιού “Continue Shopping” 2. Παρατήρηση της σελίδας που εμφανίζεται
Αναμενόμενο Αποτέλεσμα	Το σύστημα μεταφέρει τον χρήστη στη σελίδα προϊόντων

Πίνακας 11.22: Test Case TC20

TC21 – Έναρξη διαδικασίας checkout (Start Checkout Process)

Πεδίο	Περιγραφή
Test Case ID	TC21
Scenario ID	TS04
Περιγραφή Δοκιμής	Έλεγχος έναρξης της διαδικασίας checkout
Προϋποθέσεις	Υπάρχει τουλάχιστον ένα προϊόν στο καλάθι αγορών
Βήματα Εκτέλεσης	1. Μετάβαση στο καλάθι αγορών 2. Επιλογή του κουμπιού “Checkout”
Αναμενόμενο Αποτέλεσμα	Το σύστημα μεταφέρει τον χρήστη στη φόρμα checkout

Πίνακας 11.23: Test Case TC21

TC22 – Έγκυρα στοιχεία checkout (Valid Checkout Information)

Πεδίο	Περιγραφή
Test Case ID	TC22
Scenario ID	TS04
Περιγραφή Δοκιμής	Έλεγχος επιτυχούς εισαγωγής έγκυρων στοιχείων checkout
Προϋποθέσεις	Ο χρήστης βρίσκεται στη φόρμα checkout
Βήματα Εκτέλεσης	1. Εισαγωγή έγκυρου ονόματος 2. Εισαγωγή έγκυρου επωνύμου 3. Εισαγωγή έγκυρου ταχυδρομικού κώδικα 4. Επιλογή του κουμπιού “Continue”
Αναμενόμενο Αποτέλεσμα	Το σύστημα αποδέχεται τα στοιχεία και μεταφέρει τον χρήστη στη σελίδα επιβεβαίωσης παραγγελίας

Πίνακας 11.24: Test Case TC22

TC23 – Checkout με κενό καλάθι (Checkout with Empty Cart)

Πεδίο	Περιγραφή
Test Case ID	TC23
Scenario ID	TS04
Περιγραφή Δοκιμής	Έλεγχος συμπεριφοράς του συστήματος κατά την έναρξη checkout χωρίς προϊόντα στο καλάθι

Προϋποθέσεις	Το καλάθι αγορών είναι κενό
Βήματα Εκτέλεσης	1. Μετάβαση στο καλάθι αγορών 2. Επιλογή του κουμπιού “Checkout”
Αναμενόμενο Αποτέλεσμα	Το σύστημα διαχειρίζεται σωστά τη διαδικασία checkout με κενό καλάθι

Πίνακας 11.25: Test Case TC23

TC24 – Κενό όνομα χρήστη στο checkout (Empty First Name)

Πεδίο	Περιγραφή
Test Case ID	TC24
Scenario ID	TS04
Περιγραφή Δοκιμής	Έλεγχος συμπεριφοράς του συστήματος όταν δεν εισάγεται όνομα στη φόρμα checkout
Προϋποθέσεις	Ο χρήστης βρίσκεται στη φόρμα checkout
Βήματα Εκτέλεσης	1. Αφήνεται κενό το πεδίο First Name 2. Εισαγωγή επωνύμου και ταχυδρομικού κώδικα 3. Επιλογή του κουμπιού “Continue”
Αναμενόμενο Αποτέλεσμα	Το σύστημα εμφανίζει μήνυμα σφάλματος για το κενό πεδίο First Name «Error: First Name is required»

Πίνακας 11.26: Test Case TC24

TC25 – Κενό επώνυμο στο checkout (Empty Last Name)

Πεδίο	Περιγραφή
Test Case ID	TC25
Scenario ID	TS04
Περιγραφή Δοκιμής	Έλεγχος συμπεριφοράς του συστήματος όταν δεν εισάγεται επώνυμο στη φόρμα checkout
Προϋποθέσεις	Ο χρήστης βρίσκεται στη φόρμα checkout
Βήματα Εκτέλεσης	1. Εισαγωγή ονόματος 2. Αφήνεται κενό το πεδίο Last Name 3. Εισαγωγή ταχυδρομικού κώδικα 4. Επιλογή του κουμπιού “Continue”
Αναμενόμενο Αποτέλεσμα	Το σύστημα εμφανίζει μήνυμα σφάλματος για το κενό πεδίο Last Name «Error: Last Name is required»

Πίνακας 11.27: Test Case TC25

TC26 – Κενός ταχυδρομικός κώδικας (Empty Postal Code)

Πεδίο	Περιγραφή
Test Case ID	TC26
Scenario ID	TS04
Περιγραφή Δοκιμής	Έλεγχος συμπεριφοράς του συστήματος όταν δεν εισάγεται ταχυδρομικός κώδικας στη φόρμα checkout

Προϋποθέσεις	Ο χρήστης βρίσκεται στη φόρμα checkout
Βήματα Εκτέλεσης	1. Εισαγωγή ονόματος 2. Εισαγωγή επωνύμου 3. Αφήνεται κενό το πεδίο Postal Code 4. Επιλογή του κουμπιού “Continue”
Αναμενόμενο Αποτέλεσμα	Το σύστημα εμφανίζει μήνυμα σφάλματος για το κενό πεδίο Postal Code «Error: Postal Code is required»

Πίνακας 11.28: Test Case TC26

TC27 – Λειτουργία «Cancel» στο checkout (Cancel Checkout)

Πεδίο	Περιγραφή
Test Case ID	TC27
Scenario ID	TS04
Περιγραφή Δοκιμής	Έλεγχος λειτουργίας του κουμπιού “Cancel” κατά τη διαδικασία checkout
Προϋποθέσεις	Ο χρήστης βρίσκεται στη φόρμα checkout
Βήματα Εκτέλεσης	1. Επιλογή του κουμπιού “Cancel” 2. Παρατήρηση της σελίδας που εμφανίζεται
Αναμενόμενο Αποτέλεσμα	Το σύστημα επιστρέφει τον χρήστη στο καλάθι αγορών

Πίνακας 11.29: Test Case TC27

TC28 – Ολοκλήρωση παραγγελίας (Complete Checkout)

Πεδίο	Περιγραφή
Test Case ID	TC28
Scenario ID	TS04
Περιγραφή Δοκιμής	Έλεγχος επιτυχούς ολοκλήρωσης παραγγελίας
Προϋποθέσεις	Ο χρήστης έχει εισαγάγει έγκυρα στοιχεία στη φόρμα checkout και βρίσκεται στη σελίδα επισκόπησης παραγγελίας (checkout overview)
Βήματα Εκτέλεσης	1. Επιλογή του κουμπιού “Finish” 2. Παρατήρηση της σελίδας επιβεβαίωσης
Αναμενόμενο Αποτέλεσμα	Το σύστημα ολοκληρώνει επιτυχώς την παραγγελία και εμφανίζει μήνυμα επιβεβαίωσης «Thank you for your order!» καθώς και ενημέρωση αποστολής, ενώ παρέχεται η δυνατότητα επιστροφής μέσω του κουμπιού «Back Home»

Πίνακας 11.30: Test Case TC28

TC29 – Ακύρωση διαδικασίας checkout (Cancel Checkout Process)

Πεδίο	Περιγραφή
Test Case ID	TC29
Scenario ID	TS04
Περιγραφή Δοκιμής	Έλεγχος ακύρωσης της διαδικασίας checkout πριν την ολοκλήρωση της παραγγελίας
Προϋποθέσεις	Ο χρήστης βρίσκεται στη σελίδα Checkout: Overview

Βήματα Εκτέλεσης	1. Επιλογή του κουμπιού “Cancel” πριν την ολοκλήρωση της διαδικασίας 2. Παρατήρηση της σελίδας που εμφανίζεται
Αναμενόμενο Αποτέλεσμα	Το σύστημα ακυρώνει τη διαδικασία checkout και επιστρέφει τον χρήστη στη σελίδα προϊόντων

Πίνακας 11.31: Test Case TC29

Όπως παρουσιάζεται στους Πίνακες 11.22 έως 11.31, τα test cases της κατηγορίας Checkout Process καλύπτουν βασικές λειτουργίες της διαδικασίας ολοκλήρωσης αγοράς και της επικύρωσης δεδομένων εισόδου. Η διαδικασία αυτή συνέβαλε στην αξιολόγηση της αξιοπιστίας και της λειτουργικότητας του μηχανισμού checkout της εφαρμογής.

11.4.5 Navigation and Session Test Cases

Η κατηγορία Navigation and Session περιλαμβάνει test cases που σχεδιάστηκαν για την αξιολόγηση της πλοήγησης του χρήστη μεταξύ διαφορετικών σελίδων της εφαρμογής, καθώς και της διαχείρισης της κατάστασης λειτουργίας (application state) κατά τη διάρκεια της αλληλεπίδρασης με το σύστημα.

Οι δοκιμές αυτές επικεντρώνονται στη λειτουργία στοιχείων πλοήγησης της εφαρμογής, όπως η επιστροφή στην αρχική σελίδα προϊόντων, η προβολή όλων των προϊόντων, η διαδικασία αποσύνδεσης χρήστη και η επαναφορά της κατάστασης της εφαρμογής.

Οι Πίνακες 11.32 έως 11.35 παρουσιάζουν αναλυτικά τα test cases που χρησιμοποιήθηκαν για την αξιολόγηση της λειτουργικότητας πλοήγησης και διαχείρισης συνεδρίας της εφαρμογής.

TC30 – Λειτουργία «Back Home» (Back Home Functionality)

Πεδίο	Περιγραφή
Test Case ID	TC30
Scenario ID	TS05
Περιγραφή Δοκιμής	Έλεγχος λειτουργίας του κουμπιού “Back Home” μετά την ολοκλήρωση παραγγελίας
Προϋποθέσεις	Ο χρήστης έχει ολοκληρώσει την παραγγελία και βρίσκεται στη σελίδα επιβεβαίωσης
Βήματα Εκτέλεσης	1. Επιλογή του κουμπιού «Back Home» 2. Παρατήρηση της σελίδας που εμφανίζεται 3. Έλεγχος του καλαθιού αγορών
Αναμενόμενο Αποτέλεσμα	Το σύστημα μεταφέρει τον χρήστη στη σελίδα προϊόντων και το καλάθι αγορών είναι κενό

Πίνακας 11.32: Test Case TC30

TC31 – Λειτουργία «All Items» (All Items Functionality)

Πεδίο	Περιγραφή
Test Case ID	TC31
Scenario ID	TS05
Περιγραφή Δοκιμής	Έλεγχος λειτουργίας της επιλογής “All Items” στο μενού πλοήγησης
Προϋποθέσεις	Ο χρήστης είναι συνδεδεμένος στην εφαρμογή
Βήματα Εκτέλεσης	1. Επιλογή του μενού πλοήγησης 2. Επιλογή «All Items» 3. Παρατήρηση της σελίδας που εμφανίζεται
Αναμενόμενο Αποτέλεσμα	Το σύστημα μεταφέρει τον χρήστη στη σελίδα προϊόντων

Πίνακας 11.33: Test Case TC31

TC32 – Λειτουργία αποσύνδεσης χρήστη (Logout Functionality)

Πεδίο	Περιγραφή
Test Case ID	TC32
Scenario ID	TS05
Περιγραφή Δοκιμής	Έλεγχος διαδικασίας αποσύνδεσης χρήστη από την εφαρμογή
Προϋποθέσεις	Ο χρήστης είναι συνδεδεμένος στο σύστημα
Βήματα Εκτέλεσης	1. Επιλογή του μενού πλοήγησης 2. Επιλογή της ενέργειας «Logout» 3. Παρατήρηση της σελίδας που εμφανίζεται
Αναμενόμενο Αποτέλεσμα	Το σύστημα αποσυνδέει τον χρήστη και τον επιστρέφει στη σελίδα σύνδεσης

Πίνακας 11.34: Test Case TC32

TC33 – Λειτουργία «Reset App State» (Reset App State Functionality)

Πεδίο	Περιγραφή
Test Case ID	TC33
Scenario ID	TS05
Περιγραφή Δοκιμής	Έλεγχος λειτουργίας επαναφοράς της κατάστασης της εφαρμογής
Προϋποθέσεις	Υπάρχουν προϊόντα στο καλάθι αγορών
Βήματα Εκτέλεσης	1. Άνοιγμα του μενού πλοήγησης 2. Επιλογή της λειτουργίας “Reset App State” 3. Παρατήρηση του καλαθιού
Αναμενόμενο Αποτέλεσμα	Το σύστημα επαναφέρει την κατάσταση της εφαρμογής και αφαιρεί τα προϊόντα από το καλάθι αγορών - Το καλάθι αδειάζει και ο μετρητής προϊόντων μηδενίζεται

Πίνακας 11.35: Test Case TC33

Όπως παρουσιάζεται στους Πίνακες 11.32 έως 11.35, τα test cases της κατηγορίας Navigation καλύπτουν βασικές λειτουργίες πλοήγησης και διαχείρισης συνεδρίας χρήστη στην εφαρμογή. Η διαδικασία αυτή συνέβαλε στην αξιολόγηση της συνολικής συμπεριφοράς της εφαρμογής κατά την αλληλεπίδραση του χρήστη με τις λειτουργίες πλοήγησης και διαχείρισης κατάστασης του συστήματος.

11.4.6 Input Field Validation Test Cases

Η κατηγορία Input Field Validation περιλαμβάνει test cases που σχεδιάστηκαν για την αξιολόγηση της συμπεριφοράς της εφαρμογής κατά τη διαχείριση και εκκαθάριση πεδίων εισαγωγής δεδομένων. Οι δοκιμές αυτές επικεντρώνονται στη λειτουργικότητα των φορμών εισαγωγής στοιχείων και στη σωστή διαχείριση της αλληλεπίδρασης του χρήστη με τα πεδία δεδομένων της εφαρμογής.

Η αξιολόγηση της συμπεριφοράς των πεδίων εισαγωγής είναι ιδιαίτερα σημαντική σε web εφαρμογές, καθώς σχετίζεται άμεσα με τη χρηστικότητα, την αξιοπιστία και τη σωστή λειτουργία των μηχανισμών εισαγωγής δεδομένων.

Οι Πίνακες 11.36 έως 11.37 παρουσιάζουν αναλυτικά τα test cases που χρησιμοποιήθηκαν για την αξιολόγηση της λειτουργικότητας των πεδίων εισαγωγής δεδομένων της εφαρμογής.

TC34 – Εκκαθάριση πεδίων σύνδεσης (Clear Login Fields)

Πεδίο	Περιγραφή
Test Case ID	TC34
Scenario ID	TS06
Περιγραφή Δοκιμής	Έλεγχος λειτουργίας εκκαθάρισης των πεδίων σύνδεσης χρήστη
Προϋποθέσεις	Ο χρήστης βρίσκεται στη σελίδα σύνδεσης της εφαρμογής
Βήματα Εκτέλεσης	1. Εισαγωγή μη έγκυρων ή/και κενών στοιχείων στα πεδία username και password 2. Επιλογή του κουμπιού «Login» 3. Εμφάνιση μηνύματος σφάλματος 4. Εμφάνιση του εικονιδίου «X» στα πεδία εισαγωγής 5. Επιλογή του εικονιδίου «X»
Αναμενόμενο Αποτέλεσμα	Τα πεδία username και password εκκαθαρίζονται επιτυχώς

Πίνακας 11.36: Test Case TC34

TC35 – Εκκαθάριση πεδίων checkout (Clear Checkout Fields)

Πεδίο	Περιγραφή
Test Case ID	TC35
Scenario ID	TS06
Περιγραφή Δοκιμής	Έλεγχος λειτουργίας εκκαθάρισης των πεδίων της φόρμας checkout
Προϋποθέσεις	Ο χρήστης βρίσκεται στη φόρμα checkout
Βήματα Εκτέλεσης	1. Εισαγωγή μη έγκυρων ή/και κενών στοιχείων στα πεδία First Name, Last Name ή Zip Code 2. Επιλογή του κουμπιού «Continue» 3. Εμφάνιση μηνύματος σφάλματος 4. Εμφάνιση του εικονιδίου «X» στα πεδία εισαγωγής 5. Επιλογή του εικονιδίου «X»
Αναμενόμενο Αποτέλεσμα	Τα πεδία της φόρμας checkout εκκαθαρίζονται επιτυχώς

Πίνακας 11.37: Test Case TC35

Όπως παρουσιάζεται στους Πίνακες 11.36 έως 11.37, τα test cases της κατηγορίας Input Field Validation καλύπτουν βασικές λειτουργίες διαχείρισης πεδίων εισαγωγής δεδομένων της εφαρμογής. Η διαδικασία αυτή συνέβαλε στην αξιολόγηση της συμπεριφοράς των φορμών εισαγωγής και της συνολικής λειτουργικότητας της διεπαφής χρήστη της εφαρμογής.

11.5 Συμπεράσματα

Στο παρόν κεφάλαιο παρουσιάστηκε ο σχεδιασμός των test scenarios και των test cases που χρησιμοποιήθηκαν για την αξιολόγηση της εφαρμογής SauceDemo. Η διαδικασία σχεδιασμού οργανώθηκε με βάση τις βασικές λειτουργικές ενότητες της εφαρμογής, επιτρέποντας τη συστηματική κάλυψη κρίσιμων λειτουργιών, όπως η σύνδεση χρηστών, η διαχείριση προϊόντων, το καλάθι αγορών, η διαδικασία checkout και οι λειτουργίες πλοήγησης της εφαρμογής.

Συνολικά σχεδιάστηκαν τριάντα πέντε (35) test cases, τα οποία καλύπτουν τόσο θετικά όσο και αρνητικά σενάρια χρήσης. Η οργανωμένη καταγραφή των δοκιμών συνέβαλε στη δημιουργία ενός δομημένου πλαισίου testing, το οποίο υποστηρίζει την αποτελεσματική αξιολόγηση της λειτουργικότητας και της συμπεριφοράς της εφαρμογής.

Τα test cases που παρουσιάστηκαν στο παρόν κεφάλαιο χρησιμοποιούνται στο επόμενο κεφάλαιο για την εκτέλεση των δοκιμών και την καταγραφή των αποτελεσμάτων της διαδικασίας manual testing.

12 Εκτέλεση Manual Testing και Αποτελέσματα

12.1 Εισαγωγή

Στο παρόν κεφάλαιο παρουσιάζεται η διαδικασία εκτέλεσης των δοκιμών (test execution) που σχεδιάστηκαν στο προηγούμενο κεφάλαιο, καθώς και τα αποτελέσματα που προέκυψαν από την εφαρμογή της μεθοδολογίας manual testing στη διαδικτυακή εφαρμογή SauceDemo.

Οι δοκιμές πραγματοποιήθηκαν με στόχο την αξιολόγηση της λειτουργικότητας, της συμπεριφοράς και της αξιοπιστίας της εφαρμογής σε διαφορετικά σενάρια χρήσης. Η διαδικασία testing εκτελέστηκε χειροκίνητα από τον tester, ακολουθώντας τα βήματα που είχαν καθοριστεί στα αντίστοιχα test cases.

Κατά τη διάρκεια της διαδικασίας δοκιμών εκτελέστηκαν όλα τα test cases που παρουσιάστηκαν στο προηγούμενο κεφάλαιο, καλύπτοντας λειτουργίες όπως η σύνδεση χρηστών, η διαχείριση προϊόντων, το καλάθι αγορών, η διαδικασία checkout και οι λειτουργίες πλοήγησης της εφαρμογής.

Για κάθε test case καταγράφηκαν:

- το αναμενόμενο αποτέλεσμα,
- το πραγματικό αποτέλεσμα της εκτέλεσης,
- η κατάσταση επιτυχίας ή αποτυχίας της δοκιμής (Pass / Fail),
- καθώς και πιθανές παρατηρήσεις σχετικά με τη συμπεριφορά της εφαρμογής.

Η καταγραφή και η ανάλυση των αποτελεσμάτων συνέβαλαν στην αξιολόγηση της ποιότητας της εφαρμογής και στον εντοπισμό λειτουργικών ή χρηστικών προβλημάτων που εμφανίστηκαν κατά τη διαδικασία testing.

12.2 Αποτελέσματα Εκτέλεσης Δοκιμών

Στην παρούσα ενότητα παρουσιάζονται τα αποτελέσματα που προέκυψαν από την εκτέλεση των test cases που σχεδιάστηκαν στο προηγούμενο κεφάλαιο. Η διαδικασία εκτέλεσης πραγματοποιήθηκε χειροκίνητα (manual testing) και αφορούσε τον έλεγχο των βασικών λειτουργιών της εφαρμογής SauceDemo.

Για κάθε test case καταγράφηκε η κατάσταση επιτυχίας ή αποτυχίας της δοκιμής, καθώς και σύντομες παρατηρήσεις σχετικά με τη συμπεριφορά της εφαρμογής κατά τη διάρκεια της εκτέλεσης. Η συστηματική καταγραφή των αποτελεσμάτων συνέβαλε στην αξιολόγηση της αξιοπιστίας της εφαρμογής και στον εντοπισμό πιθανών σφαλμάτων ή δυσλειτουργιών.

Ο Πίνακας 12.1 παρουσιάζει συνοπτικά τα αποτελέσματα εκτέλεσης των test cases.

Test Case ID	Scenario ID	Status	Παρατηρήσεις
TC01	TS01	PASS	Επιτυχής σύνδεση χρήστη με έγκυρα στοιχεία
TC02	TS01	PASS	Εμφάνιση μηνύματος σφάλματος για λανθασμένο password
TC03	TS01	PASS	Εμφάνιση μηνύματος σφάλματος για λανθασμένο username
TC04	TS01	PASS	Εμφάνιση μηνύματος σφάλματος για κενό username
TC05	TS01	PASS	Εμφάνιση μηνύματος σφάλματος για κενό password
TC06	TS01	PASS	Αποτροπή σύνδεσης χωρίς εισαγωγή στοιχείων
TC07	TS02	PASS	Ορθή εμφάνιση λίστας προϊόντων
TC08	TS02	PASS	Ορθή εμφάνιση εικόνων προϊόντων
TC09	TS02	PASS	Ορθή εμφάνιση περιγραφών προϊόντων
TC10	TS02	PASS	Ορθή ταξινόμηση προϊόντων (Α-Ζ)
TC11	TS02	PASS	Ορθή ταξινόμηση προϊόντων (Ζ-Α)
TC12	TS02	PASS	Ορθή ταξινόμηση προϊόντων κατά αύξουσα τιμή
TC13	TS02	PASS	Ορθή ταξινόμηση προϊόντων κατά φθίνουσα τιμή
TC14	TS03	PASS	Επιτυχής προσθήκη προϊόντος στο καλάθι
TC15	TS03	PASS	Επιτυχής προσθήκη πολλαπλών προϊόντων
TC16	TS03	PASS	Επιτυχής αφαίρεση προϊόντος από τη σελίδα προϊόντων
TC17	TS03	PASS	Επιτυχής αφαίρεση προϊόντος από το καλάθι
TC18	TS03	PASS	Ορθή ενημέρωση μετρητή καλαθιού αγορών
TC19	TS03	PASS	Ορθή προβολή καλαθιού αγορών
TC20	TS05	PASS	Επιτυχής λειτουργία «Continue Shopping»
TC21	TS04	PASS	Επιτυχής μετάβαση στη διαδικασία checkout
TC22	TS04	PASS	Επιτυχής αποδοχή έγκυρων στοιχείων checkout
TC23	TS04	FAIL	Επιτρέπεται ολοκλήρωση checkout με κενό καλάθι αγορών
TC24	TS04	PASS	Εμφάνιση μηνύματος σφάλματος για κενό First Name
TC25	TS04	PASS	Εμφάνιση μηνύματος σφάλματος για κενό Last Name

TC26	TS04	PASS	Εμφάνιση μηνύματος σφάλματος για κενό Postal Code
TC27	TS04	PASS	Επιτυχής λειτουργία «Cancel» στη φόρμα checkout
TC28	TS04	PASS	Επιτυχής ολοκλήρωση παραγγελίας
TC29	TS04	PASS	Επιτυχής ακύρωση διαδικασίας checkout
TC30	TS05	PASS	Επιτυχής λειτουργία «Back Home»
TC31	TS05	PASS	Επιτυχής λειτουργία «All Items»
TC32	TS05	PASS	Επιτυχής αποσύνδεση χρήστη
TC33	TS05	PASS	Επιτυχής λειτουργία «Reset App State»
TC34	TS06	FAIL	Τα πεδία σύνδεσης δεν εκκαθαρίζονται μέσω του εικονιδίου (X)
TC35	TS06	FAIL	Τα πεδία checkout δεν εκκαθαρίζονται μέσω του εικονιδίου (X)

Πίνακας 12.1: Αποτελέσματα Εκτέλεσης Test Cases

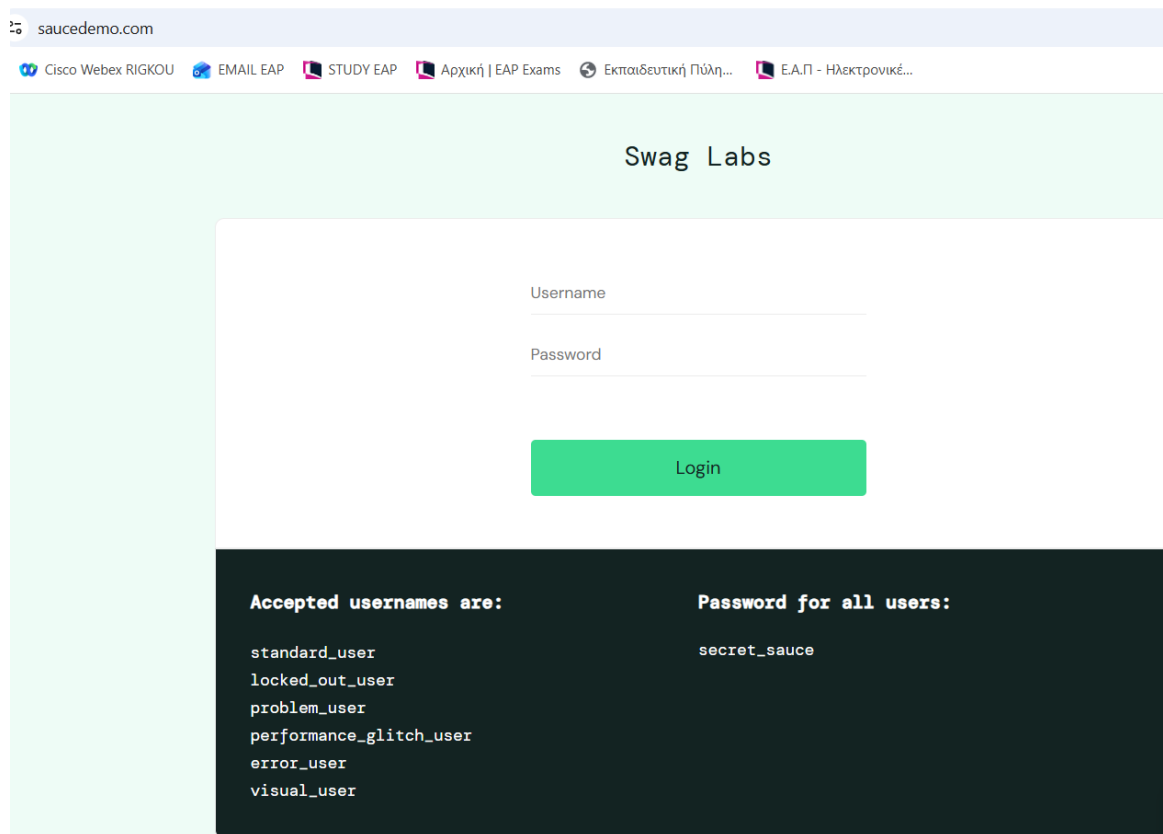
Όπως παρουσιάζεται στον Πίνακα 12.1, η πλειονότητα των test cases ολοκληρώθηκε επιτυχώς, γεγονός που υποδηλώνει ικανοποιητικό επίπεδο λειτουργικότητας της εφαρμογής. Ωστόσο, κατά τη διαδικασία testing εντοπίστηκαν ορισμένες δυσλειτουργίες που σχετίζονται με τη διαδικασία checkout και τη λειτουργία εκκαθάρισης πεδίων εισαγωγής δεδομένων.

12.3 Στιγμιότυπα Εκτέλεσης Δοκιμών

Για τις ανάγκες τεκμηρίωσης των δοκιμών λογισμικού (software testing) χρησιμοποιήθηκε η δημόσια διαθέσιμη εκπαιδευτική πλατφόρμα Sauce Demo της εταιρείας Sauce Labs (διαθέσιμη στο saucedemo.com). Στο παρόν κεφάλαιο παρουσιάζεται ένας περιορισμένος και ενδεικτικός αριθμός στιγμιότυπων (screenshots) από τα βασικότερα σενάρια δοκιμών που εκτελέστηκαν, για λόγους οικονομίας του κειμένου και ακαδημαϊκής παράθεσης σύμφωνα με τον Ν. 2121/1993. Η χρήση στιγμιότυπων οθόνης συμβάλλει στην οπτική επιβεβαίωση των αποτελεσμάτων των δοκιμών και διευκολύνει την κατανόηση της συμπεριφοράς της εφαρμογής κατά τη διάρκεια της διαδικασίας testing.

Τα screenshots που παρουσιάζονται στην παρούσα ενότητα αποτυπώνουν βασικά στάδια της αλληλεπίδρασης του χρήστη με την εφαρμογή, όπως η διαδικασία σύνδεσης, η προβολή προϊόντων, η διαχείριση του καλαθιού αγορών, η διαδικασία checkout και ο εντοπισμός σφαλμάτων που καταγράφηκαν κατά την εκτέλεση των δοκιμών.

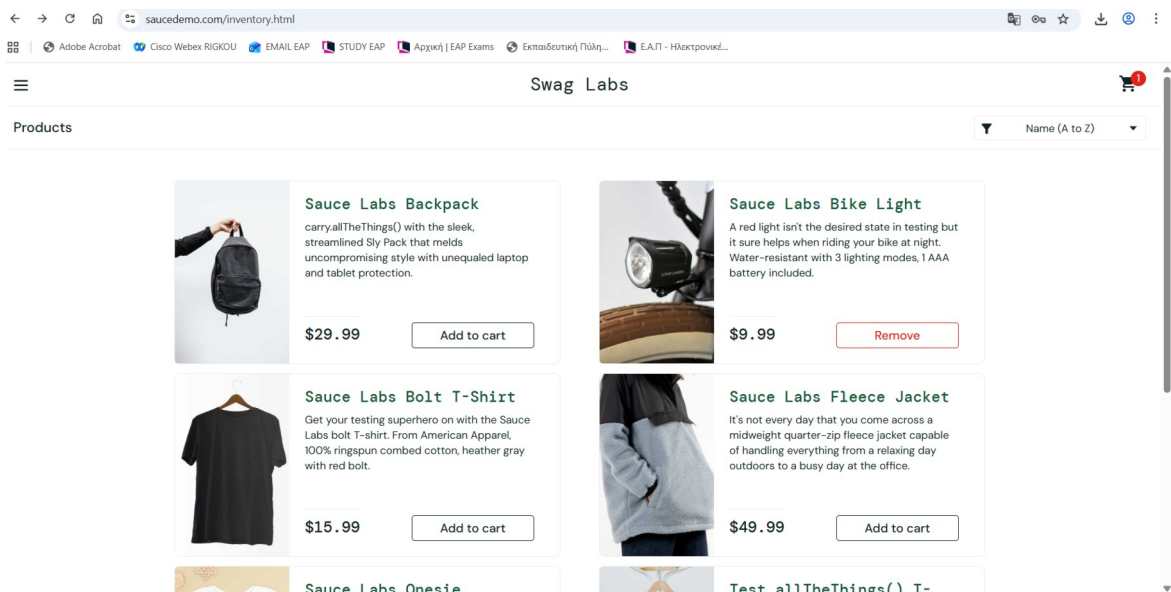
Στην Εικόνα 12.1 παρουσιάζεται η σελίδα σύνδεσης της εφαρμογής, όπου ο χρήστης εισάγει τα στοιχεία πρόσβασης στο σύστημα.



Εικόνα 12.1.Σελίδα σύνδεσης χρήστη

Πηγή: Διαδικτυακή πλατφόρμα δοκιμών Sauce Demo (saucedemo.com)

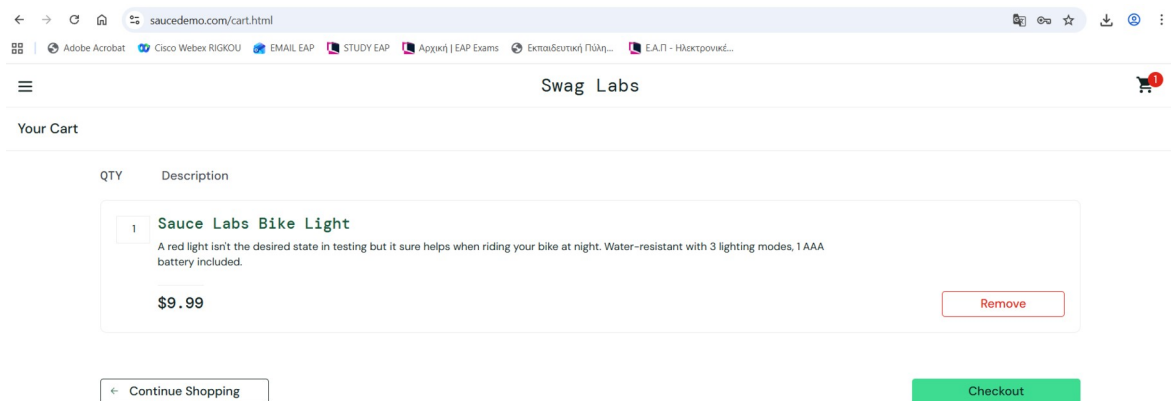
Στην Εικόνα 12.2 παρουσιάζεται η διαδικασία προσθήκης προϊόντος στο καλάθι αγορών μέσω της επιλογής του κουμπιού «Add to Cart».



Εικόνα 12.2. Προσθήκη προϊόντος στο καλάθι αγορών

Πηγή: Διαδικτυακή πλατφόρμα δοκιμών Sauce Demo (saucedemo.com)

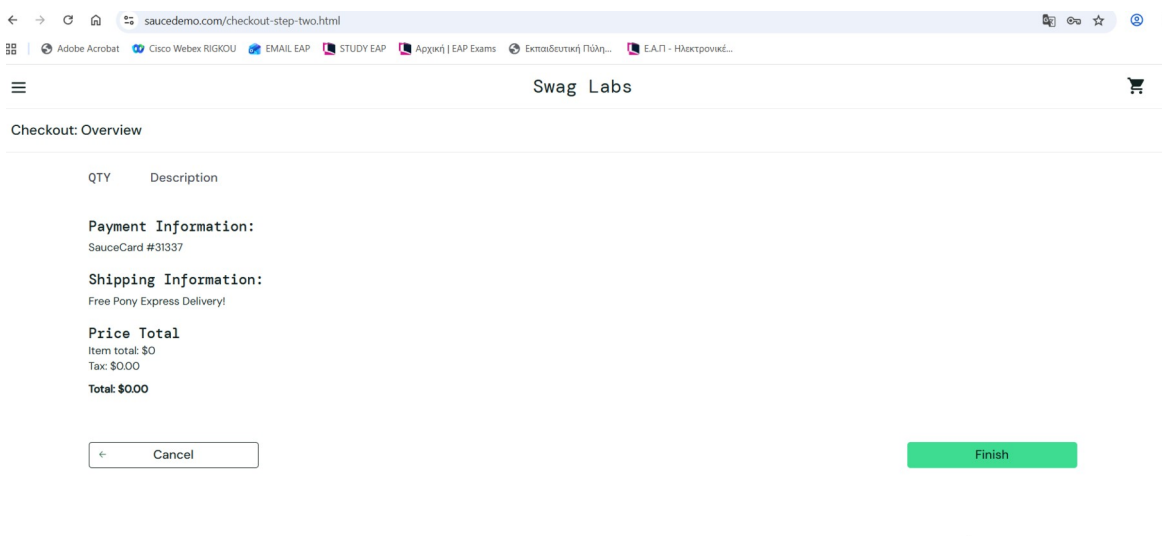
Στην Εικόνα 12.3 παρουσιάζεται η σελίδα καλαθιού αγορών, όπου εμφανίζονται τα προϊόντα που έχουν επιλεγεί από τον χρήστη.



Εικόνα 12.3. Προβολή καλαθιού αγορών

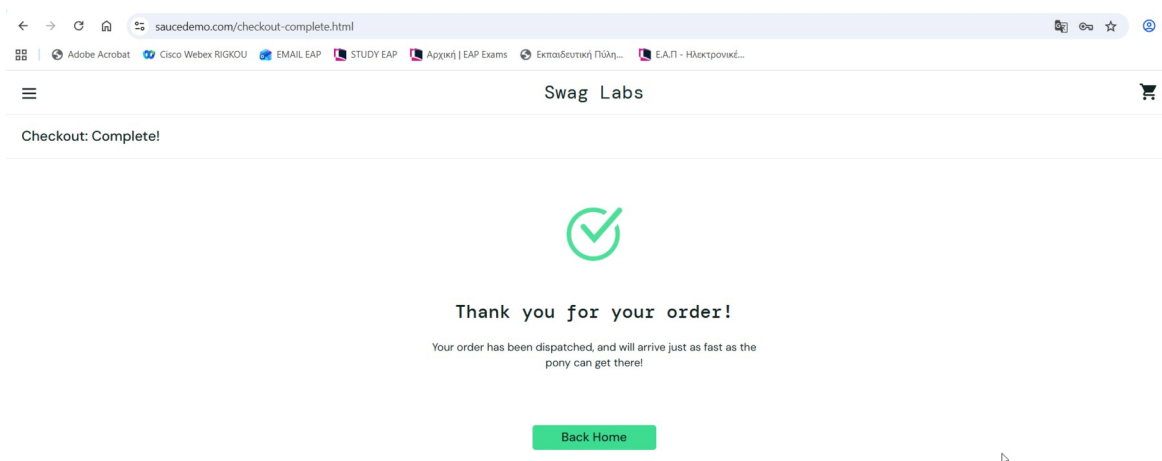
Πηγή: Διαδικτυακή πλατφόρμα δοκιμών Sauce Demo (saucedemo.com)

Στις Εικόνες 12.4 και 12.5 παρουσιάζεται η διαδικασία checkout με κενό καλάθι αγορών, καθώς και η επιτυχής ολοκλήρωση της διαδικασίας παρά την απουσία προϊόντων στο καλάθι.



Εικόνα 12.4. Διαδικασία checkout με κενό καλάθι αγορών

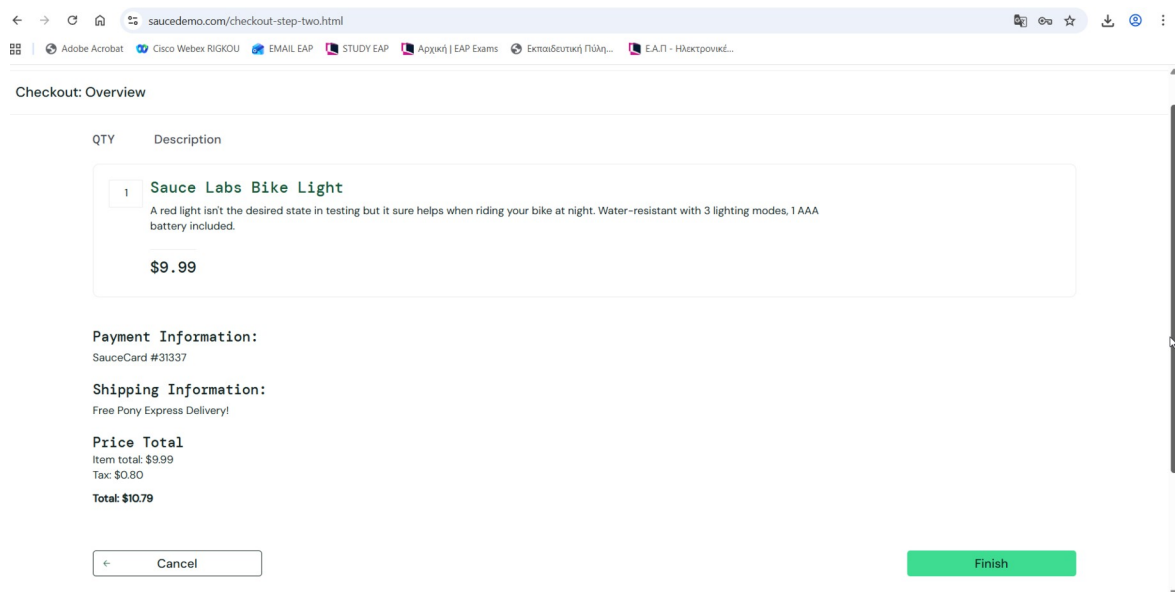
Πηγή: Διαδικτυακή πλατφόρμα δοκιμών Sauce Demo (saucedemo.com)



Εικόνα 12.5. Επιτυχής ολοκλήρωση checkout με κενό καλάθι

Πηγή: Διαδικτυακή πλατφόρμα δοκιμών Sauce Demo (saucedemo.com)

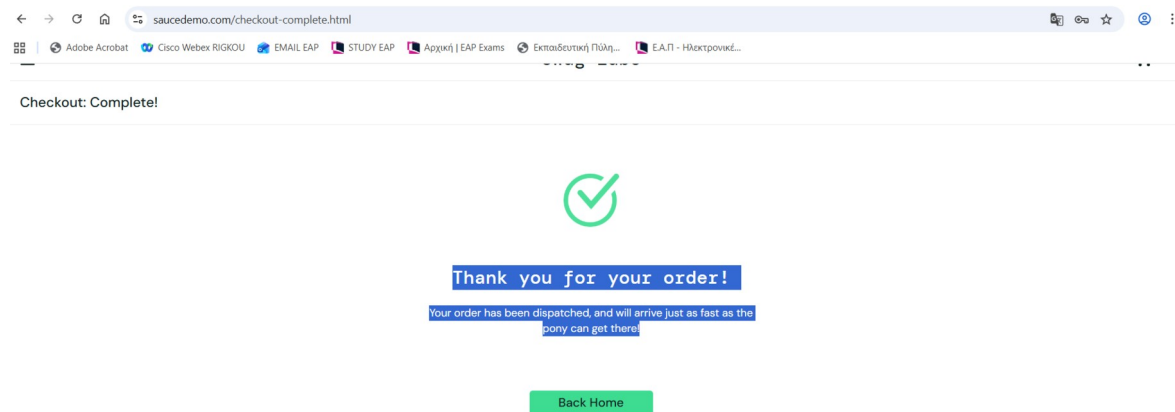
Στην Εικόνα 12.6 παρουσιάζεται η σελίδα επισκόπησης παραγγελίας (checkout overview), όπου εμφανίζονται τα στοιχεία της παραγγελίας πριν την ολοκλήρωση της αγοράς. Από τη συγκεκριμένη σελίδα ο χρήστης μπορεί να ολοκληρώσει την αγορά επιλέγοντας «Finish» ή να την ακυρώσει επιλέγοντας «Cancel».



Εικόνα 12.6.Σελίδα επισκόπησης παραγγελίας

Πηγή: Διαδικτυακή πλατφόρμα δοκιμών Sauce Demo (sauceDemo.com)

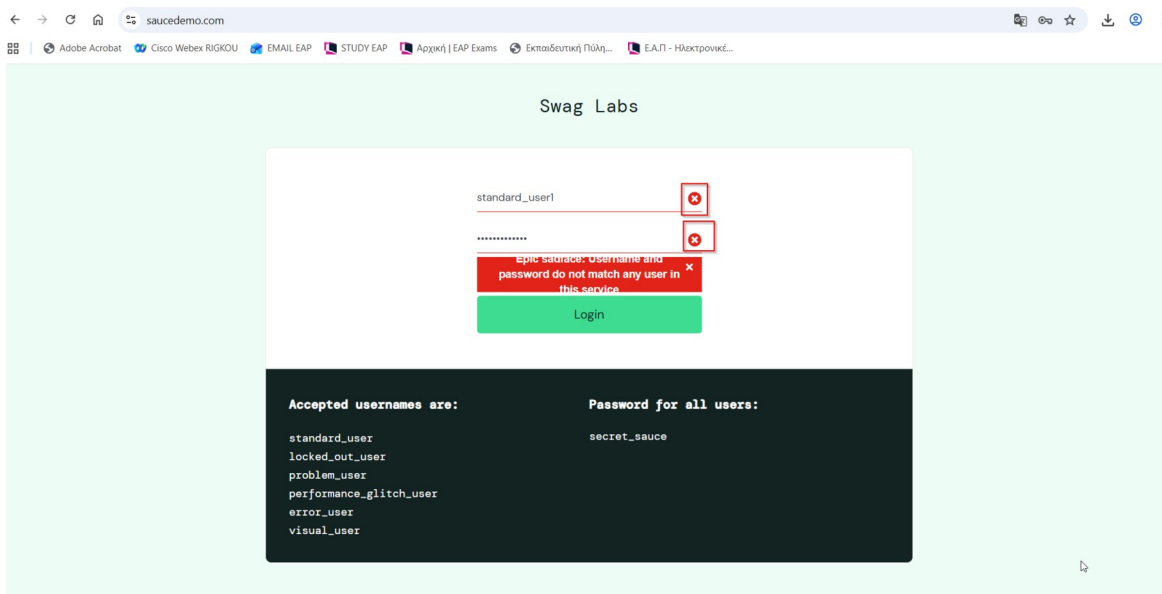
Στην Εικόνα 12.7 παρουσιάζεται η επιτυχής ολοκλήρωση της διαδικασίας αγοράς και η εμφάνιση σχετικού μηνύματος επιβεβαίωσης.



Εικόνα 12.7.Επιτυχής ολοκλήρωση παραγγελίας

Πηγή: Διαδικτυακή πλατφόρμα δοκιμών Sauce Demo (sauceDemo.com)

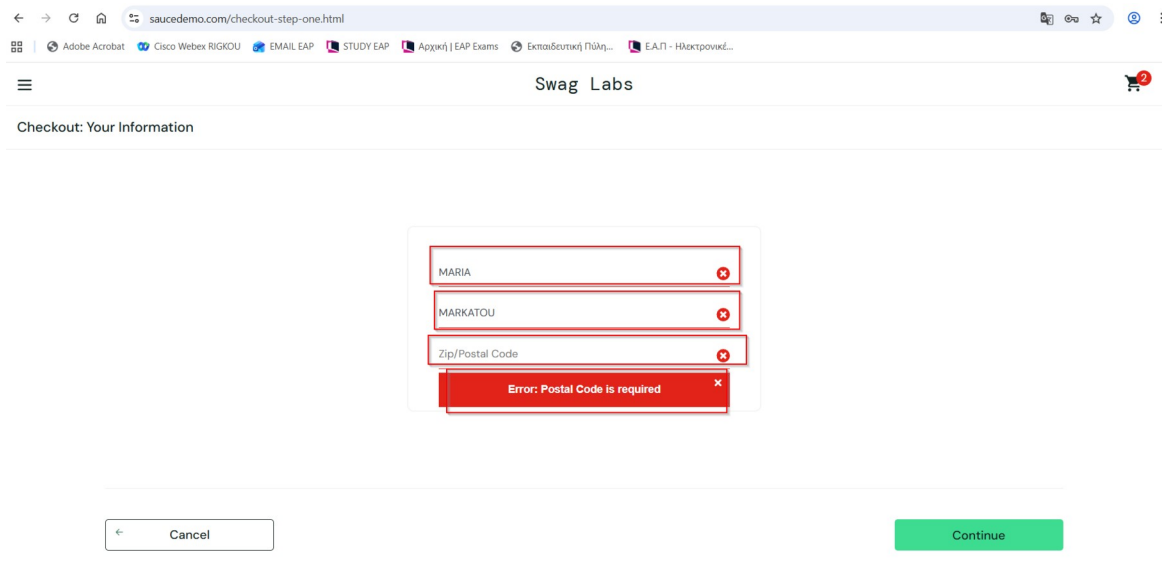
Στην Εικόνα 12.8 παρουσιάζεται το σφάλμα που εντοπίστηκε στη φόρμα σύνδεσης της εφαρμογής, όπου το εικονίδιο διαγραφής (X) δεν εκκαθαρίζει τα δεδομένα των πεδίων εισαγωγής.



Εικόνα 12.8.Εντοπισμός σφάλματος στη φόρμα σύνδεσης

Πηγή: Διαδικτυακή πλατφόρμα δοκιμών Sauce Demo (saucedemo.com)

Στην Εικόνα 12.9 παρουσιάζεται αντίστοιχο σφάλμα στη φόρμα checkout, όπου το εικονίδιο διαγραφής (X) δεν διαγράφει τα δεδομένα των πεδίων εισαγωγής.



Εικόνα 12.9.Εντοπισμός σφάλματος στη φόρμα checkout

Πηγή: Διαδικτυακή πλατφόρμα δοκιμών Sauce Demo (saucedemo.com)

12.4 Αναφορά Σφαλμάτων (Defect Reports)

Κατά τη διαδικασία εκτέλεσης των δοκιμών στην εφαρμογή SauceDemo εντοπίστηκαν defects που σχετίζονται τόσο με τη διεπαφή χρήστη όσο και με τη λειτουργική συμπεριφορά του συστήματος. Τα defects που καταγράφηκαν παρουσιάζονται στις ακόλουθες υποενότητες.

12.4.1 Defect 01 – Δυσλειτουργία εικονιδίου διαγραφής (Clear Icon)

Defect ID: DEF01

Τίτλος Defect: Το εικονίδιο διαγραφής (X) δεν αφαιρεί τα δεδομένα εισαγωγής

Severity: Medium

Priority: Medium

Status: Open

Περιβάλλον Δοκιμών: Windows 11, Google Chrome

Επηρεαζόμενες Λειτουργίες: Φόρμα σύνδεσης (Login Form), Φόρμα στοιχείων checkout (Checkout Information Form)

Περιγραφή

Κατά τη χρήση των πεδίων εισαγωγής δεδομένων εμφανίζεται εικονίδιο διαγραφής (X) μετά την εμφάνιση μηνύματος σφάλματος. Το εικονίδιο αυτό, σύμφωνα με συνήθειες πρακτικές σχεδιασμού διεπαφής χρήστη, αναμένεται να επιτρέπει τη διαγραφή των δεδομένων που έχουν εισαχθεί στα πεδία της φόρμας. Ωστόσο, κατά την επιλογή του δεν πραγματοποιείται καμία ενέργεια.

Το πρόβλημα εντοπίστηκε τόσο στη φόρμα σύνδεσης όσο και στη φόρμα checkout, γεγονός που υποδηλώνει κοινή δυσλειτουργία στη συμπεριφορά των στοιχείων διεπαφής χρήστη της εφαρμογής.

Βήματα Αναπαραγωγής – Login Form

1. Μετάβαση στη σελίδα σύνδεσης
2. Εισαγωγή λανθασμένων ή κενών στοιχείων
3. Επιλογή του κουμπιού «Login»
4. Εμφάνιση μηνύματος σφάλματος

5. Επιλογή του εικονιδίου «X»

Αναμενόμενο Αποτέλεσμα

Τα δεδομένα των πεδίων εισαγωγής διαγράφονται επιτυχώς.

Πραγματικό Αποτέλεσμα

Τα δεδομένα παραμένουν στα πεδία εισαγωγής χωρίς να πραγματοποιείται διαγραφή.

Βήματα Αναπαραγωγής – Checkout Form

1. Προσθήκη προϊόντος στο καλάθι αγορών
2. Μετάβαση στη διαδικασία checkout
3. Εισαγωγή δεδομένων στη φόρμα
4. Πρόκληση σφάλματος εισαγωγής
5. Επιλογή του εικονιδίου «X»

Αναμενόμενο Αποτέλεσμα

Το περιεχόμενο των πεδίων εισαγωγής διαγράφεται επιτυχώς.

Πραγματικό Αποτέλεσμα

Τα δεδομένα παραμένουν στα πεδία εισαγωγής χωρίς να πραγματοποιείται διαγραφή.

Τεκμηρίωση

- Εικόνα 12.8 – Εντοπισμός σφάλματος στη φόρμα σύνδεσης
- Εικόνα 12.9 – Εντοπισμός σφάλματος στη φόρμα checkout

Επίδραση

Το συγκεκριμένο πρόβλημα επηρεάζει τη χρηστικότητα της εφαρμογής, καθώς δημιουργεί εσφαλμένη προσδοκία στον χρήστη σχετικά με τη λειτουργία του εικονιδίου διαγραφής και ενδέχεται να προκαλέσει σύγχυση κατά τη διαδικασία εισαγωγής δεδομένων.

12.4.2 Defect 02 – Ολοκλήρωση παραγγελίας με κενό καλάθι

Defect ID: DEF02

Τίτλος Defect: Επιτρέπεται ολοκλήρωση παραγγελίας με κενό καλάθι

Severity: High

Priority: High

Status: Open

Περιβάλλον Δοκιμών: Windows 11, Google Chrome

Επηρεαζόμενες Λειτουργίες: Διαδικασία checkout (Checkout Process)

Περιγραφή

Κατά τη διαδικασία testing διαπιστώθηκε ότι το σύστημα επιτρέπει την ολοκλήρωση της διαδικασίας αγοράς ακόμη και όταν το καλάθι αγορών είναι κενό. Παρόλο που δεν υπάρχουν προϊόντα προς αγορά και το συνολικό ποσό της παραγγελίας είναι μηδενικό, το κουμπί «Finish» παραμένει ενεργό και η εφαρμογή ολοκληρώνει επιτυχώς τη διαδικασία checkout.

Η συγκεκριμένη συμπεριφορά θεωρείται λειτουργική δυσλειτουργία, καθώς παραβιάζει τη βασική επιχειρησιακή λογική μιας εφαρμογής ηλεκτρονικού εμπορίου.

Βήματα Αναπαραγωγής

1. Διασφάλιση ότι το καλάθι αγορών είναι κενό
2. Μετάβαση στη διαδικασία checkout
3. Συμπλήρωση των απαιτούμενων στοιχείων
4. Επιλογή του κουμπιού «Finish»

Αναμενόμενο Αποτέλεσμα

Το σύστημα δεν επιτρέπει την ολοκλήρωση της παραγγελίας και εμφανίζει κατάλληλο μήνυμα ενημέρωσης ή σφάλματος.

Πραγματικό Αποτέλεσμα

Η διαδικασία checkout ολοκληρώνεται επιτυχώς και εμφανίζεται μήνυμα επιβεβαίωσης παραγγελίας παρά την απουσία προϊόντων στο καλάθι αγορών.

Τεκμηρίωση

- Εικόνα 12.4 – Διαδικασία checkout με κενό καλάθι αγορών
- Εικόνα 12.5 – Επιτυχής ολοκλήρωση checkout με κενό καλάθι

Επίδραση

Το συγκεκριμένο σφάλμα επηρεάζει τη βασική επιχειρησιακή λογική της εφαρμογής και ενδέχεται να οδηγήσει σε λανθασμένες ή μη έγκυρες συναλλαγές κατά τη διαδικασία ολοκλήρωσης αγοράς.

12.5 Exploratory Testing

Εκτός από τα προκαθορισμένα test cases, πραγματοποιήθηκε και exploratory testing με στόχο την αξιολόγηση της συμπεριφοράς της εφαρμογής σε μη τυπικά σενάρια χρήσης και σε απρόβλεπτες ενέργειες του χρήστη. Η συγκεκριμένη προσέγγιση επιτρέπει τον εντοπισμό σφαλμάτων και λειτουργικών αδυναμιών που ενδέχεται να μην καλύπτονται από τα δομημένα σενάρια ελέγχου.

Η διαδικασία exploratory testing πραγματοποιήθηκε παράλληλα με την εκτέλεση των βασικών test cases και επικεντρώθηκε κυρίως στη συμπεριφορά της εφαρμογής κατά τη διάρκεια της διαδικασίας checkout και της πλοήγησης στον περιηγητή ιστού.

12.5.1 Exploratory Test 01 – Ανανέωση σελίδας κατά τη διαδικασία checkout

Test ID: EXP01

Περιγραφή

Έλεγχος της συμπεριφοράς της εφαρμογής κατά την ανανέωση της σελίδας (browser refresh) κατά τη διάρκεια της διαδικασίας checkout.

Προϋποθέσεις

Ο χρήστης βρίσκεται στη διαδικασία checkout.

Βήματα Εκτέλεσης

1. Έναρξη της διαδικασίας checkout
2. Ανανέωση της σελίδας μέσω του περιηγητή ιστού

Αναμενόμενο Αποτέλεσμα

Η εφαρμογή διατηρεί την τρέχουσα κατάσταση της διαδικασίας checkout χωρίς απώλεια δεδομένων.

Πραγματικό Αποτέλεσμα

Η εφαρμογή διατήρησε επιτυχώς την τρέχουσα κατάσταση της διαδικασίας checkout μετά την ανανέωση της σελίδας, χωρίς απώλεια δεδομένων ή διακοπή της ροής χρήσης.

12.5.2 Exploratory Test 02 – Χρήση του κουμπιού επιστροφής (Back) του περιηγητή **Test ID: EXP02**

Περιγραφή

Έλεγχος της συμπεριφοράς της εφαρμογής κατά τη χρήση του κουμπιού επιστροφής (Back) του περιηγητή ιστού κατά τη διάρκεια της διαδικασίας checkout.

Προϋποθέσεις

Ο χρήστης βρίσκεται σε στάδιο της διαδικασίας checkout.

Βήματα Εκτέλεσης

1. Μετάβαση στη διαδικασία checkout
2. Επιλογή του κουμπιού «Back» του περιηγητή

Αναμενόμενο Αποτέλεσμα

Η εφαρμογή διαχειρίζεται κατάλληλα την πλοήγηση μέσω του περιηγητή

Πραγματικό Αποτέλεσμα

Η εφαρμογή επιστρέφει στη φόρμα checkout μετά τη χρήση του κουμπιού επιστροφής (Back) του περιηγητή, ωστόσο τα δεδομένα που είχαν εισαχθεί στα πεδία της φόρμας δεν διατηρούνται και απαιτείται εκ νέου εισαγωγή τους από τον χρήστη.

12.5.3 Συμπεράσματα Exploratory Testing

Τα αποτελέσματα του exploratory testing έδειξαν ότι η εφαρμογή διαχειρίζεται αποτελεσματικά ορισμένες μη τυπικές ενέργειες του χρήστη, όπως η ανανέωση της σελίδας κατά τη διάρκεια της διαδικασίας checkout, χωρίς απώλεια δεδομένων ή διακοπή της λειτουργικής ροής.

Ωστόσο, κατά τη χρήση του κουμπιού επιστροφής (Back) του περιηγητή παρατηρήθηκε ότι η εφαρμογή επιστρέφει στη φόρμα checkout χωρίς να διατηρεί τα δεδομένα που είχαν

εισαχθεί προηγουμένως από τον χρήστη. Η συγκεκριμένη συμπεριφορά ενδέχεται να επηρεάσει αρνητικά τη συνολική εμπειρία χρήστη, καθώς απαιτείται εκ νέου εισαγωγή των στοιχείων της φόρμας.

Συνολικά, η εφαρμογή παρουσιάζει ικανοποιητική διαχείριση καταστάσεων σε μη τυπικά σενάρια χρήσης, με περιθώρια βελτίωσης στη διατήρηση δεδομένων κατά την πλοήγηση μέσω λειτουργιών του περιηγητή ιστού.

12.6 Μετρικές Δοκιμών

Η καταγραφή και η ανάλυση των μετρικών δοκιμών (test metrics) αποτελούν σημαντικό μέρος της διαδικασίας testing, καθώς επιτρέπουν την ποσοτική αξιολόγηση της ποιότητας της εφαρμογής και της αποτελεσματικότητας της διαδικασίας ελέγχου.

Στο πλαίσιο της παρούσας εργασίας εκτελέστηκαν συνολικά τριάντα πέντε (35) test cases, τα οποία κάλυψαν βασικές λειτουργικές ενότητες της εφαρμογής SauceDemo, όπως η διαδικασία σύνδεσης χρηστών, η προβολή προϊόντων, η διαχείριση καλαθιού αγορών, η διαδικασία checkout και οι λειτουργίες πλοήγησης της εφαρμογής.

Από το σύνολο των test cases:

- τριάντα δύο (32) ολοκληρώθηκαν επιτυχώς,
- ενώ τρία (3) test cases απέτυχαν, οδηγώντας στον εντοπισμό δύο (2) διακριτών σφαλμάτων.

Το ποσοστό επιτυχίας των δοκιμών ανήλθε σε 91,4%, γεγονός που υποδηλώνει υψηλό επίπεδο λειτουργικότητας της εφαρμογής, παρά την ύπαρξη ορισμένων λειτουργικών και χρηστικών αδυναμιών που απαιτούν βελτίωση.

Ο Πίνακας 12.2 παρουσιάζει συνοπτικά τις βασικές μετρικές των δοκιμών που πραγματοποιήθηκαν.

Μετρική	Τιμή
Συνολικά Test Cases	35
Επιτυχημένες Δοκιμές	32
Αποτυχημένες Δοκιμές	3
Εντοπισμένα Defects	2
Ποσοστό Επιτυχίας	91.4%

Πίνακας 12.2: Συνοπτικές Μετρικές Δοκιμών

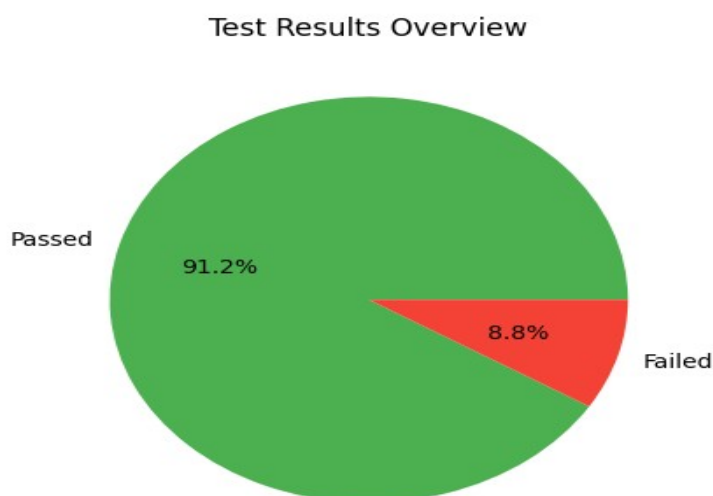
Οι μετρικές αυτές παρέχουν μια συνολική εικόνα της διαδικασίας testing και συμβάλλουν στην αξιολόγηση της σταθερότητας και της αξιοπιστίας της εφαρμογής.

12.7 Οπτικοποίηση Αποτελεσμάτων Δοκιμών

Για την καλύτερη κατανόηση και παρουσίαση των αποτελεσμάτων της διαδικασίας testing δημιουργήθηκαν διαγράμματα οπτικοποίησης, τα οποία αποτυπώνουν τη συνολική κατανομή των αποτελεσμάτων των δοκιμών και των σφαλμάτων που εντοπίστηκαν κατά την εκτέλεση των test cases.

Η χρήση διαγραμμάτων συμβάλλει στην ευκολότερη ανάλυση των αποτελεσμάτων και επιτρέπει τη συνοπτική παρουσίαση της απόδοσης της εφαρμογής ως προς τις δοκιμές που πραγματοποιήθηκαν.

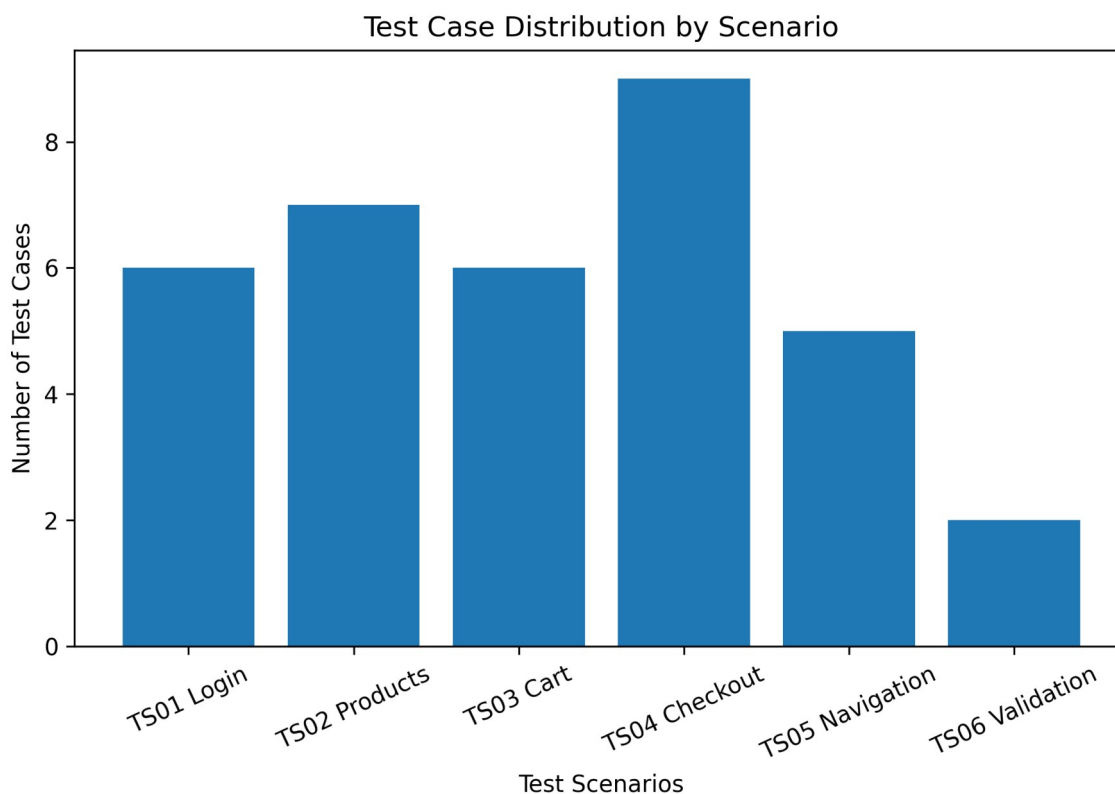
Στο σχήμα 1 παρουσιάζεται η κατανομή των αποτελεσμάτων των δοκιμών με βάση τις επιτυχημένες και αποτυχημένες εκτελέσεις test cases.



Σχήμα 1: Κατανομή αποτελεσμάτων δοκιμών (Pass / Fail)

Όπως παρατηρείται, η πλειονότητα των test cases ολοκληρώθηκε επιτυχώς, γεγονός που υποδηλώνει υψηλό επίπεδο λειτουργικότητας της εφαρμογής.

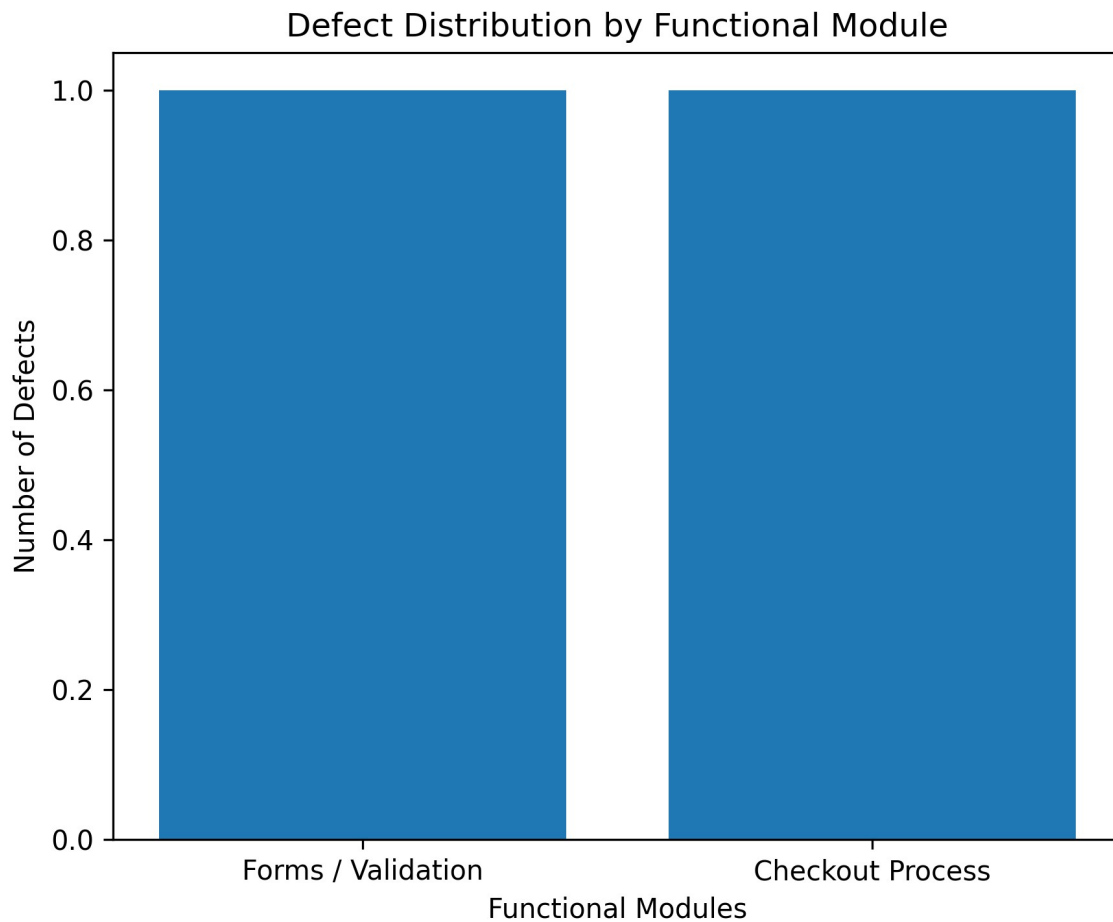
Στο σχήμα 2 παρουσιάζεται η κατανομή των test cases ανά λειτουργική ενότητα της εφαρμογής.



Σχήμα 2: Κατανομή test cases ανά λειτουργική ενότητα

Το διάγραμμα αποτυπώνει τη συνολική κάλυψη των βασικών λειτουργιών της εφαρμογής από τα test cases που σχεδιάστηκαν και εκτελέστηκαν στο πλαίσιο της παρούσας εργασίας.

Στο σχήμα 3 παρουσιάζεται η κατανομή των σφαλμάτων ανά λειτουργική ενότητα της εφαρμογής. Όπως φαίνεται από το διάγραμμα, τα σφάλματα που εντοπίστηκαν σχετίζονται κυρίως με τις φόρμες εισαγωγής δεδομένων και τη διαδικασία checkout, περιοχές οι οποίες θεωρούνται ιδιαίτερα κρίσιμες για τη συνολική λειτουργικότητα και την εμπειρία χρήστη της εφαρμογής.



Σχήμα 3: Κατανομή των εντοπισμένων σφαλμάτων ανά λειτουργική ενότητα της εφαρμογής

12.8 Ανάλυση Αποτελεσμάτων

Η εκτέλεση των δοκιμών έδειξε ότι η εφαρμογή παρουσιάζει σε γενικές γραμμές σταθερή και αξιόπιστη λειτουργία. Από το σύνολο των test cases, η πλειονότητα ολοκληρώθηκε επιτυχώς, γεγονός που υποδηλώνει υψηλό επίπεδο λειτουργικότητας στις βασικές λειτουργικές ενότητες της εφαρμογής.

Τα σφάλματα που εντοπίστηκαν αφορούν κυρίως:

- τη χρηστικότητα της διεπαφής χρήστη (clear icon),
- και τη λειτουργική λογική της διαδικασίας checkout (empty cart checkout).

Το πρώτο σφάλμα χαρακτηρίζεται ως μέτριας σοβαρότητας (Medium Severity), καθώς επηρεάζει κυρίως τη χρηστικότητα της εφαρμογής και την εμπειρία χρήστη. Αντίθετα, το

δεύτερο σφάλμα χαρακτηρίζεται ως υψηλής σοβαρότητας (High Severity), καθώς σχετίζεται με βασική λειτουργία της επιχειρησιακής λογικής της εφαρμογής και ενδέχεται να οδηγήσει σε μη έγκυρες διαδικασίες ολοκλήρωσης αγοράς.

Τα αποτελέσματα των δοκιμών επιβεβαιώνουν ότι η συστηματική προσέγγιση σχεδιασμού test cases που εφαρμόστηκε στην παρούσα εργασία συνέβαλε στον αποτελεσματικό εντοπισμό σφαλμάτων σε κρίσιμα σημεία της εφαρμογής.

Επιπλέον, η συνδυαστική χρήση δομημένων test cases και exploratory testing επέτρεψε την πληρέστερη αξιολόγηση της συμπεριφοράς του συστήματος, αναδεικνύοντας τόσο λειτουργικά όσο και χρηστικά ζητήματα που ενδέχεται να επηρεάσουν τη συνολική εμπειρία χρήστη.

12.9 Συμβολή της Παρούσας Εργασίας

Η παρούσα εργασία δεν περιορίζεται μόνο στην εκτέλεση δοκιμών λογισμικού, αλλά περιλαμβάνει τον σχεδιασμό και την εφαρμογή μιας ολοκληρωμένης και δομημένης μεθοδολογίας testing σε περιβάλλον web εφαρμογής.

Συγκεκριμένα, η συμβολή της εργασίας εντοπίζεται:

- στον συστηματικό σχεδιασμό test scenarios και test cases με βάση βασικές λειτουργικές ροές χρήστη,
- στην εφαρμογή manual testing με οργανωμένη διαδικασία καταγραφής αποτελεσμάτων,
- στην αξιοποίηση exploratory testing για την αξιολόγηση μη τυπικών σεναρίων χρήσης,
- στην καταγραφή, ανάλυση και κατηγοριοποίηση σφαλμάτων με βάση τη σοβαρότητα και την επίδρασή τους,
- στη χρήση μετρικών δοκιμών και διαγραμμάτων οπτικοποίησης για την ποσοτική αξιολόγηση της διαδικασίας testing,
- καθώς και στην ανάλυση των αποτελεσμάτων με στόχο την αξιολόγηση της λειτουργικότητας και της αξιοπιστίας της εφαρμογής.

Η προσέγγιση που ακολουθήθηκε διαφοροποιείται από μια απλή διαδικασία εκτέλεσης δοκιμών, καθώς ενσωματώνει στοιχεία μεθοδολογίας, τεκμηρίωσης, ανάλυσης και αξιολόγησης αποτελεσμάτων, προσδίδοντας ερευνητικό και ακαδημαϊκό χαρακτήρα στην εργασία.

Επιπλέον, η εργασία αναδεικνύει τη σημασία της συστηματικής διαδικασίας testing στη βελτίωση της ποιότητας λογισμικού και στη διασφάλιση της ορθής λειτουργίας web εφαρμογών σε πραγματικά σενάρια χρήσης.

12.10 Συμπεράσματα

Στο κεφάλαιο αυτό παρουσιάστηκε η διαδικασία εκτέλεσης manual testing στην εφαρμογή SauceDemo, καθώς και η ανάλυση των αποτελεσμάτων που προέκυψαν από την εφαρμογή των test scenarios και test cases που σχεδιάστηκαν στο προηγούμενο κεφάλαιο.

Η εκτέλεση των δοκιμών επέτρεψε την αξιολόγηση βασικών λειτουργικών ενοτήτων της εφαρμογής, όπως η διαδικασία σύνδεσης χρηστών, η προβολή προϊόντων, η διαχείριση καλαθιού αγορών και η διαδικασία ολοκλήρωσης αγοράς. Τα αποτελέσματα έδειξαν ότι η εφαρμογή παρουσιάζει σε γενικές γραμμές σταθερή λειτουργική συμπεριφορά, με περιορισμένο αριθμό σφαλμάτων.

Παράλληλα, μέσω της διαδικασίας exploratory testing αξιολογήθηκε η συμπεριφορά της εφαρμογής σε μη τυπικά σενάρια χρήσης, γεγονός που συνέβαλε στην πληρέστερη κατανόηση της λειτουργικότητας και της εμπειρίας χρήστη της εφαρμογής.

Η συστηματική καταγραφή των αποτελεσμάτων, η ανάλυση των σφαλμάτων και η χρήση μετρικών δοκιμών επέτρεψαν την εξαγωγή συμπερασμάτων σχετικά με την ποιότητα και την αξιοπιστία της εφαρμογής.

Στο επόμενο κεφάλαιο παρουσιάζεται η διαδικασία αυτοματοποίησης επιλεγμένων test cases με χρήση εργαλείων automation testing, καθώς και η συγκριτική αξιολόγηση μεταξύ manual testing και automated testing.

13 Αυτοματοποιημένες Δοκιμές και Συγκριτική Αξιολόγηση

13.1 Εισαγωγή

Η διαδικασία ελέγχου λογισμικού αποτελεί κρίσιμο στάδιο στον κύκλο ζωής ανάπτυξης εφαρμογών, καθώς συμβάλλει στη διασφάλιση της ποιότητας, της αξιοπιστίας και της λειτουργικής ορθότητας ενός συστήματος (ISTQB, 2023). Στο πλαίσιο της παρούσας εργασίας, η αξιολόγηση της εφαρμογής SauceDemo πραγματοποιήθηκε μέσω δύο βασικών προσεγγίσεων: του χειροκίνητου ελέγχου (manual testing) και του αυτοματοποιημένου ελέγχου (automated testing).

Σκοπός του παρόντος κεφαλαίου είναι η εις βάθος συγκριτική αξιολόγηση των δύο αυτών προσεγγίσεων, με έμφαση τόσο στη διαδικασία εκτέλεσης όσο και στα αποτελέσματα που προέκυψαν. Η ανάλυση δεν περιορίζεται σε θεωρητικό επίπεδο, αλλά βασίζεται σε πραγματική εφαρμογή και πειραματική αξιολόγηση, χρησιμοποιώντας συγκεκριμένα σενάρια δοκιμών τα οποία σχεδιάστηκαν και υλοποιήθηκαν στα προηγούμενα κεφάλαια.

Η σημασία της συγκεκριμένης αξιολόγησης έγκειται στο γεγονός ότι, στη σύγχρονη ανάπτυξη λογισμικού, η επιλογή μεταξύ manual και automated testing — ή ο συνδυασμός τους — επηρεάζει άμεσα την αποδοτικότητα της ομάδας ανάπτυξης, το κόστος συντήρησης και την ποιότητα του τελικού προϊόντος (Sommerville, 2020).

Επιπλέον, στο παρόν κεφάλαιο επιχειρείται η ποσοτική και ποιοτική αξιολόγηση των δύο προσεγγίσεων, με στόχο την εξαγωγή συμπερασμάτων σχετικά με την καταλληλότητά τους σε διαφορετικά περιβάλλοντα ανάπτυξης λογισμικού. Η ανάλυση βασίζεται σε πραγματικά δεδομένα εκτέλεσης, τα οποία προέκυψαν από την εφαρμογή των σεναρίων δοκιμών στην υπό μελέτη πλατφόρμα.

Η παρούσα μελέτη προσεγγίζει τον έλεγχο λογισμικού όχι μόνο ως τεχνική διαδικασία εκτέλεσης δοκιμών, αλλά και ως αντικείμενο συγκριτικής αξιολόγησης διαφορετικών μεθοδολογικών προσεγγίσεων. Στο πλαίσιο αυτό διερευνάται η αποτελεσματικότητα του χειροκίνητου και του αυτοματοποιημένου ελέγχου ως προς την αποδοτικότητα εκτέλεσης, την ικανότητα εντοπισμού σφαλμάτων και την καταλληλότητά τους σε διαφορετικά περιβάλλοντα ανάπτυξης λογισμικού.

13.2 Μεθοδολογική Προσέγγιση

Η συγκριτική ανάλυση που πραγματοποιήθηκε στο πλαίσιο της παρούσας εργασίας βασίστηκε σε μια συστηματική και δομημένη μεθοδολογική προσέγγιση, με στόχο την αντικειμενική αξιολόγηση των δύο βασικών μεθόδων ελέγχου λογισμικού, δηλαδή του manual testing και του automated testing (ISTQB, 2023).

Αρχικά πραγματοποιήθηκε ο σχεδιασμός των test scenarios και των αντίστοιχων test cases, τα οποία παρουσιάστηκαν στο Κεφάλαιο 11. Τα σενάρια επιλέχθηκαν με γνώμονα την κάλυψη βασικών και κρίσιμων λειτουργιών της εφαρμογής SauceDemo. Συγκεκριμένα, περιλαμβάνουν τη διαδικασία σύνδεσης χρήστη, τη διαχείριση προϊόντων στο καλάθι αγορών και τη διαδικασία ολοκλήρωσης αγοράς, οι οποίες αποτελούν χαρακτηριστικές λειτουργίες μιας εφαρμογής ηλεκτρονικού εμπορίου.

Στη συνέχεια, τα ίδια σενάρια εκτελέστηκαν με δύο διαφορετικές προσεγγίσεις, ώστε να διασφαλιστεί η συγκρισιμότητα των αποτελεσμάτων. Η πρώτη προσέγγιση αφορούσε τη χειροκίνητη εκτέλεση των δοκιμών, τα αποτελέσματα της οποίας παρουσιάστηκαν στο Κεφάλαιο 12. Η δεύτερη προσέγγιση αφορούσε την αυτοματοποιημένη εκτέλεση μέσω του Selenium WebDriver, κατά την οποία οι ίδιες ενέργειες υλοποιήθηκαν με χρήση προγραμματιστικού κώδικα.

Κατά τη διάρκεια της εκτέλεσης δόθηκε ιδιαίτερη έμφαση στην καταγραφή ποσοτικών και ποιοτικών δεδομένων, όπως ο χρόνος εκτέλεσης, η σταθερότητα των αποτελεσμάτων και η δυνατότητα επαναληψιμότητας των δοκιμών. Παράλληλα, αξιολογήθηκε η ικανότητα κάθε προσέγγισης να εντοπίζει σφάλματα τόσο σε λειτουργικό επίπεδο όσο και σε επίπεδο εμπειρίας χρήστη.

Τέλος, πραγματοποιήθηκε συγκριτική αξιολόγηση των δύο μεθόδων με βάση συγκεκριμένα κριτήρια, όπως η αποδοτικότητα, η αξιοπιστία, η ευελιξία και η ικανότητα εντοπισμού σφαλμάτων, με στόχο την εξαγωγή τεκμηριωμένων συμπερασμάτων σχετικά με την καταλληλότητά τους σε διαφορετικά περιβάλλοντα ανάπτυξης λογισμικού.

Η μεθοδολογική αυτή προσέγγιση επιτρέπει την εμπειρική αξιολόγηση των δύο τεχνικών ελέγχου υπό κοινές συνθήκες εκτέλεσης, συμβάλλοντας στην εξαγωγή συγκριτικών συμπερασμάτων σχετικά με την αποτελεσματικότητα, τα πλεονεκτήματα και τους περιορισμούς κάθε μεθόδου.

13.3 Περιβάλλον Υλοποίησης και Εργαλεία

Η ανάπτυξη και η εκτέλεση των αυτοματοποιημένων δοκιμών πραγματοποιήθηκε σε περιβάλλον που υποστηρίζει την ανάπτυξη εφαρμογών Java και τη χρήση του Selenium WebDriver για την αυτοματοποίηση αλληλεπιδράσεων με διαδικτυακές εφαρμογές. Στις επόμενες υποενότητες παρουσιάζονται τα εργαλεία που χρησιμοποιήθηκαν, καθώς και ζητήματα που σχετίζονται με την υλοποίηση, την εκτέλεση και τη συντήρηση των αυτοματοποιημένων δοκιμών.

13.3.1 Τεχνολογική Υποδομή

Η υλοποίηση των αυτοματοποιημένων δοκιμών πραγματοποιήθηκε στο περιβάλλον ανάπτυξης IntelliJ IDEA, το οποίο παρέχει ολοκληρωμένες δυνατότητες συγγραφής, εκτέλεσης και διαχείρισης κώδικα Java. Η επιλογή του συγκεκριμένου περιβάλλοντος βασίστηκε στη σταθερότητα και στις δυνατότητες που προσφέρει για ανάπτυξη και αποσφαλμάτωση εφαρμογών.

Η γλώσσα προγραμματισμού που χρησιμοποιήθηκε είναι η Java, η οποία επιλέχθηκε λόγω της ευρείας χρήσης της στον τομέα του ελέγχου λογισμικού και της πλήρους συμβατότητάς της με το Selenium WebDriver. Παράλληλα, υποστηρίζει αντικειμενοστραφή προγραμματισμό, γεγονός που διευκολύνει την οργάνωση και τη διαχείριση των αυτοματοποιημένων δοκιμών.

Το Selenium WebDriver αποτέλεσε το βασικό εργαλείο αυτοματοποίησης που χρησιμοποιήθηκε στην παρούσα εργασία. Μέσω αυτού κατέστη δυνατή η προσομοίωση ενεργειών του χρήστη στον φυλλομετρητή, όπως η εισαγωγή δεδομένων, η επιλογή στοιχείων διεπαφής χρήστη και η πλοήγηση μεταξύ διαφορετικών σελίδων της εφαρμογής.

Η προσέγγιση που υλοποιήθηκε είχε κυρίως εκπαιδευτικό και πειραματικό χαρακτήρα, με στόχο τη συγκριτική αξιολόγηση του automated σε σχέση με το manual testing και όχι την ανάπτυξη ενός πλήρως επεκτάσιμου πλαισίου αυτοματοποίησης.

Η διαχείριση των εξαρτήσεων πραγματοποιήθηκε με το Maven, το οποίο διευκολύνει την ενσωμάτωση βιβλιοθηκών και εξασφαλίζει τη σωστή λειτουργία του έργου. Επιπλέον, χρησιμοποιήθηκε το WebDriverManager για την αυτόματη διαχείριση του προγράμματος οδήγησης του φυλλομετρητή, μειώνοντας την ανάγκη χειροκίνητης ρύθμισης.

Οι δοκιμές εκτελέστηκαν στον φυλλομετρητή Google Chrome, προκειμένου να εξασφαλιστούν ρεαλιστικές συνθήκες εκτέλεσης, καθώς πρόκειται για έναν από τους πιο διαδεδομένους φυλλομετρητές.

Για τη διευκόλυνση της παρουσίασης του περιβάλλοντος υλοποίησης, ο Πίνακας 13.1 συνοψίζει τα εργαλεία και τις τεχνολογίες που αξιοποιήθηκαν κατά την ανάπτυξη και εκτέλεση των αυτοματοποιημένων δοκιμών.

Εργαλείο / Τεχνολογία	Ρόλος
Java	Ανάπτυξη αυτοματοποιημένων δοκιμών
Selenium WebDriver	Αυτοματοποίηση αλληλεπίδρασης με την εφαρμογή
IntelliJ IDEA	Περιβάλλον ανάπτυξης
Maven	Διαχείριση εξαρτήσεων και βιβλιοθηκών
WebDriverManager	Αυτόματη διαχείριση του ChromeDriver
ChromeDriver	Εκτέλεση των δοκιμών

Πίνακας 13.1: Εργαλεία και Τεχνολογίες Υλοποίησης

13.3.2 Ζητήματα Υλοποίησης και Εκτέλεσης

Κατά την ανάπτυξη και εκτέλεση των αυτοματοποιημένων δοκιμών προέκυψαν ορισμένα τεχνικά ζητήματα τα οποία έπρεπε να αντιμετωπιστούν, ώστε να διασφαλιστεί η σταθερή και αξιόπιστη λειτουργία των σεναρίων ελέγχου.

Ένα από τα σημαντικότερα ζητήματα αφορούσε τον συγχρονισμό των ενεργειών του αυτοματοποιημένου σεναρίου με τη φόρτωση των στοιχείων της εφαρμογής. Οι σύγχρονες διαδικτυακές εφαρμογές περιλαμβάνουν δυναμικά στοιχεία διεπαφής χρήστη, τα οποία ενδέχεται να μην είναι άμεσα διαθέσιμα κατά την εκτέλεση των ενεργειών του Selenium WebDriver.

Στο πλαίσιο της παρούσας εργασίας χρησιμοποιήθηκε η μέθοδος Thread.sleep() για την προσωρινή παύση της εκτέλεσης μεταξύ συγκεκριμένων ενεργειών. Η επιλογή αυτή πραγματοποιήθηκε με στόχο την απλοποίηση της υλοποίησης και τη διευκόλυνση της καταγραφής στιγμιότυπων οθόνης κατά την εκτέλεση των δοκιμών.

Ωστόσο, στη σύγχρονη πρακτική του αυτοματοποιημένου ελέγχου προτιμώνται μηχανισμοί δυναμικής αναμονής, όπως οι WebDriverWait και ExpectedConditions, οι οποίοι επιτρέπουν την αναμονή μέχρι να ικανοποιηθεί μία συγκεκριμένη συνθήκη. Η προσέγγιση

αυτή συμβάλλει στη βελτίωση της αξιοπιστίας των δοκιμών και στη μείωση αποτυχιών που οφείλονται σε καθυστερήσεις φόρτωσης των στοιχείων της εφαρμογής.

Επιπλέον, κατά την εκτέλεση των δοκιμών παρατηρήθηκε η εμφάνιση αναδυόμενου παραθύρου από τον φυλλομετρητή Google Chrome, το οποίο σχετίζεται με τη διαχείριση κωδικών πρόσβασης. Το παράθυρο αυτό επηρέαζε την ομαλή εκτέλεση των δοκιμών, καθώς κάλυπτε στοιχεία της σελίδας και εμπόδιζε την αλληλεπίδραση του Selenium WebDriver με αυτά.

Για την αντιμετώπιση του προβλήματος χρησιμοποιήθηκε περιήγηση σε κατάσταση Incognito, η οποία εξασφαλίζει καθαρό περιβάλλον εκτέλεσης χωρίς αποθηκευμένα δεδομένα, κωδικούς πρόσβασης ή άλλους μηχανισμούς που ενδέχεται να επηρεάσουν την εκτέλεση των αυτοματοποιημένων δοκιμών.

Η αντιμετώπιση των παραπάνω ζητημάτων συνέβαλε στη βελτίωση της αξιοπιστίας και της επαναληψιμότητας των δοκιμών, επιτρέποντας την ορθή αξιολόγηση της λειτουργικότητας της εφαρμογής.

13.3.3 Συντήρηση και Επεκτασιμότητα

Ένας από τους σημαντικότερους παράγοντες στην ανάπτυξη αυτοματοποιημένων δοκιμών είναι η δυνατότητα συντήρησης και επεκτασιμότητας του κώδικα. Σε περιβάλλοντα ανάπτυξης λογισμικού όπου η εφαρμογή μεταβάλλεται συχνά, η δομή και η οργάνωση των δοκιμών επηρεάζουν άμεσα την αξιοπιστία και το κόστος συντήρησης της διαδικασίας ελέγχου.

Στην παρούσα εργασία, η υλοποίηση των αυτοματοποιημένων δοκιμών πραγματοποιήθηκε με απλοποιημένη αρχιτεκτονική προσέγγιση, με στόχο την κατανόηση της βασικής λειτουργίας του Selenium WebDriver και τη συγκριτική αξιολόγηση του αυτοματοποιημένου ελέγχου σε σχέση με τον χειροκίνητο έλεγχο. Για τον λόγο αυτό, τα σενάρια υλοποιήθηκαν ως ανεξάρτητα προγράμματα χωρίς τη χρήση σύνθετων αρχιτεκτονικών προτύπων αυτοματοποίησης.

Στη σύγχρονη πρακτική του ελέγχου λογισμικού, η ανάπτυξη μεγάλων συνόλων αυτοματοποιημένων δοκιμών βασίζεται συχνά σε αρχιτεκτονικές όπως το Page Object Model (POM), το οποίο επιτρέπει τον διαχωρισμό της λογικής των δοκιμών από τα στοιχεία

διεπαφής χρήστη. Η προσέγγιση αυτή συμβάλλει στη βελτίωση της συντηρησιμότητας του κώδικα και μειώνει τον αντίκτυπο πιθανών αλλαγών στη διεπαφή της εφαρμογής.

Επιπλέον, σε παραγωγικά περιβάλλοντα χρησιμοποιούνται συχνά πλαίσια δοκιμών όπως τα JUnit και TestNG, τα οποία παρέχουν δυνατότητες ομαδοποίησης δοκιμών, δημιουργίας αναφορών αποτελεσμάτων και αυτοματοποιημένης εκτέλεσης μεγάλου αριθμού test scenarios.

Παρότι οι τεχνικές αυτές δεν υλοποιήθηκαν στην παρούσα εργασία, η αναφορά τους κρίνεται σημαντική, καθώς αποτελούν καθιερωμένες πρακτικές στη σύγχρονη ανάπτυξη αυτοματοποιημένων δοκιμών λογισμικού.

Η προσέγγιση που ακολουθήθηκε μπορεί να αποτελέσει βάση για μελλοντική επέκταση του αυτοματοποιημένου ελέγχου της εφαρμογής, μέσω της προσθήκης νέων σεναρίων και της υιοθέτησης πιο προηγμένων τεχνικών αυτοματοποίησης.

13.4 Υλοποίηση Αυτοματοποιημένων Δοκιμών

Στην ενότητα αυτή παρουσιάζεται η ανάπτυξη και η εκτέλεση των αυτοματοποιημένων δοκιμών που υλοποιήθηκαν για την εφαρμογή SauceDemo. Τα σενάρια που αυτοματοποιήθηκαν βασίζονται σε επιλεγμένα test cases τα οποία σχεδιάστηκαν στο Κεφάλαιο 11 και εκτελέστηκαν χειροκίνητα στο Κεφάλαιο 12.

Στόχος της αυτοματοποίησης ήταν η επαλήθευση βασικών λειτουργιών της εφαρμογής μέσω επαναλαμβανόμενης και αξιόπιστης εκτέλεσης δοκιμών. Για κάθε σενάριο αναπτύχθηκε αντίστοιχο αυτοματοποιημένο script με χρήση του Selenium WebDriver και της γλώσσας προγραμματισμού Java, το οποίο προσομοιώνει τις ενέργειες ενός πραγματικού χρήστη κατά την αλληλεπίδρασή του με την εφαρμογή.

Οι αυτοματοποιημένες δοκιμές εκτελέστηκαν στο περιβάλλον που περιγράφηκε στην προηγούμενη ενότητα και τα αποτελέσματά τους καταγράφηκαν με σκοπό την αξιολόγηση της λειτουργικότητας της εφαρμογής και τη σύγκριση της διαδικασίας αυτοματοποιημένου και χειροκίνητου ελέγχου.

Για τη συγγραφή και βελτιστοποίηση των σεναρίων αυτοματοποιημένων δοκιμών (automated test cases) σε περιβάλλον Java/Selenium, χρησιμοποιήθηκαν εργαλεία Τεχνητής Νοημοσύνης (AI assistants) ως βοηθητικά μέσα ανάπτυξης του κώδικα.

13.4.1 Αναλυτική Εκτέλεση Test Scenario: Επιτυχής Σύνδεση Χρήστη

Η διαδικασία σύνδεσης χρήστη αποτελεί μία από τις βασικότερες λειτουργίες της εφαρμογής, καθώς σχετίζεται άμεσα με την αυθεντικοποίηση και την πρόσβαση σε προστατευμένες λειτουργίες. Η ορθή λειτουργία του συγκεκριμένου σεναρίου είναι ιδιαίτερα σημαντική, καθώς επηρεάζει τόσο την ασφάλεια όσο και τη συνολική εμπειρία χρήστη.

Χειροκίνητη Εκτέλεση

Κατά τη χειροκίνητη εκτέλεση της δοκιμής, ο ελεγκτής ξεκίνησε ανοίγοντας τον φυλλομετρητή και μεταβαίνοντας στη σελίδα της εφαρμογής. Στη συνέχεια εντόπισε τα πεδία εισαγωγής ονόματος χρήστη και κωδικού πρόσβασης και εισήγαγε τα έγκυρα δεδομένα.

Ακολούθως επέλεξε το κουμπί σύνδεσης και παρατήρησε τη συμπεριφορά της εφαρμογής. Η επιτυχία της διαδικασίας επιβεβαιώθηκε μέσω της μετάβασης στη σελίδα προϊόντων, γεγονός που υποδηλώνει ότι η αυθεντικοποίηση ολοκληρώθηκε επιτυχώς. Η διαδικασία αυτή απαιτεί ιδιαίτερη προσοχή, καθώς οποιοδήποτε σφάλμα στην εισαγωγή των στοιχείων σύνδεσης μπορεί να οδηγήσει σε αποτυχία του σεναρίου.

Αυτοματοποιημένη Εκτέλεση

Στην αυτοματοποιημένη εκτέλεση, η ίδια διαδικασία υλοποιήθηκε μέσω του Selenium WebDriver, όπου οι ενέργειες του χρήστη αναπαράγονται μέσω προγραμματιστικού κώδικα.

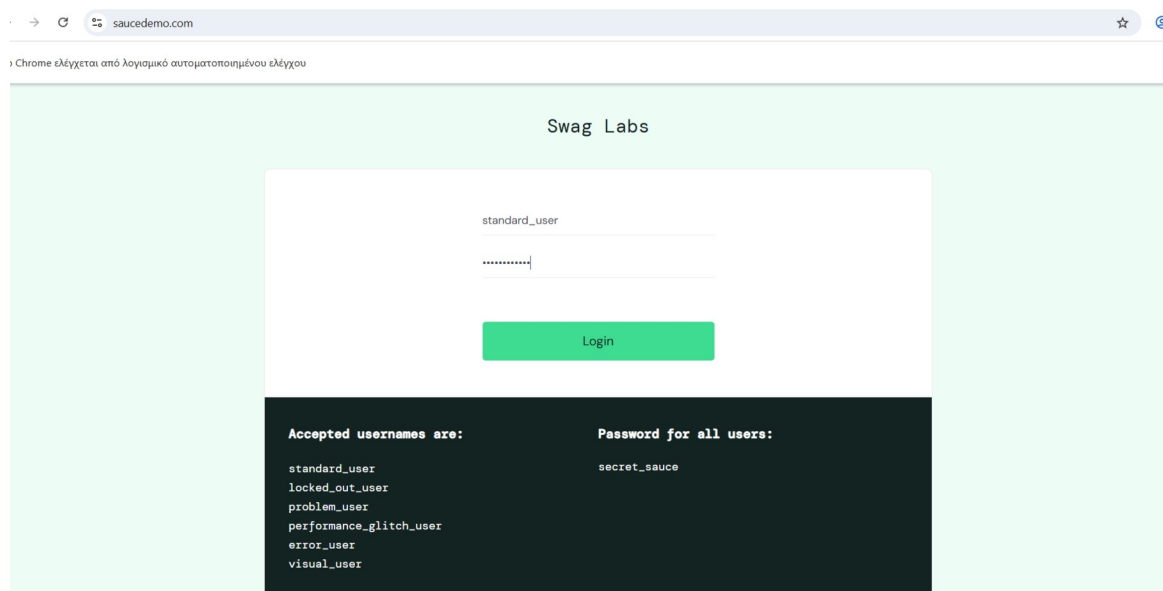
```
driver.findElement(By.id("user-name")).sendKeys("standard_user");  
driver.findElement(By.id("password")).sendKeys("secret_sauce");  
driver.findElement(By.id("login-button")).click();
```

Κώδικας 13.1 – Υλοποίηση αυτοματοποιημένης διαδικασίας σύνδεσης

Ο κώδικας εντοπίζει τα στοιχεία της σελίδας μέσω των μοναδικών αναγνωριστικών τους και εκτελεί τις αντίστοιχες ενέργειες με ακρίβεια. Η επιτυχία της δοκιμής επαληθεύεται

μέσω ελέγχου της διεύθυνσης URL, η οποία οδηγεί στη σελίδα προϊόντων, επιβεβαιώνοντας ότι η εφαρμογή μεταφέρθηκε στη σωστή σελίδα.

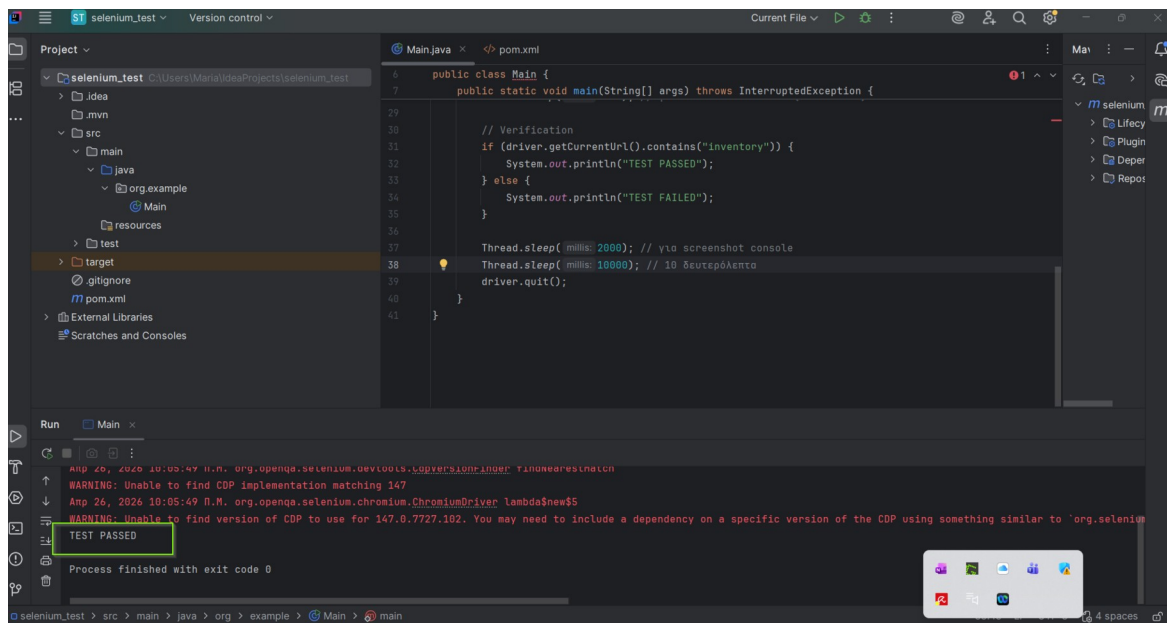
Η αυτόματη συμπλήρωση των στοιχείων σύνδεσης παρουσιάζεται στην Εικόνα 13.1, όπου αποτυπώνεται η εκτέλεση της διαδικασίας χωρίς ανθρώπινη παρέμβαση.



Εικόνα 13.1: Αυτόματη συμπλήρωση στοιχείων σύνδεσης μέσω Selenium WebDriver

Πηγή: Διαδικτυακή πλατφόρμα δοκιμών Sauce Demo (saucedemo.com)

Παράλληλα, η επιτυχής ολοκλήρωση της δοκιμής επιβεβαιώνεται και μέσω της κονσόλας εκτέλεσης, όπως φαίνεται στην Εικόνα 13.2, όπου εμφανίζεται το μήνυμα επιτυχίας της δοκιμής.



Εικόνα 13.2: Επιτυχής εκτέλεση του αυτοματοποιημένου σεναρίου στην κονσόλα του IntelliJ (TEST PASSED)

Κατά την εκτέλεση εμφανίστηκαν ορισμένα προειδοποιητικά μηνύματα, τα οποία δεν επηρέασαν την επιτυχή ολοκλήρωση της δοκιμής.

Σε σύγκριση με τη χειροκίνητη εκτέλεση, η αυτοματοποιημένη διαδικασία ολοκληρώνεται σε σημαντικά μικρότερο χρόνο και με υψηλό βαθμό επαναληψιμότητας, μειώνοντας σημαντικά την πιθανότητα ανθρώπινου σφάλματος. Το συγκεκριμένο σενάριο αναδεικνύει τα βασικά πλεονεκτήματα της αυτοματοποίησης σε κρίσιμες λειτουργίες της εφαρμογής.

13.4.2 Αναλυτική Εκτέλεση Test Scenario: Αποτυχημένη Σύνδεση

Η διαδικασία αποτυχημένης σύνδεσης αποτελεί ένα κρίσιμο σενάριο ελέγχου, καθώς αξιολογεί την ικανότητα του συστήματος να διαχειρίζεται λανθασμένα δεδομένα εισόδου και να παρέχει κατάλληλη ανατροφοδότηση στον χρήστη. Η ορθή εμφάνιση μηνυμάτων σφάλματος είναι ιδιαίτερα σημαντική για τη χρηστικότητα και την ασφάλεια της εφαρμογής.

Χειροκίνητη Εκτέλεση

Κατά τη χειροκίνητη εκτέλεση, ο ελεγκτής ακολούθησε την ίδια διαδικασία με το σενάριο επιτυχούς σύνδεσης, με τη διαφορά ότι εισήγαγε σκόπιμα λανθασμένο κωδικό πρόσβασης. Συγκεκριμένα, πραγματοποιήθηκαν τα εξής βήματα: εισαγωγή έγκυρου ονόματος χρήστη, εισαγωγή λανθασμένου κωδικού πρόσβασης και επιλογή του κουμπιού σύνδεσης.

Μετά την εκτέλεση των παραπάνω ενεργειών, η εφαρμογή δεν επέτρεψε την πρόσβαση και εμφάνισε μήνυμα σφάλματος, το οποίο ενημέρωνε τον χρήστη ότι τα στοιχεία σύνδεσης δεν είναι έγκυρα.

Η επιτυχία της δοκιμής βασίστηκε στην οπτική επιβεβαίωση της εμφάνισης του μηνύματος σφάλματος. Ωστόσο, η διαδικασία αυτή παρουσιάζει έναν βαθμό υποκειμενικότητας, καθώς εξαρτάται από την προσοχή του ελεγκτή και δεν διασφαλίζει πλήρη επαναληψιμότητα.

Αυτοματοποιημένη Εκτέλεση

Στην αυτοματοποιημένη εκτέλεση, η ίδια διαδικασία υλοποιήθηκε μέσω του Selenium WebDriver. Η βασική διαφοροποίηση σε σχέση με το προηγούμενο σενάριο είναι η χρήση λανθασμένου κωδικού πρόσβασης, με στόχο την πρόκληση αποτυχίας σύνδεσης και την επαλήθευση της αντίδρασης του συστήματος.

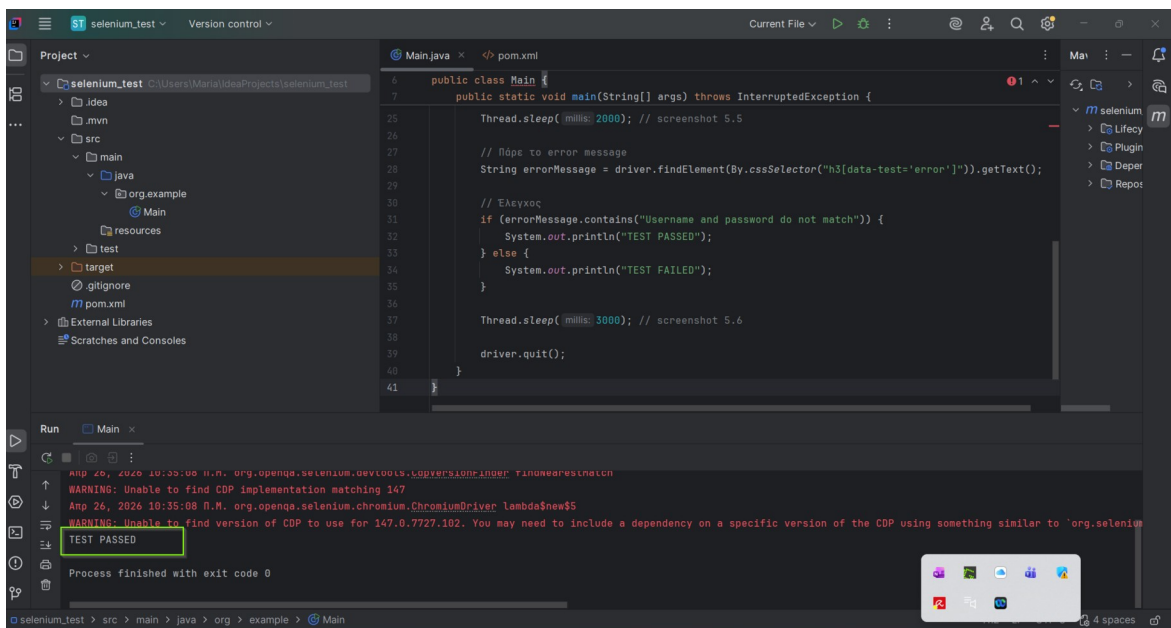
```
driver.findElement(By.id("user-name")).sendKeys("standard_user");  
driver.findElement(By.id("password")).sendKeys("wrong_password");  
driver.findElement(By.id("login-button")).click();  
  
String errorMessage = driver.findElement(By.cssSelector("h3[data-  
test='error']")).getText();  
  
if (errorMessage.contains("Username and password do not match")) {  
    System.out.println("TEST PASSED");  
} else {  
    System.out.println("TEST FAILED");  
}
```

Κώδικας 13.2 – Υλοποίηση αυτοματοποιημένης αποτυχημένης σύνδεσης

Κατά την εκτέλεση του αυτοματοποιημένου σεναρίου, το σύστημα εισάγει τα στοιχεία σύνδεσης και επιχειρεί την είσοδο στην εφαρμογή. Δεδομένου ότι ο κωδικός πρόσβασης είναι λανθασμένος, η εφαρμογή επιστρέφει το αντίστοιχο μήνυμα σφάλματος.

Ο αυτοματοποιημένος έλεγχος δεν βασίζεται σε οπτική επιβεβαίωση, αλλά ανακτά το περιεχόμενο του μηνύματος και το συγκρίνει με το αναμενόμενο αποτέλεσμα. Με τον τρόπο αυτό διασφαλίζεται μεγαλύτερη αντικειμενικότητα και ακρίβεια κατά την αξιολόγηση της λειτουργικότητας.

Η επιτυχής ολοκλήρωση της δοκιμής επιβεβαιώνεται και μέσω της κονσόλας εκτέλεσης, όπως φαίνεται στην Εικόνα 13.3.



Εικόνα 13.3: Αποτέλεσμα εκτέλεσης του αυτοματοποιημένου σεναρίου στην κονσόλα του IntelliJ (TEST PASSED)

Η επιτυχία της δοκιμής καθορίστηκε μέσω σύγκρισης του περιεχομένου του μηνύματος με το αναμενόμενο αποτέλεσμα. Η προσέγγιση αυτή εξαλείφει την ανάγκη οπτικής επιβεβαίωσης και επιτρέπει την αντικειμενική αξιολόγηση του αποτελέσματος.

Σε σύγκριση με τη χειροκίνητη εκτέλεση, η αυτοματοποιημένη διαδικασία παρέχει αυξημένη ακρίβεια και επαναληψιμότητα, καθώς η επαλήθευση πραγματοποιείται μέσω άμεσου ελέγχου των δεδομένων που επιστρέφει η εφαρμογή και όχι μέσω ανθρώπινης παρατήρησης.

13.4.3 Αναλυτική Εκτέλεση Test Scenario: Προσθήκη Προϊόντος στο Καλάθι

Η λειτουργία προσθήκης προϊόντος στο καλάθι αποτελεί βασικό στοιχείο της εμπειρίας χρήστη σε εφαρμογές ηλεκτρονικού εμπορίου, καθώς σχετίζεται άμεσα με τη διαδικασία αγοράς. Η ορθή λειτουργία του συγκεκριμένου σεναρίου είναι ιδιαίτερα σημαντική, καθώς επηρεάζει τη ροή της παραγγελίας και τη συνολική αξιοπιστία του συστήματος.

Χειροκίνητη Εκτέλεση

Κατά τη χειροκίνητη εκτέλεση, ο χρήστης, μετά την επιτυχή σύνδεση στην εφαρμογή, εντόπισε ένα προϊόν από τη λίστα διαθέσιμων προϊόντων και επέλεξε το κουμπί προσθήκης στο καλάθι. Μετά την ολοκλήρωση της ενέργειας, παρατηρήθηκε αλλαγή στο εικονίδιο του

καλαθιού, όπου εμφανίστηκε η ένδειξη «1», υποδηλώνοντας ότι το προϊόν προστέθηκε επιτυχώς.

Η επιβεβαίωση της επιτυχίας βασίστηκε στην οπτική παρατήρηση της αλλαγής στη διεπαφή χρήστη. Ωστόσο, η διαδικασία αυτή απαιτεί προσοχή από τον ελεγκτή και ενδέχεται να εισάγει έναν βαθμό υποκειμενικότητας κατά την αξιολόγηση του αποτελέσματος.

Αυτοματοποιημένη Εκτέλεση

Στην αυτοματοποιημένη εκτέλεση, η ίδια διαδικασία υλοποιήθηκε μέσω του Selenium WebDriver, όπου οι ενέργειες του χρήστη εκτελούνται μέσω προγραμματιστικού κώδικα.

Το βασικό τμήμα του κώδικα που υλοποιεί τη λειτουργία παρουσιάζεται παρακάτω.

```
driver.findElement(By.id("add-to-cart-sauce-labs-backpack")).click();

String cartValue = "";

try {

    cartValue =
driver.findElement(By.className("shopping_cart_badge")).getText();

} catch (Exception e) {

    System.out.println("Cart not found");

}

if (cartValue.equals("1")) {

    System.out.println("TEST PASSED");

} else {

    System.out.println("TEST FAILED");

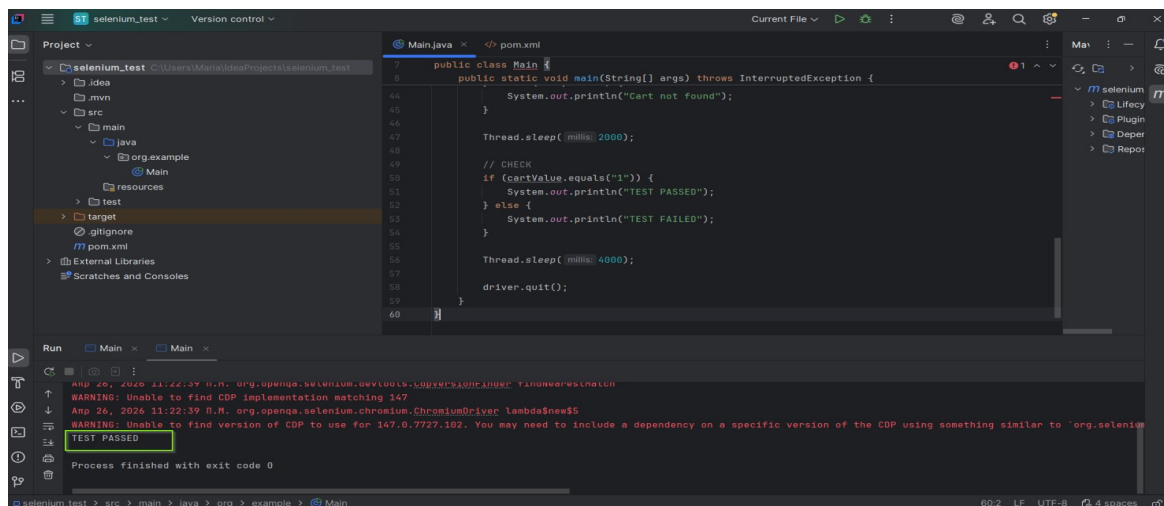
}
```

Κώδικας 13.3 – Υλοποίηση αυτοματοποιημένης προσθήκης προϊόντος στο καλάθι

Κατά την εκτέλεση του σεναρίου, το σύστημα επιλέγει το κουμπί προσθήκης προϊόντος και στη συνέχεια ανακτά την τιμή που εμφανίζεται στο εικονίδιο του καλαθιού. Η επιτυχία της ενέργειας επιβεβαιώνεται μέσω της εμφάνισης της ένδειξης «1» στο καλάθι, γεγονός που υποδηλώνει ότι το προϊόν προστέθηκε επιτυχώς.

Παράλληλα, μετά την επιτυχή προσθήκη του προϊόντος, το κουμπί «Add to Cart» μετατρέπεται σε «Remove», επιβεβαιώνοντας ότι το προϊόν βρίσκεται πλέον στο καλάθι αγορών.

Η επιτυχής ολοκλήρωση της δοκιμής επιβεβαιώνεται και μέσω της κονσόλας εκτέλεσης, όπως παρουσιάζεται στην Εικόνα 13.4.



Εικόνα 13.4: Αποτέλεσμα εκτέλεσης του αυτοματοποιημένου σεναρίου στην κονσόλα του IntelliJ (TEST PASSED)

Σε αντίθεση με τη χειροκίνητη εκτέλεση, η αυτοματοποίηση επιτρέπει την άμεση και επαναλαμβανόμενη επαλήθευση της λειτουργίας χωρίς την ανάγκη οπτικής επιβεβαίωσης από τον χρήστη. Η σύγκριση των αποτελεσμάτων πραγματοποιείται μέσω προγραμματιστικού ελέγχου, εξασφαλίζοντας αυξημένη ακρίβεια, αξιοπιστία και επαναληψιμότητα στη διαδικασία ελέγχου.

13.4.4 Αναλυτική Εκτέλεση Test Scenario: Αφαίρεση Προϊόντος από το Καλάθι

Η λειτουργία αφαίρεσης προϊόντος από το καλάθι αποτελεί βασική ενέργεια διαχείρισης αγορών, καθώς επιτρέπει στον χρήστη να τροποποιεί το περιεχόμενο της παραγγελίας του πριν από την ολοκλήρωσή της. Η ορθή λειτουργία της είναι σημαντική για τη διατήρηση της συνέπειας των δεδομένων και τη συνολική αξιοπιστία της εφαρμογής.

Χειροκίνητη Εκτέλεση

Κατά τη χειροκίνητη εκτέλεση, ο χρήστης, αφού είχε προηγουμένως προσθέσει ένα προϊόν στο καλάθι, επέλεξε το κουμπί «Remove». Μετά την εκτέλεση της ενέργειας,

παρατηρήθηκε ότι το προϊόν αφαιρέθηκε επιτυχώς και η ένδειξη του καλαθιού αγορών εξαφανίστηκε.

Η επιτυχία της δοκιμής επιβεβαιώθηκε μέσω της οπτικής παρατήρησης της απουσίας της ένδειξης στο εικονίδιο του καλαθιού, γεγονός που υποδηλώνει ότι δεν υπάρχει πλέον προϊόν καταχωρημένο στο καλάθι αγορών.

Αυτοματοποιημένη Εκτέλεση

Στην αυτοματοποιημένη εκτέλεση, η ίδια διαδικασία υλοποιήθηκε μέσω του Selenium WebDriver. Το σύστημα επιλέγει το κουμπί αφαίρεσης προϊόντος και στη συνέχεια ελέγχει εάν η ένδειξη του καλαθιού εξακολουθεί να υπάρχει.

Το βασικό τμήμα του κώδικα που υλοποιεί τη λειτουργία παρουσιάζεται παρακάτω.

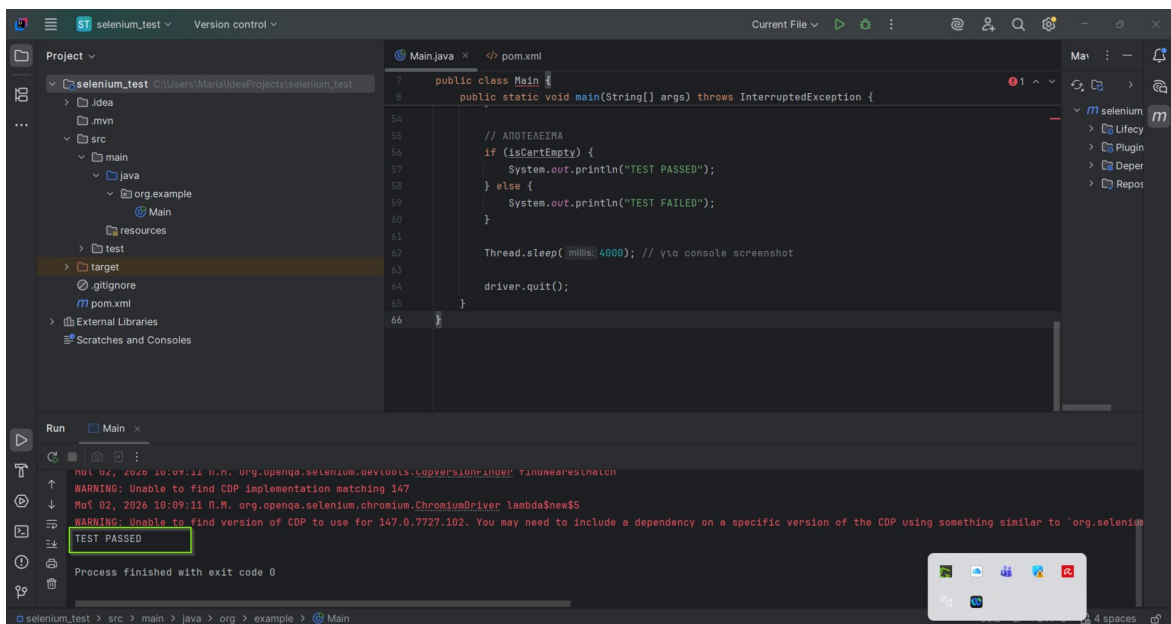
```
driver.findElement(By.id("remove-sauce-labs-backpack")).click();
boolean isEmpty;
try {
    driver.findElement(By.className("shopping_cart_badge"));
    isEmpty = false;
} catch (Exception e) {
    isEmpty = true;
}
if (isEmpty) {
    System.out.println("TEST PASSED");
} else {
    System.out.println("TEST FAILED");
}
```

Κώδικας 13.4 – Υλοποίηση αυτοματοποιημένης αφαίρεσης προϊόντος από το καλάθι

Κατά την εκτέλεση του σεναρίου, το σύστημα αφαιρεί το προϊόν από το καλάθι και στη συνέχεια επιχειρεί να εντοπίσει την ένδειξη του καλαθιού. Σε περίπτωση που το συγκεκριμένο στοιχείο δεν εντοπιστεί, θεωρείται ότι η αφαίρεση πραγματοποιήθηκε επιτυχώς.

Η επιτυχία της δοκιμής καθορίζεται μέσω ελέγχου της μη ύπαρξης του στοιχείου, γεγονός που διασφαλίζει αντικειμενικότητα και ακρίβεια κατά την αξιολόγηση του αποτελέσματος.

Η επιτυχής ολοκλήρωση της δοκιμής επιβεβαιώνεται και μέσω της κονσόλας εκτέλεσης, όπως παρουσιάζεται στην Εικόνα 13.5.



Εικόνα 13.5: Αποτέλεσμα εκτέλεσης του αυτοματοποιημένου σεναρίου στην κονσόλα του IntelliJ (TEST PASSED)

Σε αντίθεση με τη χειροκίνητη εκτέλεση, η αυτοματοποίηση επιτρέπει τον άμεσο έλεγχο της κατάστασης του καλαθιού χωρίς την ανάγκη οπτικής επιβεβαίωσης από τον ελεγκτή. Με τον τρόπο αυτό μειώνεται η πιθανότητα ανθρώπινου σφάλματος και αυξάνεται η αξιοπιστία και η επαναληψιμότητα των αποτελεσμάτων.

13.4.5 Αναλυτική Εκτέλεση Σεναρίου Δοκιμής: Διαδικασία Checkout

Η διαδικασία ολοκλήρωσης αγοράς αποτελεί ένα από τα σημαντικότερα σενάρια ελέγχου της εφαρμογής, καθώς περιλαμβάνει πολλαπλά διαδοχικά βήματα και απαιτεί την εισαγωγή στοιχείων από τον χρήστη. Η ορθή λειτουργία της είναι καθοριστική για την επιτυχή ολοκλήρωση της διαδικασίας αγοράς και τη συνολική εμπειρία χρήστη.

Χειροκίνητη Εκτέλεση

Κατά τη χειροκίνητη εκτέλεση, ο χρήστης, αφού είχε προηγουμένως προσθέσει ένα προϊόν στο καλάθι, επέλεξε το εικονίδιο του καλαθιού και στη συνέχεια το κουμπί ολοκλήρωσης αγοράς (Checkout). Ακολούθως συμπλήρωσε τα απαιτούμενα στοιχεία, όπως το όνομα, το επώνυμο και τον ταχυδρομικό κώδικα, και προχώρησε στο επόμενο στάδιο της διαδικασίας.

Η επιτυχία της δοκιμής επιβεβαιώθηκε μέσω της μετάβασης στη σελίδα επιβεβαίωσης στοιχείων παραγγελίας (Checkout Overview), γεγονός που υποδηλώνει ότι τα δεδομένα καταχωρήθηκαν επιτυχώς και η διαδικασία εκτελέστηκε σωστά.

Αυτοματοποιημένη Εκτέλεση

Στην αυτοματοποιημένη εκτέλεση, η ίδια διαδικασία υλοποιήθηκε μέσω του Selenium WebDriver, όπου οι ενέργειες του χρήστη εκτελούνται διαδοχικά μέσω προγραμματιστικού κώδικα.

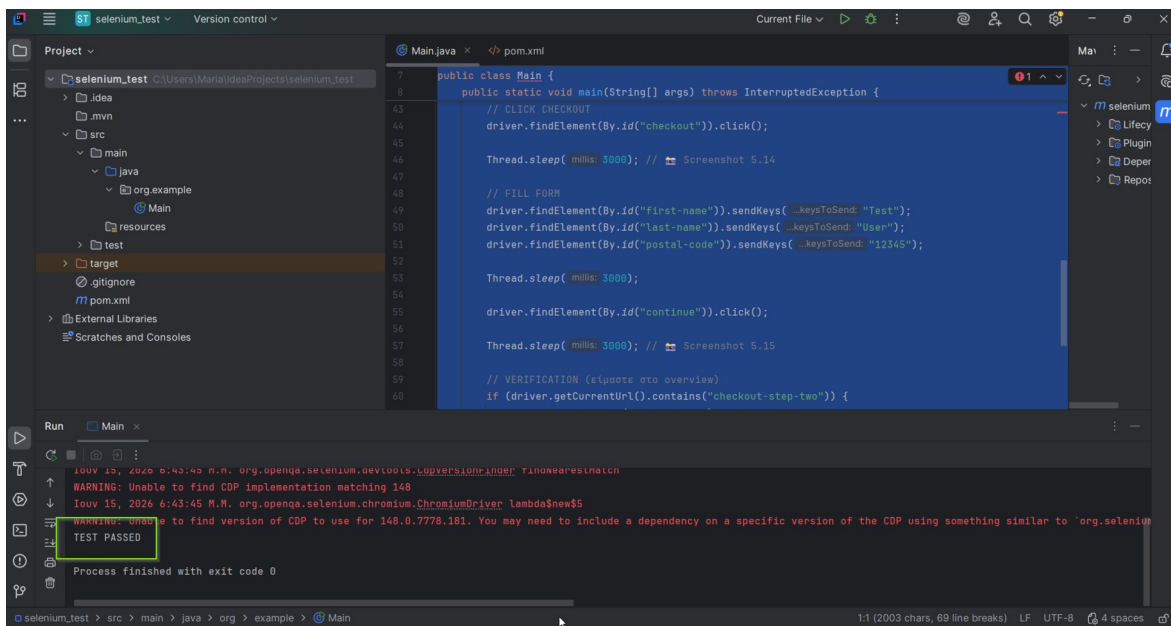
```
driver.findElement(By.className("shopping_cart_link")).click();
driver.findElement(By.id("checkout")).click();
driver.findElement(By.id("first-name")).sendKeys("Test");
driver.findElement(By.id("last-name")).sendKeys("User");
driver.findElement(By.id("postal-code")).sendKeys("12345");
driver.findElement(By.id("continue")).click();
if (driver.getCurrentUrl().contains("checkout-step-two")) {
    System.out.println("TEST PASSED");
} else {
    System.out.println("TEST FAILED");
}
```

Κώδικας 13.5 – Υλοποίηση αυτοματοποιημένης διαδικασίας checkout

Κατά την εκτέλεση του σεναρίου, το σύστημα μεταβαίνει στο καλάθι αγορών, επιλέγει την έναρξη της διαδικασίας checkout, συμπληρώνει τα απαιτούμενα στοιχεία χρήστη και προχωρά στο επόμενο στάδιο της διαδικασίας.

Η επιτυχία της δοκιμής επαληθεύεται μέσω ελέγχου της διεύθυνσης URL της εφαρμογής. Εφόσον η εφαρμογή μεταβεί στη σελίδα επιβεβαίωσης στοιχείων παραγγελίας, θεωρείται ότι η διαδικασία ολοκληρώθηκε επιτυχώς.

Παράλληλα, η επιτυχής ολοκλήρωση της δοκιμής επιβεβαιώνεται και μέσω της κονσόλας εκτέλεσης, όπου εμφανίζεται το μήνυμα επιτυχίας της δοκιμής.



Εικόνα 13.6: Αποτέλεσμα εκτέλεσης του αυτοματοποιημένου σεναρίου στην κονσόλα του IntelliJ (TEST PASSED)

Η αυτοματοποίηση της διαδικασίας checkout επιτρέπει την επαλήθευση μιας σύνθετης ακολουθίας ενεργειών με ακρίβεια και συνέπεια, μειώνοντας σημαντικά τον χρόνο εκτέλεσης σε σύγκριση με τη χειροκίνητη προσέγγιση. Παράλληλα, εξασφαλίζει υψηλό βαθμό επαναληψιμότητας και συμβάλλει στην αξιόπιστη αξιολόγηση κρίσιμων λειτουργιών της εφαρμογής.

13.5 Ανάλυση Εντοπισμένων Defects

Στο πλαίσιο της εκτέλεσης των χειροκίνητων δοκιμών που παρουσιάστηκαν στο Κεφάλαιο 12, εντοπίστηκαν defects τα οποία σχετίζονται τόσο με τη διεπαφή χρήστη όσο και με τη λογική λειτουργίας της εφαρμογής. Η ανάλυση των defects αυτών κρίνεται ιδιαίτερα σημαντική, καθώς συμβάλλει στην κατανόηση των δυνατοτήτων και των περιορισμών των διαφορετικών προσεγγίσεων ελέγχου λογισμικού.

Συγκεκριμένα, κατά τη διαδικασία του χειροκίνητου ελέγχου εντοπίστηκαν δύο βασικά defects.

Το πρώτο defect αφορά τη δυσλειτουργία του εικονιδίου διαγραφής (clear icon) στα πεδία εισαγωγής δεδομένων. Παρά την εμφάνιση του σχετικού εικονιδίου μετά από σφάλμα εισαγωγής, η επιλογή του δεν οδηγεί στη διαγραφή των δεδομένων, γεγονός που

υποδηλώνει ασυνέπεια στη διεπαφή χρήστη και επηρεάζει αρνητικά τη χρηστικότητα της εφαρμογής.

Το δεύτερο defect αφορά τη δυνατότητα ολοκλήρωσης της διαδικασίας αγοράς με κενό καλάθι αγορών. Παρά το γεγονός ότι δεν υπάρχουν προϊόντα προς αγορά, το σύστημα επιτρέπει την εκτέλεση της ενέργειας ολοκλήρωσης της παραγγελίας και εμφανίζει μήνυμα επιτυχούς ολοκλήρωσης. Το συγκεκριμένο defect επηρεάζει τη βασική επιχειρησιακή λογική της εφαρμογής και θεωρείται ιδιαίτερα σημαντικό.

Σύμφωνα με τη διεθνή ορολογία του ελέγχου λογισμικού, τα defects μπορούν να ταξινομηθούν σε διαφορετικές κατηγορίες ανάλογα με το σημείο εμφάνισης και τον αντίκτυπό τους στη λειτουργία του συστήματος (ISTQB, 2023).

Στην παρούσα εργασία, η δυσλειτουργία του clear icon ταξινομείται ως σφάλμα διεπαφής χρήστη (User Interface Defect), καθώς σχετίζεται με τη συμπεριφορά στοιχείων της διεπαφής και επηρεάζει κυρίως τη χρηστικότητα της εφαρμογής. Αντίθετα, η δυνατότητα ολοκλήρωσης αγοράς με κενό καλάθι ταξινομείται ως σφάλμα επιχειρησιακής λογικής (Business Logic Defect), καθώς το σύστημα επιτρέπει την εκτέλεση μιας λειτουργίας που παραβιάζει τη βασική ροή λειτουργίας μιας εφαρμογής ηλεκτρονικού εμπορίου.

Η ταξινόμηση των σφαλμάτων κρίνεται σημαντική στη διαδικασία testing, καθώς επιτρέπει την αξιολόγηση της σοβαρότητας και της επίδρασης κάθε προβλήματος στη λειτουργία του συστήματος.

Από πλευράς σοβαρότητας (severity), τα δύο defects παρουσιάζουν διαφορετικό επίπεδο επίδρασης στη λειτουργία της εφαρμογής. Η αξιολόγηση της σοβαρότητας ενός σφάλματος βασίζεται στον βαθμό επίδρασής του στη λειτουργικότητα και στη συνολική ποιότητα του συστήματος (ISTQB, 2023). Το defect του clear icon χαρακτηρίζεται ως χαμηλής έως μεσαίας σοβαρότητας, καθώς επηρεάζει κυρίως τη χρηστικότητα και την εμπειρία χρήστη, χωρίς να επηρεάζει κρίσιμες λειτουργίες του συστήματος. Αντίθετα, το defect που επιτρέπει την ολοκλήρωση αγοράς με κενό καλάθι θεωρείται υψηλής σοβαρότητας, καθώς επηρεάζει τη βασική επιχειρησιακή λογική της εφαρμογής και παραβιάζει τη φυσιολογική ροή μιας διαδικασίας ηλεκτρονικού εμπορίου. Η ύπαρξη αντίστοιχων defects σε πραγματικό περιβάλλον παραγωγικής λειτουργίας θα μπορούσε να οδηγήσει σε ασυνέπεια δεδομένων ή σε λανθασμένη επεξεργασία παραγγελιών.

Ιδιαίτερο ενδιαφέρον παρουσιάζει το γεγονός ότι τα παραπάνω defects εντοπίστηκαν αποκλειστικά μέσω του manual testing και δεν ανιχνεύθηκαν κατά την εκτέλεση του automated testing μέσω του Selenium WebDriver.

Ο λόγος για τον οποίο τα συγκεκριμένα defects δεν εντοπίστηκαν από τα αυτοματοποιημένα σενάρια έγκειται στο γεγονός ότι η αυτοματοποίηση βασίζεται σε προκαθορισμένα test cases και συγκεκριμένα αναμενόμενα αποτελέσματα. Στην περίπτωση του clear icon, το πρόβλημα σχετίζεται με τη συμπεριφορά της διεπαφής χρήστη και απαιτεί οπτική παρατήρηση και αλληλεπίδραση πέραν των τυπικών σεναρίων ελέγχου. Αντίστοιχα, το defect της ολοκλήρωσης αγοράς με κενό καλάθι δεν συμπεριλήφθηκε ως αρνητικό σενάριο στα αυτοματοποιημένα tests και, κατά συνέπεια, δεν ήταν δυνατό να εντοπιστεί κατά την αυτοματοποιημένη εκτέλεση.

Η παρατήρηση αυτή αναδεικνύει έναν από τους σημαντικότερους περιορισμούς του automated testing, δηλαδή την εξάρτησή του από την πληρότητα και την ποιότητα των test cases που έχουν σχεδιαστεί. Η αποτελεσματικότητα των αυτοματοποιημένων δοκιμών συνδέεται άμεσα με την ποιότητα του σχεδιασμού των test scenarios και των αναμενόμενων αποτελεσμάτων που έχουν καθοριστεί κατά τη φάση σχεδιασμού (Pressman & Maxim, 2020). Σε αντίθεση με το manual testing, όπου ο tester μπορεί να διερευνήσει μη αναμενόμενα σενάρια και να αξιολογήσει δυναμικά τη συμπεριφορά της εφαρμογής, η αυτοματοποίηση εκτελεί αυστηρά προκαθορισμένες ενέργειες χωρίς δυνατότητα προσαρμογής σε απρόβλεπτες καταστάσεις.

Επιπλέον, τα αποτελέσματα του exploratory testing που παρουσιάστηκαν στο Κεφάλαιο 12 επιβεβαιώνουν τη σημασία της ανθρώπινης παρέμβασης στη διαδικασία ελέγχου, καθώς επιτρέπουν την ανίχνευση σφαλμάτων που δεν μπορούν εύκολα να εντοπιστούν μέσω αυτοματοποιημένων διαδικασιών. Ο διερευνητικός έλεγχος θεωρείται ιδιαίτερα αποτελεσματικός στον εντοπισμό μη αναμενόμενων προβλημάτων χρηστικότητας και λειτουργικότητας, τα οποία συχνά δεν καλύπτονται από προκαθορισμένα test scenarios (ISTQB, 2023).

Συνολικά, η ανάλυση των εντοπισμένων σφαλμάτων καταδεικνύει ότι, ενώ η αυτοματοποίηση είναι ιδιαίτερα αποτελεσματική στην επαλήθευση προκαθορισμένων και επαναλαμβανόμενων λειτουργικών απαιτήσεων, το manual testing παραμένει απαραίτητο

για την ανίχνευση ζητημάτων χρηστικότητας, επιχειρησιακής λογικής και μη αναμενόμενων συμπεριφορών του συστήματος.

Η ανάλυση αυτή επιβεβαιώνει ότι οι δύο προσεγγίσεις λειτουργούν συμπληρωματικά και όχι ανταγωνιστικά. Η αυτοματοποίηση προσφέρει ταχύτητα, συνέπεια και επαναληψιμότητα, ενώ το manual testing συμβάλλει στον εντοπισμό προβλημάτων που απαιτούν ανθρώπινη κρίση και διερευνητική προσέγγιση. Ως εκ τούτου, ο συνδυασμός των δύο μεθόδων αποτελεί μια ολοκληρωμένη στρατηγική για τη διασφάλιση της ποιότητας λογισμικού.

13.6 Συγκριτική Ανάλυση και Συζήτηση

Η συγκριτική ανάλυση μεταξύ του manual και του automated testing αποτελεί βασικό στόχο της παρούσας εργασίας, καθώς επιτρέπει την αξιολόγηση των δύο προσεγγίσεων ως προς την αποδοτικότητα, την αξιοπιστία και την καταλληλότητά τους σε διαφορετικά test scenarios.

Με βάση τα αποτελέσματα που παρουσιάστηκαν στα Κεφάλαια 12 και 13, προκύπτει ότι οι δύο μέθοδοι παρουσιάζουν σημαντικές διαφορές ως προς τον τρόπο εκτέλεσης, την ταχύτητα, την επαναληψιμότητα και τα αποτελέσματα που παράγουν.

Το manual testing, όπως εφαρμόστηκε στο Κεφάλαιο 12, βασίζεται στην άμεση αλληλεπίδραση του ελεγκτή με την εφαρμογή, επιτρέποντας την αξιολόγηση της συμπεριφοράς της από την οπτική γωνία του τελικού χρήστη. Η προσέγγιση αυτή αποδείχθηκε ιδιαίτερα αποτελεσματική στον εντοπισμό defects που σχετίζονται με τη χρηστικότητα και τη λογική λειτουργίας του συστήματος, όπως η δυσλειτουργία του εικονιδίου διαγραφής (clear icon) και η δυνατότητα ολοκλήρωσης αγοράς με κενό καλάθι αγορών.

Αντίθετα, το automated testing, το οποίο υλοποιήθηκε μέσω του Selenium WebDriver στο παρόν κεφάλαιο, βασίζεται στην εκτέλεση προκαθορισμένων σεναρίων μέσω προγραμματιστικού κώδικα. Η προσέγγιση αυτή προσφέρει σημαντικά πλεονεκτήματα ως προς την ταχύτητα εκτέλεσης και την επαναληψιμότητα των δοκιμών, καθώς επιτρέπει την εκτέλεση των ίδιων ενεργειών με ακρίβεια και χωρίς ανθρώπινη παρέμβαση.

Ωστόσο, η αυτοματοποίηση παρουσιάζει περιορισμούς ως προς την ικανότητα εντοπισμού μη αναμενόμενων defects, καθώς εξαρτάται σε μεγάλο βαθμό από τα σενάρια που έχουν υλοποιηθεί. Όπως διαπιστώθηκε, τα σφάλματα που εντοπίστηκαν κατά τη χειροκίνητη διαδικασία δεν ανιχνεύθηκαν από τα αυτοματοποιημένα σενάρια, γεγονός που υπογραμμίζει τη σημασία της ανθρώπινης παρέμβασης στη διαδικασία ελέγχου.

Τα αποτελέσματα της παρούσας μελέτης υποδεικνύουν ότι η αποτελεσματικότητα μιας διαδικασίας ελέγχου δεν εξαρτάται αποκλειστικά από τον βαθμό αυτοματοποίησης, αλλά κυρίως από την ποιότητα του σχεδιασμού των test scenarios και την πληρότητα της στρατηγικής ελέγχου που εφαρμόζεται.

Παρότι οι αυτοματοποιημένες δοκιμές παρουσίασαν υψηλή ταχύτητα και σταθερότητα κατά την εκτέλεση, η περιορισμένη κάλυψη διερευνητικών και μη προβλεπόμενων σεναρίων είχε ως αποτέλεσμα τη μη ανίχνευση συγκεκριμένων σφαλμάτων. Αντίθετα, το manual testing επέτρεψε τη δυναμική αξιολόγηση της συμπεριφοράς της εφαρμογής και την ανακάλυψη προβλημάτων που δεν είχαν συμπεριληφθεί στα προκαθορισμένα αυτοματοποιημένα σενάρια.

Η παρατήρηση αυτή αναδεικνύει ότι η αποδοτικότητα μιας διαδικασίας testing δεν θα πρέπει να αξιολογείται μόνο με βάση την ταχύτητα εκτέλεσης, αλλά και με βάση την ικανότητα εντοπισμού κρίσιμων defects και την ποιότητα της συνολικής κάλυψης των δοκιμών.

Για την πληρέστερη κατανόηση των διαφορών μεταξύ των δύο προσεγγίσεων, ο Πίνακας 13.2 παρουσιάζει συνοπτική σύγκριση των βασικών χαρακτηριστικών τους.

Κριτήριο	Manual Testing	Automated Testing
Τρόπος εκτέλεσης	Άμεση αλληλεπίδραση χρήστη	Εκτέλεση μέσω κώδικα
Ταχύτητα εκτέλεσης	Χαμηλή	Υψηλή
Επαναληψιμότητα	Περιορισμένη	Πολύ υψηλή
Ακρίβεια	Εξαρτάται από τον tester	Πολύ υψηλή
Εντοπισμός defects	Ιδιαίτερα αποτελεσματικός σε διερευνητικά σενάρια	Περιορίζεται στα υλοποιημένα σενάρια
Ευελιξία	Υψηλή	Περιορισμένη
Κόστος σε βάθος χρόνου	Υψηλό	Χαμηλότερο
Ανθρώπινη παρέμβαση	Απαραίτητη	Ελάχιστη

Πίνακας 13.2: Σύγκριση Χειροκίνητου και Αυτοματοποιημένου Ελέγχου

Η ανάλυση του πίνακα καταδεικνύει ότι το automated testing υπερτερεί σε περιπτώσεις όπου απαιτείται ταχύτητα, συνέπεια και επαναληψιμότητα, όπως στις δοκιμές παλινδρόμησης (regression testing). Αντίθετα, το manual testing είναι περισσότερο κατάλληλο για διερευνητικές δοκιμές και για την αξιολόγηση της εμπειρίας χρήστη.

Επιπλέον, τα αποτελέσματα της παρούσας εργασίας επιβεβαιώνουν ότι οι δύο προσεγγίσεις δεν λειτουργούν ανταγωνιστικά αλλά συμπληρωματικά. Η χρήση manual testing στα αρχικά στάδια επιτρέπει την ανακάλυψη defects και την καλύτερη κατανόηση της συμπεριφοράς της εφαρμογής, ενώ η αυτοματοποίηση μπορεί στη συνέχεια να αξιοποιηθεί για την επαναλαμβανόμενη εκτέλεση των ίδιων δοκιμών με σημαντικά μεγαλύτερη αποδοτικότητα.

Συνεπώς, η βέλτιστη στρατηγική ελέγχου προκύπτει από τον συνδυασμό των δύο μεθόδων, αξιοποιώντας τα πλεονεκτήματα της καθεμίας. Η προσέγγιση αυτή συνάδει με τις κατευθυντήριες οδηγίες του International Software Testing Qualifications Board (ISTQB), σύμφωνα με τις οποίες η αποτελεσματική διασφάλιση ποιότητας απαιτεί τη συνδυαστική χρήση διαφορετικών τεχνικών και επιπέδων ελέγχου λογισμικού (ISTQB, 2023).

13.6.1 Ο Ρόλος του Automated Testing σε Σύγχρονα Περιβάλλοντα DevOps

Στα σύγχρονα περιβάλλοντα ανάπτυξης λογισμικού, ο αυτοματοποιημένος έλεγχος αποτελεί βασικό στοιχείο των πρακτικών DevOps και των διαδικασιών Continuous Integration / Continuous Delivery (CI/CD). Η αυτοματοποίηση των δοκιμών επιτρέπει την εκτέλεση ελέγχων σε κάθε νέα αλλαγή κώδικα, συμβάλλοντας στον έγκαιρο εντοπισμό σφαλμάτων και στη διατήρηση της σταθερότητας του συστήματος (Humble & Farley, 2011).

Στο πλαίσιο των διαδικασιών CI/CD, οι αυτοματοποιημένες δοκιμές εκτελούνται συχνά μετά από κάθε νέα έκδοση ή τροποποίηση της εφαρμογής, επιτρέποντας την άμεση επαλήθευση της λειτουργικότητας του συστήματος. Η προσέγγιση αυτή είναι ιδιαίτερα σημαντική στις δοκιμές παλινδρόμησης, όπου απαιτείται η επαναλαμβανόμενη εκτέλεση μεγάλου αριθμού test scenarios (Humble & Farley, 2011).

Η δυνατότητα επαναληψιμότητας και η υψηλή ταχύτητα εκτέλεσης καθιστούν την αυτοματοποίηση ιδιαίτερα κατάλληλη για ευέλικτα (Agile) και DevOps περιβάλλοντα ανάπτυξης λογισμικού. Παράλληλα, η ενσωμάτωση αυτοματοποιημένων δοκιμών σε

διαδικασίες συνεχούς ολοκλήρωσης συμβάλλει στη βελτίωση της ποιότητας λογισμικού και στη μείωση του κόστους εντοπισμού σφαλμάτων σε μεταγενέστερα στάδια ανάπτυξης (Pressman & Maxim, 2020).

Ωστόσο, ακόμη και στα σύγχρονα DevOps περιβάλλοντα, το manual testing εξακολουθεί να διαδραματίζει σημαντικό ρόλο, ιδιαίτερα σε διερευνητικές δοκιμές, στην αξιολόγηση χρηστικότητας και σε περιπτώσεις όπου απαιτείται ανθρώπινη κρίση και αξιολόγηση της εμπειρίας χρήστη.

13.7 Συμπεράσματα

Η υλοποίηση των αυτοματοποιημένων δοκιμών στο πλαίσιο της παρούσας εργασίας επέτρεψε την επιτυχή επαλήθευση βασικών λειτουργικών σεναρίων της εφαρμογής SauceDemo μέσω του Selenium WebDriver. Η εκτέλεση των δοκιμών κατέδειξε τα βασικά πλεονεκτήματα του automated testing, όπως η αυξημένη ταχύτητα εκτέλεσης, η υψηλή επαναληψιμότητα και η δυνατότητα αξιόπιστης εκτέλεσης προκαθορισμένων test scenarios χωρίς ανθρώπινη παρέμβαση (Pressman & Maxim, 2020).

Παράλληλα, η ανάλυση των αποτελεσμάτων ανέδειξε ότι η αποτελεσματικότητα των αυτοματοποιημένων δοκιμών εξαρτάται σε σημαντικό βαθμό από την ποιότητα του σχεδιασμού των test scenarios, την πληρότητα της στρατηγικής testing και την ορθή υλοποίηση των αυτοματοποιημένων scripts. Η παρατήρηση αυτή επιβεβαιώνεται και από τα ευρήματα της παρούσας μελέτης, όπου ορισμένα defects εντοπίστηκαν αποκλειστικά μέσω του manual και exploratory testing.

Τα αποτελέσματα καταδεικνύουν ότι το automated και το manual testing δεν αποτελούν ανταγωνιστικές αλλά συμπληρωματικές προσεγγίσεις διασφάλισης ποιότητας λογισμικού. Η αυτοματοποίηση είναι ιδιαίτερα αποτελεσματική σε επαναλαμβανόμενες διαδικασίες και δοκιμές παλινδρόμησης, ενώ το manual testing παραμένει απαραίτητος για την αξιολόγηση της χρηστικότητας, της εμπειρίας χρήστη και μη αναμενόμενων συμπεριφορών του συστήματος (ISTQB, 2023).

Συνολικά, η παρούσα μελέτη επιβεβαιώνει ότι η βέλτιστη στρατηγική ελέγχου λογισμικού προκύπτει από τον συνδυασμό χειροκίνητων και αυτοματοποιημένων δοκιμών,

αξιοποιώντας τα πλεονεκτήματα κάθε προσέγγισης στο πλαίσιο μιας ολοκληρωμένης διαδικασίας διασφάλισης ποιότητας.

13.8 Συνολική Αξιολόγηση των Δύο Προσεγγίσεων

Η συγκριτική αξιολόγηση του manual και του automated testing στο πλαίσιο της παρούσας εργασίας ανέδειξε ότι οι δύο προσεγγίσεις παρουσιάζουν διαφορετικά πλεονεκτήματα και περιορισμούς στη διαδικασία διασφάλισης ποιότητας λογισμικού.

Το manual testing αποδείχθηκε ιδιαίτερα αποτελεσματικό στην ανίχνευση μη αναμενόμενων συμπεριφορών της εφαρμογής και προβλημάτων που σχετίζονται με τη χρηστικότητα και τη συνολική εμπειρία χρήστη. Η διερευνητική προσέγγιση (exploratory testing) επέτρεψε τη δυναμική αξιολόγηση της εφαρμογής και συνέβαλε στον εντοπισμό defects τα οποία δεν είχαν προβλεφθεί κατά τον αρχικό σχεδιασμό των test scenarios (ISTQB, 2023).

Αντίστοιχα, το automated testing παρουσίασε σημαντικά πλεονεκτήματα ως προς την ταχύτητα εκτέλεσης, τη σταθερότητα και τη δυνατότητα επαναληψιμότητας των δοκιμών. Η αυτοματοποίηση αποδείχθηκε ιδιαίτερα αποτελεσματική σε επαναλαμβανόμενα και σαφώς καθορισμένα test scenarios, όπως αυτά που συναντώνται συχνά στις δοκιμές παλινδρόμησης (regression testing) και στις διαδικασίες συνεχούς ολοκλήρωσης λογισμικού (Pressman & Maxim, 2020).

Τα αποτελέσματα της μελέτης υποδεικνύουν ότι καμία από τις δύο προσεγγίσεις δεν μπορεί να θεωρηθεί απόλυτα επαρκής όταν χρησιμοποιείται μεμονωμένα. Το manual testing παρέχει ευελιξία, προσαρμοστικότητα και ανθρώπινη κρίση, ενώ το automated testing προσφέρει ταχύτητα, συνέπεια και δυνατότητα επαναλαμβανόμενης εκτέλεσης μεγάλου αριθμού δοκιμών.

Συνεπώς, η συνδυαστική αξιοποίηση manual και automated testing αποτελεί την πλέον ολοκληρωμένη στρατηγική διασφάλισης ποιότητας λογισμικού, καθώς επιτρέπει την αξιοποίηση των πλεονεκτημάτων κάθε προσέγγισης ανάλογα με τις απαιτήσεις του συστήματος και το είδος των δοκιμών που εκτελούνται. Η προσέγγιση αυτή συνάδει με τις σύγχρονες πρακτικές του ελέγχου λογισμικού, σύμφωνα με τις οποίες η επίτευξη υψηλού

επιπέδου ποιότητας προϋποθέτει τη συνδυαστική εφαρμογή διαφορετικών τεχνικών και μεθόδων ελέγχου (ISTQB, 2023).

13.9 Ποσοτική Αξιολόγηση Αποτελεσμάτων

Η αξιολόγηση των δύο προσεγγίσεων ελέγχου στην παρούσα εργασία δεν περιορίστηκε μόνο σε ποιοτικά χαρακτηριστικά, αλλά περιέλαβε και την ανάλυση ποσοτικών δεδομένων που προέκυψαν από την εκτέλεση των δοκιμών. Η χρήση μετρικών επιτρέπει την αντικειμενική αποτίμηση της αποτελεσματικότητας της διαδικασίας testing και τη συνολική αξιολόγηση της ποιότητας της εφαρμογής.

Οι συνοπτικές μετρικές των δοκιμών παρουσιάστηκαν στον Πίνακα 12.2 του προηγούμενου κεφαλαίου. Από το σύνολο των 35 δοκιμών που εκτελέστηκαν, οι 32 ολοκληρώθηκαν επιτυχώς, ενώ τρεις δοκιμές απέτυχαν. Επιπλέον, κατά τη διαδικασία του manual testing εντοπίστηκαν συνολικά δύο defects, τα οποία αφορούσαν τη διεπαφή χρήστη και τη λογική λειτουργίας της εφαρμογής.

Το ποσοστό επιτυχίας των δοκιμών ανήλθε σε 91,4%, γεγονός που υποδηλώνει ότι η εφαρμογή παρουσιάζει σε γενικές γραμμές σταθερή και αξιόπιστη λειτουργία στις βασικές λειτουργίες που αξιολογήθηκαν. Παράλληλα, η ύπαρξη αποτυχημένων δοκιμών και εντοπισμένων defects αναδεικνύει τη σημασία της διαδικασίας testing για την αποκάλυψη προβλημάτων που ενδέχεται να επηρεάσουν τη λειτουργικότητα και την εμπειρία χρήστη.

Ιδιαίτερο ενδιαφέρον παρουσιάζει η κατανομή των defects ανά μέθοδο ελέγχου, η οποία παρουσιάζεται στον Πίνακα 13.3.

Μέθοδος Ελέγχου	Εντοπισμένα Defects
Manual Testing	2
Automated Testing	0

Πίνακας 13.3: Εντοπισμός Defects ανά Μέθοδο Ελέγχου

Όπως φαίνεται στον Πίνακα 13.3, τα defects που καταγράφηκαν στην παρούσα εργασία εντοπίστηκαν αποκλειστικά μέσω της διαδικασίας manual testing. Αντίθετα, οι αυτοματοποιημένες δοκιμές δεν εντόπισαν νέα defects, καθώς περιορίστηκαν στην επαλήθευση των προκαθορισμένων λειτουργικών απαιτήσεων που είχαν υλοποιηθεί στα αντίστοιχα test scenarios.

Η αυτοματοποίηση των δοκιμών είναι γενικά αποδεκτό ότι συμβάλλει στη μείωση του χρόνου εκτέλεσης, καθιστώντας την ιδιαίτερα αποδοτική σε περιπτώσεις όπου απαιτείται επαναλαμβανόμενη εκτέλεση μεγάλου αριθμού test cases, όπως στις δοκιμές παλινδρόμησης (regression testing) (Pressman & Maxim, 2020).

Επιπλέον, η αυτοματοποίηση παρουσίασε υψηλή σταθερότητα κατά την εκτέλεση των σεναρίων, καθώς κάθε επανάληψη των δοκιμών παρήγαγε τα ίδια αποτελέσματα χωρίς αποκλίσεις. Αντίθετα, κατά τη χειροκίνητη εκτέλεση παρατηρήθηκαν μικρές διαφοροποιήσεις που σχετίζονται με τον ανθρώπινο παράγοντα, όπως η ταχύτητα εκτέλεσης των ενεργειών και η εμπειρία του tester.

Ένα ακόμη σημαντικό στοιχείο της αξιολόγησης είναι η δυνατότητα επαναληψιμότητας. Τα αυτοματοποιημένα tests μπορούν να εκτελεστούν επανειλημμένα με ελάχιστη επιπλέον προσπάθεια, ενώ το manual testing απαιτεί εκ νέου εκτέλεση από τον tester για κάθε επανάληψη.

Τα παραπάνω ευρήματα επιβεβαιώνουν ότι το automated testing υπερτερεί ως προς την αποδοτικότητα εκτέλεσης, τη σταθερότητα των αποτελεσμάτων και τη δυνατότητα επαναληψιμότητας. Ωστόσο, η αποτελεσματικότητα μιας διαδικασίας testing δεν μπορεί να αξιολογηθεί αποκλειστικά μέσω ποσοτικών δεικτών, αλλά θα πρέπει να λαμβάνει υπόψη και την ικανότητα εντοπισμού κρίσιμων defects, τη λειτουργική κάλυψη (functional coverage) και τη συνολική ποιότητα των test scenarios (ISTQB, 2023).

Συνολικά, τα αποτελέσματα της ποσοτικής αξιολόγησης επιβεβαιώνουν ότι η εφαρμογή παρουσιάζει υψηλό βαθμό λειτουργικής ορθότητας, ενώ παράλληλα αναδεικνύουν τη σημασία της συνδυαστικής χρήσης manual και automated testing για την πληρέστερη αξιολόγηση της ποιότητας λογισμικού.

13.10 Περιορισμοί της Μελέτης

Παρά τα σημαντικά ευρήματα που προέκυψαν από την παρούσα μελέτη, εντοπίζονται ορισμένοι περιορισμοί οι οποίοι θα πρέπει να ληφθούν υπόψη κατά την ερμηνεία των αποτελεσμάτων.

Αρχικά, η ανάλυση βασίστηκε σε συγκεκριμένο αριθμό test cases, τα οποία επικεντρώθηκαν κυρίως στις βασικές λειτουργίες της εφαρμογής, όπως η σύνδεση χρήστη,

η διαχείριση καλαθιού και η διαδικασία ολοκλήρωσης αγοράς. Αν και τα σενάρια αυτά καλύπτουν κρίσιμες λειτουργικές ενότητες, δεν περιλαμβάνουν πιο σύνθετες περιπτώσεις, όπως edge cases ή σενάρια υψηλής πολυπλοκότητας, τα οποία ενδέχεται να επηρεάσουν διαφορετικά τα αποτελέσματα.

Επιπλέον, η μελέτη πραγματοποιήθηκε σε μία μόνο εφαρμογή, τη SauceDemo, γεγονός που περιορίζει τη δυνατότητα γενίκευσης των συμπερασμάτων σε άλλες εφαρμογές ή διαφορετικά περιβάλλοντα ανάπτυξης λογισμικού. Διαφορετικά συστήματα ενδέχεται να παρουσιάζουν διαφορετική συμπεριφορά ως προς την αποτελεσματικότητα των μεθόδων ελέγχου και των εργαλείων αυτοματοποίησης.

Όσον αφορά την αυτοματοποίηση, η υλοποίηση επικεντρώθηκε αποκλειστικά σε λειτουργικές δοκιμές (functional testing) μέσω του Selenium WebDriver και δεν επεκτάθηκε σε άλλες κατηγορίες δοκιμών, όπως έλεγχος απόδοσης (performance testing), ασφάλειας (security testing) ή φόρτου (load testing). Ως εκ τούτου, τα συμπεράσματα της παρούσας μελέτης αφορούν κυρίως τον λειτουργικό έλεγχο εφαρμογών ιστού και δεν μπορούν να γενικευθούν σε όλες τις κατηγορίες ελέγχου λογισμικού.

Επιπρόσθετα, η αυτοματοποίηση υλοποιήθηκε με απλοποιημένη προσέγγιση, χωρίς τη χρήση προηγμένων τεχνικών όπως testing frameworks (π.χ. JUnit ή TestNG) ή αρχιτεκτονικών προτύπων όπως το Page Object Model (POM). Η επιλογή αυτή πραγματοποιήθηκε με στόχο την κατανόηση της βασικής λειτουργίας του Selenium WebDriver και τη συγκριτική αξιολόγηση του automated testing σε σχέση με το manual testing. Ωστόσο, περιορίζει την επεκτασιμότητα και τη συντηρησιμότητα του κώδικα των δοκιμών, ιδιαίτερα σε περιβάλλοντα όπου απαιτείται η διαχείριση μεγάλου αριθμού test cases (Selenium, 2024).

Επιπλέον, η χρήση στατικών μηχανισμών αναμονής (Thread.sleep) αντί δυναμικών μηχανισμών συγχρονισμού αποτελεί έναν ακόμη περιορισμό της υλοποίησης, καθώς σε πιο σύνθετα περιβάλλοντα testing ενδέχεται να επηρεαστεί η σταθερότητα και η αποδοτικότητα των αυτοματοποιημένων δοκιμών. Στη σύγχρονη πρακτική του automated testing προτιμώνται μηχανισμοί δυναμικής αναμονής, όπως οι explicit waits, οι οποίοι συμβάλλουν στη βελτίωση της αξιοπιστίας και της σταθερότητας των αυτοματοποιημένων σεναρίων (Selenium, 2024).

Τέλος, η αξιολόγηση του manual testing επηρεάζεται από τον ανθρώπινο παράγοντα, καθώς η ακρίβεια και η αποτελεσματικότητα των δοκιμών εξαρτώνται από την εμπειρία, την προσοχή και την κρίση του tester. Αυτό ενδέχεται να οδηγήσει σε διαφοροποιήσεις στα αποτελέσματα ή στην παράλειψη ορισμένων defects, ιδιαίτερα σε περιπτώσεις όπου δεν ακολουθείται αυστηρά καθορισμένη διαδικασία ελέγχου (ISTQB, 2023).

Συνολικά, οι παραπάνω περιορισμοί δεν αναιρούν την εγκυρότητα των αποτελεσμάτων της παρούσας εργασίας, αλλά αναδεικνύουν τα όρια της μελέτης και υποδεικνύουν κατευθύνσεις για μελλοντική έρευνα. Η επέκταση της αξιολόγησης σε περισσότερες εφαρμογές, η χρήση πιο σύνθετων test scenarios και η αξιοποίηση προηγμένων τεχνικών αυτοματοποίησης θα μπορούσαν να προσφέρουν πληρέστερη εικόνα σχετικά με τη συγκριτική αποτελεσματικότητα του manual και του automated testing.

13.11 Τελική Αξιολόγηση

Η παρούσα εργασία είχε ως βασικό στόχο τη συγκριτική αξιολόγηση του manual και του automated testing σε περιβάλλον εφαρμογής ιστού, με τη χρήση της πλατφόρμας SauceDemo ως μελέτη περίπτωσης. Μέσω της συστηματικής εφαρμογής και των δύο προσεγγίσεων κατέστη δυνατή η εις βάθος ανάλυση των χαρακτηριστικών, των πλεονεκτημάτων και των περιορισμών κάθε μεθόδου ελέγχου.

Τα αποτελέσματα της μελέτης ανέδειξαν ότι το manual testing αποτελεί απαραίτητο εργαλείο για την αξιολόγηση της εμπειρίας χρήστη και την ανίχνευση μη αναμενόμενων σφαλμάτων, ιδιαίτερα σε περιπτώσεις όπου απαιτείται ευελιξία, δημιουργική σκέψη και ανθρώπινη κρίση. Η δυνατότητα διερευνητικής προσέγγισης (exploratory testing) συνέβαλε καθοριστικά στον εντοπισμό defects που δεν είχαν προβλεφθεί κατά τον σχεδιασμό των test scenarios (ISTQB, 2023).

Αντίστοιχα, το automated testing, το οποίο υλοποιήθηκε μέσω του Selenium WebDriver, απέδειξε την αξία του σε περιπτώσεις επαναλαμβανόμενων και τυποποιημένων δοκιμών. Η αυξημένη ταχύτητα εκτέλεσης, η συνέπεια των αποτελεσμάτων και η δυνατότητα επαναληψιμότητας καθιστούν την αυτοματοποίηση αναπόσπαστο μέρος της σύγχρονης διαδικασίας ανάπτυξης λογισμικού και των πρακτικών συνεχούς ολοκλήρωσης και παράδοσης λογισμικού (Pressman & Maxim, 2020).

Η συγκριτική ανάλυση που πραγματοποιήθηκε επιβεβαιώνει ότι καμία από τις δύο προσεγγίσεις δεν είναι επαρκής από μόνη της για την πλήρη αξιολόγηση μιας εφαρμογής. Το manual testing, αν και ιδιαίτερα ευέλικτο, είναι περισσότερο χρονοβόρο και επηρεάζεται από τον ανθρώπινο παράγοντα, ενώ το automated testing, παρότι ιδιαίτερα αποδοτικό, περιορίζεται από τα προκαθορισμένα σενάρια που έχουν σχεδιαστεί και υλοποιηθεί.

Συνεπώς, η βέλτιστη στρατηγική ελέγχου λογισμικού βασίζεται στον συνδυασμό των δύο μεθόδων, αξιοποιώντας τα πλεονεκτήματα κάθε προσέγγισης. Η χρήση manual testing στα αρχικά στάδια επιτρέπει την καλύτερη κατανόηση της εφαρμογής και τον εντοπισμό κρίσιμων defects, ενώ η αυτοματοποίηση μπορεί στη συνέχεια να αξιοποιηθεί για την επαναλαμβανόμενη εκτέλεση των δοκιμών με υψηλή αποδοτικότητα και συνέπεια.

Τέλος, η παρούσα εργασία συμβάλλει στην κατανόηση της πρακτικής εφαρμογής των τεχνικών ελέγχου λογισμικού και αναδεικνύει τη σημασία της κατάλληλης μεθοδολογικής προσέγγισης στη διαδικασία διασφάλισης ποιότητας. Τα ευρήματα της μελέτης μπορούν να αποτελέσουν βάση για περαιτέρω έρευνα και εφαρμογή σε πιο σύνθετα περιβάλλοντα ανάπτυξης, ενισχύοντας τη σημασία της συνδυαστικής χρήσης manual και automated testing για την επίτευξη υψηλού επιπέδου ποιότητας λογισμικού.

13.12 Μελλοντική Εργασία

Η παρούσα εργασία επικεντρώθηκε στη συγκριτική αξιολόγηση του manual και του automated ελέγχου λογισμικού μέσω της εφαρμογής SauceDemo και της χρήσης του Selenium WebDriver. Παρότι τα αποτελέσματα που προέκυψαν ήταν ιδιαίτερα χρήσιμα για την κατανόηση των δύο προσεγγίσεων, υπάρχουν αρκετές δυνατότητες επέκτασης της μελέτης στο μέλλον.

Αρχικά, η υλοποίηση των αυτοματοποιημένων δοκιμών θα μπορούσε να βελτιωθεί μέσω της χρήσης αρχιτεκτονικών προτύπων όπως το Page Object Model (POM), το οποίο συμβάλλει στη βελτίωση της συντηρησιμότητας και της επεκτασιμότητας του κώδικα των δοκιμών. Επιπλέον, η αξιοποίηση testing frameworks, όπως τα JUnit και TestNG, θα επέτρεπε την καλύτερη οργάνωση των test cases, την αυτοματοποιημένη παραγωγή αναφορών και την ευκολότερη διαχείριση μεγαλύτερων συνόλων δοκιμών.

Παράλληλα, μελλοντική έρευνα θα μπορούσε να επεκτείνει την αξιολόγηση σε περισσότερες εφαρμογές και διαφορετικά περιβάλλοντα ανάπτυξης λογισμικού, προκειμένου να εξεταστεί η γενικευσιμότητα των συμπερασμάτων της παρούσας μελέτης. Η χρήση μεγαλύτερου αριθμού σεναρίων δοκιμών, συμπεριλαμβανομένων edge cases και πιο σύνθετων επιχειρησιακών ροών, θα μπορούσε να προσφέρει πληρέστερη εικόνα σχετικά με την αποτελεσματικότητα των δύο προσεγγίσεων.

Επιπλέον, ενδιαφέρον θα παρουσίαζε η επέκταση της αυτοματοποίησης σε άλλες κατηγορίες ελέγχου, όπως οι δοκιμές απόδοσης (performance testing), ασφάλειας (security testing) και φόρτου (load testing), ώστε να επιτευχθεί πιο ολοκληρωμένη αξιολόγηση της ποιότητας λογισμικού.

Τέλος, η ενσωμάτωση των αυτοματοποιημένων δοκιμών σε περιβάλλοντα συνεχούς ολοκλήρωσης και συνεχούς παράδοσης λογισμικού (CI/CD) θα μπορούσε να αποτελέσει αντικείμενο περαιτέρω μελέτης, καθώς αποτελεί μία από τις βασικές πρακτικές των σύγχρονων διαδικασιών ανάπτυξης λογισμικού (Pressman & Maxim, 2020).

Μία ακόμη ενδιαφέρουσα κατεύθυνση για μελλοντική έρευνα αφορά τη χρήση εργαλείων Τεχνητής Νοημοσύνης για την υποστήριξη της ανάπτυξης αυτοματοποιημένων δοκιμών. Τα σύγχρονα συστήματα παραγωγικής τεχνητής νοημοσύνης μπορούν να συνδράμουν στη δημιουργία test cases, στη συγγραφή automation scripts και στην παραγωγή τεκμηρίωσης. Μελλοντικές μελέτες θα μπορούσαν να συγκρίνουν την ποιότητα και την αποτελεσματικότητα του παραγόμενου κώδικα από διαφορετικά μοντέλα τεχνητής νοημοσύνης, αξιολογώντας παράγοντες όπως η ορθότητα, η συντηρησιμότητα και η πληρότητα των αυτοματοποιημένων δοκιμών.

Οι παραπάνω κατευθύνσεις μπορούν να συμβάλουν στην περαιτέρω διερεύνηση της αποτελεσματικότητας του manual testing και του automated testing, καθώς και στην αξιοποίηση νέων τεχνολογιών που αναμένεται να διαδραματίσουν σημαντικό ρόλο στο μέλλον της διασφάλισης ποιότητας λογισμικού.

Βιβλιογραφία

Ακολουθούν οι βιβλιογραφικές αναφορές (πηγές) της Εργασίας.

Ammann, P., & Offutt, J. (2016). *Introduction to software testing* (2nd ed.). Cambridge University Press.

Beck, K., Beedle, M., van Bennekum, A., et al. (2001). *Manifesto for Agile Software Development*.

Beizer, B. (1990). *Software Testing Techniques* (2nd ed.). Van Nostrand Reinhold.

Black, R. (2002). *Managing the Testing Process* (2nd ed.). Wiley.

Boehm, B. W. (1988). *A Spiral Model of Software Development and Enhancement*. Computer, 21(5), 61–72.

Crispin, L., & Gregory, J. (2014). *More agile testing: Learning journeys for the whole team*. Addison-Wesley.

Fagan, M. E. (1976). *Design and Code Inspections to Reduce Errors in Program Development*. IBM Systems Journal.

Fewster, M., & Graham, D. (1999). *Software test automation: Effective use of test execution tools*. Addison-Wesley.

Forsgren, N., Humble, J., & Kim, G. (2018). *Accelerate: The Science of Lean Software and DevOps*. IT Revolution Press.

Fowler, M. (2006). *Continuous Integration*. Martin Fowler.

Graham, D., van Veenendaal, E., Evans, I., & Black, R. (2012). *Foundations of software testing: ISTQB certification* (3rd ed.). Cengage Learning.

Humble, J., & Farley, D. (2011). *Continuous delivery: Reliable software releases through build, test, and deployment automation*. Addison-Wesley.

IEEE. (1990). *IEEE Standard Glossary of Software Engineering Terminology (IEEE Std 610.12-1990)*. IEEE.

IEEE. (2008). *IEEE Standard for Software and System Test Documentation (IEEE Std 829-2008)*. IEEE.

- IEEE. (2008). *IEEE Standard for Software Reviews and Audits (IEEE Std 1028-2008)*. IEEE.
- IEEE. (2009). *IEEE Standard Classification for Software Anomalies (IEEE Std 1044-2009)*. IEEE.
- IEEE. (2014). *IEEE Standard for Software Quality Assurance Processes (IEEE Std 730-2014)*. IEEE.
- IEEE. (2016). *IEEE Standard for System, Software, and Hardware Verification and Validation (IEEE Std 1012-2016)*. IEEE.
- International Software Testing Qualifications Board. (2023). *Certified tester foundation level syllabus version 4.0*. <https://istqb.org>
- ISO/IEC. (2011). *ISO/IEC 25010:2011 Systems and software engineering—Systems and software quality requirements and evaluation (SQuaRE)—System and software quality models*. ISO.
- ISO/IEC. (2016). *ISO/IEC 25022: Measurement of Quality in Use*. ISO.
- ISO/IEC. (2016). *ISO/IEC 25023: Systems and Software Engineering—Systems and Software Quality Requirements and Evaluation (SQuaRE)—Measurement of System and Software Product Quality*. ISO.
- ISO/IEC/IEEE. (2013). *ISO/IEC/IEEE 29119-1:2013 Software and Systems Engineering—Software Testing—Part 1: Concepts and Definitions*. ISO.
- ISO/IEC/IEEE. (2021). *ISO/IEC/IEEE 29119-2: Software Testing—Part 2: Test Processes*. ISO.
- ISO/IEC/IEEE. (2022). *ISO/IEC/IEEE 29119-1: Software and Systems Engineering—Software Testing—Part 1: General Concepts*. ISO.
- Kaner, C., Bach, J., & Pettichord, B. (2002). *Lessons learned in software testing: A context-driven approach*. Wiley.
- Kitchenham, B., & Pfleeger, S. L. (1996). *Software Quality: The Elusive Target*. IEEE Software, 13(1), 12–21.
- Laprie, J. C. (1992). *Dependability: Basic Concepts and Terminology*. Springer.

- Meszaros, G. (2007). *xUnit test patterns: Refactoring test code*. Addison-Wesley.
- Myers, G. J., Sandler, C., & Badgett, T. (2011). *The art of software testing* (3rd ed.). Wiley.
- Nielsen, J. (1994). *Usability Engineering*. Morgan Kaufmann.
- OWASP Foundation. (2023). *OWASP Top 10: The ten most critical web application security risks*. <https://owasp.org/www-project-top-ten/>
- Pressman, R. S., & Maxim, B. R. (2020). *Software engineering: A practitioner's approach* (9th ed.). McGraw-Hill Education.
- Royce, W. W. (1970). *Managing the Development of Large Software Systems*. Proceedings of IEEE WESCON.
- Sauce Labs. (n.d.). *Sauce Demo Testing Platform*. Retrieved from saucedemo.com
- Selenium. (2024). *Selenium documentation*. <https://www.selenium.dev/documentation/>
- Sommerville, I. (2016). *Software engineering* (10th ed.). Pearson.
- World Wide Web Consortium. (2023). *Web Content Accessibility Guidelines (WCAG) 2.2*. <https://www.w3.org/TR/WCAG22/>

Παράρτημα Α: Πλήρης Κώδικας Αυτοματοποιημένων Δοκιμών

Στο παρόν παράρτημα παρατίθενται ενδεικτικά οι πλήρεις υλοποιήσεις των βασικών αυτοματοποιημένων σεναρίων δοκιμών που χρησιμοποιήθηκαν στο πλαίσιο της παρούσας εργασίας. Ο κώδικας αναπτύχθηκε σε γλώσσα Java με χρήση του Selenium WebDriver.

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import io.github.bonigarcia.wdm.WebDriverManager;

public class Main {
    public static void main(String[] args) throws InterruptedException {
        // Setup driver
        WebDriverManager.chromedriver().setup();
        WebDriver driver = new ChromeDriver();
        // Open site
        driver.get("https://www.saucedemo.com/");
        driver.manage().window().maximize();
        // WAIT για screenshot
        Thread.sleep(2000);
        // Login (automation)
        driver.findElement(By.id("user-name")).sendKeys("standard_user");
        driver.findElement(By.id("password")).sendKeys("secret_sauce");
        Thread.sleep(2000); // για να πάρεις screenshot (Εικόνα 5.1)
        driver.findElement(By.id("login-button")).click();
        Thread.sleep(2000); // για screenshot result (Εικόνα 5.2)
        // Verification
        if (driver.getCurrentUrl().contains("inventory")) {
            System.out.println("TEST PASSED");
        } else {
            System.out.println("TEST FAILED");
        }
        Thread.sleep(2000); // για screenshot console
        Thread.sleep(10000); // 10 δευτερόλεπτα
        driver.quit();
    }
}
```

Πλήρης Κώδικας 13.1 – Υλοποίηση αυτοματοποιημένης διαδικασίας σύνδεσης

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import io.github.bonigarcia.wdm.WebDriverManager;

public class Main {
    public static void main(String[] args) throws InterruptedException {
        WebDriverManager.chromedriver().setup();
        WebDriver driver = new ChromeDriver();
        driver.get("https://www.saucedemo.com/");
        driver.manage().window().maximize();
        Thread.sleep(2000);
        // Λάθος password
        driver.findElement(By.id("user-name")).sendKeys("standard_user");
        driver.findElement(By.id("password")).sendKeys("wrong_password");
        Thread.sleep(2000); // screenshot 5.4
        driver.findElement(By.id("login-button")).click();
        Thread.sleep(2000); // screenshot 5.5
        // Πάρε το error message
        String errorMessage = driver.findElement(By.cssSelector("h3[data-
test='error']")).getText();
        // Έλεγχος
        if (errorMessage.contains("Username and password do not match")) {
            System.out.println("TEST PASSED");
        } else {
            System.out.println("TEST FAILED");
        }
        Thread.sleep(3000); // screenshot 5.6
        driver.quit();
    }
}
```

Πλήρης Κώδικας 13.2 – Υλοποίηση αυτοματοποιημένης αποτυχημένης σύνδεσης

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.chrome.ChromeDriver;
import io.github.bonigarcia.wdm.WebDriverManager;
import org.openqa.selenium.chrome.ChromeOptions;
public class Main {
    public static void main(String[] args) throws InterruptedException {
        WebDriverManager.chromedriver().setup();
        // Incognito για να μην εμφανίζεται popup
        ChromeOptions options = new ChromeOptions();
        options.addArguments("--incognito");
        WebDriver driver = new ChromeDriver(options);
        driver.get("https://www.saucedemo.com/");
        driver.manage().window().maximize();
        Thread.sleep(2000);
        // LOGIN
        driver.findElement(By.id("user-name")).sendKeys("standard_user");
        driver.findElement(By.id("password")).sendKeys("secret_sauce");
        Thread.sleep(2000);
        driver.findElement(By.id("login-button")).click();
        Thread.sleep(4000); // περιμένει να φορτώσει
        // ADD TO CART
        driver.findElement(By.id("add-to-cart-sauce-labs-backpack")).click();
        Thread.sleep(5000); // για screenshot
        // ΠΑΡΕ CART VALUE (χωρίς crash)
        String cartValue = "";
        try {
            cartValue =
driver.findElement(By.className("shopping_cart_badge")).getText();
        } catch (Exception e) {
            System.out.println("Cart not found");
        }
        Thread.sleep(2000);
        // CHECK
        if (cartValue.equals("1")) {
            System.out.println("TEST PASSED");
        } else {
            System.out.println("TEST FAILED");
        }
    }
}
```

```
    }  
    Thread.sleep(4000);  
    driver.quit();  
}  
}
```

Πλήρης Κώδικας 13.3 – Υλοποίηση αυτοματοποιημένης προσθήκης προϊόντος στο καλάθι

```
import org.openqa.selenium.By;  
import org.openqa.selenium.WebDriver;  
import org.openqa.selenium.chrome.ChromeDriver;  
import io.github.bonigarcia.wdm.WebDriverManager;  
import org.openqa.selenium.chrome.ChromeOptions;  
public class Main {  
    public static void main(String[] args) throws InterruptedException {  
        // Setup driver  
        WebDriverManager.chromedriver().setup();  
        // Incognito mode για αποφυγή popups  
        ChromeOptions options = new ChromeOptions();  
        options.addArguments("--incognito");  
        WebDriver driver = new ChromeDriver(options);  
        // Άνοιγμα εφαρμογής  
        driver.get("https://www.saucedemo.com/");  
        driver.manage().window().maximize();  
        Thread.sleep(2000);  
        // LOGIN  
        driver.findElement(By.id("user-name")).sendKeys("standard_user");  
        driver.findElement(By.id("password")).sendKeys("secret_sauce");  
        Thread.sleep(2000);  
        driver.findElement(By.id("login-button")).click();  
        Thread.sleep(4000); // φόρτωση inventory
```

```
// ADD PRODUCT (προϋπόθεση για remove)

driver.findElement(By.id("add-to-cart-sauce-labs-backpack")).click();

Thread.sleep(3000);

// REMOVE PRODUCT

driver.findElement(By.id("remove-sauce-labs-backpack")).click();

Thread.sleep(3000); // για screenshot

// ΕΛΕΓΧΟΣ αν το cart badge ΔΕΝ υπάρχει

boolean isCartEmpty;

try {

    driver.findElement(By.className("shopping_cart_badge"));

    isCartEmpty = false;

} catch (Exception e) {

    isCartEmpty = true;

}

// ΑΠΟΤΕΛΕΣΜΑ

if (isCartEmpty) {

    System.out.println("TEST PASSED");

} else {

    System.out.println("TEST FAILED");

}

Thread.sleep(4000); // για console screenshot

driver.quit();

}
```

Πλήρης Κώδικας 13.4 – Υλοποίηση αυτοματοποιημένης αφαίρεσης προϊόντος από το καλάθι

```
import org.openqa.selenium.By;

import org.openqa.selenium.WebDriver;

import org.openqa.selenium.chrome.ChromeDriver;

import io.github.bonigarcia.wdm.WebDriverManager;

import org.openqa.selenium.chrome.ChromeOptions;

public class Main {

    public static void main(String[] args) throws InterruptedException {

        WebDriverManager.chromedriver().setup();

        ChromeOptions options = new ChromeOptions();

        options.addArguments("--incognito");

        WebDriver driver = new ChromeDriver(options);

        // Open site

        driver.get("https://www.saucedemo.com/");

        driver.manage().window().maximize();

        Thread.sleep(2000);

        // LOGIN

        driver.findElement(By.id("user-name")).sendKeys("standard_user");

        driver.findElement(By.id("password")).sendKeys("secret_sauce");

        Thread.sleep(2000);

        driver.findElement(By.id("login-button")).click();

        Thread.sleep(3000);

        // ADD PRODUCT

        driver.findElement(By.id("add-to-cart-sauce-labs-backpack")).click();

        Thread.sleep(2000);

        // GO TO CART

        driver.findElement(By.className("shopping_cart_link")).click();

        Thread.sleep(3000); // 🖼️ Screenshot 5.13

        // CLICK CHECKOUT
```

```
driver.findElement(By.id("checkout")).click();

Thread.sleep(3000); // 🖼️ Screenshot 5.14

// FILL FORM

driver.findElement(By.id("first-name")).sendKeys("Test");
driver.findElement(By.id("last-name")).sendKeys("User");
driver.findElement(By.id("postal-code")).sendKeys("12345");

Thread.sleep(3000);

driver.findElement(By.id("continue")).click();

Thread.sleep(3000); // 🖼️ Screenshot 5.15

// VERIFICATION (είμαστε στο overview)

if (driver.getCurrentUrl().contains("checkout-step-two")) {
    System.out.println("TEST PASSED");
} else {
    System.out.println("TEST FAILED");
}

Thread.sleep(4000);

driver.quit();
}
}
```

Πλήρης Κώδικας 13.5 – Υλοποίηση αυτοματοποιημένης διαδικασίας checkout

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.