Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής
Πρόγραμμα Μεταπτυχιακών Σπουδών
«Πληροφορική»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Εφαρμογή αξιολόγησης διαδικτυακών πόρων με λειτουργικότητα ειδοποιήσεων σε πραγματικό χρόνο**
Ονοματεπώνυμο Φοιτητή	**Χαράλαμπος Μπουζόπουλος**
Πατρώνυμο	**Ανδροκλής**
Αριθμός Μητρώου	**ΜΠΠΛ 12041**
Επιβλέπων	**Χρήστος Δουληγέρης, Καθηγητής**
Συνεπιβλέπων	**Δρ. Βασίλειος Μενεκλής**

Ημερομηνία Παράδοσης **Ιούνιος 2016**

Τριμελής Εξεταστική Επιτροπή

Χρήστος Δουληγέρης Καθηγητής	Δημήτριος Βέργαδος Αναπληρωτής Καθηγητής	Χαράλαμπος Κωνσταντόπουλος Επίκουρος Καθηγητής

Περίληψη

Στις μέρες μας το διαδίκτυο αποτελεί αναπόσπαστο κομμάτι της καθημερινότητας ολοένα και περισσότερων χρηστών. Καθοριστικό παράγοντα στην τάση αυτή αποτελούν οι εφαρμογές κοινωνικής δικτύωσης οι οποίες εντυπωσιάζουν τους χρήστες με τις λειτουργίες πραγματικού χρόνου που παρέχουν.

Ορισμένες τεχνολογικές ανάγκες που διέπουν τα σύγχρονα κοινωνικά δίκτυα είναι η αποθήκευση μεγάλου όγκου δεδομένων, η γρήγορη επεξεργασία δεδομένων και η αποστολή της πληροφορίας στους χρήστες σε πραγματικό χρόνο. Στην παρούσα διπλωματική εργασία εκμεταλλευόμενοι δημοφιλή πρότυπα σχεδίασης, πλαίσια εργασίας (Laravel) και βάσεις δεδομένων (Redis) παρουσιάζεται ένα κοινωνικό δίκτυο με χαρακτηριστικά εμπνευσμένα από υπάρχοντα δίκτυα (Facebook, Twitter).

Η εφαρμογή κοινωνικής δικτύωσης που αναπτύχθηκε αποτελείται από χρήστες, προφίλ χρηστών, σχέσεις μεταξύ χρηστών και βαθμολογίες για πόρους του διαδικτύου. Η υλοποίηση είναι βασισμένη στην αρχιτεκτονική MVC ενώ παράλληλα χρησιμοποιούνται διάφορες πλατφόρμες και τεχνολογίες (Node.js, Ajax, Websockets) για την παροχή ενημερώσεων στους χρήστες σε πραγματικό χρόνο.

Abstract

Nowadays, the Internet is an integral part of the everyday lives for an increasing number of users. A key factor to this trend are the social networking applications which astound users with the real time features they provide.

Modern social netrowking applications have many technical requirements some of which include the storage of large amount of data, the fast data processing and the sending of information to users in real time. This thesis by leveraging popular design patterns, frameworks (Laravel) and databases (Redis) presents a social networking application with features inspired by famous applications (Facebook, Twitter).

The social networking application developed in this thesis consists of users, user profiles, relationships between users and rankings of internet resources. The application was implemented using the principles of the MVC architecture and of modern technologies (Node.js, Ajax, Websockets) to provide updates to users in real time.

Περιεχόμενα

Εισαγωγή

Επικοινωνία, ψυχαγωγία, γνωριμίες, διαφήμιση, δημοσίευση περιεχομένου, είναι βασικά θέματα της καθημερινότητας τα οποία όπως αναφέρει το DigitalTrends (2014) έχουν υποστεί μεγάλη βελτίωση από την χρήση των κοινωνικών δικτύων. Ιστότοποι όπως το Facebook, το Twitter , το Linkedin, το Instagram και το myspace είναι μερικοί από τους πολλούς που έφεραν μεγάλες αλλαγές στον τρόπο που χρησιμοποιούμε το διαδίκτυο Η εξέλιξη και η αξιοποίηση των σύγχρονων τεχνολογιών του διαδικτύου κατέστησαν δυνατή την αλληλεπίδραση μεταξύ χρηστών σε πραγματικό χρόνο. Δόθηκε η δυνατότητα στους χρήστες να αποκτήσουν ενεργό και κυρίως άμεσο ρόλο στη διαμόρφωση και στη διάδοση του περιεχομένου των ιστοτόπων. Τίποτα όμως από αυτά δεν θα ήταν δυνατό χωρίς τους ιστότοπους πραγματικού χρόνου.

Τι είναι οι ιστότοποι πραγματικού χρόνου

Όλοι οι ιστότοποι του διαδικτύου χωρίζονται σε δύο κατηγορίες, στους στατικούς και στους δυναμικούς. Οι στατικοί ιστότοποι είναι οι πρώτοι που εμφανίστηκαν και αποτελούνται από ιστοσελίδες των οποίων το περιεχόμενο παραμένει αμετάβλητο για όλους τους επισκέπτες και μπορεί να τροποποιηθεί μόνο μετά από παρέμβαση κάποιου αρμόδιου διαχειριστή. Σαν αποτέλεσμα οι στατικές ιστοσελίδες είναι εύκολες, γρήγορες και οικονομικές στη δημιουργία τους ωστόσο απαιτούν προγραμματιστικές γνώσεις για να μπορέσουν να ανανεωθούν. Επίσης, οι στατικές ιστοσελίδες δεν παρέχουν φιλικό και χρήσιμο περιβάλλον για το χρήστη και το περιεχόμενο τους μπορεί να θεωρηθεί πολύ γρήγορα ξεπερασμένο. Η δεύτερη και πλέον ευρέως διαδεδομένη κατηγορία ιστοτόπων είναι οι δυναμικοί ιστότοποι.

Οι ιστότοποι αυτοί είναι εξέλιξη των στατικών ιστοτόπων και αποτελούνται από ιστοσελίδες των οποίων το περιεχόμενο μπορεί να ανανεωθεί εύκολα και γρήγορα. Μία αλλαγή στο περιεχόμενο μπορεί να είναι αποτέλεσμα κάποιας ενέργειας του χρήστη όπως η συμπλήρωση ενός πεδίου κειμένου, η επιλογή τιμής από μία λίστα, το πάτημα ενός κουμπιού ή ενός συνδέσμου. Επίσης το περιεχόμενο μπορεί να τροποποιηθεί αυτόματα κατά την διάρκεια φόρτωσης της ιστοσελίδας ανάλογα με την χώρα προέλευσης του επισκέπτη, την ώρα της επίσκεψης και γενικά διάφορους μεταβλητούς παράγοντες. Η λειτουργικότητα που επιθυμούν να έχουν οι σημερινοί ιστότοποι σε συνδυασμό με τις ολοένα και αυξανόμενες ανάγκες των χρηστών του διαδικτύου έκαναν τους δυναμικούς ιστοτόπους να αποτελούν την μοναδική επιλογή για την κάλυψη των απαιτήσεων αυτών.

Οι ιστότοποι πραγματικού χρόνου ανήκουν στους δυναμικούς ιστοτόπους. Το χαρακτηριστικό που τους κάνει να έχουν μεγάλη σημασία είναι πως όταν υπάρχει κάτι νέο να προβληθεί τότε το περιεχόμενο της ιστοσελίδας ανανεώνεται αυτόματα χωρίς να χρειάζεται να πραγματοποιηθεί κάποια ενέργεια από τον χρήστη ή να γίνει ανανέωση ολόκληρης της ιστοσελίδας. Όλοι θα έχουμε παρατηρήσει την αυτόματη λήψη ειδοποιήσεων και μηνυμάτων στο Facebook και στο Twitter ακόμα και όταν έχουμε αφήσει την καρτέλα στον φυλλομετρητή αδρανή. Επίσης, σύμφωνα με τον Huber (2012), οι ιστότοποι πραγματικού χρόνου παρέχουν πολλά πλεονεκτήματα και στις επιχειρήσεις. Πιο συγκεκριμένα οι εργαζόμενοι έχουν την δυνατότητα χρήσης συνεργατικών εργαλείων, όπως φόρουμ και πλατφορμών επικοινωνίας, για την άμεση επικοινωνία μεταξύ τους, μπορούν να εργαστούν ταυτόχρονα σε διάφορα έγγραφα καθώς και να αλληλεπιδρούν με τους πελάτες με σκοπό την προώθηση προϊόντων και ιδεών ή την απάντηση σε κάποιο ερώτημα ή σχόλιο. Πέραν από την βελτίωσης τη επικοινωνίας και της παραγωγικότητας μέσα σε μία επιχείρηση ευνοείται ακόμα η αποφυγή αλληλοκάλυψης εργασιών, η βελτίωση του χρόνου λήψης των απόψεων και των αξιολογήσεων (feedback) των χρηστών και γενικά ενισχύεται η συνολική εμπειρία που απολαμβάνει ένας πελάτης. Αυτή η τεχνολογία σε συνδυασμό με την ανάγκη του ανθρώπου για παρακολούθηση των γεγονότων σε πραγματικό χρόνο είναι που πυροδότησαν την μεγάλη απήχηση των κοινωνικών δικτύων.

Ορισμός και χαρακτηριστηκά κοινωνικών δικτύων

Με τον όρο ιστότοπος κοινωνικής δικτύωσης (social network sites, SNSs) χαρακτηρίζεται σύμφωνα με τους Boyd και Ellison (2007) μία διαδικτυακή εφαρμογή που παρέχει στον χρήστη την δυνατότητα να φτιάξει ένα προσωπικό προφίλ, να δημιουργήσει μια σχέση με κάποιον άλλο χρήστη και να αυξήσει το δίκτυο των επαφών του με άτομα που ήδη γνωρίζει ή που γνώρισε αξιοποιώντας τις λειτουργίες του ιστοτόπου.

Γενικά υπάρχουν δύο μοντέλα σχέσεων σε ό,τι αφορά στα κοινωνικά δίκτυα, το συμμετρικό και το ασύμμετρο. Στο συμμετρικό μοντέλο για να δημιουργηθεί μία σχέση μεταξύ δύο χρηστών πρέπει και οι δύο να την επιθυμούν. Αποτελεί μία αμφίδρομη σχέση για την διατήρηση της οποίας είναι απαραίτητη η συμφωνία και των δύο συμβαλλόμενων μερών. Βασικό παράδειγμα τέτοιων σχέσεων μεταξύ χρηστών είναι οι φιλίες στο Facebook, MySpace, Blackplanet και MiGente. Από την άλλη στο ασύμμετρο μοντέλο η σχέση μεταξύ δύο χρηστών είναι μίας κατεύθυνσης και μπορεί να είναι ή να μην είναι αμοιβαία. Για να δημιουργηθεί μια σχέση στο μοντέλο αυτό δεν είναι απαραίτητη η αποδοχή από τον δεύτερο χρήστη. Παράδειγμα τέτοιων σχέσεων είναι η κατάσταση ακόλουθου (follower) και αυτού που ακολουθείται (following) που συμβαίνει στο Twitter, στο Pinterest και στους κύκλους επαφών του Google Plus.

Η αρχική λειτουργία των κοινωνικών δικτύων σύμφωνα με τον Chapman (2009) ήταν η γνωριμία μεταξύ των χρηστών, η δημιουργία προσωπικών σχέσεων και η αλληλεπίδραση μεταξύ τους μέσα από την ανταλλαγή μηνυμάτων.. Στην συνέχεια με την συνεχή αύξηση του πλήθους των κοινωνικών δικτύων προστέθηκαν νέοι τρόποι αλληλεπίδρασης όπως το ανέβασμα ψηφιακού υλικού (photobucket, flicker και youtube), η δημοσίευση περιεχομένου (digg και reddit) και ο σχολιασμός αυτών. Ωστόσο η μεγάλη απήχηση που βρήκαν σε συνδυασμό με τη ραγδαία αύξηση των χρηστών έκαναν τα κοινωνικά δίκτυα να αποκτήσουν εμπορική σημασία. Η διαφήμιση, η προβολή και η προώθηση συγκεκριμένου περιεχομένου έγιναν αναπόσπαστο κομμάτι αυτών. Η μεγάλη ζήτηση ωστόσο οδήγησε και σε μεγάλη προσφορά με αποτέλεσμα να υπάρχει μια πολύ μεγάλη ποικιλία κοινωνικών δικτύων καθένα από τα οποία να εστιάζει σε ένα συγκεκριμένο ρόλο. Η εξειδίκευση σε μία συγκεκριμένη λειτουργία είναι αυτό που διαφοροποιεί πλέον τα κοινωνικά δίκτυα μεταξύ τους.

Ταυτότητα της εφαρμογής

Η συγκεκριμένη μεταπτυχιακή διατριβή ασχολείται με την ανάλυση, το σχεδιασμό και την υλοποίηση ενός κοινωνικού δικτύου. Βασικοί παράγοντες ήταν η αυτοματοποίηση και απλοποίηση των διαδικασιών και των ενεργειών που χρειάζονται για να ολοκληρωθεί μία λειτουργία. Μέσα από τη χρήση σύγχρονων τεχνολογιών και μεθόδων ανάπτυξης διαδικτυακών εφαρμογών υλοποιήθηκε ένα κοινωνικό δίκτυο με δυνατότητα φιλοξενίας πολλών χρηστών ταυτόχρονα και διαχείρισης μεγάλου όγκου δεδομένων. Η εφαρμογή διαθέτει όλα τα χαρακτηριστικά που ορίζουν ένα κοινωνικό δίκτυο, όλες οι βασικές λειτουργίες γίνονται σε πραγματικό χρόνο και η δημιουργία μίας σχέσης μεταξύ δύο χρηστών ακολουθεί το ασύμμετρο μοντέλο.

Σκοπός της εφαρμογής είναι μέσα από μία απλή και γρήγορη διαδικασία να βαθμολογείται οποιαδήποτε ιστοσελίδα του διαδικτύου. Ο χρήστης μέσα από μια φιλική προς αυτόν διεπαφή έχει τη δυνατότητα να εγγραφεί στο σύστημα και να επεξεργαστεί το προφίλ του. Ανεξάρτητα όμως από την πληρότητα του προφίλ από την στιγμή που θα γίνει επιτυχής είσοδος στην εφαρμογή ο χρήστης μπορεί να ξεκινήσει την διαδικασία βαθμολόγησης ιστοσελίδων του διαδικτύου παρέχοντας απλά στο σύστημα τον αντίστοιχο εξωτερικό σύνδεσμο (URL). Επίσης, η βαθμολογία μπορεί να συνοδεύεται από μια προαιρετική κριτική του χρήστη για την συγκεκριμένη ιστοσελίδα.

Η πρόσφατη δραστηριότητα κάθε χρήστη καθώς και αυτή του κύκλου των επαφών του είναι άμεσα προσβάσιμη με την μορφή χρονοδιαγράμματος (timeline) στην προσωπική του αρχική σελίδα. Το χρονοδιάγραμμα αποτελείται από μία ακολουθία χρονικά διατεταγμένων βαθμολογιών παρέχοντας έτσι εμπειρία παρόμοια με αυτή που προσφέρουν όλα τα σύγχρονα κοινωνικά δίκτυα. Επίσης κατάλληλες ειδοποιήσεις (notifications) εμφανίζονται σε προκαθορισμένα σημεία όταν συμβούν κάποια συγκεκριμένα γεγονότα. Οι ειδοποιήσεις και η ανανέωση του χρονοδιαγράμματος γίνονται αυτόματα και σε πραγματικό χρόνο.

Στόχος της εφαρμογής δεν ήταν η γνωριμία και η επικοινωνία μεταξύ των χρηστών αλλά η εύκολη και γρήγορη καταχώρηση μίας βαθμολογίας για μία ιστοσελίδα. Για το λόγο αυτό δυνατότητες όπως η αποστολή μηνύματος σε άλλο χρήστη και σχολιασμός της δραστηριότητας κάποιου δεν υλοποιηθηκαν στην πρώτη έκδοση της εφαρμογής. Αυτό όμως που θα παρατηρήσει κανείς από την πρώτη του επίσκεψη είναι ένα φιλικό και εύχρηστο περιβάλλον με έμφαση στην εύκολη καταχώρηση βαθμολογιών και κριτικών για ιστοσελίδες. Επίσης η διαχείριση υπαρχόντων βαθμολογιών, η προβολή λιστών με βάση κάποια κριτήρια και η αναζήτηση περιεχομένου είναι διαδικασίες γρήγορες και απλές.

Η εφαρμογή υλοποιήθηκε υπό την συνεργασία τριών φοιτητών. Για το λόγο αυτό χρησιμοποιήθηκαν συνεργαστικές πλατφόρμες διαχείρισης κώδικα, παρακολούθησης ζητημάτων και οργάνωσης εργασιών. Επίσης η συνεργασία αυτή επέτρεψε να δοθεί ιδιαίτερη έμφαση σε θέματα λειτουργικότητας, ευχρηστίας, ταχύτητας αποθήκευσης και ανάκτησης δεδομένων, ασφάλειας και αξιοπιστίας. Οι διπλωματικές εργασίες που σχετίζονται με την εφαρμογή είναι η παρούσα, η εργασία με τίτλο «Εφαρμογή αξιολόγησης διαδικτυακών πόρων με χαρακτηριστικά κοινωνικού δικτύου» καθώς επίσης και η εργασία με τίτλο «Εφαρμογή αξιολόγησης διαδικτυακών πόρων με λειτουργικότητα αποθήκευσης αρχείων στο νέφος».

Δομή του κειμένου

Στο κεφάλαιο αυτό, το οποίο αποτελεί και το πρώτο της μεταπτυχιακής διατριβής, παρουσιάστηκε ο όρος των κοινωνικών δικτύων και των ιστότοπων πραγματικού χρόνου. Επίσης είδαμε το σκοπό, το στόχο και τα χαρακτηριστικά της διαδικτυακής εφαρμογής που κατασκευάσαμε.

Στο δεύτερο κεφάλαιο παρουσιάζονται βιβλιογραφικές πηγές σχετικές με τα θέματα που απασχολούν τη συγκεκριμένη μεταπτυχιακή διατριβή, οι οποιές και αποτέλεσαν πηγή έμπνευσης και επιρροής για τη συγκεκριμένη διαδικτυακή εφαρμογή

Στο τρίτο κεφάλαιο παρουσιάζονται όλες οι τεχνολογίες που χρησιμοποιήθηκαν και αναφέρεται που χρησιμοποιήθηκε η κάθε μία από αυτές κατά την διαδικασία υλοποίησης της εφαρμογής. Οι τεχνολογίες αυτές είναι η μη σχεσιακή βάση δεδομένων Redis, το κατανεμημένο σύστημα ελέγχου ροής των εργασιών Git, η μέθοδος Ajax, η πλατφόρμα Node.js με διάφορες υποστηρικτικές βιβλιοθήκες και τα πλαίσια εργασίας (framework) Codeigniter και Laravel. Επίσης θα αναφερθούν οι λόγοι που συντέλεσαν για την μετάβαση από το Codeigniter framework στο Laravel framework.

Στο τέταρτο κεφάλαιο παρουσιάζεται η αρχιτεκτονική του συστήματος και η λειτουργικότητα κάθε τμήματος που το αποτελεί. Επίσης, με την χρήση της γλώσσας μοντελοποίησης UML παρουσιάζεται η διαδικασία σχεδίασης και ανάπτυξης της εφαρμογής μέσα από περιπτώσεις χρήσης, διαγράμματα ακολουθίας και δραστηριοτήτων. Επίσης, γίνεται σύγκριση των δύο διαφορετικών μοντέλων διάταξης δεδομένων που σχεδιάστηκαν και αναφέρονται οι λόγοι που συντέλεσαν στην τελική επιλογή.

Στο πέμπτο κεφάλαιο θα παρουσιαστούν σημεία που δεν υλοποιήθηκαν σύμφωνα με παρόμοια χαρακτηριστικά άλλων κοινωνικών δικτύων, για να μην υπάρχει εμβάθυνση πέρα από το θέμα που αφορά την εφαρμογή, καθώς και προτάσεις για μελλοντικές προσθήκες και βελτιώσεις. Επίσης θα αναφερθούν τα συμπεράσματα μας και όλα όσα αποκομίσαμε κατά τη διάρκεια εκπόνησης της παρούσας μεταπτυχιακής διατριβής.

Σχετική Βιβλιογραφία

Στο κεφάλαιο αυτό παρουσιάζονται εργασίες που μελετήθηκαν κατά την διάρκεια εκπόνησης της παρούσας διπλωματικής εργασίας. Το αντικείμενο τους είναι παρόμοιο με διάφορες λειτουργίες που μας απασχόλησαν και θέλαμε να υοθετήσουμε. Για τον λόγο αυτό αποτέλεσαν τόσο πηγή έμπνευσης όσο και σημεία αναφοράς.

Βιβλιογραφία

Progit

Chacon, S., Straub, B.

Τα κατανεμημένα συστήματα ελέγχου ροής των εργασιών και συγκεκριμένα το Git, είναι τα πιο διαδεδομένα εργαλεία διαχείρισης εκδόσεων των αρχείων σε ένα έργο. Στην εργασία αυτή (Chacon και Straub, 2014) παρουσιάζεται αρχικά τι είναι τα συστήματα αυτά και για ποιο λόγο έχουν τόσο μεγάλη απήχηση. Στην συνέχεια περιγράφεται λεπτομερώς ο τρόπος εγκατάστασης και λειτουργίας του Git, τα μέρη από το οποία αποτελείται καθώς και διάφορες τεχνικές και εντολές απαραίτητες για κάθε περίπτωση έργου. Ακόμα περιγράφεται ο τρόπος αξιοποίησης του Git μέσα από διαδικτυακούς ιστοτόπους όπως το GitHub ενώ στο τέλος παρουσιάζονται σενάρια και παραδείγματα χρήσης του.

Τα αναλυτικά παραδείγματα του συγγράμματος σε συνδυασμό με τα εγχειρίδια και τις λίστες των εντολών βοήθησαν σε μεγάλο βαθμό την ομαλή και αποτελεσματική διαχείριση της ροής των εργασιών με τα υπόλοιπα μέλη της ομάδας.

Data access for high scalable solutions: Using SQL, NoSQL, and Polyglot Persistence

McMurtry, D. et al.

Σύμφωνα με τους συγγραφείς οι ανάγκες μίας σύγχρονης εφαρμογής περιλαμβάνουν την επεξεργασία μεγάλου όγκου δεδομένων. Επίσης, είναι απαραίτητο η αποθήκευση και ανάκτηση των δεδομένων αυτών να γίνεται με τον πιο γρήγορο και αποτελεσματικό τρόπο, οδηγώντας έτσι σε αρκετές περιπτώσεις την μετάβαση από τις σχεσιακές βάσεις δεδομένων (SQL) στις μη (NoSQL). Σε αυτό το σύγγραμμα (McMurtry, D. et al.) παρουσιάζονται τα διάφορα είδη βάσεων δεδομένων, τα κριτήρια για την καταλληλότερη επιλογή ανάλογα με τη φύση της εφαρμογής καθώς και τρόποι σχεδίασης και υλοποίησης κάθε είδους. Στην αρχή αναφέρονται τα προβλήματα που αντιμετωπίζουν οι σχεσιακές βάσεις δεδομένων και πως αυτά θα μπορούσαν να λυθούν μεσα από την χρήσης μίας άλλης βάσης. Αφού γίνει μία αναφορά σε τεχνικές και αρχές σχεδίασης για την βελτιστοποίηση της απόδοσης των σχεσιακών βάσεων ακολουθεί η παρουσίαση και ανάλυση των μη σχεσιακών. Βάσεις δεδομένων μορφής Κλειδιού/Τιμής (Key/Value Data Store), Εγγράφου (Document Database), Στήλης-Οικογένειας(Column-Family Database) και Γραφήματος (Graph Database) είναι οι περιπτώσεις που παρουσιάζονται και αναλύονται στα επόμενα κεφάλαια. Το τελικό συμπέρασμα των συγγραφέων είναι πως η βέλτιστη λύση είναι μια υβριδική υλοποίηση στην οποία η χρήση σχεσιακών και μη βάσεων δεδομένων καλύπτει τις ανάγκες επεκτασιμότητας, διαθεσιμότητας, συνέπειας και υψηλής ταχύτητας ανάκτησης και αποθήκευσης δεδομένων κάθε σύγχρονης εφαρμογής.

Η ανάγκη αποτελεσματικής και γρήγορης διαχείρισης μεγάλου όγκου δεδομένων σε συνδυασμό με την αναφορά του συγγράμματος στη χρήση μη σχεσιακών βάσεων δεδομένων σε μηχανές αναζήτησης και σε εφαρμογές κοινωνικής δικτύωσης για την εξασφάλιση υψηλής απόδοσης επηρέασε την διαδικασία

επιλογής της βάσης που θα χρησιμοποιούταν στην παρούσα εργασία. Επίσης, τέθηκε η βάση για περαιτέρω έρευνα σε θέματα επεκτασιμότητας και καταμερισμού των εργασιών που σύμφωνα με τους συγγραφείς θα πρέπει να ενσωματώνει κάθε σύγχρονη λύση.

Comet and Reverse Ajax, The Next-Generation Ajax 2.0

Crane, D., McCarthy, P.

Ένα από τα βασικά χαρακτηριστικά του πρωτοκόλλου HTTP είναι πως η επικοινωνία μεταξύ πελάτη (client) και εξυπηρετητή (server) ξεκινάει πάντα από την μεριά του πελάτη. Στο σύγγραμμα (Crane και McCarthy, 2008) αυτό αναφέρονται ορισμένες συνηθισμένες περιπτώσεις, όπως η παροχή δεδομένων στους χρήστες, η μεταξύ τους επικοινωνία και συνεργασία καθώς και η παρακολούθηση της προόδου μίας εργασίας, για τις οποίες θα ήταν ιδανικό η επικοινωνία να ξεκινούσε από την μεριά του εξυπηρετητή. Παρουσιάζονται οι μέθοδοι Polling και Piggybacking, με τις οποίες θα ήταν εφικτή η προσομοίωση αυτής της μη συμβατικής επικοινωνίας. Η μέθοδος Polling βασίζεται στην, ανεξάρτητη από τις ενέργειες του χρήστη, αποστολή αιτημάτων σε τακτά χρονικά διαστήματα από τον πελάτη στον εξυπηρετητή σχετικά με την ύπαρξη κάποιας ενημέρωσης. Κάθε ένα από τα αιτήματα αντιστοιχεί και σε μία απάντηση από τον εξυπηρετητή, η οποία συνήθως αποτελείται από την πληροφορία πως δεν υπάρχει καμία ενημέρωση. Η μέθοδος Piggybacking αποτελεί μία βελτίωση της μεθόδου Polling και βασίζεται στην προσάρτηση των αιτημάτων αυτών σε οποιαδήποτε ενέργεια του χρήστη και των απαντήσεων σε οποιαδήποτε απάντηση του εξυπηρετητή. Με τον τρόπο αυτό όταν ο χρήστης στέλνει ένα αίτημα στον εξυπηρετητή αυτό συνοδεύεται από ένα αίτημα σχετικά με την ύπαρξη διαθέσιμης ενημέρωσης και αντίστοιχα η απάντηση του εξυπηρετητή στο αίτημα του χρήστη περιέχει και την απάντηση στο αίτημα αυτό. Ωστόσο επισημαίνεται πως θα υπήρχαν αρκετά μειονεκτήματα με τις μεθόδους αυτές όπως η υπερφόρτωση του εξυπηρετητή από τα συνεχόμενα αιτήματα του πελάτη, σχετικά με την ύπαρξη κάποιας ενημέρωσης, καθώς και ο μικρός όγκος δεδομένων που μπορεί να μεταφερθεί.

Στην συνέχεια παρουσιάζεται το μοντέλο Comet ως τον τρόπο αντιμετώπισης των ζητημάτων αυτών. Το Comet αποτελεί μία συλλογή τεχνικών οι οποίες βασίζονται στις μεθόδους Streaming και Long Polling για την επίτευξη της επικοινωνίας αυτής. Η μέθοδος Streaming χαρακτηρίζεται από την δημιουργία μίας μόνιμης σύνδεσης TCP μεταξύ πελάτη και εξυπηρετητή για την αποστολή αιτημάτων HTTP και απαντήσεων που σχετίζονται με το Comet. Η τεχνική δημιουργίας κρυφών iframe και ενσωμάτωσης ετικετών script για την επεξεργασία των δεδομένων που στέλνει ο εξυπηρετητής βασίζεται στην μέθοδο αυτή. Η μέθοδος Long Polling βασίζεται στην νοοτροπία της μεθόδου Polling με την βασική διαφορά πως ένα αίτημα HTTP παραμένει ενεργό μέχρι να υπάρξει μία ενημέρωση ή μέχρι να ληξει (time-out). Με τον τρόπο αυτό ο εξυπηρετητής απαντάει σε ένα αίτημα μόνο όταν υπάρχει κάποια διαθέσιμη ενημέρωση ενώ σε αντίθετη περίπτωση το αίτημα παραμένει ανοιχτό. Ο πελάτης όταν λάβει την απάντηση στέλνει αμέσως καινούργιο αίτημα και περιμένει μέχρι αυτό να απαντηθεί. Το μοντέλο Comet χρησιμοποιεί το πρωτόκολλο HTTP με αποτέλεσμα να δεσμεύεται από την αδυναμία έναρξης μίας επικοινωνίας από την μεριά του εξυπηρετητή. Με την χρήση των μεθόδων και των τεχνικών που αναφέρθηκαν το Comet προσπαθεί να κρύψει τους περιορισμούς του πρωτοκόλλου και να προσομοιώσει με τον όσο το δυνατόν λιγότερο επιβαρυντικό για τον εξυπηρετητή τρόπο μία αμφίδρομη επικοινωνία. Τέλος παρουσιάζεται ένα μοντέλο μετάδοσης δεδομένων το οποίο ασχολείται με την δημιουργία καναλιών (channels) στα οποία οι πελάτες εγγράφονται (subscribe) και όταν ο εξυπηρετητής θέλει να μεταδώσει πληροφορίες στον πελάτη δημοσιεύει (publish) τα δεδομένα αυτά στο αντίστοιχο κανάλι. Το παραπάνω μοντέλο είναι γνωστό ως Δημοσίευση/ Συνδρομή (Publish – Subscribe) και παρουσιάστηκε μέσα από το πρωτόκολλο Bayeux.

Το μοντέλο αυτό έθεσε την βάση για περαιτέρω έρευνα σχετικά με τεχνολογίες που μπορεί να χρησιμοποιηθούν για την επίτευξη πραγματικά αμφίδρομης και σε πραγματικό χρόνο επικοινωνίας μεταξύ πελάτη και εξυπηρετητή με σκοπό την υλοποίηση του συστήματος ενημερώσεων και ειδοποιήσεων της παρούσας διπλωματικής εργασίας.

Node Up and Running, Scalable Server-Side Code with Javascript

Hugbes-Croucber, T., Wilson, M.

Στο συγκεκριμένο σύγγραμμα (Hugbes-Croucber και Wilson, 2012) παρουσιάζεται τι είναι το Node.js και πώς μπορεί να χρησιμοποιηθεί για την χρήση της προγραμματιστικής γλώσσας Javascript τόσο στον φυλλομετρητή (browser) όσο και στον εξυπηρετητή (server). Στην συνέχεια παρουσιάζονται τα συμβάντα (events), η αρχιτεκτονική επανάληψής τους (event-loop architecture) και το μοντέλο προγραμματισμού βάσει αυτών (event-driven programming model). Μέσα από παραδείγματα εφαρμογών εισόδου-εξόδου δεδομένων αναλύεται η προγραμματιστική διεπαφή του Node.js καθώς και η δυνατότητα αύξησης των λειτουργιών του μέσα από την εγκατάσταση εξωτερικών βιβλιοθηκών. Παρουσιάζεται η βιβλιοθήκη Express.JS η οποία αποτελεί ένα πλαίσιο εργασίας για την ανάπτυξη διαδικτυακών εφαρμογών και η βιβλιοθήκη Socket.IO η οποία είναι απαραίτητη για την αμφίδρομη και σε πραγματικό χρόνο επικοινωνία μεταξύ πελάτη και εξυπηρετητή. Τέλος, ανακαλύπτουμε πως μπορεί το Node.js να υποστηρίξει και να συνδεθεί με διάφορες βάσεις δεδομένων, σχεσιακές και μη, μία από τις οποίες είναι και η Redis.

Ο τρόπος λειτουργίας του Node.js καθώς και οι οδηγίες για την εγκατάσταση και την χρήση των εξωτερικών βιβλιοθηκών βοήθησαν στην κατανόηση των αρχών που διέπουν το προγραμματιστικό μοντέλο βάσει συμβάντων. Επίσης τέθηκε η βάση για περαιτέρω έρευνα πάνω στον τρόπο αποστολής στον φυλλομετρητή δεδομένων σε πραγματικό χρόνο με σκοπό την υλοποίηση ενός συστήματος αυτόματης παροχής ειδοποιήσεων στους χρήστες σχετικά με την πρόσφατη δραστηριότητα του κύκλου των επαφών τους καθώς και ενημερώσεων στο χρονολόγιο τους.

Professional Node.js, Building Javascript Based Scalable Software

Teixeira, P.

Στην αναζήτηση για περισσότερες πληροφορίες σχετικά με το μοντέλο Δημοσίευση/ Συνδρομή και την αμφίδρομη σε πραγματικό χρόνο επικοινωνία μελετήθηκε το συγκεκριμένο σύγγραμα (Teixeira, 2012). Αρχικά για εισαγωγικούς λόγους παρουσιάζεται το Node.js, οι βασικές λειτουργίες του και το σύστημα διεπαφής του. Στην συνέχεια περνάει σε πιο εξειδικευμένα θέματα όπως αυτά του ασύγχρονου προγραμματισμού, κατασκευής διαδικτυακών εφαρμογών, επικοινωνίας σε πραγματικό χρόνο και σύνδεση με βάσεις δεδομένων. Γίνεται εκτενής αναφορά και περιγραφή των εξωτερικών βιβλιοθηκών και πιο συγκεκριμένα του πλαισίου εργασίας Express.Js και του Socket.IO για την δημιουργία υποδοχών (sockets) και την σύνδεση πελατών. Ο τρόπος λειτουργίας της βιβλιοθήκης Socket.IO βασίζεται στο πρωτόκολλο Websocket που παρέχει η HTML5 και σκοπός του είναι η δημιουργία μίας TCP σύνδεση μεταξύ πελάτη και εξυπηρετητή. Για την επίτευξη της σύνδεσης αυτής πραγματοποιείται μία “χειραψία” (WebSocket Handshake) μεταξύ πελάτη και εξυπηρετητή η οποία ξεκινάει σαν ένα συνηθισμένο HTTP αίτημα και στην συνέχεια ζητάει από τον εξυπηρετητή την αναβάθμιση της σύνδεσης σε WebSocket. Με την ολοκλήρωση της χειραψίας η σύνδεση αλλάζει σε κατάσταση ανταλλαγής δεδομένων (data transfer mode) μεταξύ των δύο πλευρών και επιτρέπει την αποστολή μηνυμάτων χωρίς την χρήση επικεφαλίδων HTTP ή άλλων χειραψιών . Με τον τρόπο αυτό επιτυγχάνεται μία αμφίδρομη ταυτόχρονη επικοινωνία κατά την οποία ο πελάτης και ο εξυπηρετητής μπορούν να ανταλλάζουν μηνύματα χωρίς να χρειάζεται να περιμένουν ο ένας τον άλλο. Επίσης αναλύεται μέσα από κατατοπιστικά παραδείγματα ο τρόπος δημιουργίας εφαρμογών με σκοπό την διαχείριση μεγάλου αριθμού χρηστών και επίτευξης αμφίδρομης και σε πραγματικού χρόνο επικοινωνίας μεταξύ πελάτη και εξυπηρετητή. Τέλος γίνεται συνδιασμός των παραπάνω με μία εξωτερική βιβλιοθίκη που επιτρέπει την επικοινωνία με την μη σχεσιακή βάση δεδομένων Redis για πρόσβαση στα δεδομένα της βάσης και χρήσης της σαν κόμβο επικοινωνίας για την αξιοποίηση του μοντέλου Δημοσίευση/ Συνδρομή.

Η παρουσίαση από τον συγγραφέα μίας εφαρμογής η οποία θα χρησιμοποιεί την παλτφόρμα Node.JS για την εξυπηρέτηση αιτημάτων αμφίδρομης επικοινωνίας, την βιβλιοθήκη Socket.IO για την χρήση του πρωτοκόλλου Websocket και την συνεχή σύνδεση του φυλλομετρητή με την πλατφόρμα αυτή και την

βάση δεδομένων Redis σαν σημείο επικοινωνίας της πλατφόρμας με την εφαρμογή αποτέλεσε πηγή έμπνευσης για την αρχιτεκτονική της παρούσας εφαρμογής. Το σύστημα παροχής ειδοποιήσεων και ενημερώσεων που υλοποιήθηκε βασίστηκε στην αξιοποίηση των τεχνολογιών αυτών καθώς και στην εφαρμογή του μοντέλου Δημοσίευση/ Συνδρομή.

Building Scalable Apps with Redis and Node.js

Johanan, J.

Το συγκεκριμένο σύγγραμμα (Johanan, 2014) αποτελεί έναν αναλυτικό οδηγό για την υλοποίηση μίας διαδικτυακής εφαρμογής βασισμένη στην πλατφόρμα Node.js και την βάση δεδομένων Redis. Στην αρχή παρουσιάζεται το πλαίσιο εργασίας Express.Js και αναλύεται πως μπορεί να χρησιμοποιηθεί για να γίνονται στον εξυπηρετητή όλες οι ενέργειες που απαιτούνται μόνο με την χρήση της προγραμματιστικής γλώσσας Javascript. Αμέσως μετά παρουσιάζεται η εξωτερική βιβλιοθήκη Socket.IO και περιγράφεται η διαδικασία σύνδεσης και συνεργασίας με το Express.Js. Στην συνέχεια βλέπουμε τον τρόπο διασύνδεσης της μη σχεσιακής βάσης δεδομένων Redis με το Node.js μέσα από μία εξωτερική βιβλιοθήκη. Επίσης βλέπουμε τις ενέργειες που χρειάζονται για να έχουμε πρόσβαση στα δεδομένα της βάσης καθώς και την διαδικασία για την υλοποίηση της μεθόδου Δημοσίευση/ Συνδρομή. Τέλος ακολουθεί η ανάλυση μεθόδων και τεχνικών για τον αποτελεσματικό προγραμματισμό με την γλώσσα Javascript, την υλοποίηση εφαρμογών με βασικό κριτίριο την επεκτασιμότητα και την αποσφαλμάτωση του κώδικα μας.

Αν και η θεωρητική βάση για την δημιουργία εφαρμογών πραγματικού χρόνου είχε αποκτηθεί ύστερα από την μελέτη προηγούμενων συγγραμμάτων, το συγκεκριμένο σύγγραμμα αποτέλεσε τον οδηγό που ακολουθήθηκε βήμα προς βήμα. Μέσα από τα τεκμηριωμένα και αναλυτικά παραδείγματα μπόρεσε να μετατραπεί η θεωρία σε κώδικα και να πάρει το σύστημα ειδοποιήσεων και ενημερώσεων της παρούσας εφαρμογής την τελική του μορφή.

Website optimization, Speed, Search Engine & Conversion Rate Secrets

King, A. B.

Το συγκεκριμένο σύγγραμμα (King, 2008) αποτελεί έναν αναλυτικό οδηγό για την βελτιστοποίηση των ιστοσελίδων. Αποτελείται από μεθόδους , τεχνικές και πρότυπα σχετικά με την αύξηση της επισκεψιμότητας μίας ιστοσελίδας, την ταχύτερη ανταπόκρισης της και την βελτίωση της εμπειρία από την πλοήγηση σε αυτή. Στην αρχή παρουσιάζεται η διαδικασία βελτιστοποίησης της ιστοσελίδας για τις μηχανές αναζήτησης (SEO) και αναλύονται δύο μελέτες περίπτωσης. Στην συνέχεια παρουσιάζονται τρόποι για την καλύτερη αξιοποίηση της HTML και του CSS με σκοπό την ταχύτερη προβολή της ιστοσελίδας, την ελαχιστοποίηση των αιτημάτων HTTP και την ταχύτερη φόρτωση των γραφικών και των πολυμέσων. Επίσης δίνεται έμφαση σε μεθόδους τόσο για την καλύτερη συγγραφή κώδικα σε Javascript όσο και την δημιουργία πιο ισχυρών AJAX εφαρμογών. Τέλος παρουσιάζεται πως είναι δυνατή η επίτευξη υψηλότερης απόδοσης στον εξυπηρετητή από την βελτιστοποίηση του παραλληλισμού και τον έλεγχο της κρυφής μνήμης (Cache control) και στον πελάτη από την ασύγχρονη φόρτωση των αρχείων της Javascript (Scripts) και την φόρτωση πόρων μόνο όταν ζητηθούν.

Μέσα από τα παραδείγματα και τις μεθόδους που παρουσιάστηκαν ήταν δυνατή η βελτιστοποίηση της ταχύτητας, της αξιοπιστίας και της αποτελεσματικότητας της παρούσας εφαρμογής.

Little redis book

Seguin, K.

Σκοπός του συγγράμματος (Seguin, 2014) αυτού είναι η γνωριμία με την μη σχεσιακή βάση δεδομένων

Redis. Αποτελεί ένα πλήρες εγχειρίδιο για την εγκατάσταση της βάσης σε διάφορα λειτουργικά συστήματα, την αρχικοποίηση της και την σύνδεση σε αυτή. Επίσης δίνει έμφαση στα χαρακτηριστικά των πέντε τύπων δεδομένων που μπορεί να φιλοξενήσει η Redis καθώς και σε αρχιτεκτονικές μοντελοποίησης δεδομένων βάσει των τύπων αυτών. Στην συνέχεια παρουσιάζονται προχωρημένες λειτουργίες που παρέχει η Redis όπως Pipelining, Replication, Transactions, Lua Scripting και άλλα. Τέλος αναφέρονται μέθοδοι για την δημιουργία αντιγράφων ασφαλείας της βάσης καθώς και την δυνατότητα επεκτασιμότητας μέσα από την διανομή της σε παραπάνω από ένα εξυπηρετητές.

Το σύγγραμμα αυτό ήταν ο οδηγός για την δημιουργία της διάταξης δεδομένων (Data layout) που χρησιμοποιήθηκε από την παρούσα εφαρμογή καθώς και για την εγκατάσταση και λειτουργία της Redis. Επίσης τέθηκε η βάση για πιθανή μελλοντική βελτίωση της απόδοσης της βάσης μέσω του διαμοιρασμού των δεδομένων της σε ένα σύνολο διασυνδεδεμένων υπομονάδων (partitioning).

Codebright

Rees, D.

Σκοπός του συγγράμματος (Rees, 2013) αυτού είναι η παρουσίαση του PHP πλαισίου εργασίας Laravel και η δυνατότητα ανάπτυξης διαδικτυακών εφαρμογών από αρχάριους προγραμματιστές. Στην αρχή αναφέρονται οδηγίες σχετικά με την εγκατάσταση και την παραμετροποίηση του πλαισίου εργασίας. Στην συνέχεια παρουσιάζονται και αναλύονται ένα προς ένα τα μέρη που αποτελούν το Laravel. Πιο συγκεκριμένα βλέπουμε την χρήση των Routes, την εφαρμογή διάφορων φίλτρων (Filters) ανάλογα με την περίσταση, την χρήση των Controllers και τον μοντέλων (Models) , την δημιουργία προτύπων (Templates) με το Blade, την δημιουργία και επεξεργασία φορμών και άλλα πολλά. Τέλος γίνεται ο συνδυασμός όσων παρουσιάστηκαν υπό την μορφή παραδείγματος σε μία εφαρμογή με σκοπό την πλήρη κατανόηση της λειτουργικότητας και των δυνατοτήτων του πλαισίου εργασίας.

Το σύγγραμμα αυτό αποτέλεσε τον βασικό οδήγο και εγχειρίδιο για την υλοποίηση της λειτουργικότητας της παρούσας εφαρμογής καθώς και για την αξιοποίηση του συνόλου των δυνατοτήτων που παρέχει το Laravel.

Εργασίες

Design and implementation of a simple Twitter clone using only the Redis key-value store as database and PHP

Sanfilippo, S., 2011

Σε αυτή την εργασία (Sanfilippo, 2011) αναλύεται η διαδικάσια σχεδιασμού και υλοποίησης μια διαδικτυακής εφαρμογής , βασισμένη στην νοοτροπιά του Twitter, με την χρήση της γλώσσας προγραμματισμού PHP και της μη σχεσιακής βάσης δεδομένων Redis. Στην αρχή γίνεται μία μικρή εισαγωγή σχετικά με τις βάσεις δεδομένων της μορφής Κλειδί/Τιμή και παρουσιάζονται οι πέντε διαφορετικοί τύποι δεδομένων που διαθέτει η Redis. Στην συνέχεια παρουσιάζεται η διάταξη των δεδομένων (Data layout) που ακολούθησε ο συγγραφέας καθώς και διάφορες λειτουργίες όπως η πιστοποίηση των χρηστών, η δημιουργία μιας ασύμετρης σχέσης μεταξύ τους, η ενημέρωση της τιμής ενός κλειδιού και άλλα. Τέλος γίνεται αναφορά για την δυνατότητα επεκτασιμότητας της Redis και παρέχονται αντίστιχοι εξωτερικοί σύνδεσοι.

Ο ρόλος της εργασίας αυτής ήταν καθοριστικός για την υλοποίηση της παρούσας διπλωματικής εργασίας. Αποτέλε την βάση πάνω στην οποία χτίστικαν αρκετές δυνατότητες της εφαρμογής και οι οποίες στην συνέχεια εξελίχθηκαν από την χρήση του πλαισίου εργασίας Laravel.

Client-side web perfomance optimizations

Schröter, J., Germany, 2011

Η ταχύτητα μίας διαδικτυακής εφαρμογής έχει μεγάλη σημασία για έναν χρήστη σύμφωνα με τον Schröter και για τον λόγο αυτό είναι σημαντικό να υπάρχουν χαμηλοί χρόνοι απόκρισης. Σύμφωνα με την εργασία αυτή ο μέσος χρόνος φόρτωσης μίας ιστοσελίδας επηρεάζεται 10 – 20 % από τον εξυπηρετητή και 80 – 90 % από τον πελάτη. Για τον λόγο αυτό παρουσιάζονται διάφορες τεχνικές βελτιστοποίησης του χρόνου που χρειάζεται να φορτωθεί μια ιστοσελίδα. Οι τεχνικές αυτές περιλαμβάνουν την ελάττωση των αιτημάτων HTTP, την προσωρινή αποθήκευση αρχείων στον φυλλομετρητή, την ελαχιστοποίηση και συμπίεση των αρχείων CSS και Javascript και την χρήση κατάλληλης μορφής εικόνων. Επίσης στις τεχνικές συμπεριλαμβάνονται η χρήση μορφών δεδομένων με μικρό μέγεθος όπως το JSON, η αφαίρεση αχρείαστων ετικετών μεταδεδομένων (Meta tags), η φόρτωση των εξωτερικών αρχείων Javascript στο τέλος της σελίδας, η φόρτωση ενός πόρου εκ των προτέρων ή όταν ζητηθεί ανάλογα με την περίπτωση και άλλα.

Η εργασία αυτή επηρέασε σε μεγάλο βαθμό την συγγραφή του HTML και CSS κώδικα και πολλές από τις τεχνικές που παρουσιάστηκαν εφαρμόστηκαν στην παρούσα διπλωματική εργασία.

Σύνοψη

Στο κεφάλαιο αυτό παρουσιάστηκαν όλα τα συγγράμματα και οι εργασίες που μελετήθηκαν. Το αντικείμενο τους είναι παρόμοιο με αυτό της εφαρμογής μας και για τον λόγο αυτό αποτέλεσαν τόσο πηγή έμπνευσης όσο και εγχειρίδια υλοποίησης διάφορων λειτουργιών. Για κάθε ένα έγινε μια περιληπτική παρουσίαση και αναφέρθηκε ο τρόπος που συνέβαλε στην εκπόνηση της παρούσας διπλωματικής εργασίας. Με το τέλος του κεφαλαίου αυτού ολοκληρώνεται και το θεωρητικό μέρος της εργασίας.

Στο κεφάλαιο που ακολουθεί θα ασχοληθούμε με τις τεχνολογίες που χρησιμοποιήθηκαν για την υλοποίηση της εφαρμογής και θα δούμε πως αξιοποιήθηκε κάθε μία από αυτές.

Τεχνολογίες

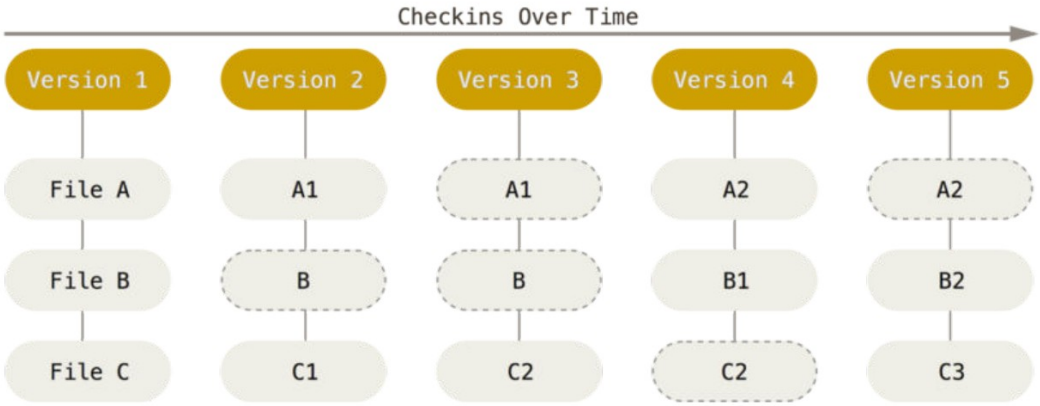
Στο κεφάλαιο αυτό παρουσιάζονται οι τεχνολογίες που χρησιμοποιήθηκαν για την υλοποίηση της παρούσας εφαρμογής. Η επιλογή τους έγινε βάσει των σύγχρονων δυνατοτήτων και της μεγάλης ευελιξίας που παρέχουν για την δημιουργία επεκτάσιμων εφαρμογών. Στην αρχή παρουσιάζεται το κατανεμημένο συστήματα ελέγχου ροής των εργασιών Git και ο ιστότοπος BitBucket. Στην συνέχεια αναλύεται το PHP πλαίσιο εργασίας Laravel μαζί με μία συνοπτική σύγκρισή του με το πλαίσιο εργασίας Codeigniter, το οποίο ήταν και η αρχική επιλογή. Τέλος παρουσιάζονται η μη σχεσιακή βάση δεδομένων Redis, η πλατφόρμα ανάπτυξης Javascript εφαρμογών Node.js ,μαζί με τις υποστηρικτικές βιβλιοθήκες Express.JS, Socket.IO και redis, και η μέθοδος Ajax .

Git

Το Git όπως αναφέρουν οι Chacon και Straub (2014) είναι ένα κατανεμημένο σύστημα ελέγχου ροής των εργασιών και χρησιμοποιείται για την διαχείριση των εκδόσεων των αρχείων σε ένα έργο. Πρόκειται για ένα σύστημα το οποίο όχι μόνο κρατάει το ιστορικό της κατάστασης όλων των αρχειών αλλά παρέχει και την δυνατότητα μετάβασης σε μία προγενέστερη κατάσταση. Για να μπορέσει να γίνει η αρχικοποίηση του Git πρέπει να οριστεί πρώτα ένα χώρος αποθήκευσης (repository) ο οποίος μπορεί να βρίσκεται είτε στον τοπικό υπολογιστή είτε σε έναν απομακρυσμένο εξυπηρετητή (server). Ανεξάρτητα όμως με το που βρίσκεται, ένα αντίγραφο του έργου καθώς και του ιστορικού όλων των αλλαγών που έχουν γίνει θα υπάρχει στον υπολογιστή που έχει πρόσβαση στον χώρο αυτό. Οποιαδήποτε στιγμή κατά την υλοποίηση του έργου ο χρήστης μπορεί να υποβάλει τις αλλαγές του και το Git είναι υπεύθυνο για την ενημέρωση των αρχείων. Η διαδικασία υποβολής των αλλαγών είναι γνωστή ως “commit” και κάθε μία ταυτοποιείται από μία ακολουθία σαράντα δεκαεξαδικών χαρακτήρων γνωστή ως “hash”. Στην περίπτωση που ο χώρος αποθήκευσης φιλοξενείται σε έναν εξυπηρετητή τότε για να την ολοκλήρωση της υποβολής των αλλαγών που πραγματοποιήθηκαν τοπικά απαιτείται ένα ακόμα βήμα , το “σπρώξιμο” (push) τους. Το βήμα αυτό δεν είναι παρά η αποστολή του commit στον εξυπηρετητή. Αντίστοιχα για να είναι πάντα ενημερωμένο το τοπικό αντίγραφο με όλες τις αλλαγές που έχουν γίνει από άλλους χρήστες πριν από κάθε commit απαιτείται το “τράβηγμα” (pull) των ενημερώσεων.

Οι τρεις δυνατές καταστάσεις στις οποίες μπορεί να βρίσκεται ένα αρχείο που διαχειρίζεται το Git είναι τροποποιημένο (modified), έτοιμο για αποθήκευση (staged) και αποθηκευμένο (committed). Αποθηκευμένο θεωρείται ένα αρχείο που έχει αποθηκευτεί επιτυχώς στο παρελθόν στην τοπική βάση δεδομένων του Git μέσα από ένα προγενέστερο commit. Τροποποιημένο είναι ένα αρχείο του οποίου το περιεχόμενο έχει αλλαχθεί και οι αλλαγές αυτές δεν έχουν ακόμα αποθηκευτεί στην βάση δεδομένων. Τέλος έτοιμο για αποθήκευση είναι ένα αρχείο όταν έχει επιλεχθεί ,βάσει της τωρινής του έκδοσης, να συμπεριληφθεί στο επόμενο commit.

Σύμφωνα με τους Chacon και Straub (2014) το Git αντιμετωπίζει τα αρχεία σαν ένα σύνολο στιγμιοτύπων ενός μικρού συστήματος αρχείων. Με τον τρόπο αυτό κάθε φορά που γίνεται η υποβολή ή η αποθήκευση των αλλαγών σε ένα έργο το Git “φωτογραφίζει” την κατάσταση όλων των αρχείων και αποθηκεύει το στιγμιότυπο αυτό. Στην περίπτωση που ένα αρχείο έχει τροποποιηθεί σε σχέση με την προηγούμενη του κατάσταση τότε το αρχείο αυτό αποθηκεύεται. Σε αντίθετη περίπτωση , για λόγους εξοικονόμησης χώρου, αποθηκεύεται ένας σύνδεσμος προς την παλαιότερη έκδοση του.



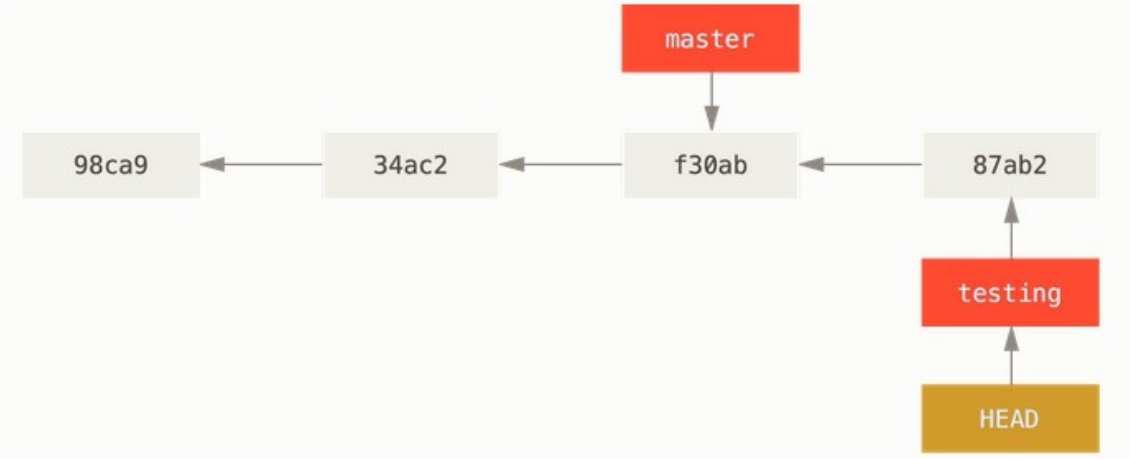
Στιγμιότυπα της κατάστασης των αρχείων κατά την περίοδο υλοποίησης

Όταν δημιουργείται ένα commit το Git αποθηκεύει ένα αντικέιμενο το οποίο περιέχει έναν δείκτη (pointer) για το στιγμιότυπο που αντιπροσωπεύει, το όνομα και το email του δημιουργού του, το κείμενο που γράφτηκε και δείκτες για το ένα ή περισσότερα commit που προηγήθηκαν ακριβώς πριν απο αυτό.

Ένα ακόμα βασικό χαρακτηριστικό του Git είναι τα κλαδιά (branches). Τα κλαδιά αποτελούν ένα τρόπο διαφοροποίησης των εργασιών που λαμβάνουν μέρος κατά την διαδικασία ανάπτυξης μιας εφαρμογής. Κάθε κλαδί δεν είναι παρά ένας δείκτης προς ένα commit. Πιο συγκεκριμένα κατά την αρχικοποίηση του Git το κλαδί που δημιουργείται ονομάζεται “master” και ο δείκτης του κλαδιού αυτού μετακινείται αυτόματα έτσι ώστε να δείχνει πάντα το πιο πρόσφατο commit. Επίσης υπάρχει ένας ειδικός δείκτης ο “Head” που δίχνει σε πιο κλαδί βρισκόμαστε. Κατά την δημιουργία ενός νέου κλαδιού δημιουργείται ένας νέος δείκτης που δείχνει στο ίδιο commit με το master.

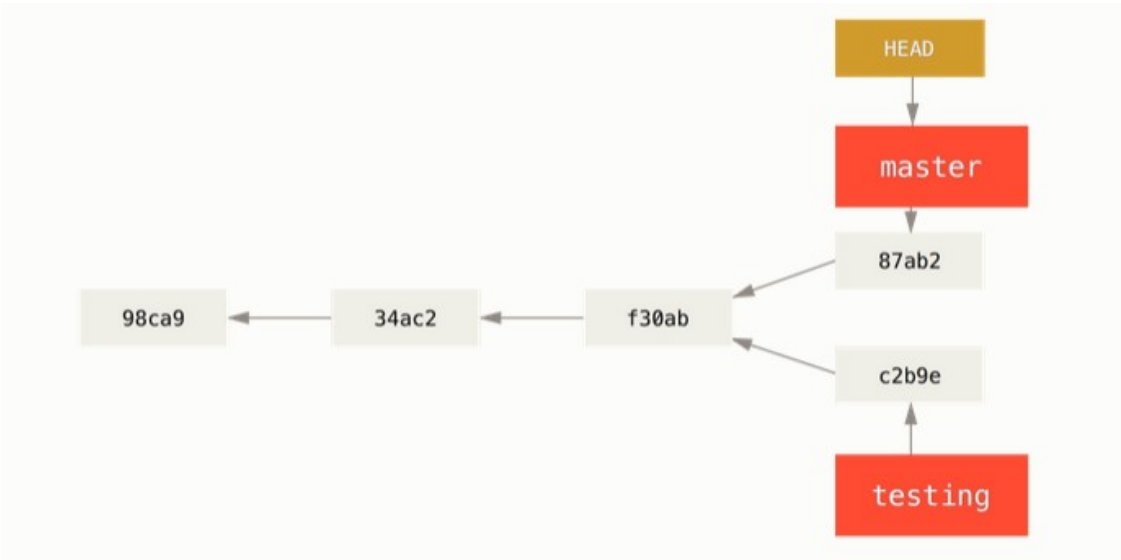
Δημιουργία ενός νέου κλαδιού

Μόλις ολοκληρωθεί η δημιουργία αυτή είναι δυνατή η μετάβαση στο νέο κλαδί, κάνοντας έτσι και τον δείκτη Head να δείχνει εκεί. Αξίζει να αναφερθεί πως κατά την υποβολή ενός νέου commit μόνο ο δείκτης του επιλεγμένου κλαδιού θα επηρεαστεί. Αν λοιπόν επιλεγεί το νέο κλαδί και υποβληθεί από εκεί ένα commit τότε ο δείκτης του θα μετακινηθεί για να δείχνει το commit αυτό ενώ ο δείκτης του master θα μείνει ανεπηρέαστος.



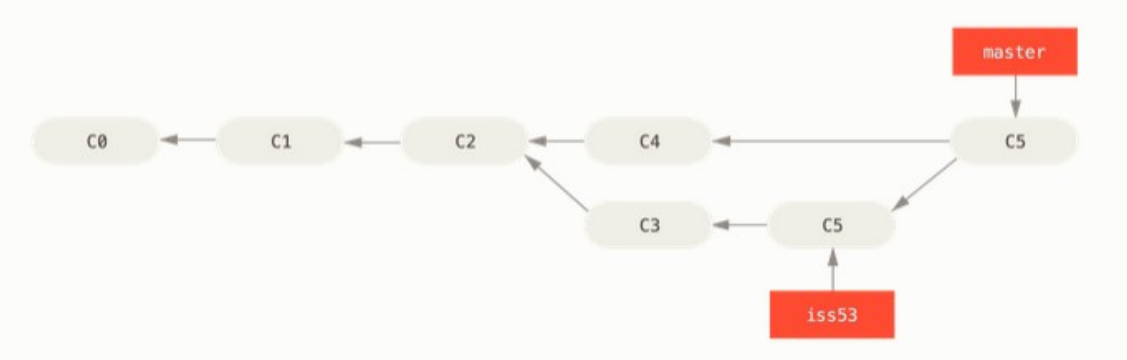
Μετάβαση στο νέο κλαδί και υποβολή νέου commit

Στην συνέχεια η επιστροφή στο κλαδί master και η συνέχιση της υποβολής των commit έχει σαν αποτέλεσμα δύο commit να έχουν τον ίδιο γονέα. Δεν υπάρχει κανένας περιορισμός από το Git για το πόσα κλαδιά μπορούν να υπάρχουν σε ένα έργο άρα και για το πόσα commit μπορούν να έχουν τον ίδιο γονέα.



Δύο ή περισσότερα commit μπορούν να έχουν τον ίδιο γονέα

Τέλος ένα commit μπορεί να έχει παραπάνω από ένα γονέα κάτι που συμβαίνει από την συγχώνευση δύο κλαδιών. Κατά την διαδικασία αυτή πολλές φορές υπάρχουν συγκρούσεις από τις διαφορετικές αλλαγές που έγιναν στο ίδιο αρχείο. Αν μία τέτοια σύγκρουση συμβεί και το Git δεν μπορεί να την επιλύσει αυτόματα τότε πρέπει να γίνει από τους συγγραφείς των commit. Στην περίπτωση όμως που δεν υπάρχουν συγκρούσεις ή που η επίλυση τους μπορεί να γίνει αυτόματα τότε η συγχώνευση είναι επιτυχής και ένα commit προέρχεται από δύο γονείς.



Συγχώνευση του κλαδιού στο master

Πλεονεκτήματα χρήσης του Git;

Όπως έχουμε αναφέρει ένα project μπορεί να βρίσκεται αποθηκευμένο σε έναν απομακρυσμένο εξυπηρετητή και ταυτόχρονα να υπάρχει ένα αντίγραφο του σε κάθε υπολογιστή που έχει πρόσβαση στον χώρο αυτό. Οποιαδήποτε αλλαγή γίνει στο τοπικό αντίγραφο στην συνέχεια υποβάλλεται στον απομακρυσμένο εξυπηρετητή για να ανανεωθεί και εκεί το περιεχόμενο των αρχείων που τροποποιήθηκαν. Η ύπαρξη πολλαπλών αντιγράφων, όσοι δηλαδή και οι χρήστες που έχουν πρόσβαση στον απομακρυσμένο χώρο, επιτρέπει την αντικατάσταση των αρχείων στον εξυπηρετητή σε περίπτωση απώλειας ή αλλοίωσης των δεδομένων.

Ένα ακόμα βασικό πλεονέκτημα του Git είναι η δημιουργία κλαδιών και η δυνατότητα συγχώνευσης τους παρέχοντας έτσι μεγάλη ανεξαρτησία και ευελιξία. Έτσι είναι δυνατή η ύπαρξη κλαδιού με commits που αποτελούν την σταθερή (stable) έκδοση της εφαρμογής , κλαδιού μόνο με λειτουργίες που βρίσκονται υπο κατασκευή και κλαδιού για δοκιμές και πειραματισμό. Ακόμα ένα κλαδί μπορεί να αποτελεί μία αυτοτελή υπό κατασκευή λειτουργία η οποία με την ολοκλήρωση της θα συγχωνευτεί με το υπόλοιπο έργο. Γενικά τα κλαδιά σε συνδυασμό με την εύκολη διαχείριση τους παρέχουν μεγάλη αυτονομία καθώς και την δυνατότητα ύπαρξης ανεξάρτητων μεταξύ τους παραλλαγών του ίδιου αρχείου.

Το Git είναι επίσης γρήγορο και μικρό. Όλες οι λειτουργίες που παρέχει εκτελούνται τοπικά και η μόνη επικοινωνία που χρειάζεται με τον εξυπηρετητή είναι για την αποστολή ή την λήψη των commit. Με την τοπική διεξαγωγή όλων των διαδικασιών επιτρέπεται η συνέχιση των εργασιών ακόμα και όταν δεν υπάρχει κάποια σύνδεση στο δίκτυο ή τον διακομιστή. Επίσης το Git είναι πολύ αποδοτικό στην συμπίεση των δεδομένων με αποτέλεσμα να καταλαμβάνει μικρό χώρο σε κάθε υπολογιστή παρόλο που κρατάει όλες τις εκδόσεις για όλα τα αρχεία καθ'όλη την διάρκεια ανάπτυξης του έργου.

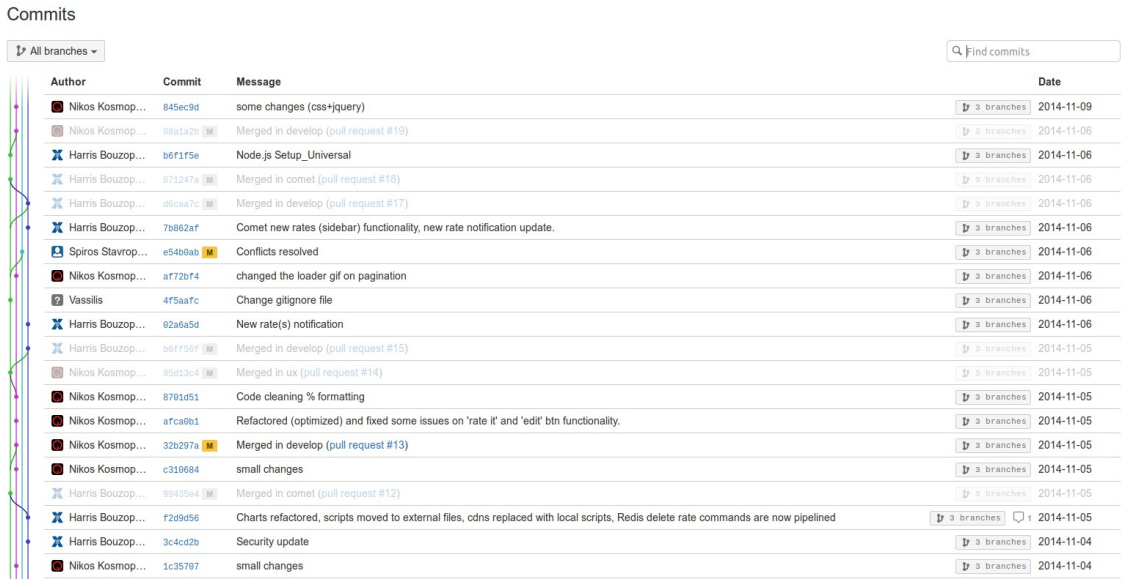
Επιπλέον το Git εξασφαλίζει την κρυπτογραφική ακεραιότητα του κάθε δεδομένου που υποβάλλεται σε ένα έργο. Κάθε commit που υποβάλλεται ταυτοποιείται από ένα hash και οποιαδήποτε αναφορά γίνεται μέσω αυτού. Το hash αυτό υπολογίζεται ανάλογα με το περιεχόμενο του κάθε commit, την ημερομηνία και ώρα υποβολής του και αρκετές άλλες παραμέτρους. Με την χρήση αυτού του μοντέλου δεδομένων καθίσταται αδύνατη η τροποποίηση ενός αρχείου, μίας ημερομηνίας, ενός μηνύματος ή και γενικά οποιουδήποτε δεδομένου έχει ήδη υποβληθεί χωρίς να επηρεαστούν τα πάντα μετά από αυτό.

Όταν ένα αρχείο έχει τροποποιηθεί τότε πρέπει υποχρεωτικά να συμπεριληφθεί στο επόμενο commit έτσι ώστε τα τοπικά αρχεία με τα απομακρυσμένα να γίνουν ίδια. Ωστόσο είναι δυνατή η επιλογή των

αρχείων που θέλουμε να υποβάλουμε αφήνοντας τα υπόλοιπα να συμπεριληφθούν σε ένα μελλοντικό commit. Με τον τρόπο αυτό είναι δυνατή η γρήγορη υποβολή κάποιων αλλαγών στον εξυπηρετητή αφήνοντας τα υπόλοιπα του αρχεία ανεπηρέαστα από τις πιθανές τοπικές αλλαγές. Όταν ένα αρχείο έχει επιλεγεί για να υποβληθεί τότε βρίσκεται στο ενδιάμεσο στάδιο (Staging Area) κατά το οποίο συνεχίζει να είναι δυνατή η επεξεργασία του. Τέλος το git είναι δωρεάν και ελεύθερο για κάθε χρήση.

Bitbucket

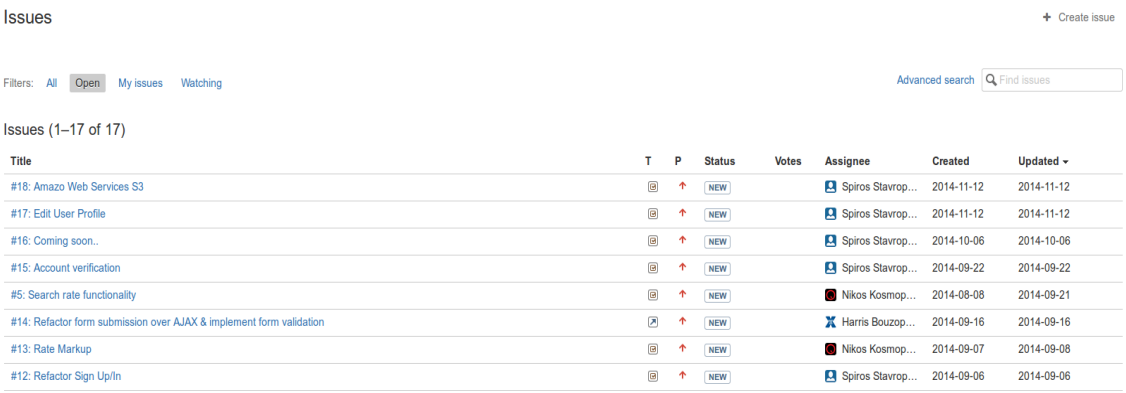
Το Bitbucket είναι ένας διαδικτυακός ιστότοπος φιλοξενίας έργων. Επιτρέπει την δημιουργία ενός απομακρυσμένου χώρου αποθήκευσης (Remote Repository) και την αποθήκευση αρχείων στον χώρο αυτό. Κάθε χρήστης δημιουργεί έναν προσωπικό λογαριασμό από τον οποίο μπορεί να αλληλεπιδρά με όλους του χώρους που είναι μέλος. Όλες οι διαδικασίες για την λήψη και την αποστολή αρχείων από και προς το Bitbucket γίνονται με την χρήση του Git.Ο ιστότοπος αυτός αποτελεί ένα γραφικό περιβάλλον εργασίας για την διεξαγωγή ορισμένων λειτουργιών του Git όπως η δημιουργία και διαχείριση κλαδιών καθώς επίσης και η συγχώνευση τους. Επίσης διαθέτει ένα κεντρικό μενού για την γρήγορη πρόσβαση στις λειτουργίες αυτές καθώς και την διευκόλυνση της πλοήγησης στα διάφορα μέρη που απαρτίζουν το έργο. Ακόμα παρέχει την δυνατότητα προβολής του ιστορικό όλων των commits μαζί με τα μηνύματα που τα συνοδεύουν, την ημερομηνία υποβολή τους και τον συγγραφέα καθώς και μια γραφική αναπαράσταση για το κλαδί στο οποίο ανήκουν. Το hash του κάθε commit αποτελεί σύνδεσμο για μία νέα σελίδα στην οποία παρουσιάζεται το σύνολο των αρχείων που τροποποιήθηκαν, το πλήθος των αλλαγών που έγινε σε κάθε ένα καθώς και ποιες ήταν οι αλλαγές αυτές.



Ιστορικό υποβολής των commit

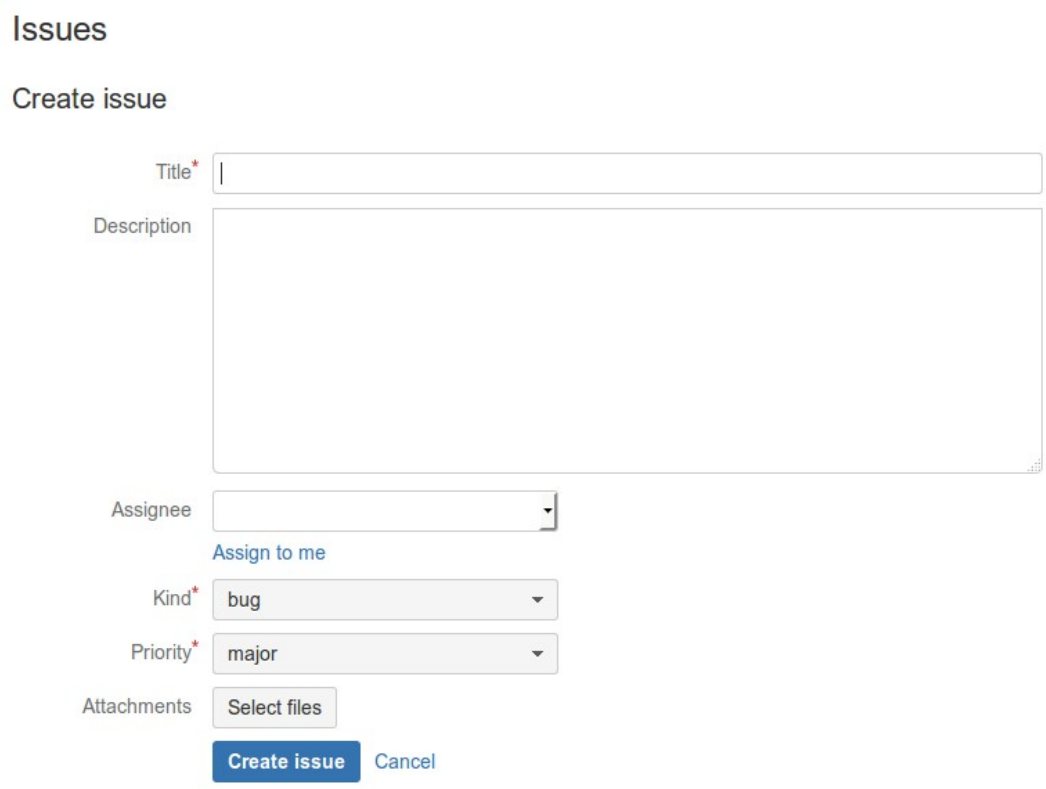
Το Bitbucket πέρα από την φιλοξενία έργων και το γραφικό περιβάλλον λειτουργίας του Git παρέχει στους χρήστες του δύο ακόμα πολύ χρήσιμα εργαλεία, την παρακολούθηση ζητημάτων (Issue Tracking) και τα εγχειρίδια (Wiki).

Τα ζητήματα είναι διάφορα θέματα που προκύπτουν κατά την διάρκεια υλοποίησης ενός έργου και τα οποία πρέπει να επιλυθούν. Η δημιουργία τους, η προβολή τους και η διαχείριση τους μπορεί να γίνει εύκολα και γρήγορα μέσα από την γραφική διεπαφή που προσφέρει το bitbucket. Επίσης είναι δυνατή η εφαρμογή ορισμένων φίλτρων για την κατάλληλη ομαδοποίηση τους.



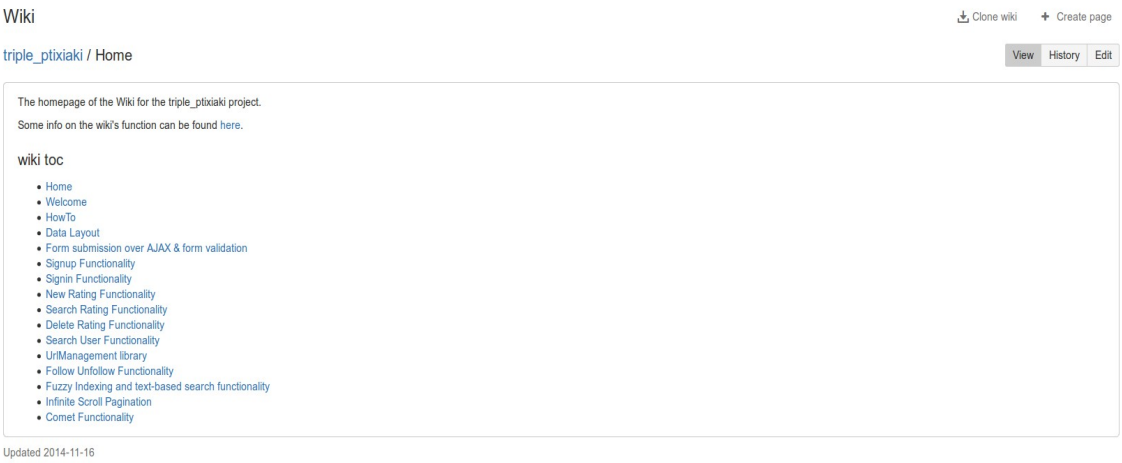
Σελίδα προβολής όλων των ζητημάτων

Ένα ζήτημα αποτελείται από έναν τίτλο, μία περιγραφή, τον χρήστη στον οποίο ανατίθεται, τον τύπο του ζητήματος, τον βαθμό προτεραιότητας του και από πιθανά συνημμένα αρχεία. Ο τύπος ενός ζητήματος μπορεί να είναι σφάλμα (Bug), πρόταση βελτιστοποίησης (Enchancement), πρόταση νέας ιδέας (Proposal) ή μία εργασία (Task). Η προτεραιότητα του μπορεί να είναι μηδαμινή (Trivial), μικρή (Minor), μεγάλη (Major), κρίσιμη (Critical) ή να αποτελεί κάποιο εμπόδιο (blocker).



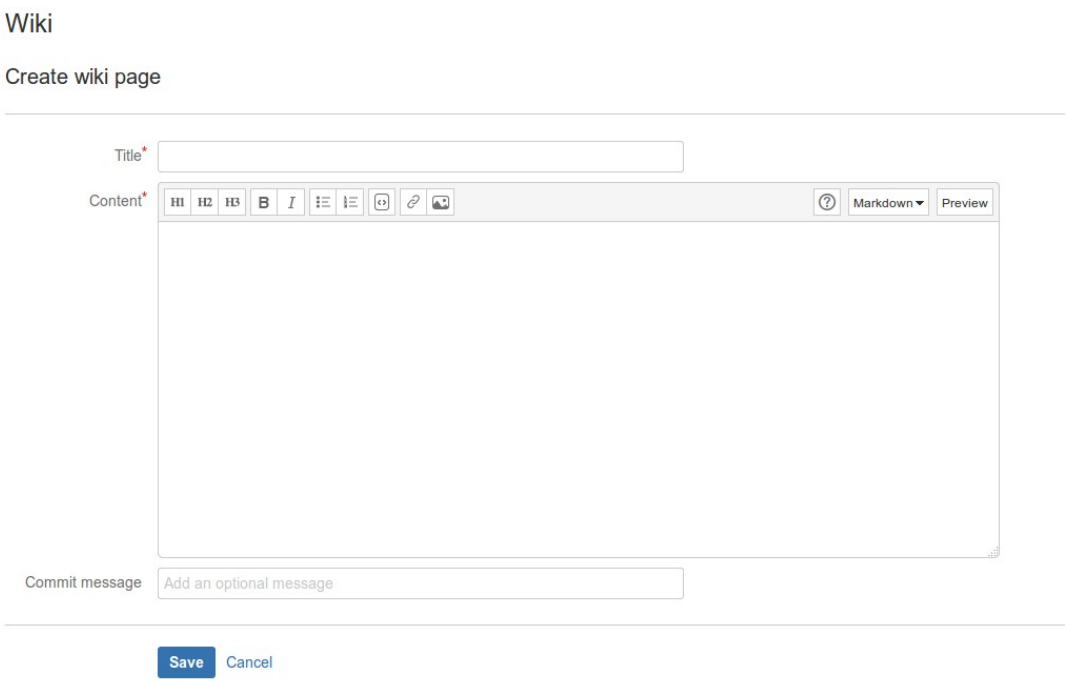
Δημιουργία νέου ζητήματος

Η σελίδα των εγχειριδίων είναι ένα μέρος δημιουργίας και αποθήκευσης εγγράφων. Αποτελεί έναν αυτόνομο χώρο αποθήκευσης και για τον λόγο αυτό όχι μόνο μπορεί να κλονοποιηθεί (clone) και να επεξεργαστεί τοπικά από τον κάθε χρήστη αλλά και κάθε καταχώρηση σε αυτόν αντιμετωπίζεται σαν ένα commit . Η αρχική σελίδα είναι στην ευχέρεια της κάθε ομάδας για το πως θα την διαμορφώσει. Για την παρούσα διπλωματική η σελίδα αυτή αποτελείται από έναν πίνακα περιεχομένων για όλα τα εγχειρίδια που υλοποιήσαμε για κάθε μία από τις λειτουργίες της εφαρμογής.



Αρχική σελίδα των εγχειριδίων

Η δημιουργία και επεξεργασία ενός εγγράφου σε απευθείας σύνδεση με τον εξυπηρετητή είναι αρκετά γρήγορη και εύκολη διαδικασία. Ένα έγγραφο αποτελείται από έναν τίτλο, το περιεχόμενο του και ένα συνοδευτικό μήνυμα για το commit αυτό.



Δημιουργία νέου εγχειριδίου

Ο ιστότοπος Bitbucket αποτέλεσε τον κόμβο επικοινωνίας για τα μέλη της ομάδας που πήραν μέρος στην υλοποίηση της παρούσας εργασίας και συνέβαλε στην μεταξύ τους αρμονική και αποτελεσματική συνεργασία. Πέραν από την παροχή ενός γραφικού περιβάλλοντος για την αξιοποίηση των δυνατοτήτων του Git συνέβαλε καθοριστικά και στον τρόπο οργάνωσης όλων των εργασιών. Με την χρήση των ζητημάτων κατέστη δυνατή η εύκολη και γρήγορη ανάθεση των εργασιών αυτών καθώς και η δυνατότητα παρακολούθησης της πορείας ολοκλήρωσης τους. Για κάθε λειτουργία που διαθέτει η παρούσα εφαρμογή δημιουργήθηκε και ένα εγχειρίδιο. Με τον τρόπο αυτό σε κάθε μέλος της ομάδας ήταν διαθέσιμο ένα σύνολο αναλυτικών οδηγών σχετικά με τον τρόπο υλοποίησης και εφαρμογής της κάθε λειτουργίας.

Πλαίσιο εργασίας Laravel

Για την επιλογή του πλαισίου εργασίας που θα χρησιμοποιόταν για την υλοποίηση της παρούσας εφαρμογής απαιτήθηκε αρκετός χρόνος. Αρχικά είχε επιλεγεί το πλαίσιο εργασίας Codeigniter και βάσει αυτού ξεκίνησε η διαδικασία υλοποίησης. Ωστόσο περαιτέρω έρευνα πάνω σε θέματα που απασχολούν διεθνώς την προγραμματιστική κοινότητα ανάπτυξης διαδικτυακών εφαρμογών αποτέλεσε τον λόγο μετάβασης στο Laravel framework. Οι λόγοι που συντέλεσαν στην μετάβαση αυτή θα παρουσιαστούν συνοπτικά στην συνέχεια.

Τι είναι το Laravel

Το Laravel είναι ένα αντικειμενοστρεφές (Object-Oriented) πλαίσιο εργασίας το οποίο χρησιμοποιείται για την ανάπτυξη διαδικτυακών εφαρμογών. Εκτελείται στην μεριά του εξυπηρετητή και είναι βασισμένο στην προγραμματιστική γλώσσα PHP. Ο ρόλος του είναι η εξυπηρέτηση των HTTP αιτημάτων που θα δέχεται από τους πελάτες βάσει του κώδικα και των λειτουργιών που έχει προγραμματιστεί να εκτελεί.

Η αρχιτεκτονική ανάπτυξης εφαρμογών που παρέχει βασίζεται στο μοντέλο Μοντέλο – Προβολή – Ελεγκτής (Model – View – Controller, MVC) του οποίου βασικό χαρακτηριστικό είναι η τμηματοποίηση της εφαρμογής σε τρία διασυνδεδεμένα τμήματα. Ο διαχωρισμός αυτός γίνεται βάσει των τριών λειτουργιών που απαρτίζουν την αρχιτεκτονική μιας εφαρμογής με αυτές να είναι η πρόσβαση στα δεδομένα και διαχείριση της λογικής της εφαρμογής, δεύτερον η παρουσίαση των δεδομένων και τρίτον η επεξεργασία δεδομένων με σκοπό την παρουσίαση τους ή την αξιοποίηση τους σε περαιτέρω λειτουργίες. Πιο συγκεκριμένα στην αρχιτεκτονική MVC τα μοντέλα είναι η καρδιά της εφαρμογής. Αποτελούν το τμήμα εκείνο στο οποίο βρίσκεται όλη η λογική της εφαρμογής και είναι υπεύθυνα τόσο για την πρόσβαση των δεδομένων όσο και την εκτέλεση του συνόλου των λειτουργιών που έχουν υλοποιηθεί. Από την άλλη οι προβολές αποτελούν τον τρόπο αλληλεπίδρασης του χρήστη με την εφαρμογή. Είναι το τμήμα εκείνο το οποίο ασχολείται με την παρουσίαση και την συλλογή των δεδομένων και για τον λόγο αυτό αποτελείται κυρίως από HTML κώδικα. Τέλος οι ελεγκτές αποτελούν τον ρόλο του ενδιάμεσου για την επικοινωνία της παρουσίασης με το μοντέλο και ασχολούνται με την διασύνδεση και διαχείριση όλων των τμημάτων. Ένας ελεγκτής ασχολείται με την επιλογή της προβολής που πρέπει να εμφανιστεί, του μοντέλου που πρέπει να κληθεί, την παραλαβή και διοχέτευση δεδομένων από και προς τις προβολές και τα μοντέλα , την κατάλληλη επεξεργασία τους καθώς και την μεταβίβαση του ελέγχου και της ροής των εργασιών σε έναν άλλο ελεγκτή.

Το Laravel συνοδεύεται από τεκμηριωμένα εγχειρίδια με πλούσια παραδείγματα καθώς και από την υποστήριξη μεγάλου μέρους της κοινότητας των προγραμματιστών διαδικτυακών εφαρμογών. Σύμφωνα με τον Skvorc (2013) το Laravel αποτελεί το πιο δημοφιλές PHP πλαίσιο εργασίας για το 2014 με ποσοστό 25.85%.

Χαρακτηρηστικά του Laravel

Το σύνολο των δυνατοτήτων που παρέχει το Laravel στους χρήστες είναι τόσο μεγάλο που θα μπορούσε να αποτελεί μία εργασία απο μόνο του. Σκοπός της παρούσας ενότητας είναι η παρουσίαση μερικών ιδιαίτερων λειτουργιών που διαθέτει το Laravel και οι οποίες το κάνουν το πιο δημοφιλές PHP πλαίσιο εργασίας.

Αρχικά αξίζει να σημειωθεί πως υποστηρίζεται η χρήση κατάλληλων εξωτερικών πακέτων (bundles) με σκοπό την αύξηση των δυνατοτήτων του. Τα πακέτα αυτά βρίσκονται στον απομακρυσμένο χώρο αποθήκευσης Packagist.org και η λήψη και εγκατάσταση τους γίνεται εύκολα και γρήγορα με την χρήση του composer.

Ένα ακόμα βασικό χαρακτηριστικό είναι η αυτόματη κλήση κλάσεων (autoloading). Με τον τρόπο αυτό είναι δυνατή η αξιοποίηση των εξωτερικών πακέτων καθώς και η συνεργασία με πλαίσια εργασίας και βιβλιοθήκες που είναι ήδη ενσωματωμένα. Δύο τέτοια εργαλεία που αξίζει να σημειωθούν είναι το πλαίσιο εργασίας PHPUnit που χρησιμοποιείται για την αποσφαλμάτωση του κώδικα και η βιβλιοθήκη Predis που χρησιμοποιείται για την επικοινωνία με την βάση δεδομένων Redis.

Πέρα από τις βιβλιοθήκες για την επικοινωνία με τις μη σχεσιακές βάσεις δεδομένων το Laravel διαθέτει ένα σύστημα χαρτογράφησης σχεσιακών δεδομένων με αντικείμενα (Object-Relational mapping , ORM) το οποίο ονομάζεται Eloquent. Το Eloquent επιτρέπει την εύκολη διαχείριση σχεσιακών βάσεων δεδομένων καθώς κάθε πίνακας της βάσης αντιστοιχείται με ένα μοντέλο μέσω του οποίου είναι δυνατή η αλληλεπίδραση της εφαρμογής με τον πίνακα αυτό.

Επίσης το Laravel παρέχει ελεγκτές αρχιτεκτονικής REST (RESTful controllers) για την κατασκευή μίας προγραμματιστικής διεπαφής για την εφαρμογή (API) με σκοπό την δημιουργία επεκτάσιμων διαδικτυακών εφαρμογών.

Ένα πολύ χρήσιμο χαρακτηριστικό του Laravel είναι το πρότυπο συγγραφής των παρουσιάσεων (Views) Blade. Με το Blade είναι δυνατή η επεξεργασία και διαχείριση των δεδομένων που περνάει ο ελεγκτής στην παρουσίαση καθώς και η ενσωμάτωση τους στον HTML κώδικα χωρίς να είναι απαραίτητη η χρήση των PHP ετικετών (<? php ?>). Το blade δεν επιβραδύνει καθόλου την απόδοση του Laravel καθώς το τελικό αρχείο που δημιουργείται είναι ένα μεταγλωττισμένο αρχείο με τις κατάλληλες PHP ετικέτες το οποίο καλείται όποτε χρειάζεται να προβληθεί η παρουσιάση από την οποία προέρχεται. Επίσης η χρήση όλων των έτοιμων συναρτήσεων της PHP καθώς και δημιουργία προτύπων (

templates) για την επέκταση τους από άλλες παρουσιάσεις γίνεται εύκολα και γρήγορα.

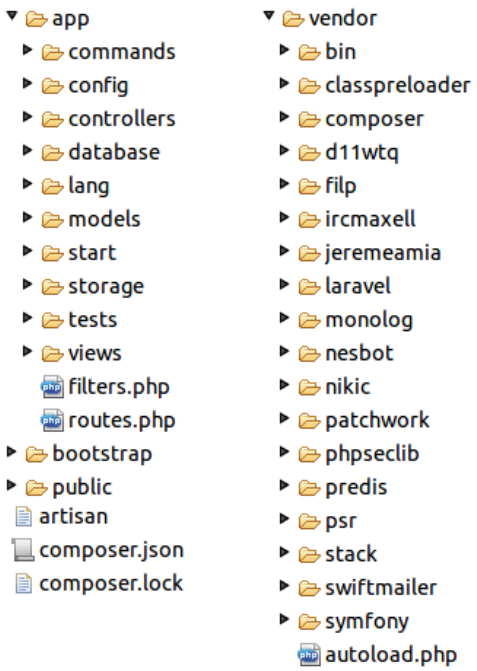
Η σχεδίαση πάνω στην οποία είναι βασισμένη η υλοποίηση του Laravel είναι αυτή της αντιστροφής του ελέγχου (Inversion of control, IoC). Βάσει της σχεδιάσης αυτής τα τμήματα του προγράμματος λαμβάνουν την ροή ελέγχου από μία επαναχρησιμοποιούμενη βιβλιοθήκη, σε αντίθεση με την παραδοσιακή μέθοδο κατά την οποία τα τμήματα του προγράμματος καλούν τις βιβλιοθήκες για την εκτέλεση μίας λειτουργίας. Η σχεδίαση αυτή αποτελεί ένα ισχυρό εργαλείο για την διαχείριση των εξαρτήσεων μεταξύ των κλάσεων και παρέχει μεγάλη ευελιξία καθώς οι εξαρτήσεις αυτές δεν χρειάζεται να είναι γραμμένες με αυστηρό κώδικα (hard coded) αλλά να εγχέονται κατά την διάρκεια εκτέλεσης της εφαρμογής.

Το Laravel διαθέτει επίσης ένα σύστημα ελέγχου της έκδοσης για τις σχεσιακές βάσεις δεδομένων, τα Migrations. Με τον τρόπο αυτό εξασφαλίζεται πως ότι αλλαγές και αν γίνουν στην δομή της βάσης από κάποιον χρήστη, όλοι οι υπόλοιποι χρήστες που συμμετέχουν στην ανάπτυξη του έργου θα είναι πάντα ενήμεροι για τις αλλαγές αυτές.

Τέλος, το Laravel διαθέτει ενσωματωμένο σύστημα σελιδοποίησης (pagination) και ταυτοποίησης των χρηστών (authentication), δυνατότητα δρομολόγησης (routing) με την χρήση ανώνυμων συναρτήσεων (closures) , φιλικά URL και δυνατότητα κρυπτογράφησης των δεδομένων

Δομή εφαρμογής Laravel

Μια εφαρμογή που δημιουργείται με το Laravel έχει πολύ συγκεκριμένη δομή. Αυτή αποτελείται από ένα σύνολο καταλόγων (directories), κάθε ένας από τους οποίους έχει συγκεκριμένη λειτουργία. Οι κατάλογοι αυτοί είναι ο app, ο bootstrap, ο public και ο vendor και κάθε ένας από αυτούς αποτελείται από ένα υποσύνολο καταλόγων και αρχείων. Ο χρήστης έχει την δυνατότητα ανάλογα με τις απαιτήσεις του έργου να προσθέσει κατάλληλα περισσότερους καταλόγους και αρχεία.



Τυπική δομή εφαρμογής Laravel

Από τους τέσσερις αυτούς καταλόγους ο “app” αποτελεί την καρδία της εφαρμογής καθώς εκεί βρίσκεται ο κύριος κώδικας. Μέσα στον κατάλογο αυτό βρίσκονται οι δρομολογητές (routes), τα φίλτρα, ο κατάλογος με τους ελεγκτές, ο κατάλογος με τα μοντέλα, ο κατάλογος με τις παρουσιάσεις και μερικοί

ακόμα κατάλογοι που αξίζει να σημειωθούν. Αυτοί είναι ο κατάλογος “config” ο οποίος περιέχει όλα τα αρχεία με τις παραμέτρους της εφαρμογής (configuration files), ο κατάλογος “database” ο οποίος περιέχει τα migrations, ο κατάλογος “storage” ο οποίος περιέχει τα μεταγλωττισμένα πρότυπα του Blade και αρχεία που δημιουργεί αυτόματα το Laravel και ο κατάλογος tests ο οποίος περιέχει τις πιθανές αυτοματοποιημένες δοκιμές.

Ο δεύτερος κατάλογος που βρίσκεται στο ανώτερο επίπεδο ο “bootstrap” περιέχει αρχεία για την εκκίνηση του πλαισίου εργασίας καθώς και για την παραμετροποίηση της διαδικασίας αυτόματης κλήσης κλάσεων.

Ο κατάλογος “public” περιέχει το αρχείο index.php το οποίο και αποτελεί τον εμπρόσθιο ελεγκτή (front controller) της εφαρμογής. Αυτό είναι το πρώτο αρχείο που εκτελεί ο εξυπηρετητής μετά από κάθε αίτημα που δέχεται από τον φυλλομετρήτη και είναι εκείνο που είναι υπεύθυνο για την εκκίνηση του πλαισίου εργασίας. Επίσης ο κατάλογος αυτός περιέχει αρχεία που πρέπει να είναι άμεσα προσβάσιμα όπως εικόνες, αρχεία CSS και Javascript.

Ο κατάλογος “vendor” περιέχει όλες τις εξωτερικές βιβλιοθήκες οι οποίες είναι απαραίτητες για την αξιοποίηση όλων των λειτουργιών του Laravel. Σε αυτόν τον κατάλογο ανήκει η βιβλιοθήκη Predis για την επικοινωνία με την βάση δεδομένων Redis που αναφέραμε πιο πριν. Το εργαλείο με το οποίο γίνεται η διαχείριση των εξαρτήσεων που έχει το Laravel είναι ο cosmposer.

Πέρα από τους τέσσερις αυτούς καταλόγους βρίσκονται στο ανώτερο επίπεδο υπάρχουν ακόμα δύο αρχεία που πρέπει να αναφερθούν, το “composer.json” και το “composer.lock”. Το πρώτο περιγράφει τις εξαρτήσεις της εφαρμογής, δηλαδή τις εξωτερικές βιβλιοθήκες που ο composer είναι υπεύθυνος να κατεβάσει (Packagist.org) και να εγκαταστήσει. Επίσης περιέχει του καταλόγους των οποίων οι κλάσεις πρέπει να φορτωθούν αυτόματα καθώς και μεταδεδομένα για την όνομα του πλαισίου, την έκδοση του και άλλα. Το αρχείο “composer.lock” περιέχει την λίστα με την ακριβή έκδοση των βιβλιοθηκών που είναι εγκατεστημένες. Είναι το αρχείο εκείνο που θα τρέξει ο composer κατά την εγκατάσταση του έργου σε έναν τοπικό υπολογιστή προκειμένου να δει πια έκδοση της βιβλιοθήκης να εγκαταστήσει.

Σύγγριση Laravel με Codeigniter

Κατά την διαδικασία επιλογής του πλαισίου εργασίας για την υλοποίηση της παρούσας διπλωματικής εργασίας βασικό κριτήριο ήταν η δυνατότητα συγγραφής δομημένου, επεκτάσιμου και συντηρήσιμου κώδικα. Αρχική επιλογή αποτέλεσε το Codeigniter εξαιτίας της μεγάλης δημοτικότητας που ήδη γνωρίζαμε πως έχει, την υποστήριξη του από μία μεγάλη προγραμματιστική κοινότητα και τα πολύ καλά τεκμηριωμένα εγχειρίδια που διαθέτει. Ωστόσο κατά την αναζήτηση σε διάφορους ιστότοπους τρόπους προσθήκης στο Codeigniter ορισμένων λειτουργιών που δεν έχει αλλά θα θέλαμε στην εφαρμογή μας συναντήσαμε το πλαίσιο εργασίας Laravel. Η επιλογή για μετάβαση απο το ένα πλαίσιο στο άλλο δεν ήταν καθόλου δύσκολη καθώς το Laravel υπερτερεί σε πολλά πράγματα. Το πιο βασικό πλεονέκτημα που παρατηρήθηκε είναι πως το Laravel έχει ενσωματωμένες πολλές λειτουργίες τις οποίες για να πετύχει το Codeigniter πρέπει να χρησιμοποιήσει εξωτερικές βιβλιοθήκες.

Πιο συγκεκριμένα το Laravel έχει ενσωματωμένο σύστημα ασφάλειας και πιστοποίησης για τους χρήστες και τα δεδομένα τους σε αντίθεση με το Codeigniter που χρειάζεται μία εξωτερική βιβλιοθήκη, όπως το Ion Auth.Επίσης για την διαδικασία δοκιμών και αποσφαλμάτωσης το Laravel ενσωματώνει το πλαίσιο εργασίας PHPUnit. Σε αντίθεση το Codeigniter προσφέρει μία στοιχειώδη κλάση για τις διαδικασίες αυτές ή την επιλογή ενσωμάτωσης από τον χρήστη του PHPUnit.

Δύο ακομα βασικά πλεονεκτήματα που παρουσιάζει το Laravel είναι τα migrations και το Eloquent. Το Codeigniter όχι μόνο δεν διαθέτει μία μέθοδο για την διαχείριση της έκδοσης μίας σχεσιακής βάσης δεδομένων αλλά και για την δυνατότητα αντιστοίχισης των πινάκων της σε αντικείμενα πρέπει να ενσωματώσει εξωτερικό σύστημα ORM.

Για την συγγραφή και την διαχείριση του περιεχομένου των παρουσιάσεων καθώς και για την δημιουργία επεκτάσιμων προτύπων το Laravel διαθέτει το Blade. Αντίθετα στο εγχειρίδιο του Codeigniter επισημαίνεται στους χρήστες πως το σύστημα συγγραφής προτύπων που ενσωματώνει επιβραδύνει λίγο την ταχύτητα της εφαρμογής καθώς και ότι το σύστημα αυτό αποτελεί μια στοιχειώδη υλοποίηση.

Τέλος αξίζει να αναφερθεί πως υπάρχουν και άλλες διαφορές στην απόδοσης και την ευελιξία, όπως σε θέματα δρομολόγησης και προσωρινής αποθήκευσης (caching), ωστόσο οι παραπάνω ήταν οι βασικοί λόγοι που συντέλεσαν στην μετάβαση αυτή.

Η εξυπηρέτηση των HTTP αιτημάτων καθώς και η υλοποίηση του συνόλου των δυνατοτήτων της παρούσας εφαρμογής πραγματοποιήθηκαν με την χρήση του Laravel. Μέσα από τον μεγάλο αριθμό έτοιμων λειτουργιών που παρέχει, όπως διαχείριση των Cookies, δημιουργία συμβάντων και επικύρωση φορμών κατέστη δυνατή η εύκολη και γρήγορη δημιουργία μίας πολύπλοκης λειτουργικότητας με σκοπό την ενσωμάτωση όλων των χαρακτηριστικών που ορίζουν ένα κοινωνικό δίκτυο.

Redis

Η Redis είναι μία μη σχεσιακή βάση δεδομένων, τύπου Κλειδί/Τιμή (key-value). Έχει δημιουργηθεί με την προγραμματιστική γλώσσα ANSI C και έχει φτιαχτεί για τα λειτουργικά συστήματα Linux και OSX. Βασικό χαρακτηριστικό της είναι η προσωρινής αποθήκευση των δεδομένων (in-memory dataset), δηλαδή η αποθήκευση γίνεται στην μνήμη RAM, με αποτέλεσμα να είναι εξαιρετικά γρήγορη. Η Redis δεν αποτελεί την κλασσική μορφή βάσης Κλειδιού/Τιμή καθώς η τιμή που δέχεται ένα κλειδί δεν περιορίζεται μόνο σε μία ακολουθία χαρακτήρων (string) αλλά μπορεί να υποστηρίξει ακόμα τέσσερις διαφορετικές δομές (lists, sets, sorted sets και hashes) οι οποίες και θα παρουσιαστούν στην συνέχεια. Επίσης είναι δυνατή η ακαριαία εκτέλεση (atomic operation) ορισμένων λειτουργιών στα κλειδιά αυτά ,όπως η αύξηση/μείωση της τιμής του, η προσθήκη μιας τιμής σε μία λίστα και άλλα, κάνοντας έτσι αδύνατη την παρεμπόδιση ολοκλήρωσης της λειτουργίας αυτής. Η τιμή ενός κλειδιού αντιμετωπίζεται σαν μία ροή δεδομένων χωρίς κάποια συγκεκριμένη διαμόρφωση (binary safe) με αποτέλεσμα να μπορεί να είναι από μια ακολουθία χαρακτήρων μέχρι και μία εικόνα με μέγιστο μέγεθος τα 512MB.

Η Redis είναι μία ευρέως διαδεδομένη βάση δεδομένων και έχει μεγάλη απήχηση τόσο σε μεγάλες όσο και σε μικρές εταιρείες. Χρησιμοποιείται σήμερα από πολύ γνωστούς και μεγάλους ιστότοπους για τους οποίους η ταχύτητα και η υψηλή απόδοση είναι πολύ βασικοί παράγοντες. Πιο συγκεκριμένα η Redis χρησιμοποιείται από το κοινωνικό δίκτυο Twitter, τον ιστότοπο για προγραμματιστικές ερωτήσεις Stackoverflow, τον απομακρυσμένο χώρο φιλοξενίας Git έργων GitHub, τον ιστότοπο οπτικών σελιδοδικτών Pinterest, την κινέζικη υπηρεσία microblogging Weibo, τον ιστότοπο μικρών αγγελιών Craigslist, την εφαρμογή ανταλλαγής φωτογραφιών Snapchat, τον ιστότοπο ενημέρωσης Digg και τον ιστότοπο φιλοξενίας φωτογραφιών και βίντεο Flickr.

Δομές δεδομένων

Όπως αναφέρθηκε η Redis υποστηρίζει πέντε διαφορετικές δομές δεδομένων. Κάθε μία από τις δομές αυτές έχει συγκεκριμένα χαρακτηριστικά και ο συνδυασμός του μπορεί να προσφέρει λύση ακόμα και στις πιο απαιτητικές εφαρμογές.

Η πιο απλή δομή δεδομένων που μπορεί να ανατεθεί σε ένα κλειδί είναι μία ακολουθία χαρακτήρων (Strings). Καθώς και τα κλειδιά της Redis αποτελούν μία ακολουθία χαρακτήρων με την δομή αυτή έχουμε την χαρτογράφηση μιας ακολουθίας με μία άλλη. Η δομή αυτή διαθέτει εντολές ακαριαίας εκτέλεσης, με αποτέλεσμα να αποτελεί την βέλτιστη λύση για μετρητές (counters), εξασφαλίζοντας πως δεν υπάρξει ποτέ συνθήκη ανταγωνισμού από δύο πελάτες για το ίδιο κλειδί.

Η δεύτερη δομή δεδομένων που υποστηρίζεται είναι αυτή της διασυνδεδεμένης λίστας (lists). Η δομή αυτή έχει την μορφή ουράς καθώς αποτελείται από μία διατεταγμένη συλλογή στοιχειών κάθε ένα από τα οποία έχει έναν δείκτη για το επόμενο στοιχείο που ακολουθεί. Βασικό πλεονέκτημα της δομής αυτής είναι ο σταθερός χρόνο προσθήκης στοιχειών στην πρώτη και την τελευταία θέση της λίστας, ανεξάρτητα με το μέγεθος της, κάνοντας την έτσι την ιδανική λύση για την αποθήκευση των πιο πρόσφατων ενημερώσεων σε ένα κοινωνικό δίκτυο.

Στην συνέχεια έχουμε μια δομή η οποία μοιάζει σε αρκετά σημεία με τα strings, τα hashes. Ωστόσο η βασική διαφορά είναι πως η δομή αυτή αποτελείται από ζεύγη πεδίων και τιμών με αποτέλεσμα σε ένα hash να μπορούν να αποθηκευτούν παραπάνω από μία τιμές και για κάθε τιμή να υπάρχει ένα

προσδιοριστικό χαρακτηριστικό. Το hash είναι μια δομή δεδομένων πολύ βολική για την αντιπροσώπευση αντικειμένων. με μόνο περιορισμό ως προς το πλήθος των ζευγαριών την διαθέσιμη μνήμη του συστήματος.

Στην συνέχεια έχουμε την δομή σετ (set) η οποία αποτελείται από μια συλλογή μη επαναλαμβανόμενων στοιχειών. Οι τιμές που αποθηκεύονται σε ένα set είναι ακολουθίες χαρακτήρων και δεν έχουν συγκεκριμένη ταξινόμηση με αποτέλεσμα να μπορούν να επιστραφούν σε οποιαδήποτε σειρά. Το γεγονός ότι κάθε τιμή είναι μοναδική σε συνδυασμό με την ύπαρξη ενός συνόλου εντολών για την αλληλεπίδραση μεταξύ των set ,όπως η ένωση δύο ή η εύρεση της τομής, τα καθιστά ιδανικά για την έκφραση σχέσεων μεταξύ των αντικειμένων.

Η τελευταία δομή δεδομένων που υποστηρίζεται είναι τα ταξινομημένα σετ (sorted sets). Η δομή αυτή μοιάζει σαν ένα συνδυασμό των set και των hashes καθώς αποτελείται από μοναδικές μη επαναλαμβανόμενες τιμές κάθε μία από τις οποίες συνοδεύεται από μία βαθμολογία. Η βαθμολογία αυτή είναι ένας δεκαδικός αριθμός βάσει του οποίου γίνεται η ταξινόμηση των τιμών κατά αύξουσα σειρά. Επίσης η ταξινόμηση αυτή δεν γίνεται όταν ζητηθούν τα στοιχεία του sorted set αλλά κατά την διαδικασία εισαγωγής ενός νέου στοιχείου. Τέλος στην περίπτωση που δύο στοιχεία έχουν ακριβώς την ίδια βαθμολογία η ταξινόμηση τους γίνεται βάσει του λεξικογραφικού μήκους της τιμής του κάθε στοιχείου.

Χαρακτηριστικά της Redis

Το βασικό χαρακτηριστικό της Redis είναι η προσωρινή αποθήκευση των δεδομένων στην μνήμη. Ωστόσο τα δεδομένα αυτά δεν χάνονται κατά την επανεκκίνηση του συστήματος καθώς υπάρχει δυνατότητα μόνιμης αποθήκευσης (persistence) τους στον σκληρό δίσκο με δύο διαφορετικούς τρόπους. Ο πρώτος τρόπος ονομάζεται RDB και έχει να κάνει με τον ορισμό τακτών χρονικών διαστημάτων κατά τα οποία η Redis δημιουργεί μία νέα διαδικασία η οποία αποτελεί αντίγραφο του εαυτού της (διαδικασία fork) με σκοπό την εγγραφή των δεδομένων της στον σκληρό δίσκο σε ένα αρχείο που ονομάζεται dump.rdb. Η διαδικασία αυτή ονομάζεται snapshotting και επιτρέπει στην Redis την αντιγραφή του περιεχομένου της χωρίς την παύση της λειτουργίας της. Ο δεύτερος τρόπος ονομάζεται AOF και αποτελείται από την καταγραφή στο αρχείο appendonly.aof κάθε εντολής εγγραφής που στέλνεται στην Redis. Η καταγραφή αυτή μπορεί να οριστεί να γίνεται κάθε δευτερόλεπτο ή με την λήψη της κάθε εντολής χωρίς να επηρεαστεί η απόδοση της Redis καθώς υλοποιείται στο παρασκήνιο (background) με την χρήση νήματος (thread). Επίσης όταν το αρχείο γίνει πολύ μεγάλο εκτελείται η διαδικασία fork με σκοπό την επανεγγραφή του αρχείου το ποίο θα διαθέτει το ελάχιστο δυνατό σύνολο των ενεργειών που απαιτούνται για την δημιουργία του τρέχοντος συνόλου δεδομένων. Κατά την επανεκκίνηση της Redis το αρχείο αυτό χρησιμοποιείται για την ανακατασκευή της βάσης. Για την ελαχιστοποίηση των πιθανοτήτων απώλειας δεδομένων είναι δυνατή η υβριδική υλοποίηση και των δύο μεθόδων στην ίδια βάση.

Η Redis υποστηρίζει την αντιγραφή των δεδομένων σε διάφορους εξυπηρετητές και την συνεργασία με αυτούς (replication). Με την διαδικασία αυτή σε μία κύρια βάση (master) μπορούν να είναι συνδεδεμένες βοηθητικές υπομονάδες (slaves) κάθε μία από τις οποίες θα έχει ένα ενημερωμένο αντίγραφο του συνόλου των δεδομένων και θα μπορεί να εξυπηρετήσει αιτήματα από την εφαρμογή για την ανάγνωση δεδομένων. Η διαδικασία συγχρονισμού της βοηθητικής υπομονάδας ,με σκοπό την ενημέρωση των δεδομένων της, είναι διαδικασία που επιτρέπει την παράλληλη εξυπηρέτηση αιτημάτων τόσο στην κύρια βάση όσο και στην ίδια την υπομονάδα.Επίσης μπορεί να οριστεί η αποθήκευση των δεδομένων στον σκληρό δίσκο να γίνεται σε μία ή περισσότερες από τις συνδεδεμένες υπομονάδες παρέχοντας έτσι μείωση των εργασιών της κύριας βάσης. Η υλοποίηση αυτή επιτρέπει ακόμα την αύξηση της ταχύτητας ανάγνωσης καθώς τα αιτήματα για ανάγνωση εξυπηρετούνται από τις συνδεδεμένες υπομονάδες και η κύρια βάση ασχολείται μόνο για την εγγραφή πληροφοριών. Ταυτόχρονα η μέθοδος αυτή προσφέρει την δυνατότητα επεκτασιμότητας στην εφαρμογή και ασφάλεια από την πιθανότητα απώλειας δεδομένων καθώς υπάρχουν πολλαπλά αντίγραφα.

Μία ακόμα δυνατότητα που παρέχει η Redis είναι αυτή του διαμοιρασμού των δεδομένων (partitioning) σε πολλαπλές υπομονάδες. Σε αντίθεση με την προηγούμενη περίπτωση όπου κάθε υπομονάδα εξαρτώταν από την κύρια βάση και διέθετε ένα πλήρες αντίγραφο των δεδομένων, τώρα κάθε υπομονάδα

είναι ανεξάρτητη και διαθέτει ένα μέρος του συνόλου των πληροφοριών. Η μέθοδος αυτή δεν επιτρέπει την χρήση εντολών που έχουν να κάνουν με πολλαπλά κλειδιά, παρέχει ωστόσο σημαντικές βελτιώσεις στην απόδοση του συστήματος. Συγκεκριμένα επιτρέπει την ύπαρξη μεγαλύτερων βάσεων καθώς αξιοποιεί το άθροισμα της μνήμης του συνόλου των υπολογιστών που συμμετέχουν στον διαμοιρασμό. Επίσης επιτρέπει την κλιμάκωση της υπολογιστικής ισχύς , αξιοποιώντας τους πολλαπλούς πυρήνες των υπολογιστών, καθώς και το εύρος ζώνης του δικτύου , αξιοποιώντας τους προσαρμογείς δικτύου. Μια υλοποίηση της μεθόδου αυτής αποτελεί το Redis Cluster το ποίο επιτρέπει τον αυτόματο διαμερισμό των δεδομένων καθώς και την δυνατότητα συνέχισης των εργασιών ακόμα και όταν μερικές υπομονάδες είναι εκτώς λειτουργίας. Την περίοδο που γράφτηκε η παρούσα διπλωματική εργασία το Redis Cluster βρισκόταν ακόμα σε δοκιμαστικό στάδιο (beta stage) οπότε δεν θα μπούμε σε περισσότερες λεπτομέρειες.

Πριν παρουσιαστεί η δυνατότητα της Redis για εκτέλεση ενός συνόλου εντολών σαν μία (transactions) πρέπει να αναφερθεί η τεχνική pipelining που υποστηρίζει. Το πρωτόκολλο Αίτηση/Απάντηση (Request/Response protocol) που χρησιμοποιεί η Redis συνήθως υλοποιείται με την αποστολή μίας αίτησης από τον πελάτη στον εξυπηρετητή, την επεξεργασία της αίτησης, την δημιουργία της απάντησης και στην συνέχεια την παρεμπόδιση οποιασδήποτε άλλης ενέργειας του εξυπηρετητή μέχρι ο πελάτης να παραλάβει την απάντηση. Ωστόσο με την τεχνική pipelining το πρωτόκολλο αυτό μπορεί να υλοποιηθεί με τέτοιο τρόπο ώστε ο εξυπηρετητής να είναι σε θέση να μπορεί να επεξεργαστεί νέες αιτήσεις ακόμα και αν ο πελάτης δεν έχει λάβει ακόμα την απάντηση. Με την αξιοποίηση της τεχνικής αυτής η λειτουργία transactions επιτρέπει την αποστολή ενός συνόλου εντολών, την επεξεργασία τους σαν μία και την χρήση του αποτελέσματος μίας εντολής σαν είσοδο για την επόμενη. Η μέθοδος αυτή διασφαλίζει πως οι εντολές θα εκτελεστούν διαδοχικά, κάνοντας αδύνατη την παρεμπόδιση της ροής εκτέλεσης από εντολές που στάλθηκαν από άλλους πελάτες. Επίσης η μέθοδος εγγυάται πως είτε θα εκτελεστούν όλες οι εντολές ή καμία κάνοντας την έτσι διαδικασία ακαριαίας εκτέλεσης.

Πρότυπο Δημοσίευση/Συνδρομή

Πέρα από τις ιδιαίτερες δυνατότητες της Redis που παρουσιάστηκαν ένα ακόμα βασικό χαρακτηριστικό της είναι πως διαθέτει ενσωματωμένη υλοποίηση του προτύπου Δημοσίευση/Συνδρομή. Βάσει του προτύπου αυτού ο αποστολέας (publisher) δεν στέλνει τα μηνύματα κατευθείαν στον συνδρομητή (subscriber) αλλά σε ένα κανάλι χωρίς να γνωρίζει αν και πόσοι συνδρομητές υπάρχουν. Οι συνδρομητές με την σειρά τους εγγράφονται στα κανάλια που τους ενδιαφέρουν και λαμβάνουν μηνύματα μόνο από τα κανάλια αυτά χωρίς να γνωρίζουν αν και πόσοι αποστολείς υπάρχουν.

Η Redis διαθέτει ένα σύνολο κατάλληλων εντολών για την λειτουργία του μοντέλου αυτού. Για τον συνδρομητή είναι διαθέσιμες οι εντολές Subscribe, για την εγγραφή σε ένα κανάλι, Unsubscribe, για την απεγγραφή από κάποιο κανάλι, Psubscribe, για την εγγραφή σε όσα κανάλια ταυτίζονται με ένα μοτίβο, και Punsubscribe , για την απεγγραφή από τα κανάλια που ταυτίζονται με ένα μοτίβο. Για τον αποστολέα του μηνύματος η μόνη διαθέσιμη εντολή είναι η Publish συνοδευόμενη από το όνομα του καναλιού και το μήνυμα. Επίσης υπάρχει η εντολή PubSub για τον έλεγχο της κατάστασης του συστήματος Δημοσίευση/Συνδρομή η οποία συνοδευόμενη από κατάλληλα ορίσματα παρέχει πληροφορίες για το ποια είναι τα ενεργά κανάλια, το πλήθος των συνοδρομητών σε συγκεκριμένα κανάλια και το πλήθος των μοτίβων στα οποία έχουν εγγραφεί χρήστες..

Η Redis αποτελεί την βάση δεδομένων που χρησιμοποιήθηκε για τις ανάγκες αποθήκευσης και ανάκτησης δεδομένων της παρούσας εφαρμογής. Επίσης, ο συνδυασμός του μοντέλου Δημοσίευση/Συνδρομή που ενσωματώνει και της υποστήριξη που παρέχει για αυτή η πλατφόρμα Node.JS, όπως θα δούμε στην συνέχεια, συντέλεσε στην λειτουργία της σαν κόμβο επικοινωνίας μεταξύ της πλατφόρμας αυτής και του εξυπηρετητή αιτημάτων HTTP. Με τον τρόπο αυτό κατέστη δυνατή η υλοποίηση του μηχανισμού παροχής σε πραγματικό χρόνο ενημερώσεων και ειδοποιήσεων που συναντάτε στην παρούσα διπλωματική εργασία.

Node.JS

Το Node.js είναι μία πλατφόρμα που εκτελείται στην μεριά του εξυπηρετητή και παρέχει ένα περιβάλλον ανάπτυξης γρήγορων και επεκτάσιμων διαδικτυακών εφαρμογών. Οι εφαρμογές αυτές γράφονται με την γλώσσα προγραμματισμού Javascript και μπορούν να εκτελεστούν εντός του περιβάλλοντος αυτού. Η αρχιτεκτονική σχεδίασης του είναι βασισμένη τόσο στα συμβάντα (event-driven) όσο και στην μη παρεμπόδιση των διαδικασιών εισόδου/εξόδου δεδομένων (non-blocking I/O) κάνοντας το έτσι ιδανικό για εφαρμογές πραγματικού χρόνου με υψηλή χρήση δεδομένων (data intensive). Το Node.JS είναι γραμμένο με τις γλώσσες προγραμματισμού C, C++ και Javascript και μπορεί να εκτελεστεί στα λειτουργικά συστήματα Linux, OSX, Windows FreeBSD και IBM i. Επίσης συνοδεύεται από το npm (Node Package Manager) το οποίο αποτελεί έναν απομακρυσμένο χώρο αποθήκευσης βιβλιοθηκών για το Node.js ενώ ταυτόχρονα επιτρέπει την εγκατάσταση τους καθώς και την διαχείριση των εκδόσεων και των εξαρτήσεων τους. Τρεις τέτοιες βιβλιοθήκες χρησιμοποιήθηκαν στην παρούσα διπλωματική εργασία και παρουσιάζονται στην συνέχεια.

Βιβλιοθήκη Express.JS

Αρχικά χρησιμοποιείται η βιβλιοθήκη για την ανάπτυξη διαδικτυακών εφαρμογών Express.JS. Πρόκειται για ένα ελαφρύ και ευέλικτο πλαίσιο εργασίας το οποίο επιτρέπει την τμηματοποίηση της εφαρμογής βάσει του μοντέλου MVC. Παράλληλα παρέχει ένα σύνολο λειτουργιών κατάλληλων για την εξυπηρέτηση αιτημάτων, την διαχείριση των δρομολογήσεων και γενικά ότι δυνατότητες παρέχει ένας εξυπηρετητής HTTP αιτημάτων. Η βιβλιοθήκη αυτή αποτέλεσε τον συνδετικό κρίκο για την συνεργασία όλων των βιβλιοθηκών που χρησιμοποιήθηκαν στο Node.JS.

Βιβλιοθήκη Redis

Η επόμενη βιβλιοθήκη που χρησιμοποιείται είναι η redis. Σκοπός της βιβλιοθήκης είναι η σύνδεση και αλληλεπίδραση του Node.JS με την βάση δεδομένων Redis. Για να μπορέσει να γίνει η σύνδεση δημιουργείται ένα αντικείμενο στο οποίο φορτώνεται η βιβλιοθήκη (require('redis')). Με την ολοκλήρωση της διαδικασίας αυτής είναι δυνατή η χρήση όλων των διαθέσιμων εντολών που υποστηρίζει η Redis με την μορφή συναρτήσεων (functions) πάνω στο αντικείμενο αυτό. Κάθε συνάρτηση συνοδεύεται από ένα σύνολο παραμέτρων καθώς και από μία μέθοδο επανάκλησης (callback).

Βιβλιοθήκη Socket.IO

Η τελευταία βιβλιοθήκη που χρησιμοποιήθηκε ήταν η Socket.IO. Η βιβλιοθήκη αυτή επιτρέπει την ανάπτυξη διαδικτυακών εφαρμογών με δυνατότητα αμφίδρομης και σε πραγματικό χρόνο επικοινωνίας μεταξύ πελάτη και εξυπηρετητή. Για την επίτευξη της επικοινωνίας αυτής υποστηρίζεται το πρωτόκολλο WebSocket της HTML5 καθώς και ένα σύνολο εφεδρικών μηχανισμών για την περίπτωση παλαιότερης έκδοσης φυλλομετρητή όπως Adobe® Flash® Socket, AJAX long polling, AJAX multipart streaming, Forever Iframe και JSONP Polling. Ανεξάρτητα όμως από το πρωτόκολλο που χρησιμοποιείται η βιβλιοθήκη παρέχει για όλα την ίδια προγραμματιστική διεπαφή. Οι εφαρμογές που δημιουργούνται αποτελούνται από δύο μέρη, το μέρος του πελάτη το οποίο υλοποιείται στον φυλλομετρητή και το μέρος του εξυπηρετητή το οποίο υλοποιείται στο περιβάλλον του Node.JS. Στην μερία του πελάτη πρέπει να υλοποιηθούν δύο βασικές λειτουργίες για να μπορέσει να είναι επιτυχής η επικοινωνία. Πρώτα πρέπει να γίνει η σύνδεση με τον εξυπηρετητή που τρέχει στην πλατφόρμα Node.JS με την εντολή “io.connect()” η οποία δέχεται σαν παράμετρο την IP του εξυπηρετητή και την θύρα που θα συνδεθεί. Στην συνέχεια με την εντολή “node.on( ), function(data){}”δηλώνεται το όνομα του καναλιού που είναι συνδρομητής ο πελάτης ακολουθούμενη από μία διαδικασία επανάκλησης για τον ορισμό της συμπεριφοράς που επιθυμούμε να έχει ο φυλλομετρητής όταν παραλάβει το μήνυμα. Αξίζει να σημειωθεί πως η εγγραφή του πελάτη στο κανάλι γίνεται στην πλευρά του εξυπηρετητή και όχι με την συνάρτηση “on”. Το όρισμα που

δέχεται η συνάρτηση αυτή αποτελεί την προσδιοριστική επικεφαλίδα που θα συνοδεύει το μήνυμα που θα στείλει ο εξυπηρετητής στους συνδρομητές του καναλιού και που επιθυμούμε να αναγνωρίσει ο πελάτης για να προβεί στις κατάλληλες ενέργειες. Με τον τρόπο αυτό εάν ο ίδιος πελάτης είναι συνδρομητής σε περισσότερα από ένα κανάλια είναι δυνατή η αντιστοίχιση του κάθε μηνύματος με το κανάλι που ανήκει. Στην μεριά του εξυπηρετητή για να μπορέσει να είναι επιτυχής η επικοινωνία πρέπει να δημιουργηθεί ένα αντικείμενο το οποίο θα δέχεται εισερχόμενες συνδέσεις. Η εντολή “io.listen(server).on('connection', function(client){ }” δέχεται τις συνδέσεις αυτές και μέσα στην συνάρτηση επανάκλησης της τοποθετούνται οι εντολές για την δημιουργία ενός στιγμιότυπου της βάσης Redis (redis.createClient()), την εγγραφή του συνδρομητή στο κανάλι (redisClient.subscribe()), την παραλαβή του μηνύματος όταν δημοσιευθεί στην Redis (redisClient.on('message', function(channel, message){}), την αποστολή του μηνύματος στον πελάτη (client.emit(channel, message)) καθώς και τον τερματισμό της σύνδεσης με τον πελάτη αυτό (redisClient.quit()).

Η πλατφόρμα Node.JS αποτέλεσε τον εξυπηρετητή αιτημάτων αμφίδρομης και σε πραγματικό χρόνο επικοινωνίας που υλοποιήθηκε στην παρούσα εφαρμογή. Για την επίτευξη της επικοινωνίας αυτής ήταν απαραίτητη η χρήση της βιβλιοθήκης Socket.IO η οποία εφαρμόζει το πρωτόκολλο WebSocket και επιτρέπει την δημιουργία μίας TCP σύνδεσης μεταξύ πελάτη και εξυπηρετητή με σκοπό την ταυτόχρονα αμφίδρομη μεταξύ τους ανταλλαγή μηνυμάτων. Επίσης η βιβλιοθήκη redis επέτρεψε την επικοινωνία της πλατφόρμας με την βάση δεδομένων Redis και την αξιοποίηση της σαν κόμβο επικοινωνίας με σκοπό την λήψη των μηνυμάτων που δημοσιεύει στα κανάλια της ο εξυπηρετητής HTTP αιτημάτων και την προώθηση τους στους αντίστοιχους πελάτες.

Ajax

Η Ajax (Asynchronous Javascript And XML) δεν είναι μία καθεαυτού τεχνολογία αλλά αποτελεί μία νέα προσέγγιση για τον συνδυασμού ενός συνόλου υπαρχόντων τεχνολογιών. Οι τεχνολογίες αυτές περιλαμβάνουν την HTML ή την XHTML, το CSS, την Javascript, το DOM, την XML, την XSLT και το αντικείμενο XMLHttpRequest. Όταν οι τεχνολογίες αυτές συνδυαστούν στο μοντέλο του Ajax τότε παρέχεται η δυνατότητα στις εφαρμογές να επικοινωνούν ασύγχρονα με τον εξυπηρετητή και να λαμβάνουν δεδομένα από αυτόν χωρίς να επηρεάζεται η συμπεριφορά της σελίδας που προβάλλεται. Τα δεδομένα μπορούν να χρησιμοποιηθούν στην συνέχεια για την ενημέρωση ενός συγκεκριμένου τμήματος της σελίδας αυτής χωρίς να απαιτείται η επαναφόρτωση της. Η ασύγχρονη αυτή επικοινωνία είναι γρήγορη και επιτρέπει την μερική ενημέρωση του περιεχόμενου που προβάλλεται κάνοντας έτσι τις διαδικτυακές εφαρμογές πιο γρήγορες και πιο ανταποκριτικές σε ενέργειες χρηστών. Στις σύγχρονες υλοποιήσεις της μεθόδου Ajax δεν χρησιμοποιείται τόσο πολύ η XML καθώς έχει αντικατασταθεί από την τεχνολογία JSON η οποία όχι μόνο είναι ελαφρύτερη αλλά αποτελεί και μέρος της Javascript. Μερικά παραδείγματα μεγάλων εφαρμογών που χρησιμοποιείται η μέθοδος Ajax είναι οι χάρτες του Google, το Gmail, το Youtube, το Twitter και το Facebook.

Στην παρούσα διπλωματική εργασία η μέθοδος αυτή υλοποιήθηκε σε τρεις περιπτώσεις. Αρχικά χρησιμοποιήθηκε για την αποστολή της φόρμας για την διαγραφή μίας βαθμολογίας που έχει υποβάλει ένας χρήστης. Στην συνέχεια για την λήψη των πέντε πιο πρόσφατων δημόσιων βαθμολογιών που έγιναν ανεξαρτήτως χρήστη. Τέλος για την λήψη της HTML σήμανσης (markup) αυτών των πέντε βαθμολογιών καθώς και της σήμανσης για τις πρόσφατες βαθμολογίες που εμφανίζονται στο χρονολόγιο του κάθε χρήστη.

Σύνοψη

Στο κεφάλαιο αυτό έγινε η παρουσίαση των τεχνολογιών που χρησιμοποιήθηκαν για την υλοποίηση της παρούσας διπλωματικής εργασίας καθώς και μία συνοπτική αναφορά για το που χρησιμοποιήθηκε κάθε μία από αυτές

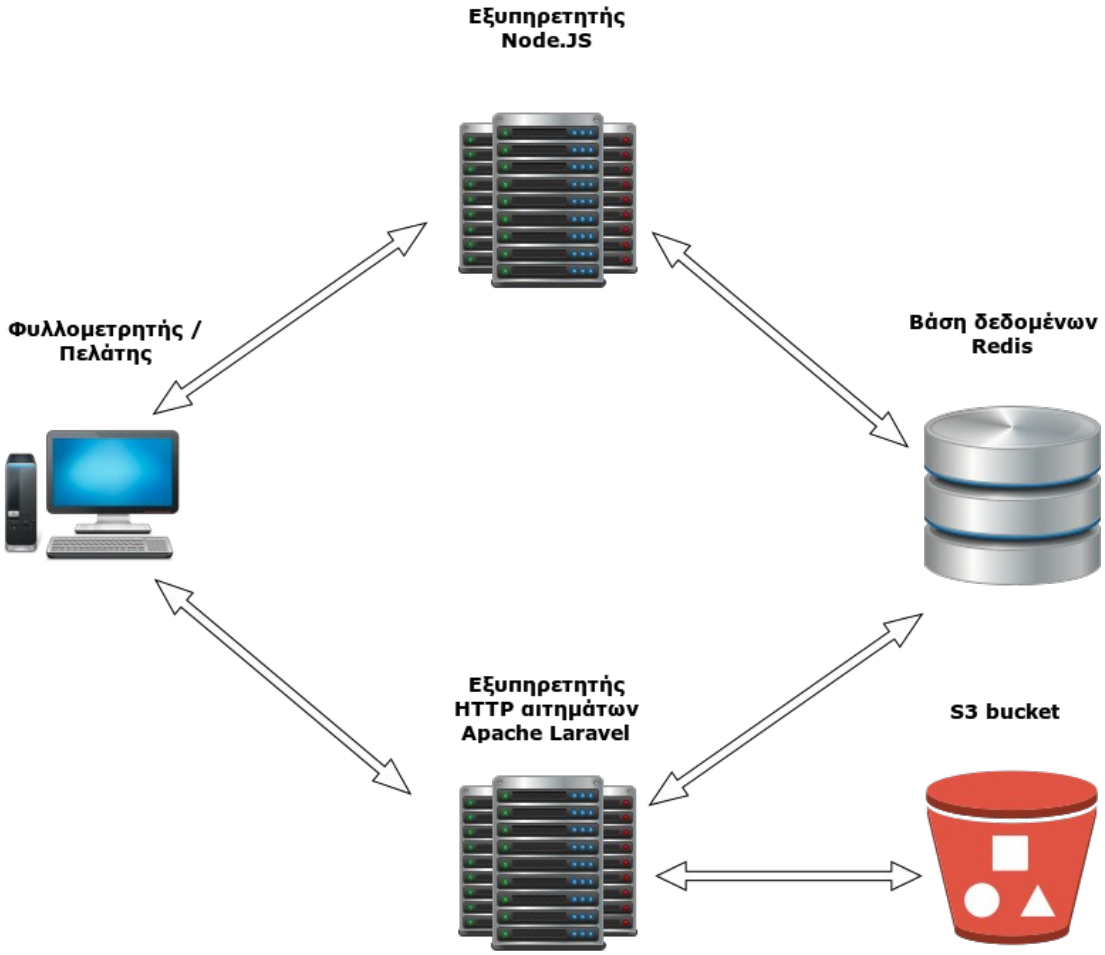
Στο κεφάλαιο που ακολουθεί παρουσιάζεται λεπτομερώς η αρχιτεκτονική της εφαρμογής ,βάσει των τεχνολογιών που χρησιμοποιήθηκαν, καθώς και η διαδικασία σχεδίασης και ανάπτυξης της με την χρήση της μεθοδολογίας UML. Τέλος παρουσιάζεται η σύγκριση των δύο διαφορετικών μοντέλων διάταξης δεδομένων που σχεδιάστηκαν.

Αρχιτεκτονική

Στο κεφάλαιο αυτό παρουσιάζεται η αρχιτεκτονική του συστήματος μέσα από την ανάλυση των επιμέρους επιπέδων/τμημάτων που το αποτελούν. Στη συνέχεια παρουσιάζεται η διαδικσία ανάλυσης και σχεδιασμού της εφαρμογής, βάσει της γλώσσας μοντελοποίησης UML, μέσα από διαγράμματα ακολουθίας, δραστηριοτήτων και περιπτώσεων χρήσης.Τέλος παρουσιάζονται οι δύο διαφορετικές εκδοχές που σχεδιάστηκαν και υλοποιήθηκαν για τον τρόπο διάταξης των δεδομένων στην Redis καθώς και μία συνοπτική σύγκριση τους.

Αρχιτεκτονική συστήματος

Η αρχιτεκτονική της εφαρμογής που υλοποιήθηκε για την παρούσα διπλωματική εργασία είναι πολυεπίπεδη και αποτελείται από ένα σύνολο διασυνδεδεμένων τμημάτων. Τα τέσσερα διακριτά επίπεδα που την διαμορφώνουν είναι το επίπεδο πελατών, το επίπεδο εξυπηρέτησης HTTP αιτημάτων, το επίπεδο εξυπηρέτησης αιτημάτων αμφίδρομης και πραγματικού χρόνου επικοινωνίας (πρωτοκόλλου WebSocket, Adobe® Flash® Socket κτλπ) και το επίπεδο αποθήκευσης δεδομένων. Κάθε ένα από τα επίπεδα αυτά παρουσιάζεται στην συνέχεια.



Αρχιτεκτονική συστήματος

Επίπεδο πελατών

Στο επίπεδο αυτό υλοποιείται η επικοινωνία και αλληλεπίδραση του χρήστη με την εφαρμογή και αποτελείται από ένα φυλλομετρητή. Βασικός σκοπός του φυλλομετρητή είναι τόσο η προβολή των ιστοσελίδων που του στέλνει ο εξυπηρετητής όσο και η αποστολή σε αυτόν του συνόλου των ενεργειών του χρήστη. Η διαδικασία αυτή υλοποιείται μέσα από την αποστολή HTTP αιτημάτων στον εξυπηρετητή, τη λήψη των απαντήσεων που του αποστέλλονται και την επεξεργασία και προβολή των πόρων και των δεδομένων που περιέχουν οι απαντήσεις αυτές. Επίσης στο επίπεδο αυτό γίνεται η εγκαθίδρυση της σύνδεσης με τον εξυπηρετητή Node.js καθώς και η επεξεργασία των δεδομένων που δέχεται από αυτόν με σκοπό την προβολή τους στον χρήστη.

Επίπεδο εξυπηρέτησης αιτημάτων HTTP

Στο επίπεδο αυτό γίνεται η εξυπηρέτηση όλων αιτημάτων που στέλνονται από το χρήστη μέσω του πρωτοκόλλου HTTP και αποτελείται από τον εξυπηρετητή παγκόσμιου ιστού Apache. Ο Apache είναι ένα πρόγραμμα γραμμένο με τις γλώσσες προγραμματισμού C, C++ και XML και ασχολείται με τη λήψη των HTTP αιτημάτων, την επεξεργασία τους βάσει του κώδικα που έχει υλοποιηθεί, τη δημιουργία κατάλληλων απαντήσεων και την αποστολή των απαντήσεων αυτών πίσω στον φυλλομετρητή. Μέσα από τη διαδικασία αυτή επιτυγχάνεται η παροχή δεδομένων στο φυλλομετρητή, ανάλογων με τις ενέργειες του χρήστη, παρέχοντας έτσι τη λειτουργικότητα που σχεδιάστηκε να προσφέρει η εφαρμογή.

Το περιεχόμενο του επιπέδου αυτού είναι όλα τα αρχεία που απαρτίζουν την εφαρμογή. Πιο συγκεκριμένα στο επίπεδο αυτό φιλοξενείται το πλαίσιο εργασίας Laravel με όλες τις υποστηρικτικές του βιβλιοθήκες, τα αρχεία με τον PHP κωδικά, οι παρουσιάσεις (Views), τα αρχεία CSS και Javascript καθώς και όλοι οι πόροι της εφαρμογής (φωτογραφίες, έγγραφα κτλπ). Για το λόγο αυτό μία απάντηση που δημιουργείται σε ένα αίτημα πέρα από τα δεδομένα περιλαμβάνει κώδικα HTML, αρχεία CSS, Javascript και τους ανάλογους πόρους προς προβολή.

Επίπεδο εξυπηρέτησης αιτημάτων αμφίδρομης και πραγματικού χρόνου επικοινωνίας

Στο επίπεδο αυτό γίνεται η εξυπηρέτηση των αιτημάτων που στέλνονται όταν έχει επιτευχθεί μία αμφίδρομη και σε πραγματικό χρόνο επικοινωνία με το φυλλομετρητή. Το λογισμικό που χρησιμοποιείται για την λήψη, επεξεργασία και απάντηση των αιτημάτων αυτών είναι η πλατφόρμα Node.JS μαζί με τις υποστηρικτικές βιβλιοθήκες Socket.IO και redis. Το λογισμικό που χρησιμοποείται για την δημιουργία των καναλιών, την εγγραφή χρηστών σε αυτά και την δημοσίευση των αιτημάτων στους εκάστοτε εγγεγραμμένους χρήστες είναι η βάση δεδομένων Redis. Η λειτουργικότητα αυτή επιτυγχάνεται μέσω του μοντέλου Δημοσιεύσεων/Συνδρομών (Publish/subscribe) που διαθέτει ενσωματωμένο η Redis. Επίσης εδώ πέρα από τον κώδικα που έχει συγγραφεί για να εκτελεί η πλατφόρμα Node.JS κατά την εξυπηρέτηση εισερχόμενων συνδέσεων φιλοξενείται και ο κώδικας των εξωτερικών βιβλιοθηκών που χρησιμοποιεί.

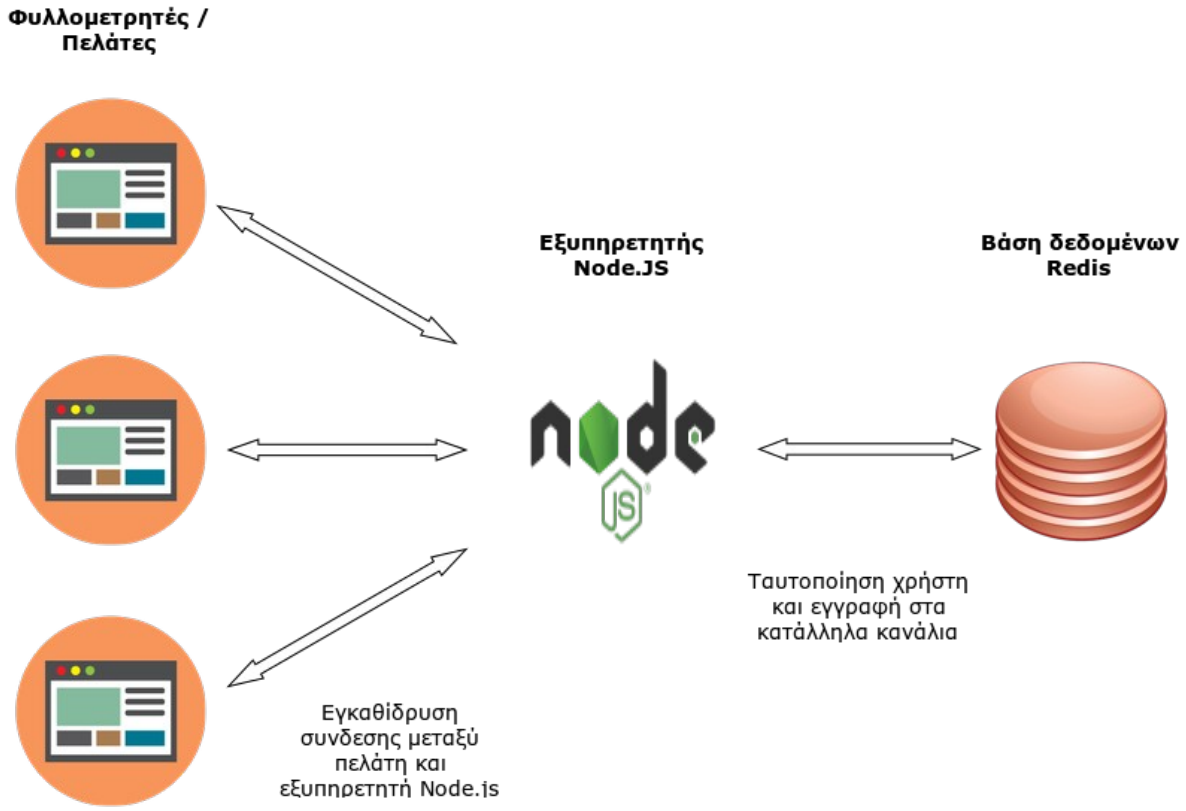
Οι ενέργειες που πραγματοποιούνται στο επίπεδο αυτό είναι αρχικά η εγκαθίδρυση της σύνδεσης με το φυλλομετρητή και η εγγραφή (subscribe) του χρήστη στα κατάλληλα κανάλια της βάσης δεδομένων Redis. Στην συνέχεια γίνεται η παραλαβή των μηνυμάτων που δημοσιεύει (publish) ο εξυπηρετητής HTTP αιτημάτων στα ενεργά κανάλια της Redis και τέλος πραγματοποιείται η αποστολή του κάθε μηνύματος στον πελάτη για τον οποίο προορίζεται.

Επίπεδο αποθήκευσης δεδομένων

Στο επίπεδο αυτό εκτελούνται δύο διακριτές λειτουργίες, η μόνιμη αποθήκευση δεδομένων και η προσωρινή αποθήκευση δεδομένων. Οι δύο λειτουργίες αυτές υλοποιούνται μέσω ενός ενεργού στιγμιότυπου της μη σχεσιακής βάσης δεδομένων Redis και κάθε μία εξυπηρετεί διαφορετικό σκοπό.

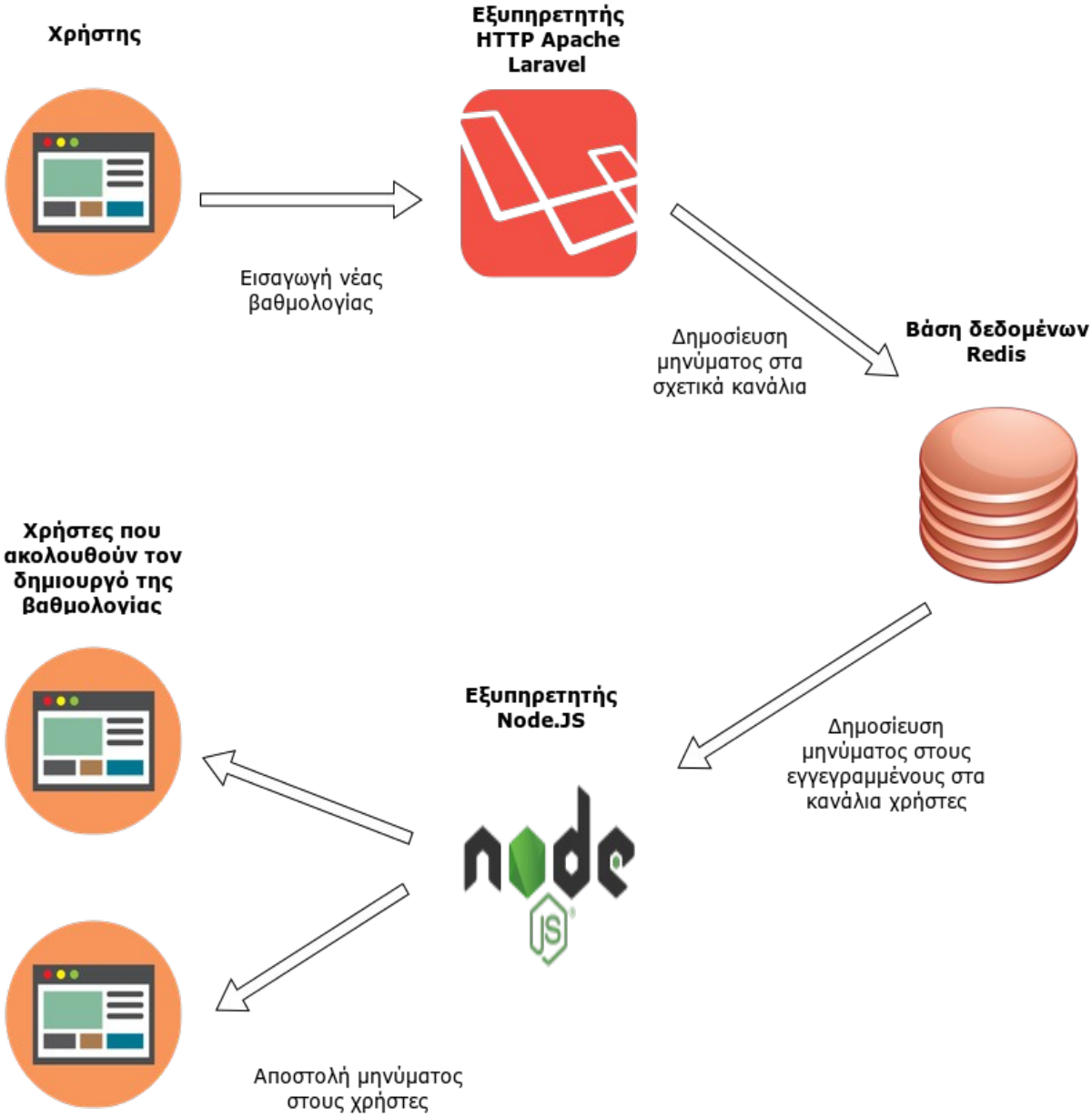
Η λειτουργία μόνιμης αποθήκευσης δεδομένων που παρέχει η Redis χρησιμοποιείται για την αποθήκευση δεδομένων που αφορούν τα προσωπικά στοιχεία των χρηστών, πληροφορίες σχετικά με τις βαθμολογίες που έχουν εισαχθεί στο σύστημα καθώς και διάφορες σχέσεις μεταξύ των δεδομένων αυτών με σκοπό την δημιουργία ευέλικτων και αποδοτικών ευρετηρίων (indexes). Σε καμία περίπτωση δε θέλουμε τα δεδομένα αυτά να χαθούν κατά την επανεκκίνηση του συστήματος και για τον λόγο αυτό αποθηκεύονται στο σκληρό δίσκο. Κατά τη διεξαγωγή μίας λειτουργίας που απαιτείται ανάκτηση ή αποθήκευση δεδομένων ο εξυπηρετητής HTTP αιτημάτων επικοινωνεί, μέσω της βιβλιοθήκης redis του πλαισίου εργασίας Laravel, με τη βάση δεδομένων Redis για την ολοκλήρωση της.

Η λειτουργία προσωρινής αποθήκευσης δεδομένων ασχολείται με την υλοποίηση του ενσωματωμένου μοντέλου Δημοσιεύσεων/Συνδρομών (Publish/subscribe) που διαθέτει η Redis Για την υλοποίηση του μοντέλου αυτού η Redis είναι υπεύθυνη τόσο για την εγγραφή των χρηστών στα κατάλληλα κανάλια όσο και για τη δημοσίευση των μηνυμάτων στα κανάλια αυτά. Τα μηνύματα που δημοσιεύονται δεν αποθηκεύονται μόνιμα καθώς οι πληροφορίες που διαθέτουν έχουν ήδη αποθηκευτεί στην βάση από τον εξυπηρετητή HTTP αιτημάτων. Πιο συγκεκριμένα ο εξυπηρετητής αιτημάτων Node.JS αφού εγκαθιδρύσει επιτυχώς, μέσω της βιβλιοθήκης Socket.IO, μία σύνδεση με το φυλλομετρητή επικοινωνεί με την Redis για την εγγραφή του χρήστη σε τρία κανάλια, δύο προσωπικά και ένα κοινό για όλους. Τα προσωπικά κανάλια χρησιμοποιούνται στη δημοσίευση βαθμολογιών που πρέπει να προβληθούν στο χρονολόγιο του χρήστη καθώς και ενημερώσεις σχετικά με τη δραστηριότητα του κύκλου των επαφών του. Το τρίτο κανάλι χρησιμοποιείται για την δημοσίευση των πιο πρόσφατων δημόσιων (Public) βαθμολογιών. Τα διαπιστευτήρια του κάθε χρήστη (auth) για την ικανότητα εγγραφής του στα προσωπικά του κανάλια βρίσκονται αποθηκευμένα στην Redis.



Εγγραφή συνδρομητή στα κανάλια της Redis

Όταν ολοκληρωθεί από τον εξυπηρετητή αιτημάτων HTTP μια λειτουργία της εφαρμογής η οποία συνοδεύεται από τη δημοσίευση ενός μηνύματος στα κανάλια αυτά (όπως προσθήκη νέας βαθμολογίας και ενημέρωση του χρονολογίου των ακόλουθων) τότε ο εξυπηρετητής επικοινωνεί με την Redis και πραγματοποιεί την εγγραφή του μηνύματος στο ανάλογο κανάλι. Μόλις ολοκληρωθεί η εγγραφή του μηνύματος η Redis αναλαμβάνει τη δημοσίευση του σε όλους τους συνδρομητές του καναλιού.



Δημοσίευση μηνύματος σε κανάλι

Η δημοσίευση υλοποιείται μέσω του εξυπηρετητή Node.JS καθώς λαμβάνει από τη Redis το μήνυμα και το κανάλι από το οποίο αυτό προέρχεται και αναλαμβάνει μέσω της βιβλιοθήκης Socket.IO την αποστολή του στους κατάλληλους φυλλομετρητές. Η Redis αποτελεί τον συνδετικό κρίκο στη διαδικασία επικοινωνίας μεταξύ των δύο εξυπηρετητών.

Ανάλυση και σχεδιασμός εφαρμογής

Για την ανάλυση και το σχεδιασμό της παρούσας εφαρμογής χρησιμοποιείται η ενοποιημένη γλώσσα μοντελοποίησης (Unified Modeling Language) UML. Πρόκειται για μία γραφιστική γλώσσα μοντελοποίησης γενικής χρήσης η οποία προέκυψε από την ενοποίηση πολλών υπαρχόντων και διαφορετικών μεταξύ τους γλωσσών. Χρησιμοποιείται για τη σχηματική αναπαράσταση των προδιαγραφών και των απαιτήσεων μίας εφαρμογής καθώς και για την δημιουργία και τεκμηρίωση των τμημάτων που αποτελούν ένα σύστημα λογισμικού.

Μία μεθοδολογία ανάπτυξης λογισμικού αποτελείται από μία γλώσσα μοντελοποίησης και μία διαδικασία. Η γλώσσα μοντελοποίησης βοηθάει στην περιγραφή του σχεδιασμού ενώ η διαδικασία ορίζει τον τρόπο δημιουργίας του σχεδιασμού αυτού. Τον ρόλο της γλώσσας μοντελοποίησης θα τον αναλάβει η γλώσσα UML η οποία παρέχει ένα σύνολο διαγραμμάτων ως εργαλεία συμβολισμού. Ένα μέρος των διαγραμμάτων εισάγονται στην διαδικασία ανάλυσης (πχ τάξεις, συσχετισμοί) ενώ ένα άλλο μέρος εισάγεται στον σχεδιασμό (π.χ. ιδιότητες). Ο ρόλος της διαδικασίας θα υλοποιηθεί μέσα από την ανάπτυξη των τεσσάρων φάσεων του αντικειμενοστραφούς μοντέλου ανάπτυξης λογισμικού (Rational Unified Process) RUP το οποίο και θα παρουσιαστεί στην συνέχεια.

Κατηγορίες διαγραμμάτων UML

Η UML ορίζει ένα σύνολο διαγραμμάτων για την αναπαράσταση των διαφορετικών απόψεων του συστήματος κατά την φάση ανάπτυξης και υλοποίησης ενός λογισμικού. Οι διάφορες κατηγορίες καθώς και διαγράμματα που περιέχουν η κάθε μία από αυτές παρουσιάζονται στην συνέχεια:

- Διαγράμματα περιπτώσεων χρήσης (use case diagrams)
- Διαγράμματα δομής
  - Διαγράμματα κλάσεων (class diagrams)
  - Διαγράμματα αντικειμένων (object diagrams)
- Διαγράμματα συμπεριφοράς
  - Διαγράμματα καταστάσεων (state chart diagrams)
- Διαγράμματα δραστηριοτήτων (activity diagrams)
- Διαγράμματα αλληλεπίδρασης
  - Διαγράμματα ακολουθίας (sequence diagrams)
  - Διαγράμματα συνεργασίας (collaboration diagrams)
- Διαγράμματα δομής υλοποίησης
  - Διαγράμματα συστατικών (component diagrams)
  - Διάγραμμα ανάπτυξης (deployment diagram)

Μοντέλο αντικειμενοστραφούς σχεδίασης λογισμικού RUP

Το αντικειμενοστραφές μοντέλο Rational Unified Process έχει αναπτυχθεί από τους δημιουργούς της αντικειμενοστραφούς γλώσσας μοντελοποίησης UML οι οποίοι και το συνιστούν ως διαδικασία ανάπτυξης λογισμικού. Επίσης προτείνεται ο κύκλος ζωής του λογισμικού να είναι επαναληπτικός, δηλαδή η ανάπτυξη να προχωρεί σε μία σειρά επαναλήψεων μέχρι να εξελιχθεί το τελικό προϊόν.

Η διαδικασία RUP αφορά κυρίως την ανάλυση απαιτήσεων και τον σχεδιασμό ενός λογισμικού και αποτελείται από ένα σύνολο οδηγιών σχετικά με τεχνικές και απόψεις για την ανάπτυξη του. Είναι δομημένη σε δύο διαστάσεις τον χρόνο και τα τμήματα της διαδικασίας με κάθε διάσταση να έχει τα δικά της χαρακτηριστικά.

Η δόμηση ενός έργου σε σχέση με την διάσταση του χρόνου ακολουθεί τις εξής φάσεις:

1. Έναρξη (Inception): Καθορίζει την προοπτική του έργου

2. Εκπόνηση μελέτης(Elaboration): Σχεδιασμός των απαιτούμενων δραστηριοτήτων και πόρων. Καθορισμός των χαρακτηριστικών και σχεδιασμός της αρχιτεκτονικής

3. Κατασκευή (Construction): Ανάπτυξη του προϊόντος σε μια σειρά βηματικών επαναλήψεων

4. Μετάβαση (Transition): Προμήθεια του προϊόντος στην κοινότητα χρηστών (παραγωγή, διανομή, εκπαίδευση).

Η δόμηση έργου σύμφωνα με τη διάσταση των τμημάτων της διαδικασίας περιλαμβάνει τις ακόλουθες δραστηριότητες:

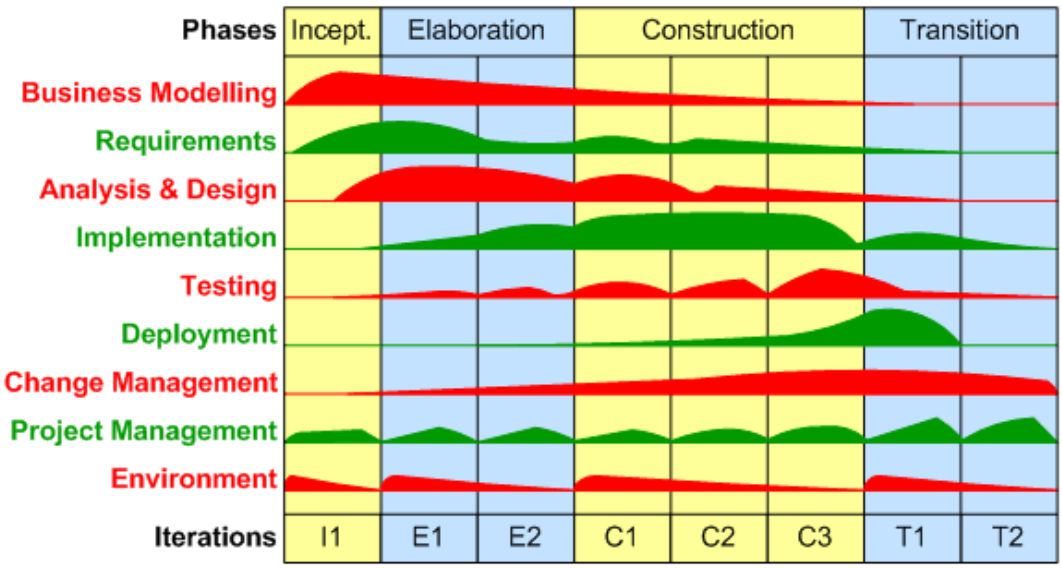
1. Σύλληψη απαιτήσεων (Requirements capture): Μια αφήγηση του τι πρέπει να κάνει το σύστημα

2. Ανάλυση και σχεδιασμός (Analysis and design): Μια περιγραφή του πως θα υλοποιηθεί το σύστημα

3. Υλοποίηση (Implementation): Η παραγωγή του κώδικα

4. Έλεγχος (Test): Η επαλήθευση του συστήματος

Το παρακάτω σχήμα παρουσιάζει μία γραφική αναπαράσταση των διαστάσεων αυτών. Ο οριζόντιος άξονας αναπαριστά τον χρόνο και παρουσιάζει τις πτυχές του κύκλου ζωής της διαδικασίας ανάπτυξης λογισμικού. Η διάσταση αυτή περιγράφεται σε σχέση με τις φάσεις (phases) και τις επαναλήψεις (iterations). Ο κάθετος άξονας παρουσιάζει το περιεχόμενο και δείχνει τους κλάδους-τομείς (disciplines) ,που ομαδοποιούν λογικά τη διαδικασία του περιεχομένου.



Κύκλος ζωής ανάπτυξης λογισμικού

Όπως δείχνουν τα «σαμαράκια» στο παραπάνω σχήμα, η σχετική έμφαση των κλάδων (disciplines) αλλάζει κατά την διάρκεια ζωής του έργου. Για παράδειγμα, στις αρχικές επαναλήψεις ο περισσότερος χρόνος δαπανάται για τις απαιτήσεις του συστήματος, καθώς σε μεταγενέστερες επαναλήψεις περισσότερος χρόνος δαπανάται στην υλοποίηση της εφαρμογής. Πρέπει να θυμόμαστε, ωστόσο, ότι κάθε κλάδος πρέπει να λαμβάνεται υπόψιν σε κάθε επανάληψη.

Σύλληψη απαιτήσεων

Στην ενότητα αυτή προσδιορίζονται οι εργασίες που επιτελεί η εφαρμογή καθώς και ορισμένοι περιορισμοί που ισχύουν. Οι εργασίες αυτές παρουσιάζονται υπό την μορφή λειτουργικών απαιτήσεων, δηλαδή ορίζουν τι πρέπει να κάνει και πως πρέπει να συμπεριφέρεται το σύστημα.

Όπως είχε αναφερθεί και στο πρώτο κεφάλαιο η εφαρμογή υλοποιήθηκε υπό την συνεργασία τριών φοιτητών. Για την αποφυγή της επανάληψης οι απαιτήσεις που παρουσιάζονται αφορούν μόνο τα τμήματα της εφαρμογής που καλύπτει η παρούσα διπλωματική εργασία. Στην συνέχεια παρουσιάζεται ένα μέρος του συνόλου των λειτουργικών απαιτήσεων της εφαρμογής:

1. Η εφαρμογή πρόκειται να είναι ένα κοινωνικό δίκτυο

2. Ένας νέος χρήστης πρέπει να πραγματοποιήσει εγγραφεί στο σύστημα για την απόκτηση λογαριασμού εισόδου
    - Ο λογαριασμός αποτελείται από μία διεύθυνση ηλεκτρονικού ταχυδρομείου και έναν κωδικό

3. Η αρχική οθόνη της εφαρμογής αποτελεί το χρονολόγιο του κάθε χρήστη.

4. Στο αριστερό τμήμα κάθε σελίδας υπάρχει ένα μενού γρήγορης προσπέλασης που θα επιτρέπει:
    - Την γρήγορη εισαγωγή βαθμολογίας για μία ιστοσελίδα
    - Την προβολή των πέντε πιο πρόσφατων δημόσιων βαθμολογιών της εφαρμογής

5. Στην κορυφή κάθε σελίδας υπάρχει ένα κεντρικό μενού από το οποίο ο χρήστης μπορεί να επιλέξει να προβάλει:
    - Την σελίδα εισαγωγής νέας βαθμολογίας
    - Την σελίδα με όλες τις βαθμολογίες που έχει εισάγει στο σύστημα
    - Την σελίδα με τους χρήστες που ακολουθεί
    - Την σελίδα με τους χρήστες που τον ακολουθούν
    - Την σελίδα αναζήτησης καταχωρημένων βαθμολογιών
    - Την σελίδα αναζήτησης χρηστών
    - Τις σελίδες με τις κατατάξεις των βαθμολογιών
    - Την σελίδα με το ιστορικό των ειδοποιήσεων
    - Την σελίδα του προσωπικού του προφίλ

6. Ο χρήστης μπορεί να διαγράψει την βαθμολογία, και την κριτική αν υπάρχει, που έχει καταχωρήσει για μία ιστοσελίδα

7. Πριν την υποβολή μίας φόρμας ο φυλλομετρητής πρέπει να ελέγξει αν όλα τα πεδία έχουν συμπληρωθεί βάσει των προδιαγραφών του συστήματος

8. Οι φόρμες που υποβάλλονται στο σύστημα για την ολοκλήρωση ορισμένων διαδικασιών πρέπει να υποβάλλονται με την μέθοδο Ajax. Οι διαδικασίες αυτές είναι:
    - Εισαγωγή νέας βαθμολογίας και κριτικής
    - Εισαγωγή νέας βαθμολογίας από το μενού γρήγορης προσπέλασης
    - Επεξεργασία υπάρχουσας βαθμολογίας
    - Διαγραφή υπάρχουσας βαθμολογίας
    - Αναζήτηση χρηστών

9. Όταν υποβάλλεται μία βαθμολογία πρέπει να αποθηκεύεται ένα στιγμιότυπο εικόνας για την ιστοσελίδα που καταχωρείται.
    - Εάν έχει υποβληθεί στο παρελθόν από οποιοδήποτε χρήστη βαθμολογία ια την ίδια ιστοσελίδα χρήση του υπάρχοντος στιγμιότυπου
    - Σε αντίθετη περίπτωση δημιουργία καινούργιου

10. Από το κεντρικό μενού ο χρήστης μπορεί να επιλέξει να προβάλει τις παρακάτω λίστες με την κατάταξη των ιστοσελίδων:
    - Λίστα με τις πιο πρόσφατες βαθμολογίες στο σύστημα
    - Λίστα με τις ιστοσελίδες που συγκέντρωσαν την υψηλότερη βαθμολογία
    - Λίστα με τις ιστοσελίδες που συγκέντρωσαν την χαμηλότερη βαθμολογία
    - Λίστα με τις ιστοσελίδες που συγκέντρωσαν τις περισσότερες ψήφους

11. Από οποιαδήποτε σελίδα ο χρήστης μπορεί να λαμβάνει αυτόματα και σε πραγματικό χρόνο ειδοποιήσεις όταν:
    - Ένας χρήστης τον ακολουθεί για πρώτη φορά
    - Βαθμολογείται μία ιστοσελίδα από το προφίλ του από έναν χρήστη που ήδη τον ακολουθεί

12. Η λίστα με τις πέντε πιο πρόσφατες βαθμολογίες στο μενού γρήγορης προσπέλασης ανανεώνονται αυτόματα και σε πραγματικό χρόνο

13. Ο χρήστης όταν βρίσκεται στο χρονολόγιο του (αρχική σελίδα) λαμβάνει αυτόματα και σε πραγματικό χρόνο βαθμολογίες που εισήγαγαν άτομα που ακολουθεί.

Περιπώσεις χρήσης

Οι περιπτώσεις χρήσης περιγράφουν τη συμπεριφορά ενός υστήματος από την οπτική γωνία ενός χρήστη. Επιτρέπουν τον ορισμό των ορίων του συστήματος και του περιβάλλοντος του. Αποτελούν μία εικόνα της λειτουργικότητας του συστήματος το οποίο ενεργοποιείται για να ανταποκριθεί σε έναν εξωτερικό ενεργοποιό. Από την ανάλυση των απαιτήσεων οι περιπτώσεις χρήσης που προκύπτουν είναι:

Περίπτωση χρήσης: Εγγραφή στο σύστημα

Βασική ροή

- Το σύστημα εμφανίζει την οθόνη εισόδου / εγγραφής στον χρήστη
- Ο χρήστης πληκτρολογεί την διεύθυνση ηλεκτρονικού ταχυδρομείου και τον κωδικό του στα αντίστοιχα πεδία και πατάει στο κουμπί “Signup”
- Το σύστημα δημιουργεί έναν νέο χρήστη
- Το σύστημα στέλνει email στην διεύθυνση που εισήγαγε ο χρήστης για την ενεργοποίηση του λογαριασμού
- Ο χρήστης εισέρχεται αυτόματα στο σύστημα και μεταφέρεται στην αρχική του σελίδα

Εναλλακτική ροή 1

- Η διεύθυνση ηλεκτρονικού ταχυδρομίου που εισήγαγε ο χρήστης χρησιμοποιείται ήδη. Ο χρήστης ενημερώνεται με κατάλληλο μήνυμα και απαιτείται εισαγωγή διαφορετικής διεύθυνσης

Εναλλακτική ροή 2

- Κάποιο από τα δύο απαιτούμενα πεδία δεν συμπληρώθηκε ή η συμπλήρωση του δεν ήταν σύμφωνη με τις προδιαγραφές του συστήματος. Ο χρήστης ενημερώνεται με κατάλληλο μήνυμα και απαιτείται η σωστή εισαγωγή των στοιχείων

Περίπτωση χρήσης: Διαγραφή βαθμολογίας

Βασική ροή

- Το σύστημα εμφανίζει την καρτέλα την βαθμολογία και τις υπόλοιπες πληροφορίες για μία ιστοσελίδα
- Ο χρήστης πατάει το κουμπί με το κόκκινο σύμβολο “X” στην κάτω δεξιά γωνία της καρτέλας
- Το σύστημα διαγράφει την βαθμολογία αυτή και αφαιρεί την καρτέλα από τον φυλλομετρητή του χρήστη

Εναλλακτική ροή

- Η βαθμολογία που επιθυμεί να διαγράψει ο χρήστης δεν έχει γίνει από αυτόν: το σύστημα δεν εμφανίζει το κουμπί για την διαγραφή της

Περίπτωση χρήσης: Υποβολή και επικύρωση φόρμας

Βασική ροή

- Το σύστημα εμφανίζει την φόρμα στην χρήστη
- Ο χρήστης πατάει το κουμπί για την υποβολή της φόρμας

- Η φόρμα επικυρώνεται από τον φυλλομετρητή και αποστέλλεται στο σύστημα με την μέθοδο Ajax
- Το σύστημα εκτελεί τις απαραίτητες ενέργειες ανάλογα με την περίπτωση
- Εμφανίζονται στον χρήστη τα αποτελέσματα της υποβολής του: λίστα χρηστών, επιτυχή καταχώρηση βαθμολογίας κτλπ

Εναλλακτική ροή 1

- Τα πεδία της φόρμας δεν συμπληρώθηκαν βάσει των προδιαγραφών του συστήματος με αποτέλεσμα να μην γίνει επιτυχής επικύρωση από τον φυλλομετρητη: Η φόρμα δεν αποστέλλεται και εμφανίζεται κατάλληλο μήνυμα στον χρήστη

Εναλλακτική ροή 2

- Ο χρήστης έχει απενεργοποιήσει την γλώσσα Javascript στον φυλλομετρητή του: Η φόρμα αποστέλλεται και η επικύρωση γίνεται στον εξυπηρετητή. Εάν η υποβολή είναι επιτυχής το σύστημα επιστρέφει την σελίδα μαζί με την επιτυχή εκτέλεση της λειτουργίας από την υποβολή της φόρμας. Σε αντίθετη περίπτωση επιστρέφεται η σελίδα που είναι ο χρήστης μαζί με κατάλληλο μήνυμα

Περίπτωση χρήσης: Καταχώρηση στιγμιότυπου εικόνας ιστοσελίδας

Βασική ροή

- Το σύστημα εμφανίζει στον χρήστη την φόρμα υποβολής βαθμολογίας
- Ο χρήστης εισάγει όλα τα απαραίτητα στοιχεία και υποβάλει την φόρμα
- Το σύστημα καταχωρεί την βαθμολογία και χρησιμοποιεί σαν στιγμιότυπο εικόνας το υπάρχον στιγμιότυπο που είναι αποθηκευμένο για την ιστοσελίδα αυτή
- Ο χρήστης λαμβάνει κατάλληλο μήνυμα για την επιτυχή καταχώρηση της βαθμολογίας

Εναλλακτική ροή 1

- Η ιστοσελίδα που βαθμολόγησε ο χρήστη δεν έχει βαθμολογιθεί ξανά στο παρελθόν: το σύστημα χρησιμοποιεί μία διαδικτυακή εφαρμογή για την απόκτηση του στιγμιότυπου εικόνας και το αποθηκεύει στην βάση

Εναλλακτική ροή 2

- Η ιστοσελίδα που βαθμολόγησε ο χρήστη δεν έχει βαθμολογηθεί ξανά στο παρελθόν και η διαδικτυακή εφαρμογή είναι υπερφορτωμένη από αιτήματα: το σύστημα χρησιμοποιεί μία προεπιλεγμένη φωτογραφία σαν στιγμιότυπο εικόνας

Περίπτωση χρήσης: Προβολή των πιο πρόσφατων βαθμολογιών

- Το σύστημα εμφανίζει στο κεντρικό μενού αναπτυσσόμενη (dropdown) λίστα με τις σελίδες κατάταξης ιστοσελίδων
- Ο χρήστης επιλέγει τον σύνδεσμο για την σελίδα με τις πιο πρόσφατες βαθμολογίες
- Το σύστημα κάνει ανάκτηση των δεδομένων από την βάση δεδομένων
- Το σύστημα εμφανίζει στον χρήστη την σελίδα με τις ιστοσελίδες

Περίπτωση χρήσης: Προβολή των ιστοσελίδων με την υψηλότερη βαθμολογία

- Το σύστημα εμφανίζει στο κεντρικό μενού αναπτυσσόμενη (dropdown) λίστα με τις σελίδες κατάταξης ιστοσελίδων
- Ο χρήστης επιλέγει τον σύνδεσμο για την σελίδα με τις ιστοσελίδες με την υψηλότερη βαθμολογία
- Το σύστημα κάνει ανάκτηση των δεδομένων από την βάση δεδομένων
- Το σύστημα εμφανίζει στον χρήστη την σελίδα με τις ιστοσελίδες

Περίπτωση χρήσης: Προβολή των ιστοσελίδων με την χαμηλότερη βαθμολογία

- Το σύστημα εμφανίζει στο κεντρικό μενού αναπτυσσόμενη (dropdown) λίστα με τις σελίδες κατάταξης ιστοσελίδων
- Ο χρήστης επιλέγει τον σύνδεσμο για την σελίδα με τις ιστοσελίδες με την χαμηλότερη βαθμολογία
- Το σύστημα κάνει ανάκτηση των δεδομένων από την βάση δεδομένων
- Το σύστημα εμφανίζει στον χρήστη την σελίδα με τις ιστοσελίδες

Περίπτωση χρήσης: Προβολή των ιστοσελίδων με τις περισσότερες βαθμολογίες

- Το σύστημα εμφανίζει στο κεντρικό μενού αναπτυσσόμενη (dropdown) λίστα με τις σελίδες κατάταξης ιστοσελίδων
- Ο χρήστης επιλέγει τον σύνδεσμο για την σελίδα με τις ιστοσελίδες με τις περισσότερες βαθμολογίες
- Το σύστημα κάνει ανάκτηση των δεδομένων από την βάση δεδομένων
- Το σύστημα εμφανίζει στον χρήστη την σελίδα με τις ιστοσελίδες

Περίπτωση χρήσης: Λήψη ενημερώσεων στο χρονολόγιο του χρήστη

- Ο χρήστης βρίσκεται στο χρονολόγιο του (αρχική σελίδα)
- Το σύστημα εμφανίζει ένα μήνυμα με το πλήθος των ενημερώσεων για το χρονολόγιο του χρήστη
- Ο χρήστης κάνει κλικ στο μήνυμα αυτό
- Οι ειδοποιήσεις προσαρτώνται στο χρονολόγιο του χρήστη

Περίπτωση χρήσης: Λήψη ειδοποιήσεων

- Ο χρήστης βρίσκεται σε οποιαδήποτε σελίδα του συστήματος
- Το σύστημα εμφανίζει ένα εικονίδιο με το πλήθος των ειδοποιήσεων στην δεξιά μεριά του κεντρικού μενού
- Ο χρήστης επιλέγει το εικονίδιο αυτό
- Το σύστημα εμφανίζει την σελίδα με το ιστορικό όλων των ειδοποιήσεων του χρήστη

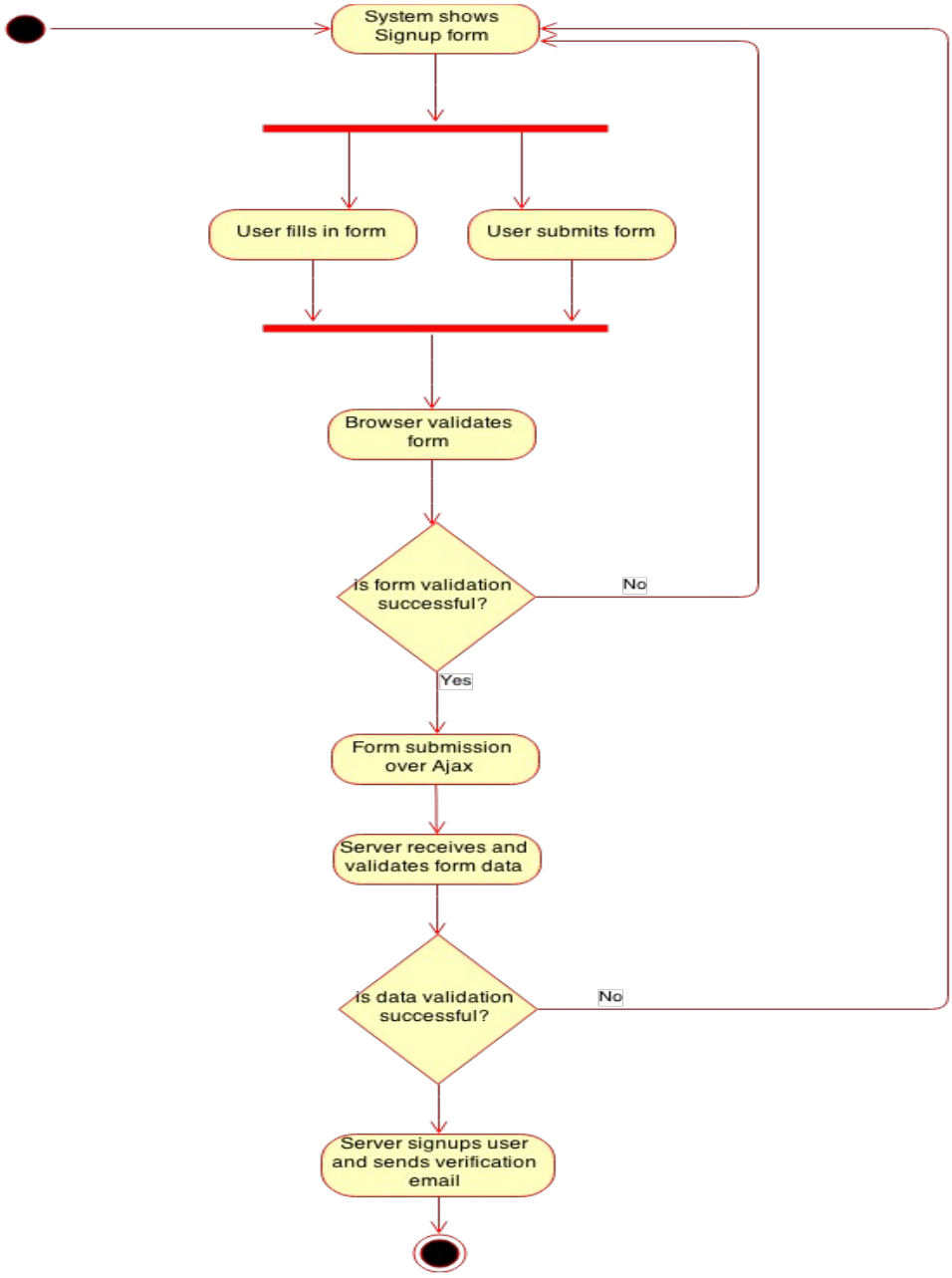
Περίπτωση χρήσης: Λήψη πρόσφατων βαθμολογιών στο μενού γρήγορης προσπέλασης

- Ο χρήστης βρίσκεται σε οποιαδήποτε σελίδα του συστήματος
- Το σύστημα ενημερώνει αυτόματα τις βαθμολογίες στο μενού

Διαγράμματα δραστηριοτήτων

Τα διαγράμματα δραστηριοτήτων χρησιμοποιούνται για την περιγραφή της ροής των εργασιών μίας περίπτωσης χρήσης. Μπορούν να χρησιμοποιηθούν συμπληρωματικά μαζί με μία περιγραφή κειμένου και παρουσιάζουν την γενική λειτουργία μίας οντότητας με βάση απλούστερες λειτουργίες που ονομάζονται δραστηριότητες. Σκοπός των διαγραμμάτων αυτών είναι η εστίαση σε ροές που προκαλούνται από εσωτερική επεξεργασία. Χρησιμοποιούνται σε περιπτώσεις που συμβαίνουν ασύγχρονα συμβάντα ή που η πλειοψηφία των συμβάντων αναπαριστά εσωτερικά παραγόμενες δράσεις. Στην συνέχεια παρουσιάζονται τα διαγράμματα δραστηριοτήτων της παρούσας εφαρμογής.

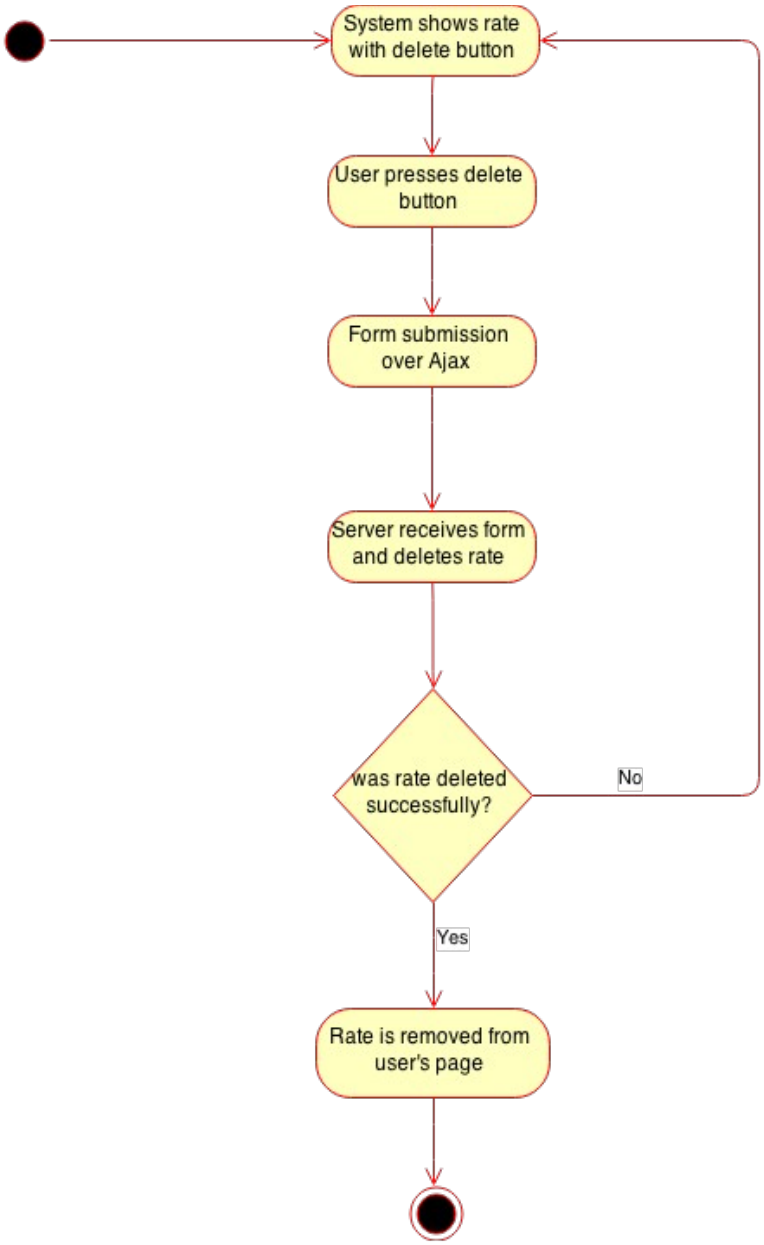
Διάγραμμα δραστηριοτήτων: Εγγραφή στο σύστημα



Στο παραπάνω διάγραμμα παρουσιάζεται η ροή των εργασιών για την εγγραφή στο σύστημα. Αρχικά το σύστημα εμφανίζει την σελίδα με την φόρμα εγγραφής την οποία ο χρήστης πρέπει να συμπληρώσει για να μπορέσει να υποβληθεί επιτυχώς. Πριν την διαδικασία υποβολής ο φυλλομετρητής ελέγχει εάν εχουν συμπληρωθεί και τα δύο πεδία (διεύθυνση ηλεκτρονικού ταχυδρομείου, συνθηματικό), εάν η διεύθυνση ηλεκτρονικού ταχυδρομείου έχει έγκυρη μορφή και εάν το συνθηματικό έχει μήκος τουλάχιστον τεσσάρων χαρακτήρων. Μόλις ολοκληρωθεί επιτυχώς η επικύρωση της φόρμας στέλνεται με την μέθοδο Ajax στον εξυπηρετητή. Στην μεριά του εξυπηρετητή γίνεται πάλι η επικύρωση της φόρμας και στην περίπτωση που είναι επιτυχής πραγματοποιείται η εγγραφή του χρήστη και η αποστολή του email για την ενεργοποίηση του λογαριασμού του. Η διπλή διαδικασία επικύρωσης συμβαίνει για την αποφυγή μη

αποδεκτών δεδομένων καθώς η γλώσσα Javascript μπορεί να είναι απενεργοποιημένη στον φυλλομετρητή του χρήστη με αποτέλεσμα να μην έχει πραγματοποιηθεί η πρώτη επικύρωση. Κάθε αποτυχημένη διαδικασία επικύρωσης έχει σαν αποτέλεσμα την προβολή της φόρμας στον χρήστη μάζι με κατάλληλο μήνυμα

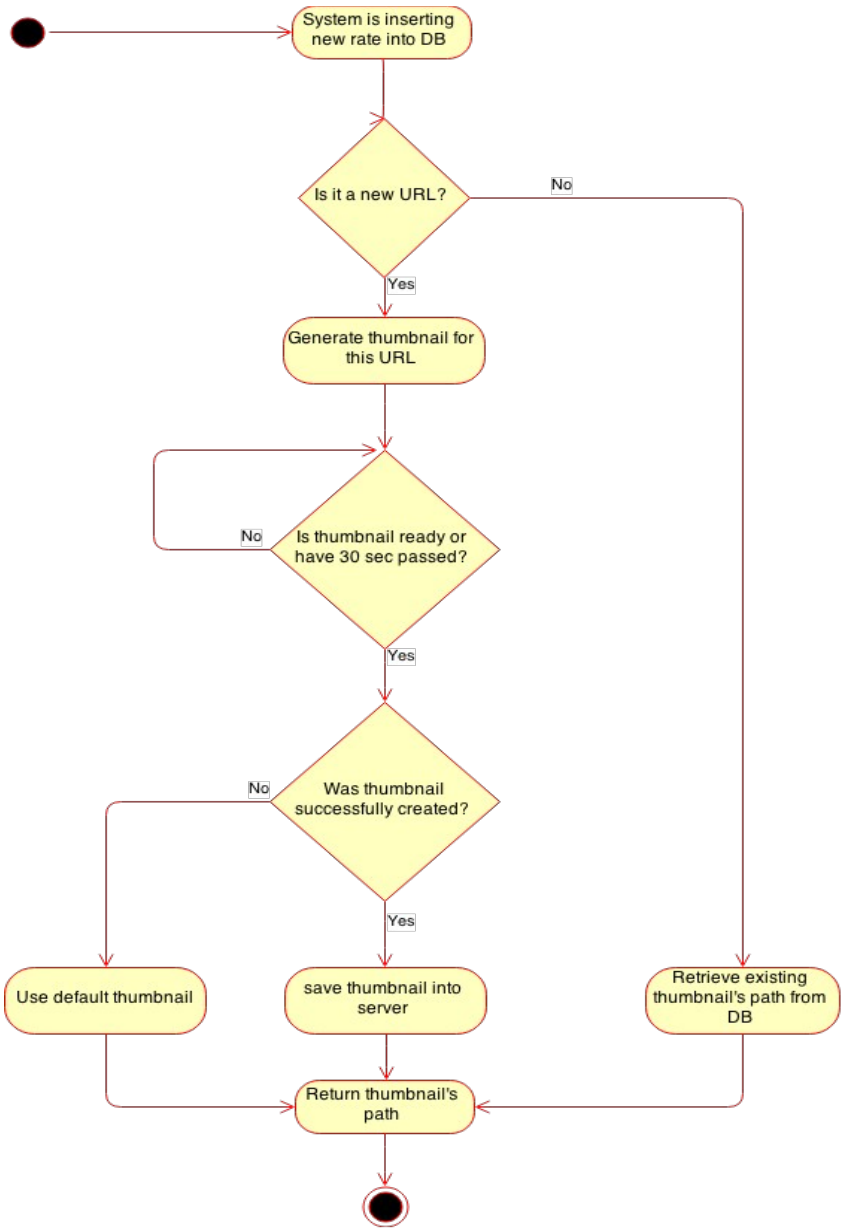
Διάγραμμα δραστηριοτήτων: Διαγραφή βαθμολογίας



Στο παραπάνω διάγραμμα παρουσιάζεται η ροή των εργασιών για την διαγραφή μίας βαθμολογίας του χρήστη από το σύστημα. Αρχικά το σύστημα εμφανίζει μία καρτέλα στην οποία περιέχονται τα στοιχεία για αυτή την βαθμολογία (διεύθυνση, βαθμός, κριτική, ημερομηνία υποβολής, μικρογραφία εικόνας, όνομα χρήστη) μαζί με τα κουμπιά για την επεξεργασία ή την διαγραφή της. Ο χρήστης πατώντας στο

κουμπί για την διαγραφή η φόρμα στέλνεται στον εξυπηρετητή με την μέθοδο Ajax. Το σύστημα εξετάζει την ορθότητα των στοιχείων του χρήστη για την διαγραφή της συγκεκριμένης βαθμολογίας. Εάν ο χρήστης που επέλεξε την διαγραφή της βαθμολογίας είναι ο ίδιος με αυτόν που την είχε καταχωρήσει στο σύστημα τότε η βαθμολογία διαγράφεται επιτυχώς. Κατά την επιτυχή διαγραφή μίας βαθμολογίας ο χρήστης παρατηρεί την καρτέλα της να αφαιρείται από την σελίδα στην οποία βρίσκεται. Σε αντίθετη περίπτωση εμφανίζεται μήνυμα σφάλματος.

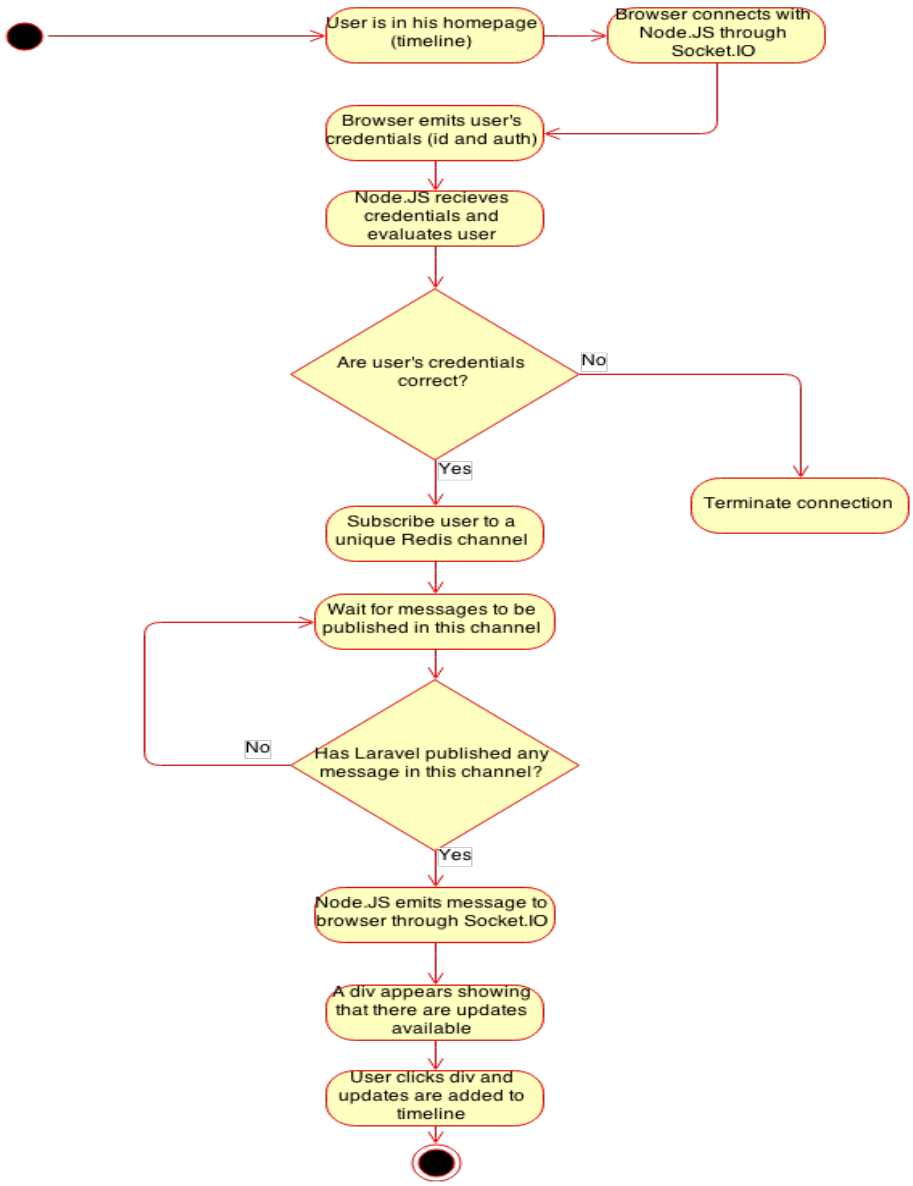
Διάγραμμα δραστηριοτήτων: Καταχώρηση στιγμιότυπου εικόνας ιστοσελίδας



Στο παραπάνω διάγραμμα παρουσιάζεται η ροή των εργασιών για την καταχώρηση του στιγμιότυπου εικόνας μίας ιστοσελίδας. Κατά τη διαδικασία εισαγωγής μίας νέας βαθμολογίας στο σύστημα εξετάζεται

εάν η διεύθυνση (URL) της βαθμολογίας αυτής έχει καταχωρηθεί ξανά στο παρελθόν. Στην περίπτωση που η διεύθυνση υπάρχει ήδη στη βάση δεδομένων τότε ανακτάται και επιστρέφεται το μονοπάτι (path) του υπάρχοντος στιγμιότυπου. Σε αντίθετη περίπτωση το σύστημα χρησιμοποιεί μία διαδικτυακή εφαρμογή για τη δημιουργία του στιγμιότυπου. Μόλις το στιγμιότυπο είναι έτοιμο το σύστημα το αποθηκεύει στον εξυπηρετητή και επιστρέφει το μονοπάτι στην κύρια διαδικασία. Στην περίπτωση που ο χρόνος αναμονής είναι πάνω από τριάντα δευτερόλεπτα ή η διαδικτυακή εφαρμογή δεν μπορεί να εξυπηρετήσει για κάποιο λόγο αιτήματα χρησιμοποιείται ένα προεπιλεγμένο στιγμιότυπο για τη διεύθυνση αυτή.

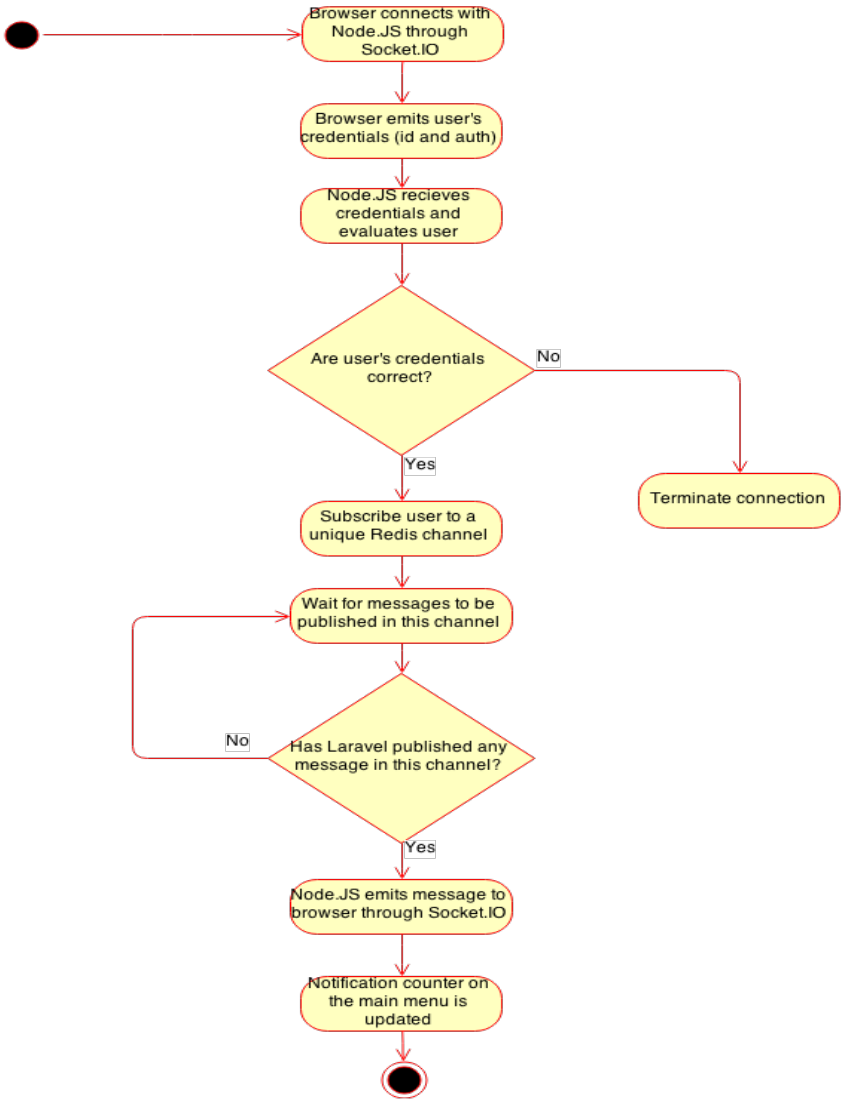
Διάγραμμα δραστηριοτήτων: Λήψη ενημερώσεων στο χρονολόγιο του χρήστη



Στο παραπάνω διάγραμμα παρουσιάζεται η ροή των εργασιών για την αυτόματη ενημέρωση του

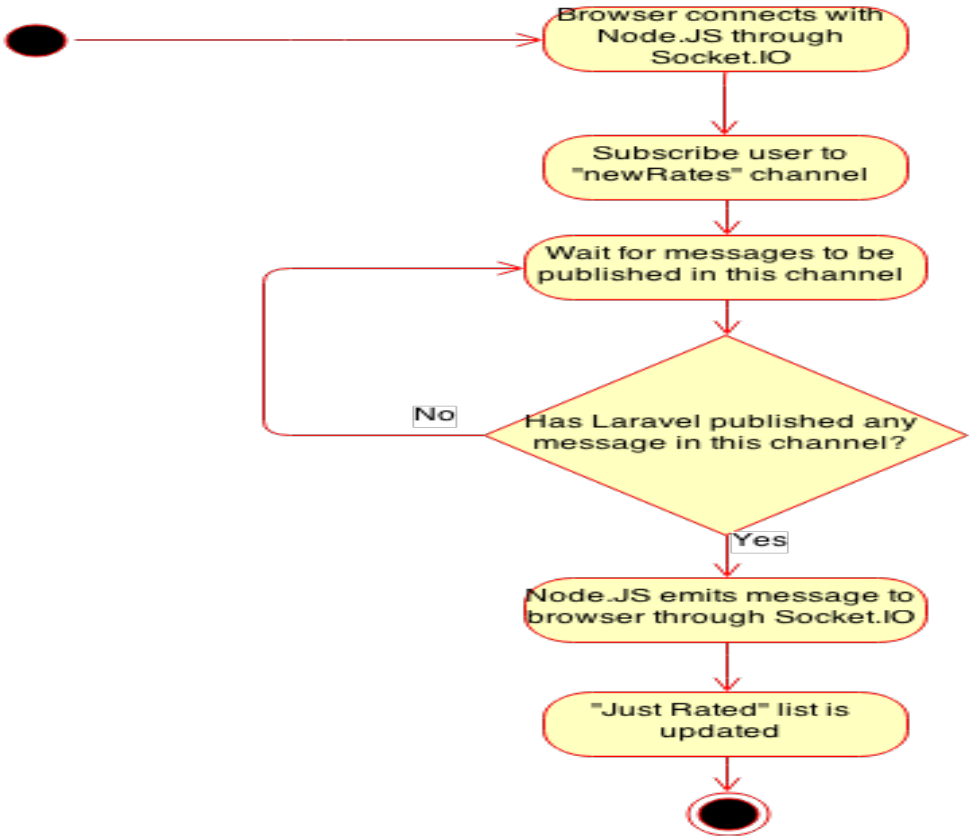
χρονολογίου ενός χρήστη. Αρχικά ο χρήστης πρέπει να συνδεθεί στο σύστημα και να βρίσκεται στο χρονολόγιο του για να ξεκινήσει η διαδικασία σύνδεσης του φυλλομετρητή με τον εξυπηρετητή Node.JS μέσω της βιβλιοθήκης Socket.IO. Μόλις η σύνδεση πραγματοποιηθεί ο φυλλομετρητής στέλνει τα διαπιστευτήρια του χρήστη (ID και cookie) για να μπορέσει να γίνει η ταυτοποίηση αυτού από τον εξυπηρετητή Node.JS. Το Node.JS ανακτά από την Redis τα στοιχεία του χρήστη και εάν ταυτίζονται με εκείνα που του έχει στείλει ο φυλλομετρητής τότε γίνεται η εγγραφή του χρήστη στο προσωπικό του κανάλι. Σε αντίθετη περίπτωση το Node.S τερματίζει την σύνδεση με τον φυλλομετρητή. Στη συνέχεια το Node.JS περιμένει να γίνει η δημοσίευση κάποιου μηνύματος στο κανάλι αυτό από τον εξυπηρετητή HTTP αιτημάτων. Όταν δημοσιευθεί ένα μήνυμα γίνεται η ανάκτηση του από την Redis και στέλνεται στο φυλλομετρητή μέσω του Socket.IO. Με τη λήψη του μηνύματος ο φυλλομετρητής εμφανίζει ένα κατάλληλο μήνυμα σχετικά με το πλήθος των διαθέσιμων ενημερώσεων. Ο χρήστης πατώντας στο μήνυμα αυτό παρατηρεί τις ενημερώσεις να προσαρτώνται στο χρονολόγιο του.

Διάγραμμα δραστηριοτήτων: Λήψη ειδοποιήσεων



Στο παραπάνω διάγραμμα παρουσιάζεται η ροή των εργασιών για την αυτόματη λήψη ειδοποιήσεων. Οι εργασίες ξεκινούν μόλις πραγματοποιηθεί επιτυχής είσοδος του χρήστη στο σύστημα και είναι ακριβώς οι ίδιες με αυτές που πραγματοποιούνται για τη λήψη των ενημερώσεων στο χρονολόγιο του χρήστη. Η μόνη διαφορά είναι ως προς τον τρόπο που επεξεργάζονται τα δεδομένα που στέλνονται από τον Node.JS στον φυλλομετρητή. Μόλις γίνει η παραλαβή τους ο μετρητής των ειδοποιήσεων στην δεξιά μεριά του κεντρικού μενού ανανεώνεται ανάλογα με το πλήθος των διαθέσιμων ειδοποιήσεων. Όταν ο χρήστης πατήσει στον μετρητή αυτό μεταφέρεται στη σελίδα με το ιστορικό όλων των ειδοποιήσεων του.

Διάγραμμα δραστηριοτήτων: Λήψη πιο πρόσφατων βαθμολογιών



Στο παραπάνω διάγραμμα παρουσιάζεται η ροή των εργασιών για την αυτόματη λήψη των πιο πρόσφατων βαθμολογιών. Μόλις πραγματοποιηθεί επιτυχής είσοδος του χρήστη στο σύστημα ο φυλλομετρητής συνδέεται με τον εξυπηρετητή Node.JS μέσω της βιβλιοθήκης Socket.IO. Στην συνέχεια πραγματοποιείται η εγγραφή του χρήστη σε ένα κοινό κανάλι για όλους τους χρήστες. Μόλις καταχωρηθεί μία δημόσια βαθμολογία στο σύστημα γίνεται ταυτόχρονα και η δημοσίευση της στο κανάλι αυτό. Το Node.JS στέλνει το μήνυμα στο φυλλομετρητή όλων των συνδρομητών του καναλιού. Με την λήψη του μηνύματος ο φυλλομετρητής ενημερώνει τη λίστα με τις πιο πρόσφατες βαθμολογίες η οποία βρίσκεται στο μενού γρήγορη προσπέλασης.

Διαγράμματα ακολουθίας

Τα διαγράμματα ακολουθίας είναι διαγράμματα τα οποία αναπαριστούν την αλληλεπίδραση, δηλαδή τον τρόπο που διαφορετικά αντικείμενα συνεργάζονται μεταξύ τους, με σκοπό την πραγματοποίηση μίας επιθυμητής πράξης ή αποτελέσματος. Μέσα από τα διαγράμματα αυτά γίνεται η περιγραφή της εκτέλεσης μίας εργασίας στην οποία λαμβάνουν μέρος αντικείμενα διάφορων συνεργαζόμενων κλάσεων.

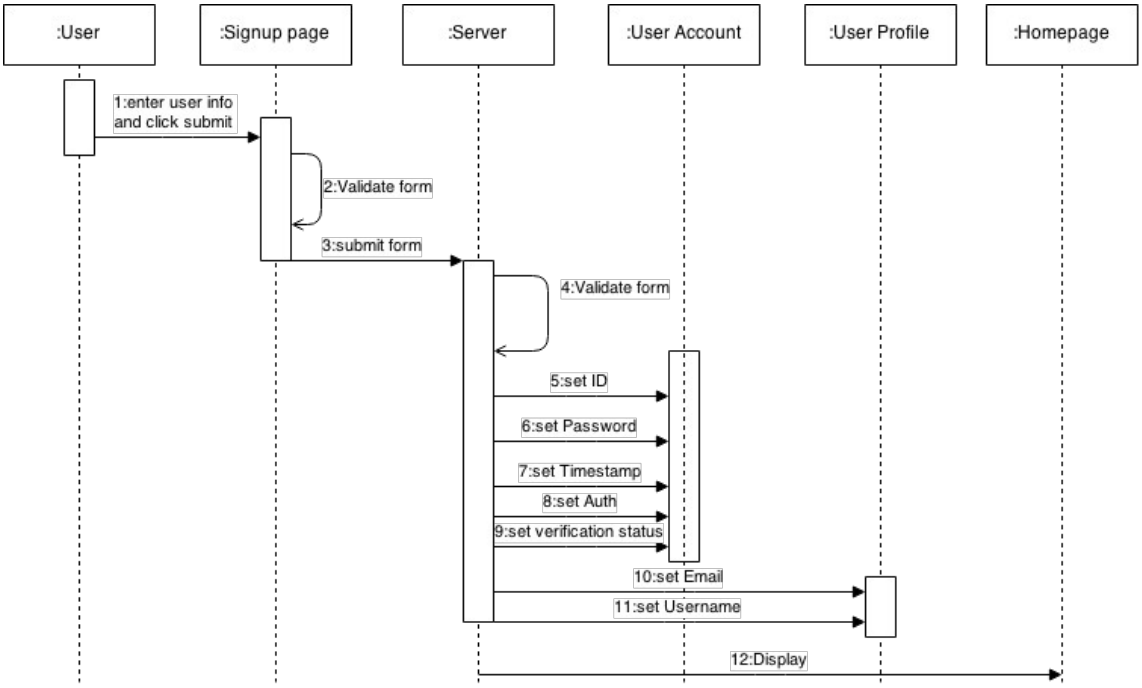
Τα διαγράμματα ακολουθίας αναπαριστώνται από δύο διαστάσεις:

1. Την οριζόντια διάσταση η οποία αναπαριστά τα αντικείμενα των κλάσεων

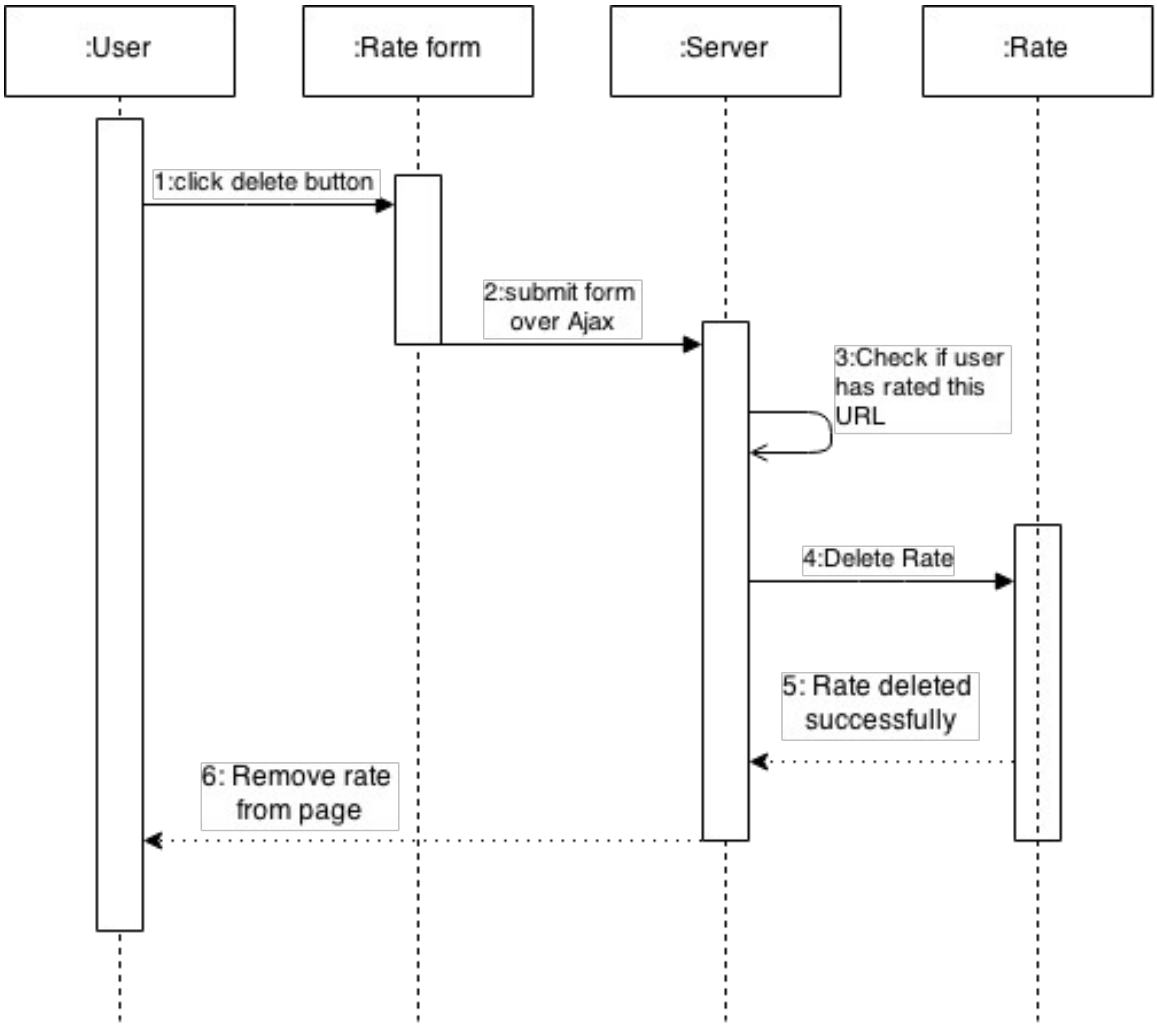
2. Την κάθετη διάσταση η οποία αναπαριστά το χρόνο

Η ροή του χρόνου είναι κάθετη και τα βέλη κλήσεων δείχνουν την μετάβαση από την μια χρονική στιγμή στην άλλη.

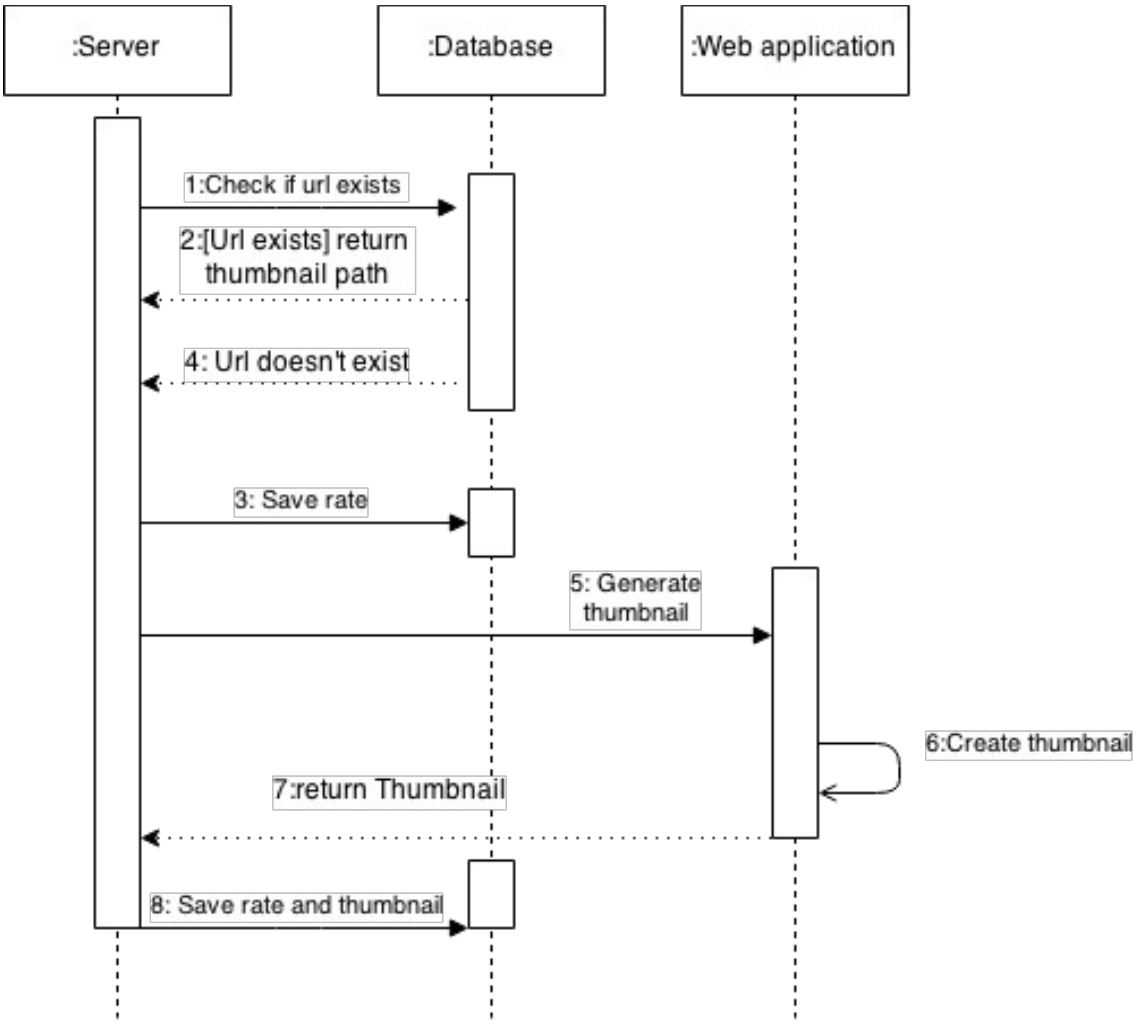
Διάγραμμα ακολουθίας: Εγγραφή στο σύστημα



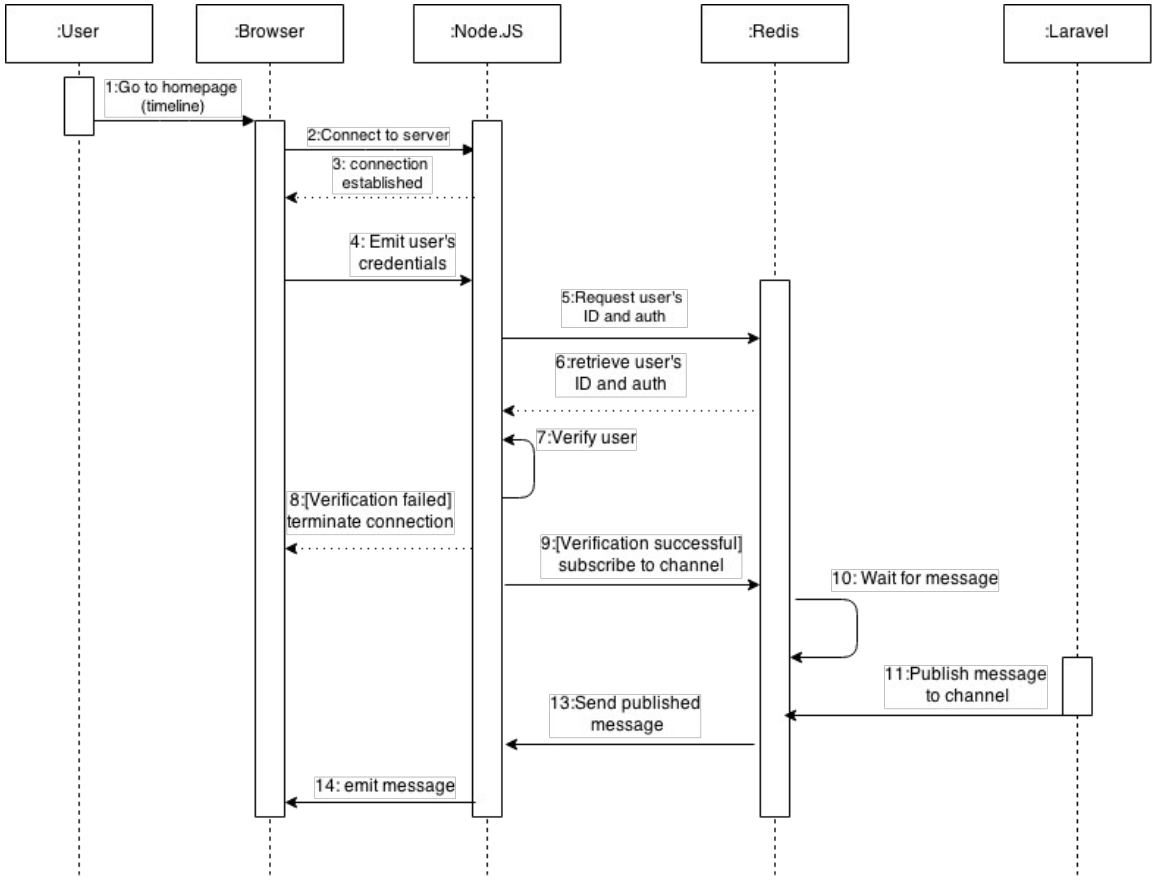
Διάγραμμα ακολουθίας: Διαγραφή βαθμολογίας



Διάγραμμα ακολουθίας: Καταχώρηση στιγμιότυπου εικόνας ιστοσελίδας



Διάγραμμα ακολουθίας: Λήψη ενημερώσεων στο χρονολόγιο του χρήστη



Διάταξη δεδομένων

Η βάση δεδομένων Redis παρέχει πολύ μεγάλη ευελιξία ως προς τον τρόπο που μπορούν να διαταχθούν τα δεδομένα της. Το γεγονός αυτό σε συνδυασμό με τον μεγάλο όγκο δεδομένων που προκύπτει από ένα κοινωνικό δίκτυο μας ανάγκασε να ψάξουμε το πιο αποδοτικό και γρήγορο μοντέλο αποθήκευσης.

Για τις ανάγκες της εφαρμογής σχεδιάστηκαν και υλοποιήθηκαν δύο μέθοδοι διάταξης δεδομένων (data layout). Η διάταξη που χρησιμοποιείται για την αποθήκευση δεδομένων που αφορούν τον χρήστη είναι ίδια και στις δύο μεθόδους. Βασικό χαρακτηριστικό των δύο μεθόδων είναι η αξιοποίηση των δομών δεδομένων που παρέχει η Redis για την διαχείριση και αποθήκευση της διεύθυνσης μίας ιστοσελίδας (URL) και της βαθμολογίας (rate) της. Η διαφορά που διέπει τις δύο μεθόδους είναι εάν θα πρέπει η διεύθυνση αυτή και η βαθμολογία που δέχεται από κάθε χρήστη να αποτελούν μία οντότητα ή να είναι ξεχωριστές.

Διάταξη δεδομένων: Διεύθυνση και βαθμολογία ξεχωριστές οντότητες

Σύμφωνα με την μέθοδο αυτή όσες πληροφορίες αφορούν μία διεύθυνση ιστοσελίδας θα πρέπει να αποτελούν ξεχωριστή οντότητα από τις πληροφορίες που αφορούν μία βαθμολογία για την διεύθυνση αυτή. Με τον τρόπο αυτό γίνεται διαχωρισμός των πληροφοριών σχετικά με έναν πόρο (resource) του

διαδικτύου και των πληροφοριών που εισάγει ο χρήστης για τον πόρο αυτό. Πιο συγκεκριμένα η μέθοδος αυτή αποτελείται από την εξής διάταξη:

Resource		
Redis Hash με πληροφορίες για το resource		
Key	Field	Value
rid:xxx	url	string
	thumbnail	string
	timestamp	float
	title	string
	average_score	float

Rate		
Redis Hash με πληροφορίες για το rate ενός resource		
Key	Field	Value
rate:xxx	uid	<uid>
	rid	<rid>
	score	float
	review	string
	timestamp	float
	private	boolean

Διάταξη δεδομένων: Διεύθυνση και βαθμολογία κοινή οντότητα

Σύμφωνα με την μέθοδο αυτή οι πληροφορίες για μία διεύθυνση καθώς και η βαθμολογία της αποτελούν μία οντότητα. Το αποτέλεσμα είναι σε ένα Redis hash να υπάρχουν οι πληροφορίες για τον πόρο του διαδικτύου και οι πληροφορίες που έχει καταχωρήσει ο χρήστης για τον πόρο αυτό. Πιο συγκεκριμένα η μέθοδος αυτή αποτελείται από την εξής διάταξη:

Rid		
Redis Hash με πληροφορίες για το resource και το rate		
Key	Field	Value
	url	string
	thumbnail	string
	timestamp	float
	title	string
	average_score	float
	uid	<uid>
	rid	<rid>
	score	float
	review	string
	timestamp	float
	private	boolean

Σύγκριση μεθόδων

Τα πλεονεκτήματα που παρουσιάζει η ύπαρξη ξεχωριστού hash για τις δύο αυτές οντότητες είναι πως είναι δυνατός ο διαχωρισμός των πληροφοριών του πόρου από τις πληροφορίες που εισήγαγε ο χρήστης. Με τον τρόπο αυτό για την αλλαγή μίας πληροφορίας (πχ μέσος όρος) ενός πόρου απαιτείται μόνο η επεξεργασία του αντίστοιχου πεδίου στο hash και όχι η επεξεργασία όλων των hash. Το βασικό μειονέκτημα της μεθόδου αυτής είναι πως χρειάζονται επιπλέον πληροφορίες για την αντιστοίχιση κάθε rate με τον πόρο που αντιπροσωπεύει.

Τα πλεονεκτήματα που παρουσιάζει η ύπαρξη κοινού hash για τον πόρο του διαδικτύου και την βαθμολογία είναι πως το σύνολο των πληροφοριών βρίσκεται συγκεντρωμένο σε ένα σημείο και η ανάκτηση του δεν χρειάζεται καμία επιπλέον διαδικασία. Ωστόσο το βασικό μειονέκτημα της μεθόδου αυτής είναι η ύπαρξη της ίδιας πληροφορίας για έναν πόρο σε κάθε βαθμολογία που έχει δημιουργηθεί για αυτόν. Με τον τρόπο αυτό για την ανανέωση μίας πληροφορίας για έναν πόρο (πχ μέσος όρος) απαιτείται η επεξεργασία όλων των hash που σχετίζονται με τον πόρο αυτό.

Το μοντέλο που επιλέχθηκε τελικά για την διάταξη των δεδομένων της παρούσας εφαρμογής είναι το δεύτερο. Η απόδοση και των δύο διατάξεων όσων αναφορά την ταχύτητα εκτέλεσης τους όσο και τον χώρο που καταλαμβάνουν ήταν παρόμοιες. Το βασικό κριτήριο που συντέλεσε στην επιλογή αυτή ήταν η χρήση ενός μοντέλου το οποίο θα διέφερε από την λογική ύπαρξης μίας οντότητας για κάθε στοιχείο της βάσης η οποία συνηθίζεται με τις σχεσιακές βάσεις δεδομένων.

Σύνοψη

Στο κεφάλαιο αυτό παρουσιάστηκαν τα επιμέρους τμήματα που αποτελούν την αρχιτεκτονική της παρούσας εφαρμογής.

Στη συνέχεια παρουσιάστηκε η γλώσσα μοντελοποίησης UML, η διαδικασία RUP και παρουσιάστηκε η διαδικασία της σύλληψης των απαιτήσεων που διέπουν την εφαρμογή αυτή.

Επίσης παρουσιάστηκε η διαδικασία ανάλυσης και σχεδιασμού των λειτουργιών που κάλυψε η παρούσα εργασία μέσα από περιπτώσεις χρήσης, διαγραμμάτων ακολουθίας και δραστηριοτήτων.

Τέλος παρουσιάστηκων οι δύο μέθοδοι διάταξης δεδομένων που σχεδιάστηκαν και υλοποιήθηκαν καθώς και μία συνοπτική σύγκριση τους.

Στο επόμενο κεφάλαιο παρουσιάζονται τα συμπεράσματα από την εκπόνηση της παρούσας διπλωματικής, ορισμένες ιδέες που δεν υλοποιήθηκαν για να υπάρχει εμβάθυνση πέρα από το θέμα που αφορά στην εφαρμογή καθώς και μελλοντικές προσθήκες και βελτιώσεις

Επίλογος

Στο κεφάλαιο αυτό παρουσιάζονται τα συμπεράσματα που εξήχθησαν και η εμπειρία που αποκτήθηκε από την υλοποίηση της παρούσας διπλωματικής εργασίας. Επίσης αναφέρονται λειτουργίες που δεν υλοποιήθηκαν στην παρούσα έκδοση της εφαρμογής καθώς ήταν ιδιαίτερα απαιτητικές και θα δινόταν εμβάθυνση πέραν των ορίων της παρούσας εργασίας. Τέλος παρουσιάζονται και ιδέες και προτάσεις για μελλοντικές βελτιώσεις που επιδέχεται η εφαρμογή.

Συμπεράσματα

Ο σκοπός της παρούσας εργασίας ήταν η ανάλυση, ο σχεδιασμός και η υλοποίηση ενός κοινωνικού δικτύου το οποίο θα μπορούσε να ανταποκριθεί στις υψηλές απαίτησεις που χαρακτηρίζουν τις σύγχρονες διαδικτυακές εφαρμογές. Για τον λόγο αυτό δόθηκε ιδιαίτερη βάση στον τομέα της αρχιτεκτονικής των δεδομένων, τόσο στον τρόπο αποθήκευσης και ανάκτησης τους όσο και στην ροή τους μεταξύ πελάτη και εξυπηρετητή σε πραγματικό χρόνο. Η εκπόνηση της εργασίας αυτής είχε σαν αποτέλεσμα την γνωριμία με σύγχρονες τεχνολογίες και το συνδυασμό τους για την επίτευξη του καλύτερου δυνατού αποτελέσματος.

Η μελέτη των τρόπων ανάπτυξης διαδικτυακών εφαρμογών σε συνδυασμό με την κατάλληλη καθοδήγηση που υπήρχε συντέλεσαν στη γνωριμία και εξοικείωση με την διαδικασία σχεδιασμού πληροφοριακών συστημάτων. Η αρχιτεκτονική δόμησης εφαρμογών Μοντέλου Παρουσίασης Ελεγκτή (MVC) συνέβαλε στην κατανόηση του συνόλου των λειτουργιών που απαρτίζουν μία εφαρμογή και στην πιο αποτελεσματική αξιοποίηση των δυνατοτήτων τους.

Η χρήση του Git έθεσε νέο ορίζοντα ως προς τον τρόπο διαχείρισης των αρχείων σε ένα έργο, την οργάνωση τους και την αποθήκευση τους σε απομακρυσμένους χώρους. Αποτελεί το εργαλείο που θα πρέπει να χρησιμοποιεί κάθε σύγχρονος προγραμματιστής ακόμα και όταν ασχολείται σε προσωπικό επίπεδο με το πιο απλό έργο.

Το Laravel αποτελεί αναμφίβολα το δημοφιλέστερο και πληρέστερο PHP πλαίσιο εργασίας. Το σύνολο των δυνατοτήτων που διαθέτει καθώς και ο τρόπος που τις διαχειρίζεται κάνουν τη διαδικασία δημιουργίας εφαρμογών αποτελεσματική και αξιόπιστη. Επίσης η συνεχής αναάθμιση του από τους δημιουργούς του συντελεί στην ενσωμάτωση των πιο πρόσφατων λειτουργιών που μπορούν να βρεθούν σε ένα πλαίσιο εργασίας. Ενδεικτικά η παρούσα εργασία υλοποιήθηκε με την έκδοση 4.2.11 και κατά την ολοκλήρωση της ήταν ήδη διαθέσιμη η έκδοση 5.0.

Η μη σχεσιακή βάση δεδομένων Redis αποτελεί την καρδία της εφαρμογής. Σε ένα κοινωνικό δίκτυο όπου οι χρήστες και τα δεδομένα ολοένα και αυξάνονται είναι απαραίτητη η ταχύτατη διεκπεραίωση όλων των διαδικασιών εγγραφής και ανάκτησης δεδομένων για την βελτιστοποίηση του συνόλου της απόδοσης. Επίσης το μοντέλο Δημοσίευση/Συνδρομή που ενσωματώνει αποτελεί την ιδανική λύση για την υλοποίηση συστημάτων αποστολής μηνυμάτων σε συγκεκριμένους χρήστες, ιδιαίτερα όταν ο αριθμός αυτών είναι αρκετά μεγάλος.

Η πλατφόρμα Node.JS σε συνδυασμό με την βιβλιοθήκη Socket.IO αποτελεί την ιδανική λύση για την επίτευξη ταυτόχρονης αμφίδρομης επικοινωνίας μεταξύ πελάτη και εξυπηρετητή με σκοπό την ανταλλαγή δεδομένων σε πραγματικό χρόνο. Το πρωτόκολλο WebSocket της HTML5 καθιστά δυνατή την επικοινωνία αυτή βάζοντας στο περιθώριο τεχνολογίες και μεθόδους , όπως το Comet και το Ajax, με τις οποίες μέχρι πρότινος ήταν δυνατή η μερική προσομοίωση της.

Το βασικό πλεονέκτημα της παρούσας εργασίας ήταν η δυνατότητα δημιουργίας μίας ομάδας όπου μέσα από την αρμονική και αποτελεσματική συνεργασία θα πραγματοποιούταν η ανάπτυξη ενός σύγχρονου κοινωνικού δικτύου. Με τον τρόπο αυτό κάθε μέλος της ομάδας αποτίναξε την νοοτροπία της ατομικής συγγραφής ενός έργου και έγινε μέρος ενός συνόλου με κοινό σκοπό και στόχο. Η συνεργασία αυτής βελτίωσε τον τρόπο επικοινωνίας μεταξύ των μελών και έκανε τη διαδικασία σχεδιασμού ενός έργου εύκολη και ευχάριστη. Επίσης ο καταμερισμός του μεγάλου όγκου των εργασιών σε συνδυασμό με τις εβδομαδιαίες προθεσμίες αύξησαν το αίσθημα ευθύνης του κάθε μέλους, βελτίωσαν τις οργανωτικές του

ικανότητας και προήγαγαν το ομαδικό πνεύμα.

Συνοψίζοντας αξίζει να σημειωθεί πως ανεξάρτητα από την ενσωμάτωση σύγχρονων τεχνολογιών σε διάφορους τομείς της εφαρμογής το αποτέλεσμα δεν θα ήταν τόσο ικανοποιητικό εάν δεν υπήρχε άψογη συνεργασία, συνεχή ομαδική προσπάθεια και κατάλληλη καθοδήγηση.

Περιορισμοί της παρούσας εργασίας

Παρά το γεγονός πως η παρούσα εφαρμογή αποτελεί ένα πλήρες λειτουργικό κοινωνικό δίκτυο υπάρχει ένα σύνολο λειτουργιών που δεν υλοποιήθηκαν. Ο λόγος για τον οποίων δε συμπεριλήφθηκαν οι λειτουργίες αυτές στην παρούσα έκδοση είναι πως χαρακτηρίζονται από εξαιρετική πολυπλοκότητα με αποτέλεσμα η έκταση της παρούσας εργασίας να ξεπερνούσε το επιθυμητό όριο. Οι λειτουργίες αυτές παρουσιάζονται συνοπτικά στην συνέχεια.

Μία δυνατότητα που είχε συμπεριληφθεί κατά την φάση σχεδιασμού της εφαρμογής ήταν ένα σύστημα παροχής κατάλληλων συστάσεων (recommendation system) στους χρήστες. Με το σύστημα αυτό θα ήταν δυνατή η προτροπή στους χρήστες πιθανών ιστοσελίδων που θα τους ενδιέφερε να βαθμολογήσουν. Οι ιστοσελίδες αυτές θα προέκυπταν βάσει των προτιμήσεων του κάθε χρήστη και θα βασίζονταν στη δραστηριότητά του στα πλαίσια της εφαρμογής. Επίσης θα ήταν δυνατή η προτροπή ατόμων που ίσως ενδιαφερόταν ο χρήστης να ακολουθήσει και η οποία θα βασιζόταν τόσο στις προτιμήσεις του όσο και στις προτιμήσεις των ατόμων που ήδη ακολουθεί.

Μία ακόμα δυνατότητα είναι ένα σύστημα κατηγοριοποίησης των ιστοσελίδων. Με το σύστημα αυτό θα ήταν δυνατή η απόδοση ετικετών σε κάθε ιστοσελίδα ανάλογα με το περιεχόμενο της και στη συνέχεια η κατηγοριοποίηση της βάσει αυτών. Η υλοποίηση του θα βοηθούσε τόσο στην πιο αποτελεσματική λειτουργία του συστήματος παροχής κατάλληλων συστάσεων όσο και στην πιο αποδοτική διαχείριση και αναζήτηση των ιστοσελίδων από τους χρήστες.

Η τελευταία λειτουργία που δεν υλοποιήθηκε στην παρούσα έκδοση της εφαρμογής είναι ένα σύστημα ανταλλαγής μηνυμάτων μεταξύ των χρηστών με σκοπό την παροχή ενός τρόπου για την μεταξύ τους επικοινωνία. Το σύστημα αυτό θα παρείχε στο κάθε χρήστη ένα πλήρες ιστορικό του συνόλου των συνομιλιών του.

Μελλοντικές βελτιώσεις

Μία σημαντική βελτίωση θα ήταν η δημιουργία μίας σειράς ενεργειών παρασκηνίου κατά την καταχώρηση μίας βαθμολογίας για μία ιστοσελίδα. Με τον τρόπο αυτό διαδικασίες όπως ο υπολογισμός του νέου μέσου όρου, η λήψη του στιγμιότυπου εικόνας καθώς και η προώθηση της βαθμολογίας στην λίστα των ακολούθων του χρήστη θα μπορούν να γίνουν ασύγχρονα χωρίς να επιβραδύνουν τον χρόνο ολοκλήρωσης της διαδικασίας καταχώρησης. Επίσης μία ακόμα βελτίωση θα ήταν η χρήση της μεθόδου Ajax για την πλειοψηφία των αιτημάτων που στέλνονται στον εξυπηρετητή καθώς και για την διαδικασία εγγραφής και σύνδεσης του χρήστη.

Οι μελλοντικές προσθήκες άμεσης προτεραιότητας είναι όσες αναφέρθηκαν στην προηγούμενη ενότητα. Ωστόσο υπάρχουν κάποιες ακόμα οι οποίες θα συνέβαλαν στην βελτιστοποίηση της εφαρμογής. Πιο συγκεκριμένα η πλατφόρμα Node.JS μαζί με την βιβλιοθήκη Socket.IO θα μπορούσαν να χρησιμοποιηθούν για τη βελτίωση της διαδικασίας ανταλλαγής μηνυμάτων μεταξυ των χρηστών που αναφέρθηκε και την αναβάθμιση της σε σε ένα σύστημα ανταλλαγής μηνυμάτων σε πραγματικό χρόνο (Instant messaging, IM).Μία ακόμα προσθήκη θα ήταν η δυνατότητα χρήσης του πρωτοκόλλου ανοιχτής ταυτοποίησης (Open Authentication, Oauth) βάσει του οποίου θα ήταν δύνατή η είσοδος στην εφαρμογή χωρίς την ανάγκη δημιουργίας νέου λογαριασμού αλλά με τη χρήση των διακριτικών ενός υπάρχοντος. Ο λογαριασμός αυτός θα μπορεί να προέρχεται από μία άλλη εφαρμογή κοινωνικής δικτύωσης, όπως το Facebook, Twitter και Google Plus, και με τον τρόπο αυτό οι χρήστες θα μπορούν να ταυτοποιηθούν χωρίς να είναι αναγκασμένοι μοιραστούν τα διαπιστευτήρια τους. Τέλος η εφαρμογή θα ενσωματώνει ένα σύνολο διαθέσιμων φυσικών γλωσσών καθώς και έναν μηχανισμό εντοπισμού της χώρας προέλευσης του χρήστη με σκοπό την προεπιλογή της καταλληλότερης γλώσσας. Ανεξάρτητα από την προεπιλεγμένη

γλώσσα ο χρήστης θα έχει την δυνατότητα της αλλαγής της με σκοπό την προβολή του περιεχομένου της εφαρμογής σύμφωνα με τις προσωπικές του προτιμήσεις.

Σύνοψη

Στο κεφάλαιο αυτό παρουσιάστηκε το σύνολο των εμπειριών που αποκομίστηκε από την εκπόνηση της παρούσας διπλωματικής εργασίας. Επίσης παρουσιάστηκαν τα συμπεράσματα που εξήχθησαν καθώς και ιδέες για ορισμένες βελτιώσεις και προσθήκες που επιδέχεται η εφαρμογή.

Με την ενότητα αυτή ολοκληρώνεται το πέμπτο και τελευταίο κεφάλαιο της παρούσας διπλωματικής εργασίας. Ακολουθεί το σύνολο των βιβλιογραφικών πηγών που μελετήθηκαν καθώς και το παράρτημα στο οποίο είναι διαθέσιμος ο πηγαίος κώδικας.

Βιβλιογραφία

BOYD, D. M. and ELLISON, N. B., (2007) *Social Network Sites: Definition, History, and Scholarship* [Online] Volume 13, Issue 1, pages 210–230, Available from: http://onlinelibrary.wiley.com/doi/10.1111/j.1083-6101.2007.00393.x/full [Accessed: 6th December 14]

CHACON, S. and STRAUB, B. (2014) *Pro Git* 2nd Ed. s.l.: Apress

CHAPMAN, C., (2009) *The History and Evolution of Social Media* [Online] Available from: http://www.webdesignerdepot.com/2009/10/the-history-and-evolution-of-social-media/ [Accessed: 5th December 14]

CRANE, D. and MCCARTHY, P. (2008) *Comet and Reverse Ajax: The Next-Generation Ajax 2.0*. s.l.: Apress

Design and implementation of a simple Twitter clone using only the Redis key-value store as database and PHP (2011) [Online] Available from: http://redis.io/topics/twitter-clone [Accessed: 2th August 14]

DIGITAL TRENDS (2014) *The History of Social Networking* [Online] Available from: http://www.digitaltrends.com/features/the-history-of-social-networking/ [Accessed: 8th December 14]

GAFITESCU, D. (2013) *Goodbye CodeIgniter, Hello Laravel* [Online] Available from: http://www.sitepoint.com/goodbye-codeigniter-hello-laravel/ [Accessed: 10th December 14]

HUBER, S., (2012) *The real-time web* [Online] Available from: http://community.aiim.org/blogs/serge-huber/2012/05/10/the-real-time-web [Accessed: 2th December 14]

HUGHES-CROUCHER, T. and WILSON, M. (2012) *Node: Up and Running* Sebastopol: O'Reilly Media

JOHANAN, J. (2014) *Building Scalable Apps with Redis and Node.js* Birmingham:Packt Publishing

KING, A. B. (2008) *Website Optimization* Sebastopol: O'Reilly Media

LARAVEL (2014) *Documentation* [Online] Available from: http://laravel.com/docs/4.2 [Accessed: 10th July 14]

MCMURTRY, D. et al. (2013) *Data Access for Highly-Scalable Solutions: Using SQL, NoSQL, and Polyglot Persistence* s.l.: Microsoft

MOZZILA DEVELOPER NETWORK (2014) *Ajax* [Online] Available from: https://developer.mozilla.org/en-US/docs/AJAX [Accessed: 13th jult 14]

NODEJS (2014) *Documentation* [Online] Available from: http://nodejs.org/documentation/ [Accessed: 20th October 14]

REDIS (2014) *Documentation* [Online] Available from: http://redis.io/documentation
[Accessed: 15th July 14]

REES, D. (2013) *Laravel: Code Bright* s.l.: Leanpub

SCHRÖTER, J., (2011) *Client-side web performance optimizations* [Online] Available from: http://www.slideshare.net/jakob.schroeter/clientside-web-performance-optimization
[Accessed: 5th September 14]

SEGUIN, K. (2014) *The little Redis Book* [Online] Available from: https://github.com/karlseguin/the-little-redis-book
[Accessed: 8th August 14]

SKVORC, B. (2013) *Best PHP Frameworks for 2014* Available from: http://www.sitepoint.com/best-php-frameworks-2014/
[Accessed: 10th December 14]

TEIXEIRA, P. (2013) *Professional Node.js®: Building JavaScript-Based Scalable Software* Indianapolis:John Wiley & Sons

ΒΙΡΒΟΥ, Μ. Κ. [No date] *Αντικειμενοστρεφής Τεχνολογία Λογισμικού* Πειραιάς: Εκδόσεις Βαρβαρήγου

Παράρτημα

Μοντέλα

BaseModel.php

```php
<?php

    /*
     * Model for global declaration of Redis connection on models.
     * Every model that uses redis connection must extends BaseModel.
     */



    class BaseModel
    {
        protected $Redis;

        public function __construct() {

            $this->Redis = Redis::connection();
        }

    }

```

FollowerGraph.php

```php
<?php


    /*
     *  Follower Graph actions
     */

    class FollowerGraph extends BaseModel
    {
        public function __construct()
        {
            parent::__construct(); //current constructor inherit parent from BaseModel
        }


        public function followAction($data)
        {
            $this -> Redis -> zadd('followers:'.$data['foreign_uid'], microtime(true), $data['current_uid']);   //add current user in sorted set named followers:*  uid*
            $this -> Redis -> zadd('followings:'.$data['current_uid'], microtime(true), $data['foreign_uid']);  //add foreign user in sorted set named followings:* uid*

            $User     = new User();
            $userData = $User->getUserData($data['foreign_uid']);

            // give values because every view with follow/unfollow action contains variables below
            $userData['followings'] = TRUE;
            $userData['sameuser']   = FALSE;

            // Increase notifications counter
            $this -> Redis -> incr('uncheckedNotifications:'. $data['foreign_uid']);
            // Add notification to user's sorted set
            $this -> Redis -> zadd('notifications:' . $data['foreign_uid'], microtime(true), $data['current_uid'].':follows you');
            // If "foreign user" is online notify him
            if (!empty($this -> Redis -> hget('uid:' . $data['foreign_uid'], 'auth')))
            {
                $notification = array(
                    'uid' => $data['foreign_uid'],
                );
                Event::fire('cometNotification', array($notification));
            }

            return $userData;
        }


        public function unFollowAction($data)
        {
            $this -> Redis -> zrem('followers:'.$data['foreign_uid'], $data['current_uid']);  //remove current user from sorted set named followers:*  uid*
            $this -> Redis -> zrem('followings:'.$data['current_uid'], $data['foreign_uid']); //remove foreign user from sorted set named followings:*  uid*
            // Remove notification from sorted set
            $this -> Redis -> zrem('notifications:' . $data['foreign_uid'], $data['current_uid'].':follows you');

            $User     = new User();
            $userData = $User->getUserData($data['foreign_uid']);

            $userData['followings'] = FALSE;
            $userData['sameuser']   = FALSE;

            $rids = $this -> Redis -> lrange('own_ratings:'.$data['foreign_uid'], '0', '-1');   //get all rating from the followed user

            foreach ($rids as $rid)
            {
                $this->Redis->lrem('timeline_ratings:'.$data['current_uid'], '0', $rid);    //remove all ratings(above action) from follower user
            }

            return $userData;
        }

        /*
         *  Retrieve followers from database. Use $uid for each user and $currentUid
         *  for giving value to same user index e.g. When results of action "show uid followers"
         *  contains the signed in user(current) then need to give value to same user variable (hide follow/unfollow button)
         */
        public function getFollowers($uid, $currentUid, $start)
        {
            $semistop  = $start + 4;

            //get followers of $uid
            $followers = $this -> Redis -> zrange('followers:'.$uid, $start, $semistop);

            $recordsNo = $this -> Redis -> zcard('followers:'.$uid); // count records

            if (empty($followers))
```

```
82          {
83              //for error message
84              return $results = FALSE;
85          }
86
87          // instance of User model
88          $User = new User();
89
90          foreach ($followers as $fUid)
91          {
92              // method of user model. get data of user with id the given argument
93              $results[$fUid] = $User->getUserData($fUid);
94
95              $isfolloiwng    =  $this -> Redis -> zscore('followings:'.$uid, $fUid);
96
97              if (!empty($isfolloiwng))
98              {
99                  $results[$fUid]['followings'] = TRUE;
100             }
101             else
102             {
103                 $results[$fUid]['followings'] = FALSE;
104             }
105
106             if ($currentUid == $fUid)
107             {
108                 $results[$fUid]['sameuser'] = TRUE;
109             }
110             else
111             {
112                 $results[$fUid]['sameuser'] = FALSE;
113             }
114         }
115
116         if ($semistop >= $recordsNo - 1)
117         {
118             return array(
119
120                 'results'   => $results,
121                 'semistop'  => FALSE,
122                 'uid'       => $uid
123             );
124         }
125
126         return array(
127
128             'results'   => $results,
129             'semistop'  => $semistop,
130             'uid'       => $uid
131         );
132     }
133
134
135     public function getFollowings($uid, $currentUid, $start)
136     {
137         $semistop  = $start + 4;
138
139         $followings = $this -> Redis -> zrange('followings:'.$uid, $start, $semistop);
140
141         $recordsNo = $this -> Redis -> zcard('followings:'.$uid);
142
143         if (empty($followings))
144         {
145             return $results = FALSE;
146         }
147
148         $User = new User();
149
150         foreach ($followings as $fUid)
151         {
152             $results[$fUid] = $User->getUserData($fUid);
153
154             $results[$fUid]['followings'] = TRUE;
155
156             if ($currentUid == $fUid)
157             {
158                 $results[$fUid]['sameuser'] = TRUE;
159             }
160             else
161             {
162                 $results[$fUid]['sameuser'] = FALSE;
163             }
164         }
165
166         if ($semistop >= $recordsNo - 1)
```

```
167             {
168                 return array(
169 
170                     'results'   => $results,
171                     'semistop'  => FALSE,
172                     'uid'       => $uid
173                 );
174             }
175 
176             return array(
177 
178                 'results'   => $results,
179                 'semistop'  => $semistop,
180                 'uid'       => $uid
181             );
182         }
183     }
184 
185 
186 
187 
```

Rate.php

```php
<?php

class Rate extends BaseModel
{

    public function __construct()
    {
        parent::__construct();
    }

    /* Function newRate($rdata)
     * Inserts a new rate in db
     * Input: $rdata-> array of new rate data
     */
    public function newRate($rdata)
    {
        //print_r($rdata);return;
        //get unique rid
        $rid = $this -> Redis -> incr('nextRateId');

        //if capture thumb plugin fails store into rid thumb field the below
        if (!$rdata['thumb'])
        {
            $this->Redis->hset('rid:' . $rid, 'thumb', '/images/thumbnails/NoPhotoAvailable.jpg');
        }
        else
        {
            // call static method from libraries/StorageFactory.php and get
            // the appropriate instance based on config item placed app/config/app.php
            $storage = StorageFactory::getStorage();

            $dataR['rid']       =  $rid;
            $dataR['thumbUrl']  =  $rdata['thumb'];

            $storage->storeRateThumb($dataR);
        }

        //insert data to hashes and sets
        //hash    rid:xxx   url     title  score   e.t.c.
        $this -> Redis -> hmset('rid:' . $rid, 'url', $rdata['url'], 'title', $rdata['title'], 'timestamp', microtime(true), 'uid', $rdata['uid'], 'score', $rdata['score'], 'review', $rdata['review'], 'public', $rdata['public']);
        //hash    url:<string>   uid  rid
        $this -> Redis -> hset('url:' . $rdata['url'], $rdata['uid'], $rid);
        //hash    url:<string>:rscores   rid  score
        $this -> Redis -> hset('url:' . $rdata['url'] . ':rscores', $rid, $rdata['score']);
        //hash    url:<string>:uscores   uid  score
        $this -> Redis -> hset('url:' . $rdata['url'] . ':uscores', $rdata['uid'], $rdata['score']);
        //set     uid:xxx:rated_urls    set-of-urls
        $this -> Redis -> sadd('uid:' . $rdata['uid'] . ':rated_urls', $rdata['url']);
        //set     uid:xxx:rids          set of rids
        $this -> Redis -> sadd('uid:' . $rdata['uid'] . ':rids', $rid);
        // add the rid into the score indexing set
        $this -> Redis -> sadd('score:' . $rdata['score'], $rid);
        //if rate is public, add it to the public rates set:
        if ($rdata['public'])
        {
            $this -> Redis -> sadd('uid:' . $rdata['uid'] . ':publicRids', $rid);
            $this -> addToChart($rdata['url']);
            Event::fire('cometNewRate', $rid);
        }
        //take care of the average score
        if ($rdata['newUrl'])//new rate on non-existing url
        {
            //average score = score
            $this -> Redis -> set('url:' . $rdata['url'] . ':average_score', $rdata['score']);
        }
        else//case = existingURL:new rate on existing url
        {
            //compute new average
            $this -> average_score($rdata['url']);
        }
        //push rate to user's + followers lists
        $this -> fanout($rdata['uid'], $rid, 'new', $rdata['public']);

        //add the rid to the keywords index sets)
        if (array_key_exists('keywords', $rdata))
        {
            foreach ($rdata['keywords'] as $keyword)
            {
                //add the rid in the set of this word
                $this -> Redis -> sadd('word:' . $keyword, $rid);
            }
```

```
82          }
83
84          return TRUE;
85      }//end-of-newRate method
86
87      ////////////////////////////////////////////////////////////////////////////////////////////////
88
89      /*
90       * Function edit($data)
91       * Edits a Rate
92       * Input: array of edited rate data
93       * Updates a rate (new score,review*,timestamp) and repushes it to lists (timelines + own)
94       */
95      public function edit($data)
96      {
97
98          // get rate id - if needed
99          if (empty($data['rid']))
100         {
101             $data['rid'] = $this -> Redis -> hget('url:' . $data['url'], $data['uid']);
102         }
103         if ($data['public'] === "old")// edit case from quick rate sidebar form
104         {
105             //retrieve the old value
106             $data['public'] = $this -> Redis -> hget('rid:' . $data['rid'], 'public');
107         }
108         //check if new review is given (if not, old one stays)
109         if (empty($data['review']))
110         {
111             //retrieve the old review
112             $data['review'] = $this -> Redis -> hget('rid:' . $data['rid'], 'review');
113         }
114         //get the old score
115         $oldScore = $this -> Redis -> hget('rid:' . $data['rid'], 'score');
116         //remove the rid form the old score index set:
117         $this -> Redis -> srem('score:' . $oldScore, $data['rid']);
118         //add the rid to the new score index set
119         $this -> Redis -> sadd('score:' . $data['score'], $data['rid']);
120         //update rid hash fields
121         $this -> Redis -> hmset('rid:' . $data['rid'], 'score', $data['score'], 'review', $data['review'], 'timestamp', microtime(true), 'public', $data['public']);
122         //update new score to hashes 2,3
123         $this -> Redis -> hset('url:' . $data['url'] . ':rscores', $data['rid'], $data['score']);
124         $this -> Redis -> hset('url:' . $data['url'] . ':uscores', $data['uid'], $data['score']);
125         //compute new average
126         $this -> average_score($data['url']);
127         //fanout edited rate
128         $this -> fanout($data['uid'], $data['rid'], 'edit', $data['public']);
129         //if public, add it to user's set of public rids (if already there, nothing happens)
130         if ($data['public'])
131         {
132             $this -> Redis -> sadd('uid:' . $data['uid'] . ':publicRids', $data['rid']);
133             $this -> addToChart($data['url']);
134         }
135         else//remove from public set the rid
136         {
137             $this -> Redis -> srem('uid:' . $data['uid'] . ':publicRids', $data['rid']);
138         }
139
140         return TRUE;
141     }
142
143     ////////////////////////////////////////////////////////////////////////////////////////////////
144
145     /*
146      * Function fanout($uid, $rid, $case, $public)
147      * Input: user Id, rate Id, $case -> edit or new rate(on existing or not-existing url), $public (e.g. only public rates shoyl be fanned out in others timelines)
148      * Pushes a new/edited rate to user's and followers lists
149      */
150     private function fanout($uid, $rid, $case, $public)
151     {
152
153         //push rate to user's lists
154         if ($case == 'edit')
155         {
156             //remove rate from user's lists to repush it
157             $this -> Redis -> lrem('own_ratings:' . $uid, '0', $rid);
158             $this -> Redis -> lrem('timeline_ratings:' . $uid, '0', $rid);
159         }
160         //(re)push rate
161         $this -> Redis -> lpush('own_ratings:' . $uid, $rid);
162         $this -> Redis -> lpush('timeline_ratings:' . $uid, $rid);
163
164         //Get user's followers and push rate to their timelines
165         $followers = $this -> Redis -> zrange('followers:' . $uid, '0', '-1');
166
```

```
167         if (!empty($followers))
168         {
169             switch ($case)
170             {
171                 case 'edit' :
172                     foreach ($followers as $follower_uid)
173                     {
174                         //remove rid from timeline
175                         $this -> Redis -> lrem('timeline_ratings:' . $follower_uid, '0', $rid);
176                         //if is public, repush it
177                         if ($public)
178                         {
179                             $this -> Redis -> lpush('timeline_ratings:' . $follower_uid, $rid);
180                             $isOnline = $this -> Redis -> hget('uid:' . $follower_uid, 'auth');
181                             if (!empty($isOnline))
182                             {
183                                 $update = array(
184                                     'updateTimeline' => $follower_uid,
185                                     'ridBase' => $rid
186                                 );
187                                 Event::fire('cometTimelineUpdate', array($update));
188                             }
189                         }
190 
191                     }
192                     break;
193 
194                 case 'new' :
195                     //if new rate is public, push it to all followers' timelines
196                     if ($public)
197                     {
198                         foreach ($followers as $follower_uid)
199                         {
200                             $this -> Redis -> lpush('timeline_ratings:' . $follower_uid, $rid);
201                             $isOnline = $this -> Redis -> hget('uid:' . $follower_uid, 'auth');
202                             // If follower is online update timeline
203                             if (!empty($isOnline))
204                             {
205                                 $update = array(
206                                     'updateTimeline' => $follower_uid,
207                                     'ridBase' => $rid
208                                 );
209                                 Event::fire('cometTimelineUpdate', array($update));
210                             }
211                         }
212                     }
213 
214                     break;
215             }
216 
217         }
218     }
219 
220     ////////////////////////////////////////////////////////////////////////////////////////////////
221 
222     /* Function average_score($url)
223      * Computes and updates average score for a resource
224      * (needs to run on edit rate + new rate on existing url)
225      */
226     private function average_score($url)
227     {
228         //get all the scores for the url
229         $scores = $this -> Redis -> hvals('url:' . $url . ':rscores');
230         //get the number of scores
231         $sum = $this -> Redis -> hlen('url:' . $url . ':rscores');
232         $average = array_sum($scores) / $sum;
233         $average = number_format($average, 1, '.', '');
234         $this -> Redis -> set('url:' . $url . ':average_score', $average);
235     }
236 
237     ////////////////////////////////////////////////////////////////////////////////////////////////
238 
239     /* Function search($sdata)
240      * Takes search data (criteria,uid e.t.c.)  passed from Rate Controller
241      * Returns multidimensional array $searchResults holding the rating objects (rid:xxx hashes) as arrays
242      */
243     public function search($sdata)
244     {
245         //initialize search results array (will be returned to controller)
246         $searchResults = false;
247         //$rates = false;
248         //redis keyword-index is in form : 'word:keyword [set of rids]' so we add the string 'word:' before each keyword the controller sent
249         array_walk($sdata['keywords'], function(&$value)
250         {
251             $value = 'word:' . $value;
```

```
252     });
253     //store ALL the rids associated with every keyword to a temporary redis set (will be deleted after intersection(s) needed for the search)
254     //this temp set needs to be unique (otherwise conflicts may arise when 2+ users search simultaneously)
255     $this -> Redis -> sunionstore($sdata['uid'] . ':temp1', $sdata['keywords']);
256     //e.g. 3:temp1 (set of rids) where 3=the id of the user
257     // if no rids found , return false
258     if (!$this -> Redis -> scard($sdata['uid'] . ':temp1'))
259     {
260         return false;
261     }
262     //repeat the above process to retrieve rids based on score range, IF search is advanced search
263     if (array_key_exists('scores', $sdata))
264     {
265         //e.g. if score[0]=2, score[1]=5, we want the range of scores in an array: 2,3,4,5
266         $scores = range($sdata['scores'][0], $sdata['scores'][1]);
267 
268         //add the string 'score:' before each score (to get them in the form of redis index keys score:x)
269         array_walk($scores, function(&$value, $key)
270         {
271             $value = 'score:' . $value;
272         });
273         //get all the rids within score range in a temp set
274         $this -> Redis -> sunionstore($sdata['uid'] . ':temp3', $scores);
275         //intersect the 2 temp sets to get only the rids matching the keywords AND the score range (and store the result in uid:temp1 set)
276         $this -> Redis -> sinterstore($sdata['uid'] . ':temp1', $sdata['uid'] . ':temp3', $sdata['uid'] . ':temp1');
277         //delete the temp3 set
278         $this -> Redis -> del($sdata['uid'] . ':temp3');
279     }
280 
281     switch ($sdata['searchIn'])
282     {
283         case '"own"' :
284             //intersect user's own rids with rids associated with search keywords (and scores if advanced search)
285             $rids = $this -> Redis -> sinter('uid:' . $sdata['uid'] . ':rids', $sdata['uid'] . ':temp1');
286             rsort($rids);
287             break;
288 
289         case '"followings"' :
290             //get all followings' uids
291             $fuids = $this -> Redis -> zrange('followings:' . $sdata['uid'], 0, -1);
292 
293             //convert the above uids in 'uid:xxx:publicRids' -> the key of the set with the public rids of each user
294             array_walk($fuids, function(&$value, $key)
295             {
296                 $value = 'uid:' . $value . ':publicRids';
297             });
298 
299             //put all followings' public rids in a temp redis set
300             $this -> Redis -> sunionstore($sdata['uid'] . ':temp2', $fuids);
301 
302             //intersect followings' public rids with rids associated with search keywords
303             $rids = $this -> Redis -> sinter($sdata['uid'] . ':temp2', $sdata['uid'] . ':temp1');
304             rsort($rids);
305 
306             break;
307         case '"both"' :
308             //get all followings' uids
309             $fuids = $this -> Redis -> zrange('followings:' . $sdata['uid'], 0, -1);
310 
311             //convert the above uids(xxx) in 'uid:xxx:publicRids' -> the key of the set with the public rids of each user
312             array_walk($fuids, function(&$value, $key)
313             {
314                 $value = 'uid:' . $value . ':publicRids';
315             });
316 
317             //gather all  followings' public rids in a temp redis set
318             $this -> Redis -> sunionstore($sdata['uid'] . ':temp2', $fuids);
319 
320             //add to this set the user's own rids (public or not)
321             $this -> Redis -> sunionstore($sdata['uid'] . ':temp2', $sdata['uid'] . ':temp2', 'uid:' . $sdata['uid'] . ':rids');
322 
323             //intersect own+followings' rids with rids associated with search keywords
324             $rids = $this -> Redis -> sinter($sdata['uid'] . ':temp2', $sdata['uid'] . ':temp1');
325             rsort($rids);
326 
327             break;
328     }//end-of-switch
329 
330     //delete the remaining temp redis sets
331     $this -> Redis -> del($sdata['uid'] . ':temp1');
332     $this -> Redis -> del($sdata['uid'] . ':temp2');
333 
334     $i = 0;
335 
336     foreach ($rids as $rid)
```

```
337         {
338             //insert the rid object (array) into the searchResults array that will be returned to the controller
339             $searchResults[$i] = $this -> Redis -> hgetall('rid:' . $rid);
340             //print_r($searchResults[$i]);return;
341             //add rid, avg score and following's username
342             $searchResults[$i]['rid'] = $rid;
343             $searchResults[$i]['average_score'] = $this -> Redis -> get('url:' . $this -> Redis -> hget('rid:' . $rid, 'url') . ':average_score');
344             $searchResults[$i]['username'] = $this -> Redis -> hget('uid:' . $searchResults[$i]['uid'], 'username');
345             //number of total ratings on this resource
346             $searchResults[$i]['count'] = $this -> Redis -> hlen('url:' . $searchResults[$i]['url']);
347             //check if user has rated it (we need to show this info on other's ratings, e.g. 'You have rated this')
348             if ($this -> Redis -> sismember('uid:' . $sdata['uid'] . ':rated_urls', $searchResults[$i]['url']))
349             {
350                 $searchResults[$i]['metoo'] = true;
351             }
352 
353             $i++;
354         }
355         return $searchResults;
356     }
357 
358     ////////////////////////////////////////////////////////////////////////////////
359     public function delete($rid, $user)
360     {
361         // Retrieve url
362         $url = $this -> Redis -> hget('rid:' . $rid, 'url');
363         // Retrieve uid
364         $uid = $this -> Redis -> hget('rid:' . $rid, 'uid');
365         // Retrieve thumb url
366         $thumb  = $this -> Redis -> hget('rid:' . $rid, 'thumb');
367 
368         $rate['title'] = $this -> Redis -> hget('rid:' . $rid, 'title');
369         $rate['average_score'] = $this -> Redis -> get('url:' . $this -> Redis -> hget('rid:' . $rid, 'url') . ':average_score');
370         // Check if user is on of the raters
371         if ($uid != $user)
372         {
373             return FALSE;
374         }
375         // count raters
376         $countRaters = $this -> Redis -> hlen('url:' . $url);
377         $rate['raters'] = $countRaters;
378         // Commands associated with Delete rate, see use cases 4 & 5 (_Data_Layout/Use_Cases - Queries.doc)
379         //if rate is public, delete it from the user's set of public rids
380         if ($this -> Redis -> hget('rid:' . $rid, 'public'))
381         {
382             $this -> Redis -> srem('uid:' . $uid . ':publicRids', $rid);
383         }
384         $redis = $this -> Redis -> pipeline();
385         $redis -> del('rid:' . $rid);
386         $redis -> hdel('url:' . $url, $uid);
387         $redis -> hdel('url:' . $url . ':rscores', $rid);
388         $redis -> hdel('url:' . $url . ':uscores', $uid);
389         $redis -> lrem('own_ratings:' . $uid, 0, $rid);
390         $redis -> lrem('timeline_ratings:' . $uid, 0, $rid);
391         $redis -> srem('uid:' . $uid . ':rated_urls', $url);
392         $redis -> srem('uid:' . $uid . ':rids', $rid);
393         $redis -> execute();
394 
395         if ($countRaters == 1)
396         {
397             $this -> Redis -> del('url:' . $url . ':average_score');
398 
399             if (strpos($thumb, 's3') !== FALSE)
400             {
401                 $thumbName = explode('com/', $thumb);
402 
403                 $s3obj  = AWS::get('s3');
404 
405                 $result = $s3obj->deleteObject(array(
406 
407                     'Bucket' => 'ratethumb',
408                     'Key'    => $thumbName[1]
409                 ));
410             }
411             else
412             {
413                 $thumbName = explode('/', $thumb);
414 
415                 unlink('images/thumbnails/'.$thumbName[3]);
416             }
417         }
418         else
419         {
420             $this -> average_score($url);
421         }
```

```
422         $followers = $this -> Redis -> zrange('followers:' . $uid, 0, -1);
423         foreach ($followers as $follower)
424         {
425             $this -> Redis -> lrem('timeline_ratings:' . $follower, 0, $rid);
426             // remove notification from users sorted set
427             $notificationRid = $this -> Redis -> hget('url:' . $url, $follower);
428             $this -> Redis -> zrem('notifications:' . $follower, $uid . ':rated:' . $notificationRid);
429         }
430         $this -> removeFromChart($url);
431         return array('rate' => $rate);
432     }
433
434     private function addToChart($url)
435     {
436         $votes = $this -> Redis -> hlen('url:' . $url . ':rscores');
437         if ($votes >= 4)
438         {
439             $average_score = $this -> Redis -> get('url:' . $url . ':average_score');
440             $this -> Redis -> zadd('scoreBasedChart', $average_score, $url);
441         }
442         $this -> Redis -> zadd('popularityBasedChart', $votes, $url);
443
444     }
445
446     private function removeFromChart($url)
447     {
448         $redis = $this -> Redis -> pipeline();
449         $redis -> zrem('scoreBasedChart', $url);
450         $redis -> zrem('popularityBasedChart', $url);
451         $redis -> execute();
452     }
453
454 }
455
```

User.php

```php
<?php

class User extends BaseModel {

    public function __construct()
    {
        parent::__construct();
    }

    /*
     * Return user data with id the givewn argument
     */
    public function getUserData($uid)
    {
        $uid_data['uid']        = $uid;
        $uid_data['fname']      = $this->Redis->hget ('uid:'.$uid, 'fname');
        $uid_data['lname']      = $this->Redis->hget ('uid:'.$uid, 'lname');
        $uid_data['email']      = $this->Redis->hget ('uid:'.$uid, 'email');
        $uid_data['sex']        = $this->Redis->hget ('uid:'.$uid, 'sex');
        $uid_data['age']        = $this->Redis->hget ('uid:'.$uid, 'age');
        $uid_data['thumb']      = $this->Redis->hget ('uid:'.$uid, 'thumb');
        $uid_data['username']   = $this->Redis->hget ('uid:'.$uid, 'username');
        $uid_data['verified']   = $this->Redis->hget ('uid:'.$uid, 'verified');
        $uid_data['ratings']    = $this->Redis->llen ('own_ratings:'.$uid);
        $uid_data['numOfFollowers']  = $this->Redis->zcard('followers:'.$uid);
        $uid_data['numOfFollowings'] = $this->Redis->zcard('followings:'.$uid);

        return $uid_data;
    }

    // Method called from app/controllers/UserController.php
    public function signup($formData)
    {
        // Get next user id
        $id = $this -> Redis -> incr( 'next_user_id' );

        // Generate username from email
        $formData['username'] = strstr($formData['email'], '@', TRUE);

        // Build the validation constraint set.
        $rules = array(

            'username' => 'already_exists:username'
        );

        // Create a new validator instance
        $validator = Validator::make( $formData, $rules );

        if ( $validator -> fails() )
        {
        // Append username with id
            $formData['username'] .= $id;
        }

        Cookie::queue('auth', $formData['auth'], time() + 3600 * 24 * 365);

        // Save to Database
        $this -> Redis -> hmset('uid:'.$id,

            'email'         , $formData['email'],
            'password'      , $formData['password'],
            'username'      , $formData['username'],
            'verifyCode'    , $formData['verifyCode'],
            'timestamp'     , microtime(true),
            'verified'      , 0,
            'auth'          , $formData['auth']
        );

        $metaphone = new DoubleMetaPhone;
        $keywords  = $metaphone->getKeywords($formData['username']);

        // A way to retrieve uid from username, email, auth and verifyCode.
        $this -> Redis -> hset('username', $formData['username'], $id);
        $this -> Redis -> hset('email', $formData['email'], $id);
        $this -> Redis -> hset('auth', $formData['auth'], $id);
        $this -> Redis -> hset('verifyCode', $formData['verifyCode'], $id);
        //required for search user fanctionality
        $this -> Redis -> sadd('user:'.$formData['username'], $id);
    }

    /*
```

```
82      * Implement pagianation on search user. use $start
83      * argument to know form where to start the pagination
84      */
85     public function pagination($start, $currentUid)
86     {
87         $semiStop   = $start + 4; //show 5 records on every page of panination
88
89         // get from the temporary sorted set the records defined by start & semistop
90         $resultUids = $this -> Redis ->zrange('search', $start, $semiStop);
91
92         $recordsNo  = $this -> Redis ->zcard('search'); // count records
93
94         // prepare uid results to be ready for the controller
95         foreach ($resultUids as $uid)
96         {
97             //check if current user is same with search user
98             if ($uid == $currentUid)
99             {
100                 $results[$uid]['sameuser']  = TRUE; //first index of $results is the user id. Important to seperate users data.
101             }
102             else $results[$uid]['sameuser'] = FALSE;
103
104             //retrieve and strore in $results array each user data
105             $results[$uid]['uid']       = $uid;
106             $results[$uid]['username']  = $this -> Redis -> hget('uid:'.$uid, 'username');
107             $results[$uid]['lname']     = $this -> Redis -> hget('uid:'.$uid, 'lname');
108             $results[$uid]['fname']     = $this -> Redis -> hget('uid:'.$uid, 'fname');
109
110             //initialize variable for follow relationship
111             $results[$uid]['followings']  = FALSE;
112
113             //check if user from search result exist in current user followings sorted set
114             $existInFollowings = $this->Redis->zscore('followings:'.$currentUid, $uid);
115
116             //get own_ratings number of the user
117             $results[$uid]['ratings'] = $this->Redis->llen ('own_ratings:'.$uid);
118
119             //check if current user have followigns
120             if ($this->Redis->exists('followings:'.$currentUid))
121             {
122                 if (!empty($existInFollowings))
123                 {
124                     $results[$uid]['followings']  = TRUE;
125                 }
126             }
127         }
128
129         // remove or set (as mark up) the form who call pagination method.
130         if ($semiStop >= $recordsNo - 1)
131         {
132             return array(
133
134                 'results' => $results,
135                 'semistop'   => FALSE
136             );
137         }
138
139         return array(
140
141             'results' => $results,
142             'semistop'   => $semiStop
143         );
144     }
145
146     /*
147      * Serach user action
148      */
149     public function search($data)
150     {
151         $results = FALSE;
152         $count   = 0;
153         $start   = 0;
154         // zunionstore doesn't accept multidimensional array. So need to
155         // separate cases for one or two coming strings from search field
156         if (sizeof($data['keywords'] ) == 1)
157         {
158             $this -> Redis -> zunionstore('search', 1, $data['keywords'][0]);
159         }
160         else
161         {
162             $this -> Redis -> zunionstore('search', 2, $data['keywords'][0], $data['keywords'][1]);
163         }
164
165         if (!$this->Redis->exists('search') )
166         {
```

```
167             return $results;
168         }
169
170         $results = $this->pagination($start, $data['uid']);
171
172         return array(
173
174             'results'   => $results['results'],
175             'semistop'  => $results['semistop']
176         );
177     }
178
179     /*
180      * make changes on signed in user profile data after edit action
181      */
182     public function editProfile($uData)
183     {
184         $oldValues[]  = $this -> Redis -> hget('uid:'.$uData['uid'], 'username');
185         $oldValues[]  = $this -> Redis -> hget('uid:'.$uData['uid'], 'fname');
186         $oldValues[]  = $this -> Redis -> hget('uid:'.$uData['uid'], 'lname');
187
188         $newValues[]  = $uData['username'];
189         $newValues[]  = $uData['fname'];
190         $newValues[]  = $uData['lname'];
191
192         //names of hash fields in which foreach loop will set the new values
193         $nameFields[] = 'username';
194         $nameFields[] = 'fname';
195         $nameFields[] = 'lname';
196
197         $metaphone = new DoubleMetaPhone;
198
199         /*
200          * foreach loop makes changes for every field if have to
201          */
202         foreach ($newValues as $key => $newVal)
203         {
204             //check if the user has change the profile fields
205             if ($newVal !== $oldValues[$key])
206             {
207                 // when metaphone get empty variable return empty array
208                 // this cause exception error 'variable undefined' when use the metaphone result
209                 if (!empty($oldValues[$key]))
210                 {
211                     $oldKeywords  = $metaphone->getKeywords($oldValues[$key]);
212                     $this -> Redis -> srem('users:'.$oldKeywords[0], $uData['uid']); //remove old metaphone
213                 }
214
215                 if (!empty($newVal))
216                 {
217                     $keywords = $metaphone->getKeywords($newVal);
218                     $this -> Redis -> sadd('users:'.$keywords[0], $uData['uid']); //add new metaphone
219                 }
220
221                 $this -> Redis -> hset('uid:'.$uData['uid'], $nameFields[$key], $newVal); //store new values
222             }
223         }
224
225         // create age field in user data if date of birth was set by the user
226         if (isset($uData['age']))
227         {
228             $this -> Redis -> hset('uid:'.$uData['uid'], 'age', $uData['age']);
229         }
230         // statment to avoid undefined index error in case that user do not select sex
231         if (Input::has('sex'))
232         {
233             $this -> Redis -> hset('uid:'.$uData['uid'], 'sex', $uData['sex']);
234         }
235
236         // if user have change the password
237         if (isset($uData['password']))
238         {
239             $this -> Redis -> hset('uid:'.$uData['uid'], 'password', md5($uData['password']));
240         }
241         //print_r($this->getUserData($uData['uid']));return;
242         //return new user data
243         return $this->getUserData($uData['uid']);
244     }
245
246     /*
247      * Get User Avatar data and call from libraries/StorageFactory.php the storeAvatarThumb() method
248      * which store avatar thumb into place where factory class determines after config item check
249      */
250     public function storeAvatar($dataU)
251     {
```

```
252         // call static method from libraries/StorageFactory.php and get
253         // the appropriate instance based on config item placed app/config/app.php
254         $storage = StorageFactory::getStorage();
255
256         $storage->storeAvatarThumb($dataU);
257
258         return;
259     }
260 }
261
```

Ελεγκτές

HomeController.php

```php
<?php

class HomeController extends BaseController
{

    public function __construct()
    {
        parent::__construct();
        // check login authentication always on class instantiation
        if (!$this -> isSignedin())
        {
            // use the global before filter to redirect from a constructor
            $this -> beforeFilter(function()
            {
                return View::make('hello');
                // show signin/signup options
            });
        }
    }

    /*
     * If user is signedin ( checked in the constructor),show homepage  (timeline)
     * When the view is loaded, jquery takes action to show the timeline (does pagination if needed)
     */
    public function showHome()
    {
        //count the total rates of the timeline
        $totalRids = $this -> Redis -> llen("timeline_ratings:" . $this -> Udata['uid']);
        if (!$totalRids)//no rates to show
        {
            return View::make('home', array(
                'username' => $this -> Udata['username'],
                'verified' => $this -> Udata['verified'],
                'message' => '<i class="fa fa-info-circle text-info"></i> Your timeline is empty..',
                'uid'      => $this ->Udata['uid']
            ));
        }

        //get the first 5 rids(change this to the number of rates we want to paginate)
        $rids = $this -> Redis -> lrange("timeline_ratings:" . $this -> Udata['uid'], 0, 4);
        $i = 0;
        foreach ($rids as $rid)
        {
            //get all the rid hash info
            $rates[$i] = $this -> Redis -> hgetall('rid:' . $rid);
            //add rid, avg score,rater's username and total # of ratings('count') on that resource
            $rates[$i]['rid'] = $rid;
            $rates[$i]['average_score'] = $this -> Redis -> get('url:' . $this -> Redis -> hget('rid:' . $rid, 'url') . ':average_score');
            $rates[$i]['username'] = $this -> Redis -> hget('uid:' . $rates[$i]['uid'], 'username');
            $rates[$i]['count'] = $this -> Redis -> hlen('url:' . $rates[$i]['url']);
            //check if I have rated it (to show 'rate it' btn  in rate markup)
            if ($this -> Redis -> sismember('uid:' . $this -> Udata['uid'] . ':rated_urls', $rates[$i]['url']))
            {
                $rates[$i]['metoo'] = true;
            }
            $i = $i + 1;
        }
        // //compute the pages needed for pagination
        // $items = 5;  //how many rates to paginate
        // $pages = ceil($totalRids/$items); //needed for jquery on home view
        //show home
        return View::make('home', array(
            'username' => $this -> Udata['username'],
            'verified' => $this -> Udata['verified'],
            'uid' => $this -> Udata['uid'],
            'rates' => $rates,
            'totalRids' => $totalRids
        ));
    }

}

```

BaseController.php

```php
<?php

class BaseController extends Controller
{

    protected $Redis;
    // declaration of redis variable
    protected $Udata;
    // declaration of user data

    public function __construct()
    {
        $this -> Redis = Redis::connection();
    }

    /**
     * Setup the layout used by the controller.
     *
     * @return void
     */

    protected function setupLayout()
    {
        if (!is_null($this -> layout))
        {
            $this -> layout = View::make($this -> layout);
        }
    }

    /*
     * Method to confirm if user is signed in.
     * Check if user is signed in
     */
    protected function isSignedin()
    {
        if (isset($_COOKIE['auth']))
        {
            //laravel send cookie with encrypt format
            $cookie = Crypt::decrypt($_COOKIE['auth']);

            // get user id below data set

            $uid        = $this -> Redis -> hget ("auth", $cookie);

            // get all user data
            $fname      = $this -> Redis -> hget ('uid:'.$uid, 'fname');
            $lname      = $this -> Redis -> hget ('uid:'.$uid, 'lname');
            $email      = $this -> Redis -> hget ('uid:'.$uid, 'email');
            $sex        = $this -> Redis -> hget ('uid:'.$uid, 'sex');
            $age        = $this -> Redis -> hget ('uid:'.$uid, 'age');
            $thumb      = $this -> Redis -> hget ('uid:'.$uid, 'thumb');
            $username   = $this -> Redis -> hget ('uid:'.$uid, 'username');
            $uid_auth   = $this -> Redis -> hget ('uid:'.$uid, 'auth');
            $isActive   = $this -> Redis -> hget ('uid:'.$uid, 'verified');

            //check if auth filed and cookie are not equal
            if ($uid_auth != $cookie)
            {
                return FALSE;
            }

            //if verify code is false means that user doesn't have reply to verification email
            if (!$isActive)
            {
                //get current and signed up datetime
                $currentDT = microtime(true);
                $signedupDT = $this -> Redis -> hget('uid:' . $uid, 'timestamp');
                $result = $currentDT - $signedupDT;

                // check if this period is rather than 24 hours
                // and if is true force him sign out
                if ($result > 86400)
                {
                    $this -> Redis -> hdel('uid:' . $uid, 'auth');
                    $this -> Redis -> hdel('auth', $uid_auth);

                    return FALSE;
                }
            }

            $this->Udata = array(
```

```
82
83                 'uid'        => $uid,
84                 'fname'      => $fname,
85                 'lname'      => $lname,
86                 'email'      => $email,
87                 'age'        => $age,
88                 'sex'        => $sex,
89                 'thumb'      => $thumb,
90                 'username'   => $username,
91                 'verified'   => $isActive
92             );
93
94             // Share with every view a notification counter and a hidden field value
95             $notificationsCount = $this -> Redis -> get('uncheckedNotifications:' . $this -> Udata['uid']);
96             if($notificationsCount==0)$notificationsCount=null;
97             View::share(array(
98                 'myUid' => $uid,
99                 'notificationsCount' => $notificationsCount
100            ));
101
102            return TRUE;
103        }
104        return FALSE;
105    }
106
107 }
108
```

FollowerGraphController.php

```php
<?php

class FollowerGraphController extends BaseController
{

    /*
     *  Controller of follow graph actions
     */

    public function __construct()
    {
        parent::__construct(); //inherit variable of redis connection from parent

        if (!$this -> isSignedin())
        {
            // use the global before filter to redirect from constructor
            $this -> beforeFilter(function()
            {
                //Said laravel to remember where it was (URL) and send to '/'
                return Redirect::guest('/') -> withErros('You are not signed in...');
            });
        }
    }


    public function follow()
    {
        $formdata = Input::all(); // get foreign user id
        $formdata['current_uid'] = $this->Udata['uid']; // get current user id

        $follow   = new FollowerGraph();
        $userData = $follow->followAction($formdata);
        $page     = false;

        //check if you follow the user (to show appropriate follow/unfollow button) and if he's following you (just to show this info in the view)
        $userData['isFollower']  = $this->Redis->zrank('followers:'.$this->Udata['uid'], $userData['uid']);
        $userData['isFollowing'] = $this->Redis->zrank('followings:'.$this->Udata['uid'], $userData['uid']);

        //get (the new) number of your followings to update sidebar's mini profile panel info
        $numOfFollowings = $this->Redis->zcard('followings:'.$this->Udata['uid']);

        if(array_key_exists('page', $formdata))  //form submitted from userProfile view
        {
            $page = $formdata['page']; //will return with the json response
            //render the userProfileMarkup that will replace the current (will chang the 'follow' btn to 'unfollow' and the # of followings info)
            $html = View::make('userProfileMarkup')->with('user',$userData)->with('username', $this->Udata["username"])->render();
        }
        else
        {
            // render cause need file with updated results for updating over AJAX
            $html = View::make('userMarkup')->with('user', $userData)->render();
        }

        $message = 'You are now following '.$userData['username'];

        return Response::json(array(

            'status'    => TRUE,
            'formCase'  => 'followerGraph',
            'view'      => $html,
            'uid'       => $formdata['foreign_uid'],
            'user'      => $userData,
            'numOfFollowings' => $numOfFollowings,
            'page'      => $page,
            'message'   => $message
        ));
    }

    public function unfollow()
    {
        $formdata = Input::all();
        $formdata['current_uid'] = $this->Udata['uid'];

        $unfollow = new FollowerGraph();
        $userData = $unfollow->unFollowAction($formdata);
        $page     = false;

        //check if you follow the user (to show appropriate follow/unfollow button) and if he's following you (just to show this info in the view)
        $userData['isFollower'] = $this->Redis->zrank('followers:'.$this->Udata['uid'], $userData['uid']);
        $userData['isFollowing'] = $this->Redis->zrank('followings:'.$this->Udata['uid'], $userData['uid']);

```

```
82      //get (the new) number of your followings to update sidebar's mini profile panel info
83      $numOfFollowings = $this->Redis->zcard('followings:'.$this->Udata['uid']);
84
85      if(array_key_exists('page', $formdata))  //form submitted from userProfile view
86      {
87          $page = $formdata['page']; //will return with the json response
88          //render the userProfileMarkup that will replace the current (will chang the 'follow' btn to 'unfollow' and the # of followings info)
89          $html = View::make('userProfileMarkup')->with('user',$userData)->with('username', $this->Udata["username"])->render();
90      }
91      else
92      {
93          // render cause need file with updated results for updating over AJAX
94          $html = View::make('userMarkup')->with('user', $userData)->render();
95      }
96      $message = 'You no longer follow '.$userData['username'];
97      return Response::json(array(
98
99          'status'    => TRUE,
100         'formCase' => 'followerGraph',
101         'view'      => $html,
102         'uid'       => $formdata['foreign_uid'],
103         'user'      => $userData,
104         'page'      => $page,
105         'message'   => $message,
106         'numOfFollowings' => $numOfFollowings
107     ));
108 }
109
110 /*
111  *  Show followers
112  */
113 public function followers($uid = null)
114 {
115     //get uid as argument(optional) if no uid parsed means that user who ask for followers
116     ///is the signed in user(current) else..
117     if ($uid == null)
118     {
119         $uid      = $this->Udata['uid'];
120         $username = $this->Udata['username'];
121     }
122     else
123     {
124         $username = $this->Redis->hget('uid:'.$uid, 'username');
125     }
126
127     $start      = 0;
128     $followers  = new FollowerGraph();
129     $results    = $followers->getFollowers($uid, $this->Udata['uid'], $start);
130
131     if (!$results)
132     {
133         return View::make('showFollowersFollowings')->with('results'  , $results)
134                                                     ->with('username' , $this->Udata['username'])
135                                                     ->with('verified' , $this->Udata['verified'])
136                                                     ->with('fUsername', $username)
137                                                     ->with('semistop' , $results['semistop'])
138                                                     ->with('action'   , 'Followers')
139                                                     ->withErrors('Nothing found..');
140     }
141
142     return View::make('showFollowersFollowings')->with('results'  , $results['results'])
143                                                 ->with('username' , $this->Udata['username'])
144                                                 ->with('verified' , $this->Udata['verified'])
145                                                 ->with('fUsername', $username)
146                                                 ->with('semistop' , $results['semistop']) // point where the pagination stops
147                                                 ->with('uid'      , $results['uid'])
148                                                 ->with('action'   , 'Followers');
149 }
150
151 /*
152  *  Show followings
153  */
154 public function followings($uid = null)
155 {
156     if ($uid == null)
157     {
158         $uid      = $this->Udata['uid'];
159         $username = $this->Udata['username'];
160     }
161     else
162     {
163         $username = $this->Redis->hget('uid:'.$uid, 'username');
164     }
165
166     $start      = 0;
```

```
167         $followings = new FollowerGraph();
168         $results    = $followings->getFollowings($uid, $this->Udata['uid'], $start);
169
170         if (!$results)
171         {
172             return View::make('showFollowersFollowings')->with('results'  , $results)
173                                                         ->with('username' , $this->Udata['username'])
174                                                         ->with('verified' , $this->Udata['verified'])
175                                                         ->with('fUsername', $username)
176                                                         ->with('semistop' , $results['semistop'])
177                                                         ->with('action'   , 'Followings')
178                                                         ->withErrors('Nothing found..');
179         }
180
181         return View::make('showFollowersFollowings')->with('results'  , $results['results'])
182                                                     ->with('username' , $this->Udata['username'])
183                                                     ->with('verified' , $this->Udata['verified'])
184                                                     ->with('fUsername', $username)
185                                                     ->with('semistop' , $results['semistop'])
186                                                     ->with('uid'      , $results['uid'])
187                                                     ->with('action'   , 'Followings');
188     }
189
190     /*
191      * Pagination on followers followings results
192      */
193     public function pagination()
194     {
195         $start      = Input::get('semistop');// where stop the previus count
196         $uid        = Input::get('uid');     // id of user whose followers - followings be presented
197         $from       = Input::get('from');    // which action come from followers or followings
198         $followers  = new FollowerGraph();
199
200         if ($from == 'Followers')
201         {
202             $results = $followers->getFollowers($uid, $this->Udata['uid'], ++$start);
203         }
204         else
205         {
206             $results = $followers->getFollowings($uid, $this->Udata['uid'], ++$start);
207         }
208
209         if (Request::ajax())
210         {
211             $html = View::make('userBase')->with('results', $results['results'])
212                                           ->render();
213
214             $more = View::make('moreUsers')->with('semistop', $results['semistop'])
215                                            ->with('uid', $results['uid'])
216                                            ->render();
217
218             return Response::json(array(
219
220                 'status' => TRUE,
221                 'formCase' => 'moreUsers',
222                 'view'   => $html,
223                 'more'   => $more
224             ));
225         }
226     }
227 }
228
229
230
```

ProfileController.php

```php
<?php

/*
 * User profile
 */

class ProfileController extends BaseController
{

    public function __construct()
    {
        parent::__construct();
        //current constructor inherit parent from BaseController

        if (!$this -> isSignedin())
        {
            // use the global before filter to redirect from a constructor
            $this -> beforeFilter(function()
            {
                //Said laravel to remember where it was (URL) and send to '/'
                return Redirect::guest('/') -> withErrors('Sign in is required.');
            });
        }
    }

    /*
     * Show profile of signed in user
     */
    public function mine()
    {
        // Instanciate and get data from user with id the given argument
        $User        = new User();
        $this->Udata = $User->getUserData($this->Udata['uid']);

        $this->Udata['sameuser'] = TRUE;
        //if a dob was set (field 'age' in user hash, timestamp format), calculate his age
        if(!empty($this->Udata['age']))
        {
            $this->Udata['age'] = $this->calculateAge($this->Udata['age']);
        }


        return View::make('userPage')->with('user', $this->Udata)
                                     ->with('username', $this->Udata['username'])
                                     ->with('verified', $this->Udata['verified']);
    }

    /*
     * Show profile of foreign user
     */
    public function preview($uid)
    {
        if ($uid == $this->Udata['uid'])
        {
            return Redirect::to('profile');
        }

        // Instanciate and get data from user with id the given argument
        $User     = new User();
        $uid_data = $User->getUserData($uid);
        $uid_data['sameuser'] = FALSE;
        //if a dob was set (field 'age' in user hash, timestamp format), calculate his age
        if(!empty($uid_data['age']))
        {
            $uid_data['age'] = $this->calculateAge($uid_data['age']);
        }
        //check if you follow the user (to show appropriate follow/unfollow button) and if he's following you (just to show this info in the view)
        $uid_data['isFollower'] = $this->Redis->zrank('followers:'.$this->Udata['uid'], $uid);
        $uid_data['isFollowing'] = $this->Redis->zrank('followings:'.$this->Udata['uid'], $uid);

        return View::make('userPage')->with('user', $uid_data)
                                     ->with('username', $this->Udata['username'])
                                     ->with('verified', $uid_data['verified']);
    }

    /*
     * Show edit profile view of signed in user
     */
    public function edit()
    {
        //if a dob was set (field 'age' in user hash, timestamp format), convert to d-m-y date
```

```
82          if(!empty($this->Udata['age']))
83          {
84              $this->Udata['age'] = date('d-m-Y',$this->Udata['age']);
85          }
86
87          return View::make('editProfile')->with('user', $this->Udata)
88                                          ->with('username', $this->Udata['username'])
89                                          ->with('verified', $this->Udata['verified']);
90      }
91
92      /*
93       * get post variables from edit profile view and proceed to changes
94       */
95      public function handleEdit()
96      {
97          $formData = Input::all();
98
99          $rules = array( 'username' => 'min:2' );
100
101         if (Input::has('fname'))// if first name has value add rule to validator
102         {
103             $rules['fname'] = 'min:2';
104         }
105
106         if (Input::has('lname'))
107         {
108             $rules['lname'] = 'min:2';
109         }
110
111         if(Input::has('dob'))
112         {
113             $rules['dob'] = 'date_format:"d-m-Y"';
114         }
115         $messages = array( 'min' => 'At least two characters' );// custom validator message for min rule
116
117         $validator = Validator::make($formData, $rules, $messages);
118
119         if ($validator -> fails())
120         {
121             // Validator fails, form fields dont comply with the rules
122             return Redirect::to('edit') -> withErrors($validator);
123         }
124
125         // check exist changes in security fields and parse values to validator
126         if (Input::has('oldPassword') || Input::has('newPassword') || Input::has('confirmPassword'))
127         {
128             // get user email because need it for exist_password validation rule
129             $email = $this->Redis->hget('uid:'.$this->Udata['uid'], 'email');
130
131             // Build the validation constraint set.
132             $rules = array(
133
134                 'oldPassword'     => 'required|exist_password:'.$email.','.$formData['oldPassword'],
135                 'newPassword'     => 'required|min:4',
136                 'confirmPassword' => 'required|password_confirmed:'.$formData['newPassword'],
137
138             );
139
140             //custom validator message for exist_password rule
141             $messages = array(
142                     'exist_password' => 'Wrong Password',
143                     'dob'            => 'invalid date format'
144                     );
145
146             // Create a new validator instance
147             $validator = Validator::make($formData, $rules, $messages);
148
149             if ($validator -> fails())
150             {
151                 // Validator fails, form fields dont comply with the rules
152                 return Redirect::to('edit') -> withErrors($validator);
153             }
154
155             $formData['password'] = $formData['newPassword'];
156         }
157
158         $formData['uid'] = $this->Udata['uid'];
159         if(Input::has('dob'))
160         {
161             $formData['age'] = strtotime($formData['dob']);
162         }
163         //print_r($formData);return;
164         //parse data to user model and return the changes
165         $User     = new User();
166         $userData = $User->editProfile($formData);
```

```
167
168
169         return  Redirect::to('edit')->with('user'    , $userData)
170                                     ->with('username', $userData['username'])
171                                     ->with('verified', $userData['verified'])
172                                     ->with('message' , 'Changes Saved');
173     }
174
175     /*
176      * Get avatar thumb from editProfile.blade.php
177      */
178     public function getAvatar()
179     {
180         $input = array('avatar' => Input::file('avatar')); // give name to input filed
181
182         $rules = array('avatar' => 'required|image|max:1000');
183
184         $messages = array('avatar.required' => 'Select an image first');
185
186         $validator = Validator::make($input, $rules, $messages);
187
188         if ($validator -> fails())
189         {
190             // Validator fails, form fields dont comply with the rules
191             return Redirect::to('edit')->withErrors($validator);
192         }
193
194         $avatar   = Input::file('avatar'); // instance from uploaded file
195
196         $s3Data['filepath'] = $avatar->getRealPath();               // input type files temporary saved into server. use this path to skip move method
197         $s3Data['filetype'] = $avatar->getMimeType();               // get type of file
198         $fileExtension      = $avatar->getClientOriginalExtension();    // get extension
199         $s3Data['filename'] = 'avatar'.$this->Udata['uid'].'.'.$fileExtension; // create a new file name with prefix avatar plus the uid
200         $s3Data['bucket']   = 'avatarthumb';
201         $s3Data['uid']      = $this->Udata['uid'];
202
203         $User   = new User();
204         $User->storeAvatar($s3Data);
205
206         return Redirect::to('edit');
207     }
208
209
210     /**
211      * Calculate age in years based on timestamp and reference timestamp
212      * If the reference $now is set to 0, then current time is used
213      *
214      * @param int $timestamp
215      * @param int $now
216      * @return int
217      */
218     public function calculateAge($timestamp = 0, $now = 0) {
219     # default to current time when $now not given
220     if ($now == 0)
221     $now = time();
222
223     # calculate differences between timestamp and current Y/m/d
224     $yearDiff = date("Y", $now) - date("Y", $timestamp);
225     $monthDiff = date("m", $now) - date("m", $timestamp);
226     $dayDiff = date("d", $now) - date("d", $timestamp);
227
228     # check if we already had our birthday
229     if ($monthDiff < 0)
230     $yearDiff--;
231     elseif (($monthDiff == 0) && ($dayDiff < 0))
232     $yearDiff--;
233
234     # set the result: age in years
235     $result = intval($yearDiff);
236
237     # deliver the result
238     return $result;
239     }
240 }
241
```

RateController.php

```php
<?php

class RateController extends BaseController
{

    protected $uid;

    public function __construct()
    {
        parent::__construct();
        // check login authentication always on class instantiation
        if (!$this -> isSignedin())
        {
            // use the global before filter to redirect from a constructor
            $this -> beforeFilter(function()
            {
                //Said laravel to remember where it was (URL) and send to '/'
                return Redirect::guest('/') -> with('message', 'You are not signed in...');
            });
        }
    }

    ////////////////////////////////////////////////////////////////////////////////////
    /*
     * Function newRate()
     * Called on 'New rate' menu button pressed
     * Shows new rate form (app/views/newRate.blade.php)
     *
     */
    public function newRate()
    {

        return View::make('newRate', array(
            'uid' => $this -> Udata['uid'],
            'username' => $this -> Udata['username'],
            'verified' => $this -> Udata['verified']
        ));
    }

    ////////////////////////////////////////////////////////////////////////////////////
    /*
     * Function handleNewRate()
     * Called on new rate form submit (From quick(sidebar) or normal (newrate view))
     * Gets form input data
     * Runs checks and validation rules
     * Calculates data needed (resource title, e.t.c.)
     * Calls appropriate method from rate model (app/models/Rate.php) to insert/update rate into db (newRate() or edit())
     */

    public function handleNewRate()
    {

        //get data from form
        $rdata = Input::all();
        // if rate is quick rate (no review input), create an empty review index and set validation rules
        if (!array_key_exists('review', $rdata))
        {
            $form = 'quickNewRate';
            //returned with json response
            $rdata['review'] = "";
            $rules = array('urlQuick' => 'required|activeurl');
            // //trim any forward slash, spaces, tabs and linebreaks before inserting to db
            // // (e.g. we want www.foo.com & www.foo.com/ to be treated as same resource)
            // $rdata['urlQuick']   = rtrim($rdata['urlQuick'], "/ \t\n\r");
            $uri = $rdata['uri'];
            // The URI will be included into json response on form success.
            // The function formSuccessQuickRate(data) will use it to decide if the page should be refreshed (case page = home or myrates) or not
        }
        else// new rate from newRate view (not from sidebar's quick rate form)
        {
            $form = 'normalNewRate';
            $rules = array('urlNormal' => 'required|activeurl');
            // //trim any forward slash, spaces, tabs and linebreaks
            // $rdata['urlNormal'] = rtrim($rdata['urlNormal'], "/ \t\n\r");
            $uri = null;
            // we don't need the URI in this case (new rate from newRate view)
        }

        //set custom validation error messages:
        //if laravel's default messages are used, the form's input names are shown (e.g. 'the urlQuick field is required') - not elegant
        $messages = array(
```

```
82              'required' => 'The url field is required',
83              'activeurl' => 'The url is not a valid URL'
84          );
85
86          // client-sided (javascript) validation of the form succeeded, continue with server-sided validation
87
88          // Create a new validator instance (pass the custom messages too)
89          $validator = Validator::make($rdata, $rules, $messages);
90
91          if ($validator -> fails())
92          {
93              // Validator fails, return JSON encoded response
94              return Response::json(array(
95                  'status' => FALSE,
96                  'errors' => array($validator -> messages() -> all())
97              ));
98          }
99
100         // server-sided validation ok, continue:
101         // 'normalize' $rdata['urlNormal'] / $rdata['urlquick'] to $rdata['url'] before passing to model
102         if (array_key_exists('urlNormal', $rdata))// 'normal' form submitted
103         {
104             $rdata['url'] = $rdata['urlNormal'];
105             unset($rdata['urlNormal']);
106         }
107         else//quick form submitted
108         {
109             $rdata['url'] = $rdata['urlQuick'];
110             unset($rdata['urlQuick']);
111         }
112         //
113         //trim any forward slash, spaces, tabs and linebreaks before inserting to db
114         $rdata['url'] = rtrim($rdata['url'], "/ \t\n\r");
115         //remove 'www.' if exists
116         $rdata['url'] = str_replace("http://www.", "http://", $rdata['url']);
117         //set the id of the user-rater
118         $rdata['uid'] = $this -> Udata['uid'];
119         //check if url exists, and if so, check if it is edit case (only possible from quick rate)
120         //initialize to true and change if needed
121         $rdata['newUrl'] = true;
122         //initialize to newRate and change if needed
123         $case = 'newRate';
124         // check if case is edit:
125         // a)check if url exists in db
126         $UrlManagement = new UrlManagement;
127         if ($UrlManagement -> urlExists($rdata['url']))
128         {
129             $rdata['newUrl'] = false;
130             // b)check if edit case
131
132             if ($this -> Redis -> sismember('uid:' . $rdata['uid'] . ':rated_urls', $rdata['url']))
133             {
134                 $case = 'edit';
135
136                 if ($form == 'quickNewRate')
137                 {
138                     $rdata['public'] = 'old';
139                     //for the model to know that case is edit from quick rate form=>keep the old public/private setting
140                 }
141             }
142         }
143         //take care of the rest of the rating data (only if case = newRate, not needed for edit )
144         if ($case != 'edit')
145         {
146             // set the rest of the rate fields
147             $rdata['title'] = $this -> getTitle($rdata['url']);
148             $rdata['thumb'] = $this -> getThumb($rdata['url']);
149         }
150
151         if (empty($rdata['public']))//case quick rate or 'off' in new rate form
152         {
153             $rdata['public'] = 0;
154             //default value
155         }
156
157         // get the search keywords associated with the rating title (only if rate = new rate -new rid is created)
158         if ($case == "newRate")
159         {
160             $mp = new DoubleMetaPhone;
161             $keywords = $mp -> getKeywords($rdata['title']);
162             $rdata['keywords'] = $keywords;
163         }
164         // bootstrap switch plugin sends public as 'on' if checked, convert it to '1' (for the model to store it in the redis hash as 1)
165         if ($rdata['public'] === "on")
166         {
```

```
167             $rdata['public'] = 1;
168         }
169 
170         // ready to pass new rate data to model
171         // new rate model instance  (app/models/Rate.php)
172         $Rate = new Rate;
173         //call appropriate model function
174         $rid = false;
175         switch ($case)
176         {
177             case 'newRate' :
178                 $Rate -> newRate($rdata);
179                 //just a message for the json response (or the redirect if js is disabled)
180                 $message = 'New rate inserted';
181                 break;
182             case 'edit' :
183                 $Rate -> edit($rdata);
184                 //get the rid - needed for the response (so as to know which rate to update if in timeline or my rates view)
185                 $rid = $this -> Redis -> hget('url:' . $rdata['url'], $this -> Udata['uid']);
186                 //just a message for the json response
187                 $message = 'Rate edited';
188                 break;
189         }
190 
191         // rate inserted succesfully, return JSON encoded response
192 
193         return Response::json(array(
194             'status' => TRUE,
195             'formCase' => $form,
196             'message' => $message,
197             'uri' => $uri,
198             'rid' => $rid
199         ));
200 
201     }
202 
203     ////////////////////////////////////////////////////////////////////////////
204     public function handleRateIt()
205     {
206         $data = Input::all();
207         $form = 'rateIt';
208         //returned with json response
209         $data['uid'] = $this -> Udata['uid'];
210         $data['newUrl'] = false;
211 
212         if (array_key_exists('public', $data))// public is 'on' - if public switch was off, the form does not post a 'public' parameter
213         {
214             $data['public'] = 1;
215         }
216         else
217         {
218             $data['public'] = 0;
219         }
220         $data['thumb'] = $this -> getThumb($data['url']);
221         //get the metaphones , for the model to add the rid into the keyword(s) set(s)
222         $mp = new DoubleMetaPhone;
223         $keywords = $mp -> getKeywords($data['title']);
224         $data['keywords'] = $keywords;
225 
226         $message = 'Rate inserted';
227         //for the json response
228 
229         // ready to pass new rate data to model
230         // new rate model instance  (app/models/Rate.php)
231         $Rate = new Rate;
232         $Rate -> newRate($data);
233 
234         //get the rid of the rate just created
235         $rid = $this -> Redis -> hget('url:' . $data['url'], $this -> Udata['uid']);
236         //get the new avg score
237         $newAvgScore = $this -> Redis -> get('url:' . $data['url'] . ':average_score');
238         //get the new num of raters on that resource
239         $newCount = $this -> Redis -> hlen('url:' . $data['url']);
240         //if user is on home, return the new rate markup and prepend it to the timeline/myrates
241         $newRate = false;
242         $rids = false;
243         if ($data['target'] === 'home')
244         {
245             //create the rate markup to be returned to the view
246             //get the data needed for the markup
247             $rate = $this -> Redis -> hgetall('rid:' . $rid);
248             $rate['rid'] = $rid;
249             $rate['username'] = $this -> Udata['username'];
250             $rate['average_score'] = $newAvgScore;
251             $rate['count'] = $newCount;
```

```
252             $username = $this -> Udata['username'];
253
254             $newRate = View::make('rateMarkup', array(
255                 'rate' => $rate,
256                 'username' => $username
257             )) -> render();
258
259         }
260         // Get original rid
261         $originalRid = str_replace('rid', '', $data['originalRid']);
262         //Get original uid
263         $originalUid = $this -> Redis -> hget('rid:' . $originalRid, 'uid');
264         // Increace notifications counter
265         $this -> Redis -> incr('uncheckedNotifications:' . $originalUid);
266         // Add notification to user's sorted set
267         $this -> Redis -> zadd('notifications:' . $originalUid, microtime(true), $this -> Udata['uid'] . ':rated:' . $originalRid);
268
269         $isOnline = $this -> Redis -> hget('uid:' . $originalUid, 'auth');
270         // If "original user" is online notify him
271         if (!empty($isOnline))
272         {
273             $notification = array('uid' => $originalUid, );
274             Event::fire('cometNotification', array($notification));
275         }
276         return Response::json(array(
277             'status' => TRUE,
278             'formCase' => $form,
279             'message' => $message,
280             'originalRid' => $data['originalRid'], //to know which rate's 'rate it' button was pressed
281             'newrate' => $newRate,
282             'newAvgScore' => $newAvgScore,
283             'newNumOfRaters' => $newCount
284         ));
285     }
286
287     ////////////////////////////////////////////////////////////////////////////
288     /*
289      * Function getTitle($url)
290      * Gets the title of the rating on the given resource
291      */
292     private function getTitle($url)
293     {
294         //if url is already a rated url , just get title from an existing rate hash
295         if (($this -> Redis -> hlen('url:' . $url)) > 0)
296         {
297             // get the title hash field of the first rid hash found (for that specific url)
298             $rids = $this -> Redis -> hvals('url:' . $url);
299             $title = $this -> Redis -> hget('rid:' . $rids[0], 'title');
300             return $title;
301         }
302         //if rate is new rate on non-existing url, use the appropriate lib function to get the title
303
304         //new instance of 'UrlManagement' library class ( /libraries/UrlManagement.php)
305         $UrlManagement = new UrlManagement;
306         $title = $UrlManagement -> urlGetTitle($url);
307         return $title;
308     }
309
310     ////////////////////////////////////////////////////////////////////////////
311
312     /*
313      * Function getThumb($url)
314      * Creates and stores a thumb of the rated resource.
315      * Returns the name (rid) of the stored thumb
316      * Functionality identical to getTitle.
317      */
318     private function getThumb($url)
319     {
320         if (($this -> Redis -> hlen('url:' . $url)) > 0)
321         {
322             $rids = $this -> Redis -> hvals('url:' . $url);
323             $thumb = $this -> Redis -> hget('rid:' . $rids[0], 'thumb');
324             return $thumb;
325         }
326
327         $UrlManagement = new UrlManagement;
328         $thumb = $UrlManagement -> urlGetThumb($url);
329         return $thumb;
330     }
331
332     ////////////////////////////////////////////////////////////////////////////
333     /*
334      * Function searchRate()
335      * Shows the search Rate form view  (/app/views/searchRate.blade.php - 'search rate' menu item)
336      */
```

```
337     public function searchRate()
338     {
339         return View::make('searchRate', array(
340             'uid' => $this -> Udata['uid'],
341             'username' => $this -> Udata['username'],
342             'verified' => $this -> Udata['verified']
343         ));
344     }
345 
346     ////////////////////////////////////////////////////////////////////////
347 
348     /*
349      * Function handleSearchRate()
350      * Called on search rate form submission
351      * Gets form input data
352      * Prepares the input search data to be sent to search() function in Rate model
353      */
354 
355     public function handleSearchRate()
356     {
357         $formdata = Input::all();
358 
359         $rules = array(
360             'searchIn' => 'required', // form's checkboxes own/followings/both  (where to search)
361             'searchQuery' => 'required'
362         );
363         //set a custom error message
364         $messages = array(
365             'searchIn.required' => 'Tell me where to search',
366             'searchQuery.required' => 'Type something'
367         );
368 
369         $validator = Validator::make($formdata, $rules, $messages);
370 
371         if ($validator -> fails())
372         {
373             $messages = $validator -> messages();
374             // redirect to search form view (/app/views/searchRate.blade.php) with errors messagebag
375             Input::flashonly('searchQuery', 'searchIn');
376             // keep the old inputs
377             return Redirect::to('searchrate') -> withErrors($validator);
378         }
379 
380         // Validation succeeded, prepare data to be sent to model
381 
382         // $criteria array will be passed to model (holding search criteria, user id and 'searchIn' option)
383         // Set 'search in' option
384         $searchIn = '"both"';
385         // assume both checkboxes are checked
386         if (sizeof($formdata['searchIn']) == 1)// only one checkbox is checked
387         {
388             // get checked checkbox's name ('own' or 'followings')
389             $searchIn = key($formdata['searchIn']);
390         }
391 
392         //split the search query into keywords (based on which the search will take place)
393         $mp = new DoubleMetaPhone;
394         $keywords = $mp -> getKeywords($formdata['searchQuery']);
395         if (!count($keywords))
396         {
397             return Redirect::to('searchrate') -> with('message', 'Nothing found!') -> withInput(Input::only('searchQuery', 'searchIn'));
398         }
399         $criteria = array(
400             'uid' => $this -> Udata['uid'],
401             'searchIn' => $searchIn,
402             'keywords' => $keywords
403         );
404 
405         //if user clicked advanced search, add rating score range  in $criteria array to be passed to model
406         if (array_key_exists('advanced', $formdata))
407         {
408             // sort rating range (if for example user entered score from 10 to 8 (and not from 8 to 10))
409             $scores[0] = $formdata['fromScore'];
410             $scores[1] = $formdata['toScore'];
411             sort($scores);
412             //add them to criteria array
413             $criteria['scores'] = $scores;
414         }
415 
416         //new rate model instance
417         $Rate = new Rate;
418         $searchResults = $Rate -> search($criteria);
419         //print_r($searchResults);return;
420         if (!$searchResults)
421         {
```

```
422             return Redirect::to('searchrate') -> with('message', 'Nothing found!') -> withInput(Input::only('searchQuery', 'searchIn'));
423
424         }
425         return View::make('searchRateResults', array(
426             'username' => $this -> Udata['username'],
427             'verified' => $this -> Udata['verified'],
428             'searchResults' => $searchResults,
429             'searchQuery' => $formdata['searchQuery']
430         ));
431
432     }
433
434     ////////////////////////////////////////////////////////////////////////////////
435     /*
436      * Function handleDeleteRate()
437      * Called on delete rate form submission
438      * Gets form input data
439      * Sends rid and uid to delete() function in Rate model
440      * delete() functions returns true or false.
441      */
442     public function handleDeleteRate()
443     {
444         $formdata = Input::only('rid');
445         $Rate = new Rate;
446         // Status is either FALSE or contains rate array
447         $status = $Rate -> delete($formdata['rid'], $this -> Udata['uid']);
448         if ($status)
449         {
450             $message = 'Rate deleted';
451             $errors = FALSE;
452         }
453         else
454         {
455             $message = FALSE;
456             $errors = ' Something went wrong, mate...';
457         }
458         //get the number of public rates, to update if needed the info displayed on page header(e.g myRates: #total - #PUBLIC rates)
459         $numOfPublic = $this -> Redis -> scard('uid:' . $this -> Udata['uid'] . ':publicRids');
460         if (Request::ajax())
461         {
462             return Response::json(array(
463                 'status' => $status,
464                 'formCase' => 'deleteRate',
465                 'message' => $message,
466                 'errors' => $errors,
467                 'rid' => $formdata['rid'],
468                 'numOfPublic' => $numOfPublic
469             ));
470         }
471
472         return Redirect::back();
473     }
474
475     ////////////////////////////////////////////////////////////////////////////////
476
477     /*
478      * Handles the edit rate form (shown on pressing 'edit' button on a rate - NOT edit case form quick or new rate form)
479      */
480
481     public function handleEditRate()
482     {
483
484         //Get form post data:
485         $formdata = Input::all();
486
487         if (!array_key_exists('public', $formdata))//public checkbox not checked, set to 0
488         {
489             $formdata['public'] = 0;
490         }
491         else
492         {
493             $formdata['public'] = 1;
494         }
495         $formdata['uid'] = $this -> Udata['uid'];
496         //needed by the edit() method in model
497         //new model instance
498         $Rate = new Rate;
499         $status = $Rate -> edit($formdata);
500         // assume things went wrong, overriden if not
501         $message = FALSE;
502         $errors = ' Something went wrong';
503         //if edited succesfully,set $message, $errors accordingly and get the all the rating info
504         //(needed for the response, as the whole rate markup will be returned to replace the old one)
505         if ($status)
506         {
```

```
507             $rate = $this -> Redis -> hgetall('rid:' . $formdata['rid']);
508             //add the rid, username, #of raters on the resource (count) , avg score (the markup expects them)
509             $rate['rid'] = $formdata['rid'];
510             $rate['username'] = $this -> Udata['username'];
511             $rate['average_score'] = $this -> Redis -> get('url:' . $this -> Redis -> hget('rid:' . $rate['rid'], 'url') . ':average_score');
512             $rate['count'] = $this -> Redis -> hlen('url:' . $rate['url']);
513             $username = $this -> Udata['username'];
514             $errors = FALSE;
515             $message = "Rate edited";
516         }
517 
518         //get the number of public rates, to update if needed the info displayed on page header( #total - #PUBLIC rates)
519         $numOfPublic = $this -> Redis -> scard('uid:' . $this -> Udata['uid'] . ':publicRids');
520 
521         //Prepare the edited rate markup. Will be included in the response and will replace the old rate markup
522         $editedRate = View::make('rateMarkup', array(
523             'rate' => $rate,
524             'username' => $username
525         )) -> render();
526 
527         return Response::json(array(
528             'status' => $status,
529             'formCase' => 'editRate',
530             'message' => $message,
531             'errors' => $errors,
532             'targetRateDiv' => $formdata['target'], //to know which rate will be replaced
533             'editedRate' => $editedRate,
534             'numOfPublic' => $numOfPublic,
535         ));
536     }
537 
538     ////////////////////////////////////////////////////////////////////////////////////////////////
539 
540     public function myRates()
541     {
542         //count total and public own rates
543         $totalRids = $this -> Redis -> llen("own_ratings:" . $this -> Udata['uid']);
544         $numOfPublicRates = $this -> Redis -> scard('uid:' . $this -> Udata['uid'] . ':publicRids');
545         if (!$totalRids)//no rates to show
546         {
547             return View::make('myRates', array(
548                 'username' => $this -> Udata['username'],
549                 'verified' => $this -> Udata['verified'],
550                 'message' => 'You have no rates, mate...',
551                 'public' => $numOfPublicRates,
552                 'totalRids' => $totalRids
553             ));
554         }
555         $rids = $this -> Redis -> lrange("own_ratings:" . $this -> Udata['uid'], 0, 4);
556         if (!$rids)
557         {
558             return false;
559         }
560         $i = 0;
561         foreach ($rids as $rid)
562         {
563             //get all the rid hash info
564             $rates[$i] = $this -> Redis -> hgetall('rid:' . $rid);
565             //add rid, avg score,rater's username and total # of ratings('count') on that resource
566             $rates[$i]['rid'] = $rid;
567             $rates[$i]['average_score'] = $this -> Redis -> get('url:' . $this -> Redis -> hget('rid:' . $rid, 'url') . ':average_score');
568             $rates[$i]['username'] = $this -> Redis -> hget('uid:' . $rates[$i]['uid'], 'username');
569             $rates[$i]['count'] = $this -> Redis -> hlen('url:' . $rates[$i]['url']);
570             //check if I have rated it (to show 'rate it' btn or 'You have rated it' alert in rate markup)
571             if ($this -> Redis -> sismember('uid:' . $this -> Udata['uid'] . ':rated_urls', $rates[$i]['url']))
572             {
573                 $rates[$i]['metoo'] = true;
574             }
575             $i = $i + 1;
576         }
577 
578         return View::make('myRates', array(
579             'username' => $this -> Udata['username'],
580             'verified' => $this -> Udata['verified'],
581             'rates' => $rates,
582             'public' => $numOfPublicRates,
583             //'pages'   => $pages,
584             'totalRids' => $totalRids
585         ));
586     }
587 
588     /*
589      * Function that returnes the 10 top rated urls.
590      * Only urls that meet criteria (minimum 5 votes ) are included.
591      */
```

```
592     public function topRated()
593     {
594         $rates = null;
595         // Array with url as key and average score as value
596         $urlsAndAverageScores = $this -> Redis -> zrevrange('scoreBasedChart', 0, 9, array('withscores' => TRUE));
597         for ($i = 0; $i < count($urlsAndAverageScores); $i++)
598         {
599             // Gets all rids associated with this url
600             $rids = $this -> Redis -> hvals('url:' . $urlsAndAverageScores[$i][0]);
601             // Get title and thumbnail for this url from the first rid (random selection)
602             $rates[$i]['title'] = $this -> Redis -> hget('rid:' . $rids[0], 'title');
603             $rates[$i]['thumb'] = $this -> Redis -> hget('rid:' . $rids[0], 'thumb');
604             // Set average score and url
605             $rates[$i]['url'] = $urlsAndAverageScores[$i][0];
606             $rates[$i]['average_score'] = $this -> Redis -> get('url:' . $rates[$i]['url'] . ':average_score');
607             // Number of voters
608             $rates[$i]['count'] = count($rids);
609         }
610         return View::make('topRated', array(
611             'username' => $this -> Udata['username'],
612             'verified' => $this -> Udata['verified'],
613             'rates' => $rates
614         ));
615     }
616 
617     /*
618      * Function that returnes the 10 most rated urls.
619      */
620     public function mostRated()
621     {
622         $rates = null;
623         // Array with url as key and average score as value
624         $urlsAndRaters = $this -> Redis -> zrevrange('popularityBasedChart', 0, 9, array('withscores' => TRUE));
625         for ($i = 0; $i < count($urlsAndRaters); $i++)
626         {
627             // Gets all rids associated with this url
628             $rids = $this -> Redis -> hvals('url:' . $urlsAndRaters[$i][0]);
629             // Get title and thumbnail for this url from the first rid (random selection)
630             $rates[$i]['title'] = $this -> Redis -> hget('rid:' . $rids[0], 'title');
631             $rates[$i]['thumb'] = $this -> Redis -> hget('rid:' . $rids[0], 'thumb');
632             // Set average score and url
633             $rates[$i]['url'] = $urlsAndRaters[$i][0];
634             $rates[$i]['average_score'] = $this -> Redis -> get('url:' . $rates[$i]['url'] . ':average_score');
635             // Number of voters
636             $rates[$i]['count'] = count($rids);
637         }
638         return View::make('mostRated', array(
639             'username' => $this -> Udata['username'],
640             'verified' => $this -> Udata['verified'],
641             'rates' => $rates
642         ));
643     }
644 
645     /*
646      * Function that returnes the 10 newest rates.
647      * If rids in DB are less than 10, all of them are going to be returned.
648      */
649     public function newestRates()
650     {
651         // Response returned when newest rates are accessed from sidebar
652         if (Request::ajax())
653         {
654 
655             $rates = null;
656             $rid = $this -> Redis -> get('nextRateId');
657             $i = 0;
658             while ($rid > 0 && $i < 5)
659             {
660                 $check = $this -> Redis -> hget('rid:' . $rid, 'url');
661                 if (!empty($check))
662                 {
663                     $rates[$i]['rid'] = $rid;
664 
665                     $i++;
666                     $rid--;
667                 }
668                 else
669                 {
670                     $rid--;
671                 }
672             }
673             $rates = array_reverse($rates);
674             return $rates;
675 
676         }
```

```
677
678         $rates = null;
679         $rid = $this -> Redis -> get('nextRateId');
680         $i = 0;
681         while ($rid > 0 && $i < 10)
682         {
683             $rates[$i] = $this -> Redis -> hgetall('rid:' . $rid);
684             if (!empty($rates[$i]))
685             {
686                 $rates[$i]['rid'] = $rid;
687                 $rates[$i]['average_score'] = $this -> Redis -> get('url:' . $this -> Redis -> hget('rid:' . $rid, 'url') . ':average_score');
688                 $rates[$i]['count'] = $this -> Redis -> hlen('url:' . $rates[$i]['url']);
689                 $rates[$i]['username'] = $this -> Redis -> hget('uid:' . $rates[$i]['uid'], 'username');
690                 $i++;
691                 $rid--;
692             }
693             else
694             {
695                 $rid--;
696             }
697         }
698         //print_r($rates[1]);return;
699         return View::make('newestRates', array(
700             'username' => $this -> Udata['username'],
701             'verified' => $this -> Udata['verified'],
702             'rates' => $rates
703         ));
704     }
705
706     /*
707      * Function that returnes the 50 worst rated urls.
708      * Only urls that meet criteria (minimum 5 votes ) are included.
709      */
710     public function worstRated()
711     {
712         $rates = null;
713         // Array with url as key and average score as value
714         $urlsAndAverageScores = $this -> Redis -> zrange('scoreBasedChart', 0, 49, array('withscores' => TRUE));
715         for ($i = 0; $i < count($urlsAndAverageScores); $i++)
716         {
717             // Gets all rids associated with this url
718             $rids = $this -> Redis -> hvals('url:' . $urlsAndAverageScores[$i][0]);
719             // Get title and thumbnail for this url from the first rid (random selection)
720             $rates[$i]['title'] = $this -> Redis -> hget('rid:' . $rids[0], 'title');
721             $rates[$i]['thumb'] = $this -> Redis -> hget('rid:' . $rids[0], 'thumb');
722             // Set average score and url
723             $rates[$i]['url'] = $urlsAndAverageScores[$i][0];
724             $rates[$i]['average_score'] = $this -> Redis -> get('url:' . $rates[$i]['url'] . ':average_score');
725             // Number of voters
726             $rates[$i]['count'] = count($rids);
727         }
728         return View::make('worstRated', array(
729             'username' => $this -> Udata['username'],
730             'verified' => $this -> Udata['verified'],
731             'rates' => $rates
732         ));
733     }
734
735     ////////////////////////////////////////////////////////////////////////////////////////////////////
736
737     public function showUserRates($uid)
738     {
739         //check that the user for which the rates are requested exists
740         if (!$this -> Redis -> exists('uid:' . $uid))
741         {
742             return View::make('userRates', array(
743                 'username' => $this -> Udata['username'],
744                 'verified' => $this -> Udata['verified'],
745                 'message' => 'Oops..You have requested the ratings of a user that does not exist'
746             ));
747         }
748
749         //if the requested user is the same user, show 'my Rates'
750         if ($uid == $this -> Udata['uid'])
751
752         {
753             return Redirect::action('RateController@myRates');
754
755         }
756
757         //count total and public rates
758         $totalRids = $this -> Redis -> llen("own_ratings:" . $uid);
759         $numOfPublicRates = $this -> Redis -> scard('uid:' . $uid . ':publicRids');
760
761         //create a temp redis list of the user's public rids (needed to lrange it for pagination)
```

```
762         $rates = false;
763         if ($numOfPublicRates)
764         {
765             $publicRids = $this -> Redis -> smembers('uid:' . $uid . ':publicRids');
766             sort($publicRids);
767             foreach ($publicRids as $prid)
768             {
769                 $this -> Redis -> lpush('tempList:' . $uid, $prid);
770             }
771             $list = 'tempList:' . $uid;
772 
773             $rids = $this -> Redis -> lrange($list, 0, 4);
774             if (!$rids)
775 
776             {
777                 return false;
778             }
779             $i = 0;
780             foreach ($rids as $rid)
781             {
782                 //get all the rid hash info
783                 $rates[$i] = $this -> Redis -> hgetall('rid:' . $rid);
784                 //add rid, avg score,rater's username and total # of ratings('count') on that resource
785                 $rates[$i]['rid'] = $rid;
786                 $rates[$i]['average_score'] = $this -> Redis -> get('url:' . $this -> Redis -> hget('rid:' . $rid, 'url') . ':average_score');
787                 $rates[$i]['username'] = $this -> Redis -> hget('uid:' . $rates[$i]['uid'], 'username');
788                 $rates[$i]['count'] = $this -> Redis -> hlen('url:' . $rates[$i]['url']);
789                 //check if I have rated it (to show 'rate it' btn or 'You have rated it' alert in rate markup)
790                 if ($this -> Redis -> sismember('uid:' . $this -> Udata['uid'] . ':rated_urls', $rates[$i]['url']))
791                 {
792                     $rates[$i]['metoo'] = true;
793                 }
794                 $i = $i + 1;
795             }
796 
797         }
798 
799         return View::make('userRates', array(
800             'username' => $this -> Udata['username'], //the user that is logged in
801             'verified' => $this -> Udata['verified'],
802             'rates' => $rates,
803             'user' => $this -> Redis -> hget('uid:' . $uid, 'username'), //the user of whom the rates i want to show
804             'totalRids' => $totalRids,
805             'public' => $numOfPublicRates,
806             'uid' => $uid,
807         ));
808 
809     }
810 
811     ////////////////////////////////////////////////////////////////////////
812     /*
813      * Pagination
814      * called with ajax, returns html markup with the rates of one 'page'
815      */
816 
817     public function paginate()
818     {
819         //get the number of page to paginate as sent with ajax
820         $currentRates = Input::get('currentRates');
821 
822         //get the view for which pagination is requested  (timeline-myRates-userRates)
823         $case = Input::get('case');
824         // redis 'lrange' command will be used to retrieve the rids we need.
825         //set the list to apply lrange for each case
826         switch ($case)
827         {
828             case 'timeline' :
829                 $list = "timeline_ratings:" . $this -> Udata['uid'];
830                 $totalRids = $this -> Redis -> llen($list);
831                 break;
832             case 'myRates' :
833                 $list = "own_ratings:" . $this -> Udata['uid'];
834                 $totalRids = $this -> Redis -> llen($list);
835                 break;
836             case 'userRates' :
837                 $userId = Input::get('userId');
838 
839                 //in this case we want to retrieve the public rates of the user, stored in a redis set (not a list as above cases)
840                 $publicSet = 'uid:' . $userId . ':publicRids';
841                 //count the public rids of the user
842                 $totalRids = $this -> Redis -> scard($publicSet);
843                 //create a temp redis list of public rids (needed to lrange it for pagination)
844                 $publicRids = $this -> Redis -> smembers($publicSet);
845                 sort($publicRids);
846                 foreach ($publicRids as $prid)
```

```
847             {
848                 $this -> Redis -> lpush('tempList:' . $userId, $prid);
849             }
850             $list = 'tempList:' . $userId;
851             break;
852         }//end of switch
853         // set num of rates per page
854         $items = 5;
855         //$pages = ceil($totalRids/$items);
856         //initialize rates array (will hold each rate's data needed for the rate markup)
857         $rates = array();
858         $page = "empty";
859 
860         if ($totalRids - $currentRates > 0)//if any more rates to show
861         {
862             //set the range of items to fetch from the ratings list
863             $start = $currentRates;
864             $end = $start + $items - 1;
865             //get the rids to show
866             $rids = $this -> Redis -> lrange($list, $start, $end);
867 
868             $i = 0;
869             foreach ($rids as $rid)
870             {
871                 //get all the rid hash info
872                 $rates[$i] = $this -> Redis -> hgetall('rid:' . $rid);
873                 //add rid, avg score,rater's username and total # of ratings('count') on that resource
874                 $rates[$i]['rid'] = $rid;
875                 $rates[$i]['average_score'] = $this -> Redis -> get('url:' . $this -> Redis -> hget('rid:' . $rid, 'url') . ':average_score');
876                 $rates[$i]['username'] = $this -> Redis -> hget('uid:' . $rates[$i]['uid'], 'username');
877                 $rates[$i]['count'] = $this -> Redis -> hlen('url:' . $rates[$i]['url']);
878                 //check if I have rated it (to show 'rate it' btn or 'You have rated it' alert in rate markup)
879                 if ($this -> Redis -> sismember('uid:' . $this -> Udata['uid'] . ':rated_urls', $rates[$i]['url']))
880                 {
881                     $rates[$i]['metoo'] = true;
882                 }
883                 $i = $i + 1;
884             }
885             $page = "";
886             //create the html to return (a 'page' with the rates fetched)
887             foreach ($rates as $rate)
888             {
889                 $page = $page . View::make('rateMarkup', array(
890                     'rate' => $rate,
891                     'username' => $this -> Udata['username']
892                 )) -> render();
893             }
894 
895         }
896 
897         //delete the temp list (if case was userRates)
898         if ($case === 'userRates')
899         {
900             $this -> Redis -> del($list);
901         }
902         //return the page with tha rate markups
903         return $page;
904     }
905 
906     /*
907      * Returns rate markup along with the rate data.
908      * Called by AJAX in Home.
909      */
910     public function rateMarkup()
911     {
912 
913         $rid = Input::get('rate');
914         $rate = $this -> Redis -> hgetall('rid:' . $rid);
915         $rate['rid'] = $rid;
916         $rate['average_score'] = $this -> Redis -> get('url:' . $rate['url'] . ':average_score');
917         $rate['username'] = $this -> Redis -> hget('uid:' . $rate['uid'], 'username');
918         $rate['count'] = $this -> Redis -> hlen('url:' . $rate['url']);
919 
920         //check if I have rated it (to show 'rate it' btn or 'You have rated it' alert in rate markup)
921         if ($this -> Redis -> sismember('uid:' . $this -> Udata['uid'] . ':rated_urls', $rate['url']))
922         {
923             $rate['metoo'] = true;
924         }
925 
926         return View::make('rateMarkup', array(
927             'username' => $this -> Udata['username'],
928             'verified' => $this -> Udata['verified'],
929             'uid' => $this -> Udata['uid'],
930             'rate' => $rate
931         ));
```

```
932     }
933
934     /*
935      * Returns minified rate markup along with the rate data.
936      * Called by AJAX in signedInBase.
937      */
938     public function miniRateMarkup()
939     {
940
941         $rid = Input::get('rate');
942         $rate['url'] = $this -> Redis -> hget('rid:' . $rid, 'url');
943         $rate['score'] = $this -> Redis -> hget('rid:' . $rid, 'score');
944         $rate['title'] = $this -> Redis -> hget('rid:' . $rid, 'title');
945         $rate['rid'] = $rid;
946         return View::make('minifiedRateMarkup', array('rate' => $rate, ));
947     }
948
949     /*
950      *  Returns all user's notifications and resets unchecked notifications counter
951      */
952     public function notifications()
953     {
954         // Change notifications counter to 0
955         $notificationsCount = $this -> Redis -> set('uncheckedNotifications:' . $this -> Udata['uid'], 0);
956         // Get user's notifications
957         $notificationsPlainText = $this -> Redis -> zrevrange('notifications:' . $this -> Udata['uid'], 0, -1, array('withscores' => TRUE));
958         // Edit each result, values are of the form uid:text ( follow action ) or uid:text:rid ( rate action )
959         $notifications = false;
960         for ($i = 0; $i < count($notificationsPlainText); $i++)
961         {
962             // Seperate value to uid , text and rid ( if exists )
963             $uidAndMessage = explode(':', $notificationsPlainText[$i][0]);
964             $username = $this -> Redis -> hget('uid:' . $uidAndMessage[0], 'username');
965             $notifications[$i]['username'] = $username;
966             $notifications[$i]['uid'] = $uidAndMessage[0];
967             $notifications[$i]['text'] = $uidAndMessage[1];
968             $notifications[$i]['timestamp'] = $notificationsPlainText[$i][1];
969             // If rid exists
970             if (!empty($uidAndMessage[2]))
971             {
972                 $notifications[$i]['rid'] = $uidAndMessage[2];
973                 $notifications[$i]['title'] = $this -> Redis -> hget('rid:' . $notifications[$i]['rid'], 'title');
974                 $notifications[$i]['url'] = $this -> Redis -> hget('rid:' . $notifications[$i]['rid'], 'url');
975                 $notifications[$i]['score'] = $this -> Redis -> hget('url:' . $notifications[$i]['url'].':uscores', $notifications[$i]['uid']);
976             }
977         }
978
979         return View::make('notifications', array(
980             'username' => $this -> Udata['username'],
981             'verified' => $this -> Udata['verified'],
982             'notificationsCount' => null,
983             'notifications' => $notifications,
984         ));
985     }
986
987 } //end-of-class
988
```

UserController.php

```php
<?php

class UserController extends BaseController {

    private $isSignedin; // stored result of isSignedin method

    public function __construct()
    {
        parent::__construct();
        //current constructor inherit parent from BaseController.

        // exist hear because need to use return value and user data: id, username
        // in signout, searchUser, handleSearchUser methods
        $this->isSignedin = $this->isSignedin();
    }

    /**
     * Retrieves values from form fields and creates new user
     * Calls signupUser() method in model app/models/User.php
     *
     */
    public function handleSignup()
    {
        // Fetch all request data.
        $formData = Input::all();

        // Build the validation constraint set.
        $rules = array(

            'emailUp'     => 'required|email|already_exists',
            'passwordUp'  => 'required|min:4'
        );

        // Create a new validator instance
        $validator = Validator::make($formData, $rules);

        if ($validator -> fails())
        {
            // Validator fails, form fields dont comply with the rules
            return Redirect::to('/') -> withErrors($validator) -> withInput(Input::only('emailUp'));
        }
        //  Calculate the md5 hash of the string
        $formData['password']   = md5($formData['passwordUp']);
        $formData['email']      = $formData['emailUp'];
        $formData['verifyCode'] = md5(rand());
        $formData['auth']       = md5(rand());

        // Call method for user creation
        $User = new User;
        $User -> signup($formData);

        // send email and use views/emails/verify.blade.php file
        Mail::send('emails.verify', $formData, function($message)
        {
            //$message->from('triple@gmail.com', 'Laravel' );
            $message->to(Input::get('emailUp') , 'Recipient')->subject('Welcome to the Laravel 4 App!');
        });

        // Successful signup
        return Redirect::to('/');
    }

    /*
     * Get the values from sign in form and confirm if
     * the email and the password exist in database
     */
    public function handleSignin() {

        $formData = Input::all();

        $rules = array(

            'emailIn'       => 'required|exist_email',
            'passwordIn'    => 'required|exist_password:'.$formData['emailIn'].','.$formData['passwordIn']
            );

        //create instance of validator and pass form data and the rules
        $validator = Validator::make($formData, $rules);

        if ($validator -> fails())
        {
```

```
82              // Return to Sign In form with Errors and the email been given
83              return Redirect::to('/') -> withErrors($validator) -> withInput(Input::only('emailIn'));
84          }
85
86          $currentUid = $this -> Redis -> hget('email', $formData['emailIn']);
87
88          $formData['uid']      = $currentUid;
89          $formData['auth']     = md5(rand()); //create a md5 number for authentication code
90
91          //Set cookie auth and parse to the next response
92          Cookie::queue('auth', $formData['auth'], time() + 3600 * 24 * 365);
93
94          //Create new field in user hash named auth with value a random md5 number.
95          //Store in a new hash with field the md5 number the value of user id.
96          $this -> Redis -> hset('uid:' . $formData['uid'], 'auth', $formData['auth']);
97          $this -> Redis -> hset('auth', $formData['auth'], $formData['uid']);
98
99          //Redirect to url that was brefore the system redirect user to sign in. Else to default (profile)
100         return Redirect::intended('/');
101     }
102
103     /*
104      * Method reliable for the email verification.
105      * Get the verification code that comes from the reply
106      * email and proceed to the account activation.
107      */
108     public function confirm($verifyCodeSent)
109     {
110         if (empty($verifyCodeSent))
111         {
112             return Redirect::to('/') -> withErrors('Error with account verification');
113         }
114
115         $uid =  $this -> Redis -> hget('verifyCode', $verifyCodeSent);
116
117         $verifyCodeStored = $this -> Redis -> hget('uid:'.$uid, 'verifyCode');
118
119         if ($verifyCodeStored != $verifyCodeSent)
120         {
121             return Redirect::to('/') -> withErrors('Error with account verification');
122         }
123
124         //$dbAuth = $this -> Redis -> hget('uid:'.$uid, 'auth');
125         //$this -> Redis -> hdel('uid:'.$uid, 'auth');
126         //$this -> Redis -> hdel('auth', $dbAuth);
127         $this -> Redis -> hdel('uid:'.$uid, 'verifyCode');
128         $this -> Redis -> hdel('uid:'.$uid, 'timestamp');
129         $this -> Redis -> hdel('verifyCode', $verifyCodeStored);
130         $this -> Redis -> hset('uid:'.$uid, 'verified', 1);
131
132         return Redirect::to('/') -> withErrors('Your account has been activated!');
133     }
134
135     /*
136      * Sign out Method
137      */
138     public function signout()
139     {
140         if (!$this->isSignedin)
141         {
142             return Redirect::to('/');
143         }
144
145         $cookie = Crypt::decrypt($_COOKIE['auth']);
146
147         //Retrieve user data
148         $uid     = $this -> Redis -> hget('auth', $cookie);
149         $DBAuth  = $this -> Redis -> hget('uid:' . $uid, 'auth');
150
151         //Delete md5 number from hush filed to set user sign out
152         $this -> Redis -> hdel('uid:' . $uid, 'auth');
153         $this -> Redis -> hdel('auth', $DBAuth);
154
155         return Redirect::to('/');
156     }
157
158     /*
159      * Make search user view
160      */
161     public function searchUser()
162     {
163         //parse to userSignedin.blade.php
164         return View::make('searchUser') ->with('username', $this->Udata['username'])
165                                         ->with('verified', $this->Udata['verified'])
166                                         ->with('semistop', FALSE);
```

```
167     }
168
169     /*
170      * Search user method
171      */
172     public function handleSearchUser()
173     {
174         $formdata['searchUser'] = Input::get('searchUser');
175
176         // Build the validation constraint set.
177         $rules = array('searchUser' => 'required');
178
179         // Create a new validator instance
180         $validator = Validator::make($formdata, $rules);
181
182         if ($validator -> fails())
183         {
184             if (Request::ajax())
185             {
186                 // render cause need searchUserResults.php file with validator erros for updating over AJAX
187                 $html = View::make('userBase')->withErrors($validator)->render();
188
189                 return Response::json(array(
190
191                     'status' => TRUE,
192                     'formCase' => 'searchUser',
193                     'view'   => $html
194                 ));
195             }
196
197             return Redirect::to('searchuser')->withErrors($validator);
198         }
199
200         //get current user id
201         $formdata['uid']      = $this->Udata['uid'];
202         $metaphone            = new DoubleMetaPhone;
203         $formdata['keywords'] = $metaphone->getKeywords($formdata['searchUser']);
204
205         // protect from metaphone return nothing
206         if (!count($formdata['keywords']))
207         {
208             if (!Request::ajax())
209             {
210                 return Redirect::to('searchuser')->withErrors('Nothing found..');
211             }
212
213             $html = View::make('userBase')->withErrors('Nothing found..')->render();
214
215             return Response::json(array(
216
217                 'status' => TRUE,
218                 'formCase' => 'searchUser',
219                 'view'   => $html
220             ));
221         }
222
223         // search for max two different strings from search field. Remove the rests
224         if (sizeof($formdata['keywords'] ) > 2)
225         {
226             array_splice($formdata['keywords'], 2);
227         }
228
229         // add prefix user: before every string coming from search user field
230         array_walk($formdata['keywords'], function(&$value, $key)
231         {
232             $value = 'users:'.$value;
233         });
234
235         //instance of User model and call search function
236         $User    = new User();
237         $results = $User->search($formdata);
238
239         if (!$results)
240         {
241             if (Request::ajax())
242             {
243                 // render cause need searchUserResults.php file with the results of search for updating over AJAX
244                 $html = View::make('userBase')->withErrors('Nothing found..')
245                                               ->with('semistop', FALSE)
246                                               ->render();
247
248                 $more = View::make('moreUsers')->with('semistop', $results['semistop'])->render();
249
250                 return Response::json(array(
251
```

```
252                 'status' => TRUE,
253                 'formCase' => 'searchUser',
254                 'view'   => $html,
255                 'more'   => $more
256             ));
257         }
258 
259         return Redirect::to('searchuser')->withErrors('Nothing found..');
260     }
261 
262     if (Request::ajax())
263     {
264         $html = View::make('userBase')->with('results', $results['results'])
265                                       ->with('semistop', $results['semistop'])
266                                       ->render();
267 
268         $more = View::make('moreUsers')->with('semistop', $results['semistop'])->render();
269 
270         return Response::json(array(
271 
272             'status' => TRUE,
273             'formCase' => 'searchUser',
274             'view'   => $html,
275             'more'   => $more
276         ));
277     }
278 
279     return View::make('searchUser')->with('results', $results['results'])
280                                    ->with('semistop', $results['semistop'])
281                                    ->with('username', $this->Udata['username'])
282                                    ->with('verified' , $this->Udata['verified']);
283 }
284 
285 /*
286  * Show more search users results every time user scroll at the
287  * bottom of the page
288  */
289 public function moreUsers()
290 {
291     $start   = Input::get('semistop');
292     $User    = new User();
293     $results = $User->pagination(++$start, $this->Udata['uid']); //++$start avoid show the last result as first
294 
295     if (Request::ajax())
296     {
297         $html = View::make('userBase')->with('results', $results['results'])
298                                       ->with('semistop', $results['semistop'])
299                                       ->render();
300 
301         $more = View::make('moreUsers')->with('semistop', $results['semistop'])->render();
302 
303         return Response::json(array(
304 
305             'status' => TRUE,
306             'formCase' => 'moreUsers',
307             'view'   => $html,
308             'more'   => $more
309         ));
310     }
311 }
312 ////////////////////////////
313 public function getSidebarProfileInfo()
314 {
315     $info['ratings'] = $this ->Redis-> llen('own_ratings:'.$this->Udata['uid']);
316     $info['followers'] = $this ->Redis-> zcard('followers:'.$this->Udata['uid']);
317     $info['followings'] = $this ->Redis-> zcard('followings:'.$this->Udata['uid']);
318 
319     return Response::json(array(
320         'numOfRatings'      => $info['ratings'],
321         'numOfFollowers'    => $info['followers'],
322         'numOfFollowings'   => $info['followings']
323     ));
324 
325 }
326 
327 }
```

Websocket – server

server.js

```
1   // Initialization
2   var express = require('express');
3   var redis = require('redis');
4   var io = require('socket.io');
5   var http = require('http');
6   var app = express();
7   var server = http.createServer(app).listen(3000, 'localhost');
8
9   console.log('Server successfully started...');
10  // Client connection
11  io.listen(server).on('connection', function(client)
12  {
13      var redisClientNews = redis.createClient();
14      redisClientNews.subscribe('newRates');
15      var redisClientNotifications = redis.createClient();
16      var redisClient = redis.createClient();
17
18      client.on('uDataNotifications', function(data)
19      {
20          var subscribed = false;
21          var redisClientAuth = redis.createClient();
22          var uDataNotifications = data.split(',');
23          redisClientAuth.hget('uid:' + uDataNotifications[0], 'auth', function(err, reply)
24          {
25              if (reply == uDataNotifications[1])
26              {
27                  redisClientNotifications.subscribe('notifications:' + uDataNotifications[0]);
28                  subscribed = true;
29              }
30          });
31          redisClientAuth.quit();
32          redisClientAuth.on("end", function(err)
33          {
34              if (!subscribed)
35              {
36                  client.disconnect();
37              }
38          });
39      });
40
41      // Recieve credentials and subscribe client to Timeline update channel or kill connection
42      client.on('uData', function(data)
43      {
44
45          var subscribed = false;
46          var redisClientAuth = redis.createClient();
47          var uData = data.split(',');
48          redisClientAuth.hget('uid:' + uData[0], 'auth', function(err, reply)
49          {
50              if (reply == uData[1])
51              {
52                  redisClient.subscribe('timeline:' + uData[0]);
53                  subscribed = true;
54              }
55          });
56          redisClientAuth.quit();
57          redisClientAuth.on("end", function(err)
58          {
59              if (!subscribed)
60              {
61                  client.disconnect();
62              }
63          });
64      });
65      // When a string is published on channel Timeline:xxx send it to client
66      redisClient.on('message', function(channel, message)
67      {
68          client.emit(channel, message);
69      });
70
71      redisClientNews.on('message', function(channel, message)
72      {
73          client.emit(channel, message);
74      });
75
76      redisClientNotifications.on('message', function(channel, message)
77      {
78          client.emit(channel, message);
79      });
80
81      client.on('disconnect', function()
82      {
```

```
83            redisClient.quit();
84            redisClientNews.quit();
85            redisClientNotifications.quit();
86        });
87
88    });
89
90
```

Βιβλιοθήκες

StorageFactory.php

```php
<?php

    /*
     * Design pattern factory
     * Every instance of the class return the appropriate instance
     * based on config item 'cloudStorage' placed into app/config/app.php
     */

    class StorageFactory
    {
        public static function getStorage()
        {
            if (Config::get('app.cloudStorage'))
            {
                $storage = new CloudStorage();
            }

            else
            {
                $storage = new LocalStorage();
            }

            return $storage;
        }
    }

    /*
     * Specify the method below
     */

    interface StorageInterface
    {
        public function storeAvatarThumb($dataA);
        public function storeRateThumb($dataR);
    }

    /*
     * Contains Redis instance and intreface's methods
     */

    class Storage implements StorageInterface
    {
        protected $Redis;

        public function __construct()
        {
            $this->Redis = Redis::connection();
        }

        public function storeAvatarThumb($dataA)
        {
            return;
        }

        public function storeRateThumb($dataR)
        {
            return;
        }
    }

    /*
     * Hanlde local storage of User avatar & Rate avatr
     */

    class LocalStorage extends Storage
    {
        function __construct()
        {
            parent::__construct();
        }

        public function storeAvatarThumb($dataL)
        {
            //move the uploaded file from server's temp location to 'public_path()./images/avatar/'
            Input::file('avatar')->move( public_path().'/images/avatar/', $dataL['filename']);

            // store url into user data
            $this->Redis->hset ('uid:'.$dataL['uid'], 'thumb', 'images/avatar/'.$dataL['filename']);

            return;
        }

        public function storeRateThumb($dataL)
```

```
80          {
81              // store url into rid data
82              $this->Redis->hset('rid:' . $dataL['rid'], 'thumb', $dataL['thumbUrl']);
83
84              return;
85          }
86      }
87
88      /*
89       * Hanlde cloud storage of User avatar & Rate avatar
90       */
91      class CloudStorage extends Storage
92      {
93          function __construct()
94          {
95              parent::__construct();
96          }
97
98          public function storeAvatarThumb($dataC)
99          {
100             $url = $this->s3PutRequest($dataC);
101
102             // store url in user data
103             $this->Redis->hset ('uid:'.$dataC['uid'], 'thumb', $url);
104
105             return;
106         }
107
108         public function storeRateThumb($dataC)
109         {
110             $thumbId = $this->Redis->get('nextThumbId');
111
112             //prepare data for parsing to s3PutRequest()
113             $dataC['filename']  = 'thumb'.$thumbId.'.jpg';
114             $dataC['filepath']  = public_path().$dataC['thumbUrl'];
115             $dataC['filetype']  = 'image/jpeg';
116             $dataC['bucket']    = 'ratethumb';
117
118             $url = $this->s3PutRequest($dataC);
119
120             // store url in rid data
121             $this->Redis->hset('rid:'.$dataC['rid'], 'thumb', $url);
122
123             // thumbnail captured when a new rate created and stored into server.
124             // Delete that thumb after cloud storage action
125             unlink($dataC['filepath']);
126
127             return;
128         }
129
130         private function s3PutRequest($DataS3)
131         {
132             //instance of AWS SDK
133             $s3obj = AWS::get('s3');
134
135             // put request store given thumb and return the desirable url
136             $url = $s3obj->putObject(array(
137
138                     'Bucket'     => $DataS3['bucket'],   // sub folder of s3 storage
139                     'Key'        => $DataS3['filename'], // set the name of the uploaded file
140                     'SourceFile' => $DataS3['filepath'], // where find the file
141                     'ACL'        => 'public-read',       // set access permissions
142                     'ContentType'=> $DataS3['filetype'], // set type of file
143                 ));
144
145             return $url['ObjectURL'];
146         }
147     }
148
149
150
151
152
153
```

UrlManagement.php

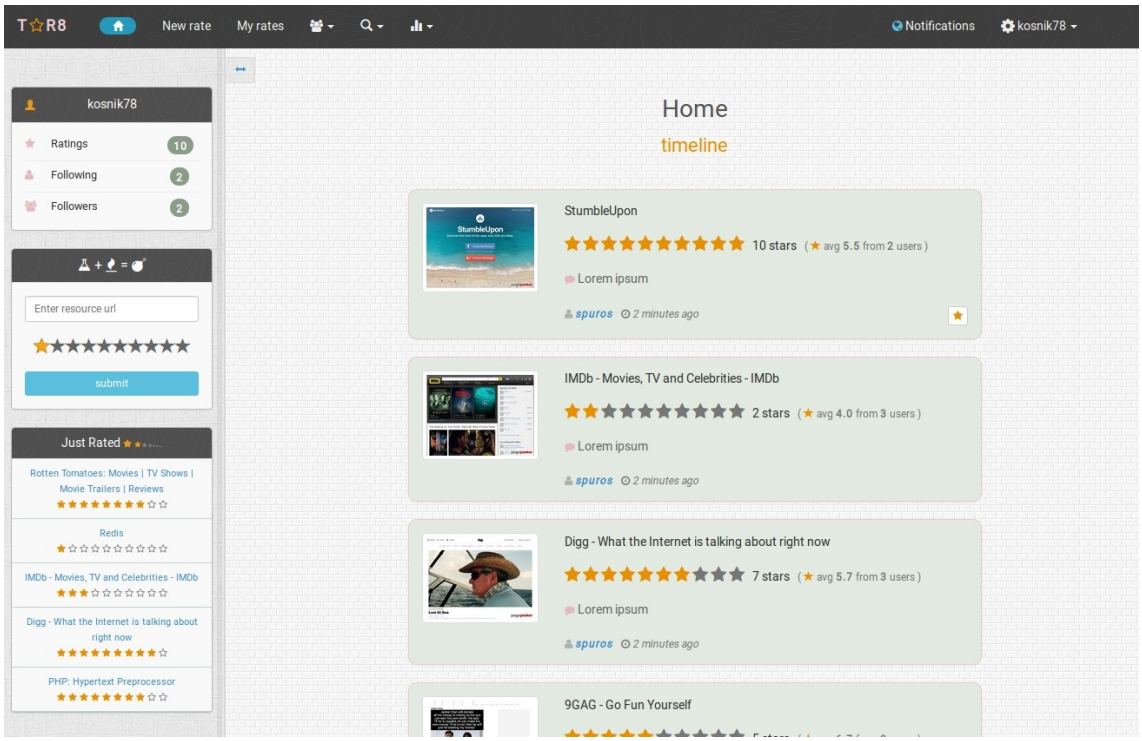
```php
<?php
class UrlManagement extends BaseModel
{
    public function __construct()
    {
        parent::__construct();
    }

    //class variable $mimetypes: array with all the known filetypes
    public $mimeTypes = array(
        "323" => "text/h323",
        "acx" => "application/internet-property-stream",
        "ai" => "application/postscript",
        "aif" => "audio/x-aiff",
        "aifc" => "audio/x-aiff",
        "aiff" => "audio/x-aiff",
        "asf" => "video/x-ms-asf",
        "asr" => "video/x-ms-asf",
        "asx" => "video/x-ms-asf",
        "au" => "audio/basic",
        "avi" => "video/x-msvideo",
        "axs" => "application/olescript",
        "bas" => "text/plain",
        "bcpio" => "application/x-bcpio",
        "bin" => "application/octet-stream",
        "bmp" => "image/bmp",
        "c" => "text/plain",
        "cat" => "application/vnd.ms-pkiseccat",
        "cdf" => "application/x-cdf",
        "cer" => "application/x-x509-ca-cert",
        "class" => "application/octet-stream",
        "clp" => "application/x-msclip",
        "cmx" => "image/x-cmx",
        "cod" => "image/cis-cod",
        "cpio" => "application/x-cpio",
        "crd" => "application/x-mscardfile",
        "crl" => "application/pkix-crl",
        "crt" => "application/x-x509-ca-cert",
        "csh" => "application/x-csh",
        "css" => "text/css",
        "dcr" => "application/x-director",
        "der" => "application/x-x509-ca-cert",
        "dir" => "application/x-director",
        "dll" => "application/x-msdownload",
        "dms" => "application/octet-stream",
        "doc" => "document/msword",
        "dot" => "application/msword",
        "dvi" => "application/x-dvi",
        "dxr" => "application/x-director",
        "eps" => "application/postscript",
        "etx" => "text/x-setext",
        "evy" => "application/envoy",
        "exe" => "application/octet-stream",
        "fif" => "application/fractals",
        "flr" => "x-world/x-vrml",
        "gif" => "image/gif",
        "gtar" => "application/x-gtar",
        "gz" => "application/x-gzip",
        "h" => "text/plain",
        "hdf" => "application/x-hdf",
        "hlp" => "application/winhlp",
        "hqx" => "application/mac-binhex40",
        "hta" => "application/hta",
        "htc" => "text/x-component",
        "ico" => "image/x-icon",
        "ief" => "image/ief",
        "iii" => "application/x-iphone",
        "ins" => "application/x-internet-signup",
        "isp" => "application/x-internet-signup",
        "png" => "image/png",
        "jfif" => "image/pipeg",
        "jpe" => "image/jpeg",
        "jpeg" => "image/jpeg",
        "jpg" => "image/jpeg",
        "js" => "application/x-javascript",
        "latex" => "application/x-latex",
        "lha" => "application/octet-stream",
        "lsf" => "video/x-la-asf",
        "lsx" => "video/x-la-asf",
        "lzh" => "application/octet-stream",
        "m13" => "application/x-msmediaview",
```

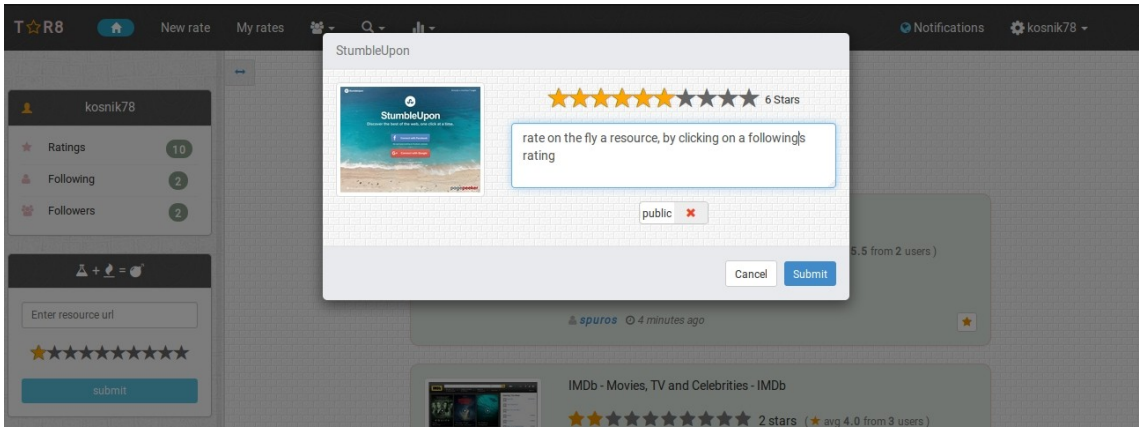
```
82      "m14" => "application/x-msmediaview",
83      "m3u" => "audio/x-mpegurl",
84      "man" => "application/x-troff-man",
85      "mdb" => "application/x-msaccess",
86      "me" => "application/x-troff-me",
87      "mht" => "message/rfc822",
88      "mhtml" => "message/rfc822",
89      "mid" => "audio/mid",
90      "mny" => "application/x-msmoney",
91      "mov" => "video/quicktime",
92      "movie" => "video/x-sgi-movie",
93      "mp2" => "video/mpeg",
94      "mp3" => "audio/mpeg",
95      "mpa" => "video/mpeg",
96      "mpe" => "video/mpeg",
97      "mpeg" => "video/mpeg",
98      "mpg" => "video/mpeg",
99      "mpp" => "application/vnd.ms-project",
100     "mpv2" => "video/mpeg",
101     "ms" => "application/x-troff-ms",
102     "mvb" => "application/x-msmediaview",
103     "nws" => "message/rfc822",
104     "oda" => "application/oda",
105     "p10" => "application/pkcs10",
106     "p12" => "application/x-pkcs12",
107     "p7b" => "application/x-pkcs7-certificates",
108     "p7c" => "application/x-pkcs7-mime",
109     "p7m" => "application/x-pkcs7-mime",
110     "p7r" => "application/x-pkcs7-certreqresp",
111     "p7s" => "application/x-pkcs7-signature",
112     "pbm" => "image/x-portable-bitmap",
113     "pdf" => "document/pdf",
114     "pfx" => "application/x-pkcs12",
115     "pgm" => "image/x-portable-graymap",
116     "pko" => "application/ynd.ms-pkipko",
117     "pma" => "application/x-perfmon",
118     "pmc" => "application/x-perfmon",
119     "pml" => "application/x-perfmon",
120     "pmr" => "application/x-perfmon",
121     "pmw" => "application/x-perfmon",
122     "pnm" => "image/x-portable-anymap",
123     "pot" => "application/vnd.ms-powerpoint",
124     "ppm" => "image/x-portable-pixmap",
125     "pps" => "application/vnd.ms-powerpoint",
126     "ppt" => "application/vnd.ms-powerpoint",
127     "prf" => "application/pics-rules",
128     "ps" => "application/postscript",
129     "pub" => "application/x-mspublisher",
130     "qt" => "video/quicktime",
131     "ra" => "audio/x-pn-realaudio",
132     "ram" => "audio/x-pn-realaudio",
133     "ras" => "image/x-cmu-raster",
134     "rgb" => "image/x-rgb",
135     "rmi" => "audio/mid",
136     "roff" => "application/x-troff",
137     "rtf" => "application/rtf",
138     "rtx" => "text/richtext",
139     "scd" => "application/x-msschedule",
140     "sct" => "text/scriptlet",
141     "setpay" => "application/set-payment-initiation",
142     "setreg" => "application/set-registration-initiation",
143     "sh" => "application/x-sh",
144     "shar" => "application/x-shar",
145     "sit" => "application/x-stuffit",
146     "snd" => "audio/basic",
147     "spc" => "application/x-pkcs7-certificates",
148     "spl" => "application/futuresplash",
149     "src" => "application/x-wais-source",
150     "sst" => "application/vnd.ms-pkicertstore",
151     "stl" => "application/vnd.ms-pkistl",
152     "stm" => "text/html",
153     "svg" => "image/svg+xml",
154     "sv4cpio" => "application/x-sv4cpio",
155     "sv4crc" => "application/x-sv4crc",
156     "t" => "application/x-troff",
157     "tar" => "application/x-tar",
158     "tcl" => "application/x-tcl",
159     "tex" => "application/x-tex",
160     "texi" => "application/x-texinfo",
161     "texinfo" => "application/x-texinfo",
162     "tgz" => "application/x-compressed",
163     "tif" => "image/tiff",
164     "tiff" => "image/tiff",
165     "tr" => "application/x-troff",
166     "trm" => "application/x-msterminal",
```

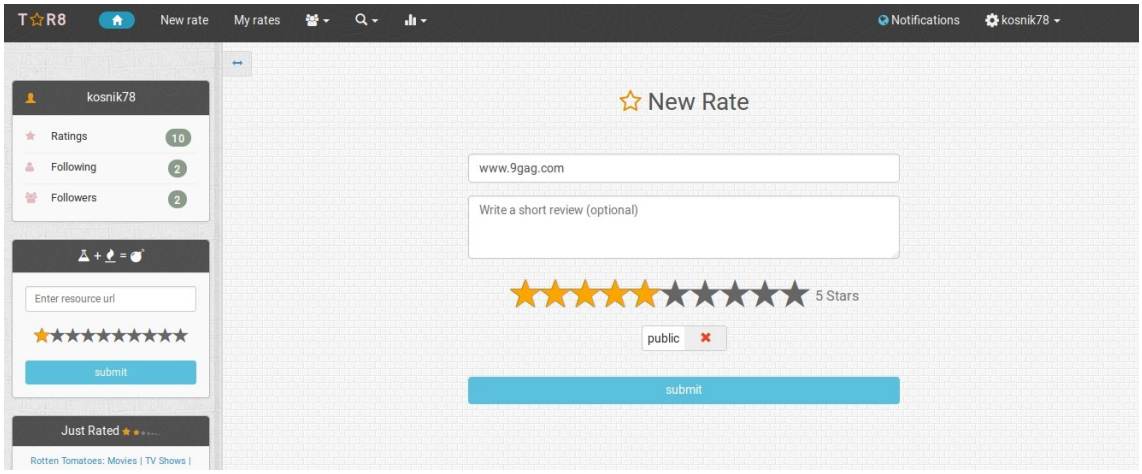
```
167         "tsv" => "text/tab-separated-values",
168         "txt" => "text/plain",
169         "uls" => "text/iuls",
170         "ustar" => "application/x-ustar",
171         "vcf" => "text/x-vcard",
172         "vrml" => "x-world/x-vrml",
173         "wav" => "audio/x-wav",
174         "wcm" => "application/vnd.ms-works",
175         "wdb" => "application/vnd.ms-works",
176         "wks" => "application/vnd.ms-works",
177         "wmf" => "application/x-msmetafile",
178         "wps" => "application/vnd.ms-works",
179         "wri" => "application/x-mswrite",
180         "wrl" => "x-world/x-vrml",
181         "wrz" => "x-world/x-vrml",
182         "xaf" => "x-world/x-vrml",
183         "xbm" => "image/x-xbitmap",
184         "xla" => "application/vnd.ms-excel",
185         "xlc" => "application/vnd.ms-excel",
186         "xlm" => "application/vnd.ms-excel",
187         "xls" => "application/vnd.ms-excel",
188         "xlsx" => "vnd.ms-excel",
189         "xlt" => "application/vnd.ms-excel",
190         "xlw" => "application/vnd.ms-excel",
191         "xof" => "x-world/x-vrml",
192         "xpm" => "image/x-xpixmap",
193         "xwd" => "image/x-xwindowdump",
194         "z" => "application/x-compress",
195         "zip" => "application/zip"
196     );
197
198     /*
199      * Function urlExists($url)
200      * Checks if url exists in database.
201      *
202      */
203     public static function urlExists($url)
204     {
205         $Redis = Redis::connection();
206
207         if (!$Redis -> hlen('url:' . $url))
208         {
209             return false;
210         }
211         return true;
212
213     }
214
215     ////////////////////////////////////////////////////////////////////
216
217     /*
218      * Function urlType($url)
219      * Returns the type of url: 'webpage' or filetype (image, document e.t.c)
220      */
221     public function urlType($url)
222     {
223         //get rid of variables in the url - if any
224         $url = strtok($url, '?');
225         $urlinfo = pathinfo($url);
226         //example: url="http://www.davey.com/media/1001/home-tree.png"
227         //pathinfo returns:
228         //Array ( [dirname] => http://www.davey.com/media/1001 [basename] => home-tree.png [extension] => png [filename] => home-tree )
229
230         //check if is webpage  (case url has no extension  or extension does not refer to known filetype)
231         if (empty($urlinfo['extension']) || !array_key_exists($urlinfo['extension'], $this -> mimeTypes))
232         {
233             $type = "webpage";
234             return $type;
235         }
236         // url is file, get filetype
237         $ext = Str::lower($urlinfo['extension']);
238         $type = strtok($this -> mimeTypes[$ext], '/');
239         //e.g. if $ext='jpg', $mimeTypes['jpg'] = image/jpeg, strtok gives 'image'
240         return $type;
241     }
242
243     ////////////////////////////////////////////////////////////////////////////////////
244     /*
245      * Function urlGetTitle($url)
246      * Returns a title for a rate
247      * Calls  function UrlType() to get the type of url, and:
248      * 1.If  url is a webpage,set as title the html 'title' tag of the <head> section.
249      * 2.If extension exists and is a known filetype, then title=filename
250      */
251
```

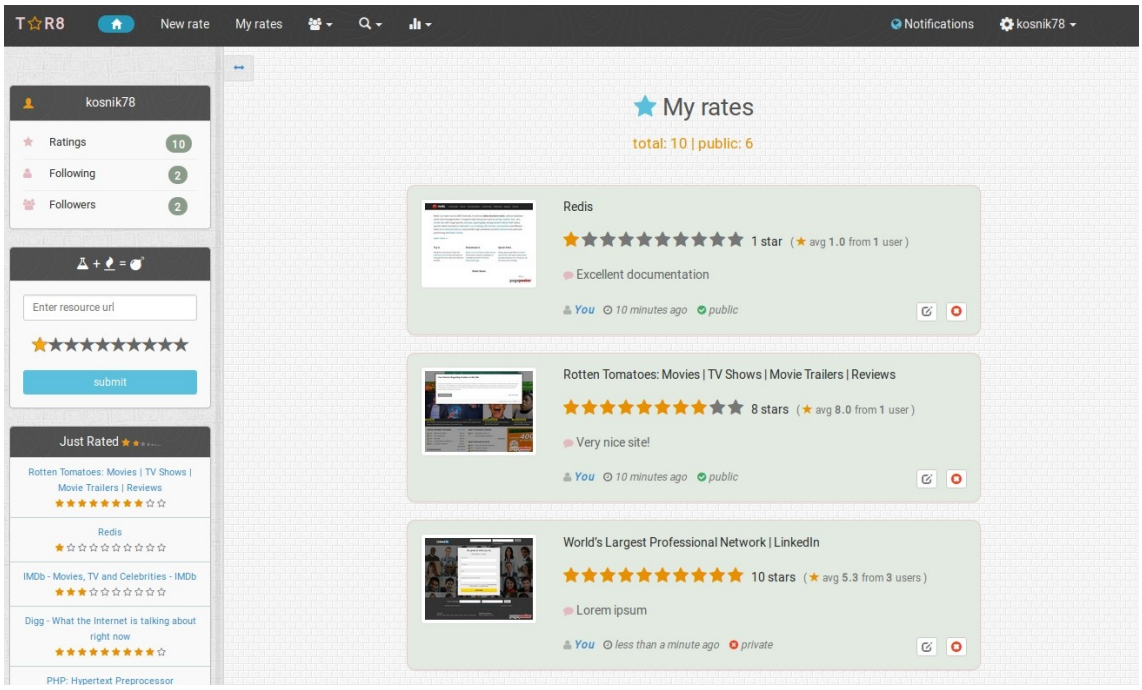
```
252     public function urlGetTitle($url)
253     {
254         $type = $this -> urlType($url);
255         if ($type == 'webpage')
256         {
257             include_once ('simple_html_dom.php');
258             // Create a DOM object
259             $html = new simple_html_dom();
260             // Load HTML from a URL
261             $html -> load_file($url);
262             //get title
263             if ($html -> find('title', 0))
264             {
265                 $title = trim($html -> find('title', 0) -> innertext);
266                 return $title;
267             }
268             // if no title retrieved, set url basename as title
269             $url = strtok($url, '?');
270             $urlinfo = pathinfo($url);
271             $title = $urlinfo['basename'];
272             return $title;
273         }
274 
275         //type is file, set title=filename
276         $url = strtok($url, '?');
277         $urlinfo = pathinfo($url);
278         $title = $urlinfo['filename'];
279         return $title;
280 
281     }
282 
283     public function urlGetThumb($url)
284     {
285         // Online Thumb generator - Download Image
286         $thumb = 'http://free.pagepeeker.com/v2/thumbs.php?size=x&url=' . $url;
287         // Online Thumb generator - Check if thumb is ready Image
288         $thumbReady = 'http://free.pagepeeker.com/v2/thumbs_ready.php?size=x&url=' . $url;
289         // Ignore HTTP error file_get_contents() function might throw
290         $context = stream_context_create(array('http' => array('ignore_errors' => true), ));
291         // Get JSON code API returns
292         $json = file_get_contents($thumbReady, false, $context);
293         // Decode JSON code
294         $response = json_decode($json);
295         $i = 0;
296         // Check for up to 30 seconds if thumb is ready
297         while (!$response -> IsReady && $i < 30)
298         {
299             sleep(1);
300             $json = file_get_contents($thumbReady, false, $context);
301             $response = json_decode($json);
302             $i++;
303         }
304         if (!$response -> IsReady)
305         {
306             return $url = FALSE;
307         }
308         else
309         {
310             // Get thumb
311             $thumb = file_get_contents($thumb, false, $context);
312             // Get next thumb ID
313             $thumbId = $this -> Redis -> incr('nextThumbId');
314             $name = 'thumb' . $thumbId . '.jpg';
315             $destinationPath = public_path() . '/images/thumbnails/' . $name;
316             // Save thumb
317             file_put_contents($destinationPath, $thumb);
318 
319             $url = '/images/thumbnails/' . $name;
320 
321             return $url;
322         }
323     }
324 }
325 
```

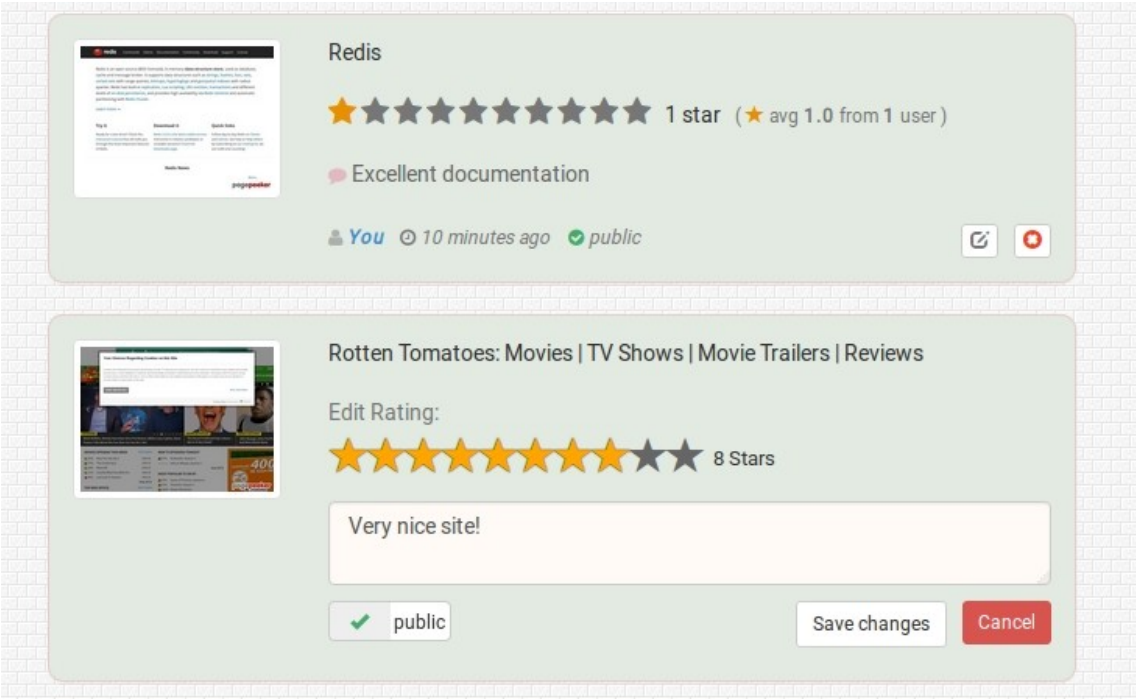
Στιγμιότυπα εφαρμογής

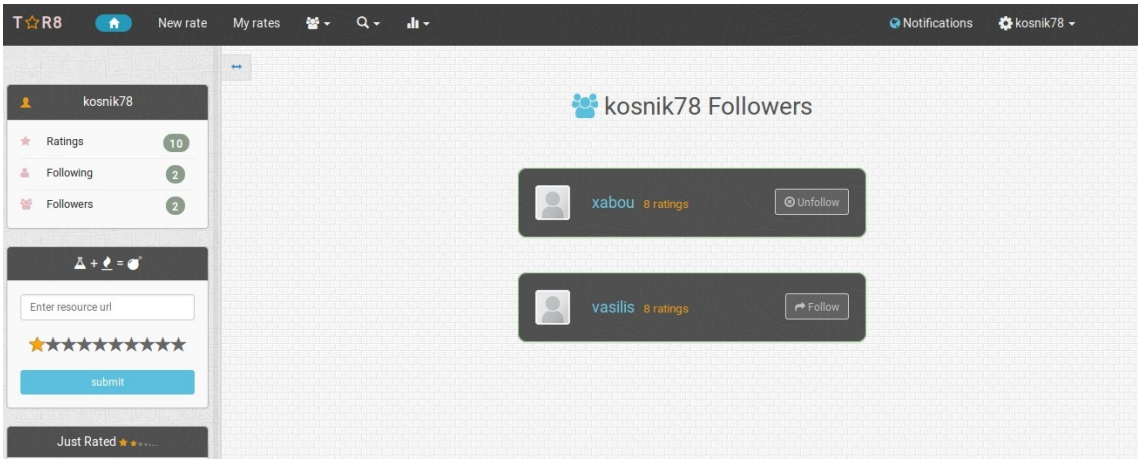


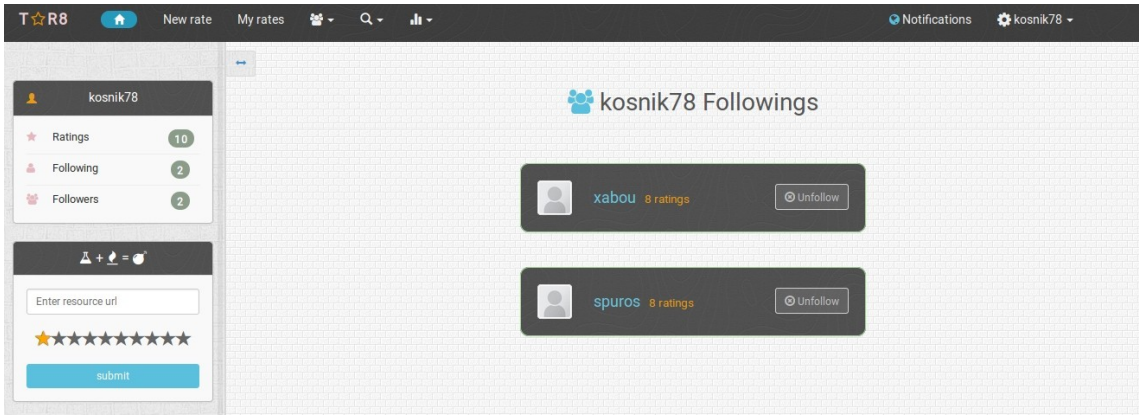


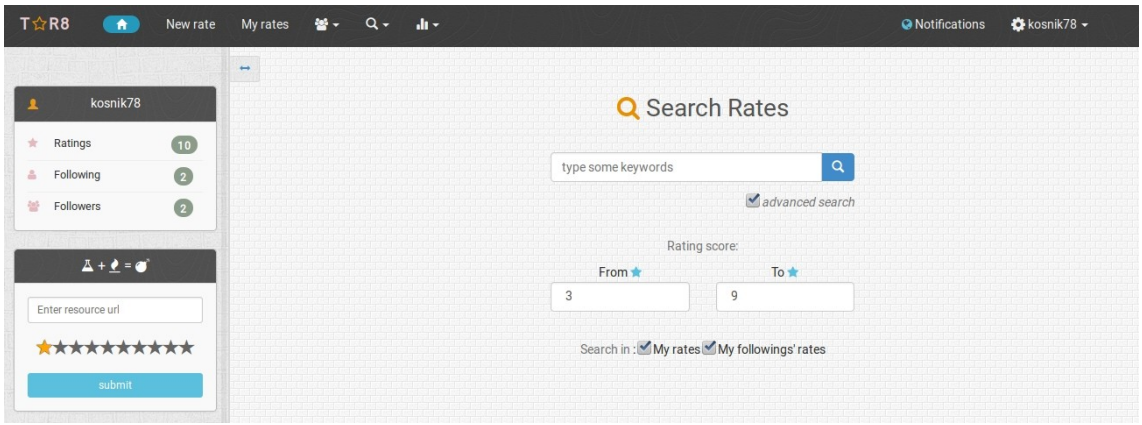

T☆R8
New rate
My rates
Notifications
kosnik78
kosnik78
Ratings 10
Following 2
Followers 2
Enter resource url
submit
Just Rated
Rotten Tomatoes: Movies | TV Shows |
☆ New Rate
www.9gag.com
Write a short review (optional)
5 Stars
public
submit

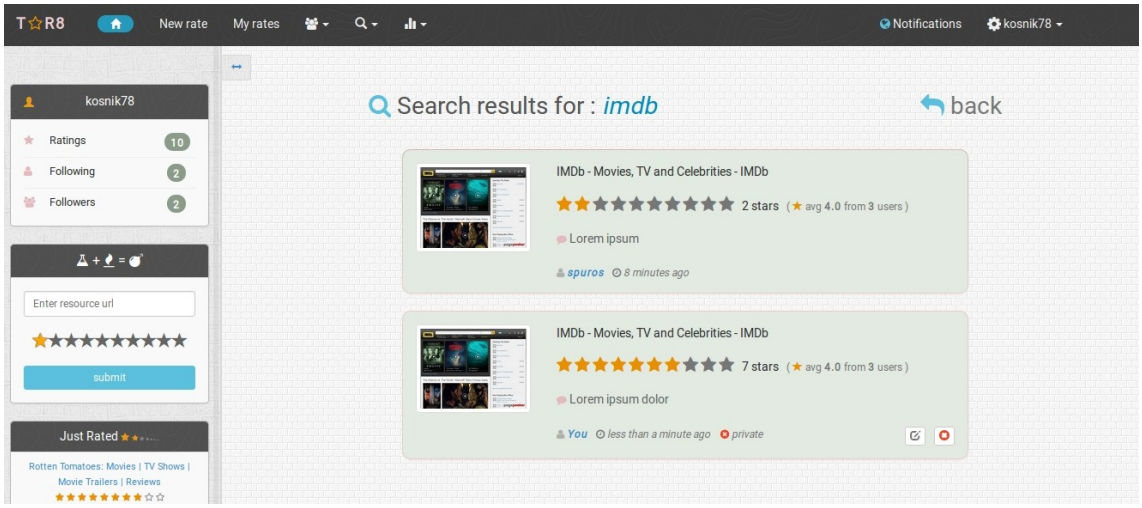

T☆R8
New rate
My rates
Notifications
kosnik78
kosnik78
Ratings 10
Following 2
Followers 2
Enter resource url
submit
Just Rated
Rotten Tomatoes: Movies | TV Shows | Movie Trailers | Reviews
Redis
IMDb - Movies, TV and Celebrities - IMDb
Digg - What the Internet is talking about right now
PHP: Hypertext Preprocessor
My rates
total: 10 | public: 6
Redis
1 star (avg 1.0 from 1 user)
Excellent documentation
You 10 minutes ago public
Rotten Tomatoes: Movies | TV Shows | Movie Trailers | Reviews
8 stars (avg 8.0 from 1 user)
Very nice site!
You 10 minutes ago public
World's Largest Professional Network | LinkedIn
10 stars (avg 5.3 from 3 users)
Lorem ipsum
You less than a minute ago private

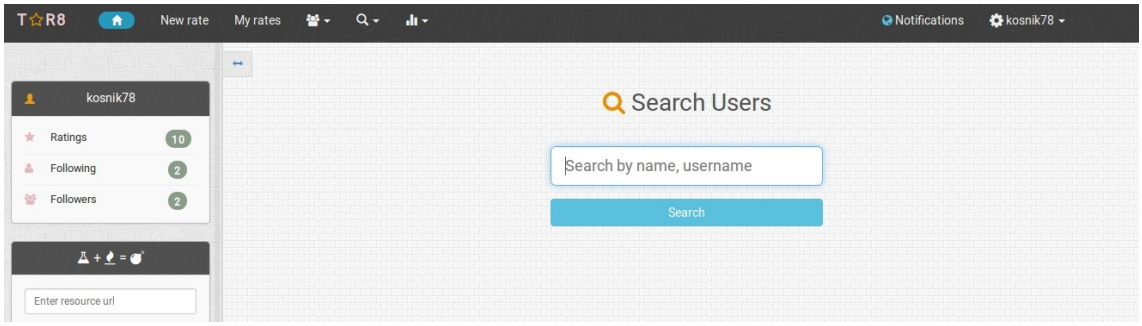

Redis
1 star (avg 1.0 from 1 user)
Excellent documentation
You 10 minutes ago public
Rotten Tomatoes: Movies | TV Shows | Movie Trailers | Reviews
Edit Rating:
8 Stars
Very nice site!
public
Save changes
Cancel

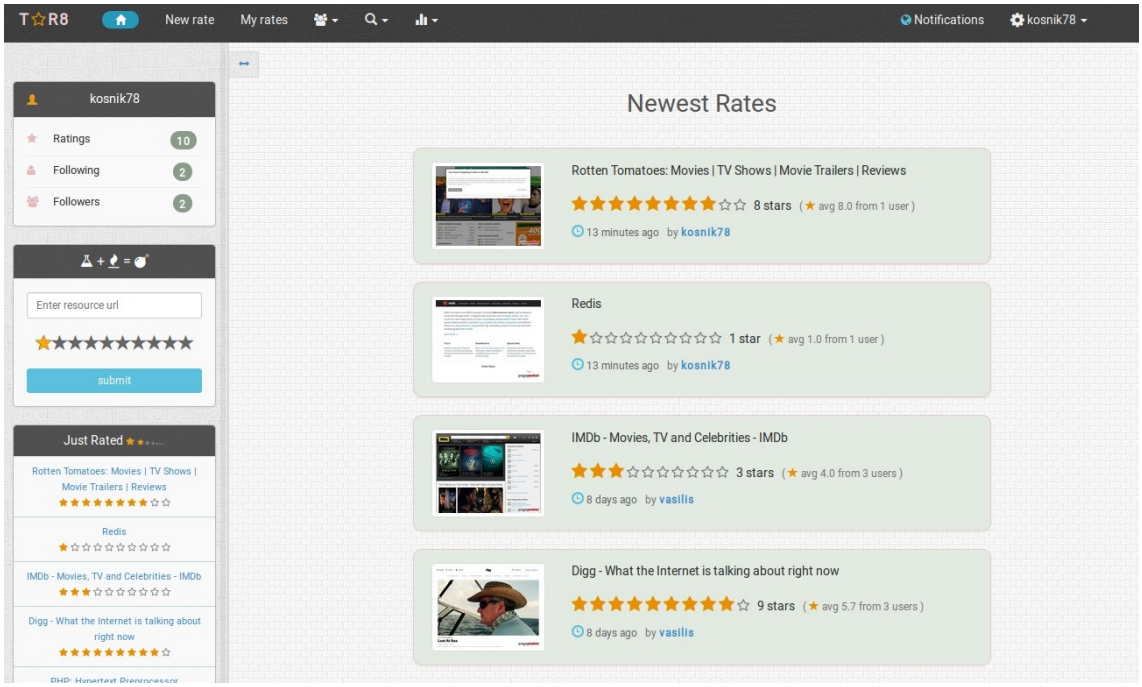

T☆R8
New rate
My rates
Notifications
kosnik78
kosnik78
Ratings 10
Following 2
Followers 2
Enter resource url
submit
Just Rated
kosnik78 Followers
xabou 8 ratings
Unfollow
vasilis 8 ratings
Follow

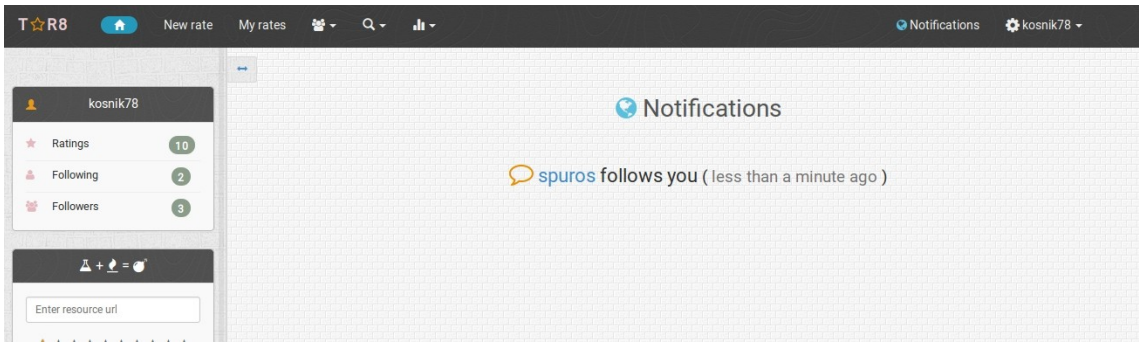

T☆R8
New rate
My rates
Notifications
kosnik78
kosnik78
Ratings 10
Following 2
Followers 2
Enter resource url
submit
kosnik78 Followings
xabou 8 ratings
Unfollow
spuros 8 ratings
Unfollow

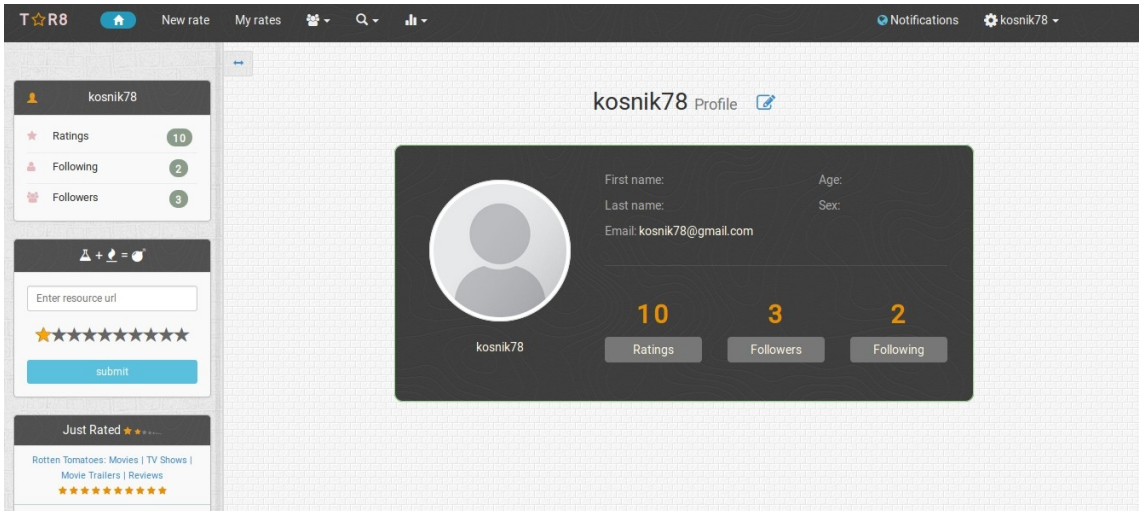

T☆R8
New rate
My rates
Notifications
kosnik78
kosnik78
Ratings 10
Following 2
Followers 2
Enter resource url
submit
Search Rates
type some keywords
advanced search
Rating score:
From
To
3
9
Search in : My rates My followings' rates

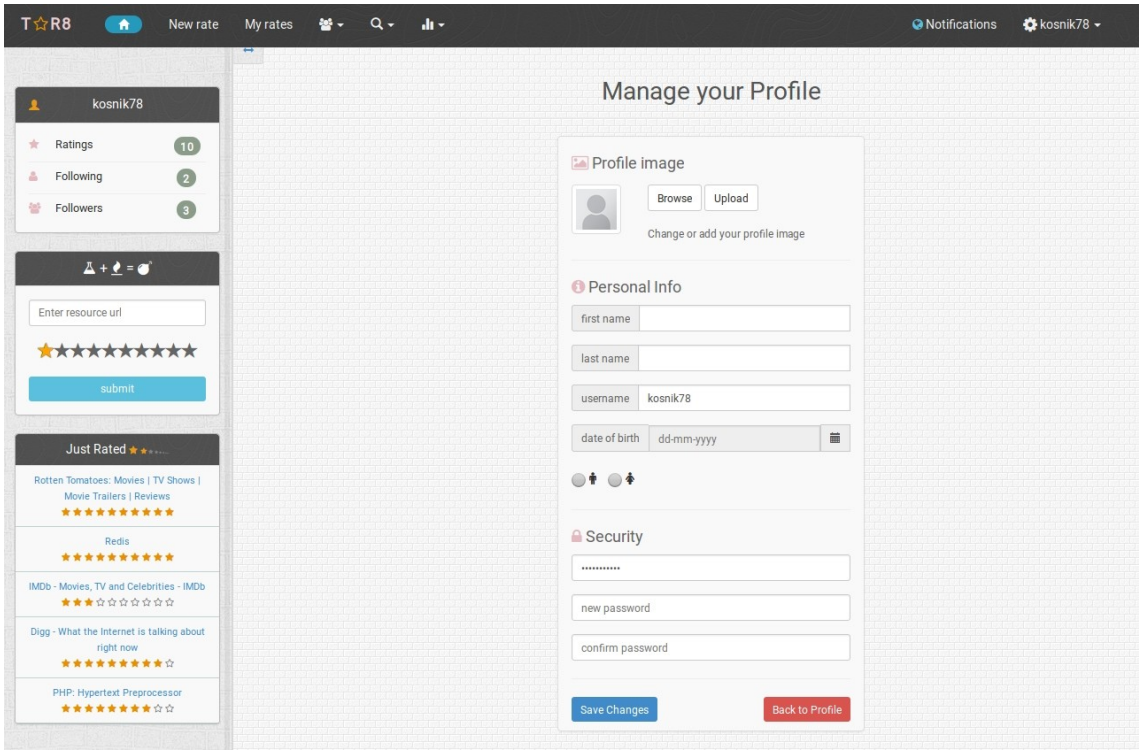

T☆R8
New rate
My rates
Notifications
kosnik78
kosnik78
Ratings 10
Following 2
Followers 2
Enter resource url
submit
Just Rated
Rotten Tomatoes: Movies | TV Shows | Movie Trailers | Reviews
Search results for : imdb
back
IMDb - Movies, TV and Celebrities - IMDb
2 stars (avg 4.0 from 3 users)
Lorem ipsum
spuros 8 minutes ago
IMDb - Movies, TV and Celebrities - IMDb
7 stars (avg 4.0 from 3 users)
Lorem ipsum dolor
You less than a minute ago private


T☆R8
New rate
My rates
Notifications
kosnik78
kosnik78
Ratings 10
Following 2
Followers 2
Enter resource url
Search Users
Search by name, username
Search


T☆R8
New rate
My rates
Notifications
kosnik78
kosnik78
Ratings 10
Following 2
Followers 2
Enter resource url
submit
Just Rated
Rotten Tomatoes: Movies | TV Shows | Movie Trailers | Reviews
Redis
IMDb - Movies, TV and Celebrities - IMDb
Digg - What the Internet is talking about right now
Newest Rates
Rotten Tomatoes: Movies | TV Shows | Movie Trailers | Reviews
8 stars (avg 8.0 from 1 user)
13 minutes ago by kosnik78
Redis
1 star (avg 1.0 from 1 user)
13 minutes ago by kosnik78
IMDb - Movies, TV and Celebrities - IMDb
3 stars (avg 4.0 from 3 users)
8 days ago by vasilis
Digg - What the Internet is talking about right now
9 stars (avg 5.7 from 3 users)
8 days ago by vasilis


T☆R8
New rate
My rates
Notifications
kosnik78
kosnik78
Ratings 10
Following 2
Followers 3
Enter resource url
Notifications
spuros follows you (less than a minute ago)


T☆R8
New rate
My rates
Notifications
kosnik78
kosnik78
Ratings 10
Following 2
Followers 3
Enter resource url
submit
Just Rated
Rotten Tomatoes: Movies | TV Shows | Movie Trailers | Reviews
kosnik78 Profile
First name:
Last name:
Email: kosnik78@gmail.com
Age:
Sex:
kosnik78
10
Ratings
3
Followers
2
Following


T☆R8
New rate
My rates
Notifications
kosnik78
kosnik78
Ratings 10
Following 2
Followers 3
Enter resource url
submit
Just Rated
Rotten Tomatoes: Movies | TV Shows | Movie Trailers | Reviews
Redis
IMDb - Movies, TV and Celebrities - IMDb
Digg - What the Internet is talking about right now
PHP: Hypertext Preprocessor
Manage your Profile
Profile image
Browse
Upload
Change or add your profile image
Personal Info
first name
last name
username
kosnik78
date of birth
dd-mm-yyyy
Security
new password
confirm password
Save Changes
Back to Profile