ΠΑΝΕΠΙΣΤΗΜΙΟ ΘΕΣΣΑΛΙΑΣ
ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ / ΜΗΧΑΝΙΚΩΝ
ΥΠΟΛΟΓΙΣΤΩΝ

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Σχεδιασμός και αξιολόγηση μεθόδων ανάπτυξης οδηγούμενης από ελέγχους σε υπηρεσίες IoT

Συγγραφέας: Γεώργιος ΧΑΛΒΑΤΖΗΣ

Επιβλέπων Καθηγητής: Αθανάσιος ΚΟΡΑΚΗΣ

26 Φεβρουαρίου 2020

Abstract

Η παρούσα πτυχιακή εργασία πραγματεύεται την αξιοποίηση και αξιολόγηση μιας νέας μεθόδου ανάπτυξης λογισμικού, η οποία βασίζεται και οδηγείται από ελέγχους (test-driven development ή TDD). Στόχος είναι να δημιουργηθεί μια γρηγορότερη και βαθύτερη κατανόηση των απαιτήσεων της εφαρμογής, να διασφαλιστεί η αποδοτικότητα του ελεγχόμενου κώδικα και να διατηρηθεί μια συνεχής εστίαση στην ποιότητα της εφαρμογής. Για τους σκοπούς αυτούς εισάγεται μια νέα τεχνική προγραμματισμού, όπου παράλληλα με τον κώδικα παραγωγής γράφεται κώδικας ελέγχου ο οποίος δεν συνεισφέρει στην λειτουργικότητα της εφαρμογής, αλλά εποπτεύει και εξασφαλίζει την σωστή λειτουργία του κώδικα παραγωγής. Γίνεται πρώτα μια επεξήγηση και εμβάθυνση στους τρόπους και τις μορφές ελέγχου του κώδικα παραγωγής. Έπειτα ακολουθεί μια εφαρμογή και αξιολόγηση των μεθόδων αυτών πάνω σε μια υπηρεσία IoT. Στα πλαίσια της διπλωματικής εργασίας έγινε ανάπτυξη συγκεκριμένων tests για τον έλεγχο της IoT υπηρεσίας “Agronit”. Αυτό έγινε εφικτό με την αξιοποίηση ορισμένων εργαλείων. Χρησιμοποιήθηκε μεταξύ άλλων τα test frameworks “Mocha-Chai” και “Jest” για τον έλεγχο του back-end και front-end λογισμικού και το ”Selenium” για την συνολική εποπτεία της εφαρμογής. Η επίδραση της εφαρμογής του testing γίνεται εμφανής στο στάδιο της αποσφαλμάτωσης, όπου εντοπίζονται ελαττώματα στον κώδικα γρήγορα, στο πιο πρώιμο στάδιο της ανάπτυξης. Επιπλέον οδηγεί σε καλύτερο σχεδιασμό του λογισμικού και βελτιώνει τον χρόνο ανάπτυξης του κώδικα, ενώ διασαφηνίζει την εικόνα του μοντέλου του κώδικα που έχει ο προγραμματιστής, αυξάνοντας την αυτοπεποίθηση και την παραγωγικότητα. Η οδηγούμενη από ελέγχους ανάπτυξη απευθύνεται στο ευρύ σύνολο των προγραμματιστών και απώτερος σκοπός είναι να εδραιωθεί ως κύρια προγραμματιστική τεχνική του μέλλοντος.

Ευχαριστίες

Ευχαριστώ την οικογένειά μου και τους φίλους μου που με στήριξαν καθ όλη την διάρκεια των σπουδών μου και της περάτωσης της παρούσας πτυχιακής.

Ευχαριστώ τον καθηγητή *Αθανάσιο ΚΟΡΑΚΗ.*

Τέλος ευχαριστώ τους *Αντώνης ΚΑΛΚΑΝΟΦ* και *Χρήστος ΘΕΟΛΟΓΟΥ*, οι οποίοι με βοήθησαν και με καθοδήγησαν από την αρχή μέχρι το τέλος.

Πίνακας περιεχομένων

Abstract 2
Ευχαριστίες 3
Έλεγχος και Δοκιμή 5
Ανάπτυξη Οδηγούμενη από Ελέγχους 5
Τα είδη των δοκιμών 6
Unit Tests 7
Stubs και Mocks 8
Integration Tests 9
Διαφορετικές προσεγγίσεις για integration testing 9
End-to-End Tests 10
Καταγραφή λειτουργιών χρήστη 10
Δημιουργία υποθέσεων 11
Κατασκευή περιπτώσεων δοκιμής 11
Συνεχής Ενσωμάτωση/ Συνεχής Παραγωγή & Παράδοση 12
Περιορισμοί και δυσκολίες 13
Εφαρμογή και αξιολόγηση των μεθόδων ελέγχου σε υπηρεσία Iot 14
Εισαγωγή στην Vue.js 14
Εισαγωγή στις υπηρεσίες Internet Of Things 16
Η IoT υπηρεσία Agronit 16
Unit testing & Integration testing / Μέθοδοι και αξιολόγηση 17
Εφαρμογή και Παρατηρήσεις 20
End-to-end testing / Μέθοδοι και αξιολόγηση 21
Εφαρμογή Μεθόδων 26
Επίλογος 28

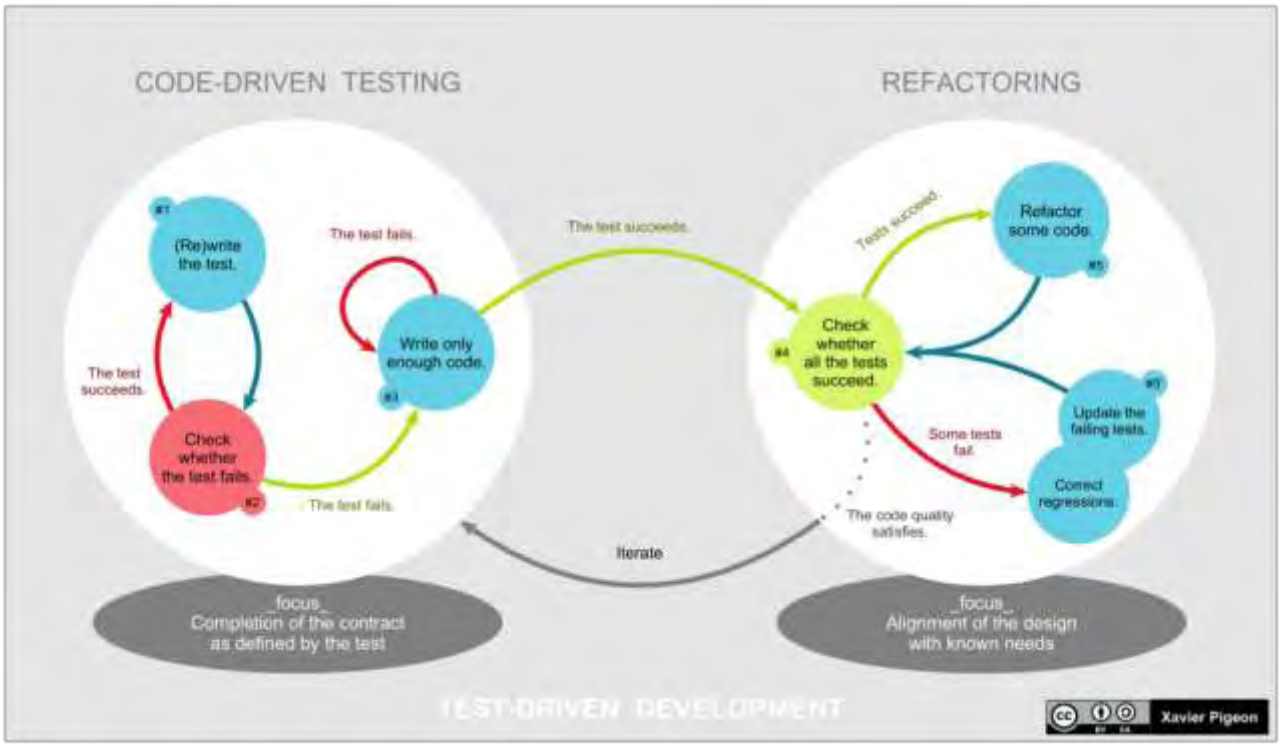
Έλεγχος και Δοκιμή

Ως έλεγχο (test) ορίζουμε ένα σύνολο κώδικα ο οποίος γράφεται για μια εφαρμογή με σκοπό να εξετάσει την λειτουργία της και να διασφαλίσει ότι η λειτουργία αυτή ανταποκρίνεται στο σχεδιασμό της και συμπεριφέρεται όπως προβλέπεται. Είναι ανεξάρτητος από τον κώδικα παραγωγής, καθώς δεν προσφέρει επιπλέον λειτουργικότητα και έχει βοηθητικό και συμπληρωματικό ρόλο στην διαδικασία της ανάπτυξης.

Ως δοκιμή (testing) ορίζουμε την εκτέλεση των ανωτέρω ελέγχων για να εξακριβωθεί αν ολοκληρώνονται επιτυχώς. Αυτό γίνεται σε κάποιο περιβάλλον εκτέλεσης που ορίζει ο προγραμματιστής.

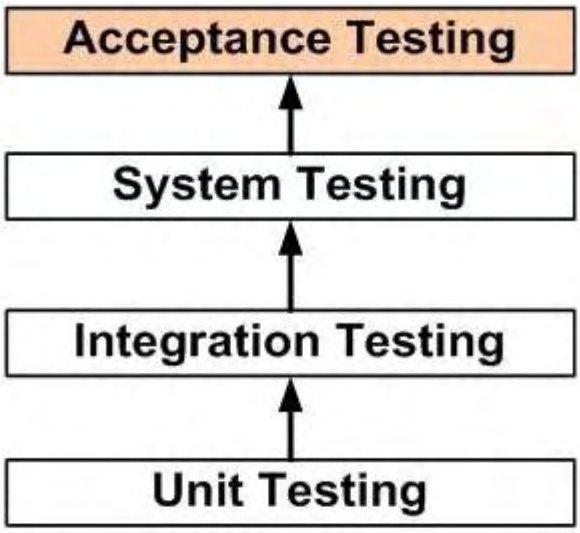
Ανάπτυξη Οδηγούμενη από Ελέγχους

Η οδηγούμενη από ελέγχους ανάπτυξη αποτελεί μια επαναστατική προσέγγιση για την ανάπτυξη λογισμικού. Αποτελεί μια τεχνική προγραμματισμού με βασικό χαρακτηριστικό χαρακτηριστικό την “test-first ανάπτυξη”. Η ιδέα της ανάπτυξης αυτής συνοψίζεται ως εξής:



Αρχικά δημιουργείται ένας έλεγχος για κάθε μια λειτουργικότητα της εφαρμογής. Προστίθεται βασικός κώδικας στο σώμα του κάθε ελέγχου ο οποίος κάνει τον κάνει να περάσει με επιτυχία. Ο κώδικας ελέγχου στη συνέχεια αναδιαμορφώνεται ώστε να εστιάζει στον κώδικα παραγωγής, δηλαδή κάποια συγκεκριμένη λειτουργικότητα της εφαρμογής. Ο προγραμματιστής τρέχει όλους τους ελέγχους και εφόσον κάποιος αποτύχει, κάνει τις απαραίτητες αλλαγές στον κώδικα παραγωγής μέχρι να τρέξουν όλοι επιτυχώς.

Τα είδη των δοκιμών



Κατά την εφαρμογή της οδηγούμενης από ελέγχους ανάπτυξης εμφανίζονται τρία διαφορετικά στάδια δοκιμής της εξεταζόμενης εφαρμογής. Δημιουργούνται πρώτα οι έλεγχοι για τις μικρότερες δοκιμαστικές μονάδες, γνωστοί ως “unit

tests”, έπειτα για τις σύνθετες συμπεριφορές μεταξύ αυτών (“integration tests”) και μια ένας συνολικός έλεγχος ο οποίος εξετάζει ολόκληρη την εφαρμογή “από άκρη σε άκρη” γνωστός ως “system testing” ή “end-to-end testing”.

Unit Tests

Οι έλεγχοι μονάδας αποτελούν αυτοματοποιημένους ελέγχους οι οποίοι διασφαλίζουν ότι το κομμάτι μιας εφαρμογής, γνωστό ως “μονάδα”, ανταποκρίνεται στο σχεδιασμό της και συμπεριφέρεται όπως προβλέπεται. Στον διαδικαστικό προγραμματισμό, μια μονάδα είναι συνηθέστερα μια μεμονωμένη λειτουργία ή διαδικασία. Στον αντικειμενοστραφή προγραμματισμό, μια μονάδα είναι συχνά μια ολόκληρη διεπαφή, όπως μία κλάση, αλλά μπορεί να είναι και μια μεμονωμένη μέθοδος.

Το στάδιο συγγραφής και εφαρμογής των παραπάνω ελέγχων ονομάζεται “Unit Testing”. Είναι το πιο σημαντικό στάδιο καθώς εξετάζει την σωστή λειτουργία των θεμέλιων λίθων της εφαρμογής.

Προκειμένου οι έλεγχοι μονάδας να είναι αποτελεσματικοί, πρέπει να πληρούν ορισμένες προϋποθέσεις. Οι σωστοί έλεγχοι μονάδας:

- Έχουν γρήγορους χρόνους εκτέλεσης.
- Τρέχουν σε απομόνωση (θα πρέπει να είναι δυνατή η αναδιάταξή τους).
- Είναι ανεξάρτητοι (η εκτέλεση ενός ελέγχου δεν εξαρτάται από την εκτέλεση άλλου).
- Χρησιμοποιούν δεδομένα τα οποία τους κάνουν ευανάγνωστους και ευκατανόητους.
- Χρησιμοποιούν πραγματικά δεδομένα (π.χ. αντίγραφα από τον κώδικα παραγωγής) όταν είναι απαραίτητο.
- Αντιπροσωπεύουν ένα βήμα προς τον γενικό σκοπό της εφαρμογής.

Εικόνα 1.1: Παράδειγμα μιας συνάρτησης/μονάδας

Στην εικόνα 1.1 η upperMerge() ειναι μια

λειτουργία-συνάρτηση της εφαρμογής η οποία δέχεται σαν παράμετρο μια συμβολοσειρά και μετατρέπει όλους τους χαρακτήρες της σε κεφαλαίους, αφαιρώντας τους κενούς χαρακτήρες. Το unit test για τον κώδικα αυτό φαίνεται στην εικόνα 1.2. Αποτελεί ταυτόχρονα μια “μονάδα” επειδή δεν υπάρχει αλληλεξάρτηση της upperMerge με κάποια άλλη λειτουργία της εφαρμογής. Στην εικόνα 1.3 φαίνεται η εκτέλεση του ελέγχου μονάδας και το αποτέλεσμα εκτέλεσης.

Έστω ότι ο κώδικας του σχήματος 1.1 μεταβάλλεται κατά τέτοιο τρόπο ώστε το αποτέλεσμα της upperMerge να γράφεται σε αρχείο, χρησιμοποιώντας την μονάδα “FileWriter” της εφαρμογής. Τώρα η upperMerge δεν θεωρείται πλέον μονάδα επειδή δημιουργήθηκε αλληλοεξάρτηση με την fileWriter. Πρέπει με κάποιο τρόπο να απομονωθεί από αυτήν. Η FileWriter αποτελεί μια “προβληματική” λειτουργία όσον αφορά τον έλεγχο.

Εικόνα 1.2: Έλεγχος μονάδας για τον κώδικα της εικόνας 1.1

Stubs και Mocks

Λύση στο παραπάνω πρόβλημα δίνουν τα αντικείμενα εξομοίωσης (mock objects ή mocks) και τα stubs. Τα αντικείμενα εξομοίωσης χρησιμοποιούνται στον κώδικα ελέγχου μονάδας με σκοπό να μετατρέψουν μια λειτουργία της εφαρμογής σε μονάδα. Απώτερος στόχος και των δύο τεχνικών εξομοίωσης είναι να ελαχιστοποιηθούν οι εξαρτήσεις που περιέχονται μέσα σε κλάσεις ή συναρτήσεις έτσι ώστε ο κώδικας να γίνει πιο απλός και επικεντρωμένος σε αυτό που προσπαθεί να αποδείξει.

Τα mocks εξασφαλίζουν ότι η συνάρτηση που προκαλεί την αλληλοεξάρτηση καλείται με τις σωστές παραμέτρους και έπειτα παρακάμπτουν την εκτέλεσή της, προχωρώντας στον υπόλοιπο κώδικα.

Εικόνα 1.3: Συνάρτηση που δεν μπορεί να θεωρηθεί μονάδα

Καταγράφουν δηλαδή και επιβεβαιώνουν προσδοκίες.

Τα stubs από την άλλη έχουν διαφορετικό σκοπό. Επιτρέπουν τον προγραμματιστή να αντικαταστήσει την κατάσταση και συμπεριφορά μιας λειτουργίας της οποίας ο κώδικας δεν έχει συνήθως γραφεί ακόμη. Υπάρχει η δυνατότητα να δημιουργηθεί μια κλάση η οποία υλοποιεί την “προβληματική” κλάση ή διεπαφή. Οι μέθοδοι αυτής της κλάσης θα έχουν όλες τις εξόδους προκαθορισμένες από τον προγραμματιστή.

Integration Tests

Μετά την επιτυχή ολοκλήρωση όλων των ελέγχων μονάδας ακολουθεί το επόμενο στάδιο δοκιμών. Σε αυτό το στάδιο οι ενότητες λογισμικού συνδυάζονται μεταξύ τους και δοκιμάζονται ως ομάδα. Πρόκειται για τον έλεγχο ενοποίησης (integration testing) ο οποίος διεξάγεται για να αξιολογηθεί η συμμόρφωση ενός συστήματος ή ενός κατασκευαστικού στοιχείου με συγκεκριμένες λειτουργικές απαιτήσεις. Πραγματοποιείται μετά τη δοκιμή μονάδας και πριν από τη δοκιμή συστήματος. Ο έλεγχος ενσωμάτωσης έχει στην είσοδό του κομμάτια κώδικα τα οποία έχουν υποβληθεί σε δοκιμή μονάδων, τα ομαδοποιεί σε μεγαλύτερα μεγέθη, εφαρμόζει δοκιμές που ορίζονται σε ένα πρόγραμμα δοκιμών ενοποίησης και παράγει ως έξοδο το ελεγμένο και ενοποιημένο σύστημα που είναι πλέον έτοιμο για την τελική δοκιμή συστήματος.

Διαφορετικές προσεγγίσεις για integration testing

- Η *Big Bang είναι μια προσέγγιση* όπου όλες ή οι περισσότερες μονάδες της εφαρμογής συνδυάζονται μεταξύ τους και δοκιμάζονται με την μία. Αυτή η προσέγγιση χρησιμοποιείται όταν η ομάδα προγραμματιστών που έχει αναλάβει τις δοκιμές παραλαμβάνει ένα έτοιμο λογισμικό όπου ο κώδικας παραγωγής έχει ολοκληρωθεί και εκκρεμεί το testing. Εδώ εξετάζονται μόνο οι αλληλεπιδράσεις των μονάδων μεταξύ τους και όχι όλο το σύστημα σαν σύνολο.

- Η *Bottom Up / Top Down* είναι δυο αντιδιαμετρικές προσεγγίσεις. Στην πρώτη, όλες οι μονάδες χαμηλότερης ιεραρχίας εξετάζονται ανεξάρτητα η μία από την άλλη και έπειτα εξετάζονται οι λειτουργίες οι οποίες εξαρτώνται από αυτές τις μονάδες κ.ο.κ. Στην top-down προσέγγιση το testing ξεκινάει από τις

λειτουργίες της εφαρμογής που βρίσκονται ψηλά στην ιεραρχία και αναπόφευκτα περιέχουν πληθώρα αλληλεξαρτήσεων με χαμηλότερου επιπέδου λειτουργίες. Οι λειτουργίες που βρίσκονται στο πιο χαμηλό επίπεδο εξομοιώνονται με την χρήση των stubs ή mocks.

- Η *Sandwich/Hybrid* είναι ένας συνδυασμός των Top Down και Bottom Up.

End-to-End Tests

Η δοκιμή από άκρη σε άκρη είναι μια μεθοδολογία δοκιμής λογισμικού κατά την οποία ελέγχεται η ροή μιας εφαρμογής από την αρχή μέχρι το τέλος. Στόχος είναι να πραγματοποιηθεί μια προσομοίωση χρήσης της εφαρμογής, όπως ακριβώς θα την χειριζόταν ένας χρήστης. Αποτελεί το τελικό στάδιο των δοκιμών όπου τα επιμέρους τμήματα του συστήματος ταυτοποιούνται για την ενσωμάτωσή τους στο συνολικό σύστημα και επιβεβαιώνεται η ακεραιότητα των δεδομένων. Χρησιμοποιούνται δεδομένα που προσομοιώνουν περιβάλλοντα πραγματικού χρόνου. Στα περιβάλλοντα αυτά είναι απαραίτητη η ακριβής μετάδοση πληροφορίας ώστε να εξασφαλιστεί η επικοινωνία της εφαρμογής με το δίκτυο, το hardware, την βάση δεδομένων καθώς και με άλλες εφαρμογές.

Το framework σχεδιασμού της δοκιμής από-άκρη-σε-άκρη αποτελείται από τρία βασικά μέρη:

➔ Καταγραφή όλων των ορισμένων από τον χρήστη λειτουργιών

➔ Δημιουργία υποθέσεων

➔ Κατασκευή υποθετικών σεναρίων χρήσης

Καταγραφή λειτουργιών χρήστη

Σε αυτό το στάδιο καταγράφονται τα χαρακτηριστικά του συστήματος και τα διασυνδεδεμένα μεταξύ τους επιμέρους τμήματα. Καταγράφονται οι σχέσεις μεταξύ των συναρτήσεων και τα δεδομένα εισόδου/εξόδου για κάθε χαρακτηριστικό ή συνάρτηση. Η συνάρτηση έχει την δυνατότητα να είναι ανεξάρτητη και επαναχρησιμοποιήσιμη. Ένα παράδειγμα αποτελεί η χρήση εφαρμογής μιας τράπεζας. Εδώ θα καταγραφούν οι εξής λειτουργίες:

α) Είσοδος στο προσωπικό λογαριασμό του τραπεζικού συστήματος του χρήστη

β) Αποσύνδεση από τον λογαριασμό.

γ) Έλεγχος του υπολοίπου του λογαριασμού

δ) Μεταφορά ενός χρηματικού ποσού από τον προσωπικό λογαριασμό του χρήστη σε έναν άλλο (αλληλεπίδραση με ξένο υποσύστημα).

ε) Έλεγχος εκ νέου του υπολοίπου για να εξακριβωθεί ότι αφαιρέθηκε το ποσό.

Δημιουργία υποθέσεων

Αφού έχουν πλέον καταγραφεί όλες οι λειτουργίες ακολουθεί η κατασκευή ενός συνόλου εισόδων και εξόδων για κάθε μια από αυτές. Οι υποθέσεις αυτές καλύπτουν επίσης και θέματα ακολουθίας συμβάντων και συγχρονισμού. Για την παραπάνω λειτουργία "Έλεγχος εκ νέου του υπολοίπου" μπορούν να δημιουργηθούν δύο υποθέσεις:

- Έλεγχος του υπολοίπου μετά από εικοσιτέσσερις ώρες (αν το ποσό μεταφέρθηκε σε άλλη τράπεζα)
- Εξακρίβωση ότι εμφανίζεται μήνυμα σφάλματος όταν ο χρήστης επιχειρεί να μεταφέρει ποσό μεγαλύτερο από το υπόλοιπό του.

Κατασκευή περιπτώσεων δοκιμής

Με την ολοκλήρωση των δυο παραπάνω σταδίων αρχίζει η κατασκευή σεναρίων χρήσης της εφαρμογής. Για το παραπάνω παράδειγμα της εφαρμογής τραπεζικού λογαριασμού έχουμε το εξής ακολουθιακό σενάριο:

➔ Σύνδεση στο σύστημα

➔ Έλεγχος του διαθέσιμου υπολοίπου

➔ Μεταφορά του τραπεζικού υπολοίπου

Για το παραπάνω παράδειγμα δημιουργούνται μια ή περισσότερες περιπτώσεις ελέγχου οι οποίες χρησιμοποιούν τις υποθέσεις που ορίστηκαν στο προηγούμενο βήμα.

Η μεγαλύτερη πρόκληση γι αυτό το τελικό στάδιο δοκιμών είναι να υπάρχει επαρκής γνώση ολόκληρου του συστήματος μαζί με τα υποσυστήματά του.

Συνεχής Ενσωμάτωση/ Συνεχής Παραγωγή & Παράδοση

Η **Συνεχής Ενσωμάτωση/Συνεχής Παραγωγή (CI/CD)** είναι μια μέθοδος παραγωγής λογισμικού όπου εισάγεται ο αυτοματισμός σε όλα τα στάδια της παραγωγής. Στην σύγχρονη ανάπτυξη λογισμικού, στόχος αποτελεί η ταυτόχρονη εργασία πολλαπλών προγραμματιστών πάνω σε διαφορετικά χαρακτηριστικά της ίδιας εφαρμογής. Αν όμως η πολιτική ενός οργανισμού επιβάλλει την συνένωση όλου του παραχθέντος κώδικα την ίδια χρονική στιγμή (γνωστή ως μέρα συνένωσης), τότε η εργασία που προκύπτει ενδέχεται να είναι κουραστική και απαιτητική σε χρόνο, καθώς υπάρχει περίπτωση οι αλλαγές ενός προγραμματιστή που δουλεύει σε απομόνωση να συγκρουστούν με τις αλλαγές που γίνονται ταυτόχρονα από άλλους προγραμματιστές.

Η **Συνεχής Ενσωμάτωση** επιτρέπει τους προγραμματιστές να εφαρμόζουν παράλληλα ο ένας με τον άλλον αλλαγές στον κώδικα. Μόλις ο προγραμματιστής συγγράψει και ανεβάσει κώδικα, αυτός επαληθεύεται αυτόματα στο στάδιο κατασκευής της εφαρμογής, εκτελώντας κάποια επίπεδα αυτοματοποιημένων δοκιμών. Μεγάλο ποσοστών των δοκιμών αυτών είναι οι δοκιμές μονάδας και ενσωμάτωσης. Με αυτόν τον τρόπο επιβεβαιώνεται ότι οι αλλαγές δεν έχουν οδηγήσει σε λανθασμένη λειτουργία της εφαρμογής. Αν οι αυτοματοποιημένες δοκιμές εντοπίσουν κάποια σύγκρουση μεταξύ καινούργιου και του ήδη υπάρχοντα κώδικα, η συνεχής ενσωμάτωση διευκολύνει και επιταχύνει την διαδικασία αποσφαλμάτωσης.

Μετά το στάδιο της αυτόματης εφαρμογής των δοκιμών, ακολουθεί η αποθήκευση του επαληθευμένου κώδικα σε ένα κοινώς συμφωνημένο διαδικτυακό αποθηκευτικό χώρο, γνωστό ως “repository”. Η **Συνεχής Παράδοση** αναλαμβάνει την αυτοματοποίηση της διαδικασίας αυτής. Στόχος της μεθόδου είναι να υπάρχει μια βάση κώδικα, η οποία να είναι πάντα έτοιμη για την μετεξέλιξή της σε ένα περιβάλλον παραγωγής. Στο τέλος αυτής της μεθόδου, η ομάδα ανάπτυξης είναι ικανή να μεταφέρει την εφαρμογή στο στάδιο παραγωγής εύκολα και γρήγορα.

Το τελικό στάδιο μιας ορθής CI/CD ροής είναι η **Συνεχής Ανάπτυξη**. Αποτελεί επέκταση της συνεχούς παράδοσης και αυτοματοποιεί την διαδικασία της μεταφοράς της εφαρμογής στο στάδιο της παραγωγής. Στηρίζεται σε μεγάλο βαθμό στις σωστά σχεδιασμένες αυτοματοποιημένες δοκιμές. Στην πράξη, συνεχής ανάπτυξη σημαίνει ότι οι αλλαγές που επέφερε ο προγραμματιστής μπορούν άμεσα να θεωρηθούν ως μόνιμες (υποθέτοντας ότι έχουν περάσει επιτυχώς το στάδιο των αυτοματοποιημένων δοκιμών). Αυτό σημαίνει ότι διευκολύνεται η παραλαβή και ενσωμάτωση προτάσεων από τους χρήστες της εφαρμογής.

Οι μέθοδοι αυτοί στο σύνολό τους καθιστούν την ανάπτυξη εφαρμογών λιγότερο ριψοκίνδυνη και ενθαρρύνουν την παράλληλη συνεισφορά εργασίας, μεγιστοποιώντας έτσι το προγραμματιστικό δυναμικό της ομάδας και μειώνοντας κατα συνέπεια τον συνολικό χρόνο ανάπτυξης.

Περιορισμοί και δυσκολίες

Η δοκιμή του λογισμικού είναι ένα συνδυαστικό πρόβλημα. Για παράδειγμα, κάθε δυαδική δήλωση απόφασης απαιτεί τουλάχιστον δύο δοκιμές: μία με αποτέλεσμα "αληθές" και μία με αποτέλεσμα "ψευδές". Ως αποτέλεσμα, για κάθε γραφή γραμμής κώδικα, οι προγραμματιστές χρειάζονται συχνά πολλαπλές σειρές κώδικα δοκιμής. Αυτό προφανώς απαιτεί χρόνο και η επένδυσή του μπορεί να μην αξίζει τον κόπο. Υπάρχουν προβλήματα που δεν μπορούν να δοκιμαστούν εύκολα - για παράδειγμα αυτά που είναι μη ντετερμινιστικά ή αυτά τα οποία κάνουν χρήση του παράλληλου προγραμματισμού (πολυνηματικά). Επιπλέον, ο κώδικας για μια δοκιμή μονάδας είναι πιθανό να περιέχει τόσα προβλήματα όσα ο κώδικας που δοκιμάζει.

Μια άλλη πρόκληση που σχετίζεται με τη σύνταξη των δοκιμών είναι η δυσκολία δημιουργίας ρεαλιστικών και χρήσιμων δοκιμών. Είναι απαραίτητο να δημιουργηθούν σχετικές αρχικές συνθήκες έτσι ώστε το μέρος της υπό δοκιμή εφαρμογής να συμπεριφέρεται ως μέρος του πλήρους συστήματος. Εάν αυτές οι αρχικές συνθήκες δεν έχουν ρυθμιστεί σωστά, η δοκιμή δεν θα ελέγξει τον κώδικα σε

ρεαλιστικό πλαίσιο, πράγμα που μειώνει την αξία και την ακρίβεια των αποτελεσμάτων των δοκιμών.

Για να επιτευχθούν τα επιδιωκόμενα οφέλη από τις δοκιμές, απαιτείται αυστηρή πειθαρχία καθόλη τη διάρκεια της διαδικασίας ανάπτυξης λογισμικού. Είναι σημαντικό να τηρούνται προσεκτικά αρχεία όχι μόνο για τις δοκιμές που έχουν εκτελεστεί αλλά και για όλες τις αλλαγές που έγιναν στον κώδικα παραγωγής αυτής ή οποιασδήποτε άλλης μονάδας στο λογισμικό. Η χρήση ενός συστήματος ελέγχου έκδοσης είναι απαραίτητη. Εάν μια μεταγενέστερη έκδοση της μονάδας αποτύχει σε μια συγκεκριμένη δοκιμή που είχε προηγουμένως περάσει, το λογισμικό ελέγχου έκδοσης μπορεί να παράσχει μια λίστα με τις αλλαγές πηγαίου κώδικα (εάν υπάρχουν) που έχουν εφαρμοστεί στη μονάδα από τότε.

Είναι επίσης απαραίτητο να εφαρμοστεί μια βιώσιμη διαδικασία για να διασφαλιστεί ότι οι αποτυχίες των δοκιμαστικών περιπτώσεων επανεξετάζονται τακτικά και αντιμετωπίζονται αμέσως. Εάν μια τέτοια διαδικασία δεν εφαρμοστεί και δεν ενσωματωθεί στη ροή εργασίας της ομάδας, η εφαρμογή θα εξελίσσεται εκτός συγχρονισμού με τη μονάδα δοκιμών μονάδας, αυξάνοντας τα ψευδώς θετικά και μειώνοντας την αποτελεσματικότητα της δοκιμαστικής σουίτας.

Εφαρμογή και αξιολόγηση των μεθόδων ελέγχου σε υπηρεσία Iot

Εισαγωγή στην Vue.js

Η **Vue.js** είναι ένα ανοιχτού-κώδικα προγραμματιστικό πλαίσιο προβολής μοντέλων σε Javascript, το οποιο κατασκευάζει διεπαφές χρήστη και εφαρμογές μίας-σελίδας. Δημιουργηθηκε από τον Evan You και συντηρείται από τον ίδιο και από τα υπόλοιπα ενεργά στελέχη, τα οποία προέρχονται από εταιρίες όπως η Netlify και η Netguru.

Η Vue.js διαθέτει μια αυξανόμενα υιοθετίσημη αρχιτεκτονική η οποία εστιάζει στην απόδοση με την βοήθεια δηλώσεων και στην σύνθεση των συστατικών σελίδας.

```HTML
<template>
  <view>
    <text>{{ message }}</text>
  </view>
</template>
```

```JS
<script>
export default {
  data: function() {
    return {
      message: "Hello World"
    };
  }
};
</script>
```

Εικόνα 2.1: Παράδειγμα δηλωτικής απόδοσης στην Vue.js

```HTML
<p v-if="seen">Now you see me</p>
```

Εικόνα 2.2: Παράδειγμα χρήσης οδηγίας στην Vue.js

Προχωρημένα χαρακτηριστικά τα οποία είναι προαπαιτούμενα στην ανάπτυξη περίπλοκων λειτουργιών, όπως δυναμική απόδοση συστατικών επιπέδου-σελίδας, διαχείρηση κατάστασης και κατασκευή εργαλείων, προσφέρονται διαμέσου επισήμων βιβλιοθηκών και πακέτων που συντηρούνται επίσημα, με την Nuxt.js να αποτελεί μία από τις πιο δημοφιλείς λύσεις.

Η Vue.js επιτρέπει τον προγραμματιστή να επεκτείνει την HTML με χαρακτηριστικά HTML τα οποία ονομάζονται οδηγίες. Οι οδηγίες προσφέρουν λειτουργικότητα στις εφαρμογές HTML. Είναι ήδη προ-εγκατεστημένες, αλλά ο χρήστης έχει την δυνατότητα να δημιουργήσει δικές του. Στην Εικόνα 2.2, η οδηγία “v-if” θα αφαιρέσει ή εμφανίσει το στοιχείο p, ανάλογα με την αλήθεια της τιμής της έκφρασης “seen”.

Εισαγωγή στις υπηρεσίες Internet Of Things

Το **Διαδίκτυο των πραγμάτων** ή **Ίντερνετ των πραγμάτων** (**Internet of things**) αποτελεί το δίκτυο επικοινωνίας πληθώρας συσκευών, οικιακών συσκευών, αυτοκινήτων καθώς και κάθε αντικειμένου που ενσωματώνει ηλεκτρονικά μέσα, λογισμικό, αισθητήρες και συνδεσιμότητα σε δίκτυο ώστε να επιτρέπεται η σύνδεση και η ανταλλαγή δεδομένων. Απλούστερα, η φιλοσοφία του IoT είναι η σύνδεση όλων των ηλεκτρονικών συσκευών μεταξύ τους (σε τοπικό δίκτυο) ή με δυνατότητα σύνδεσης στο διαδίκτυο.

Η έννοια "Things" (πράγματα) δεν είναι αυστηρά συνδεδεμένη με ορισμένα προϊόντα. Αναφέρεται σε μία ευρεία ποικιλία συσκευών εντελώς διαφορετικά μεταξύ τους, όπως για παράδειγμα αυτοκίνητα με ενσωματωμένους αισθητήρες, κάμερες, κλιματιστικά, φώτα, συστήματα ασφαλείας, smartwatches ακόμα και αυτοκίνητα των οποίων οι περίπλοκοι αισθητήρες εντοπίζουν αντικείμενα στην πορεία τους. Είναι μερικά από τα πολλά προϊόντα τεχνολογίας. Βασικό χαρακτηριστικό όλων είναι η σύνδεση μεταξύ τους με απώτερο σκοπό την δυνατότητα του χρήστη να τα ελέγχει από έναν υπολογιστή ή κινητό. Ο όρος Internet of Things αποδόθηκε την δεκαετία του 1990 από τον Kevin Ashton.

Το Διαδύκτιο των πραγμάτων είναι μία από τις τρεις κορυφαίες τεχνολογικές εξελίξεις της επόμενης δεκαετίας (μαζί με το Mobile Internet και την αυτοματοποίηση του knowledgework) και αποτελεί το επόμενο μεγάλο βήμα στον χώρο της τεχνολογίας.Ο όρος Internet of Things επινοήθηκε στα τέλη της δεκαετίας του 1990 από τον επιχειρηματία Kevin Ashton.Ο Ashton, ο οποίος είναι ένας από τους ιδρυτές του Auto-ID center στο MIT, ήταν μέρος μιας ομάδας που ανακάλυψε τον τρόπο να συνδέσει τα αντικείμενα με το Διαδίκτυο μέσω μιας ετικέτας RFID.

Η IoT υπηρεσία Agronit

Η Agronit αποτελεί μια υπηρεσία Internet Of Things. Είναι μια πλατφόρμα η οποία διευθύνει και αυτοματοποιεί την διαδικασία εγκατάστασης και παρακολούθησης ενός ασύρματου δίκτυου αισθητήρων. Κάθε χρήστης έχει την δυνατότητα να οργανώσει την δικιά του τοπολογία από αισθητήρες και να συλλέξει δεδομένα και στατιστικά.

Η υπηρεσια Agronit επιλέχτηκε για την δοκιμή των λειτουργιών της καθώς είναι γραμμένη σε Javascript, η οποία υποστηρίζεται και προτείνεται από τις περισσότερες βιβλιοθήκες ελέγχου λογισμικού. Επιπλέον το front-end μέρος της έχει συνταχτεί σε Vue.js, η οποία είναι αρκετά σαφής και απλή και δεν θα ξενίσει κάποιον που έρχεται σε πρώτη επαφή με τις δοκιμές.

Jest

Η **Jest** είναι μια βιβλιοθήκη η οποία δημιουργήθηκε και συντηρείται από το Facebook. Έγινε η πιο δημοφιλής βιβλιοθήκη το 2017. Βασίζεται στην βιβλιοθήκη **Jasmine.** Το Facebook αντικαθιστά τακτικά μεγάλος μέρος της λειτουργικότητάς του και προσθέτει καινούργια χαρακτηριστικά.

- **Επιδόσεις** - Η Jest είναι πιο γρήγορη για μεγάλα έργα με πολλά αρχεία ελέγχων και αυτό το πετυχαίνει με την υλοποίηση ενός έξυπνου μηχανισμού παράλληλων ελέγχων.
- **Διεπαφή χρήστη (UI)** – Απλή και βολική.
- **Globals**- Δημιουργεί αυτόματα Globals για τους ελέγχους. Τα Globals είναι μέθοδοι και αντικείμενα τα οποία παρέχονται από την βιβλιοθήκη και δεν χρειάζεται να γίνει η εισαγωγή τους. Αυτό μπορεί να θεωρηθεί ως μειονέκτημα επειδή κάνει τα tests λιγότερο ευέλικτα και ελέγξιμα, αλλά στις περισσότερες περιπτώσεις διευκολύνει τον προγραμματιστή.
- **Έλεγχος με χρήση στιγμιοτύπων –** Το “jest-snapshot” αναπτύσσεται και συντηρείται από το Facebook, αλλά μπορεί να χρησιμοποιηθεί σε οποιοδήποτε προγραμματιστικό πλαίσιο σαν μέρος της ενσωμάτωσης του πλαισίου του εργαλείου ή χρησιμοποιώντας τα σωστά plugins.
- **Βελτιωμένη προσομοίωση λειτουργιών** – Το Jest παρέχει εργαλεία για την προσομοίωση πολύπλοκων λειτουργιών το οποίο έχει σαν αποτέλεσμα την βελτίωση της ταχύτητας. Για παράδειγμα μια συνάρτηση που μεταξύ άλλων κάνει αίτηση μέσω δικτύου, μπορεί να παρακάμψει την αίτηση αυτή μέσω της λειτουργίας mock.
- **Αξιοπιστία –** Παρόλο που είναι μια σχετικά νέα βιβλιοθήκη, η έκδοση της Jest σταθεροποιήθηκε και θεωρείται πλέον αξιόπιστη. Υποστηρίζεται από όλα τα ενσωματωμένα περιβάλλοντα ανάπτυξης και τα εργαλεία.
- **Πληθώρα εργαλείων** – Περιέχει assertions*, spies**, και mocks τα οποία είναι παρεμφερή με αυτά που παρέχει η βιβλιοθήκη Sinon Οι βιβλιοθήκες μπορούν εύκολα να χρησιμοποιηθούν σε περίπτωση που χρειάζονται κάποια ιδιαίτερα χαρακτηριστικά.

* Ως assertion ορίζεται μια λογική έκφραση, η οποία επαληθεύει μια προσδοκία για το υπό έλεγχο αντικείμενο.
**Ως spy ορίζεται μια λειτουργία η οποία καταγράφει τα ορίσματα, την τιμή που επιστρέφεται, τυχόν σφάλματα που προκλήθηκαν, καθώς και τον αριθμό κλήσεων μιας συγκεκριμένης συνάρτησης.

Jasmine

Η **Jasmine** είναι μια βιβλιοθήκη για ελέγχους, πάνω στην οποία βασίζεται η βιβλιοθήκη Jest. Δημιουργήθηκε πολύ πιο πριν από την Jest και υπάρχουν πολλά άρθρα, εργαλεία και εγχειρίδια τα οποία δημιουργήθηκαν όλα από την κοινότητα. Η ομάδα της Angular προτείνει την χρήση της Jasmine αντί της Jest , παρόλο που η Jest είναι ικανή να τρέξει αποτελεσματικά tests πάνω σε κώδικα γραμμένο σε Angular.

- **Έτοιμη για χρήση** – Περιέχει όλα τα απαραίτητα μέσα για την δημιουργία test.
- **Globals** – Περιέχει όλα τα σημαντικά χαρακτηριστικά στο φάσμα των globals.
- **Κοινότητα** - Υπάρχει στην αγορά από το 2009 και έχει συγκεντρώσει ένα αχανές ποσό από άρθρα, προτάσεις και εργαλεία που βασίζονται σε αυτήν.
- **Angular**- Έχει ευρέως διαδεδομένη υποστήριξη για την Angular για όλες τις εκδόσεις της και προτείνεται από το επίσημο εγχειρίδιο χρήσης της Angular.

Mocha

Η **Mocha** είναι η πιο διαδεδομένη βιβλιοθήκη. Αντίθετα με την Jasmine, χρησιμοποιείται από τον προγραμματιστή **σε συνδυασμό** με εργαλεία άλλων βιβλιοθηκών, και κυρίως εργαλεία για mocking και spying (συνήθως οι βιβλιοθήκες Sinon και Chai οι οποίες εξετάζονται παρακάτω).
Αυτό σημαίνει ότι η Mocha είναι λίγο πιο δύσκολο να εγκατασταθεί και διαιρείται σε πληθώρα βιβλιοθηκών, αλλά είναι πολύ ευέλικτη και ανεκτή σε προσθήκες και επεκτάσεις.

Για παράδειγμα, έστω ότι ο προγραμματιστής χρησιμοποιεί την βιβλιοθήκη Chai για τις επαληθεύσεις (assertions) . Έστω ότι θέλει να βρει διαφορετικό τρόπο με τον οποίο να γίνονται αυτές οι επαληθεύσεις, έχει την δυνατότητα να αντικαταστήσει την βιβλιοθήκη επαληθεύσεων της Chai με μία δικιά του. Αυτό είναι υλοποιήσιμο και στην Jasmine, αλλά στην Mocha αυτή η αλλαγή θα είναι πιο προφανής και ξεκάθαρη.

- **Κοινότητα** – Περιέχει πληθώρα από plugins και επεκτάσεις για τον έλεγχο ιδιαίτερων σεναρίων.
- **Επεκτασιμότητα** – Πολύ επεκτάσιμη, σε βαθμό που τα plugins, οι επεκτάσεις και οι βιβλιοθήκες έχουν σχεδιαστεί για να τρέχουν πάνω της.
- **Υποστήριξη** – Έχει ενθουσιώδη κοινότητα και πληθώρα από εγχειρίδια και tutorials που μπορούν να βρεθούν στο διαδύκτιο.
- **Globals –** Δημιουργεί Globals από μόνη της, αλλά όχι επαληθεύσεις, spies και mocks όπως η Jasmine.

Sinon & Chai

Η Chai είναι η πιο δημοφιλής βιβλιοθήκη για τις επαληθεύσεις. Περιέχει πολλά plugins και επεκτάσεις και ένα αχανές εγχειρίδιο χρήσης με όλες τις συναρτήσεις και λειτουργίες που προσφέρει. Η Sinon παρέχει spies για ελέγχους, καθώς και stubs & mocks για την Javascript τα οποία είναι συμβατά με οποιοδήποτε περιβάλλον ελέγχου.

Ava

Η **Ava** είναι μια μινιμαλιστική βιβλιοθήκη για ελέγχους η οποία έχει την δυνατότητα να τρέξουν τα tests παράλληλα.

- **Έτοιμη για χρήση** – Περιέχει τα πάντα για την άμεση δημιουργία test (εκτός από spying τα οποία μπορούν εύκολα να προστεθούν από το χρήστη). Τρέχει σε Node.js.
- **Globals** – Δεν δημιουργεί δικά της test globals, έτσι ώστε να υπάρχει μεγαλύτερος έλεγχος στην διαδικασία παραγωγής των tests.
- **Απλότητα** – Απλή δομή και επαληθεύσεις δίχως κάποιο περίπλοκο API, ενώ υποστηρίζει πολλά εξελιγμένα χαρακτηριστικά.
- **Ταχύτητα** - Τρέχει τα tests παράλληλα σαν ξεχωριστές διεργασίες του Node.js.
- **Έλεγχος με στιγμιότυπα** - υποστηρίζεται σαν κομμάτι της βιβλιοθήκης.

```
describe('Login.vue', function () {
    it('Does not sign in with wrong credentials', async() =>{
        await expect(BackEndApi.postCalls('/auth/login', wrongInput)).to.be.rejecte
      })
    it('Signs In Succesfully with correct credentials', async() =>{

      res =await BackEndApi.postCalls('/auth/login', loginInput)
      expect(res.data.user).to.have.property('email')
    })
    it('Server returns correct e-mail', async () =>{
        expect(res.data.user.email).to.equal('test351@email.com')
    })
    it('Server returns correct firstname', async () =>{
        expect(res.data.user.firstName).to.equal('George');
    })
  })
```

Εικόνα 2.1: Στιγμιότυπο κώδικα ελέγχου (unit tests)

```
  lastName: 'testLast',
  password: '12345678',
  address: 'testAddr',
  email: 'test351@email.com',
  phone: '12345678',
  job: 'testJob',
  dob: 2020-02-25T18:01:21.314Z,
  status: 'NOT OK' }
    √ Does not register duplicate user  (192ms)

  Login.vue
{ email: 'te@email.com', password: '12345678' }
    √ Does not sign in with wrong credentials (94ms)
{ email: 'test351@email.com', password: '12345678' }
    √ Signs In Succesfully with correct credentials (110ms)
    √ Server returns correct e-mail
    √ Server returns correct firstname


  8 passing (2s)

 MOCHA  Tests completed successfully
```

Εικόνα 2.2: Στιγμιότυπο εκτέλεσης από unit & integration tests

Μετά από δοκιμασία των παραπάνω δοκιμών μονάδας και ενσωμάτωσης, επιλέχτηκε η Mocha ως η βιβλιοθήκη για την εφαρμογή τους πάνω στην υπηρεσία Agronit. Αυτό έγινε για τους εξής λόγους:

- **Ευελιξία** – Η βιβλιοθήκη Mocha κατέστησε εφικτή την εγκατάσταση και εκμετάλλευση βοηθητικών βιβλιοθηκών και συγκεκριμένα των Sinon & Chai, οι οποίες αποδείχτηκαν πολύτιμες, καθώς παρέχουν πληθώρα λειτουργιών και διαθέτουν εκτενή υποστήριξη στο διαδίκτυο.
- **Συγγραφή & οργάνωση κώδικα ελέγχου** – Ο έλεγχος για κάθε μια λειτουργία της εφαρμογής ορίζεται μέσω της λειτουργίας describe() και χωρίζεται σε πολλά μέρη μέσω της λειτουργίας it(). Ο τελικός κώδικας ελέγχου είναι αρκετά ευανάγνωστος.
- **Συμβατότητα με Iot υπηρεσίες** – Σε πολλές περιπτώσεις και συγκεκριμένα στους ελέγχους ενσωμάτωσης, ο έλεγχος πραγματοποιείται σε συνάρτηση όπου υπάρχει επικοινωνία με δίκτυο και αναμονή για δεδομένα. Βιβλιοθήκες όπως η Jest και η Jasmine τερματίζουν πρόωρα την εκτέλεση των ελέγχων, πριν προλάβουν τα δεδομένα να μεταφερθούν, με αποτέλεσμα να εξάγουν λάθος συμπεράσματα. Το Mocha διαθέτει βιβλιοθήκη που επιτρέπει την αναμονή / προσωρινή παύση της εκτέλεσης των ελέγχων, μέχρι να μεταφερθούν τα δεδομένα διαμέσου του δικτύου.

Selenium

Το Selenium και εργαλεία που βασίζονται σε αυτό κυριάρχησαν στην αγορά για χρόνια. Δεν έχει γραφτεί μόνο για τον έλεγχο εφαρμογών αλλά μπορεί επίσης να ελέγξει έναν φυλλομετρητή, αξιοποιώντας έναν οδηγό (WebDriver) ο οποίος ελέγχει φυλλομετρητές χρησιμοποιώντας πρόσθετα και επεκτάσεις.

Node.js <=> WebDriver <=>
FF/Chrome/IE/Safari drivers <=> browser

Ο προγραμματιστής μπορεί να αποκτήσει πρόσβαση στον WebDriver του Selenium με πολλούς διαφορετικούς τρόπους και χρησιμοποιώντας ποικίλες γλώσσες προγραμματισμού, ακόμη και με εργαλεία που δεν χρησιμοποιούν πραγματικό προγραμματισμό.
Ο WebDriver εισάγεται σαν βιβλιοθήκη σε ένα προγραμματιστικό πλαίσιο ελέγχου και τα test γράφονται σαν μέρος του. Αναλαμβάνει την επικοινωνία του κώδικα γραμμένου σε Node.js με όλους τους ευρέως διαδεδομένους φυλλομετρητές. Ο WebDriver από μόνος του μπορεί να είναι επαρκής, αλλά δημιουργήθηκαν πολλές διαφορετικές βιβλιοθήκες για να τον επεκτείνουν ακόμη περισσότερο και να κάνουν την χρήση του πιο εύκολη.

Protractor

Το **Protractor** είναι μια βιβλιοθήκη που επεκτείνει το Selenium και παρέχει βελτιωμένη σύνταξη και ειδικά ενσωματωμένα εργαλεία.

- **Angular –** Παρέχει ειδικές λειτουργίες και το επίσημο documentation της Angular προτείνει την χρήση αυτού του εργαλείου. Μπορεί να χρησιμοποιηθεί και σε άλλα προγραμματιστικά πλαίσια της Javascript.
- **Αναφορά σφαλμάτων –** Έξυπνος μηχανισμός και φιλικά προς τον χρήστη μηνύματα.
- **Υποστήριξη –** Είναι διαθέσιμη η υποστήριξη για την Typescript και η βιβλιοθήκη χειρίζεται και διατηρείται από την ομάδα της Angular.

WebdriverIO

Το WebdriverIO έχει την δικιά του υλοποίηση του Webdriver του Selenium**.**

- **Σύνταξη** – Απλό στην χρήση και εύκολο στην ανάγνωση.

• **Ευέλικτο** – Απλή, ευέλικτη και επεκτάσιμη βιβλιοθήκη.
• **Κοινότητα** – Έχει καλή υποστήριξη και ενθουσιώδη κοινότητα από προγραμματιστές.

Nightwatch

Το Nightwatch έχει την δικιά του υλοποίηση του Selenium WebDriver. Παρέχει το δικό του προγραμματιστικό πλαίσιο ελέγχου, χρησιμοποιώντας έναν διακομιστή για ελέγχους, λειτουργίες επαλήθευσης δεδομένων και ποικίλα άλλα εργαλεία.

• **Προγραμματιστικό πλαίσιο** – Μπορεί να χρησιμοποιηθεί και σε άλλα προγραμματιστικά πλαίσια, αλλά είναι ιδιαίτερα χρήσιμο να τρέξουν τα tests από άκρη-σε-άκρη όχι ως μέρος ενός άλλου πλαισίου.
• **Σύνταξη** – φαίνεται ως η πιο εύκολη στην χρήση και στην ανάγνωση.
• **Αναφορά Σφαλμάτων** – Πολύ φιλικό περιβάλλον για τον χρήστη.
• **Υποστήριξη** – Δεν έχει υποστήριξη για Typescript, και η βιβλιοθήκη αυτή φαίνεται να έχει λιγότερη υποστήριξη από άλλες, αλλά η δημοτικότητά της αυξάνεται σταθερά.

Appium

Το **Appium** παρέχει ένα API παρόμοιο με αυτό του Selenium για τον end-to-end έλεγχο ιστοσελίδων και εφαρμογών σε μία συσκευή κινητού χρησιμοποιώντας τα παρακάτω εργαλεία:

•**iOS 9.3+**: XCUITest της Apple.
•**Before iOS 9.3**: Το UIAutomation της Apple.
•**Android 4.2+**: Τα UiAutomator/UiAutomator2 της Google.
•**Android 2.3+**: Το Instrumentation της Google. (Η υποστήριξη για το Instrumentation παρέχεται από ένα ξεχωριστό εργαλείο, το Selendroid)
•**Windows Phone**: Ο οδηγός WinAppDriver της Microsoft.

TestCafe

Το **TestCafe** είναι μια εναλλακτική λύση για εργαλεία βασισμένα στο Selenium. Έγινε εργαλείο ανοικτού κώδικα το 2016. Το TestCafe είχε επίσης μια έκδοση με πληρωμή που προσέφερε εργαλεία που δεν χρησιμοποιούσαν προγραμματισμό. Θα αντικατασταθεί σύντομα από το TestCafe Studio το οποίο θα είναι δωρεάν.

Εγκαθίσταται και τρέχει ως Javascript script σε μια ιστοσελίδα, αντί να ελέγξει τον ίδιο τον φυλλομετρητή, όπως κάνει το Selenium. Αυτό το επιτρέπει να τρέχει σε οποιοδήποτε φυλλομετρητή, συμπεριλαμβανομένων και των συσκευών κινητής τηλεφωνίας και να έχει πλήρη έλεγχο πάνω στην εκτέλεση του βρόχου εκτέλεσης της Javascript.

- **Γρήγορο στην εγκατάσταση** – Εγκαθίσταται μέσω του npm.
- **Φυλλομετρητές και συσκευές** – Υποστηρίζει πληθώρα φυλλομετρητών και συσκευών για τα tests. Αυτό περιλαμβάνει την εκτέλεση των τεστ σε Headless Chrome και Headless Firefox τα οποία θα επεξηγηθούν πιο κάτω.
- **Παράλληλος έλεγχος**- Το TestCafe μπορεί να τρέξει ταυτόχρονα σε πολλά στιγμιότυπα του φυλλομετρητή. Αυτό μειώνει δραματικά τους χρόνους εκτέλεσης των δοκιμών.
- **Αναφορά Σφαλμάτων –** Φιλικό προς τον χρήστη περιβάλλον.
- **Ξεχωριστό περιβάλλον –** Έχει δικιά του δομή για τα tests και αξιοποιεί μια διεπαφή χρήστη στην οποία συντάσσονται τα tests. Αυτό μπορεί να αποδειχθεί ιδιαίτερα βολικό, αλλά ταυτόχρονα ξενίζει τον προγραμματιστή, καθώς τα tests συνήθως τρέχουν το ένα ξεχωριστά από το άλλο.

Cypress

Το Cypress είναι ένας άμεσος ανταγωνιστής του TestCafe. Επιχειρεί σαν το TestCafe να εγκαθίσταται και να τρέχει στον φυλλομετρητή σαν script, αλλά με πιο βολικό και σύγχρονο τρόπο.
Η διαφορά μεταξύ τους είναι ότι το Cypress εκτελείται στον φυλλομετρητή και ελέγχει τα tests από εκεί ενώ το TestCafe τρέχει σε Node.js and και ελέγχει τα tests μέσα από μια σειριακή επικοινωνία με το εγκατεστημένο του script εντός του φυλλομετρητή.

Πρόκειται για καινούργια βιβλιοθήκη (μετατράπηκε από κλειστή σε δημόσια beta έκδοση τον Οκτώβριο του 2017) και έχει ήδη αποκτήσει πολλούς υποστηρικτές.

- **Παράλληλοι έλεγχοι**
- **Εγχειρίδιο χρήσης** – Απλό και ξεκάθαρο.
- **Εύκολη πρόσβαση στις μεταβλητές εφαρμογής** – Χωρίς χρήση σειριοποησιμότητας (Ένα παράδειγμα τέτοιας χρήσης είναι η μετατροπή των αντικειμένων σε JSON μορφή, η αποστολή τους στο Node.js σαν κείμενο και η επανα-μετατροπή τους σε αντικείμενα).
- **Πολύ βολικά εργαλεία εκτέλεσης & αποσφαλμάτωσης**
- **Δεν υποστηρίζει όλους τους φυλλομετρητές** – Υποστηρίζει μόνο Chrome προς το παρόν.

- **Λείπουν κάποια σενάρια χρήσης –** Ένα παράδειγμα είναι η έλλειψη προσομοίωσης χρήσης της συρόμενης μπάρας σε HTML5.
- **Χρησιμοποιεί Mocha –** Αυτό έχει ως αποτέλεσμα τα tests από άκρη-σε-άκρη να έχουν την ίδια δομή με τα unit και integration tests, εφόσον έχει επιλεχθεί το mocha γι αυτά.

Puppeteer

Το **Puppeteer** είναι μια βιβλιοθήκη για Node.js, που αναπτύχθηκε από την Google. Παρέχει ένα βολικό Node.js API για τον έλεγχο του Chrome ή του Headless Chrome. Το **Headless Chrome** είναι μια έκδοση Chrome αριθμού 59+ η οποία εκτελείται με την σημαία “--headless”. Όταν το Chrome εκτελείται σε headless μορφή, επιτρέπει τον έλκεγχό του από ένα API , και το Puppeteer είναι ένα εργαλείο σε JavaScript το οποίο παρέχει η Google provides to γι αυτό το σκοπό.

Ένας άλλος φυλλομετρητής που υποστηρίζει την headless μορφή είναι ο firefox. Πολλά διαφορετικά εργαλεία για ελέγχους μπορούν να χρησιμοποιήσουν τους Headless Chrome και Firefox, όπως τα TestCafe, Karma, Cypress.

- **Κοινότητα** - Το Puppeteer είναι σχετικά καινούργιο, αλλά έχει μεγάλη κοινότητα η οποία αναλαμβάνει την δημιουργία εργαλείων.
- **Ταχύτητα** - Χρησιμοποιεί την τελευταία έκδοση της μηχανής του Chrome, και ως εκ τούτου είναι πολύ γρήγορο.

΄Ενα μεγάλο μειονέκτημα του Headless Chrome (και επομένως του Puppeteer) είναι ότι δεν υποστηρίζει επεκτάσεις όπως το Flash.

Εφαρμογή και Παρατηρήσεις

Μετά από δοκιμασία των παραπάνω μεθόδων ελέγχου από άκρη-σε-άκρη, επιλέχτηκε η Nightwatch ως η βιβλιοθήκη για την εφαρμογή τους πάνω στην υπηρεσία Agronit. Αυτό έγινε για τους εξής λόγους:

- **Περιβάλλον αποσφαλμάτωσης** – Μετά από εκτέλεση των ελέγχων, τα μηνύματα λάθους ή επιτυχίας της δοκιμής ήταν τα πιο ευανάγνωστα και φιλικά

προς τον χρήστη, με ταυτόχρονη απουσία περιττών μηνυμάτων της εφαρμογής.

- **Συγγραφή κώδικα** – Οι εντολές για τον έλεγχο του φυλλομετρητή είναι απλές και ευανάγνωστες και η βιβλιοθήκη παρέχει επιλογείς για την εύκολη πρόσβαση στοιχείων HTML, το οποίο είναι πολύ σημαντικό για την δοκιμή του front-end κώδικα της Agronit.

Ο προγραμματιστής ενθαρρύνεται να αξιοποιήσει οποιαδήποτε από τις βιβλιοθήκες, αλλά ειδικά αυτές που βασίζονται στον Selenium WebDriver, καθώς η ευελιξία και η απλοικότητά τους δεν θα ξενίσει κάποιον που έρχεται σε πρώτη επαφή με τις δοκιμές εφαρμογών.

Εφαρμογή Μεθόδων

```
32          // d) Go to user profile & update name
33          .click('button.v-btn.v-btn--icon.v-btn--large.theme--dark')
34          .pause(2000)
35          .waitForElementVisible('a[href="#/profile"]')
36          .click('a[href="#/profile"]')
37          .click('input#first')
38          .setValue('input#first', ['', [browser.Keys.CONTROL, "a"]])
39          .keys('\uE017') // delete them using delete key
40          .setValue('input#first', 'George')
```

Εικόνα 3.1. Στιγμιότυπο κώδικα ελέγχου end-to-end δοκιμής

Κατασκευάζονται δύο υποθετικά σενάρια χρήσης. Στο πρώτο, η βιβλιοθήκη Nightwatch αναλαμβάνει τον έλεγχο του φυλλομετρητή και πραγματοποιεί τις εξείς ενέργειες:

1. Επίσκεψη στην ιστοσελίδα της Agronit και σύνδεση σε λογαριασμό χρήστη.
2. Επίσκεψη στο προφίλ του.
3. Αλλαγή του ονόματος του χρήστη.
4. Έξοδος από τον λογαριασμό.

Το δεύτερο σενάριο χρήσης έχει ως εξής:

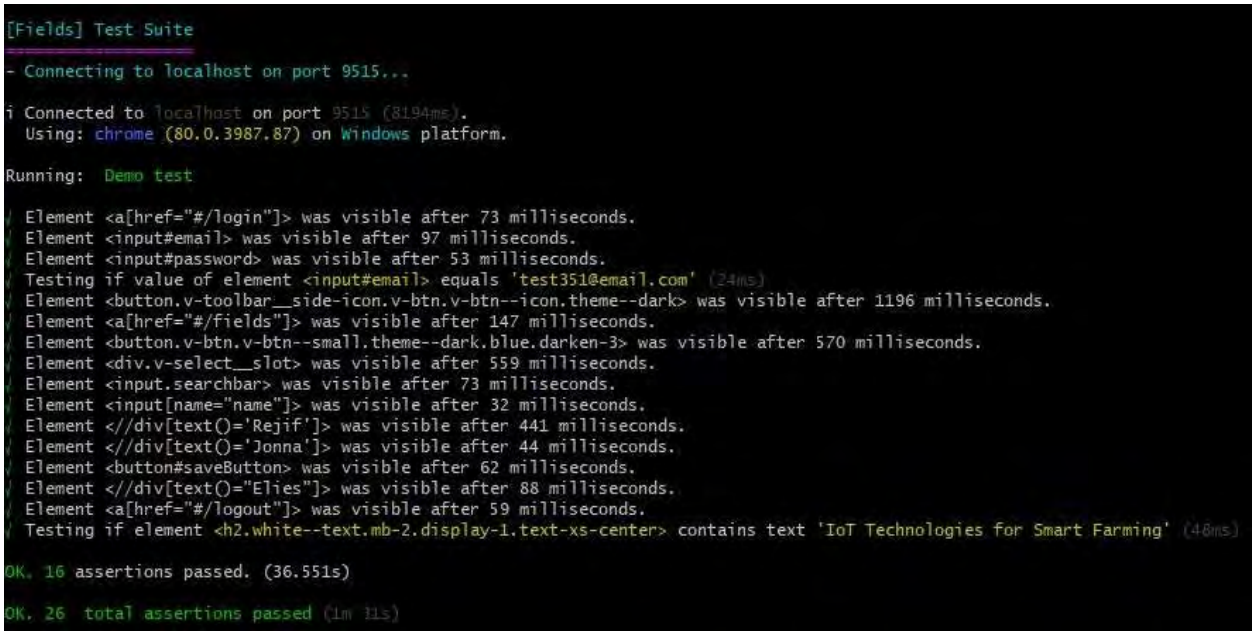
1. Επίσκεψη στην ιστοσελίδα της Agronit και σύνδεση σε λογαριασμό χρήστη.
2. Δημιουργία νέου αγροκτήματος με την συμπλήρωση όλων των απαραίτητων στοιχείων του.
3. Έλεγχος αν δημιουργήθηκε σωστά το καινούργιο αγρόκτημα.
4. Αποσύνδεση από την εφαρμογή.

Το κάθε σενάριο είναι αποθηκευμένο σαν ξεχωριστός κώδικας στο δέντρο ανάπτυξης κώδικα της υπηρεσίας Agronit, στα αρχεία editProfile.js και fields.js αντίστοιχα. Ο χρήστης δίνει την εντολή “npm run nightwatch” και η βιβλιοθήκη εντοπίζει την τοποθεσία των δύο κωδίκων ελέγχου / σεναρίων χρήσης και τα εκτελεί σειριακά. Κατά την εκτέλεση δημιουργείται μια διεργασία Chrome, η οποία είναι ελεγχόμενη από την Nightwatch, και εκτελεί όλες τις εντολές που όρισε ο χρήστης στα σενάρια.

Στην εικόνα 3.1 φαίνεται μέρος του κώδικα ελέγχου του πρώτου σεναρίου.

Στους παρόντες ελέγχους είναι σημαντικό να εντοπιστεί γρήγορα και αποτελεσματικά από τον κώδικα ένα στοιχείο (όπως ένα κουμπί ή μια φόρμα εισόδου), με σκοπό να γίνουν ενέργειες πάνω σε αυτό, οι οποίες θα εξομοιώνουν την χρήση από έναν πραγματικό χρήστη. Ο εντοπισμός αυτός αποδυκνείεται ιδιαίτερα δύσκολος, ειδικά όταν το στοιχείο HTML είναι εμφωλευμένο μέσα σε άλλα.

Ο εντοπισμός ενός στοιχείου γίνεται με με βάση το αναγνωριστικό τους, ή κάποιο γνώρισμα που τα διακρίνει από τα υπόλοιπα. Πολλές φορές όμως ο δημιουργός της υπο-έλεγχο εφαρμογής δεν έχει μεριμνήσει για τον ονοματισμό τους, με αποτέλεσμα ο εντοπισμός αυτός να γίνεται χρονοβόρος και μη αποδοτικός. Η διαφορά αυτή φαίνεται στην Εικόνα 3.1, στις γραμμές 33 και 37. Στην πρώτη περίπτωση το στοιχείο εντοπίζεται με βάση την CSS τεχνοτροπία του (λόγω απουσίας αναγνωριστικού), ενώ στην δεύτερη με βάση το αναγνωριστικό.



Εικόνα 3.2: Στιγμιότυπο εκτέλεσης του δεύτερου σεναρίου end-to-end δοκιμής

Επίλογος

Ο προγραμματιστής που αναπτύσσει εφαρμογές και επιθυμεί να εισαχθεί στον κόσμο των δοκιμών, ενθαρρύνεται σε πρώτο στάδιο να συγγράφει κώδικα παραγωγής συμβατό με μεθόδους ελέγχου. Απώτερος σκοπός αποτελεί η εξοικοίωση του προγραμματιστή με την χρήση των βιβλιοθηκών και η κατανόηση των ωφελών της δοκιμής εφαρμογών, έτσι ώστε να υιοθετήσει εν καιρώ την “test-first” προγραμματιστική τεχνική, δηλαδή την ανάπτυξη οδηγούμενη από ελέγχους.

Βιβλιογραφία

[1] (Paper) Unit-Test-using-Test-Driven-Development-Approach-to-Support-Reusability

[2] (Paper) Evolution-of-Testing-with-Respect-to-the-Programming-Paradigms

[3] https://medium.com/@giltayar/testing-your-frontend-code-part-i-introduction-7e307eac4446

[4] http://agiledata.org/essays/tdd.html

[5] http://softwaretestingfundamentals.com/integration-testing/

[6] https://www.tutorialspoint.com/software_testing_dictionary

[7] https://vuejs.org/v2/guide/unit-testing.html

[8] https://www.softwaretestinghelp.com/

[9] https://medium.com/welldone-software/an-overview-of-javascript-testing-in-2019-264e19514d0a

[10] https://selenium.dev/documentation/en/

[11] https://mochajs.org/index.html