ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΣΥΣΤΗΜΑΤΩΝ ΜΕΤΑΔΟΣΗΣ ΠΛΗΡΟΦΟΡΙΑΣ
ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΥΛΙΚΩΝ

Σχεδίαση και ανάπτυξη μηχανής αναζήτησης ελληνικών και αγγλικών όρων εφαρμόζοντας πολλαπλές τεχνολογίες βάσεων δεδομένων και πειραματική σύγκριση τεχνικών συγχρονισμού τους

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Φοίβος, Γ. Μπουρίκας

Ραφαήλ, Ν. Ρέτζος

Επιβλέπων : Ιάκωβος Βενιέρης
Καθηγητής ΕΜΠ

Αθήνα, Οκτώβριος 2016

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΣΥΣΤΗΜΑΤΩΝ ΜΕΤΑΔΟΣΗΣ ΠΛΗΡΟΦΟΡΙΑΣ
ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΥΛΙΚΩΝ

Σχεδίαση και ανάπτυξη μηχανής αναζήτησης ελληνικών και αγγλικών όρων εφαρμόζοντας πολλαπλές τεχνολογίες βάσεων δεδομένων και πειραματική σύγκριση τεχνικών συγχρονισμού τους

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Φοίβος, Γ. Μπουρίκας
Ραφαήλ, Ν. Ρέτζος

Επιβλέπων : Ιάκωβος Βενιέρης
Καθηγητής ΕΜΠ

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 24η Οκτωβρίου 2016.

.....	.....	.....
Ι. Βενιέρης	Δ.Θ. Κακλαμάνη	Ν. Ουζούνογλου
Καθηγητής ΕΜΠ	Καθηγήτρια ΕΜΠ	Καθηγητής ΕΜΠ

Αθήνα, Οκτώβριος, 2016

.....
Φοίβος, Γ. Μπουρίκας
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

.....
Ραφαήλ, Ν. Ρέτζος
Διπλωματούχος Ηλεκτρολόγος Μηχανικός και Μηχανικός Υπολογιστών Ε.Μ.Π.

Περίληψη

Οι μηχανές αναζήτησης είναι τις τελευταίες δύο δεκαετίες ένα μεγάλο κομμάτι της ζωής και της καθημερινότητάς μας. Σε συνδυασμό με την εκρηκτική ανάπτυξη των έξυπνων υπολογιστικών συσκευών τα τελευταία χρόνια, αποτελούν πλέον ένα εργαλείο που χρησιμοποιούν οι περισσότεροι άνθρωποι στον κόσμο και βασίζονται πάνω του για πολλές από τις πληροφορίες που χρειάζονται.

Ο σκοπός της παρούσας διπλωματικής εργασίας είναι η σχεδίαση και ανάπτυξη μηχανής αναζήτησης η οποία θα επιτρέπει την χρήση ελληνικών ή λατινικών χαρακτήρων για την πληκτρολόγηση των λέξεων-κλειδιών που χρησιμοποιεί ο χρήστης, αγγίζοντας έτσι τις ανάγκες της σύγχρονης εποχής για μεγαλύτερη ευελιξία στην χρήση της ψηφιακής γλώσσας στην Ελλάδα, ή όπως είναι περισσότερο γνωστή, τα «greeklish». Επιπλέον, θα προσφέρει την δυνατότητα για εκμετάλλευση της γεωγραφικής τοποθεσίας του χρήστη προκειμένου να του παρέχει τα καλύτερα δυνατά αποτελέσματα με βάση την απόστασή του από τα μέρη τα οποία αναζητεί.

Μία ακόμα πτυχή που θα μελετηθεί είναι ο σχεδιασμός του συνολικού συστήματος διαχείρισης των δεδομένων, στο οποίο και θα χρησιμοποιηθούν πολλαπλές τεχνολογίες βάσεων δεδομένων, ώστε να συνδυαστούν χαρακτηριστικά τα οποία είναι σημαντικά για την λειτουργία της μηχανής αναζήτησης, αλλά δεν προσφέρονται από ένα μοναδικό σύστημα. Ταυτόχρονα, θα μελετηθούν τα θέματα που προκύπτουν από μία τέτοια αρχιτεκτονική και πιο συγκεκριμένα ο συγχρονισμός των δεδομένων μεταξύ των διαφορετικών συστημάτων, και θα μετρηθεί πειραματικά η επίδοση των παρεχόμενων εργαλείων για την εξαγωγή συμπερασμάτων σχετικά με τις καταλληλότερες λύσεις.

Τέλος, θα σχεδιασθούν και αναπτυχθούν εφαρμογές για φυλλομετρητή διαδικτύου και για έξυπνα κινητά τηλέφωνα με λειτουργικό σύστημα Android, τα οποία θα δρουν ως αποτύπωση του σεναρίου χρήσης της μηχανής αναζήτησης, και θα προσφέρουν διεπαφή χρήστη για την δοκιμή και παρουσίαση της μηχανής σε πραγματικές συνθήκες.

Λέξεις κλειδιά: Elasticsearch, Android, εφαρμογή, μηχανή αναζήτησης, greeklish, NoSQL βάση δεδομένων, Django, Συγχρονισμός Δεδομένων.

Abstract

Search engines have been a key part of our everyday lives for the last two decades. Combined with the rapid growth of mobile computing devices the last few years, they know present a tool that is being used by the biggest percentage of the world's population and depended upon for most of the information needed.

The purpose of this thesis is the design and development of a search engine that allows the use of Greek and/or Latin characters in typing the search keywords, that way touching the needs of the modern era for higher flexibility in the use of the digital language in Greece, or in the way that it is more known, the "greeklish" language. Furthermore, it will have the capability of utilize the app user's location to provide him with the best possible results based on the distance between him and the places he is searching for.

Another issue that we will study is the design of the complete data management system, which will use multiple database technologies, so it can combine features that are important for the functions of the search engine, but are not provided from a singular system. At the same time, we will study the issues that arise from such an architecture, and specifically the data synchronization between the different systems, and we will measure the performance of the provided tools so we can conclude on the appropriate solutions.

Finally, we will design and develop a web browser application and an Android application that will act as a representation of the use case for the search engine, and will provide a user interface for testing and showing the engine in real time.

Keywords: Elasticsearch, Android, application, search engine, greeklish, NoSQL databases, Django, Data Synchronization.

Ευχαριστίες

Για την εκπόνηση της παρούσας διπλωματικής εργασίας αλλά και γενικότερα για την ολοκλήρωση του κύκλου των σπουδών μας, θέλουμε να ευχαριστήσουμε όλους τους ανθρώπους που μας βοήθησαν.

Αρχικά, θα θέλαμε να ευχαριστήσουμε θερμά τον επιβλέποντα της διπλωματικής εργασίας μας, καθηγητή Ιάκωβο Βενιέρη, ο οποίος μας έδωσε την δυνατότητα να ασχοληθούμε με ένα αντικείμενο που μας ενδιέφερε πραγματικά, αλλά και για την πολύτιμη βοήθειά του κατά τη διάρκεια εκπόνησης της διπλωματικής εργασίας. Επίσης θα θέλαμε να ευχαριστήσουμε τα μέλη της επιτροπής, καθηγήτρια Δήμητρα-Θεοδώρα Κακλαμάνη και τον καθηγητή Νικόλαο Ουζούνογλου, για την σημαντική συμβολή τους στην επιτυχημένη περαίωση της διπλωματικής εργασίας.

Ιδιαιτέρες ευχαριστίες θα θέλαμε να απευθύνουμε στον υποψήφιο Διδάκτορα Εμμανουήλ Καραμανή, χωρίς τη βοήθεια του οποίου θα ήταν αδύνατη η ολοκλήρωση της διπλωματικής εργασίας, μιας και παρείχε υποστήριξη, καθοδήγηση και συμβουλές σε κάθε δυσκολία που αντιμετωπίσαμε. Επίσης θα θέλαμε να ευχαριστήσουμε τον υποψήφιο Διδάκτορα Πέτρο Φλώριο Μπάκαλο, αλλά και όλο το προσωπικό του Εργαστηρίου Ευφυών Επικοινωνιών και Δικτύων Ευρείας Ζώνης που μας προσέφεραν την βοήθεια τους, όποτε αυτή ζητήθηκε.

Τέλος, ο καθένας μας προσωπικά θα θέλαμε να ευχαριστήσουμε τις οικογένειες μας.

Εγώ, ο Φοίβος Μπουρίκας θα ήθελα να ευχαριστήσω τους γονείς μου για την στήριξη που μου παρείχαν καθ’ όλη τη διάρκεια των σπουδών μου.

Εγώ, ο Ραφαήλ Ρέτζος θα ήθελα να ευχαριστήσω τους γονείς και τις δυο αδερφές μου, που με στήριξαν σε όλη τη διάρκεια σπουδών μου.

Πίνακας περιεχομένων

1

Εισαγωγή

1.1 Οι δυσκολίες του ελληνικού αλφαβήτου στον ψηφιακό κόσμο

Με την ραγδαία ανάπτυξη της τεχνολογίας στην εποχή της πληροφορίας, η ψηφιακή γλώσσα έχει εισβάλει στην καθημερινότητα των κοινωνιών παγκοσμίως. Συχνά παρατηρείται λοιπόν η εγγενής δυσκολία των επίσημων γλωσσών να ανταπεξέλθουν στις απαιτήσεις των χρηστών του διαδικτύου. Συγκεκριμένα, η γραφή ελληνικών λέξεων με λατινικούς χαρακτήρες, γνωστή και ως «greeklish», έχει επικρατήσει στην διαδικτυακή επικοινωνία, με αποκορύφωμα την σχεδόν αποκλειστική χρήση τους από την νέα γενιά. Παρά το γεγονός ότι έχουν υπάρξει αντιδράσεις και προβληματισμός για την σκοπιμότητά και τους πιθανούς κινδύνους που προκύπτουν, το φαινόμενο αυτό είναι τελικά διεθνές, με άλλες χώρες των οποίων οι επίσημες γλώσσες δεν χρησιμοποιούν το λατινικό αλφάβητο να αντιμετωπίζουν τις ίδιες δυσκολίες.

Ένας από τους βασικούς λόγους αυτής της εξέλιξης, είναι η αδυναμία των χρηστών να χρησιμοποιήσουν ελληνικές γραμματοσειρές σε χώρες του εξωτερικού, σε υπολογιστές που δεν τους ανήκουν (π.χ. υπολογιστές βιβλιοθηκών, πανεπιστημίων, internet cafe κ.λπ.) και οι οποίοι δε διαθέτουν λογισμικό με ελληνικούς χαρακτήρες. Σε αυτές τις περιπτώσεις καθίσταται υποχρεωτική η χρήση των «greeklish».

Επίσης, η συντριπτική πλειονότητα των προγραμμάτων λογισμικού διεθνώς χρησιμοποιούν διεπαφή χρήστη εξ' ολοκλήρου στην αγγλική, επηρεάζοντας έτσι τους χρήστες, τόσο οπτικά (οπτική συνήθεια) όσο και λεξιλογικά. Γεγονός που έχει συντελέσει στο να αποκτήσει η ψηφιακή αυτή γλώσσα ευρεία αποδοχή στους χρήστες του διαδικτύου και να συνδεθεί αναπόσπαστα με την ηλεκτρονική και διαδικτυακή κουλτούρα .

Προκύπτει λοιπόν η ανάγκη να προσαρμοστούν οι προσφερόμενες διαδικτυακές υπηρεσίες στην νέα αυτή πραγματικότητα, με μηχανισμούς τέτοιους ώστε να παρέχεται η δυνατότητα διαχείρισης

της επικοινωνίας των χρηστών μεταξύ τους αλλά και αμεσότερης αλληλεπίδρασης ανθρώπου – μηχανής.

Συγκεκριμένα στις μηχανές αναζήτησης είναι ιδιαιτέρως σημαντικό να προσφέρεται η δυνατότητα στον χρήστη να επιλέξει την «διάλεκτο» με την οποία επιθυμεί να πραγματοποιήσει μία αναζήτηση, είτε αυτή είναι η κλασσική ελληνική, είτε το ψηφιακό της παράγωγό με λατινικούς χαρακτήρες, είτε ακόμα ακόμα οποιαδήποτε μίξη των δύο. Βλέπουμε έτσι τις μεγάλες εταιρίες ανάπτυξης search engines όπως η Google ή η Yahoo! να έχουν ενσωματώσει μεθόδους μετάφρασης και επεξεργασίας των «greeklish», και να προσφέρουν στον χρήστη «έξυπνους» αλγορίθμους αναγνώρισης χαρακτήρων και λέξεων.

1.2. Ανάγκη για γρήγορη αναζήτηση και ανάλυση σε μεγάλα δεδομένα

Ο όρος «μεγάλα δεδομένα», ή όπως είναι γνωστά διεθνώς «Big Data», χρησιμοποιείται για να περιγράψει σύνολα δεδομένων τα οποία είναι τόσο ογκώδη ή/και περίπλοκα, ώστε οι παραδοσιακές μέθοδοι επεξεργασίας δεδομένων αποδεικνύονται ανεπαρκείς να τα διαχειριστούν αποτελεσματικά. Τα Big Data αποτελούν ένα πάρα πολύ σημαντικό τομέα τεχνολογίας στην εποχή της πληροφορίας, και χρησιμοποιούνται εκτενώς για την άντληση χρήσιμης πληροφορίας μέσω ανάλυσης τους που μπορεί να αφορά επιχειρηματικές τάσεις, επιστημονικές μελέτες, διαφημιστικές στρατηγικές, ιατρικές εφαρμογές και πολλά ακόμα.

Ο όγκος των διαθέσιμων δεδομένων αυξάνεται ραγδαία μέρα με τη μέρα, καθώς συλλέγονται εύκολα πολλές φθηνές κινητές ηλεκτρονικές συσκευές, αισθητήρες, logs λογισμικού, ασύρματα δίκτυα κλπ. Η τεχνολογική χωρητικότητα αποθήκευσης δεδομένων παγκοσμίως έχει πάνω κάτω διπλασιαστεί κάθε 40 μήνες από το 1980, υπολογίζεται ότι κάθε μέρα στην σημερινή εποχή παράγονται 2.5 Exabytes (2.5×10^{18} bytes) δεδομένα.

Τα κλασικά σχεσιακά συστήματα βάσεων δεδομένων συχνά συναντούν δυσκολίες στη διαχείριση των Big Data. Συνήθως απαιτείται μεγάλης κλίμακας παραλληλοποίηση του λογισμικού που τρέχει σε δεκάδες, εκατοντάδες ή ακόμα και χιλιάδες εξυπηρετητές.

Τα Big Data έχουν αυξήσει κατακόρυφα την ζήτηση για επιστήμονες διαχείρισης πληροφοριών παγκοσμίως. Το 2010, η χρηματική αξία της βιομηχανίας λογισμικού που εξειδικεύεται στην διαχείριση και ανάλυση δεδομένων υπολογιζόταν στα 100 εκατομμύρια δολάρια με αυξητική τάση

τουλάχιστον 10% ετησίως. Αυτός ο ρυθμός ανάπτυξης είναι περίπου διπλάσιος από τον αντίστοιχο της βιομηχανίας λογισμικού σαν σύνολο.

Οι ανεπτυγμένες οικονομίες χρησιμοποιούν τεχνολογίες μεγάλων δεδομένων όλο και περισσότερο. Υπολογίζεται ότι υπάρχουν 4.6 δισεκατομμύρια συνδέσεις κινητής τηλεφωνίας παγκοσμίως, και περίπου 1 με 2 δισεκατομμύρια άνθρωποι με πρόσβαση στο διαδίκτυο. Μεταξύ του 1990 και του 2005, περισσότεροι από 1 δισεκατομμύριο άνθρωποι εισήχθησαν στην μέση κοινωνική τάξη, κάτι που σημαίνει ότι περισσότεροι άνθρωποι απέκτησαν μεγαλύτερη πρόσβαση στο διαδίκτυο και την πληροφορία, με αποτέλεσμα να ενισχύουν με τη σειρά τους την παραγωγή της. Οι παγκόσμιες δυνατότητες αποδοτικού διαμοιρασμού πληροφορίας μέσω τηλεπικοινωνιακών δικτύων ήταν 281 Petabytes (2.5×1015 Bytes) το 1986, 471 Petabytes το 1993, 2.2 Exabytes το 2000, 65 Exabytes το 2007, και προβλέψεις τοποθετούν την ποσότητα διαδικτυακής κίνησης για τα επόμενα χρόνια πάνω από 700 Exabytes ετησίως.

1.3. Χρήση NoSQL βάσεων δεδομένων στις σύγχρονες εφαρμογές

Ο όρος NoSQL έχει προκύψει ως η απάντηση στις κλασικές σχεσιακές βάσεις δεδομένων, και πρακτικά ενσωματώνει μία ευρεία ποικιλία από διαφορετικές τεχνολογίες βάσεων δεδομένων οι οποίες έχουν αναπτυχθεί για την εξυπηρέτηση αναγκών των σύγχρονων εφαρμογών.

- Οι προγραμματιστές δουλεύουν με εφαρμογές οι οποίες δημιουργούν και επεξεργάζονται τεράστιους όγκους νέων, ραγδαία μεταβαλλόμενων τύπων δεδομένων – δομημένων, ημι-δομημένων, αδόμητων καθώς και πολυμορφικών δεδομένων.
- Παρωχημένος θεωρείται πλέον ο 12μηνος ως 18μηνος κύκλος ανάπτυξης τύπου «καταρράκτη». Στην σύγχρονη εποχή η ανάπτυξη λογισμικού πολύ συχνά συντελείται σε μικρές, ευέλικτες ομάδες, με ταχεία απόδοση και διαρκείς αναβαθμίσεις κώδικα κάθε βδομάδα ή μήνα, σε μερικές περιπτώσεις ακόμα και πολλές φορές την ίδια μέρα.
- Εφαρμογές που κάποτε εξυπηρετούσαν ένα πεπερασμένο κοινό υλοποιούνται σήμερα ως υπηρεσίες οι οποίες πρέπει να είναι διαρκώς διαθέσιμες και λειτουργικές, σε πολλές και διαφορετικές συσκευές, εξυπηρετώντας εκατομμύρια χρηστών σε παγκόσμια κλίμακα.
- Οι οργανισμοί και οι επιχειρήσεις στρέφονται πλέον σε εύκολα κλιμακώσιμες αρχιτεκτονικές χρησιμοποιώντας λογισμικό ανοιχτού κώδικα, commodity servers και

υπολογιστικά νέφη στη θέση των πρότερων μεγάλων μονολιθικών εξυπηρετητών και υποδομών αποθήκευσης και διαχείρισης δεδομένων.

Η δημιουργία και σχεδίαση των σχεσιακών βάσεων δεδομένων δεν μπορούσε να λάβει υπόψιν της τις ανάγκες αυτές της σημερινής εποχής και αναπόφευκτα αποτυγχάνει να αντιμετωπίσει τις προκλήσεις που περιγράψαμε, και να εξυπηρετήσει τους σκοπούς των σύγχρονων εφαρμογών. Επίσης δεν διαθέτει τρόπους και μεθόδους για να εκμεταλλευτεί την κοινή χρήση πόρων και την διαθέσιμη υπολογιστική ισχύ της σημερινής υποδομής σύγχρονων υπολογιστικών κέντρων.

Οι NoSQL βάσεις δεδομένων περιλαμβάνουν πληθώρα τύπων δεδομένων και μεθόδων αποθήκευσης και διαχείρισης τους. Ενδεικτικά αναφέρουμε τις τέσσερις σημαντικότερες κατηγορίες:

Βάσεις Εγγράφων (Document Databases): κάθε κλειδί συνδέεται με μία πολύπλοκη δομή δεδομένων γνωστή ως έγγραφο (document). Τα έγγραφα μπορούν να περιέχουν πολλά διαφορετικά ζευγάρια κλειδιών-τιμών, ή κλειδιών-πινάκων, ή ακόμα και ένθετων/εμφωλευμένων εγγράφων. Σε αυτή την κατηγορία εντάσσονται οι MongoDB και Elasticsearch.

Γράφοι (Graph Stores): χρησιμοποιούνται για να αποθηκευτεί πληροφορία πάνω σε δίκτυα δεδομένων, όπως για παράδειγμα κοινωνικές συνδέσεις και δίκτυα. Παραδείγματα βάσεων: Neo4J και Giraph.

Αποθήκες Κλειδιού-Τιμής (Key-Value Stores): η απλούστερη μορφή NoSQL βάσεων. Κάθε αντικείμενο στην βάση αποθηκεύεται ως ένα ζεύγος κλειδιού-τιμής, όπου το κλειδί είναι το όνομα του αντικειμένου. Παραδείγματα αυτού του τύπου βάσεων αποτελούν οι Riak και Berkeley DB. Κάποιες από τις βάσεις επιτρέπουν κάθε τιμή να έχει και τύπο δεδομένου, π.χ. «integer» (ακέραιος), το οποίο προσθέτει λειτουργικότητα.

Αποθήκες «Πλατιάς Στήλης» (Wide-Column Stores): όπως οι Cassandra και HBase είναι βελτιστοποιημένες για ερωτήματα πάνω σε μεγάλα σύνολα δεδομένων, και αποθηκεύουν στήλες δεδομένων μαζί, αντί για γραμμές όπως στις κλασσικές σχεσιακές βάσεις.

Τα πλεονεκτήματα χρήσης των NoSQL βάσεων κατά την ανάπτυξη και διάθεση των σύγχρονων εφαρμογών επικεντρώνονται στην καλύτερη απόδοση, την ευέλικτη διαχείριση των δεδομένων, την ανώτερη δυνατότητα κλιμάκωσης σε μεγάλα σύνολα δεδομένων (datasets). Προσφέρουν αποτελεσματική διαχείριση τεράστιων όγκων δομημένων και αδόμητων δεδομένων με μεταβαλλόμενους τύπους, γρήγορη αναζήτηση στη schema της βάσης και ενισχυμένη δυνατότητα συχνών προσθηκών λειτουργικού κώδικα, αντικειμενοστραφή προγραμματισμό που είναι εύκολος

στη χρήση και ευέλικτος, καθώς και γεωγραφικά διανεμημένη κλιμακωτή αρχιτεκτονική στη θέση της παλαιάς, παραδοσιακής, ακριβής μονολιθικής αρχιτεκτονικής.

1.4. Σχεδιασμός σύνθετων συστημάτων και συγχρονισμός δεδομένων

Κάθε βήμα εξέλιξης της τεχνολογίας παρουσιάζει νέα, όλο και πιο σύνθετα προβλήματα, οι λύσεις των οποίων αποτελούν συνεχή πρόκληση. Στον τομέα του προγραμματισμού τα λογισμικά συστήματα καλούνται να παρέχουν συνεχώς αυξανόμενο αριθμό υπηρεσιών, συνδυάζοντας πολλά μικρότερα υποσυστήματα, πάντα συνεργαζόμενα σε αρμονία και με αποτελεσματικότητα. Η αρχιτεκτονική τέτοιων συστημάτων είναι πολύ σημαντικό να καθορίζει επαρκώς τον τρόπο λειτουργίας σε διάφορα σενάρια χρήσης και να πετυχαίνει τους λειτουργικούς στόχους. Ο σχεδιασμός της αρχιτεκτονικής σύνθετων συστημάτων επικεντρώνεται σε δύο κυρίως θέματα. Τον ακριβή προσδιορισμό λειτουργίας των αντίστοιχων υποσυστημάτων και τις μεταξύ τους σχέσεις και αλληλεπιδράσεις.

Συγκεκριμένα σε εφαρμογές όπου κύριο πρόβλημα και ζητούμενο αποτελεί ο συγχρονισμός δεδομένων μεταξύ υποσυστημάτων, ο τρόπος που αλληλοεπιδρούν αυτά αλλά και οι κανόνες που ορίζονται για την λειτουργία τους, αποτελούν την αφετηρία για τις επιλογές σχεδιασμού της συνολικής αρχιτεκτονικής.

1.5. Κρισιμότητα των Geodata στην εποχή του Internet of Things

Καθώς το Internet of Things (IoT) εξαπλώνεται σε πάνω από 21 δις συσκευές μέχρι το 2020, η ποσότητα των γεωγραφικών δεδομένων μαζί με όλα τα άλλα δεδομένα που παράγονται, θα αυξηθεί φυσικά δραματικά. Η κίνηση προς ένα κόσμο με συνδεδεμένα αυτοκίνητα, έξυπνες συσκευές και αισθητήρες θα παράγει αγορές υψηλής αξίας οι οποίες θα επικεντρώνονται στην ανάλυση γεωγραφικών δεδομένων. Εξάλλου το βασικό σκεπτικό του Internet of Things είναι να διασυνδέσει ανθρώπους, συσκευές και δίκτυα πετυχαίνοντας έτσι μεγαλύτερη γνώση και περισσότερες δυνατότητες.

Πρόσθετα, τα τελευταία χρόνια η ανάλυση γεωγραφικών δεδομένων έχει αποκτήσει μεγαλύτερη αξία καθώς βοηθά στην εξαγωγή συμπερασμάτων ως προς τη γεωγραφική θέση, την βελτιστοποίηση ροών εργασίας, μείωσης κόστους, διαχείρισης κινδύνου και της βελτίωσης της εμπειρίας των πελατών. Γεωγραφικά δεδομένα παρέχουν σε οργανισμούς και επιχειρήσεις ευκαιρίες να χρησιμοποιήσουν τις μεγάλες ταχύτητες επεξεργασίας δεδομένων για να εξάγουν διαφοροποιημένες και οξυδερκείς αποφάσεις. Στη συνέχεια παρατίθενται παραδείγματα τομέων όπου τα γεωγραφικά δεδομένα αξιοποιούνται με μεγάλη επιτυχία.

- Χρηματοπιστωτικές Υπηρεσίες: Ο εντοπισμός απάτης είναι πρόβλημα τεράστιας αξίας, που απασχολεί καταναλωτές και τράπεζες. Γεωγραφικά δεδομένα μπορούν να βοηθήσουν σε τέτοιες περιπτώσεις με την συσχέτιση δεδομένων που παράγονται κατά την χρήση υπηρεσιών και την σύγκριση με ιστορικά δεδομένα ή τον εντοπισμό ανωμαλιών.
- Λιανεμπόριο: Στην εποχή όπου τα φυσικά καταστήματα προσπαθούν να συνυπάρξουν με τα συνεχώς αναπτυσσόμενα διαδικτυακά αντίστοιχά τους, τα γεωγραφικά δεδομένα μπορούν να παίξουν το ρόλο του συνδετικού κρίκου που θα προσφέρει την συνολικά καλύτερη εξυπηρέτηση των καταναλωτών με βάση την τοποθεσία τους.
- Ασφάλιση: Η ύπαρξη και η ανάλυση των γεωγραφικών δεδομένων είναι κρίσιμη για τον τομέα των ασφαλίσεων καθώς συχνά διαπιστώνεται κάποια σύνδεση μεταξύ κινδύνου και τοποθεσίας. Οι εταιρίες έτσι μπορούν να εξάγουν προβλέψεις για τους κινδύνους κάθε περίπτωσης και να εκτιμήσουν καλύτερα την πιθανότητα να υπάρξουν οικονομικές ζημίες.

Τελικώς, εύκολα μπορούμε να διακρίνουμε τη κρισιμότητα της άμεσης αξιοποίησης των γεωγραφικών δεδομένων στις περισσότερες εφαρμογές. Έτσι, είναι σημαντικό να αναφερθεί πως οι εξελίξεις στις επιδόσεις τόσο της μνήμης, όσο και της επεξεργαστικής ισχύος των ηλεκτρονικών υπολογιστών θα συμβάλουν σημαντικά στην ανάδειξη νέων τρόπων όπου θα μπορούν να χρησιμοποιηθούν γεωγραφικά δεδομένα.

1.6. Αντικείμενο διπλωματικής εργασίας

Το αντικείμενο της διπλωματικής εργασίας είναι η μελέτη, ο σχεδιασμός και η ανάπτυξη ενός ολοκληρωμένου συστήματος το οποίο θα λειτουργεί ως μηχανή αναζήτησης. Θα παρέχεται η δυνατότητα στο χρήστη να χρησιμοποιήσει λέξεις κλειδιά με ελληνικούς ή λατινικούς χαρακτήρες. Χρησιμοποιούνται επίσης γεωγραφικά δεδομένα για την καλύτερη υποστήριξη αναζητήσεων των χρηστών. Θα υλοποιηθεί διαδικτυακή εφαρμογή καθώς και εφαρμογή στο λειτουργικό σύστημα για έξυπνες συσκευές Android της Google, βασισμένη πάνω σε ένα σενάριο χρήσης της μηχανής αναζήτησης. Στη συνέχεια, θα διερευνηθούν οι καταλληλότεροι τρόποι αποθήκευσης των δεδομένων και αποδοτικής συνεργασίας/συγχρονισμού των διάφορων υποσυστημάτων.

Για την υλοποίηση της μηχανής αναζήτησης θα εκμεταλλευτούμε την παραμετροποιησημότητα που παρέχεται από το σύστημα βάσης δεδομένων Elasticsearch, και θα χρειαστεί να μελετήσουμε τις δυνατότητες εξειδικευμένων ερωτημάτων προς τη βάση, καθώς και τις αδυναμίες που παρατηρούνται στο σύστημα αυτό, σε ότι αφορά την μόνιμη λειτουργία του ως αποθήκη δεδομένων.

Για την υλοποίηση της διαδικτυακής εφαρμογής θα χρησιμοποιήσουμε το σύστημα ανάπτυξης εφαρμογών Django Web Framework, το οποίο είναι ένα μοντέρνο εργαλείο στο web design και αποκτά όλο και μεγαλύτερο κοινό. Το σύστημα αυτό μας παρέχει τη δυνατότητα να δημιουργήσουμε τόσο το frontend της εφαρμογής, όσο και το backend του εξυπηρετητή, βασισμένα στην ίδια φιλοσοφία σχεδίασης.

Η ανάπτυξη της Android εφαρμογής θα συντελεστεί σύμφωνα με τα πρότυπα της Google, χρησιμοποιώντας το επίσημο εργαλείο ανάπτυξης Android Studio. Θα χρειαστεί να εξοικειωθούμε με τα διάφορα πακέτα εργαλείων που προσφέρονται, καθώς και με τις επικρατούσες πρακτικές διεπαφής με το χρήστη.

Για την διαχείριση των δεδομένων θα χρησιμοποιηθεί το σύστημα βάσης δεδομένων MongoDB, το οποίο προσφέρει ασφαλή και αξιόπιστη μόνιμη αποθήκευση, άμεση διαθεσιμότητα, παρέχοντας επίσης δυνατότητες κατακερματισμού και αυτόματης δημιουργίας αντιγράφων ασφαλείας των δεδομένων.

Τέλος, θα πραγματοποιηθεί μελέτη απόδοσης και επίδοσης μεθόδων συγχρονισμού μεταξύ των συστημάτων διαχείρισης των δεδομένων, Elasticsearch και MongoDB, περιλαμβάνοντας υπάρχουσες λύσεις αλλά και ανάπτυξη μίας νέας λύσης ως επίπεδο σύγκρισης αυτών.

1.7. Διάρθρωση της εργασίας

Στα πλαίσια της διπλωματικής εργασίας, προσεγγίσαμε τη δημιουργία του ολοκληρωμένου συστήματος βηματικά, όπως θα παρουσιαστεί στα επόμενα κεφάλαια. Στο δεύτερο κεφάλαιο, παρουσιάζονται οι τεχνολογίες που απαιτούνται για την ανάπτυξη ενός τέτοιου project. Στο τρίτο κεφάλαιο, αναλύονται τα διάφορα θεωρητικά προβλήματα καθώς και η προσέγγιση των λύσεων τους. Στο τέταρτο κεφάλαιο, παρουσιάζεται η αρχιτεκτονική του συστήματος, καθώς και οι εκάστοτε λεπτομέρειες σχεδίασης. Στο πέμπτο κεφάλαιο καταγράφονται οι μέθοδοι υλοποίησης των διαφόρων υποσυστημάτων, και παρουσιάζονται οι μετρήσεις που προέκυψαν από την μελέτη απόδοσης των μεθόδων συγχρονισμού. Τέλος, στο έκτο κεφάλαιο, καταγράφονται συμπεράσματα και πιθανές μελλοντικές επεκτάσεις.

2

Τεχνολογίες

2.1. Elasticsearch

Η Elasticsearch είναι ένα σύστημα διαχείρισης βάσεων δεδομένων (DBMS) με κυρίως σκοπό την αναζήτηση κειμένου. Παρέχει μία κατανεμημένη μηχανή αναζήτησης «πλήρους κειμένου» (full-text search) με δυνατότητα εξυπηρέτησης πολλαπλών χρηστών, μία HTTP διεπαφή και JSON έγγραφα χωρίς schema. Έχει αναπτυχθεί σε Java και είναι λογισμικό ανοιχτού κώδικα κάτω από τους όρους της άδειας χρήσης Apache License. Η Elasticsearch είναι χτισμένη πάνω από τον Apache Lucene, και είναι η δημοφιλέστερη μηχανή αναζήτησης στην βιομηχανία δεδομένων ακολουθούμενη από το Apache Solr, το οποίο είναι επίσης βασισμένο στο Lucene.

Η Elasticsearch μπορεί να χρησιμοποιηθεί για αναζήτηση σε κάθε είδους έγγραφα. Προσφέρει κλιμακώσιμη αναζήτηση, έχει αναζήτηση σχεδόν πραγματικού χρόνου (real-time search) και υποστηρίζει ύπαρξη πολλών χρηστών. Είναι κατανεμημένη, κάτι που σημαίνει ότι τα indexes της (αντίστοιχα με βάση δεδομένων σε παραδοσιακά DMBS) μπορούν να χωριστούν σε shards και κάθε shard να έχει από καθόλου μέχρι πολλά αντίγραφα. Κάθε κόμβος φιλοξενεί ένα ή περισσότερα shards, και λειτουργεί ως συντονιστής ώστε να διαμοιράζει τις ενέργειες στα κατάλληλα shards. Υποστηρίζεται αυτόματη αναπροσαρμογή φόρτου εργασίας μεταξύ των κόμβων.

Το σύστημα αυτό χρησιμοποιεί το εργαλείο Lucene, μία πολύ ισχυρή βιβλιοθήκη ανοιχτού κώδικα για αναζήτηση πλήρους κειμένου (full-text search), και προσπαθεί να διαθέσει όλα τα χαρακτηριστικά του μέσω του JSON και Java API που παρέχει.

Αξιοσημείωτοι χρήστες της Elasticsearch σήμερα είναι οι Adobe, Facebook, Mozilla, Quora, Foursquare, GitHub, Stack Exchange, Netflix και άλλοι.

2.2. MongoDB

Το MongoDB είναι ένα ελεύθερο και ανοιχτού κώδικα σύστημα διαχείρισης βάσεων δεδομένων, το οποίο αποθηκεύει δεδομένα σε μορφή εγγράφων (documents) και είναι ανεξάρτητο πλατφόρμας (cross-platform). Ανήκοντας στην κατηγορία των NoSQL βάσεων δεδομένων, αποφεύγει την παραδοσιακή δομή των σχεσιακών βάσεων με χρήση πινάκων, και χρησιμοποιεί στη θέση τους αρχεία τύπου JSON με δυναμική schema (συγκεκριμένα αρχεία BSON), κάνοντας την ένταξη των δεδομένων σε σύγχρονες εφαρμογές ευκολότερη και γρηγορότερη. Αναπτύσσεται από την MongoDB Inc. και τηρεί τους όρους των αδειών GNU Affero General Public License και Apache Licence.

Κάποια από τα χαρακτηριστικά του MongoDB:

- Ad-hoc queries: υποστηρίζει αναζήτηση ανά πεδίο, κανονικές εκφράσεις και ερωτήματα εύρους.
- Ευρετηριοποίηση: κάθε πεδίο σε ένα BSON έγγραφο μπορεί να ευρετηριοποιηθεί.
- Replica Sets: προσφέρουν υψηλή διαθεσιμότητα καθώς και ασφάλεια, χρησιμοποιώντας δύο ή παραπάνω αντίγραφα των αρχικών δεδομένων.
- Διαμοιρασμός φόρτου (load balancing): χρησιμοποιεί shards για να μπορεί να κλιμακώσει οριζόντια, δηλαδή τα δεδομένα κατανέμονται και χωρίζονται σε εύρη, και στη συνέχεια αποθηκεύονται σε διαφορετικά shards τα οποία μπορούν να βρίσκονται σε διαφορετικούς εξυπηρετητές.
- Δυνατότητα χρήσης MapReduce αλγορίθμου για επεξεργασία δεδομένων σε μεγάλα κομμάτια.
- Αποθήκη αρχείων: κυρίως χρήση του MongoDB που χρησιμοποιεί τις παραπάνω δυνατότητες και λειτουργεί σε κατανεμημένο περιβάλλον μέσω της χρήσης shards.

2.3. Python

Η Python είναι μία ευρέως διαδεδομένη υψηλού επιπέδου, γενικού σκοπού γλώσσα προγραμματισμού, λειτουργεί με interpreter και ανήκει στις δυναμικές γλώσσες προγραμματισμού. Η φιλοσοφία σχεδίασής της δίνει έμφαση στην αναγνωσιμότητα του κώδικα, και το συντακτικό της επιτρέπει στους προγραμματιστές να εκφράζονται χρησιμοποιώντας λιγότερες γραμμές κώδικα

από για παράδειγμα γλώσσες σαν τη C++ ή την Java. Η γλώσσα παρέχει δυνατότητες για παραγωγή καθαρού και σαφή κώδικα σε μικρή ή μεγάλη κλίμακα.

Η Python υποστηρίζει πλήθος προγραμματιστικών μοντέλων, μεταξύ άλλων αντικειμενοστραφή, προστακτικό και συναρτησιακό προγραμματισμό. Χρησιμοποιεί δυναμικό σύστημα τύπων και αυτόματο διαχειριστή μνήμης και έχει μεγάλη ποικιλία βιβλιοθηκών.

Οι interpreters της Python είναι διαθέσιμοι σε πολλά λειτουργικά συστήματα, επιτρέποντας έτσι στον κώδικα Python να τρέχει σε μεγάλη ποικιλία συστημάτων. Χρησιμοποιώντας εργαλεία όπως το Py2exe ή το Pyinstaller, ο κώδικας Python μπορεί να μετατραπεί σε ανεξάρτητα εκτελέσιμα προγράμματα για κάποια από τα πιο δημοφιλή λειτουργικά συστήματα, οπότε λογισμικό γραμμένο στη γλώσσα αυτή μπορεί να διανεμηθεί και να χρησιμοποιηθεί σε τέτοια περιβάλλοντα χωρίς την ανάγκη εγκατάστασης και interpreter στο σύστημα.

Από το 2003, η Python κατατάσσεται με συνέπεια στις πρώτες δέκα πιο δημοφιλείς γλώσσες προγραμματισμού.

2.4. Java

Η Java είναι μία γενικού σκοπού, υψηλού επιπέδου γλώσσα προγραμματισμού, η οποία είναι αντικειμενοστραφής, βασισμένη σε κλάσεις, ειδικά σχεδιασμένη να έχει όσο λιγότερες εξαρτήσεις υλοποίησης (dependencies) είναι δυνατό, και υποστηρίζει ταυτοχρονισμό. Ο μεταγλωττισμένος κώδικάς της μπορεί να τρέξει σε οποιαδήποτε πλατφόρμα υποστηρίζει Java χωρίς ανάγκη για τοπική μεταγλώττιση. Οι εφαρμογές που είναι γραμμένες σε αυτή τη γλώσσα τυπικά μεταγλωττίζονται σε bytecode, ο οποίος μπορεί να τρέξει μόνο σε JVM (Java Virtual Machine) ανεξάρτητα από την αρχιτεκτονική του συστήματος πάνω στο οποίο τρέχει το ίδιο το JVM. Εδώ και πολλά χρόνια η Java είναι μία από τις δημοφιλέστερες και περισσότερο χρησιμοποιούμενες γλώσσες στον κόσμο, συγκεκριμένα για διαδικτυακές υπηρεσίες εξυπηρετητή-πελάτη, με αναφερόμενη ύπαρξη εννιά εκατομμυρίων προγραμματιστών που την χρησιμοποιούν σήμερα. Αρχικά αναπτύχθηκε από τον James Gosling στην εταιρία Sun Microsystems και εκδόθηκε το 1995. Το συντακτικό της είναι επηρεασμένο από τις C και C++, αλλά έχει λιγότερες λειτουργίες χαμηλού επιπέδου από αυτές.

Κατά τη δημιουργία της, αποφασίστηκαν πέντε πρωταρχικοί στόχοι:

1. Πρέπει να είναι «απλή, αντικειμενοστραφής, και γνώριμη»
2. Πρέπει να είναι «ευέλικτη και ασφαλής»
3. Πρέπει να είναι «ανεξάρτητη αρχιτεκτονικής και μεταφέρσιμη»
4. Πρέπει να εκτελείται με «υψηλή απόδοση»
5. Πρέπει να είναι «μεταφρασμένη (interpreted), να υποστηρίζει νήματα εκτέλεσης και να είναι δυναμική»

Η Java είναι μία από τις σημαντικότερες γλώσσες της εποχής μας και χρησιμοποιείται ευρέος στη βιομηχανία λογισμικού.

2.5. Django Web Framework

Το Django είναι ένα ελεύθερο και ανοιχτού κώδικα εργαλείο ανάπτυξης διαδικτυακών εφαρμογών, γραμμένο σε Python, το οποίο ακολουθεί το μοντέλο-όψη-πρότυπο (model-view-template) αρχιτεκτονικό μοτίβο.

Ο πρωταρχικός στόχος του είναι να διευκολύνει την δημιουργία πολύπλοκων ιστοσελίδων που βασίζουν την λειτουργία τους κατά μεγάλο βαθμό στη σύνδεση με βάσεις δεδομένων. Το Django δίνει έμφαση στην συντηρισημότητα των μερών του, καθώς και στην δυνατότητα προσθήκης νέων μερών με ευκολία, στην γρήγορη ανάπτυξη, και στην αρχή της μη επαναληψιμότητας. Η Python χρησιμοποιείται παντού, ακόμα και για αρχεία ρυθμίσεων και μοντέλα δεδομένων. Το Django επίσης προσφέρει προαιρετική δυνατότητα για τη δημιουργία admin interface, με δυνατότητες create, read, update και delete.

Κάποιες πολύ γνωστές ιστοσελίδες που χρησιμοποιούν το Django είναι οι Instagram, Mozilla, The Washington Times, Disqus, Bitbucket και άλλες.

Τα κυρίως χαρακτηριστικά του είναι:

- Ελαφρύς και ανεξάρτητος εξυπηρετητής για τους σκοπούς της ανάπτυξης και του testing.
- Σύστημα σειριοποίησης το οποίο μπορεί να μεταφράζει φόρμες HTML και τιμές κατάλληλες για αποθήκευση στη βάση.
- Σύστημα templates που υλοποιεί το μοντέλο της κληρονομικότητας, δανεισμένο από τον αντικειμενοστραφή προγραμματισμό.
- Σύστημα για caching δεδομένων.

- Εσωτερικό σύστημα αποστολής σημάτων μεταξύ μερών της εφαρμογής για την επικοινωνία γεγονότων μεταξύ τους.
- Σύστημα σειριοποίησης το οποίο μπορεί να παράγει XML ή/και JSON αναπαραστάσεις των μοντέλων δεδομένων του Django.
- Σύστημα επέκτασης των δυνατοτήτων της μηχανής templates.

2.6. Android OS

Το Android είναι το λειτουργικό σύστημα για κινητές έξυπνες συσκευές της Google, το οποίο είναι βασισμένο στον πυρήνα του Linux και σχεδιασμένο πρωταρχικά για οθόνες αφής. Η διεπαφή χρήστη είναι κυρίως βασισμένη σε άμεση απόκριση, χρησιμοποιώντας χειρονομίες αφής (touch gestures) που περίπου αντιστοιχούν σε κινήσεις της πραγματικής ζωής, όπως ολίσθηση, χτύπημα και τσίμπημα για την επεξεργασία αντικειμένων στην οθόνη, και ένα εικονικό πληκτρολόγιο για εισαγωγή κειμένου. Πέρα από τις συσκευές οθόνης αφής το λειτουργικό σύστημα Android χρησιμοποιείται και σε τηλεοράσεις, αυτοκίνητα, ρολόγια, laptops, παιχνιδομηχανές, ψηφιακές φωτογραφικές μηχανές και άλλες συσκευές. Για αυτό το λόγο είναι και το λειτουργικό σύστημα με τον μεγαλύτερο αριθμό συσκευών στο οποίο λειτουργεί.

Αρχικά αναπτύχθηκε από την εταιρία Android, Inc., την οποία η Google αγόρασε το 2005, ενώ το σύστημα Android αποκαλύφθηκε το 2007 ταυτόχρονα με την Open Handset Alliance, με την οποία αρχικά 84 εταιρίες συμφώνησαν να παράγουν συσκευές οι οποίες υποχρεωτικά θα πρέπει να είναι συμβατές με το σύστημα Android, ώστε να προωθηθεί η λογική του ανοιχτού λογισμικού. Σύμφωνα με στοιχεία του 2013, το Google Play Store, η οποία είναι και η βασική πλατφόρμα διάθεσης λογισμικού για τις συσκευές Android, παρέχει στους χρήστες της πάνω από ένα εκατομμύριο εφαρμογές, με πάνω από πενήντα δις εγκαταστάσεις. Το 71% των προγραμματιστών αναπτύσσουν τις εφαρμογές τους σε Android, και το 40% των προγραμματιστών θεωρούν το Android το βασικό λειτουργικό σύστημα πάνω στο οποίο επικεντρώνονται.

Ο πηγαίος κώδικας του Android είναι ανοικτός και δωρεάν διαθέσιμος στο κοινό, παρότι τα δικαιώματα πολλών από τις κύριες εφαρμογές του, ανήκουν στην Google.

2.7. REST API

Οι Representational state transfer (REST) διαδικτυακές εφαρμογές παρέχουν διαλειτουργικότητα (interoperability) μεταξύ συστημάτων στο Internet. Στην ουσία ορίζουν ένα προκαθορισμένο σετ από ατομικές λειτουργίες, σύμφωνα με τις οποίες συστήματα μπορούν να επικοινωνήσουν ή να μοιραστούν πόρους σε μορφή κειμένου. Αρχικά οι πόροι αυτοί αποτελούσαν έγγραφα ή αρχεία ορισμένα από μια διεύθυνση URL, όμως σήμερα συμπεριλαμβάνουν οποιαδήποτε οντότητα μπορεί να αναγνωριστεί, ονομαστεί ή χειριστεί στο Internet. Η μετάδοση τις περισσότερες φορές γίνεται μέσω HTTP, και παρέχουν λειτουργικότητες όπως GET, POST, PUT και DELETE. Η μορφή των αποκρίσεων σε τέτοιες λειτουργίες συνήθως είναι XML, HTML ή JSON. Με αυτούς τους τρόπους τα REST συστήματα στοχεύουν σε γρήγορες αποδόσεις, αξιοπιστία και κλιμακοσιμότητα, εκμεταλλευόμενα την επαναχρησιμοποίηση πόρων που μπορούν να επεξεργαστούν χωρίς να επηρεάζεται το συνολικό σύστημα, ακόμα και αν αυτό βρίσκεται σε λειτουργία.

2.8. JSON

Η μορφή αρχείων JSON είναι ένα πρότυπο ανοιχτού κώδικα που χρησιμοποιεί κείμενο κατανοητό από ανθρώπους για την αποστολή δεδομένων σε ζευγάρια στοιχείων-πληροφορίας (attribute-value). Έχει καταλήξει να είναι ο πιο διαδεδομένος τρόπος αποστολής δεδομένων μεταξύ browsers και servers, αντικαθιστώντας τον προκάτοχο του, το XML.

Είναι ανεξάρτητο γλώσσας προγραμματισμού παρότι αρχικά ξεκίνησε από την γλώσσα JavaScript. Η επέκταση των αρχείων JSON είναι η ".json" ενώ οι βασικοί τύποι που υποστηρίζονται είναι οι number, String, Boolean, Array, Object και null.

3

Ανάλυση

3.1. Απαιτήσεις μηχανής αναζήτησης κειμένου με βάση την Elasticsearch

3.1.1. Full text search

Το κυριότερο χαρακτηριστικό μίας μηχανής αναζήτησης τέτοιου τύπου είναι η υποστήριξη πλήρους κειμένου (full-text search). Αυτό πραγματοποιείται μέσω της δημιουργίας ευρετηρίων πάνω στα αρχεία τα οποία θέλουμε να αναζητήσουμε. Για να γίνει κατανοητός ο τρόπος επεξεργασίας των δεδομένων που απαιτείται ώστε να υπάρχει στη συνέχεια η δυνατότητα γρήγορης εύρεσης αποτελεσμάτων, μπορεί κανείς να αναλογιστεί τα ευρετήρια που βρίσκονται στο τέλος πολλών επιστημονικών βιβλίων. Το ευρετήριο σε αυτή την περίπτωση είναι χρήσιμο στην περίπτωση που ο αναγνώστης θέλει να χρησιμοποιήσει το βιβλίο σαν αναφορά. Για παράδειγμα, σε ένα βιβλίο για λειτουργικά συστήματα, θέλει να διαβάσει τις παραγράφους που αναφέρουν την λέξη «χρονοδρομολογητής». Μπορεί λοιπόν να ανατρέξει στο ευρετήριο στο τέλος του βιβλίου και με βάση τη λέξη, να εντοπίσει τις σελίδες του βιβλίου που είναι σχετικές.

Στο ευρετήριο λοιπόν, επικεντρωνόμαστε στις λέξεις αντί για το σύνολο του κειμένου. Αυτή είναι και η ιδέα πίσω από τις μηχανές αναζήτησης. Στην διαδικασία δημιουργίας ευρετηρίου, η οποία βρίσκεται στην καρδιά κάθε μηχανής αναζήτησης, ανατρέπεται ουσιαστικά η σειρά ανάγνωσης, δηλαδή αντί να κοιτάμε ολόκληρο το έγγραφο, κοιτάμε τις λέξεις του συγκεκριμένα, και τις αντιστοιχίζουμε με το έγγραφο σε κατάλληλες εγγραφές, ώστε να έχουμε στη συνέχεια τη δυνατότητα να εντοπίσουμε το έγγραφο στο οποίο βρίσκεται κάποια από αυτές τις λέξεις.

Η διαδικασία αυτή μας επιτρέπει τελικά, την στιγμή της αναζήτησης, να εντοπίσουμε γρήγορα και εύκολα την λέξη/φράση που επιθυμούμε, χωρίς να χρειαστεί να ανατρέξουμε το κείμενο εκατομμυρίων εγγράφων, έχοντας κάνει όλη τη σημαντική δουλειά κατά την διάρκεια της εισαγωγής των εγγράφων στην βάση δεδομένων της μηχανής. Επιπλέον, μπορούμε έτσι να χρησιμοποιήσουμε εξιδεικευμένες δομές δεδομένων που είναι κατάλληλες για να ενισχύσουν την ταχύτητα και την αποτελεσματικότητα των αναζητήσεων.

Η διαδικασία δημιουργίας ευρετηρίων απαιτεί μεγάλη υπολογιστική ισχύ, και επίσης μία σημαντική ποσότητα λειτουργιών I/O (input/output). Όμως, αυτό σημαίνει ότι το κυρίως κομμάτι της δουλειάς γίνεται μία φορά στην καταχώρηση και στη συνέχεια οι αναζητήσεις απαιτούν αμελητέα υπολογιστική ισχύ και γίνονται πάρα πολύ γρήγορα, ιδιαίτερα όταν είναι δυνατόν να αποθηκεύσουμε ολόκληρο το ευρετήριο στην μνήμη.

Η λειτουργικότητα που περιεγράφηκε παραπάνω, προσφέρεται συνολικά σε εξαιρετική υλοποίηση στην βιβλιοθήκη ανοιχτού κώδικα Apache Lucene, η οποία αποτελεί μία βιβλιοθήκη μηχανής αναζήτησης κειμένου πολύ υψηλής απόδοσης, με πλήρη υποστήριξη, και είναι γραμμένη εξ' ολοκλήρου σε Java. Είναι μία τεχνολογία κατάλληλη για σχεδόν οποιαδήποτε εφαρμογή που απαιτεί πλήρη αναζήτηση κειμένου.

3.1.2. Υποστήριξη Αναζήτησης με Greeklish και γεωγραφικά κριτήρια

Με βάση την ανάλυση που πραγματοποιήσαμε στην προηγούμενη υπό-ενότητα, καθίσταται σαφές ότι οι λέξεις που δημιουργούνται είναι απόλυτες, και οι αναζητήσεις στερούνται ευελιξίας. Δημιουργείται λοιπόν η ανάγκη να αλλάξουμε τον τρόπο με τον οποίο δημιουργείται το ευρετήριο. Συγκεκριμένα, θα πρέπει να έχουμε τη δυνατότητα οι λέξεις που εισάγονται στο ευρετήριο, να αναλύονται με τρόπο τέτοιο, ώστε να μπορούν να αντιστοιχιστούν οι χαρακτήρες τους τόσο με τους ελληνικούς όσο και με τους αντίστοιχους λατινικούς. Έτσι, την ώρα της αναζήτησης θα μπορούμε να χρησιμοποιήσουμε οποιοδήποτε συνδυασμό ελληνικών και λατινικών χαρακτήρων και το ευρετήριο να αναγνωρίσει την κατάλληλη λέξη.

Επιπλέον, θέλουμε το σύστημα να υποστηρίζει τη λειτουργικότητα γεωγραφικής αναζήτησης βάσει γεωγραφικών δεδομένων που περιλαμβάνονται στα έγγραφα.

Τέλος, σημαντικό κριτήριο αποτελεί η δυνατότητα κλιμάκωσης του συστήματος για πολύ μεγάλους όγκους δεδομένων ή πολύ μεγάλο αριθμό ταυτόχρονων αιτημάτων. Γρήγορα γίνεται αντιληπτό ότι η χρήση ενός μονολιθικού εξυπηρετητή που «τρέχει» σε ένα μόνο μηχάνημα με περιορισμένους

υπολογιστικούς πόρους και χωρίς δυνατότητες επέκτασης ανάλογα με τις μελλοντικές ανάγκες του συστήματος, δεν κρίνεται επαρκής. Η προσθήκη νέων κόμβων επεξεργασίας και αποθήκευσης δεδομένων καθίσταται ιδιαίτερα περίπλοκη σε αυτή την αρχιτεκτονική.

Όλα αυτά τα δευτερογενή προβλήματα καλείται να λύσει η τεχνολογία Elasticsearch, η οποία υλοποιείται πάνω από το Apache Lucene και προσφέρει εξαιρετικές δυνατότητες κλιμάκωσης και τροποποιήσιμους αναλυτές κειμένου.

3.1.3. Δυνατότητες της μηχανής αναζήτησης με χρήση της Elasticsearch

Σε αυτή την ενότητα, θα περιγράψουμε περιληπτικά την επίλυση των προβλημάτων που παρατέθηκαν παραπάνω, και θα αναλύσουμε τα πλεονεκτήματα της χρήσης της τεχνολογίας Elasticsearch για την ανάπτυξη μηχανής αναζήτησης κειμένου.

- Παραμετροποιήσιμοι αναλυτές κειμένου: Προσφέρεται η δυνατότητα επιλογής του τρόπου με τον οποίο η μηχανή θα επεξεργάζεται τις λέξεις και τα σύμβολα κατά την διαδικασία εισαγωγής τους στο ευρετήριο. Αυτό επιτρέπει την υποστήριξη διαφορετικών γλωσσών στο ίδιο ευρετήριο, και άρα επιλύει το πρόβλημα των Greeklish.
- Geodata: Υποστηρίζεται η γεωγραφική ανάλυση δεδομένων, μέσω των ειδικών πεδίων εγγραφών «Geodata» που παρέχει η Elasticsearch, τα οποία λειτουργούν με όρισμα τις συντεταγμένες και προσφέρουν υπηρεσίες γεωγραφικής αναζήτησης.
- Κλιμάκωση: Κάθε εξυπηρετητής μπορεί να κρατάει ένα ή περισσότερο κομμάτια του ευρετηρίου, και όποτε προστίθενται νέοι κόμβοι στο σύμπλεγμα (cluster), συνδέονται αυτόματα. Κάθε αναζήτηση συνδυάζει αποτελέσματα από τους κατάλληλους κόμβους, με ταχύτητα και έξυπνους αλγορίθμους που χρησιμοποιούνται από την Elasticsearch. Κάθε τέτοιο κομμάτι ευρετηρίου, καλείται «shard» και υπάρχει η δυνατότητα να μετακινηθεί στο σύμπλεγμα πολύ εύκολα.
- REST API: Όλη η επικοινωνία με την Elasticsearch γίνεται μέσω απλών αιτημάτων σύμφωνα με το διαδικτυακό πρότυπο REST. Οι απαντήσεις καθώς και τα πολύπλοκα αιτήματα είναι πάντα σε μορφή JSON, η οποία είναι αναγνώσιμη τόσο από ανθρώπους όσο και από μηχανές, και ιδιαίτερα απλή.
- Χρήση της βιβλιοθήκης αναζήτησης Lucene: Το Lucene είναι μία εξαιρετική βιβλιοθήκη αναζήτησης, η οποία προσφέρει τεχνολογίες αιχμής και εξελίσσεται ραγδαία, διαθέτοντας μεγάλη κοινότητα χρηστών και προγραμματιστών.

- Υποστήριξη προχωρημένων εργαλείων αναζήτησης: Έτοιμες, χρήσιμες αλλά και πολύπλοκες συναρτήσεις προσφέρονται για την εξειδίκευση της αναζήτησης με βάση της ανάγκες της εφαρμογής, με δυνατότητες για βαθμολόγηση, κατάταξη και φιλτράρισμα των επιστρεφόμενων αποτελεσμάτων.

3.2. Επιλογή MongoDB για αποθήκευση δεδομένων

Παρότι η Elasticsearch μπορεί θεωρητικά να χρησιμοποιηθεί ως η κύρια βάση αποθήκευσης των δεδομένων μίας εφαρμογής, στην πράξη παρουσιάζονται συγκεκριμένα μειονεκτήματα, που υπενθυμίζουν ότι η αρχή σχεδίασής της είναι κυρίως αυτή της μηχανής αναζήτησης και δευτερευόντως της αποθήκης δεδομένων. Έτσι, έχει επικρατήσει η χρήση ξεχωριστού συστήματος αποθήκευσης, με συγχρονισμό μεταξύ των δύο συστημάτων. Η αρχιτεκτονική μίας τέτοιας λύσης θα αναλυθεί στην επόμενη ενότητα.

Το μεγαλύτερο και πιο διαδεδομένο NoSQL σύστημα διαχείρισης δεδομένων-εγγράφων (document type data) είναι το MongoDB. Είναι εύκολο στη χρήση και έχει πολλά πλεονεκτήματα σε σχέση με τις κλασικές σχεσιακές βάσεις. Επιπλέον, ταιριάζει εξαιρετικά με την Elasticsearch, καθώς και οι δύο είναι NoSQL συστήματα, και οι δύο έχουν μη ορισμένη Schema, και άρα η μεταφορά/συγχρονισμός μεταξύ τους είναι άμεσος και εύκολα κατανοητός.

Το MongoDB δείχνει τις δυνατότητές του σε περιπτώσεις που οι σχεσιακές βάσεις δεν είναι η κατάλληλη λύση, όπως σε εφαρμογές με αδόμητα, ημι-δομημένα και πολυμορφικά δεδομένα, καθώς και εφαρμογές με ανάγκη για μεγάλη κλιμακωσιμότητα (scalability).

Κάποιες από τις δυνατότητες του MongoDB οι οποίες μας οδήγησαν στο να το επιλέξουμε ως την κύρια αποθήκη δεδομένων για το σύστημά μας:

- Ισχυρή συνέπεια δεδομένων, εγγυημένη απόδοση των τελευταίων χρονικά αλλαγών στις εγγραφές, με χρήση των replica sets
- Ευελιξία, μοντέλα δεδομένων τα οποία εξυπηρετούν αλλαγές στους τύπους των δεδομένων εύκολα και γρήγορα
- Ασφάλεια, με εισαγωγή πολλών επιπέδων χρηστών με διαφορικά δικαιώματα πάνω στα δεδομένα

3.3. Συγχρονισμός δεδομένων μεταξύ συστημάτων

Η χρήση δυο ξεχωριστών συστημάτων βάσεων δεδομένων προϋποθέτει κάποιο είδος συγχρονισμού μεταξύ των δεδομένων των συστημάτων, έτσι ώστε να επιτυγχάνεται συνέπεια και αξιοπιστία. Πιο συγκεκριμένα, το MongoDB, το οποίο λειτουργεί ως η κύρια αποθήκη δεδομένων, θα είναι το πρώτο σημείο εισαγωγής των νέων δεδομένων και θα αποθηκεύει όλη την πληροφορία που αφορά κάθε εγγραφή. Στο MongoDB δίνεται η δυνατότητα ευελιξίας της μορφής και των πεδίων των εγγραφών (schema), οπότε θα αποθηκεύεται συγκεκριμένα η πληροφορία που παρέχεται στο σύστημα στα κατάλληλα πεδία. Τα δεδομένα αυτά θα πρέπει να μεταφέρονται στο τμήμα του συστήματος που αναλαμβάνει τις υπηρεσίες αναζήτησης και λειτουργεί με Elasticsearch. Εκεί θα μας χρησιμεύσει μόνο ένα μέρος της πληροφορίας, επομένως υπάρχει ανάγκη μετατροπής της schema των δεδομένων κατά την διαδικασία του συγχρονισμού, ώστε να μην αποθηκεύουμε πληροφορίες που δεν χρειάζονται στις αναζητήσεις.

Σε θεωρητικό επίπεδο, ο συγχρονισμός μπορεί να συντελείται με δύο τρόπους.

Αρχικά, η απλούστερη μέθοδος είναι η αντιγραφή των νέων δεδομένων, που έχουν εισαχθεί στην MongoDB, στην Elasticsearch ανά τακτά χρονικά διαστήματα (ημερησίως, εβδομαδιαία κλπ.). Μια τέτοια προσέγγιση προσφέρει το πλεονέκτημα της επιλογής της χρονικής στιγμής στην οποία θα γίνεται ο συγχρονισμός, με αποτέλεσμα να εξοικονομούνται πολύτιμοι υπολογιστικοί πόροι σε περιόδους φόρτου του συστήματος. Έτσι η μηχανή αναζήτησης μπορεί να πετύχει γρηγορότερες ανταποκρίσεις στις ερωτήσεις των χρηστών, αφού διαθέτει όλους τους πόρους σε αυτή τη λειτουργία. Ταυτόχρονα όμως, δεν δίνεται η δυνατότητα άμεσης διάθεσης των εισερχόμενων δεδομένων στους χρήστες έως ότου υπάρξει η ευκαιρία να συντελεστεί ο συγχρονισμός. Ανάλογα το σενάριο χρήσης του συστήματος, και την ποσότητα των εγγραφών σε σχέση με τις αναγνώσεις στην Elasticsearch, επιλέγεται τι από τα δυο είναι σημαντικότερο.

Εναλλακτικά, μπορεί να χρησιμοποιηθεί ένας άμεσος συγχρονισμός κάθε νέας εγγραφής που εισάγεται στην MongoDB, με την Elasticsearch. Παρατηρούμε ότι μια τέτοια υλοποίηση είναι πιο περίπλοκη, καθώς χρειάζεται να εντοπίζονται οι νεοεισαχθέντες εγγραφές και να μεταφέρονται μόνο αυτές. Σε ευθεία αντιπαραβολή με την προηγούμενη μέθοδο, καταναλώνονται υπολογιστικοί πόροι κατά τη διάρκεια του συγχρονισμού, άκριτα, με αποτέλεσμα να υπάρχει πιθανότητα επιβάρυνσης του συστήματος σε περίοδο φόρτου λόγω μεγάλης επισκεψιμότητας. Παρόλα αυτά συνηθίζεται να χρησιμοποιείται αύτη η μέθοδος περισσότερο, καθώς θεωρείται πολύ σημαντική η άμεση διάθεση των νέων δεδομένων στους χρήστες σε υπηρεσίες αναζήτησης.

Αναζητώντας λοιπόν υπάρχουσες λύσεις συγχρονισμού δεδομένων μεταξύ των συστημάτων MongoDB και Elasticsearch, εντοπίσαμε τις εξής τρεις υλοποιήσεις: Elasticsearch Rivers, Compose Transporter, και Mongo-Connector. Θα μελετήσουμε τα χαρακτηριστικά και τις επιδόσεις του κάθε εργαλείου καθώς και θα αναπτύξουμε δική μας υλοποίηση συγχρονισμού σε Python και Java σαν μέτρο σύγκρισης.

3.4. Σενάριο Χρήσης Μηχανής Αναζήτησης

Μια μηχανή αναζήτησης είναι ένα σύστημα ανάκτησης πληροφορίας σχεδιασμένου να βρίσκει δεδομένα τα οποία είναι αποθηκευμένα σε ένα υπολογιστικό σύστημα. Τα αποτελέσματα συνήθως παρουσιάζονται σε λίστα και αποκαλούνται «hits». Η πιο γνωστή και διαδεδομένη χρήση μηχανών αναζήτησης παρατηρείται στις διαδικτυακές search engines. Αντικείμενο αυτών μπορεί να είναι είτε απλά κείμενα, μετα-δεδομένα είτε άλλες μορφές δεδομένων.

Για την καλύτερη ανάδειξη των χαρακτηριστικών της μηχανής αναζήτησής μας, επιλέξαμε το σενάριο χρήσης όπου κάθε εγγραφή αντιστοιχεί σε ένα μέρος (place) η οποία περιέχει πληροφορίες κειμένου και γεωγραφικής τοποθεσίας. Συγκεκριμένα ο χρήστης θα μπορεί να πραγματοποιήσει αναζητήσεις με λέξεις κλειδιά, σε ελληνικούς ή λατινικούς χαρακτήρες, είτε μίγμα αυτών. Θα παρέχεται η δυνατότητα να αναζητηθεί το εκάστοτε μέρος με βάση το όνομα, το είδος ή την διεύθυνσή του. Επίσης, η μηχανή αναζήτησης θα μπορεί να αξιοποιήσει τη γεωγραφική τοποθεσία του χρήστη για πιο εξατομικευμένα αποτελέσματα. Τέλος, μεγάλη σημασία θα δοθεί στην κατάλληλη βαθμολόγηση και κατάταξη των επιστρεφόμενων αποτελεσμάτων με κριτήρια σχετικότητας κειμένου (text relevance), αλλά και με χρήση εξειδικευμένων συναρτήσεων βαθμολόγησης με βάση τη δημοφιλία του κάθε place όπως αυτή προκύπτει από τις κριτικές άλλων χρηστών.

3.5. Υλοποίηση Web και Android εφαρμογής

3.5.1. Απαιτήσεις Web και Android εφαρμογής

Προκειμένου να παρουσιάσουμε την υλοποίηση του σεναρίου χρήσης της εργασίας αυτής σε πραγματικές συνθήκες, υλοποιήσαμε εφαρμογή αναζήτησης σημείων ενδιαφέροντος (places) η οποία λειτουργεί σε φυλλομετρητή ιστού (web browser) και προσφέρει όλες τις απαραίτητες για τον χρήστη λειτουργίες και υπηρεσίες που απαιτούνται από μία μηχανή αναζήτησης.

Πρέπει όμως να λάβουμε υπόψιν μας, ότι τα τελευταία χρόνια, οι κινητές έξυπνες συσκευές επικρατούν όλο και εντονότερα στην καθημερινότητα των ανθρώπων. Προσφέρουν υπηρεσίες όπως επικοινωνία, αναζήτηση πληροφοριών, διασκέδαση και πολλά ακόμα, και όλα αυτά με άμεση πρόσβαση, καθώς η έξυπνη συσκευή βρίσκεται συνήθως στην τσέπη ή την τσάντα μας. Γίνεται λοιπόν προφανές πως για οποιαδήποτε προσφερόμενη υπηρεσία λογισμικού, είναι ιδιαίτερα σημαντικό να αναπτύσσεται και αντίστοιχη υλοποίηση για τέτοιες κινητές συσκευές όπως τα smartphones ή τα tablets. Παράλληλα λοιπόν με την Web εφαρμογή, αναπτύξαμε και εφαρμογή αναζήτησης σημείων ενδιαφέροντος (places) στο λειτουργικό σύστημα Android της Google για κινητές συσκευές.

Οι προδιαγραφές που θα πρέπει να ικανοποιούνται από την εφαρμογή αυτή περιγράφονται παρακάτω:

- Θα πρέπει να δίνεται στον χρήστη δυνατότητα να εισάγει λέξη/λέξεις-κλειδιά για να πραγματοποιηθεί αναζήτηση πάνω σε αυτήν/αυτές.
- Παροχή επιλογής μέσω κουμπιού για την λειτουργία επιπλέον του γεωγραφικού κριτηρίου στα αποτελέσματα. Ο χρήστης να μπορεί έτσι να επιλέξει να του εμφανιστούν κυρίως αποτελέσματα από places τα οποία είναι εύκολα και γρήγορα προσβάσιμα από αυτόν. Αυτή η επιλογή παρέχεται αποκλειστικά στην mobile εφαρμογή, αφού σε αυτήν έχει νόημα να τίθενται γεωγραφικά κριτήρια, αφού ο χρήστης θα την χρησιμοποιεί εν κινήσει.
- Autocomplete και διορθωτική παρέμβαση σε περίπτωση ορθογραφικών λαθών από πλευράς χρήστη.
- Παρουσίαση αποτελεσμάτων με εμφάνιση χρήσιμων στοιχείων πληροφορίας όταν επιλεγεί κάποιο place από τον χρήστη, με εμφάνιση marker σε χάρτη για την τοποθεσία του συγκεκριμένου place.

3.5.2. Σενάριο χρήσης

3.5.2.1. Αναζήτηση – Βασική Ροή Γεγονότων

Χρήστης	**Εφαρμογή**	**Server**
1. Εκκίνηση εφαρμογής. 2. Ο χρήστης κάνει κλικ στην μπάρα αναζήτησης. 3. Ο χρήστης πληκτρολογεί χαρακτήρες.		
	4. Αποστολή του input του χρήστη στο server με κάθε νέα εισαγωγή χαρακτήρα.	
		5. Δημιουργία ερωτήματος (query) auto-complete προς την Elasticsearch και αποστολή του. 6. Επιστροφή αποτελεσμάτων στην εφαρμογή.
	7. Εμφάνιση στην οθόνη drop down λίστας με αποτελέσματα με auto-complete.	
8. Ο χρήστης κάνει κλικ σε μία από τις προτεινόμενες επιλογές της λίστας.		

	9. Αποστολή της επιλογής του χρήστη στον server.	
		10. Εύρεση της συγκεκριμένης εγγραφής στην MongoDB και επιστροφή της στην εφαρμογή.
	11. Εμφάνιση των λεπτομερειών του place που επέλεξε ο χρήστης.	

3.5.2.2. Αναζήτηση – Εναλλακτική Ροή Γεγονότων

Χρήστης	**Εφαρμογή**	**Server**
8. Ο χρήστης πατάει Enter ή κάνει κλικ στο κουμπί «Go» στην μπάρα αναζήτησης.		
	9. Αποστολή του input του χρήστη στο server με ολόκληρη τη λέξη/φράση κλειδί.	
		10. Δημιουργία ερωτήματος (query) προς την Elasticsearch για την λέξη/φράση κλειδί του χρήστη και αποστολή του.

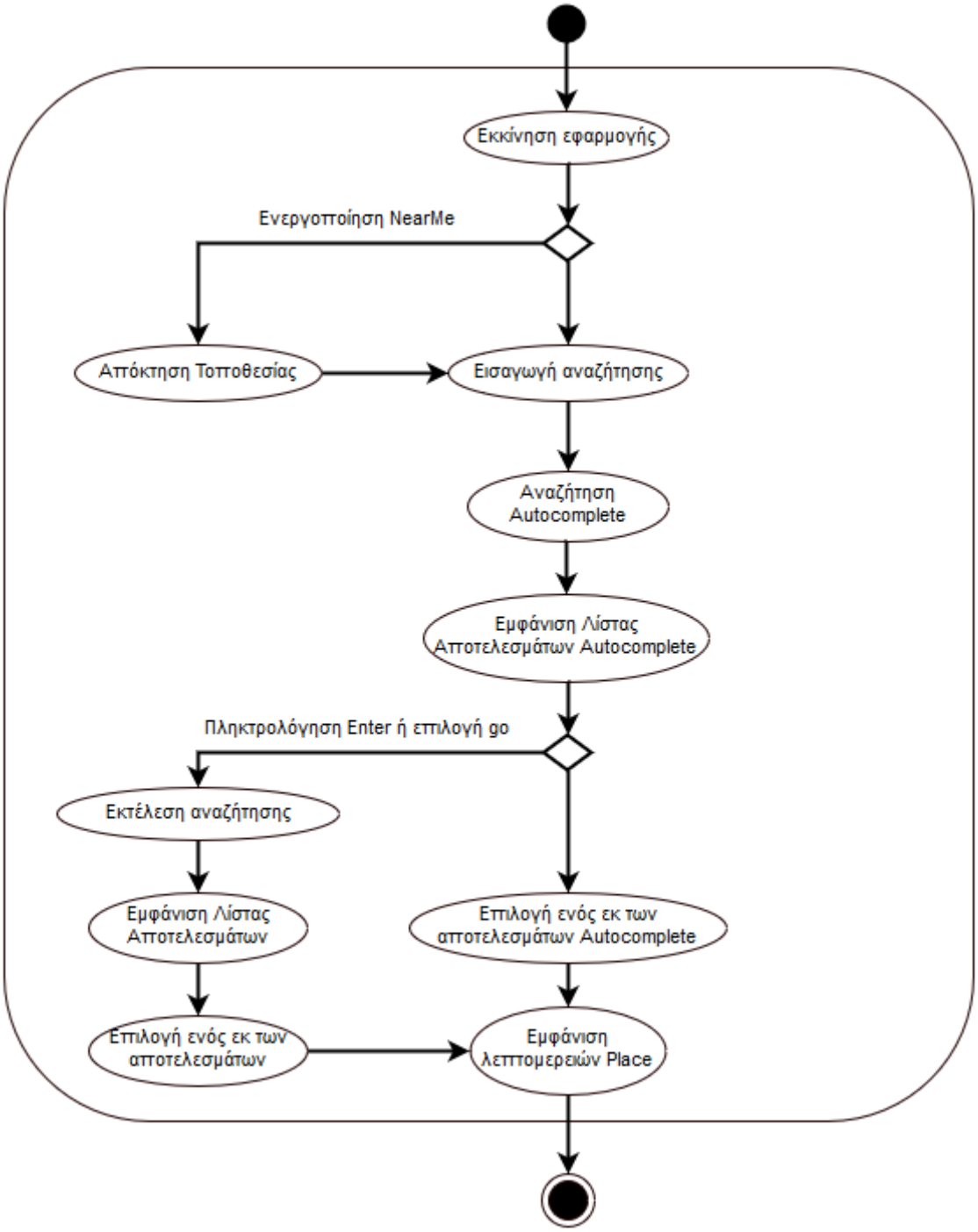
		11. Επιστροφή αποτελεσμάτων στην εφαρμογή.
	12. Εμφάνιση των αποτελεσμάτων σε σελίδες (αν είναι αρκετά ώστε να χρειάζονται πάνω από μία).	
13. Ο χρήστης επιλέγει ένα από τα αποτελέσματα, οπότε και το σενάριο συνεχίζει στην βασική ροή, αριθμός 9. Εναλλακτικά κάνει κλικ σε ένα από τα κουμπιά για αλλαγή σελίδας.		

3.5.2.2. Αναζήτηση – Εναλλακτική Ροή Γεγονότων με επιλογή NearMe για την εφαρμογή Android

Χρήστης	**Εφαρμογή**	**Server**
1. Εκκίνηση εφαρμογής. **2. Ο χρήστης ενεργοποιεί το κουμπί «NearMe»** 3. Ο χρήστης κάνει κλικ στην μπάρα αναζήτησης.		

4. Ο χρήστης πληκτρολογεί χαρακτήρες.		
	. . .	
		14. Δημιουργία ερωτήματος (query) προς την Elasticsearch για την λέξη/φράση κλειδί του χρήστη και αποστολή του, **συμπεριλαμβάνοντας γεωγραφικό κριτήριο κατάταξης**. 15. Επιστροφή αποτελεσμάτων στην εφαρμογή.
	16. Εμφάνιση των αποτελεσμάτων σε σελίδες (αν είναι αρκετά ώστε να χρειάζονται πάνω από μία).	

3.5.2.3. Διάγραμμα Δραστηριοτήτων



4

Αρχιτεκτονική Συστήματος και Σχεδίαση

Η φιλοσοφία σχεδίασης του συστήματος που αναπτύξαμε βασίζεται στις αρχές της αρχιτεκτονικής συστημάτων λογισμικού, και πραγματοποιήθηκε με γνώμονα τις ανάγκες των χρηστών, αλλά και την ευελιξία, αποδοτικότητα και ασφάλεια του συστήματος. Σε αυτό το κεφάλαιο θα περιγράψουμε τη διαδικασία ορισμού της αρχιτεκτονικής, των υποσυστημάτων, των μοντέλων, των διεπαφών και των δεδομένων με σκοπό την ικανοποίηση των απαιτήσεων των συστήματος.

Προτού συνεχίσουμε την περαιτέρω περιγραφή της σχεδίασης, είναι σημαντικό να αναφερθεί ότι το σύστημα μας δεν δημιουργείται εν κενώ, δηλαδή επηρεάζεται από το περιβάλλον στο οποίο δημιουργείται. Οφείλει λοιπόν να αντεπεξέρχεται, όχι μόνο στις αρχικές απαιτήσεις των σχεδιαστών, αλλά και στις ανάγκες των χρηστών. Επίσης, θα πρέπει να λαμβάνεται πρόβλεψη και για την επίλυση προβλημάτων που μπορούν να προκύψουν στο μέλλον, επιπλέον των τωρινών. Η συνολική διαδικασία από την αρχική σχεδίαση ως την τελική υλοποίηση περιλαμβάνει την λεπτομερή εξέταση όλων των σχετικών παραγόντων και τον συνυπολογισμό όλων των απαιτούμενων προδιαγραφών, δημιουργώντας έτσι ένα χρήσιμο σύστημα βασισμένο σε βαθιά τεχνική και αναλυτική κατάρτιση.

Η σχεδίαση αποτελεί ένα σημαντικό βήμα της ανάπτυξης συστήματος, και ξεκινάει αφότου η φάση ανάλυσης έχει ολοκληρωθεί. Είναι σημαντικό να αναφέρουμε ότι το αποτέλεσμα που προκύπτει από την διαδικασία ανάλυσης και οι προδιαγραφές που προκύπτουν από αυτή, χρησιμοποιούνται ως «είσοδος» στην φάση σχεδίασης, το οποίο κατόπιν οδηγεί στην αποτελεσματική υλοποίηση.

4.1. Περιγραφή συνολικής λειτουργίας συστήματος και σχηματική δομή

Η φιλοσοφία σχεδίασης που ακολουθήσαμε είναι αυτή της αρχιτεκτονικής λογισμικού βασισμένου σε υποσυστήματα (component based software architecture). Σε αυτή τη φιλοσοφία, επικεντρωνόμαστε στο διαχωρισμό της σχεδίασης σε ξεχωριστά λειτουργικά και λογικά υποσυστήματα, τα οποία ορίζουν συγκεκριμένα πρότυπα επικοινωνίας και αλληλεπίδρασης μεταξύ τους. Προσφέρει έτσι ένα υψηλότερο επίπεδο αφαίρεσης και χωρίζει το πρόβλημα σε υπο-προβλήματα, το κάθε ένα από τα οποία αντιστοιχίζεται με ένα από τα υποσυστήματα.

Το σύστημα θα διαχωριστεί σε τρία κυρίως υποσυστήματα, τα οποία διαθέτουν ξεχωριστή λειτουργικότητα, στόχευση και μπορούν να θεωρηθούν σχετικώς ανεξάρτητα.

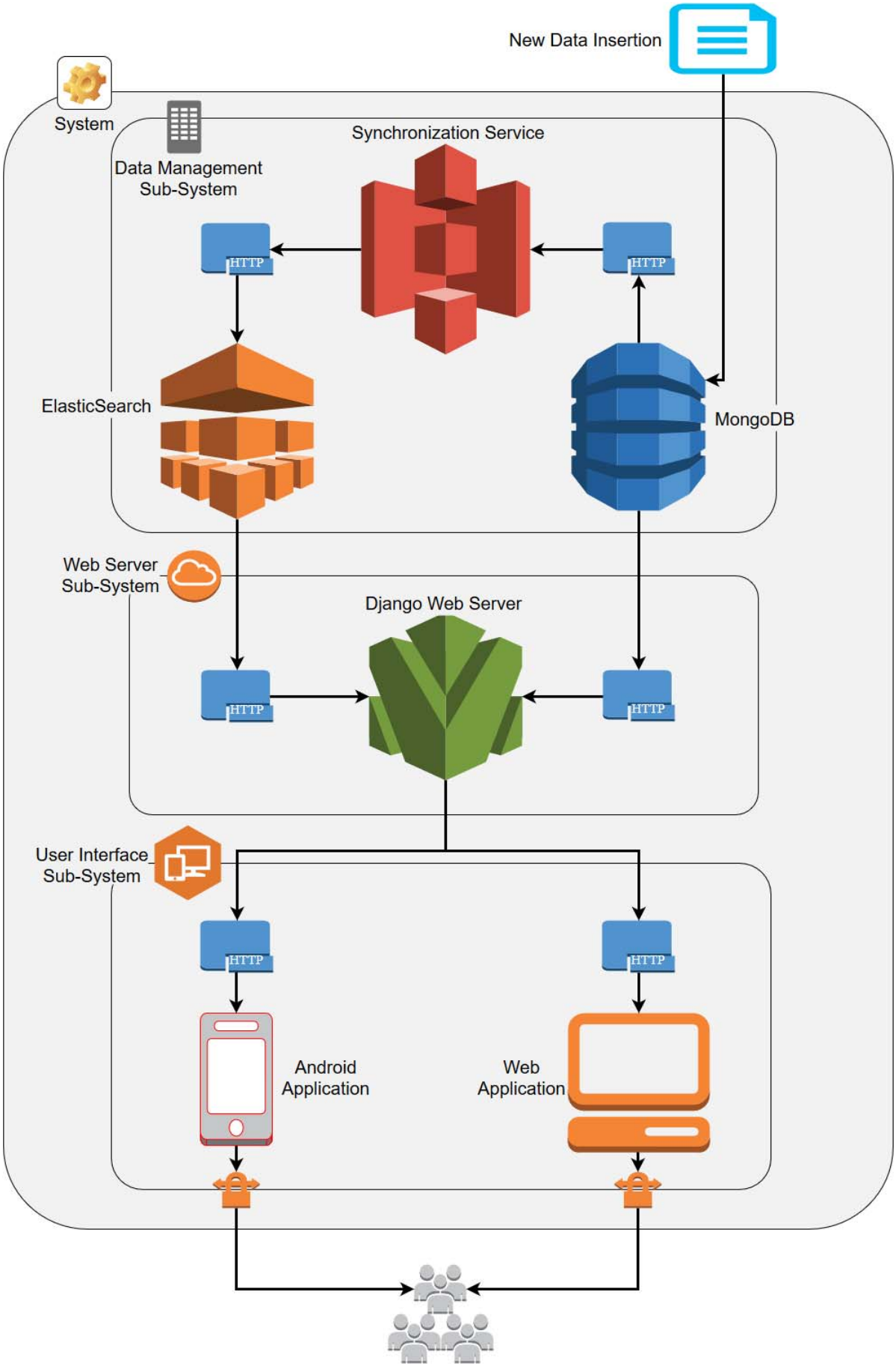
Αρχικά, ορίζεται το υποσύστημα που πραγματοποιεί την διαχείριση των δεδομένων. Σε αυτό περιλαμβάνεται η κυρίως αποθήκη δεδομένων που χρησιμοποιούμε, με χρήση του NoSQL συστήματος βάσης δεδομένων MongoDB, το σύστημα αναζήτησης δεδομένων σε δευτερεύουσα βάση, με χρήση του συστήματος Elasticsearch, καθώς και η υπηρεσία συγχρονισμού των δύο, με χρήση μίας επιλεγμένης λύσης από αυτές που θα μελετήσουμε στο αντίστοιχο υπο-κεφάλαιο. Η επικοινωνία και αλληλεπίδραση μεταξύ των εξαρτημάτων αυτών πραγματοποιείται με αιτήματα του πρωτοκόλλου HTTP, και η ροή της πληροφορίας πραγματοποιείται ως εξής:

- Η κυρίως πηγή των δεδομένων εδράζεται στη βάση που βρίσκεται στο MongoDB. Όταν υπάρχουν νέα δεδομένα εισέρχονται απευθείας σε αυτή. Παράλληλα, προσφέρεται κατάλληλη διεπαφή για ανάκτηση της πλήρους πληροφορίας κάθε εγγραφής μετά από αίτημα του επόμενου συστήματος που πραγματοποιεί τις αναζητήσεις.
- Η υπηρεσία συγχρονισμού παρακολουθεί την κατάσταση (state) της παραπάνω βάσης και μεταφέρει τις νέες εγγραφές στο ευρετήριο (index) της Elasticsearch όταν εισαχθούν νέα δεδομένα, τροποποιώντας την μορφή τους (schema), με κατάλληλο τρόπο ώστε να διατηρείται η ομοιομορφία των δεδομένων στο index και να συγκρατούνται μόνο τα χρήσιμα για τις αναζητήσεις πεδία.
- Το ευρετήριο που κρατείται στην Elasticsearch παραμένει ενημερωμένο και προσφέρει την κατάλληλη διεπαφή στο επόμενο σύστημα για πολύπλοκες και γρήγορες αναζητήσεις στα δεδομένα με βάση τις προδιαγραφές που έχουν τεθεί.

Στη συνέχεια, ορίζεται το υποσύστημα που πραγματοποιεί την διαχείριση των λειτουργιών που προσφέρονται στον τελικό χρήστη και αναλαμβάνει την επικοινωνία των τελικών εφαρμογών με το υποσύστημα διαχείρισης των δεδομένων. Το υπο-σύστημα αυτό αποτελείται από τον εξυπηρετητή διαδικτύου (Web Server) και τις προσφερόμενες μεθόδους διεπαφής με αυτόν, τόσο από την πλευρά των εφαρμογών χρήστη όσο και από την πλευρά του υποσυστήματος διαχείρισης των δεδομένων. Ο πυρήνας του εξυπηρετητή έχει υλοποιηθεί πάνω σε τεχνολογία Django Web Framework, η οποία προσφέρει ασφάλεια επικοινωνίας και ισχυρή απόδοση, καθώς και συντηρισημότητα και επεκτασιμότητα υπηρεσιών. Η αλληλεπίδραση του εξυπηρετητή με το υποσύστημα δεδομένων πραγματοποιείται μέσω αιτημάτων του μέσω του πρωτοκόλλου HTTP στην βάση στο MongoDB ή στο index στην Elasticsearch, ανάλογα με την υπηρεσία που καλείται να προσφέρει στον χρήστη, τα οποία λαμβάνουν απάντηση μέσω του ιδίου πρωτοκόλλου. Η αλληλεπίδραση με τις εφαρμογές χρήστη πραγματοποιείται μέσω αιτημάτων HTTP της εκάστοτε εφαρμογής χρήστη προς τον server, τα οποία πληρούν τις προδιαγραφές ενός RESTful API το οποίο έχουμε υλοποιήσει και απαντώνται από τον server με κατάλληλο τρόπο και αφού έχει πραγματοποιήσει επεξεργασία του αιτήματος και πιθανώς επικοινωνία με το υποσύστημα δεδομένων.

Τέλος, θεωρούμε το τρίτο υποσύστημα το οποίο αποτελείται από τις υλοποιήσεις των εφαρμογών χρήστη, στο φυλλομετρητή ιστοσελίδων (browser) και στο λειτουργικό σύστημα έξυπνων κινητών συσκευών Android. Αυτές είναι που αναλαμβάνουν και την αλληλεπίδραση με τον χρήστη μέσω κατάλληλων διεπαφών, και μεταφέρουν τα αιτήματά του στα κατώτερα επίπεδα του συνολικού συστήματος για εξυπηρέτηση, ενώ επιστρέφουν και εμφανίζουν τα αποτελέσματα σε μορφή αναγνώσιμη από τον άνθρωπο. Και πάλι χρησιμοποιείται το πρωτόκολλο HTTP για επικοινωνία με τον εξυπηρετητή, τηρώντας τις προδιαγραφές που επιβάλλονται από το RESTful API.

Στην επόμενη σελίδα παρουσιάζεται σχηματικά η περιγραφείσα αρχιτεκτονική συστήματος:


New Data Insertion
System
Data Management
Sub-System
Synchronization Service
HTTP
HTTP
ElasticSearch
MongoDB
Web Server
Sub-System
Django Web Server
HTTP
HTTP
User Interface
Sub-System
HTTP
HTTP
Android
Application
Web
Application

4.2. Web εφαρμογή

4.2.1. Σελίδες της εφαρμογής

Η εφαρμογή που σχεδιάσαμε αποτελείται από ορισμένες οθόνες που λειτουργούν ως το μέσο διεπαφής των χρηστών. Αυτές είναι οι εξής:

- **Αρχική σελίδα**, στην οποία παρουσιάζεται στον χρήστη ένα πλαίσιο εισαγωγής λέξεων ή φράσεων πάνω στις οποίες θα εκτελεστεί η αναζήτηση. Θα περιλαμβάνει επίσης κάποια αρχικά αποτελέσματα ακριβώς από κάτω για γρηγορότερη πλοήγηση. Μετά την εισαγωγή ο χρήστης θα μπορεί να μεταβεί είτε στην σελίδα αποτελεσμάτων, είτε κατ' ευθείαν στην σελίδα λεπτομερειών εφόσον επέλεξε κάποιο από τα αρχικά αποτελέσματα.
- **Σελίδα αποτελεσμάτων**, στην οποία θα παρουσιάζονται σε λίστα τα αποτελέσματα που επέστρεψε η εφαρμογή με βάση τις λέξεις κλειδιά που όρισε ο χρήστης νωρίτερα. Τα αποτελέσματα θα ομαδοποιούνται ανά 15 και θα δίνεται η δυνατότητα εναλλαγής τους, ή επιλογής ενός εκ των αποτελεσμάτων για μετάβαση στην σελίδα λεπτομερειών.
- **Σελίδα λεπτομερειών**, στην οποία θα εμφανίζονται όλες οι σχετικές πληροφορίες με το συγκεκριμένο place, καθώς και χάρτης με την ακριβή του τοποθεσία.

4.2.2. Περιγραφή templates

Όνομα	**Base.html** Το πρότυπο html πάνω στο οποίο χτίζονται όλα τα υπόλοιπα
HTML elements	• **H1** Περιλαμβάνει τον τίτλο της εφαρμογής και αποτελεί υπερσύνδεσμο για την αρχική σελίδα • **div container** Αποτελεί το πλαίσιο μέσα στο οποίο οι επόμενες σελίδες να αναπτύξουν το εκάστοτε περιεχόμενο • **div footer** Περιλαμβάνει συνδέσμους για υλοποιήσεις της εφαρμογής σε συστήματα έξυπνων τηλεφώνων

Όνομα	**Search.html** Η αρχική σελίδα της εφαρμογής
HTML elements	• **form** Αποτελεί το πλαίσιο εισαγωγής για τον χρήστη • **ul** Η λίστα που θα περιλαμβάνει τα αρχικά αποτελέσματα

Όνομα	**results.html** Η σελίδα αποτελεσμάτων της εφαρμογής
HTML elements	• **table** Ο πίνακας που περιέχει τα αποτελέσματα • **div pagination** Η λειτουργία εναλλαγής ομάδων αποτελεσμάτων

Όνομα	**details.html** Η σελίδα λεπτομερειών της εφαρμογής
HTML elements	• **script** Το σενάριο απεικόνισης του χάρτη με την τοποθεσία • **table** Ο πίνακας με τις υπόλοιπες πληροφορίες που αφορούν το place • **li reviews** Η λίστα με τις κριτικές που αφορούν το place

4.3. Android εφαρμογή

4.3.1. Οθόνες της εφαρμογής

Η εφαρμογή που σχεδιάσαμε αποτελείται από ορισμένες οθόνες που λειτουργούν ως το μέσο διεπαφής των χρηστών. Αυτές είναι οι εξής:

- **Αρχική οθόνη**, η οποία παρέχει στον χρήστη πλαίσιο εισαγωγής όρων για την αναζήτηση, λίστα αρχικών αποτελεσμάτων, και μοχλό για την έναρξη της λειτουργίας αναζήτησης με βάση την τοποθεσία του χρήστη
- **Οθόνη αποτελεσμάτων**, στην οποία θα παρουσιάζονται σε λίστα τα αποτελέσματα, το καθένα από τα οποία θα περιέχει κάποιες από τις κυριότερες πληροφορίες
- **Οθόνη λεπτομερειών**, στην οποία θα παρουσιάζονται όλες οι πληροφορίες για το επιλεγμένο place, και χάρτης με την τοποθεσία του

4.3.2. Περιγραφή κλάσεων και αλληλεπιδράσεων

4.3.2.1. Διαγράμματα κλάσεων

<table>
<tr><td>Όνομα Κλάσης</td><td colspan="2">MainActivity</td></tr>
<tr><td>Extends</td><td colspan="2">AppCompatActivity</td></tr>
<tr><td>Implements</td><td colspan="2">ConnectionCallbacks
OnConnectionFailedListener</td></tr>
<tr><td rowspan="6">Μεταβλητές</td><td>listView</td><td>listView</td></tr>
<tr><td>PLAY_SERVICES_RESOLUTION_REQUST</td><td>int</td></tr>
<tr><td>MY_PERMISSION_ACCESS_FINE_LOCATION</td><td>int</td></tr>
<tr><td>MLastLocation</td><td>location</td></tr>
<tr><td>mainCommunicator</td><td>MainCommunicator</td></tr>
<tr><td>mGoogleApiClient</td><td>GoogleApiClient</td></tr>
<tr><td>Μέθοδοι</td><td colspan="2">• onCreate
Καλείται στην δημιουργία της οθόνης και αρχικοποιεί διάφορες λειτουργίες
• buildGoogleApiClient
Δημιουργεί το αντικείμενο google api client
• checkPlayServices
Ελέγχει αν η εφαρμογή έχει συνδεθεί σωστά με τα PlayServices
• onStart
Καλείται στην εκκίνηση της οθόνης και ξεκινά την σύνδεση με τα PlayServices
• onResume
Καλείται στην συνέχιση της οθόνης και καλεί τον έλεγχο της σύνδεσης με τα PlayServices
• onConnectionFailed
Καλείται σε περίπτωση αποτυχίας σύνδεσης</td></tr>
</table>

	• **onConnected** Καλείται όταν γίνει η σύνδεση και αποκτά την τοποθεσία • **onConnectionSuspended** Σε περίπτωση αποσύνδεσης, κάνει την επανασύνδεση • **displayListView** Αναλαμβάνει όλη την λειτουργικότητα των αντικειμένων που αλληλοεπιδρούν με τον χρήστη. • **goToResultsActivity** Ετοιμάζει τις πληροφορίες και στην συνέχεια την μετάβαση προς οθόνη αποτελεσμάτων

Όνομα Κλάσης	**MainCustomAdapter**	
Extends	ArrayAdapter<LiveResult>	
Μεταβλητές	vi	LayoutInflater
Μέθοδοι	• **getView** Ετοιμάζει την παρουσίαση του κάθε αποτελέσματος	

Όνομα Κλάσης	**ResultsActivity**	
Extends	AppCompatActivity	
Μεταβλητές	listView	listView
	dataAdapter	ResultsCustomAdapter
	resultsCommunicator	ResultsCommunicator
	MLastLocation	location
	mGoogleApiClient	GoogleApiClient

Μέθοδοι	• **onCreate** Καλείται στην δημιουργία της οθόνης και αρχικοποιεί διάφορες λειτουργίες • **displayListView** Αναλαμβάνει όλη την λειτουργικότητα των αντικειμένων που αλληλοεπιδρούν με τον χρήστη.

Όνομα Κλάσης	**ResultsCommunicator**	
Extends	HttpCommunicator	
Μέθοδοι	• **processResults** Αναλαμβάνει την μορφοποίηση των πληροφοριών των αποτελεσμάτων για την κατάλληλη προβολή στην οθόνη αποτελεσμάτων	

Όνομα Κλάσης	**ResultsCustomAdapter**	
Extends	ArrayAdapter<Place>	
Μεταβλητές	vi	LayoutInflater
Μέθοδοι	• **getView** Ετοιμάζει την παρουσίαση του κάθε αποτελέσματος	

Όνομα Κλάσης	DetailsActivity
Extends	AppCompatActivity
Implements	OnMapReadyCallback

Μεταβλητές	googleMap	GoogleMap
	detailsCommunicator	DetailsCommunicator
Μέθοδοι	• **onCreate** Καλείται στην δημιουργία της οθόνης και αρχικοποιεί διάφορες λειτουργίες • **onMapReady** Καλείται όταν ο χάρτης έχει δημιουργηθεί και καλεί την παραμετροποίηση των πληροφοριών του • **setUpMap** Ρυθμίζει τις διάφορες παραμέτρους του χάρτη	

Όνομα Κλάσης	**DetailsCommunicator**
Extends	HttpCommunicator
Μέθοδοι	• **processResults** Αναλαμβάνει την μορφοποίηση των πληροφοριών των αποτελεσμάτων για την κατάλληλη προβολή στην οθόνη λεπτομερειών

Όνομα Κλάσης	**LiveResult**	
Μεταβλητές	name	String
	id	String
Μέθοδοι	• **getId** Επιστρέφει την τιμή της μεταβλητής id • **setId** Θέτει την τιμή της μεταβλητής id • **getName** Επιστρέφει την τιμή της μεταβλητής name	

	• **setName** Θέτει την τιμή της μεταβλητής name

Όνομα Κλάσης	**Place**	
Μεταβλητές	name	String
	id	String
	type	String
	rating	Float
	address	String
	phoneNumber	String
Μέθοδοι	• **getId** Επιστρέφει την τιμή της μεταβλητής id • **setId** Θέτει την τιμή της μεταβλητής id • **getName** Επιστρέφει την τιμή της μεταβλητής name • **setName** Θέτει την τιμή της μεταβλητής name • **getType** Επιστρέφει την τιμή της μεταβλητής type • **setType** Θέτει την τιμή της μεταβλητής type • **getRating** Επιστρέφει την τιμή της μεταβλητής rating • **setRating** Θέτει την τιμή της μεταβλητής rating	

	• **getAddress** Επιστρέφει την τιμή της μεταβλητής address • **setAddress** Θέτει την τιμή της μεταβλητής address • **getPhoneNumber** Επιστρέφει την τιμή της μεταβλητής phoneNumber • **setPhoneNumber** Θέτει την τιμή της μεταβλητής phoneNumber

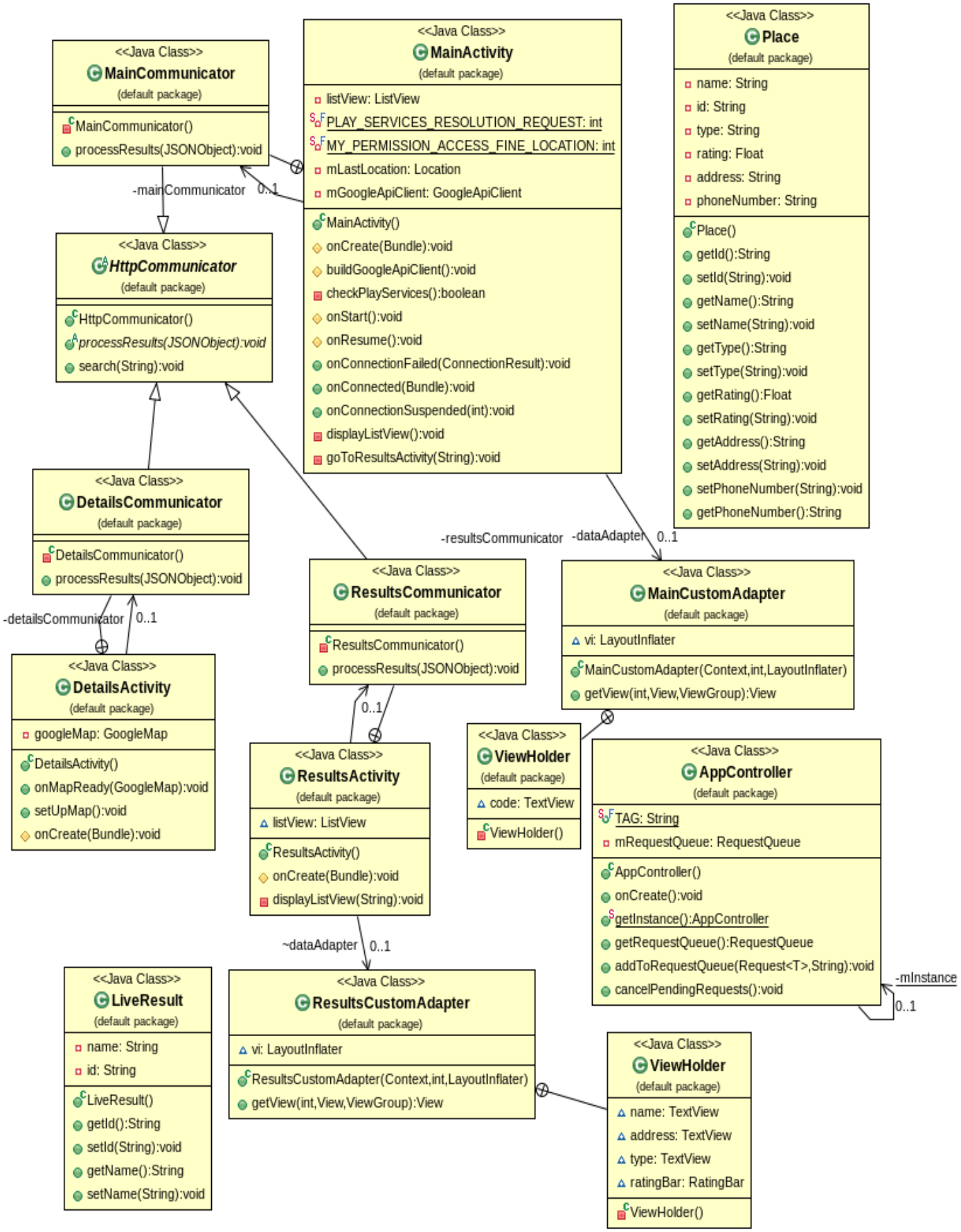
Όνομα Κλάσης	**HttpCommunicator**	
Μεταβλητές	name	String
	id	String
Μέθοδοι	• **processResults** Abstract μέθοδος για την εκάστοτε επεξεργασία των αποτελεσμάτων σε κάθε οθόνη • **search** Μέθοδος επικοινωνίας με τον server	

Όνομα Κλάσης	**AppController**	
Extends	Application	
Μεταβλητές	TAG	String
	mInstance	AppController
	mRequestQueue	RequestQueue

Μέθοδοι	• **onCreate** Καλείται στην δημιουργία της εφαρμογής και αποκτά το reference στο αντικείμενο της κλάσης για μελλοντική χρήση • **getInstance** Επιστρέφει το reference στο αντικείμενο της κλάσης • **getRequestQueue** Δημιουργεί και επιστρέφει την RequestQueue • **addToRequestQueue** Προσθέτει νέα Requests στην RequestQueue • **cancelPendingRequests** Ακυρώνει όλες τις εκκρεμείς Requests

4.3.2.1. Διάγραμμα αλληλεπιδράσεων

Το διάγραμμα αλληλεπιδράσεων παρουσιάζεται στην επόμενη σελίδα.


<<Java Class>>
MainCommunicator
(default package)
MainCommunicator()
processResults(JSONObject):void
<<Java Class>>
MainActivity
(default package)
listView: ListView
PLAY_SERVICES_RESOLUTION_REQUEST: int
MY_PERMISSION_ACCESS_FINE_LOCATION: int
mLastLocation: Location
mGoogleApiClient: GoogleApiClient
MainActivity()
onCreate(Bundle):void
buildGoogleApiClient():void
checkPlayServices():boolean
onStart():void
onResume():void
onConnectionFailed(ConnectionResult):void
onConnected(Bundle):void
onConnectionSuspended(int):void
displayListView():void
goToResultsActivity(String):void
<<Java Class>>
Place
(default package)
name: String
id: String
type: String
rating: Float
address: String
phoneNumber: String
Place()
getId():String
setId(String):void
getName():String
setName(String):void
getType():String
setType(String):void
getRating():Float
setRating(String):void
getAddress():String
setAddress(String):void
setPhoneNumber(String):void
getPhoneNumber():String
-mainCommunicator
0..1
<<Java Class>>
HttpCommunicator
(default package)
HttpCommunicator()
processResults(JSONObject):void
search(String):void
<<Java Class>>
DetailsCommunicator
(default package)
DetailsCommunicator()
processResults(JSONObject):void
-detailsCommunicator
0..1
-resultsCommunicator
-dataAdapter
0..1
<<Java Class>>
ResultsCommunicator
(default package)
ResultsCommunicator()
processResults(JSONObject):void
<<Java Class>>
MainCustomAdapter
(default package)
vi: LayoutInflater
MainCustomAdapter(Context,int,LayoutInflater)
getView(int,View,ViewGroup):View
<<Java Class>>
DetailsActivity
(default package)
googleMap: GoogleMap
DetailsActivity()
onMapReady(GoogleMap):void
setUpMap():void
onCreate(Bundle):void
0..1
<<Java Class>>
ResultsActivity
(default package)
listView: ListView
ResultsActivity()
onCreate(Bundle):void
displayListView(String):void
<<Java Class>>
ViewHolder
(default package)
code: TextView
ViewHolder()
<<Java Class>>
AppController
(default package)
TAG: String
mRequestQueue: RequestQueue
AppController()
onCreate():void
getInstance():AppController
getRequestQueue():RequestQueue
addToRequestQueue(Request<T>,String):void
cancelPendingRequests():void
-mInstance
0..1
~dataAdapter
0..1
<<Java Class>>
LiveResult
(default package)
name: String
id: String
LiveResult()
getId():String
setId(String):void
getName():String
setName(String):void
<<Java Class>>
ResultsCustomAdapter
(default package)
vi: LayoutInflater
ResultsCustomAdapter(Context,int,LayoutInflater)
getView(int,View,ViewGroup):View
<<Java Class>>
ViewHolder
(default package)
name: TextView
address: TextView
type: TextView
ratingBar: RatingBar
ViewHolder()

4.4. Django Web Server

4.4.1. Λειτουργίες του Server

Ο server αποτελεί το συνδετικό υποσύστημα μεταξύ της διεπαφής χρήστη και του υποσυστήματος διαχείρισης δεδομένων, επομένως οι λειτουργίες του είναι ιδιαιτέρως σημαντικές για την ομαλή, ασφαλή, και προσδοκώμενη λειτουργία του συνολικού συστήματος. Αυτές περιγράφονται παρακάτω:

- Προσφέρει ένα RESTful API με το οποίο μπορεί να επικοινωνήσει κάποια εφαρμογή (όπως η Android εφαρμογή στην περίπτωσή μας), προσφέροντας επεκτασιμότητα και τηρώντας τα πρότυπα που κυριαρχούν στο διαδίκτυο σήμερα.

- Παρέχει κατάλληλη μέθοδο αντιστοιχισμένη με ένα συγκεκριμένο σύνδεσμο (link) για την αναζήτηση στην Elasticsearch με σύνθετο query που ματσάρει συγκεκριμένα πεδία και βαθμολογεί τα αποτελέσματα με συγκεκριμένα κριτήρια.

- Παρέχει κατάλληλη μέθοδο αντιστοιχισμένη με ένα συγκεκριμένο σύνδεσμο (link) για την αναζήτηση στην Elasticsearch με auto-complete query με βάση το input του χρήστη.

- Παρέχει κατάλληλη μέθοδο αντιστοιχισμένη με ένα συγκεκριμένο σύνδεσμο (link) για την ανάκτηση της πλήρους εγγραφής με όλα τα πεδία από την βάση δεδομένων στην MongoDB.

4.4.2. Περιγραφή μεθόδων

<table>
<tr><td>Όνομα Αρχείου</td><td colspan="2">views.py</td></tr>
<tr><td rowspan="2">Μεταβλητές</td><td>mongo_client</td><td>index_name</td></tr>
<tr><td>elastic_client</td><td>max_size</td></tr>
<tr><td>Μέθοδοι</td><td colspan="2">

rest_name
Προσφέρει RESTful απάντηση με αποτελέσματα αναζήτησης με σύνθετο query στην Elasticsearch με βάση λέξη-κλειδί που περιλαμβάνεται στο HTTP request.
rest_near
Προσφέρει RESTful απάντηση με αποτελέσματα αναζήτησης με σύνθετο query στην Elasticsearch που περιλαμβάνει και γεωγραφικό κριτήριο κατάταξης με βάση λέξη-κλειδί και συντεταγμένων χρήστη, που περιλαμβάνονται στο HTTP request.
rest_autocomplete
Προσφέρει RESTful απάντηση με αποτελέσματα αναζήτησης με auto-complete query στην Elasticsearch με βάση λέξη-κλειδί που περιλαμβάνεται στο HTTP request.
rest_id
Προσφέρει RESTful απάντηση με λεπτομέρειες εγγραφής στην MongoDB με βάση πεδίο «id» που περιλαμβάνεται στο HTTP request.
search_results
Εμφανίζει αποτελέσματα σε συγκεκριμένο template της Web εφαρμογής μετά από αναζήτηση του χρήστη για κάποια λέξη/φράση κλειδί.
search_page
Εμφανίζει την αρχική οθόνη της Web εφαρμογής σε κατάλληλο template.
live_search
Καλείται με κάθε νέο χαρακτήρα που πληκτρολογεί ο χρήστης στην μπάρα αναζήτησης της Web εφαρμογής και εμφανίζει auto-complete αποτελέσματα.
get_by_id
Εμφανίζει λεπτομέρειες εγγραφής που ανακτά από την MongoDB με βάση

</td></tr>
</table>

	πεδίο «_id» που περιλαμβάνεται στο HTTP request, σε κατάλληλο template της Web εφαρμογής.

Όνομα Αρχείου	**helpers.py**	
Μεταβλητές	mongo_client	index_name
	elastic_client	max_size
Μέθοδοι	• **find** Επιστρέφει πίνακα με αποτελέσματα αναζήτησης με σύνθετο query στην Elasticsearch. • **find_nearme** Επιστρέφει πίνακα με αποτελέσματα αναζήτησης με σύνθετο query στην Elasticsearch που περιλαμβάνει και γεωγραφικό κριτήριο κατάταξης. **autocomplete** Επιστρέφει πίνακα με αποτελέσματα αναζήτησης με auto-complete query στην Elasticsearch. • **parse_results** Επεξεργάζεται τα αποτελέσματα από απάντηση της Elasticsearch σε οποιοδήποτε query και τα τοποθετεί με κατάλληλο τρόπο σε πίνακα για χρήση από τις μεθόδους που απαντάνε στις εφαρμογές.	

Όνομα Αρχείου	**queries.py**
Μέθοδοι	• **searchQuery** Κατασκευάζει σύνθετο query για αποστολή στην Elasticsearch με βάση λέξη-κλειδί που δέχεται σαν όρισμα. • **autocompleteQuery** Κατασκευάζει autocomplete query για αποστολή στην Elasticsearch με βάση λέξη-κλειδί που δέχεται σαν όρισμα. • **nearSearchQuery** Κατασκευάζει σύνθετο query για αποστολή στην Elasticsearch που περιλαμβάνει και γεωγραφικό κριτήριο κατάταξης με βάση λέξη-κλειδί και συντεταγμένες που δέχεται σαν όρισματα.

4.5. Elasticsearch

4.5.1. Schema

Η μορφή αποθήκευσης δεδομένων (schema) η οποία επιλέχθηκε για το index της Elasticsearch περιέχει μέρος μόνο των αρχικών πεδίων των εγγράφων (documents) που αντιπροσωπεύουν την πλήρη πληροφορία της βάσης δεδομένων και τοποθετείται στην αντίστοιχη βάση στη MongoDB. Αυτό συμβαίνει για διάφορους λόγους. Αρχικά, οι αναζητήσεις πραγματοποιούνται σε κάποια πεδία, όπως το όνομα, η διεύθυνση και ο τύπος των places. Η κατάταξη των αποτελεσμάτων γίνεται με βάση συγκεκριμένα κριτήρια, όπως η βαθμολογία που έχει λάβει το κάθε place από τους υπόλοιπους χρήστες και η γεωγραφική τοποθεσία του σε σύγκριση με αυτή του χρήστη, αλλά και την εσωτερική βαθμολόγηση της αναζήτησης κειμένου της Elasticsearch με βάση την σχετικότητα κειμένου. Έτσι, δεν υπάρχει ανάγκη αποθήκευσης για παράδειγμα της διεύθυνσης ιστοτόπου του εκάστοτε place, καθώς αποτελεί περιττή πληροφορία για το index της Elasticsearch και επιπλέον αυξάνει τις απαιτήσεις χώρου (storage) της Elasticsearch, αλλά και επιβαρύνει χρονικά την ένταξη των νέων εγγραφών στο ευρετήριο.

Τελικά, η schema των δεδομένων μας στην Elasticsearch φαίνεται στο παρακάτω έγγραφο-παράδειγμα, στη μορφή που αποθηκεύεται σε αυτή:


```
{
    "_index": "elasticplaces",
    "_type": "places",
    "_id": "57ea6f4eeed65740abf8d504",
    "_score": 1,
    "_source": {
        "geometry": {
            "location": {
                "lat": 37.9510091,
                "lng": 23.9354903
            }
        },
        "formatted_address": "Irakleitou 15",
        "name": "Galaxy Bar",
        "types": [
            "bar",
            "restaurant"
        ]
    }
}
```

Τα πεδία «_index», «_type», «_id» και «_score» αποτελούν τα μετα-δεδομένα της Elasticsearch, ενώ το πεδίο «_source» περιέχει το έγγραφο με τα πεδία τα οποία αποθηκεύουμε στην βάση για να τα χρησιμοποιήσουμε για αναζητήσεις. Βλέπουμε λοιπόν ότι περιλαμβάνονται τα πεδία «geometry», «formatted_address», «name» και «types». Το πεδίο «geometry» είναι σύνθετο και περιέχει ένα πεδίο «location», το οποίο είναι ένας πίνακας που περιέχει το γεωγραφικό μήκος (latitude) και πλάτος (longitude) του place.

4.5.2. Σχεδίαση απαντήσεων της μηχανής με βάση τις ανάγκες του χρήστη

Για να σχεδιάσουμε τα ερωτήματα που θέλουμε να κάνουμε στην βάση εκ μέρους του χρήστη των εφαρμογών, οφείλουμε να αναλογιστούμε με ποιο τρόπο θα χρησιμοποιεί την πληροφορία αυτή ο χρήστης και ποιες θα είναι οι ανάγκες του τις οποίες οφείλουμε να καλύψουμε, καθώς και πως μπορούμε να βελτιώσουμε τα αποτελέσματα που θα του παρουσιάσουμε. Στο δικό μας σενάριο χρήσης λοιπόν, οι χρήστες αναζητούν places, «μέρη ενδιαφέροντος» δηλαδή, όπως εστιατόρια, μπαρ, καταστήματα κλπ.

Ο προφανής και απλούστερος τρόπος αναζήτησης είναι να γνωρίζει ο χρήστης εκ των προτέρων το όνομα του place για το οποίο θέλει να μάθει πληροφορίες, και να πληκτρολογήσει αυτό το όνομα στην μπάρα αναζήτησης της εφαρμογής. Σε αυτή την περίπτωση, θα πρέπει το ερώτημα που θα κάνουμε στην Elasticsearch να «ματσάρει» αυτή τη λέξη-κλειδί που δεχόμαστε από τον χρήστη με τα ονόματα των places (πεδίο name) στην βάση.

Μία δεύτερη περίπτωση είναι ο χρήστης να γνωρίζει σε ποια οδό βρίσκεται το place που ψάχνει, και ίσως να μην θυμάται το όνομα του απ' έξω, ή εναλλακτικά να θέλει να αναζητήσει όλα τα places στην συγκεκριμένη οδό. Τώρα, θα ζητήσουμε από την Elasticsearch να «ματσάρει» την λέξη-κλειδί με όλες τις διευθύνσεις από places (πεδίο formatted_address) που υπάρχουν στη βάση.

Τέλος, διακρίνουμε την περίπτωση να επιθυμεί ο χρήστης να αναζητήσει όλα τα places ενός συγκεκριμένου τύπου, για παράδειγμα να θέλει να βρει εστιατόρια. Τότε θα πληκτρολογήσει έναν τύπο στη μπάρα αναζήτησης, και θα χρειαστεί το ερώτημά μας στην Elasticsearch να «ματσάρει» όλα τα πεδία που καθορίζουν τους τύπους των places, και να επιστρέψει αυτά που ταιριάζουν.

Θα πρέπει επίσης να εξετάσουμε την πιθανότητα ο χρήστης να θελήσει να εισάγει περισσότερες από μία λέξεις-κλειδιά που να αντιστοιχούν σε περισσότερες από μία «κρίσιμες» πληροφορίες (με τον όρο κρίσιμες εννοούμε πληροφορίες που αντιστοιχούν σε συγκεκριμένο πεδίο). Για παράδειγμα, μπορεί ο χρήστης να θέλει να εισάγει όνομα οδού αλλά και τύπο place, για να

εντοπίσει για παράδειγμα όλα τα μπαρ που υπάρχουν στην οδό Αριστοφάνους. Θα πληκτρολογήσει λοιπόν «μπαρ αριστοφάνους». Θα πρέπει τότε να ζητήσουμε να «ματσαριστούν» και οι δύο λέξεις με πεδία διευθύνσεων αλλά και τύπων, ώστε να προκύψουν αποτελέσματα που είτε είναι μπαρ, είτε είναι στην αριστοφάνους, είτε και τα δύο μαζί (προφανώς επίσης που έχουν στο όνομα μπαρ, ή έχουν στο όνομά τους την λέξη «αριστοφάνους» κ.ο.κ).

Προκύπτει συμπερασματικά η ανάγκη να είναι περίπλοκα τα ερωτήματα στη βάση και να περιλαμβάνουν όλες τις παραπάνω περιπτώσεις και συνδυασμούς λέξεων-κλειδιών και πεδίων στα οποία «στοχεύει» να αναζητήσει ο χρήστης. Αυτή η ανάγκη θα ικανοποιηθεί με την χρήση των εξειδικευμένων ερωτημάτων-queries «ματσαρίσματος» (matching) που παρέχει η Elasticsearch, και πιο συγκεκριμένα με χρήση του Boolean ερωτήματος «should», κάτω από το οποίο θα ζητήσουμε να γίνεται match η λέξη/φράση κλειδί με το όνομα, την διεύθυνση και τον τύπο. Αυτό θα έχει ως αποτέλεσμα, με βάση το documentation, να επιστραφούν όλα τα αποτελέσματα τα οποία έχουν τουλάχιστον ένα σωστό ταίριασμα (δηλαδή μία έστω από τις λέξεις-κλειδιά που έχουν χρησιμοποιηθεί να βρεθεί σε ένα τουλάχιστον από τα πεδία που ζητήσαμε – εν προκειμένω name, formatted_address και types). Προφανώς θα επιστραφεί και κάθε αποτέλεσμα που έχει παραπάνω από ένα σωστά ταιριάσματα, και το οποίο μάλλον θα είναι και πιο σχετικό με αυτό που ψάχνει ο χρήστης από αυτό που έχει μόνο ένα ταίριασμα. Θα αναλύσουμε το συγκεκριμένο θέμα στην επόμενη υπο-ενότητα.

4.5.3. Κατάταξη αποτελεσμάτων αναζήτησης

Αφότου λοιπόν σχεδιάσαμε τα ερωτήματα ως προς τις λέξεις-κλειδιά των αναζητήσεων, σειρά τώρα έχει ο τρόπος, και πιο συγκεκριμένα η σειρά, με την οποία θα επιστρέφονται τα αποτελέσματα στον χρήστη. Είναι προφανές ότι ο χρήστης επιθυμεί να του παρουσιαστούν σε αρχικό επίπεδο τα πιο σχετικά με την αναζήτησή του αποτελέσματα, ώστε να μπορεί να βρει το/τα μέρη που αναζητεί χωρίς να χρειάζεται να ανατρέξει σε όλες τις σελίδες των αποτελεσμάτων.

Ο μηχανισμός με τον οποίο η Elasticsearch κατατάσσει τα αποτελέσματα που μας επιστρέφει ονομάζεται «scoring» και μπορεί να επεκταθεί και να προσαρμοστεί στις ανάγκες της αναζήτησης, ανάλογα με τα κριτήρια που θέσουμε στα ερωτήματά μας.

Ας αναλύσουμε λοιπόν τα κριτήρια με τα οποία θα κατατάξουμε τα αποτελέσματα που θα ικανοποιήσουν τις συνθήκες matching που αναλύσαμε παραπάνω.

4.5.3.1. Με βάση το κείμενο

Πρώτα και σημαντικότερα, θέλουμε να εμφανίζονται ψηλότερα τα αποτελέσματα που είναι περισσότερο σχετικά με τις λέξεις κλειδιά του χρήστη ως προς την ομοιότητα κειμένου (text relevance). Η Elasticsearch ήδη λαμβάνει υπόψιν της στο score που αναθέτει σε κάθε αποτέλεσμα την ομοιότητα κειμένου, αλλά θα χρειαστεί να αναθέσουμε βάρη σημαντικότητας στα πεδία που γίνονται matched ώστε να ορίσουμε ποια είναι σημαντικότερα από άλλα κατά τις αναζητήσεις του χρήστη. Συγκεκριμένα, είναι σημαντικό για τον χρήστη να εμφανίζονται νωρίτερα αποτελέσματα τα οποία έχουν κάνει match την λέξη-κλειδί με την οποία αναζήτησε με το όνομα κάποιου place, κατόπιν με βάση τον τύπο, και τέλος με βάση την διεύθυνση. Αυτό είναι σχετικά προφανές, καθώς το πιο λογικό για τους περισσότερους χρήστες είναι να αναζητήσουν με λέξη-κλειδί το όνομα, και σε ότι αφορά την διάκριση ανάμεσα σε τύπο και διεύθυνση επιλέξαμε τον τύπο να θεωρείται σημαντικότερος διότι θεωρούμε πως σπάνια οι χρήστες γνωρίζουν και θυμούνται συγκεκριμένες διευθύνσεις, ενώ ακόμα και αν δεν ξέρουν τίποτα για ένα μέρος μπορούν πάντα να αναζητήσουν με βάση τον τύπο του.

Επιπλέον, η σχετικότητα κειμένου εντοπίζεται και στο ζήτημα του σε πόσα από τα πεδία που ψάχνουμε υπάρχει ταίριασμα με τη λέξη/φράση κλειδί που αναζητά ο χρήστης. Δηλαδή αν για κάποια λέξη σε ένα αποτέλεσμα υπάρχει ταίριασμα και στο όνομα αλλά και στον τύπο, αυτό το αποτέλεσμα θα παίρνει παραπάνω score από ένα στο οποίο το ταίριασμα υπάρχει μόνο στο όνομα. Η σχετικότητα αυτή εντοπίζεται από τον μηχανισμό αναζήτησης της Elasticsearch, και τα βάρη που εισάγουμε κατευθύνουν την επιλογή κατάταξης με βάση την σημαντικότητα του πεδίου ακόμα και όταν υπάρχουν πολλαπλά ταιριάσματα.

4.5.3.2. Με βάση την βαθμολογία χρηστών

Η χρήση αυτού του κριτηρίου είναι σχετικά προφανής. Οι χρήστες επιθυμούν να εμφανίζονται με μεγαλύτερη προτεραιότητα αποτελέσματα τα οποία είναι δημοφιλέστερα ανάμεσα σε άλλους χρήστες, και έχουν βαθμολογηθεί από αυτούς με υψηλή προτίμηση. Θα συμπεριλάβουμε λοιπόν στα ερωτήματά μας την ειδική συνάρτηση που παρέχεται από την Elasticsearch «field_value_factor», η οποία θα δίνει βαρύνουσα σημασία στο πεδίο των εγγράφων «rating», που αντιπροσωπεύει τις βαθμολογίες που έχει λάβει αυτό το place από άλλους χρήστες.

4.5.3.3. Με βάση την τοποθεσία

Στην περίπτωση που διαθέτουμε την γεωγραφική τοποθεσία του χρήστη, μία ιδιαίτερη διευκόλυνση για αυτόν είναι το να κατατάξουμε τα αποτελέσματα συμπεριλαμβάνοντας στα κριτήρια βαθμονόμησης την απόσταση του κάθε place από την τοποθεσία του. Προφανώς λοιπόν αν ο χρήστης αναζητήσει ένα κατάστημα που ανήκει σε αλυσίδα καταστημάτων, θα εμφανιστούν στα αποτελέσματά του πιο πάνω τα καταστήματα της αλυσίδας που βρίσκονται πιο κοντά στην περιοχή του, και χαμηλότερα τα πιο απομακρυσμένα. Για να πετύχουμε την καταμέτρηση αυτού του κριτηρίου στο score της Elasticsearch, θα χρησιμοποιήσουμε την ειδική συνάρτηση Gauss, η οποία είναι μία φθίνουσα συνάρτηση που αντιπροσωπεύει τη σημαντικότητα της απόστασης από τον χρήστη. Συγκεκριμένα, τέλειο score (σε ότι αφορά την απόσταση) λαμβάνουν αποτελέσματα σε μία συγκεκριμένη ακτίνα γύρω από την τοποθεσία του χρήστη, ενώ το score των αποτελεσμάτων μειώνεται με αυξανόμενο ρυθμό (και όχι απλά γραμμικά – εξ ου και η επιλογή της συνάρτησης Gauss έναντι της απλής γραμμικής), όσο απομακρύνονται τα places από την τοποθεσία του χρήστη. Αυτό έχει την λογική ότι ακόμα και αν δεν είναι ακριβώς δίπλα στον χρήστη τα places, μπορεί παρόλα αυτά να τον ενδιαφέρουν, αν όμως είναι αρκετά μακριά τότε θα τον ενδιαφέρουν πολύ λιγότερο, είτε είναι πολύ μακριά είτε είναι πάρα πολύ μακριά.

4.5.4. Παραμετροποίηση του Index

Η Elasticsearch μας δίνει την δυνατότητα να παραμετροποιήσουμε το ευρετήριο της που θα χρησιμοποιήσουμε κατά την δημιουργία του, προτού εισάγουμε οποιαδήποτε δεδομένα. Θα εκμεταλλευτούμε λοιπόν αυτή τη δυνατότητα προκειμένου να χρησιμοποιήσουμε κάποια από τα χαρακτηριστικά της τα οποία χρειαζόμαστε για την λειτουργικότητα της μηχανής αναζήτησης που σχεδιάζουμε.

Καταρχάς θα χρειαστούμε κάποια mappings λατινικών και ελληνικών χαρακτήρων προκειμένου να υποστηρίζεται αυτή η δυνατότητα αναζήτησης είτε σε επίσημα ελληνικά είτε σε greeklish. Θα θέσουμε ενδεικτικά τις γνωστότερες αντιστοιχίσεις που χρησιμοποιούνται συνήθως σε αυτή την ψηφιακή διάλεκτο, περιλαμβάνοντας και κάποιους διφθόγγους, όπως για παράδειγμα το «ks» το οποίο χρησιμοποιείται αντί του «ξ» πολύ συχνά. Τα mappings αυτά εντάσσονται σε ένα φίλτρο χαρακτήρων (char_filter) που δημιουργούμε κατά την αρχικοποίηση του index.

Επίσης θα δημιουργήσουμε δύο ακόμα φίλτρα, ένα για να διαχειρίζεται την αντιστοίχιση μικρών-κεφαλαίων γραμμάτων και ένα για να επεξεργάζεται τις λέξεις εντός των δεδομένων-εγγράφων και να παράγει κάποιους αναγραμματισμούς λέξεων.

Αυτοί οι αναγραμματισμοί θα παραχθούν την ώρα του indexing του κάθε document, και θα μπορούν να χρησιμοποιηθούν άμεσα σε κάθε αναζήτηση στην συνέχεια προκειμένου να βελτιώνεται η αναζήτηση λέξεων κλειδιών. Δίνεται λοιπόν η δυνατότητα έτσι, να επιστραφούν αποτελέσματα στο χρήστη τα οποία περιέχουν τις λέξεις-κλειδιά ως κομμάτια άλλων λέξεων που υπάρχουν σε έγγραφα, ή λέξεων που μοιάζουν με αυτές που χρησιμοποίησε ο χρήστης. Για παράδειγμα αν υπάρξει κάποιο ορθογραφικό λάθος από πλευράς χρήστη, οι αναγραμματισμοί θα επιτρέψουν στην Elasticsearch να επιστρέψει αποτελέσματα τα οποία όντως αναζητά ο χρήστης. Ακόμα ένα παράδειγμα είναι ότι αν δοθεί ένα μέρος μίας λέξης, και πάλι θα επιστραφούν αποτελέσματα που περιέχουν ολόκληρη αυτή τη λέξη. Προφανώς οι αναγραμματισμοί που δημιουργούνται δεν μπορούν να είναι όλοι οι πιθανοί, καθώς αυτό δεν μπορεί να κλιμακώσει ρεαλιστικά για εκατομμύρια έγγραφα, οπότε θέτουμε ένα πάνω όριο γραμμάτων που θα χρησιμοποιούνται για να δημιουργηθούν αναγραμματισμοί για κάθε λέξη, και συγκεκριμένα τα πρώτα οκτώ γράμματα.

Στην συνέχεια, δημιουργούμε έναν δικό μας analyzer, στον οποίο εντάσσουμε τα δύο φίλτρα που δημιουργήσαμε συν το προκαθορισμένο “lowercase” φίλτρο της Elasticsearch, καθώς και το char_filter που δημιουργήσαμε για τα mappings των greeklish. Ορίζουμε τα πεδία των documents στα οποία θα πρέπει να εφαρμόζεται αυτός ο analyzer κατά το indexing των δεδομένων, και αυτά είναι τα name, formatted_address, types.

Τέλος, θέτουμε το πεδίο «location» των εγγράφων να είναι τύπου «geo_point», και αυτό θα μας επιτρέψει να χρησιμοποιήσουμε τις γεωγραφικές υπηρεσίες της Elasticsearch στις αναζητήσεις, καθώς σε αυτό το πεδίο υπάρχει ένας πίνακας με τις συντεταγμένες του κάθε place με βάση τις προδιαγραφές των Geo points της Elasticsearch.

4.5.5. Χρήση Python API

Για την επικοινωνία με την Elasticsearch τόσο από την πλευρά του server, όσο και μέσω άλλων προγραμμάτων σεναρίου (scripts) για διάφορες λειτουργίες, όπως ο συγχρονισμός των δεδομένων ή η αρχικοποίηση του index, χρησιμοποιήθηκε το επίσημο Python API (elasticsearch-py) που παρέχεται από την Elasticsearch, και ο κώδικας γράφτηκε στη γλώσσα προγραμματισμού Python.

Στο παρακάτω σχεδιάγραμμα φαίνονται οι μέθοδοι του API που χρησιμοποιήσαμε σε διάφορα σημεία του συστήματος.

Μέθοδος	Περιγραφή
`Elasticsearch()`	Επιστρέφει ένα αντικείμενο που χρησιμοποιείται ως client της Elasticsearch. Οι υπόλοιπες μέθοδοι καλούνται πάνω σε αυτό.
`search()`	Εκτελεί αναζήτηση σε ένα index της Elasticsearch παίρνοντας ως όρισμα το ερώτημα (query) σε μορφή JSON και επιστρέφει τα αποτελέσματα επίσης σε μορφή JSON.
`indices.delete()`	Διαγράφει ένα συγκεκριμένο index από την βάση της Elasticseach.
`indices.create()`	Δημιουργεί ένα νέο index παίρνοντας ως όρισμα το όνομα και τις ρυθμίσεις που θα έχει αυτό.
`index()`	Προσθέτει ένα JSON έγγραφο σε ένα συγκεκριμένο index, κάνοντας το αναζητήσιμο.
`get()`	Επιστρέφει ένα έγγραφο από ένα συγκεκριμένο index της Elasticsearch, παίρνοντας ως όρισμα το id του εγγράφου.
`helpers.bulk()`	Επιτρέπει την εκτέλεση πολλών index/delete εντολών σε μία μοναδική κλήση API .

4.6. MongoDB

4.6.1. Schema

Το σύστημα χρησιμοποιεί την MongoDB ως την κύρια αποθήκη δεδομένων, από την οποία μεταφέρονται τα χρήσιμα πεδία στην Elasticsearch για αναζητήσεις. Αποθηκεύεται λοιπόν η πλήρης πληροφορία των δεδομένων σε αυτή, περιλαμβάνοντας τόσο τα πεδία τα οποία χρειάζεται η Elasticsearch, όσο και τα επιπλέον στοιχεία που έχουμε για κάθε place, όπως πχ η διεύθυνση ιστοτόπου. Η schema είναι ευέλικτη στην MongoDB, κάτι το οποίο ταιριάζει στην περίπτωσή μας καθώς τα διάφορα έγγραφα μπορούν να περιέχουν κάποια πεδία που άλλα δεν περιέχουν. Θα παρουσιάσουμε εδώ την πλήρη schema των δεδομένων μας, με την υποσημείωση ότι δεν είναι ίδια για όλα τα έγγραφα, αλλά αυτή θα ήταν η συνολική μορφή κάποιου εγγράφου που θα περιείχε όλα τα πεδία ανεξαιρέτως, που είναι διαθέσιμα. Ένα παράδειγμα εγγράφου στην MongoDB με όλη την schema:


```
{
    "formatted_address": "48 Pirrama Road, Pyrmont NSW, Australia",
    "formatted_phone_number": "(02) 9374 4000",
    "geometry": {
        "location": {
            "lat": -33.866971,
            "lng": 151.195875
        }
    },
    "icon":
"http://maps.gstatic.com/mapfiles/place_api/icons/generic_business-
71.png",
    "name": "Google Sydney",
    "_id": "ChIJN1t_tDeuEmsRUsoyG83frY4",
    "rating": 4.7,
    "reviews": [
        {
            "aspects": [
                {
                    "rating": 3,
                    "type": "quality"
                }
            ],
            "author_name": "Simon Bengtsson",
            "author_url":
"https://plus.google.com/104675092887960962573",
            "language": "en",
            "rating": 5,
            "text": "Just went inside to have a look at Google.
Amazing.",
            "time": 1338440552869
```

```
        },
        {
            "aspects": [
                {
                    "rating": 3,
                    "type": "quality"
                }
            ],
            "author_name": "Felix Rauch Valenti",
            "author_url":
"https://plus.google.com/103291556674373289857",
            "language": "en",
            "rating": 5,
            "text": "Best place to work :-)",
            "time": 1338411244325
        },
        {
            "aspects": [
                {
                    "rating": 3,
                    "type": "quality"
                }
            ],
            "author_name": "Chris",
            "language": "en",
            "rating": 5,
            "text": "Great place to work, always lots of free food!",
            "time": 1330467089039
        }
    ],
    "types": [
        "establishment"
    ],
    "url": "http://maps.google.com/maps/place?cid=10281119596374313554",
    "website": "http://www.google.com.au/"
}
```


4.6.2. Replica sets

Η MongoDB παρέχει την δυνατότητα χρήσης των replica sets στην βάση που χρησιμοποιούμε. Αυτά προσφέρουν ασφάλεια δεδομένων και αυξημένη διαθεσιμότητα, και είναι η βάση για όλες τις χρήσεις της στην παραγωγή. Ένα replica set είναι μία ομάδα από διεργασίες οι οποίες διατηρούν το ίδιο σύνολο δεδομένων. Κατ' αυτόν τον τρόπο τα δεδομένα μπορούν να διαμοιραστούν σε πολλούς διαφορετικούς εξυπηρετητές βάσεων δεδομένων, με αποτέλεσμα να υπάρχουν αντίγραφα ασφαλείας σε κάθε περίπτωση, αλλά και να είναι άμεσα διαθέσιμα ανεξάρτητα από τις συνθήκες δικτύου κάποιου μοναδικού εξυπηρετητή. Επιπλέον, σε κάποιες περιπτώσεις, μπορεί να επιτευχθεί αυξημένη ταχύτητα ανάγνωσης καθώς τα ερωτήματα στην βάση μπορούν να αποστέλλονται σε διαφορετικούς εξυπηρετητές ανάλογα με την ισορροπία φόρτου εργασίας ανάμεσά τους.

Η λειτουργία αυτή της MongoDB, δημιουργεί ένα log με timestamps των αλλαγών που συμβαίνουν σε κάθε replica set, και το οποίο χρησιμοποιεί προκειμένου να κρατάει συγχρονισμένα όλα τα αντίγραφα της βάσης μεταξύ τους. Αυτό το log εκμεταλλεύονται στη συνέχεια και όλα τα εργαλεία συγχρονισμού από MongoDB σε Elasticsearch, και μπορούν έτσι να παρατηρούν τις αλλαγές στην βάση αυτή προκειμένου να γνωρίζουν ποια ακριβώς είναι τα νέα έγγραφα που εισήχθησαν και πρέπει να αντιγραφούν (με μετατροπή της schema ενδιάμεσα) στο index της Elasticsearch.

4.6.3. Χρήση Python API

Για την επικοινωνία με την MongoDB τόσο από την πλευρά του server, όσο και μέσω άλλων προγραμμάτων σεναρίου (scripts) για διάφορες λειτουργίες, όπως ο συγχρονισμός των δεδομένων ή η εισαγωγή των νέων δεδομένων στη βάση, χρησιμοποιήθηκε το επίσημο Python API (pymongo) που παρέχεται από την MongoDB, και ο κώδικας γράφτηκε στη γλώσσα προγραμματισμού Python. Στο παρακάτω σχεδιάγραμμα φαίνονται οι μέθοδοι του API που χρησιμοποιήσαμε σε διάφορα σημεία του συστήματος.

Μέθοδος	Περιγραφή
`MongoClient()`	Επιστρέφει ένα αντικείμενο που χρησιμοποιείται ως client της MongoDB. Οι υπόλοιπες μέθοδοι καλούνται πάνω σε αυτό.
`find()`	Επιστρέφει ένα σύνολο εγγράφων από τη συλλογή (collection) τα οποία πληρούν τα κριτήρια που τίθενται ως όρισμα.
`find_one()`	Επιστρέφει ένα έγγραφο από ένα συγκεκριμένο collection της MongoDB, παίρνοντας ως όρισμα το id του εγγράφου.
`insert()`	Προσθέτει ένα JSON έγγραφο σε ένα συγκεκριμένο collection.
`insert_many()`	Επιτρέπει την εκτέλεση πολλών insert εντολών σε μία μοναδική κλήση API .
`count()`	Επιστρέφει το πλήθος των εγγραφών σε μία συλλογή.

4.7. Μέθοδοι συγχρονισμού δεδομένων

Ο σχεδιασμός του συγχρονισμού μεταξύ των δεδομένων είναι ένα πολύ σημαντικό πρόβλημα που συναντάται στο σύστημα μας λόγω της ύπαρξης δύο διαφορετικών συστημάτων διαχείρισης βάσεων δεδομένων, της Elasticsearch και της MongoDB. Υπάρχουν δύο κυρίως θέματα σε αυτό το υποσύστημα: 1) θα πρέπει οι δύο βάσεις να έχουν τα ίδια δεδομένα, παρότι η εισαγωγή τους στο σύστημα πραγματοποιείται μέσω εισαγωγής τους στην MongoDB, και 2) θα πρέπει η schema των εγγράφων (documents) να μεταβάλλεται κατά τη διαδικασία συγχρονισμού, αφού τα δεδομένα στην Elasticsearch θα έχουν μέρος μόνο των πεδίων των αρχικών δεδομένων.

Στις παρακάτω ενότητες περιγράφεται η κάθε μέθοδος συγχρονισμού που χρησιμοποιήσαμε και ο τρόπος με τον οποίο επιλύονται τα δύο προβλήματα που τέθηκαν.

4.7.1. Elasticsearch Rivers

Τα rivers είναι plugins της Elasticsearch τα οποία λειτουργούν μέσω αυτής για να «τραβάνε» δεδομένα από κάποια εξωτερική πηγή, τα οποία κατόπιν προστίθενται σε κάποιο προσδιορισμένο index. Τα rivers έχουν τύπο, ο οποίος ουσιαστικά αντιπροσωπεύει την πηγή των εισερχόμενων δεδομένων. Συγκεκριμένα στην δική μας υλοποίηση, ο τύπος του river που χρησιμοποιούμε είναι «mongodb», αφού λαμβάνουμε δεδομένα από μία συλλογή σε μία βάση MongoDB.

Θα χρησιμοποιήσουμε επίσης την επιλογή «include_fields» για να δείξουμε στο river ποια ακριβώς πεδία της schema που έχουν τα έγγραφα στην MongoDB θέλουμε να εισάγονται στην Elasticsearch.

4.7.2. Mongo-connector

Πρόκειται για ένα εργαλείο συγχρονισμού δεδομένων για MongoDB. Δημιουργεί ένα «pipeline» από ένα MongoDB σύμπλεγμα (cluster) προς ένα ή παραπάνω άλλα συστήματα, όπως Elasticsearch, Solr ή ένα άλλο MongoDB cluster. Συγχρονίζει δεδομένα από την βάση στην MongoDB στο άλλο σύστημα και στην συνέχεια παρακολουθεί το oplog της MongoDB για να εντοπίζει αλλαγές στη βάση σε πραγματικό χρόνο. Επίσης δημιουργεί ένα index «mongodb_meta» στην Elasticsearch το οποίο χρησιμοποιείται για να καταγράφεται πότε πραγματοποιήθηκε η τελευταία αλλαγή σε κάποιο έγγραφο.

Σε αυτό το εργαλείο θα χρησιμοποιήσουμε την επιλογή «`"fields": ["_id", "name", "formatted_address", "types", "geometry.location", "rating"]`» προκειμένου να μετατρέψουμε την schema κατά το πέρασμα στην Elasticsearch.

4.7.3. Transporter

Ακόμα ένα εργαλείο συγχρονισμού δεδομένων που μεταξύ άλλων δίνει την δυνατότητα για συγχρονισμό και μετατροπή δεδομένων από MongoDB σε Elasticsearch. Για την μετατροπή των δεδομένων, και πιο συγκεκριμένα της schema αυτών, χρησιμοποιούνται οι ειδικοί Transformers που παρέχονται και γράφονται σε JavaScript, προκειμένου να υποδείξουν την τελική μορφή των δεδομένων που θα εισαχθούν στην Elasticsearch.

4.7.4. Υλοποίηση λύσης συγχρονισμού σε Python και Java

Επιπλέον των παραπάνω εργαλείων συγχρονισμού που δοκιμάσαμε, αναπτύξαμε και δικές μας λύσεις προκειμένου να συγκρίνουμε την απόδοση των ήδη υπαρχόντων, χρησιμοποιώντας τες ως benchmark. Η πρώτη υλοποιείται ως πρόγραμμα σεναρίου (script) το οποίο είναι γραμμένο σε Python, διαβάζει τα δεδομένα από την MongoDB collection στην οποία τα έχουμε αποθηκεύσει, επιλέγει τα πεδία τα οποία είναι απαραίτητα για το index της Elasticsearch και τα εισάγει στο index με εντολές πλήθους (bulk) ώστε να εξασφαλίζεται η μεγαλύτερη δυνατή ταχύτητα στην διεπαφή με την βάση. Η δεύτερη λύση έχει γραφτεί με χρήση της γλώσσας Java και διαβάζει τα δεδομένα από την MongoDB, κατασκευάζει το κατάλληλο αντικείμενο για κάθε έγγραφο με τα πεδία που θέλουμε να κρατήσουμε στην Elasticsearch και τα εισάγει στο index με εντολές bulk. Σημειώνεται ότι τα προγράμματα αυτά δεν υλοποιούν διαρκή συγχρονισμό όπως τα παραπάνω εργαλεία, αλλά συγχρονίζουν μία φορά όλα τα έγγραφα από την μία βάση στην άλλη. Αυτή η λειτουργία αρκεί για την σύγκριση απόδοσης που θέλουμε να πραγματοποιήσουμε.

5

Υλοποίηση

5.1. Django Web Server

Σε αυτό το υπο-κεφάλαιο θα αναλύσουμε την λειτουργία του Server που αναπτύξαμε σε Django και ο οποίος διαχειρίζεται όλα τα αιτήματα στην μηχανή αναζήτησης, είτε αυτά προέρχονται από την web εφαρμογή, είτε από την εφαρμογή για το λειτουργικό σύστημα Android. Αξίζει να σημειώσουμε σε αυτό το σημείο, πως υλοποιούμε ένα RESTful interface για τις μεθόδους που απαντάνε στην Android εφαρμογή, το οποίο μπορεί να χρησιμοποιηθεί για οποιαδήποτε άλλη εφαρμογή που θα θελήσει να χρησιμοποιήσει το σύστημα μηχανής αναζήτησης, καθώς με απλά HTTP requests είναι δυνατό να αναζητήσει και να λάβει απαντήσεις μέσω του interface που δημιουργήσαμε.

Το αρχείο με τις κυρίως μεθόδους που παρέχουν την λειτουργικότητα του server φαίνεται παρακάτω:

views.py

```
# coding: utf-8
from django.shortcuts import get_object_or_404, render
from django.http import HttpResponseRedirect
from django.core.urlresolvers import reverse
import json, requests
from bson.objectid import ObjectId
from apps import mongo_client, elastic_client, index_name, max_size
from helpers import find, find_nearme, autocomplete
from django.core.paginator import Paginator, EmptyPage,
PageNotAnInteger
from django.http import JsonResponse

# homepage for the web app
def search_page(request):
    return render(request, "searchapp/search.html")

# methods for the rest interface for mobile app
```

```
# search by keyword
def rest_name(request):
    search_key = request.GET['search']
    results = find(search_key, 20)
    return JsonResponse({'res' : results})

# search by keyword and include geodata scoring
def rest_near(request):
    search_key = request.GET['search']
    search_lat = request.GET['lat']
    search_lon = request.GET['lon']
    results = find_nearme(search_key, search_lat, search_lon, 20)
    return JsonResponse({'res' : results})

# autocomplete results for live search
def rest_autocomplete(request):
    search_key = request.GET['search']
    results = autocomplete(search_key, 5)
    return JsonResponse({'res' : results})

# get full document from mongodb
def rest_id(request):
    separator = ','
    search_id = request.GET['id']
    res = result = mongo_client.find_one({"_id": search_id})
    result = {}
    result['id'] = res['_id']
    result['name'] = res['name']
    result['formatted_address'] =
res['formatted_address'].split(separator, 1)[0]
    result['lat'] = res['geometry']['location']['lat']
    result['lon'] = res['geometry']['location']['lng']
    try:
        result['rating'] = res['rating']
    except KeyError:
        result['rating'] = 0
    try:
        result['types'] = res['types']
    except KeyError:
        result['types'] = [""]
    try:
        result['website'] = res['website']
    except KeyError:
        result['website'] = " "
    try:
        result['formatted_phone_number'] =
res['formatted_phone_number']
    except KeyError:
        result['formatted_phone_number'] = " "

    return JsonResponse({'res' : result})

# query elasticsearch with keyword, display at results.html
def search_results(request):
    search_key = request.GET['search']
    results = find(search_key, max_size)
    paginator = Paginator(results, 15) # Show 15 results per page
    page = request.GET.get('page')
    try:
        page_results = paginator.page(page)
    except PageNotAnInteger:
```

```
        # If page is not an integer, deliver first page.
        page_results = paginator.page(1)
    except EmptyPage:
        # If page is out of range (e.g. 9999), deliver last page of
results.
        page_results = paginator.page(paginator.num_pages)
    return render(request, "searchapp/results.html", {'res' :
page_results, 'search_key' : search_key})

# query mongodb with id, display at details.html
def get_by_id(request, _id):
    result = mongo_client.find_one({"_id": _id})
    result['id'] = result['_id']
    return render(request, "searchapp/details.html", {'res' : result})

# ajax-livesearch
def live_search(request):
    if request.method == "POST":
        search_key = request.POST['search_text']
        results = autocomplete(search_key, 5)
    return render(request, 'searchapp/ajax_search.html', {'results' :
results})
```

Βλέπουμε δύο ομάδες μεθόδων, τις rest και τις απλές. Οι rest απαντούν στην mobile εφαρμογή, διαβάζουν τα κατάλληλα δεδομένα από το HTTP request το οποίο παίρνουν σαν όρισμα, καλούν την αντίστοιχη μέθοδο αναζήτησης που βρίσκεται στο αρχείο helpers.py και τέλος επιστρέφουν ένα αρχείο JSON με τα αποτελέσματα. Για την web εφαρμογή υπάρχουν τέσσερις μέθοδοι. Αρχικά, η μέθοδος που εμφανίζει την αρχική ιστοσελίδα. Η μέθοδος για αναζήτηση στην Elasticsearch, η οποία αφού πάρει αποτελέσματα από την βάση, τα εμφανίζει στην σελίδα results.html με χρήση paginator για την περίπτωση που τα αποτελέσματα είναι πολλά. Η μέθοδος για ανάκτηση του πλήρους εγγράφου από τη MongoDB και εμφάνιση του στην σελίδα details.html. Και τέλος, η μέθοδος που με χρήση ajax επιτρέπει την εμφάνιση αποτελεσμάτων με drop down menu στην αρχική σελίδα καθώς ο χρήστης πληκτρολογεί.

Το αρχείο με τις μεθόδους αναζήτησης:

helpers.py

```
from queries import searchQuery, nearSearchQuery, autocompleteQuery
from apps import mongo_client, elastic_client, index_name, max_size

def find_nearme(search_key, lat, lon, results_size):
    search_query = nearSearchQuery(search_key, lat, lon)
    return parse_results(search_query, results_size)

def find(search_key, results_size):
    search_query = searchQuery(search_key)
    return parse_results(search_query, results_size)

def autocomplete(search_key, results_size):
```

```
    search_query = autocompleteQuery(search_key)
    return parse_results(search_query, results_size)

def parse_results(search_query, results_size):
    separator = ','
    results = []
    res = elastic_client.search(index=index_name, body=search_query,
size=results_size)
    for item in res['hits']['hits']:
        temp = {}
        temp['name'] = item['_source']['name']
        temp['formatted_address'] =
item['_source']['formatted_address'].split(separator, 1)[0]
        temp['id'] = item['_id']
        temp['types'] = item['_source']['types']
        try:
            temp['rating'] = item['_source']['rating']
        except KeyError:
            temp['rating'] = "-"
        results.append(temp)
    return results
```

Όλες οι μέθοδοι παίρνουν τα κατάλληλα ορίσματα, κατασκευάζουν το query προς την Elasticsearch με χρήση της αντίστοιχης μεθόδου που βρίσκεται στο αρχείο queries.py, και στην συνέχεια επιστρέφουν τα επεξεργασμένα αποτελέσματα. Η επεξεργασία γίνεται στην μέθοδο parse_results, η οποία στέλνει την ερώτηση στην βάση, δέχεται τα αποτελέσματα και τα τοποθετεί σε πίνακα κατάλληλα διαμορφωμένο για χρήση από τις εφαρμογές.

Το αρχείο με τις μεθόδους που κατασκευάζουν τα queries προς την Elasticsearch:

queries.py

```
# coding: utf-8
def searchQuery(search_key):
    return  {
            "query":{
               "function_score":{
                  "query":{
                     "bool":{
                        "should":[
                           {
                              "match":{
                                 "name":{
                                    "query":search_key,
                                    "boost":5
                                 }
                              }
                           },
                           {
                              "match":{
                                 "name.greeklish":{
                                    "query":search_key,
                                    "boost":5
                                 }
```

```
                        }
                    },
                    {
                        "match":{
                            "formatted_address":{
                                "query":search_key,
                                "boost":2
                            }
                        }
                    },
                    {
                        "match":{
                            "formatted_address.greeklish":{
                                "query":search_key,
                                "boost":2
                            }
                        }
                    },
                    {
                        "match":{
                            "types":{
                                "query":search_key,
                                "boost":3
                            }
                        }
                    },
                    {
                        "match":{
                            "types.greeklish":{
                                "query":search_key,
                                "boost":3
                            }
                        }
                    }
                ]
            }
        },
        "functions":[
            {
                "field_value_factor":{
                    "field":"rating",
                    "factor":1.2,
                    "missing":2
                }
            }
        ],
        "boost_mode": "avg"
    }
  }
}


def autocompleteQuery(search_key):
  return  {
    "query":{
      "multi_match" : {
        "query":search_key,
        "type":"phrase_prefix",
        "max_expansions" : 1000,
```

```
        "fields":[ "name^5", "name.greeklish^5", "types^3",
"types.greeklish^3", "formatted_address^2",
"formatted_address.greeklish^2"]
      }
    }
  }

def nearSearchQuery(search_key, lat, lon):
    return  {
                "query":{
                   "function_score":{
                      "query":{
                         "bool":{
                            "should":[
                               {
                                  "match":{
                                     "name":{
                                        "query":search_key,
                                        "boost":5
                                     }
                                  }
                               },
                               {
                                  "match":{
                                     "name.greeklish":{
                                        "query":search_key,
                                        "boost":5
                                     }
                                  }
                               },
                               {
                                  "match":{
                                     "formatted_address":{
                                        "query":search_key,
                                        "boost":2
                                     }
                                  }
                               },
                               {
                                  "match":{
                                     "formatted_address.greeklish":{
                                        "query":search_key,
                                        "boost":2
                                     }
                                  }
                               },
                               {
                                  "match":{
                                     "types":{
                                        "query":search_key,
                                        "boost":3
                                     }
                                  }
                               },
                               {
                                  "match":{
                                     "types.greeklish":{
                                        "query":search_key,
                                        "boost":3
                                     }
                                  }
```

```
                        }
                    ]
                }
            },
            "functions":[
                {
                    "field_value_factor":{
                        "field":"rating",
                        "factor":1.2,
                        "modifier":"sqrt",
                        "missing":2
                    }
                },
                {
                    "gauss":{
                        "location":{
                            "origin":{
                                "lat":lat,
                                "lon":lon
                            },
                            "offset":"1km",
                            "scale":"2km"
                        }
                    },
                    "weight":1.2
                }
            ],
            "boost_mode":"avg"
        }
    }
}
```

Χρησιμοποιούμε τρία διαφορετικά queries, τα οποία κατασκευάζονται από αντίστοιχες μεθόδους. Το autocomplete query χρησιμοποιεί multi_match με τύπο phrase_prefix ώστε να επιστρέφει αποτελέσματα με βάση τις λέξεις/φράσεις που έχουν ως prefix την λέξη/φράση κλειδί που εισήγαγε ο χρήστης. Τα άλλα δύο queries είναι παρόμοια, με προσθήκη του γεωγραφικού κριτηρίου στο near query, το οποίο θα προσμετράται στην βαθμονόμηση (scoring) των αποτελεσμάτων που επιστρέφονται. Χρησιμοποιούν λοιπόν το bool query, συγκεκριμένα με should ερώτημα που σημαίνει ότι έστω ένα από τα match να ισχύει, το έγγραφο είναι κατάλληλο αποτέλεσμα. Τα τρία match γίνονται στο όνομα, την διεύθυνση και τον τύπο των places. Επίσης γίνονται τρία ίδια match στα sub-fields τους. Δίνουμε μεγαλύτερο βάρος στο πόσο σχετικό με την αναζήτηση είναι το όνομα, μετά ο τύπος και τέλος η διεύθυνση (βάρη 5, 3 και 1 αντίστοιχα). Αυτά τα βάρη θα χρησιμοποιηθούν στο εσωτερικό σύστημα βαθμολόγησης πλήρους κειμένου (full-text search) της Elasticsearch. Στην συνέχεια επιλέγουμε τις συναρτήσεις βαθμολόγησης (scoring) που θα χρησιμοποιήσουμε επιπλέον της βαθμολόγησης σχετικότητας κειμένου που γίνεται αυτόματα. Χρησιμοποιούμε λοιπόν την συνάρτηση field_value_factor για να συμπεριλάβουμε την βαθμολογία άλλων χρηστών (rating) για το κάθε place, με παράγοντα 1.2 και συνάρτηση υπολογισμού την τετραγωνική ρίζα., καθώς και χρησιμοποιώντας την τιμή 2 για όσα places δεν

έχουν καθόλου rating. Αυτές οι ρυθμίσεις προέκυψαν μετά από πειραματισμούς με τα αποτελέσματά μας, όπως προτείνεται από την ίδια την Elasticsearch. Τέλος, χρησιμοποιούμε την συνάρτηση gauss για να συμπεριλάβουμε και το γεωγραφικό κριτήριο στο scoring, χρησιμοποιώντας τις ρυθμίσεις offset 1 χιλιόμετρο και scale 2 χιλιόμετρα, δηλαδή το score θα είναι το ιδανικό αν το place βρίσκεται 1 χιλιόμετρο σε ακτίνα από τον χρήστη, ενώ το score θα πέσει στο 50% αν βρίσκεται σε 2 χιλιόμετρα. Δίνουμε βάρος 1.2 σε αυτή τη συνάρτηση για τον υπολογισμό του συνολικού score. Επιλέγουμε το τελικό score να είναι ο μέσος όρος του score που προκύπτει από αυτές τις συναρτήσεις και αυτό που δίνεται από την σχετικότητα κειμένου (boost_mode: avg).

Το αρχείο με τα links και τις αντιστοιχήσεις με τις κατάλληλες μεθόδους:

urls.py

```
from django.conf.urls import url

from . import views

app_name = 'searchapp'
urlpatterns = [
    url(r'^$', views.search_page, name='search_page'),
    url(r'^results/$', views.search_results, name='search_results'),
    url(r'^details/(?P<_id>[\w\-]+)', views.get_by_id,
name='get_by_id'),
    url(r'^live_search', views.live_search, name='live_search'),
    url(r'^rest/name/$', views.rest_name, name='rest_name'),
    url(r'^rest/autocomplete/$', views.rest_autocomplete,
name='rest_autocomplete'),
    url(r'^rest/id/$', views.rest_id, name='rest_id'),
    url(r'^rest/near/$', views.rest_near, name='rest_near'),
]
```

Το αρχείο apps.py με κάποιες αρχικοποιήσεις και δημιουργίες συνδέσεων με τις βάσεις:

apps.py

```
from __future__ import unicode_literals

from django.apps import AppConfig
from pymongo import MongoClient
from elasticsearch import Elasticsearch

# Establish Connection with Elasticsearch
elastic_client = Elasticsearch(['http://83.212.96.164:9200'])
index_name = 'elasticplaces'
max_size = 50

# Establish Connection with MongoDB
mongo_client = MongoClient()['elasticPlaces']['places']
```

```
class SearchappConfig(AppConfig):
    name = 'searchapp'
```

5.2. *Android εφαρμογή*

5.2.1. *Υλοποίηση της εφαρμογής*

Στο σημείο αυτό θα μελετήσουμε τον κώδικα της Android εφαρμογής. Λόγω του μεγάλου μεγέθους των αρχείων, θα επικεντρωθούμε στις σημαντικότερες και πιο ουσιώδεις μεθόδους της κάθε κλάσης.

MainActivity.java

```
private void displayListView() {

        dataAdapter = new MainCustomAdapter(this,
R.layout.live_result_item, (LayoutInflater)
getSystemService(Context.LAYOUT_INFLATER_SERVICE));

        listView.setAdapter(dataAdapter);
        listView.setTextFilterEnabled(true);
        listView.setEmptyView(findViewById(R.id.emptyElement));
        listView.setOnItemClickListener(new
AdapterView.OnItemClickListener() {
            public void onItemClick(AdapterView<?> parent, View view,
                                    int position, long id) {
                LiveResult selectedItem = (LiveResult)
parent.getItemAtPosition(position);


                Intent intent = new Intent(MainActivity.this,
DetailsActivity.class);
                intent.putExtra("id", selectedItem.getId());
                startActivity(intent);
            }
        });
        EditText editText = (EditText) findViewById(R.id.myFilter);
        editText.addTextChangedListener(new TextWatcher() {

            public void afterTextChanged(Editable s) {

            }

            public void beforeTextChanged(CharSequence s, int start,
int count, int after) {
                AppController.getInstance().cancelPendingRequests();
```

```
            }

            public void onTextChanged(CharSequence s, int start, int
before, int count) {
                dataAdapter.clear();
                try {
                    String input= "";
                    String url =
"http://83.212.96.164/searchapp/rest/autocomplete/?search=";
                    try {
                        input =
java.net.URLEncoder.encode(s.toString(), "utf-8");
                    }
                    catch (java.io.UnsupportedEncodingException e) {
                        e.printStackTrace();
                    }
                    url = url + input;
                    mainCommunicator.search(url);
                } catch (JSONException e) {
                    e.printStackTrace();
                }
            }
        });

        editText.setOnEditorActionListener(new
TextView.OnEditorActionListener() {
            @Override
            public boolean onEditorAction(TextView v, int actionId,
KeyEvent event) {
                if (actionId == EditorInfo.IME_ACTION_SEARCH) {
                    String input = "";
                    try {
                        input =
java.net.URLEncoder.encode(v.getText().toString(), "utf-8");
                    }
                    catch (java.io.UnsupportedEncodingException e) {
                        e.printStackTrace();
                    }
                    goToResultsActivity(input);
                    return true;
                }
                return false;
            }
        });
}
```

```
private void goToResultsActivity(String text) {
    Intent intent = new Intent(this, ResultsActivity.class);
    intent.putExtra("text", text);

    if ( ContextCompat.checkSelfPermission( this,
android.Manifest.permission.ACCESS_FINE_LOCATION ) !=
PackageManager.PERMISSION_GRANTED ) {

        ActivityCompat.requestPermissions( this, new String[] {
android.Manifest.permission.ACCESS_FINE_LOCATION  },
MY_PERMISSION_ACCESS_FINE_LOCATION );
    }
    if (((Switch) findViewById(R.id.NearMe)).isChecked()) {
        mLastLocation = LocationServices.FusedLocationApi
                .getLastLocation(mGoogleApiClient);
        if (mLastLocation != null) {
```

```
            double latitude = mLastLocation.getLatitude();
            double longitude = mLastLocation.getLongitude();
            intent.putExtra("lat", latitude);
            intent.putExtra("lon", longitude);

        }
    }
    startActivity(intent);
}
```

Η MainActivity αποτελεί την αρχική οθόνη της εφαρμογής. Με την μέθοδο displayListView επιτυγχάνεται η λειτουργικότητα του autocomplete, όπου δημιουργείται ένας Adapter, μέσα στον οποίο αποθηκεύονται προσωρινά τα αποτελέσματα. Με την προσθήκη ενός TextChangedListener γίνεται αναζήτηση στον server με κάθε νέα πληκτρολόγηση του χρήστη, και στην συνέχεια γεμίζει ξανά ο Adapter με τα νέα αποτελέσματα. Αντίστοιχα ένας ItemClickListener αναλαμβάνει την μετάβαση στην DetailsActivity σε περίπτωση επιλογής ενός εκ των αποτελεσμάτων, και ένας EditorActionListener καλεί την μέθοδο goToResults όταν ο χρήστης ολοκληρώσει την πληκτρολόγηση. Στην μέθοδο goToResults, μετά από έναν απλό έλεγχο για το αν είναι ενεργοποιημένη η λειτουργία NearMe, γίνεται η εκκίνηση της DetailsActivity με όλα τα απαραίτητα δεδομένα να βρίσκονται στο αντικείμενο intent.

ResultsActivity

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_results);
    listView = (ListView) findViewById(R.id.listView);
    displayListView((String)
getIntent().getSerializableExtra("text"));
}
```

```
private void displayListView(String searchText) {
    dataAdapter = new ResultsCustomAdapter(this, R.layout.result_item,
(LayoutInflater) getSystemService(Context.LAYOUT_INFLATER_SERVICE));
    listView.setAdapter(dataAdapter);
    listView.setTextFilterEnabled(true);
    listView.setEmptyView(findViewById(R.id.emptyElement));
    listView.setOnItemClickListener(new
AdapterView.OnItemClickListener() {
        public void onItemClick(AdapterView<?> parent, View view,
                                int position, long id) {
            Place selectedItem = (Place)
parent.getItemAtPosition(position);

            Intent intent = new Intent(ResultsActivity.this,
DetailsActivity.class);
            intent.putExtra("id", selectedItem.getId());
            startActivity(intent);
        }
    });
    dataAdapter.clear();
    try {
```

```
        String url =
"http://83.212.96.164/searchapp/rest/name/?search=" + searchText;
        Double lat = (Double) getIntent().getSerializableExtra("lat");
        Double lon = (Double) getIntent().getSerializableExtra("lon");
        if ((lat != null) && (lon != null)) {
            url = "http://83.212.96.164/searchapp/rest/near/?search="
+ searchText + "&lat=" + lat + "&lon=" + lon;
        }
        resultsCommunicator.search(url);
    } catch (JSONException e) {
        e.printStackTrace();
    }
}
```

Η κλάση ResultsActivity.java παρουσιάζει τα αποτελέσματα σε λίστα εξάγοντας κατά την δημιουργία της (κλήση OnCreate) την λέξη/φράση κλειδί που εισήγαγε ο χρήστης στην MainActivity. Στην μέθοδο displayListView εκτελείται η αναζήτηση και χρησιμοποιώντας ξανά έναν Adapter γίνεται η αποθήκευση κάθε Place. Με χρήση του ItemClickListener μεταβαίνει στην DetailsActivity σε περίπτωση επιλογής κάποιου Place, με μοναδική πληροφορία το Id.

DetailsActivity.java

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);

    setContentView(R.layout.activity_details);
    MapFragment mapFragment = (MapFragment) getFragmentManager()
            .findFragmentById(R.id.DetailsMap);
    mapFragment.getMapAsync(this);
    String id = (String) getIntent().getSerializableExtra("id");

    try {
        String url = "http://83.212.96.164/searchapp/rest/id/?id=" +
id;
        detailsCommunicator.search(url);
    } catch (JSONException e) {
        e.printStackTrace();
    }
}
```

```
public void setUpMap() {
    googleMap.setMapType(GoogleMap.MAP_TYPE_NORMAL);
    googleMap.setTrafficEnabled(true);
    googleMap.setIndoorEnabled(true);
    googleMap.setBuildingsEnabled(true);
    googleMap.getUiSettings().setZoomControlsEnabled(true);
}
```

Αμέσως μετά την δημιουργία της κλάσης (μέθοδος onCreate), εκτελείται η συλλογή όλων των πληροφοριών του Place, που προκύπτει από το δοσμένο Id, από τον server. Η μέθοδος setUpMap

προετοιμάζει τις παραμέτρους του χάρτη για την προβολή της τοποθεσίας του Place στο κάτω μέρος της οθόνης.

HttpCommunicator.java

```
public abstract void processResults(JSONObject jsonResults);
```

```
public void search(String url ) throws JSONException {
    JsonObjectRequest postRequest = new
JsonObjectRequest(Request.Method.GET, url, null,
            new Response.Listener<JSONObject>() {
                @Override
                public void onResponse(JSONObject jsonResponse) {
                        processResults(jsonResponse);
                }
            },
            new Response.ErrorListener() {
                @Override
                public void onErrorResponse(VolleyError volleyError) {
                    volleyError.printStackTrace();
                }
            }
    ) {
    };
    int socketTimeout = 30000; //milliseconds
    RetryPolicy policy = new DefaultRetryPolicy(socketTimeout,
DefaultRetryPolicy.DEFAULT_MAX_RETRIES,
DefaultRetryPolicy.DEFAULT_BACKOFF_MULT);
    postRequest.setRetryPolicy(policy);
    AppController.getInstance().addToRequestQueue(postRequest,
"json_obj_req");
}
```

Σε κάθε μια από τις προηγούμενες κλάσεις, χρειάστηκε σε κάποιο σημείο να γίνει σύνδεση με τον server και στη συνέχεια να επεξεργαστούν και να προβληθούν τα αποτελέσματα. Σε κάθε περίπτωση οι διαφοροποιήσεις είναι μικρές, και για αυτό τον λόγο δημιουργήθηκε ξεχωριστά η κλάση HttpCommunicator. Στην κλάση αυτή, μέσω της μεθόδου search και της abstract μεθόδου processResults επιτυγχάνονται η ανωτέρα λειτουργικότητα.

AppController

```
public void onCreate() {
        super.onCreate();
        mInstance = this;
    }
```

```
public static synchronized AppController getInstance() {
```

```
        return mInstance;
    }
public RequestQueue getRequestQueue() {
        if (mRequestQueue == null) {
            mRequestQueue =
Volley.newRequestQueue(getApplicationContext());
        }
        return mRequestQueue;
    }
public <T> void addToRequestQueue(Request<T> req, String tag) {
        req.setTag(TextUtils.isEmpty(tag) ? TAG : tag);
        getRequestQueue().add(req);
    }
public void cancelPendingRequests() {
        if (mRequestQueue != null) {
            mRequestQueue.cancelAll(TAG);
        }
    }
```

Στην κλάση AppController ορίζονται οι μέθοδοι, που μέσω της βιβλιοθήκης volley κάνουν εφικτή την επικοινωνία με τον server. Για την αποφυγή «race conditions» δημιουργείται μια ουρά από αιτήματα, με τις αντίστοιχες λειτουργίες προσθήκης και αφαίρεσης νέων ή αντίστοιχα διεκπεραιωμένων αιτήσεων.

Place.java

```
public String getId() { return id; }
public void setId(String id) { this.id = id; }
public String getName() { return name; }
public void setName(String name) { this.name = name; }
public String getType() { return type; }
public void setType(String type) { this.type = type;  }
public Float getRating() { return rating; }
public void setRating(String rating) { this.rating =
Float.parseFloat(rating);  }
public String getAddress() { return address; }
public void setAddress(String address) { this.address = address; }
public void setPhoneNumber(String phoneNumber) { this.phoneNumber =
phoneNumber; }
public String getPhoneNumber() { return phoneNumber; }
```

Η κλάση Place αντιπροσωπεύει τα αντίστοιχα Places από την βάση δεδομένων της MongoDB, με επιλεγμένα έξι πεδία τα οποία παρουσιάζουμε στον χρήστη στην DetailsView οθόνη. Στον κώδικα φαίνονται όλες οι μέθοδοι getters και setters.

LiveResult.java

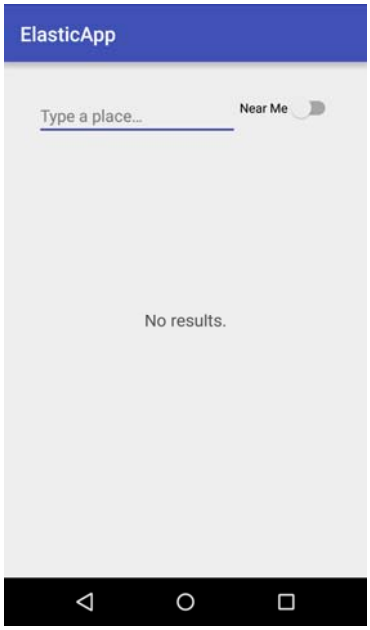
```
public String getId() { return id; }
public void setId(String id) { this.id = id; }
public String getName() { return name; }
public void setName(String name) { this.name = name; }
```

Αντίστοιχα για τα αντικείμενα των αρχικών αποτελεσμάτων του autocomplete χρησιμοποιείται η κλάση LiveResult με τους δικούς τις getters και setters.

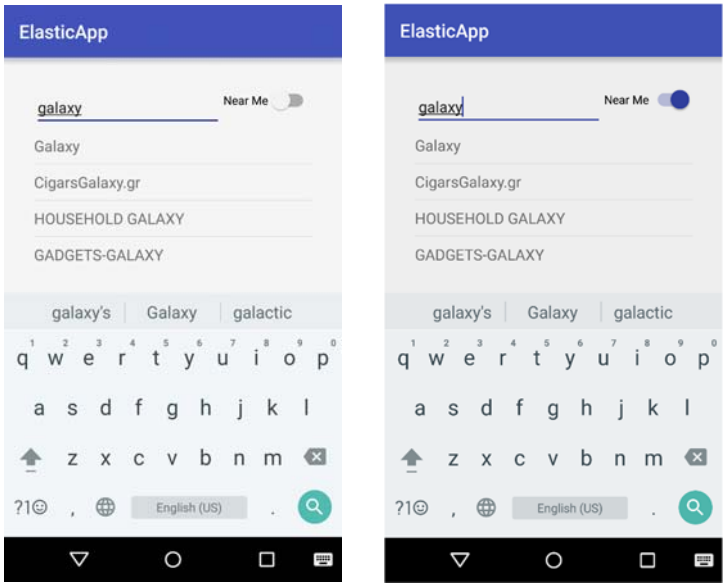
5.2.2. Παράδειγμα Χρήσης

Στην ενότητα αυτή θα περιγράψουμε ένα παράδειγμα χρήσης της εφαρμογής από έναν υποθετικό χρήστη, για την καλύτερη κατανόηση τόσο της λειτουργικότητας της εφαρμογής όσο και της αλληλεπίδρασης του χρήστη με αυτή.

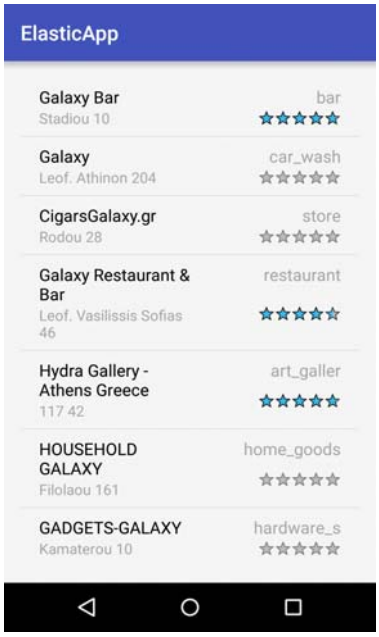
Ο χρήστης ξεκινά με την έναρξη της εφαρμογής και αμέσως εμφανίζεται η αρχική οθόνη.



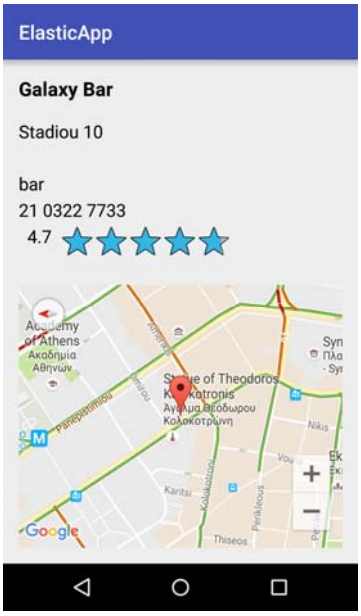
Η ετικέτα “Type a place…” υποδεικνύει το πλαίσιο μέσα στο οποίο πρέπει να επιλέξει ο χρήστης ώστε να εμφανιστεί το πληκτρολόγιο. Στη συνέχεια μπορεί να πληκτρολογήσει την λέξη/φράση που τον ενδιαφέρει, ενώ με κάθε μια επιλογή πλήκτρου η εφαρμογή παρουσιάζει στην λίστα τα αποτελέσματα της λειτουργίας autocomplete.



Σε αυτό το σημείο ο χρήστης μπορεί να διαλέξει 3 διαφορετικούς τρόπους να συνεχίσει. Αρχικά μπορεί να επιλέξει αμέσως ένα από τα αποτελέσματα που του προτάθηκαν στην λίστα και να μεταβεί στην τελική οθόνη DetailsActivity, ή να εκτελέσει την αναζήτηση πατώντας το εικονίδιο ολοκλήρωσης εισαγωγής. Στο πάνω-δεξιό άκρο της οθόνης υπάρχει «διακόπτης» Near Me ο οποίος αφού η εφαρμογή διαπιστώσει πως οι υπηρεσίες τοποθεσίας είναι διαθέσιμες, μπορεί ο χρήστης να τον ενεργοποιήσει ή να τον απενεργοποιήσει, προσθέτοντας έτσι ή αφαιρώντας αντίστοιχα στην αναζήτηση γεωγραφικά κριτήρια.



Μετά την ολοκλήρωση της εισαγωγής και τη μετάβαση στην οθόνη αποτελεσμάτων, ο χρήστης μπορεί να δει μια λίστα από Places, να κάνει scroll down, και να επιλέξει ένα από αυτά ώστε να μεταβεί στην τελευταία οθόνη που περιέχονται όλες οι λεπτομέρειες.



Τέλος, εδώ μπορεί να δει όλες τις λεπτομέρειες και να χρησιμοποιήσει τον διαδραστικό χάρτη στο κάτω μέρος της οθόνης.

5.3. *Web εφαρμογή*

5.3.1. Υλοποίηση της εφαρμογής

Θα παρουσιάσουμε σε αυτό το σημείο τα templates με βάση τα οποία δημιουργούνται οι οθόνες της εφαρμογής.

Το αρχείο base.html αποτελεί τον σκελετό όλων των σελίδων, καθώς περιέχει στοιχεία που περιέχονται σε όλη την εφαρμογή αμετάβλητα όπως ο τίτλος και το footer.

base.html

```
<html>
  <head>
      <link
href="http://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min
.css" rel="stylesheet" />
      <link
href='https://fonts.googleapis.com/css?family=Roboto:500,300,700,400'
rel='stylesheet' type='text/css'>
      <link rel="stylesheet" type="text/css"
href="/static/searchapp/search.css" />
      <script src="/static/searchapp/assets/js/jquery-
2.0.0.min.js"></script>
      <script src="/static/searchapp/assets/js/ajax.js"></script>
```

```
      {% block extra-head %}{% endblock %}
  </head>

  <body>
    <div class="header">
      <div class="container">
        <h1><a href="/searchapp" style="text-decoration : none; color
: #000000;"> Elastic Web </a></h1>
      </div>
    </div>
    <div class="main">
      <div class="container">
        <div class="row">
          <div id="custom-search-input">
            {% block inside-green-layout %}{% endblock %}
          </div>
        </div>
      </div>
    </div>
    <div class="footer">
      <div class="container">
        <h2>Available for iPhone and Android.</h2>
        <img src="/static/searchapp/img/app-store.png" width="360px"
/>
        <img src="/static/searchapp/img/google-play-badge.png"
width="360px" />
      </div>
    </div>
  </body>
</html>
```

Το αρχείο search.html, το οποίο δημιουργεί την search bar με χρήση φόρμας <form>, και το χώρο όπου θα εμφανιστούν τα αρχικά autocomplete αποτελέσματα σε μια λίστα <ul>.

search.html

```
{% extends "searchapp/base.html" %}

{% block extra-head %} <title> Search </title> {% endblock %}

{% block inside-green-layout %}

<h3>Elastic Search Box</h3>
{% csrf_token %}
<form class="input-group col-md-12" method="get"
action="/searchapp/results/">
  <input type="text" class="  search-query form-control"
placeholder="Search for a place" id="search" name="search"
autocomplete="off" />
    <span class="input-group-btn" method="get"
action="/searchapp/search/">
      <button class="btn btn-danger" type="submit" >
        <span class=" glyphicon glyphicon-search"></span>
      </button>
    </span>
</form>
<ul id="search-results" class="list-unstyled col-md-6">
</ul>
```

```
{% endblock %}
```

Το αρχείο results.html, το οποίο παρουσιάζει την λίστα των αποτελεσμάτων κάνοντας χρήση πίνακα <table> για τις επιμέρους λεπτομέρειες, αλλά και την σελιδοποίηση ανά ομάδα.

results.html

```
{% extends "searchapp/base.html" %}

{% block extra-head %} <title> Results </title> {% endblock %}

{% block inside-green-layout %}
<table class="table table-hover">
    <thead>
      <tr>
        <th>Name</th>
        <th>Address</th>
        <th>Rating</th>
      </tr>
    </thead>
    <tbody>
    {% for result in res %}
        <tr>
            <td><a href="/searchapp/details/{{ result.id  }}"> {{
result.name  }}</a></td>
            <td><a href="/searchapp/details/{{ result.id  }}"> {{
result.formatted_address  }}</a></td>
            <td><a href="/searchapp/details/{{ result.id  }}"> {{
result.rating  }}</a></td>
          </tr>
    {% endfor %}
  </tbody>
</table>
<div class="pagination">
    <span class="step-links">
        {% if res.has_previous %}
        <a href="?search={{search_key}}&page={{
res.previous_page_number }}">previous</a>
        {% endif %}
        <span class="current">
            Page {{ res.number }} of {{ res.paginator.num_pages }}.
        </span>
        {% if res.has_next %}
        <a href="?search={{search_key}}&page={{ res.next_page_number
}}">next</a>
        {% endif %}
    </span>
</div>

{% endblock %}
```

Το αρχείο details.html, το οποίο παρουσιάζει όλες τις πληροφορίες για το επιλεγμένο Place σε μορφή πίνακα <table>, αλλά και την ακριβή τοποθεσία σε χάρτη κάνοντας χρήση ενός <script>.

details.html

```
{% extends "searchapp/base.html" %}

{% block extra-head %}
<script type="text/javascript"
src="http://maps.google.com/maps/api/js?sensor=false"></script>
<script
src="http://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/js/bootstrap.min.j
s"></script>
<script
src="https://ajax.googleapis.com/ajax/libs/jquery/1.12.0/jquery.min.js
"></script>
<script>
    $(document).ready(function() {
        var latlng = new google.maps.LatLng("{{
res.geometry.location.lat }}", "{{ res.geometry.location.lng }}");
        var mapOptions = {
            zoom: 15,
            center: latlng,
            mapTypeControl: false,
            navigationControlOptions: {style:
google.maps.NavigationControlStyle.SMALL},
            mapTypeId: google.maps.MapTypeId.ROADMAP
        };
        map = new google.maps.Map($('.map')[0], mapOptions);

        var marker = new google.maps.Marker({
            position: latlng,
            map: map,
            title:"{{ res.name }}"
        });
    });
</script>
<title> Details </title>
{% endblock %}

{% block inside-green-layout %}
<div class="col-md-7 col-xs-6">
    <table class="table">
        <tbody>
            <tr>
                <td> Name </td>
                <td> {{ res.name }} </td>
            </tr>
            <tr>
                <td> Rating </td>
                <td> {{ res.rating }} </td>
            </tr>
            <tr>
                <td> Website </td>
                <td> <a href="{{ res.website }}"> {{ res.website }}
</a></td>
            </tr>
            <tr>
                <td> Address </td>
                <td> {{ res.formatted_address }} </td>
```

```
            </tr>
            <tr>
                <td> Phone Number </td>
                <td> {{ res.formatted_phone_number }} </td>
            </tr>
            <tr>
                <td> Types </td>
                <td>
                {% for type in res.types %}
                    {{ type }}          
                {% endfor %}
                </td>
            </tr>
            <tr>
                <td> <a href="#demo" data-toggle="collapse">Reviews
&darr;</a></td>
                <td> {{ res.user_ratings_total }} </td>
            </tr>
            <tr><td></td><td></td></tr>
        </tbody>
    </table>
    <div id="demo" class="collapse">
        {% for review in res.reviews %}
            <li class="list-group-item"> {{ review.author_name }}  :
{{ review.rating }} &#9733;
            {{ review.text }} </li>
        {% endfor %}
    </div>
</div>
<div class="col-md-5 ">
    <div class="map" style="width: 300px; height: 300px;"></div>
</div>
{% endblock %}
```

Το αρχείο ajax_search.html, το οποίο αναλαμβάνει να δημιουργήσει τα αρχικά αποτελέσματα autocomplete σε λίστα <li>, τα οποία θα προστεθούν στον χώρο της search.html.

ajax_search.html

```
<link
href="http://maxcdn.bootstrapcdn.com/bootstrap/3.3.6/css/bootstrap.min
.css" rel="stylesheet" />
<link
href='https://fonts.googleapis.com/css?family=Roboto:500,300,700,400'
rel='stylesheet' type='text/css'>
<link rel="stylesheet" type="text/css"
href="/static/searchapp/search.css" />

{% if results.count > 0 %}

{% for result in results %}
    <li class="list-group-item">
        <a href="details/{{ result.id  }}"> {{ result.name  }}, {{
result.formatted_address }}
    </li>
{% endfor %}
```

```
{% else %}

<li>None to show!</li>

{% endif %}
```

Το αρχείο ajax.js, το οποίο παρέχει την λειτουργικότητα της αναζήτησης με κάθε νέα πληκτρολόγηση του χρήστη, χωρίς να περνάμε σε διαφορετική html σελίδα.

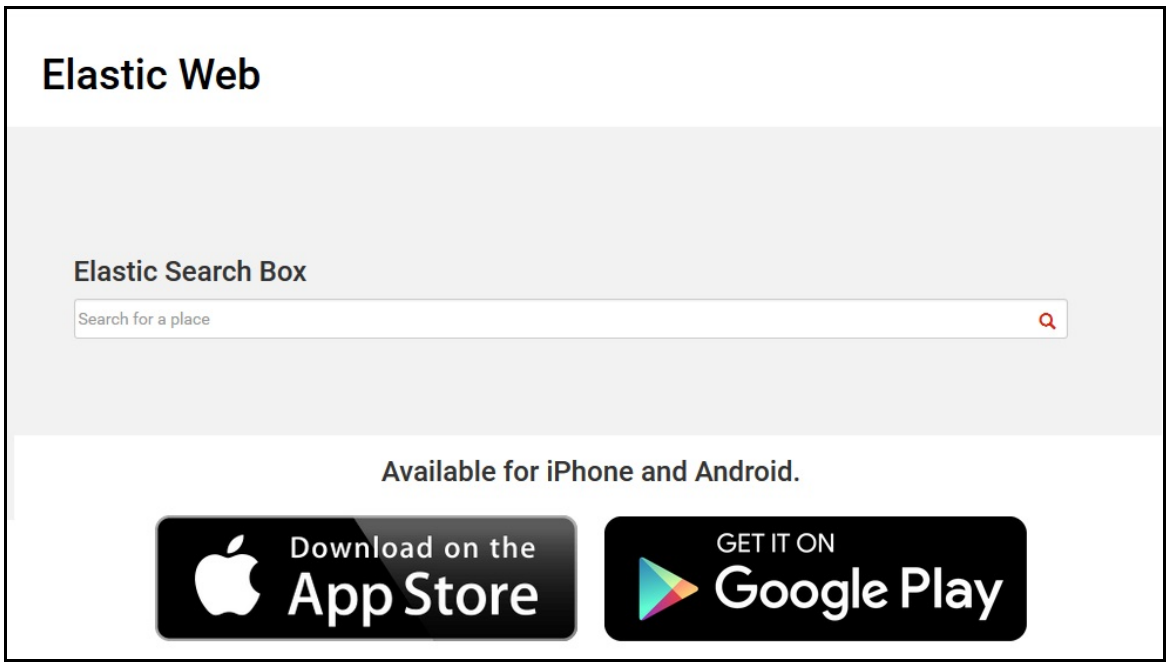
ajax.js

```
$(function(){

    $('#search').keyup(function() {

        $.ajax({
            type: "POST",
            url: "/searchapp/live_search/",
            data: {
                'search_text' : $('#search').val(),
                'csrfmiddlewaretoken' :
$("input[name=csrfmiddlewaretoken]").val()
            },
            success: searchSuccess,
            dataType: 'html'
        });

    });

});

function searchSuccess(data, textStatus, jqXHR)
{
    $('#search-results').html(data);
}
```

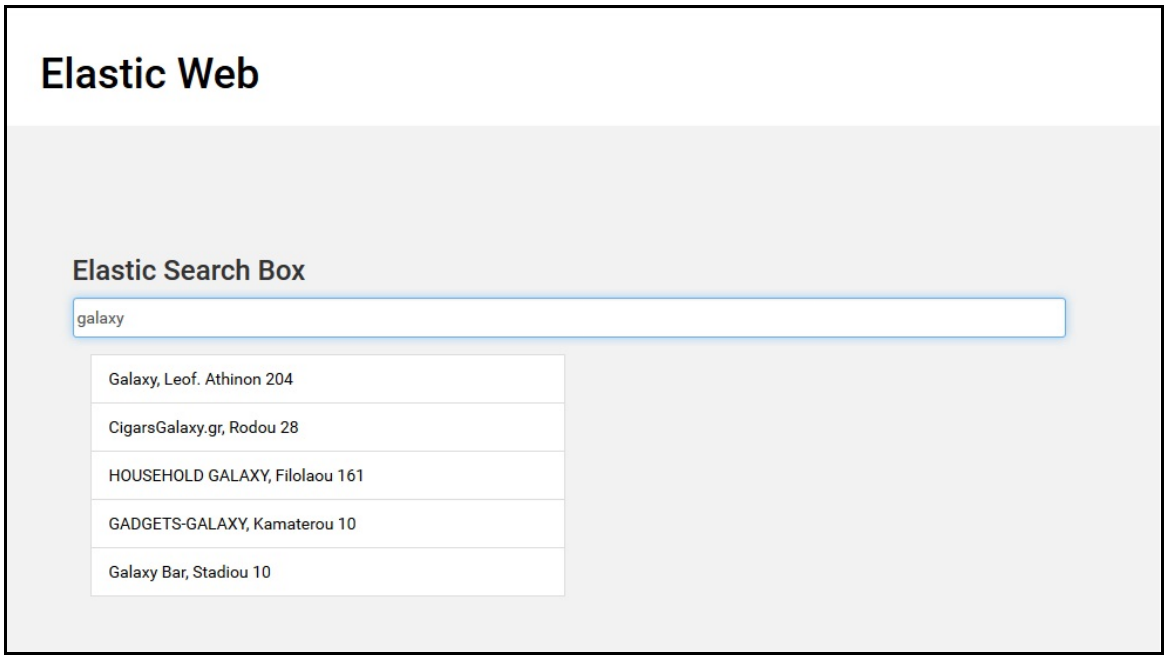
5.3.2. Παράδειγμα Χρήσης

Αντιστοίχως με προηγούμενη ενότητα όπου παρουσιάσαμε ένα παράδειγμα χρήσης για την εφαρμογή Android, τώρα θα περιγράψουμε ένα παράδειγμα χρήσης για την Web εφαρμογή.

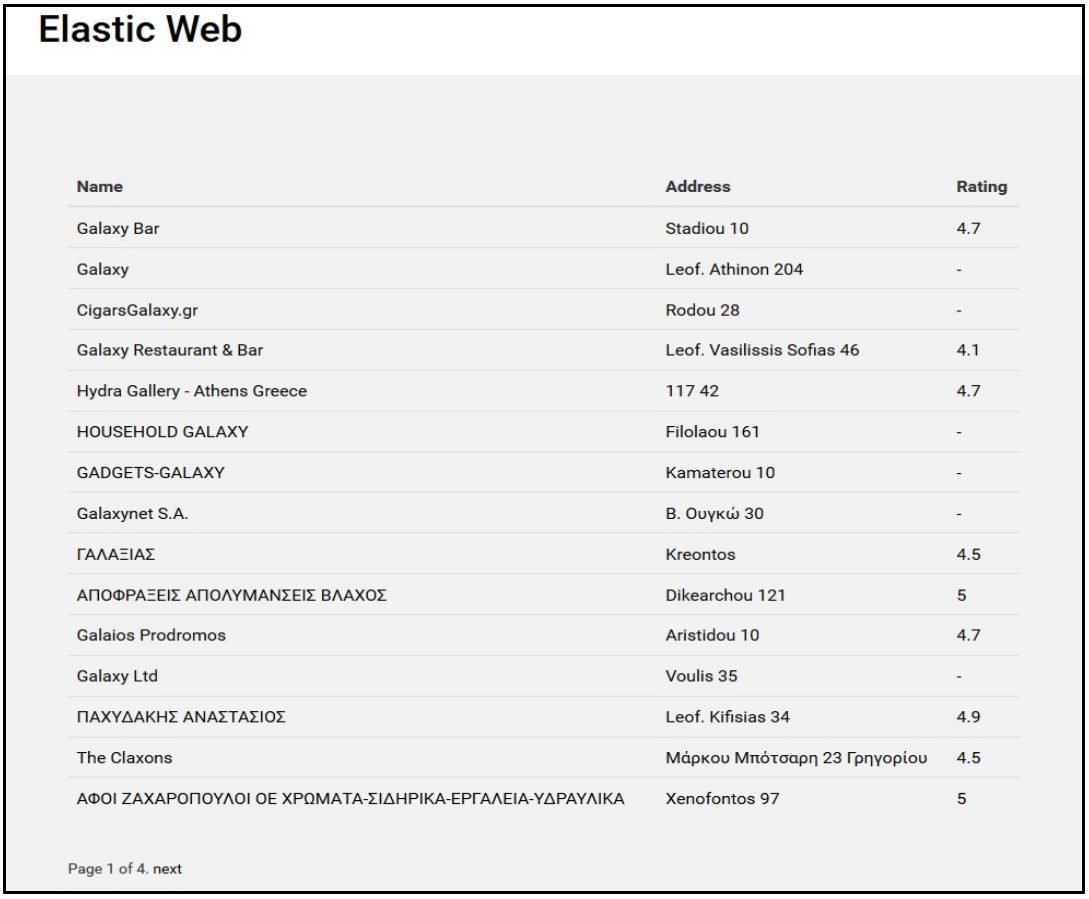
Στην επόμενη εικόνα δείχνουμε την αρχική οθόνη όπου ο χρήστης μπορεί να επιλέξει το πλαίσιο «Search for a place» και να αρχίσει να πληκτρολογεί.



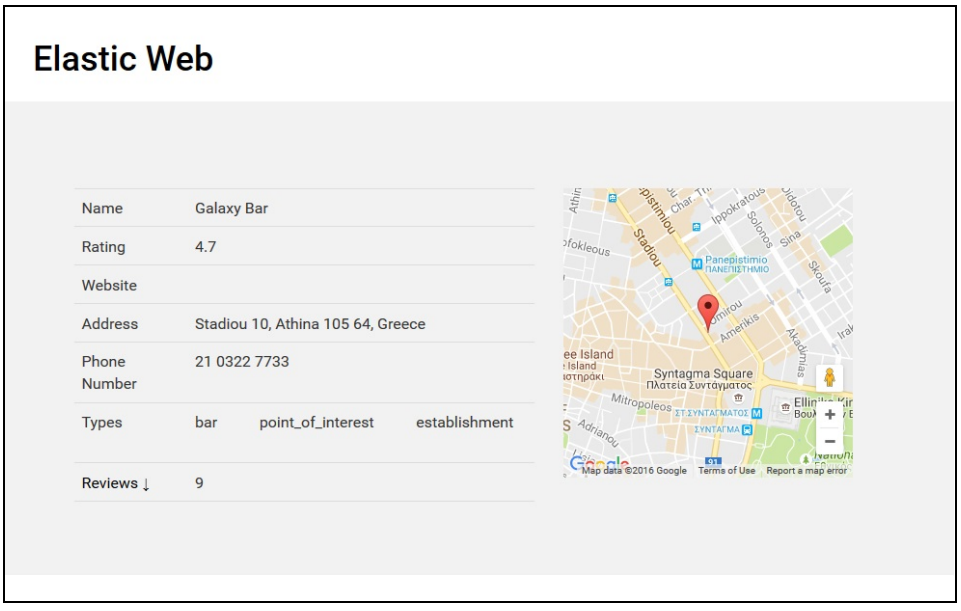
Με την εισαγωγή του κάθε γράμματος παρουσιάζονται τα αρχικά αποτελέσματα του autocomplete.



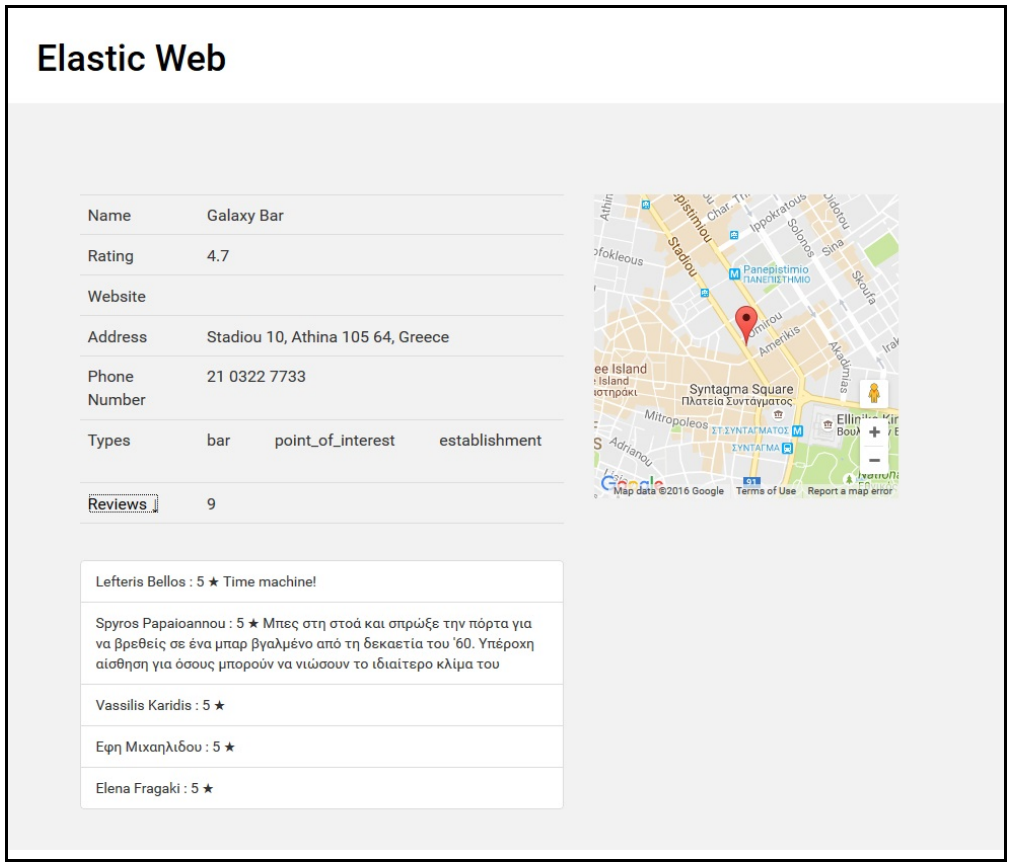
Με την επιλογή ενός εκ των αποτελεσμάτων θα μπορούσε ο χρήστης να μεταβεί απ' ευθείας στην τελική σελίδα λεπτομερειών, ή με το τέλος εισαγωγής («enter») μεταβαίνει στην σελίδα αποτελεσμάτων.



Τα αποτελέσματα ομαδοποιούνται (σελιδοποίηση) ανά 15, και μπορεί ο χρήστης να αλλάξει ομάδα με το κουμπί next στο κάτω μέρος της οθόνης. Σε κάθε περίπτωση μετά την επιλογή κάποιου αποτελέσματος εμφανίζονται όλες οι λεπτομέρειες με τον χάρτη τοποθεσίας του στην οθόνη λεπτομερειών.



Το κείμενο κάθε «κριτικής» είναι αρχικά κρυμμένο για εξοικονόμηση χώρου. Η λεζάντα «Reviews» χρησιμεύει και ως διακόπτης για την εμφάνιση τους.



5.4. Elasticsearch - Δημιουργία του Index

Παρακάτω φαίνεται το πρόγραμμα σε Python το οποίο τρέχουμε για την δημιουργία του index της Elasticsearch το οποίο θα χρησιμοποιείται για τις αναζητήσεις.

index.py

```
# coding: utf-8

from elasticsearch import Elasticsearch
es = Elasticsearch(['http://83.212.96.164:9200'])

es.indices.delete(index='elasticplaces', ignore=[400, 404])


es.indices.create(
    index="elasticplaces",
    body={
        "settings": {
            "analysis": {
                "char_filter": {
```

```
                "my_mapping": {
                    "type": "mapping",
                    "mappings" : [
            "α => a", "η => i",  "ν => n",  "υ => y",  "ΕΥ => ef",
            "ά => a", "ή => i",  "Ν => n",  "Υ => y",  "αυ => af", "ου => ou",
            "Α => a", "Η => i",  "ξ => ks", "φ => f",  "αύ => af", "ού => ou",
            "Ά => a", "Ή => i",  "Ξ => ks", "Φ => f",  "Αυ => af", "Ου => ou",
            "β => b", "θ => th", "ο => o",  "χ => x",  "ΑΥ => af", "Ού => ou",
            "Β => b", "Θ => th", "ό => o",  "Χ => x",  "γκ => g",  "ΟΥ => ou",
            "γ => g", "ι => i",  "Ο => o",  "ψ => ps", "Γκ => g",  "ύ => y",
            "Γ => g", "ί => i",  "Ό => o",  "Ψ => ps", "ΓΚ => g",
            "δ => d", "Ι => i",  "π => p",  "ω => o",  "γγ => g",
            "Δ => d", "Ί => i",  "Π => p",  "ώ => o",  "Γγ => g",
            "ε => e", "κ => k",  "ρ => r",  "Ω => o",  "ΓΓ => g",
            "έ => e", "Κ => k",  "Ρ => r",  "Ώ => o",  "μπ => b",
            "Ε => e", "λ => l",  "σ => s",  "ς => s",  "ευ => ef", "Μπ => b",
            "Έ => e", "Λ => l",  "Σ => s",  "οι => i", "εύ => ef", "ΜΠ => b",
            "ζ => z", "μ => m",  "τ => t",  "οί => i", "Ευ => ef",
            "Ζ => z", "Μ => m",  "Τ => t",  "Οι => i", "Εύ => ef"
                    ]
                },
                "my_mapping_two": {
                    "type": "mapping",
                    "mappings" : [
            "α => a", "η => h",  "ν => n", "υ => i", "Οί => i",  "ΕΥ => ef", "ΝΤ => d",
            "ά => a", "ή => h",  "Ν => n", "Υ => i", "ΟΙ => i",  "αυ => af", "ου => ou",
            "Α => a", "Η => h",  "ξ => ks","φ => f", "αι => e",  "αύ => af", "ού => ou",
            "Ά => a", "Η => h",  "Ξ => ks","Φ => f", "αί => e",  "Αυ => af", "Ου => ou",
            "β => v", "θ => th", "ο => o", "χ => x", "Αι => e",  "ΑΥ => af", "Ού => ou",
            "Β => v", "Θ => th", "ό => o", "Χ => x", "Αί => e",  "γκ => g",  "ΟΥ => ou",
            "γ => g", "ι => i",  "Ο => o",  "ψ => ps", "ΑΙ => e",  "Γκ => g",  "ύ => i",
            "Γ => g", "ί => i",  "Ό => o",  "Ψ => ps", "ει => i",  "ΓΚ => g",
            "δ => d", "Ι => i",  "π => p",  "ω => w",  "εί => i",  "γγ => g",
            "Δ => d", "Ί => i",  "Π => p",  "ώ => w",  "Ει => i",  "Γγ => g",
            "ε => e", "κ => k",  "ρ => r",  "Ω => w",  "Εί => i",  "ΓΓ => g",
            "έ => e", "Κ => k",  "Ρ => r",  "Ώ => w",  "ΕΙ => i",  "μπ => b",
            "Ε => e", "λ => l",  "σ => s",  "ς => s",  "ευ => ef", "Μπ => b",
            "Έ => e", "Λ => l",  "Σ => s",  "οι => i", "εύ => ef", "ΜΠ => b",
            "ζ => z", "μ => m",  "τ => t",  "οί => i", "Ευ => ef", "ντ => d",
            "Ζ => z", "Μ => m",  "Τ => t",  "Οι => i", "Εύ => ef", "Ντ => d"
                    ]
                }
            },
            "filter" : {
                "myGreekLowerCaseFilter" : {
                    "type" : "lowercase",
                    "language" : "greek"
                },
                "ngrams_filter": {
                    "type": "ngram",
                    "min_gram": 3,
                    "max_gram": 8
                }
            },
            "analyzer": {
                "my_analyzer": {
                    "type": "custom",
                    "char_filter": ["my_mapping"],
                    "tokenizer": "standard",
                    "filter": ["lowercase", "myGreekLowerCaseFilter",
"ngrams_filter"]
```

```
                    },
                    "my_analyzer_two": {
                        "type": "custom",
                        "char_filter": ["my_mapping_two"],
                        "tokenizer": "standard",
                        "filter": ["lowercase", "myGreekLowerCaseFilter",
"ngrams_filter"]
                    }
                }
            }
        },
        "mappings": {
            "places": {
                "properties": {
                    "name": {
                        "type": "string",
                        "analyzer": "my_analyzer",
                        "fields" : {
                            "greeklish": {
                                "type": "string",
                                "analyzer": "my_analyzer_two"
                            }
                        }
                    },
                    "formatted_address": {
                        "type": "string",
                        "analyzer": "my_analyzer",
                        "fields" : {
                            "greeklish": {
                                "type": "string",
                                "analyzer": "my_analyzer_two"
                            }
                        }
                    },
                    "types": {
                        "type": "string",
                        "analyzer": "my_analyzer",
                        "fields" : {
                            "greeklish": {
                                "type": "string",
                                "analyzer": "my_analyzer_two"
                            }
                        }
                    },
                    "location": {
                        "type": "geo_point"
                    }
                }
            }
        }
    },
    # Will ignore 400 errors, remove to ensure you're prompted
    ignore=400
)
```

Αρχικά διαγράφουμε το index αν υπάρχει ήδη, αφού θέλουμε να δημιουργήσουμε ένα νέο χωρίς δεδομένα μέσα. Στη συνέχεια δημιουργούμε μέσω της μεθόδου indices.create() του Python API

της Elasticsearch το index, με όνομα “elasticplaces”, και στο σώμα του query θέτουμε τις διάφορες ρυθμίσεις που απαιτούνται, όπως αυτές περιεγράφηκαν στο υπο-κεφάλαιο 4.5.4.

5.5. Συγχρονισμός

5.5.1. Υλοποίηση

Στις επόμενες υπο-ενότητες θα παρουσιάσουμε τις μεθόδους συγχρονισμού που χρησιμοποιήσαμε και τις διάφορες απαραίτητες παραμετροποιήσεις της καθεμιάς. Όλες οι εντολές που αναφέρονται παρακάτω χρησιμοποιούνται σε terminal λειτουργικού Ubuntu 14.04 LTS.

5.5.1.1. Elasticsearch River

Ενεργοποιούμε την χρήση river στο index της Elasticsearch που χρησιμοποιούμε με την εξής εντολή, την οποία στέλνουμε με HTTP request στη θύρα 9200 του localhost στην οποία «ακούει» η Elasticsearch:

```
curl -XPUT "localhost:9200/_river/es_river/_meta" -d'{
    "type": "mongodb",
    "mongodb": {
        "db": "elasticPlaces",
        "collection": "places",
        "options": {
            "include_fields": ["_id", "name", "formatted_address",
"types", "geometry.location", "rating"]
        }
        },
    "index": {
        "name": "elasticplaces",
        "type": "places"
    }
}'
```

Ο τύπος του river είναι «mongodb», με το οποίο καταλαβαίνει το σύστημα ότι θα συγχρονίσουμε δεδομένα που θα διαβάζονται από κάποια βάση σε σύστημα MongoDB, οπότε και ενεργοποιείται το κατάλληλο plugin, το οποίο παίρνει ως ορίσματα την βάση και την συλλογή από τα οποία θα διαβάσει, καθώς και κάποιες επιλογές (options). Συγκεκριμένα χρησιμοποιήσαμε την επιλογή «include_fields», με την οποία καθορίζουμε ποια πεδία θα εισάγονται στο index, καθώς αυτά είναι τα χρήσιμα για αναζητήσεις.

Απενεργοποιούμε την χρήση river στο index με την εξής εντολή:

```
curl -XDELETE 'http://localhost:9200/_river/'
```

5.5.1.2. Mongo-connector

Χρησιμοποιούμε την παρακάτω εντολή προκειμένου να ενεργοποιήσουμε το πρόγραμμα mongo-connector ώστε να ξεκινήσει να συγχρονίζει δεδομένα:

```
mongo-connector -c config.json
```

Το αρχείο config.json είναι το αρχείο παραμετροποίησης του mongo-connector με όλες τις απαραίτητες ρυθμίσεις, και φαίνεται παρακάτω:

```
{
    "__comment__": "Configuration options starting with '__' are
disabled",
    "__comment__": "To enable them, remove the preceding '__'",

    "mainAddress": "localhost:27017",
    "oplogFile": "oplog.timestamp",
    "noDump": false,
    "batchSize": -1,
    "verbosity": 2,
    "continueOnError": false,

    "logging": {
        "type": "file",
        "filename": "mongo-connector.log",
        "__format": "%(asctime)s [%(levelname)s] %(name)s:%(lineno)d -
%(message)s",
        "__rotationWhen": "D",
        "__rotationInterval": 1,
        "__rotationBackups": 10,

        "__type": "syslog",
        "__host": "localhost:514"
    },

    "authentication": {
        "__adminUsername": "username",
        "__password": "password",
        "__passwordFile": "mongo-connector.pwd"
    },

    "__comment__": "For more information about SSL with MongoDB, please
see http://docs.mongodb.org/manual/tutorial/configure-ssl-clients/",
    "__ssl": {
```

```
        "__sslCertfile": "Path to certificate to identify the local
connection against MongoDB",
        "__sslKeyfile": "Path to the private key for sslCertfile. Not
necessary if already included in sslCertfile.",
        "__sslCACerts": "Path to concatenated set of certificate
authority certificates to validate the other side of the connection",
        "__sslCertificatePolicy": "Policy for validating SSL
certificates provided from the other end of the connection. Possible
values are 'required' (require and validate certificates), 'optional'
(validate but don't require a certificate), and 'ignored' (ignore
certificates)."
    },

    "fields": ["_id", "name", "formatted_address", "types",
"geometry.location", "rating"],

    "namespaces": {
        "__include": ["db.source1", "db.source2", "db2.*"],
        "__exclude": ["db3.source1", "db3.*"],
        "__mapping": {
            "db.source1": "db.dest1",
            "db.source2": "db.dest2",
            "db2.*": "db2_map_*.dest"
        },
    "__gridfs": ["db.fs"]
    },

    "docManagers": [
        {
            "docManager": "elastic2_doc_manager",
            "targetURL": "localhost:9200",
            "__bulkSize": 1000,
            "__uniqueKey": "_id",
            "__autoCommitInterval": null
        }
    ]
}
```

Όλες οι ρυθμίσεις με "__" στην αρχή είναι απενεργοποιημένες. Συγκεκριμένα χρησιμοποιούμε εδώ την ρύθμιση docManager, προκειμένου να ενεργοποιήσουμε το πακέτο του mongo-connector για Elasticsearch, την ρύθμιση targetURL για να του υποδείξουμε σε ποια διεύθυνση και θύρα «ακούει» η Elasticsearch, και τέλος την ρύθμιση fields για να κρατήσει μόνο τα συγκεκριμένα πεδία κατά τον συγχρονισμό.

5.5.1.3. Transporter

Χρησιμοποιούμε την παρακάτω εντολή για να ενεργοποιήσουμε το πρόγραμμα Transporter ώστε να ξεκινήσει να συγχρονίζει τις δύο βάσεις:

```
transporter run --config ~/Desktop/transporter\ config/config.yaml
~/Desktop/transporter\ config/mongo_es_sync.js
```

Οπότε παίρνει το αρχείο παραμετροποίησης (configuration file) config.yaml από το path που του δώσαμε και το JavaScript αρχείο mongo_es_sync.js που προσδιορίζει το πώς θα γίνει ο συγχρονισμός.

Το config.yaml:

```
# api:
#   interval: 60s
#   uri: "http://requestb.in/13gerls1"
#   key: "48593282-b38d-4bf5-af58-f7327271e73d"
#   pid: "something-static"
nodes:
  localmongo:
    type: mongodb
    uri: mongodb://localhost/elasticPlaces
    tail: true
  es:
    type: elasticsearch
    uri: http://localhost:9200/
  timeseries:
    type: influx
    uri: influxdb://root:root@localhost:8086/compose
  debug:
    type: file
    uri: stdout://
  foofile:
    type: file
    uri: file:///tmp/elasticPlaces
```

Περιέχει ρυθμίσεις για το που βρίσκονται οι βάσεις και τα αρχεία για logs κλπ.

Το mongo_es_sync.js:

```
// create a pipeline that reads documents from a file, transforms
them, and writes them
Source({name:"localmongo", namespace:"elasticPlaces.places"})
.transform({filename: "transformer.js",
namespace:"elasticPlaces.places"})
.save({name:"es", namespace:"elasticplaces.places"});
```

Προσδιορίζει τα namespaces για τη συλλογή και τη βάση στη MongoDB, το index και type στην Elasticsearch καθώς και το αρχείο του μετατροπέα (transformer), το οποίο είναι ένα JavaScript αρχείο το οποίο χρησιμοποιείται για την επεξεργασία του κάθε εγγράφου κατά το πέρασμα από την μία βάση στην άλλη.

Το transformer.js:

```
module.exports = function(doc) {
  var new_data = {};
  new_data["_id"] = doc["data"]["_id"]["$oid"];
  new_data["geometry"] = doc["data"]["geometry"];
  new_data["name"] = doc["data"]["name"];
  new_data["formatted_address"] = doc["data"]["formatted_address"];
  new_data["rating"] = doc["data"]["rating"];
  new_data["types"] = doc["data"]["types"];
  doc["data"] = new_data
  //console.log("transformer: " + JSON.stringify(doc));
  return doc
}
```

5.5.1.4. Υλοποίηση λύσης συγχρονισμού σε Python

```
# coding: utf-8

import pymongo, pyelasticsearch, elasticsearch
from pymongo import MongoClient
from bson.objectid import ObjectId
from elasticsearch import helpers


# Establish Connection with Elasticsearch
elastic_client_official   = elasticsearch.Elasticsearch()
elastic_client_unofficial = pyelasticsearch.client.ElasticSearch()


# Establish Connection with MongoDB
mongo_client = MongoClient()['elasticPlaces']['places']
bulk_size = 1000

# using the official elasticsearch api for python
def sync_official():
    array = []
    packet = 0
    for place in mongo_client.find({}, {"name": 1, "types": 1,
"formatted_address": 1, "rating": 1, "geometry.location": 1}):
        place["_id"] = str(place["_id"])
        place["_index"] = "elasticplaces"
        place["_type"] = "places"
        array.append(place)
        packet += 1

        if (packet == bulk_size):
            helpers.bulk(elastic_client_official, array)
            packet = 0
            array = []

    if (array):
        helpers.bulk(elastic_client_official, array)

# using the pyelasticsearch unofficial api
def sync_unofficial():
    array = []
    packet = 0
    for place in mongo_client.find({}, {"name": 1, "types": 1,
"formatted_address": 1, "rating": 1, "geometry.location": 1}):
```

```
        place["_id"] = str(place["_id"])
        array.append(place)
        packet += 1
        if (packet == bulk_size):
            elastic_client_unofficial.bulk_index("elasticplaces",
"places", array, id_field="_id")
            packet = 0
            array = []
    if (array):
        elastic_client_unofficial.bulk_index("elasticplaces",
"places", array, id_field="_id")

sync_official()
```

Υλοποιήσαμε δύο διαφορετικές μεθόδους για συγχρονισμό χρησιμοποιώντας το επίσημο και το ανεπίσημο Python API για να συγκρίνουμε την επίδοσή τους, και διαπιστώσαμε ότι είναι πανομοιότυπες.

5.5.1.5. Υλοποίηση λύσης συγχρονισμού σε Java

```
package java_sync;

import com.mongodb.MongoClient;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoCursor;
import com.mongodb.client.MongoDatabase;
import com.mongodb.client.model.Projections;

import java.io.IOException;
import java.net.InetAddress;
import java.util.List;
import java.util.logging.Level;

import org.bson.Document;
import org.elasticsearch.action.bulk.BulkRequestBuilder;
import org.elasticsearch.action.bulk.BulkResponse;
import org.elasticsearch.client.transport.TransportClient;
import org.elasticsearch.common.settings.Settings;
import org.elasticsearch.common.transport.InetSocketTransportAddress;
import org.elasticsearch.common.xcontent.XContentBuilder;

import static org.elasticsearch.common.xcontent.XContentFactory.*;

public class JavaSync {
    @SuppressWarnings("unchecked")
    public static void main(String[] args) throws IOException {

        long bulkBuilderLength = 0;
        Settings settings = Settings.settingsBuilder()
                .put("cluster.name", "elasticsearch").build();
        TransportClient client =
TransportClient.builder().settings(settings).build()
                .addTransportAddress(new
InetSocketTransportAddress(InetAddress.getByName("localhost"), 9300));
        BulkRequestBuilder bulkRequest = client.prepareBulk();
```

```
java.util.logging.Logger.getLogger("org.mongodb.driver").setLevel(Leve
l.SEVERE);
        MongoClient mongoClient = new MongoClient( "localhost" , 27017
);
        MongoDatabase mongodb =
mongoClient.getDatabase("elasticPlaces");
        MongoCollection<Document> collection =
mongodb.getCollection("places");
        MongoCursor<Document> cursor =
collection.find().projection(Projections.include("name", "_id",
"types", "formatted_address", "rating", "geometry")).iterator();
        while (cursor.hasNext()) {
            Document place = cursor.next();
            String id = place.get("_id").toString();
            String name = place.get("name").toString();
            String formatted_address =
place.get("formatted_address").toString();
            String rating = place.get("rating").toString();
            String lat = ((Document) ((Document)
place.get("geometry")).get("location")).get("lat").toString();
            String lon = ((Document) ((Document)
place.get("geometry")).get("location")).get("lng").toString();
            List<String> types = (List<String>) place.get("types");
            XContentBuilder builder = jsonBuilder().startObject()
                    .field("name", name)
                    .field("formatted_address", formatted_address)
                    .field("rating", rating)
                    .startObject("location")
                        .field("lat", lat)
                        .field("lon", lon)
                    .endObject()
                    .startArray("types");
                    for (String type : types) {
                        builder.value(type);
                    }
                    builder.endArray();
            builder.endObject();
            bulkRequest.add(client.prepareIndex("elasticplaces",
"places", id).setSource(builder));
            bulkBuilderLength++;

            //batch size 1000
            if(bulkBuilderLength % 1000== 0){
                  BulkResponse bulkRes =
bulkRequest.execute().actionGet();
                  if(bulkRes.hasFailures()){
                  }
                  bulkRequest = client.prepareBulk();
            }
        }
        if(bulkRequest.numberOfActions() > 0){
           System.out.println("##### " + bulkBuilderLength + " data
indexed.");
           BulkResponse bulkRes = bulkRequest.execute().actionGet();
           if(bulkRes.hasFailures()){
              System.out.println("##### Bulk Request failure with
error: " +   bulkRes.buildFailureMessage());
           }
           bulkRequest = client.prepareBulk();
        }
```

```
        client.close();
        mongoClient.close();
    }
}
```

5.5.1.6. Υλοποίηση generator δεδομένων

Επιπλέον υλοποιήσαμε σε Python ένα generator για να παράγουμε πολύ μεγάλο αριθμό δεδομένων, ώστε να μπορούμε να μετρήσουμε την απόδοση των λύσεων συγχρονισμού σε κλιμακούμενα μεγέθη. Ο κώδικας παρουσιάζεται παρακάτω:

gen.py

```
# coding: utf-8
from __future__ import division
import random, string, pymongo, sys

collection = pymongo.MongoClient()['elasticPlaces']['places']
size = int(sys.argv[1])
bulk_size = 1000

def main():
    arr = []
    cstart = collection.count()
    packet = 0
    for i in range(size):
        place = {
            "name" : "" ,
            "geometry" : {
            "location" : {
                "lat" : 0.0,
                "lng" : 0.0
            }
            },
            "url" : "",
            "formatted_address" : "",
            "types" : ["bar", "route"],
            "icon" : ""
        }
        place['name'] = randomword(50)
        place['geometry']['location']['lat'] =
random.randint(379500009, 379999999)/10000000
        place['geometry']['location']['lng'] =
random.randint(237200000, 239999999)/10000000
        place['url'] = randomword(100)
        place['formatted_address'] = randomword(50)
        place['icon'] = randomword(50)
```

```
        arr.append(place)
        packet += 1
        if (packet == bulk_size or i == (size - 1)):
            collection.insert_many(arr, False)
            packet = 0
            arr = []

    cstop = collection.count()
    print(cstop - cstart)

def randomword(length):
   return ''.join(random.choice(string.lowercase) for i in
range(length))

main()
```

5.5.2. Μετρήσεις

Για την σύγκριση των διαφόρων μεθόδων συγχρονισμού που χρησιμοποιήσαμε, πραγματοποιήσαμε μετρήσεις χρόνου ολοκλήρωσης για διαφορετικό πλήθος δεδομένων που συγχρονίζονται. Προκειμένου οι μετρήσεις να είναι αξιόπιστες, φροντίσαμε να διατηρούνται σταθεροί οι υπόλοιποι παράγοντες που θα μπορούσαν να επηρεάσουν την εκτέλεση των προγραμμάτων. Συγκεκριμένα, χρησιμοποιήσαμε εικονικές μηχανές μέσω του προγράμματος εικονικοποίησης (virtualization) VirtualBox της Oracle. Το host μηχάνημα έχει τα παρακάτω χαρακτηριστικά:

CPU	Intel Core i7-6700 3.4GHz
OS	Windows 10 Education
RAM	16 GB DDR4 2400 MHz
HDD	1TB 7200rpm
Motherboard	Gigabyte B150-HD3P-CF

Η κάθε μια από τις εικονικές μηχανές είχε στη διάθεση της ένα πυρήνα του επεξεργαστή, 6144 MB RAM, και 100 GB χώρο στον σκληρό δίσκο. Σε όλες τις εικονικές μηχανές έτρεχε λειτουργικό Ubuntu 14.04 LTS.

Παρακάτω παρουσιάζονται δέκα μετρήσεις (σε δευτερόλεπτα) για κάθε μέγεθος δεδομένων κάθε μεθόδου συγχρονισμού, οι μέσοι όροι αυτών και το διάγραμμα επίδοσης:

Αριθμός Δεδομένων: 1000				
Python	**Java**	**Mongo-Connector**	**Transporter**	**River**
0.700	2.347	2.110	0.830	0.088
0.611	1.902	1.480	0.441	0.059
0.648	1.774	0.586	0.453	0.077
1.239	1.665	0.874	0.445	0.081
0.617	1.469	0.567	0.326	0.058
0.588	1.282	0.831	0.323	0.069
0.609	1.300	0.666	0.330	0.067
0.588	1.340	0.490	0.440	0.077
0.590	1.289	0.482	0.459	0.070
0.586	1.267	0.536	0.343	0.075

Αριθμός Δεδομένων: 50000				
Python	**Java**	**Mongo-Connector**	**Transporter**	**River**
18.145	7.278	25.772	10.971	6.341
13.636	7.407	20.468	11.355	6.317
13.110	7.186	19.104	9.897	5.256
13.144	6.575	17.569	9.588	5.160
13.123	7.066	17.483	9.970	5.605
12.642	6.875	17.017	10.013	4.873
12.622	6.478	17.285	9.874	5.107
14.142	6.762	18.135	9.199	4.934
12.630	7.450	17.149	16.681	5.082
12.641	6.486	18.198	10.517	4.919

Αριθμός Δεδομένων: 100000				
Python	**Java**	**Mongo-Connector**	**Transporter**	**River**
25.676	10.488	36.568	20.495	10.582

26.159	10.961	36.269	21.161	11.021
26.168	10.999	36.404	20.612	11.166
26.664	10.545	38.030	21.434	11.303
25.644	10.763	37.961	21.032	11.344
25.121	11.021	40.322	21.633	12.067
25.172	11.313	46.068	20.853	10.934
25.645	12.340	45.564	21.202	12.059
26.140	10.528	44.772	21.206	11.550
26.135	10.769	44.273	20.929	11.426

Αριθμός Δεδομένων: 250000				
Python	**Java**	**Mongo-Connector**	**Transporter**	**River**
63.743	22.437	117.333	51.579	32.040
63.218	24.447	120.257	54.483	31.824
64.208	24.633	99.453	54.275	31.323
70.207	26.689	96.542	62.700	31.976
64.753	24.237	97.892	59.364	35.085
64.235	22.655	99.438	56.890	36.646
63.734	23.924	102.228	54.919	29.488
63.729	24.010	94.233	52.726	29.702
63.186	23.022	92.105	54.569	29.532
63.259	23.832	92.127	55.326	29.499

Αριθμός Δεδομένων: 500000				
Python	**Java**	**Mongo-Connector**	**Transporter**	**River**
139.460	47.416	195.157	104.770	64.763
131.367	46.752	193.042	126.102	66.788

130.848	54.426	195.717	119.156	65.376
130.859	47.727	202.299	117.127	66.389
130.822	58.101	197.314	112.608	67.252
140.370	51.413	193.536	118.888	64.881
130.855	53.435	185.902	114.492	71.220
128.881	49.996	194.954	115.059	72.263
133.944	49.491	192.431	119.845	67.133
136.402	49.592	192.629	118.643	68.610

Αριθμός Δεδομένων: 1000000				
Python	**Java**	**Mongo-Connector**	**Transporter**	**River**
273.598	101.822	386.895	225.367	145.255
278.212	104.368	406.181	230.631	139.916
282.726	102.660	399.263	272.619	147.113
268.615	101.019	411.690	208.525	143.521
276.113	104.008	437.049	217.645	138.465
273.232	102.683	424.772	222.001	141.447
243.587	102.173	412.301	227.848	145.010
271.633	104.860	414.844	220.496	133.601
276.175	102.933	403.190	230.269	135.211
276.201	109.717	393.813	208.914	143.036

Αριθμός Δεδομένων: 10000000				
Python	**Java**	**Mongo-Connector**	**Transporter**	**River**
3145.127	1497.950	4468.100	2665.783	1941.577
3141.372	1608.546	4818.488	2772.905	1977.050

3142.426	1520.199	4766.036	2846.668	2053.442
3158.277	1616.987	4643.622	2955.847	1789.995
3150.184	1532.846	4680.565	3024.851	1842.348
3142.456	1492.989	4751.338	2606.102	1809.083
3141.357	1515.486	4917.713	2803.483	1958.320
3142.668	1533.474	4644.831	2850.384	1940.788
3158.954	1583.846	4624.073	2700.862	2005.138
3150.278	1508.685	4644.153	2721.259	1998.359

Μέσος όρος μετρήσεων για κάθε μέγεθος

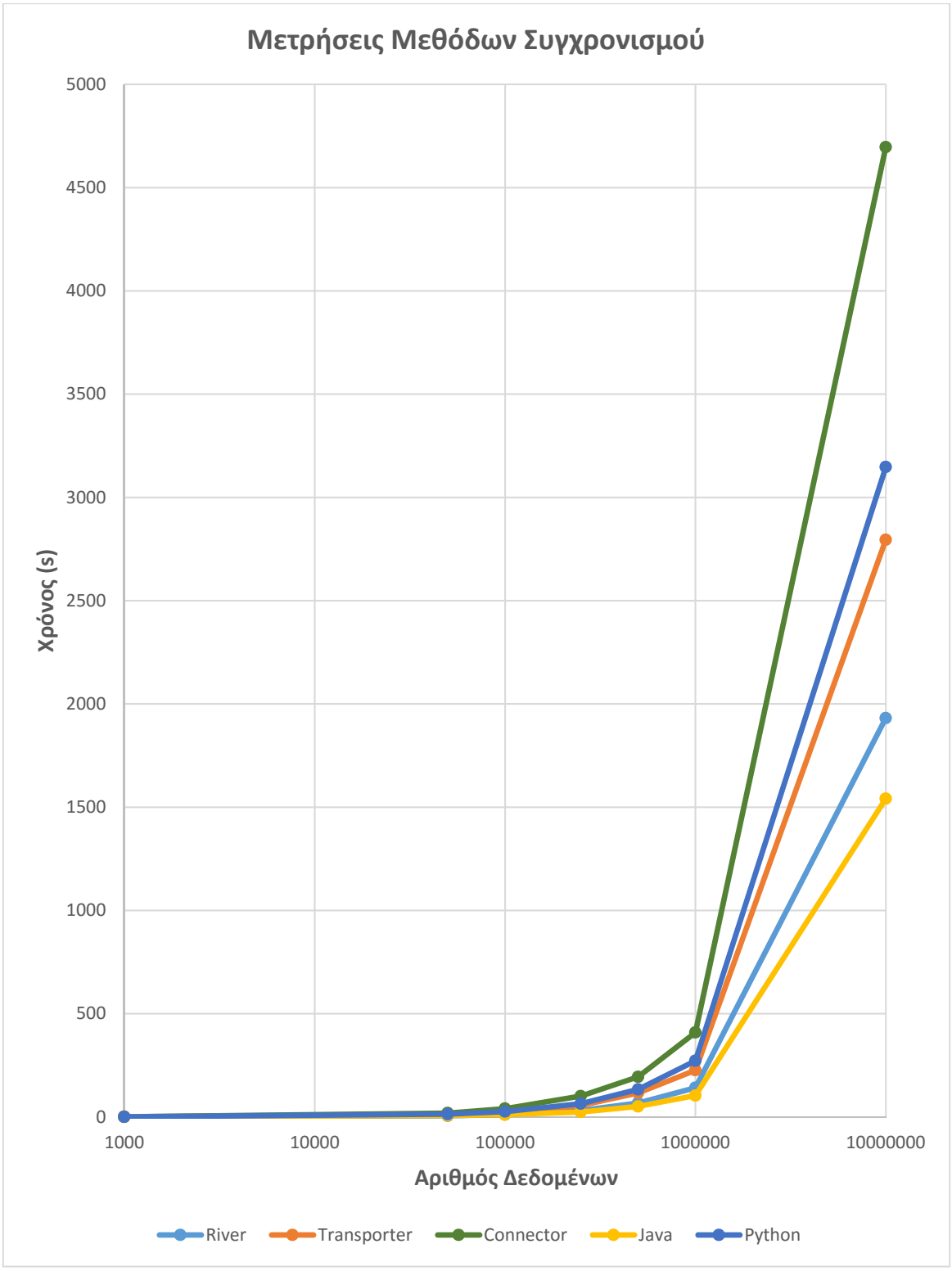
Μέγεθος	**Python**	**Java**	**Mongo-Connector**	**Transporter**	**River**
1000	0.678	1.564	0.862	0.439	0.072
50000	13.584	6.956	18.818	10.807	5.359
100000	25.852	10.973	40.623	21.056	11.345
250000	64.427	23.989	101.161	55.683	31.712
500000	133.381	50.835	194.298	116.669	67.468
1000000	272.009	103.624	409.000	226.432	141.257
10000000	3147.310	1541.101	4695.892	2794.814	1931.610

Παρατηρούμε λοιπόν πως η υλοποίηση μας σε Java είναι συνολικά η γρηγορότερη μέθοδος, με εξαίρεση το πολύ μικρό μέγεθος δεδομένων για πλήθος χιλίων εγγραφών. Η καθυστέρηση σε αυτό το σημείο πιθανότατα οφείλεται στο γεγονός ότι η Java χρειάζεται να κάνει κάποιες αρχικοποιήσεις κάθε φορά που τρέχει το πρόγραμμα αυτό, οπότε στην μέτρηση περιλαμβάνεται το χρονικό διάστημα αυτό, ενώ σε μεγαλύτερα πλήθη εγγραφών αυτό το σταθερό χρονικό διάστημα αρχικοποίησης γίνεται πλέον αμελητέο. Επιπλέον, τα τρία εργαλεία που χρησιμοποιήσαμε τρέχουν διαρκώς και συγχρονίζουν, σε αντίθεση με τις δύο δικές μας υλοποιήσεις σε Python και Java, τις οποίες εκκινούμε κάθε φορά που θέλουμε να συγχρονίσουμε τις δύο βάσεις. Επομένως, τα Mongo-Connector, Transporter και River πραγματοποιούν τις διάφορες αρχικοποιήσεις τους όταν εκκινούν να λειτουργούν και όχι κατά τη διάρκεια του συγχρονισμού. Η πολύ υψηλή επίδοση της υλοποίησης Java είναι απόλυτα αναμενόμενη, αφού τα προγράμματα γραμμένα σε αυτή τη γλώσσα «μιλάνε» απευθείας με την Elasticsearch και όχι μέσω HTTP, ενώ και όλα τα υπόλοιπα APIs για

Elasticsearch είναι υλοποιημένα με βάση το API για Java, οπότε προφανώς εισάγονται καθυστερήσεις λόγω των επιπλέον επιπέδων αφαίρεσης.

Πολύ γρήγορη επίσης παρουσιάζεται και η επίδοση του River. Αυτό είναι σχετικά αναμενόμενο, καθώς το River είναι πρακτικά μέρος της Elasticsearch, υλοποιείται με χρήση εξιδεικευμένου index και απλά δέχεται plugin για να καταλαβαίνει η Elasticsearch τα δεδομένα που εισάγονται από MongoDB (ή οποιαδήποτε άλλη εξωτερική πηγή). Παρότι τα Elasticsearch Rivers έχουν εξαιρετική επίδοση, είναι deprecated (καταργούνται) από την έκδοση της 2.0 και μετά, ακριβώς επειδή αποτελούν μέρος της Elasticsearch, καθώς αυτό το γεγονός έχει την τάση να επηρεάζει την σταθερότητα των clusters. Τα plugins που διατίθενται για τις διάφορες εξωτερικές πηγές δεδομένων δεν αναπτύσσονται επίσημα από την ομάδα ανάπτυξης της Elasticsearch, οπότε δεν εγγυάται η σταθερότητά τους.

Ακολουθεί το διάγραμμα επίδοσης των λύσεων συγχρονισμού:


Μετρήσεις Μεθόδων Συγχρονισμού
5000
4500
4000
3500
3000
2500
2000
1500
1000
500
0
Χρόνος (s)
1000
10000
100000
1000000
10000000
Αριθμός Δεδομένων
River
Transporter
Connector
Java
Python

6

Επίλογος

6.1. Συμπεράσματα

Ολοκληρώνοντας τη συγγραφή της διπλωματικής εργασίας, αλλά και κατά τη διάρκεια της υλοποίησης των εφαρμογών και συστημάτων, καταλήξαμε σε διάφορα χρήσιμα συμπεράσματα, μέσα από την ενασχόλησή μας με διάφορες τεχνολογίες και φιλοσοφίες σχεδίασης συστημάτων.

Αναπτύσσοντας μία μηχανή αναζήτησης, σημαντικό ρόλο κατέχει ο προσδιορισμός των αναγκών των χρηστών, όχι μόνο όπως τις φαντάζονται και τις σχεδιάζουν αρχικά οι προγραμματιστές, αλλά και η διαρκής αναγνώριση τους και προσαρμογή των δεδομένων και των κριτηρίων εμφάνισης των αποτελεσμάτων αντίστοιχα. Συγκεκριμένα στη δική μας υλοποίηση, σχεδιάσαμε με βάση την δική μας ανάγνωση αναγκών των χρηστών, αλλά κατά τη συντήρηση και αναβάθμιση (update) της εφαρμογής συνίσταται επανεξέταση και κατάλληλη προσαρμογή κριτηρίων.

Πειραματιστήκαμε με διαφορετικά συστήματα διαχείρισης βάσεων δεδομένων (DBMS) και καταλήξαμε στον συνδυασμό MongoDB σαν αποθήκη δεδομένων (data store) και Elasticsearch ως προσωρινή αποθήκη και κυρίως ως ευρετήριο για εξιδεικευμένες αναζητήσεις κειμένου. Ο συνδυασμός δύο συστημάτων με διαφορετικές δυνατότητες και πλεονεκτήματα παρέχει ευελιξία στην ανάπτυξη εφαρμογών και δίνει στον προγραμματιστή την ευχέρεια να ενσωματώσει συνδυασμούς χαρακτηριστικών παρέχοντας τελικά πιο ολοκληρωμένες υπηρεσίες στον χρήστη. Φυσικά, η πολυπλοκότητα του συστήματος αυξάνεται και απαιτείται προσεκτική σχεδίαση για την αποφυγή ασυνεπειών στα δεδομένα αλλά και πολλά ακόμα ζητήματα.

Σε συνέχεια του παραπάνω, ασχοληθήκαμε ιδιαίτερα με τον συγχρονισμό των δεδομένων και διαφορετικές μεθόδους και εργαλεία για την επίτευξή του. Συμπερασματικά, υπάρχει η δυνατότητα να αναπτύξει ο προγραμματιστής δική του λύση χρησιμοποιώντας ένα από τα παρεχόμενα API για κάποια γλώσσα και να την προσαρμόσει ακριβώς στις ανάγκες της εφαρμογής του, αλλά υφίστανται επίσης έτοιμα εργαλεία γενικής χρήσεως που λειτουργούν ικανοποιητικά και

προσφέρουν δυνατότητες παραμετροποίησης. Πάντως η καλύτερη επίδοση επιτυγχάνεται στην πρώτη περίπτωση, που όμως απαιτεί και περισσότερο χρόνο και δουλειά από τον προγραμματιστή.

Τέλος, θεωρούμε σημαντικό να αναφέρουμε πως σε μία μηχανή αναζήτησης, η ποσότητα και η ποιότητα των δεδομένων είναι εξίσου σημαντικά με τον μηχανισμό και τις τεχνικές ερωτημάτων που χρησιμοποιούνται. Στα πλαίσια της διπλωματικής εργασίας δεν είχαμε την δυνατότητα να συλλέξουμε αρκετά εκτενή δεδομένα από χρήστες ώστε να μπορέσουμε να εκμεταλλευτούμε τεχνικές αυτοματοποίησης αναζητήσεων με βάση τις αναζητήσεις άλλων χρηστών και άλλες παρόμοιες εξελιγμένες λειτουργίες που χρησιμοποιούνται από γνωστές μηχανές αναζήτησης όπως το Google, το Yahoo κλπ.

6.2. Μελλοντικές επεκτάσεις

Στο υπο-κεφάλαιο αυτό θα αναλύσουμε πιθανές μελλοντικές επεκτάσεις για το σύστημα που αναπτύξαμε. Ξεκινώντας, η διεπαφή του χρήστη με το σύστημα πραγματοποιείται αποκλειστικά στην πλατφόρμα Android και σε Web browser. Θα ήταν ιδιαίτερα βοηθητικό για τους χρήστες η διάθεση της εφαρμογής και σε άλλες δημοφιλείς πλατφόρμες όπως το iOS της Apple και το Windows Phone της Microsoft.

Το σύστημα ως έχει χρησιμοποιεί δεδομένα που εισάγει ο διαχειριστής του συστήματος, και αυτό το καθιστά λιγότερο ευέλικτο και διαδραστικό με τους χρήστες, καθώς και περιορίζει την κλιμάκωση των διαθέσιμων δεδομένων. Μια πολύ χρήσιμη λοιπόν μελλοντική επέκταση θα μπορούσε να είναι η ανάπτυξη δυνατότητας για τους χρήστες να εισάγουν νέα Places καθώς και να μπορούν να δημοσιεύουν κριτικές και βαθμολογίες για υπάρχοντα.

Καθώς το μέγεθος των συνόλων δεδομένων (data sets) που χρησιμοποιούμε στις μέρες μας αυξάνεται εκθετικά, παρουσιάζεται η ανάγκη για κατανεμημένα συστήματα και κλιμάκωση των λειτουργιών, δυνατότητα που παρέχεται από την Elasticsearch και την MongoDB. Συνεπώς, θα πρέπει στο μέλλον να τεθούν σε λειτουργία απομακρυσμένοι εξυπηρετητές, σε μορφή Cluster.

7

Βιβλιογραφία

[1] Java, http://www.oracle.com/technetwork/java/index.html

[2] Elasticsearch, https://www.elastic.co/products/elasticsearch

[3] MongoDB, https://www.mongodb.com/

[4] Python, https://www.python.org/

[5] Stackoverflow, http://stackoverflow.com/

[6] Abraham Silberschatz, Henry F. Korth, S. Sudarshan, Συστήματα Βάσεων Δεδομένων, 2011

[7] Alan Dix, Janet Finlay, Gregory D. Abowd, Beale Russell, Επικοινωνία ανθρώπου – υπολογιστή, 2007

[8] Bill Phillips, Chris Stewart, Android Programming: The Big Nerd Ranch Guide, 2015

[9] Neil Smyth, Android Studio Development Essentials, 2015

[10] Java, http://www.oracle.com/technetwork/java/index.html

[11] Kathy Sierra and Bert Bates, Head First Java, 2005

[12] Pressman R.S., Τεχνολογία Λογισμικού, 2012

[13] Google Maps, https://developers.google.com/maps/

[14] Django Project, https://www.djangoproject.com/