ΣΧΟΛΗ ΜΗΧΑΝΙΚΩΝ

ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΗΛΕΚΤΡΟΝΙΚΩΝ ΣΥΣΤΗΜΑΤΩΝ

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΕΛΕΓΧΟΙ ΛΟΓΙΣΜΙΚΟΥ:

ΕΙΔΗ, ΑΥΤΟΜΑΤΟΠΟΙΗΣΗ ΚΑΙ ΜΕΛΕΤΗ ΠΕΡΙΠΤΩΣΗΣ ΕΛΕΓΧΟΥ ΑΠΟΔΟΣΗΣ

Του φοιτητή
ΣΤΗΒ ΤΡΑΣΑ
Αρ. Μητρώου: 093436

Επιβλέπων
ΑΝΤΩΝΗΣ ΣΙΔΗΡΟΠΟΥΛΟΣ
Επίκουρος Καθηγητής

Σεπτέμβριος 2022

Τίτλος Π.Ε.: ΕΛΕΓΧΟΙ ΛΟΓΙΣΜΙΚΟΥ: ΕΙΔΗ, ΑΥΤΟΜΑΤΟΠΟΙΗΣΗ ΚΑΙ ΜΕΛΕΤΗ ΠΕΡΙΠΤΩΣΗΣ ΕΛΕΓΧΟΥ ΑΠΟΔΟΣΗΣ

Κωδικός Π.Ε.: 21285

Ονοματεπώνυμο φοιτητή: ΣΤΗΒ ΤΡΑΣΑΣ

Ονοματεπώνυμο εισηγητή: ΑΝΤΩΝΗΣ ΣΙΔΗΡΟΠΟΥΛΟΣ

Ημερομηνία ανάληψης Π.Ε.: 24/07/2021

Ημερομηνία περάτωσης Π.Ε.: 04/09/2022

Βεβαιώνω ότι είμαι ο συγγραφέας αυτής της εργασίας και ότι κάθε βοήθεια την οποία είχα για την προετοιμασία της είναι πλήρως αναγνωρισμένη και αναφέρεται στην εργασία. Επίσης, έχω καταγράψει τις όποιες πηγές από τις οποίες έκανα χρήση δεδομένων, ιδεών, εικόνων και κειμένου, είτε αυτές αναφέρονται ακριβώς είτε παραφρασμένες. Επιπλέον, βεβαιώνω ότι αυτή η εργασία προετοιμάστηκε από εμένα προσωπικά, ειδικά ως πτυχιακή εργασία, στο Τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του ΔΙ.ΠΑ.Ε.

Πρόλογος

Ο όρος "Έλεγχος λογισμικού" είναι ένας ευρύς όρος που καλύπτει ένα μεγάλο φάσμα δραστηριοτήτων. Ο έλεγχος λογισμικού μπορεί να γίνει στον κώδικα, στο γραφικό περιβάλλον του χρήστη, στη βάση δεδομένων, στην απόδοση του συστήματος και γενικά σε όλες τις περιοχές που σχετίζονται με το υπό-κατασκευή λογισμικό. Αυτοί οι έλεγχοι μπορούν να εκτελεστούν από έναν χρήστη χειρωνακτικά, μπορούν να γίνουν όμως και επαναληπτικά χωρίς μεγάλο κόστος σε χρόνο και χρήμα, με τη χρήση εργαλείων αυτοματοποίησης των ελέγχων.

Στόχος αυτής της πτυχιακής είναι η μελέτη και η κατανόηση των διαφόρων ειδών ελέγχου λογισμικού και η πρακτική τους εφαρμογή σε συγκεκριμένη μελέτη περίπτωσης. Συγκεκριμένα, σε μια ιστοσελίδα του τμήματος ΜΠΗΣ θα εφαρμοστεί αυτοματοποιημένος έλεγχος με δύο τρόπους, ενώ στη συνέχεια θα μελετηθεί η περίπτωση ελέγχου απόδοσης σε δύο άλλες ιστοσελίδες του τμήματος. Τα αποτελέσματα των ελέγχων απόδοσης και τα συμπεράσματα που θα προκύψουν, θα καταγραφούν και θα αναλυθούν στο τέλος αυτής της εργασίας.

Περίληψη

Αντικείμενο της παρούσας εργασίας είναι η μελέτη των ελέγχων λογισμικού και η εφαρμογή τους με τη χρήση εξειδικευμένων εργαλείων, σε δικτυακούς τόπους του τμήματος Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του Διεθνούς Πανεπιστημίου της Ελλάδος.

Αρχικά, γίνεται μια ανασκόπηση για το τι είναι οι έλεγχοι λογισμικού, παρουσιάζοντας τις επτά βασικές αρχές, τα επίπεδα και τα είδη των ελέγχων. Επίσης, αναλύονται οι βασικές κατηγορίες των ελέγχων και οι τεχνικές που αυτές περιλαμβάνουν.

Στη συνέχεια, παρουσιάζεται μια από τις μεγαλύτερες τάσεις τα τελευταία χρόνια στον τομέα των ελέγχων λογισμικού, η αυτοματοποίηση των ελέγχων. Η αυτοματοποίηση μπορεί να γίνει με τη χρήση κώδικα ή με τη χρήση εργαλείων που δεν απαιτούν τη συγγραφή κώδικα. Στο πρώτο μέρος της αυτοματοποίησης μελετήθηκε το Selenium, το δημοφιλέστερο εργαλείο για την αυτοματοποίηση με κώδικα. Αφού αναφέρθηκαν τα εργαλεία και οι δυνατότητές του, δημιουργήθηκε ένα σενάριο ελέγχου για την επίδειξη της λειτουργίας του, ελέγχοντας την ιστοσελίδα διπλωματικών του τμήματος ΜΠΗΣ. Στο δεύτερο μέρος της αυτοματοποίησης, παρουσιάστηκε το Leapwork, ένα εργαλείο ρομποτικής αυτοματοποίησης διεργασιών χωρίς τη χρήση κώδικα και δημιουργήθηκε το ίδιο σενάριο ελέγχου για την ιστοσελίδα διπλωματικών.

Έπειτα, παρουσιάζεται ένα σημαντικό είδος μη λειτουργικού ελέγχου, ο έλεγχος απόδοσης. Γίνεται μια ανασκόπηση των διαφόρων ειδών ελέγχου απόδοσης και της διαδικασίας που ακολουθείται για τη διενέργεια ενός ολοκληρωμένου ελέγχου απόδοσης. Με τη χρήση του Apache JMeter, πραγματοποιήθηκε έλεγχος απόδοσης στα δύο συστήματα ανακοινώσεων του τμήματος ΜΠΗΣ, αναλύοντας τη διαδικασία δημιουργίας και εκτέλεσης των σεναρίων ελέγχου.

Τέλος, παρουσιάζονται και αναλύονται τα αποτελέσματα του ελέγχου απόδοσης, επιβεβαιώνοντας ότι το “aBoard”, η υπηρεσία ανακοινώσεων που αντικατέστησε το “apps”, είναι γενικά καλύτερη.

SOFTWARE TESTING: TYPES, AUTOMATION AND A PERFORMANCE TESTING CASE STUDY

STIV TRASAS

Abstract

The subject of this thesis is the study of software testing and its application, using specialised tools, on websites of the Department of Information and Electronic Engineering of the International Hellenic University.

Initially a review of what is software testing is done, presenting the seven basic principles, the levels and types of tests. Also, the basic categories of test techniques are analysed.

Subsequently, an introduction is done to one of the biggest trends in recent years in software testing, test automation. Automation can be done by writing code or by using tools that don't require from the user to write code. In the first part of automation, Selenium was studied, the most popular tool for coded automation. After mentioning its tools and capabilities, a test script was created for demonstration purposes, to test the thesis website of the Department of Information and Electronic Engineering. In the second part of automation, Leapwork, a robotic process automation tool was presented and the same test script was created for the thesis website.

Thereafter, an important type of non-functional testing is presented, performance testing. A review of the various types of performance tests is done and the process followed to conduct a comprehensive performance test is analysed. A performance test was conducted on the announcements systems of the Department of Information and Electronic Engineering, using Apache JMeter, while presenting the whole procedure.

Finally, the performance test results are presented and analysed, confirming that "aBoard", the announcements service that replaced "apps", is better in general.

Ευχαριστίες

Με την ολοκλήρωση της παρούσας πτυχιακής εργασίας, θα ήθελα να ευχαριστήσω τον επιβλέποντα καθηγητή μου κ. Αντώνη Σιδηρόπουλο που δέχτηκε με χαρά να με επιβλέψει σε αυτό το θέμα που επέλεξα να ασχοληθώ. Η συμβολή και οι συμβουλές του ήταν πολύτιμες καθ' όλη τη διάρκεια εκπόνησης της πτυχιακής, μέσα σ' ένα εξαιρετικό κλίμα συνεργασίας.

Περιεχόμενα

Κατάλογος Εικόνων

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Συντομογραφίες

ΔΙΠΑΕ	Διεθνές Πανεπιστήμιο Ελλάδος
ΜΠΗΣ	Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων
Π.Ε.	Πτυχιακή Εργασία
API	Application Programming Interface
DOM	Document Object Model
POM	Project Object Model
DOS	Denial of Service
HTTP	Hypertext Transfer Protocol
IDE	Integrated Development Environment
FIFO	First In, First Out

Κεφάλαιο 1ο: Έλεγχοι λογισμικού

1.1 Εισαγωγή

Στην εποχή μας, η τεχνολογία είναι ιδιαίτερα ανεπτυγμένη και αυτό έχει ως αποτέλεσμα τις αυξημένες απαιτήσεις στο κάθε είδους λογισμικό που δημιουργείται. Ειδικότερα σε κάποιους τομείς (όπως υγεία, αεροπλοΐα, πυρηνική ενέργεια, κλπ.) η ποιότητα του λογισμικού είναι εξαιρετικά κρίσιμης σημασίας για τη ζωή των ανθρώπων.

Ποιότητα λογισμικού είναι ο βαθμός κατά τον οποίο ένα σύστημα, ένα στοιχείο, ή μια διαδικασία ανταποκρίνεται σε συγκεκριμένες απαιτήσεις [1].

Ο έλεγχος του λογισμικού είναι μια από τις βασικές φάσεις του κύκλου ζωής ανάπτυξης λογισμικού. Οι φάσεις αυτές είναι η ανάλυση, η σχεδίαση, η υλοποίηση, ο έλεγχος και η διανομή και συντήρηση του λογισμικού. Ο έλεγχος ποιότητας λογισμικού ορίζεται ως το σύνολο των δραστηριοτήτων που είναι σχεδιασμένες έτσι ώστε να αξιολογούν την ποιότητα ενός ανεπτυγμένου προϊόντος [1].

Ο κύκλος ζωής ελέγχου λογισμικού αποτελείται από τις εξής δραστηριότητες: Ανάλυση απαιτήσεων, προγραμματισμός των ελέγχων, σχεδίαση των ελέγχων, προετοιμασία περιβάλλοντος εκτέλεσης ελέγχων, εκτέλεση των ελέγχων και ολοκλήρωση των ελέγχων [2].

1.2 Οι επτά αρχές του ελέγχου λογισμικού

Υπάρχουν επτά βασικές αρχές στον έλεγχο λογισμικού, οι οποίες έχουν καθιερωθεί από το International Software Testing and Qualifications Board (ISTQB), και είναι πολύ διαδεδομένες στον τομέα του ελέγχου λογισμικού [2].

1. **Ο έλεγχος λογισμικού δείχνει την παρουσία προβλημάτων και όχι την απουσία**

   Ο έλεγχος λογισμικού μπορεί να δείξει ότι υπάρχουν προβλήματα μειώνοντας την πιθανότητα να παραμείνουν στο λογισμικό, αλλά δεν μπορεί να αποδείξει ότι δεν υπάρχουν άλλα προβλήματα.

2. **Ο πλήρης έλεγχος λογισμικού είναι αδύνατος**

   Είναι αδύνατο να ελεγχθεί ένα σύστημα πλήρως, εξετάζοντας όλους τους συνδυασμούς εισόδων και συνθηκών. Για αυτό και χρησιμοποιούνται τεχνικές ανάλυσης του ρίσκου και προτεραιότητες ώστε να επιλεχθεί η λειτουργικότητα που θα ελεγχθεί.

3. **Ο πρόωρος έλεγχος λογισμικού εξοικονομεί χρόνο και χρήμα**

   Όσο πιο γρήγορα ξεκινάει ο έλεγχος λογισμικού στον κύκλο ζωής του λογισμικού, τόσο πιο γρήγορα εντοπίζονται τυχόν προβλήματα και συνεπώς μειώνεται το κόστος δαπανηρών αλλαγών.

4. **Τα προβλήματα του λογισμικού συνήθως ομαδοποιούνται**

   Συνήθως, πολλά προβλήματα του λογισμικού συγκεντρώνονται σε συγκεκριμένες λειτουργικότητες. Οι ομάδες ελέγχου λογισμικού πρέπει να χρησιμοποιούν αυτή την πληροφορία, ώστε να επικεντρώνουν τους ελέγχους τους σε αυτά τα μέρη του λογισμικού.

5. **Το παράδοξο των φυτοφαρμάκων**

Αν σε ένα φυτό ρίχνουμε συνεχώς το ίδιο φυτοφάρμακο, ένα συγκεκριμένο παράσιτο μπορεί να μην ξαναεμφανιστεί αλλά μπορούν να εμφανιστούν διαφορετικά παράσιτα. Έτσι και στον έλεγχο λογισμικού αν εκτελούμε τον ίδιο έλεγχο συνεχώς τότε κάποια στιγμή δεν θα βρίσκει προβλήματα, αλλά μπορεί να έχουν δημιουργηθεί νέα προβλήματα που δεν καλύπτονται από τον συγκεκριμένο έλεγχο.

6. **Ο έλεγχος λογισμικού εξαρτάται από το περιβάλλον που πραγματοποιούνται οι έλεγχοι**

   Μπορεί να υπάρξουν διαφορετικές προσεγγίσεις ως προς τον τρόπο που θα πραγματοποιηθεί ο έλεγχος λογισμικού. Για παράδειγμα, διαφορετικά θα ελεγχθεί ένα ζωτικής σημασίας ιατρικό λογισμικό από μία απλή εμπορική εφαρμογή κινητού.

7. **Η απουσία προβλημάτων είναι πλάνη**

   Όπως προαναφέρθηκε στις δύο πρώτες αρχές, είναι αδύνατο η ομάδα ελέγχου λογισμικού να εκτελέσει όλους τους πιθανούς ελέγχους και να βρει όλα τα πιθανά προβλήματα. Είναι πλάνη να πιστεύουμε ότι επειδή όλα τα προβλήματα που βρέθηκαν έχουν επιδιορθωθεί, θα εξασφαλιστεί η επιτυχία του συστήματος. Είναι πιθανό το τελικό σύστημα να είναι δύσχρηστο, να μην ικανοποιεί τις προσδοκίες του πελάτη ή να είναι κατώτερο συγκριτικά με άλλα ανταγωνιστικά συστήματα.

1.3 Επίπεδα ελέγχων

Ανάλογα με το επίπεδο του συστήματος που ελέγχεται, ο δυναμικός έλεγχος διακρίνεται σε [3]:

Έλεγχος μονάδας

Ο έλεγχος μονάδας (unit testing) αποτελεί τον πρώτο έλεγχο που πρέπει να πραγματοποιηθεί στο σύστημα. Εξετάζεται, αν κάθε μονάδα του λογισμικού λειτουργεί σωστά και σύμφωνα με τις προδιαγραφές που έχουν τεθεί. Οι έλεγχοι αυτοί δημιουργούνται από τους προγραμματιστές που έχουν γράψει τον κώδικα, επειδή απαιτείται γνώση της λειτουργικότητας του κώδικα και πρόσβαση σε αυτόν.

Αυτό το επίπεδο ελέγχου είναι ιδιαίτερα σημαντικό, καθώς συνήθως ενσωματώνεται στη διαδικασία δημιουργίας του build ενός προγράμματος και μπορεί να προσφέρει γρήγορο εντοπισμό σφαλμάτων στο αρχικό στάδιο. Όσο νωρίτερα εντοπίζονται τα σφάλματα, τόσο περισσότερο μειώνεται το κόστος επιδιόρθωσης τους.

Οι δύο βασικές μετρικές για τους ελέγχους μονάδας είναι η "Επιτυχία-Αποτυχία" και η κάλυψη του κώδικα (code coverage). Ένας έλεγχος μονάδας όταν εκτελεστεί, είτε θα επιτύχει είτε θα αποτύχει. Η μετρική της κάλυψης του κώδικα δείχνει το ποσοστό στο οποίο οι γραμμές του κώδικα έχουν χρησιμοποιηθεί μέσω των ελέγχων μονάδας.

Έλεγχος ενσωμάτωσης

Ο έλεγχος ενσωμάτωσης (integration testing) επικεντρώνεται στην επικοινωνία μεταξύ των διαφόρων συστατικών ενός συστήματος με στόχο να επαληθεύσει ότι η λειτουργική και μη λειτουργική συμπεριφορά των διεπαφών είναι όπως έχει σχεδιαστεί.

Υπάρχουν δύο διαφορετικά επίπεδα ελέγχων ενσωμάτωσης: έλεγχος ενσωμάτωσης συστατικών (component) και έλεγχος ενσωμάτωσης συστήματος. Ο πρώτος επικεντρώνεται στις αλληλεπιδράσεις και τις διεπαφές μεταξύ των ενσωματωμένων συστατικών. Ο δεύτερος επικεντρώνεται στις

αλληλεπιδράσεις και τις διεπαφές μεταξύ συστημάτων, πακέτων και microservices, ενώ μπορεί επίσης να καλύψει και αλληλεπιδράσεις με εξωτερικούς οργανισμούς.

Ο έλεγχος ενσωμάτωσης συστατικών είναι συνήθως ευθύνη των προγραμματιστών, ενώ ο έλεγχος ενσωμάτωσης συστήματος είναι γενικά ευθύνη των μηχανικών διασφάλισης ποιότητας λογισμικού.

Έλεγχος συστήματος

Ο έλεγχος συστήματος επικεντρώνεται στη συμπεριφορά και τις δυνατότητες ολόκληρου του συστήματος, ελέγχοντας τις από άκρο σε άκρο (end to end) εργασίες που μπορεί να εκτελέσει το σύστημα και έχοντας ως στόχο να επιβεβαιώσει ότι το σύστημα είναι ολοκληρωμένο και ότι θα λειτουργήσει όπως αναμένεται.

Ο έλεγχος συστήματος συνήθως διεξάγεται από ανεξάρτητη ομάδα πραγματοποίησης των ελέγχων η οποία βασίζεται σε μεγάλο βαθμό στις προδιαγραφές, οπότε ελλείψεις ή προβλήματα στις προδιαγραφές μπορεί να οδηγήσουν σε έλλειψη κατανόησης σχετικά με τη συμπεριφορά του συστήματος.

Έλεγχος αποδοχής

Ο έλεγχος αποδοχής (acceptance testing) αποτελεί τον τελευταίο έλεγχο πριν το λογισμικό παραδοθεί στον πελάτη και μέσω αυτού του ελέγχου διασφαλίζεται ότι το σύστημα που έχει αναπτυχθεί πληροί τις απαιτήσεις του πελάτη. Μπορεί να εντοπιστούν ελαττώματα κατά τη διάρκεια του ελέγχου αποδοχής, όμως η εύρεση ελαττωμάτων συχνά δεν είναι στόχος και η εύρεση σημαντικού αριθμού ελαττωμάτων κατά τη διάρκεια του ελέγχου αποδοχής θεωρείται σημαντικός κίνδυνος για το έργο. Ο έλεγχος αποδοχής μπορεί επίσης να ικανοποιεί νομικές ή ρυθμιστικές απαιτήσεις.

Υπάρχουν διαφορετικές μορφές ελέγχων αποδοχής. Οι πιο συνηθισμένες είναι:

- **User acceptance testing (UAT)**

  Επικεντρώνεται στην επικύρωση της καταλληλότητας χρήσης του συστήματος από τους τελικούς χρήστες σε πραγματικό ή περιβάλλον προσομοίωσης.

- **Operational acceptance testing**

  Διασφαλίζεται ότι οι διαχειριστές του συστήματος μπορούν να διατηρήσουν το σύστημα σε κανονική λειτουργία για τους χρήστες στο περιβάλλον παραγωγής, ακόμα και υπό δύσκολες συνθήκες.

- **Contractual and regulatory acceptance testing**

  Ελέγχεται ότι το σύστημα ικανοποιεί τα κριτήρια αποδοχής που έχουν οριστεί στο συμβόλαιο που έχει υπογραφεί. Ο έλεγχος ρυθμιστικής αποδοχής διασφαλίζει ότι τηρούνται κυβερνητικοί, νομικοί ή κανονισμοί ασφαλείας.

- **Alpha and beta testing**

  Τα alpha και beta testing χρησιμοποιούνται συνήθως από εταιρίες εμπορικού λογισμικού που θέλουν να λάβουν σχόλια από δυνητικούς ή υπάρχοντες πελάτες προτού διατεθεί το λογισμικό στην αγορά. Οι έλεγχοι alpha πραγματοποιούνται στις εγκαταστάσεις της εταιρίας ανάπτυξης από δυνητικούς ή υπάρχοντες πελάτες ή από ανεξάρτητη ομάδα που πραγματοποιεί τους ελέγχους. Οι έλεγχοι beta πραγματοποιούνται από δυνητικούς ή υπάρχοντες πελάτες στις εγκαταστάσεις των τελευταίων.

1.4 Είδη ελέγχων

Τα είδη ελέγχων λογισμικού περικλείουν ένα σύνολο δραστηριοτήτων ελέγχου που επικεντρώνεται σε συγκεκριμένα χαρακτηριστικά του συστήματος [2].

Λειτουργικοί έλεγχοι

Οι λειτουργικοί έλεγχοι περιλαμβάνουν ελέγχους που αξιολογούν λειτουργίες που πρέπει να εκτελεί το σύστημα. Οι απαιτήσεις λειτουργίας του συστήματος είναι απαραίτητες για τη δημιουργία αυτών των ελέγχων. Περιλαμβάνουν κυρίως ελέγχους τύπου μαύρου κουτιού (black box testing) καθώς δεν μας ενδιαφέρει ο πηγαίος κώδικας του συστήματος αλλά η συμπεριφορά του.

Μη λειτουργικοί έλεγχοι

Οι μη λειτουργικοί έλεγχοι δεν αξιολογούν το πώς συμπεριφέρεται ένα σύστημα αλλά το πόσο καλά συμπεριφέρεται. Ελέγχουν χαρακτηριστικά του συστήματος όπως η ευχρηστία, η απόδοση, η ασφάλεια, η προσβασιμότητα, κλπ. Για παράδειγμα ο έλεγχος ευχρηστίας εξετάζει πόσο εύχρηστο είναι το σύστημα για τον χρήστη και αν του προκαλεί δυσκολίες για την εκτέλεση κάποια ενέργειας. Ο έλεγχος προσβασιμότητας εξετάζει την ευχρηστία του συστήματος και εντοπίζει προβλήματα για άτομα με ειδικές ανάγκες.

Δομικοί έλεγχοι

Στους δομικούς ελέγχους (ή αλλιώς ελέγχους λευκού κουτιού) εξετάζεται η εσωτερική δομή και υλοποίηση του συστήματος. Αυτοί οι έλεγχοι μπορούν να πραγματοποιηθούν σε οποιοδήποτε επίπεδο, αλλά εφαρμόζονται κυρίως στο επίπεδο μονάδας και επίπεδο ενσωμάτωσης. Η πληρότητα και το εύρος κάλυψης αυτών των ελέγχων μετριέται με τη μετρική της κάλυψης κώδικα (code coverage) που προσδιορίζει το ποσοστό κάλυψης των γραμμών του κώδικα από τους ελέγχους.

Έλεγχοι που σχετίζονται με τις αλλαγές του συστήματος

Όταν πραγματοποιούνται αλλαγές σε ένα σύστημα είτε για τη διόρθωση ενός προβλήματος είτε λόγω νέας λειτουργικότητας, θα πρέπει να γίνουν έλεγχοι για να επιβεβαιώσουν ότι οι αλλαγές έχουν διορθώσει το πρόβλημα στην πρώτη περίπτωση ή ότι έχει υλοποιηθεί σωστά η λειτουργικότητα στη δεύτερη περίπτωση χωρίς να προκαλέσει νέα προβλήματα. Αυτοί οι έλεγχοι ονομάζονται έλεγχοι επιβεβαίωσης (confirmation testing) και έλεγχοι παλινδρόμησης (regression testing).

1.5 Κατηγορίες και τεχνικές ελέγχων

Υπάρχουν δύο βασικές κατηγορίες ελέγχου λογισμικού, ο στατικός έλεγχος (static testing) όπου το πρόγραμμα δεν εκτελείται και ο δυναμικός έλεγχος (dynamic testing) όπου το πρόγραμμα εκτελείται.

1.5.1 Στατικός έλεγχος

Στο στατικό έλεγχο, δεν απαιτείται η εκτέλεση του προγράμματος που ελέγχεται όπως γίνεται στον δυναμικό έλεγχο, αλλά βασιζόμαστε είτε στη χειροκίνητη εξέταση των προϊόντων της εργασίας (π.χ. επιθεωρήσεις) είτε στην αξιολόγηση του κώδικα ή άλλων προϊόντων εργασίας με τη χρήση εργαλείων (στατική ανάλυση).

1.5.1.1 Επιθεωρήσεις

Οι επιθεωρήσεις (reviews) μπορεί να ποικίλλουν από ανεπίσημες σε επίσημες. Οι ανεπίσημες χαρακτηρίζονται από το γεγονός ότι δεν ακολουθούν μια καθορισμένη διαδικασία και δεν έχουν επίσημα τεκμηριωμένα αποτελέσματα. Οι επίσημες επιθεωρήσεις χαρακτηρίζονται από συμμετοχή

της ομάδας, τεκμηριωμένα αποτελέσματα και καθορισμένες διαδικασίες για τη διεξαγωγή της επιθεώρησης. Όλοι οι τύποι επιθεώρησης μπορούν να βοηθήσουν στον εντοπισμό σφαλμάτων και ο επιλεγμένος τύπος επιθεώρησης θα πρέπει να βασίζεται στις ανάγκες του έργου, στους διαθέσιμους πόρους, στην εταιρική κουλτούρα, στην πολυπλοκότητα του προϊόντος εργασίας που πρόκειται να εξεταστεί και τυχόν νομικές ή ρυθμιστικές απαιτήσεις [2].

Τα τέσσερα πιο συνηθισμένα είδη επιθεώρησης περιγράφονται παρακάτω.

Άτυπη επιθεώρηση

Ο κύριος στόχος της άτυπης επιθεώρησης (informal review) είναι ο εντοπισμός πιθανών σφαλμάτων, ενώ χρησιμεύει επίσης και για την εξεύρεση λύσεων ή τη γρήγορη επίλυση μικροπροβλημάτων. Δεν βασίζεται σε καθορισμένη διαδικασία και το πιθανότερο είναι να μην περιλαμβάνει συνάντηση επιθεώρησης. Συνήθως πραγματοποιείται από κάποιον συνάδελφο και χρησιμοποιείται πολύ συχνά σε ευέλικτες μεθοδολογίες ανάπτυξης λογισμικού.

Περιήγηση

Οι κύριοι στόχοι της περιήγησης (walkthrough) είναι ο εντοπισμός σφαλμάτων, η εξέταση εναλλακτικών τρόπων υλοποίησης και η αξιολόγηση της συμμόρφωσης σε πρότυπα και προδιαγραφές, ενώ χρησιμεύει και για την ανταλλαγή ιδεών και την εκπαίδευση των συμμετεχόντων. Περιλαμβάνει συνάντηση επιθεώρησης, η οποία καθοδηγείται από τον συντάκτη του προϊόντος εργασίας και η ύπαρξη γραφέα είναι υποχρεωτική. Στο τέλος δημιουργούνται αρχεία καταγραφής πιθανών σφαλμάτων και αναφορά επιθεώρησης.

Τεχνική επιθεώρηση

Οι κύριοι στόχοι της τεχνικής επιθεώρησης (technical review) είναι η επίτευξη συναίνεσης και η ανίχνευση πιθανών σφαλμάτων. Πιθανοί στόχοι είναι επίσης η αξιολόγηση της ποιότητας και η οικοδόμηση εμπιστοσύνης για το προϊόν εργασίας, η δημιουργία νέων ιδεών και η εξέταση εναλλακτικών υλοποιήσεων. Οι επιθεωρητές πρέπει να είναι τεχνικά ομότιμοι του συντάκτη και ειδικοί στον τομέα. Απαιτείται ατομική προετοιμασία των συμμετεχόντων πριν από τη συνάντηση αξιολόγησης, η οποία ιδανικά καθοδηγείται από εκπαιδευμένο συντονιστή και η ύπαρξη γραφέα είναι υποχρεωτική. Στο τέλος δημιουργούνται αρχεία καταγραφής πιθανών σφαλμάτων και αναφορά επιθεώρησης.

Επισκόπηση

Οι κύριοι στόχοι της επισκόπησης (inspection) είναι ο εντοπισμός πιθανών σφαλμάτων, η αξιολόγηση της ποιότητας και η αποτροπή παρόμοιων σφαλμάτων στο μέλλον μέσω της εκπαίδευσης του συντάκτη και της ανάλυσης των βαθύτερων αιτιών δημιουργίας των σφαλμάτων. Αυτό το είδος επιθεώρησης ακολουθεί μια καθορισμένη διαδικασία με επίσημα τεκμηριωμένα αποτελέσματα, βάση κανόνων και λιστών ελέγχου. Οι επιθεωρητές πρέπει να είναι τεχνικά ομότιμοι του συντάκτη και ειδικοί στον τομέα ή σε άλλους σχετικούς τομείς. Απαιτείται ατομική προετοιμασία των συμμετεχόντων πριν από τη συνάντηση αξιολόγησης, η οποία καθοδηγείται από εκπαιδευμένο συντονιστή και η ύπαρξη γραφέα είναι υποχρεωτική. Στο τέλος δημιουργούνται αρχεία καταγραφής πιθανών σφαλμάτων και αναφορά επιθεώρησης, ενώ συλλέγονται και μετρικές που χρησιμοποιούνται για τη βελτίωση της διαδικασίας ανάπτυξης λογισμικού συμπεριλαμβανομένης της διαδικασίας επισκόπησης.

Πίνακας 1.1: Ρόλοι στη διαδικασία επιθεώρησης

Ρόλος	Περιγραφή
Συντάκτης	Δημιουργεί το προϊόν εργασίας και διορθώνει τα σφάλματα που θα βρεθούν
Επιθεωρητής	Βρίσκει πιθανά σφάλματα στο προϊόν εργασίας. Μπορεί να είναι ειδικός στον τομέα ή συνάδελφος του συντάκτη
Γραφέας	Καταγράφει τα πιθανά ελαττώματα που εντοπίστηκαν κατά τη διάρκεια της επιθεώρησης όπως επίσης και αποφάσεις από τη συνάντηση επιθεώρησης
Συντονιστής	Εξασφαλίζει την αποτελεσματική διεξαγωγή των συναντήσεων επιθεώρησης και μεσολαβεί αν χρειάζεται μεταξύ των διαφόρων απόψεων
Επικεφαλής	Αναλαμβάνει τη συνολική ευθύνη για την επιθεώρηση και αποφασίζει ποιος θα συμμετάσχει, πότε και πού θα γίνει

1.5.1.2 Στατική ανάλυση

Η στατική ανάλυση είναι μια μέθοδος που εξετάζει τον πηγαίο κώδικα και πραγματοποιείται αυτόματα από ένα λογισμικό πριν την εκτέλεση του κώδικα. Ο πηγαίος κώδικας αναλύεται για δομές που είναι γνωστό ότι σχετίζονται με σφάλματα λογισμικού ή ευπάθειες ασφάλειας. Συνήθως χρησιμοποιείται για τη συμμόρφωση με πρότυπα κωδικοποίησης (π.χ. το MISRA για κρίσιμα για την ασφάλεια συστήματα) ή βιομηχανικά πρότυπα (π.χ. το ISO 26262, ένα λειτουργικό πρότυπο ασφάλειας που χρησιμοποιείται στην αυτοκινητοβιομηχανία) [4]. Όταν ανιχνεύεται μια δομή υψηλού κινδύνου, το εργαλείο στατικής ανάλυσης αναφέρει μια παραβίαση, για να ελεγχθεί στη συνέχεια από έναν προγραμματιστή και να επιδιορθωθεί.

Τα πλεονεκτήματα των εργαλείων στατικής ανάλυσης είναι η ταχύτητα, η ακρίβεια και η λεπτομέρεια. Χρειάζεται χρόνος για τους προγραμματιστές προκειμένου να επιθεωρήσουν τον κώδικα με μη αυτόματο τρόπο, όμως τα αυτοματοποιημένα εργαλεία είναι πολύ πιο γρήγορα. Ο στατικός έλεγχος του κώδικα βρίσκει τα προβλήματα νωρίς και προσδιορίζει ακριβώς που βρίσκεται το σφάλμα στον κώδικα. Έτσι, τα σφάλματα μπορούν να διορθωθούν πιο γρήγορα. Επίσης, οι χειροκίνητες επιθεωρήσεις του κώδικα είναι επιρρεπείς σε ανθρώπινο λάθος, ενώ τα αυτοματοποιημένα εργαλεία δεν είναι. Σαρώνουν κάθε γραμμή κώδικα για να εντοπίσουν πιθανά προβλήματα. Τέλος, ο έλεγχος λογισμικού δεν μπορεί να καλύψει κάθε πιθανή διαδρομή εκτέλεσης του κώδικα, ενώ ένα εργαλείο στατικής ανάλυσης μπορεί να αναλύσει εις βάθος τον κώδικα και να εντοπίσει πιθανά προβλήματα [5].

1.5.2 Δυναμικός έλεγχος

Στο δυναμικό έλεγχο, ελέγχουμε το λογισμικό για προβλήματα μέσω της εκτέλεσης του πηγαίου κώδικα. Έτσι μπορούν να εντοπιστούν προβλήματα που δεν εντοπίζονται εύκολα με τη χρήση του στατικού ελέγχου. Ο δυναμικός έλεγχος χωρίζεται σε τρείς κατηγορίες τεχνικών ελέγχου, τις τεχνικές μαύρου κουτιού, τις τεχνικές άσπρου κουτιού και τις τεχνικές που βασίζονται στην εμπειρία [3].

1.5.2.1 Τεχνικές μαύρου κουτιού

Οι τεχνικές του μαύρου κουτιού (black box) βασίζονται στις προδιαγραφές, επειδή οι έλεγχοι που εκτελούνται έχουν δημιουργηθεί με βάση τις προδιαγραφές του συστήματος. Επικεντρώνονται στις

εισόδους και εξόδους του συστήματος ελέγχοντας ποιο είναι το αποτέλεσμα που παράγεται για κάποια συγκεκριμένη είσοδο. Οι δημοφιλέστερες τεχνικές μαύρου κουτιού είναι οι παρακάτω.

Διαχωρισμός ισοδυναμίας (equivalence partitioning)

Με αυτήν την τεχνική τα δεδομένα χωρίζονται σε κλάσεις με τέτοιο τρόπο ώστε το σύστημα για κάθε κλάση να έχει την ίδια συμπεριφορά. Για κάθε κλάση υπάρχουν οι αποδεκτές τιμές στις οποίες το σύστημα πρέπει να ανταποκρίνεται και οι μη αποδεκτές όπου το σύστημα θα πρέπει να εμφανίζει μήνυμα λάθους. Συνεπώς, αν μια τυχαία αποδεκτή τιμή δουλεύει σωστά, τότε συμπεραίνουμε ότι όλες οι τιμές που είναι αποδεκτές από αυτήν την κλάση θα δουλεύουν σωστά.

Ανάλυση οριακής τιμής (boundary value analysis)

Αυτή η τεχνική αποτελεί επέκταση του διαχωρισμού ισοδυναμίας και ελέγχει τις ακραίες τιμές μιας κλάσης. Έτσι ο έλεγχος επικεντρώνεται στη μικρότερη και μεγαλύτερη τιμή που μπορεί να λάβει το σύστημα σαν είσοδο όπως επίσης και τις μη αποδεκτές τιμές που είναι κοντά στις αποδεκτές.

Έλεγχος πίνακα αποφάσεων (decision table testing)

Η συγκεκριμένη τεχνική είναι χρήσιμη για πολύπλοκους επιχειρηματικούς κανόνες. Αφού εντοπίσουμε τις εισόδους και εξόδους του συστήματος, τις καταγράφουμε σε έναν πίνακα όπου σε κάθε στήλη ανάλογα με το συνδυασμό, αντιστοιχεί μια απόφαση.

Έλεγχος μετάβασης κατάστασης (state transition testing)

Με αυτήν την τεχνική εξετάζεται κάθε κατάσταση του λογισμικού και τα πιθανά αποτελέσματα που παράγονται ανάλογα με τις επιλογές του χρήστη.

Έλεγχος περιπτώσεων χρήσης (use case testing)

Σε αυτήν την τεχνική, οι έλεγχοι προκύπτουν από τις περιπτώσεις χρήσης, οι οποίες έχουν ήδη αναπτυχθεί σε αρχικά στάδια του κύκλου ζωής ανάπτυξης λογισμικού. Οι περιπτώσεις χρήσης περιγράφουν συγκεκριμένη χρήση του συστήματος από έναν χρήστη.

1.5.2.2 Τεχνικές λευκού κουτιού

Οι τεχνικές του λευκού κουτιού (white box) βασίζονται στη δομή του συστήματος, επειδή οι έλεγχοι που εκτελούνται αφορούν την εσωτερική δομή του συστήματος και τον κώδικα. Αυτές οι τεχνικές μπορούν να χρησιμοποιηθούν σε όλα τα επίπεδα ελέγχου, αλλά συνήθως χρησιμοποιούνται στο επίπεδο μονάδας.

Έλεγχος και κάλυψη εντολών (statement testing and coverage)

Με αυτήν την τεχνική ελέγχονται οι εκτελέσιμες εντολές στον κώδικα. Η κάλυψη μετριέται ως ο αριθμός των εντολών που εκτελέστηκαν από τους ελέγχους, διαιρεμένος με το συνολικό αριθμό εντολών στο αντικείμενο ελέγχου, εκφρασμένο σε ποσοστό.

Έλεγχος και κάλυψη αποφάσεων (decision testing and coverage)

Με αυτήν την τεχνική ελέγχονται οι αποφάσεις και τα πιθανά αποτελέσματα στον κώδικα. Για παράδειγμα σε μια εντολή IF θα ελεγχθεί ο κώδικας και για το αληθές αποτέλεσμα και για το ψευδές.

1.5.2.3 Τεχνικές που βασίζονται στην εμπειρία

Η τελευταία κατηγορία τεχνικών δυναμικού ελέγχου είναι αυτές που βασίζονται στην εμπειρία του ατόμου που πραγματοποιεί τους ελέγχους. Αυτές οι τεχνικές βασίζονται στις γνώσεις, στην εμπειρία

και στη διαίσθηση του μηχανικού διασφάλισης ποιότητας λογισμικού και μπορεί να είναι χρήσιμες όταν χρησιμοποιούνται συμπληρωματικά με τις προηγούμενες τεχνικές.

Πρόβλεψη σφαλμάτων (error guessing)

Ένας έμπειρος tester γνωρίζει καλά το σύστημα, πως θα έπρεπε να λειτουργεί και τι λάθη κάνουν συνήθως οι προγραμματιστές, οπότε μπορεί να ελέγξει το σύστημα και να εντοπίσει πιθανά σφάλματα.

Διερευνητικοί έλεγχοι (exploratory testing)

Αυτή η τεχνική αποτελεί μια άτυπη τεχνική ελέγχου λογισμικού και συνήθως χρησιμοποιείται όταν οι προδιαγραφές είναι λίγες ή ανεπαρκείς, ή υπάρχει σημαντική πίεση χρόνου. Με αυτήν την τεχνική ο tester χρησιμοποιεί το σύστημα εκτελώντας βασικές λειτουργίες, ενώ μπορεί να χρησιμοποιηθεί επίσης ως μέθοδος εκμάθησης του συστήματος για νέους tester.

Έλεγχοι βάσει λίστας (checklist based testing)

Αυτή η τεχνική βασίζεται σε λίστες ελέγχου οι οποίες έχουν δημιουργηθεί από τους testers. Είναι χρήσιμη γιατί παρέχει μια μορφή καθοδήγησης στους testers, ως προς τα σενάρια που πρέπει να ελεγχθούν, βοηθώντας τους να μην παραλείψουν κάτι σημαντικό.

1.6 Επίλογος

Σε αυτό το κεφάλαιο παρουσιάστηκε το θεωρητικό υπόβαθρο των ελέγχων λογισμικού. Αρχικά, αναφέρθηκαν οι επτά αρχές του ελέγχου λογισμικού όπως αυτές έχουν καθιερωθεί από το ISTQB. Στη συνέχεια, αναλύθηκαν τα τέσσερα επίπεδα ελέγχων, όπως και τα είδη των ελέγχων. Τέλος, παρουσιάστηκαν αναλυτικά οι δύο βασικές κατηγορίες ελέγχων, οι στατικοί και οι δυναμικοί έλεγχοι. Οι στατικοί έλεγχοι διακρίνονται περαιτέρω, στις επιθεωρήσεις και στη στατική ανάλυση, ενώ οι δυναμικοί έλεγχοι διακρίνονται στις τεχνικές μαύρου κουτιού, λευκού κουτιού και σε αυτές που βασίζονται στην εμπειρία.

Κεφάλαιο 2ο: Αυτοματοποίηση ελέγχων με κώδικα - Selenium

2.1 Εισαγωγή

Η τεχνολογία εξελίσσεται με γρήγορους ρυθμούς και οι εταιρείες που ασχολούνται με την παραγωγή λογισμικού έχουν την ανάγκη να παραδίδουν νέες εκδόσεις των προϊόντων τους στους τελικούς χρήστες, σε πιο τακτά χρονικά διαστήματα, ακόμα και σε καθημερινή βάση. Οι σύγχρονες πρακτικές ανάπτυξης λογισμικού όπως είναι η συνεχής ενσωμάτωση (Continuous Integration, CI) και η συνεχής παράδοση (Continuous Delivery, CD) απαιτούν γρήγορες και αυτοματοποιημένες διαδικασίες για τον έλεγχο του λογισμικού που οδεύει προς το περιβάλλον παραγωγής [6-7]. Η ανυπαρξία αυτοματοποίησης στο στάδιο του ελέγχου δημιουργεί ένα σημείο συμφόρησης στο σύγχρονο κύκλο ανάπτυξης λογισμικού [8].

2.2 Αυτοματοποίηση ελέγχων με τη χρήση του Selenium

Το Selenium είναι ένα έργο ανοιχτού λογισμικού που περιλαμβάνει εργαλεία και βιβλιοθήκες που υποστηρίζουν την αυτοματοποίηση προγραμμάτων περιήγησης, επιτρέποντας τον έλεγχο και την εξομοίωση της αλληλεπίδρασης ενός χρήστη με το πρόγραμμα περιήγησης. Υποστηρίζει τις δημοφιλέστερες γλώσσες προγραμματισμού: Java, Python, C#, Ruby, JavaScript, Perl, PHP. Επίσης υποστηρίζει τα εξής προγράμματα περιήγησης: Google Chrome, Mozilla Firefox, Microsoft Edge, Internet Explorer, Safari.

Η ανάπτυξη του Selenium ξεκίνησε το 2004 όπου ο Jason Huggings, εργαζόμενος στη ThoughtWorks στο Σικάγο, ανέπτυξε ένα JavaScript module με την ονομασία "JavaScriptTestRunner" για τον έλεγχο μιας εσωτερικής ιστοσελίδας της εταιρείας [9]. Η ονομασία Selenium προέκυψε μετά από ένα αστείο που έκανε ο Huggings, λέγοντας στους συνεργάτες του ότι "η δηλητηρίαση από Υδράργυρο (Mercury) μπορεί να θεραπευτεί με τη λήψη συμπληρώματος Σεληνίου (Selenium)", αναφερόμενος στο δημοφιλές λογισμικό ελέγχων εκείνη την περίοδο, το QuickTest Professional της Mercury Interactive [10].

Το Selenium έχει εξελιχθεί ως το δημοφιλέστερο εργαλείο για την αυτοματοποίηση ελέγχων για τους παρακάτω λόγους:

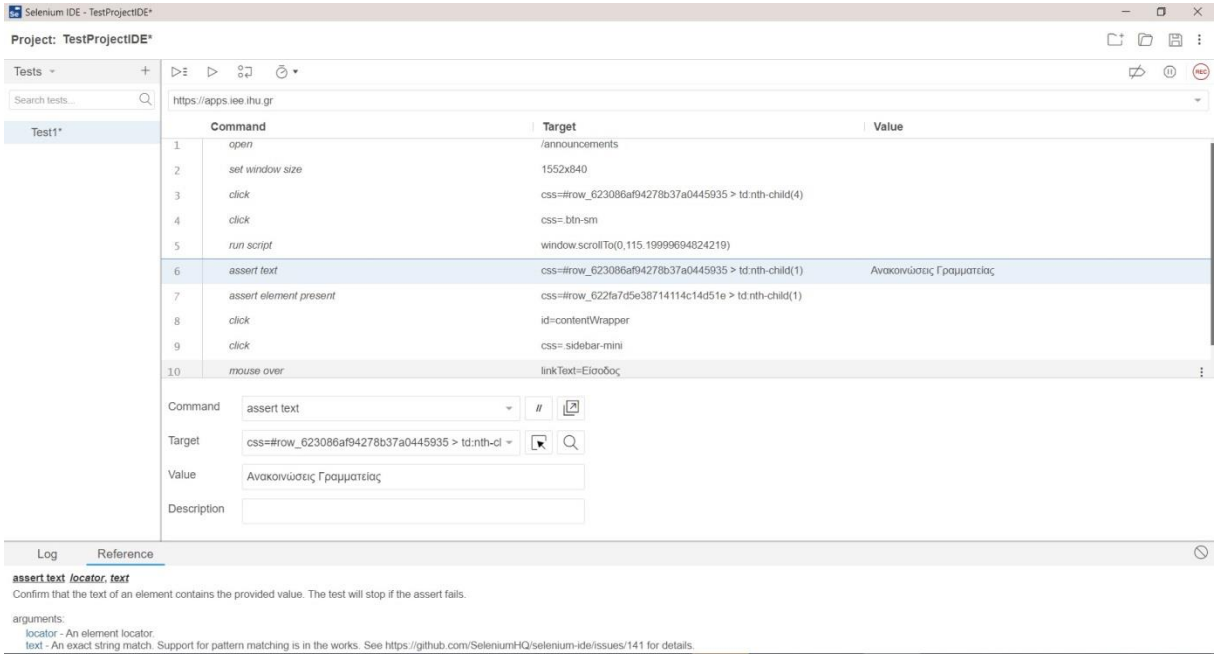
- Είναι έργο ανοιχτού λογισμικού, συνεπώς δεν υπάρχει κόστος για άδεια χρήσης
- Υποστηρίζει πολλές γλώσσες προγραμματισμού
- Είναι συμβατό με πολλά λειτουργικά συστήματα (Linux, UNIX, Mac, Windows)
- Υποστηρίζει τα δημοφιλέστερα προγράμματα περιήγησης
- Ο WebDriver μπορεί να ενσωματωθεί σε διάφορα framework, όπως το Maven για build automation, το JUnit/TestNG για τις βιβλιοθήκες ελέγχων ή ακόμα και το Jenkins για την αυτόματη εκτέλεση των ελέγχων μετά από κάθε αλλαγή στον κώδικα
- Έχει μεγάλη υποστήριξη από την κοινότητα

2.3 Εργαλεία του Selenium

Το Selenium είναι μια σουίτα λογισμικού η οποία αποτελείται από τα παρακάτω εργαλεία:

2.3.1 Selenium IDE

Το Selenium IDE (Integrated Development Environment) είναι μια επέκταση που αρχικά ήταν διαθέσιμη μόνο για τον Mozilla Firefox και από τον Μάρτιο του 2020 έγινε διαθέσιμη και στον Google Chrome. Υποστηρίζει την ανάπτυξη σεναρίων ελέγχου με δυνατότητα καταγραφής της αλληλεπίδρασης του χρήστη με το πρόγραμμα περιήγησης.



Εικόνα 2.1: Το γραφικό περιβάλλον του Selenium IDE

Είναι ένα εύκολο στη χρήση εργαλείο το οποίο μπορεί να χρησιμοποιηθεί από χρήστες με ελάχιστη εμπειρία στην ανάπτυξη αυτοματοποιημένων ελέγχων για ιστοσελίδες, καθώς έχει γραφικό περιβάλλον για την καταγραφή και επεξεργασία των σεναρίων ελέγχου και δεν απαιτεί καμία άλλη εγκατάσταση πέραν της επέκτασης στο πρόγραμμα περιήγησης. Παρέχει επίσης τη δυνατότητα μετατροπής και εξαγωγής των σεναρίων ελέγχου σε όλες τις υποστηριζόμενες από το Selenium γλώσσες προγραμματισμού.

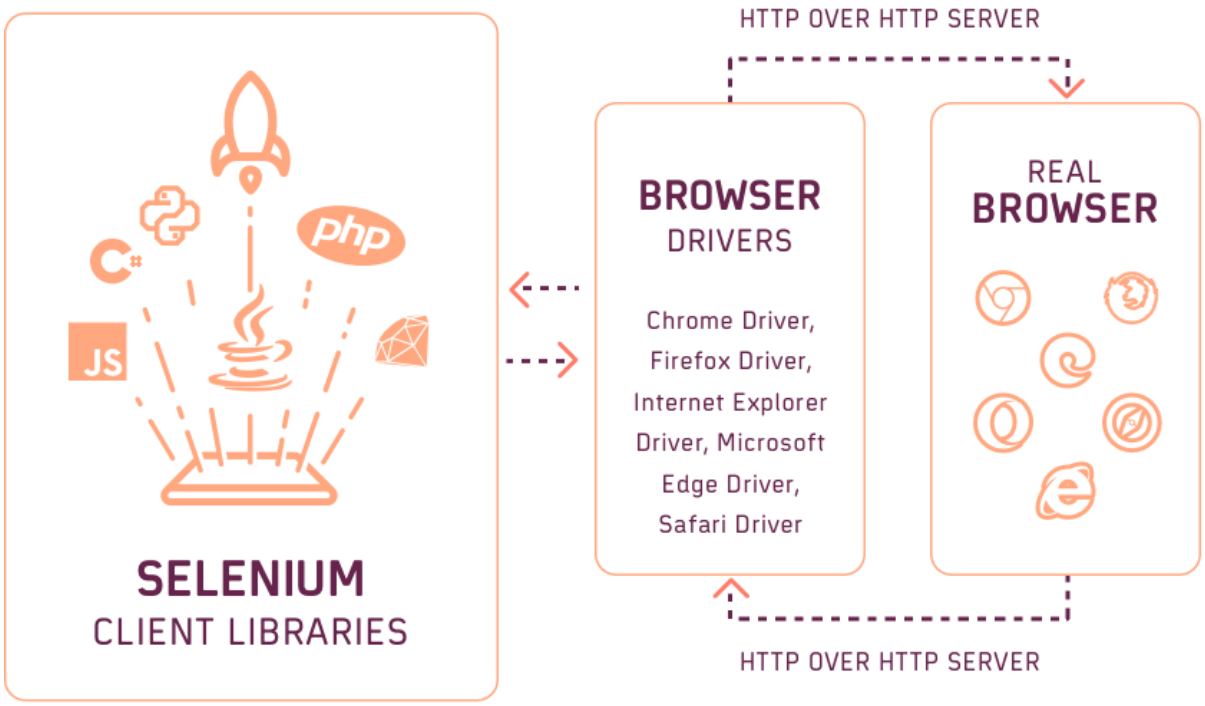
Το Selenium IDE έχει και κάποιους περιορισμούς οι οποίοι το καθιστούν ακατάλληλο για επαγγελματική χρήση. Ενδεικτικά, τέτοιοι περιορισμοί είναι: η αδυναμία δημιουργίας επαναχρησιμοποιούμενων συστατικών, η έλλειψη δυνατότητας εισαγωγής δεδομένων π.χ. στοιχεία χρηστών από πηγές δεδομένων (αρχεία Excel, βάση δεδομένων), η αδυναμία καταγραφής αναδυόμενων παραθύρων [11 - 12].

2.3.2 Selenium WebDriver

Το Selenium WebDriver είναι το πιο χρήσιμο εργαλείο της σουίτας εργαλείων του Selenium. Παρέχει ένα σύνολο διεπαφών που επιτρέπουν τον εντοπισμό και το χειρισμό στοιχείων του DOM σε ιστοσελίδες όπως επίσης και τον έλεγχο της συμπεριφοράς ενός πράκτορα χρήστη με το χειρισμό συμβάντων ποντικιού και πληκτρολογίου.

Με το Selenium στην έκδοση 4, ο WebDriver χρησιμοποιεί πλέον μόνο το WebDriver W3C πρωτόκολλο για την επικοινωνία με το πρόγραμμα περιήγησης, το οποίο έχει πολλά πλεονεκτήματα

σε σύγκριση με το JSON Wire πρωτόκολλο που χρησιμοποιούνταν σε προγενέστερες εκδόσεις του Selenium [13-14]. Δε χρειάζεται κωδικοποίηση και αποκωδικοποίηση των αιτημάτων διεπαφής καθώς όλα τα σύγχρονα προγράμματα περιήγησης χρησιμοποιούν το W3C πρωτόκολλο, συνεπώς ο WebDriver επικοινωνεί απευθείας με το πρόγραμμα περιήγησης. Αυτό οδηγεί σε μεγαλύτερη σταθερότητα των ελέγχων μεταξύ διαφορετικών προγραμμάτων περιήγησης. Επίσης, η διεπαφή Actions έχει ανανεωθεί με την προσθήκη νέων δυνατοτήτων (π.χ. ενέργειες πολλαπλής αφής).



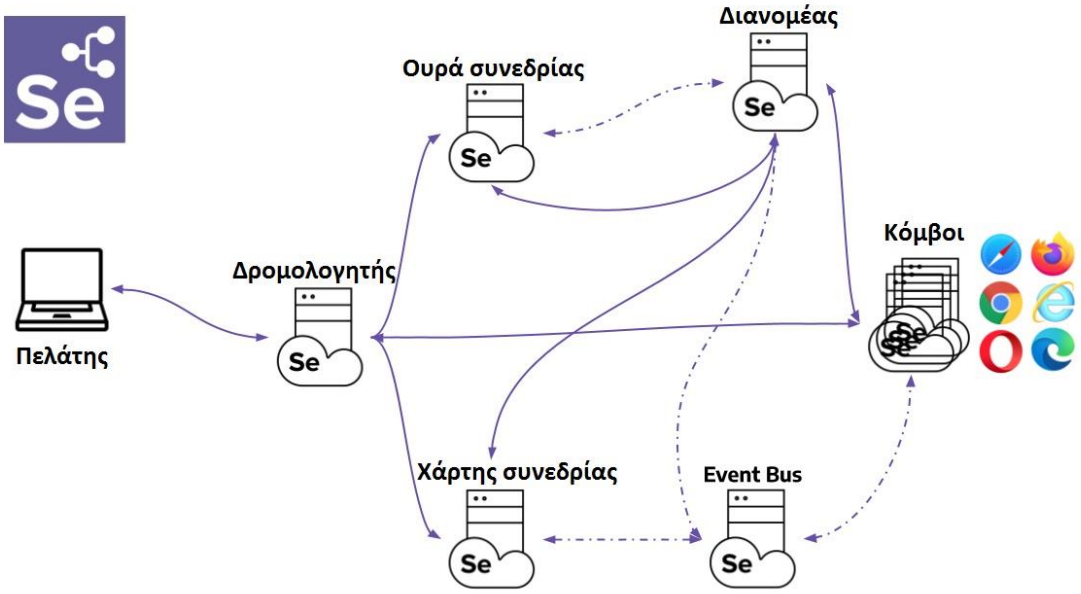
Σχήμα 2.1: Η αρχιτεκτονική του Selenium WebDriver, Πηγή: [8]

2.3.3 Selenium Grid

Το Selenium Grid είναι ένα άλλο εργαλείο του Selenium το οποίο διευκολύνει την παράλληλη εκτέλεση σεναρίων ελέγχου σε πολλαπλά μηχανήματα, επιτρέποντας την κεντρική διαχείριση διαφορετικών προγραμμάτων περιήγησης ή διαφορετικών εκδόσεων τους [15]. Το Selenium 4 υποστηρίζει τρία είδη grid: αυτόνομης λειτουργίας, κλασσικό grid (hub – node) και πλήρως κατανεμημένο [16].

Τα συστατικά του Selenium Grid 4 στην πλήρως κατανεμημένη λειτουργία είναι [Σχήμα 2.2]:

- Δρομολογητής: Δέχεται τα αιτήματα νέα συνεδρίας και τα προωθεί στον σωστό παραλήπτη
- Ουρά συνεδρίας: Διατηρεί όλα τα νέα αιτήματα συνεδρίας με σειρά FIFO
- Διανομέας: Επιλέγει τον κατάλληλο κόμβο στον οποίο πρέπει να εκτελεστεί το σενάριο ελέγχου
- Χάρτης συνεδρίας: Διατηρεί τις αντιστοιχίσεις αναγνωριστικού συνεδρίας - κόμβου
- Κόμβος: Μηχάνημα στο οποίο εκτελούνται τα σενάρια ελέγχων
- Event bus: Χρησιμεύει ως διαδρομή επικοινωνίας μεταξύ των άλλων συστατικών του grid



Σχήμα 2.2: Selenium Grid 4, Πηγή: [9]

Στην πλήρως κατανεμημένη λειτουργία του το Selenium Grid 4 λειτουργεί με τον εξής τρόπο. Ο δρομολογητής είναι το σημείο εισόδου στο Grid και όταν λαμβάνει νέα αιτήματα συνεδρίας τα τοποθετεί σε μια ουρά συνεδρίας. Ο διανομέας διαβάζει την ουρά συνεδρίας και επιλέγει το καταλληλότερο slot που φιλοξενείται σε κάποιον κόμβο και προωθεί το νέο αίτημα συνεδρίας σε αυτόν τον κόμβο. Όταν ο κόμβος έχει ξεκινήσει τη συνεδρία, ο διανομέας τοποθετεί την αντιστοίχιση του αναγνωριστικού συνεδρίας και της διεύθυνσης του κόμβου σε έναν χάρτη συνεδρίας. Όταν ο δρομολογητής λαμβάνει αιτήματα για ενεργές συνεδρίες, χρησιμοποιεί τον χάρτη συνεδρίας για να εντοπίσει τον κόμβο στον οποίο θα προωθήσει το αίτημα, χωρίς τη διαμεσολάβηση του διανομέα [17].

2.4 Δυνατότητες του WebDriver

Τα βήματα που ακολουθούμε για τη δημιουργία ενός σεναρίου ελέγχου με τη χρήση του WebDriver είναι τα εξής. Αρχικά δημιουργούμε ένα στιγμιότυπο του WebDriver, περιηγούμαστε στην επιθυμητή ιστοσελίδα και εντοπίζουμε τα στοιχεία ιστού (web elements) μέσω των εντοπιστών. Έπειτα εκτελούμε τις ενέργειες χρήστη που επιθυμούμε σε αυτά τα στοιχεία και συγκρίνουμε τα αποτελέσματα με τα αναμενόμενα αποτελέσματα.

2.4.1 Εντοπιστές

Τα στοιχεία ιστού είναι αντικείμενα στο DOM και προκειμένου να εκτελεστούν λειτουργίες σε αυτά πρέπει να εντοπιστούν μέσω των εντοπιστών (locators) [18]. Για να αναζητήσουμε στοιχεία ιστού το Selenium έχει τη μέθοδο “findElement” η οποία βρίσκει και επιστρέφει ένα μόνο στοιχείο ιστού με βάση τα κριτήρια αναζήτησης του locator. Αν τα στοιχεία είναι περισσότερα του ενός, τότε επιστρέφει το πρώτο αποτέλεσμα της αναζήτησης. Επίσης παρέχει και τη μέθοδο “findElements” η οποία επιστρέφει όλα τα στοιχεία που βρέθηκαν.

```
WebElement element = driver.findElement (By.<Locator>);
List <WebElement> list = driver.findElements (By.<Locator>);
```

Το Selenium μέσω της κλάσης "By" παρέχει τις παρακάτω μεθόδους για τον καθορισμό της στρατηγικής που θα χρησιμοποιηθεί ώστε να εντοπιστεί το στοιχείο ιστού [19].

- id
- name
- className
- cssSelector
- linkText
- partialLinkText
- tagName
- xpath

Στην έκδοση 4 του Selenium προστέθηκε μια νέα κατηγορία locators, οι σχετικοί locators. Αυτοί είναι χρήσιμοι όταν δεν είναι εύκολο να δημιουργηθεί ένας locator για το επιθυμητό στοιχείο ιστού, αλλά είναι πιο εύκολο να περιγραφεί η θέση του σε σχέση με ένα στοιχείο που ήδη έχει έναν locator.

- above
- below
- leftOf
- rightOf
- near

Στο παρακάτω παράδειγμα, το πεδίο του email εντοπίζεται σε σχέση με το πεδίο του password, καθώς βρίσκεται πάνω από αυτό.

```
WebElement passwordField = driver.findElement(By.id("password"));
WebElement emailField = driver.findElement(with(By.tagName("input"))
                                            .above(passwordField));
```

2.4.2 Πληροφορίες που μπορούν να εξαχθούν από τα στοιχεία ιστού

Αφού εντοπιστεί ένα στοιχείο ιστού στο DOM, μπορούμε να πάρουμε κάποιες πληροφορίες για αυτό το στοιχείο με τις παρακάτω μεθόδους:

- isDisplayed, επιστρέφει true αν το στοιχείο εμφανίζεται στην ιστοσελίδα
- isEnabled, επιστρέφει true αν το στοιχείο είναι ενεργοποιημένο
- isSelected, επιστρέφει true αν το στοιχείο είναι επιλεγμένο
- tagName, επιστρέφει το tag name του στοιχείου
- getRect, επιστρέφει τις διαστάσεις και τις συντεταγμένες του στοιχείου μέσω των μεθόδων getX, getY, getWidth, getHeight
- getCSSValue, επιστρέφει την τιμή της καθορισμένης ιδιότητας του στοιχείου
- getText, επιστρέφει το κείμενο του στοιχείου

Στο παρακάτω παράδειγμα, ανακτούμε το χρώμα του κειμένου του συγκεκριμένου στοιχείου ιστού.

```
String cssValue = driver.findElement(By.linkText("Sample text"))
                                        .getCssValue("color");
```

2.4.3 Εντολές αλληλεπίδρασης

Το Selenium παρέχει τις εξής μεθόδους για την αλληλεπίδραση με κάποιο στοιχείο ιστού:

- click
- sendKeys, πληκτρολογεί το κείμενο σε ένα επεξεργάσιμο στοιχείο
- clear, επαναφέρει το περιεχόμενο του στοιχείου
- select, αυτή η κλάση περιλαμβάνει τις μεθόδους selectByVisibleText, selectByValue, selectByIndex και χρησιμοποιείται σε dropdown μενού

Επίσης, μέσω του Actions API, ο WebDriver υποστηρίζει πολλές εντολές που μπορούν να πραγματοποιηθούν με ποντίκι, πληκτρολόγιο, πένα και τροχό κύλισης. Ενδεικτικά, παρέχονται οι παρακάτω μέθοδοι:

- clickAndHold
- doubleClick
- keyDown
- scrollToElement
- scrollByAmount

2.4.4 Αναμονές

Κατά την εκτέλεση σεναρίων ελέγχου με το Selenium είναι πολύ συχνό το "ElementNotVisibleException" στην περίπτωση που αναζητούμε στοιχείο το οποίο δεν είναι ακόμα διαθέσιμο στο DOM. Για να αποφευχθεί αυτή η εξαίρεση, πρέπει να χρησιμοποιηθούν οι εντολές αναμονής (waits). Όταν χρησιμοποιείται μια εντολή αναμονής, το σενάριο ελέγχου τίθεται σε παύση για συγκεκριμένο χρονικό διάστημα προτού εκτελεστεί το επόμενο βήμα του σεναρίου. Κατά τη διάρκεια αυτής της παύσης, ο WebDriver ελέγχει εάν το στοιχείο ιστού υπάρχει, είναι ορατό, μπορεί να κλικαριστεί, κλπ [20]. Το Selenium παρέχει τρείς τύπους αναμονών: Implicit, Explicit και Fluent.

Implicit

Με την Implicit αναμονή, ο WebDriver θα περιμένει μετά από κάθε εντολή για το ορισμένο χρονικό διάστημα, προτού εμφανίσει ένα "NoSuchElementException". Αφού οριστεί η Implicit αναμονή, ισχύει για όλη τη διάρκεια που το πρόγραμμα περιήγησης είναι ανοιχτό. Αυτή η αναμονή όμως αυξάνει τον χρόνο εκτέλεσης του σεναρίου ελέγχου, καθώς μετά από κάθε εντολή θα αναμένει τον καθορισμένο χρόνο ακόμα και αν το στοιχείο ιστού έχει εμφανιστεί νωρίτερα.

```
driver.manage().timeouts().implicitlyWait(10, TimeUnit.SECONDS);
```

Explicit

Με τη χρήση της Explicit αναμονής, ο WebDriver περιμένει την ικανοποίηση μίας συγκεκριμένης συνθήκης ή την υπέρβαση του μέγιστου χρόνου, προτού εμφανίσει το "ElementNotVisibleException".

Αυτή η αναμονή είναι πιο "έξυπνη" από την Implicit, καθώς η εκτέλεση του σεναρίου ελέγχου θα συνεχιστεί μόλις ικανοποιηθεί η συνθήκη (π.χ. με την εμφάνιση του στοιχείου) και χωρίς να αναμένει τη συμπλήρωση του μέγιστου χρόνου. Η συνθήκη ελέγχεται με ορισμένη συχνότητα που είναι προκαθορισμένη. Για τη δήλωση της Explicit wait απαιτείται η χρήση της κλάσης ExpectedConditions, η οποία περιλαμβάνει μεθόδους όπως οι elementToBeClickable, presenceOfElementLocated, textToBePresentInElement, visibilityOfElementLocated κλπ.

```
WebElement test = new WebDriverWait(driver, Duration.ofSeconds(10))
.until(ExpectedConditions.elementToBeClickable(By.xpath("//a/h3")));
```

Fluent

Η Fluent αναμονή είναι παρόμοια με την Explicit, με τη διαφορά ότι σε αυτή μπορεί να οριστεί η συχνότητα με την οποία ελέγχεται η συνθήκη που έχει οριστεί. Επίσης, αυτή η αναμονή μπορεί να ρυθμιστεί έτσι ώστε να αγνοεί συγκεκριμένους τύπους εξαιρέσεων κατά την αναμονή [21].

```
FluentWait wait = new FluentWait(driver);
wait.withTimeout(5000, TimeUnit.MILLISECONDS);
wait.pollingEvery(250, TimeUnit.MILLISECONDS);
wait.ignoring(NoSuchElementException.class)

wait.until(ExpectedConditions.alertIsPresent());
```

2.5 Σενάριο ελέγχου ιστοσελίδας τμήματος ΜΠΗΣ

Σε αυτό το σενάριο θα επισκεφτούμε την ιστοσελίδα διπλωματικών του τμήματος Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του Διεθνούς Πανεπιστημίου Ελλάδος, η οποία βρίσκεται στη διεύθυνση https://thesis.iee.ihu.gr. Αφού κάνουμε login και ταυτοποιηθούμε ως φοιτητής του τμήματος, θα ελέγξουμε ότι η διπλωματική που έχει καταχωρηθεί στο χρήστη έχει τον τίτλο αυτής της εργασίας και ο επιβλέπων διδάσκων είναι ο κ. Αντώνης Σιδηρόπουλος.

2.5.1 Εγκατάσταση περιβάλλοντος ανάπτυξης

Για τη δημιουργία και την εκτέλεση των σεναρίων ελέγχου θα χρειαστούμε ένα IDE, όπως το Eclipse. Η γλώσσα προγραμματισμού για το σενάριο θα είναι η Java. Στο Eclipse θα χρειαστεί να εγκαταστήσουμε το Maven, ένα εργαλείο αυτοματοποίησης της δημιουργίας των build, όπου στο αρχείο POM (Project Object Model) εισάγουμε τα dependencies του project [22]. Στη περίπτωση μας εισάγουμε τον Selenium WebDriver για την αυτοματοποίηση του προγράμματος περιήγησης, το TestNG για τους ισχυρισμούς (assertions) και κάποια άλλα απαραίτητα στοιχεία για την εκτέλεση των σεναρίων. Μπορούμε να βρούμε τα dependencies από το κεντρικό αποθετήριο του Maven στην ιστοσελίδα https://mvnrepository.com.

Παρακάτω παρατίθεται ο κώδικας του αρχείου POM.

Κώδικας 2.1: Το αρχείο POM του Maven

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
https://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>IHU</groupId>
  <artifactId>thesisTestAutomation</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>SeleniumTest</name>

<dependencies>
<dependency>
    <groupId>org.seleniumhq.selenium</groupId>
    <artifactId>selenium-java</artifactId>
    <version>4.1.4</version>
</dependency>
<dependency>
    <groupId>org.testng</groupId>
    <artifactId>testng</artifactId>
    <version>7.5</version>
    <scope>test</scope>
</dependency>
<dependency>
       <groupId>org.slf4j</groupId>
       <artifactId>slf4j-api</artifactId>
       <version>1.7.5</version>
   </dependency>
   <dependency>
       <groupId>org.slf4j</groupId>
       <artifactId>slf4j-simple</artifactId>
       <version>1.6.4</version>
   </dependency>
</dependencies>

 <build>
    <pluginManagement>
      <plugins>
        <plugin>
          <groupId>org.apache.maven.plugins</groupId>
          <artifactId>maven-surefire-plugin</artifactId>
          <version>2.22.0</version>
                   <configuration>
                          <suiteXmlFiles>>
                          <suiteXmlFile>testng.xml</suiteXmlFile>
                          </suiteXmlFiles>
                   </configuration>
        </plugin>
        <plugin>
            <artifactId>maven-compiler-plugin</artifactId>
            <version>3.8.1</version>
            <configuration>
           <source>1.8</source>
           <target>1.8</target>
            </configuration>
        </plugin>
      </plugins>
    </pluginManagement>
</build>

</project>
```

2.5.2 Το σενάριο ελέγχου με τη χρήση του Selenium WebDriver και TestNG

Αρχικά πρέπει να εισάγουμε τις κλάσεις που θα χρησιμοποιήσουμε στη συνέχεια, όπως την "WebDriverWait" και "ExpectedConditions" για να κάνουμε χρήση των αναμονών και την "annotations" για το χαρακτηρισμό μεθόδων σε μια κλάση. Σε μια κλάση μπορούμε να προσθέσουμε περισσότερες από μια μεθόδους ελέγχου, απλά προσθέτοντας το annotation "@Test" πάνω από την αρχή της μεθόδου. Οι μέθοδοι που έχουν τα annotation "@BeforeMethod" και "@AfterMethod" εκτελούνται πριν και μετά από κάθε έλεγχο αντίστοιχα.

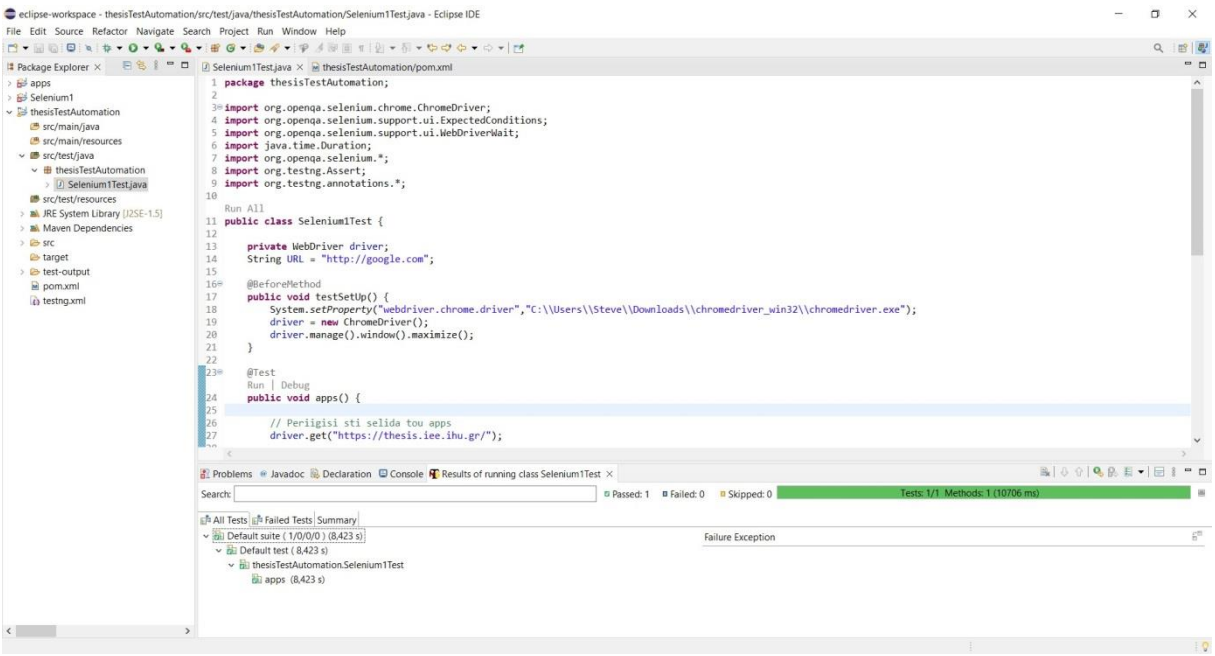
Στη συνέχεια στη μέθοδο "testSetUp" που θα εκτελεστεί πριν από κάθε μέθοδο ελέγχου, καθορίζουμε τον οδηγό (driver) του προγράμματος περιήγησης και ξεκινούμε μια νέα συνεδρία (session). Η μέθοδος "thesis" περιλαμβάνει όλα τα βήματα για τον εντοπισμό των στοιχείων ιστού και την αλληλεπίδραση με αυτά, όπως επίσης και τον έλεγχο των ζητούμενων στοιχείων. Τέλος, η μέθοδος "tearDown" περιέχει μια μοναδική εντολή η οποία κλείνει όλα τα παράθυρα του προγράμματος περιήγησης και τερματίζει τη συνεδρία.

Κώδικας 2.2: Το σενάριο ελέγχου σε Java, με χρήση WebDriver και TestNG

```
package thesisTestAutomation;

import org.openqa.selenium.chrome.ChromeDriver;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;
import java.time.Duration;
import org.openqa.selenium.*;
import org.testng.Assert;
import org.testng.annotations.*;

public class Selenium1Test {

	private WebDriver driver;

	@BeforeMethod
	public void testSetUp() {
		//Καθορισμός του προγράμματος οδήγησης για τον περιηγητή
		 System.setProperty("webdriver.chrome.driver","C:\\Users\\
			Steve\\Downloads\\chromedriver_win32\\chromedriver.exe");
		//Έναρξη νέας συνεδρίας
		driver = new ChromeDriver();
		//Μεγιστοποίηση του παραθύρου του προγράμματος περιήγησης
		driver.manage().window().maximize();
	}

	@Test
	public void thesis() {

		// Περιήγηση στη σελίδα του thesis
		driver.get("https://thesis.iee.ihu.gr/");

		// Επιλογή εισόδου χρήστη
		driver.findElement(By.xpath("/html/body/div/div/nav/div/
					a/span")).click();

		//Πληκτρολόγηση του username
		driver.findElement(By.id("username")).sendKeys("it******");

		//Πληκτρολόγηση του password
		driver.findElement(By.id("password")).sendKeys("********");
```

```
            //Επιλογή Login
            driver.findElement(By.id("loginButton")).click();

            //Εύρεση της καρτέλας «Η Διπλωματική μου» με τη χρήση αναμονής
            WebElement diplomatikiTab = new WebDriverWait(driver,
            Duration.ofSeconds(10)).until(ExpectedConditions.elementToBeClickab
            le(By.xpath("//a[contains(., 'Η Διπλωματική Μου')]")));
            diplomatikiTab.click();

            //Εύρεση του τίτλου διπλωματικής με τη χρήση αναμονής
            WebElement titlos = new WebDriverWait(driver,
            Duration.ofSeconds(10))
            .until(ExpectedConditions.visibilityOfElementLocated(By.xpath("/ht
            ml/body/div/div/div/div/div[2]/div/div/table/tbody[2]/tr/td[1]")));

            //Σύγκριση του πραγματικού τίτλου με τον αναμενόμενο
            Assert.assertEquals(("Έλεγχοι λογισμικού: είδη, αυτοματοποίηση και
            μελέτη περίπτωσης ελέγχου απόδοσης"), titlos.getText());

            //Σύγκριση του πραγματικού επιβλέποντα με τον αναμενόμενο
            Assert.assertEquals(("ΑΝΤΩΝΗΣ ΣΙΔΗΡΟΠΟΥΛΟΣ"),
            driver.findElement(By.xpath("/html/body/div/div/div/div/div[2]/div
            /div/table/tbody[2]/tr/td[2]")).getText());

    @AfterMethod
    public void tearDown() {
            driver.quit();
    }
}
```

Μετά την εκτέλεση του σεναρίου ελέγχου επιβεβαιώνουμε το επιτυχές αποτέλεσμα (Εικόνα 2.2).



Εικόνα 2.2: Αποτέλεσμα ελέγχου στο TestNG

2.6 Επίλογος

Σε αυτό το κεφάλαιο παρουσιάστηκε η αυτοματοποίηση των ελέγχων με κώδικα και συγκεκριμένα με το Selenium.

Αρχικά αναφέρθηκαν τα τρία εργαλεία που παρέχει η σουίτα του Selenium στην έκδοση 4: το Selenium IDE – ένα πρόσθετο προγράμματος περιήγησης με γραφικό περιβάλλον για τη δημιουργία και καταγραφή σεναρίων ελέγχου, ο Selenium WebDriver – ένα framework για τον εντοπισμό και χειρισμό στοιχείων ιστού όπως και το χειρισμό συμβάντων ποντικιού και πληκτρολογίου, το Selenium Grid – για την παράλληλη εκτέλεση σεναρίων ελέγχου σε πολλαπλά μηχανήματα, επιτρέποντας την κεντρική διαχείριση διαφορετικών προγραμμάτων περιήγησης.

Στη συνέχεια παρουσιάστηκαν αναλυτικά οι δυνατότητες του σημαντικότερου εργαλείου της σουίτας, του Selenium WebDriver και πιο συγκεκριμένα ο τρόπος με τον οποίο εντοπίζονται τα στοιχεία ιστού με τη χρήση των locators, οι πληροφορίες που μπορούμε να εξάγουμε από αυτά τα στοιχεία και οι τρόποι με τους οποίους μπορούμε να αλληλεπιδράσουμε με αυτά. Επίσης αναφέρθηκαν οι αναμονές που υποστηρίζει ο WebDriver.

Τέλος, υλοποιήθηκε ένα αυτοματοποιημένο σενάριο ελέγχου της ιστοσελίδας διπλωματικών του τμήματος Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του Διεθνούς Πανεπιστημίου Ελλάδος με τη χρήση του Selenium WebDriver σε γλώσσα προγραμματισμού Java.

Κεφάλαιο 3ο: Αυτοματοποίηση ελέγχων χωρίς κώδικα - RPA

3.1 Εισαγωγή

Η παγκόσμια έλλειψη εργατικού δυναμικού στον τομέα της Πληροφορικής [23] έχει δημιουργήσει την ανάγκη ύπαρξης εργαλείων που δε χρειάζονται γνώσεις προγραμματισμού για τη χρήση τους στον κύκλο ζωής ανάπτυξης λογισμικού.

Για το στάδιο των ελέγχων λογισμικού, μπορούν να χρησιμοποιηθούν εργαλεία RPA τα οποία δίνουν τη δυνατότητα στις εταιρίες να προσλάβουν εργαζομένους που θα διενεργούν τον έλεγχο λογισμικού χωρίς να έχουν υπόβαθρο γνώσεων προγραμματισμού. Αυτά τα εργαλεία παρέχουν γραφικό περιβάλλον χρήστη για τη δημιουργία των σεναρίων ελέγχου και δυνατότητες καταγραφής των κινήσεων του χρήστη. Η αλληλεπίδραση του χρήστη παρουσιάζεται με βήματα στο γραφικό περιβάλλον της εφαρμογής ενώ υπάρχει και η δυνατότητα συγγραφής κώδικα για πιο πολύπλοκες λειτουργίες, όπως για παράδειγμα η επικοινωνία με μια διεπαφή (API).

3.2 Εργαλεία ρομποτικής αυτοματοποίησης διεργασιών

Η αυτοματοποίηση επιχειρηματικών διαδικασιών με τη χρήση λογισμικού περιγράφεται με τον όρο RPA – Robotic Process Automation. Ο όρος "ρομπότ" χρησιμοποιείται μεταφορικά για να περιγράψει το λογισμικό το οποίο εκτελεί τις επαναλαμβανόμενες ψηφιακές εργασίες που βασίζονται σε κανόνες. Τα συστήματα RPA αντιγράφουν τις κινήσεις του χρήστη και την αλληλεπίδραση του με μια εφαρμογή και στη συνέχεια επαναλαμβάνουν αυτή την αλληλουχία ενεργειών με αυτοματοποιημένο τρόπο απευθείας στο γραφικό περιβάλλον της εφαρμογής.

Το RPA είναι ένας από τους πιο ανερχόμενους κλάδους στη αγορά Πληροφορικής. Η αγορά του RPA το 2021 εκτιμήθηκε στα 1.9$ δισεκατομμύρια, ενώ το 2022 αποτιμάται στα 2.3$ δισεκατομμύρια και προβλέπεται να επεκταθεί με ετήσιο ρυθμό ανάπτυξης 38%, για να οδηγηθεί στα 30$ δισεκατομμύρια το 2030 [24]. Κατά την περίοδο της πανδημίας COVID-19 οι εταιρίες υιοθέτησαν πρακτικές εργασίας από απόσταση, το οποίο είχε ως αποτέλεσμα την ανάγκη για μεγαλύτερη αυτοματοποίηση διαδικασιών και αυτός ήταν ένας παράγοντας ο οποίος επιτάχυνε ακόμα περισσότερο την ανάπτυξη της αγοράς RPA.

Η UiPath, μια ηγέτιδα εταιρία στο χώρο, ανακοίνωσε το 2019 ότι συγκέντρωσε 568$ εκατομμύρια σε επενδύσεις και σε δύο περίπου χρόνια είχε μια τεράστια ανάπτυξη, με την αξία της να εκτοξεύεται από τα 110$ εκατομμύρια στα 7$ δις, ενώ τα ετήσια έσοδά της ανέβηκαν από τα 8$ εκατομμύρια στα 200$ εκατομμύρια. Το 2020 η Microsoft ανακοίνωσε την εξαγορά της ελληνικής Softomotive, δημιουργό του RPA προγράμματος WinAutomation, έναντι τιμήματος που ξεπερνούσε τα 100€ εκατομμύρια. Η Microsoft θα ενσωμάτωνε στο δικό της προϊόν, Microsoft Power Automate, τις δυνατότητες του WinAutomation [25].

3.3 Χρήση του RPA στον έλεγχο λογισμικού

Υπάρχουν κάποιες διαφορές στα παραδοσιακά εργαλεία RPA και σε αυτά που προορίζονται κυρίως για αυτοματοποίηση των ελέγχων. Ο σκοπός της αυτοματοποίησης των ελέγχων είναι να ελέγξει ότι μια διαδικασία λειτουργεί, ενώ ο σκοπός του RPA είναι να ολοκληρώσει μια διαδικασία. Με βάση αυτή τη διαφορά στο σκοπό τους, είναι φανερό ότι τα παραδοσιακά εργαλεία RPA δεν καλύπτουν πλήρως τις απαιτήσεις σε δυνατότητες για την αυτοματοποίηση ελέγχων λογισμικού.

Κάποιες δυνατότητες που είναι απαραίτητες στην αυτοματοποίηση ελέγχων είναι η αποσφαλμάτωση και η δυνατότητα αναφοράς. Ο χρήστης ενός RPA εργαλείου δεν περιμένει να υπάρχουν σφάλματα στην εφαρμογή στην οποία αυτοματοποιεί διαδικασίες. Αντιθέτως, ο χρήστης ενός εργαλείου αυτοματοποίησης ελέγχων αναζητά τα σφάλματα στην εφαρμογή και όταν αυτά βρεθούν, θέλει να μπορεί να τα εντοπίσει και να τα διορθώσει όσο το δυνατόν πιο εύκολα. Συνεπώς, τα εργαλεία αυτοματοποίησης ελέγχων χωρίς κώδικα είναι απαραίτητο να παρέχουν δυνατότητες αναφοράς και αποσφαλμάτωσης με αναλυτική καταγραφή των βημάτων και συμβάντων και στιγμιότυπα οθόνης ή βίντεο για την γρηγορότερη εύρεση των σφαλμάτων.

Στον παρακάτω πίνακα φαίνονται οι κύριες διαφορές των εξειδικευμένων εργαλείων αυτοματοποίησης ελέγχων και των παραδοσιακών εργαλείων RPA [26].

Πίνακας 3.1: Διαφορές εργαλείων αυτοματοποίησης ελέγχων και RPA

	Αυτοματοποίηση ελέγχων	**RPA**
Ιδιοκτησία	Ανήκει στο τμήμα Πληροφορικής της εταιρίας	Ανήκει σε οποιοδήποτε τμήμα τη εταιρίας
Σκοπός	Έλεγχος διαδικασιών	Ολοκλήρωση διαδικασιών
Πεδίο εφαρμογής	Συνήθως σε μία εφαρμογή	Συνήθως σε πολλές εφαρμογές
Πεδίο γνώσης	Απαιτεί γνώση του τομέα	Απαιτεί γνώση των διαδικασιών
Πηγή δεδομένων	Δεδομένα ελέγχου	Πραγματικά δεδομένα
Περιβάλλον	Περιβάλλον δοκιμών	Περιβάλλον παραγωγής
Τι αυτοματοποιείται	Σενάρια ελέγχου	Διαδικασίες

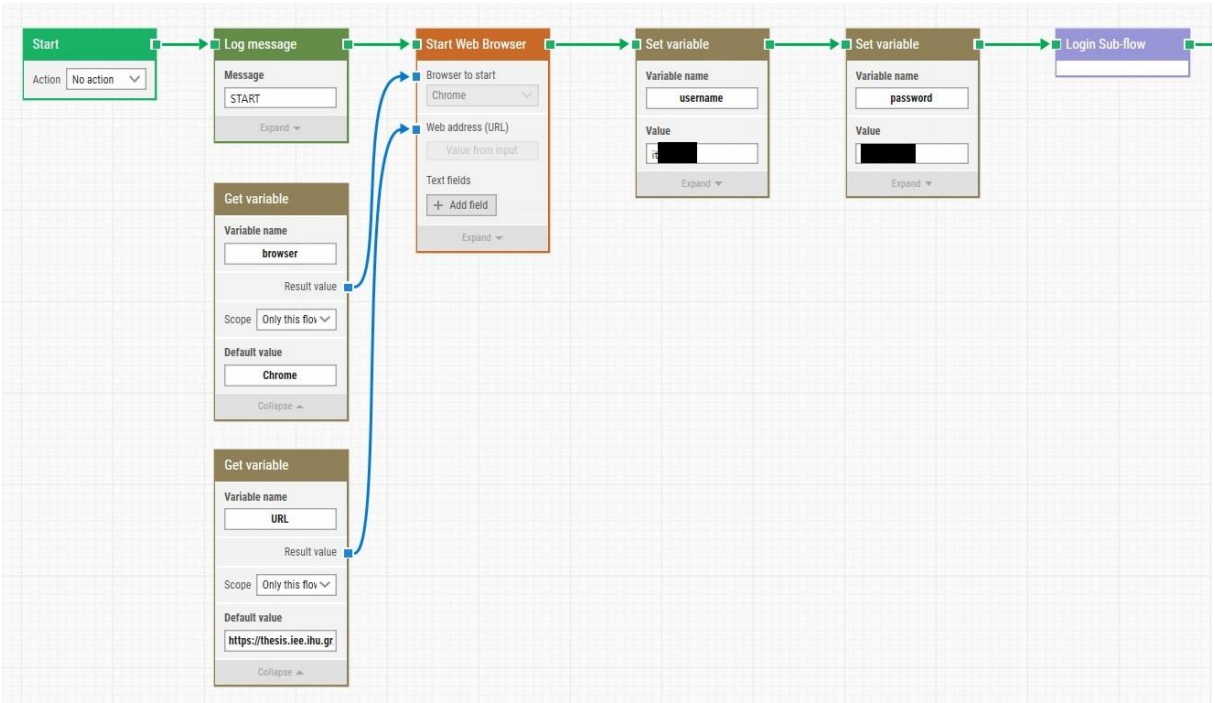
3.4 Σενάριο ελέγχου ιστοσελίδας τμήματος ΜΠΗΣ

Για λόγους σύγκρισης, χρησιμοποιήσαμε το ίδιο σενάριο ελέγχου όπως και στην περίπτωση του ελέγχου με τη χρήση του Selenium.

Παρακάτω φαίνονται διαδοχικά τα βήματα που απαιτούνται για την εκτέλεση του σεναρίου ελέγχου με τη χρήση ενός εργαλείου RPA, του Leapwork. Τα βήματα οπτικοποιούνται με τη χρήση blocks-τετραγώνων τα οποία συνδέονται μεταξύ τους με βελάκια που δείχνουν την αλληλουχία των ενεργειών, η οποία αποτελεί και τη ροή (flow) του σεναρίου ελέγχου. Τα blocks κατηγοριοποιούνται ανάλογα με τη λειτουργικότητα τους (Εικόνα 3.1).

Εικόνα 3.1: Οι κατηγορίες των blocks

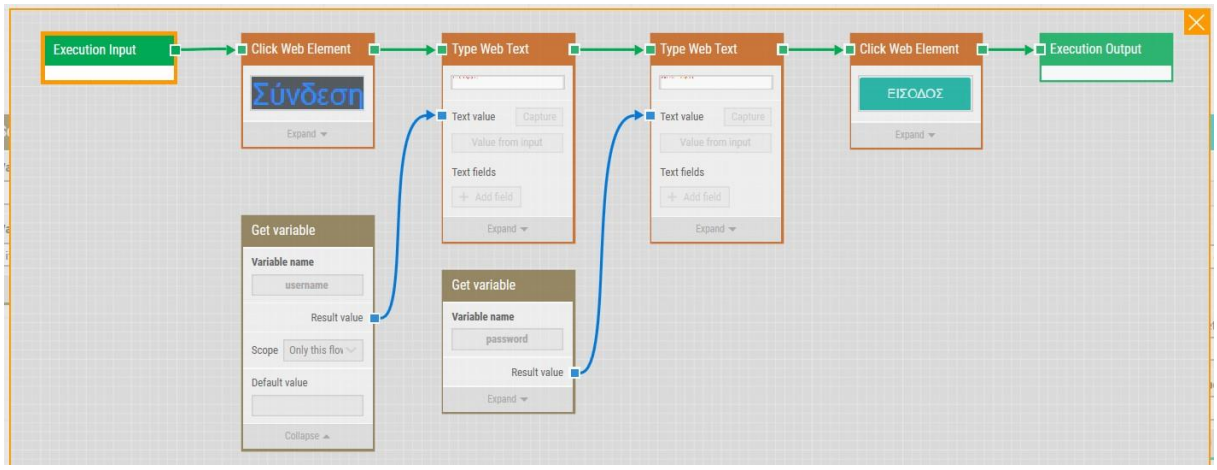
Ξεκινώντας τη δημιουργία της ροής του σεναρίου ελέγχου, προσθέτουμε ένα block που εμφανίζει το μήνυμα "START" στην κονσόλα και ένα block για την εκκίνηση του προγράμματος περιήγησης. Αυτό έχει ως εισόδους, τις εξόδους δυο άλλων block, οι οποίες παίρνουν τις μεταβλητές για το πρόγραμμα περιήγησης και το URL που θα χρησιμοποιήσει η ροή ελέγχου.



Εικόνα 3.2: Τα πρώτα βήματα της ροής

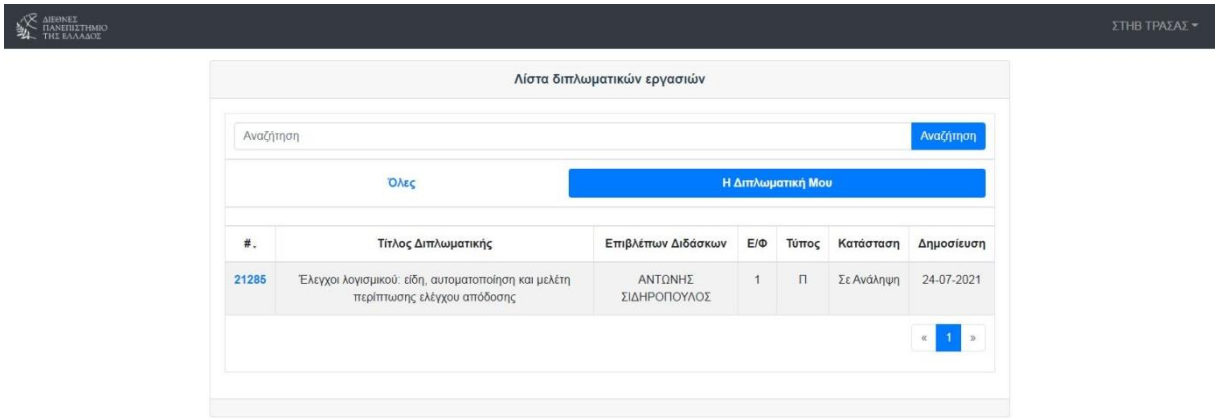
Η λειτουργία του block “Get variable” είναι πολύ χρήσιμη καθώς μπορούμε να ορίσουμε την τιμή των μεταβλητών για ένα σύνολο σεναρίων ελέγχου, π.χ. να ορίσουμε ότι θέλουμε να εκτελέσουμε τους ελέγχους στο πρόγραμμα περιήγησης Chrome και αργότερα με την αλλαγή μόνο μιας μεταβλητής να εκτελέσουμε πάλι το σύνολο των ελέγχων στο πρόγραμμα περιήγησης Firefox. Στη συνέχεια με τα δύο block “Set variable” ορίζουμε τις τιμές των username και password του χρήστη που θα χρησιμοποιηθούν για το login.

Το Leapwork δίνει τη δυνατότητα ομαδοποίησης βημάτων μέσα σε ένα block (subflow), το οποία είναι εξαιρετικά χρήσιμο για την επαναχρησιμοποίηση ροών που χρησιμοποιούνται συχνά, όπως το login. Συνεπώς, έχουμε δημιουργήσει μια υπο-ροή για το Login η οποία περιλαμβάνει όλα τα απαραίτητα βήματα για την πραγματοποίηση της αυθεντικοποίησης του χρήστη. Τα βήματα αυτά είναι η επιλογή της σύνδεσης του χρήστη, η είσοδος του username και password και η επιλογή της εισόδου ως ταυτοποιημένος χρήστης. Αφού τελειώσει η εκτέλεση των βημάτων της υπο-ροής, επιστρέφει στη κεντρική ροή του σεναρίου ελέγχου για την εκτέλεση των υπόλοιπων βημάτων.

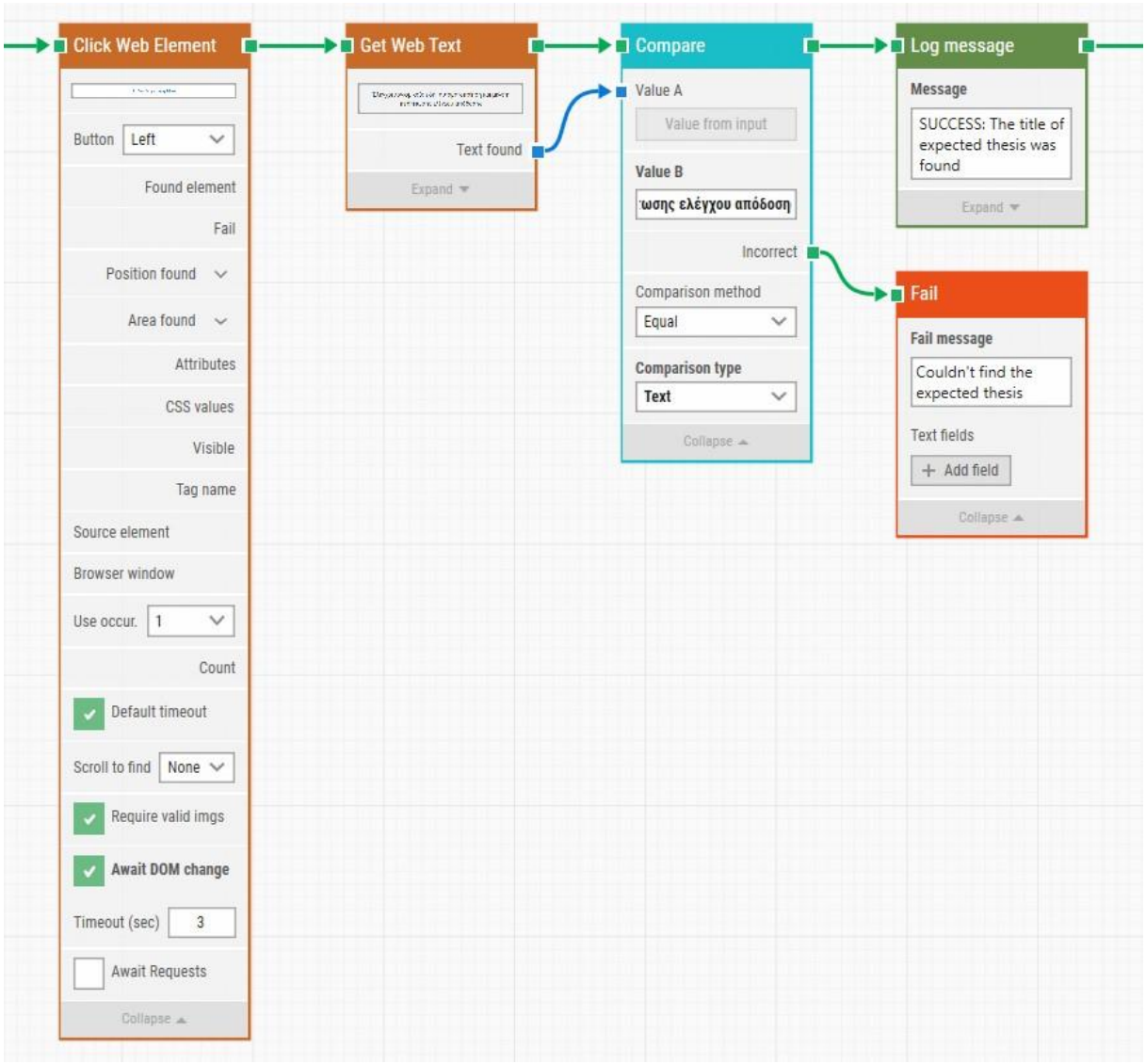


Εικόνα 3.3: Τα βήματα της υπο-ροής για το Login

Πλέον ως ταυτοποιημένος χρήστης του συστήματος, εμφανίζεται η σελίδα των διπλωματικών εργασιών. Το επόμενο βήμα του σεναρίου είναι η επιλογή της καρτέλας “Η Διπλωματική μου” όπου θα γίνουν οι έλεγχοι για τον τίτλο της διπλωματικής και τον επιβλέποντα καθηγητή.



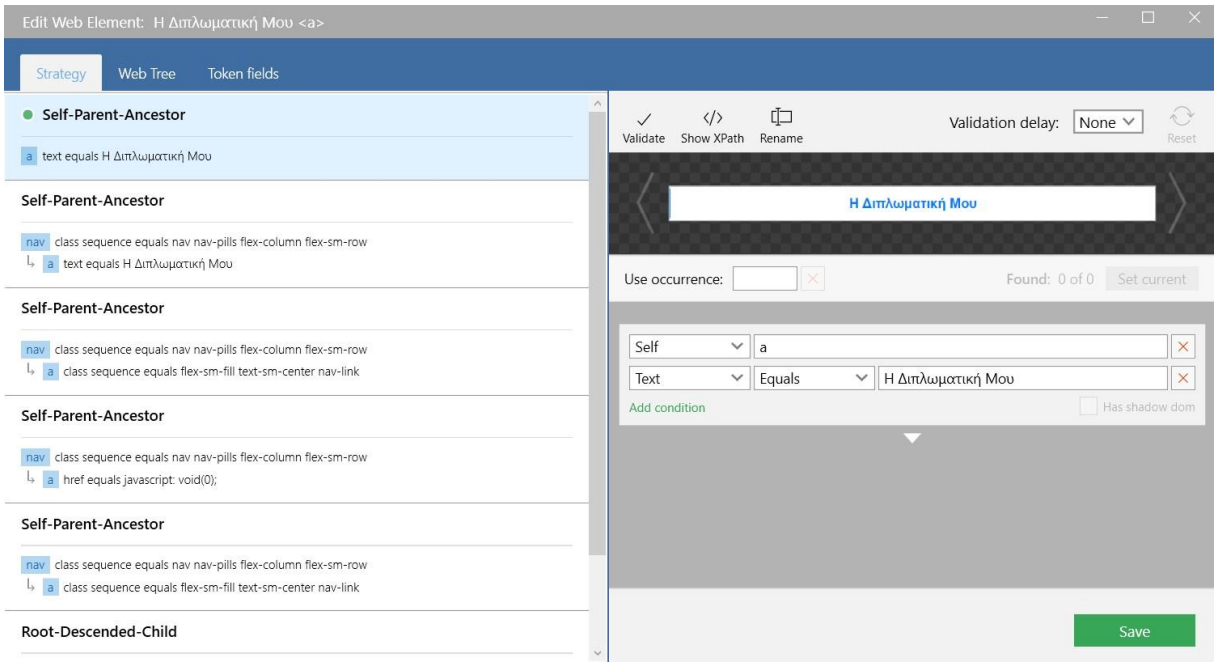
Εικόνα 3.4: Η ιστοσελίδα διπλωματικών του τμήματος ΜΠΗΣ



Εικόνα 3.5: Τα βήματα ελέγχου του τίτλου της διπλωματικής

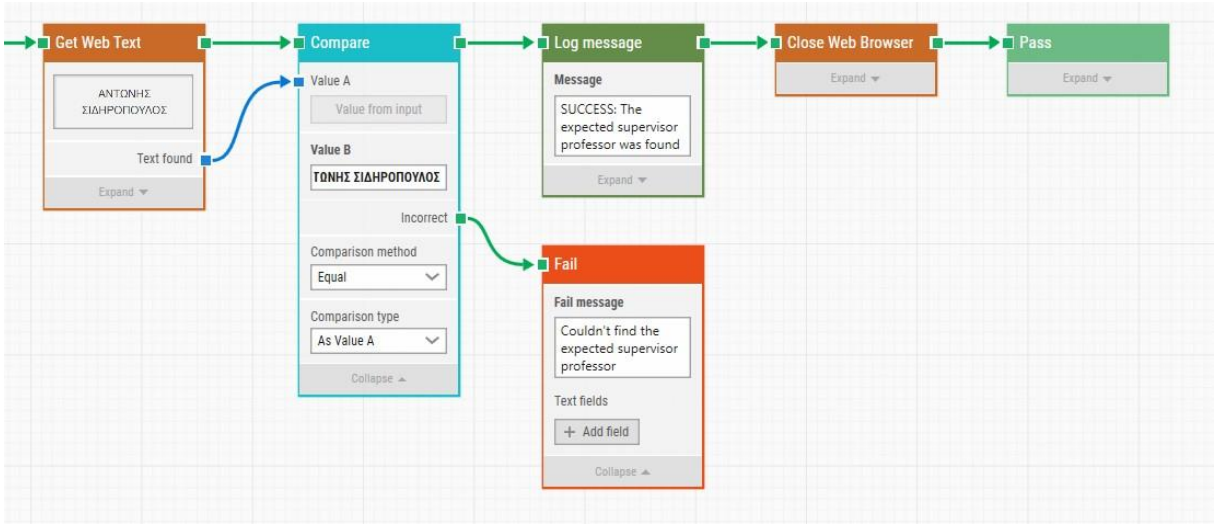
Με το block “Click web element” επιλέγουμε την καρτέλα “ Η Διπλωματική μου”. Όλα τα blocks παρέχουν επιπλέον ρυθμίσεις σχετικά με τη δυνατότητα αναμονής για την εμφάνιση του στοιχείου ιστού, επιλογές ποντικιού και πληκτρολογίου και χαρακτηριστικά του στοιχείου. Επίσης, μπορούμε να ορίσουμε τη στρατηγική που ακολουθείται για τον εντοπισμό του στοιχείου στο DOM, είτε επιλέγοντας από τις προτεινόμενες επιλογές του Leapwork είτε προσθέτοντας δικές μας επιλογές. Στην Εικόνα 3.6 έχουμε επιλέξει την προτεινόμενη βασική στρατηγική, στην οποία το στοιχείο ιστού εντοπίζεται από το κείμενό του, στην προκειμένη περίπτωση η καρτέλα “Η Διπλωματική μου” εντοπίζεται από το κείμενο που εμφανίζεται στο κουμπί της καρτέλας.

Στη συνέχεια, αφού έχουν εμφανιστεί στην οθόνη τα στοιχεία που θέλουμε να ελέγξουμε, με τα blocks “Get web text” βρίσκουμε το ζητούμενο κείμενο και το συγκρίνουμε με το αναμενόμενο κείμενο. Έπειτα εμφανίζουμε στη κονσόλα αντίστοιχο μήνυμα για την επιτυχή ή όχι έκβαση του ελέγχου. Αυτός ο έλεγχος μπορεί να γίνει και με πιο απλό τρόπο, με τη χρήση του block “Find web element” όπου αν το στοιχείο ιστού βρεθεί σημαίνει ότι είναι και το ζητούμενο, δεν απαιτείται επιπλέον σύγκριση πραγματικού και αναμενόμενου κειμένου.



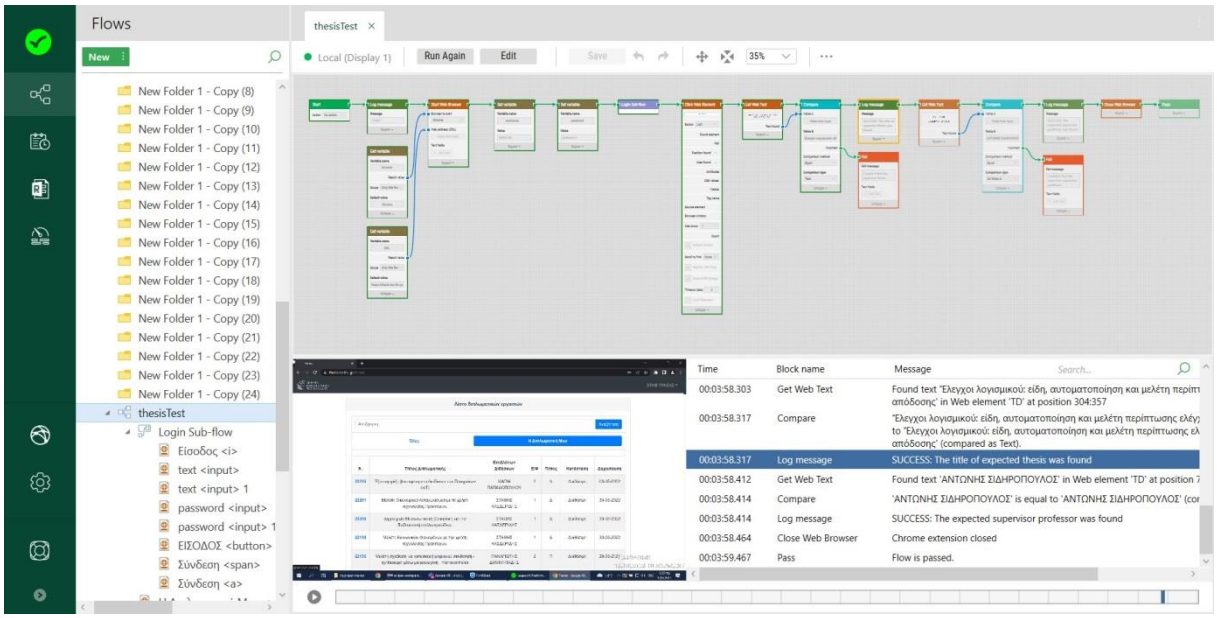
Εικόνα 3.6: Επιλογή στρατηγικής για τον εντοπισμό του στοιχείου ιστού

Τέλος, αφού κλείσουμε το πρόγραμμα περιήγησης με το block “Close web browser” ολοκληρώνεται επιτυχώς η ροή του σεναρίου ελέγχου.



Εικόνα 3.7: Τα τελευταία βήματα της ροής

Στην Εικόνα 3.8 εμφανίζεται το περιβάλλον ανάπτυξης του Leapwork και πιο συγκεκριμένα η στιγμή μετά την εκτέλεση του σεναρίου ελέγχου όπου φαίνεται και το επιτυχές αποτέλεσμα της εκτέλεσης στην κονσόλα. Επίσης, μια χρήσιμη δυνατότητα του εργαλείου για την αποσφαλμάτωση είναι το βίντεο που αποθηκεύει μετά από κάθε εκτέλεση, κάνοντας εξαιρετικά εύκολη την ανεύρεση των λαθών και την επίλυση τους.



Εικόνα 3.8: Το περιβάλλον ανάπτυξης του Leapwork

Το Leapwork παρέχει επίσης δυνατότητες προγραμματισμού εκτέλεσης των ελέγχων και αναφοράς των αποτελεσμάτων. Πιο συγκεκριμένα, μπορούμε να προσθέσουμε τα σενάρια ελέγχου σε λίστες και να ορίσουμε το πότε θα εκτελούνται αυτές οι λίστες. Αυτή είναι μια χρήσιμη δυνατότητα καθώς μας επιτρέπει να κατηγοριοποιήσουμε τα σενάρια σε λίστες ανάλογα με τη λειτουργικότητα που ελέγχουν και να εκτελούμε τις λίστες σε ξεχωριστές ημέρες της εβδομάδας. Είναι συχνή πρακτική να εκτελούνται οι έλεγχοι το βράδυ, όταν οι υπάλληλοι της εταιρίας δεν εργάζονται και το περιβάλλον πραγματοποίησης των ελέγχων είναι ελεύθερο προς χρήση. Όταν τελειώσει η εκτέλεση των σεναρίων ελέγχου που περιέχει μια λίστα, μπορούμε να ορίσουμε την αποστολή email με συνοπτικά αποτελέσματα για την έκβαση των ελέγχων.

3.5 Επίλογος

Σε αυτό το κεφάλαιο παρουσιάστηκε ένας εναλλακτικός τρόπος αυτοματοποίησης των ελέγχων λογισμικού χωρίς κώδικα, με τη χρήση εργαλείων RPA. Αρχικά, αναφέρθηκαν κάποια οικονομικά στοιχεία για την ολοένα αυξανόμενη αγορά του RPA αλλά και το γεγονός ότι τα εργαλεία RPA έχουν προσθέσει δυνατότητες αναφοράς και αποσφαλμάτωσης στα προϊόντα τους ούτως ώστε να μπορούν να χρησιμοποιηθούν αποτελεσματικότερα για την αυτοματοποίηση ελέγχων λογισμικού. Στη συνέχεια, παρουσιάστηκε αναλυτικά η υλοποίηση του σεναρίου ελέγχου στην ιστοσελίδα διπλωματικών του τμήματος ΜΠΗΣ, με τη χρήση του εργαλείου Leapwork αναφέροντας κάποιες από τις δυνατότητες του εργαλείου.

Κεφάλαιο 4ο: Μελέτη περίπτωσης ελέγχου απόδοσης

4.1 Εισαγωγή

Οι μη λειτουργικοί (non-functional) έλεγχοι λογισμικού είναι αυτοί που χρησιμοποιούνται για τον έλεγχο μη λειτουργικών πτυχών μια εφαρμογής, όπως είναι η απόδοση, η χρηστικότητα, η αξιοπιστία, η διαθεσιμότητα, κλπ. Έχουν σχεδιαστεί έτσι ώστε να ελέγχουν την ετοιμότητα ενός συστήματος σύμφωνα με μη λειτουργικές παραμέτρους, οι οποίες δεν δοκιμάζονται με τους λειτουργικούς ελέγχους. Έχουν ως στόχο να συλλέξουν και να παράξουν μετρήσεις και μετρικές για εσωτερική έρευνα και ανάπτυξη του προϊόντος, για να βελτιώσουν τη γνώση της συμπεριφοράς του προϊόντος και να συμβάλλουν στη μείωση του κινδύνου και του κόστους στο παραγωγικό περιβάλλον, που σχετίζεται με τις μη λειτουργικές παραμέτρους [27-28].

Ένα χαρακτηριστικό παράδειγμα μη λειτουργικού ελέγχου είναι ο προσδιορισμός του πλήθους των χρηστών που μπορούν να χρησιμοποιήσουν ταυτόχρονα μια εφαρμογή, χωρίς να αντιμετωπίσουν προβλήματα ή καθυστερήσεις. Αυτός είναι ο έλεγχος απόδοσης λογισμικού (performance testing).

Το τμήμα Μηχανικών Πληροφορικής και Ηλεκτρονικών Συστημάτων του Διεθνούς Πανεπιστημίου Ελλάδος, χρησιμοποιούσε μέχρι τον Μάιο του 2022 το σύστημα ανακοινώσεων του “apps”. Στα τέλη του ιδίου μήνα παρουσιάστηκε η νέα υπηρεσία ανακοινώσεων του τμήματος με την ονομασία “aBoard” η οποία αντικατέστησε το apps. Το aBoard δημιουργήθηκε εξ’ ολοκλήρου από την αρχή και βασίζεται σε ένα νέο API που στοχεύει να διορθώσει τα γνωστά προβλήματα απόδοσης και αρχιτεκτονικής που είχε το apps.

Σε αυτό το κεφάλαιο θα ασχοληθούμε με τη διαδικασία του ελέγχου της απόδοσης των δύο υπηρεσιών με τη χρήση του Apache JMeter και θα συγκρίνουμε τα αποτελέσματα για να επιβεβαιώσουμε ότι η απόδοση του aBoard είναι καλύτερη συγκριτικά με το apps.

4.2 Έλεγχοι απόδοσης

Ο έλεγχος απόδοσης είναι μια διαδικασία ελέγχου λογισμικού που χρησιμοποιείται για τον έλεγχο της ταχύτητας, του χρόνου απόκρισης, της αξιοπιστίας, της σταθερότητας, της επεκτασιμότητας και της χρήσης πόρων μιας εφαρμογής υπό συγκεκριμένο φόρτο.

Ο κύριος σκοπός του ελέγχου απόδοσης είναι ο εντοπισμός και η εξάλειψη σημείων συμφόρησης απόδοσης στην εφαρμογή. Οι τρείς κύριοι στόχοι είναι η εξαγωγή πληροφοριών σχετικά με τη ταχύτητα της εφαρμογής (αν η εφαρμογή ανταποκρίνεται γρήγορα), την επεκτασιμότητα (τον προσδιορισμό του μέγιστου αριθμού χρηστών που μπορεί να χειριστεί η εφαρμογή) και τη σταθερότητα (αν η εφαρμογή είναι σταθερή κάτω από διαφορετικό φόρτο).

Με τον έλεγχο απόδοσης θα καθοριστεί αν η υπό έλεγχο εφαρμογή ικανοποιεί τα παραπάνω κριτήρια και θα ενημερωθούν τα ενδιαφερόμενα μέρη. Ο έλεγχος απόδοσης επίσης διενεργείται σε κρίσιμες εφαρμογές (π.χ. ιατρικός εξοπλισμός) για να διασφαλιστεί ότι θα λειτουργούν για μεγάλο χρονικό διάστημα χωρίς αποκλίσεις.

Η διανομή μιας εφαρμογής στην παραγωγή με χαμηλές μετρήσεις απόδοσης λόγω ανύπαρκτου ή ελλιπούς ελέγχου απόδοσης, είναι πιθανό να επιφέρει σοβαρή ζημιά στην εταιρία λόγω της κακής φήμης [29].

4.2.1 Είδη ελέγχων απόδοσης

Μπορούν να οριστούν διαφορετικά είδη ελέγχων απόδοσης [30-31]:

Έλεγχος απόδοσης

Ο έλεγχος απόδοσης (performance testing) είναι ένας γενικός όρος που περιλαμβάνει κάθε είδους έλεγχο που επικεντρώνεται στην απόδοση/απόκριση του συστήματος σε διαφορετικά μεγέθη φόρτου.

Έλεγχος φόρτου

Ο έλεγχος φόρτου (load testing) επικεντρώνεται στην ικανότητα ενός συστήματος να χειρίζεται αυξανόμενα επίπεδα φόρτου. Αυτός ο φόρτος μπορεί να είναι ο αναμενόμενος αριθμός χρηστών που χρησιμοποιούν ταυτόχρονα το σύστημα, εκτελώντας συγκεκριμένο αριθμό αιτημάτων σε ένα καθορισμένο χρονικό διάστημα.

Έλεγχος ακραίας κατάστασης

Ο έλεγχος ακραίας κατάστασης (stress testing) χρησιμοποιείται για την εύρεση των ανώτατων ορίων χωρητικότητας ενός συστήματος. Έτσι προσδιορίζεται το σημείο στο οποίο το σύστημα θα αποτύχει και αναλύεται η αντίδραση του συστήματος για να επιβεβαιωθεί η ικανότητα επαναφοράς του.

Έλεγχος ξαφνικής μεταβολής

Ο έλεγχος της ξαφνικής μεταβολής του φόρτου (spike testing) που δημιουργείται από πολύ μεγάλο αριθμό χρηστών, έχει ως στόχο να διαπιστωθεί αν η απόδοση θα επηρεαστεί σημαντικά, αν το σύστημα θα αποτύχει ή αν θα είναι σε θέση να διαχειριστεί τις μεγάλες αλλαγές στο φόρτο.

Έλεγχος διαμόρφωσης

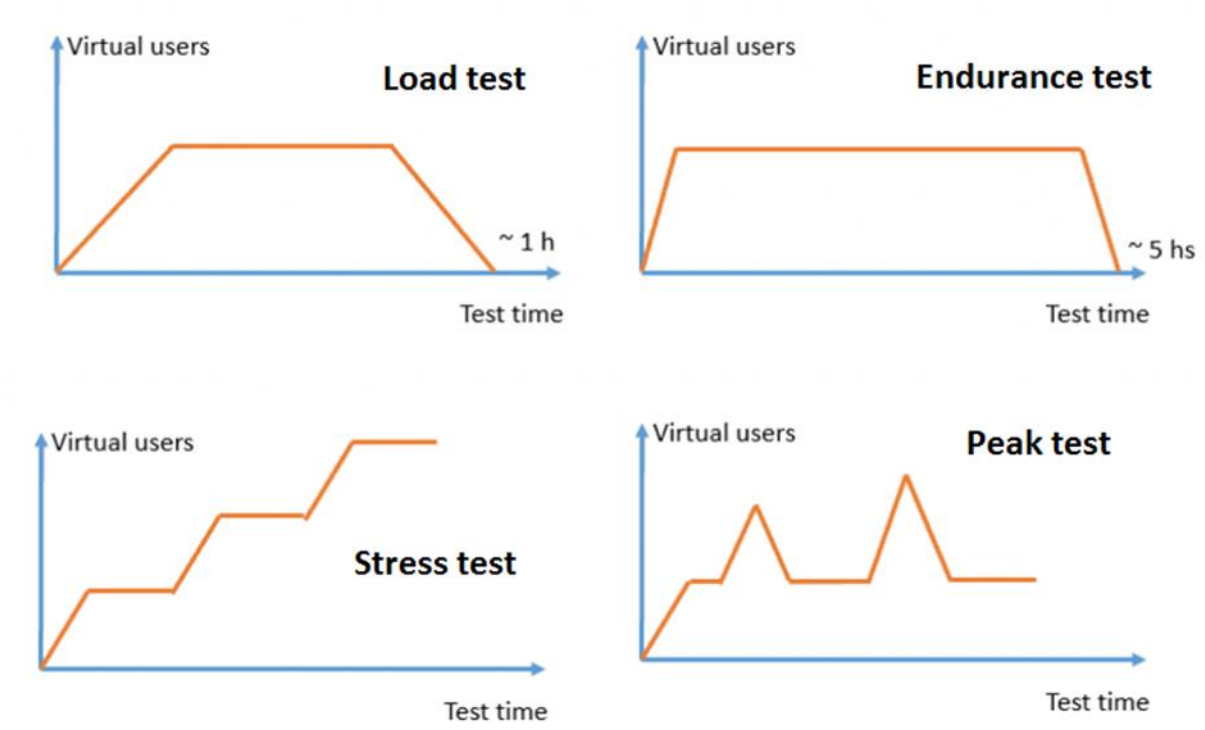
Στον έλεγχο διαμόρφωσης (configuration testing), αντί να ελέγχεται η απόδοση με τη δημιουργία φόρτου, δημιουργούνται έλεγχοι για τον προσδιορισμό των επιπτώσεων που θα έχουν οι αλλαγές διαμόρφωσης στοιχείων του συστήματος στην απόδοση και συμπεριφορά του συστήματος. Ένα τυπικό παράδειγμα είναι ο πειραματισμός με διάφορες μεθόδους εξισορρόπησης φορτίου (load-balancing).

Έλεγχος αντοχής

Ο έλεγχος αντοχής (endurance testing ή soak testing) γίνεται συνήθως για να καθοριστεί αν το σύστημα μπορεί να αντέξει το συνεχές αναμενόμενο φόρτο. Κατά τη διάρκεια του ελέγχου, παρακολουθείται η χρήση της μνήμης για τον εντοπισμό πιθανών διαρροών (memory leaks) και οι συνδέσεις με βάσεις δεδομένων, ώστε να διασφαλιστεί ότι δεν υπάρχουν προβλήματα χωρητικότητας πόρων που μπορεί να επηρεάσουν την απόδοση. Είναι σημαντικό να εξασφαλιστεί ότι η ρυθμαπόδοση (throughput) και οι χρόνοι απόκρισης μετά από κάποια μακρά περίοδο συνεχούς δραστηριότητας είναι εξίσου καλοί όσο στην αρχή του ελέγχου.

Έλεγχος χωρητικότητας

Με τον έλεγχο χωρητικότητας (capacity testing) προσδιορίζεται το πλήθος των χρηστών ή συναλλαγών που το σύστημα μπορεί να υποστηρίζει και παράλληλα να ικανοποιεί του καθορισμένους στόχους απόδοσης. Συνεπώς χρησιμοποιείται για την εύρεση της μέγιστης χωρητικότητας κάτω από την οποία το σύστημα θα λειτουργεί σύμφωνα με τις απαιτούμενες προδιαγραφές ή τη συμφωνία επιπέδου υπηρεσιών (Service Level Agreement – SLA).



Σχήμα 4.1: Η γραφική απεικόνιση των δημοφιλέστερων ειδών ελέγχου απόδοσης, Πηγή: [30]

4.2.2 Απαιτούμενες δραστηριότητες για τον έλεγχο απόδοσης

Οι βασικές δραστηριότητες που απαιτούνται για την πραγματοποίηση του ελέγχου απόδοσης είναι κατά σειρά [31-32]:

Προσδιορισμός του περιβάλλοντος ελέγχου

Σε αυτό το αρχικό βήμα περιλαμβάνεται ο προσδιορισμός του υλικού, του λογισμικού, των δεδομένων ελέγχου (test data) και των εργαλείων που είναι διαθέσιμα για το σχεδιασμό των ελέγχων.

Προσδιορισμός των μετρικών απόδοσης

Σε αυτό το βήμα προσδιορίζονται οι μετρικές απόδοσης που θα χρησιμοποιηθούν ως κριτήρια αποδοχής και ισοδυναμούν με χαρακτηριστικά που οι χρήστες εκλαμβάνουν ως καλή απόδοση. Τέτοιες μετρικές είναι ο χρόνος απόκρισης (π.χ. η φόρτωση της αρχικής σελίδας να μη διαρκεί πάνω από 3 δευτ.), η ρυθμαπόδοση (π.χ. το σύστημα να μπορεί να υποστηρίξει 2000 χρήστες σε ώρες αιχμής) και η εκμετάλλευση πόρων (π.χ. η χρήση του επεξεργαστή να μην υπερβαίνει το 70%).

Σχεδιασμός των σεναρίων ελέγχου απόδοσης

Στη φάση του σχεδιασμού των σεναρίων, κύριος στόχος είναι η προσομοίωση πραγματικών καταστάσεων προκειμένου να παρέχονται αξιόπιστα δεδομένα για τη χρήση του συστήματος.

Διαμόρφωση του περιβάλλοντος ελέγχου

Προετοιμασία του περιβάλλοντος ελέγχου σε επίπεδο υλικού, εργαλείων που θα χρησιμοποιηθούν για την παρακολούθηση των πόρων κατά τη διάρκεια των ελέγχων και δικτύου.

Υλοποίηση των σεναρίων ελέγχου που σχεδιάστηκαν

Σε αυτή τη φάση υλοποιούνται στο επιλεγμένο λογισμικό παραγωγής φόρτου, τα ρεαλιστικά σενάρια που προσομοιάζουν τις κινήσεις ενός χρήστη στην εφαρμογή. Επίσης, απαιτείται μια διαδικασία για τη δημιουργία των δεδομένων ελέγχου (test data), τα οποία αντιπροσωπεύουν τα δεδομένα του περιβάλλοντος παραγωγής σε όγκο και τύπο, ώστε να μπορεί να προσομοιωθεί η πραγματική χρήση τους.

Εκτέλεση των ελέγχων

Η εκτέλεση των ελέγχων πραγματοποιείται όταν διεξάγεται ο έλεγχος της απόδοσης με τη χρήση εργαλείων ελέγχου απόδοσης. Αυτή η φάση περιλαμβάνει την επικύρωση για άλλη μια φορά του περιβάλλοντος ελέγχου, τον συντονισμό της εκτέλεσης και την παρακολούθηση κατά τη διάρκεια της εκτέλεσης όλων των ορατών δεδομένων, την αποθήκευση των αποτελεσμάτων.

Ανάλυση αποτελεσμάτων, αναφορά αποτελεσμάτων και επανεκτέλεση των ελέγχων

Το τελευταίο βήμα περιλαμβάνει την ανάλυση και δημιουργία αναφοράς για τα αποτελέσματα των ελέγχων. Τα αποτελέσματα παρέχονται στους ενδιαφερόμενους σε μια συνοπτική έκθεση ελέγχου. Εκφράζονται με τη χρήση μετρικών και διαγραμμάτων για να είναι πιο εύκολα κατανοητά. Αν παρουσιαστούν αστοχίες ή σημεία που χρήζουν βελτίωσης, τα σενάρια ελέγχου πρέπει να επανεξεταστούν και να εκτελεστούν πάλι οι έλεγχοι.

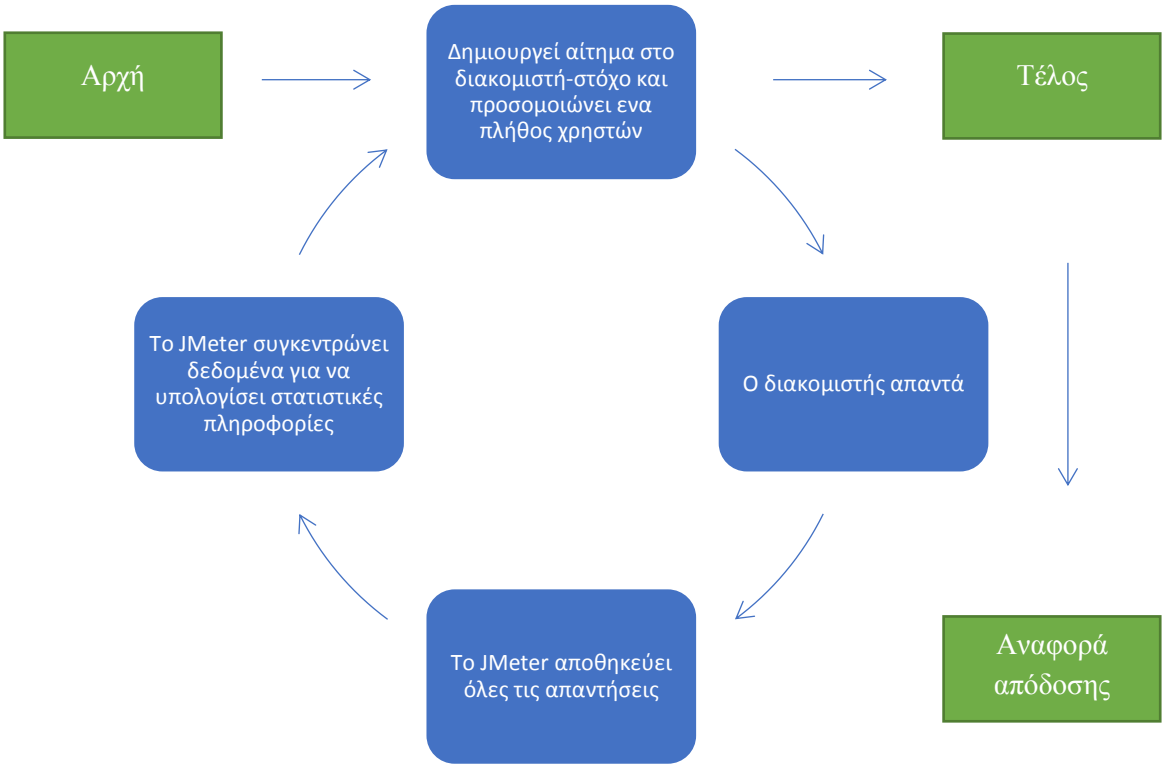
4.3 Το Apache JMeter

Το JMeter είναι ένα λογισμικό ανοιχτού κώδικα σε Java που έχει αναπτυχθεί από το “The Apache Software Foundation”. Η πρώτη του έκδοση κυκλοφόρησε το 1998, όμως λαμβάνει συχνά ενημερώσεις για να προσαρμόζεται στις τεχνολογικές εξελίξεις. Μπορεί να χρησιμοποιηθεί για την ανάλυση και τη μέτρηση της απόδοσης σε ένα πλήθος υπηρεσιών, με έμφαση στις διαδικτυακές υπηρεσίες. Επίσης μπορεί να χρησιμοποιηθεί ως εργαλείο ελέγχου μονάδας (unit test) για JDBC συνδέσεις βάσεων δεδομένων, FTP, LDAP, JMS, HTTP και TCP συνδέσεων.

Ο τρόπος που λειτουργεί είναι απλός, όπως φαίνεται και στο Σχήμα 4.2. Προσομοιώνει ένα πλήθος χρηστών στέλνοντας αιτήματα στο διακομιστή-στόχο και δημιουργεί στατιστικά με τη χρήση διαγραμμάτων για την απόδοση του διακομιστή. Εκτός των λειτουργιών που προσφέρει με την αρχική εγκατάσταση του, βασίζεται και σε πρόσθετα (plugins) για τη προσθήκη επιπλέον λειτουργικότητας.

Προσφέρει γραφικό περιβάλλον για τη δημιουργία, την καταγραφή και την αποσφαλμάτωση των σεναρίων ελέγχου και λειτουργία γραμμής εντολών που είναι και η προτεινόμενη για την τελική εκτέλεση του ελέγχου χωρίς να επηρεάζεται η απόδοση από το γραφικό περιβάλλον.

Επιπλέον, έχει και λειτουργία δημιουργίας ολοκληρωμένης αναφοράς των αποτελεσμάτων του ελέγχου απόδοσης σε μορφή HTML, παρέχοντας αναλυτικά διαγράμματα και στατιστικά.



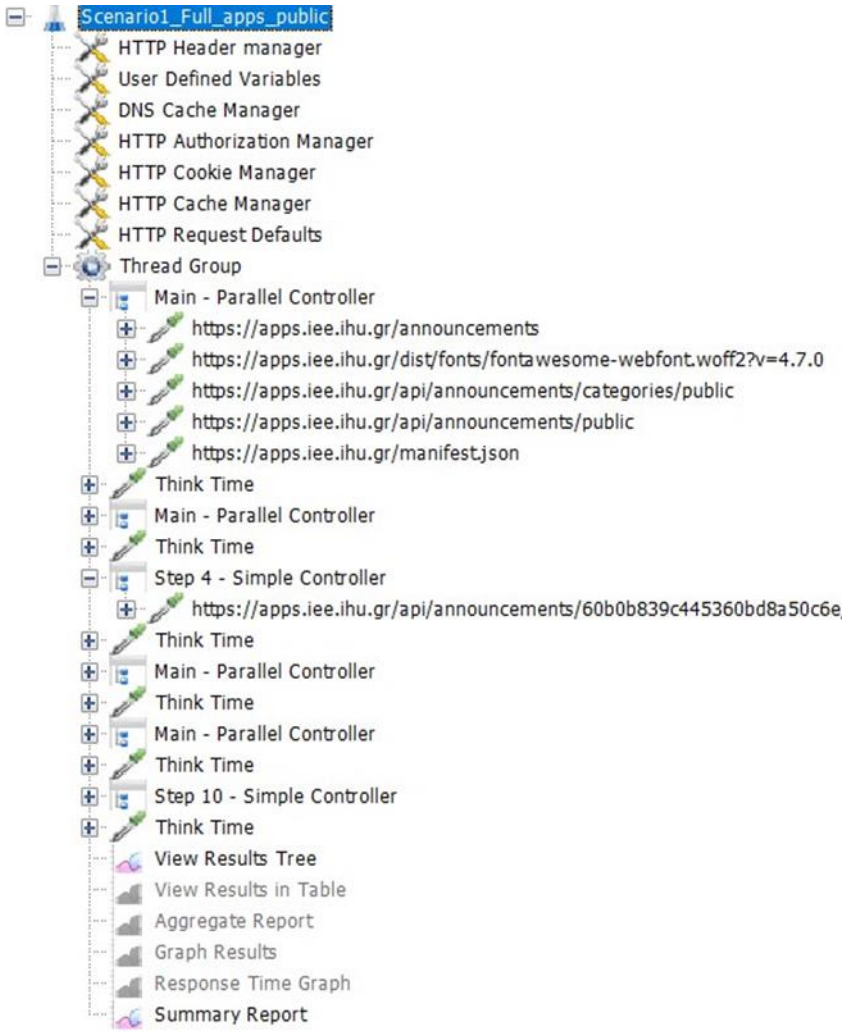
Σχήμα 4.2: Ο τρόπος λειτουργίας του JMeter

4.3.1 Στοιχεία ενός σχεδίου ελέγχου

Ένα ολοκληρωμένο σχέδιο ελέγχου είναι απαραίτητο να περιέχει τα παρακάτω στοιχεία [33]:

Σχέδιο ελέγχου

Το σχέδιο ελέγχου (test plan) περιέχει όλα τα βήματα που θα εκτελέσει το JMeter. Ένα ολοκληρωμένο σχέδιο ελέγχου περιλαμβάνει μια ή περισσότερες ομάδες νημάτων, λογικούς ελεγκτές, ελεγκτές παραγωγής δειγμάτων, ακροατές, χρονοδιακόπτες, ισχυρισμούς και στοιχεία διαμόρφωσης.

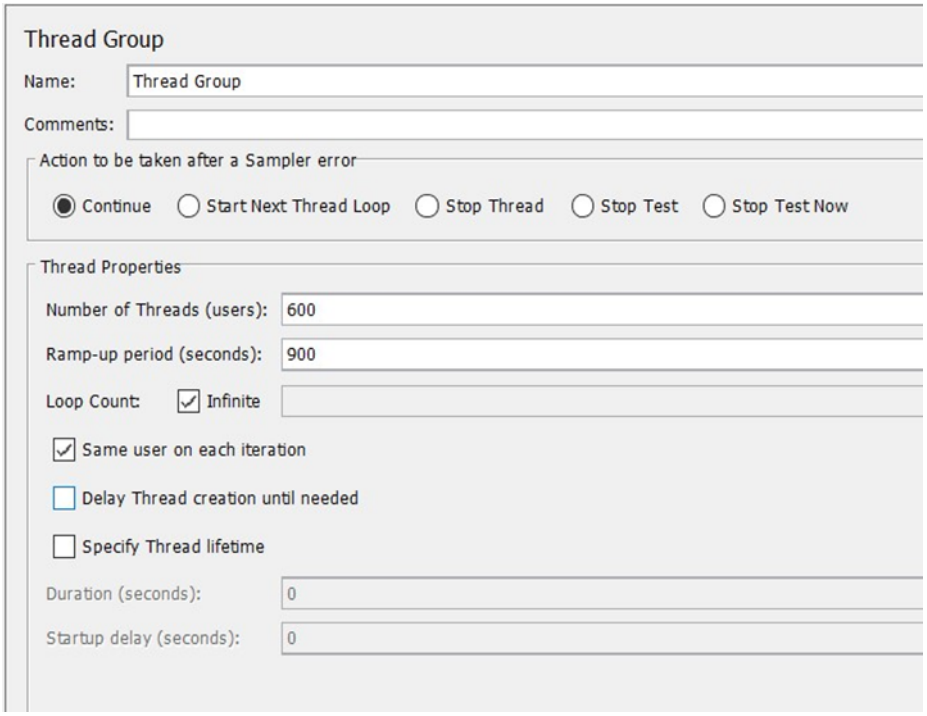


Εικόνα 4.1: Το σχέδιο ελέγχου του apps

Ομάδα νημάτων

Τα στοιχεία που περιέχονται στην ομάδα νημάτων (thread group) είναι το σημείο έναρξης κάθε σχεδίου ελέγχου. Όλοι οι ελεγκτές και οι δειγματολήπτες πρέπει να βρίσκονται σε κάποια ομάδα νημάτων. Άλλα στοιχεία, όπως οι ακροατές, μπορούν να τοποθετηθούν στο κάτω μέρος του σχεδίου ελέγχου, όπου σε αυτή την περίπτωση θα εφαρμοστούν σε όλες τις ομάδες νημάτων.

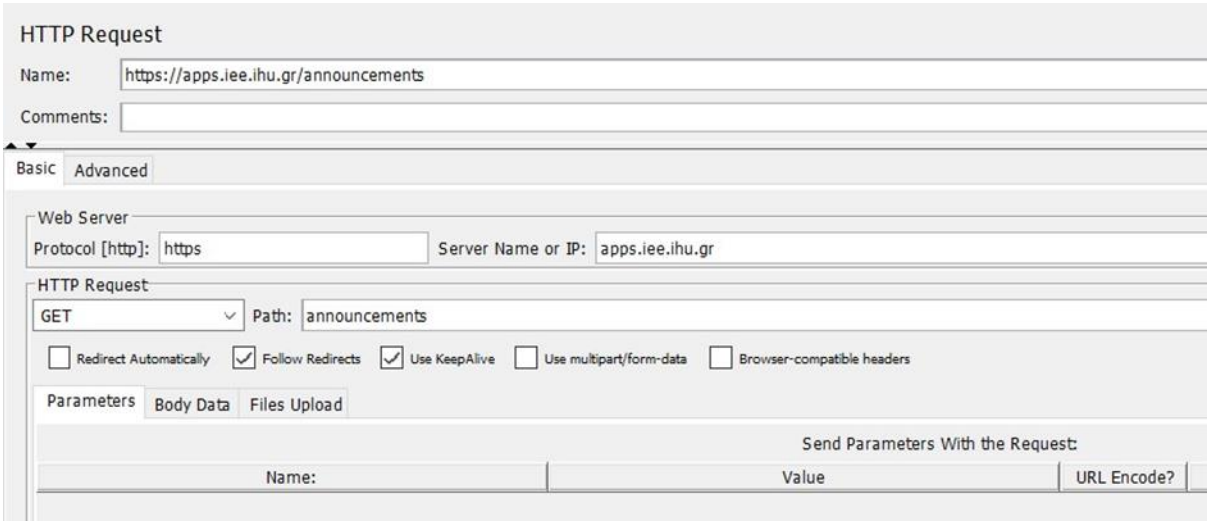
Το στοιχείο της ομάδας νημάτων ελέγχει τον αριθμό των νημάτων τα οποία θα χρησιμοποιήσει το JMeter για την εκτέλεση του ελέγχου και αντιπροσωπεύει ουσιαστικά τον αριθμό των χρηστών. Τα στοιχεία ελέγχου για μια ομάδα νημάτων επιτρέπουν τον καθορισμό του αριθμού των νημάτων, της περιόδου ανόδου και του αριθμού των επαναλήψεων για την εκτέλεση του ελέγχου.



Εικόνα 4.2: Τα στοιχεία ελέγχου για μια ομάδα νημάτων

Ελεγκτές

Το JMeter έχει δύο είδη ελεγκτών (controllers), τους δειγματολήπτες (samplers) και τους λογικούς ελεγκτές. Οι δειγματολήπτες δίνουν την εντολή στο JMeter να στείλει αιτήματα σε έναν διακομιστή και να περιμένει για μια απάντηση. Περιλαμβάνει δειγματολήπτες για HTTP request, FTP request, JDBC request, LDAP request, TCP request, Mail request, κλπ. Μια χρήσιμη ρύθμιση στον δειγματολήπτη HTTP request, είναι η επιλογή “Retrieve embedded resources” για την αυτόματη μεταφόρτωση όλων των αρχείων που συνοδεύουν μια HTML ιστοσελίδα.



Εικόνα 4.3: Δειγματολήπτης για HTTP request

Οι λογικοί ελεγκτές επιτρέπουν τον καθορισμό της λογικής που χρησιμοποιεί το JMeter για να αποφασίσει το πότε θα στείλει τα αιτήματα και καθορίζουν τη σειρά με την οποία εκτελούνται οι δειγματολήπτες. Περιλαμβάνει τον απλό ελεγκτή, τον ελεγκτή βρόγχου, τον if controller, τον while controller, τον switch controller κλπ. Εκτός των βασικών ελεγκτών, μπορούν να προστεθούν και επιπλέον ελεγκτές μέσω πρόσθετων του JMeter, για παράδειγμα ο παράλληλος ελεγκτής είναι ιδιαίτερα χρήσιμος για την παράλληλη αποστολή αιτημάτων σε έναν διακομιστή.

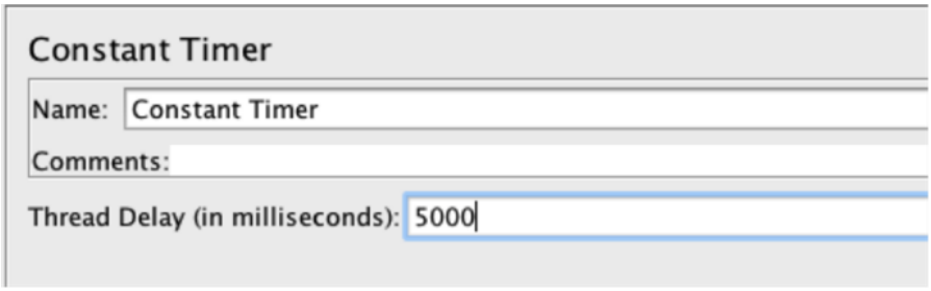
Ακροατές

Οι ακροατές (listeners) παρέχουν πρόσβαση στις πληροφορίες που συγκεντρώνει το JMeter για τα σενάρια ελέγχου κατά τη διάρκεια εκτέλεσης τους. Πέρα από τη δυνατότητα συγκέντρωσης των αποτελεσμάτων του ελέγχου, οι ελεγκτές παρέχουν δυνατότητες προβολής, αποθήκευσης και ανάγνωσης αποθηκευμένων αποτελεσμάτων. Τα αποτελέσματα μπορούν να αποθηκευτούν σε μορφή JTL (JMeter Text Logs) και μπορεί να είναι XML ή CSV, με τη δυνατότητα να επιλεχθούν τα επιθυμητά πεδία που θέλουμε να αποθηκευτούν. Οι ακροατές μπορεί να χρησιμοποιήσουν μεγάλο μέγεθος μνήμης αν υπάρχουν πολλοί δειγματολήπτες, γι' αυτό και συστήνεται η χρήση του JMeter από τη γραμμή εντολών κατά την εκτέλεση του ελέγχου και η απενεργοποίηση των ακροατών. Με τη χρήση του flag –l κατά την εκτέλεση του σχεδίου ελέγχου, προστίθεται ένας ακροατής για την αποθήκευση των αποτελεσμάτων στο αρχείο που θα ορίσουμε.

Περιλαμβάνει ακροατές όπως οι Graph Results, Summary Report, View Results Tree, View Results in Table, κλπ.

Χρονοδιακόπτες

Το JMeter από προεπιλογή εκτελεί τους δειγματολήπτες σε μια ομάδα νημάτων, με τη σειρά και χωρίς παύση. Για τη προσομοίωση ρεαλιστικών σεναρίων είναι απαραίτητη η χρήση χρονοδιακοπτών (timers) ανάμεσα στους δειγματολήπτες, έτσι ώστε η αποστολή αιτημάτων στο διακομιστή να γίνεται με παύσεις, όπως θα γινόταν στη περίπτωση όπου ένας πραγματικός χρήστης χρησιμοποιούσε την εφαρμογή. Περιλαμβάνει τους Constant Timer, Uniform Random Timer, Gaussian Random Timer, κλπ.



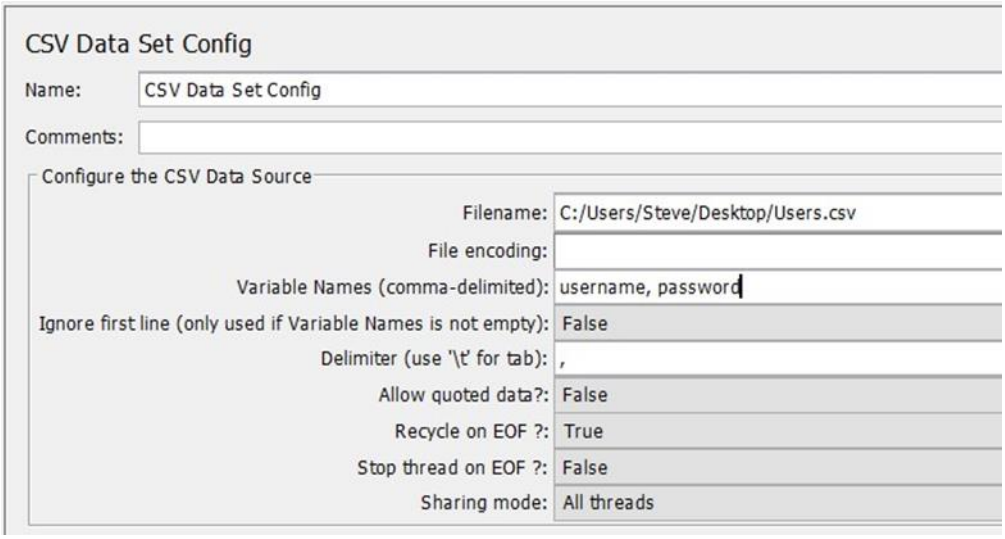
Εικόνα 4.4: Χρήση του Constant Timer για παύση 5 δευτερολέπτων

Ισχυρισμοί

Οι ισχυρισμοί (assertions) μας επιτρέπουν να ελέγχουμε τις απαντήσεις που λαμβάνονται από τον διακομιστή. Με τη χρήση τους μπορούμε να επιβεβαιώσουμε ότι η εφαρμογή λειτουργεί με τον αναμενόμενο τρόπο. Για παράδειγμα μπορούμε να επιβεβαιώσουμε ότι η απάντηση σε ένα αίτημα περιέχει συγκεκριμένο κείμενο, με τη χρήση κανονικών εκφράσεων. Αν το αποτέλεσμα του ισχυρισμού είναι αρνητικό, δηλαδή το ζητούμενο κείμενο δεν βρεθεί στην απάντηση του διακομιστή, το JMeter θα χαρακτηρίσει το αίτημα ως αποτυχημένο.

Στοιχεία διαμόρφωσης

Τα στοιχεία διαμόρφωσης, όπως φαίνονται στην Εικόνα 4.1, περιλαμβάνουν τα HTTP Cookie Manager, HTTP Cache Manager, HTTP Header Manager, CSV Data Set Config κλπ. Για παράδειγμα, αν θέλουμε να δημιουργήσουμε ένα σενάριο ελέγχου όπου 100 μοναδικοί χρήστες θα κάνουν login σε μια εφαρμογή, μπορούμε με τη χρήση του στοιχείου CSV Data Set Config να διαβάσουμε τα δεδομένα (username, password) των 100 χρηστών από ένα αρχείο CSV και με τη χρήση μεταβλητών να τα χρησιμοποιήσουμε στο σχέδιο ελέγχου.



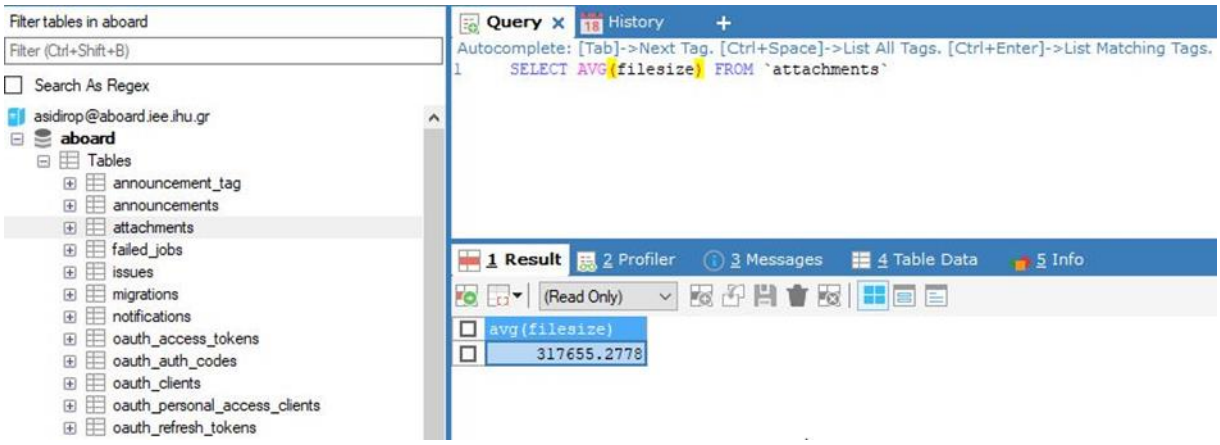
Εικόνα 4.5: Το στοιχείο διαμόρφωσης CSV Data Set Config

4.4 Σενάρια για τον έλεγχο απόδοσης του apps-aBoard

Για την αξιολόγηση της απόδοσης των δύο υπηρεσιών ανακοινώσεων, δημιουργήθηκαν δύο διαφορετικά σενάρια ελέγχου.

Το πρώτο σενάριο προσομοιάζει την περιήγηση του χρήστη στη σελίδα ανακοινώσεων, με ενέργειες που περιλαμβάνουν το άνοιγμα ανακοινώσεων, τη μεταφόρτωση συνημμένων αρχείων των ανακοινώσεων, την περιήγηση σε διαφορετική σελίδα των ανακοινώσεων και τη χρήση της λειτουργίας αναζήτησης ανακοινώσεων.

Το δεύτερο σενάριο προσομοιάζει τη μεταφόρτωση συνημμένου αρχείου pdf δύο ανακοινώσεων από μεγάλο αριθμό χρηστών σε μικρό χρονικό διάστημα. Αυτό το σενάριο μπορεί να προκύψει όταν δημοσιευθεί μια σημαντική ανακοίνωση και οι χρήστες αφού λάβουν ειδοποίηση με email για την ανακοίνωση ή ενημερωθούν από τα μέσα κοινωνικής δικτύωσης του τμήματος ΜΠΗΣ, σπεύσουν να την αναγνώσουν σε σύντομο χρονικό διάστημα. Τα δύο αρχεία είναι μικρού (199KB) και μεσαίου (648KB) μεγέθους και επιλέχθηκε αυτό το μέγεθος των αρχείων μετά την εύρεση του μέσου όρου του μεγέθους (317KB) όλων των αρχείων στην υπηρεσία ανακοινώσεων aBoard. (Εικόνα 4.6).



Εικόνα 4.6: Εύρεση του μέσου όρου των αρχείων στην υπηρεσία ανακοινώσεων aBoard

Αναλυτική περιγραφή Σεναρίου 1:

Ο χρήστης επισκέπτεται την κεντρική σελίδα ανακοινώσεων και επιλέγει την πρώτη ανακοίνωση. Μετά την πάροδο 30 δευτ. επιστρέφει πίσω και επιλέγει μια δεύτερη ανακοίνωση από την κεντρική σελίδα, η οποία περιέχει συνημμένο αρχείο μικρού μεγέθους (199KB). Αφού διαβάσει την ανακοίνωση και μεταφορτώσει το συνημμένο αρχείο, μετά την πάροδο 30 δευτ. από την μεταφόρτωση επιστρέφει πίσω στην κεντρική σελίδα.

Στη συνέχεια επιλέγει να μεταβεί στη δεύτερη σελίδα ανακοινώσεων, όπου θα επιλέξει μια ακόμα ανακοίνωση. Όταν παρέλθει χρόνος 30 δευτ. από το άνοιγμα της ανακοίνωσης, ο χρήστης επιστρέφει πίσω και αυτή τη φορά επιλέγει να χρησιμοποιήσει τη λειτουργία αναζήτησης για να εντοπίσει μια συγκεκριμένη ανακοίνωση με συνημμένο αρχείο μεσαίου μεγέθους (648KB). Αφού κάνει την αναζήτηση, επιλέγει το πρώτο αποτέλεσμα και μεταφορτώνει το συνημμένο αρχείο. Μετά την πάροδο 30 δευτ. ολοκληρώνεται η αλληλουχία βημάτων για αυτό το σενάριο.

Αναλυτική περιγραφή Σεναρίου 2:

Ο χρήστης χρησιμοποιεί τη διεύθυνση ενός pdf αρχείου μικρού μεγέθους (199KB) και μετά την πάροδο 30 δευτ. χρησιμοποιεί την διεύθυνση ενός δεύτερου pdf αρχείου μεσαίου μεγέθους (648KB). Αφού παρέλθουν 30 δευτ. ολοκληρώνεται η αλληλουχία βημάτων για αυτό το σενάριο.

4.5 Δημιουργία των σεναρίων ελέγχου

Τα δύο συστήματα ανακοινώσεων βασίζονται σε διαφορετικά API, συνεπώς οι κλήσεις που πραγματοποιούνται προς τον διακομιστή για τις ίδιες ενέργειες, είναι διαφορετικές. Σκοπός αυτού του ελέγχου απόδοσης είναι η σύγκριση της απόδοσης των δύο συστημάτων. Ως εκ τούτου, υλοποιήσαμε το ίδιο σενάριο ελέγχου και για τα δύο συστήματα, χρησιμοποιώντας σε κάθε περίπτωση το API κάθε συστήματος.

Στον Πίνακα 4.1 φαίνονται οι διαφορές του API των δύο υπηρεσιών και δίπλα σε κάθε βήμα ο χρόνος αναμονής για την εκτέλεση του επόμενου βήματος. Ο συνολικός χρόνος εκτέλεσης είναι ίδιος και για τα δύο σενάρια ελέγχου.

Πίνακας 4.1: Οι διαφορές του API των δύο υπηρεσιών

apps		**aBoard**	
1) Κεντρική σελίδα	30"	1) Κεντρική σελίδα	10"
		2) Πρώτη ανακοίνωση	20"
3) Κεντρική σελίδα	15"	3) Κεντρική σελίδα	5"
		4a) Δεύτερη ανακοίνωση	10"
4b) Μεταφόρτωση μικρού pdf	20"	4b) Μεταφόρτωση μικρού pdf	20"
5) Κεντρική σελίδα	30"	5) Κεντρική σελίδα	5"
		6) Δεύτερη σελίδα ανακοινώσεων	5"
		7)Τρίτη ανακοίνωση	20"
8) Κεντρική σελίδα	25"	8) Κεντρική σελίδα	5"
		9) Αναζήτηση τέταρτης ανακοίνωσης	10"
		10a) Τέταρτη ανακοίνωση	10"
10b) Μεταφόρτωση μεσαίου pdf	30"	10b) Μεταφόρτωση μεσαίου pdf	30"

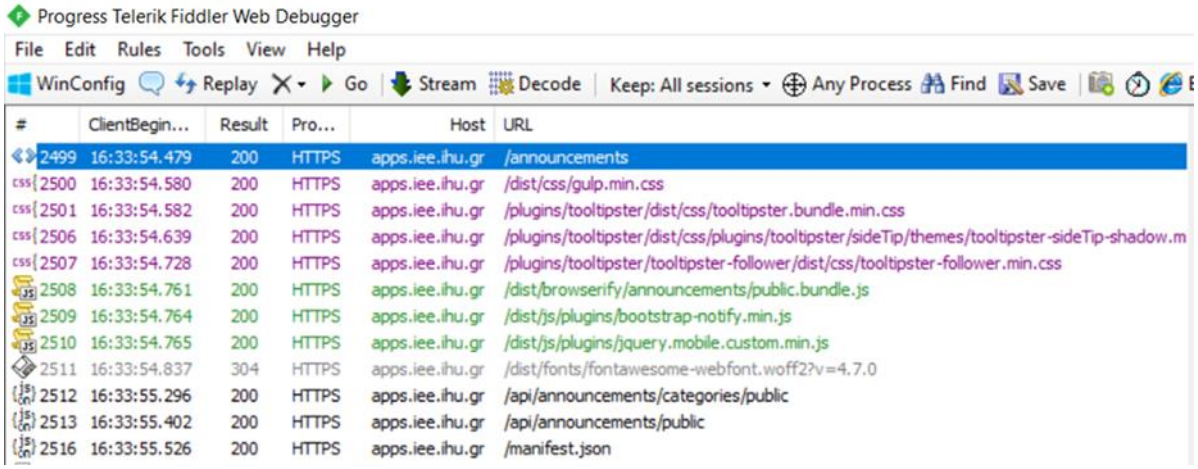
Μια σημαντική διαφορά του apps είναι ότι η κλήση GET announcements/public φορτώνει μεγάλο πλήθος των ανακοινώσεων (θεωρητικά όλες) του συστήματος, ενώ στο aBoard φορτώνει τις ανακοινώσεις μόνο μιας σελίδας. Για το λόγο αυτό, όπως φαίνεται στον Πίνακα 4.1 στα βήματα 2, 4α, 6, 7, 9 και 10α στο apps δεν γίνεται καμία κλήση προς το διακομιστή επειδή οι ανακοινώσεις έχουν ήδη φορτωθεί από το βήμα στο οποίο επισκεπτόμαστε την κεντρική σελίδα του συστήματος ανακοινώσεων.

Η διαδικασία που ακολουθήσαμε για να δημιουργήσουμε τα σενάρια στην έκδοση 5.4.1 του JMeter είναι η εξής.

Αρχικά έπρεπε να καταγράψουμε την κίνηση στο δίκτυο κατά τη διάρκεια εκτέλεσης του σεναρίου με χειροκίνητο τρόπο. Για το σκοπό αυτό χρησιμοποιήσαμε το Telerik Fiddler, ένα χρήσιμο εργαλείο για τον εντοπισμό σφαλμάτων σε εφαρμογές ιστού, το οποίο καταγράφει τις κινήσεις στο δίκτυο και μας δίνει τη δυνατότητα να επιθεωρήσουμε τα εισερχόμενα και εξερχόμενα δεδομένα.

Αφού εκτελέσαμε όλα τα βήματα του σεναρίου και για τα δύο συστήματα ανακοινώσεων, κάναμε εξαγωγή της καταγραφής των κλήσεων που αφορούν κάθε βήμα ξεχωριστά, σε αρχείο της μορφής .har (HTTP Archive format). Στη συνέχεια μετατρέψαμε τα αρχεία .har σε μορφή .jmx με τη χρήση

του μετατροπέα στην ιστοσελίδα https://converter.blazemeter.com/. Το πρότυπο αρχείου .jmx χρησιμοποιείται από το JMeter για την αποθήκευση των project του. Συνεπώς, τώρα μπορούμε να εισάγουμε στο JMeter τις κλήσεις που καταγράψαμε προηγουμένως με το Fiddler και να δημιουργήσουμε τα βήματα του σεναρίου.



Εικόνα 4.7: Η καταγραφή του δικτύου κατά τη χρήση του apps

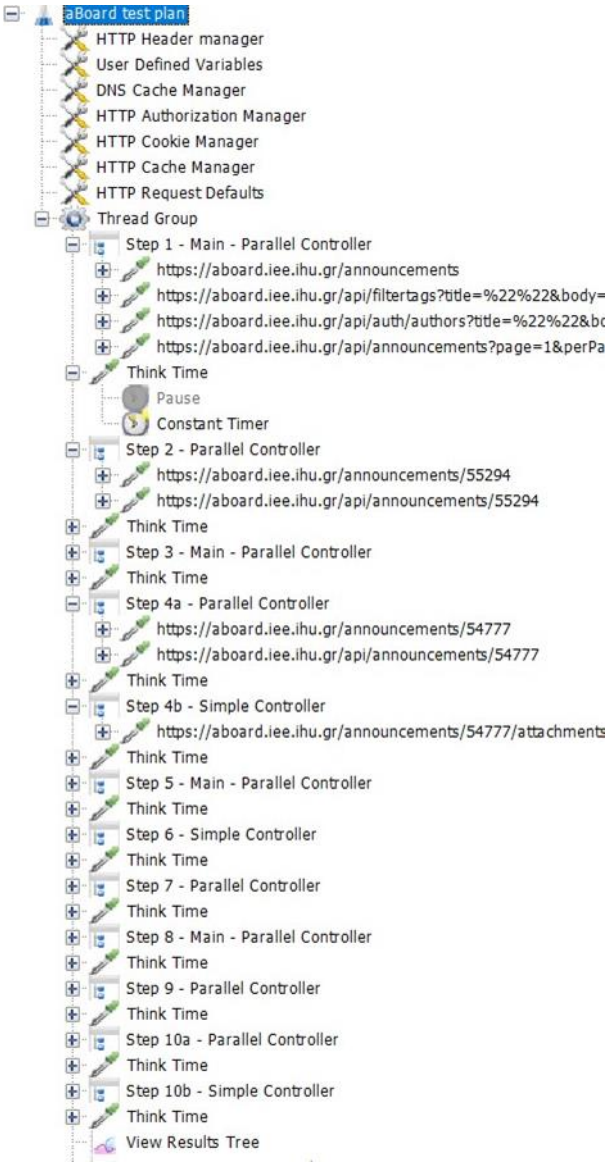
Προσθέσαμε ελεγκτές για κάθε βήμα του σεναρίου, ακολουθώντας τα βήματα του Πίνακα 4.1. Στην Εικόνα 4.1 φαίνεται η δομή του σεναρίου για το apps, όπου για το βήμα της κεντρικής σελίδας ανακοινώσεων χρησιμοποιούμε έναν παράλληλο ελεγκτή στον οποίο προσθέτουμε δειγματολήπτες HTTP request για όλες τις κλήσεις που πραγματοποιούνται παράλληλα όταν ένας χρήστης επισκέπτεται την κεντρική σελίδα.

Για τα βήματα 4β και 10β, στα οποία γίνεται η μεταφόρτωση των δύο αρχείων, χρησιμοποιούμε απλό ελεγκτή στον οποίο προσθέτουμε έναν δειγματολήπτη HTTP Request. Ενδιάμεσα των βημάτων έχουμε προσθέσει δειγματολήπτες Flow Action Control οι οποίοι περιέχουν έναν χρονοδιακόπτη Constant Timer ανάλογα με το χρόνο αναμονής που έχει κάθε βήμα, όπως φαίνεται στον Πίνακα 4.1.

Επίσης προσθέσαμε τα στοιχεία διαμόρφωσης HTTP Cookie Manager και HTTP Cache Manager με ενεργοποιημένη την επιλογή καθαρισμού των cookies/cache μετά από κάθε επανάληψη.

Για τις δοκιμαστικές εκτελέσεις του σχεδίου ελέγχου, έχουμε προσθέσει ακροατές ώστε να βλέπουμε τη στιγμή της εκτέλεσης, δεδομένα για την διενέργεια του ελέγχου. Όλοι οι ακροατές απενεργοποιήθηκαν πριν την τελική εκτέλεση του ελέγχου, καθώς επηρεάζουν την απόδοση του συστήματος διενέργειας του ελέγχου.

Το σχέδιο ελέγχου για το aBoard (Εικόνα 4.8) ακολουθεί αντίστοιχα τα βήματα του Πίνακα 4.1.

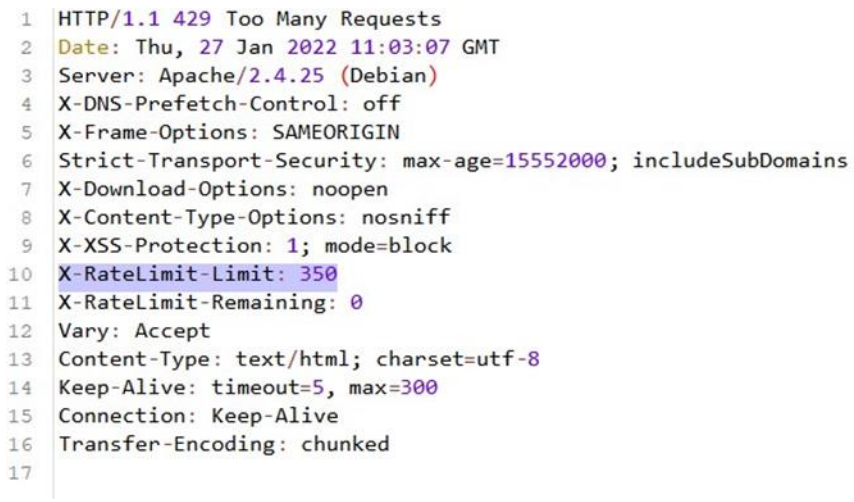


Εικόνα 4.8: Το σχέδιο ελέγχου για το aBoard

4.6 Εκτέλεση των σεναρίων

Πριν την τελική εκτέλεση των σχεδίων ελέγχου, πραγματοποιήσαμε δοκιμαστικές εκτελέσεις και για τα δύο συστήματα, ώστε να επιβεβαιώσουμε την ορθή λειτουργία τους.

Αρχικά, ανακαλύψαμε ότι οι διακομιστές και των δύο συστημάτων είχαν ενεργοποιημένο rate limiter, για τον περιορισμό επιθέσεων τύπου DOS. Συνεπώς, μετά από κάποιο χρονικό διάστημα εκτέλεσης του ελέγχου όπου ο αριθμός των αιτημάτων αυξανόταν, λαμβάναμε απαντήσεις “429 Too many requests” (Εικόνα 4.7). Αφού απενεργοποιήσαμε τον rate limiter και στους δύο διακομιστές, εκτελέσαμε και πάλι του ελέγχους για να επιβεβαιώσουμε την αλλαγή.



Εικόνα 4.9: Η απάντηση 429 λόγω του rate limiter

Το επόμενο βήμα ήταν να ελέγξουμε τα χαρακτηριστικά του υλικού κάθε διακομιστή ώστε τα αποτελέσματα του ελέγχου να μην επηρεάζονται από τη μεγαλύτερη επεξεργαστική ισχύ τους ενός ή τη μεγαλύτερη μνήμη RAM. Οι δύο διακομιστές φιλοξενούνται σε εικονικές μηχανές οπότε ήταν εύκολο να επιλέξουμε ίδιες ρυθμίσεις επεξεργαστικής ισχύος και μνήμης. Ο υπολογιστής που διενεργούσε τους ελέγχους είχε μνήμη 16GB και επεξεργαστή i5-8250U 1.6GHz.

Για την τελική εκτέλεση των σεναρίων επιλέξαμε την απευθείας σύνδεση του υπολογιστή που θα εκτελούσε τον έλεγχο, με το διακομιστή μέσω καλωδίου Ethernet. Ο διακομιστής φιλοξενείται σε ειδικό χώρο στο τμήμα ΜΠΗΣ (Εικόνα 4.10). Η εκτέλεση των ελέγχων έγινε από τη γραμμή εντολών με την παρακάτω εντολή χωρίς τη χρήση του γραφικού περιβάλλοντος του JMeter, με αποθήκευση των αποτελεσμάτων σε αρχεία της μορφής .jtl.

```
jmeter -n -t my_test.jmx -l log.jtl
```

Η εντολή έχει τις παρακάτω παραμέτρους:

-n: εκτέλεση σε μη γραφικό περιβάλλον

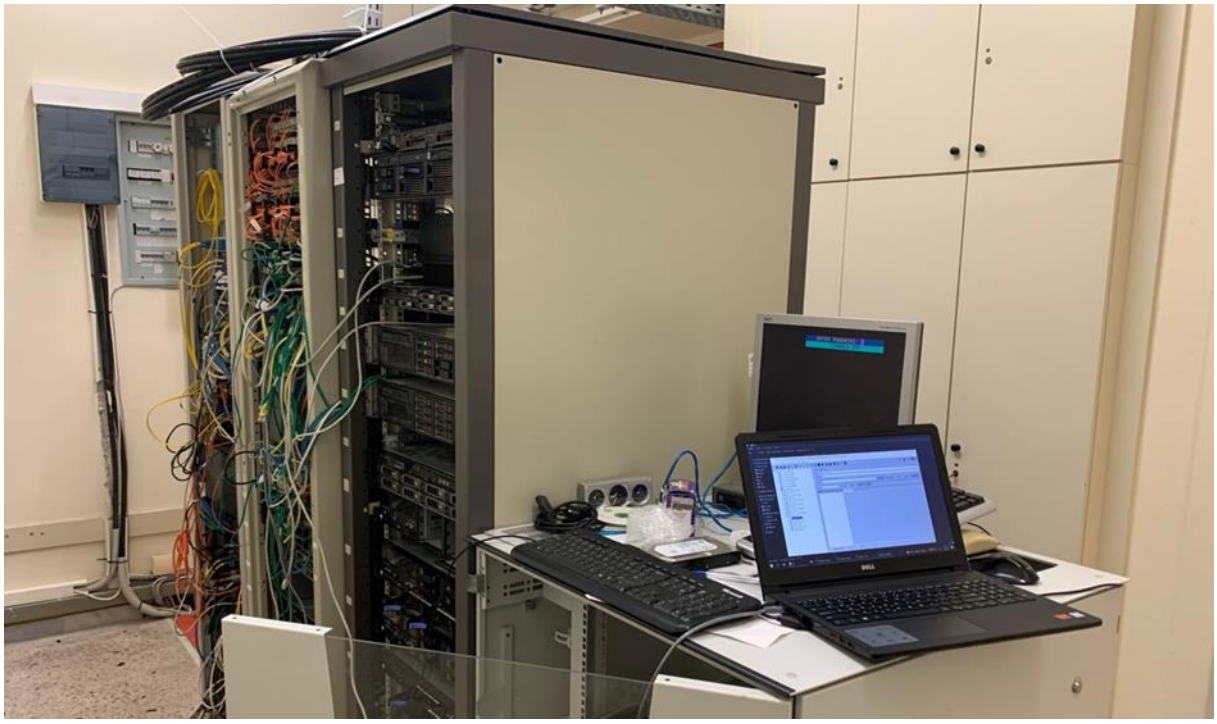
-t: προσδιορίζει τη διαδρομή του αρχείου .jmx που θα εκτελεστεί

-l: προσδιορίζει τη διαδρομή του αρχείου JTL στο οποίο θα αποθηκευτούν όλα τα δεδομένα

Η διενέργεια των ελέγχων πραγματοποιήθηκε με τις ρυθμίσεις που φαίνονται στην Εικόνα 4.2 για το σενάριο 1, δηλαδή 600 χρήστες οι οποίοι προστίθενται σταδιακά μέσα σε διάστημα 15 λεπτών και εκτελούν επαναληπτικά τα βήματα του σχεδίου ελέγχου. Για το σενάριο 2 οι χρήστες ήταν 500 για το ίδιο χρονικό διάστημα.

Ιδανικά, το περιβάλλον εκτέλεσης των ελέγχων θα πρέπει να είναι ένα απομονωμένο περιβάλλον προ-παραγωγής το οποίο θα προσομοιάζει τις συνθήκες του περιβάλλοντος παραγωγής. Στην περίπτωση μας υπήρχε μόνο το περιβάλλον παραγωγής, συνεπώς οι έλεγχοι θα μπορούσαν να εκτελεστούν μόνο σε αυτό.

Κατά τη διάρκεια εκτέλεσης των ελέγχων, επιθεωρούσαμε τα γραφήματα απόδοσης του επεξεργαστή, της μνήμης και του δικτύου που παρέχονται από το XenCenter, το λογισμικό που χρησιμοποιείται για τη διαχείριση των εικονικών μηχανών που φιλοξενούν τους διακομιστές.



Εικόνα 4.10: Η διενέργεια των ελέγχων στο χώρο που φιλοξενείται ο διακομιστής

4.7 Δημιουργία αναφοράς εκτέλεσης ελέγχων

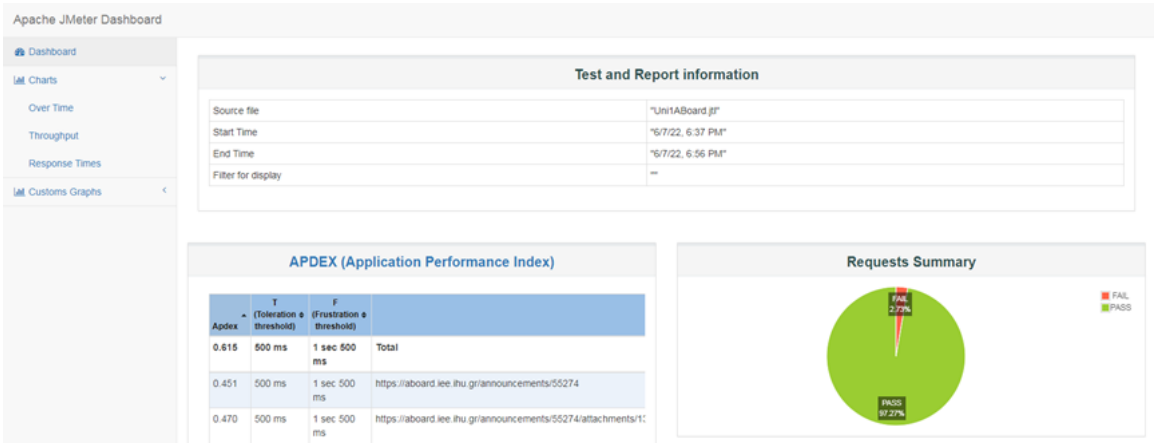
Έχοντας στη διάθεση μας τα αρχεία .jtl με τα δεδομένα των αποτελεσμάτων, μπορούμε να δημιουργήσουμε το HTML Dashboard Report (Εικόνα 4.11) με την παρακάτω εντολή, το οποίο είναι μια αναφορά με διαγράμματα και στατιστικά στοιχεία για τον έλεγχο που διενεργήθηκε. Η παρουσίαση των αποτελεσμάτων γίνεται στο επόμενο κεφάλαιο.

```
jmeter -g log.jtl -o folder
```

Η εντολή έχει τις παρακάτω παραμέτρους:

-g: προσδιορίζει τη διαδρομή του αρχείου .jtl που θα χρησιμοποιηθεί για τη δημιουργία της αναφοράς

-o: προσδιορίζει τη διαδρομή του φακέλου στον οποίο θα δημιουργηθεί η αναφορά σε μορφή HTML



Εικόνα 4.11: Το DashBoard Report για το aBoard

4.8 Επίλογος

Στο κεφάλαιο αυτό ασχοληθήκαμε με τη μελέτη μιας περίπτωσης ελέγχου απόδοσης των δύο υπηρεσιών ανακοινώσεων του τμήματος ΜΠΗΣ.

Αρχικά, αναφέρθηκε το θεωρητικό υπόβαθρο για τους ελέγχους απόδοσης αναλύοντας τα διάφορα είδη των ελέγχων και τις δραστηριότητες που απαιτούνται για να πραγματοποιηθεί ένας ολοκληρωμένος έλεγχος απόδοσης.

Στη συνέχεια, παρουσιάστηκε το εργαλείο που χρησιμοποιήθηκε για τη διενέργεια των ελέγχων, το Apache JMeter, αναφέροντας τις δυνατότητες του και τα απαραίτητα στοιχεία που περιέχει ένα σχέδιο ελέγχου. Επίσης, παρουσιάστηκαν τα δύο σενάρια που επιλέχθηκαν για τους ελέγχους απόδοσης και η μεθοδολογία με την οποία δημιουργήθηκαν στο JMeter.

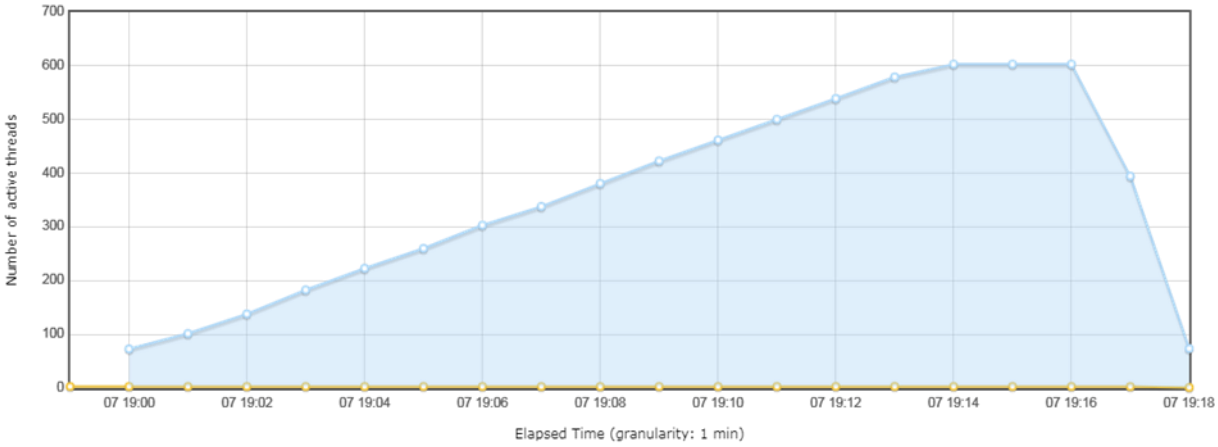
Τέλος, έγινε αναφορά στη διαδικασία εκτέλεσης των ελέγχων και στον τρόπο δημιουργίας αναφοράς διαγραμμάτων με τη χρήση του JMeter. Στο επόμενο κεφάλαιο, παρουσιάζονται και αναλύονται τα αποτελέσματα των ελέγχων απόδοσης για τις δύο υπηρεσίες ανακοινώσεων.

Κεφάλαιο 5ο: Ανάλυση αποτελεσμάτων ελέγχου απόδοσης

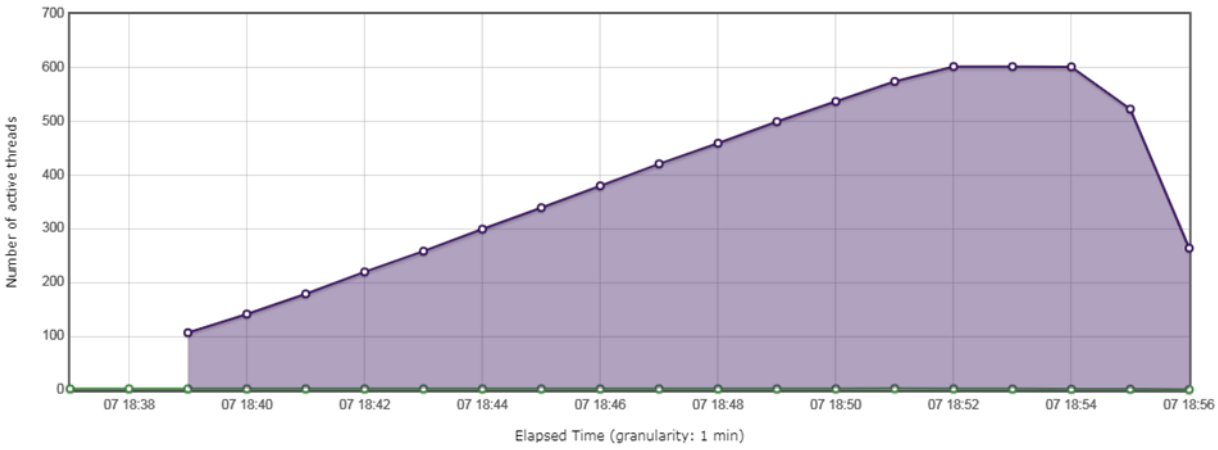
5.1 Εισαγωγή

Για την παρουσίαση των αποτελεσμάτων του ελέγχου απόδοσης χρησιμοποιήσαμε τα διαγράμματα που αποτελούν μέρος της αναφοράς που δημιουργήσαμε με το JMeter, το HTML Dashboard Report.

Αρχικά, στο Σχήμα 5.1 και Σχήμα 5.2 παρουσιάζεται ο ρυθμός με τον οποίο αυξάνονται οι χρήστες κατά τη διάρκεια εκτέλεσης του ελέγχου στα δύο συστήματα. Σε διάστημα 15 λεπτών από την έναρξη του ελέγχου, οι χρήστες προστίθενται σταδιακά έως ότου φτάσουν τον αριθμό των 600 χρηστών, οι οποίοι θα εκτελούν επαναλαμβανόμενα το σενάριο ελέγχου. Στο τέλος του διαγράμματος δίνουμε εντολή διακοπής του ελέγχου, για αυτό και οι χρήστες μειώνονται με γρήγορο ρυθμό μέχρι να μηδενιστούν.



Σχήμα 5.1: : Ενεργοί χρήστες κατά τη διάρκεια εκτέλεσης του ελέγχου στο apps (Σενάριο 1)



Σχήμα 5.2: Ενεργοί χρήστες κατά τη διάρκεια εκτέλεσης του ελέγχου στο aBoard (Σενάριο 1)

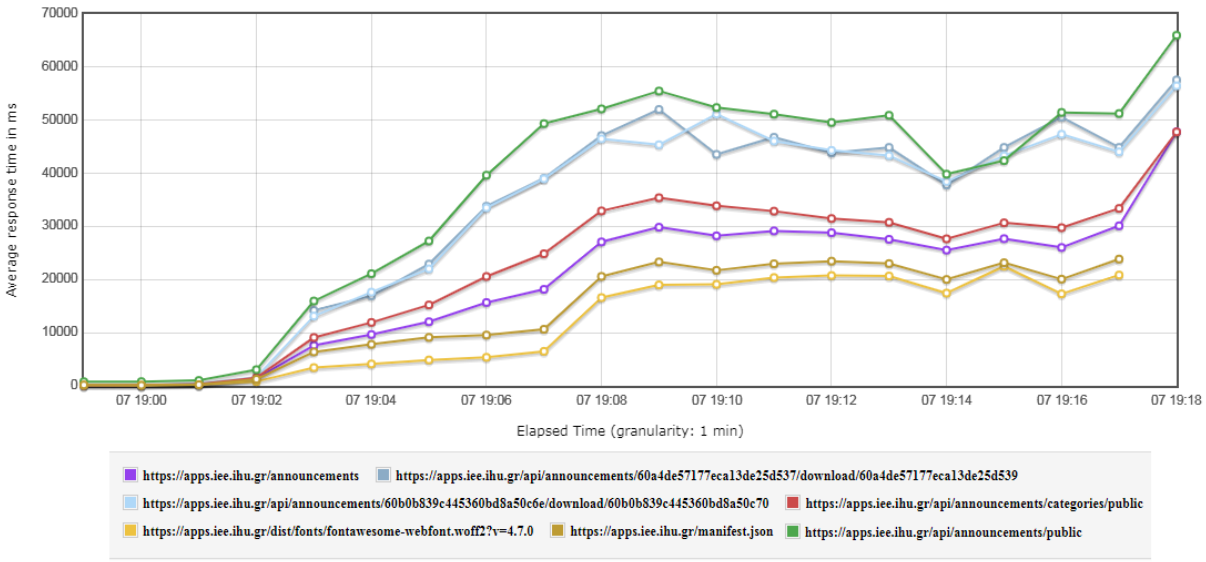
5.2 Αποτελέσματα για το σενάριο 1

Στο πρώτο σενάριο, προσομοιώνουμε την περιήγηση χρηστών στην υπηρεσία ανακοινώσεων με διάφορες ενέργειες που θα έκανε ένας πραγματικός χρήστης.

5.2.1 Μέσος χρόνος απόκρισης

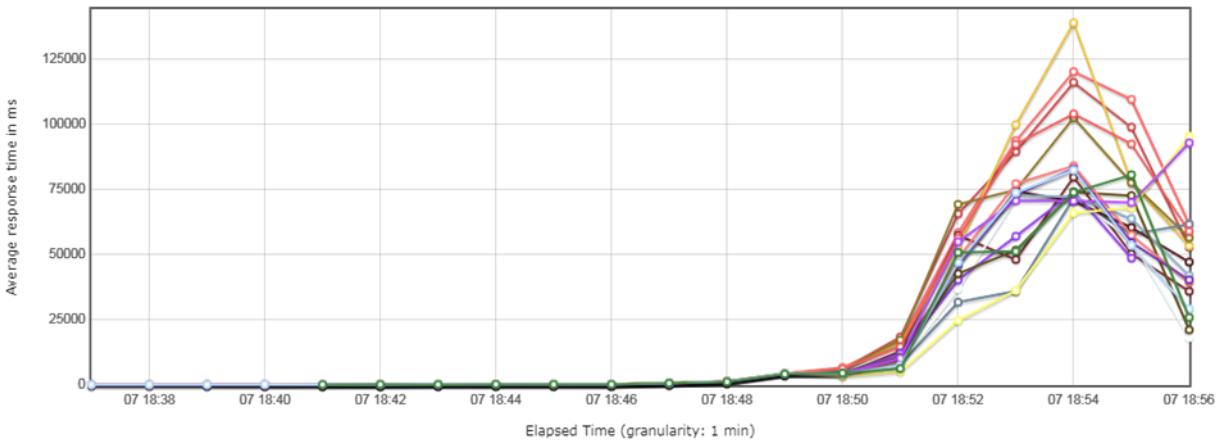
Στο Σχήμα 5.2, παρουσιάζεται ο μέσος χρόνος απόκρισης των αιτημάτων κατά τη διάρκεια εκτέλεσης του ελέγχου. Παρατηρούμε, ότι στα πρώτα δυόμιση λεπτά εκτέλεσης του ελέγχου (19:01) ο χρόνος απόκρισης είναι στα φυσιολογικά επίπεδα, όμως από εκείνο το σημείο και μετά υπάρχει μια αύξηση στο χρόνο απόκρισης όλων των αιτημάτων. Σε αυτό το σημείο, ο αριθμός των ενεργών χρηστών είναι 135 και ο μέσος χρόνος απόκρισης κυμαίνεται από 807 ms (για το embedded resource: font) έως 3045 ms (για την announcements/public).

Όσο προστίθενται περισσότεροι χρήστες οι οποίοι εκτελούν το σχέδιο ελέγχου, ο μέσος χρόνος απόκρισης συνεχίζει να αυξάνεται, με αποτέλεσμα να παρατηρούνται σημαντικές καθυστερήσεις στην απόκριση του διακομιστή.



Σχήμα 5.3: Μέσος χρόνος απόκρισης για το apps (Σενάριο 1)

Στο Σχήμα 5.4 παρουσιάζεται το αντίστοιχο διάγραμμα για το aBoard. Είναι φανερό ότι υπάρχει σημαντική διαφορά συγκριτικά με το apps, καθώς δεν παρατηρούνται καθυστερήσεις στον μέσο χρόνο απόκρισης μέχρι το ενδέκατο λεπτό εκτέλεσης του ελέγχου (18:48). Σε αυτό το σημείο, ο αριθμός των ενεργών χρηστών είναι 457 και ο μέσος χρόνος απόκρισης κυμαίνεται από 41 ms έως 1654 ms (για την announcements/54777). Από εκείνο το σημείο και για τα επόμενα 2 λεπτά παρατηρείται αύξηση του μέσου χρόνου απόκρισης, ενώ στη συνέχεια η αύξηση είναι εξαιρετικά σημαντική καθιστώντας στη πραγματικότητα αδύνατη τη χρήση του συστήματος.



Σχήμα 5.4: Μέσος χρόνος απόκρισης για το aBoard (Σενάριο 1)

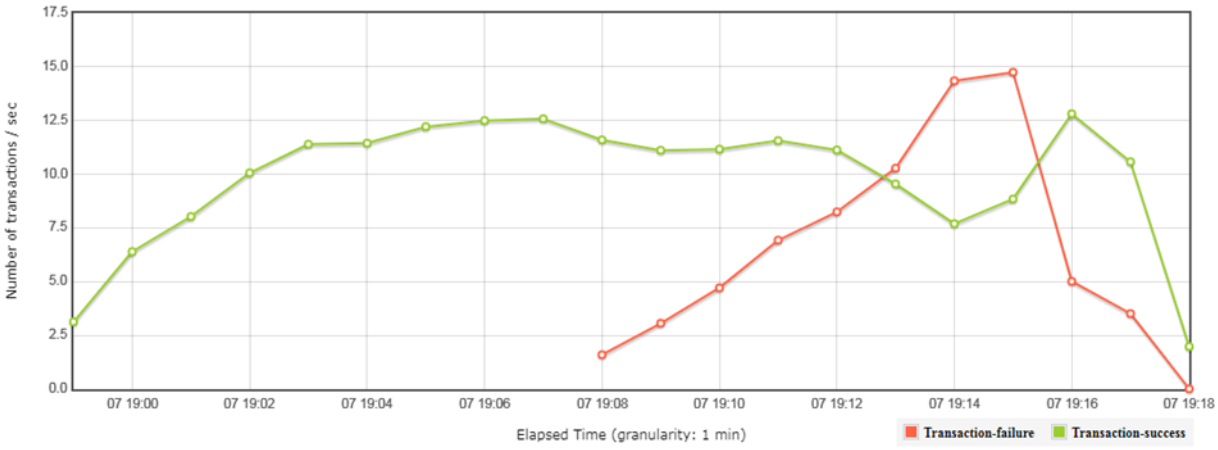
5.2.2 Αριθμός συναλλαγών ανά δευτερόλεπτο

Στο Σχήμα 5.5 παρουσιάζεται ο αριθμός των ολοκληρωμένων συναλλαγών ανά δευτερόλεπτο για το apps. Στον έλεγχο απόδοσης, η ρυθμαπόδοση συχνά αναφέρεται ως "Συναλλαγές ανά δευτερόλεπτο" (transactions per second, TPS) [34]. Η ρυθμαπόδοση συστήματος (system throughput) είναι μια μετρική του αριθμού των συναλλαγών που το σύστημα επεξεργάζεται σε μια μονάδα χρόνου, για παράδειγμα ο αριθμός των HTTP αιτημάτων ανά δευτερόλεπτο [29].

Παρατηρούμε ότι στο ένατο λεπτό εκτέλεσης του ελέγχου, ξεκινούν οι αποτυχημένες συναλλαγές λόγω των σφαλμάτων που υπάρχουν στον Πίνακα 5.1.

Πίνακας 5.1: Ο τύπος των σφαλμάτων που εμφανίστηκαν κατά την εκτέλεση του ελέγχου στο apps

Τύπος σφάλματος	Ποσοστό επί των σφαλμάτων	Ποσοστό επί όλων των δειγμάτων
Non HTTP response code: org.apache.http.conn.HttpHostConnectException/Non HTTP response message: Connect to apps.iee.ihu.gr:443 [apps.iee.ihu.gr/195.251.123.163] failed: Connection timed out: no further information	75.24%	20.33%
Non HTTP response code: java.net.SocketException/Non HTTP response message: Connection reset	15.01%	4.06%
Non HTTP response code: org.apache.http.NoHttpResponseException/Non HTTP response message: apps.iee.ihu.gr:443 failed to respond	4.87%	1.32%
Non HTTP response code: java.net.SocketTimeoutException/Non HTTP response message: Read timed out	3.49%	0.94%

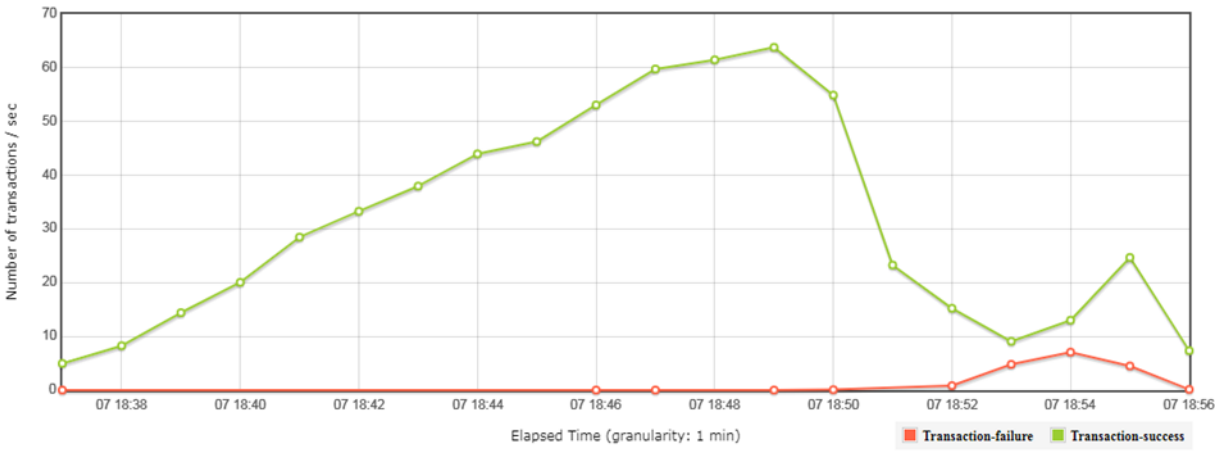


Σχήμα 5.5: Αριθμός συναλλαγών/δευτ. για το apps (Σενάριο 1)

Αντίστοιχα, για το aBoard, ο αριθμός των ολοκληρωμένων συναλλαγών αρχίζει να μειώνεται στο σημείο όπου αρχίζει η αύξηση του μέσου χρόνου απόκρισης του διακομιστή (18:49). Επιπλέον, τρία λεπτά αργότερα εμφανίζονται τα πρώτα σφάλματα, τα οποία υπάρχουν στον Πίνακα 5.2. Το συνολικό ποσοστό των σφαλμάτων στο σύνολο των δειγμάτων για το aBoard, είναι σημαντικά μικρότερο (2.73%) έναντι του apps (27.02%).

Πίνακας 5.2: Ο τύπος των σφαλμάτων που εμφανίστηκαν κατά την εκτέλεση του ελέγχου στο aBoard

Τύπος σφάλματος	Ποσοστό επί των σφαλμάτων	Ποσοστό επί όλων των δειγμάτων
Non HTTP response code: javax.net.ssl.SSLHandshakeException/Non HTTP response message: Remote host terminated the handshake	84.46%	2.30%
Assertion failed	11.82%	0.32%



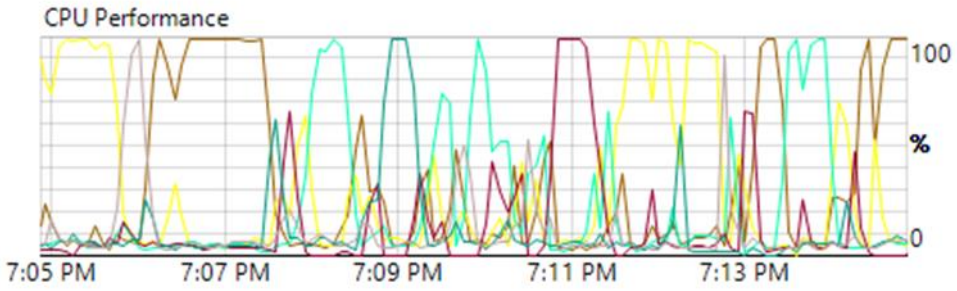
Σχήμα 5.6: : Αριθμός συναλλαγών/δευτ. για το aBoard (Σενάριο 1)

5.2.3 Χρήση επεξεργαστή του διακομιστή

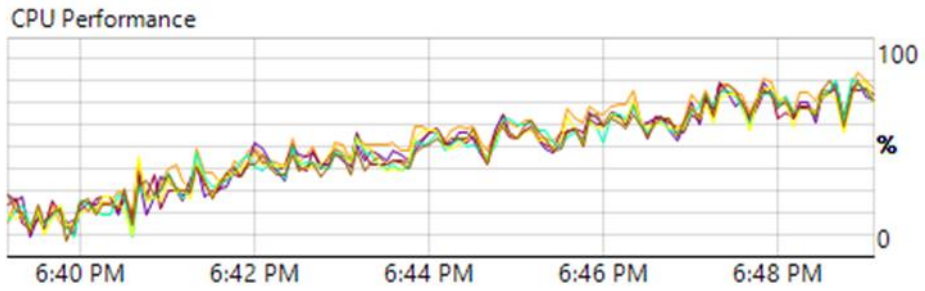
Κατά τη διάρκεια διενέργειας των ελέγχων, παρακολουθούσαμε τα γραφήματα απόδοσης του διακομιστή για τον επεξεργαστή, τη μνήμη και το δίκτυο. Παρακάτω παρουσιάζονται μόνο τα γραφήματα του επεξεργαστή καθώς τα υπόλοιπα δύο δεν παρουσίαζαν ενδιαφέρον ως προς τον τομέα της απόδοσης.

Στο Σχήμα 5.7, παρουσιάζεται η χρήση του επεξεργαστή στο διακομιστή του apps κατά τη διάρκεια διενέργειας του ελέγχου. Ο επεξεργαστής αποτελείται από 6 πυρήνες και παρατηρούμε ότι η χρήση τους εναλλάσσεται χωρίς να φτάνουν όλοι ταυτόχρονα στο μέγιστο της λειτουργίας τους.

Αντίστοιχα, στο Σχήμα 5.8, παρουσιάζεται η χρήση του επεξεργαστή στο διακομιστή του aBoard κατά τη διάρκεια διενέργειας του ελέγχου. Αυτός ο επεξεργαστής έχει τα ίδια χαρακτηριστικά ισχύος με αυτόν του apps. Παρατηρούμε όμως ένα διαφορετικό μοτίβο χρήσης, όπου όλοι οι πυρήνες χρησιμοποιούνται με παρόμοιο ποσοστό, το οποίο σταδιακά αυξάνεται και πλησιάζει το 80%.



Σχήμα 5.7: Χρήση επεξεργαστή στον διακομιστή του apps (Σενάριο 1)



Σχήμα 5.8: Χρήση επεξεργαστή στον διακομιστή του aBoard (Σενάριο 1)

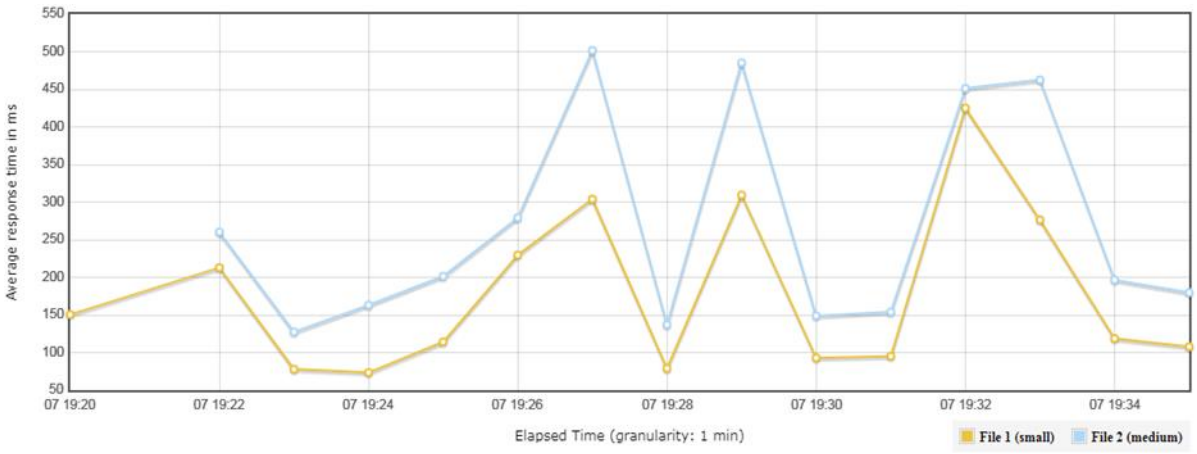
5.3 Αποτελέσματα για το Σενάριο 2

Στο δεύτερο σενάριο, προσομοιώνουμε τη μαζική μεταφόρτωση αρχείων.

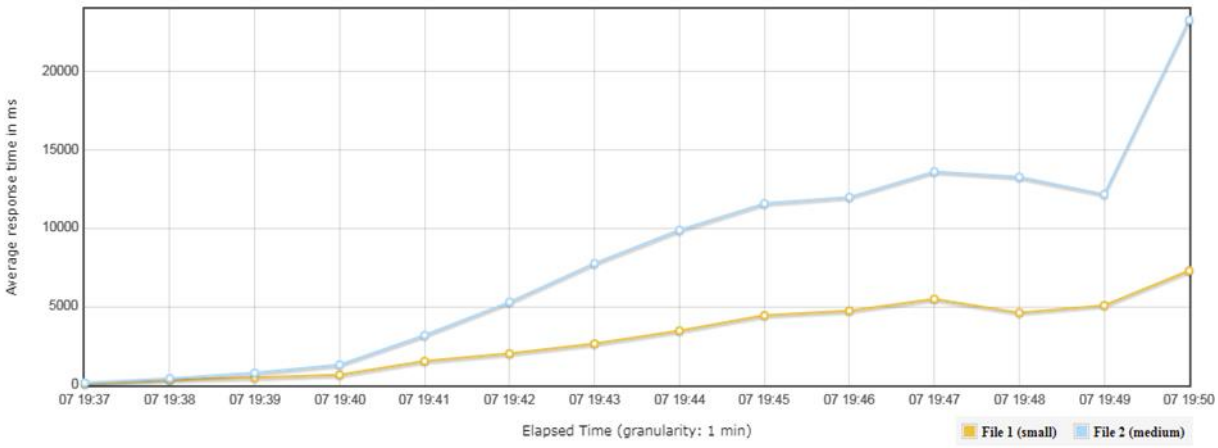
5.3.1 Μέσος χρόνος απόκρισης

Στα διαγράμματα του μέσου χρόνου απόκρισης είναι εμφανής μια μεγάλη διαφορά υπέρ του apps. Πιο συγκεκριμένα, στο δεύτερο λεπτό διενέργειας του ελέγχου, ο μέσος χρόνος απόκρισης για το apps

είναι 213 ms για το μικρό αρχείο και 260 ms για το αρχείο μεσαίου μεγέθους. Για το aBoard, οι χρόνοι είναι 475 ms και 787 ms αντίστοιχα. Ενώ στα δύο πρώτα λεπτά ο μέσος χρόνος απόκρισης είναι φυσιολογικός και για τα δύο συστήματα, από το τρίτο λεπτό και τους 155 χρήστες ξεκινάει μια εκθετική αύξηση του μέσου χρόνου απόκρισης για το aBoard, ενώ και η χρήση του επεξεργαστή του διακομιστή κυμαίνεται σε υψηλά επίπεδα. Στο apps ο μέγιστος χρόνος απόκρισης φτάνει μόλις τα 502 ms.



Σχήμα 5.9: Μέσος χρόνος απόκρισης για το apps (Σενάριο 2)

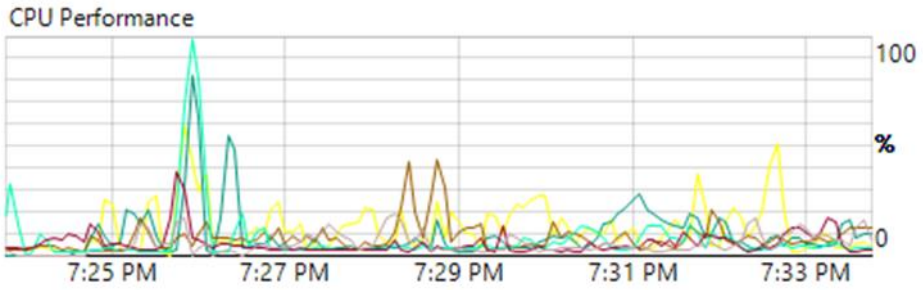


Σχήμα 5.10: Μέσος χρόνος απόκρισης για το aBoard (Σενάριο 2)

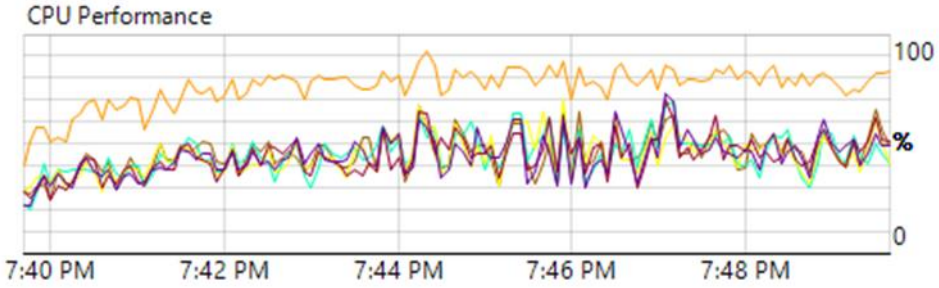
5.3.2 Χρήση επεξεργαστή του διακομιστή

Στα διαγράμματα χρήσης του επεξεργαστή, παρατηρούμε πάλι μια εμφανή διαφορά υπέρ του apps. Πιο συγκεκριμένα, στο apps εκτός από ένα σημείο στιγμιαίας μεταβολής της χρήσης του επεξεργαστή στο έκτο λεπτό (19:26) διενέργειας του ελέγχου, ο μέσος όρος χρήσης του επεξεργαστή κυμαίνεται σε επίπεδα κάτω του 30%. Στο aBoard, από το τρίτο λεπτό παρατηρούμε μεγάλη αύξηση στη χρήση του επεξεργαστή η οποία προσεγγίζει το 90% μετά το έβδομο λεπτό διενέργειας του ελέγχου, για έναν πυρήνα και κυμαίνεται στο 50% για τους υπόλοιπους. Αυτή η αυξημένη χρήση του επεξεργαστή οφείλεται στο query που πραγματοποιείται στη βάση κάθε φορά που λαμβάνεται ένα αίτημα

μεταφόρτωσης ενός αρχείου. Αυτό το σημείο χρήζει περαιτέρω διερεύνησης για τον εντοπισμό πιθανών βελτιώσεων.



Σχήμα 5.11: Χρήση επεξεργαστή στο διακομιστή του apps (Σενάριο 2)



Σχήμα 5.12: Χρήση επεξεργαστή στο διακομιστή του aBoard (Σενάριο 2)

Παρατηρώντας τα ιδιαίτερα αυξημένα ποσοστά χρήσης του επεξεργαστή και τους μεγάλους χρόνους απόκρισης για το aBoard κατά τη διάρκεια του ελέγχου, αποφασίσαμε να διερευνήσουμε τις ρυθμίσεις του διακομιστή που θα μπορούσαν να επηρεάσουν αυτά τα αποτελέσματα.

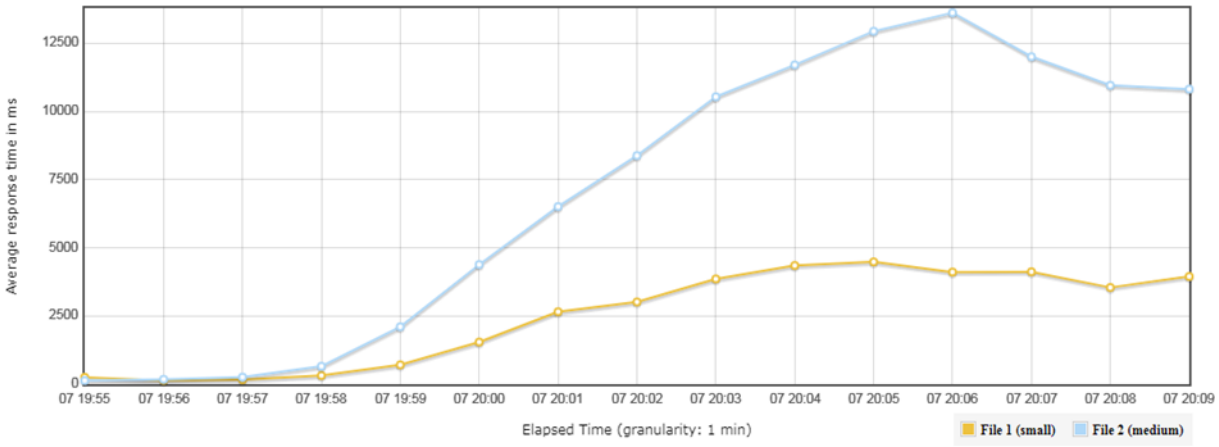
Στο aBoard χρησιμοποιείται για την php το module php-fpm (FastCGI Process Manager) [35]. Αυτό αναλαμβάνει να παρακάμπτει το κλασικό μοντέλο CGI που είναι πολύ αργό. Αυτό οφείλεται στο ότι για κάθε αίτηση από τον πελάτη, στο διακομιστή δημιουργείται μια νέα διεργασία με το μοντέλο CGI. Το php-fpm αναλαμβάνει να "προδημιουργήσει" διεργασίες, έτσι ώστε να αποφεύγεται η επαναλαμβανόμενη δημιουργία διεργασίας για το ίδιο πρόγραμμα. Οι αρχικές ρυθμίσεις επέτρεπαν τη δημιουργία μόνο 5 διεργασιών (το μεταβάλαμε σε 255), οι αρχικές διεργασίες ήταν 2 (το μεταβάλλαμε σε 10) καθώς και το πλήθος των διαθέσιμων διεργασιών ήταν 3 (το μεταβάλλαμε σε 30).

```
; Note: This value is mandatory.
pm.max_children = 255

; The number of child processes created on startup.
; Note: Used only when pm is set to 'dynamic'
; Default Value: (min_spare_servers + max_spare_servers) / 2
pm.start_servers = 10

; The desired minimum number of idle server processes.
; Note: Used only when pm is set to 'dynamic'
; Note: Mandatory when pm is set to 'dynamic'
pm.min_spare_servers = 1

; The desired maximum number of idle server processes.
; Note: Used only when pm is set to 'dynamic'
; Note: Mandatory when pm is set to 'dynamic'
pm.max_spare_servers = 30

; The number of seconds after which an idle process will be killed.
; Note: Used only when pm is set to 'ondemand'
; Default Value: 10s
;pm.process_idle_timeout = 10s;

; The number of requests each child process should execute before respawning.
; This can be useful to work around memory leaks in 3rd party libraries. For
; endless request processing specify '0'. Equivalent to PHP_FCGI_MAX_REQUESTS.
; Default Value: 0
;pm.max_requests = 500
```

Εικόνα 5.1: Οι ρυθμίσεις του php-fpm module

5.3.3 Επανεκτέλεση ελέγχων για το σενάριο 2

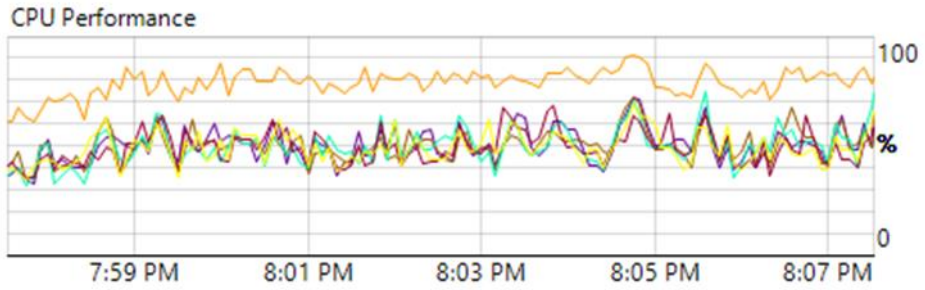
Μετά την εφαρμογή των αλλαγών στις ρυθμίσεις του php-fpm, εκτελέσαμε ξανά το σχέδιο ελέγχου για το σενάριο 2 στο aBoard. Ο μέσος χρόνος απόκρισης είναι πλέον μειωμένος σε σύγκριση με την πρώτη εκτέλεση του ελέγχου (Πίνακας 5.3). Μετρήσαμε τους χρόνους απόκρισης σε συγκεκριμένα στιγμιότυπα των δύο εκτελέσεων του ελέγχου και παρατηρήσαμε ότι ιδιαίτερα στην αρχή του ελέγχου, το ποσοστό μείωσης του μέσου χρόνου απόκρισης είναι ιδιαίτερα υψηλό (έως 67% στο 2ο λεπτό) με φθίνουσα πορεία όσο προχωράει ο έλεγχος και αυξάνεται ο αριθμός των χρηστών, όμως ο χρόνος απόκρισης παραμένει βελτιωμένος σε όλη τη διάρκεια του ελέγχου. Για τη χρήση του επεξεργαστή (Σχήμα 5.14) δεν παρατηρούμε διαφορές συγκριτικά με την πρώτη εκτέλεση.



Σχήμα 5.13: Μέσος χρόνος απόκρισης για το aBoard μετά τις αλλαγές στις ρυθμίσεις (Σενάριο 2)

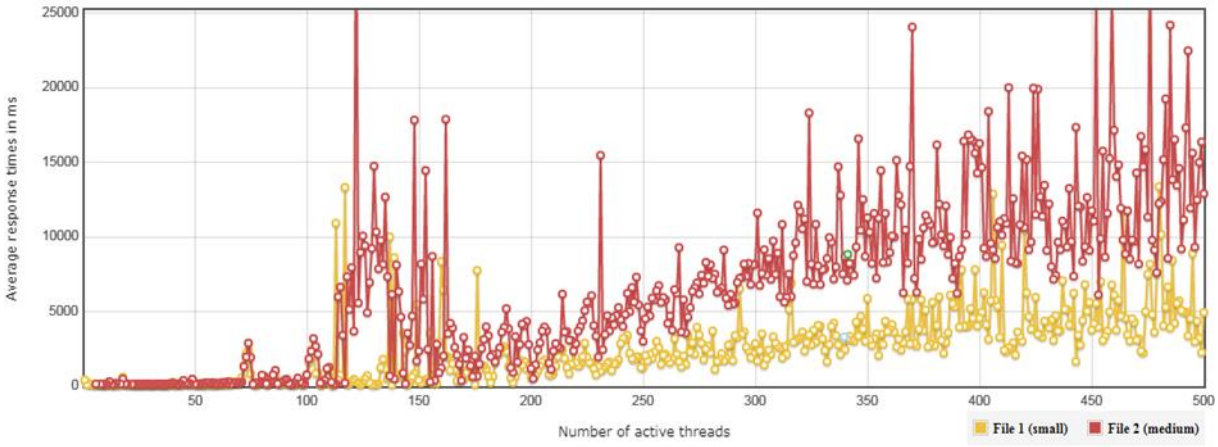
Πίνακας 5.3: Σύγκριση μέσου χρόνου απόκρισης των δύο εκτελέσεων

Λεπτό	1η εκτέλεση ελέγχου	2η εκτέλεση ελέγχου	% μείωσης
2'	475ms, 787 ms	160 ms, 259 ms	66%-67%
4'	1535 ms, 3191 ms	708 ms, 2105 ms	34%-54%
8'	4452ms, 11598ms	3859ms, 10540 ms	9%-13%
12'	5096 ms, 12189 ms	4122 ms, 12009 ms	1.4%-19%

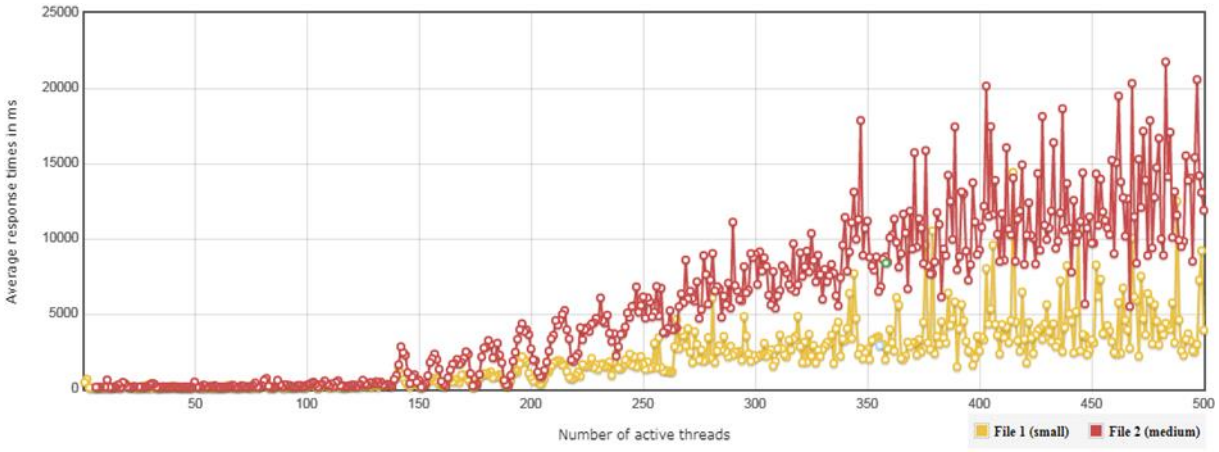


Σχήμα 5.14: Χρήση επεξεργαστή στο διακομιστή του aBoard μετά τις αλλαγές στις ρυθμίσεις (Σενάριο 2)

Στα διαγράμματα του μέσου χρόνου απόκρισης ανά αριθμό ενεργών χρηστών, είναι πιο φανερό ότι υπάρχει βελτίωση στην απόδοση του διακομιστή μετά τις αλλαγές των προεπιλεγμένων ρυθμίσεων.



Σχήμα 5.15: Μέσος χρόνος απόκρισης ανά αριθμό ενεργών χρηστών στο aBoard πριν τις αλλαγές στις ρυθμίσεις (Σενάριο 2)



Σχήμα 5.16: Μέσος χρόνος απόκρισης ανά αριθμό ενεργών χρηστών στο aBoard μετά τις αλλαγές στις ρυθμίσεις (Σενάριο 2)

5.4 Επίλογος

Σε αυτό το κεφάλαιο παρουσιάστηκαν και αναλύθηκαν τα αποτελέσματα των ελέγχων απόδοσης που πραγματοποιήθηκαν στις δύο υπηρεσίες ανακοινώσεων του τμήματος ΜΠΗΣ, το apps και το aBoard.

Αρχικά, για το σενάριο 1 παρουσιάστηκαν διαγράμματα μέσου χρόνου απόκρισης, αριθμού συναλλαγών και χρήσης του επεξεργαστή του διακομιστή. Από την ανάλυση των αποτελεσμάτων προέκυψε ότι το σύστημα του aBoard μπορεί να διαχειριστεί περισσότερους χρήστες, πριν αρχίσουν να υπάρχουν αισθητές καθυστερήσεις στο χρόνο απόκρισης.

Στη συνέχεια, για το σενάριο 2 παρουσιάστηκαν διαγράμματα μέσου χρόνου απόκρισης και χρήσης του επεξεργαστή του διακομιστή. Από την ανάλυση των αποτελεσμάτων προέκυψε ότι το apps συμπεριφέρεται καλύτερα σε αυτό το σενάριο, όταν ο φόρτος αρχίζει να αυξάνεται.

Τέλος, αφού έγιναν αλλαγές βελτιστοποίησης στις ρυθμίσεις του διακομιστή του aBoard, ο έλεγχος απόδοσης για το σενάριο 2 εκτελέστηκε ξανά στο aBoard, δείχνοντας βελτιωμένη απόδοση.

Κεφάλαιο 6ο: Συμπεράσματα και προτάσεις βελτίωσης

6.1 Συμπεράσματα

Ο έλεγχος λογισμικού διακρίνεται σε διάφορες κατηγορίες και είδη. Ο σκοπός αυτής της εργασίας ήταν η απόκτηση γνώσεων σε ένα μέρος αυτού του πεδίου, στην αυτοματοποίηση των ελέγχων λογισμικού και στη διεξαγωγή ενός ολοκληρωμένου ελέγχου απόδοσης. Επιλέχθηκαν αυτοί οι δύο τομείς, λόγω προσωπικού ενδιαφέροντος του συγγραφέα αλλά και επειδή αποτελούν τις μεγαλύτερες τάσεις στον έλεγχο λογισμικού τα τελευταία χρόνια.

Η αυτοματοποίηση ελέγχων λογισμικού έχει γίνει αναγκαία στους οργανισμούς που θέλουν να παράγουν λογισμικό με πολύ γρήγορους ρυθμούς και η ζήτηση στην αγορά εργασίας για μηχανικούς αυτοματοποίησης ελέγχων λογισμικού είναι εξαιρετικά μεγάλη. Σε αυτή την εργασία ασχοληθήκαμε και με τους δύο τρόπους που μπορούν να αυτοματοποιηθούν οι έλεγχοι, με ή χωρίς τη χρήση κώδικα. Διαπιστώσαμε ότι η εκμάθηση του Selenium WebDriver είναι εύκολη για κάποιον ο οποίος έχει βασικές γνώσεις προγραμματισμού, ενώ το ίδιο εύκολη είναι και η χρήση των βασικών δυνατοτήτων του. Στο δεύτερο μέρος της αυτοματοποίησης, χρησιμοποιήσαμε ένα εργαλείο αυτοματοποίησης ελέγχων χωρίς τη χρήση κώδικα και διαπιστώσαμε ότι η εκμάθηση και η χρήση του είναι εύκολη ακόμα και για κάποιον που δεν έχει γνώσεις προγραμματισμού, χρειάζονται μόνο βασικές γνώσεις εντοπισμού των στοιχείων ιστού.

Ο έλεγχος της απόδοσης του λογισμικού είναι πολύ σημαντικός για τους οργανισμούς, επειδή η διανομή μιας εφαρμογής στην παραγωγή με χαμηλές μετρήσεις απόδοσης λόγω ανύπαρκτου ή ελλιπούς ελέγχου απόδοσης, είναι πιθανό να επιφέρει σοβαρή ζημιά στον οργανισμό λόγω της κακής φήμης. Σε αυτή την εργασία ασχοληθήκαμε με τη μελέτη μιας περίπτωσης ελέγχου απόδοσης, ακολουθώντας όλα τα βήματα τα οποία είναι απαραίτητα για έναν ολοκληρωμένο έλεγχο απόδοσης, από το σχεδιασμό μέχρι και την ανάλυση των αποτελεσμάτων. Διαπιστώσαμε ότι η εκμάθηση και η χρήση του Apache JMeter απαιτεί τεχνικές γνώσεις και εμπειρία για την αποτελεσματική διενέργεια του ελέγχου απόδοσης. Επίσης, η ανάλυση των αποτελεσμάτων είναι ένα σημαντικό μέρος του ελέγχου απόδοσης και χρειάζεται συνδυασμός γνώσεων και δεδομένων για τον εντοπισμό των σημείων της εφαρμογής που χρήζουν βελτίωσης.

Από τα αποτελέσματα των ελέγχων απόδοσης συμπεράναμε ότι το aBoard είναι γενικά καλύτερο συγκριτικά με το apps. Εντοπίσαμε όμως σημεία βελτίωσης για το aBoard με την εκτέλεση του δεύτερου σεναρίου, όπως στις ρυθμίσεις του διακομιστή και στο query που πραγματοποιείται στη βάση κατά τη μεταφόρτωση αρχείων.

6.2 Μελλοντικές βελτιώσεις

Σχετικά με το κεφάλαιο της αυτοματοποίησης ελέγχων με κώδικα, θα μπορούσαν να χρησιμοποιηθούν και άλλες γλώσσες προγραμματισμού από αυτές που υποστηρίζει το Selenium για την ανάπτυξη σεναρίων ελέγχου. Επίσης, θα μπορούσαν να εξεταστούν και άλλα framework αυτοματοποίησης, όπως το Cypress και το Cucumber. Για την αυτοματοποίηση ελέγχων χωρίς κώδικα, θα μπορούσαν να χρησιμοποιηθούν και άλλα δημοφιλή εργαλεία, όπως το UiPath και το Automation Anywhere και να γίνει σύγκριση του τρόπου λειτουργίας και των δυνατοτήτων τους. Και στους δύο τρόπους αυτοματοποίησης, θα μπορούσαν να αναπτυχθούν περισσότερα σενάρια ελέγχου για την επίδειξη πιο πολύπλοκων δυνατοτήτων των εργαλείων.

Σχετικά με το κεφάλαιο του ελέγχου απόδοσης, θα μπορούσαν να δημιουργηθούν περισσότερα σενάρια ελέγχου, καλύπτοντας διαφορετικά είδη ελέγχων απόδοσης. Επίσης, θα μπορούσαν να εξεταστούν τα ευρήματα από τα αποτελέσματα του ελέγχου απόδοσης και να εφαρμοστούν όλες οι αναγκαίες βελτιώσεις στο σύστημα, ώστε να εξαλειφθούν οι αδυναμίες του.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] IEEE, “IEEE Standard for Software Quality Assurance Processes,” IEEE Std 730-2014 (Revision of IEEE Std 730-2002), pp. 1–138, Jun. 2014.

[2] International Software Testing Qualifications Board, “Certified Tester Foundation Level Syllabus”, 2018. [Online]. Available: https://www.istqb.org/certifications/certified-tester-foundation-level . [Accessed: 03/09/2022]

[3] G. J. Myers and C. Sandler, *The Art of Software Testing*. John Wiley & Sons, New Jersey, USA, 2004.

[4] Perforce, “What is static analysis? Static code analysis overview”, 2020. [Online]. Available: https://www.perforce.com/blog/sca/what-static-analysis . [Accessed: 03/09/2022]

[5] Γ. Ζαρογιάννης, “Η χρήση της στατικής ανάλυσης στην παραγωγή ασφαλούς λογισμικού”, Διπλωματική εργασία, Τμήμα Ψηφιακών Συστημάτων, Πανεπιστήμιο Πειραιώς, 2013. [Online]. Available: https://dione.lib.unipi.gr/xmlui/bitstream/handle/unipi/8591/Zarogiannis_Georgios.pdf . [Accessed: 03/09/2022]

[6] J. Humble and D. Farley, *Continuous delivery: reliable software releases through build, test, and deployment automation*. Pearson Education, 2010.

[7] Yuqing Wang, Mika V. Mäntylä, Zihao Liu, Jouni Markkula, “Test automation maturity improves product quality—Quantitative study of open source projects using continuous integration,” Journal of Systems and Software, Volume 188, 2022.

[8] E. F. Collins and V. F. de Lucena, "Software Test Automation practices in agile development environment: An industry experience report," 2012 7th International Workshop on Automation of Software Test (AST), 2012, pp. 57-63.

[9] Selenium, “Selenium History”, 2022. [Online]. Available: https://www.selenium.dev/history . [Accessed: 03/09/2022]

[10] InfoWorld, “Open source Selenium web app test suite to support iPhone and Android”, 2011. [Online]. Available: https://web.archive.org/web/20110505181545/http://news.techworld.com/applications/3272444/open-source-selenium-web-app-test-suite-to-support-iphone-and-android . [Accessed: 03/09/2022]

[11] BrowserStack, “What is Selenium IDE”, 2021. [Online]. Available: https://www.browserstack.com/guide/what-is-selenium-ide . [Accessed: 03/09/2022]

[12] Leapwork, “Selenium IDE vs Leapwork”, 2022. [Online]. Available: https://www.leapwork.com/blog/selenium-ide-vs-leapwork . [Accessed: 03/09/2022]

[13] Lambdatest, “Selenium 4 Is Now W3C Compliant: All You Need To Know”, 2020. [Online]. Available: https://www.lambdatest.com/blog/selenium4-w3c-webdriver-protocol/ . [Accessed: 03/09/2022]

[14] W3C, “WebDriver W3C Working Draft”, 2022. [Online]. Available: https://www.w3.org/TR/webdriver . [Accessed: 03/09/2022]

[15] Selenium, “Selenium Grid 4”, 2022. [Online]. Available: https://www.selenium.dev/documentation/grid/ . [Accessed: 03/09/2022]

[16] BrowserStack, “Selenium Grid 4 Tutorial”, 2021. [Online]. Available: https://www.browserstack.com/guide/selenium-grid-4-tutorial . [Accessed: 03/09/2022]

[17] Saucelabs, “What’s coming in Selenium 4: The new Selenium Grid”, 2021. [Online]. Available: https://saucelabs.com/blog/whats-coming-in-selenium-4-the-new-selenium-grid . [Accessed: 03/09/2022]

[18] M. Leotta, D. Clerissi, F. Ricca, and C. Spadaro, “Comparing the maintainability of selenium WebDriver test suites employing different locators: a case study.” In Proceedings of the 2013 International Workshop on Joining AcadeMiA and Industry Contributions to testing Automation (JAMAICA 2013). Association for Computing Machinery, New York, NY, USA, 53–58.

[19] M. Leotta, A. Stocco, F. Ricca and P. Tonella, "Using Multi-Locators to Increase the Robustness of Web Test Cases," 2015 IEEE 8th International Conference on Software Testing, Verification and Validation (ICST), 2015.

[20] K. Presler-Marshall, E. Horton, S. Heckman and K. Stolee, "Wait, Wait. No, Tell Me. Analyzing Selenium Configuration Effects on Test Flakiness," 2019 IEEE/ACM 14th International Workshop on Automation of Software Test (AST), 2019.

[21] Tools QA, “Advance WebDriver Waits”, 2021. [Online]. Available: https://www.toolsqa.com/selenium-webdriver/advance-webdriver-waits/ . [Accessed: 03/09/2022]

[22] B. Varanasi, *Introducing Maven: A Build Tool for Today's Java Developers*. Apress, 2019.

[23] Gartner, “Gartner Survey Reveals Talent Shortages as Biggest Barrier to Emerging Technologies Adoption”, 2021. [Online]. Available: https://www.gartner.com/en/newsroom/press-releases/2021-09-13-gartner-survey-reveals-talent-shortages-as-biggest-barrier-to-emerging-technologies-adoption . [Accessed: 03/09/2022]

[24] Grand View Research, “Robotic Process Automation Market Size Report 2022-2030”, 2022. [Online]. Available: https://www.grandviewresearch.com/industry-analysis/robotic-process-automation-rpa-market . [Accessed: 03/09/2022]

[25] Capital.gr, “Την εξαγορά της ελληνικών συμφερόντων Softomotive ανακοίνωσε η Microsoft”, 2020. [Online]. Available: https://www.capital.gr/epixeiriseis/3454676/tin-exagora-tis-ellinikon-sumferonton-softomotive-anakoinose-i-microsoft . [Accessed: 03/09/2022]

[26] Leapwork, “Test Automation and RPA Tools: What's the Difference?”, 2022. [Online]. Available: https://www.leapwork.com/blog/test-automation-and-rpa-tools-whats-the-difference . [Accessed: 03/09/2022]

[27] C. R. Camacho, S. Marczak and D. S. Cruzes, "Agile Team Members Perceptions on Non-functional Testing: Influencing Factors from an Empirical Study," 2016 11th International Conference on Availability, Reliability and Security (ARES), 2016, pp. 582-589.

[28] Dave, M. S., Patel, M. H., Prajapati, M. A. (2018). Non-functional testing in agile development, IJIRT 5(7).

[29] International Software Testing Qualifications Board, “Certified Tester Performance Testing Syllabus”, 2018. [Online]. Available: https://www.istqb.org/certifications/performance-tester

[30] Abstracta, “Types of performance tests”, 2015. [Online]. Available: https://abstracta.us/blog/performance-testing/types-performance-tests/ . [Accessed: 03/09/2022]

[31] Molyneaux, Ian. *The art of application performance testing: from strategy to tools*. O'Reilly Media, Inc., 2014.

[32] J. Meier, C. Farre, P. Bansode, S. Barber and D Rea, *Performance testing guidance for web applications: patterns & practices*. Microsoft Press, USA, 2007.

[33] The Apache Software Foundation, “Apache JMeter Documentation”, 2022. [Online]. Available: https://jmeter.apache.org/. [Accessed: 03/09/2022]

[34] Test Guild, “What is throughput in performance testing”, 2022. [Online]. Available: https://testguild.com/performance-testing-what-is-throughput/ . [Accessed: 03/09/2022]

[35] Immad Uddin Khan, “PHP-FPM Cuts Web App Loading Times by 300%”, 2021. [Online]. Available: https://www.cloudways.com/blog/php-fpm-on-cloud/ . [Accessed: 03/09/2022]