Αριστοτέλειο Πανεπιστήμιο Θεσσαλονίκης
Πρόγραμμα Μεταπτυχιακών Σπουδών στην Πληροφορική κατεύθυνση: Πληροφοριακά Συστήματα

Εργασία εξαμήνου στο μάθημα “Επαλήθευση Λογισμικού”

Θέμα: “*Frontend testing in Web Applications (Unit + End to End testing)*”

Ο μεταπτυχιακός φοιτητής:
Κουρουτζίδης Θεόδωρος – Νικόλαος
ΑΜ: 568

Φεβρουάριος 2015

Περιεχόμενα

Εισαγωγή

Τα τελευταία χρόνια οι διαδικτυακές εφαρμογές (web applications) έχουν λάβει μια μεγάλη μερίδα της αγοράς και παράλληλα οδήγησαν στη δημιουργία νέων εργαλείων. Αυτό οφείλεται κατά κύριο λόγο στην σημαντική βελτίωση των υποδομών δικτύωσης και πρόσβασης στον παγκόσμιο ιστό. Η ραγδαία αυτή αύξηση των εφαρμογών ήταν αναμενόμενο να “γεννήσει” και την ανάγκη επαλήθευσης των προϊόντων – λογισμικών. Όπως και στις κοινές standalone εφαρμογές, έτσι και στις web based υπάρχει η ανάγκη για δοκιμές – tests σε διάφορα επίπεδα. Στο παρόν έργο θα δοθεί έμφαση στο κομμάτι των “δοκιμών μονάδων” ή αλλιώς unit Testing καθώς και σε End to End Integration Testing (e2e). Οι δοκιμές αυτές θα πραγματοποιηθούν με γνώμονα το client side κομμάτι των εφαρμογών αυτών ή αλλιώς frontend.

Σκοπός

Η εργασία αυτή αποσκοπεί στη χρήση εργαλείων δημιουργίας δοκιμών και την ενσωμάτωσή τους σε ένα υπό ανάπτυξη λογισμικό. Συνοπτικά, το λογισμικό αυτό αποτελεί μια διαδικτυακή πλατφόρμα κρατήσεων – ενοικιάσεων καλοκαιρινών καταλυμάτων με τίτλο E-Res (E-Resort). Έτσι με τη χρήση των εργαλείων αυτών θα συνταχθούν μερικά ενδεικτικά unit tests και e2e integration tests. Κύριος στόχος είναι ο αναγνώστης να κατανοήσει πλήρως τους λόγους για τους οποίους είναι αναγκαία η δημιουργία και διεξαγωγή των παραπάνω δοκιμών σε μια web based εφαρμογή. Παράλληλα επιδιώκεται μια στοιχειώδης εξοικείωση με τα εργαλεία αυτά με στόχο τη μετέπειτα χρήση τους σε μεγαλύτερα έργα. Τέλος γίνεται μια εκτενής αναφορά στα κριτήρια επιλογής των σεναρίων δοκιμών κάτι που μπορεί να επιταχύνει σημαντικά τη διαδικασία της επαλήθευσης του λογισμικού, αποτρέποντας την εκτέλεση “ανούσιων” δοκιμών.

Θεωρητικό υπόβαθρο

Όπως έχει ήδη αναφερθεί, οι δοκιμές που θα αναπτυχθούν αφορούν στην επαλήθευση μιας web based εφαρμογής που βασίζεται στην αρχιτεκτονική πελάτη / εξυπηρετητή (client – server). Αυτό σημαίνει πως ο πηγαίος κώδικας χωρίζεται σε δύο κομμάτια, αυτό που εκτελείται στον εξυπηρετητή (server) και αυτό που εκτελείται στον πελάτη (client) και συγκεκριμένα στον φυλλομετρητή (browser) του χρήστη. Η εφαρμογή E-res σε ό,τι αφορά το server side (backend) κομμάτι υλοποιήθηκε σε Java, ενώ το client side (frontend) σε Angularjs (αποτελείται κυρίως από Javascript αλλά και jquery). Στο παρόν έργο θα μας απασχολήσει το frontend κομμάτι καθώς οι δοκιμές σε επίπεδο backend πραγματοποιούνται με εργαλεία που έχουν περιγραφεί καιαναλυθεί εκτενώς στο παρελθόν (πχ Junit).
Έτσι, λέγοντας Frontend Testing εννοούμε το σύνολο των δοκιμών που απευθύνονται στον κώδικα που εκτελείται στον client. Συγκεκριμένα θα αναλυθούν δυο υποπεριπτώσεις δοκιμών του frontend:

- Unit Testing: Όπως και στο backend έτσι και στο frontend, είναι ωφέλιμη η δοκιμή μονάδων

με τη διαφορά ότι στο AngularJs οι “μονάδες” ορίζονται διαφορετικά. Ως ένα framework που βοηθάει στη δημιουργία δυναμικών διαδικτυακών εφαρμογών ακολουθεί “πιστά” ένα τυπικό MVC Design Pattern (Model – View – Controller). Κάθε Controller έχει πρόσβαση στο δικό του Scope, το οποίο και περιλαμβάνει ένα σύνολο μεταβλητών και συναρτήσεων. Επιπλέον, όπως θα περιγραφεί και στις επόμενες ενότητες, κάθε Controller μπορεί να κάνει Inject και μερικά services που επιτελούν συγκεκριμένες λειτουργίες οι οποίες χρησιμοποιούνται και από άλλους Controllers παράλληλα. Όλα τα παραπάνω συνθέτουν ένα Scope και ταυτόχρονα ένα σύνολο από “Units” ανά Scope τα οποία καλούμαστε να “επαληθεύσουμε”. Ουσιαστικά τα unit tests αποτελούν την πρώτη γραμμή ελέγχου και διαπίστωσης σφαλμάτων.

- E2E Integration Testing: Καθώς η υπό ανάπτυξη εφαρμογή μεγαλώνει είναι λογικό να αυξάνεται και η πολυπλοκότητα αυτής. Κάτι τέτοιο καθιστά ιδιαιτέρως δύσκολη την διαπίστωση για το αν τα νέα “Features” (νέες λειτουργίες) λειτουργούν με τον αναμενόμενο τρόπο. Βοηθούν σημαντικά στην διαπίστωση bugs που προκύπτουν με την αλληλεπίδραση διαφόρων Units. Για παράδειγμα ακόμα και αν δυο units έχουν περάσει με επιτυχία τα αντίστοιχα unit tests, είναι πολύ πιθανόν η μεταξύ τους αλληλεπίδραση να οδηγεί σε μη αναμενόμενες συμπεριφορές. Τέτοιας φύσεως προβλήματα θα εξορυχθούν με τη βοήθεια του E2E Integration Testing.

Frontend Unit Testing

Στο στάδιο αυτό πρέπει ο προγραμματιστής να διασπάσει το σύνολο της εφαρμογής σε μικρότερα κομμάτια – Units. Για κάθε ένα ξεχωριστά πρέπει να επιβεβαιώσει πως λειτουργεί με τον αναμενόμενο τρόπο καθιστώντας την εφαρμογή αξιόπιστη. Είναι αναγκαίο λοιπόν να συνταχθούν δοκιμές μονάδων για τα βασικότερα στοιχεία – δυνατότητες της εφαρμογής. Στην Javascript η μικρότερη δυνατή μονάδα είναι μια συνάρτηση (function)

Διαθέσιμα εργαλεία

- Jasmine - *"Jasmine is a behavior-driven development framework for testing JavaScript code."*
- Mocha - *"The fun, simple, flexible JavaScript test framework."*
- QUnit - *"A JavaScript Unit Testing framework, used by jQuery."*

Στην περίπτωσή μας επιλέχθηκε το Jasmine και αυτό γιατί παρέχει ίσως το καλύτερο documentation τόσο σε θεωρητικό όσο και σε πρακτικό επίπεδο. Ακόμα συνεργάζεται άψογα με το βασικό frontend Framework που χρησιμοποιήθηκε για την κατασκευή του λογισμικού, το AngularJs.

Ενδεικτικά σενάρια δοκιμών μονάδων σχετικά με την εφαρμογή Eres

- Δοκιμή συνάρτησης που εμφανίζει ένα μήνυμα σφάλματος
- Δοκιμή συνάρτησης που πραγματοποιεί ταυτοποίηση χρηστών
- Δοκιμή συνάρτησης που εκτελεί διαφόρων ειδών ταξινομήσεις
- Δοκιμή συνάρτησης που καθιστά ένα διαμέρισμα ως διαθέσιμο ή μη

Ακόμα και να υλοποιήσουμε αρκετά unit tests δεν μπορούμε να είμαστε ποτέ σίγουροι για την εγκυρότητα του τελικού αποτελέσματος. Αυτό που μπορούμε όμως να σιγουρέψουμε είναι πως κάθε μονάδα λειτουργεί σωστά, δοκιμάζοντάς την ανεξάρτητα από τις υπόλοιπες. Έτσι υπάρχουν αρκετές ελπίδες όταν λειτουργήσουν όλες οι μονάδες (Units) μαζί, να λειτουργήσουν σωστά και χωρίς απρόσμενες συμπεριφορές.

Τέλος καλό είναι να αναφερθεί πως παίζει αρκετά σημαντικό ρόλο, το χρονικό διάστημα που θα επιλέξει ο καθένας για να σχεδιάσει τα εκάστοτε σενάριο δοκιμών. Από τη μια δεν προτιμάται ο σχεδιασμός όλων τον σεναρίων στην αρχή του έργου (αφού δεν είναι εκ των προτέρων γνωστές όλες οι λειτουργίες) αλλά ούτε και ο σχεδιασμός των σεναρίων στο τέλος του έργου. Το ιδανικότερο είναι η συγγραφή των unit tests να πραγματοποιείται επαναληπτικά και σε μικρούς κύκλους κατά τη διάρκεια υλοποίησης του έργου. Για παράδειγμα αφού υλοποιηθεί ένα “user story”, είναι ίσως η καταλληλότερη στιγμή να συγγραφούν όλες οι δοκιμές που αφορούν τα Units που απαρτίζουν το user story αυτό. Για παράδειγμα ένα User story θα μπορούσε να είναι: “Αφού πραγματοποιείται η ταυτοποίηση του χρήστη εμφανίζεται η λίστα με τα διαθέσιμα ενοικιαζόμενα διαμερίσματα”.

Θα μπορούσαμε να ταυτίσουμε τη διαδικασία του unit testing με το παρακάτω παράδειγμα: Κατά τη φάση ελέγχου ενός αεροπλάνου ελέγχονται ξεχωριστά διάφορα τμήματά του. Συγκεκριμένα για να ελεγχθεί η μηχανή, ο κατασκευαστής της παρέχει καύσιμο και αέρα και αναμένει να λειτουργήσει. Αν δεν λειτουργήσει κάτι πάει στραβά. Για το σύστημα διεύθυνσης του αεροσκάφους ο κατασκευαστής δοκιμάζει να στρέψει προς μια κατεύθυνση το τιμόνι και αναμένει τα πτερύγια του αεροσκάφους να κινηθούν αναλόγως.

Frontend Integration Testing

Ο συγκεκριμένος τύπος δοκιμών αφορά στη δημιουργία end – to – end tests με κύριο στόχο τη δοκιμή μιας ροής εργασιών ή σεναρίων χρήσης όπου επαληθεύεται η κατάσταση της εφαρμογής και της γραφικής διεπαφής (π.χ σφάλματα ή exceptions ή ορθή λειτουργία κλπ). Επιπλέον σημαντικό ρόλο στην όλη διαδικασία παίζει η αυτοματοποίηση των δοκιμών ούτως ώστε να εκτελούνται με ευέλικτο τρόπο με εργαλεία όπως το Selinium ή το PhantomJS . Τα εργαλεία αυτά διευκολύνουν σημαντικά το έργο κάθε Tester αφού ενσωματώνονται και σε κατάλληλο Task runner όπως για παράδειγμα τον Grunt. Με χρήση μίας και μόνο έκφρασης ο Tester έχει τη δυνατότητα να εκτελέσει το σύνολο των δοκιμών που ανέπτυξε αποφεύγοντας την επαναλαμβανόμενη – χειροκίνητη εκτέλεση όλων των σεναρίων.

Διαθέσιμα εργαλεία

- Protractor "*an end-to-end test framework for AngularJS applications*"
- CasperJS - *"CasperJS is a navigation scripting & testing utility which uses headless browsers."*

Ανάμεσα στα δυο εργαλεία που προαναφέρθηκαν επιλέχθηκε το Protractor κυρίως για λόγους που αφορούν το πλήρες Documentation αλλά και την ευκολία χρήσης και συγγραφής των δοκιμών.

Χαρακτηριστικά παραδείγματα e2e tests θα μπορούσαν να είναι τα παρακάτω:

- Ο χρήστης επιχειρεί να εισέλθει σε μια υπο-σελίδα χωρίς να κάνει login
- Ο χρήστης κάνει login πατάει το πλήκτρο προβολής διαμερισμάτων και επιλέγει να δει ένα από αυτά
- Ο χρήστης κάνει login και επιλέγει να δει ένα από τα διαμερίσματα.

Θα μπορούσαμε να παρομοιάσουμε τη διαδικασία του e2e testing με αυτή ενός robot το οποίο είναι προγραμματισμένο να κάνει μια σειρά ενεργειών όπως π.χ το να οδηγήσει ένα αεροπλάνο. Του λέμε να εισέλθει στο αεροπλάνο, να κλείσει την πόρτα, να βάλει μπρος τις μηχανές, να απογειωθεί και εν τέλη το ρωτάμε αν απογειώθηκε με επιτυχία.
Σε ό,τι αφορά το καταλληλότερο χρονικό διάστημα στο οποίο ιδανικά συντάσσονται τα integration tests: όταν τα user stories έχουν δημιουργηθεί, είναι μια καλή στιγμή να συγγραφούν και τα e2e tests που θα ανταποκρίνονται σειρά ενεργειών του εκάστοτε user story. Το θετικό είναι πως αυτού του είδους τα test είναι ιδιαίτερα ανθεκτικά σε ενδεχόμενες αλλαγές στα units του λογισμικού.

Κρίσημα στοιχεία προς επαλήθευση

Θα μπορούσε κανείς να κατηγοριοποιήσει τα σημεία μιας διαδικτυακής εφαρμογής που πρέπει να δοκιμασθούν επαρκώς:

- Functionality Testing
- UI & Usability Testing
- Security Testing
- Load Testing

Functionality Testing

Σε ό,τι αφορά το σύνολο των λειτουργιών που εμπεριέχονται σε κάθε λογισμικό, ο χρήστης "απαιτεί" την εύρυθμη , ταχύτατη και απρόσκοπτη λειτουργία του. Αυτό οδηγεί στο συμπέρασμα πως πρέπει να ελεγχθούν εξαντλητικά εκείνα τα στοιχεία της εφαρμογής όπου βοηθούν τον χρήστη να εκπληρώσει – παράξει κάποιο τελικό αποτέλεσμα. Μερικά από τα συνηθέστερα λειτουργικά στοιχεία μιας εφαρμογής που χρήζουν άμεσης επαλήθευσης είναι:

- **Forms (Φόρμες):** Επαλήθευση των ενεργών στοιχείων – συνδέσμων της φόρμας τα οποία δρομολογούν οποιαδήποτε διαδικασία χρησιμοποιώντας τα δεδομένα που προέρχονται από τα πεδία της φόρμας.
- **File manipulation and calculations:** Η διαχείριση αρχείων (Upload – Download) είναι

ακόμη ένα στοιχείο που πρέπει να ελεγθεί προκειμένου να διαπιστωθεί αν αφενώς ολοκληρώνεται με επιτυχία και αφετέρου αν με την ολοκλήρωση του upload/download παράγεται η επιθυμητή έξοδος.

- **Search:** Στις περιπτώσεις που μια εφαρμογή ενσωματώνει πεδία αναζήτησης είναι ορθό να ελεγχθεί αν η αναζήτηση είναι λειτουργική και γρήγορη καθώς και αρκετά “έξυπνη” ώστε να παράξει σχετικά αποτελέσματα.
- **Media components:** Δοκιμή των πολυμεσικών στοιχείων (εικόνες, βίντεο, ήχους) Τα στοιχεία αυτά πρέπει να εμφανίζονται με τέτοιο τρόπο ώστε να μην επηρεάζουν αρνητικά τη λειτουργία του συνόλου της εφαρμογής.
- **Scripts and libraries:** Πολύ σημαντικός ο έλεγχος συμβατότητας των στοιχείων της εφαρμογής (scripts, libraries) με το σύνολο των Browsers που είναι διαθέσιμοι (είτε νέοι είτε παλαιότεροι). Στην περίπτωση οποιασδήποτε ασυμβατότητας με κάποια έκδοση ενός browser ο χρήστης πρέπει να είναι ενήμερος.

User interface and usability testing

Εκτός από το όλο functionality της εφαρμογής πρέπει να ελεγχθεί και το “φαίνεσθαι” αυτής. Πράγμα που μας οδηγεί να ελέγξουμε τα παρακάτω:

Navigation: Έλεγχος των σημαντικότερων συνδέσμων – πλήκτρων για να διαπιστωθεί αν οδηγούν εκεί που προβλέπεται.

Accessibility: Δοκιμή προσβασιμότητας της εφαρμογής από άτομα με ειδικές ικανότητες ή διαφορετικών γλωσσικών – πολιτισμικών ιδιαιτεροτήτων.

Cross browser testing: Έλεγχος των οπτικών στοιχείων – μορφοποιήσεων της εφαρμογής, καθώς αυτές μπορεί να μην εμφανίζονται με τον ίδιο τρόπο στο σύνολο των φυλλομετρητών (browsers) ή να μην εμφανίζονται καθόλου.

Error messages and warnings: Ακόμα και αν η εφαρμογή δεν προβλέπεται να παρουσιάσει προβλήματα μετά την διαδικασία της επαλήθευσης, πρέπει να προβλεφθούν απρόσμενες καταστάσεις. Σε αυτές τις καταστάσεις πρέπει να σιγουρευθεί πως θα εμφανισθούν τα αντίστοιχα μηνύματα σφαλμάτων – προειδοποιήσεων.

Layouts: Δοκιμή της εφαρμογής όχι μόνο σε πολλούς browsers αλλά και σε διαφορετικές αναλύσεις (tablet , κινητά κλπ).

Security testing

Οι περισσότερες εφαρμογές αποθηκεύουν πληροφορίες των χρηστών που πολλές φορές είναι προσωπικές. Έτσι πρέπει να παραμείνουν ασφαλείς σε όλη τη διάρκεια ζωής του λογισμικού, ακολουθώντας δυο κανόνες:

1. Τα απόρρητα δεδομένα να παραμείνουν απόρρητα.
2. Η εφαρμογή να δίνει πρόσβαση σε οποιαδήποτε απόρρητη πληροφορία μόνο μετά την

ταυτοποίηση του χρήστη, με την προϋπόθεση πως ο δεύτερος έχει τα ανάλογα δικαιώματα πρόσβασης.

Οι Crackers μπορούν οποιαδήποτε στιγμή να “επιτεθούν” στην εφαρμογή, έτσι καλό είναι αυτός που ελέγχει την εφαρμογή να γνωρίζει τις βασικές μεθόδους επίθεσης όπως:

Cross-site scripting: ο επιτιθέμενος παραποιεί τμήματα από client scripts που εκτελούνται στον client – θύμα με το που επισκέπτεται την ανάλογη τοποθεσία

SQL injection: ενθυλάκωση Sql εκφράσεων σε κάποιο από τα πεδία κειμένου της εφαρμογής με τελικό στόχο την παραποίηση,προσθήκη,διαγραφή,εισαγωγή οποιουδήποτε στοιχείου στη βάση δεδομένων της εφαρμογής

DDoS (Distributed Denial of Service) attacks: ταυτόχρονα ερωτήματα – αιτήματα προς την εφαρμογή από διαφορετικούς αποστολείς οδηγούν πολλές φορές στην άρνηση απόκρισης ή στην πλήρη αποτυχία αυτής.

Προκειμένου να πραγματοποιηθούν έλεγχοι για τα παραπάνω κακόβουλα στοιχεία μπορούν να δράσουν μερικοί “White hat hackers”, “ενάντια” στην εφαρμογή ή εναλλακτικά να χρησιμοποιηθούν έτοιμα εργαλεία Vulnerability testing όπως το Grabber.

Load testing

Όπως είναι φυσιολογικό, οι χρήστες αναμένουν η εφαρμογή να λειτουργεί επ'αόριστον όπως λειτουργούσε και κατά την εγκατάσταση. Κάτι τέτοιο εκ των πραγμάτων είναι αδύνατον καθώς καθημερινά ο όγκος των αποθηκευμένων δεδομένων καθώς και των αιτημάτων αυξάνεται εκθετικά. Είναι πολύ σημαντικό λοιπόν να ελέγξουμε κατά πόσο η εφαρμογή μας είναι σε θέση να ανταπεξέλθει σε μεγάλους φόρτους. Έτσι δεν έχουμε παρά να δοκιμάσουμε να περάσουμε πολλά “Dummy” δεδομένα στην εφαρμογή μας καθώς και να αναθέσουμε στην εφαρμογή τη διαχείριση πολλών ταυτόχρονων χρηστών.

Εργαλεία

Grunt

Το Grunt διευκολύνει τόσο την εγκατάσταση testing frameworks (Jasmine, Protractor κλπ) όσο και στην εκτέλεση των δοκιμών. Μπορεί να διαρρυθμιστεί κατάλληλα ώστε με το που συμβεί η οποιαδήποτε μεταβολή σε κάποιο από τα πηγαία αρχεία, αυτομάτως να επανεκτελεσθούν και τα σενάρια δοκιμών. Σε συνδυασμό με το εργαλείο Karma ένα command σε γραμμή εντολών θα μπορούσε να είναι "#grunt karma:unit" (εκτέλεση unit tests) ή "grunt protractor" (εκτέλεση e2e tests).

Karma

Χρησιμοποιήθηκε για την αυτοματοποιημένη εκτέλεση των unit tests. Αυτό γιατί εμπεριέχει το εργαλείο Jasmine που είναι και το βασικό εργαλείο συγγραφής των unit tests. Επιπλέον μας παρέχει τη δυνατότητα συγγραφής e2e Tests καθώς εμπεριέχει και το εργαλείο "Angular scenario runner". Δυστυχώς το εργαλείο αυτό είναι προσωρινά deprecated και επομένως χρησιμοποιήθηκε στη θέση του το εργαλείο Protractor. Άρα λοιπόν το Karma σαν ένα ευρύτερο framework θα μπορούσε να περιγραφεί ως ένας "Test Runner" που λύνει πραγματικά τα χέρια του tester.

Protractor

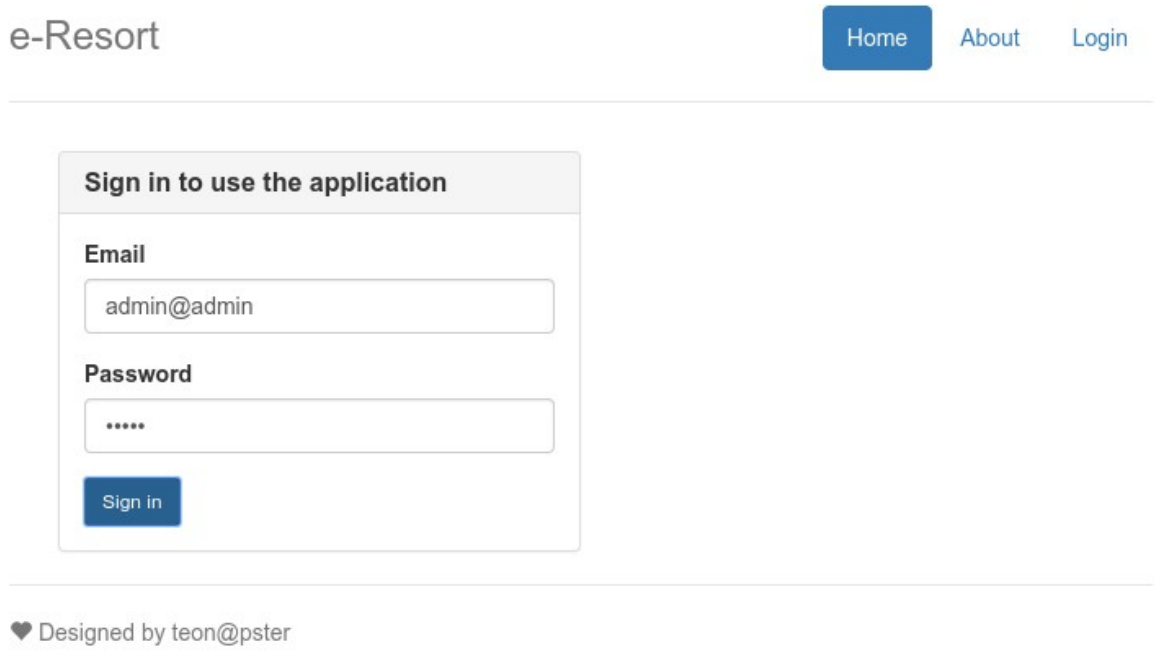
Ένα από τα διασημότερα frameworks που επικεντρώνονται στο e2e testing. Χρησιμοποιώντας ιδιαίτερα απλή σύνταξη επιτρέπει την δημιουργία σύνθετων δοκιμών ικανών να διανύσουν μεγάλα μονοπάτια – σενάρια χρήσης της εφαρμογής. Επίσης ως ολοκληρωμένο σύστημα e2e testing παρέχει ένα σύνολο συναρτήσεων ικανών να προσπελάσουν με εύκολο τρόπο στοιχεία DOM – html.

Yeoman

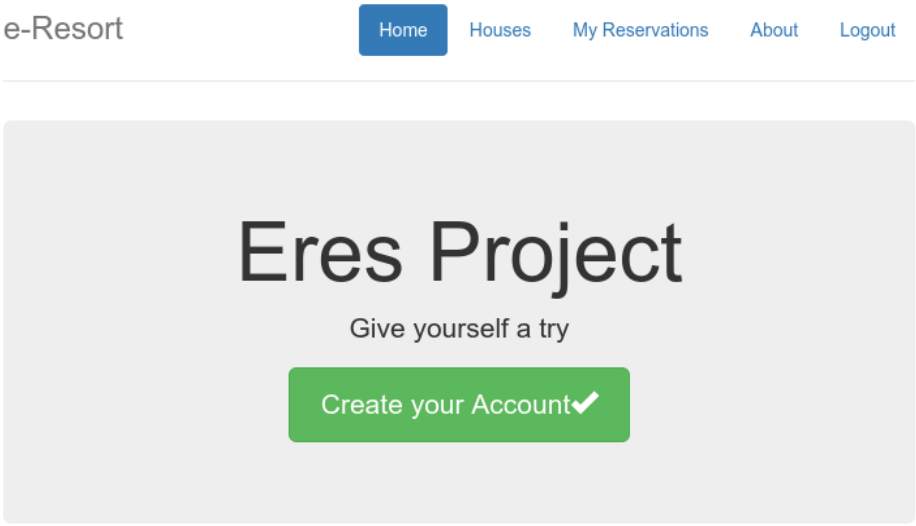
Το εργαλείο αυτό αν και είχε δευτερεύοντα ρόλο στη δημιουργία αλλά και την επαλήθευση του E-res αποτελεί ένα βοήθημα που επιτάχυνε τόσο τη φάση της υλοποίησης όσο και τη φάση των δοκιμών. Αυτό γιατί με σύντομες εκφράσεις στη γραμμή εντολών, μπορεί να "αρχικοποιήσει"views,models,controllers,tests κ.α. . Ο αγγλικός όρος "App skeleton Creator" είναι άκρως περιγραφικός.

Περιγραφή του προς επαλήθευση λογισμικού

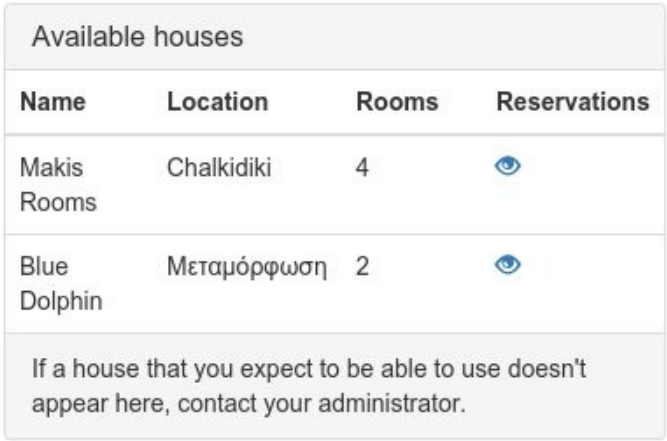
Όπως προαναφέρθηκε, στα πλαίσια της εργασίας αυτής αναπτύχθηκε μια εφαρμογή προκειμένου να εκτελεστούν μερικά σενάρια δοκιμών ικανά να μας επιτρέψουν να εξάγουμε χρήσιμα και ασφαλή συμπεράσματα. Ωστόσο σαν εφαρμογή δεν είναι ολοκληρωμένη σε όλους τους τομείς κάτι το οποίο δεν θα μας απασχολήσει καθώς δεν θα δοκιμασθούν παρά μόνον μερικά σενάρια χρήσης και μερικές μονάδες – units. Τα στοιχεία που θα παρουσιασθούν παρακάτω είναι αυτά που θα δοκιμασθούν τόσο σε επίπεδο μονάδας όσο και σε επίπεδο σεναρίου χρήσης (e2e test)
Η “οθόνη” υποδοχής αποτελείται από μια φόρμα εισαγωγής διαπιστευτηρίων (Authentication form). Η συμπλήρωση των διαπιστευτηρίων είναι υποχρεωτική και δεν μπορεί να παρακαμφθεί.



Αφού ο χρήστης πληκτρολογήσει τα στοιχεία του εισέρχεται στο σύστημα έχοντας πλέον ως κύρια οθόνη μια φόρμα με γενικές πληροφορίες.

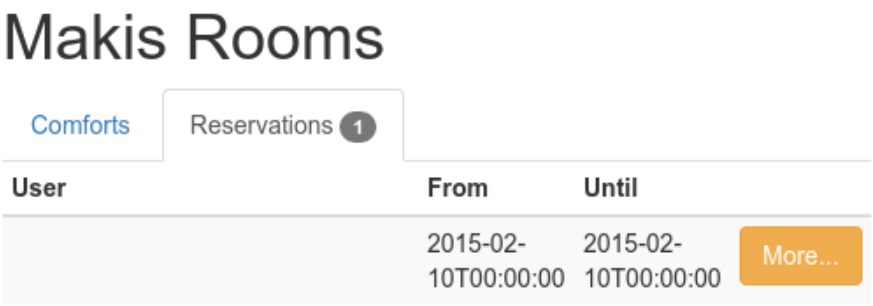


Όντας συνδεδεμένος κανείς μπορεί να μεταβεί στην καρτέλα “Houses” όπου μπορεί να παρατηρήσει τα διαθέσιμα προς ενοικίαση διαμερίσματα.



Για κάθε ένα διαμέρισμα αναφέρεται η τοποθεσία του καθώς και το σύνολο των δωματίων που διαθέτει το καθένα. Αν επιθυμεί να δει περισσότερες πληροφορίες κάποιο συγκεκριμένο διαμέρισμα (όπως πχ αν έχει μπαλκόνι, αν είναι διαθέσιμο κλπ) υπάρχει ο ανάλογος σύνδεσμος. Πατώντας εκεί ο χρήστης αντικρίζει την παρακάτω οθόνη:

όπου διακρίνει δυο καρτέλες (Tab Panels) όπου στην πρώτη (Comforts) βλέπει το σύνολο των διαθέσιμων πληροφοριών για το τρέχον διαμέρισμα και στην δεύτερη (Reservations) μπορεί να δει τις κρατήσεις που έχουν γίνει (και πάλι για το συγκεκριμένο διαμέρισμα). Για κάθε κράτηση φαίνονται δυο ημερομηνίες, η ημερομηνία έναρξης και λήξης της κράτησης.



Tests που αναπτύχθηκαν

Το σύνολο των δοκιμών (unit tests αλλά και e2e tests) που αναπτύχθηκαν για την εφαρμογή E-res ενσωματώθηκε στα παρακάτω αρχεία:

- houselistNB.js (no backend): περιλαμβάνει unit tests τα οποία που προσομοιώνουν τη λειτουργία του server με πλασματικά δεδομένα
- houselistAsync.js (Asynchronized calls): περιλαμβάνει unit tests όπου γίνεται ορατή η συνηθέστερη μέθοδος δοκιμής συναρτήσεων που βασίζονται σε ασύγχρονη λογική (asynchronized callbacks).
- Login_spec.js: περιλαμβάνει e2e tests (end to end) που επαληθεύουν τις διαδικασίες ταυτοποίησης των χρηστών
- houselist_spec.js: στο αρχείο αυτό ενσωματώθηκαν e2e (end to end) έλεγχοι που επαληθεύουν την εύρυθμη λειτουργία παρουσίασης και προβολής των διαμερισμάτων του λογισμικού

Όλα τα παραπάνω αρχεία μπορούν να βρεθούν στα επισυναπτόμενα αρχεία της εφαρμογής που συνοδεύουν την παρούσα τεκμηρίωση. Συγκεκριμένα, τα unit tests βρίσκονται στον υπο-φάκελο *Eres/src/main/webapp/test/spec/controllers* και τα e2e tests στον υπο-φάκελο *Eres/src/main/webapp/e2eTests/*

Πριν προχωρήσουμε στην ανάλυση των παραπάνω δοκιμών έχει αρκετή αξία να αναδειχθεί ο τρόπος με τον οποίον αυτοματοποιήθηκε η διεξαγωγή αυτών. Συγκεκριμένα δημιουργήθηκαν μερικά js scripts που αφορούν κυρίως το Grunt το οποίο διαχειρίζεται την εκτέλεση όλων των δοκιμών. Παρακάτω παραθέτουμε τα σημαντικότερα τμήματα του script *Gruntfile.js* που βρίσκεται στην τοποθεσία */Eres/src/main/webapp*. Για λόγους μορφοποίησης δεν παρουσιάζεται ολόκληρο το αρχείο αλλά τα σημαντικότερα σημεία αυτού.Στο παρακάτω κομμάτι ουσιαστικά καλούνται τα επιμέρους script του karma (karma.conf.js) και του protractor (protractor.conf.js)

```
/ Define the configuration for all the tasks
  grunt.initConfig({

  // karma settings
    karma: {
      unit: {
        configFile: 'test/karma.conf.js',
        singleRun: true
      }
    }

  // Protractor Settings
  protractor: {
    options: {
     configFile: "protractor.conf.js", //protractor config file
     keepAlive: true, // If false, the grunt process stops when the test fails.
     noColor: false, // If true, protractor will not use colors in its output.
     args: {
```

```
        // Arguments passed to the command
      }
     },
    chrome: {
     options: {
      args: {
       browser: "chrome" // define default browser to run e2e tests
      }
     }
    }
.
.
.
connect: { // Define grunt default port
      options: {
        port: 9000,
        // Change this to '0.0.0.0' to access the server from outside.
        hostname: 'localhost',
        livereload: 35729
      },
}
```

Στα επιμέρους js scripts karma.conf.js και protractor.conf.js που προαναφέρθηκαν οι ρυθμίσεις είναι αρκετά απλές και στην πλειοψηφία τους αφορούν τον ορισμό της τοποθεσίας στην οποία βρίσκονται τα εκάστοτε tests. Για παράδειγμα στο protractor.conf.js έχουμε

```
exports.config = {
  // The address of a running selenium server.
  seleniumAddress: 'http://localhost:4444/wd/hub',

  // Spec patterns are relative to the location of this config.
  specs: [
    'e2eTests/*_spec.js'
  ],


  multiCapabilities: [{
    'browserName': 'chrome'
  }],
```

Στην πρώτη γραμμή αναφέρεται η τοποθεσία του WebDriver που χρησιμοποιεί το protractor για να τρέξει σε πραγματικό browser τα e2e tests επιτρέποντας και οπτική παρακολούθηση της εξέλιξης του εκάστοτε test. Ακολούθως ορίζεται ένα μονοπάτι – pattern στο οποίο βρίσκονται όλα τα e2e tests και τέλος περιγράφεται ο προεπιλεγμένος browser που θα χρησιμοποιηθεί (στην περίπτωσή μας ο google chrome)

Στο σημείο αυτό ολοκληρώνεται η διαδικασία αυτοματοποίησης όλων των test κάτι που σημαίνει πως η εκτέλεση όλων τον unit test μπορεί να πραγματοποιηθεί με την παρακάτω εντολή:

grunt karma:unit

ενώ η εκτέλεση των e2e test με την εντολή:

grunt protractor

Unit Tests που αναπτύχθηκαν

Όπως ειπώθηκε και σε προηγούμενα κεφάλαια, προκειμένου να υλοποιηθούν τα unit tests χρησιμοποιήθηκε το Karma Framework σε συνδυασμό με το Jasmine Framework. Πριν γίνει εκτενής περιγραφή των δοκιμών που υλοποιήθηκαν και εκτελέστηκαν αξίζει να αναδειχθεί ο τρόπος διεξαγωγής και σύνταξης αυτών. Συγκεκριμένα ένα Jasmine unit test πέρα από το ότι αποτελεί ένα Js αρχείο, στην τυπική του μορφή έχει την παρακάτω δομή:

```
describe("Test a specific branch", function () {
    //init variables here

    beforeEach(function () {
            .....
            .....
      //runs before each test case
    });

    afterEach(function(){
            .....
            .....
      //runs after each test case
    });

      //First Test case
    it("changes state", function () {
      //first test validations here
            .....
    })

      //Second Test case
    it("adds states", function () {
      //second test validation here
            .....
    })
});
```

Το παραπάνω ενδεικτικό κομμάτι κώδικα απαρτίζεται απο τέσσερις συναρτήσεις:

- Τη συνάρτηση describe, η οποία με μοναδικό όρισμα ένα αλφαριθμητικό – τίτλο περικλείει ένα σύνολο από δοκιμές.
- Τη συνάρτηση it η οποία με μοναδικό όρισμα ένα αλφαριθμητικό – τίτλο περιγράφει μια δοκιμή μονάδας. Μέσα σε αυτή δηλαδή μπορούμε να κάνουμε τους επιθυμητούς ελέγχους. Αξίζει να σημειωθεί πως μπορεί να επαναχρησιμοποιηθεί πολλές φορές μέσα στο ίδιο σύνολο δοκιμών – describe, κάτι που φαίνεται και στο παραπάνω παράδειγμα.
- Τη συνάρτηση beforeEach() η οποία εκτελείται πρίν από κάθε unit test (it). Αποτελεί ένα ιδιαιτέρως χρήσιμο εργαλείο καθώς μας επιτρέπει να αποφύγουμε επαναλαμβανόμενα κομμάτια κώδικα που αρχικοποιούν τα εκάστοτε test cases.
- Τη συνάρτηση afterEach() η οποία εκτελείται μετά από κάθε unit test (it). Ακριβώς η αντίστροφη λειτουργία σε σχέση με την beforeEach()

Προκειμένου να δοκιμασθεί μια πραγματική εφαρμογή θα χρειασθεί να αντιμετωπισθούν μερικές επιπλέον “δυσκολίες”, κάτι που καθιστά ελαφρώς πιο σύνθετα τα test cases που αναπτύχθηκαν σε σχέση με το παραπάνω παράδειγμα. Συγκεκριμένα ο Unit Tester θα πρέπει να λύσει τα παρακάτω προβλήματα:

1. Με δεδομένο ότι το AngularJS Framework διαθέτει ένα ισχυρό MVC (Model – View – Controller) System αλλά και ένα πανίσχυρο dependency Injection system προκύπτει μια τεχνική δυσκολία. Πώς θα αρχικοποιηθούν τα modules (που περιέχουν είτε Controllers είτε Services) προκειμένου να ελεγχθούν μονάδες που ενσωματώθηκαν σε κάποιο από αυτά.
2. Πολλές συναρτήσεις από διάφορους Controllers ή Services “ζητούν” δεδομένα από τον server (μέσω http calls) ο οποίος και απαντάει χρησιμοποιώντας Jax-Rs Rest services στην περίπτωσή μας. Κατά τη φάση των δοκιμών όμως, το server side κομμάτι μπορεί να μην έχει υλοποιηθεί ακόμη ή απλά το κόστος λειτουργίας του να είναι υψηλό. Έτσι προκύπτει η ανάγκη ολοκληρωτικής αποδέσμευσης του client side τμήματος με το server side, κάτι που επιτυγχάνεται εύκολα με τη χρήση “πλασματικών δεδομένων”.
3. Τέλος, δεν είναι λίγες οι φορές όπου μερικές μονάδες που ελέγχονται έχουν ασύγχρονη λογική. Πράγμα που σημαίνει πως δεν ισχύει η σειριακή εκτέλεση των εντολών. Αυτό σημαίνει πως ο έλεγχός τους πρέπει να σχεδιασθεί με τέτοιο τρόπο που να διακόπτεται η ροή του ελέγχου μέχρις ότου επιστραφούν τα επιθυμητά αποτελέσματα της συνάρτησης.

Με γνώμονα τις προαναφερθείσες δυσκολίες και έχοντας κατά νου την εφαρμογή E-res που αναπτύχθηκε στα πλαίσια της εργασίας αυτής θα ακολουθήσει εκτενής ανάλυση των δοκιμών που δημιουργήθηκαν.

Προκειμένου να λύσουμε τα δυο πρώτα προβλήματα που προέκυψαν δημιουργήθηκε το αρχείο houselistNB.js που περιέχει 3 δοκιμές καθώς και τη συνάρτηση beforeEach που αρχικοποιεί τα προαπαιτούμενα στοιχεία.

.....

```
// load the main application
      beforeEach(module('webappApp'));

  // Initialize the controller and a mock scope
  beforeEach(inject(function ($injector) {
      // Set up the mock http service responses
           $httpBackend = $injector.get('$httpBackend');
           // backend definition common for all tests
           var authUserState = $httpBackend.when('GET',USER_STATE_URL)
                                           .respond(userStatus);
           var loginView= $httpBackend.when('GET', 'views/login.html')
                  .respond(returned);
           var mainView= $httpBackend.when('GET', 'views/main.html')
                  .respond(returned);

           var authRequestHandler = $httpBackend.when('GET', HOUSE_LIST_URL)
                                           .respond(returned);
```

```
            $rootScope = $injector.get('$rootScope');
            // The $controller service is used to create instances of controllers
            var $controller = $injector.get('$controller');

            createController = function() {
              return $controller('HouselistCtrl', {'$scope' : $rootScope });
            };
  }));
```

Είναι εύκολα ορατό το πώς γίνεται inject τόσο ο HouselistCtrl Controller του οποίου τις συναρτήσεις θα ελέγξουμε στη συνέχεια, όσο και το $httpBackend service που μας επιτρέπει να ορίσουμε “πλασματικές απαντήσεις” του διακομιστή στην περίπτωση που εκτελεσθεί πχ ένα ερώτημα GET σε ένα συγκεκριμένο URL.
Η πρώτη δοκιμή αφορά στον έλεγχο για το αν η συνάρτηση που αναλαμβάνει να λάβει από τον διακομιστή τη λίστα των διαθέσιμων διαμερισμάτων επιστρέφει μη κενό περιεχόμενο μήκους 1:

```
 it('should return a valid list of Eres houses - no backend needed', function ()
{
      //controller handles async call promises
        var controller = createController();
        $httpBackend.flush(); // enable dummy backend reply

        $rootScope.validateHouses();
        expect($rootScope.errorMsg).toBe(false); //no error msg is enabled
        expect($rootScope.houses.length).toBe(1); // one house
        dump($rootScope.houses);
  });
```

Συνεχίζοντας, το επόμενο Unit test πραγματοποιεί έλεγχο της συνάρτησης που διαπιστώνει αν ένα διαμέρισμα είναι διαθέσιμο μια συγκεκριμένη μέρα. Η απάντηση αναμένεται να είναι θετική (true) αφού γνωρίζουμε εκ των πρωτέρων τις πλασματικές τιμές (returned[0]) που δόθηκαν ως είσοδος.

```
it('should return an unavailable house',function(){
        var controller = createController();
        $httpBackend.flush(); // enable dummy backend reply

        var availability =
$rootScope.checkAvailability(returned[0],"2015/06/11");
        expect(availability).toBe(true); // its available
  })
```

Να σημειωθεί πως στα παραπάνω παραδείγματα η μεταβλητή returned περιέχει πλασματικές εγγραφές η περιγραφή των οποίων κρίθηκε ανούσια.
Το τελευταίο παράδειγμα επικεντρώνεται στην κατανόηση του ιδανικού τρόπου διαχείρισης ασύγχρονων κλήσεων εντός του Unit test. Με δεδομένη μια ασύγχρονη κλήση υλοποιημένη σε AngularJs όπως την παρακάτω

```
      var fetchHousesDef =function(){
              //Creating a deferred object
```

```
            var deferred = $q.defer();

            //Calling Web API to fetch house list
            $http.get('rest/house/list').success(function(data){
              //Passing data to deferred's resolve function on successful
                completion
              deferred.resolve(data);
          }).error(function(){
            //Sending a friendly error message in case of failure
            deferred.reject("An error occured while fetching houses");
          });
            return deferred;
        }
```

θα μπορούσε να χρησιμοποιηθεί το παρακάτω σενάριο ελέγχου ώστε να ελεγχθεί με ασφάλεια η συνάρτηση fetchHousesDef

```
  it('should simulate async promise', inject(function($q, $rootScope) {
        //define an async call
        var deferred = houseService.fetchHousesDef();

        // asyn call promise handle (callback
        var promise = deferred.promise;
        promise.then(
                  function(value) { scope.houses = value; }, //On success
                  function(){scope.houses = undefined}       //On fail
        );

        //Simulate resolving of promise (fake data set)
        deferred.resolve(returned);

        $rootScope.$apply(); //Goes to Resolve – On Success line

        //==========Check if fetchHousesDef returned any house========//
        expect(scope.houses.length).not.toBe(0);

        //Validation check (raises error msg)
        scope.validateHouses();

        //==========Error must be disabled============================//
        expect(scope.errorMsg).toBe(false);
      }));
```

Σε κάθε ασύγχρονη κλήση καλό είναι να προβλέπουμε δυο περιπτώσεις ανάλογα με τη δοκιμή που πρόκειται να ακολουθήσει:

- Περίπτωση επιτυχούς εκτέλεσης και άρα επιστροφή αποτελεσμάτων
- Περίτπωση αποτυχημένης εκτέλεσης και άρα επιστροφή μηνύματος σφάλματος.

Τις δυο αυτές περιπτώσεις αναθέτουμε στη αντικείμενο promise. Ακολούθως δηλώνουμε ποιά από τις δυο περιπτώσεις επιθυμούμε να δοκιμάσουμε. Στην περίπτωσή μας γίνεται κλήση της πρώτης (επιτυχημένη εκτέλεση - resolve) προκειμένου να διαπιστώσουμε αν επιστρέφεται έστω και ένα διαμέρισμα μετά την κλήση της συνάρτησης `fetchHousesDef()`. Στην πραγματικότητα η κλήση αυτή είναι ψευδοασύγχρονη κατά το Unit testing αφού ο χρήστης είναι αυτός που ελέγχει πότε θα “επιστρέψει” η μέθοδος με τη χρήση της εντολής $apply στο τρέχον scope. Ακολούθως μπορούν να

εκτελεσθούν με τον γνωστό πλέον τρόπο έλεγχοι στα δεδομένα που επεστράφησαν με τη χρήση της μεθόδου expect(...)

E2E tests που υλοποιήθηκαν

Ως κύριο εργαλείο για τη διεξαγωγή των δοκιμών αυτών χρησιμοποιήθηκε το Protractor Framework. Και εδώ η σύνταξη ενός e2e test είναι πανομοιότυπη με αυτή του Jasmine με τη διαφορά ότι πλέον έχουμε στη διάθεσή μας εργαλεία – κλάσεις που μας δίνουν πρόσβαση σε DOM – Html elements. Επιπλέον παρέχει τη δυνατότητα εκτέλεσης των δοκιμών σε πραγματικές συνθήκες. Αυτό γιατί η εκτέλεση πραγματοποιείται σε πραγματικό browser στον οποίο και εκτελούνται ένα – ένα τα βήματα του test case επιτρέποντας στον χρήστη να κάνει και “οπτικό” έλεγχο. Πρακτικά αυτό σημαίνει πως υπάρχει τόσο η δυνατότητα πρόσβασης στο περιεχόμενό τους όσο και η δυνατότητα ελέγχου αυτών. Χαρακτηριστικό παράδειγμα αποτελεί η ικανότητα να ελέγξουμε πλήκτρα - buttons της εφαρμογής καθώς και να συμπληρώσουμε φόρμες αυτής.

Στο παρακάτω παράδειγμα, στις πρώτες γραμμές αρχικοποιούνται μερικές μεταβλητές που αντιστοιχούν σε dom Elements. Είναι εφικτή η προσπέλαση των στοιχείων αυτών με πολλούς τρόπους, όπως με το όνομα της κλάσης ή το όνομα του εκάστοτε element κ.α. Πριν την εκτέλεση όλων των δοκιμών κατευθύνουμε τον browser σε μια συγκεκριμένη διεύθυνση URL που αντιστοιχεί στη σελίδα ταυτοποίησης των χρηστών με τη χρήση της εντολής `browser.get()`

Αρχικά, στην πρώτη δοκιμή επιχειρείται η προσπέλαση μιας σελίδας που εμφανίζει τη λίστα των διαμερισμάτων, χωρίς όμως πρώτα να έχει γίνει ταυτοποίηση των διαπιστευτηρίων του χρήστη. Συνεπώς αναμένουμε από την εφαρμογή να απαγορεύσει την είσοδο στη σελίδα αυτή μεταφέροντάς μας στην οθόνη ταυτοποίησης `expect(browser.getCurrentUrl()).toEqual(loginURL).`

Αντίθετα στο δεύτερο σενάριο συμπληρώνονται στη φόρμα ταυτοποίησης τα στοιχεία ενός έγκυρου χρήστη `email.sendKeys('admin@admin')`. Ο χρήστης αυτός προβλέπεται να έχει πρόσβαση στη σελίδα αυτή. Πρώτα όμως καλό είναι να εξαλειφθούν τυχόν προηγούμενες τιμές των πεδίων Username & Password `password.clear()`.Στη συνέχεια πραγματοποιούμε είσοδο στο σύστημα πατώντας το πλήκτρο Login που αρχικοποιήθηκε νωρίτερα. Η έκφραση `loginButton.click()` εργάζεται σε αυτήν ακριβώς την κατεύθυνση και ακαριαία μας οδηγεί στη σελίδα υποδοχής του E-res. Άρα λοιπόν σε αυτή την περίπτωση αναμένουμε μετά τις παραπάνω ενέργειες το τρέχον URL να είναι διάφορο του Login URL. Τέλος μετά την πραγματοποίηση του ελέγχου αυτού εκτελούμε έξοδο από το σύστημα με τη χρήση του πλήκτρου Logout (ομοίως με το Login Button).

```
describe('Authentication Procedure', function() {

  var loginURL='http://localhost:8080/eres/app/#/login';
  var email = element(by.name('email1'));
```

```
  var password = element(by.name('passwd1'));
  var loginButton = element(by.name('loginBtn'));
  var logoutButton = element(by.name('logoutBtn'));
  var error = element(by.model('loginError'));

  beforeEach(function() {
      browser.get(loginURL);
  });



  afterEach(function(){

  });

  it('should redirect to the login page if trying to load protected page while
not authenticated', function() {


    browser.get('http://localhost:8080/eres/app/#/houseList');
    expect(browser.getCurrentUrl()).toEqual(loginURL);
  });

  it('should accept a valid email address and password', function() {
      email.clear();
    password.clear();

    email.sendKeys('admin@admin');
    password.sendKeys('admin');
    loginButton.click();
    expect(browser.getCurrentUrl()).not.toEqual(loginURL);
    logoutButton.click();
  });
});
```

Στο παρακάτω σενάριο εκτελούμε τις ίδιες ενέργειες σε ό,τι αφορά το κομμάτι της αρχικοποίησης των DOM Elements. Και εδώ επιχειρούμε είσοδο στο σύστημα χρησιμοποιώντας τα στοιχεία ενός υπαρκτού χρήστη. Εξετάζουμε και πάλι αν η είσοδός μας στη σελίδα εμφάνισης του συνόλου των διαμερισμάτων είναι πετυχημένη. Έχοντας λοιπόν εμφανίσει τη λίστα των διαμερισμάτων, επιλέγουμε τυχαία να δούμε το δεύτερο από αυτά “πατώντας” το αντίστοιχο πλήκτρο `element(by.name('viewHouse2')).click()` . Στο σημείο αυτό, αξίζει να σημειωθεί πως σημαντικό ρόλο στην ομαλή διεξαγωγή των δοκιμών αλλά και στην εύκολη πρόσβαση των DOM Elements παίζει ο τρόπος με τον οποίο δόθηκαν τα names κατά την υλοποίηση. Στην περίπτωση του E-res τα id των house view πλήκτρων δίδονται δυναμικά με βάση το Pattern viewHouse+HouseId. Με τον τρόπο αυτό παρέχουμε στον tester τη δυνατότητα να τα εντοπίσει – ξεχωρίσει χωρίς πρόβλημα. Ολοκληρώνοντας το σενάριο της δοκιμής αυτής ελέγχεται αν το τρέχον URL (browser.getCurrentUrl()) αντιστοιχεί στο διαμέρισμα που επιλέξαμε να επισκεφθούμε.

```
describe('Houselist Procedure', function() {
```

```
  var loginURL='http://localhost:8080/eres/app/#/login';
  var houselistURL = 'http://localhost:8080/eres/app/#/houseList';
  var email = element(by.name('email1'));
  var password = element(by.name('passwd1'));
  var loginButton = element(by.name('loginBtn'));
  var logoutButton = element(by.name('logoutBtn'));
  var error = element(by.model('loginError'));
  var secondHouseLink;
  it('should accept a valid email address and password', function() {
    browser.get(loginURL);
    email.clear();
    password.clear();
    email.sendKeys('admin@admin');
    password.sendKeys('admin');
    loginButton.click();
    expect(browser.getCurrentUrl()).not.toEqual(loginURL);
  });

  it('should show houselist', function() {
      browser.get(houselistURL);
    expect(browser.getCurrentUrl()).toEqual(houselistURL);
  });

  it('should visit the house with id equal to 2',function(){
        browser.get(houselistURL);
      secondHouseLink = element(by.name('viewHouse2'));
      secondHouseLink.click();
      expect(browser.getCurrentUrl()).
            toEqual("http://localhost:8080/eres/app/#/house/2");
      logoutButton.click();
  });

});
```

Συμπεράσματα

Όπως και στις standalone εφαρμογές έτσι και στις διαδικτυακές εφαρμογές υπάρχει άμεση ανάγκη για δοκιμές. Στο έργο αυτό εξετάστηκαν ίσως τα βασικότερα είδη δοκιμών σε ό,τι αφορά τις web εφαρμογές (unit & e2e testing) που όμως δεν αποτελούν μονόδρομο όπως επισημάνθηκε στο κεφάλαιο "Κρίσιμα *στοιχεία προς επαλήθευση*". Θα μπορούσε να πει κανείς πως τα δυο αυτά είδη δοκιμών αλληλοσυμπληρώνονται καθώς πετυχαίνουν την εύρεση προβλημάτων διαφορετικής φύσεως,:

- προβλήματα δηλαδή που προκύπτουν από σφάλματα που κρύβονται σε κάποια μονάδα και
- προβλήματα που προκύπτουν από το συνδυασμό δυο οι περισσοτέρων μονάδων.

Ακόμα και στην μικρή εφαρμογή που υλοποιήθηκε για τους σκοπούς της εργασίας, τα σφάλματα που διαπιστώθηκαν δεν ήταν λίγα. Κάτι που σημαίνει πως ο συνδυασμός των παραπάνω δοκιμών είχε θετικά αποτελέσματα στην επαλήθευση του E-res. Στον αντίποδα ο χρόνος που δαπανήθηκε για την εγκατάσταση και διαρρύθμιση όλων των εργαλείων που χρησιμοποιήθηκαν, δεν ήταν λίγος, κάτι που σε πραγματικά περιβάλλοντα ανάπτυξης δεν είναι (δυστυχώς) εύκολα αποδεκτό. Ωστόσο αν σκεφτεί κανείς πως τα περισσότερα σενάρια δοκιμών σχεδιάζονται μια φορά και δεν επιδέχονται (και δεν χρειάζεται) συχνές παρεμβάσεις, (ασχέτως αν ο κώδικας μεταβάλλεται) οδηγείται στο τελικό μας συμπέρασμα. Ο χρόνος που δαπανήθηκε για την δημιουργία των δοκιμών και την εγκατάσταση των εργαλείων, κερδίζεται εις διπλούν (έστω και μακροπρόθεσμα) αφού η ομάδα ανάπτυξης εστιάζει ακαριαία στην πραγματική αιτία του σφάλματος αποφεύγοντας αρκετές ώρες ατελείωτης αποσφαλμάτωσης.

Πηγές

http://www.ng-newsletter.com/posts/practical-protractor.html Strategy for Testing

https://docs.angularjs.org/guide/e2e-testing AngularJs E2E Testing

http://andyshora.com/unit-testing-best-practices-angularjs.html Unit Testing Theory

http://www.sitepoint.com/unit-and-e2e-testing-in-angularjs/ End to End Tests

http://thenextweb.com/apps/2013/11/28/guide-testing-web-app-steps-approach-testing-get-sessions/ Front end testing advises