ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ
ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

# Backbone, AngularJS, Ember: Συγκριτική ανάλυση και σενάρια χρήσης JavaScript frameworks

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

**Χρήστος Παπουτσάκης**

**Επιβλέπων :** Θεοδώρα Βαρβαρίγου
Καθηγήτρια ΕΜΠ

Αθήνα, Οκτώβριος 2017

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΕΠΙΚΟΙΝΩΝΙΩΝ, ΗΛΕΚΤΡΟΝΙΚΗΣ
ΚΑΙ ΣΥΣΤΗΜΑΤΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ

# Backbone, AngularJS, Ember: Συγκριτική ανάλυση και σενάρια χρήσης JavaScript frameworks

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

**Χρήστος Παπουτσάκης**

**Επιβλέπων :** Θεοδώρα Βαρβαρίγου
Καθηγήτρια ΕΜΠ

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 16η Οκτωβρίου 2017.

| | | |
|---|---|---|
| ..... | ..... | ..... |
| Θεοδώρα Βαρβαρίγου | Εμμανουήλ Βαρβαρίγος | Δημήτριος Ασκούνης |
| Καθηγήτρια ΕΜΠ | Καθηγητής ΕΜΠ | Καθηγητής ΕΜΠ |

Αθήνα, Οκτώβριος 2017

.....
Χρήστος Παπουτσάκης

Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

# Περίληψη

Εν έτει 2017, ο τομέας του front-end web development, δηλαδή της ανάπτυξης εφαρμογών διαδικτύου με τις οποίες αλληλεπιδρά ο χρήστης, αποτελεί “πεδίο μάχης”. Το εύρος της επιλογής ανάμεσα στα διαφορετικά εργαλεία, βιβλιοθήκες και frameworks (τα οποία αποτελούν ολοκληρωμένες λύσεις) είναι πραγματικά πολύ μεγάλο και ολοένα αυξάνεται μέρα με τη μέρα. Δικαιολογημένα λοιπόν, μία επιχείρηση ή ένας ιδιώτης, που θέλει να δημιουργήσει μία καινούρια εφαρμογή στο χώρο βρίσκεται στη δύσκολη θέση της επιλογής ανάμεσα στην τεράστια γκάμα των δυνατών λύσεων.

Σκοπός της παρούσης διπλωματικής εργασίας ήταν η σύγκριση τριών διαφορετικών JavaScript frameworks και η μελέτη των επιδόσεών τους ανάλογα με την κατά περίπτωση εφαρμογή. Συγκεκριμένα, μελετήθηκαν τα Backbone, AngularJS και Ember, τρία frameworks κατάλληλα για τη δημιουργία εφαρμογών με τις οποίες αλληλεπιδρά ο χρήστης, και καταγράφηκε η επίδοσή τους πάνω σε τρεις τύπους δοκιμαστικών εφαρμογών.

Το κυριότερο και πιο σημαντικό συμπέρασμα που προέκυψε από τη διεξαγωγή των διαφόρων πειραμάτων είναι πως δεν υφίσταται η ύπαρξη ενός καθολικού framework, ιδανικού για όλες τις περιπτώσεις και σενάρια χρήσης και σίγουρα, πριν τη λήψη της τελικής απόφασης, θα πρέπει να έχουν προσδιοριστεί όσο το δυνατόν καλύτερα το ποιες είναι οι κύριες απαιτήσεις της εφαρμογής και πόσο εύκολα μπορεί να προσαρμοστεί το κάθε framework σε αυτές. Ωστόσο, μπορούμε με ασφάλεια να εξάγουμε, ότι στην περίπτωση μίας σχετικά μικρής σε έκταση εφαρμογής η επιλογή του Backbone είναι πιο πιθανή να επιφέρει τα καλύτερα αποτελέσματα, ενώ στην περίπτωση μίας μεγαλύτερης, πολύπλοκης εφαρμογής το AngularJS αποτελεί ίσως την λιγότερο ριψοκίνδυνη λύση.

Τέλος, η καταγραφή των αποτελεσμάτων αυτών και της μεθοδολογίας που ακολουθήθηκε στις συγκεκριμένες δοκιμαστικές εφαρμογές είναι δυνατό να αποτελέσει τη βάση για την διεξαγωγή πειραμάτων και μετρήσεων πάνω σε πραγματικές, σύνθετες διαδικτυακές εφαρμογές είτε να καταστεί το βασικό κριτήριο στην επιλογή του τρόπου υλοποίησης για μία καινούρια, απλούστερη εφαρμογή που πλησιάζει σε περιεχόμενο ή εμπίπτει στην κατηγορία μίας εκ των δοκιμαστικών εφαρμογών.



**Λέξεις Κλειδιά:** angular, angularjs, backbone, ember, javascript framework, διαδικτυακή εφαρμογή, επιδόσεις εφαρμογής, αλληλεπίδραση με χρήστη

## Abstract

In year 2017, the field of front-end web development is a real "battlefield". The variety of the different tools, libraries and frameworks is already huge, and keeps growing day by day. Unsurprisingly, a company or an individual that wants to create a new application is at a loss, when it comes to deciding what tools suit best the needs of the company and the specific application.

The scope of this thesis was the comparison among three different JavaScript frameworks and the analysis of their performance depending on the application category at hand. The frameworks in question were Backbone, AngularJS and Ember, which are client side JavaScript frameworks renowned for the creation of the user interface of a web application, which is the part that the end user sees and interacts with, and their performance was recorded for each type of three different test applications.

The most important conclusion that we arrived to from conducting these experiments is that a universal framework, suitable for all purposes and use cases is non-existent, and before making the final choice, the requirements of the application and the adaptability of each framework upon them should be researched as deeply as possible. Nevertheless, it is safe enough to assume, that in case of a relatively small in size application choosing Backbone over the other two is most probably going to produce the best outcome, whereas in case of a large, complex application AngularJS might be the safest bet.

Finally, the writing down of the results and the methodology followed is possible to form the basis upon which more experiments and measurements will be conducted on real world, complex applications, or, alternatively, constitute the main criterion in order to select the most suitable way of creating a new application that is conceptually close to the tested ones.



**Keywords**: angular, angularjs, backbone, ember, javascript framework, web application, application performance, user interaction

# Πίνακας περιεχομένων

# Κεφάλαιο 1

# Εισαγωγή

## 1.1 Εφαρμογές Διαδικτύου (Web Applications)

Εφαρμογή Διαδικτύου, ή αλλιώς, διαδικτυακή εφαρμογή (web application ή web app) ονομάζεται κάθε εφαρμογή η οποία είναι διαθέσιμη στους χρήστες της μέσω του Διαδικτύου (Internet) και το μόνο που χρειάζεται ο χρήστης για να την χρησιμοποιήσει είναι ένας περιηγητής (browser). Οι εφαρμογές αυτές εκτελούνται σε απομακρυσμένους υπολογιστές οι οποίοι ονομάζονται εξυπηρετητές (servers) και μπορούν να παρέχουν τις υπηρεσίες τους σε περισσότερους του ενός χρήστη ταυτόχρονα. Μερικά από τα πλεονεκτήματά τους είναι:

- **Η άμεση πρόσβαση από οποιαδήποτε συσκευή.** Οι χρήστες μπορούν εύκολα και γρήγορα να έχουν πρόσβαση στην εφαρμογή μέσω οποιασδήποτε συσκευής η οποία έχει δυνατότητα σύνδεσης στο Internet και στην οποία υπάρχει εγκατεστημένος ένας σύγχρονος browser, ούτως ώστε να υπάρχει πλήρη συμβατότητα με όλες τις διαδικτυακές εφαρμογές.
- **Η συμβατότητα με όλα τα λειτουργικά συστήματα.** Εφόσον ικανοποιείται ο περιορισμός της ύπαρξης ενός σύγχρονου browser τότε το λειτουργικό σύστημα της συσκευής δεν παίζει ρόλο στη δυνατότητα χρήσης της εφαρμογής.
- **Η μικρή κατανάλωση πόρων και χώρου.** Οι διαδικτυακές εφαρμογές δεν εκτελούνται στον υπολογιστή του χρήστη επομένως χρησιμοποιούν λίγη επεξεργαστική ισχύ και μνήμη και μηδενικό ή ελάχιστο αποθηκευτικό χώρο στο δίσκο, αναλόγως την εφαρμογή.
- **Η γρήγορη αναβάθμιση.** Σε περίπτωση αναβάθμισης, αυτή λαμβάνει χώρα μόνο στον server, επομένως δεν απαιτείται καμία ενέργεια από την πλευρά του χρήστη.

Όμως, παρά τα πλεονεκτήματά τους, παρουσιάζουν και μερικά μειονεκτήματα, όπως:

- **Η αδυναμία χρήσης χωρίς σύνδεση στο Internet.** Σε περίπτωση που δεν υπάρχει σύνδεση στο Internet, δηλαδή η συσκευή είναι offline, τότε ο χρήστης δεν έχει τη δυνατότητα χρήσης της εφαρμογής. Μοναδική εξαίρεση αποτελούν οι PWA εφαρμογές (Progressive Web Apps), οι οποίες αποτελούν εφαρμογές τελευταίας τεχνολογίας που προσφέρουν δυνατότητα χρήσης ακόμη και χωρίς σύνδεση στο Internet, όμως προς το παρόν αποτελούν μόνο ένα πολύ μικρό κομμάτι των διαδικτυακών εφαρμογών.

- **Η μη πλήρης συμβατότητα όλων των browsers.** Συγκεκριμένα, μερικοί παλαιότεροι browsers συνήθως αντιμετωπίζουν προβλήματα συμβατότητας με τις πιο καινούριες εφαρμογές, διότι αυτές χρησιμοποιούν στοιχεία τα οποία δεν υπήρχαν όταν είχαν κατασκευαστεί εκείνοι οι browsers. Αυτό μπορεί όμως να αντιμετωπιστεί εύκολα από τον χρήστη με μία αναβάθμιση σε έναν μοντέρνο browser, και φροντίζοντας ούτως ώστε να είναι πάντα ενημερωμένος στην τελευταία έκδοση.
- **Η καθυστέρηση στον χρόνο φόρτωσης και αλληλεπίδρασης.** Αυτό το πρόβλημα εμφανίζεται κυρίως στις πιο καινούριες, σύνθετες εφαρμογές με την αυξημένη λειτουργικότητα. Είναι λογικό όσο προστίθενται νέες λειτουργίες και χαρακτηριστικά σε μία διαδικτυακή εφαρμογή, τόσο να αυξάνεται και ο χρόνος που κάνει ο browser να την φορτώσει από το Internet, και να την επεξεργαστεί κατάλληλα ούτως ώστε ο χρήστης να μπορεί να την χρησιμοποιήσει και να αλληλεπιδράσει μαζί της. Αυτό αποτελεί μείζον πρόβλημα ειδικότερα στις φορητές συσκευές όπως τα «έξυπνα» κινητά τηλέφωνα (smartphones) κλπ., και στις περιοχές που δεν υπάρχει υποδομή για γρήγορη σύνδεση με το Internet.

## 1.2 Αντικείμενο διπλωματικής

Στα πλαίσια της διπλωματικής αναπτύχθηκαν τρεις απλές διαδικτυακές εφαρμογές σε τρία διαφορετικά JavaScript frameworks, και αξιολογήθηκε η απόδοση τους σε τρεις δημοφιλείς browsers. Στόχος της είναι, να αναλύσει και να συγκρίνει τις ομοιότητες και διαφορές μεταξύ των τριών frameworks για τον εκάστοτε τύπο της εφαρμογής, και να καταλήξει σε συμπεράσματα όσον αφορά στην καταλληλότητα χρήσης του κάθε framework για τον συγκεκριμένο τύπο εφαρμογής.

### 1.2.1 Συνεισφορά

Η συνεισφορά της εργασίας συνοψίζεται ως εξής:

1) **Η εργασία παρέχει μία σύγκριση ανάμεσα στα πιο δημοφιλή πρότυπα (patterns) αρχιτεκτονικής σχεδίασης διεπαφών χρήστη (user interfaces).** Συγκεκριμένα, παρουσιάζονται τα πρότυπα MVC, MVVM, MVP και αναλύονται τα πλεονεκτήματα και μειονεκτήματα του καθενός.
2) **Η εργασία παρουσιάζει έναν διαχωρισμό των διαδικτυακών εφαρμογών σε τρεις βασικές κατηγορίες.** Βέβαια, οι σύγχρονες, πολύπλοκες διαδικτυακές εφαρμογές δε μπορούν να ενταχθούν σε μία μόνο κατηγορία, όμως η παρουσίαση των βασικών κατηγοριών βοηθά στην καλύτερη κατανόηση του περιεχομένου που πραγματεύονται όλες οι εφαρμογές.

3) **Η εργασία περιγράφει τρία από τα πιο γνωστά JavaScript frameworks και αναλύει τα δυνατά σημεία αλλά και τα μειονεκτήματα του καθενός.** Τα εν λόγω frameworks, Backbone, Angular και Ember, χρησιμοποιούνται ευρέως για την κατασκευή των user interfaces αμέτρητων διαδικτυακών εφαρμογών και είναι υπεύθυνα σε μεγάλο βαθμό για την απόδοση της εφαρμογής.
4) **Η εργασία παρέχει υλοποιήσεις τριών απλών εφαρμογών σε όλα τα προαναφερθέντα frameworks.** Έτσι, είναι δυνατόν να φανούν ξεκάθαρα οι διαφορές στην υλοποίηση για το καθένα.
5) **Η εργασία διεξάγει πειράματα πάνω στις διαφορετικές υλοποιήσεις των εφαρμογών και καταγράφει τα αποτελέσματά τους.** Η καταγραφή αυτή των αποτελεσμάτων οδηγεί σε συμπεράσματα σχετικά με την καταλληλότητα χρήσης του κάθε framework πανω στον συγκεκριμένο τύπο εφαρμογής και καταλήγει στη συνολική αξιολόγηση της κάθε υλοποίησης.

## 1.3 Οργάνωση κειμένου

Το κείμενο της διπλωματικής αποτελείται από 8 κεφάλαια συνολικά. Στο κεφάλαιο 2 αναλύεται το θεωρητικό υπόβαθρο της εργασίας, όπου δίνονται οι ορισμοί και περιγράφονται οι σχετικές έννοιες ως προς τα εργαλεία που χρησιμοποιήθηκαν. Στο κεφάλαιο 3 γίνεται η ανάλυση των απαιτήσεων του συστήματος, και η κατηγοριοποίηση των διαδικτυακών εφαρμογών, πάνω στις οποίες αναπτύχθηκαν υλοποιήσεις. Το κεφάλαιο 4 περιέχει εκτενή περιγραφή των frameworks που χρησιμοποιήθηκαν για τις υλοποιήσεις των εφαρμογών. Στο κεφάλαιο 5 παρουσιάζονται οι λεπτομέρειες της καθεμίας υλοποίησης μαζί με τα κομμάτια κώδικα που χρήζουν σχολιασμού, καθώς και οι πλατφόρμες στις οποίες εκτελούνται. Στο κεφάλαιο 6 περιγράφεται η διαδικασία διεξαγωγής των πειραμάτων σχετικά με την απόδοση της εκάστοτε υλοποίησης και η καταγραφή των αποτελεσμάτων. Τέλος, το κεφάλαιο 7 αποτελεί τον επίλογο της εργασίας, όπου δίνεται η σύνοψη, τα σημαντικότερα συμπεράσματα καθώς και πιθανές μελλοντικές επεκτάσεις, ενώ το κεφάλαιο 8 τη βιβλιογραφία, με την παράθεση όλων των πηγών που χρησιμοποιήθηκαν στο κείμενο.

# Κεφάλαιο 2

# Θεωρητικό υπόβαθρο

## 2.1 JavaScript

Η JavaScript είναι μια γλώσσα προγραμματισμού σεναρίων (scripting language) για ηλεκτρονικούς υπολογιστές. Αρχικά, αποτέλεσε μέρος της υλοποίησης των περιηγητών (browsers), ώστε τα σενάρια από την πλευρά του χρήστη (client-side scripts) να μπορούν να επικοινωνούν με τον ίδιο τον χρήστη, να ανταλλάσουν δεδομένα ασύγχρονα με το Internet (AJAX) και να αλλάζουν δυναμικά το περιεχόμενο της σελίδας (DOM).

Η JavaScript είναι μία γλώσσα με τα εξής κύρια χαρακτηριστικά:

- **Χρησιμοποιεί διερμηνέα (interpreter).** Αυτό σημαίνει πως ο κώδικάς της εκτελείται γραμμή-γραμμή και όχι όλος μαζί αφότου μεταγλωττιστεί σε κώδικα μηχανής (machine code), όπως θα γινόταν σε μία γλώσσα που χρησιμοποιεί μεταγλωττιστή (compiler).
- **Είναι δυναμική (dynamically typed) με ασθενές σύστημα τύπων (weak typing).** Δηλαδή, ο έλεγχος των τύπων των μεταβλητών γίνεται κατά τη διάρκεια εκτέλεσης του προγράμματος και η γλώσσα δεν είναι αυστηρή όσον αφορά τους τύπους, το οποίο σημαίνει πως μετατροπές τύπων (type casts) κλπ επιτρέπονται.
- **Οι συναρτήσεις είναι αντικείμενα πρώτης τάξης (first class functions).** Αυτό ουσιαστικά σημαίνει πως μπορούν να χρησιμοποιηθούν όπως οποιαδήποτε άλλη τιμή, δηλαδή οι συναρτήσεις μπορούν να πάρουν ως όρισμα άλλες συναρτήσεις, να επιστρέφουν συναρτήσεις κλπ.
- **Υποστηρίζει διαφορετικά στυλ προγραμματισμού (multi-paradigm).** Ο προγραμματιστής είναι ελεύθερος να επιλέξει ανάμεσα σε προστακτικό (imperative), αντικειμενοστρεφές (object-oriented) και συναρτησιακό (functional) στυλ προγραμματισμού και συνδυασμούς αυτών.
- **Βασίζεται στην κληρονομικότητα μέσω πρωτοτύπων (prototypal inheritance).** Αυτό αποτελεί βασική διαφορά σε σχέση με τη Java και τις υπόλοιπες object-oriented γλώσσες που υλοποιούν κληρονομικότητα μέσω κλάσεων και αξίζει ιδιαιτέρως να σημειωθεί.

Πλέον, η JavaScript έχει φτάσει να χρησιμοποιείται και σε εφαρμογές εκτός ιστοσελίδων, όπως για τη δημιουργία ηλεκτρονικών παιχνιδιών (πχ unity3d), εφαρμογών για κινητά τηλέφωνα smartphones (πχ react native), εφαρμογών για ηλεκτρονικούς υπολογιστές (πχ electron) αλλά και διαδικτυακών εφαρμογών που εκτελούνται σε servers, μέσω της πλατφόρμας Node.js.

Κάτι άλλο το οποίο αξίζει να αναφερθεί, είναι, ότι ο οργανισμός τυποποίησης γλωσσών προγραμματισμού ECMA International είναι αυτός που είναι υπεύθυνος για την τυποποιημένη μορφή της γλώσσας που έχει σήμερα, και γι αυτό πολλές φορές αποκαλείται και ECMAScript, με την πιο πρόσφατη έκδοση που υποστηρίζεται από όλους τους μοντέρνους browsers να ειναι η ECMAScript 2015 ή αλλιώς ES6.

## 2.2 Πλαίσια ανάπτυξης εφαρμογών διαδικτύου (Web frameworks)

Τα πλαίσια ανάπτυξης εφαρμογών διαδικτύου ή αλλιώς web frameworks αποτελούν την πλέον ολοκληρωμένη λύση για τη δημιουργία των σύγχρονων και πολύπλοκων διαδικτυακών εφαρμογών. Λόγω των αυξημένων απαιτήσεων των εφαρμογών δεν είναι επαρκής η χρησιμοποίηση μόνο βοηθητικών εργαλείων και βιβλιοθηκών. Γι' αυτό λοιπόν τα τελευταία χρόνια, τα frameworks αποκτούν όλο και μεγαλύτερη δημοτικότητα, διότι δεν παρέχουν μόνο κάποιες ομαδοποιημένες λειτουργίες όπως οι βιβλιοθήκες, αλλά μία ολόκληρη πλατφόρμα πάνω στην οποία στηρίζεται η κατασκευή της εφαρμογής. Πλέον, η δημιουργία μιας διαδικτυακής εφαρμογής σχεδόν ποτέ δεν ξεκινά από το μηδέν, αλλά οι προγραμματιστές έχουν να πάρουν σημαντικές αποφάσεις όσον αφορά την επιλογή του καταλληλότερου framework και βιβλιοθηκών.

Ο κύριος στόχος των frameworks είναι λοιπόν να δώσουν μία κατεύθυνση στον προγραμματιστή, αποτελώντας τη βάση, το θεμέλιο πάνω στο οποίο θα «χτιστεί» η εφαρμογή. Αυτό το καταφέρνουν μέσω των εξής χαρακτηριστικών:

1) Προωθώντας την επαναχρησιμοποίηση του κώδικα (code reuse) και παρέχοντας έτοιμες υλοποιήσεις για εργασίες όπως διαχείριση συνεδρίας (session management), βιβλιοθήκες για πρόσβαση και παραμετροποίηση βάσεων δεδομένων (database access and configuration) αλλά και μηχανές για τη σχεδίαση προτύπων των ιστοσελίδων (template engines).
2) Προβλέποντας για ενέργειες όπως ο έλεγχος ταυτότητας χρήστη (user authentication), αιτήματα AJAX (AJAX requests), η δρομολόγηση της εφαρμογής (routing) και την αποθήκευση μέρους της εφαρμογής στη μνήμη του υπολογιστή του χρήστη (caching) με σκοπό την εξοικονόμηση πόρων και τη γρηγορότερη φόρτωσή της μεταγενέστερα.

### 2.2.1 Model-View-Controller (MVC )

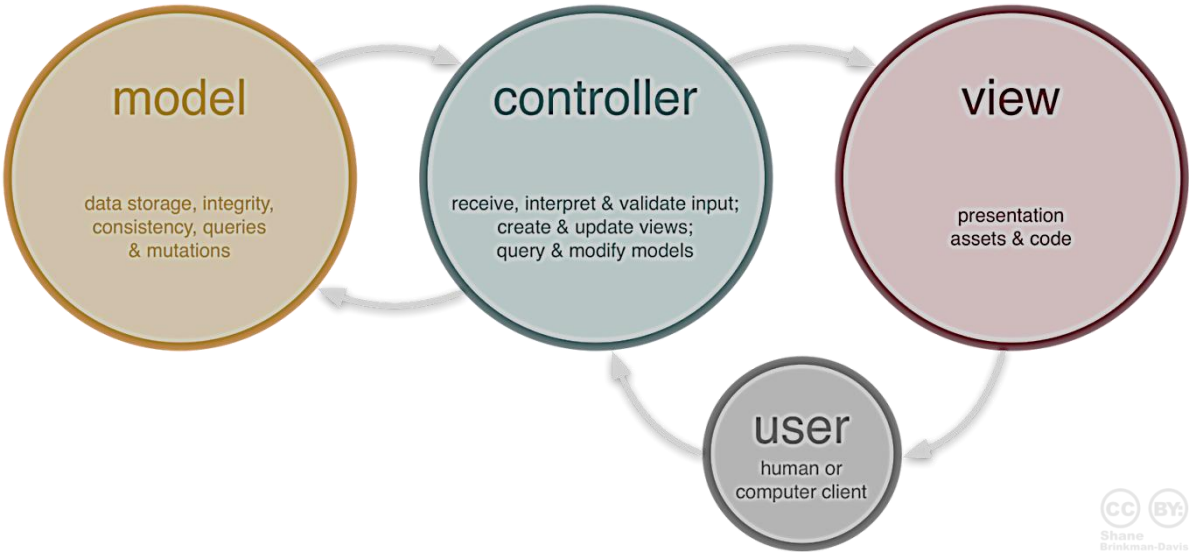
Το Model-View-Controller ή αλλιώς MVC αποτελεί ένα από τα πιο γνωστά μοτίβα αρχιτεκτονικής λογισμικού που χρησιμοποιείται για τη δημιουργία εφαρμογών με περιβάλλον αλληλεπίδρασης χρήστη (user interface). Ιστορικά, χρησιμοποιήθηκε για τη δημιουργία γραφικού περιβάλλοντος χρήστη (GUI) σε εφαρμογές για σταθερούς

υπολογιστές, όμως στην πορεία έγινε πολύ δημοφιλές και επεκτάθηκε στις διαδικτυακές εφαρμογές και ακόμα και σε εφαρμογές κινητών τηλεφώνων.

Το μοτίβο αυτό διαιρεί την εφαρμογή σε τρία διασυνδεδεμένα μέρη με σκοπό να διαχωρίσει την εσωτερική αναπάρασταση της πληροφορίας που παρουσιάζει αλλά και δέχεται από τον χρήστη. Τα βασικά μέρη από τα οποία αποτελείται το MVC είναι τα εξής:

- **Το model.** Είναι το πιο κεντρικό κομμάτι του μοτίβου, το οποίο διαχειρίζεται τα δεδομένα, τη λειτουργικότητα (business logic) και τους κανόνες της εφαρμογής.
- **Το view.** Αναπαριστά με γραφικό τρόπο την πληροφορία του model και παράγει καινούριο υλικό όταν υπάρχουν αλλαγές στα δεδομένα. Είναι αυτό που βλέπει ο χρήστης.
- **Ο controller.** Είναι υπεύθυνος για την αποστολή εντολών στο model για την ενημέρωση της κατάστασης του. Μπορεί επίσης να στέλνει εντολές στο αντίστοιχο view για την ανανέωση της παρουσίασης των δεδομένων του model μέσω του εαυτού του στο χρήστη.

Μπορεί να παρασταθεί σχηματικά με το ακόλουθο διάγραμμα:




Model-View-Controller Diagram

Οι κυριότεροι στόχοι του MVC είναι:

i. Η παράλληλη ανάπτυξη της εφαρμογής. Επειδή το μοτίβο αυτό «αποσυνδέει» τα συστατικά μέρη της εφαρμογής, οι προγραμματιστές είναι δυνατόν να εργαστούν

παράλληλα σε διαφορετικά κομμάτια χωρίς να απαιτείται ο συνεχής συντονισμός τους.

ii. Η επαναχρησιμοποίηση του κώδικα. Δημιουργώντας δομικά στοιχεία τα οποία είναι ανεξάρτητα μεταξύ τους, οι προγραμματιστές είναι σε θέση να χρησιμοποιήσουν τα ίδια αυτά στοιχεία και σε άλλες εφαρμογές. Αυτό για παράδειγμα μπορεί να γίνει κάνωντας χρήση του ίδιου ή παρόμοιου view για άλλη εφαρμογή με διαφορετικό model, χωρίς να χρειάζεται αλλαγή στον τρόπο παρουσίασης των δεδομένων στο χρήστη.

Η μεγάλη δημοφιλία του MVC, έχει δώσει , όπως ήταν αναμενόμενο, βάση στο να δημιουργηθούν και πολλές παραλλαγές του, όπως τα Model-View-ViewModel και Model-View-Presenter, τα οποία θα δούμε στη συνέχεια.

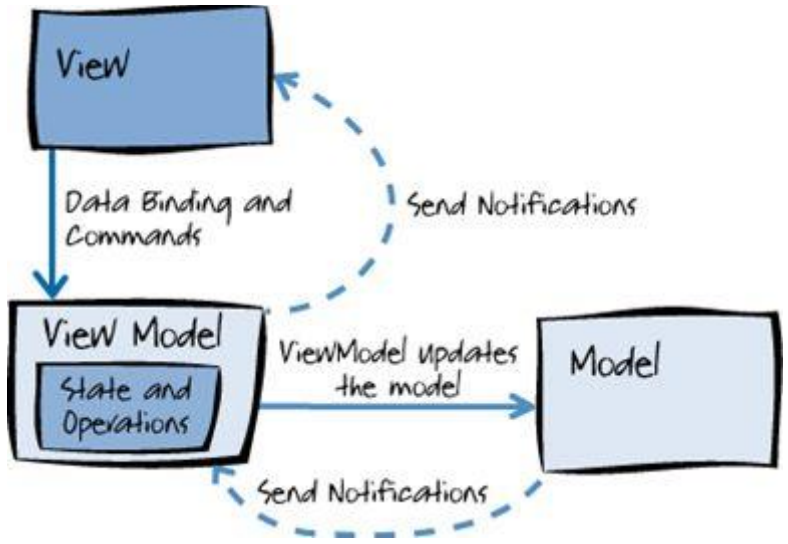
### 2.2.2 Model-View-ViewModel (MVVM)

Το Model-View-ViewModel ή MVVM αποτελεί ένα μοτίβο αρχιτεκτονικής λογισμικού, που έχει ως στόχο το διαχωρισμό της ανάπτυξης του γραφικού περιβάλλοντος χρήστη (GUI) από την ανάπτυξη της λειτουργικότητας (business logic) της εφαρμογής. Συγκεκριμένα, το κομμάτι του viewmodel είναι υπεύθυνο για την μετατροπή του περιεχομένου από το model σε μορφή κατάλληλη για τη διαχείριση και παρουσίασή του, και επιβλέπει ολόκληρη ή σχεδόν ολόκληρη τη λογική παρουσίασης του view. Το viewmodel είναι δυνατόν να υλοποιεί ένα μοτίβο «μεσολαβητή» (mediator pattern), οργανώνοντας έτσι την πρόσβαση στο business logic γύρω από ένα σύνολο σεναρίων χρήσης που υποστηρίζονται από το view.

Πιο αναλυτικά, τα κομμάτια τα οποία αποτελούν το MVVM είναι τα εξής:

- **Το model.** Το model αναφέρεται είτε στο μοντέλο πεδίου, που αναπαριστά τα πραγματικά δεδομένα σε μία αντικειμενοστρεφή προσέγγιση, είτε στο στρώμα που έχει πρόσβαση στα δεδομένα και στο business logic της εφαρμογής.
- **Το view.** Το view αποτελεί τη δομή και την εμφάνιση του περιεχομένου που βλέπει ο χρήστης στην οθόνη.
- **Το viewmodel.** Το viewmodel είναι μία γενικευμένη διεπαφή του view που αποκαλύπτει τις δημόσιες ιδιότητες και εντολές του. Αντί για τον controller του MVC, το MVVM έχει έναν binder. Ο binder μεσολαβεί για να γίνει η επικοινωνία ανάμεσα στο view και στον data binder, που είναι υπεύθυνος για την αντιστοίχιση και τον συγχρονισμό μεταξύ των πηγών των δεδομένων και των καταναλωτών τους.

Σχηματικά, μπορεί να παρασταθεί εύκολα με το ακόλουθο διάγραμμα:




Model-View-ViewModel Diagram

Μία πολύ γνωστή υλοποίηση του MVVM αποτελεί το JavaScript framework Ember.

### 2.2.3 Model-View-Presenter (MVP)

Το Model-View-Presenter ή MVP αποτελεί ένα μοτίβο αρχιτεκτονικής λογισμικού, επίσης εμπνευσμένο από το MVC, το οποίο χρησιμοποιείται κυρίως για την κατασκευή γραφικού περιβάλλοντος χρήστη. Στο MVP, ο presenter παίζει το ρόλο του «διαμεσολαβητή» και όλη η λογική της παρουσίασης των δεδομένων παραχωρείται σε αυτόν.

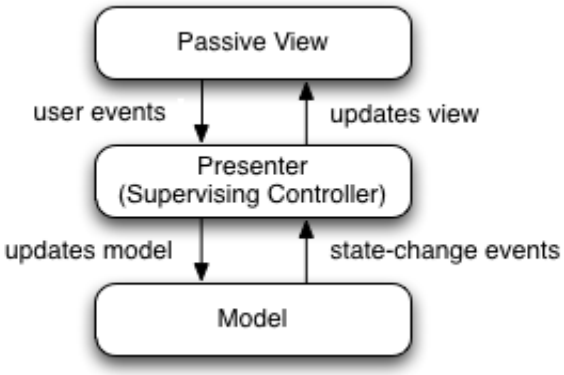
Κύριος στόχος του MVP είναι να διευκολύνει τους αυτοματοποιημένους ελέγχους των επιμέρους μονάδων (automated unit testing) και να βελτιώσει το διαχωρισμό των προβλημάτων (separation of concerns) στη λογική της παρουσίασης.

Τα κομμάτια τα οποία το αποτελούν είναι τα εξής:

- **Το model.** Είναι μία διεπαφή που περιέχει τα δεδομένα προς παρουσίαση, δηλαδή το business logic της εφαρμογής.
- **Το view.** Αποτελεί μία παθητική διεπαφή που απλά παρουσιάζει τα δεδομένα και στέλνει τις εντολές του χρήστη στον presenter μέσω γεγονότων (events) ούτως ώστε ο τελευταίος να πάρει κάποιες αποφάσεις.
- **Ο presenter.** Αλληλεπιδρά με το model και το view. Ανακτά τα δεδομένα από το model και τα επεξεργάζεται κατάλληλα για παρουσίαση στο view.

Αξίζει να αναφερθεί, ότι, ο βαθμός της ελευθερίας του view ποικίλλει μεταξύ διαφορετικών υλοποιήσεων. Στο ένα άκρο, το view είναι εντελώς παθητικό, και προωθεί όλες τις ενέργειες του χρήστη στον presenter, καλώντας μία μέθοδό του χωρίς παραμέτρους και τιμή επιστροφής, αφότου συμβεί κάποιο event. Ο presenter στη συνέχεια ανακτά δεδομένα από το view, επεξεργάζεται τα δεδομένα του model και ανανεώνει το view με τα ενημερωμένα αποτελέσματα. Κάποιες άλλες υλοποιήσεις του MVP αφήνουν μερική ελευθερία στο view, που στην περίπτωση των διαδικτυακών εφαρμογών εκτελείται στον browser του χρήστη, επομένως είναι το πιο κατάλληλο μέρος όσον αφορά το χειρισμό συγκεκριμένων ενεργειών.

Σχηματικά, έχουμε το εξής διάγραμμα:




Model-View-Presenter Diagram

Από τις πιο γνωστές υλοποιήσεις του MVP αποτελεί μέχρι και σήμερα το JavaScript framework Backbone.

### 2.2.4 Model-View-Whatever (MVW)

Το Model-View-Whatever ή MVW, παρόλο που δεν αποτελεί επίσημο μοτίβο αρχιτεκτονικής σχεδίασης, αξίζει να αναφερθεί, διότι είναι ένας όρος που χρησιμοποιείται με σκοπό να υποδηλώσει τη δυνατότητα χρήσης «οτιδήποτε» στη θέση του Controller στο MVC, ή του ViewModel στο MVVM κλπ.

Συγκεκριμένα, το JavaScript framework AngularJS το οποίο υλοποιεί το προαναφερθέν μοτίβο, αφήνει αρκετή ελευθερία στον προγραμματιστή να επιλέξει τι θέλει να χρησιμοποιήσει για την επικοινωνία μεταξύ view και model. Έτσι, ανάλογα την εφαρμογή αλλά και την προσωπική προτίμηση του προγραμματιστή, υπάρχει η δυνατότητα επιλογής ανάμεσα σε MVC, MVVM, MVP κλπ, το οποίο φυσικά είναι αρκετά ευέλικτο και βοηθάει στον ομαλό διαχωρισμό της λογικής της παρουσίασης από το business logic, συμβάλλοντας παράλληλα θετικά στην παραγωγικότητα και στη μακροπρόθεσμη συντήρηση της εφαρμογής.

# Κεφάλαιο 3

# Ανάλυση Απαιτήσεων Συστήματος

## 3.1 Εισαγωγή

Σε αυτό το κεφάλαιο θα δούμε τα επιμέρους κομμάτια που αποτελούν το σύστημά μας, δηλαδή τις τρεις κατηγορίες εφαρμογών για τις οποίες παρέχουμε υλοποιήσεις στη συνέχεια, οι οποίες είναι:

1. Το rendering, η απεικόνιση δηλαδή του περιεχομένου στην οθόνη.
2. Το data binding, δηλαδή η αντιστοίχιση πηγών δεδομένων με τους καταναλωτές τους και ο συγχρονισμός τους.
3. Το operation flow, που αποτελεί ουσιαστικά μία συνδυαστική κατηγορία, καθώς το περιεχόμενο ανανεώνεται και απεικονίζεται στην οθόνη συνεχώς.

Αυτές οι τρεις κατηγορίες σχεδόν πάντα συνυπάρχουν στην περίπτωση μίας σύγχρονης, πραγματικής διαδικτυακής εφαρμογής, όμως, για τους σκοπούς της παρούσης εργασίας θα αποτελέσουν αντικείμενο ξεχωριστής μελέτης.

## 3.2 Κατηγορίες εφαρμογών

Σε αυτό το σημείο αξίζει να σημειωθεί, ότι, η κατηγοριοποίηση των εφαρμογών δεν είναι μοναδική, δηλαδή ο αριθμός των υποκατηγοριών μίας πολύπλοκης εφαρμογής δύναται να είναι πολύ μεγάλος. Παρολ'αυτά θεωρούμε ότι οι τρεις κατηγορίες που θα δούμε αναλυτικά παρακάτω αποτελούν ένα αρκετά αντιπροσωπευτικό δείγμα, και συναντώνται στην πλειοψηφία των διαδικτυακών εφαρμογών.

### 3.2.1 Rendering

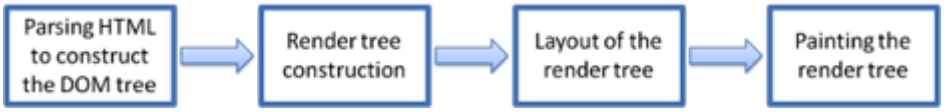
Το rendering αποτελεί μία ευρεία κατηγορία, και ταυτόχρονα τη βασικότερη όλων, όσον αφορά τις διαδικτυακές εφαρμογές. Ουσιαστικά, μιλώντας για το rendering, εννοούμε την απεικόνιση του περιεχομένου στην οθόνη του χρήστη (όπως HTML, XML, αρχεία εικόνων κλπ) μαζί με τους κανόνες που διέπουν τη διαμόρφωση τους μέσα στην εφαρμογή (όπως CSS, XSL κλπ). Αυτό πραγματοποιείται από τη μηχανή του browser (web browser engine ή αλλιώς web rendering engine) στην περίπτωση των διαδικτυακών εφαρμογών, ενώ

μία παρόμοια μηχανή υπάρχει και σε προγράμματα email (email clients), συσκευές ανάγνωσης βιβλίων σε ηλεκτρονική μορφή (e-book readers) κλπ.

Οι μηχανές αυτές είτε περιμένουν να έχουν όλο το περιεχόμενο προτού ξεκινήσουν το rendering, είτε ξεκινούν προτού το έχουν όλο, και καθώς φτάνουν νέα δεδομένα ανανεώνουν την απεικόνιση της σελίδας. Για παράδειγμα, είναι συχνό φαινόμενο εικόνες υψηλής ανάλυσης να καθυστερήσουν να έρθουν, και κατ' επέκταση η μηχανή να τις απεικονίσει τελευταίες, ακόμη και αν οι κανόνες διαμόρφωσης της σελίδας την τοποθετούν ανάμεσα από άλλο περιεχόμενο (πχ κείμενο).

Αναφορικά, ο web browser Google Chrome χρησιμοποιεί τη μηχανή Blink, ο Mozilla Firefox τη μηχανή Gecko και ο Microsoft Internet Explorer τη μηχανή Trident στις παλαιότερες εκδόσεις και την EdgeHTML στις νεότερες εκδόσεις.

Ένα διάγραμμα που δείχνει τα βήματα του rendering είναι το ακόλουθο




Rendering engine steps

Και ως παράδειγμα του πως φαίνεται η δημοφιλής σελίδα κοινωνικής δικτύωσης Facebook χωρίς να έχει ολοκληρωθεί το rendering του περιεχομένου παραθέτουμε την εξής εικόνα


Facebook partially rendered

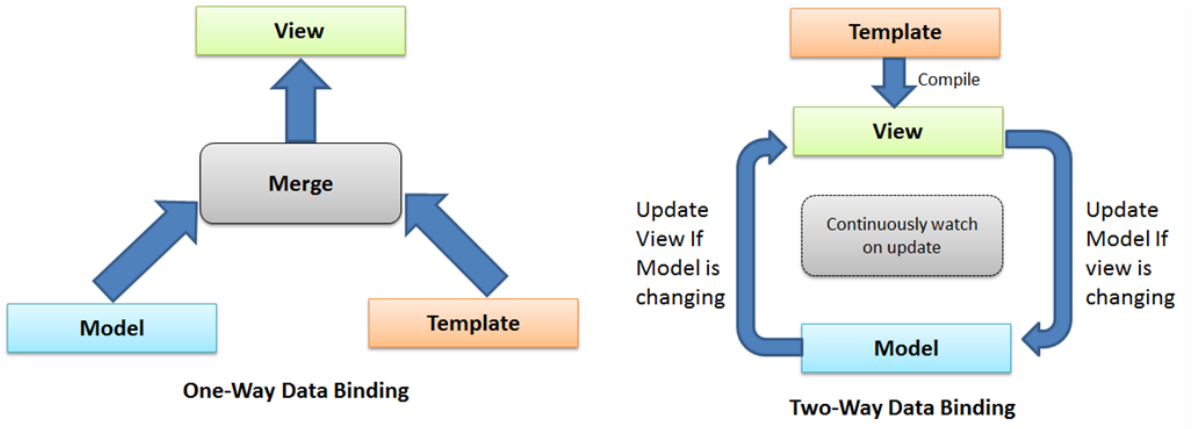
### 3.2.2 Data Binding

Το data binding αποτελεί τη δεύτερη κατηγορία εφαρμογών για την οποία παρέχονται υλοποιήσεις σε επόμενο κεφάλαιο. Πρόκειται για μία γενική τεχνική που έχει στόχο το «δέσιμο», την αντιστοίχιση δηλαδή πηγών δεδομένων μεταξύ του παρόχου και αυτού που τα χρησιμοποιεί, του λεγόμενου καταναλωτή, και κατ' επέκταση των συγχρονισμό μεταξύ τους.

Συγεκριμένα, στο UI data binding που θα μελετήσουμε εμεις, το δέσιμο πραγματοποιείται μεταξύ των αντικειμένων του περιβάλλοντος χρήστη (UI elements ή DOM elements) με τα αντικείμενα του model της εφαρμογής. Κάτι το οποίο επιτυγχάνεται στα περισσότερα frameworks με χρήση του μοτίβου Observer, ως ο μηχανισμός που κρύβεται από κάτω. Φυσικά, για να είναι αποδοτικό, το UI data binding πρέπει να έχει τρόπο να επαληθεύει την είσοδο αλλά και τον τύπο των δεδομένων.

Σε μία τέτοια διαδικασία, είναι απαραίτητο κάθε αλλαγή στα δεδομένα ενός αντικειμένου του model να αντανακλάται απευθείας στα UI elements του view που έχουν δεθεί με αυτό, αλλά και το αντίστροφο, δηλαδή μία αλλαγή στα δεδομένα ενός UI element του view να γίνεται εμφανής αμέσως στο model. Αυτό αποκαλείται αμφίδρομο data binding (two-way data binding) και διαφέρει από το μονόδρομο data binding (one-way data binding) ως προς το ότι στο τελευταίο έχουμε μόνο ανανέωση στο view όταν υπάρξει κάποια αλλαγή στα δεδομένα του model.

Σχηματικά μπορεί να γίνει ακόμη πιο κατανοητό

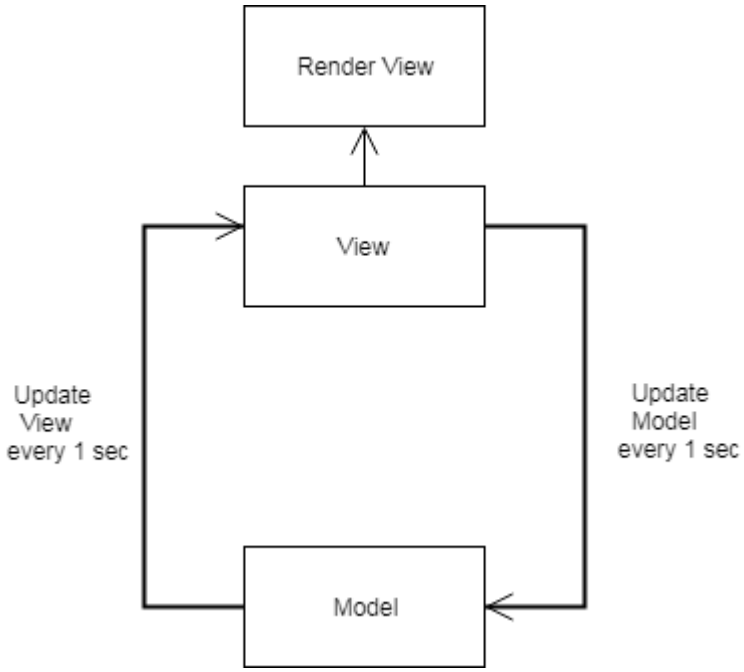



One-way and Two-way Data Binding

### 3.2.3 Operation Flow

Το operation flow αποτελεί την τρίτη και τελευταία κατηγορία εφαρμογών που θα μελετήσουμε, και ταυτόχρονα την πιο σύνθετη από τις τρεις. Ουσιαστικά, πρόκειται για μία συνδυαστική κατηγορία που περιλαμβάνει τις δύο προηγούμενες, το rendering δηλαδή και το data binding, καθώς το περιεχόμενο ανανεώνεται συνεχώς, αλλάζει δηλαδή η καταστασή του και αμέσως πραγματοποιείται ανανέωση της οθόνης, του view που βλέπει ο χρήστης.

Στο διάγραμμα που ακολουθεί φαίνεται ακόμη καλύτερα ότι αποτελεί σύνθεση των δύο προηγούμενων κατηγοριών




Operation Flow Diagram

# Κεφάλαιο 4

# Σχεδίαση Εφαρμογών

## 4.1 Εισαγωγή

Στο παρόν κεφάλαιο θα δούμε λεπτομέρειες σχετικά με την σχεδίαση των εφαρμογών και τα κομμάτια τα οποία αποτελούν την καθεμία. Φυσικά, οι υλοποιήσεις των εφαρμογών διαφέρουν κατά περίπτωση, για παράδειγμα ο τρόπος υλοποίησης του rendering σε Backbone δεν είναι ίδιος με τον τρόπο υλοποίησης του rendering σε AngularJS. Επομένως, πρέπει πρώτα να δούμε κάποια βασικά πράγματα γύρω από τα frameworks που χρησιμοποιήθηκαν, ούτως ώστε να είναι πιο κατανοητός ο τρόπος σχεδίασης των εφαρμογών στη συνέχεια.

## 4.2 Περιγραφή frameworks

Σε προηγούμενο κεφάλαιο είχαμε δει μερικά από τα πιο γνωστά μοτίβα αρχιτεκτονικής λογισμικού που χρησιμοποιούνται ευρέως για τη σχεδίαση εφαρμογών με περιβάλλον αλληλεπίδρασης χρήστη (user interface). Σε αυτή την ενότητα, θα δούμε ποιο από αυτά τα μοτίβα χρησιμοποιεί το καθένα από τα τρία frameworks που μας ενδιαφέρουν, και με τη βοήθεια του οποίου δημιουργήσαμε υλοποιήσεις για τις εφαρμογές στη συνέχεια, αλλά και τα βασικά στοιχεία του καθενός.

### 4.2.1 Backbone

Το Backbone ή αλλιώς Backbone.js είναι ένα JavaScript framework βασισμένο στο μοτίβο Model-View-Presenter (MVP) και χρησιμοποιεί ένα RESTful JSON interface, που σημαίνει ότι συμμορφώνεται με τους αρχιτεκτονικούς κανόνες REST και χρησιμοποιεί το πρότυπο JSON για την αναπαράσταση των δεδομένων. Δημιουργήθηκε στα τέλη του 2010 από τον Jeremy Ashkenas, ο οποίος είναι υπεύθυνος και για τη δημιουργία της γλώσσας προγραμματισμού CoffeeScript αλλά και της βιβλιοθήκης Underscore.

Το Backbone είναι ένα “ελαφρύ” framework, με μικρό μέγεθος, και απαιτεί για τη λειτουργία του μόνο τη βιβλιοθήκη Underscore, και προαιρετικά τη βιβλιοθήκη jQuery για την πλήρη χρήση του (χειρισμό του DOM κλπ). Συγκεκριμένα, για τον χειρισμό του DOM χρησιμοποιεί προστακτικό στυλ προγραμματισμού, ούτως ώστε να δίνει στον προγραμματιστή την ελευθερία να προσαρμόσει ο ίδιος όπως επιθυμεί την πλήρη εμπειρία

χρήσης της διαδικτυακής εφαρμογής του. Σε αυτό συμβάλλει και το γεγονός ότι παρέχει μόνο τα απολύτως απαραίτητα εργαλεία για την επεξεργασία των δεδομένων (models και collections) και του user interface (views και URLs), αφήνοντας στον προγραμματιστή την επιλογή για περαιτέρω λειτουργικότητα όπως πχ εμφωλευμένα views (nested views) και αντιστοίχιση model-view (model-view binding).

Μερικές από τις πιο γνωστές ιστοσελίδες που χρησιμοποιούν το Backbone είναι οι εξής:

- Airbnb
- edX
- Foursquare
- Hulu
- Pinterest
- Reddit

### 4.2.2 AngularJS

Το AngularJS είναι ένα JavaScript framework για την κατασκεύη του user interface διαδικτυακών εφαρμογών, το οποίο υποστηρίζεται από την Google αλλά και από μία μεγάλη κοινότητα ανεξάρτητων προγραμματιστών και εταιρειών. Σε αυτό το σημείο αξίζει να σημειώσουμε, ότι, λέγοντας AngularJS ή απλά Angular από εδώ και στο εξής, αναφερόμαστε στην πρώτη έκδοση του framework, η οποία χρησιμοποιείται ευρέως και σήμερα, αλλά διαφέρει αρκετά από τις εκδόσεις Angular 2 και Angular 2+.

Το Angular χρησιμοποιεί το μοτίβο αρχιτεκτονικής σχεδίασης Model-View-Whatever (MVW), δηλαδή παρέχει στον προγραμματιστή τη δυνατότητα επιλογής είτε controller (MVC) είτε viewmodel (MVVM) κλπ για την επικοινωνία μεταξύ view και model, αναλόγως το σενάριο χρήσης, και έχει ως πρωταρχικό στόχο να αντιμετωπίσει τα προβλήματα που εμφανίζονται κατά την κατασκευή εφαρμογών μίας σελίδας (single-page applications ή SPAs) και να διευκολύνει τη δημιουργία τους αλλά και τον έλεγχό τους (testing).

Αρχικά, το framework διαβάζει όλη την HTML σελίδα, η οποία έχει ενσωματωμένους επιπλέον, μη τυπικούς όρους μέσα στα tags (custom tag attributes), τα οποία το Angular τα ερμηνεύει ως «εντολές» (directives) για να αντιστοιχίσει κομμάτια της σελίδας με ένα model που αναπαριστάται από JavaScript μεταβλητές. Οι τιμές αυτών των μεταβλητών μπορούν να τεθούν είτε χειροκίνητα μέσω του κώδικα είτε να ανακτηθούν από στατικές και δυναμικές πηγές JSON.

Το Angular είναι βασισμένο στην αξία ότι ο δηλωτικός προγραμματισμός (declarative programming) είναι το πιο κατάλληλο στυλ προγραμματισμού για τη δημιουργία user interfaces και τη σύνδεση των διαφόρων κομματιών του λογισμικού, ενώ ο προστατικός προγραμματισμός (imperative programming) ευνοεί τον ορισμό του business logic της εφαρμογής. Το framework προσαρμόζει και επεκτείνει τον κλασσικό κώδικα HTML για να παρουσιάσει δυναμικό περιεχόμενο μέσω της τεχνικής του two-way data

binding που επιτρέπει τον αυτόματο συγχρονισμό των models και των views. Ως αποτέλεσμα, δίνεται λιγότερη έμφαση στον ρητό χειρισμό του DOM με σκοπό τη βελτίωση της απόδοσης και της ικανότητας ελέγχου του παραγόμενου αποτελέσματος (testability).

Οι σχεδιαστικοί στόχοι του Angular περιλαμβάνουν τα εξής:

- Την παροχή μίας δομής για τη δημιουργία της εφαρμογής, από το σχεδιασμό του user interface μέχρι τον καθορισμό του business logic και τον έλεγχο.
- Την αποσύνδεση του χειρισμού του DOM από τη λειτουργικότητα της εφαρμογής, η δυσκολία του οποίου μεταβάλλεται εκθετικά ανάλογα με το πως είναι δομημένος ο κώδικας.
- Την αποσύνδεση της μεριάς του χρήστη (client side) από τη μεριά του server της εφαρμογής, κάτι το οποίο ευνοεί την παράλληλη ανάπτυξη της εφαρμογής και την επαναχρησιμοποίηση του κώδικα.

Χρησιμοποιώντας μία τεχνική που ονομάζεται dependency injection, στην οποία ένα αντικείμενο παρέχει τις εξαρτήσεις, δηλαδή τις υπηρεσίες που είναι απαραίτητες για τη λειτουργία ενός άλλου αντικειμένου, το Angular καταφέρνει να φέρει κλασσικές υπηρεσίες του server, όπως για παράδειγμα controllers που είναι εξαρτώμενοι από views, στη μεριά του χρήστη μίας διαδικτυακής εφαρμογής. Συνεπώς, με αυτόν τον τρόπο μειώνεται δραστικά ο υπολογιστικός φόρτος του server.

Μερικές από τις πιο γνωστές ιστοσελίδες που κάνουν χρήση του Angular είναι οι εξής:

- Wolfram Alpha
- NBC
- Intel
- Sprint
- Forbes
- Udemy

Ας περάσουμε όμως να δούμε εν συντομία πως δουλεύει το Angular σε πιο πρακτικό επίπεδο. Κατά κύριο λόγο, χρησιμοποιεί έναν ειδικό τύπο αντικειμένου, το οποίο ονομάζεται scope, και αντιστοιχεί σε ένα model. Οι μεταβλητές που ορίζονται στο scope είναι προσβάσιμες και από το view και από τον controller, και θα μπορούσαμε να πούμε ότι το scope συμπεριφέρεται ως κόλλα και δένει το view με τον controller. Ουσιαστικά, η έννοια του scope στο Angular δεν απέχει πολύ από τον θεμελιώδη ορισμό του scope στην επιστήμη των υπολογιστών, όπου μεταφράζεται ως το κομμάτι του προγράμματος που ένας συγκεκριμένος ορισμός μεταβλητής βρίσκεται σε ισχύ.

Ένα ειδικό εργαλείο, που ονομάζεται bootstrapper είναι υπεύθυνο για τις εξής εργασίες, αφότου δημιουργηθεί το DOM:

1. Τη δημιουργία ενός νέου αντικειμένου injector, που είναι αυτό που οργανώνει την ανάκτηση των διαφόρων υπηρεσίων (services) αλλά και το dependency injection γενικότερα.
2. Τη μεταγλώττιση των directives που προσθέτουν επιπλέον λειτουργικότητα στο DOM.
3. Τη σύνδεση όλων των directives με το scope.

Τα πιο συχνά χρησιμοποιούμενα directives, τα περισσότερα εκ των οποίων θα τα συναντήσουμε και στη συνέχεια, είναι:

- **ng-app**
  Υποδηλώνει τη ρίζα της Angular εφαρμογής, κάτω από την οποία τα directives μπορούν να χρησιμοποιηθούν για να δηλώσουν δεσίματα μεταβλητών και συμπεριφοράς.
- **ng-bind**
  Θέτει το κείμενο ενός DOM element ως την αποτίμηση μίας έκφρασης. Οποιαδήποτε αλλαγή στην τιμή της έκφρασης αμέσως αντανακλάται στο DOM.
- **ng-model**
  Ίδια χρήση με το **ng-bind**, με τη διαφορά ότι χρησιμοποιεί two-way data binding μεταξύ view και scope.
- **ng-class**
  Εφαρμόζει μία κλάση, ανάλογα αν η τιμή αληθείας μίας λογικής έκφρασης (boolean) είναι αληθής ή ψευδής.
- **ng-controller**
  Υποδηλώνει έναν JavaScript controller που αποτιμά HTML εκφράσεις.
- **ng-repeat**
  Δημιουργεί ένα DOM element για κάθε ένα αντικείμενο μίας συλλογής.
- **ng-show & ng-hide**
  Αναλόγως αν η τιμή μίας λογικής έκφρασης είναι αληθής ή ψευδής εμφανίζει ή κρύβει το συγκεκριμένο DOM element.
- **ng-view**
  Το βασικό directive για το χειρισμό των routes της εφαρμογής που επιστρέφουν δεδομένα σε μορφή JSON.
- **ng-if**
  Εάν η if - συνθήκη είναι αληθής τότε το DOM element δημιουργείται, διαφορετικά καταστρέφεται.

Σε αυτό το σημείο, αξίζει να τονίσουμε ακόμη μία φορά τη σημασία του two-way data binding, ίσως του πιο σημαντικού χαρακτηριστικού του Angular. Και αυτό, διότι, οι server απαλάσσονται σε μεγάλο βαθμό από τα καθήκοντα περί templating, δηλαδή τα templates παρουσιάζονται σε απλή HTML σύμφωνα με τα δεδομένα που περιέχονται στο scope που αντιστοιχεί στο model. Το scope εντοπίζει τις αλλαγές στο model και μεταβάλλει τις HTML εκφράσεις στο view μέσω ενός controller. Αντίστοιχα, οποιεσδήποτε αλλαγές στο view αμέσως αντανακλώνται στο model. Αυτό ουσιαστικά παρακάμπτει την ανάγκη

του ενεργού χειρισμού του DOM και ενθαρρύνει την ταχεία δημιουργία των διαδικτυακών εφαρμογών. Το Angular μπορεί και εντοπίζει αυτές τις αλλαγές στο model συγκρίνοντας τις τρέχουσες τιμές με αυτές που ήταν αποθηκευμένες νωρίτερα, με μία διαδικασία που ονομάζεται dirty-checking, εν αντιθέσει με το Ember.js και το Backbone.js που ενεργοποιούν listeners κάθε φορά που μία τιμή στο model μεταβάλλεται.

Τέλος, πρέπει να αναφερθεί πως το Angular είναι αυτό που όρισε πρώτο το πρότυπο του digest cycle, ενός κύκλου δηλαδή κατα τη διάρκεια του οποίου ελέγχει εάν κάποια τιμή μίας μεταβλητής που επιβλέπει κάποιο scope έχει αλλάξει. Αυτή η προσέγγιση δυνητικά μπορεί να οδηγήσει σε αργό rendering, σε περίπτωση που ελέγχονται πάρα πολλές μεταβλητές σε κάθε κύκλο.

### 4.2.3 Ember

Το Ember ή αλλιώς Ember.js είναι ένα JavaScript framework ανοιχτού κώδικα βασισμένο στο μοτίβο Model-View-ViewModel (MVVM), το οποίο επιτρέπει στους προγραμματιστές να δημιουργήσουν διαδικτυακές εφαρμογές μίας σελίδας (single-page applications ή SPAs), ενσωματώνοντας κοινά προγραμματιστικά ιδιώματα και τις καλύτερες πρακτικές μέσα στο ίδιο το framework.

Από την αρχή, ο σχεδιασμός του Ember περιστράφηκε γύρω από τις εξής σημαντικές ιδεές:

i. Να αποτελέσει τη βάση για «φιλόδοξες» διαδικτυακές εφαρμογές. Το Ember έχει ως στόχο να παρέχει μία ολοκληρωμένη λύση για το κομμάτι της μεριάς του χρήστη (client-side) των εφαρμογών. Αυτό έρχεται σε αντίθεση με άλλα JavaScript frameworks που ξεκινούν με το να παρέχουν μία λύση για το view, και προσπαθούν έπειτα να επεκταθούν από εκεί.
ii. Να είναι έτοιμο για χρήση. Το Ember αποτελεί ένα συστατικό σε ένα σύνολο εργαλείων που συνεργάζονται για να παρέχουν μία ολοκληρωμένη στοίβα παραγωγής (development stack). Ο στόχος αυτών των εργαλείων είναι να βοηθήσουν τον προγραμματιστή να ξεκινήσει να παράγει αμέσως.
iii. Να έχει σταθερότητα αλλά όχι στασιμότητα. Αυτό πρακτικά σημαίνει ότι η συμβατότητα προς τα πίσω (backwards compatibility) είναι σημαντική και πρέπει να διατηρηθεί ενώ το framework εξελίσσεται.
iv. Να προνοεί για τα μελλοντικά πρότυπα του διαδικτύου. Το Ember είναι ένας από τους προπομπούς γύρω από πολλά πρότυπα της JavaScript και του διαδικτύου, όπως promises, web components και ES6.

Ας περάσουμε όμως να δούμε τις πέντε βασικές έννοιες που απαρτίζουν το Ember:

1. **Routes.** Στο Ember, η κατάσταση της εφαρμογής αναπαριστάται από ένα URL. Κάθε URL έχει ένα αντίστοιχο αντικείμενο δρομολόγησης (route object) που ελέγχει το τι βλέπει ο χρήστης.

2. **Models.** Κάθε route έχει ένα συσχετισμένο με αυτό model, που περιέχει τα δεδομένα που έχουν σχέση με την εκάστοτε κατάσταση της εφαρμογής. Υπάρχει η δυνατότητα ο προγραμματιστής να χρησιμοποιήσει τη βιβλιοθήκη jQuery ή κάποια αντίστοιχη για να φορτώσει αντικείμενα JSON από το server και να χρησιμοποιήσει αυτά ως models. Παρολ' αυτά, οι περισσότερες εφαρμογές κάνουν χρήση μίας βιβλιοθήκης model όπως της Ember Data για να χειριστούν αυτή την περίπτωση.
3. **Templates.** Τα templates χρησιμοποιούνται για τη δημιουργία της HTML της εφαρμογής και κατασκευάζονται με τη βοήθεια της templating γλώσσας HTMLBars. (Η HTMLBars είναι μία παραλλαγή της Handlebars που δημιουργεί στοιχεία DOM αντί για strings.)
4. **Components**. Ένα component είναι ένα μη τυπικό HTML tag (custom HTML tag). Η λειτουργικότητα του component υλοποιείται με χρήση της JavaScript και η παρουσία του ορίζεται μέσω των HTMLBars templates. Τα components ουσιαστικά είναι οι κυρίαρχοι των δεδομένων που περιέχουν. Επιπλέον, είναι δυνατό να είναι εμφωλευμένα και να επικοινωνούν με τα εξωτερικά components μέσω ενεργειών (events). Διάφορες component βιβλιοθήκες όπως το Polymer γίνεται να χρησιμοποιηθούν μαζί με το Ember.
5. **Services**. Τα services είναι μοναδικά αντικείμενα (singleton) που περιέχουν δεδομένα για συνεχόμενη χρήση όπως οι συνεδρίες χρήστη (user sessions).

Το Ember.js είναι απλά ένα κομμάτι ενός ολοκληρωμένου front end stack, κατασκευασμένου και υποστηριζόμενου από την κεντρική ομάδα του Ember. Επιγραμματικά, αναφέρουμε τα υπόλοιπα κομμάτια:

- **Ember CLI.** Το Ember CLI στοχεύει να πετύχει τον κανόνα της σύμβασης πάνω από την παραμετροποίηση (convention over configuration), δηλαδή να μειώσει τον αριθμό των αποφάσεων που έχει να πάρει ένας προγραμματιστής που χρησιμοποιεί το framework χωρίς ωστόσο να χάνει σε ευελιξία. Πρόκειται για ένα εργαλείο της γραμμής εντολών (command line), με το οποίο δημιουργείται μια νέα εφαρμογή Ember με το προεπιλεγμένο stack, τρέχοντας την εντολή 'ember new <app-name>'.
- **Ember Data.** Βιβλιοθήκη που ασχολείται με τα models και τα δεδομένα της εφαρμογής. Αντιστοιχίζει client-side models με server-side δεδομένα, και στη συνέχεια μπορεί να φορτώνει και να αποθηκεύει εγγραφές και τις εξαρτήσεις μεταξύ τους μέσω ενός RESTful JSON API.
- **Ember Inspector.** Πρόκειται για ένα πρόσθετο (extension) διαθέσιμο για τους browsers Mozilla Firefox και Google Chrome που διευκολύνει την εύρεση σφαλμάτων (debugging) σε εφαρμογές Ember.
- **Fastboot.** Ένα πρόσθετο για το Ember CLI που δίνει τη δυνατότητα στους προγραμματιστές να τρέχουν τις εφαρμογές τους στην πλατφόρμα Node.js.
- **Liquid Fire.** Παρέχει υποστήριξη για τις απεικονίσεις (animations) μίας εφαρμογής Ember.

Μερικές από τις πιο γνωστές ιστοσελίδες που κάνουν χρήση του Ember είναι οι εξής:

- Discourse
- Groupon
- LinkedIn
- Vine
- Twitch.tv
- Yahoo

# Κεφάλαιο 5

# Κώδικας Εφαρμογών και Σχολιασμός

## 5.1 Λεπτομέρειες υλοποίησης

Στην γενική περίπτωση, κάθε υλοποίηση αποτελείται από ένα αρχείο με κατάληξη .html με ενσωματωμένο τον κώδικα JavaScript, εκτός από όπου απαιτείται να βρίσκεται σε ξεχωριστό αρχείο με κατάληξη .js λόγω μεγέθους, ούτως ώστε να είναι ευανάγνωστο το τελικό αποτέλεσμα.

### 5.1.1 Υλοποίηση σε Backbone

#### 5.1.1.1 Rendering

Για την υλοποίηση του Rendering σε Backbone χρησιμοποιήθηκε η βιβλιοθήκη jQuery και η βιβλιοθήκη Underscore, ως εξωτερικές απαιτήσεις (external dependencies). Τα σημεία που χρήζουν σχολιασμού στον κώδικα είναι τα εξής:

- Στο αρχείο backbone-rendering.html, το κομμάτι

```
<ul id="items">
  <script type="x-template" id="underscore-template">
    <li>
      <%= number %>
    </li>
  </script>
</ul>
```

είναι το σημείο όπου γίνεται το rendering των δεδομένων στην οθόνη σε html, εν προκειμένω των αντικειμένων που περιέχουν αριθμούς.

- Στο αρχείο backbone-rendering.js, το κομμάτι

```
const backboneRender = () => {


  const t0 = performance.now();


  items = _.map(_.range(10000), i => {
    const item = new Item({ number: i });
    const view = new ItemView({ model: item });


    grid.appendChild(view.render().el);
    return item;
  });


  const t1 = performance.now();
  console.log(`Call to render() took ${t1 - t0} milliseconds.`);
};
```

είναι το σημείο όπου γίνεται η κατασκευή του πίνακα των αντικειμένων, το rendering τους, η μέτρηση του χρόνου γι' αυτή τη διαδικασία και η εκτύπωση του αποτελέσματος στην κονσόλα του browser.

#### 5.1.1.2 Data Binding

Για την υλοποίηση του Data Binding σε Backbone και πάλι χρησιμοποίηθηκαν οι βιβλιοθήκες jQuery και Underscore. Τα σημεία του κώδικα που αξίζει να αναλυθούν είναι τα εξής:

- Στο αρχείο backbone-data-binding.html, το κομμάτι

```
<ul id="items">
  <script type="x-template" id="underscore-template">
    <li>
      <%= number %>  <input type="text" />
    </li>
  </script>
</ul>
```

όπου πλέον για κάθε ένα στοιχείο των δεδομένων έχουμε προσθέσει ένα πεδίο κειμένου, το οποίο θα μας βοηθήσει οπτικά να δούμε τις διαφορές κατά τη διάρκεια των πειραμάτων στη συνέχεια.

- Στο αρχείο backbone-data-binding.js, το κομμάτι

```
const ItemView = Backbone.View.extend({

  template: _.template(document.querySelector('#underscore-template').innerHTML),

  events: {
   "change input": "contentChanged"
  },

  initialize() {
   _.bindAll(this, "contentChanged");
  },

  render() {
   this.el.innerHTML = this.template(this.model.attributes);
   return this;
  },

  contentChanged(evt) {
   const inp = evt.currentTarget.value;
   this.model.set({ content: inp });
  }

});
```

όπου επεκτείνουμε το προϋπάρχον Backbone.View ούτως ώστε να χειριστούμε το change event που ανακύπτει κάθε φορά που κάποιο πεδίο κειμένου αλλάζει τιμή. Αρχικά, «δένουμε» το κάθε στοιχείο με τη μέθοδο contentChanged(), κατόπιν με το που υπάρξει κάποια αλλαγή στην τιμή ενός πεδίου κειμένου συμβαίνει ένα changed event, το οποίο στη συνέχεια το χειριζόμαστε με την αποθήκευση της νέας τιμής στο εκάστοτε model.

#### 5.1.1.3 Operation Flow

Για την υλοποίηση του Operation Flow σε Backbone έγινε ξανά χρήση των βιβλιοθηκών jQuery και Underscore. Τα σημεία που παρουσιάζουν ιδιαίτερο ενδιαφέρον είναι τα εξής:

- Στο αρχείο backbone-operation-flow.html, το κομμάτι

```
<script type="x-template" id="underscore-template">
  <div class="box" id="box-<%= number %>" style="top: <%= top %>px; left: <%= left %>px; background: rgb(0,0,<%= color %>);">
    <%= content %>
  </div>
</script>
```

όπου έγινε χρήση ενός underscore template για την παρουσίαση των δεδομένων σε html.

- Στο αρχείο backbone-operation-flow.js, το κομμάτι

```
const backboneInit = () => {


  boxes = _.map(_.range(N), i => {
    const box = new Box({ number: i });
    const view = new BoxView({ model: box });


    const grid = document.querySelector('#grid');
    grid.appendChild(view.render().el);
    return box;
  });
 };


 const backboneAnimate = () => {


  if (counter == 0) {
    t0 = performance.now();
  }
  counter += 1;
  if (counter == 100) {
    t1 = performance.now();
    console.log(`Time: ${t1-t0} ms`);
  }
  for (let i = 0, l = boxes.length; i < l; i++) {
    boxes[i].tick();
  }
  window.timeout = _.defer(backboneAnimate);
};
```

όπου αρχικά κατασκευάζουμε τα δεδομένα, στη συνέχεια τα προσθέτουμε στο πλέγμα (grid) και κατόπιν ξεκινά η συνεχής ανανέωση και απεικόνισή τους (animation). Στην αρχή και στο τέλος του πρώτου κύκλου μετράμε τον χρόνο και εκτυπώνουμε στην κονσόλα του browser τη συνολική διάρκεια του κύκλου.

### 5.1.2 Υλοποίηση σε AngularJS

#### 5.1.2.1 Rendering

Για την υλοποίηση του Rendering σε AngularJS δε χρειάστηκε κάποιο external dependency. Τα σημεία του κώδικα που αξίζει να σημειώσουμε είναι τα εξής:

- Το κομμάτι

```
<ul>
    <li ng-repeat="item in data">{{item.number}}</li>
</ul>
```

αποτελεί το σημείο όπου γίνεται το rendering των δεδομένων στην οθόνη σε html, μέσω της εντολής (directive) του AngularJS ng-repeat, η οποία προσπελαύνει με τη σειρά τα αντικέιμενα του πίνακα data και εκτυπώνει το περιεχόμενό τους.

- Το κομμάτι

```
$scope.render = () => {

    const t0 = performance.now();

    for (let i=0; i<10000; i++){
      $scope.data.push({ number: i });
    }

    const t1 = performance.now();
    console.log(`Call to render() took ${t1 - t0} milliseconds.`);
};
```

αποτελεί το σημείο όπου κατασκευάζονται τα αντικείμενα, γίνονται rendered, μετράται ο χρόνος γι' αυτή τη διαδικασία και εκτυπώνεται το αποτέλεσμα στην κονσόλα του browser.

### 5.1.2.2 Data Binding

Για την υλοποίηση του Data Binding στο AngularJS αυτό που χρειάστηκε να επεκτείνουμε είναι το κομμάτι

```
<ul>
    <li ng-repeat="item in data">
      {{item.number}}
      <input ng-model="item.text" />
    </li>
</ul>
```

ώστε να κατασκευάσουμε ένα πεδίο κειμένου για κάθε ένα στοιχείο των δεδομένων. Με το ng-model directive, το ίδιο το AngularJS φροντίζει για την αντιστοίχιση μεταξύ του

συγκεκριμένου πεδίου με το στοιχείο των δεδομένων, και είναι υπεύθυνο ώστε να είναι και τα δύο ενημερωμένα.

### 5.1.2.3 Operation Flow

Για την υλοποίηση του Operation Flow σε AngularJS χρησιμοποιήθηκε η βιβλιοθήκη Underscore. Τα πιο βασικά σημεία είναι τα εξής:

- Στο αρχείο angular-operation-flow.html, το κομμάτι

```
<div id="grid">
    <div class="box-view" ng-repeat="box in boxes">
      <div class="box" ng-style="{top: box.top+'px', left: box.left+'px', background:
'rgb(0,0,'+box.color+')'}">
        {{box.content}}
      </div>
    </div>
</div>
```

όπου χρησιμοποιώντας τα AngularJS directives ng-repeat και ng-style εμφανίζουμε στην οθόνη όλα τα δεδομένα με τα επιθυμητά χαρακτηριστικά.

- Στο αρχείο angular-operation-flow.js, το κομμάτι

```
$scope.angularjsInit = () => {
  for (let i = 0; i < N; i++) {
    $scope.boxes[i] = new $scope.Box(i);
  }
 }

 $scope.angularjsAnimate = () => {
  if (counter == 0) {
    t0 = performance.now();
  }
  counter += 1;
  if (counter == 100) {
    t1 = performance.now();
    console.log(`Time: ${t1-t0} ms`);
  }
  for (let i = 0; i < N; i++) {
    $scope.boxes[i].tick();
  }
  $rootScope.$apply();
  $window.timeout = _.defer($scope.angularjsAnimate);
}
```

όπου αφού κατασκευάσουμε τα δεδομένα και τα προσθέσουμε στο grid με τη βοήθεια της ειδικής μεταβλητής $scope του AngularJS, στη συνέχεια ξεκινά η συνεχής ανανέωση και απεικόνισή τους, μαζί με τη μέτρηση της χρονικής διάρκειας του πρώτου κύκλου, η οποία εκτυπώνεται στην κονσόλα του browser.

### 5.1.3 Υλοποίηση σε Ember

#### 5.1.3.1 Rendering

Για την υλοποίηση του Rendering σε Ember χρησιμοποιήθηκε η βιβλιοθήκη jQuery, καθώς και η template engine Handlebars.js. Τα σημεία που παρουσιάζουν το μεγαλύτερο ενδιαφέρον στον κώδικα είναι:

- Το κομμάτι

```
<ul>
    {{#each item in controller}}
    <li>{{item.number}}</li>
    {{/each}}
</ul>
```

όπου κάνουμε χρήση της Handlebars.js για να γίνει το rendering των δεδομένων στην οθόνη σε html.

- Το κομμάτι

```
App.ApplicationController = Ember.ArrayController.extend({
  actions:{
    render() {

      const t0 = performance.now();

      for (let i=0; i<10000; i++){
        this.get('model').pushObject({ number: i });
      }

      const t1 = performance.now();
      console.log(`Call to render() took ${t1 - t0} milliseconds.`);
    }
  }
});
```

όπου κατασκευάζονται τα αντικείμενα, γίνονται rendered, καταγράφεται ο συνολικός χρόνος της διαδικασίας και τέλος εκτυπώνεται το αποτέλεσμα στην κονσόλα του browser.

#### 5.1.3.2 Data Binding

Για το Data Binding σε Ember το σημείο του κώδικα που φέρει αλλαγές είναι το

```
<ul>
    {{#each item in controller}}
    <li>
     {{item.number}}
     {{input value=item.text}}
    </li>
    {{/each}}
</ul>
```

όπου δημιουργούμε ένα πεδίο κειμένου για κάθε στοιχείο των δεδομένων και αντιστοιχίζουμε την τιμή του με την ιδιότητα (property) text του στοιχείου.

#### 5.1.3.3 Operation Flow

Για την υλοποίηση του Operation Flow σε Ember χρησιμοποιήθηκαν οι βιβλιοθήκες jQuery και Underscore, και η template engine Handlebars.js. Τα σημεία που χρήζουν επιπλέον σχολιασμού είναι τα εξής:

- Στο αρχείο ember-operation-flow.html, το κομμάτι

```
<script type="text/x-handlebars" id="handlebars-template" data-template-name="box">
   <div class="box" {{bindAttr id="model.number" style="model.style"}}>
     {{ model.content }}
   </div>
</script>
```

όπου χρησιμοποιούμε αυτό το handlebars template για να εμφανίσουμε τα δεδομένα στην οθόνη σε html.

- Στο αρχείο ember-operation-flow.js, το κομμάτι

```
const emberInit = () => {
  boxes = _.map(_.range(N), i => {
    const box = Box.create();
    const view = BoxView.create({ model: box });
    view.appendTo('#grid');
    box.set('number', i);
    return box;
  });
 };

 const emberAnimate = () => {
  if (counter == 0) {
    t0 = performance.now();
  }
  counter += 1;
  if (counter == 100) {
    t1 = performance.now();
    console.log(`Time: ${t1-t0} ms`);
  }

  for (let i = 0, l = boxes.length; i < l; i++) {
    boxes[i].tick();
  }
  window.timeout = _.defer(emberAnimate);
};
```

όπου αρχικά κατασκευάζουμε τα δεδομένα, στη συνέχεια τα προσθέτουμε στο grid και κατόπιν ξεκινά η συνεχής ανανέωση και απεικόνισή τους, ενώ παράλληλα μετράμε τη

συνολική διάρκεια του πρώτου κύκλου, η οποία εκτυπώνεται στην κονσόλα του browser.

Ο ολοκληρωμένος κώδικας των παραπάνω υλοποιήσεων μπορεί να βρεθεί στο σύνδεσμο Code.

## 5.2 Πλατφόρμες και προγραμματιστικά εργαλεία

Όλες οι ανωτέρω υλοποιήσεις έχουν κατασκευαστεί με χρήση του προγράμματος επεξεργασίας κώδικα (code editor) Brackets και είναι δοκιμασμένο ότι εκτελούνται επιτυχώς στις τελευταίες εκδόσεις των πιο δημοφιλών browsers. Συγκεκριμένα, τη στιγμή συγγραφής αυτού του κειμένου, αυτές είναι:

- Google Chrome 61
- Mozilla Firefox 55
- Opera 47
- Microsoft Edge 40

Δεν υπάρχουν ιδιαίτερες απαιτήσεις σε υλικό (hardware), ένας απλός desktop υπολογιστής ή ένα laptop με εγκατεστημένο κάποιον από τους προαναφερθέντες browsers (πιθανώς να υποστήριζονται και παλαιότερες εκδόσεις) αρκεί για την επιτυχή εκτέλεση όλων των κομματιών κώδικα και την παρατήρηση των αποτελεσμάτων τους. Τέλος, αξίζει να αναφερθεί, ότι είναι πολύ πιθανό κάποιες ή και όλες από τις υλοποιήσεις να τρέχουν δίχως πρόβλημα σε κινητά τηλέφωνα smartphone ή tablet (πχ με την εφαρμογή Google Chrome for Android) χωρίς ωστόσο να έχουν πραγματοποιηθεί επιβεβαιωμένες δοκιμές σε αυτά.

# Κεφάλαιο 6

# Καταγραφή Μετρήσεων

## 6.1 Μεθοδολογία ελέγχου

Η αξιολόγηση όλων των ανωτέρω υλοποιήσεων έγινε σύμφωνα με τα παρακάτω κριτήρια:

1. Το χρόνο εκτέλεσης του κρίσιμου τμήματος του κώδικα (execution time).
2. Την κατανάλωση μνήμης (memory allocation).
3. Το συνολικό χρόνο χρησιμοποίησης του επεξεργαστή (CPU time).

Όσον αφορά το χρόνο εκτέλεσης, για όλες τις κατηγορίες εφαρμογών, τον μετράμε προγραμματιστικά μέσω του κώδικα, δηλαδή καλώντας το performance.now() API ακριβώς πριν την έναρξη και αμέσως μετά τη λήξη του επίμαχου τμήματος, παίρνοντας τη διαφορά τους και εκτυπώνοντας το τελικό αποτέλεσμα στην κονσόλα του browser. Για το πείραμα αυτό, εκτελούμε δέκα διαφορετικές επαναλήψεις της κάθε υλοποίησης, λαμβάνοντας τον κάθε χρόνο και στο τέλος υπολογίζουμε το μέσο όρο τους, τον οποίο και χρησιμοποιούμε ως αντιπροσωπευτικό αποτέλεσμα για την εκάστοτε υλοποίηση.

Για το δεύτερο κριτήριο, την κατανάλωση μνήμης της κάθε υλοποίησης, αυτή την υπολογίζουμε μέσω των εργαλείων του browser (dev tools). Αρχικά, παίρνουμε ένα στιγμιότυπο του heap (heap snapshot) προτού εκτελέσουμε των κώδικα και βλέπουμε πόσα MB μνήμης είναι δεσμευμένα. Στη συνέχεια, εκτελούμε των κώδικα, και μόλις ολοκληρωθεί η εκτέλεσή του παίρνουμε άλλο ένα heap snapshot και καταγράφουμε και πάλι το πόσα MB μνήμης είναι δεσμευμένα. Τέλος, υπολογίζουμε τη διαφορά μεταξύ της τελικής και της αρχικής τιμής, και η τιμή αυτή αποτελεί την ποσότητα μνήμης που καταναλώνει το συγκεκριμένο κομμάτι κώδικα.

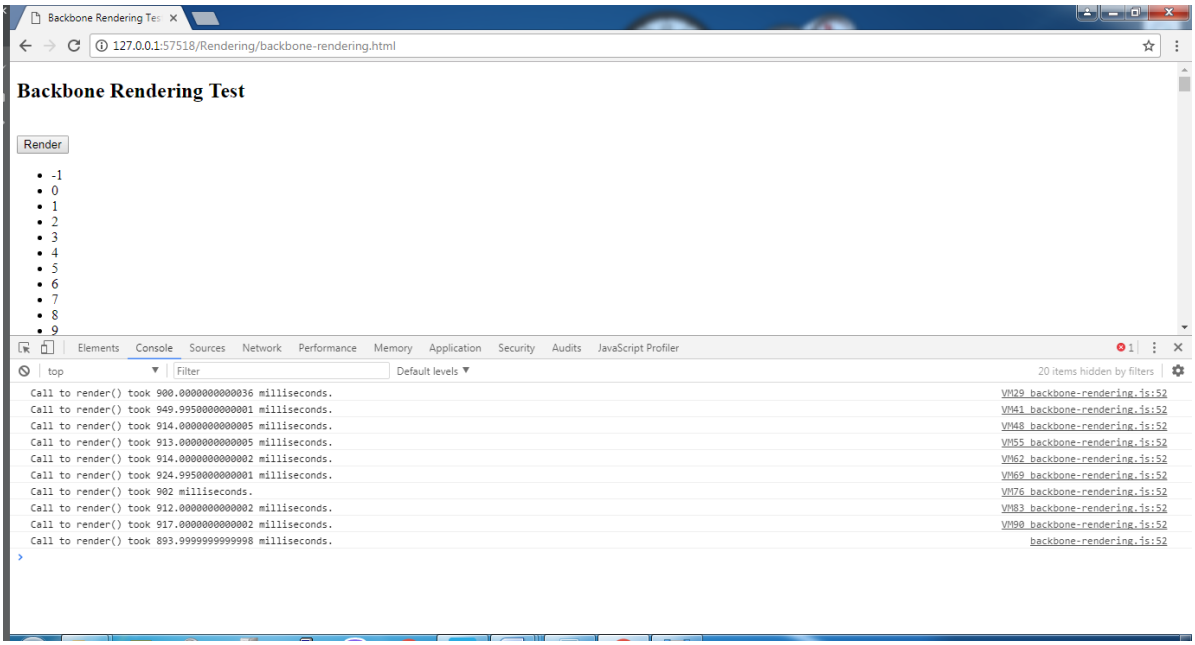
Για το τρίτο και τελευταίο κριτήριο, του συνολικού χρόνου χρησιμοποίησης του επεξεργαστή, η μέτρηση έγινε και πάλι με τη βοήθεια των dev tools του browser. Αρχικά, ξεκινάμε την εγγραφή του προφίλ μέσω της καρτέλας performance, έπειτα εκτελούμε τον κώδικα, και μόλις ολοκληρωθεί η εκτέλεση σταματάμε την εγγραφή. Τελικώς, παρατηρούμε το διάγραμμα του επεξεργαστή (CPU chart) που έχει δημιουργηθεί και τους επιμέρους χρόνους. Η τιμή στο κέντρο του διαγράμματος μείον την τιμή του πεδίου Idle είναι και η τιμή του συνολικού χρόνου χρησιμοποίησης του επεξεργαστή για τη συγκεκριμένη υλοποίηση.

## 6.2 Αναλυτική παρουσίαση ελέγχου

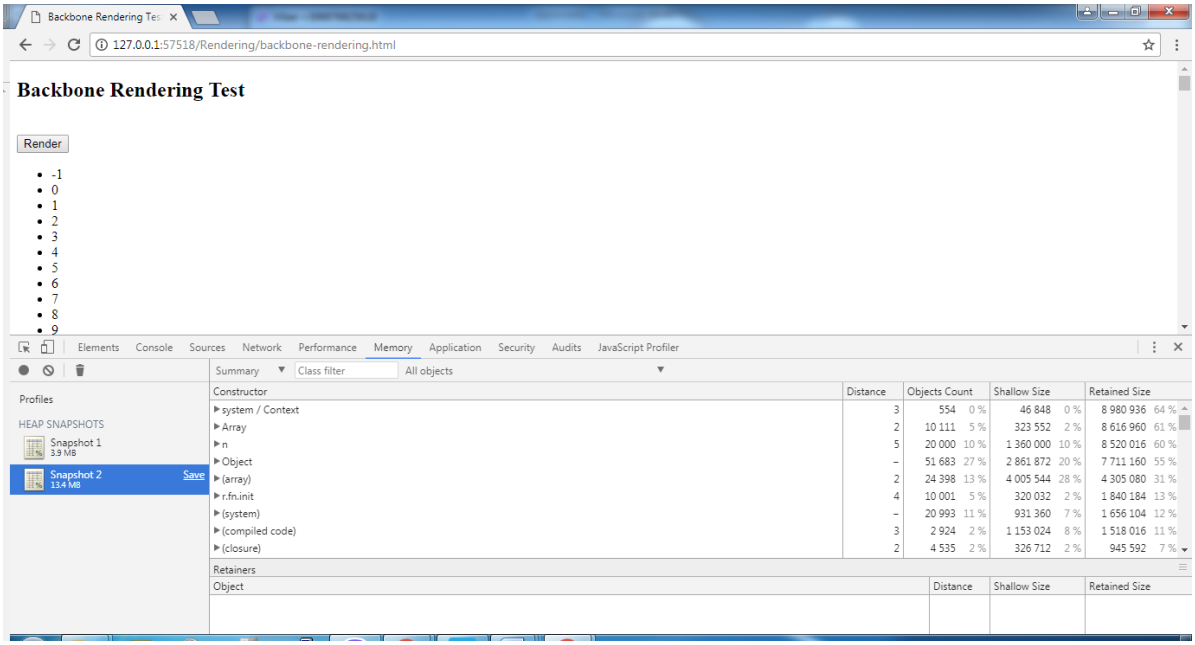
Σε αυτό το σημείο θα δούμε αναλυτικά τον τρόπο με τον οποίο διεξήχθησαν τα πειράματα για την κάθε υλοποίηση, και θα παραθέσουμε μερικά στιγμιότυπα οθόνης (screenshots) με τα αποτελέσματα. Τα πειράματά μας έχουν γίνει με χρήση του πιο δημοφιλούς browser, του Google Chrome, σε υπολογιστή με λειτουργικό σύστημα Windows 7. Ας περάσουμε όμως να δούμε τη διαδικασία που ακολουθήθηκε.

Για την κατηγορία του Rendering η διαδικασία έχει ως εξής:

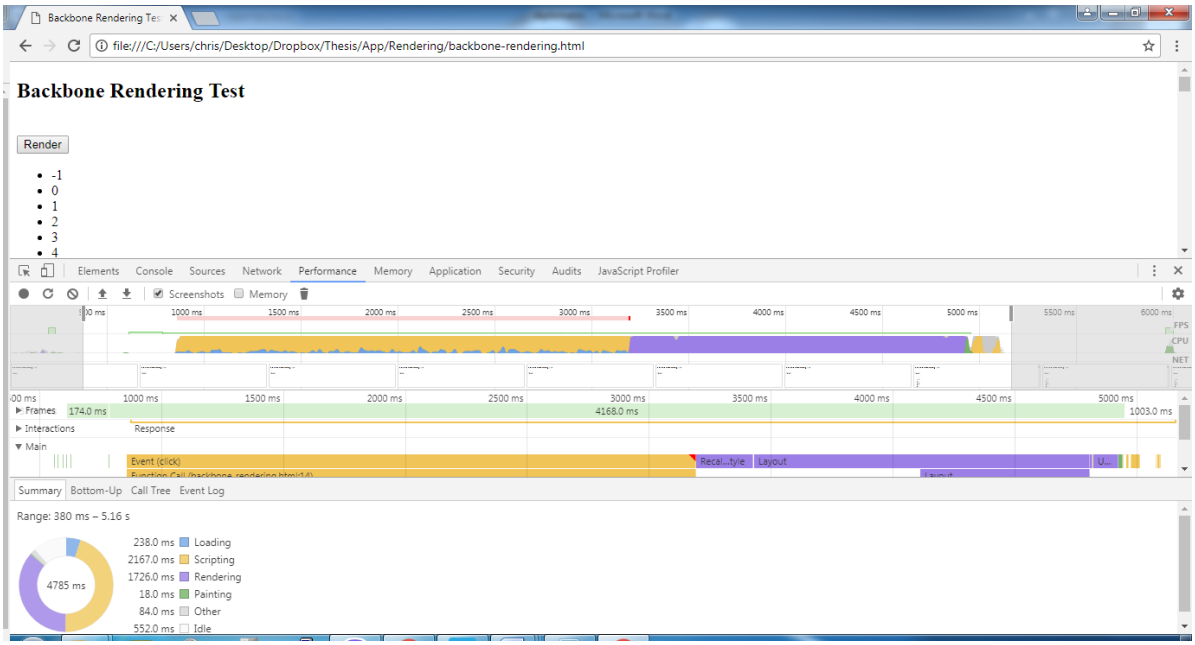
1. Ανοίγουμε ένα καινούριο παράθυρο.
2. Πληκτρολογούμε Ctrl + O, επιλέγουμε το αρχείο με κατάληξη .html και κάνουμε διπλό κλικ επάνω του.
3. Κάνουμε κλικ στο κουμπί ‘Render’, και περιμένουμε να εμφανιστούν τα υπόλοιπα αντικείμενα στην οθόνη.
4. Πληκτρολογούμε F12, και εμφανίζονται τα εργαλεία του browser (dev tools).
5. Κάνουμε κλικ στην καρτέλα Console, και παρατηρούμε την εγγραφή στην κονσόλα του browser. Κάνουμε κλικ στις ρυθμίσεις της κονσόλας, και επιλέγουμε ‘Preserve log’. Ανανεώνουμε τη σελίδα, και επαναλαμβάνουμε το βήμα 3. Επαναλαμβάνουμε το συγκεκριμένο κομμάτι μέχρις ότου να έχουμε δέκα εγγραφές με χρόνους στην κονσόλα, εκ των οποίων υπολογίζουμε το μέσο όρο. Αυτή ειναι και η αντιπροσωπευτική τιμή για το χρόνο εκτέλεσης του κρίσιμου τμήματος του κώδικα της εκάστοτε υλοποίησης.
6. Ανανεώνουμε τη σελίδα, κάνουμε κλικ στην καρτέλα Memory, ως προεπιλογή υπάρχει το ‘Take heap snaphot’, και πληκτρολογούμε Ctrl + E.
7. Επαναλαμβάνουμε το βήμα 3. Πληκτρολογούμε και πάλι Ctrl + E, και παίρνουμε το τελικό heap snapshot. Η διαφορά στις δύο τιμές (τελική – αρχική) είναι η κατανάλωση μνήμης της συγκεκριμένης υλοποίησης.
8. Κάνουμε κλικ στην καρτέλα Performance. Ανανεώνουμε τη σελίδα. Πληκτρολογούμε Ctrl + E. Επαναλαμβάνουμε το βήμα 3. Πληκτρολογούμε και πάλι Ctrl + E. Στο κάτω μέρος της καρτέλας βλέπουμε το CPU chart. Η τιμή στο κέντρο του μείον την τιμή του πεδίου Idle είναι και η τιμή του συνολικού χρόνου χρησιμοποίησης του επεξεργαστή για τη συγκεκριμένη υλοποίηση.



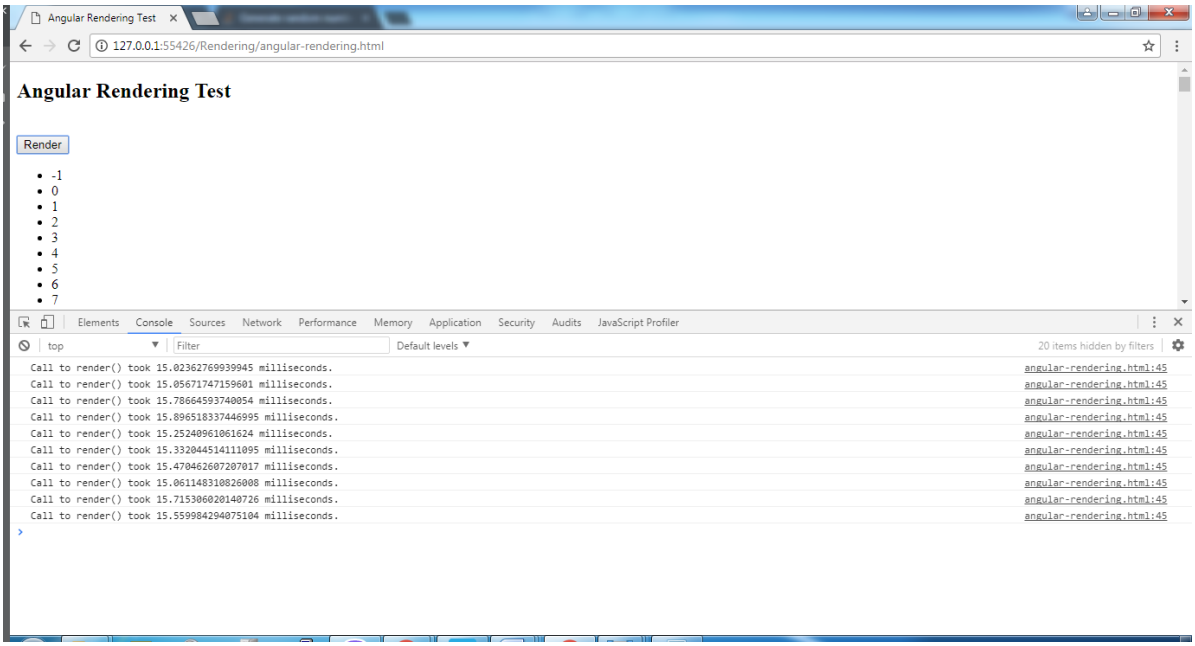

Backbone rendering test execution time



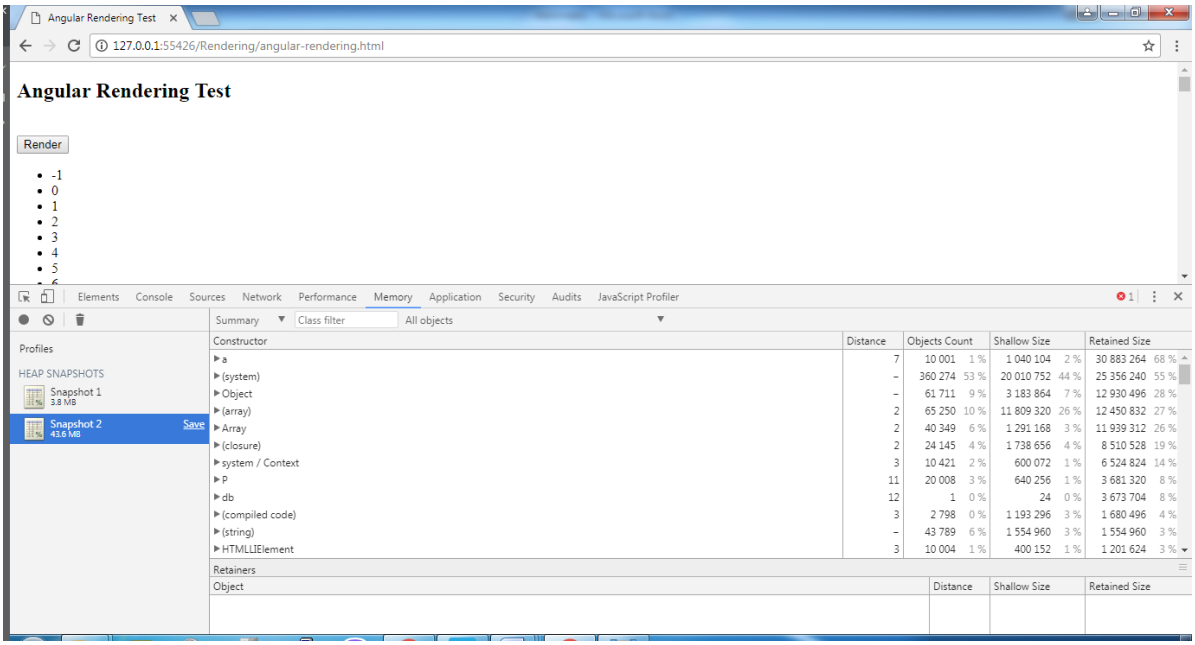

Backbone rendering test memory consumption



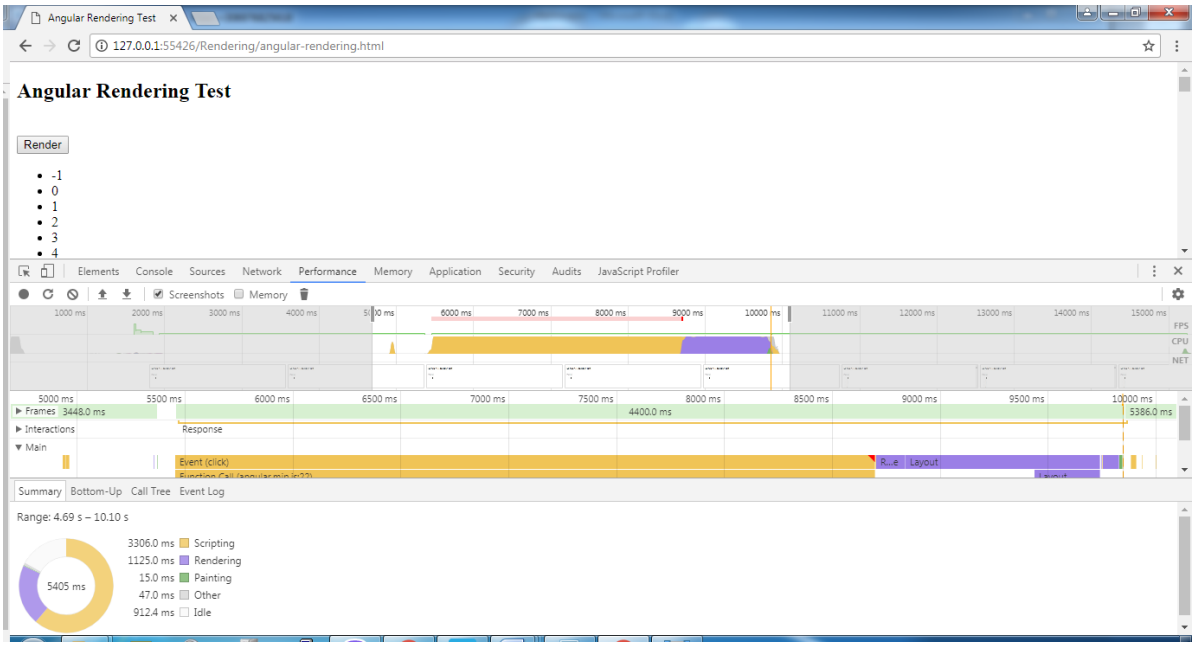

Backbone rendering test CPU time



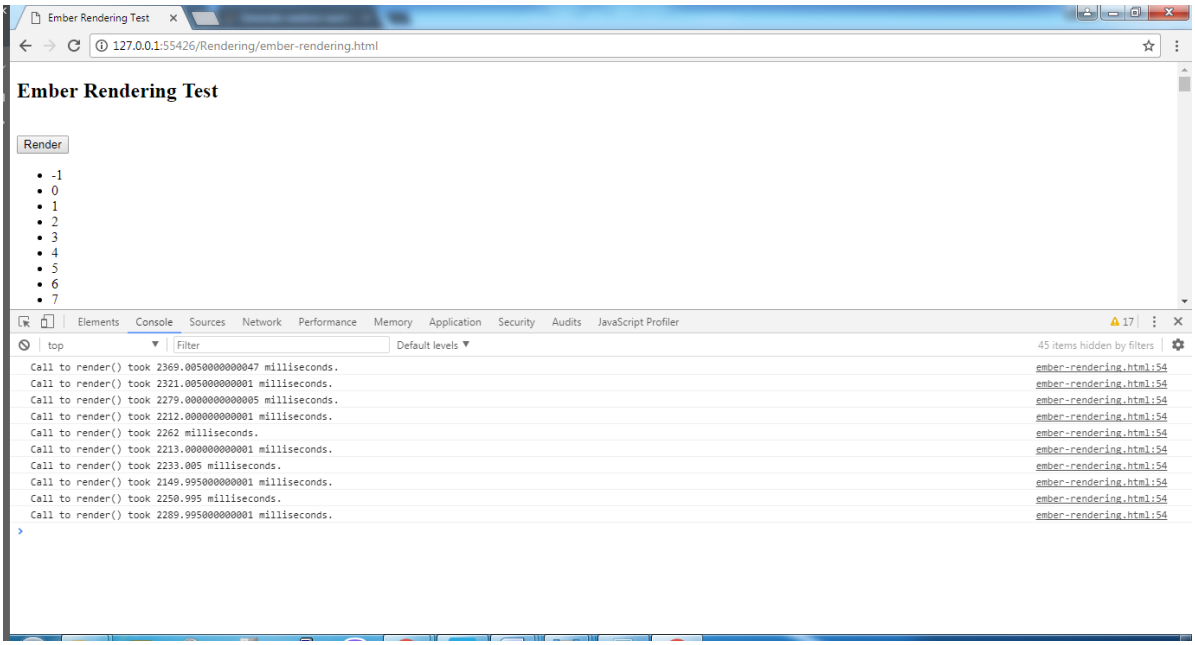

Angularjs rendering test execution time



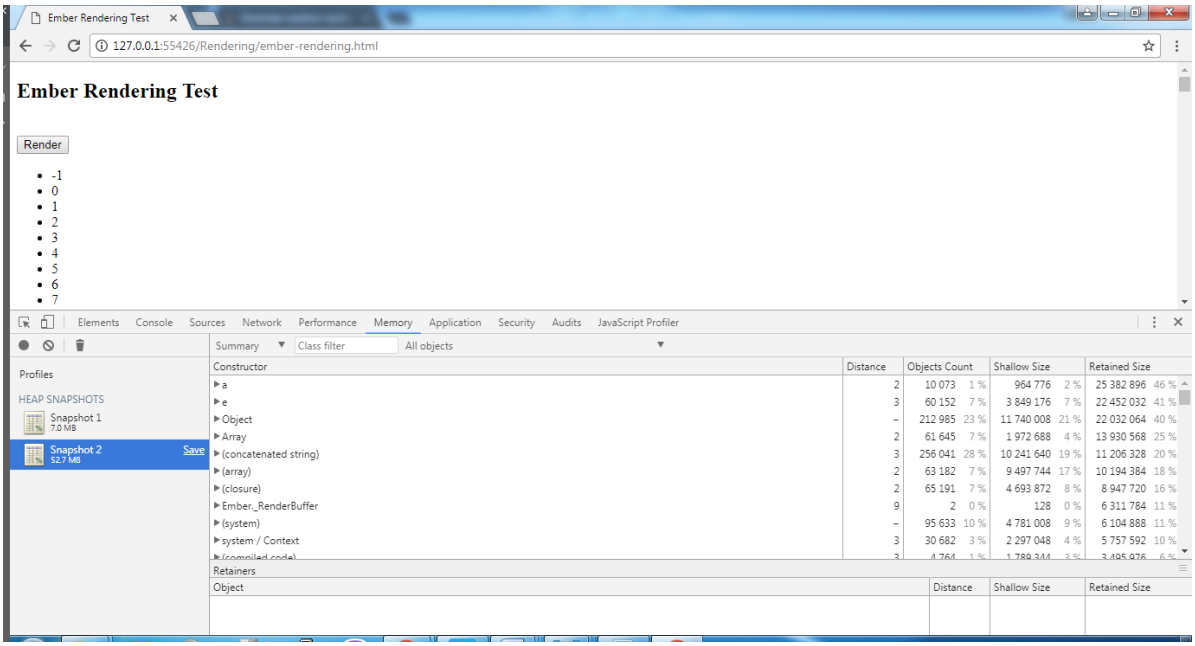

Angularjs rendering test memory consumption



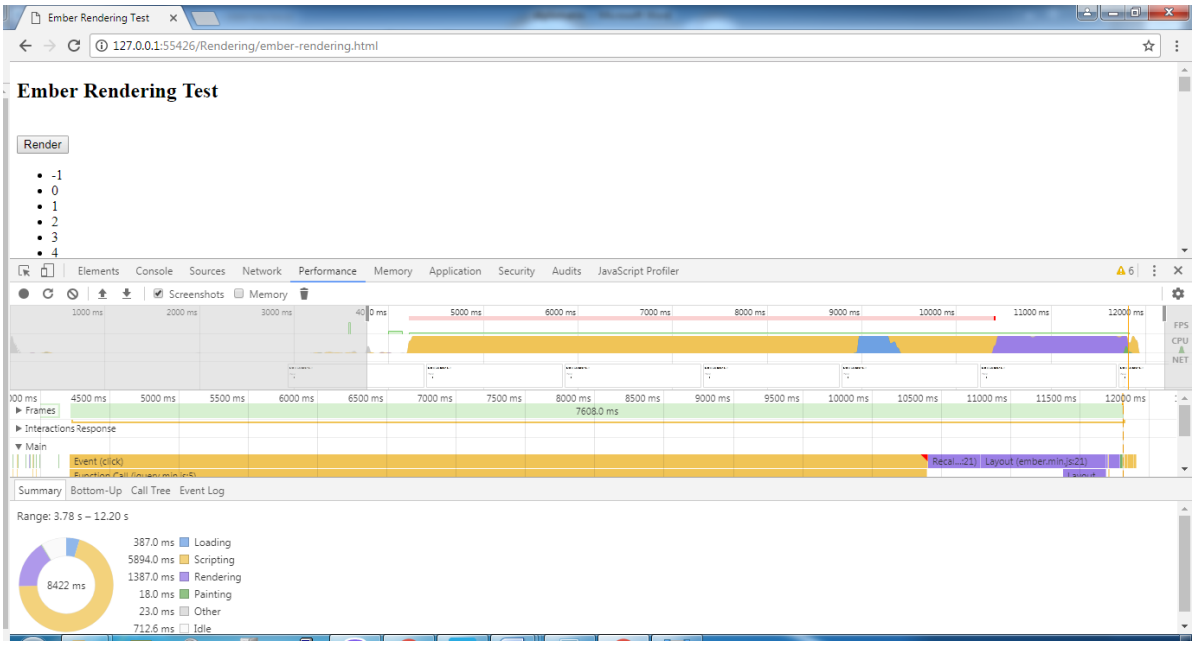

Angularjs rendering test CPU time




Ember rendering test execution time




Ember rendering test memory consumption

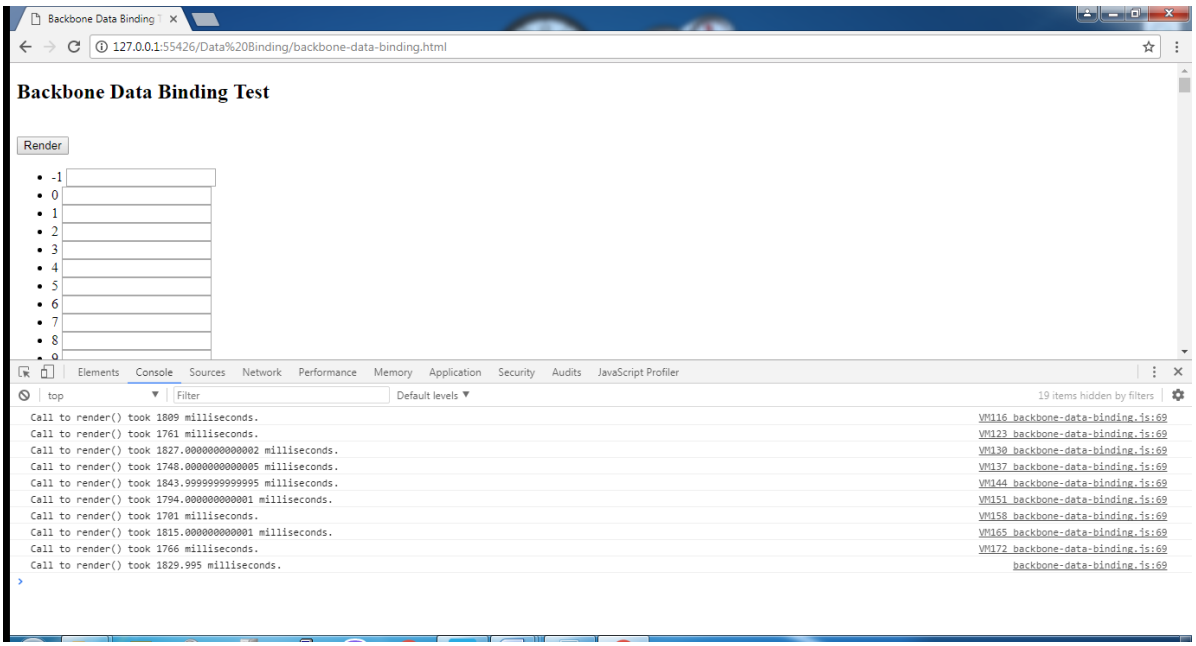



Ember rendering test CPU time

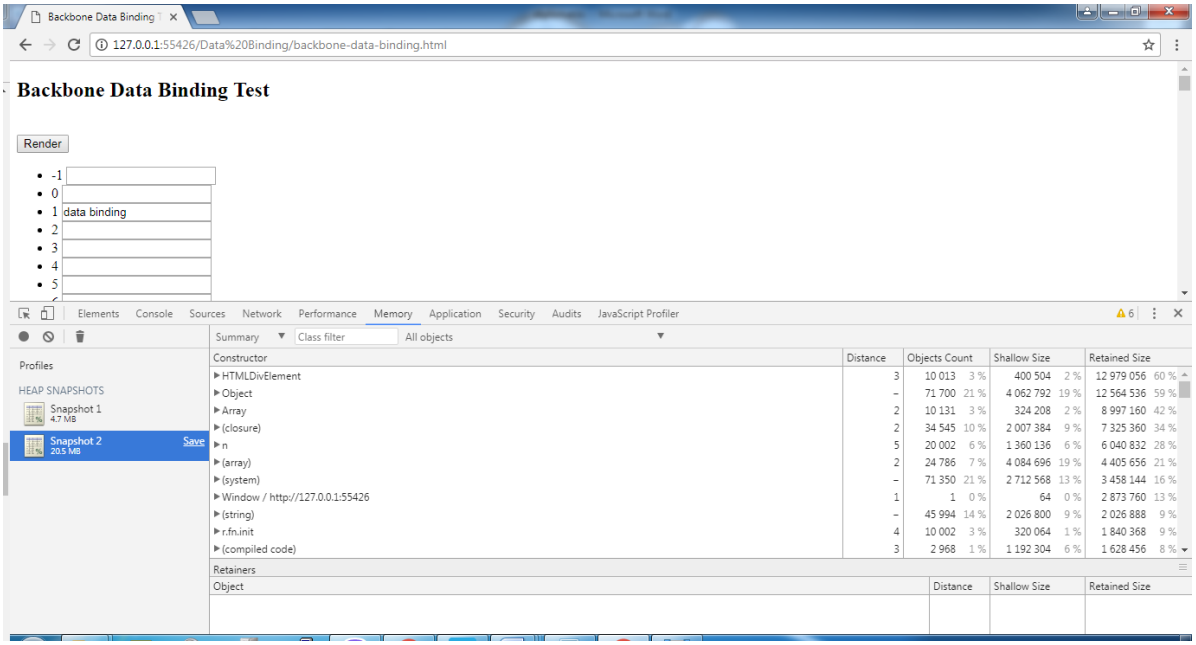
Για την κατηγορία του Data Binding η διαδικασία έχει ως εξής:

1. Ανοίγουμε ένα καινούριο παράθυρο.
2. Πληκτρολογούμε Ctrl + O, επιλέγουμε το αρχείο με κατάληξη .html και κάνουμε διπλό κλικ επάνω του.
3. Κάνουμε κλικ στο κουμπί ‘Render’, και περιμένουμε να εμφανιστούν τα υπόλοιπα αντικείμενα στην οθόνη.
4. Πληκτρολογούμε F12, και εμφανίζονται τα εργαλεία του browser (dev tools).
5. Κάνουμε κλικ στην καρτέλα Console, και παρατηρούμε την εγγραφή στην κονσόλα του browser. Κάνουμε κλικ στις ρυθμίσεις της κονσόλας, και επιλέγουμε ‘Preserve log’. Ανανεώνουμε τη σελίδα, και επαναλαμβάνουμε το βήμα 3. Επαναλαμβάνουμε το συγκεκριμένο κομμάτι μέχρις ότου να έχουμε δέκα εγγραφές με χρόνους στην κονσόλα, εκ των οποίων υπολογίζουμε το μέσο όρο. Αυτή ειναι και η αντιπροσωπευτική τιμή για το χρόνο εκτέλεσης του κρίσιμου τμήματος του κώδικα της εκάστοτε υλοποίησης.
6. Ανανεώνουμε τη σελίδα, κάνουμε κλικ στην καρτέλα Memory, ως προεπιλογή υπάρχει το ‘Take heap snaphot’, και πληκτρολογούμε Ctrl + E.
7. Επαναλαμβάνουμε το βήμα 3. Επιλέγουμε κάποιο από τα διαθέσιμα πεδία κειμένου και πληκτρολογούμε τη δοκιμαστική φράση ‘data binding’. Πληκτρολογούμε και πάλι Ctrl + E, και παίρνουμε το τελικό heap snapshot. Η διαφορά στις δύο τιμές (τελική – αρχική) είναι η κατανάλωση μνήμης της συγκεκριμένης υλοποίησης.
8. Κάνουμε κλικ στην καρτέλα Performance. Ανανεώνουμε τη σελίδα. Πληκτρολογούμε Ctrl + E. Επαναλαμβάνουμε το βήμα 3. Επιλέγουμε κάποιο από τα διαθέσιμα πεδία κειμένου και πληκτρολογούμε τη δοκιμαστική φράση ‘data binding’. Πληκτρολογούμε και πάλι Ctrl + E. Στο κάτω μέρος της καρτέλας

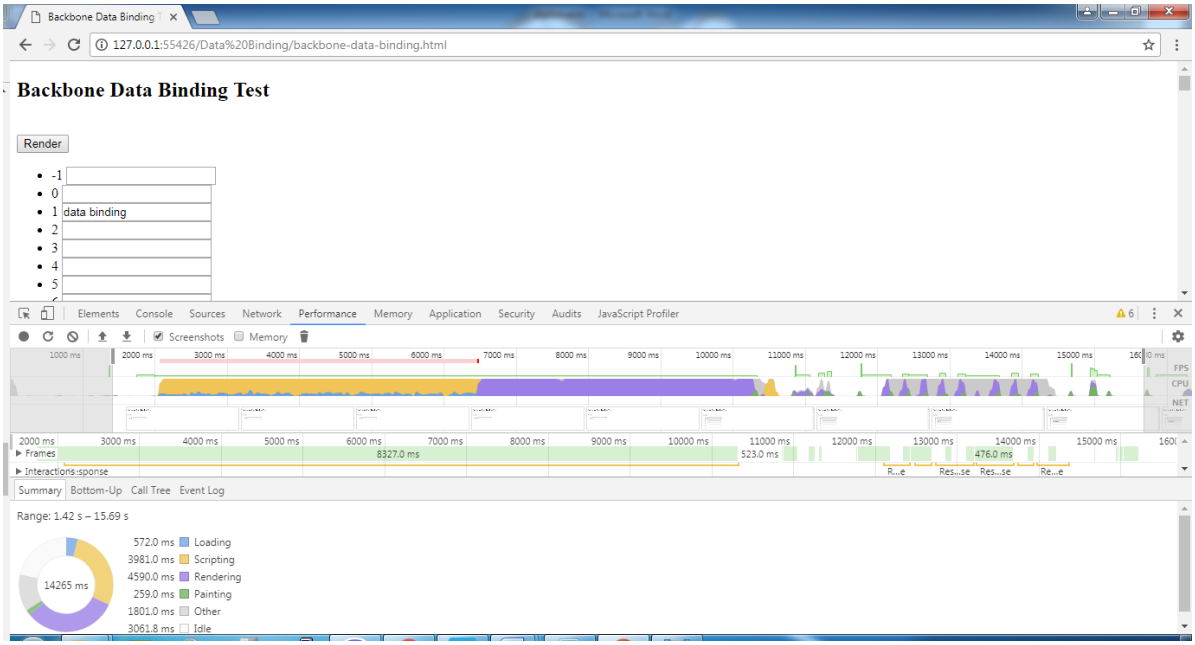
βλέπουμε το CPU chart. Η τιμή στο κέντρο του μείον την τιμή του πεδίου Idle είναι και η τιμή του συνολικού χρόνου χρησιμοποίησης του επεξεργαστή για τη συγκεκριμένη υλοποίηση.



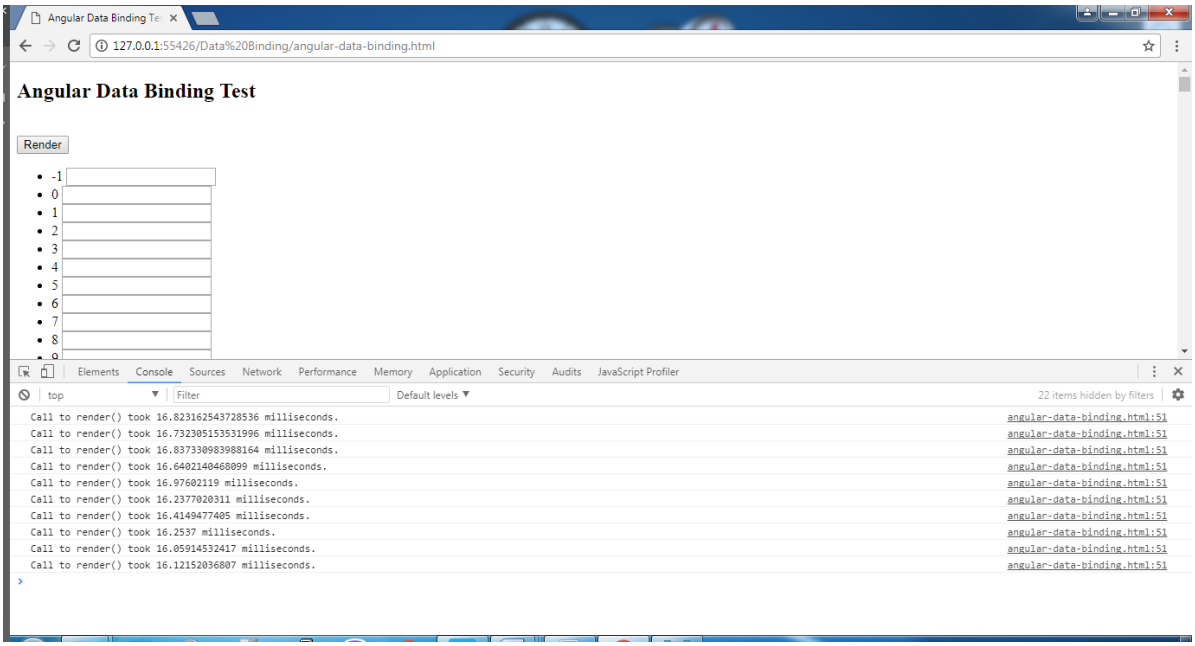

Backbone data binding test execution time



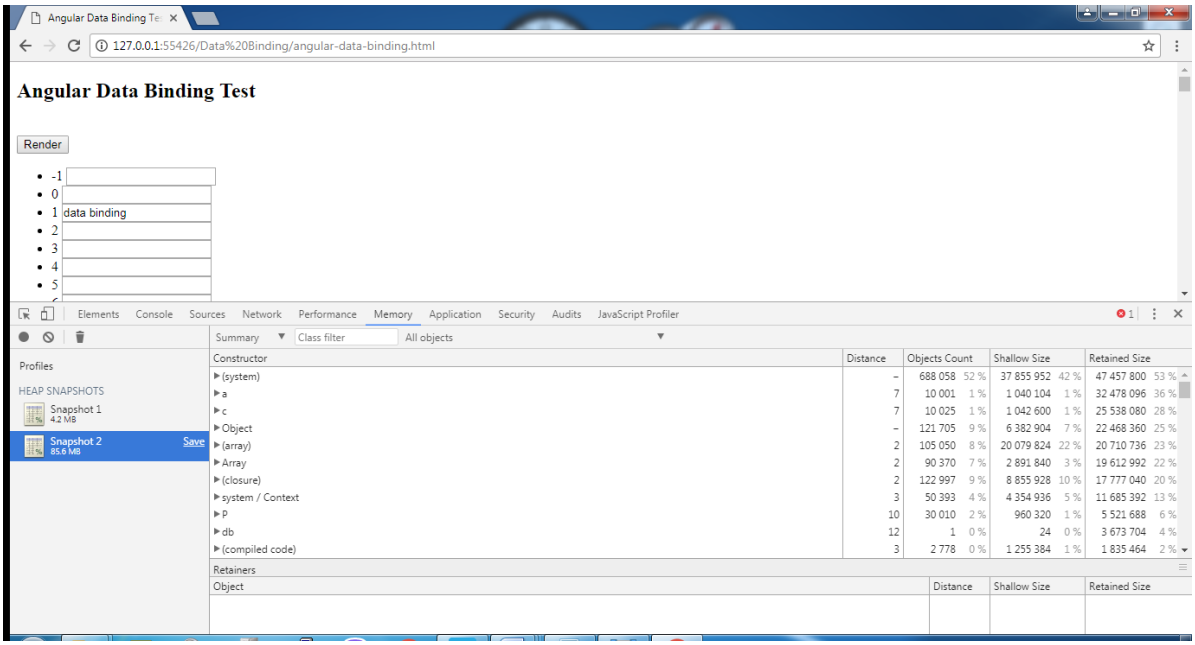

Backbone data binding test memory consumption



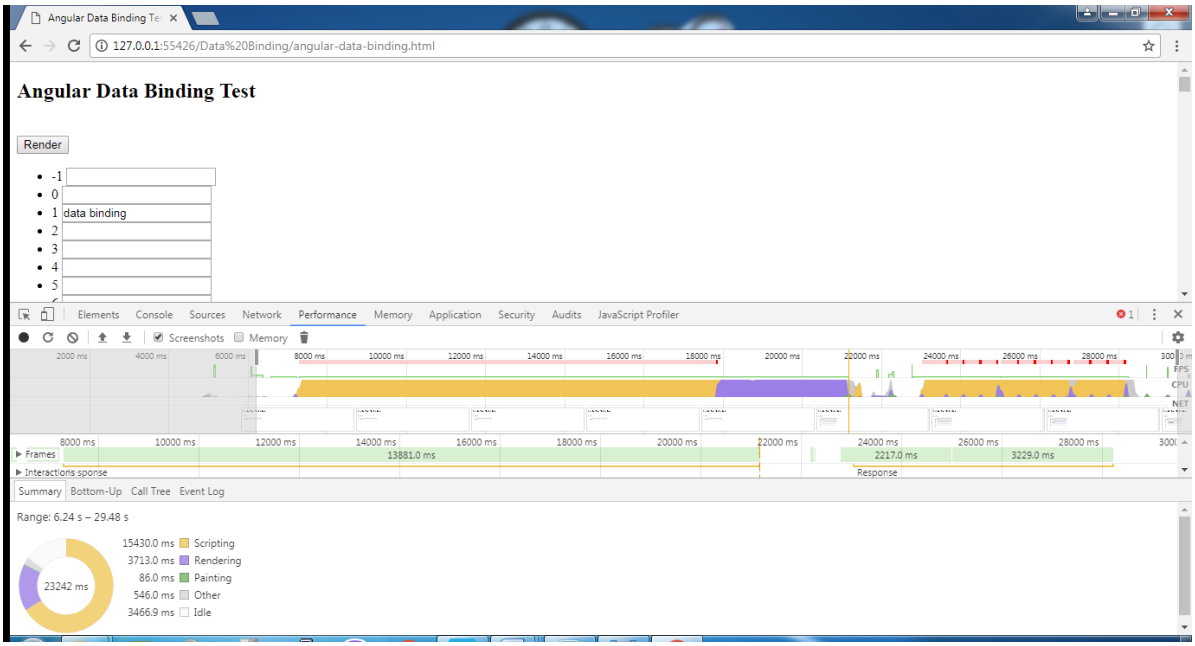

Backbone data binding test CPU time



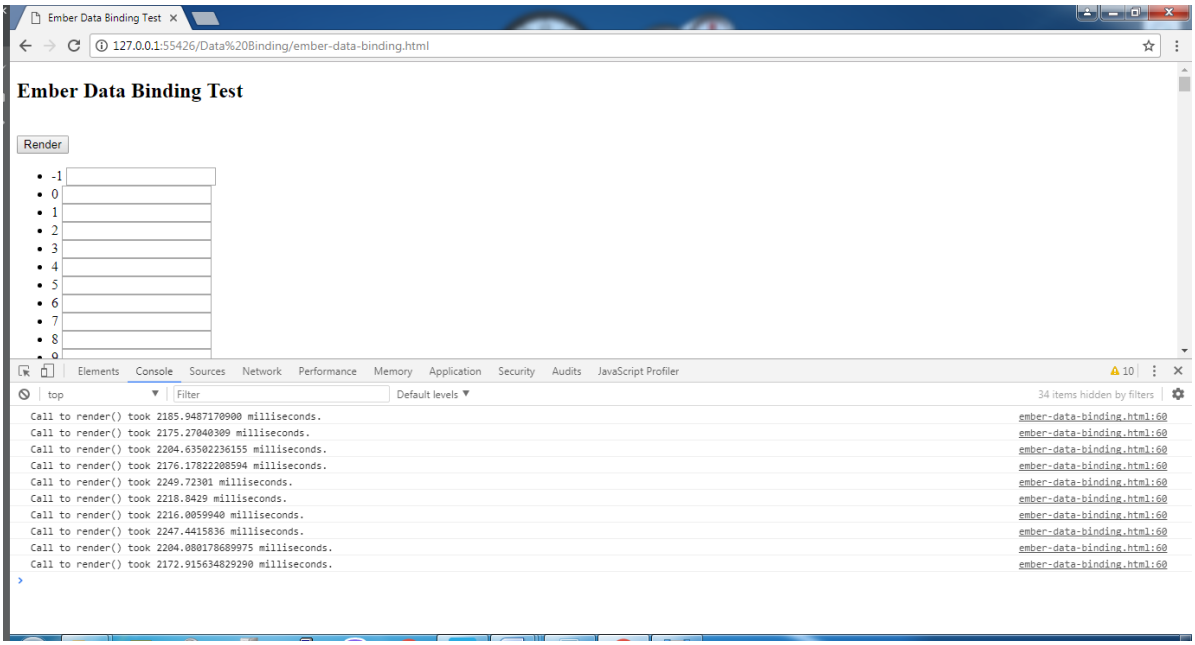

Angularjs data binding test execution time



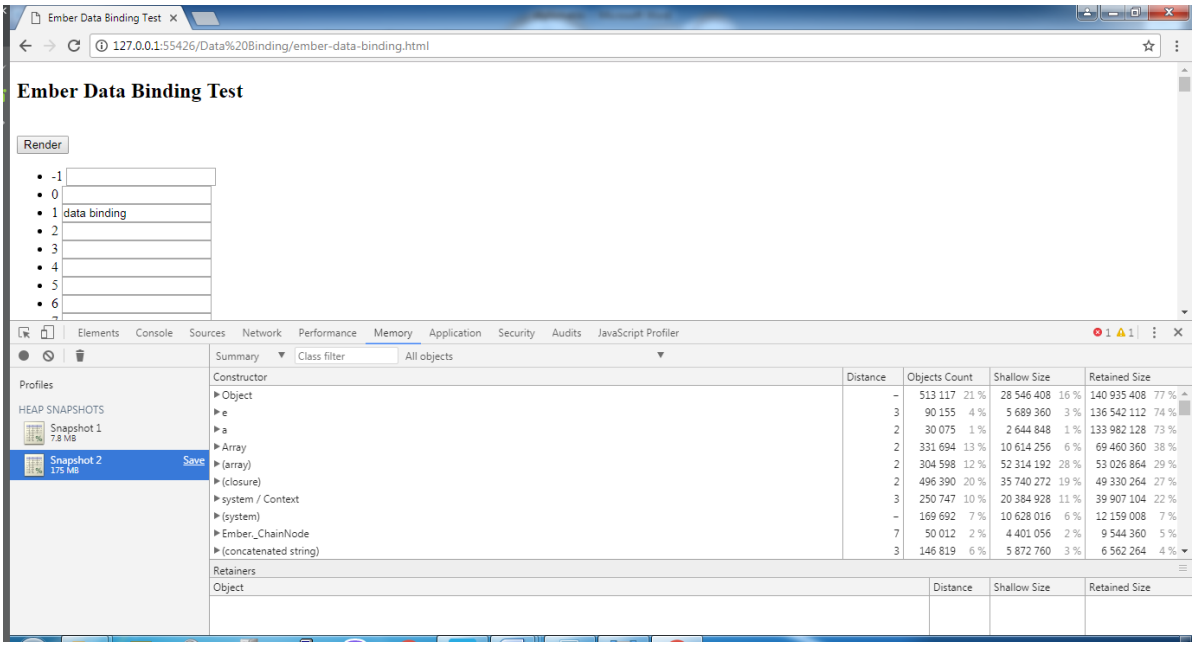

Angularjs data binding test memory consumption



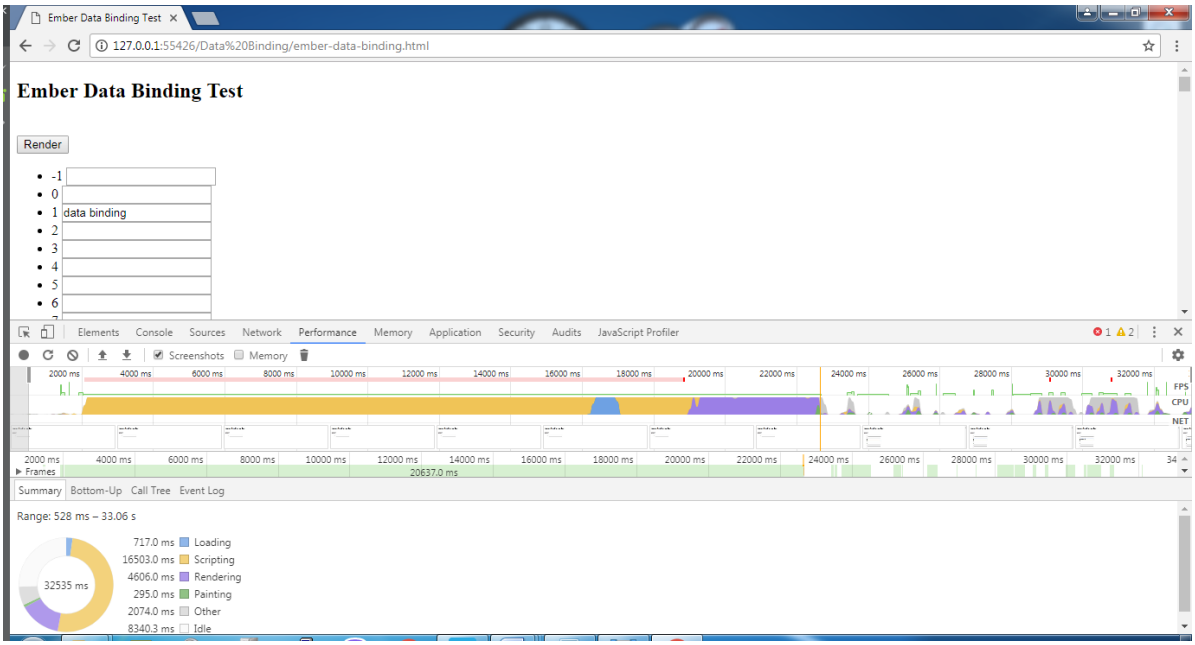

Angularjs data binding test CPU time




Ember data binding test execution time



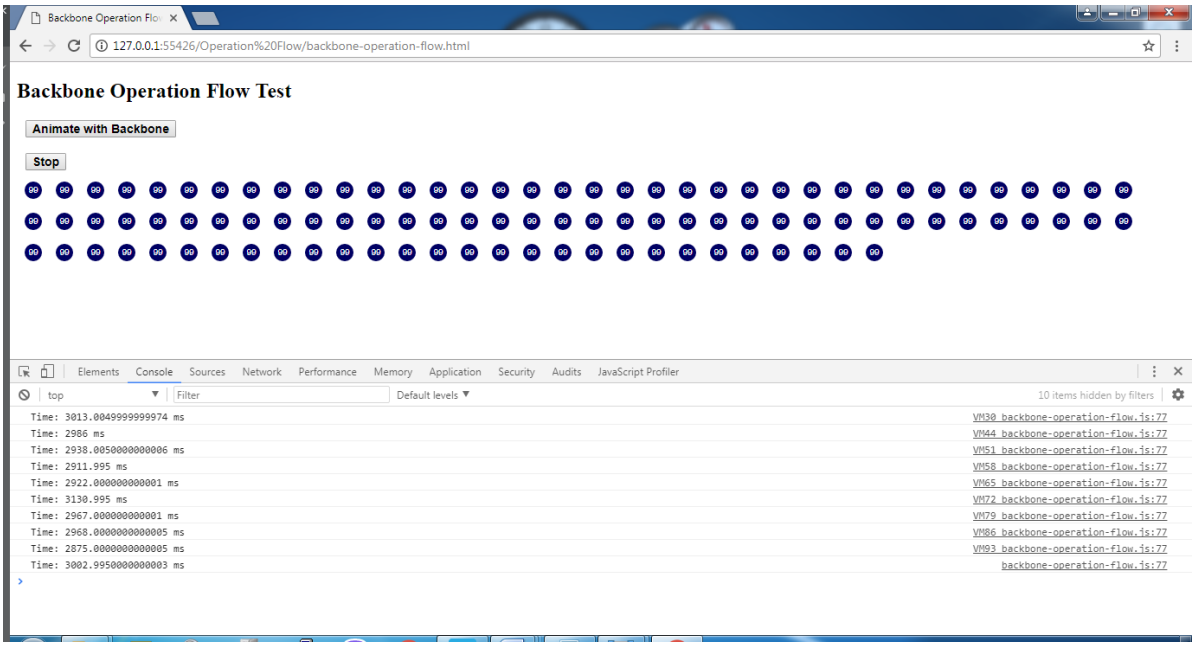

Ember data binding test memory consumption



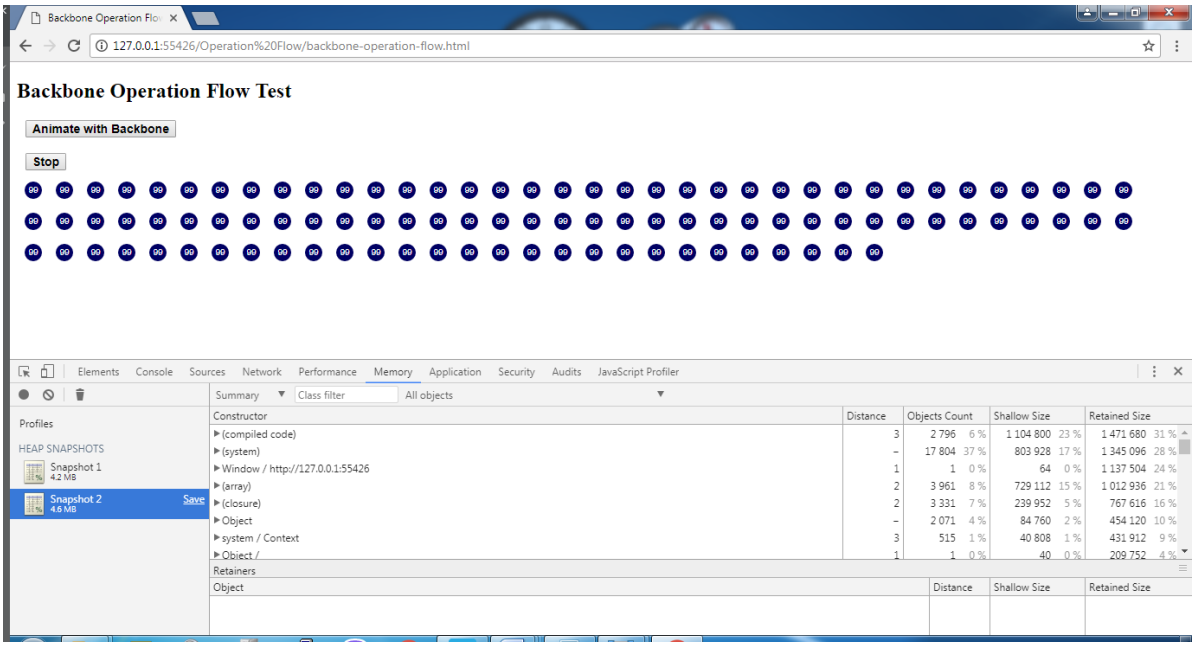

Ember data binding test CPU time

Για την κατηγορία του Operation Flow η διαδικασία έχει ως εξής:

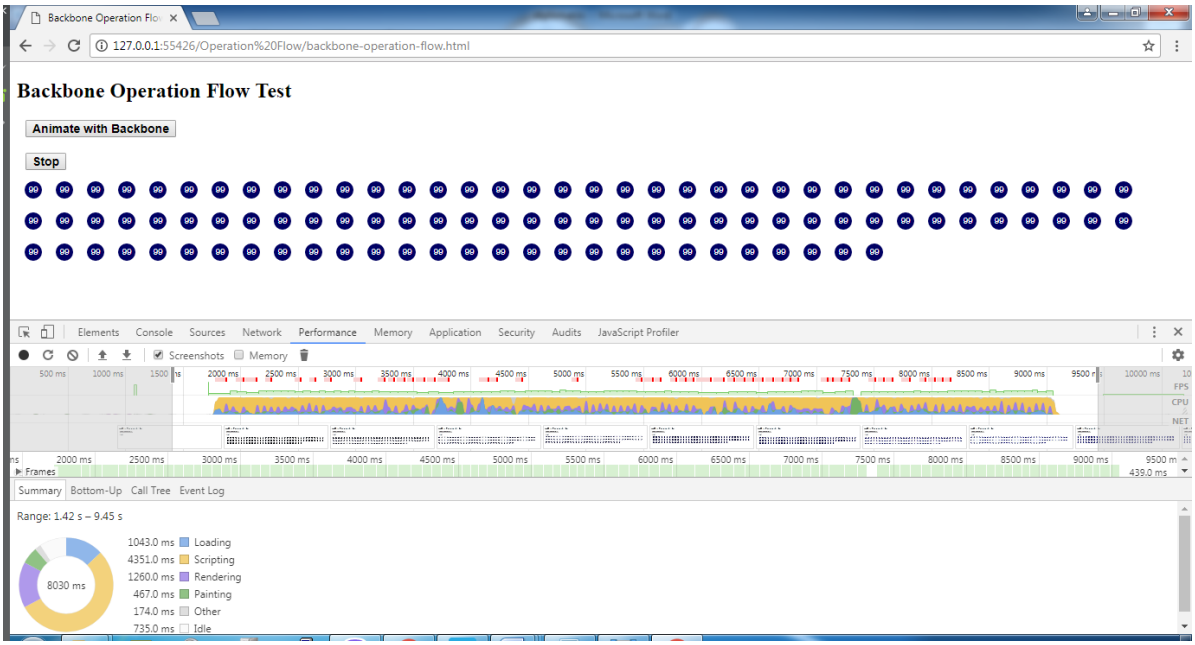
1. Ανοίγουμε ένα καινούριο παράθυρο.
2. Πληκτρολογούμε Ctrl + O, επιλέγουμε το αρχείο με κατάληξη .html και κάνουμε διπλό κλικ επάνω του.
3. Κάνουμε κλικ στο κουμπί ‘Animate with Backbone/ AngularJS/ Ember’, και περιμένουμε να ολοκληρωθεί το animation των αντικειμένων στην οθόνη.
4. Πληκτρολογούμε F12, και εμφανίζονται τα εργαλεία του browser (dev tools).
5. Κάνουμε κλικ στην καρτέλα Console, και παρατηρούμε την εγγραφή στην κονσόλα του browser. Κάνουμε κλικ στις ρυθμίσεις της κονσόλας, και επιλέγουμε ‘Preserve log’. Ανανεώνουμε τη σελίδα, και επαναλαμβάνουμε το βήμα 3. Επαναλαμβάνουμε το συγκεκριμένο κομμάτι μέχρις ότου να έχουμε δέκα εγγραφές με χρόνους στην κονσόλα, εκ των οποίων υπολογίζουμε το μέσο όρο. Αυτή ειναι και η αντιπροσωπευτική τιμή για το χρόνο εκτέλεσης του κρίσιμου τμήματος του κώδικα της εκάστοτε υλοποίησης.
6. Ανανεώνουμε τη σελίδα, κάνουμε κλικ στην καρτέλα Memory, ως προεπιλογή υπάρχει το ‘Take heap snaphot’, και πληκτρολογούμε Ctrl + E.
7. Επαναλαμβάνουμε το βήμα 3. Πληκτρολογούμε και πάλι Ctrl + E, και παίρνουμε το τελικό heap snapshot. Η διαφορά στις δύο τιμές (τελική – αρχική) είναι η κατανάλωση μνήμης της συγκεκριμένης υλοποίησης.
8. Κάνουμε κλικ στην καρτέλα Performance. Ανανεώνουμε τη σελίδα. Πληκτρολογούμε Ctrl + E. Επαναλαμβάνουμε το βήμα 3. Πληκτρολογούμε και πάλι Ctrl + E. Στο κάτω μέρος της καρτέλας βλέπουμε το CPU chart. Η τιμή στο κέντρο του μείον την τιμή του πεδίου Idle είναι και η τιμή του συνολικού χρόνου χρησιμοποίησης του επεξεργαστή για τη συγκεκριμένη υλοποίηση.



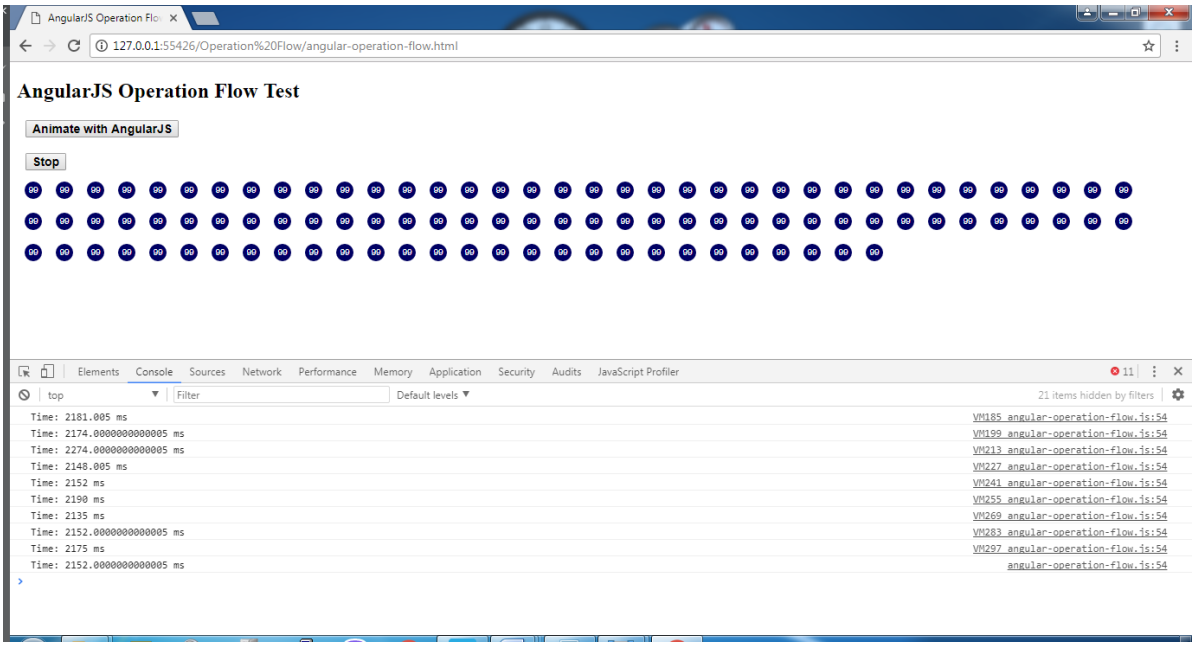

Backbone operation flow test execution time



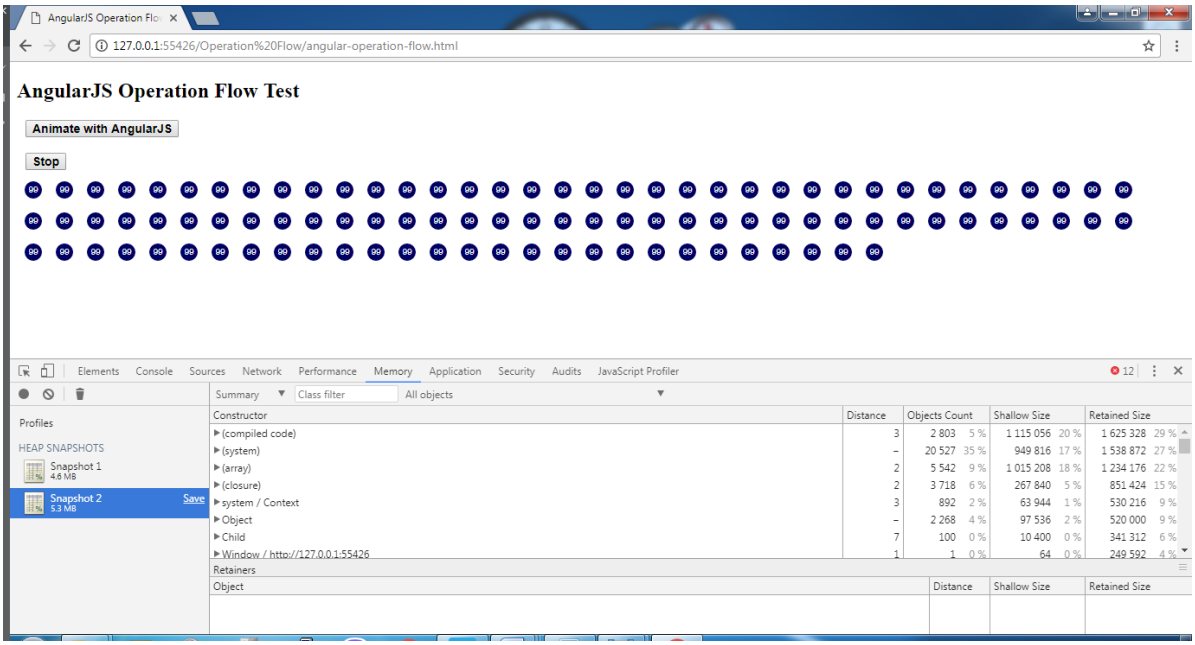

Backbone operation flow test memory consumption



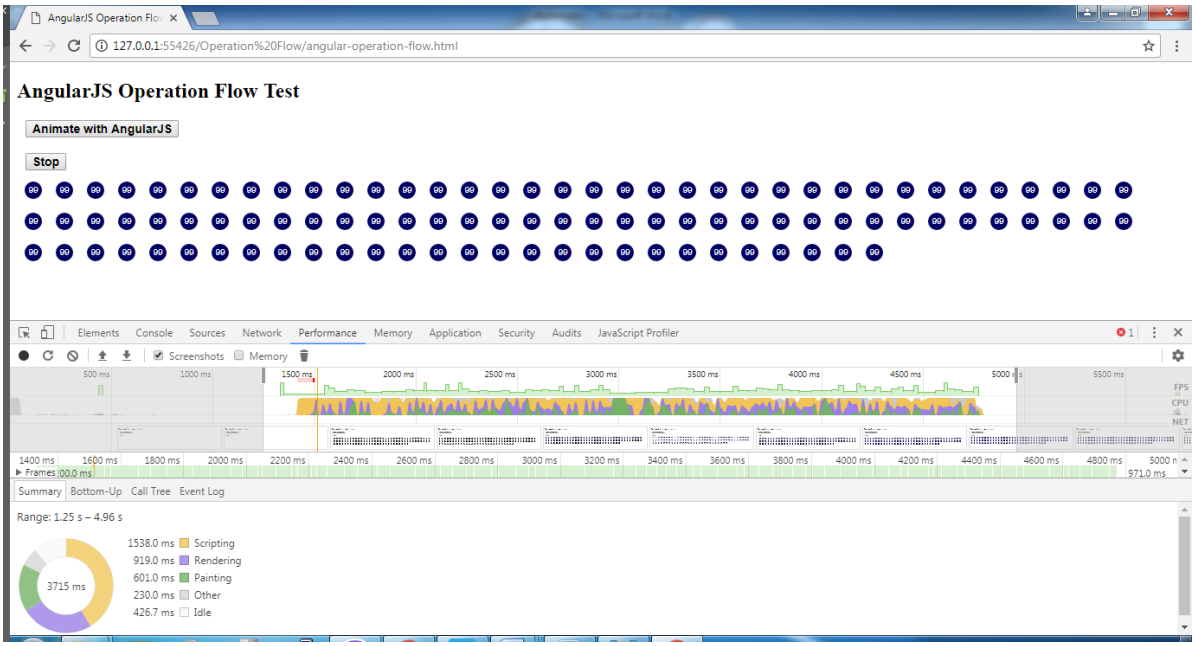

Backbone operation flow test CPU time



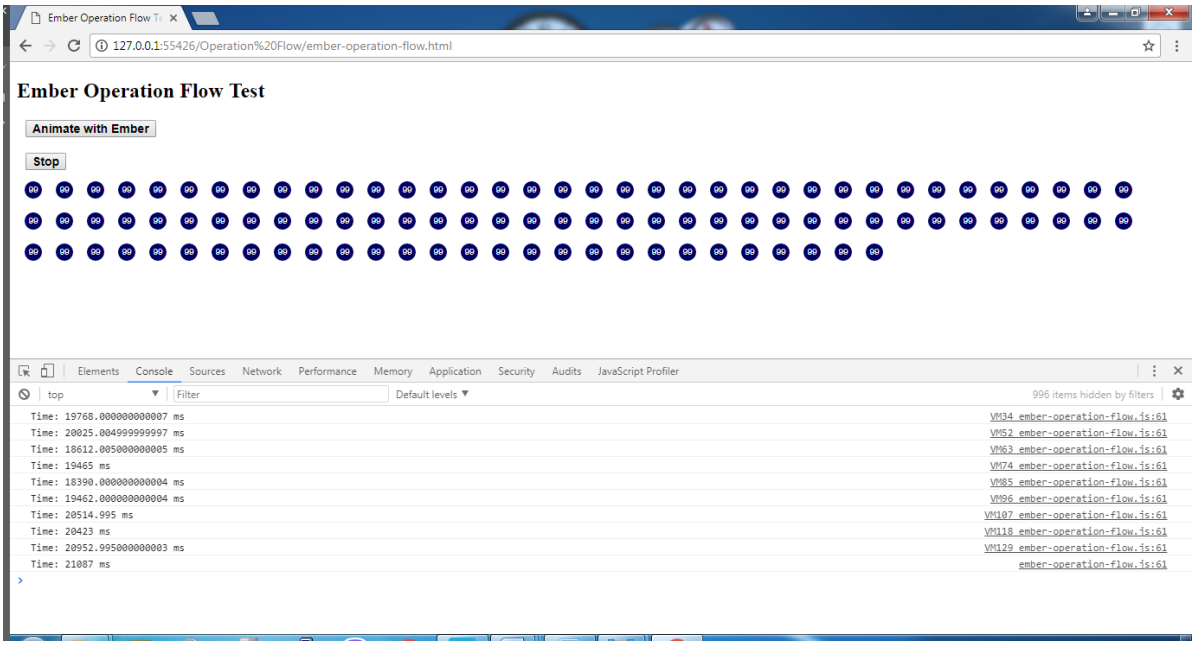

Angularjs operation flow test execution time



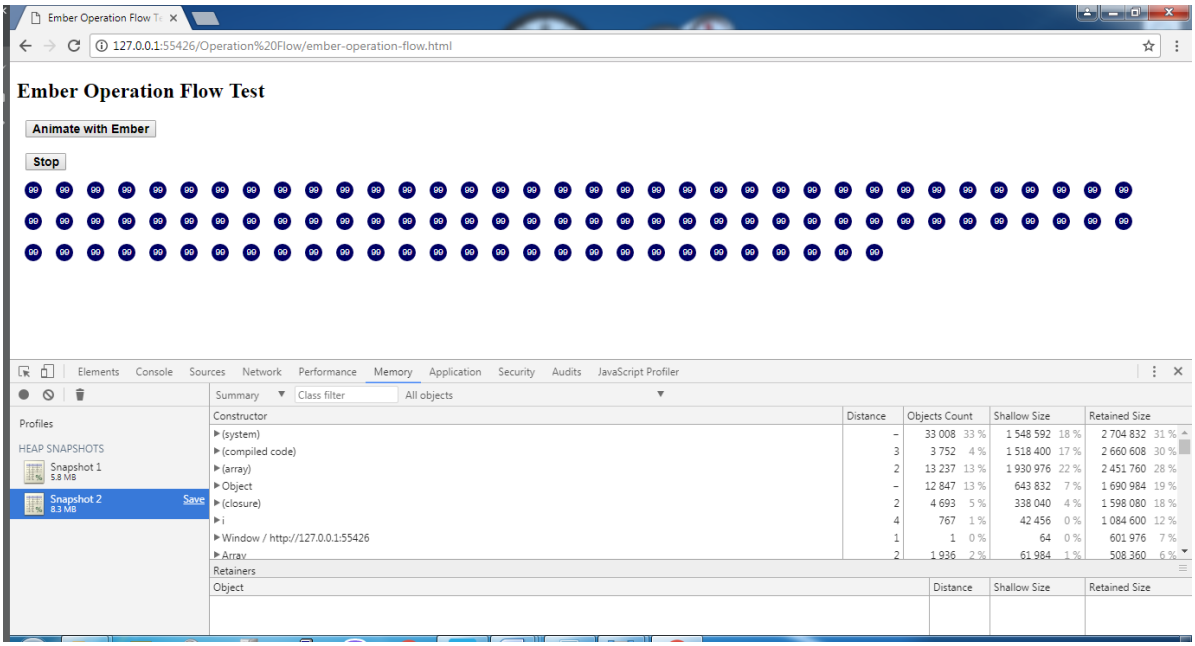

Angularjs operation flow test memory consumption



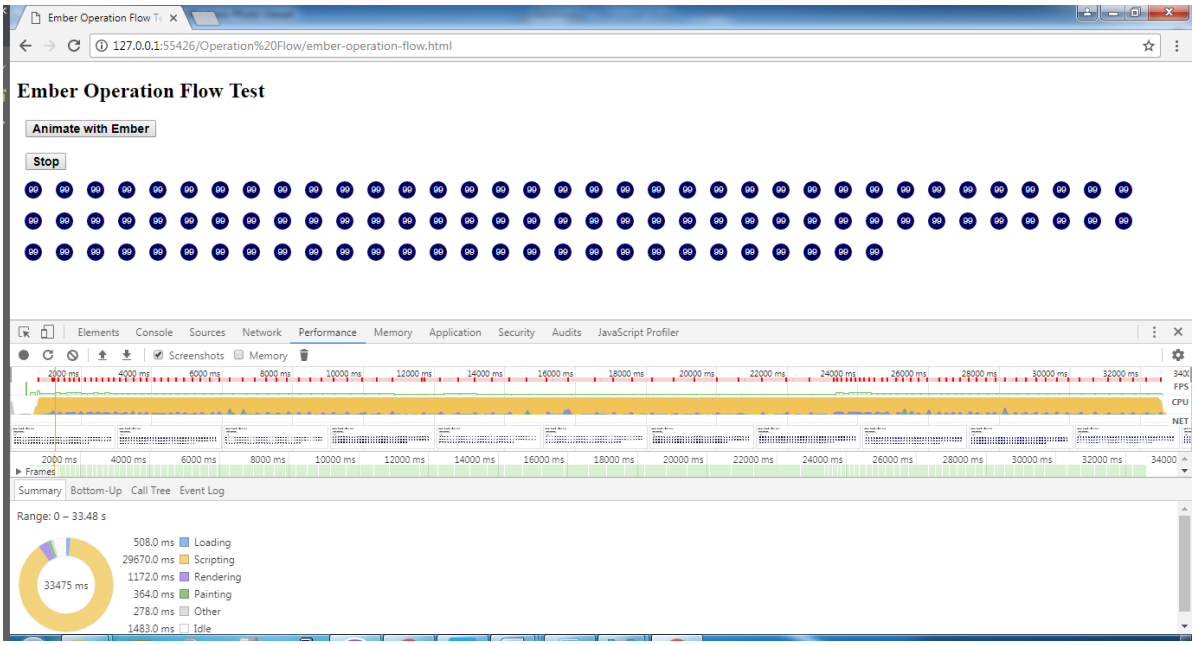

Angularjs operation flow test CPU time




Ember operation flow test execution time




Ember operation flow test memory consumption




Ember operation flow test CPU time

# Κεφάλαιο 7

# Επίλογος

## 7.1 Σύνοψη

Από τα παραπάνω βλέπουμε ότι τα αποτελέσματα των μετρήσεών μας διαφέρουν αρκετά από framework σε framework και από υλοποίηση σε υλοποίηση. Συγκεκριμένα, για την κατηγορία του Rendering, το Backbone έχει μέσο όρο χρόνου εκτέλεσης για το κρίσιμο τμήμα του κώδικα στα 914.09 ms, το AngularJS στα 15.41 ms και το Ember στα 2258 ms. Όσον αφορά την κατανάλωση μνήμης, αυτή είναι στα 9.5 MB για το Backbone, στα 39.8 MB για το AngularJS και 45.7 MB για το Ember. Τέλος, ο χρόνος χρησιμοποίησης του επεξεργαστή είναι στα 4233 ms για το Backbone, στα 4492.6 ms για το AngularJS και στα 7709.4 ms για το Ember. Συνοπτικά, παρουσιάζονται στον παρακάτω πίνακα:

| **Rendering** | Backbone | AngularJS | Ember |
|---|---|---|---|
| Execution time (ms) | 914.09 | 15.41 | 2258 |
| Memory (MB) | 9.5 | 39.8 | 45.7 |
| CPU time (ms) | 4233 | 4492.6 | 7709.4 |

Table 7.1 Rendering results

Βλέπουμε ότι, στην περίπτωση του Angular, παρά το γεγονός ότι ο χρόνος εκτέλεσης του κρίσιμου τμήματος είναι πολύ μικρός, κάτι το οποίο έχει να κάνει με το πως χειρίζεται εσωτερικά το framework τη μεταβλητή $scope και το directive ng-repeat, ο συνολικός χρόνος του επεξεργαστή είναι στα ίδια επίπεδα με το Backbone, παρουσιάζοντας σημαντική διαφορά σε σχέση με το Ember. Αυτό συμβαίνει, διότι, το Ember προσθέτει έναν observer σε κάθε αντικείμενο, ούτως ώστε να ξέρει πότε έχει μεταβληθεί η κατάστασή του, και αυτό καθιστά το rendering των αντικειμένων αρκετά πιο αργό σε σύγκριση με τα άλλα δύο framework.

Για την κατηγορία του Data Binding πήραμε τα εξής αποτελέσματα:

i. Για το χρόνο εκτέλεσης του κρίσιμου τμήματος οι μέσοι όροι ήταν 1789.39 ms, 16.5 ms και 2205.1 ms για Backbone, AngularJS και Ember αντίστοιχα.
ii. Για την κατανάλωση μνήμης πήραμε 15.4 MB, 80.9 MB και 179.5 MB για Backbone, AngularJS και Ember αντίστοιχα.

iii. Για το χρόνο χρησιμοποίησης του επεξεργαστή πήραμε 11203.2 ms, 19775.1 ms και 24194.7 ms για Backbone, AngularJS και Ember αντίστοιχα.

| **Data Binding** | Backbone | AngularJS | Ember |
|---|---|---|---|
| Execution time (ms) | 1789.39 | 16.5 | 2205.1 |
| Memory (MB) | 15.4 | 80.9 | 179.5 |
| CPU time (ms) | 11203.2 | 19775.1 | 24194.7 |

Table 7.2 Data binding results

Σε αυτή την περίπτωση, το σημαντικό είναι να παρατηρήσουμε την αποκρισιμότητα του κάθε framework αφότου έχουν εμφανιστεί όλα τα αντικείμενα στην οθόνη. Στην υλοποίηση σε Backbone και σε Ember δεν υπάρχει πρόβλημα, διαλέγουμε ένα πεδίο κειμένου, πληκτρολογούμε μία φράση και βλέπουμε οτι η εφαρμογή αποκρίνεται κανονικά, σε πραγματικό χρόνο, χωρίς καθυστέρηση. Αυτό συμβαίνει, διότι το Ember λόγω των observers που έχει προσθέσει στα αντικείμενα γνωρίζει αμέσως ποιανού αντικειμένου έχει μεταβληθεί η τιμή, ενώ το Angular πρέπει να διασχίσει όλα τα αντικείμενα και να ελέγξει σε ποιο έχει πραγματοποιηθεί η αλλαγή (dirty checking), κάτι το οποίο είναι αρκετά χρονοβόρο, το οποίο το καθιστά λιγότερο αποκρίσιμο σε σχέση με τα άλλα δύο.

Για την κατηγορία του Operation Flow καταγράψαμε τα εξής αποτελέσματα:

i. Σχετικά με το χρόνο εκτέλεσης του κρίσιμου τμήματος του κώδικα πήραμε τους παρακάτω μέσους όρους: 2971.49 ms για Backbone, 2173.3 ms για AngularJS και 19870 ms για Ember.
ii. Αναφορικά με τη μνήμη είχαμε κατανάλωση 0.4 MB για Backbone, 0.7 MB για AngularJS και 2.5 MB για Ember.
iii. Σχετικά με το χρόνο χρησιμοποίησης του επεξεργαστή μετρήσαμε 7295 ms για Backbone, 3288.3 ms για AngularJS και 31992 ms για Ember.

| **Operation Flow** | Backbone | AngularJS | Ember |
|---|---|---|---|
| Execution time (ms) | 2971.49 | 2173.3 | 19870 |
| Memory (MB) | 0.4 | 0.7 | 2.5 |
| CPU time (ms) | 7295 | 3288.3 | 31992 |

Table 7.3 Operation flow results

Εδώ αξίζει να παρατηρήσουμε την τεράστια διαφορά στο χρόνο εκτέλεσης αλλά και στο χρόνο χρησιμοποίησης της CPU του Ember, σε σχέση με τα υπόλοιπα δύο. Αυτό οφείλεται στο γεγονός ότι υπάρχει συνεχής αλλαγή στο model και ανανέωση του view, και οι observers που προσθέτει στα αντικείμενα το Ember επιφέρουν μεγάλο κόστος στο συνολικό χρόνο εκτέλεσης (overhead), διότι κάθε φορά που αλλάζει μία τιμή ανανεώνεται και το αντίστοιχο view, κάνοντας έτσι πολλή παραπάνω, περιττή δουλειά. Στην περίπτωση του Angular, το dirty checking αποδεικνύεται αρκετά χρήσιμο εδώ, αφού πρώτα αλλάζει τις τιμές όλων των αντικειμένων στο model, και στη συνέχεια ανανεώνει το view, χωρίς να χρειάζεται να κάνει οτιδήποτε άλλο.

## 7.2 Συμπεράσματα

Σε αυτό το σημείο μπορούμε με ασφάλεια να εξάγουμε τα εξής σημαντικά συμπεράσματα:

I. Δεν υπάρχει το ιδανικό framework για όλες τις περιπτώσεις και τα σενάρια χρήσης, όχι μόνο ανάμεσα σε αυτά που συγκρίναμε στην παρούσα εργασία, αλλά γενικότερα μεταξύ όλων των διαθέσιμων επιλογών που υπάρχουν στο Internet.

II. Παρά το γεγονός ότι με όλα τα frameworks μπορούμε να φτάσουμε στο ίδιο τελικό αποτελέσμα, το καθένα παρουσιάζει αρκετές διαφορές σε σχέση με τα υπόλοιπα. Πέρα από τη βασική τους διαφορά όσον αφορά το μοντέλο αρχιτεκτονικής λογισμικού που υλοποιούν (πχ MVC, MVP κλπ), παρουσιάζουν διαφορές στις εξαρτήσεις (dependencies) που απαιτούν για να λειτουργήσουν (πχ jQuery, underscore κλπ), στην template engine που χρησιμοποιούν (πχ Handlebars.js), στον τρόπο με τον οποίο επιτυγχάνουν το «δέσιμο» των αντικειμένων με το model (data binding) κλπ. Όλα αυτά τα στοιχεία καθιστούν το κάθε framework ξεχωριστό.

III. Πριν τη λήψη της απόφασης για την επιλογή ενός συγκεκριμένου framework για τη δημιουργία μίας διαδικτυακής εφαρμογής είναι πολύ σημαντικό να προσδιοριστούν όσο καλύτερα γίνεται ορισμένες μεταβλητές: ποιες είναι οι κύριες απαιτήσεις της εφαρμογής και πόσο εύκολα μπορεί να προσαρμοστεί το κάθε framework σε αυτές, σε πόσο μεγάλο κοινό απευθύνεται η εφαρμογή (scalability), ποιο είναι το εκτιμώμενο μέγεθος της εφαρμογής σε απαιτήσεις κώδικα, πόσο διαδραστική χρειάζεται να είναι, πόσο γρήγορο απαιτείται να είναι το ξεκίνημα και η αρχική εγκατάσταση των ρυθμίσεων της εφαρμογής (bootstrapping) και άλλα πολλά. Φαίνεται λοιπόν, ότι, μία τέτοια απόφαση δεν είναι καθόλου εύκολο να ληφθεί, και σίγουρα κάτι τέτοιο θα πρέπει να γίνεται έπειτα από εκτενή έρευνα.

## 7.3 Μελλοντικές επεκτάσεις

Εδώ οφείλουμε να επισημάνουμε ότι οι υλοποιήσεις που παραθέσαμε δύνανται να επεκταθούν με πολλούς τρόπους, καθώς δεν αποτελούν από μόνες τους ολοκληρώμενες, σύνθετες εφαρμογές αλλά ένα κομμάτι αυτών, όπως αναφέρθηκε και προηγουμένως. Συνεπώς, προτείνονται οι εξής επεκτάσεις:

- Η μελέτη της απόδοσης των παραπάνω υλοποιήσεων και σε άλλους browsers, το οποίο είναι χρήσιμο για την ύπαρξη μίας πιο συνολικής εικόνας σχετικά με τη συμπεριφορά του κάθε framework, σε αναλογία πάντα με τον τύπο της εφαρμογής, όχι μόνο σε desktop υπολογιστές και laptop, αλλά και σε φορητές συσκευές.

- Η εισαγωγή υλοποιήσεων και σε άλλα frameworks, όπως για παράδειγμα Angular 2, Vue, React (+Redux κ.α.), Meteor κλπ που αποτελούν από τις πιο διαδεδομένες επιλογές σήμερα, και παρέχουν την υποδομή για τη δημιουργία μοντέρνων, σύνθετων διαδικτυακών εφαρμογών.

- Η επέκταση των υλοποιήσεων ούτως ώστε να είναι πιο κοντά στα πρότυπα μίας πραγματικής εφαρμογής. Για παράδειγμα, για την κατηγορία του rendering, θα μπορούσαμε να λαμβάνουμε τη λίστα κάποιων tweets που ικανοποιούν συγκεκριμένα κριτήρια από το server μέσω κλήσεων σε API (API calls), και να εκτυπώνουμε αυτή στην οθόνη, αντί για τα δοκιμαστικά αντικείμενα. Αυτό πιθανώς θα εισήγαγε και νέες ενδιαφέρουσες μετρικές, όπως το πλήθος των αιτήσεων στο server και ο χρόνος διεκπεραίωσης αυτών.

- Η χρησιμοποίηση των υλοποιήσεων ως έχουν ως κομμάτι μίας μεγαλύτερης εφαρμογής, και η μέτρηση της συνολικής απόδοσης σε αυτό το πλαίσιο, ούτως ώστε να προκύψουν ενδιαφέροντα συμπεράσματα σχετικά με το που καταναλώνει τον περισσότερο χρόνο (μνήμη κλπ) το κάθε framework και γιατί.

# Κεφάλαιο 8

# Βιβλιογραφία


[1] https://el.wikipedia.org/wiki/Διαδικτυακή_εφαρμογή

[2] https://el.wikipedia.org/wiki/JavaScript

[3] https://en.wikipedia.org/wiki/Web_framework

[4] https://en.wikipedia.org/wiki/Model-view-controller

[5] https://stackoverflow.com/questions/40900741/mvc-variables-in-model-or-controller

[6] https://en.wikipedia.org/wiki/Model-view-viewmodel

[7] http://www.c-sharpcorner.com/UploadFile/3789b7/xamarin-guide-9-use-mvvm-pattern/

[8] https://en.wikipedia.org/wiki/Model-view-presenter

[9] http://www.gwtproject.org/articles/testing_methodologies_using_gwt.html, CC BY-SA 3.0, https://commons.wikimedia.org/w/index.php?curid=34271902

[10] https://plus.google.com/+IgorMinar/posts/DRUAkZmXjNV

[11] https://en.wikipedia.org/wiki/Web_browser_engine

[12] https://www.pathinteractive.com/blog/design-development/rendering-a-webpage-with-google-webmaster-tools/

[13] https://en.wikipedia.org/wiki/Data_binding

[14] https://en.wikipedia.org/wiki/UI_data_binding

[15] http://www.c-sharpcorner.com/UploadFile/ff2f08/data-binding-in-angularjs/

[16] https://en.wikipedia.org/wiki/Backbone.js

[17] https://en.wikipedia.org/wiki/AngularJS

[18] https://en.wikipedia.org/wiki/Ember.js

[19] http://voidcanvas.com/why-angularjs-is-generally-better-than-emberjs-and-backbonejs/

[20] http://voidcanvas.com/emberjs-vs-angularjs-performance-testing/

[21] http://jsfiddle.net/jashkenas/CGSd5/

[22] http://jsfiddle.net/mhevery/vYknU/23/

[23] https://developers.google.com/web/tools/chrome-devtools/evaluate-performance/

[24] https://developers.google.com/web/tools/chrome-devtools/rendering-tools/