Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής
Πρόγραμμα Μεταπτυχιακών Σπουδών
«Πληροφορική»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Εφαρμογή αξιολόγησης διαδικτυακών πόρων με χαρακτηριστικά κοινωνικού δικτύου**
Ονοματεπώνυμο Φοιτητή	**Νικόλαος Κοσμόπουλος**
Πατρώνυμο	**Διογένης**
Αριθμός Μητρώου	**ΜΠΠΛ/ 12022**
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

Ο παγκόσμιος ιστός έχει αλλάξει ριζικά σχεδόν όλα τα πεδία που διαμορφώνουν τις κοινωνίες των λαών. Ένα από τα πιο ραγδαίως αναπτυσσόμενα, και με μεγάλη επίδραση στις κοινωνίες, προϊόντα του παγκόσμιου ιστού αποτελούν τα κοινωνικά δίκτυα, τα οποία παίζουν καθοριστικό ρόλο στην οργάνωση και την διάδοση πληροφορίας. Η εξέλιξη και η αξιοποίηση σύγχρονων τεχνολογιών του διαδικτύου οδήγησαν στην δημιουργία ιστότοπων πραγματικού χρόνου δίνοντας τη δυνατότητα στους χρήστες του διαδικτύου να αλληλεπιδρούν, να διαμορφώνουν και να διαδίδουν περιεχόμενο σε πραγματικό χρόνο.

Στην παρούσα διπλωματική εργασία επιχειρείται να παρουσιαστεί ο τρόπος με τον οποίο μπορούν να υλοποιηθούν εφαρμογές διαδικτύου που επιτρέπουν στους χρήστες να μοιραστούν, διαδώσουν περιεχόμενο στην διαδικτυακή κοινότητα που τις χρησιμοποιεί, και να ενημερώνονται σε πραγματικό χρόνο για τις ενέργειες άλλων χρηστών.

Για τον σκοπό αυτό αναπτύχθηκε μία εφαρμογή (TR8) η οποία επιτρέπει στους χρήστες να διατηρούν ένα προσωπικό προφίλ, να σχετίζονται με άλλους χρήστες με βάση ενα σχεσιακό μοντέλο παρόμοιο με αυτό του Twitter, να βαθμολογούν ιστοσελίδες, εικόνες και κάθε είδους διαδικτυακό πόρο προσβάσιμο μέσω μιας διεύθυνσης URL, και να λαμβάνουν σε πραγματικό χρόνο ενημερώσεις για την δραστηριότητα άλλων χρηστών.

Η υλοποίηση της εφαρμογής έγινε με βάση τις αρχές της σχεδιαστικής αρχιτεκτονικής MVC και με χρήση σύγχρονων προγραμματιστικών τεχνικών.

Abstract

The world wide web has radically affected almost every aspect of today's societies. Social networks form one of the most rapidly evolving and with a significant effect on societies product of the world wide web, marking an epochal role in information dissemination. The evolution and utilization of modern web technologies led to the creation of real time websites which allow users to interact, form and publish content in real time.

This thesis is an attempt to present how web applications that allow users to share, publish and be informed in real time about content and actions of other users of the community, can be built.

For this purpose a web application was developed (TR8) , that allows the users to create and maintain a personal profile, to be related to other users with a relational model similar to the of Twitter social network, to rate websites, images and any kind of web re3source accessible by a URL address, and to receive real-time notifications about the activity of the other users.

The application was implemented using the principles of the MVC architecture and other modern programming techniques.

Περιεχόμενα

1 Εισαγωγή

Το 1989 ένας φυσικός του Cern, έσπειρε το σπόρο για μία από τις μεγαλύτερες επαναστάσεις του 20ού αιώνα. Αν και το Internet (Leiner, B. et al., 2009), ένα δίκτυο δικτύων υπολογιστών, το οποίο χρησιμοποιείτο από το 1972, επέτρεπε την ανταλλαγή αρχείων και μηνυμάτων μεταξύ υπολογιστών που ήταν συνδεδεμένοι σε αυτό, ο Tim Berners-Lee στην προσπάθειά του να βρεθεί ένας καλύτερος τρόπος διαμοιρασμού πληροφορίας μεταξύ των επιστημόνων του ινστιτούτου, επινόησε και εφάρμοσε την ιδέα της αποθήκευσης πληροφορίας σε μία κεντρική τράπεζα δεδομένων. Αυτή ήταν προσβάσιμη από οποιοδήποτε τερματικό ήταν συνδεδεμένο στο δίκτυο. Από τότε, η εξέλιξη του παγκόσμιου ιστού www (Leiner, B. et al., 2009) ήταν ραγδαία. Ειδικά από την αλλαγή της χιλιετίας και μέχρι σήμερα, ο παγκόσμιος ιστός έχει αλλάξει ριζικά σχεδόν όλα τα πεδία που διαμορφώνουν τις κοινωνίες των λαών: εκπαίδευση, ενημέρωση, αγορά προϊόντων, μουσική, ιατρική, χαρτογραφία είναι απλώς κάποια από αυτά.

Ένα από τα πιο ραγδαίως αναπτυσσόμενα, και με μεγάλη επίδραση στις κοινωνίες, προϊόντα του παγκόσμιου ιστού αποτελούν τα κοινωνικά δίκτυα (Boyd D., Ellison N., 2007), τα οποία έθεσαν το πεδίο για πρωτοφανείς συλλογικές κινητοποιήσεις, κινήματα διαμαρτυριών, συνεργασίας και ανάπτυξης. Η Αιγυπτιακή Άνοιξη (Howard P. et al., 2011) αποτελεί ένα χαρακτηριστικό παράδειγμα για τον τρόπο με τον οποίο τα κοινωνικά δίκτυα αποτέλεσαν εργαλεία που έπαιξαν καθοριστικό ρόλο στην οργάνωση και την διάδοση πληροφορίας σχετικά με τις διαδηλώσεις και άλλες δράσεις που έλαβαν τόπο στην χώρα την περίοδο εκείνη, και ταυτόχρονα έστρεψαν την προσοχή της παγκόσμιας κοινότητας στα γεγονότα που εξελίσσονταν. Αλλά και στην προγραμματιστική κοινότητα, πλατφόρμες με χαρακτηριστικά κοινωνικών δικτύων όπως το GitHub (Kosner W. A., 2012), έδωσαν τεράστια ώθηση στην ανάπτυξη και διαμοιρασμό ανοιχτού κώδικα. Περισσότεροι από 10 εκατομμύρια εγγεγραμμένοι προγραμματιστές συμμετέχουν ανεβάζοντας κώδικα, ακολουθώντας άλλους χρήστες-προγραμματιστές ή παρακολουθώντας, βελτιώνοντας και εξελίσσοντας έργα άλλων χρηστών.

Η αλληλεπίδραση λοιπόν των χρηστών του διαδικτύου καθώς και η διαμόρφωση και η διάδοση περιεχομένου σε πραγματικό χρόνο οφείλεται στην εξέλιξη και την αξιοποίηση σύγχρονων τεχνολογιών του διαδικτύου που οδήγησαν στην δημιουργία ιστότοπων πραγματικού χρόνου. Οι ιστότοποι πραγματικού χρόνου (Wikipedia 2015a) ανήκουν στους δυναμικούς ιστότοπους των οποίων το περιεχόμενο, σε αντίθεση με τους στατικούς, μπορεί να ανανεωθεί είτε λόγω κάποιας ενέργειας του χρήστη είτε λόγω άλλων αυτοματοποιημένων διαδικασιών που ενεργοποιούνται βάσει κάποιων γεγονότων (triggers). Το σύστημα ειδοποιήσεων του Facebook (notification system) όπου μία ενέργεια ενός χρήστη, όπως μία δημοσίευση (post) στο χρονολόγιό του , προκαλεί την ανανέωση του περιεχομένου στις ανοιχτές σελίδες άλλων χρηστών που σχετίζονται με τον πρώτο, αποτελεί ένα χαρακτηριστικό παράδειγμα αυτοματοποιημένης πυροδότησης αλλαγής περιεχομένου.

Η παρούσα εργασία αφορά τον σχεδιασμό και την υλοποίηση μίας εφαρμογής που ενσωματώνει τα βασικά χαρακτηριστικά ενός κοινωνικού δικτύου. Το μοντέλο συσχέτισης μεταξύ των χρηστών επιλέχθηκε να είναι μονόδρομο (one way follow design). Με βάση το μοντέλο αυτό, και σε αντίθεση με το αμφίδρομο μοντέλο 'φιλίας' (two way friending design) όπου οι χρήστες μπορεί απλά να είναι φίλοι ή όχι, ένας χρήστης έχει χρήστες που ακολουθεί, χρήστες που τον ακολουθούν, χρήστες που αλληλοακολουθούνται και χρήστες ανάμεσα στους οποίους δεν υπάρχει κανενός είδους σχέση. Η κεντρική ιδέα της εφαρμογής, είναι να δώσει τη δυνατότητα στο χρήστη να βαθμολογεί και να σχολιάζει έναν οποιοδήποτε πόρο του διαδικτύου ο οποίος είναι προσβάσιμος μέσω μίας διεύθυνσης URL (Uniform Resource Locator), (Berners-Lee, T. et al., 2005). Καθώς ο άνθρωπος από τη φύση του έχει την τάση να κατηγοριοποιεί με βάση έναν βαθμό αρεσκείας, έστω και ασυνείδητα και σε αφηρημένο επίπεδο, σχεδόν όλα τα πράγματα που βλέπει, διαβάζει ή ακούει, μία πλατφόρμα η οποία θα δίνει την δυνατότητα στον χρήστη του διαδικτύου να βαθμολογεί και να μοιράζεται οποιοδήποτε είδος πληροφορίας συναντά, θεωρήθηκε ως ενδιαφέρουσα πρόκληση. Η σελίδα χρονοδιαγράμματος

(timeline) του κάθε χρήστη περιέχει τις δημοσιεύσεις εκείνου και των χρηστών που ακολουθεί, παρόμοια με τα δημοφιλή κοινωνικά δίκτυα όπως το Twitter. Ο χρήστης μπορεί να εισαγάγει, επεξεργαστεί και διαγράψει βαθμολογίες, να δει στατιστικά στοιχεία όπως λίστα βαθμολογιών με τους δημοφιλέστερους πόρους (που έχουν βαθμολογηθεί περισσότερες φορές) ή λίστα με τους πόρους που έχουν βαθμολογηθεί με το υψηλότερο σκορ. Επιπλέον η εφαρμογή δίνει τη δυνατότητα στο χρήστη να αναζητά δημοσιεύσεις-βαθμολογήσεις βάσει κειμένου και κριτιρίων, και να διατηρεί ένα προφίλ στο οποίο μπορεί να επεξεργαστεί τις προσωπικές του πληροφορίες και να εισάγει μία φωτογραφία προφίλ. Τέλος, η εφαρμογή εμφανίζει στον χρήστη ειδοποιήσεις (notifications) σε πραγματικό χρόνο ως αποτέλεσμα ενεργειών που λαμβάνουν χώρα από άλλους χρήστες.

Η εφαρμογή που παρουσιάζεται σε αυτήν την εργασία, είναι αποτέλεσμα συνεργασίας τριών φοιτητών. Η ιδιαιτερότητα αυτή οδήγησε στην ανάγκη να γίνει χρήση συστήματος ελέγχου εκδόσεων αρχείων (Otte, S.), σε συνδυασμό με πλατφόρμες διαχείρισης κώδικα και οργάνωσης εργασιών. Οι διπλωματικές εργασίες που σχετίζονται με την εφαρμογή αυτή είναι η παρούσα και οι εργασίες με τίτλο “Εφαρμογή αξιολόγησης διαδικτυακών πόρων με λειτουργικότητα ειδοποιήσεων σε πραγματικό χρόνο” και “Εφαρμογή αξιολόγησης διαδικτυακών πόρων με λειτουργικότητα αποθήκευσης αρχείων στο νέφος”.

Στο κεφάλαιο αυτό έγινε σύντομη αναφορά στα κοινωνικά δίκτυα ως ένα από τα σημαντικότερα προϊόντα των τεχνολογιών του διαδικτύου και παρουσιάστηκε η ιδέα και τα βασικά χαρακτηριστικά της εφαρμογής που υλοποιήθηκε στα πλαίσια της παρούσας εργασίας.

Στο δεύτερο κεφάλαιο γίνεται μία συνοπτική παρουσίαση της βιλιογραφίας η οποία οδήγησε ή συνέβαλε στην σύλληψη της ιδέας, των χαρακτηριστικών, και στην υλοποίηση της εφαρμογής.

Στο τρίτο κεφάλαιο παρουσιάζονται οι τεχνολογίες οι οποίες χρησιμοποιήθηκαν για την υλοποίηση της εφαρμογής της παρούσας διπλωματικής εργασίας. Αναφορικά, αυτές είναι το σύστημα ελέγχου εκδόσεων αρχείων Git, η διαδικτυακή πλατφόρμα φιλοξενίας έργων Bitbucket, το πλαίσιο εργασίας php Laravel, η μέθοδος Ajax, η μη σχεσιακή βάση δεδομένων Redis, και το πλαίσιο εργασίας διεπαφών Bootstrap.

Στο τέταρτο κεφάλαιο παρουσιάζεται η αρχιτεκτονική του συστήματος και η διαδικασία σχεδίασης και ανάπτυξης της εφαρμογής με χρήση της γλώσσας μοντελοποίησης UML και τη μέθοδο RUP.

Στο πέμπτο κεφάλαιο αναφέρονται μελλοντικές επεκτάσεις και βελτιώσεις που επιδέχεται η εφαρμογή καθώς και τα τελικά συμπεράσματα που βγήκαν από την εκπόνηση της παρούσας μεταπτυχιακής διατριβής.

2 Σχετική Βιβλιογραφία

Στο κεφάλαιο αυτό παρουσιάζονται τα συγγράματα και οι εργασίες που μελετήθηκαν, οι οποίες ασχολούνται με θέματα, τεχνολογίες και αντικείμενα που χρησιμοποιήθηκαν ή έδωσαν ιδέες για την υλοποίηση της παρούσας διπλωματικής εργασίας.

ProGit Chacon, S., straub, B.

Σε αυτό το σύγγραμα παρουσιάζεται λεπτομερώς το σύστημα ελέγχου εκδόσεων αρχείων (Version Control System) Git. Περιγράφεται ο σκοπός και τα οφέλη της χρήσης του συστήματος αυτού, ο τρόπος εγκατάστασης και ενσωμάτωσής του σε ένα έργο, και τα επιμέρους χαρακτηριστικά και λειτουργίες που προσφέρει. Για κάθε χαρακτηριστικό ή λειτουργία, υπάρχει λεπτομερής ανάλυση του τρόπου χρήσης καθώς και του πότε και γιατί προτείνεται η κάθε ενέργεια.

Καθώς η εφαρμογή με την οποία ασχολείται η παρούσα διπλωματική εργασία υλοποιήθηκε από ομάδα τριών φοιτητών, και έπρεπε να αντιμετωπιστούν με κάποιον αποδοτικό τρόπο οι δυσκολίες που θα προέκυπταν από την συγγραφή κώδικα και ενσωμάτωση αυτού στο έργο από τρία διαφορετικά άτομα, το σύγγραμα αυτό βοήθησε στην χρήση του Git με αποτελεσματικό τρόπο στην διαχείριση της ροής εργασιών μεταξύ των μελών της ομάδας.

Codebright Rees, D., 2013

Στο σύγγραμα αυτό γίνεται μία λεπτομερής παρουσίαση του Php πλαισίου εργασίας Laravel (έκδοση 4.0), το οποίο τελικά χρησιμοποιήθηκε στην παρούσα εργασία. Αρχικά περιγράφεται η αρχιτεκτονική του πλαισίου εργασίας, οι απαιτήσεις και ο τρόπος εγατάστασης και παραμετροποίησης. Στην συνέχεια παρουσιάζονται αναλυτικά όλα τα χαρακτηριστικά και οι δυνατότητες που προσφέρει το Laravel, με απλό και κατανοητό τρόπο και πλήθος παραδειγμάτων. Ο αναγνώστης διαβάζει , μεταξύ άλλων, με ποιόν τρόπο το Laravel δέχεται και δρομολογεί τις αιτήσεις, πως μπορεί να αποκριθεί σε αυτές, μαθαίνει για την χρήση των ελεγκτών (Controllers) , Μοντέλων (Models), την δημιουργία προτύπων παρουσιάσεων (View Templating) με χρήση του Blade και πολλά άλλα.

Το σύγγραμα αυτό, όντας παράδειγμα άριστης τεκμηρίωσης, συνέβαλε, μεταξύ άλλων, στην απόφαση να χρησιμοποιηθεί τελικά το πλάισιο εργασίας Laravel και αποτέλεσε βασικό σημείο αναφοράς καθόλη την διάρκεια της υλοποίησης της παρούσας εφαρμογής.

Data access for high scalable solutions: Using SQL, NoSQL, and Polyglot Persistence McMurtry, D. et al.

Το σύγγραμα αυτό καταπιάνεται με το πώς μπορεί να σχεδιαστούν και να υλοποιηθούν εφαρμογές και υπηρεσίες οι οποίες συνδυάζουν και εκμεταλλέυονται σχεσιακές και μη σχεσιακές βάσεις δεδομένων, στο πλάισιο της ανάγκης για γρήγορη απόδοση και χειρισμό μεγάλου όγκου δεδομένων. Περιγράφονται οι διάφοροι τύποι των μη σχεσιακών βάσεων δεδομένων που είναι διαθέσιμοι, τα χαρακτηριστικά τους και διάφορες περιπτώσεις χρήσης στις οποίες μπορούν να χρησιμοποιηθούν. Καταγράφονται οι αρχές σχεδιασμού σχεσιακών βάσεων δεδομένων ώστε να επιτυγχάνονται γρήγορες αναζητήσεις, και πώς σε συνδυασμό με μη σχεσιακές βάσεις δεδομένων μπορεί να υλοποιηθεί μία σύγχρονη επαγγελματική εφαρμογή υψηλών απαιτήσεων σε απόδοση, κλιμάκωση (scalability), και συνοχή (consistency).

Το σύγγραμα αυτό προσέφερε μία σφαιρική αλλά και ταυτόχρονα πιο εξειδικευμένη εικόνα σχετικά με το ποιόν τρόπο θα μπορούσε να χρησιμοποιηθεί μία μη σχεσιακή βάση δεδομένων στην εφαρμογή που τελικά υλοποιήθηκε στο πλάισιο της παρούσας εργασίας, και συνέβαλε

στην τελική απόφαση να χρησιμοποιηθεί η Redis σαν μία αξιόπιστη λύση για τις ανάγκες που θα προέκυπταν από την κατασκευή μιας εφαρμογής με χαρακτηριστικά κοινωνικού δικτύου.

Little Redis Book Seguin, K., 2014

Το σύγγραμα αυτό αποτέλεσε οδηγό για τον σχεδιασμό και την δημιουργία της διάταξης δεδομένων της εφαρμογής της παρούσας εργασίας, καθώς αποτελεί ένα πλήρες εγχειρίδιο χρήσης της μη σχεσιακής βάσης δεδομένων Redis. Οδηγίες εγκατάστασης, παραμετροποίησης, αναλυτική παρουσίαση των τύπων δεδομένων της Redis και προτεινόμενες περιπτώσεις χρήσης, καθώς και πιο προχωρημένα ζητήματα όπως πρακτικές μόνιμης αποθήκευσης, ασφάλειας δεδομένων, επεκτασιμότητα και άλλα παρουσιάζονται σε αυτό το σύγγραμα.

Design and implementation of a simple Twitter clone using only the Redis key-value store as database and PHP Sanfilippo, S., 2011

Σε αυτήν την εργασία παρουσιάζεται ο σχεδιασμός και η υλοποίηση μίας διαδικτυακής εφαρμογής η οποία αποτελεί έναν κλώνο του Twitter, με χρήση της γλώσσας προγραμματισμού Php και της μη σχεσιακής βάσης δεδομένων Redis. Αρχικά γίνεται μία αναφορά στις βάσεις δεδομένων που λειτουργούν στην λογική κλειδιού τιμής, και παρουσιάζονται οι διάφοροι τύποι δεδομένων που παρέχει η Redis. Στην συνέχεια παρουσιάζεται η διάταξη δεδομένων που χρησιμοποιήθηκε για την αναπαράσταση και την αποθήκευση των αντικειμένων όπως οι χρήστες ή οι δημοσιεύσεις τους, καθώς και οι τύποι δεδομένων και οι τεχνικές χρήσης για λειτουργίες όπως πιστοποίηση χρηστών και το μοντέλο συσχέτισης των χρηστών μεταξύ τους (follower model). Τέλος, παρουσιάζονται κάποιες μετρικές, οι οποίες δείχνουν την πολύ γρήγορη απόδοση που μπορεί να έχει η εφαρμογή τρέχοντας ακόμα και σε έναν αργό εξυπηρετητή, και η ευκολία με την οποία μπορεί να επιτευχθεί κλιμάκωση της βάσης δεδομένων σε περισσότερους κόμβους.

Η εργασία αυτή αποτέλεσε την βάση πάνω στην οποία σχεδιάστηκαν βασικά τμήματα της διάταξης δεδομένων και υλοποιήθηκαν βασικές λειτουργίες αυτής.

Συνοψη

Στο κεφάλαιο αυτό παρουσιάστηκε η βιβλιογραφία σε επίπεδο συγγραμάτων αλλά και εργασιών που μελετήθηκαν για τις ανάγκες της παρούσας διπλωματικής εργασίας, καθώς το αντικείμενό τους ήταν παρόμοιο ή αφορούσε θέματα με τα οποία η παρούσα εργασία καταπιάνεται. Επισημάνθηκαν τα βασικά στοιχεία τους και οι λόγοι για τους οποίους αποτέλεσαν έμπνευση ή συνέβαλαν με πρακτικό τρόπο σαν εγχειρίδια για την εφαρμογή που υλοποιήθηκε. Στο επόμενο κεφάλαιο παρουσιάζονται οι τεχνολογίες που χρησιμοποιήθηκαν, εμβαθύνοντας σε πιό τεχνικές λεπτομέρειες για κάθε μία από αυτές.

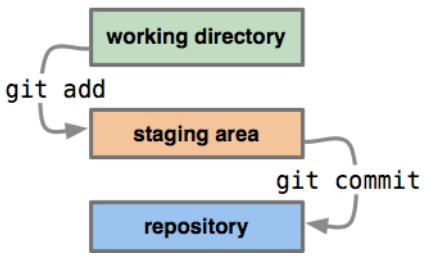
3 Τεχνολογίες

Στο κεφάλαιο αυτό παρουσιάζονται οι τεχνολογίες που επιλέχθηκαν να χρησιμοποιηθούν για την υλοποίηση της εφαρμογής. Αυτές είναι το σύστημα ελέγχου διανεμόμενης έκδοσης και διαχείρισης πηγαίου κώδικα Git (Loeliger, J., McCullough, M., 2012), η online υπηρεσία φιλοξενίας έργων που χρησιμοποιούν Git, Bitbucket, το πλαίσιο εργασίας Php Laravel (Rees, D.), το μη σχεσιακό σύστημα αποθήκευσης δεδομένων Redis (Seguin K., 2012), η μέθοδος Ajax και το Bootstrap, ένα πλαίσιο εργασίας για την ανάπτυξη της διεπαφής της εφαρμογής με το χρήστη.

3.1 Git

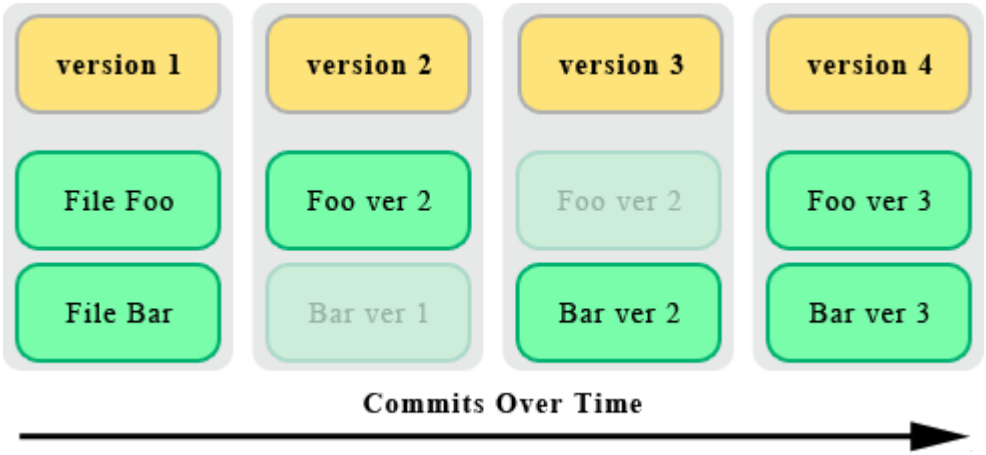
Το Git είναι το πιο διαδεδομένο σύστημα ελέγχου εκδόσεων αρχείων ενός έργου (version control system). Καταγράφει τις αλλαγές ενός αρχείου ή ενός συνόλου αρχείων μέσα στον χρόνο (snapshots), έτσι ώστε να είναι δυνατή η ανάκληση των αλλαγών αυτών σε μεταγενέστερο χρόνο αν αυτό χρεαστεί. Για να χρησιμοποιηθεί σε ένα έργο, πρέπει να γίνει αρχικοποίηση του Git σε έναν χώρο αποθήκευσης (repository), στον οποίον υπάρχουν και τα αρχεία του έργου. Ο χώρος αποθήκευσης αυτός, μπορεί να βρίσκεται σε έναν απομακρυσμένο εξυπηρετητή (server) ή σε έναν τοπικό υπολογιστή. Η συνήθης πρακτική είναι η δημιουργία ενός απομακρυσμένου αποθετηρίου (remote repository) και η κλωνοποίηση αυτού τοπικά στον υπολογιστή των συμμετέχοντων στην εγγραφή του έργου. Το Git αποτελεί δηλαδή, ένα αποκεντρωμένο σύστημα ελέγχου έκδοσης (DVCS: Decentralized Version Control System).

Όταν ο χρήστης αλλάζει στο τοπικό αποθετήριο ένα αρχείο, αυτό μπαίνει σε κατάσταση 'τροποποιημένο' (modified). Πριν αποθηκεύσει το Git την αλλαγή, το αρχείο πρέπει να περάσει στην κατάσταση 'έτοιμο προς αποθήκευση' (staged), και από εκεί στην κατάσταση 'αποθηκευμένο' (commited).



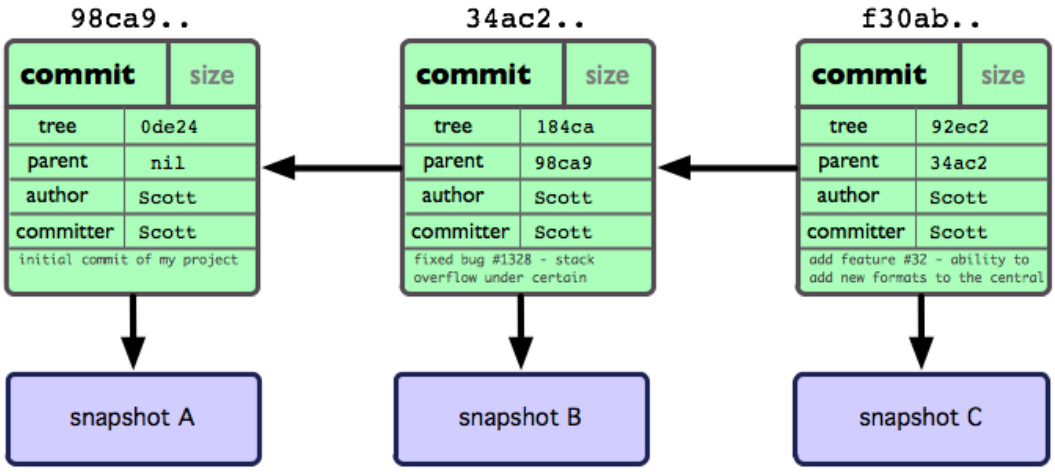
Η διαδικασία αυτή των αλλαγών (commit) μπορεί φυσικά να αφορά και ένα σύνολο αρχείων. Η συνήθης πρακτική είναι σε κάθε commit να περιέχονται αρχεία που σχετίζονται μεταξύ τους, με

την έννοια ότι αφορούν μια συγκεκριμένη λειτουργικότητα, για παράδειγμα, του έργου. Εφόσον το έργο διατηρείται και σε απομακρυσμένο αποθετήριο, τότε ο χρήστης πρέπει να ενημερώνει το αποθετήριο αυτό για τα commit που έκανε. Η ενέργεια αυτή ονομάζεται 'σπρώξιμο' (push). Αντίστροφα, και εφόσον στην ανάπτυξη του έργου συμμετέχουν περισσότερα από ένα άτομα, ο καθένας οφείλει να ενημερώνει τακτικά το τοπικό του αποθετήριο, με τα commits που έχουν σπρώξει οι υπόλοιποι χρήστες. Η ενέργεια αυτή ονομάζεται 'τράβηγμα' (pull).



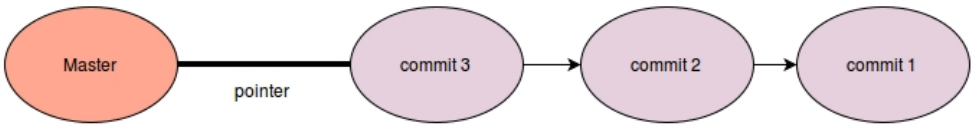
Στιγμιότυπα εκδόσεων αρχείων στον χρόνο.

Κάθε commit είναι λοιπόν ένα στιγμιότυπο των αρχείων που βρίσκονται υπό version control, χαρακτηρίζεται μοναδικά από τον αλγόριθμο SHA-1, και περιέχει πληροφορίες όπως τη χρονοσφραγίδα της καταγραφής, το όνομα (email) του χρήστη, και ένα σχόλιο σχετικό με το λόγο ή τις αλλαγές που έγιναν και αποθηκεύτηκαν. Το Git αποθηκεύει τις πληροφορίες αυτές σε ένα αντικείμενο (object) το οποίο περιέχει ένα δείκτη για το στιγμιότυπο που αντιπροσωπεύει. Η αλληλουχία των commits δεν είναι τίποτα άλλο παρά μία μονή συνδεδεμένη λίστα, όπου κάθε commit ξέρει μόνο το γονέα του (parent).



Αντικείμενα commit

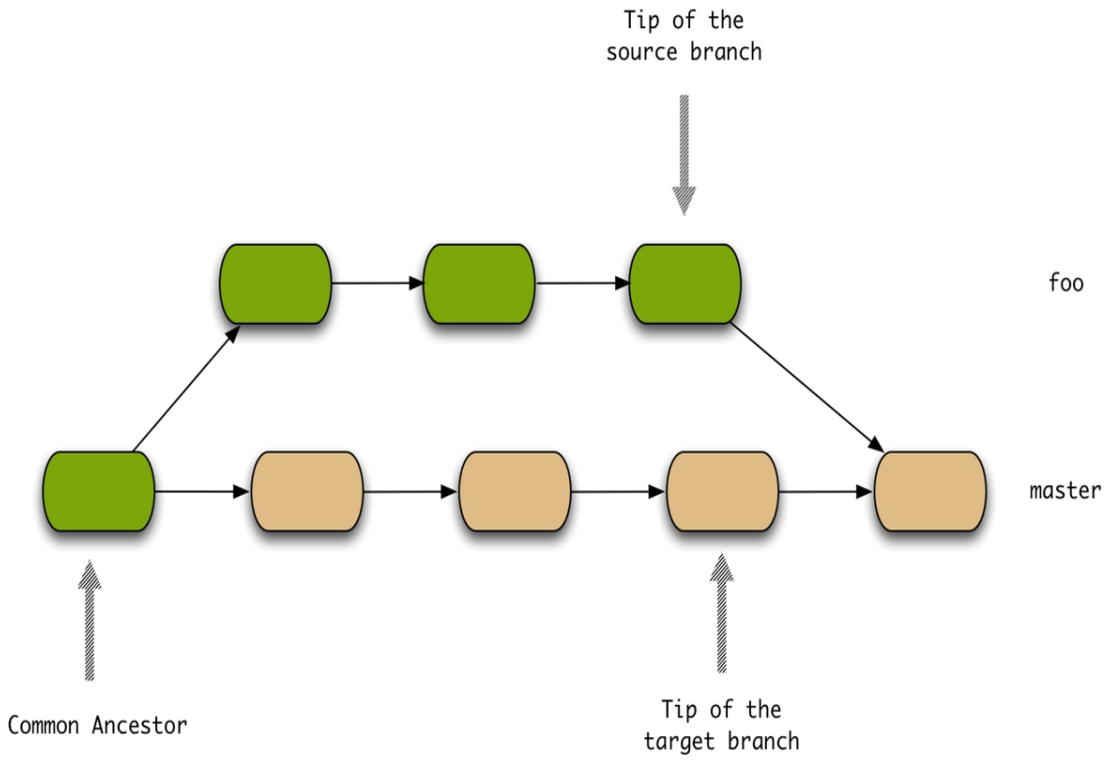
Ενα άλλο σημαντικό χαρακτηριστικό του Git είναι η ισχυρή υποστήριξη που παρέχει για μη γραμμική ανάπτυξη ενός έργου. Αυτό επιτυγχάνεται μέσω των κλαδιών (branches). Κατά την αρχικοποίηση του Git, δημιουργείται ένα κλαδί το οποίο ονομάζεται 'Master'. Ο δείκτης του κλαδιού αυτού δείχνει πάντα στο πιο προφατο commit.



Ο δείκτης του κλαδιού master δείχνει στο πιο πρόσφατο commit.

Θεωρώντας το αποθετήριο σαν ένα γράφημα από commits, όπου κάθε κόμβος δείχνει στο γονικό του, ένα νέο κλαδί δεν είναι παρά ένας νέος κόμβος ο οποίος δείχνει στο ίδιο (πιο πρόσφατο) γονικό κόμβο.

Επιπλέον, υπάρχει ένας ειδικός δείκτης που ονομάζεται 'Head', και δείχνει σε ποιό κλαδί βρισκόμαστε. Όταν δημιουργείται ένα νέο κλαδί, δημιουργείται ένας νέος δείκτης που δείχνει στο ίδιο commit με το master. Αντίστροφα, όταν συγχωνεύονται δύο κλαδιά (branch merging), ένα commit έχει 2 γονείς.



Δημιουργία κλαδιού και συγχώνευση με το master

Κατά την συγχώνευση, αν σε ένα αρχείο έγιναν διαφορετικές αλλαγές (για παράδειγμα από διαφορετικούς χρήστες οι οποίοι δούλευαν στα κλαδιά που συγχωνεύτηκαν), μπορεί να προκληθεί μία 'διένεξη' (conflict). Εφόσον το Git δεν καταφέρει να επιλύσει αυτόματα την διένεξη, αυτό πρέπει να γίνει από τους υποβολείς του commit.

Τα πλεονεκτήματα της χρήσης του Git κατά την ανάπτυξη ενός έργου είναι σημαντικά:

Η χρήση των κλαδιών βοηθά στην αποδοτικότερη ανάπτυξη, π.χ. νέων λειτουργιών του έργου. Η συνήθης πρακτική που ακολουθείται είναι η ύπαρξη ενός 'κεντρικού' κλαδιού με την σταθερή (stable) έκδοση της εφαρμογής. Η δυνατότητα δημιουργίας νέων κλαδιών δίνει την δυνατότητα στον προγραμματιστή να αναπτύξει μία νέα λειτουργικότητα της εφαρμογής πάνω σε αυτό. Εφόσον αποφασιστεί η νέα αυτή λειτουργικότητα να συμπεριληφθεί στο έργο, η συγχώνευση του κλαδιού στο οποίο αναπτύχθηκε με το κεντρικό κλαδί θα οδηγήσει στην ενσωμάτωσή της. Έτσι, το Git ενθαρρύνει την δημιουργικότητα και τον πειραματισμό, κάτι που ενισχύεται ακόμα περισσότερο από την δυνατότητα 'πισογυρίσματος' (undo) σε κάθε προηγούμενη κατάσταση, όπως αυτές ορίζονται από τα commits. Επιπλέον, διατηρώντας ένα απομακρυσμένο αποθετήριο και ταυτόχρονα τοπικά αντίγραφα στον υπολογιστή, η πιθανότητα απώλειας δεδομένων ελαχιστοποιείται, εφόσον ο χρήστης φροντίζει για τη συχνή ενημέρωση (μέσω των

ενεργειών pull-push) και των δύο.

3.2 Bitbucket

Το Bitbucket είναι μια διαδικτυακή πλατφόρμα η οποία μπορεί να φιλοξενεί έργα που χρησιμοποιούν Git. Επιλέχθηκε να χρησιμοποιηθεί στην παρούσα εργασία (φιλοξενώντας το απομακρυσμένο αποθετήριο) , καθώς παρέχει μια οπτικοποίηση των commits και γενικά των Git ενεργειών που έχουν λάβει τόπο κατά τη δημιουργία του έργου. Ο χρήστης μπορεί εύκολα να δημιουργήσει και να συγχωνεύσει κλαδιά, να προβάλει όλο το ιστορικό των commits, να δει τα αρχεία του έργου και να συγκρίνει τις αλλαγές σε κάθε αρχείο μεταξύ δύο commits. Επιπλέον, το Bitbucket παρέχει στους χρήστες τη δυνατότητα να δημιουργήσουν ζητήματα (issues). Ένα issue μπορεί να δημιουργηθεί για να περιγράψει μία λειτουργικότητα της εφαρμογής που πρέπει να υλοποιηθεί, καταγράφοντας σε αυτό τις προδιαγραφές αυτής. Ένα issue μπορεί επίσης να δημιουργηθεί για να περιγράψει κάποιο bug που πρέπει να διορθωθεί. Κάθε issue μπορεί να έχει διάφορες καταστάσεις, όπως 'ανοιχτό', 'σε αναμονή', 'κλειστό', κ.α. Κάθε commit, μπορεί να αναφέρεται σε κάποιο συγκεκριμένο issue, συμπεριλαμβάνοντας στο σχόλιό του τον κωδικό του ζητήματος.

Ειδικά για μεγάλα projects, όπου συμμετέχουν πολλά άτομα, τα issues βοηθούν στην καλύτερη οργάνωση της ομάδας (αναθέτοντας issues σε διαφορετικούς χρήστες), και στην συνολική καταγραφή και επισκόπιση της προόδου του έργου.

Ένα ακόμα σημαντικό χαρακτηριστικό του Bitbucket, τέλος, είναι η δυνατότητα δημιουργίας εγχειριδίων (wiki), όπου εκεί μπορούν να καταγράφονται η περιγραφή και οι λεπτομέρειες που αφορούν λειτουργικότητες της εφαρμογής που υλοποιήθηκαν.

Commits

All branches

Author	Commit	Message
Nikos Kosmop…	0b1e808	Favicon @ markup-css changes
Nikos Kosmop…	d791c7b	Some css & replaced 'invalid url' alert (when adding a new rate) with custom growl error notification
Nikos Kosmop…	8763801	Added a 'please wait' loader (displayed when adding a new rate as it takes sometime to complete)
Nikos Kosmop…	d6c45be	Fixed bug on user rates page
Nikos Kosmop…	9c24c1c	Final changes
Nikos Kosmop…	cf5a88d	Fixed bug: Removed 'rate it' btn functionality for non-verified users. (when they see rates of others users)
Nikos Kosmop…	c8c96a4	Final markup schanges
Spiros Stavrop…	51a938a M	Merged in aws_s3 (pull request #26) Update develop after refactor of aws s3
Spiros Stavrop…	768f09a	#18 refactor AWS S3 functionality
Spiros Stavrop…	1601d2a	code optimization baseController
Spiros Stavrop…	a847e76 M	Merged in develop (pull request #25) update aws_s3
Harris Bouzop…	4b1d245	Vendor files with conflicts commented
Harris Bouzop…	7b44108 M	Merged in comet (pull request #24) comet - > develop Notifications implementation, other fixes
Harris Bouzop…	7a961e4 M	Merged in develop (pull request #23) Conflicts: app/controllers/BaseController.php app/views/home.blade.php app/views/layouts/signedinBase.blade.php ven…
Harris Bouzop…	3e24636	Code commenting

Εμφάνιση του ιστορικού των commits και συγχωνεύσεων κλαδιών στο Bitbucket

Issues (1–17 of 17)

Title	T	P	Status	Votes	Assignee	Created	Updated
#18: Amazo Web Services S3			NEW		Spiros Stavrop…	2014-11-12	2014-11-12
#17: Edit User Profile			NEW		Spiros Stavrop…	2014-11-12	2014-11-12
#16: Coming soon..			NEW		Spiros Stavrop…	2014-10-06	2014-10-06
#15: Account verification			NEW		Spiros Stavrop…	2014-09-22	2014-09-22
#5: Search rate functionality			NEW		Nikos Kosmop…	2014-08-08	2014-09-21
#14: Refactor form submission over AJAX & implement form validation			NEW		Harris Bouzop…	2014-09-16	2014-09-16

Λίστα ζητημάτων στο BitBucket

Issues

Issue #5 NEW

Search rate functionality

Nikos Kosmopoulos created an issue 2014-08-08
We need a search form to search for ratings, and a page to show the results

Fields the user submits

- text - keywords (basic default search in ratings' titles)
- Where to search, i.e. own, followings' (or both) ratings
- score (advanced search)
- date (TBD if should be added in advanced search)

Fields the system is providing

- user_id

User acceptance criteria

- User_id should not be submitted with the form (should be provided by the system)
- We need **indexes** for fast search execution and results retrieval. **Ideally, searching and filtering operations should take place inside Redis using nothing more than sets and sunion/sinter commands**.
- Should be able to search ratings for a specific score or date (e.g. score:8 or/and date:01-01-2014), and for a specific range of scores or dates (e.g. score between 7 and 10).
- Text-based search should be able to handle different languages (e.g. greek) and show tolerance to reasonable spelling mistakes e.t.c.
- To be discussed: Submit form over Ajax

Ένα ζήτημα στο Bitbucket

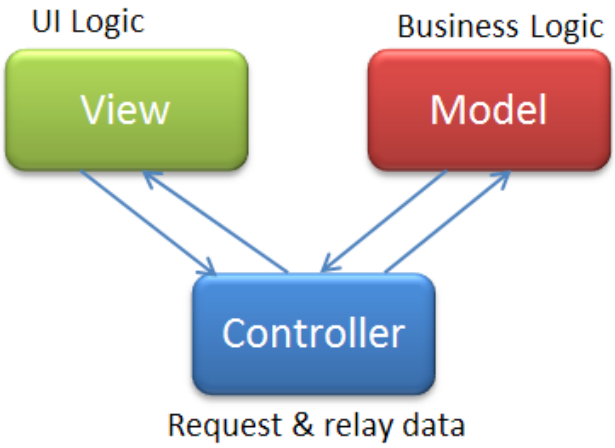
3.3 Το πλαίσιο εργασίας Laravel

Το Laravel είναι ένα δωρεάν, ανοιχτού κώδικα (open source) πλαίσιο εργασίας Php, το οποίο χρησιμοποιείται για την κατασκευή διαδικτυακών εφαρμογών. Επιλέχθηκε για την υλοποίηση της εργασίας αυτής αντί του CodeIgniter (EllisLab), που είχε επιλεγεί αρχικά, καθώς είναι ένα μοντέρνο, με ραγδαίως αυξανόμενη αποδοχή από την προγραμματιστική κοινότητα, πλαίσιο εργασίας. Βασίζεται στα τρέχοντα πρότυπα της βιομηχανίας και δεν απαιτούνται περίπλοκοι κώδικες για να εκτελεστούν λειτουργίες που κανονικά είναι (ή θα έπρεπε να είναι) απλές. Επιπλέον η τεκμηρίωση του Laravel είναι πλήρης και πάντα ανανεωμένη, και υπάρχει μια μεγάλη, ολοένα αυξανόμενη κοινότητα υποστήριξης.

Το Laravel ακολουθεί την αρχιτεκτονική Μοντέλο-Παρουσίαση-Ελεγκτή (MVC, Model-View-Controller). Σύμφωνα με το μοντέλο αυτό, η εφαρμογή διαιρείται σε τρία επίπεδα: Ο ελεγκτής (controller) ελέγχει το πώς 'τρέχει' η εφαρμογή. Κατά κανόνα, στην γενική περίπτωση, δέχεται τις αιτήσεις (requests), μιλάει με το μοντέλο (model), παίρνει δεδομένα από αυτό και αφού τα επεξεργαστεί τα στέλνει στην παρουσίαση (view), την οποία τελικά και βλέπει ο χρήστης. Μπορεί επίσης να δίνει οδηγίες στο μοντέλο για να ενημερώσει την κατάστασή του.

Το μοντέλο είναι το επίπεδο που σχετίζεται με τη βάση δεδομένων. Εκεί κατοικούν λειτουργίες διαχείρισης των δεδομένων που λαμβάνονται από τη βάση.

Η παρουσίαση αποτελεί το επίπεδο που αλληλεπιδρά με το χρήστη. Είναι στην ουσία οι HTML σελίδες που βλέπει ο χρήστης. Τις περισσότερες φορές μία παρουσίαση μιλάει με έναν ελεγκτή, ο οποίος αφου επεξεργαστεί τα δεδομένα τα στέλνει σε αυτή για να τα εμφανίσει.



Τα 3 επίπεδα της MVC αρχιτεκτονικής

Η επιλογή ενός πλαισίου εργασίας MVC αρχιτεκτονικής ήταν εξαρχής αποφασισμένη, καθώς προσφέρει διαχωρισμό προβλημάτων (Seperation of concerns), επεκτασιμότητα και ελεγξιμότητα. Μία σωστή MVC εφαρμογή είναι εκείνη που τα τρία επίπεδα είναι ξεκάθαρα καθορισμένα και δεν συμπλέκονται. Μελλοντικά μπορούν να προστεθούν νέες λειτουργίες, ή να αλλάξουν κάποιες από τις υπάρχουσες, χωρίς περίπλοκες επεμβάσεις. Τέλος, η αρχιτεκτονική αυτή προσφέρει τη δυνατότητα στην εφαρμογή να είναι ελέγξιμη, γράφοντας κώδικα-τεστς, και με τον τρόπο αυτό να συντηρείται πιο εύκολα.

Χαρακτηριστικά του Laravel

Το Laravel προσφέρει αρκετά χρήσιμα χαρακτηριστικά (features). Υποστηρίζει την χρήση συμβατών εξωτερικών πακέτων (Bundles) και βιβλιοθηκών (Libraries) με εύκολη εγκατάσταση. Ενσωματώνει το Eloquent ORM, ένα σύστημα χαρτογράφησης σχεσιακών δεδομένων με αντικείμενα, το οποίο αναπαριστά τους πίνακες της βάσης δεδομένων με κλάσεις, ένα στιγμιότυπο της οποίας (instance) είναι δεμένο με μία γραμμή-εγγραφή του πίνακα. Επιπλέον, το Laravel ενσωματώνει την βιβλιοθήκη Predis, ώστε να διευκολύνει την επικοινωνία με την μη σχεσιακή βάση δεδομένων Redis, η οποία χρησιμοποιήθηκε στην παρούσα εργασία και στην οποία θα γίνει αναφορά στην συνέχεια.

Η δημιουργία διεπαφών με την εφαρμογή (API) διευκολύνεται με τους ελεγκτές REST αρχιτεκτονικής που παρέχει το Laravel, ενώ η ελεγξιμότητα της εφαρμογής μπορεί να επιτευχθεί με χρήση της βιβλιοθήκης Php UnitTest.

Ένα ακόμα πολύ χρήσιμο χαρακτηριστικό του Laravel είναι το Blade, μία μηχανή συγγραφής προτύπων παρουσιάσεων (View Templating). Παρέχει τη δυνατότητα πιο κομψής και εύκολης διαχείρισης δεδεομένων που φτάνουν στην παρουσίαση από τους ελεγκτές, προσφέροντας πιο καθαρό κώδικα σε αντιστοιχία με την κλασική σύνταξη php εντολών σε HTML αρχεία, ενώ παράλληλα προσφέρει ένα δικό του σετ από δομές ελέγχου (control structure), όπως εντολές συνθηκών και βρόχους (conditional statements, loops). Επιπλέον, το Blade δίνει τη δυνατότητα δημιουργίας προτύπων παρουσιάσεων (templates) με πολύ αποδοτικό τρόπο, συνδυάζοντας πολλά αρχεία παρουσιάσεων.

Ανάμεσα στα υπόλοιπα χαρακτηριστικά του Laravel, τέλος, αξίζει να αναφερθεί το σύστημα ασφάλειας και πιστοποίησης (authentication) που παρέχει.

Αρχικά, σαν πλαίσιο εργασίας για την υλοποίηση της εφαρμογής είχε επιλεγεί το CodeIgniter, καθώς είναι ένα από τα πολύ διαδεδομένα πακέτα και χρησιμοποιείται ευρέως από την προγραμματιστική κοινότητα, έχει πολύ καλή τεκμηρίωση και μεγάλη κοινότητα υποστήριξης. Το Laravel σαν εναλλακτική επιλογή προέκυψε από την ανάγκη χρήσης διάφορων εξωτερικών βιβλιοθηκών μαζί με το CodeIgniter για να υποστηριχθούν διάφορες λειτουργίες και χαρακτηριστικά της εφαρμογής που υλοποιήθηκε. Για παράδειγμα, το CodeIgniter δεν διαθέτει

κάποιο σύστημα συγγραφής προτύπων για τις παρουσιάσεις, δεν προσφέρει μηχανή ασφάλειας και πιστοποίησης χρηστών, ούτε κάποιο σύστημα διαχείρισης σχεσιακής βάσης δεδομένων ή την βιβλιοθήκη υποστήριξης Redis σε php, την Predis. Αυτά και πολλά επιπλέον, περιέχονται στο Laravel. Στην εικόνα που ακολουθεί, παρουσιάζεται ένας συγκριτικός πίνακας μεταξύ των δύο πλαισίων εργασίας.

Module	**Laravel**	**Codeigniter**
Layout Control	Yes	No
ORM	Yes	No
Error Stack Trace	Yes	No
Class Auto Loading	Yes	No
–Database mySQL	Yes	Yes
–Database SQLite	Yes	Yes
–Database MSSQL	Yes	Yes
–Database PostgreSQL	Yes	Yes
–Database Cubrid	Yes	No
–ODBC drivers	Yes	No
–Database MariaDB	No	No
Authentication Library	Yes	No
External Modules	Yes (Bundles)	No
Form Validation Rules	Yes	Yes
Internationalization	Yes	Yes

Συγκριτικός πίνακας Laravel - CodeIgniter

3.4 Redis

Στην παρούσα εργασία επιλέχθηκε να χρησιμοποιηθεί, αντί για μια σχεσιακή βάση δεδομένων, η Redis. Η Redis είναι ένας εξυπηρετητής δομών δεδομένων ο οποίος ακολουθεί την λογική του κλειδιού-τιμής (key-value) για την αποθήκευση δεδομένων, με το χαρακτηριστικό οτι οι τιμές μπορούν να αποτελούν εκτός από ένα απλό αλφαρηθμητικό (string), πιο σύνθετους τύπους, στους οποίους θα γίνει αναφορά στη συνέχεια.

Η Redis αποθηκεύει στη μνήμη το σετ δεδομένων της εφαρμογής που την χρησιμοποιεί. Αυτό σημαίνει ότι το μέγεθος της μνήμης αποτελεί και το φυσικό όριο του όγκου των δεδομένων που μπορεί να αποθηκεύσει. Επιπλέον, η Redis έχει την δυνατότητα μόνιμης αποθήκευσης στο δίσκο (persistence), αλλά αυτό γίνεται ασύγχρονα, που σημαίνει οτι αν, για παράδειγμα, ο εξυπηρετητής επανεκκινήσει λόγω κάποιου προβλήματος, θα χαθούν δεδομένα που είχαν αποθηκευτεί στην μνήμη μετά τον τελευταίο συγχρονισμό στον δίσκο.

Σχετικά με την μόνιμη αποθήκευση, η Redis προσφέρει δύο επιλογές. Η πρώτη, ονομαζόμενη RDB persistence, επιτρέπει την ανά τακτά χρονικά διαστήματα αποθήκευση στιγμιοτύπων του σετ δεδομένων στον δίσκο, ενώ η δεύτερη, η AOF Persistence, καταχωρεί (log) όλες τις ενέργειες εγγραφής-διαγραφής που δέχεται ο εξυπηρετητής. Σε κάθε επανεκκίνηση του εξυπηρετητή, το ιστορικό των εγγραφών μπορεί να 'ξαναπαιχτεί' επανακτίζοντας το αρχικό σετ δεδομένων. Μία πρακτική που προτείνεται για την ελαχιστοποίηση της πιθανότητας να χαθούν δεδομένα, είναι η χρήση της AOF κάθε δευτερόλεπτο, ωριαία RDB στιγμιότυπα στον δίσκο και συγχρονισμός των παραπάνω με S3, μία υπηρεσία της Amazon για διαδικτυακή αποθήκευση δεδομένων στο σύννεφο (internet & cloud storage), έτσι ώστε να είναι δυνατή η πλήρης επαναφορά της βάσης δεδομένων σε περίπτωση απώλειας.

Ένα επιπλέον χαρακτηριστικό της Redis είναι η λειτουργία Master-Slave Replication, κατά την οποία η βάση δεδομένων μπορεί να υπάρχει σε πολλούς κόμβους-εξυπηρετητές. Με τον τρόπο αυτό είναι δυνατή η κατανομή του φόρτου εργασίας σε μία ομάδα εξυπηρετητών. Η συνήθης πρακτική είναι να υπάρχει ένας κόμβος 'Master' στον οποίο γίνονται όλες οι ενέργειες εγγραφής, και κόμβοι 'Slave', όπου εκτελούνται οι ενέργειες ανάγνωσης.

Στην συνέχεια παρουσιάζονται οι τύποι δεδομένων που προσφέρει η Redis:

Strings. Αποτελούν μία αλληλουχία δυαδικών bytes, μέχρι 512mb και αποτελούν την κλασική key-value αποθήκευση δεδομένων.

```
Key: "name"
Value: "bob"
```

Παράδειγμα String

Hashes. Ένα hash είναι μια συλλογή ζευγών κλειδιού-τιμής. Μπορεί να χρησιμοποιηθεί για την αποθήκευση ενός αντικειμένου (object) ή σαν ένας τρόπος για να αποθηκευτούν συμπαγώς πολλές μικρές τιμές. Η Redis βελτιστοποιεί με τέτοιο τρόπο την πρόσβαση και την αποθήκευση ενός hash, ώστε είναι αποδοτικότερη η αποθήκευση, για παράδειγμα, 100 hashes με 100 εσωτερικά κλειδιά-τιμές το καθένα, παρά 10.000 ζέυγη κλειδιών-τιμών ξεχωριστά σαν strings.

```
Key: "accountInfo:5531"
Value: "name" => "bob smith"
       "username" => "ultimate bob"
       "country" => "UK"
       "lastLogin" => "1383147407"
       "loginCount" => "3691"
       "lastPaymentSync" => "1383256813"
       "signup" => "1381254812"
       "isAdmin" => "false"
```

Παράδειγμα Hash

Lists. Έχουν την λογική της συνδεδεμένης λίστας. Είναι δυνατή η εισαγωγή, διαγραφή στοιχείων και η διάσχιση αυτής από την ουρά ή την κεφαλή της. Οι λίστες είναι μία καλή επιλογή για να διατηρείται η σειρά με την οποία τα στοιχεία της εισάχθηκαν σε αυτή. Επίσης, μία λίστα μπορεί να διατηρεί συγκεκριμένο μήκος (capped collection) εφαρμόζοντας όποτε απαιτείται 'ψαλίδισμα' (trimming) σε αυτή μετά από κάθε εισαγωγή στοιχείου.

```
Key: "processURLsNext"
Value: ["http://google.com/", "http://theonion.com", "http://redis.io"]

Key: "userTimeline:userId:312"
Value: ["swimming away from park", "Whoops, park on fire", "Toasting marshmallows", "At
 park (it's cold!)"]
```

Παράδειγμα Lists

Sets. Τα σετ αποτελούν μια συλλογή από strings. Είναι δυνατή η εισαγωγή, αφαίρεση, και ο έλεγχος της ύπαρξης μελών σε O(1) (σταθερός χρόνος ανεξαρτήτως του πλήθους των στοιχείων στο σετ). Στα σετ δεν είναι δυνατή η ύπαρξη δύο ή παραπάνω ίδιων μελών, κάνοντάς

τα ιδανικά για την αποθήκευση μοναδικών τιμών, π.χ. των διευθύνσεων IP που επισκεύτηκαν ένα συγκεκριμένο ιστολόγιο. Τα σετ υποστηρίζουν τις κλασικές πράξεις των συνόλων, όπως ένωση (union), τομή (intersection), διαφορά (difference), με τη δυνατότητα είτε επιστροφής των αποτελεσμάτων για άμεση χρήση, είτε την αποθήκευση αυτών σε ένα νέο σετ για μεταγενέστερη χρήση.

```
Key: "adminUsers"
Value: ("bob", "ralph", "jose", "loa")

Key: "bannedUids"
Value: ("332", "4096", "271", "737199492")

Key: "forbiddenUsernames"
Value: ("boogie", "bogus", "fake", "cybercommander", "improv", "gandalf", "oberoth")

Key: "uniqueVisitors"
Val: ("127.0.0.1", "63.23.221.7", "4.2.2.2", "8.8.8.8")
```

Παράδειγμα Sets

Sorted Sets. Είναι σύνολα με ορισμένη από τον χρήστη κατάταξη των μελών τους. Η σειρά κατάταξης κάθε μέλους καθορίζεται από ένα σκορ που το συνοδεύει. Εδώ η μοναδικότητα των στοιχείων-μελών ενός sorted set αφορά το ζευγάρι σκορ-τιμή. Μπορεί να γίνει ανάκτηση μελών ενός τέτοιου σετ με βάση το σκορ τους με αύξουσα ή φθίνουσα σειρά.

```
Key: "leaderBoard"
Value: ("smashmaster" [300], "pilot" [600], "oigo" [720])

Key: "lastLoggedInUids"
Value: ("332" [1383256813], "4096" [1383256916], "271" [1383257212], "737199492" [13832
59419])
```

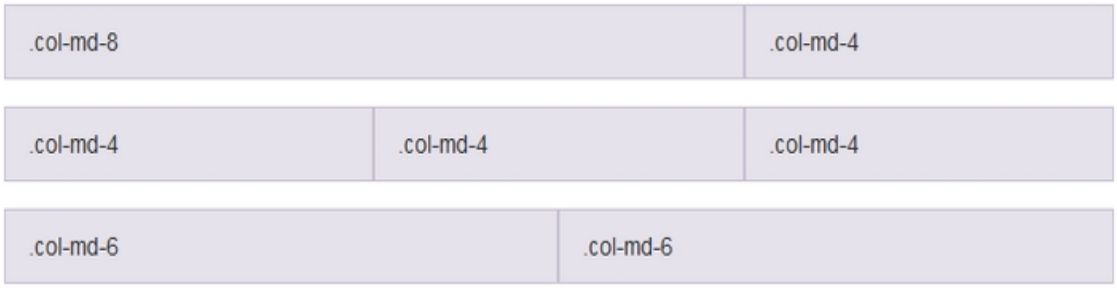
Παράδειγμα Sorted Sets

3.5 AJAX

Το AJAX (Asynchronous Javascript and XML) είναι μία τεχνολογία που χρησιμοποιείται ευρέως σε διαδικτυακές εφαρμογές που αλληλεπιδρούν με τον χρήστη. Με το Ajax, μπορεί να σταλούν δεδομένα από τον client στον εξυπηρετητή, και να ληφθούν πίσω δεδομένα απόκρισης χωρίς να ξαναφορτώσει η σελίδα. Αυτό σημαίνει οτι μπορεί να κάνει τις εφαρμογές πολύ πιο γρήγορες και δυναμικές, προσφέροντας στον χρήστη μια καλύτερη εμπειρία αλληλεπίδρασης. Επιπλέον, συμβάλλει καθοριστικά στην μείωση της χρήσης εύρους ζώνης (reduced bandwidth usage) καθώς μπορούν να ανανεωθούν κομμάτια της σελιδας χωρίς αυτή να χρειάζεται να ξαναφορτώσει. Η χρήση του AJAX απαιτεί όμως προσοχή, καθώς επειδή είναι τεχνολογία προσανατολισμένη στην πλευρά του πελάτη, μπορεί να οδηγήσει σε κενά ασφαλείας αν δεν ληφθούν τα απαραίτητα μέτρα κατά τις κλήσεις. Ένα ακόμα στοιχείο που πρέπει να λαμβάνεται υπόψην κατά τη χρήση AJAX για τις διάφορες λειτουργίες της εφαρμογής, είναι ότι το περιεχόμενο που ανανεώνεται με αυτόν τον τρόπο, δεν ανιχνεύεται από τις μηχανές αναζήτης.

3.6 Bootstrap

Το Bootsrap είναι ένα ισχυρό πλαίσιο εργασίας διεπαφών (front-end framework) για την ταχύτερη και ευκολότερη ανάπτυξη ιστοσελίδων. Βασίζεται στις τεχνολογίες HTML,CSS, Javascript και υποστηρίζει όλους τους γνωστούς φυλλομετρητές. Είναι πολύ εύκολο στην εγκατάσταση και στην χρήση, και άρα κατάλληλο για όχι τόσο έμπειρους προγραμματιστές. Προσφέρει ένα σταθερό και ευέλικτο σύστημα διάταξης (grid system) το οποίο είναι αποκρίσιμο (responsive) σε όλες τις διαστάσεις του φυλλομετρητή, άρα μία καλή επιλογή για σχεδιασμό φιλικών προς κινητά και ταμπλέτες ισττοσελίδων.



Παράδειγμα του Bootstrap Grid System

Περιλαμβάνει πλήθος από web-συστατικά (web-components), εικονίδια σε μορφή συμβολοσειρών (glyph icons) και είναι επεκτάσιμο μέσω των πολλών πρόσθετων Javascript

που είναι διαθέσιμα. Στοιχεία τυπογραφίας, πίνακες, φόρμες, κουμπιά, εικόνες και άλλα στοιχεία μπορούν να χρησιμοποιήσουν css κλάσεις που παρέχει το Bootstrap και να παραχθούν γρήγορα και εύκολα αποτελέσματα που θα χρειαζόντουσαν χρόνο και κόπο χωρίς την χρήση του.

Τέλος, έχει πολύ καλή τεκμηρίωση, μεγάλη κοινότητα υποστήριξης και υπάρχει μία πολύ μεγάλη γκάμα από templates (δωρεάν ή μη) που βασίζονται στο πλαίσο εργασίας αυτό.

```
bootstrap/
├── css/
│   ├── bootstrap.css
│   ├── bootstrap.css.map
│   ├── bootstrap.min.css
│   ├── bootstrap-theme.css
│   ├── bootstrap-theme.css.map
│   └── bootstrap-theme.min.css
├── js/
│   ├── bootstrap.js
│   └── bootstrap.min.js
└── fonts/
    ├── glyphicons-halflings-regular.eot
    ├── glyphicons-halflings-regular.svg
    ├── glyphicons-halflings-regular.ttf
    ├── glyphicons-halflings-regular.woff
    └── glyphicons-halflings-regular.woff2
```

Copy

Δομή αρχείων του Bootstrap

Συνοψη

Στο κεφάλαιο αυτό παρουσιάστηκαν οι τεχνολογίες που χρησιμοποιήθηκαν για την υλοποίηση της εφαρμογής της παρούσας διπλωματικής εργασίας. Στο επόμενο κεφάλαιο, θα περιγραφεί λεπτομερώς η αρχιτεκτονική του συστήματος της εφαρμογής που υλοποιήθηκε βάσει των τεχνολογιών που επιλέχτηκαν, η διαδικασία ανάλυσης και σχεδίασης με την μέθοδο RUP και διαγραμμάτων UML και η διάταξη των δεδομένων της εφαρμογής.

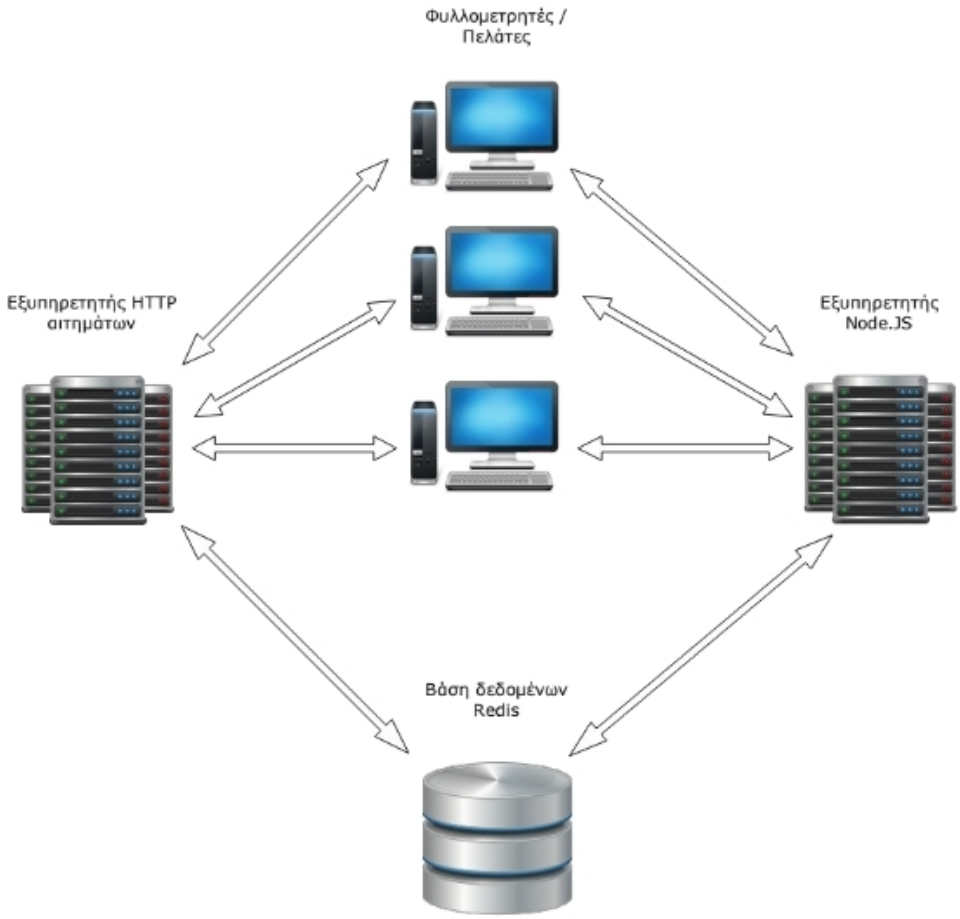
4 Αρχιτεκτονική

Στο κεφάλαιο αυτό παρουσιάζεται η αρχιτεκτονική του συστήματος, για την οποία θα αναφερθούν τα επίπεδα που την αποτελούν, η ανάλυση και ο σχεδιασμός της εφαρμογής με την μεθοδολογία UML, και η τελική διάταξη των δεδομένων (data layout) που αποφασίστηκε να υλοποιηθεί.

4.1 Αρχιτεκτονική Συστήματος

Η εφαρμογή της παρούσας διπλωματικής εργασίας αποτελείται από τα παρακάτω 4 διακριτά επίπεδα:

Το επίπεδο εξυπηρέτησης αιτημάτων HTTP (HTTP Application Server), το επίπεδο πελατών (φυλλομετρητές), το επίπεδο εξυπηρέτησης αιτημάτων επικοινωνίας αμφίδρομης και πραγματικού χρόνου και το επίπεδο αποθήκευσης δεδομένων, μόνιμης και προσωρινής.

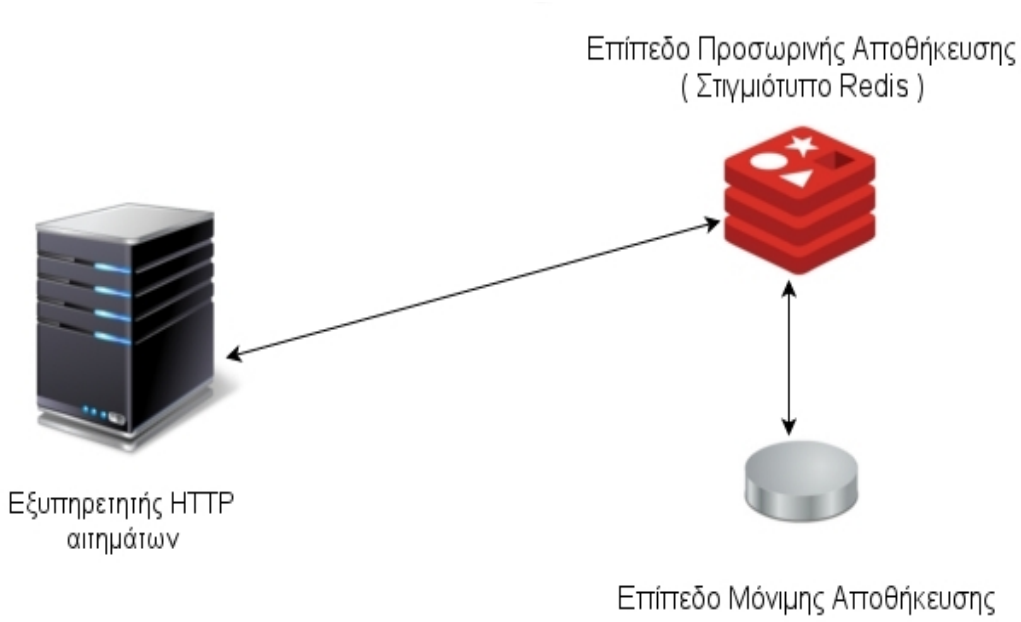


4.1.1 Επίπεδο εξυπηρέτησης HTTP αιτημάτων

Το επίπεδο αυτό δέχεται όλα τα αιτήματα που έρχονται από το επίπεδο πελατών μέσω του πρωτοκόλλου HTTP. Στο επίπεδο αυτό τρέχει ο Apache, ένα από τα δημοφιλέστερα ανοιχτού κώδικα λογισμικά εξυπηρετητή. Οι αιτήσεις των φυλλομετρητών λαμβάνονται, επεξεργάζονται από την εφαρμογή και ετοιμάζονται οι αποκρίσεις οι οποίες επιστρέφουν στον φυλλομετρητή. Στο επίπεδο αυτό κατοικεί το Laravel με τα αρχεία php της εφαρμογής, οι παρουσιάσεις, τα Css και Javascript αρχεία καθώς και τα λοιπά στοιχεία της όπως τα αρχεία εικόνων. Οι απαντήσεις που στέλνονται πίσω στο επίπεδο πελατών μπορεί να περιέχουν δεδομένα, HTML, Css και Javascript. Το επίπεδο εξυπηρέτησης αιτημάτων HTTP επικοινωνεί εκτός από το επίπεδο πελατών, και με αυτό της αποθήκευσης δεδομένων.

4.1.2 Επίπεδο Αποθήκευσης Δεδομένων.

Το επίπεδο αποθήκευσης δεδομένων αποτελείται από 2 επιμέρους επίπεδα. Το επίπεδο προσωρινής αποθήκευσης, όπου εκτελείται ένα στιγμιότυπο της Redis, και το επίπεδο μόνιμης αποθήκευσης το οποίο χρησιμοποιείται από την Redis ωστε να ελαχιστοποιείται ο κίνδυνος απώλειας δεδομένων και να είναι δυνατή η επαναφορά τους όταν επανεκκινεί ένα στιγμιότυπο Redis. Η προσωρινή αποθήκευση λαμβάνει χώρα στην μνήμη (RAM) του εξυπηρετητή από όπου και εξυπηρετούνται όλες οι ενέργειες ανάγνωσης αλλά και εγγραφής (σε πρώτο χρόνο). Το επίπεδο μόνιμης αποθήκευσης, χρησιμοποιείται από τη Redis για να αποθηκεύει στον δίσκο αντίγραφα του σετ δεδομένων από την μνήμη, καθώς και το ιστορικό όλων των ενεργειών εγγραφής ώστε να είναι δυνατή, μέσω της επανεκτέλεσης αυτών, η ανάκτηση του σετ δεδομένων.



Το Επίπεδο εξυπηρέτησης HTTP αιτημάτων επικοινωνεί με το Επίπεδο Προσωρινής Αποθήκευσης, το οποίο επικοινωνεί με το Επίπεδο Μόνιμης Αποθήκευσης.

4.1.3 Επίπεδο Πελατών

Το επίπεδο πελατών είναι αυτό στο οποίο βρίσκεται η διεπαφή με το χρήστη η οποία του επιτρέπει να αλληλεπιδρά με την εφαρμογή. Συνήθως αυτό γίνεται μέσω ενός φυλλομετρητή ο οποίος είναι ένα πρόγραμμα πλοήγησης στο διαδίκτυο. Ο φυλλομετρητής μπορεί να επεξεργάζεται αρχεία HTML, Css και Javascript και να προβάλλει τη σελίδα που προκύπτει από τον κώδικα και τα δεδομένα που έχουν φτάσει στο επίπεδο αυτό. Στο επίπεδο αυτό, η επικοινωνία με το επίπεδο του εξυπηρετητή γίνεται με HTTP αιτήσεις, με τις μεθόδους GET, POST και AJAX.

4.2 Ανάλυση και Σχεδιασμός Εφαρμογής

Η ανάλυση και ο σχεδιασμός της εφαρμογής έγινε με χρήση της Unified Modeling System (UML) η οποία είναι μία γλώσσα που χρησιμοποιείται για προδιαγραφές, αναπαράσταση με οπτικό τρόπο, δημιουργία και τεκμηρίωση των τμημάτων των συστημάτων λόγισμικού, αλλά και για μοντελοποίηση εταιρικών και άλλων συστημάτων που δεν αφορούν απαραίτητα λογισμικό.

Κάθε σύνθετο σύστημα προσεγγίζεται καλύτερα χρησιμοποιώντας ένα μικρό σύνολο

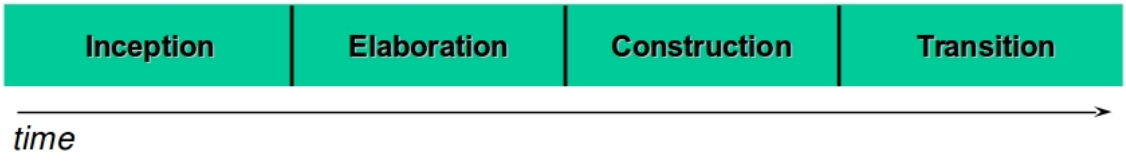
ανεξάρτητων όψεων του συστήματος. Τα καλύτερα μοντέλα συνδέονται με την πραγματικότητα, ενώ οι όψεις ενός μοντέλου μπορεί να οπτικοποιηθούν με τα παρακάτω γραφικά διαγράμματα:

- Διαγράμματα περιπτώσεων χρήσης (use case)
- Διαγράμματα κλάσεων (class)
- Διαγράμματα συμπεριφοράς
  - Διαγράμματα καταστάσεων (statechart)
  - Διαγράμματα δραστηριοτήτων (activity)
  - Διαγράμματα αλληλεπίδρασης
    - Διαγράμματα ακολουθίας (sequence)
    - Διαγράμματα συνεργασίας (collaboration)
- Διαγράμματα υλοποίησης
  - Διαγράμματα συνιστωσών (component)
  - Διαγράμματα deployment

Τα διαγράμματα αυτά παρέχουν διαφορετικές απόψεις του συστήματος κατά τη φάση της ανάλυσης και της ανάπτυξης. Το υποκείμενο μοντέλο ολοκληρώνει τις απόψεις αυτές, ούτως ώστε να μπορεί να αναλυθεί και να δομηθεί ένα πλήρες, συνεπές με τον εαυτό του σύστημα.

Παρέχει στους χρήστες μία έτοιμη προς χρήση οπτική γλώσσα μοντελοποίησης ώστε αυτοί να μπορούν να αναπτύξουν και να ανταλλάξουν κατανοητά μοντέλα.

Για την διαδικασία της ανάπτυξης του λογισμικού ακολουθήθηκε το μοντέλο Rational Unified Process (RUP) το οποίο χρησιμοποιεί την UML σαν γλώσσα μοντελοποίησης, καθοδηγείται από μελέτες χρήσης (use cases) και έχει τέσσερις φάσεις:



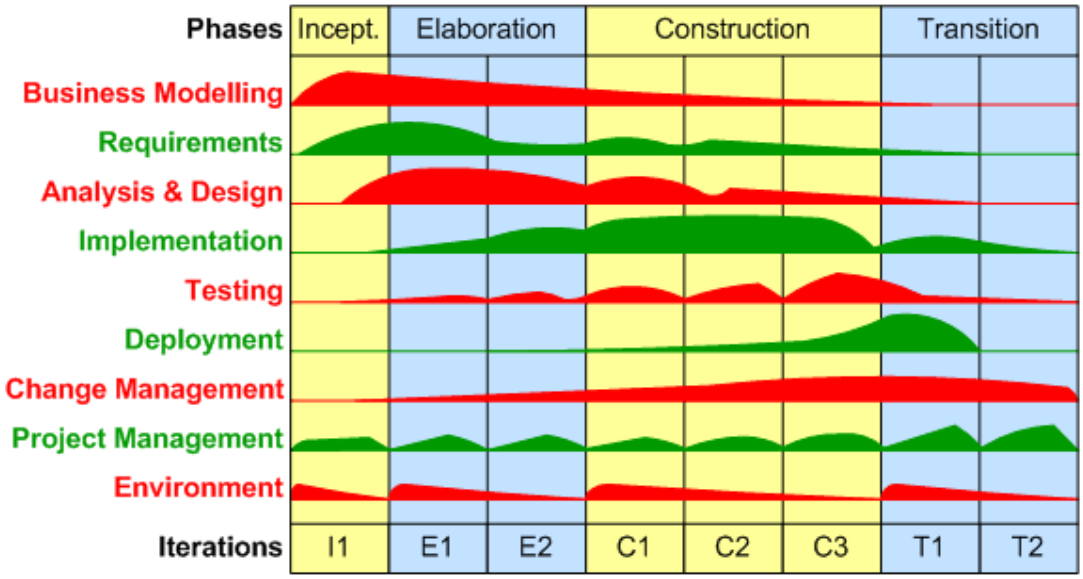
Οι τέσσερις φάσεις του μοντέλου RUP στον χρόνο

- Έναρξη (Inception): Ορισμός του έργου και της έκτασής του
- Επεξεργασία (Elaboration): Κατάστρωση μεθόδου υλοποίησης του έργου,

μοντελοποίηση χαρακτηριστικών του έργου, ορισμός της αρχιτεκτονικής του συστήματος

- Κατασκευή (Construction): Υλοποίηση του έργου
- Μετάβαση (Transition): Ανάπτυξη του συστήματος στο περιβάλλον χρήσης του

Η RUP μπορεί να περιγραφεί και αναπαρασταθεί με τη βοήθεια δύο διαστάσεων: του χρόνου και του περιεχομένου. 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.



Χρονική και χωρική διάσταση RUP

4.2.1 Σύλληψη Απαιτήσεων

Οι λειτουργικές απαιτήσεις της εφαρμογής που προέκυψαν παρουσιάζονται παρακάτω. Καθώς η εφαρμογή αυτή είναι προϊόν της συνεργασίας τριών φοιτητών, θα δοδεί έμφαση στις λειτουργίες που υλοποιήθηκαν για την παρούσα διπλωματική εργασία.

- Η εφαρμογή θα έχει χαρακτηριστικά ενός κοινωνικού δικτύου.
- Ο χρήστης θα μπορεί να εισάγει μία νέα βαθμολογία για ένα URL.
- Ο χρήστης θα μπορεί να επεξεργάζεται μία βαθμολογία που έχει καταχωρήσει.
- Η κάθε βαθμολογία θα έχει τα εξής χαρακτηριστικά:
  - Τον χρήστη που την εισήγαγε στο σύστημα
  - Τίτλο
  - Σκορ βαθμολογίας
  - Κείμενο σχολίου ή κριτικής (προαιρετικά)
  - Σημαία (flag) η οποία θα δηλώνει εάν τη βαθμολογία θα μπορούν να τη δουν άλλοι χρήστες ή θα είνα ιδιωτική.
- Ο χρήστης θα μπορεί να αναζητήσει βαθμολογίες, βάσει λέξεων-κλειδιών και, προαιρετικά, βάσει βαθμολογίας. Η αναζήτηση θα μπορεί να γίνει είτε ανάμεσα σε αυτές που έχει δημοσιεύσει ο χρήστης, είτε στις βαθμολογίες των χρηστών που ακολουθεί, είτε στο συνδυασμό των δύο αυτών συνόλων. Η αναζήτηση βάσει κειμένου (text-based search) θα πρέπει να μπορεί να γίνει ανεξαρτήτως γλώσσας, και να υπάρχει σχετική ανεκτικότητα σε μικρής έκτασης ορθογραφικά λάθη.
- Ο χρήστης θα μπορεί να δεί το προφίλ ενός χρήστη ή το δικό του. Στην οθόνη αυτή θα φαίνονται βασικές πληροφορίες του χρήστη (όνομα, ηλικία κ.λ.π.), ο αριθμός των βαθμολογήσεων που έχει κάνει, ο αριθμός των χρηστών που ακολουθεί και ο αριθμός των χρηστών που τον ακολουθούν.
- Η διάταξη προτύπου των παρουσιάσεων της εφαρμογής πρέπει να αποτελείται από 3 διακριτά μέρη:
  - Κεντρικό μενού στην κορυφή της σελίδας το οποίο θα προσφέρει επιλογές πλοήγησης στην εφαρμογή και τις απαραίτητες ενέργειες που θα μπορεί να εκτελέσει ο χρήστης.
  - Πλευρική μπάρα στο αριστερό μέρος της σελίδας με δυνατότητα γρήγορης βαθμολόγησης και βασικά στοιχεία του προφίλ του χρήστη.
  - Περιοχή προβολής περιεχομένου, ανάλογα με την σελίδα στην οποία έχει πλοηγηθεί ο χρήστης.
- Στην αρχική οθόνη της εφαρμογής, η οποία θα δείχνει το χρονολόγιο του χρήστη και στην οθόνη με τις προσωπικές βαθμολογίες του χρήστη θα πρέπει να υλοποιείται σελιδοποίηση (pagination) με χρήση AJAX.

4.2.2 Περιπτώσεις Χρήσης

Οι περιπτώσεις χρήσης που προκύπτουν από την ανάλυση των απαιτήσεων (και περιγράφουν λειτουργικότητες του συστήματος κυρίως από την πλευρά του τελικού χρήστη) είναι οι εξής:

4.2.2.1 Περίπτωση χρήσης: Εισαγωγή νέας βαθμολογίας

α. Εισαγωγή νέας βαθμολογίας από την φόρμα γρήγορης βαθμολόγησης στο πλευρικό μενού.

Βασική Ροή

- Ο χρήστης εισάγει το Url της σελίδας που θέλει να βαθμολογήσει στο σχετικό πεδίο της φόρμας
- Ο χρήστης επιλέγει σκορ βαθμολογίας από το σχετικό πεδίο της φόρμας
- Αποστέλει την βαθμολογία στο σύστημα πατώντας το κουμπί 'Submit'
- Το σύστημα δέχεται την αίτηση και αποθηκεύει την βαθμολογία
- Το σύστημα ανάλογα την σελίδα της εφαρμογής στην οποία βρισκόταν ο χρήστης κατά την υποβολή:
    - Ξαναφορτώνει τη σελίδα και δείχνει τη βαθμολογία στην κορυφή της λίστας (Αρχική σελίδα – χρονολόγιο και σελίδα λίστας βαθμολογιών του χρήστη)
    - Εμφανίζει αναδυόμενο μήνυμα επιτυχίας (λοιπές σελίδες)

Εναλλακτκή Ροή 1

- Το url της σελίδας που υπέβαλε ο χρήστης δεν είναι έγκυρο ή δεν υπάρχει. Ο χρήστης ενημερώνεται σχετικά με μήνυμα σε αναδυόμενο παράθυρο.

Εναλλακτική Ροή 2

- Ο χρήστης έχει ήδη βαθμολογήσει σε προγενέστερο χρόνο το Url που υπέβαλε. Το σύστημα επεξεργάζεται την υπάρχουσα βαθμολογία του χρήστη ανανεώνοντας το σκορ και ενημερώνει τον χρήστη με μήνυμα σε αναδυόμενο παράθυρο.
- Το σύστημα , ανάλογα την σελίδα της εφαρμογής στην οποία βρισκόταν ο χρήστης κατα την υποβολή:
    - Ξαναφορτώνει τη σελίδα και δείχνει τη βαθμολογία στην κορυφή της λίστας (Αρχική σελίδα – χρονολόγιο και σελίδα λίστας βαθμολογιών του χρήστη)
    - Εμφανίζει αναδυόμενο μήνυμα επιτυχίας (λοιπές σελίδες)

β. Εισαγωγή νέας βαθμολογίας από την σελίδα νέας βαθμολογίας της εφαρμογής

Βασική Ροή

- Ο χρήστης πατάει από το κεντρικό μενού την επιλογή 'New Rate'.

- Το σύστημα εμφανίζει την οθόνη εισαγωγής νέας βαθμολογίας (φόρμα).
- Ο χρήστης συμπληρώνει τα πεδία της φόρμας: Url, σκορ, επιλογή ιδιωτικής ή δημόσιας βαθμολόγησης, σχόλιο-κριτική.
- Αποστέλει την βαθμολογία στο σύστημα πατώντας το κουμπί 'Submit'.
- Το σύστημα δέχεται την αίτηση και αποθηκεύει την βαθμολογία.
- Εμφανίζεται αναδυόμενο μήνυμα επιτυχίας

Εναλλακτκή Ροή 1

- Το σύστημα ανιχνεύει ελλειπή στοιχεία, ή λανθασμένα στοιχεία τα οποία υποβλήθηκαν, και ενημερώνει σχετικά τον χρήστη.

Εναλλακτική Ροή 2

- Ο χρήστης έχει ήδη βαθμολογήσει σε προγενέστερο χρόνο το Url που υπέβαλε. Το σύστημα επεξεργάζεται την υπάρχουσα βαθμολογία του χρήστη ανανεώνοντας το σκορ και ενημερώνει τον χρήστη με μήνυμα σε αναδυόμενο παράθυρο.

γ. Εισαγωγή νέας βαθμολογίας από αντικείμενο βαθμολόγησης άλλου χρήστη

Βασική Ροή

- Ο χρήστης πατάει το κουμπί με το αστεράκι στην κάτω δεξιά γωνία της καρτέλας μιας βαθμολογίας ενός χρήστη που ακολουθεί. Το κουμπί αυτό είναι ορατό μόνο σε καρτέλες βαθμολογιών σελίδων τις οποίες δεν έχει βαθμολογήσει ο ίδιος.
- Το σύστημα εμφανίζει στην οθόνη ένα αναδυόμενο παράθυρο με φόρμα βαθμολόγησης της σελίδας, με προσυμπληρωμένο τον τίτλο της.
- Ο χρήστης συμπληρώνει τα πεδία της φόρμας: Σκορ, επιλογή ιδιωτικής ή δημόσιας βαθμολόγησης, σχόλιο-κριτική και πατάει το κουμπί 'Submit'.
- Το σύστημα δέχεται την αίτηση και αποθηκεύει την βαθμολογία.
- Το σύστημα , ανάλογα την σελίδα της εφαρμογής στην οποία βρισκόταν ο χρήστης κατά την υποβολή:
  - Εμφανίζει τη βαθμολογία στην κορυφή της λίστας (Αρχική σελίδα – χρονολόγιο)
  - Εμφανίζει ένα αναδυόμενο μήνυμα επιτυχίας

4.2.2.2 Περίπτωση χρήσης: Επεξεργασία βαθμολογίας

Βασική Ροή

- Ο χρήστης πατάει το κουμπί με το εικονίδιο 'μολύβι' στην κάτω δεξιά καρτέλα μιας βαθμολογίας του (Οθόνη χρονολογίου και σελίδα λίστας των βαθμολογιών του).
- Το περιεχόμενο της καρτέλας της βαθμομολογίας αλλάζει και εμφανίζεται φόρμα

βαθμολόγησης της σελίδας, με προσυμπληρωμένο τον τίτλο, το σκορ και την κριτική.

- Ο χρήστης αλλάζει τα πεδία που θέλει και υποβάλλει την φόρμα πατώντας το κουμπί 'Save changes'.
- Το σύστημα δέχεται την αίτηση και αποθηκεύει τις αλλαγές.
- Η καρτέλα βαθμολογίας επανέρχεται στην αρχική της κατάσταση (η φόρμα αποκρύπτεται) με τα νέα επεξεργασμένα στοιχεία που υπέβαλε ο χρήστης. Ένα αναδυόμενο μήνυμα ενημερώνει τον χρήστη για την επιτυχή επεξεργασία.

4.2.2.3 Περίπτωση Χρήσης: Αναζήτηση βαθμολογίας

- Ο χρήστης πατάει από το κεντρικό μενού την επιλογή 'Search Rates'.
- Το σύστημα εμφανίζει την σελίδα αναζήτησης βαθμολογιών.
- Ο χρήστης συμπληρώνει τα πεδία βάσει των οποίων θέλει να εκτελεστεί η αναζήτηση:
  - Εισάγει λέξεις-κλειδιά στο αντίστοιχο πεδίο της φόρμας
  - Επιλέγει αν η αναζήτηση θα αφορά δικές του βαθμολογίες, βαθμολογίες χρηστών που ακολουθεί ή και τα δύο.
  - Προαιρετικά, πατάει το κουμπί προχωρημένης αναζήτησης 'Advanced Search', οπότε και εμφανίζονται δύο πεδία επιλογής εύρους σκορ ('From', 'To').
- Ο χρήστης πατάει το κουμπί υποβολής της φόρμας αναζήτησης με το εικονίδιο 'Μεγενθυντικός φακός' και η φόρμα αποστέλεται.
- Το σύστημα δέχεται την αίτηση, εκτελεί την αναζήτηση και επιστρέφει τα αποτελέσματα.
- Το σύστημα εμφανίζει οθόνη αποτελεσμάτων αναζήτησης, με τις βαθμολογίες που βρέθηκαν, ή με σχετικό μήνυμα αν δεν βρέθηκε κάτι.

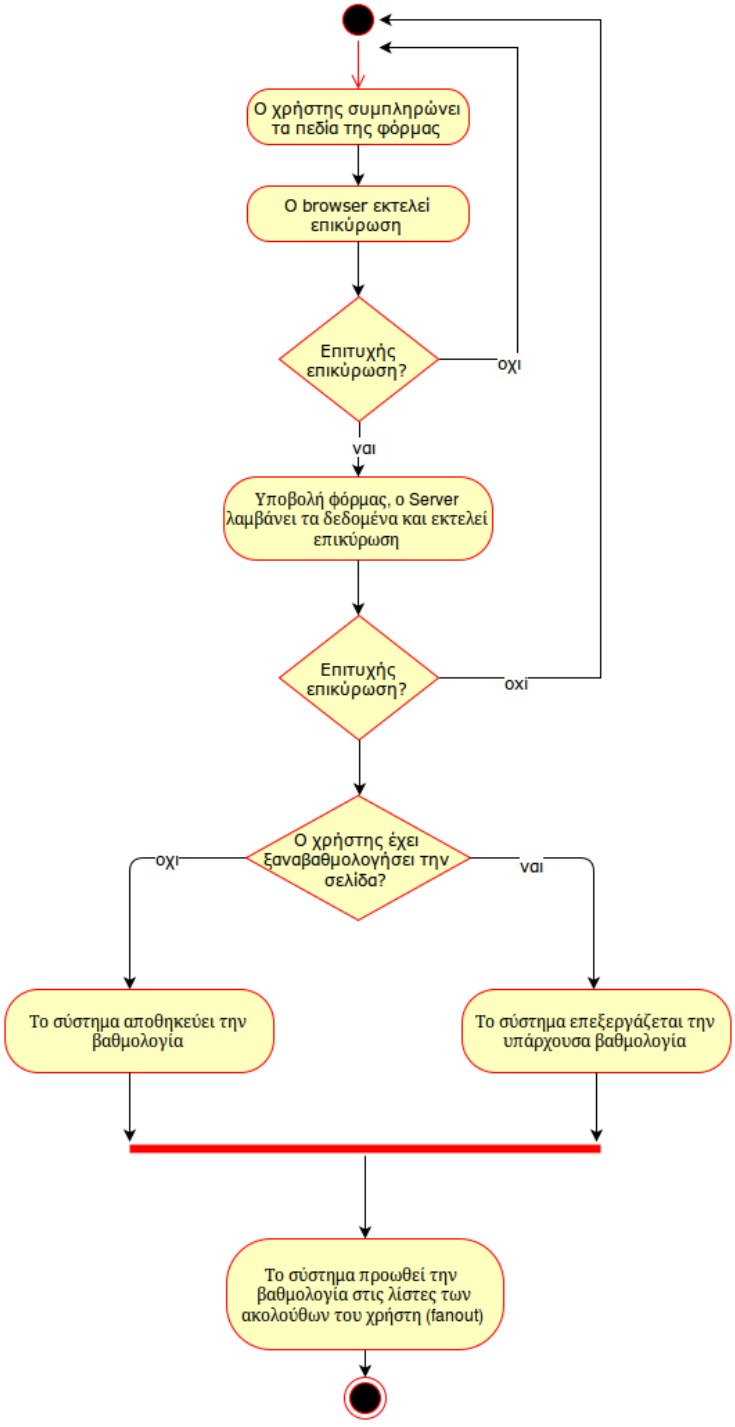
4.2.3 Διαγράμματα Δραστηριοτήτων

Στην συνέχεια παρουσιάζονται τα διαγράμματα δραστηριοτήτων για τις βασικότερες περιπτώσεις χρήσης και τις εσωτερικά παραγόμενες ενέργειες που αυτές μπορεί να προκαλούν.

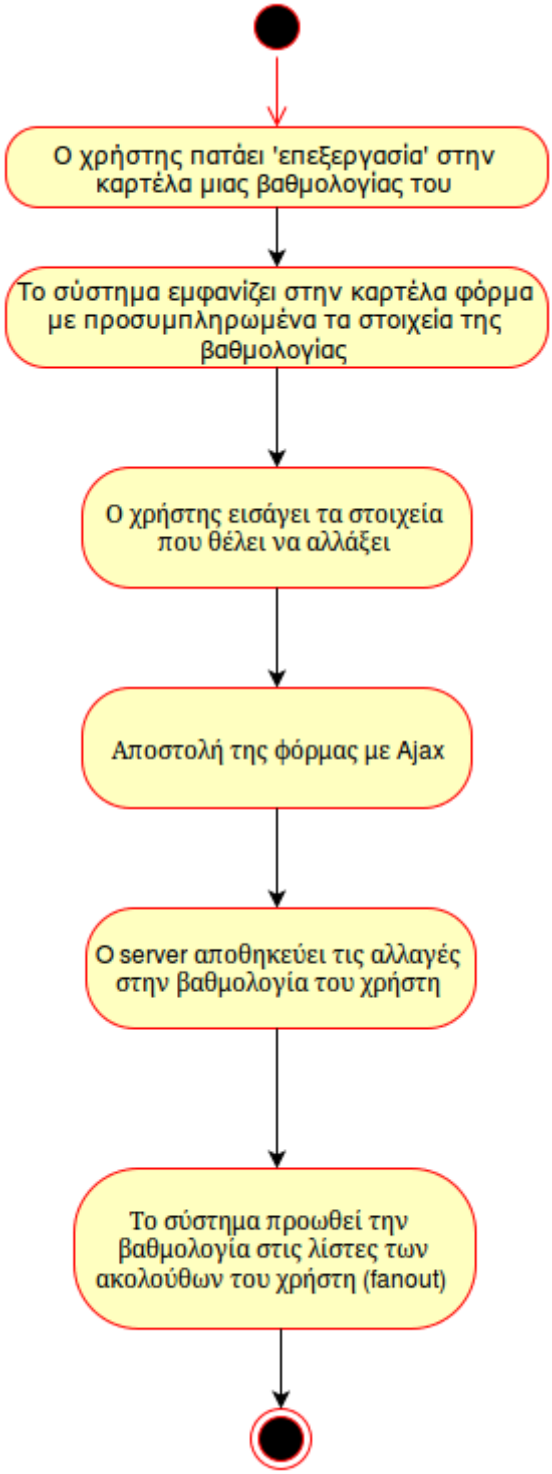
4.2.3.1 Διάγραμμα δραστηριοτήτων: Εισαγωγή βαθμολογίας στο σύστημα

Ο χρήστης μπορεί να εισαγάγει μία βαθμολογία στο σύστημα με τρεις τρόπους: Από τη σελίδα νέας βαθμολογίας, από την πλευρική μπάρα που είναι πάντα ορατή σε όλες τις σελίδες (γρήγορη βαθμολόγηση) και από μία υπάρχουσα βαθμολογία άλλου χρήστη, πατώντας το αντίστοιχο κουμπί στην καρτέλα της βαθμολογίας. Και στις τρεις περιπτώσεις ο χρήστης

συμπληρώνει πεδία σε φόρμα και την υποβάλλει. Το σύστημα δέχεται τα δεδομένα και ανάλογα με την περίπτωση, αποθηκεύει τη νέα βαθμολογία ή επεξεργάζεται την υπάρχουσα, σε περίπτωση που έχει ξαναβαθμολογηθεί από το χρήστη. Το σύστημα στη συνέχεια ενημερώνει τις λίστες του χρήστη και των ακολούθων του με τη βαθμολογία αυτή.

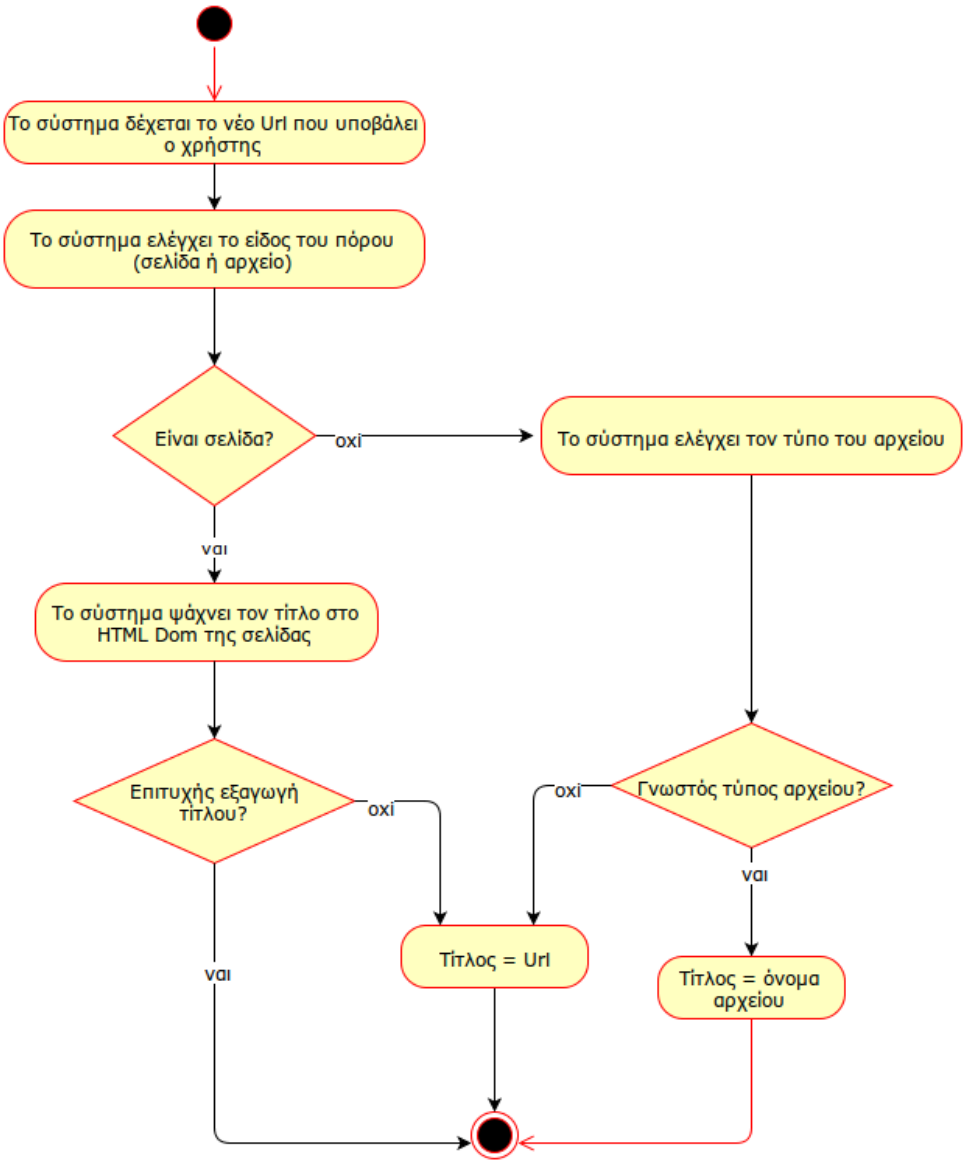


4.2.3.2 Διάγραμμα δραστηριοτήτων: Επεξεργασία βαθμολογίας



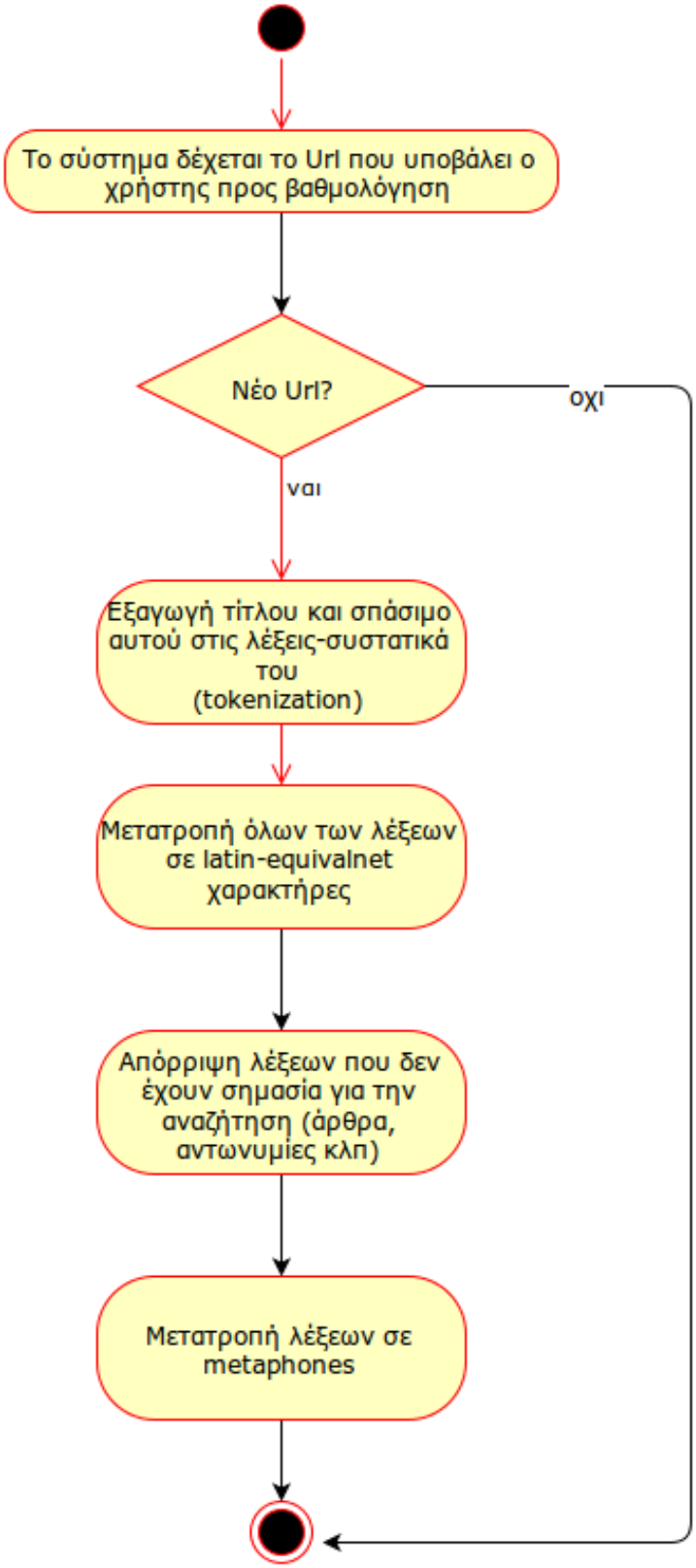
4.2.3.3 Διάγραμμα Δραστηριοτήτων: Εξαγωγή τίτλου βαθμολογίας

Κατά την εισαγωγή μίας βαθμολογίας, το σύστημα πρέπει να παραγάγει τον τίτλο αυτής. Αρχικά ελέγχεται το είδος του πόρου, αν δηλαδή πρόκειται για ιστοσελίδα ή για αρχείο. Αν είναι σελίδα, το σύστημα προσπαθεί να εξαγάγει τον τίτλο από το μοντέλο αντικειμένων εγγράφου της σελίδας (HTML Dom Tree), και συγκεκριμένα από το meta tag 'title' στο head της σελίδας. Αν δεν μπορέσει να εξάγει τον τίτλο, σαν τίτλος τίθεται το ίδιο το Url, χωρίς το πρόθεμα 'http://www'. Αν είναι αρχείο, το σύστημα θέτει σαν τίτλο το όνομα του αρχείου (χωρίς την κατάληξη) εφόσον πρόκειται για γνωστό τύπο αρχείου, αλλιώς σαν τίτλος αποθηκεύεται το Url όπως παραπάνω.



4.2.3.4 Διάγραμμα Δραστηριοτήτων: Εξαγωγή λέξεων-κλειδιών σαν metaphones από τον τίτλο βαθμολογίας

Στην παρούσα εργασία αποφασίστηκε η δημιουργία ενός ευρετηρίου για αναζήτηση βάσει κειμένου στους τίτλους των βαθμολογιών που είναι αποθηκευμένες στο σύστημα. Όλες οι λέξεις-κλειδιά στις οποίες μπορεί να γίνει αναζήτηση αποθηκεύονται σαν Metaphones. Η βασική ιδέα του αλγόριθμου που χρησιμοποιήθηκε (Double Metaphone) είναι να παράγει για μία λέξη, μία 'χοντρική' αναπαράσταση αυτής βάσει κάποιων κανόνων. Αυτό το νέο string μπορεί να αποτελεί το ίδιο κλειδί και να αντιστοιχεί και σε άλλες λέξεις που προφέρονται με παρόμοιο τρόπο. Για παράδειγμα, το metaphone της λέξης '*steven*' είναι 'STFN', ενώ το ίδιο είναι και για τις λέξεις '*steeven*' ή '*stefen*'. Ο λόγος που αποφασίστηκε να χρησιμοποιηθεί αυτή η τεχνική είναι για τη δημιουργία ενός 'ασαφούς' ευρετηρίου (fuzzy indexing) (Arehart, M. 2010), με βάση το οποίο θα μπορούν να εξάγονται αποτελέσματα σε μία αναζήτηση βάσει κειμένου η οποία θα 'συγχωρεί' τυχόντα (μικρής έκτασης) ορθογραφικά λάθη. Ο συγκεκριμένος αλγόριθμος που χρησιμοποιήθηκε μπορεί να παραγάγει για κάθε λέξη που του δίνεται από ένα έως δύο metaphones.

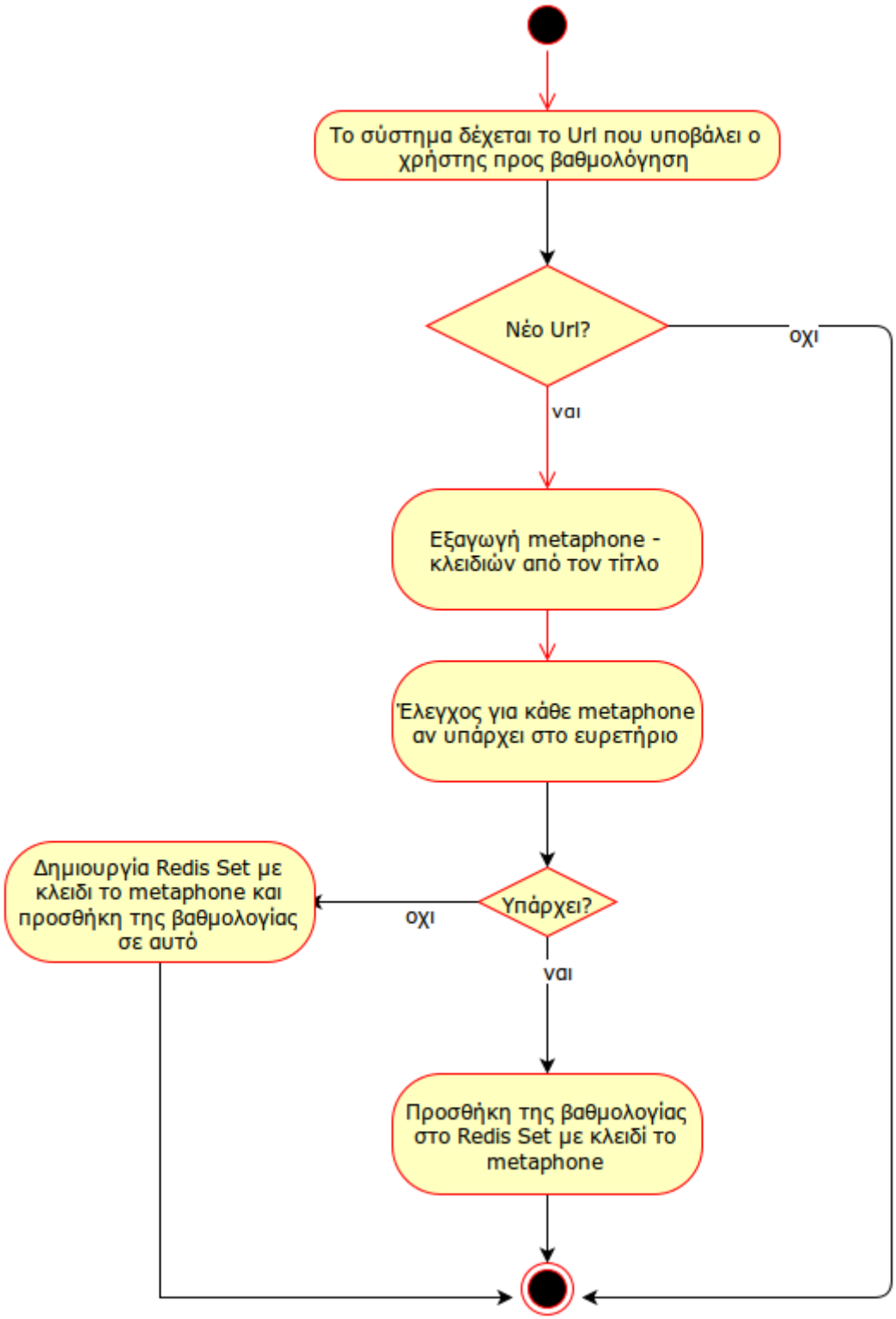

Το σύστημα δέχεται το Url που υποβάλει ο χρήστης προς βαθμολόγηση
Νέο Url?
οχι
ναι
Εξαγωγή τίτλου και σπάσιμο αυτού στις λέξεις-συστατικά του (tokenization)
Μετατροπή όλων των λέξεων σε latin-equivalnet χαρακτήρες
Απόρριψη λέξεων που δεν έχουν σημασία για την αναζήτηση (άρθρα, αντωνυμίες κλπ)
Μετατροπή λέξεων σε metaphones

4.2.3.5 Διάγραμμα Δραστηριοτήτων: Δημιουργία – ενημέρωση ασαφούς ευρετηρίου

Όταν εισάγεται μία νέα βαθμολογία στο σύστημα, γίνεται εξαγωγή των λέξεων-κλειδιών με χρήση του αλγορίθμου Double Metaphone από τον τίτλο. Για κάθε ένα metaphone, γίνεται έλεγχος αν υπάρχει στο ευρετήριο. Το ευρετήριο αποτελείται απο Redis Sets, όπου κάθε ένα έχει σαν κλειδί ένα metaphone και τα μέλη του είναι τα Ids των βαθμολογιών στους τίτλους των οποίων υπάρχει τουλάχιστον μία φορά η λέξη που αντιστοιχεί σε αυτό το metaphone. Για παράδειγμα:

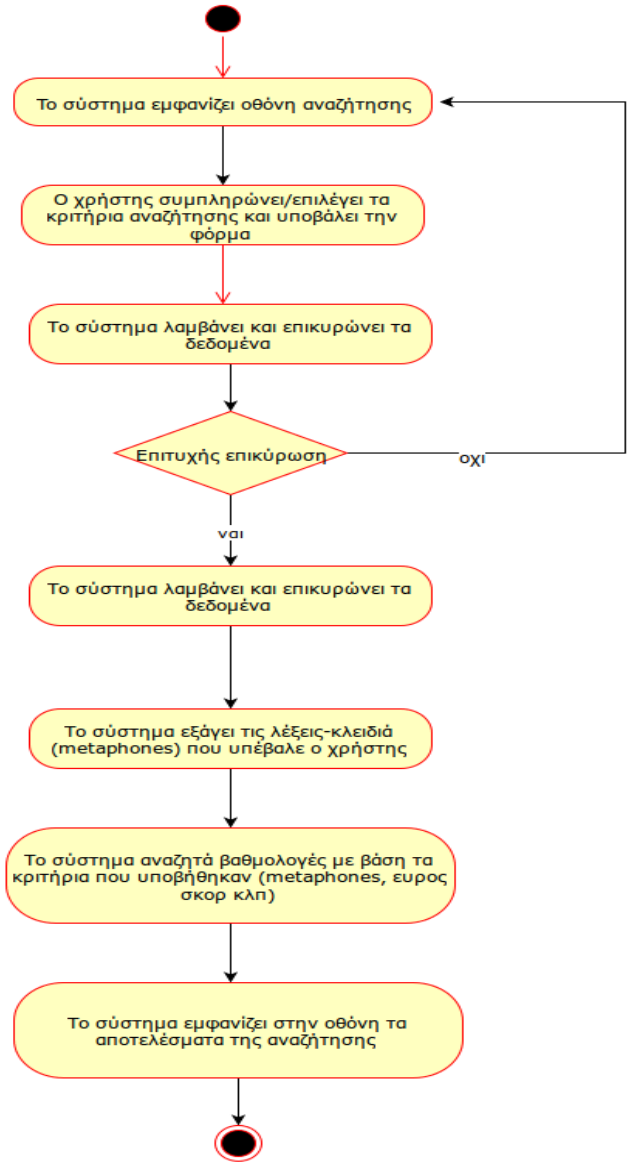
- word::PTN = SET(12, 33, 56) (metaphone λέξης κλειδιού 'python')
- word::AFMR = SET(101, 333, 156) (metaphone λέξης κλειδιού 'εφημερίδα')

Αν στο ευρετήριο υπάρχει ήδη το metaphone, το σύστημα προσθέτει στο υπάρχον σετ το Id της βαθμολογίας. Αν όχι, τότε δημιουργείται νέο σετ με πρώτο μέλος τη βαθμολογία αυτή.


Το σύστημα δέχεται το Url που υποβάλει ο χρήστης προς βαθμολόγηση
Νέο Url?
οχι
ναι
Εξαγωγή metaphone - κλειδιών από τον τίτλο
Έλεγχος για κάθε metaphone αν υπάρχει στο ευρετήριο
Δημιουργία Redis Set με κλειδι το metaphone και προσθήκη της βαθμολογίας σε αυτό
οχι
Υπάρχει?
ναι
Προσθήκη της βαθμολογίας στο Redis Set με κλειδί το metaphone

4.2.3.6 Διάγραμμα Δραστηριοτήτων: Αναζήτηση βαθμολογίας

Ο χρήστης μπορεί να αναζητήσει βαθμολογίες που έχουν υποβληθεί στο σύστημα είτε από τον ίδιο, είτε από χρήστες που τον ακολουθούν. Μπορεί να εισαγάγει λέξεις κλειδιά καθώς και εύρος σκορ βαθμολογιών.



4.3 Διάταξη Δεδομένων

Όπως έχει ήδη αναφερθεί, για την εφαρμογή της παρούσας εργασίας χρησιμοποιήθηκε η Redis για την αποθήκευση δεδομένων. Αρχικά, έπρεπε να αποφασιστεί ο τρόπος που θα αποθηκεύεται η βασική οντότητα της εφαρμογής, η βαθμολογία. Τελικά προτάθηκαν 2 σενάρια:

Σύμφωνα με το πρώτο, θα υπάρχουν 2 οντότητες οι οποίες θα χαρακτηρίζουν πλήρως μία βαθμολογία. Η μία θα ενσωματώνει όλες τις πληροφορίες που σχετίζονται με τη διεύθυνση του πόρου (Url), και η άλλη θα περιέχει τις πληροφορίες σχετικά με τις πληροφορίες που εισάγει ένας χρήστης για τον πόρο αυτό (σκορ, κριτική κ.λ.π.).

Resource			
Redis Hash με πληροφορίες για τον πόρο			
Key	Field	Field description	Value
rid:xxx	url	Διεύθυνση πόρου	string
	thumbnail	Στιγμιότυπο εικόνας	string
	timestamp	Χρονοσφραγίδα	int
	title	Τίτλος	string
	average_score	Μ.Ο. σκορ βαθμολογήσεων του πόρου	float

Rate			
Redis Hash με πληροφορίες για την βαθμολόγηση ενός πόρου			
Key	Field	Field description	Value
rate:xxx	uid	Id χρήστη	int
	rid	Id πόρου	int
	score	Σκορ βαθμολογίας	int
	review	Κριτική - Σχόλια	string
	timestamp	Χρονοσφραγίδα	int
	private	Ιδιωτική ή δημόσια βαθμολογία	boolean

Σύμφωνα με τη δεύτερη υποψήφια διάταξη, όλες οι πληροφορίες σχετικά με μία βαθμολογία που υποβάλλεται στο σύστημα αποθηκεύονται σε μία οντότητα:

Rate		
Redis Hash με όλες τις πληροφορίες για τον πόρο και την βαθμολογία του χρήστη		
Key	Field	Value
rid:xxx	url	string
	title	string
	thumbnail	string
	timestamp	int
	uid	int
	score	int
	review	string
	public	boolean
	average_score	float

Σύγκριση διατάξεων

Και οι δύο διατάξεις σε διάφορα συγκριτικά τεστ που έγιναν παρουσίασαν παρόμοια αποτελέσματα από άποψη χρόνου εκτέλεσης και όγκου δεδομένων. Η πρώτη διάταξη προσφέρει μεγαλύτερη ευελιξία ενημέρωσης και αποθήκευσης πληροφοριών που σχετίζονται με τον πόρο και όχι το χρήστη όπως ο μέσος όρος του σκορ όλων των βαθμολογήσεων που έχουν γίνει πάνω σε αυτόν. Βασικό μειονέκτημα είναι ότι χρειάζονται πρόσθετες, συχνά σύνθετες ενέργειες εγγραφής-ανάγνωσης για βασικές λειτουργίες της εφαρμογής. Αντίθετα, στη δεύτερη διάταξη, ο μέσος όρος σκορ θα έπρεπε μεν να υπολογίζεται και να ενημερώνεται σε όλες τις αποθηκευμένες βαθμολογίες αυτού του πόρου, σε κάθε νέα εισαγωγή ή επεξεργασία του σκορ βαθμολογίας από ένα χρήστη, απαιτεί όμως λιγότερες ενέργειες για τις συχνότερες λειτουργίες της εφαρμογής, όπως για την ανάκτηση μίας βαθμολογίας ενός χρήστη καθώς όλη η πληροφορία πόρου – βαθμολόγησης βρίσκεται συγκεντρωμένη σε ένα Redis Hash.

Τελικά αποφασίστηκε να χρησιμοποιηθεί η δεύτερη διάταξη, καθώς η πρώτη παραπέμπει σε σχεδιασμό αποθήκευσης σε σχεσιακή βάση δεδομένων, ενώ η Redis αποτελεί μία βάση δεδομένων στηριγμένη στην λογική κλειδιού-τιμής. Για καλύτερη απόδοση, αποφασίστηκε ο μέσος όρος του σκορ βαθμολογιών ενός πόρου να αποθηκεύεται σε ξεχωριστή δομή.

Στην συνέχεια παρουσιάζεται η διάταξη δεδομένων που χρησιμοποιήθηκε στο σύνολο της εφαρμογής.

User

Hash – Πληροφορίες σχετικές με τον χρήστη

key	Field	Value	Περιγραφή - Σχόλια
uid:xxx	email	email string	Email
	password	password string	Κωδικός εισόδου
	username	username string	Ψευδώνυμο χρήστη
	auth	authentication string	Κωδικός πιστοποίησης χρήστη
	verified	0-1	Πιστοποιημένος χρήστης
	fname	first name string	Όνομα
	lname	last name string	Επώνυμο
	age	int	Ηλικία
	sex	male-female	Φύλο
	profile_pic:	image Url string	Url εικόνας προφίλ χρήστη

Παράδειγμα: Ανάκτηση πληροφοριών του χρήστη με Id=1

```
127.0.0.1:6379> hgetall uid:1
 1) "email"
 2) "kosnik78@gmail.com"
 3) "password"
 4) "1a1dc91c907325c69271ddf0c944bc72"
 5) "username"
 6) "kosnik78"
 7) "verified"
 8) "1"
 9) "fname"
10) "Nikos"
11) "lname"
12) "Kosmopoulos"
13) "age"
14) "36"
15) "sex"
16) "male"
17) "auth"
18) "2cf533d0dc41f5c5981d1f5e8b61ad56"
19) "thumb"
20) "images/avatar/avatar1.jpg"
```

Hash – Αποθήκευση σε ένα Hash τα ζεύγη username - user id όλων των χρηστών

Key	Field	Value	Περιγραφή - Σχόλια
username	<username>	<uid>	Ανάκτηση κωδικού χρήστη από το username

Παράδειγμα: Ανάκτηση Id του χρήστη με username 'kosnik78'

```
127.0.0.1:6379> hget username kosnik78
"1"
```

Hash – Αποθήκευση σε ένα Hash τα ζεύγη email- user id όλων των χρηστών

Key	Field	Value	Περιγραφή - Σχόλια
email	<email>	<uid>	Ανάκτηση κωδικού χρήστη από το email

Παράδειγμα: Ανάκτηση Id του χρήστη με email 'kosnik78@gmail.com'

```
127.0.0.1:6379> hget email kosnik78@gmail.com
"1"
```

Followers – Followings

Sorted Set – Οι ακόλουθοι ενός χρήστη

key	members	Περιγραφή - Σχόλια
followers:xxx	uids – timestamp as score	Σε κάθε χρήστη αντιστοιχεί ένα sorted set με μέλη τα Ids των χρηστών που τον ακολουθούν. Κάθε μέλος-Id έχει σαν σκορ τη χρονοσφραγίδα που δηλώνει πότε προστέθηκε σαν μέλος του σετ.

Παράδειγμα: Ανάκτηση ακολούθων του χρήστη με Id=1

```
127.0.0.1:6379> zrange followers:1 0 -1
1) "4"
2) "2"
```

Παράδειγμα: Ανάκτηση πλήθους ακολούθων του χρήστη με Id=1

```
127.0.0.1:6379> zcard followers:1
(integer) 2
```

Sorted Set – Οι χρήστες που ακολουθεί ένας χρήστης

key	members	Περιγραφή - Σχόλια
followings:xxx	uids – timestamp as score	Σε κάθε χρήστη αντιστοιχεί ένα sorted set με μέλη τα Ids των χρηστών που αυτός ακολουθεί. Κάθε μέλος-Id έχει σαν σκορ τη χρονοσφραγίδα που δηλώνει πότε προστέθηκε σαν μέλος του σετ.

Rating

Hash – Το αντικείμενο βαθμολογίας

key	Field	Value	Περιγραφή - Σχόλια
rid:xxx	url	url string	Η διεύθυνση του πόρου
	title	title string	Τίτλος πόρου
	thumbnail	Image url string	Η διεύθυνση του στιγμιότυπου του πόρου
	timestamp	int	Χρονοσφραγίδα αποθήκευσης ή επεξεργασίας
	uid	int	Id του χρήστη της βαθμολογίας
	score	int	Σκορ βαθμολογίας
	review	text	Κείμενο κριτικής
	public	0-1	Ιδιωτική ή δημόσια βαθμολογία.

Παράδειγμα: Ανάκτηση όλων των πληροφοριών της βαθμολογίας με id=9

```
127.0.0.1:6379> hgetall rid:9
 1) "url"
 2) "http://reddit.com"
 3) "title"
 4) "reddit: the front page of the internet"
 5) "timestamp"
 6) "1416163374.7449"
 7) "uid"
 8) "2"
 9) "score"
10) "7"
11) "review"
12) "great site!"
13) "thumb"
14) "images/thumbnails/thumb6.jpg"
15) "public"
16) "1"
```

Hash – Ζεύγη κωδικού χρήστη – κωδικού βαθμολογίας για έναν πόρο

key	Field	Value	Περιγραφή - Σχόλια
url:<url string>	<uid>	<rid>	Σε κάθε πόρο αντιστοιχεί ένα hash στο οποίο για κάθε βαθμολογία που αναφέρεται σε αυτόν, αποθηκεύεται το ζεύγος κωδικός χρήστη – κωδικός βαθμολογίας

Παράδειγμα: Ανάκτηση κωδικού βαθμολογίας του site reddit.com για τον χρήστη με Id=2

```
127.0.0.1:6379> hget url:http://reddit.com 2
"9"
```

Παράδειγμα: Ανάκτηση των κωδικών όλων των χρηστών που έχουν βαθμολογήσει το site reddit.com

```
127.0.0.1:6379> hkeys url:http://reddit.com
1) "2"
2) "4"
```

Hash – Ζεύγη κωδικού βαθμολογίας – σκορ βαθμολογίας για έναν πόρο

key	Field	Value	Περιγραφή - Σχόλια
url:<url string>:rscores	<rid>	<score>	Σε κάθε πόρο αντιστοιχεί ένα hash στο οποίο για κάθε βαθμολογία που αναφέρεται σε αυτόν, αποθηκεύεται το ζεύγος κωδικός βαθμολογίας – σκορ.

Παράδειγμα: Ανάκτηση όλων των κωδικών βαθμολογιών μαζί με το σκορ τους για την σελίδα Imdb.com

```
127.0.0.1:6379> hgetall url:http://imdb.com:rscores
1) "22"
2) "6"
3) "25"
4) "3"
```

* Αποτέλεσμα: 2 ζεύγη : Βαθμολογία με κωδικό 22 και σκορ 6, βαθμολογία με κωδικό 25 και σκορ 3.

Hash – Ζεύγη κωδικού χρήστη – σκορ βαθμολογίας για έναν πόρο

key	Field	Value	Περιγραφή - Σχόλια
url:<url string>:uscores	<uid>	<score>	Σε κάθε πόρο αντιστοιχεί ένα hash στο οποίο για κάθε χρήστη που τον έχει βαθμολογήσει, αποθηκεύεται το ζεύγος κωδικός χρήστη – σκορ.

Παράδειγμα: Ανάκτηση του σκορ βαθμολογίας που έδωσε ο χρήστης με κωδικό 3 για την σελίδα imdb.com

```
127.0.0.1:6379> hget url:http://imdb.com:uscores 3
"6"
```

List – Λίστα με όλες τις βαθμολογίες ενός χρήστη

key	Values	Περιγραφή - Σχόλια
own_ratings:xxx	List of rids	Στην κορυφή της λίστας βρίσκεται πάντα η πιο πρόσφατη βαθμολογία του χρήστη

Παράδειγμα: Ανάκτηση λίστας όλων των βαθμολογιών του χρήστη με id=1

```
127.0.0.1:6379> lrange own_ratings:1 0 -1
 1) "106"
 2) "124"
 3) "121"
 4) "120"
 5) "112"
 6) "119"
 7) "114"
 8) "109"
 9) "110"
10) "113"
11) "108"
12) "107"
13) "3"
```

List – Λίστα με όλες τις βαθμολογίες του χρονολογίου ενός χρήστη

key	Values	Περιγραφή - Σχόλια
timeline_ratings:xxx	List of rids	Η λίστα αυτή κρατάει όλες τις βαθμολογίες που μπορούν να φαίνονται στο χρονολόγιο ενός χρήστη: τις δικές του και τις δημόσιες βαθμολογίες των χρηστών που ακολουθεί.

Παράδειγμα: Ανάκτηση των 5 πιο πρόσφατων βαθμολογιών του χρονολογίου του χρήστη με κωδικό 3

```
127.0.0.1:6379> lrange timeline_ratings:3 0 4
1) "102"
2) "96"
3) "64"
4) "63"
5) "61"
```

Set – Σετ με μέλη τις διευθύνσεις που έχει βαθμολογήσει ένας χρήστης

key	Members	Περιγραφή - Σχόλια
uid:xxx:rated_urls	Resource urls	Επιλέχθηκε set για δυνατότητα γρήγορου ελέγχου για το αν ο χρήστης έχει ξαναβαθμολογήσει ένα url

Παράδειγμα: Έλεγχος για το αν ο χρήστης με κωδικό 1 έχει βαθμολογήσει την σελίδα imdb.com (απάντηση αρνητικη)

```
127.0.0.1:6379> SISMEMBER uid:1:rated_urls 'http://imdb.com'
(integer) 0
```

Παράδειγμα: Ανάκτηση όλων των διευθύνσεων που έχει βαθμολογήσει ο χρήστης με κωδικό 1

```
127.0.0.1:6379> smembers uid:1:rated_urls
 1) "http://mikrokosmos.gr"
 2) "http://stumbleupon.com"
 3) "http://codeigniter.com"
 4) "http://spitogatos.gr"
 5) "http://subs4free.com"
 6) "http://spiti24.gr"
 7) "http://redis.io"
 8) "http://rottentomatoes.com"
 9) "http://tospitimou.gr"
10) "http://papasikas.gr"
```

Set – Σετ με μέλη τους κωδικούς όλων των βαθμολογιών ενός χρήστη

key	Members	Περιγραφή - Σχόλια
uid:xxx:rids	rids	

Set – Σετ με μέλη τους κωδικούς όλων των δημόσιων βαθμολογιών ενός χρήστη

key	Members	Περιγραφή - Σχόλια
uid:xxx:publicRids	rids	Περιέχει τις βαθμολογίες του χρήστη που είναι δημόσιες και άρα μπορούν να εμφανιστούν στα χρονολόγια των χρηστών που τον ακολουθούν.

Key - Value Μέσος όρος βαθμολογίας ενός πόρου

key	Members	Περιγραφή - Σχόλια
url:<url string>:average_score	rids	Ενα απλό ζευγάρι κλειδιού (διεύθυνση url) – τιμής (μεσος όρος σκορ) για κάθε πόρο

Παράδειγμα: Ανάκτηση μέσου όρου σκορ βαθμολογίας της σελίδα imdb.com

```
127.0.0.1:6379> get url:http://imdb.com:average_score
"4.5"
```

Sorted Set – Διευθύνσεις που έχουν βαθμολογηθεί με κατάταξη τον μέσο όρο βαθμολογίας τους

key	members	Περιγραφή - Σχόλια
scoreBasedChart	Url strings	Χρησιμοποιείται για την ανάκτηση των βαθμολογήσεων με τον μεγαλύτερο μέσο όρο σκορ που έχουν δημοσιευτεί

Sorted Set – Διευθύνσεις που έχουν βαθμολογηθεί με κατάταξη τον αριθμό χρηστών που έχουν βαθμολογήσει την κάθε μία

key	members	Περιγραφή - Σχόλια
popularityBasedCharts	Url strings	Χρησιμοποιείται για την ανάκτηση των 'δημοφιλέστερων' πόρων (που έχουν βαθμολογηθεί περισσότερες φορές)

Indexes

SET - ευρετήριο για αναζήτηση βάσει κειμένου

key	Members	Περιγραφή - Σχόλια
word:<keyword>	rids	Περιέχει τους κωδικούς βαθμολογιών που ο τίτλος τους περιέχει τη λέξη-κλειδί (metaphone) που προσδιορίζει το όνομα του set

Παράδειγμα: Ο χρήστης ψάχνει βαθμολογίες που περιέχουν στον τίτλο τους τη λέξη 'νεράιδα'. Το σύστημα ψάχνει για βαθμολογίες που είναι μέλη του σετ με κλειδί το metaphone της λέξης αυτής

```
127.0.0.1:6379> SMEMBERS word:NRT
1) "5"
2) "14"
3) "21"
4) "99"
```

SET - ευρετήριο για αναζήτηση βάσει του σκορ βαθμολογίας

key	Members	Περιγραφή - Σχόλια
score:x	rids	Όπου 'x' το σκορ βαθμολογίας (1-10)

Παράδειγμα: Ανάκτηση κωδικών βαθμολογιών με σκορ 7

```
127.0.0.1:6379> SMEMBERS score:7
1) "1"
2) "9"
3) "51"
4) "55"
5) "83"
6) "97"
7) "107"
8) "111"
```

Σύνοψη

Στο κεφάλαιο αυτό παρουσιάστηκαν τα επίπεδα της αρχιτεκτονικής και ο σχεδιασμός της εφαρμογής που υλοποιήθηκε για την παρούσα εργασία, με χρήση της γλώσσας μοντελοποίησης UML και της διαδικασίας RUP. Παρουσιάστηκε τέλος η διάταξη δεδομένων που χρησιμοποιήθηκε για την αποθήκευση πληροφορίας με χρήση Redis. Στο επόμενο κεφάλαιο παρουσιάζονται τα συμπεράσματα που προέκυψαν από την εκπόνηση της εργασίας αυτής, αναφέρονται ορισμένες λειτουργίες της εφαρμογής οι οποίες δεν μπόρεσαν να υλοποιηθούν, καθώς και προτάσεις για βελτίωση και επέκταση της εφαρμογής.

5 Επίλογος

5.1 Σύνοψη και συμπεράσματα

Στο κεφάλαιο αυτό παρουσιάζονται τα συμπεράσματα που προέκυψαν από την υλοποίηση της παρούσας διπλωματικής εργασίας, γίνεται αναφορά των λειτουργιών που δεν υλοποιήθηκαν στην εφαρμογή, καθώς και των μελλοντικών επεκτάσεων και βελτιώσεων που θα μπορούσαν να υλοποιηθούν σε μία επόμενη έκδοση της εφαρμογής αυτής.

Ο σκοπός της εργασίας ήταν ο σχεδιασμός και η δημιουργία ενός κοινωνικού δικτύου το οποίο αφενός θα ανταποκρινόταν, αρχιτεκτονικά και τεχνικά, στα σύγχρονα στάνταρ των μοντέρνων διαδικτυακών εφαρμογών, αφετέρου θα προσέφερε στο χρήστη τη δυνατότητα να εκφράσει, μέσω των δημοσιεύσεων στην πλατφόρμα αυτή, οι οποίες είναι μία βαθμολόγηση ενός πόρου του διαδικτύου (ιστοσελίδες, αρχεία κ.λ.π), την έμφυτη τάση όλων των ανθρώπων να αξιολογούν όλα αυτά με τα οποία έρχονται σε επαφή.

Για τις ανάγκες της εργασίας αυτής πραγματοποιήθηκε, υπό την κατάλληλη καθοδήγηση, μία ενδιαφέρουσα και εποικοδομητική έρευνα πάνω στις σύγχρονες τεχνολογίες του διαδικτύου, με έμφαση στις βέλτιστες-προτεινόμενες πρακτικές (best practicies) που αφορούν τα κοινωνικά δίκτυα. Καθώς η εφαρμογή που υλοποιήθηκε αποτελεί προϊόν συνεργασίας τριών φοιτητών, διερευνήθηκαν τρόποι και πρακτικές που θα βελτιστοποιούσαν την οργάνωση και τελικά την υλοποίση του έργου από μία ομάδα.

Το Git αποτελεί το πιο διαδεδομένο σύστημα ελέγχου εκδόσεων αρχείων, το οποίο σε συνδυασμό με την πλατφόρμα φιλοξενίας και διαχείρισης έργων Bitbucket, καθίσταται απαραίτητο εργαλείο ειδικά για πιο περίπλοκα έργα στα οποία συμμετέχουν περισσότεροι του ενός προγραμματιστές.

Η σχεδιαστική αρχή του Διαχωρισμού Εννοιών (Seperation of Concerns), σύμφωνα με την οποία μία πληροφοριακή εφαρμογή πρέπει να χωρίζεται σε διακριτά επιμέρους τμήματα, κάθε ένα απο τα οποία απευθύνεται και αντιμετωπίζει μία ξεχωριστή ανάγκη, οδήγησε στην επιλογή ενός πλαισίου εργασίας βασισμένο στην αρχιτεκτονική Μοντέλου-Παρουσίασης-Ελεγκτή (MVC), η οποία αποτελεί τη δημοφιλέστερη ίσως υλοποίηση της αρχής αυτής. Επιλέχθηκε το Laravel, σαν ένα από τα δημοφιλέστερα, σύγχρονα και αποτελεσματικά πλαίσια εργασίας Php, προσφέροντας περισσότερα χαρακτηριστικά και δυνατότητες σχεδόν απο οποιοδήποτε άλλη εναλλακτική πρόταση.

Η βάση δεδομένων Redis αποτελεί ιδανική και αξιόπιστη λύση για τύπους δεδομένων και λειτουργίες που χαρακτηρίζουν εφαρμογές κοινωνικής δικτύωσης, όπου οι χρήστες, ο όγκος και η κυκλοφορία δεδομένων αυξάνονται συνεχώς, επιβάλλοντας έτσι την ανάγκη κλιμάκωσης και την ταχύτατη επίτευξη εντολών εισόδου – εξόδου.

Το πλαίσο εργασίας διεπαφών Bootstrap αποτελεί μία από τις πιο δημοφιλείς προτάσεις για την εύκολη, γρήγορη και φιλική προς τον χρήστη δημιουργία σελίδων, με έτοιμο σύστημα αποκρίσιμου πλέγματος ώστε να μην χρειάζεται ξεχωριστή υλοποίηση για να υποστηριχθούν συσκευές όπως κινητά ή ταμπλέτες.

Μία από τις ιδιαιτερότητες αυτής της εργασίας ήταν πως ήταν πολυσυμμετοχική. Υπό την άρτια καθοδήγηση του επιβλέποντα, αναπτύχθηκαν οι συνεργατικές δεξιότητες των μελών της ομάδας, αποκομίζοντας πολύτιμα οφέλη τόσο σε προσωπικό επίπεδο, που αφορά τις διαπροσωπικές σχέσεις, όσο και σε τεχνικό επίπεδο, αφού ο σχεδιασμός και η υλοποίηση ενός προσωπικού έργου διαφέρει κατά πολύ απο την οργάνωση, τον διαμοιρασμό εργασιών και τελικά την υλοποίηση ενός έργου από μία ομάδα προγραμματιστών.

5.2 Μελλοντικές βελτιώσεις - επεκτάσεις

Στην εφαρμογή που παρουσιάζεται σε αυτήν την εργασία, υπήρξαν κάποιες λειτουργίες οι οποίες τελικά δεν υλοποιήθηκαν, κυρίως λόγω του εκτεταμένου όγκου τόσο σε χρόνο όσο και σε πολυπλοκότητα που αυτές θα προσέθεταν στο έργο σε επίπεδο έρευνας, σχεδιασμού και υλοποίησης. Μία τέτοια είναι η κατηγοριοποίηση των πόρων που βαθμολογούνται στην εφαρμογή, συσχετίζοντας ετικέτες – κατηγορίες με κάθε διεύθυνση ιστοσελίδας ή αρχείου που εισέρχεται στο σύστημα (tagging). Σε συνδυασμό με το παραπάνω, υπήρχε στα αρχικά στάδια του σχεδιασμού η ιδέα να υλοποιηθεί ένα σύστημα συστάσεων προς τον χρήστη (recommendation system) το οποίο θα πρότεινε είτε πόρους προς βαθμολόγηση, είτε μέλη του κοινωνικού δικτύου που θα μπορούσε να ακολουθεί ο χρήστης. Οι συστάσεις αυτές θα ήταν αποτέλεσμα επεξεργασίας πληροφοριών όπως, μεταξύ άλλων, οι βαθμολογίες που έχει υποβάλει γενικά ο χρήστης, οι βαθμολογίες των χρηστών που ακολουθεί ή οι βαθμολογήσεις που υπέβαλε ο χρήστης σε πόρους τους οποίους συνάντησε στις βαθμολογίες άλλων χρηστών που ακολουθεί. Τέλος, μία λειτουργία η οποία δεν υλοποιήθηκε είναι αυτή ενός συστήματος ανταλλαγής μηνυμάτων μεταξύ των χρηστών (messaging system), καθώς και η δυνατότητα σχολιασμού (commenting) βαθμολογιών και κριτκών άλλων χρηστών.

Όλες οι παραπάνω λειτουργίες που αναφέρθηκαν, αποτελούν μέρος των βελτιώσεων και επεκτάσεων που θα μπρούσαν να υλοποιηθούν στο μέλλον. Μία ακόμα βελτίωση θα ήταν η δυνατότητα εγγραφής και εισόδου ενός χρήστη στο σύστημα με χρήση των στοιχείων λογαριασμού άλλων πλατφορμών, όπως Google+ ή Facebook. Επιπλέον, η αναζήτηση δημοσιεύσεων - βαθμολογιών βάσει κειμένου θα μπορούσε να γίνει ταχύτατη και ακριβέστερη με χρήση του Elastic Search μίας πολύ ισχυρής μηχανής αναζήτησης πλήρους κειμένου. Τέλος, σε ό,τι αφορά το επίπεδο αποθήκευσης δεδομένων, ο καταμερισμός (sharding) και ένα σύστημα δημιουργίας και καταμερισμού αντιγράφων (replication) που προσφέρει η Redis θα ήταν μία απαραίτητη προσθήκη που θα βελτίωνε δραστικά τους χρόνους ανάγνωσης και αποθήκευσης δεδομένων αλλά και την ασφάλεια των αυτών, στοιχεία απαραίτητα σε ιστότοπους υψηλής επισκεψιμότητας και κυκλοφορίας δεδομένων.

6 Βιβλιογραφία

Leiner B. et al., 2009. A Brief History of the Internet. Available at: http://www.cs.ucsb.edu/~almeroth/classes/F10.176A/papers/internet-history-09.pdf

Boyd D., Ellison N., 2007. Social Network Sites: Definition, History and Scholarship. Available at: http://onlinelibrary.wiley.com/doi/10.1111/j.1083-6101.2007.00393.x/full

Howard P. et al., 2011. Opening Closed Regimes: What Was the Role of Social Media During the Arab Spring? Available at: http://papers.ssrn.com/sol3/papers.cfm?abstract_id=2595096

Kosner W. A., 2012. GitHub Is The Next Big Social Network, Powered By What You Do, Not Who You Know. Available at: http://www.forbes.com/sites/anthonykosner/2012/07/15/github-is-the-next-big-social-network-powered-by-what-you-do-not-who-you-know/

Berners-Lee T. et al, 2005. Uniform Resource Identifier (URI): Generic Syntax. Available at: http://www.rfc-editor.org/info/rfc3986

Wikipedia 2015a . Real-time Web https://en.wikipedia.org/wiki/Real-time_web

Otte, S. Version Control Systems. Available at: http://www.mi.fu-berlin.de/inf/groups/ag-tech/teaching/2008-09_WS/S_19565_Proseminar_Technische_Informatik/otte09version.pdf

Loeliger, J., McCullough, M., 2012. Version Control with Git

Rees, D. Codebright The Laravel Framework for Begginers3

Seguin K., 2012. The Little Redis Book

EllisLab. A Brief History Of CodeIgniter. Retrieved from: https://ellislab.com/codeigniter

Arehart, M. 2010. Indexing Methods for Faster and More Effective Person Name Search. Available at: http://www.lrec-conf.org/proceedings/lrec2010/pdf/166_Paper.pdf

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

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
```

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
        ));
    }

}

```

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 }
```

Βιβλιοθήκες

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
```

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",
```

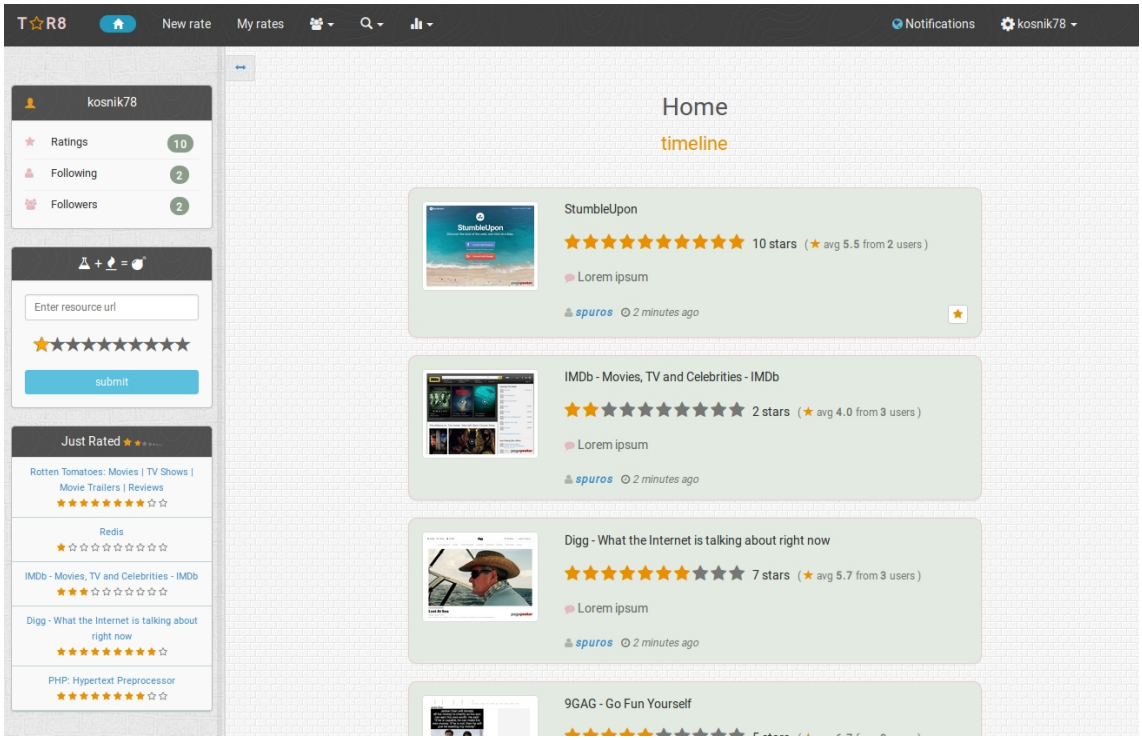
```
82      "m14" => "application/x-msmediaview",
83      "m3u" => "audio/x-mpegurl",
84      "man" => "application/x-troff-man",
85      "mdb" => "application/x-msaccess",
86      "me" => "application/x-troff-me",
87      "mht" => "message/rfc822",
88      "mhtml" => "message/rfc822",
89      "mid" => "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
```

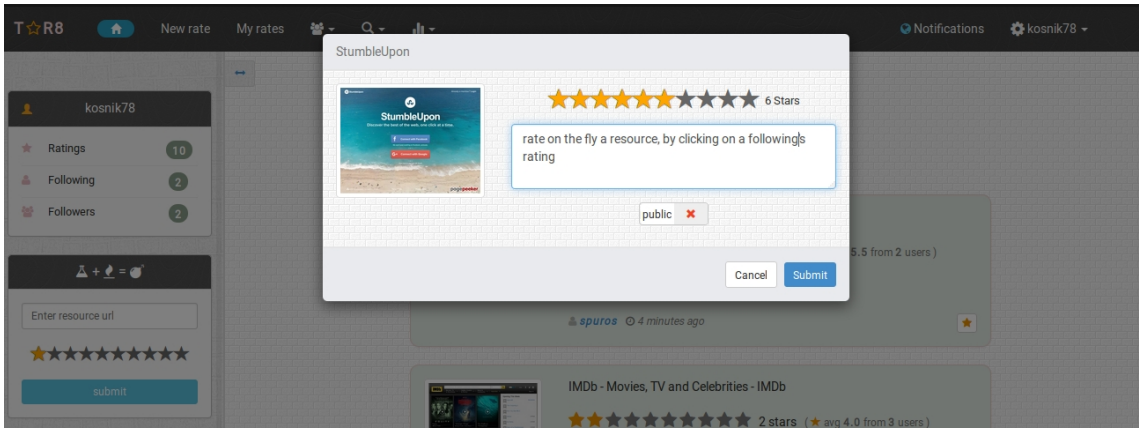
```
252     public function urlGetTitle($url)
253     {
254         $type = $this -> urlType($url);
255         if ($type == 'webpage')
256         {
257             include_once ('simple_html_dom.php');
258             // Create a DOM object
259             $html = new simple_html_dom();
260             // 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

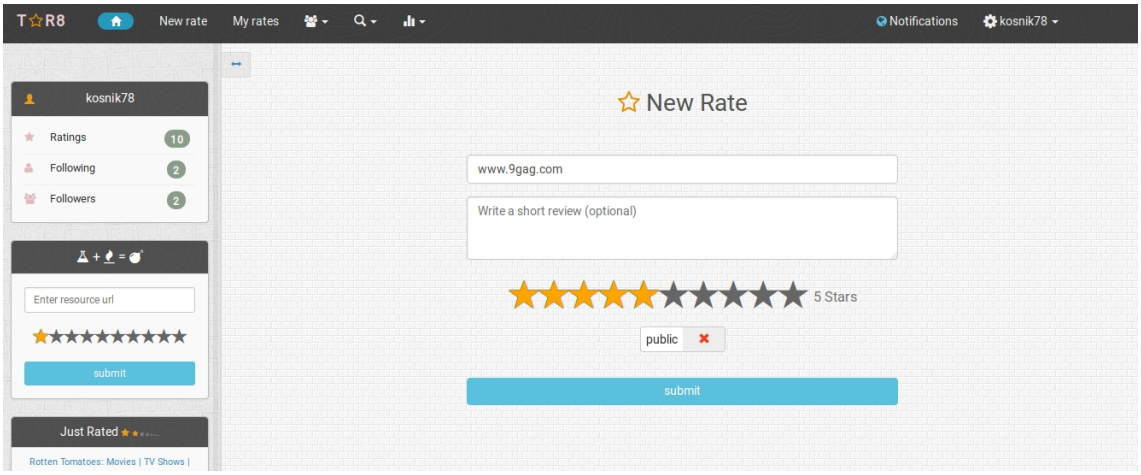
Αρχική οθόνη - Timeline



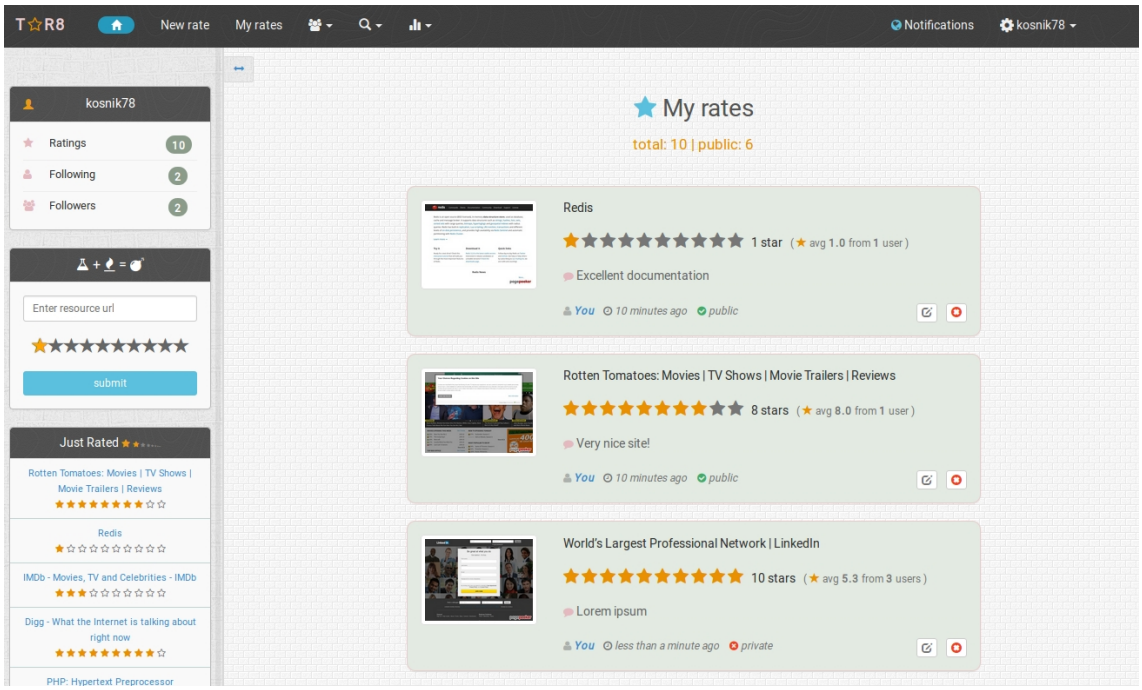
Αξιολόγηση πόρου



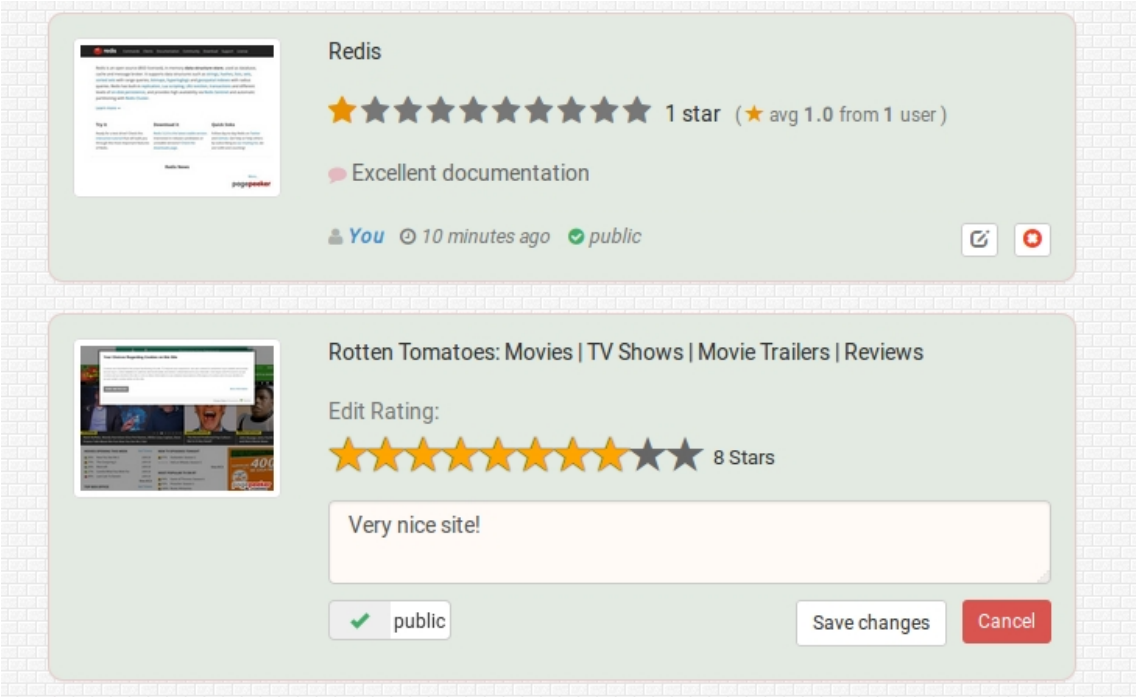
Οθόνη εισαγωγής νέας αξιολόγησης



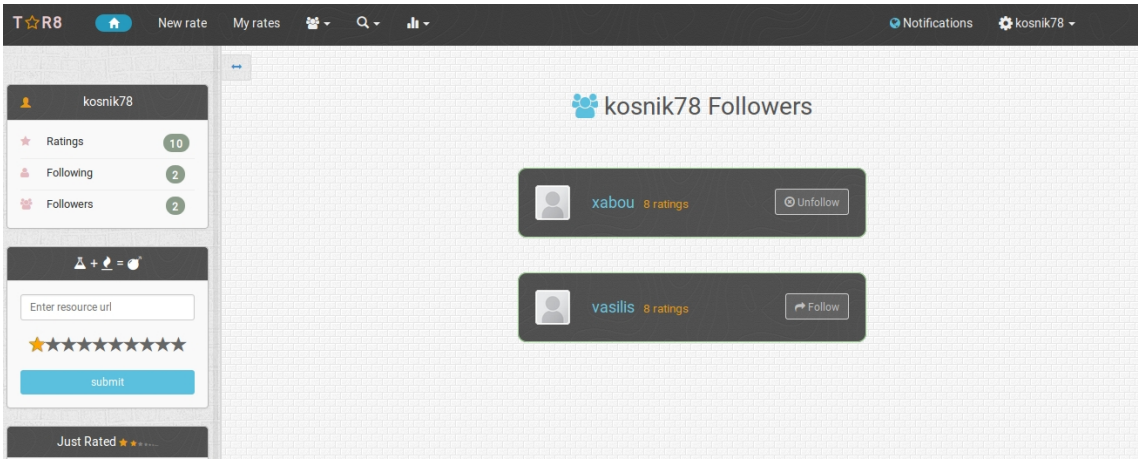
Σελίδα προσωπικών αξιολογήσεων του χρήστη



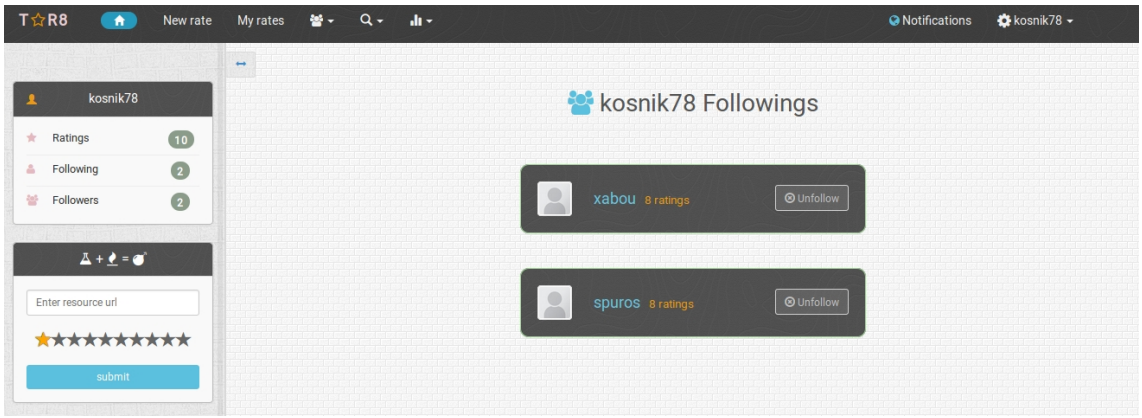
Επεξεργασία αξιολόγησης



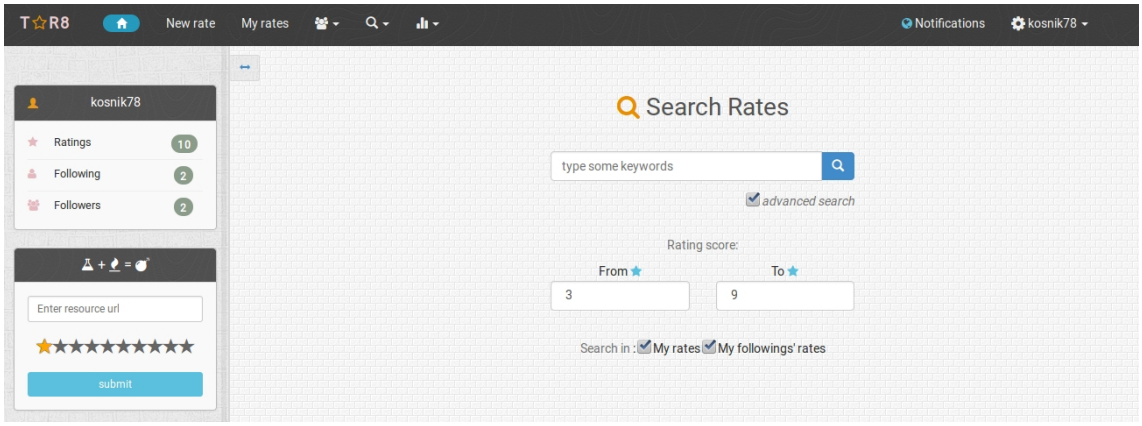
Λίστα ακολούθων του χρήστη



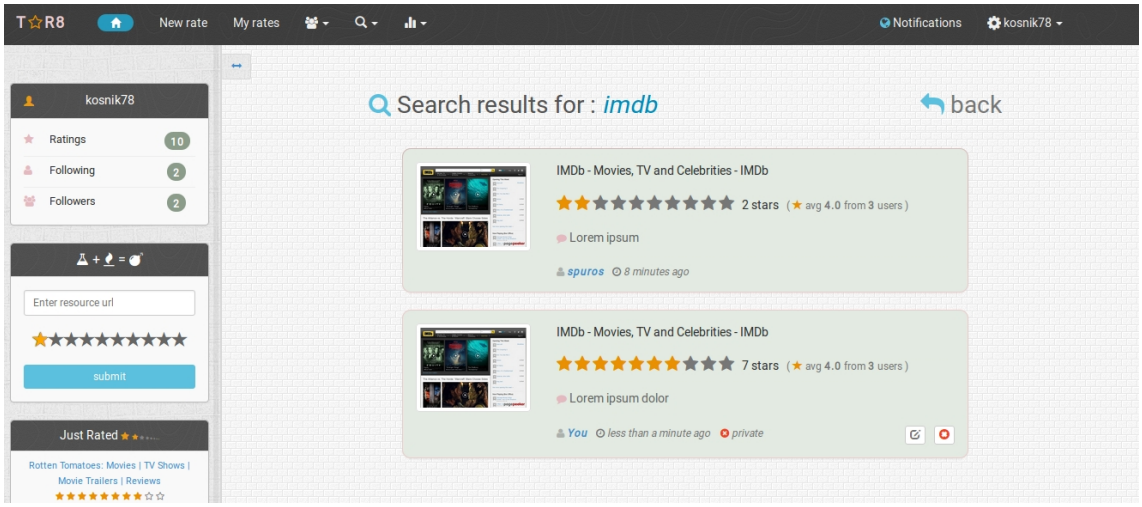
Λίστα χρηστών που ο χρήστης ακολουθεί



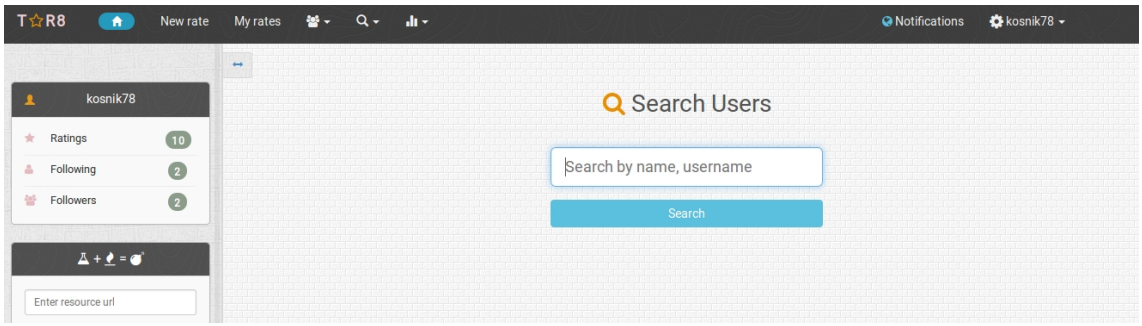
Φόρμα αναζήτησης αξιολογήσεων



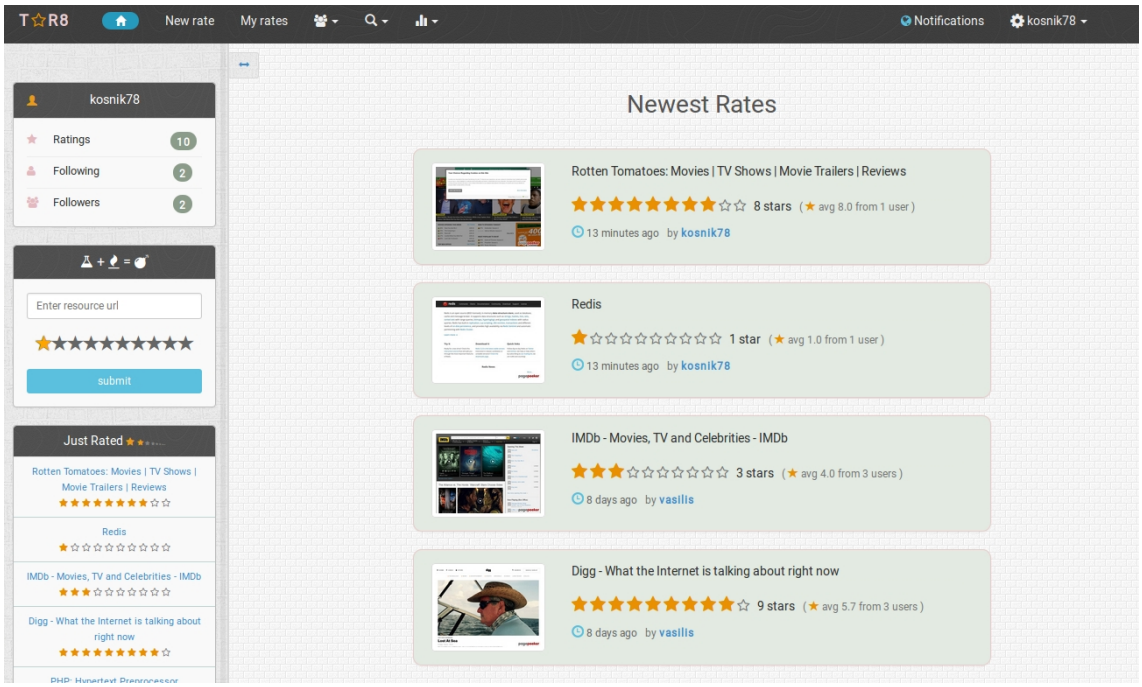
Σελίδα αποτελεσμάτων αναζήτησης αξιολογήσεων



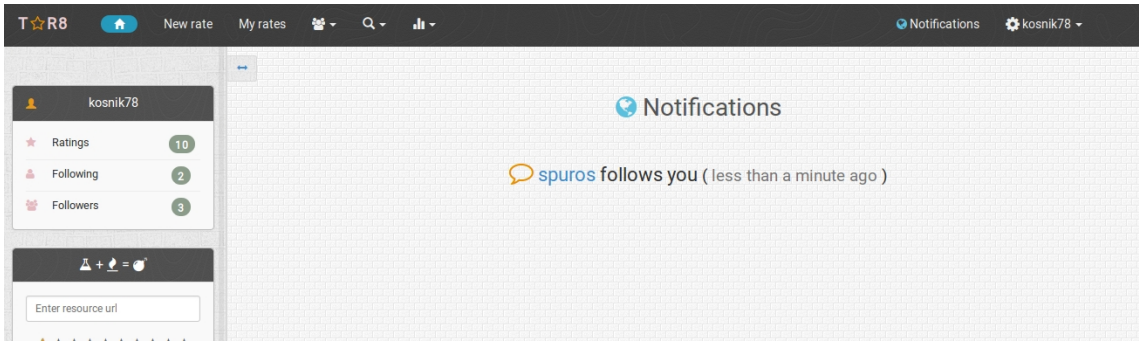
Φόρμα αναζήτησης χρηστών



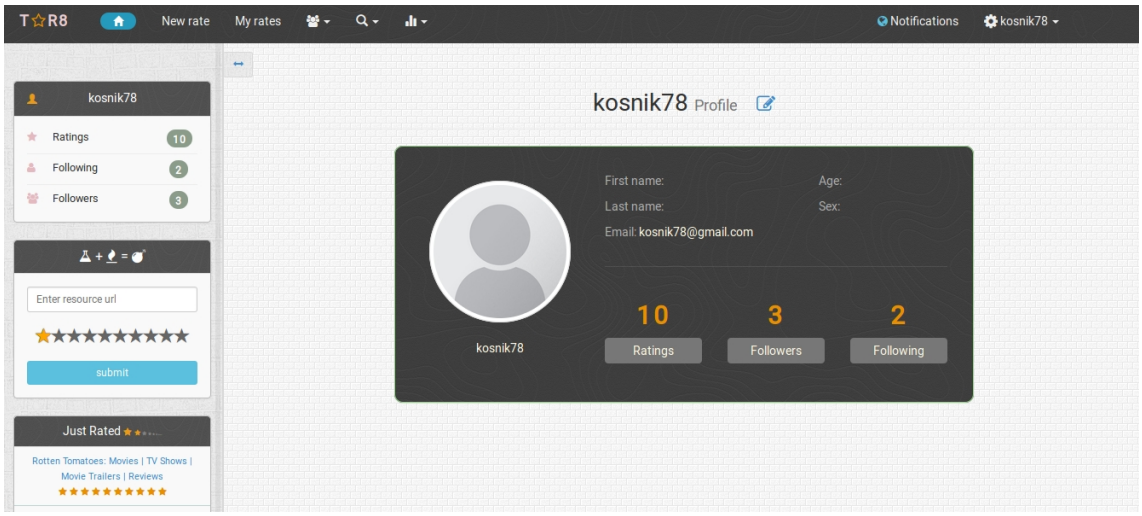
Λίστα πιο πρόσφατων αξιολογήσεων



Λίστα ειδοποιήσεων



Επισκόπηση προφίλ χρήστη



Επεξεργασία προφίλ χρήστη