# ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

2020


**ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΛΟΠΟΝΝΗΣΟΥ**

## Ενσωματωμένη αναζήτηση σε NoSQL με χρήση Elasticsearch

**Φοιτητές**: Βασιλακόπουλος Ορέστης 1653,
Τσαντουλής Παναγιώτης 1829

**Επιβλέπων Καθηγητής:** Ταμπακάς Βασίλειος

# Ευχαριστίες

Αρχικά θέλουμε να ευχαριστήσουμε τον καθηγητή μας για την ευκαιρία που μας έδωσε να ασχοληθούμε με το συγκεκριμένο και τόσο ενδιαφέρον θέμα και για την καθοδήγηση του στην εξέλιξη της πτυχιακής μας εργασίας, τον κ. Βασίλειο Ταμπακά, καθώς επίσης και για τις συμβουλές του και την καθοδήγησή του καθ' όλη την διάρκεια των σπουδών μας.
Τέλος, θα θέλαμε να ευχαριστήσουμε τις οικογένειες μας για την στήριξή τους όλα αυτά τα χρόνια.

# Περιεχόμενα

1. Εισαγωγή 556667889999. Βιβλιογραφικά στοιχεία σχετικά με NoSQL 81010101011111212121313141414151515161616171717171818181818191919202021 2121212222 2022222222232323242424242526262728323232323234353537373838394142424345 46Ανάκτηση πολλών βιβλίων - Multi Get book documents 414647484950505152 475354555657575860 546062636569696970707172747575
75
75
75

75
75
75

75
75
75
75
75
75
75
75
75
75
75
75
75
75
75
75
75

# 1. Εισαγωγή

## 1.1. Στόχος

Στόχος της παρούσας διπλωματικής εργασίας είναι η διερεύνηση των δυνατοτήτων των NoSQL βάσεων δεδομένων και πιο συγκεκριμένα η διερεύνηση του εργαλείου Elasticsearch. Σκοπός μας είναι η εγκατάσταση του Elasticsearch σε υπολογιστή που φέρει το λειτουργικό σύστημα Ubuntu χρησιμοποιώντας Dockers. Με αυτόν τον τρόπο προσδίδουμε ευελιξία στην εγκατάσταση της συγκεκριμένης βάσης δεδομένων, καθώς μπορούμε να την αφαιρέσουμε και προσθέσουμε όποτε είναι επιθυμητό. Αφού γίνει η εγκατάσταση επόμενος στόχος μας είναι η συγγραφή προγράμματος στην γλώσσα προγραμματισμού Python 3 που να εισάγει δεδομένα σχετικά με βιβλία στο Elasticsearch, θα υπάρχει μια εγγραφή για κάθε βιβλίο που θα περιέχει τα εξής δεδομένα (ID, summary, title, publish_date, authors, num_reviews, publisher). Θα γίνει υλοποίηση queries σε Python χρησιμοποιώντας την βιβλιοθήκη Elasticsearch για αναζήτηση αυτών των δεδομένων βάση των παραπάνω παραμέτρων π.χ. αναζήτηση βάση συγγραφέων κλπ. Το πρόγραμμα θα δίνει την δυνατότητα στον χρήστη να εισάγει πληροφορία για τα βιβλία καθώς και να διαγράψει πληροφορία ή να ανανεώσει. Αξιοποίηση των ιδιοτήτων του Elasticsearch για υλοποίηση πιο σύνθετων queries τα οποία θα συνδυάζουν την αναζήτηση βάσει συγκεκριμένων όρων και να κάνει exclude κάποιους άλλους. Πχ την εύρεση ενός βιβλίου που να περιέχει την λέξη " Elasticsearch " η την λέξη "Υπολογιστής" και το όνομα του συγγραφέα να μην είναι "Βασιλόπουλος". Αξιοποίηση ιδιοτήτων του Elasticsearch για αναζήτηση όμοιων βιβλίων (βάσει του summary) με ένα βιβλίο που δίνεται σαν είσοδος ο τίτλος του ή το ID του. Στην συνέχεια του κεφαλαίου γίνεται αναφορά στις ορολογίες NoSQL, Elasticsearch και Docker.

## 1.2. NoSQL Βάσεις Δεδομένων

### 1.2.1. Τι είναι τα NoSQL συστήματα

Ο όρος NoSQL, μεταφράζεται ως «Not Only SQL» και πρωτοεμφανίστηκε στις αρχές του 2009 σε μια συνάντηση στο San Francisco, κατά την οποία παρουσιάστηκαν κάποιες «νέες» βάσεις δεδομένων. Σαφής ορισμός σχετικά με ένα σύστημα NoSQL δεν υπάρχει. Αναφέρεται σε μια ευρεία ομάδα συστημάτων διαχείρισης βάσεων δεδομένων (database management system) που το κύριο χαρακτηριστικό τους είναι το ότι δεν τηρούν το RDBMS μοντέλο (Relational Database Management System), το οποίο και χρησιμοποιείται στην συντριπτική πλειοψηφία των περιπτώσεων, καθώς επίσης χρησιμοποιούν διαφορετικό από τον κλασικό τρόπο (SQL) για την διαχείριση και επεξεργασία των δεδομένων μέσα στη βάση (data manipulation). Τα NoSQL συστήματα είναι κατανεμημένες, μη σχεσιακές (non-relational) βάσεις, σχεδιασμένες για αποθήκευση δεδομένων μεγάλης κλίμακας και παράλληλη επεξεργασία δεδομένων μοιρασμένα σε έναν μεγάλο αριθμό από servers. Οι NoSQL βάσεις δεδομένων γενικώς δεν χρησιμοποιούν κάποιο δομημένο σύστημα για τα στοιχεία που περιλαμβάνουν, όπως για παράδειγμα πίνακες, οι οποίοι είναι οι δομικές μονάδες για ένα παραδοσιακό σχεσιακό σύστημα, ούτε χρησιμοποιούν κάποια Structured Query Language (SQL) [1] για την διαχείριση των δεδομένων. Χρησιμοποιούν αποκλειστικά non - relational τρόπους οργάνωσης και ανάλυσης των δεδομένων και είναι κατά κύριο λόγο βελτιστοποιημένες, ώστε να επισυνάπτουν και να ανακτούν τα δεδομένα αυτά. Τα κύρια χαρακτηριστικά τους συνοψίζονται στα εξής : 1) Η μη χρήση της SQL ως query language, 2) δεν μπορούν να εγγυηθούν ότι οι διεργασίες στην βάση δεδομένων θα γίνονται αξιόπιστα 3) έχουν μια ιδιαίτερη αρχιτεκτονική και φιλοσοφία λειτουργίας. Το NoSQL κίνημα υιοθετήθηκε από τις μεγαλύτερες εταιρείες στον χώρο του Internet, όπως η Google, η Amazon και το Facebook. Και στις 3 αυτές περιπτώσεις προέκυψαν προκλήσεις στον χειρισμό τεράστιου όγκου δεδομένων, όπου οι συμβατικές RDBMS λύσεις [2], δεν μπορούσαν να ανταπεξέλθουν. Και αυτό είναι το βασικό ατού των NoSQL συστημάτων. Έχουν την ικανότητα να αποθηκεύουν και να ανακτούν μεγάλες ποσότητες δεδομένων, «αδιαφορώντας» για τις σχέσεις μεταξύ των στοιχείων αυτών. Επίσης, οι μέθοδοι υλοποίησης και εφαρμογής τους αξιοποιούν μια αρχιτεκτονική που επιτρέπει (και ίσως διευκολύνει) την κατανεμημένη λειτουργία του συστήματος. Τα χαρακτηριστικά αυτά οδηγούν το σύστημα σε αυξημένες επιδόσεις, αφού παρέχεται η δυνατότητα

κλιμάκωσης (θεωρητικά άπειροι servers όπου κατανεμημένα θα επεξεργάζονται τα δεδομένα του συστήματος).Η σημαντική αυτή αύξηση στην απόδοση και την επεκτασιμότητα για ορισμένα μοντέλα δεδομένων, αντισταθμίζει την μειωμένη ευελιξία του χρόνου εκτέλεσης σε σύγκριση με συστήματα SQL (RDBMS). Μπορούν να υποστηρίξουν πολλαπλές δραστηριότητες, συμπεριλαμβανομένων διαφόρων αναγνωριστικών και προγνωστικών analytics, την μετατροπή δεδομένων στυλ ETL, και μη κρίσιμες OLTP (για παράδειγμα, την διαχείριση μακράς διάρκειας ή ενδοεπιχειρηματικών συναλλαγών). Αρχικά τα συστήματα αυτά υποκινήθηκαν από Web 2.0 εφαρμογές και είναι σχεδιασμένα για αυξανόμενη κλιμάκωση (scaling) σε χιλιάδες ή εκατομμύρια χρήστες που κάνουν updates καθώς και reads, σε αντίθεση με τα παραδοσιακά DBMS συστήματα και τα data warehouses. Η ιδέα είναι ότι και οι 2 τεχνολογίες μπορούν να συνυπάρξουν και η κάθε μία έχει την δική της θέση. Το κίνημα των NoSQL έχει αναδειχθεί τα τελευταία χρόνια καθώς πολύ πρωτοπόροι του Web 2.0, έχουν υιοθετήσει τα συστήματα αυτά. Εταιρείες όπως το Facebook, Twitter, Digg, Amazon, LinkedIn και η Google, όλες χρησιμοποιούν NoSQL με τον ένα ή με τον άλλο τρόπο.

1.2.2. Γιατί, που και πότε είναι χρήσιμες

Η συχνή πλέον χρήση μη-σχεσιακών βάσεων δεδομένων στον τομέα της τεχνολογίας, θα έκανε κάποιον να σκεφτεί ότι τα NoSQL συστήματα έχουν αρχίσει να εκτοπίζουν τα παραδοσιακά σχεσιακά συστήματα. Αυτό στην πραγματικότητα δεν ισχύει. Κάθε ένα από αυτά τα δύο μοντέλα έχει διακριτό ρόλο και είναι κατάλληλο για διαφορετικές εφαρμογές και διαφορετικό φόρτο εργασίας. Στο κεφάλαιο αυτό θα επικεντρωθούμε στο γιατί να επιλέξει κάποιος να πάει σε ένα NoSQL σύστημα, πάντα σε σύγκριση με ένα SQL. Τα NoSQL συστήματα δεν εμφανίστηκαν για να αντικαταστήσουν τα σχεσιακά, αλλά για να τα συμπληρώσουν. Τα NoSQL συστήματα διαχείρισης βάσεων δεδομένων είναι εξαιρετικά χρήσιμα όταν κάποιος δουλεύει με τεράστιο όγκο δεδομένων, η φύση των οποίων δεν απαιτεί κάποιο σχεσιακό μοντέλο. Για παράδειγμα εταιρείες που συλλέγουν μεγάλες ποσότητες «αδόμητων» (unstructured) δεδομένων, στρέφονται όλο και περισσότερο σε μη-σχεσιακές βάσεις [3]. Η κατανεμημένη τους φύση τα καθιστά ιδανικά για μαζική επεξεργασία δεδομένων (π χ ενοποίηση, φιλτράρισμα, διαλογή, στατιστικές ενέργειες κλπ.). Είναι επίσης πολύ καλά για ανάκτηση και ανταλλαγή δεδομένων μεταξύ μηχανημάτων (machine – to - machine), καθώς και για την επεξεργασία συναλλαγών μεγάλου όγκου, με την προϋπόθεση βέβαια χαμηλών απαιτήσεων για ACID ιδιότητες. Ως εκ τούτου, παρέχουν σχετικά φθηνή, υψηλής επεκτασιμότητας αποθήκευση μεγάλου όγκου, όπως για παράδειγμα ιστορικά δεδομένα, αρχεία καταγραφής, αρχεία τηλεφωνικών δεδομένων, ενδείξεις μετρητών και αισθητήρων, καθώς και αποθήκευση ημιδομημένων (semi - structured) ή αδόμητων δεδομένων (unstructured), όπως αρχεία ηλεκτρονικού ταχυδρομείου, αρχεία XML, έγγραφα, κλπ. Έχουν την ικανότητα της κλιμάκωσης, κάτι στο οποίο υστερούν τα παραδοσιακά συστήματα. Πέρα από τα πλεονεκτήματα κλίμακας, η ίδια η αρχιτεκτονική τους βοηθά στην βελτίωση της απόδοσής τους. Αν μια σχεσιακή βάση δεδομένων έχει εκατοντάδες χιλιάδες πίνακες, η επεξεργασία των δεδομένων που βρίσκονται στους πίνακες αυτούς θα δημιουργήσει πολλά locks στα δεδομένα, γεγονός το οποίο θα έχει σαν συνέπεια την υποβάθμιση της απόδοσης του συστήματος. Εν αντίθεση, τα NoSQL συστήματα έχουν πιο αδύνατα μοντέλα συνέπειας των δεδομένων, μπορούν να «θυσιάσουν» την συνοχή για την αποδοτικότητα. Οι NoSQL λύσεις είναι ελκυστικές επειδή μπορούν να διαχειριστούν τεράστιες ποσότητες δεδομένων, σχετικά γρήγορα, σε ένα cluster από servers, οι οποίοι μοιράζονται πόρους μεταξύ τους. Επιπλέον, οι περισσότερες NoSQL λύσεις είναι ανοιχτού κώδικα (open source), κάτι το οποίο τους δίνει πλεονέκτημα τιμής έναντι των συμβατικών εμπορικών βάσεων δεδομένων.

1.2.3. Τι προσφέρουν και τι όχι

Αφού είδαμε πιο γενικά κάποια στοιχεία σχετικά με την λειτουργία των NoSQL συστημάτων και ποιες οι βασικές διαφορές τους από τα κλασικά συστήματα, τώρα θα αναφερθούμε πιο ειδικά στα πλεονεκτήματα τους, σε πιο πρακτικό επίπεδο. Οι NoSQL βάσεις δεδομένων επικεντρώνονται στην αναλυτική επεξεργασία μεγάλης κλίμακας συνόλου δεδομένων (big data), προσφέροντας αυξημένη επεκτασιμότητα και υψηλές επιδόσεις. Παρουσιάζουν μεγάλη «ελαστικότητα» στην βελτίωση της επίδοσης τους. Σε αυτό συμβάλλει η υλοποίησή τους, που τις περισσότερες φορές γίνεται σε υποδομές cloud ή σε virtualized environments, σε αντίθεση με τα κλασικά RDBMS συστήματα. Ειδικότερα, ενώ στα RDBMS συστήματα, όταν θέλουμε να τα βελτιώσουμε, προσθέτουμε περισσότερη μνήμη RAM ή καλύτερους επεξεργαστές στις υποδομές, αντίθετα στα NoSQL συστήματα απλά προσθέτουμε κόμβους ώστε να επεξεργάζονται ακόμη περισσότερα δεδομένα

ταυτόχρονα. Ένα από τα μεγαλύτερα συν που έχουν είναι ότι υποστηρίζουν την αποθήκευση και επεξεργασία μεγάλων όγκων δεδομένων (αδόμητα, ημιδομημένα κλπ.), κάτι στο οποίο υστερούν τα παραδοσιακά συστήματα, και κάτι το οποίο έχει γίνει αναπόφευκτο κομμάτι της σύγχρονης εποχής. Βασικό πλεονέκτημα των μη-σχεσιακών συστημάτων είναι και ο οικονομικός παράγοντας. Για παράδειγμα, η συντήρηση ενός high end συστήματος RDBMS που διαχειρίζεται τεράστιο όγκο δεδομένων απαιτεί εξίσου μεγάλο κόστος (κυρίως Database Administrators). Αντίθετα, τα NoSQL συστήματα εξορισμού είναι σχεδιασμένα ώστε να απαιτούν τη λιγότερη δυνατή διαχείριση από τον ανθρώπινο παράγοντα. Αυτό επιτυγχάνεται χρησιμοποιώντας μεθόδους όπως το automatic repair, data distribution και πιο απλά μοντέλα δεδομένων. Όλα αυτά οδηγούν σε σαφώς μικρότερες ανάγκες για διαχείριση ή βελτιστοποίηση του συστήματος. Επιπλέον, οποιαδήποτε αλλαγή στο μοντέλο δεδομένων (schema) ενός συστήματος RDBMS είναι μία διαδικασία που απαιτεί ένα σεβαστό χρονικό διάστημα που η εφαρμογή που στηρίζεται στην συγκεκριμένη βάση δεδομένων δεν θα είναι προσβάσιμη (downtime) ή θα μειωθούν σε μεγάλο βαθμό τα επίπεδα λειτουργίας της υπηρεσίας, ενώ ένα NoSQL σύστημα έχει πολύ λιγότερους (έως μηδαμινούς) περιορισμούς σε αυτό το ζήτημα. Από την άλλη μεριά βέβαια, τα NoSQL συστήματα δεν μπορούν να εγγυηθούν ιδιότητες ACID, σε σχέση με τις SQL βάσεις. Στην πλειοψηφία τους "διακινδυνεύουν" την συνοχή- παρουσιάζουν κάτι που γενικά αναφέρεται ως eventual consistency. Για να πετύχουν υψηλή διαθεσιμότητα, θυσιάζουν την συνέπεια. Με ποιο απλά λόγια γενικά θεωρείται πως η ισχυρή συνέπεια είναι σύγχρονη. Μια δοσοληψία δεν θεωρείται ολοκληρωμένη παρά μόνο αν ενημερωθούν όλα τα αντίγραφα της βάσης δεδομένων. Η Eventual consistency γενικά θεωρείται ασύγχρονη. Τα αντίγραφα ενημερώνονται στο παρασκήνιο, χωρίς να μπλοκάρουν την ολοκλήρωση της δοσοληψίας. Μετά από κάποια αστοχία, μπορεί να υπάρχουν στο σύστημα πολλαπλές εκδόσεις ενός αντικειμένου, ασυνεπείς μεταξύ τους. Ένα δεύτερο μειονέκτημα είναι ότι λόγω του «νεαρού της ηλικίας» τους και του ανοιχτού τους κώδικα, δεν προσφέρουν την ασφάλεια υποστήριξης και την αξιοπιστία που παρέχουν τα RDBMS συστήματα. Όπως έχει αναφερθεί νωρίτερα οι NoSQL βάσεις δεν χρησιμοποιούν σαν γλώσσα επερωτήσεων την SQL , αλλά πιο χαμηλές σε επίπεδο γλώσσες. Αυτό έχει σαν αποτέλεσμα να δημιουργούνται περιορισμοί στην ευκολία ανάλυσης των δεδομένων (δεν υπάρχουν joins και δεν υποστηρίζονται πάντα τα ad-hoc queries).

1.2.4. Κατηγορίες NoSQL συστημάτων

Τα NoSQL συστήματα μπορούν να κατηγοριοποιηθούν ως εξής με βάση την αρχιτεκτονική τους και στο μοντέλο δεδομένων που ακολουθούν:

1. **Key-values Stores**: DynamoDB, FoundationDB, MemcacheDB, Redis, Riak, c-treeACE, Aerospike, Azure Table Storage, LevelDB, Berkeley DB, Oracle NOSQL Database, GenieDB, BangDB, Chordless, Scalaris, Tokyo Cabinet / Tyrant, Voldemort, MemcacheDB [4]
2. **Column Family Stores**: Accumulo, Cassandra, Druid, HBase, Hypertable, Amazon SimpleDB, Cloudata, MonetDB [5]
3. **Document Databases**: Clusterpoint, Apache CouchDB, Couchbase, MarkLogic, MongoDB, Elasticsearch, RethinkDB, NeDB, Terrastore,XML-dbs (BaseX, eXist, Sedna, Qizx) [6]
4. **Graph Databases**: Allegro, Neo4J, InfiniteGraph, OrientDB, Virtuoso, Stardog, Sparksee, TITAN, InfoGrid, HyperGraphDB, GraphBase.

Η πρώτη κατηγορία, οι key-value stores, βασίζονται στην ύπαρξη ενός hash table όπου υπάρχει ένα μοναδικό κλειδί (key) και ένας δείκτης (pointer) που «δείχνει» σε ένα συγκεκριμένο στοιχείο. Είναι το πιο σχετικά απλό και εύκολο στην υλοποίηση μοντέλο από τα NoSQL. Το μοντέλο αυτό συνοδεύεται από μηχανισμούς content caching, για την καλύτερη απόδοση του συστήματος σε μεγάλους όγκους δεδομένων. Στις Column family stores τα δεδομένα αποθηκεύονται σε κελιά (cells), τα οποία ομαδοποιούνται σε στήλες (columns) και όχι σε σειρές (rows) όπως στις σχεσιακές βάσεις δεδομένων. Οι στήλες αυτές με την σειρά τους ομαδοποιούνται σε οικογένειες στηλών (column families), οι οποίες μπορούν να περιλαμβάνουν θεωρητικά άπειρο αριθμό από στήλες. Με τον τρόπο αυτό επιτυγχάνονται μεγάλες ταχύτητες σε λειτουργίες search / access, λόγω του ότι όλα τα κελιά που αναφέρονται σε μια στήλη είναι μια συνεχόμενη εγγραφή στο δίσκο. Η επόμενη κατηγορία, των Document databases, ακολουθούν την λογική των key - value stores. Τα δεδομένα σε αυτή την περίπτωση (συλλογές από ζευγάρια key - value) είναι οργανωμένα, πιο φυσικά και λογικά, χωρίς να

υπάρχουν περιορισμοί από κάποιο σχήμα. Κάθε εγγραφή (record) και τα συ σχετιζόμενα μαζί της δεδομένα, θεωρούνται ως ένα document. Η τελευταία κατηγορία, αυτή των Graph databases, χρησιμοποιεί δομές γράφων και είναι βασισμένη σε κόμβους (nodes), τις σχέσεις μεταξύ αυτών των κόμβων και τις ιδιότητές τους. Αντί για πίνακες με στήλες και σειρές, εδώ υπάρχει ένα ευέλικτο γραφικό μοντέλο (graph model) που μπορεί να χρησιμοποιηθεί και να αναπτυχθεί παράλληλα σε πολλά μηχανήματα (servers – κόμβους) [7].

### 1.3. Διάρθρωση διπλωματικής

Η παρούσα διπλωματική εργασία διαρθρώνεται ως εξής:

Στο πρώτο (1ο) κεφάλαιο παρουσιάζεται μια γενική εισαγωγή των σημερινών ζητημάτων, ποιος είναι ο στόχος της διπλωματικής εργασίας, κάποια στοιχεία και πληροφορίες σχετικές με τις NoSQL databases καθώς και οι κατηγορίες των NoSQL databases που υπάρχουν αυτήν την στιγμή. Στο δεύτερο (2ο) κεφάλαιο παρουσιάζονται κάποιες παρόμοιες εφαρμογές, δηλαδή εφαρμογές οι οποίες χρησιμοποιούν το Elasticsearch καθώς και άλλα συναφή εργαλεία. Επίσης παρουσιάζονται και άλλα συναφή στοιχεία. Στο τρίτο (3ο) κεφάλαιο γίνεται εκτενής παρουσίαση των τεχνολογιών που θα χρησιμοποιηθούν στην παρούσα διπλωματική. Το τέταρτο (4ο) κεφάλαιο αποτελείται από έναν αναλυτικό οδηγό εγκατάστασης των κυριότερων εργαλείων της παρούσας διπλωματικής εργασίας. Το πέμπτο (5ο) κεφάλαιο αναλύει την αρχιτεκτονική του συστήματος της εφαρμογής. Το έκτο (6ο) κεφάλαιο αναλύει την προγραμματιστική υλοποίηση της εφαρμογής. Το έβδομο (7ο) κεφάλαιο περιλαμβάνει τον επίλογο της διπλωματικής.

## 2. Βιβλιογραφία και παρουσίαση παρόμοιων εφαρμογών

### 2.1. Εισαγωγή

Το κεφάλαιο που ακολουθεί ασχολείται με 2 θέματα. Το πρώτο είναι η παράθεση της βιβλιογραφίας που χρησιμοποιήθηκε για την κατασκευή της εφαρμογής και την τεκμηρίωση του θεωρητικού υποβάθρου της και το δεύτερο είναι η σύγκριση της λειτουργικότητας και της ιδέας της εφαρμογής με άλλες παρόμοιες. Κυρίως πρόκειται για δεδομένα σχετικά με βιβλία, περιγραφές βιβλίων, ονόματα συγγραφέων από διαφορετικές πηγές όπου κύριο χαρακτηριστικό τους είναι ο μεγάλος τους όγκος και αποθηκεύονται σε NoSQL βάσεις δεδομένων και ευρετηριάζονται με την βοήθεια του Elasticsearch ώστε να διευκολύνεται η αναζήτηση μέσα από αυτά.

### 2.2. Βιβλιογραφία

Για τις ανάγκες της ανάπτυξης της εφαρμογής και της συγγραφής της παρούσας διπλωματικής εργασίας χρησιμοποιήθηκαν αρκετά είδη πηγών. Αρχικά χρησιμοποιήθηκε συγκεκριμένη βιβλιογραφία, ιδιαίτερα για τα περισσότερο θεωρητικά ζητήματα, όσο και ιστοσελίδες που περιλαμβάνουν εγχειρίδια για τη χρήση συγκεκριμένων συστατικών της εφαρμογής, ιστολογία που περιέχουν άρθρα για μια σειρά θεμάτων που αφορούσαν την υλοποίηση συγκεκριμένων τμημάτων του κώδικα και λύσεις σε δυσκολίες τόσο σε υλοποίηση κώδικα όσο και εγκατάσταση τεχνολογιών

που έπρεπε να δουλεύουν σε απόλυτη αντιστοιχία μμεταξύ τους και ήταν αναγκαίο να ακολουθηθούν συγκεκριμένα βήματα κατά τη διαδικασία εγκατάστασής τους. Ιδιαίτερα σημαντική βοήθεια ήταν επίσης ιστοσελίδες, όπου αποτελούν κοινωνικά δίκτυα μμεταξύ προγραμματιστών, όπως το GitHub, από το οποίο αντλήθηκαν βιβλιοθήκες που αποτέλεσαν στήριγμά για την ανάπτυξη ορισμένων λειτουργιών της εφαρμογής. Αλλά και ιστοσελίδες διαδικτυακών συζητήσεων (forums), όπως το stackoverflow.com, όπου και βρέθηκαν λύσεις σε δυσκολίες κατά την ανάπτυξη του κώδικα. Ειδικότερα, η παράθεση της βιβλιογραφίας και όλων των συναφών πηγών που προαναφέρθηκαν θα χωριστεί ανάλογα με τα διάφορα τμήματα της εφαρμογής, όπως θα αναλυθούν και παρακάτω.

2.2.1. Βιβλιογραφικά στοιχεία σχετικά με NoSQL

Συνεχίζοντας με την υποδομή του πίσω μέρους (backend) της εφαρμογής, για την κατανόηση του τρόπου λειτουργίας του Elasticsearch σημαντική ήταν η εξήγηση για τη γενικότερη φιλοσοφία, δομή και λειτουργία των NoSQL βάσεων δεδομένων και τις διαφορές που έχουν αυτές οι βάσεις με τις παραδοσιακές σχεσιακές βάσεις (SQL), όπως αυτή αναπτύσσεται από τον Vaish [8]. Ιδιαίτερα το γεγονός ότι οι NoSQL βάσεις δεδομένων είναι ιδανικές για παρουσίαση δεδομένων χωρίς την προϋπόθεση συγκεκριμένου σχήματος εξαιτίας της αποθήκευσης των δεδομένων σε μμορφή JSON οδήγησε στην επιλογή ενός τέτοιου είδους βάσης ως βασικής επιλογής για τις ανάγκες της εφαρμογής.

2.2.2. Βιβλιογραφικά στοιχεία για το Elasticsearch

Όσον αφορά το Elasticsearch εξαιτίας του γεγονότος ότι πρόκειται για πολύ καινούργια τεχνολογία η βιβλιογραφία είναι περιορισμένη και οι υποστηρικτικές πηγές που υπάρχουν είναι το εγχειρίδιο της επίσημής ιστοσελίδας (Elasticsearch.org), που χρησιμοποιήθηκε κατά κόρον, διάφορα άρθρα από ιστολογία και ιστοσελίδες γραμμένα από χρήστες και το GitHub, όπως και άλλες ιστοσελίδες διαδικτυακών συζητήσεων με προεξάρχουσα το stackoverflow.com. Από τις λίγες πηγές που προχωράνε ένα βήμα πιο πέρα είναι το εγχειρίδιο των Rafał Kuć, Marek Rogoziński [9] και που βοηθάει ιδιαίτερα στην κατανόηση τόσο της λειτουργίας όσο και της ίδιας της δομής του Elasticsearch και σε θεωρητικό και σε πρακτικό επίπεδο. Βασικό επίσης για την κατανόηση του θεωρητικού υπόβαθρου του Elasticsearch στάθηκε και η μελέτη του Lucene και των υποδομών που κληροδότησε για την ανάπτυξη του Elasticsearch.

2.3. Σύγκριση με παρόμοιες εφαρμογές

Η εμφάνιση τόσο των NoSQL βάσεων δεδομένων όσο και του Elasticsearch έγινε την τελευταία δεκαετία στον χώρο, γι' αυτό έχουν αφήσει το στίγμα τους στο χώρο της πληροφορικής. Η παρούσα εφαρμογή εξάλλου δεν φιλοδοξεί να σταθεί ως μια αυτόνομή εφαρμογή, αλλά κυρίως θα είναι ιδιαίτερα χρήσιμή, αν αποτελεί ένα συγκεκριμένο τμήμα κάτι μμεγαλύτερου. Αυτές οι σκέψεις θα παρουσιαστούν εκτενέστερα στο τελευταίο κεφάλαιο της παρούσας εργασίας. Για τους παραπάνω λόγους βρέθηκαν κάποιες εργασίες, που βέβαια δεν μμπορούν να συγκριθούν απόλυτα με την παρούσα εφαρμογή, γιατί το πεδίο τους είναι αρκετά μμεγαλύτερο, αλλά ένα κομμάτι τους, κυρίως αυτό που σχετίζεται με τη συγκέντρωση και αποθήκευση κειμενικών δεδομένων και η ευρετηρίαση τους, ώστε σε επόμενο στάδιο να είναι περισσότερο εύκολη η ανάκτηση τους, βρίσκεται σε πλήρη αντιστοίχιση. Ο κοινός παρονομαστής όλων των εφαρμογών είναι αυτό που περιεγράφηκε παραπάνω, δηλαδή από κάπου εισρέουν δεδομένα που πρέπει να αποθηκευτούν, με τέτοιο τρόπο, ώστε μμετέπειτα τα δεδομένα αυτά να μμπορούν να ανακτηθούν μέσω κάποιας υπηρεσίας αναζήτησης. Τα αποτελέσματα αυτά των αναζητήσεων θα μμπορούν να αποτελούν πεδίο για περαιτέρω έρευνες που ξεπερνούν το παρόν αντικείμενο, όπως είναι η εξόρυξη πληροφορίας και η μμηχανική μάθηση.

2.3.1. ThemeStreams

Η εφαρμογή ThemeStreams [10]. Αυτή η εφαρμογή αντλεί δεδομένα από το Twitter API και αποθηκεύει κειμενικά δεδομένα που δημοσιεύονται από συγκεκριμένες κατηγορίες χρηστών που όλοι σχετίζονται με την πολιτική σκηνή της Ολλανδίας. Αυτοί οι χρήστες είναι χωρισμένοι σε 4 κατηγορίες, πολιτικοί, δημοσιογράφοι, στελέχη πολιτικών κομμάτων ή άλλων οργανισμών που σχετίζονται ευθέως με κάποιο κόμμα ή πολιτικό και τέλος ηθοποιοί και διασημότητες που εκφέρουν στα tweets τους έντονο πολιτικό λόγο. Μέχρι τη στιγμή που γραφόταν η εν λόγω αναφορά τους είχαν καταφέρει να συλλέξουν περίπου 3,5 εκατομμύρια tweets από όλα τα είδη των παραπάνω χρηστών που προσεγγιστικά φτάνουν τον αριθμό των 245 χιλιάδων. Στη συνέχεια από ένα γραφικό περιβάλλον μίας ιστοσελίδας οι χρήστες μπορούν να ανακτήσουν τα tweets, έχοντας δύο κύριες επιλογές είτε να διαλέξουν ένα από τα προκαθορισμένα θέματα και να δουν πως τα tweets σχετικά με αυτά αυξομειώνονται σε όγκο στη μμονάδα του χρόνου, παράλληλα και με τις πιο συχνές λέξεις που

διαθέτουν είτε να διαλέξουν κάποια λέξη-κλειδί και επίσης να δουν γραφικά τη συχνότητά της μέσα στο χρόνο μμαζί με τις πιο συχνές λέξεις γύρω από αυτήν.
Αυτό όμως που είναι ιδιαίτερα ενδιαφέρον είναι ότι η συγκέντρωση όλων των δεδομένων γίνεται απευθείας στο Elasticsearch που στην ουσία το χρησιμοποιούν ως την κύρια βάση δεδομένων τους. Αυτή η επιλογή, όπως θα αναλυθεί και στα συμπεράσματα στο πέμπτο κεφάλαιο, έχει μια σειρά από προβλήματα. Μερικά από αυτά είναι η έλλειψη ασφάλειας, διάρκειας (durability) και διαθεσιμότητας (availability) των δεδομένων και η μη ύπαρξη αρκετά εξελιγμένων βιβλιοθηκών και εργαλείων για περαιτέρω διευκόλυνση της διαχείρισης των δεδομένων. Όλα αυτά συντείνουν πως η ύπαρξη ενός ενδιάμεσου επιπέδου αποθήκευσης των δεδομένων σε μια NoSQL βάση δεδομένων λύνει τα παραπάνω προβλήματα και δεν αποτελεί και ιδιαίτερα δύσκολο πρόβλημα, καθώς υπάρχει η κατάλληλη υποδομή, ώστε τα δεδομένα να οδηγούνται από τη NoSQL βάση προς το Elasticsearch.

### 2.3.2. AVResearcher

Η εφαρμογή AVResearcher [11] και σκοπό έχει την αναζήτηση μεταδεδομένων συνδεδεμένων με έναν μεγάλο αριθμό από οπτικοακουστικές εκπομπές. Οι εκπομπές αυτές μμπορούν να αναζητηθούν από τους χρήστες όχι μόνο βάσει των παραδοσιακών καταλόγων που περιγράφουν το εκάστοτε πρόγραμμα, αλλά μέσω των υποτίτλων τους και των αναφορών τους στο Twitter. Τα αποτελέσματα επιστρέφονται ανάλογα με το θέμα και το χρόνο δημοσίευσής τους.
Η αρχιτεκτονική του εγχειρήματος είναι αρκετά απλή. Τα δεδομένα που είναι προς αποθήκευση αντλούνται από ξεχωριστές πηγές, δηλαδή οι κατάλογοι περιγραφής των εκπομπών από το ολλανδικό Ινστιτούτο Ήχου και Εικόνας, οι υπότιτλοι από τους υπεύθυνους των ίδιων των ολλανδικών εκπομπών και τα tweets από το Twitter API. Συγκεκριμένα για το Twitter, αποθηκεύονται 25 επίσημές θεματικές ενότητες (hashtags) που αφορούν ισάριθμές τηλεοπτικές εκπομπές. Το σημαντικό είναι ότι όλα αυτά τα δεδομένα, παρόλη την ανομοιογένεια που ενδεχομένως έχουν, ευρετηριάζονται και αποθηκεύονται με τη βοήθεια του Elasticsearch και χωρίς τη μεσολάβηση κάποιου ενδιάμεσου επιπέδου. Οι συντάκτες της εργασίας δεν αναφέρουν λεπτομέρειες για τις διευκολύνσεις που τους παρέχει αυτή τους η επιλογή, αλλά σίγουρα θα αντιμετωπίζουν τις δυσκολίες της χρήσης μόνο ενός επιπέδου αποθήκευσης, όπως αυτές εξηγήθηκαν και στην παραπάνω εφαρμογή. Παράλληλα, οι χρήστες μπορούν να έχουν πρόσβαση στα δεδομένα μέσω ενός γραφικού περιβάλλοντος που προσφέρει μια ιστοσελίδα και για κάθε αναζήτησή τους έχουν τη δυνατότητα να βλέπουν τον αριθμό των εκπομπών που περιέχουν στην περιγραφή τους τη λέξη-κλειδί, τη χρονική κατανομή των αποτελεσμάτων και τη συχνότητά των λέξεων στα αποτελέσματα που βρέθηκαν. Υπάρχει επίσης η δυνατότητα για ταυτόχρονη αναζήτηση δύο λέξεων-κλειδιών και εμφάνιση στην ίδια οθόνη των αντίστοιχων αποτελεσμάτων.

### 2.3.3. Kongress

Στο ίδιο μήκος κύματος κινείται και η εφαρμογή Kongress [12]. Η εφαρμογή έχει ως κύριο πεδίο δράσης την εξόρυξη πληροφορίας από δεδομένα που προέρχονται από μέσα κοινωνικής δικτύωσης, στην προκειμένη φάση της μόνο από το Twitter. Η εφαρμογή προορίζεται να λειτουργεί μόνο ως υπηρεσία σε έξυπνα τηλέφωνα (smartphones) και ο κύριος σκοπός της είναι η συλλογή tweets με τη χρήση του Twitter API από μέλη του αμερικανικού Κογκρέσου, τους χρήστες που αυτά τα μέλη ακολουθούν (followers), αλλά και χρήστες που ακολουθούν τα ίδια τα μέλη (followings), από τα όποια θα βγαίνει το συμπέρασμα ποιος χρήστης – μέλος του Κογκρέσου επηρεάζει περισσότερο τους συναδέλφους του μέσα στο Κογκρέσο και ποιοι χρήστες followers ή followings ασκούν μεγαλύτερη επιρροή στα μέλη του Κογκρέσου. Η εφαρμογή αυτή πηγαίνει το όλο εγχείρημα ακόμα ένα βήμα πιο πέρα, καθώς θα αποπειράται να οπτικοποιεί αυτές τις σχέσεις αλληλεπίδρασης μέσω ενός οπτικού δικτύου ανάλυσης, το οποίο μάλιστα θα έχει και γεωγραφικές διαστάσεις, αφού η θέση όλων των χρηστών θα απεικονίζεται πάνω σε γεωγραφικό χάρτη.
Όλα αυτά είναι ιδιαιτέρως ενδιαφέροντα και αποδεικνύουν πως το φιλτράρισμα και η αποθήκευση δεδομένων από κοινωνικά δίκτυα με συντεταγμένο τρόπο είναι το πρώτο βήμα, για να εφαρμοστούν πάνω σ’ αυτά πολύ πιο σύνθετες και εξειδικευμένες αναλύσεις που μπορούν να οδηγήσουν σε άκρως ενδιαφέροντα συμπεράσματα και διαπιστώσεις. Γι’ αυτό το πρώτο βήμα λοιπόν η παρούσα εφαρμογή χρησιμοποιεί το Elasticsearch ως κύρια μονάδα ευρετηρίασης και αποθήκευσης των δεδομένων.
Από αυτήν την εφαρμογή μπορεί εύκολα να αναγνωριστεί η υπεροχή του Elasticsearch στο χώρο των μηχανών αναζήτησης σε πραγματικό χρόνο και ταυτόχρονα αποδεικνύεται πως αυτές οι δυνατότητες δεν θα μπορούσαν να εξασφαλιστούν από καμία βάση δεδομένων είτε σχεσιακή είτε NoSQL. Στο Kongress λοιπόν ο χρήστης αναζητώντας μια λέξη-κλειδί μπορεί να δει αποτελέσματα από τρεις

διαφορετικές πηγές, δηλαδή από τις ψηφοφορίες των μελών του Κογκρέσου, από τα tweets που έχουν δημοσιεύσεις τα ίδια, αλλά και οι followers και οι followings τους. Για να μπορούν αυτές οι αναζητήσεις να επιστρέφουν σωστά αποτελέσματα, οι δημιουργοί της εφαρμογής έχουν κατασκευάσει 5 διαφορετικά ευρετήρια αντί για ένα. Διαθέτουν ξεχωριστά ευρετήρια για τα νομοσχέδια που ψηφίστηκαν, για τα μητρώα ψήφων των μελών του Κογκρέσου, για στοιχεία του κάθε χρήστη στο Twitter (id, URL χρήστη κτλ.), για τα ίδια τα tweets και επιπλέον ένα που αποθηκεύει τις συνδέσεις μεταξύ των μελών και των followers / followings τους. Όλα αυτά τα ευρετήρια και κυρίως το τελευταίο έχουν ως σκοπό την επιτάχυνση των αναζητήσεων και την επίτευξη μεγαλύτερης ακρίβειας στην ανάκτηση των αποτελεσμάτων.

2.3.4. Αναζήτηση στο Liber Usualis

Πρόκειται για μια εφαρμογή της οποίας τα δεδομένα προέρχονται από μια έντυπη έκδοση του 1961, 2340 σελίδων, ύμνων της καθολικής εκκλησίας με το όνομα Liber Usualis. Αυτή, αφού ψηφιοποιήθηκε μέσω λογισμικού οπτικής αναγνώρισης χαρακτήρων (OCR), τα ηλεκτρονικά έγγραφα που προέκυψαν αποθηκεύτηκαν σε μια NoSQL βάση δεδομένων και συγκεκριμένα στην CouchDB. Τα ηλεκτρονικά αυτά έγγραφα δεν αποτελούνται μόνο από κείμενα, αλλά και από μουσικές νότες. Για την ευρετηρίαση και την αποθήκευση των εγγράφων χρησιμοποιούνται συνδυαστικά τόσο η CouchDB όσο και το Elasticsearch. Το σημαντικό που πρέπει να σημειωθεί από αυτήν την εφαρμογή είναι πως η ευρετηρίαση των εγγράφων δεν έγινε στο επίπεδο του Elasticsearch, όπως στην εφαρμογή της παρούσας διπλωματικής, αλλά κατά το πρώτο επίπεδο της αποθήκευσης στη βάση δεδομένων, κάνοντας χρήση των δυνατοτήτων ευρετηρίασης των NoSQL βάσεων δεδομένων. Κυρίως επειδή πρόκειται κατά κύριο λόγο για μη εξ ολοκλήρου κειμενικά δεδομένα, δημιουργήθηκαν πολλαπλές συλλογές μέσα στην βάση δεδομένων, όπου για κάθε συλλογή υπήρχε διαφορετικό ευρετήριο με τη χρήση της μεθόδου n-gram. Δηλαδή μια συλλογή είχε χωρισμένα τα δεδομένα με 2-gram, άλλη με 3-gram κοκ. Σύμφωνα με τους δημιουργούς της εφαρμογής αυτή η επιλογή, παρόλο που δημιουργεί πληθώρα εγγραφών, βοηθά στην ταχύτερη αναζήτηση των δεδομένων που είναι μη κειμενικά δεδομένα και η πληροφορία βρίσκεται σε επίπεδο νότας, δηλαδή ενός χαρακτήρα και όχι σε επίπεδο λέξης, όπου οι χαρακτήρες είναι περισσότεροι του ενός.

2.4. Συμπεράσματα

Σε γενικές γραμμές από τις παραπάνω εφαρμογές και εργασίες που παρουσιάστηκαν φαίνεται πως η επιλογή του Elasticsearch ως κύριου τρόπου ευρετηρίασης και αποθήκευσης δεδομένων που προέρχονται από κείμενα που προέρχονται είτε από κοινωνικά δεδομένα είτε από άλλες πηγές είναι ιδιαίτερα χρήσιμή και αυτό οφείλεται κυρίως στο γεγονός ότι τόσο τα δεδομένα δίνονται σε μορφή JSON όσο και ο τρόπος αποθήκευσης στο Elasticsearch βασίζεται σ’ αυτήν την μορφή. Η επιλογή αυτή γίνεται, παρόλες τις αδυναμίες του Elasticsearch που υπογραμμίστηκαν παραπάνω. Τέλος, φάνηκε πως και η συνεργασία βάσεων δεδομένων NoSQL και Elasticsearch αποτελεί μια λύση που οδηγεί σε θετικά αποτελέσματα εξαιτίας της ομοιότητας της χρησιμοποιουμένης δομής δεδομένων και της απουσίας σχήματος.

# 3. Παρουσίαση Τεχνολογιών

## 3.1. Εισαγωγή

Το παρόν κεφάλαιο έχει ως σκοπό πρώτα να παραθέσει και κατόπιν να περιγράψει τις πιο σημαντικές τεχνολογίες που χρησιμοποιήθηκαν για την κατασκευή της εφαρμογής. Επίσης, θα γίνει προσπάθεια να καταγραφούν τα θετικά και τα τυχόν αρνητικά σημεία των τεχνολογιών, όπως αυτά έγιναν κατανοητά κατά τη διάρκεια της ανάπτυξης της εφαρμογής. Παράλληλα, με τα παραπάνω θα καταδειχθεί και ο ρόλος των τεχνολογιών με την περιγραφή της χρήσης τους στα διάφορα συστατικά που συνθέτουν τις λειτουργίες της εφαρμογής.

## 3.2. NoSQL Βάσεις Δεδομένων

Ο πιο γνωστός τύπος βάσεων δεδομένων είναι οι σχεσιακές βάσεις δεδομένων, οι οποίες πρωτοεμφανίστηκαν τη δεκαετία του 70. Αυτές οι βάσεις δεδομένων, πασίγνωστες σήμερα, δημιουργήθηκαν, έτσι ώστε να αποθηκεύουν δεδομένα βάσει ενός προκαθορισμένου μοντέλου και να προσπελαύνονται βάσει μιας συγκεκριμένης γλώσσας δημιουργίας ερωτημάτων, τη Δομημένη Γλώσσα Ερωτημάτων (SQL). Εκείνη την εποχή, ο χώρος αποθήκευσης δεδομένων ήταν ακριβός και τα σχήματα αποθήκευσης (data schemas) αρκετά απλά. Όμως, ιδιαίτερα μετά την εξάπλωση του Web 2.0, η ποσότητα των δεδομένων προς αποθήκευση αυξήθηκε σε δυσθεώρητα επίπεδα. Εκτός αυτού, η πρόσβαση στα δεδομένα γίνεται συχνότερα και ο βαθμός επεξεργασίας τους είναι πολύ μεγαλύτερος. Σ’ αυτό έχουν συντελέσει καθοριστικά οι τεράστιες ποσότητες δεδομένων που παράγονται από τις κινήσεις εκατομμυρίων χρηστών ανά τον κόσμο μέσα στα κοινωνικά δίκτυα. Δεδομένα, από τα οποία μπορούν να συναχθούν πληροφορίες με τεράστια οικονομική, εμπορική και όχι μόνο χρησιμότητα. Παράλληλα με τα παραπάνω, σήμερα τόσο οι υπολογιστικές δομές όσο και οι στρατηγικές ανάπτυξης υπολογιστικών συστημάτων έχουν αλλάξει ριζικά. Τα φθηνά σε κόστος συστήματα νεφοϋπολογιστικής (cloud computing) έχουν αντικαταστήσει τις δαπανηρές και πολύπλοκες δομές που βασίζονται σε έναν και μόνο εξυπηρετητή (Server). Έτσι, οι νέες εφαρμογές που κάνουν χρήση αυτών των νέων εξελίξεων γίνονται αποδέκτες τεράστιων όγκων δεδομένων που ζητούν χώρους αποθήκευσης, ώστε να επεξεργαστούν κατάλληλα.

Οι σχεσιακές βάσεις δεδομένων δεν σχεδιάστηκαν με σκοπό να διαχειρίζονται δεδομένα σε τόσο μεγάλη ποσότητα και σε τόσο μμεγάλη ταχύτητα που απαιτούν οι νέες εφαρμογές ούτε κατέχουν εκείνα τα χαρακτηριστικά που μμπορούν να τις καταστήσουν συμβατές με τεράστιες αποθήκες δεδομένων και με συστήματα με τεράστια υπολογιστική δεινότητα, όπως αυτά που προσφέρει σήμερα η νεφοϋπολογιστική [13].

Οι βάσεις δεδομένων NoSQL διαθέτουν μια σειρά από χαρακτηριστικά που τις διαφοροποιούν από τις παραδοσιακές σχεσιακές βάσεις δεδομένων. Βασικό χαρακτηριστικό τους είναι ότι επιτρέπουν την εισροή δεδομένων χωρίς να χρειάζεται από πριν να υπάρχει κάποιο προκαθορισμένο μοντέλο, όπως το μοντέλο οντοτήτων – συσχετίσεων (E-R Model) των σχεσιακών βάσεων. Αυτό δίνει τη δυνατότητα να γίνονται αλλαγές στη βάση δεδομένων σε πραγματικό χρόνο, χωρίς να υπάρχει η ανησυχία για τυχόν κατάρρευση του συστήματος, με το οποίο είναι συνδεδεμένη η βάση [14]. Επίσης, η έλλειψη προκαθορισμένου μοντέλου διευκολύνει τη δημιουργία οποιασδήποτε βάσης σε πολύ σύντομο χρόνο, ελαχιστοποιεί την ποσότητα του κώδικα που πρέπει να συνταχθεί και μείωνε το χρόνο που χρειάζεται για τη μετέπειτα διαχείριση και συντήρηση της βάσης.

Επόμενο χαρακτηριστικό των NoSQL βάσεων είναι η δυνατότητα του αυτόματου κατακερματισμού (auto-sharding). Αυτό σημαίνει πως οι βάσεις αυτές έχουν την εγγενή δυνατότητα να διανέμουν αυτόματα τα δεδομένα σ’ έναν αριθμό από εξυπηρετητές, χωρίς να υπάρχει η προηγούμενη απαίτηση από την εφαρμογή να έχει ορίσει τη διάταξη αυτών των εξυπηρετητών. Τόσο τα δεδομένα όσο και τα ερωτήματα αυτόματα κατανέμονται ισορροπημένα στους εξυπηρετητές. Σε περίπτωση που ένας από

τους εξυπηρετητές καταρρεύσει, υπάρχει η δυνατότητα γρήγορης αντικατάστασής του, χωρίς να διαταραχθεί η λειτουργία της εφαρμογής.
Εξίσου σημαντικό χαρακτηριστικό τους είναι η αντικατάσταση (replication). Αυτό σημαίνει πως δίνεται από την ίδια τη βάση η δυνατότητα αποκατάστασης μετά από κατάρρευση (disaster recovery), χωρίς να απαιτείται η μεσολάβηση της εφαρμογής στη διαχείριση των διαδικασιών αυτών. Ο ίδιος ο κατασκευαστής της βάσης μπορεί να δομήσει εικονικά το αποθηκευτικό περιβάλλον της βάσης σύμφωνα με τις προδιαγραφές που επιθυμεί.
Τέλος, ακόμα ένα σημαντικό πλεονέκτημά των βάσεων αυτού του είδους είναι οι ενσωματωμένες δυνατότητες προσωρινής αποθήκευσης (caching) που διαθέτουν. Μπορούν να διατηρούν στη μνήμη του συστήματος τα δεδομένα που χρησιμοποιούνται με μεγαλύτερη συχνότητα από το χρήστη. Αυτό αφαιρεί την ανάγκη διατήρησης ενός ξεχωριστού επιπρόσθετου επιπέδου προσωρινής αποθήκευσης [15].

### 3.2.1. Ορισμός και είδη NoSQL βάσεων δεδομένων

NoSQL ουσιαστικά σημαίνει “όχι σχεσιακή βάση” και “όχι SQL” [16], επομένως από τον ίδιο τον ορισμό γίνεται αντιληπτό πως υπάρχουν πολλοί διαφορετικοί τρόποι, για να εφαρμοστεί η NoSQL τεχνολογία. Γενικότερα, οι NoSQL βάσεις δεδομένων περιλαμβάνουν τις παρακάτω υλοποιήσεις:

- Βάσεις δεδομένων εγγράφων (**Document database**): Κάθε κλειδί ταιριάζει με μια ομάδα εγγράφων. Τα έγγραφα μμπορούν να διαθέτουν πολλά διαφορετικά ζευγάρια κλειδιού - τιμής ή κλειδιού-πίνακα ή ακόμα και εμφωλευμένα έγγραφα. Η βασικότερη εκπρόσωπος αυτής κατηγορίας, αλλά και μια βάση από τις πιο γνωστές στον χώρο των NoSQL βάσεων είναι το Elasticsearch.
- Βάσεις γράφων (**Graph stores**): Αυτού του είδους η υλοποίηση χρησιμοποιείται, για να αποθηκεύσει πληροφορίες σχετικές με δίκτυα, όπως παραδείγματος χάρη τα κοινωνικά δίκτυα. Τέτοιες βάσεις δεδομένων είναι η Neo4J και η HyperGraphDB.
- Βάσεις κλειδιού - τιμής (**Key-Value stores**): Αυτές οι βάσεις είναι οι πιο απλές του τύπου NoSQL. Κάθε διαφορετικό στοιχείο μέσα στη βάση είναι αποθηκευμένο ως χαρακτηριστικό όνομά ιδιότητας, ή κλειδί, μαζί με την τιμή του. Τέτοιες βάσεις είναι η Riak και η Voldemort. Η Redis επιτρέπει, επίσης, στην κάθε τιμή να έχει και ένα τύπο, όπως παραδείγματος χάρη “ακέραιος”, πράγμα που προσθέτει μεγαλύτερη λειτουργικότητα στη βάση.
- Βάσεις ευρείας στήλης (**Wide-Column stores**): Εδώ εκπρόσωποι αυτού του είδους είναι η Cassandra και HBase, οι οποίες έχουν βελτιστοποιηθεί, έτσι ώστε να υπάρχει η δυνατότητα να εφαρμοστούν ερωτήματα σε βάσεις με τεράστιες ποσότητες δεδομένων.

## 3.3. Elasticsearch

Το Elasticsearch χρησιμοποιείται ως εργαλείο για αναζήτηση λέξεων μέσα από κειμενικά δεδομένα. Το εύρος των δυνατοτήτων του είναι αρκετά μεγάλο και μερικές από αυτές τις δυνατότητες θα αναλυθούν στη συνέχεια, αλλά κατά κόρον είναι φτιαγμένο, για να αποτελεί την αναγκαία υποδομή, πάνω στην οποία δημιουργούνται μμηχανές αναζήτησης, αλλά και για να παρέχει στατιστικές αναλύσεις πάνω σε σώματα κειμένων. Στην παρούσα εφαρμογή λειτουργεί ως ένας εναλλακτικός τρόπος αποθήκευσης και αναζήτησης κειμενικών δεδομένων [17].

### 3.3.1. Χαρακτηριστικά και Δομή του Elasticsearch

Πιο συγκεκριμένα, το Elasticsearch είναι ένας ανεξάρτητος εξυπηρετητής βάσης δεδομένων ( standalone database server ), γραμμένο σε Java, που δέχεται δεδομένα και τα αποθηκεύει με τέτοιο τρόπο, ώστε να διευκολύνει αναζητήσεις πάνω σ’ αυτά. Είναι ιδιαίτερα εύχρηστο προγραμματιστικά, καθώς το κυρίως πρωτόκολλο που χρησιμοποιεί βασίζεται σε HTTP/JSON. Ο ίδιος ο μηχανισμός του HTTP είναι εξ ολοκλήρου ασύγχρονος από μόνος του, επομένως κανένα νήμα πληροφορίας που αποστέλλεται δεν χρειάζεται να περιμένει απόκριση. Το Elasticsearch προσφέρει μεγάλες δυνατότητες επεκτασιμότητας, αφήνοντας περιθώρια ποικίλων ομαδοποιήσεων, πράγμα το οποίο αντικατοπτρίζει και το ίδιο το όνομα του. Επίσης, η χρήση του μπορεί να εφαρμοστεί σε διάφορα πεδία αναζητήσεων, όπως παραδείγματος χάρη, στα προϊόντα που διαθέτει μια βάση δεδομένων ή στα άρθρα που διαθέτει ένα ιστολόγιο. Το Elasticsearch διαθέτει τις δυνατότητες να ξεπερνά τα εμπόδια που δημιουργεί στις αναζητήσεις η φυσική γλώσσα [18].

### 3.3.2. Apache Lucene

Ο πυρήνας του Elasticsearch στην ουσία είναι ένα άλλο λογισμικό, ιδιαίτερα γνωστό και με ισχυρές δυνατότητες, το Apache Lucene. Το Elasticsearch μπορεί να μελετηθεί περισσότερο σε βάθος, αν μελετηθεί παράλληλα με το Lucene και τις δικές του υποδομές. Οι αλγόριθμοι του Elasticsearch που σχετίζονται είτε με ταίριασμα κειμένου (text matching) είτε με βελτιστοποιημένα ευρετήρια όρων προς αναζήτηση προέρχονται από τις υποδομές του Lucene. Το νέο που κομίζει το Elasticsearch στην ήδη υπάρχουσα τεχνολογία του Lucene είναι το πιο εύχρηστο και ακριβές API, η επεκτασιμότητα, η δια λειτουργικότητα με γλώσσες προγραμματισμού πέραν της Java, η ευχρηστία προγραμματιστικής διαχείρισής του, η ομαδοποίηση (clustering) και η αντικατάσταση (replication), γνωρίσματα που ήδη αναφέρθηκαν στις NoSQL τεχνολογίες και καινούργιες, πιο εύχρηστες εφαρμογές, οι οποίες αποτελούν επεκτάσεις των Java κλάσεων του Lucene [19].
Το Apache Lucene είναι ένα δωρεάν λογισμικό ανοιχτού κώδικα για ανάκτηση πληροφορίας. Δημιουργήθηκε από τον Doug Cutting το 1999 και είναι γραμμένο σε Java [20]. Μπορεί να ανταποκριθεί στις ανάγκες οποιασδήποτε εφαρμογής αφορά αποθήκευση και έρευνα πάνω σε δεδομένα κειμένου. Το Elasticsearch είναι document NoSQL database και ταυτόχρονα και μηχανή αναζήτησης ανοικτού κώδικα που βασίστηκε πάνω στο Lucene. Αναπτύχθηκε σε Java το 2010 από τον Shay Bannon και χρόνο με τον χρόνο πείθει όλο και περισσότερες εταιρείες να το χρησιμοποιήσουν στις εφαρμογές τους. Διάσημοι χρήστες του είναι το Facebook, η Mozilla και το Cern. Το πλεονέκτημά του είναι ότι παρέχει ένα κατανεμημένο περιβάλλον διαχείρισης και full text αναζήτησης, σε πραγματικό χρόνο μέσω αιτημάτων πάνω στο πρωτόκολλο HTTP, εγγράφων σε μμορφή JSON. Όλες οι παραπάνω δυνατότητες που προσφέρει έγκεινται στην αρχιτεκτονική πάνω στην οποία βασίστηκε. Η βασική μονάδα πληροφορίας του Elasticsearch είναι τα έγγραφα (documents) σε μορφή JSON. Κάθε document είναι ένα σύνολο ζευγών πεδίων-κλειδιών. Το επόμενο χρήσιμο κομμάτι είναι οι αντιστοιχίες (mappings). Ομοιάζουν με τα σχήματα βάσης δεδομένων (database schemas) στις σχεσιακές βάσεις δεδομένων. Ο σκοπός του είναι να περιγράψουν το είδος των πεδίων των εγγράφων και τον τρόπο με τον οποίο πρέπει να αποθηκευτούν από το Lucene. Πολλά documents μαζί, σε συνδυασμό με το mapping τους συγκροτούν έναν τύπο (type). Ο τύπος είναι το αντίστοιχο του Elasticsearch για τους πίνακες των σχεσιακών συστημάτων οργάνωσης βάσεων δεδομένων. Πολλά types μαζί συγκροτούν έναν κατάλογο (index) δηλαδή την αντίστοιχη "βάση δεδομένων". Μια ακόμα χρήσιμή δυνατότητα που προσφέρεται είναι η δυνατότητα διαίρεσης του index σε κομμάτια που καλούνται shards. Αυτό επιτρέπει στις λειτουργίες να παραλληλοποιούνται κατά μήκος των shards, τα οποία μεγαλώνουν οριζοντίως όσο αυξάνονται τα δεδομένα, αυξάνοντας τελικά την απόδοση του συστήματος. Όπως φαίνεται στην εικόνα 6 ένας κόμβος (node) είναι ένας server ο οποίος αποθηκεύει και μπορεί να εκτελέσει ερωτήματα πάνω σε indices του Elasticsearch. Πολλοί κόμβοι μαζί συγκροτούν μια συστάδα (cluster). Όλοι μαζί οι κόμβοι της συστάδας περιέχουν όλα τα δεδομένα της εφαρμογής. Η οργάνωση σε συστάδες προσφέρει δυνατότητες καταχώρησης και αναζήτησης σε όλους του κόμβους που την απαρτίζουν. Κατ' αυτόν τον τρόπο λοιπόν εξασφαλίζεται η 'ενιαία' διαχείριση σε δεδομένα κατανεμημένα σε διαφορετικούς κόμβους [21].

### 3.3.3. Ευρετήριο

Βασικό συστατικό του Elasticsearch είναι το ευρετήριο (index), όπου και αποθηκεύονται τα δεδομένα. Όπως έχει προαναφερθεί και στην περιγραφή των NoSQL βάσεων δεδομένων, τα ευρετήρια για το Elasticsearch έχουν ακριβώς την ίδια λειτουργία με τους πίνακες των σχεσιακών βάσεων. Αλλά αντίθετα από αυτές, οι τιμές που είναι αποθηκευμένες σ' ένα ευρετήριο προορίζονται, ώστε να συνδράμουν στην επιτάχυνση μιας πιθανής αναζήτησης σε κειμενικά δεδομένα. Για την ακρίβεια το ανάλογο ενός ευρετηρίου στο Elasticsearch είναι μια συλλογή της MongoDB.

### 3.3.4. Δομή

Η κυρίως αποθηκευτική οντότητα του Elasticsearch είναι το έγγραφο. Σε αναλογία προς τις σχεσιακές βάσεις, ένα έγγραφο είναι μια γραμμή από δεδομένα σ' έναν πίνακα. Συγκρίνοντας ένα έγγραφο του Elasticsearch με ένα έγγραφο της MongoDB, και τα δύο μπορούν να έχουν διαφορετικές δομές, αλλά αυτό του Elasticsearch πρέπει να διαθέτει τους ίδιους τύπους για κοινά πεδία. Τα έγγραφα αποτελούνται από πεδία (γραμμές), αλλά κάθε πεδίο μπορεί να εμφανίζεται περισσότερες από μια φορές, τότε σ' αυτήν την περίπτωση ονομάζεται πεδίο πολλαπλών τιμών. Κάθε πεδίο διαθέτει ένα τύπο (κείμενο, νούμερο, ημερομηνία κοκ), όπως επίσης μπορεί να εμπεριέχει ένα μέρος από κάποιο έγγραφο (sub document) ή και πίνακες. Οι τύποι των πεδίων μπορούν επίσης να είναι σύνθετοι. Ο τύπος των πεδίων έχει ιδιαίτερη σημασία στο Elasticsearch, γιατί παρέχει στην εκάστοτε

μηχανή αναζήτησης πληροφορίες για το πόσο συχνές διαδικασίες, παραδείγματος χάρη σύγκρισης ή ταξινόμησης, πρέπει να λάβουν χώρα. Στην περίπτωση του Elasticsearch βέβαια ο καθορισμός αυτών των διαδικασιών επιτυγχάνεται με αυτόματο τρόπο. Από την άλλη, στις σχεσιακές βάσεις τα έγγραφα δεν απαιτείται να έχουν προκαθορισμένη δομή. Το κάθε έγγραφο μπορεί να διαθέτει διαφορετικό σύνολο πεδίων και επίσης τα πεδία δεν είναι απαραίτητο να είναι γνωστά πριν από την ανάπτυξη της εφαρμογής, παρά μόνο αν από πριν καθοριστεί κάποιου είδους σχήματος.
Εξίσου σημαντικό στο Elasticsearch είναι και ο τύπος του εγγράφου, όπου σ' ένα ευρετήριο μπορούν να αποθηκευτούν πολλές οντότητες με διαφορετικούς σκοπούς. Για παράδειγμα, σ' ένα ιστολόγιο που διαθέτει ταυτόχρονα άρθρα και σχόλια, ο τύπος του εγγράφου διαφοροποιεί εύκολα αυτά τα δύο διαφορετικά είδη κειμενικών δεδομένων. Είναι επίσης πολύ σημαντικό να επισημανθεί πως το κάθε έγγραφο μπορεί να έχει τη δική του διαφορετική δομή και πως αυτός ο διαχωρισμός βοηθά σημαντικά στη διαχείριση των δεδομένων. Όμως δεν πρέπει να αγνοούνται και οι περιορισμοί που υφίστανται, καθώς οι διαφορετικοί τύποι ενός εγγράφου δεν μπορούν να θέσουν διαφορετικούς τύπους για την ίδια οντότητα.

3.3.5. Ανεξάρτητος Εξυπηρετητής

Επίσης, το Elasticsearch έχει την ιδιότητα να λειτουργεί και ως ανεξάρτητος εξυπηρετητής, αλλά και να λειτουργεί μέσω πολλαπλών συνεργαζόμενων εξυπηρετητών, ώστε να μπορεί να επεξεργάζεται ογκώδη σύνολα δεδομένων και να διαθέτει ανοχή σε τυχόν σφάλματα. Το σύνολο αυτών των συνεργαζόμενων εξυπηρετητών ονομάζεται σύμπλεγμά (cluster) και ο καθένας από αυτούς ξεχωριστά κόμβος (node).

3.3.6. Κατακερματισμός

Σημαντικό χαρακτηριστικό του Elasticsearch αποτελεί και ο κατακερματισμός, ο οποίος είναι χρήσιμος στην περίπτωση, όπου ένας κόμβος υπερφορτωθεί από δεδομένα είτε εξαιτίας έλλειψης μνήμης τυχαίας προσπέλασης (RAM) είτε έλλειψης χώρου στο σκληρό δίσκο είτε οποιαδήποτε άλλης αδυναμίας λογισμικού. Σ' αυτές τις περιπτώσεις, υπάρχει η ευελιξία τα δεδομένα να κατακερματιστούν σε μικρότερα κομμάτια, τα λεγόμενα θραύσματα (shards), όπου το καθένα από αυτά στην ουσία αποτελεί ένα ανεξάρτητο ευρετήριο με τις ιδιότητες που κληρονομεί από το Lucene. Κάθε θραύσμα μπορεί να τοποθετηθεί σε διαφορετικό εξυπηρετητή και μ' αυτόν τον τρόπο τα δεδομένα διαχέονται σε ολόκληρο το σύμπλεγμά των εξυπηρετητών. Όταν τίθενται ερωτήματα σε ένα ευρετήριο, τα αποτελέσματα που επιστρέφονται ουσιαστικά συντίθενται από πολλαπλά θραύσματα. Το Elasticsearch αποστέλλει το εκάστοτε ερώτημα στα σχετικά θραύσματα και συνενώνει τα αποτελέσματα με τέτοιο τρόπο που δεν επιβαρύνει τη λειτουργία της εφαρμογής περαιτέρω, καθώς η εύρεση των θραυσμάτων είναι λειτουργία που διενεργείται αποκλειστικά από το ίδιο [22].

3.3.7. Εισαγωγή Δεδομένων στο Elasticsearch

Το Elasticsearch είναι όπως ειπώθηκε παραπάνω μια μηχανή αναζήτησης σε σχεδόν πραγματικό χρόνο που στηρίζεται στην αρχιτεκτονική REST [23]. Το Rest (Representational State Transfer) αποτελεί ένα σύνολο αρχών σχεδίασης μιας υπηρεσίας που επικεντρώνεται στα δεδομένα του συστήματος. Η μεταβολή της κατάστασης των δεδομένων του συστήματος περιγράφεται και μεταφέρεται στο σύστημα μέσω του πρωτοκόλλου HTTP. Τα βασικά χαρακτηριστικά του είναι δυο. Πρώτον, ότι για την επικοινωνία μμεταξύ χρήστη και συστήματος χρησιμοποιούνται αποκλειστικά HTTP αιτήματα. Μάλιστα, η βασική του λογική είναι να υπάρχει ένα-προς-ένα αντιστοιχία μεταξύ των HTTP μεθόδων και των λειτουργιών CRUD (Create Read Update Delete). Έτσι, για την δημιουργία ενός πόρου στο σύστημα χρησιμοποιούμε την μέθοδο POST, για το διάβασα ενός πόρου τη GET, για την ενημέρωση ή την επεξεργασία του την μέθοδο PUT και για την διαγραφή την μέθοδο DELETE. Δεύτερον, η συγκεκριμένη αρχιτεκτονική θεωρείται stateless, με την έννοια ότι δεν υπάρχει οποιαδήποτε εξάρτηση μεταξύ δυο REST κλήσεων. Το Elasticsearch είναι οργανωμένο σε APIs [24] (application programming interfaces), δηλαδή σε σύνολα συναρτήσεων και εξειδικευμένα εργαλεία που το καθένα επιτελεί κάποιο συγκεκριμένο σκοπό. Στο παρόν κεφάλαιο θα γίνει αναφορά στο BULK_API του Elasticsearch. Το BULK_API επιτρέπει να γίνουν πολλαπλές πράξεις εισαγωγής/ διαγραφής/ενημέρωσης σε μια κλήση του API. Αυτό έχει σαν συνέπεια την αύξηση της ταχύτητας της ευρετηρίασης (indexing speed). Η δομή που πρέπει να έχει ένα αρχείο ακολουθεί την εξής μορφή : Για κάθε εγγραφή που βρίσκεται σε μια γραμμή του αρχείου πρέπει ακριβώς από πάνω της να βρίσκεται μια γραμμή που ορίζει σαφώς τι πράξη θα πραγματοποιηθεί και που. Υποχρεωτικά στο τέλος κάθε γραμμής του αρχείου πρέπει να υπάρχει ο χαρακτήρας αλλαγής γραμμής "\n". Όσον

αφορά την εισαγωγή των δεδομένων λαμβάνοντας υπόψιν ότι είχαμε να εισάγουμε περί τις 19000 αποφάσεις η χρήση του συγκεκριμένου API ήταν κάτι παραπάνω από επιτακτική.

3.3.8. Αντικατάσταση

Τέλος, σημαντικό ρόλο στο Elasticsearch παίζουν και τα αντίγραφα που μπορεί να έχει ένα θραύσμα (shard replicas), τα όποια προσδίδουν στην εφαρμογή μεγάλο βαθμό διαθεσιμότητας όσον αφορά τα δεδομένα. Το πρωτεύον θραύσμα χρησιμοποιείται ως το μέρος, όπου κατευθύνονται οι λειτουργίες εκείνες που προκαλούν αλλαγές στο ευρετήριο. Ένα αντίγραφο (replica) είναι μια πιστή αναπαραγωγή του πρωτεύοντος θραύσματος και κάθε θραύσμα μπορεί να έχει από κανένα ως έναν αριθμό από αντίγραφα. Στην περίπτωση εκείνη, όπου το πρωτεύον θραύσμα δεν ανταποκρίνεται λόγω μη διαθεσιμότητας που οφείλεται σε αδυναμία του υλικού, τότε ένα σύμπλεγμά από εξυπηρετητές μπορεί να προαγάγει ένα αντίγραφο ως το νέο πρωτεύον θραύσμα.

3.3.9. Διαδικασία Ευρετηρίασης

Στη συνέχεια, μετά την περιγραφή της δομής και των χαρακτηριστικών του Elasticsearch, θα ήταν χρήσιμο να γίνει αναφορά για τη διαδικασία που ακολουθείται από την στιγμή που ένα έγγραφο αποθηκεύεται εντός του Elasticsearch μέχρι τη στιγμή που ανακτάται μέσω κάποιου ερωτήματος.

Η πρόσθεση δεδομένων μέσα σ' ένα ευρετήριο μπορεί να γίνει είτε απευθείας, απλώς με την αποθήκευση μερικών εγγράφων χωρίς προηγούμενο καθορισμό κάποιου ειδικού ευρετηρίου είτε μπορεί ο χρήστης να επιλέξει τον ευρετηριασμό των δεδομένων με συγκεκριμένο τρόπο. Το Elasticsearch διαθέτει αρκετές επιλογές για τον τρόπο, με τον οποίο μπορεί να γίνει ευρετηρίαση δεδομένων. Οι δύο βασικοί μέθοδοι που ακολουθούνται για τον εξειδικευμένο καθορισμό ενός ευρετηρίου είναι η ανάλυση (analysis) και η χαρτογράφηση (mapping), οι οποίες και θα αναλυθούν παρακάτω.

3.3.10. Ανάλυση

Στο Elasticsearch, με τη δημιουργία ενός ευρετηρίου ταυτόχρονα καθορίζεται και ο αριθμός των θραυσμάτων και των αντιγράφων που θα χρησιμοποιηθούν. Όπως προαναφέρθηκε, ένα θραύσμα είναι ένα κομμάτι από τα δεδομένα και το αντίγραφο είναι μια επανάληψη των ίδιων δεδομένων με τη μορφή ενός αντιγράφου ασφαλείας (backup). Όταν λοιπόν υπάρχει ένας κόμβος, τότε όλα τα θραύσματα και όλα τα αντίγραφα βρίσκονται μέσα σε αυτόν τον κόμβο. Ενώ στην περίπτωση που οι κόμβοι είναι περισσότεροι, τότε τα δεδομένα κατανέμονται ανάμεσα σ' αυτούς τους κόμβους.

Τα δεδομένα στο Elasticsearch αναλύονται σε δύο φάσεις. Η πρώτη φορά είναι όταν ένα έγγραφο αποθηκεύεται και η πληροφορία της ανάλυσής του καταγράφεται και αυτή στο ευρετήριο και η δεύτερη φορά είναι όταν διενεργείται κάποιο ερώτημα. Το Elasticsearch αναλύει το ερώτημα της αναζήτησης και ψάχνει στις αποθηκευμένες πληροφορίες του ευρετηρίου [25].

Η διαδικασία της ανάλυσης ευρετηρίου (index analysis) λειτουργεί ως ένα ρυθμιζόμενο μητρώο αναλυτών (analyzers), οι οποίοι μπορούν τόσο να αναλύσουν σε πεδία ένα έγγραφο κατά την είσοδό του όσο και ένα ερώτημα με τη μορφή ακολουθίας συμβόλων (string). Οι αναλυτές γενικότερα αποτελούνται από έναν και μόνο διαχωριστή λεκτικών μονάδων (tokenizer) και από κανένα μέχρι περισσότερα φίλτρα λεκτικών μονάδων. Ως διαχωριστής λεκτικών μονάδων ορίζεται το εργαλείο εκείνο που λαμβάνει ως είσοδο μια ακολουθίας συμβόλων (string) και επιστρέφει το σύνολο των λεκτικών μονάδων που εμπεριέχονται στην ακολουθία αυτή. Παραδείγματος χάρη, ο προκαθορισμένος αναλυτής (standard analyzer) αποτελείται από τον προκαθορισμένο διαχωριστή λεκτικών μονάδων, ο οποίος εφαρμόζει το προκαθορισμένο φίλτρο λεκτικών μονάδων, όπου κανονικοποιεί όλες τις λεκτικές μονάδες, το φίλτρο μικρογράμματων λέξεων, όπου μετατρέπει όλες τις λεκτικές μονάδες με μικρά γράμματα και το φίλτρο των stop words λέξεων, όπου απομακρύνει όλες τις λέξεις που γραμματικά είναι άρθρα, αντωνυμίες, σύνδεσμοι κτλ., δηλαδή μικρές σε μέγεθος λέξεις με περισσότερο λειτουργικό χαρακτήρα και λιγότερο σημασιολογικό [26].

3.3.11. Χαρτογράφηση

Στη συνέχεια ακολουθεί η χαρτογράφηση, κατά την οποία καθορίζεται το είδος των δεδομένων που θα τοποθετηθεί σε κάθε πεδίο. Αν δεν υπάρξει καθορισμός χαρτογράφησης, τότε το Elasticsearch θα προσπαθήσει να μαντέψει το είδος το δεδομένων και θα τα χαρτογραφήσει αυτόματα, χωρίς αυτό να συνιστά αλάνθαστη διαδικασία. Στη χαρτογράφηση, πέρα από το είδος των δεδομένων σε κάθε πεδίο, δίνεται η δυνατότητα να προσδοθεί βαρύτητα (boost) σε ορισμένα πεδία, ώστε σ' αυτά να δίνεται προτεραιότητα κατά τη διάρκεια των αναζητήσεων.

3.3.12. Elasticsearch Search API

Μπορεί κάποιος χρησιμοποιώντας την γλώσσα ερωτημάτων του Elasticsearch (Query DSL, Domain Specific Language) και το Search API να διατυπώσει ερωτήματα στη βάση. Η Query DSL χρησιμοποιεί JSON για να ορίσει τα ερωτήματα που θα γίνουν στην βάση. Ομοιάζει με ένα αφαιρετικό συντακτικό δέντρο (AST Tree) αποτελούμενο από δύο τύπους: α) Τα ερωτήματα φύλλα, που ερευνούν για μια συγκεκριμένη τιμή το πεδίο που δόθηκε όπως ερωτήματα ταιριάσματος (match) και εύρους (range), και β) Και τα σύνθετα ερωτήματα, που συνδυάζουν πολλά ερωτήματα σε ένα εξάγοντας πιο πολύπλοκα αποτελέσματα. Το αποτέλεσμα που επιστρέφει είναι μια εγγραφή JSON που στο πρώτο τμήμα της περιέχει πληροφορίες για την επιτυχία/αποτυχία της διαδικασίας στα διάφορα shards του cluster και την ποσότητα των στοιχείων που εξετάστηκαν και στην συνέχεια τα αποτελέσματα σε μορφή πεδίου-κλειδιού. Το Search API είναι η διεπαφή που επιτρέπει την εκτέλεση των ερωτημάτων που έχουν γραφτεί στη γλώσσα Query DSL. Μπορεί να εφαρμοστεί σε πολλούς τύπους (types) του ίδιου καταλόγου (index) όπως επίσης και σε πολλούς καταλόγους (indices) κατά μήκος της συστάδας (cluster) [27].

### 3.3.13 Πλεονεκτήματα Elasticsearch

Συνοψίζοντας για το Elasticsearch, θα αναφερθούν μερικές από τις περιπτώσεις, όπου η χρήση του στην κατασκευή μηχανών αναζήτησης είναι ιδιαίτερη χρήσιμή. Τα παρακάτω παραδείγματα είναι χαρακτηριστικές περιπτώσεις των προβλημάτων που μπορεί να επιλύσει.

- Έχει την ικανότητα να επιστρέφει τα καλύτερα αποτελέσματα μέσα από έναν μεγάλο αριθμό περιγραφών προϊόντων, ιδιαίτερα όταν αναζητούνται φράσεις, όπως “chef knife” για παράδειγμα. Αυτό καθίσταται δυνατό με την χρήση των διαχωριστών λεκτικών μονάδων, που περιεγράφηκαν παραπάνω.
- Δοθέντος του προηγούμενου παραδείγματος, υπάρχει η δυνατότητα να αναζητηθούν σε ποια συγκεκριμένα υποκαταστήματα υπάρχει το συγκεκριμένο προϊόν και από ποιες ποσότητες και πάνω. Σ’ αυτό μπορεί να βοηθήσει ο καθορισμός συγκεκριμένων όψεων στα επιστρεφόμενα αποτελέσματα.
- Υπάρχει και η δυνατότητα αυτό συμπλήρωσης (auto completing) στο κουτί αναζήτησης σε μερικώς γραμμένες λέξεις βασιζόμενη σε προηγούμενες αναζητήσεις.

## 3.4. Model - View - Controller

### 3.4.1. Ορισμός

Το πρότυπο Μοντέλο – Άποψη – Ελεγκτής (MVC) [28] αποτελεί μια αρχιτεκτονική ανάπτυξης λογισμικού, κατά την οποία η παρουσίαση της πληροφορίας διαχωρίζεται από την αλληλεπίδραση του χρήστη μ’ αυτήν. Αν και αρχικά αυτή η αρχιτεκτονική κατασκευάστηκε για ανάπτυξη λογισμικού που αφορούσε τους προσωπικούς υπολογιστές, σήμερα έχει γίνει γνωστή σε ευρεία κλίμακα ως αρχιτεκτονική που υιοθετείται στην ανάπτυξη εφαρμογών στον παγκόσμιο ιστό (World Wide Web) και είναι συμβατή με όλες τις γνωστές γλώσσες προγραμματισμού. Υπάρχουν πολλές εφαρμογές διαδικτυακού πλαισίου λογισμικού, εμπορικές και μη που βασίζονται πάνω σ’ αυτήν την αρχιτεκτονική, όπως το CodeIgniter που προαναφέρθηκε, το Django, το Ruby on Rails κ.α.

### 3.4.2. Πλεονεκτήματα MVC προτύπου

Στη συνέχεια, πριν περιγράφει η δομή και τα χαρακτηριστικά του MVC προτύπου, είναι χρήσιμο να αποσαφηνιστούν τα προτερήματα της χρήσης του σε μια διαδικτυακή εφαρμογή. Οποιαδήποτε εφαρμογή, λοιπόν, έχει δομηθεί πάνω στο MVC πρότυπο, έχει κατανεμημένες τις βασικές της λειτουργικότητες σε τρεις τύπους αρχείων, δηλαδή τα μοντέλα, τις απόψεις και τους ελεγκτές. Αυτή η τεχνική επιτρέπει στο κάθε κομμάτι της να σχεδιάζεται, να διαχειρίζεται και να ελέγχεται ανεξάρτητα από τα υπόλοιπα. Επίσης, ενισχύεται μ’ αυτόν τον τρόπο η δυνατότητα να παραμένει ο κώδικας τακτοποιημένος, πράγμα το οποίο διευκολύνει την επεκτασιμότητα και την επαναχρησιμοποίηση του κώδικα, την εύρεση λαθών και διόρθωσή τους, αλλά και αυξάνει τις επιδόσεις της εφαρμογής σε ταχύτητα απόκρισης. Τέλος, ο διαχωρισμός αυτός σε πολλαπλά αρχεία ευνοεί τη συνεργασία πολλών ανθρώπων πάνω στην ίδια εφαρμογή, καθώς μπορεί ο καθένας να κατέχει διακριτούς ρόλους στην ανάπτυξη και τη συντήρησή της.

### 3.4.3. Ελεγκτής

Εφόσον έγινε κατανοητή η σημασία χρήσης αυτής της αρχιτεκτονικής, ακολουθεί η ανάλυση των επιμέρους συστατικών της [29]. Αρχικά, ο ελεγκτής μπορεί να χαρακτηριστεί ως ένας

διαμεσολαβητής μεταξύ του μοντέλου και της άποψης. Οι βασικές εργασίες που εκτελεί ο ελεγκτής είναι:

- Λαμβάνει δεδομένα από τον χρήστη μέσω των αρχείων που αποτελούν το κομμάτι της άποψης.
- Διατηρεί τα δεδομένα, όπως κάθε άλλος τύπος διαδικτυακής εφαρμογής. Έχοντας υπ' όψιν τη λειτουργικότητα της PHP, τότε μπορεί να ειπωθεί πως τα δεδομένα διατηρούνται σε $_GET και $_POST μεταβλητές.
- Ο ελεγκτής είναι υπεύθυνος να καθορίζει ποιες λειτουργίες χρειάζεται να λάβουν χώρα και γι' αυτό χρησιμοποιεί το κατάλληλο μοντέλο, ώστε να τις θέσει σε εφαρμογή.
- Εφόσον δημιουργήσει ένα αντικείμενο του μοντέλου, τότε θέτει σε εφαρμογή τη λειτουργία και διατηρεί ό, τι επιστραφεί από αυτήν.
- Τέλος, ο ελεγκτής φορτώνει την κατάλληλη άποψη, ανάλογα με τους σκοπούς του χρήστη και αποστέλλει τις πληροφορίες που δέχθηκε από το μοντέλο.

3.4.4. Μοντέλο

Το μοντέλο αναπαριστά τη δομή των δεδομένων που διαχειρίζεται η εφαρμογή [30]. Είναι υπεύθυνο να διατηρεί συναρτήσεις και μεταβλητές που εμπλέκονται με τα δεδομένά που αυτό αναπαριστά. Είναι πολύ βοηθητικό στην κατανόηση της λογικής του μοντέλου, αν θεωρηθεί ως ο πυρήνας ενός αντικειμένου στον αντικειμενοστραφή προγραμματισμό. Επομένως, το μοντέλο διεκπεραιώνει τις εξής λειτουργίες:

- Λαμβάνει οδηγίες διαθέσου του ελεγκτή και επεξεργάζεται δεδομένά που συνήθως λαμβάνει από κάποια βάση δεδομένών.
- Όταν οι αναγκαίες πληροφορίες επιστραφούν από τις συναρτήσεις ή τα δεδομένά τοποθετηθούν μέσα στις κατάλληλες μεταβλητές, τότε ο ελεγκτής τις χρησιμοποιεί ως αντικείμενα.

Άρα συνοπτικά το μοντέλο προσθέτει, ενημερώνει και ανακτά δεδομένά από τη βάση δεδομένών, ευρισκόμενο σε μια συνεχή επικοινωνία με τον ελεγκτή, κατά την οποία ανταλλάσσονται δεδομένά ανάλογα με τις απαιτήσεις του χρήστη.

3.4.5. Άποψη

Τέλος, τις πληροφορίες που έχει αιτηθεί ο ελεγκτής από το μοντέλο, τις αποστέλλει στην άποψη [31], ώστε να εμφανιστούν προς το χρήστη με συγκεκριμένο τρόπο. Στην ουσία η άποψη αποτελεί ένα σύστημα εμφάνισης της πληροφορίας που δομείται με συγκεκριμένο τρόπο, έτσι ώστε να μπορεί να εμφανιστεί είτε σε μια ιστοσελίδα είτε σε μια εφαρμογή κινητού τηλεφώνου. Η άποψη μπορεί να αποτελείται από μια ή περισσότερες ιστοσελίδες που να είναι γραμμένες με τη συνεργασία της γλώσσας χαρακτηρισμού υπερκειμένου (HTML), της γλώσσας κατασκευής ιστοσελίδων δυναμικού περιεχομένου (PHP) και της γλώσσας φύλλων στυλ (CSS). Σ' αυτές τις ιστοσελίδες, επίσης, μπορούν να συνδράμουν και όλες οι υπόλοιπες γλώσσες που συντελούν στην κατασκευή ιστοσελίδων, όπως παραδείγματος χάρη η JavaScript.

Πρέπει να σημειωθεί πως μια διαδικτυακή εφαρμογή αποτελείται από σύνολα ελεγκτών, μμοντέλων και απόψεων, δηλαδή από διάφορα αρχεία που επιτελούν τις λειτουργίες που αναλύθηκαν. Ένας όμως ελεγκτής παίζει το ρόλο του βασικού ελεγκτή, ο οποίος δέχεται όλες τις αιτήσεις και καλεί τους συγκεκριμένους ελεγκτές που χειρίζονται συγκεκριμένες ενέργειες σε κάθε περίπτωση.

3.5. Γλώσσα Προγραμματισμού Python

3.5.1. Γενικά για την Python

Η Python είναι μια διερμηνευόμενη (interpreted) γλώσσα προγραμματισμού υψηλού επιπέδου [32], που σχεδιάστηκε με έμφαση στην εύκολη ανάγνωση του κώδικά της. Η ανάπτυξή της ξεκίνησε το Δεκέμβριο του 1989 από τον Guido van Rossum στο Εθνικό Ινστιτούτο Έρευνας Πληροφορικής και Μαθηματικών της Ολλανδίας (CWI). Ο δημιουργός της εμπνεύστηκε το όνομα της από τους Monty Pythons, το γνωστό γκρουπ κωδικών από τη Μεγάλη Βρετανία.

Υποστηρίζει πολλά είδη προγραμματισμού, όπως αντικειμενοστραφή, διαδικαστικό, συναρτησιακό και άλλα, και παρέχει ένα πλήρως δυναμικό σύστημα τύπων και αυτόματη διαχείριση μνήμης. Είναι μια γλώσσα προγραμματισμού γενικής χρήσης, ιδανική για κατασκευή εφαρμογών όλων των ειδών

ακόμα και δυναμικών ιστοσελίδων. Η βασική της βιβλιοθήκη είναι αρκετά μεγάλη και διαθέτει εκτενή τεκμηρίωση.

### 3.5.2 Χαρακτηριστικά της Python

Μερικά από τα βασικά της χαρακτηριστικά που την κάνουν ιδιαίτερα ελκυστική και εύκολη στη χρήση είναι τα παρακάτω.

- Η Python είναι απλή και μινιμαλιστική γλώσσα σε τέτοιο σημείο που ο κώδικας της μοιάζει με ψευδοκώδικα. Ένα ιδιαίτερο χαρακτηριστικό της γλώσσας είναι η χρήση κενών διαστημάτων (whitespace) για το διαχωρισμό των συντακτικών δομών του προγράμματος, σε αντίθεση με την πρακτική άλλων γλωσσών, όπου για τον ίδιο σκοπό χρησιμοποιούνται ειδικά σύμβολα (πχ αγκύλες). Αυτό, σε συνδυασμό με το ότι χρησιμοποιεί πλήρεις αγγλικές λέξεις στη θέση συμβόλων, καθιστούν τον κώδικα της Python ευανάγνωστο.
- Είναι γλώσσα υψηλού επιπέδου και όπως όλες οι γλώσσες αυτού του είδους αποκρύπτουν όλες τις διεργασίες που λαμβάνουν χώρα σε χαμηλό επίπεδο, όπως παραδείγματος χάρη τη διαχείριση της μνήμης.
- Είναι γλώσσα ελεύθερου και ανοιχτού κώδικα, πράγμα που σημαίνει ότι ο πηγαίος κώδικάς της μπορεί να διαχειρίζεται και να επεκτείνεται ελεύθερα και πλαισιώνεται από μια κοινότητα προγραμματιστών που αφενός διασπείρει την αποκτημένη γνώση γύρω από θέματα που την αφορούν και αφετέρου τη βελτιώνει και την ισχυροποιεί συνεχώς.
- Είναι γλώσσα ερμηνευόμενη που σημαίνει πως δεν χρειάζεται μεταγλώττιση σε δυαδικό κώδικα. Απλά το εκάστοτε πρόγραμμα εκτελείται άμεσα από τον πηγαίο κώδικα. Εσωτερικά, η Python μετατρέπει τον πηγαίο κώδικα σε μια ενδιάμεση μορφή, η οποία ονομάζεται bytecode, έπειτα μεταφράζει το bytecode στην εγγενή γλώσσα του συστήματος και τελικά τρέχει τον κώδικα. Όλη αυτή η διαδικασία διευκολύνει τη χρήση της Python, αφού δεν υπάρχει η ανάγκη μεταγλώττισής ή φόρτωσης συγκεκριμένων βιβλιοθηκών.
- Η Python υποστηρίζει τόσο διαδιακασιοστραφή όσο και αντικειμενοστραφή προγραμματισμό. Στις διαδικασιοστραφείς γλώσσες, το πρόγραμμα χτίζεται πάνω σε διαδικασίες ή συναρτήσεις, οι οποίες δεν είναι τίποτε άλλο παρά επαναχρησιμοποιήσιμά κομμάτια κώδικα. Στις αντικειμενοστραφείς γλώσσες το πρόγραμμα χτίζεται πάνω σε αντικείμενα, τα οποία συνδυάζουν δεδομένά και λειτουργικότητα. Η Python έχει έναν πολύ ισχυρό, αλλά απλό τρόπο υποστήριξης του αντικειμενοστραφούς προγραμματισμού, ειδικά όταν συγκρίνεται με γλώσσες, όπως η C++ ή η Java.
- Είναι επεκτάσιμή και έτσι υπάρχει η δυνατότητα ενσωμάτωσης της Python μέσα σε προγράμματα γραμμένα σε C/C++, για να τους επιτρέπεται η εκτέλεση σεναρίων εντολών με πιο εύκολο τρόπο.
- Τέλος, εγγενώς η Python διαθέτει μια τεράστια συλλογή από βιβλιοθήκες, που προσφέρουν μεγάλη βοήθεια, ώστε η συγγραφή του κώδικα να μην ξεκινάει από την αρχή χωρίς κανένα υπόβαθρο. Αυτές βιβλιοθήκες αφορούν κανονικές εκφράσεις, ενσωμάτωση βάσεων δεδομένών, νημάτωση (threading), γραφικά περιβάλλοντα δημιουργίας διεπαφών (GUI) κα.

## 3.6. Docker

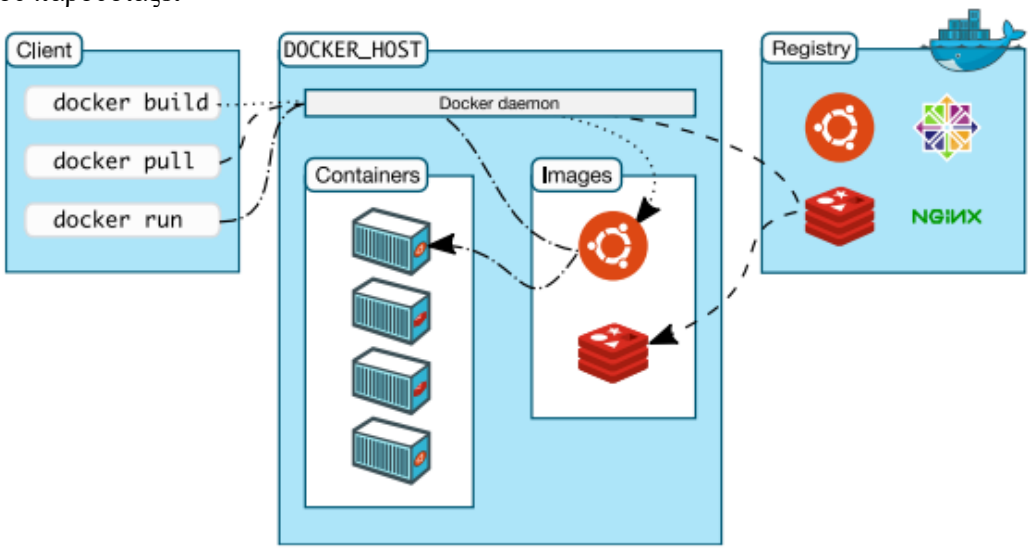
### 3.6.1. Γενικά για το Docker

Ο πυρήνας του Linux παρέχει ένα μηχανισμό για containerization μέσω του οποίου ο χρήστης μπορεί να δημιουργήσει πολλούς containers με δομή όπως των VMs σε ένα επίπεδο πάνω από το λειτουργικό σύστημα Linux, γνωστό ως Linux Based Containers (LXC). Το LXD είναι ουσιαστικά μία ανανεωμένη έκδοση του LXC, προσφέροντας περισσότερη ευελιξία και χαρακτηριστικά. Ουσιαστικά, χρησιμοποιεί τη βιβλιοθήκη LXC για την παροχή containers σε επίπεδο λειτουργικού με μικρή επιβάρυνση. Το Docker, έχει γραφτεί στη γλώσσα προγραμματισμού GoLang, και έχει γίνει συνώνυμο με την τεχνολογία των containers διότι ήταν το πιο επιτυχημένο εργαλείο για τη διάδοσή τους. Ωστόσο, δεν είναι νέα η τεχνολογία των containers, καθώς έχει ενσωματωθεί στο Linux στη μορφή των LXC για πάνω από 10 χρόνια, με τους τελευταίους να είναι τυπικό εργαλείο για ανάπτυξη ροών εργασίας και cloud ανάπτυξης. Το 2015 η εταιρία CoreOS δημιούργησε τη δική της περιγραφή App Container Image που ήταν διαφορετική από την περιγραφή των Docker Containers. Αργότερα, κατά τη διάρκεια του ίδιου έτους ανήγγειλε μια πρωτοβουλία που ονομάστηκε Open Container Project και μετέπειτα Open Container Initiative. Ο σκοπός της ήταν η ανάπτυξη βιομηχανικών προτύπων για μία container φόρμα και ενός container λογισμικού για όλες τις πλατφόρμες. Το σημείο

εκκίνησης των προτύπων OCP ήταν η τεχνολογία Docker. Χορηγοί του έργου περιλαμβάνουν τις AWS, Google, IBM, Docker, CoreOS κ.ά [33] . Η ιδέα του OCI είναι να εξασφαλιστεί ότι τα θεμελιώδη δομικά στοιχεία της τεχνολογίας των containers θα είναι τυποποιημένα ώστε όλοι να μπορούν να επωφεληθούν από αυτά. Αυτό σημαίνει ότι, αντί να δαπανώνται πόροι αναπτύσσοντας ανταγωνιστικές τεχνολογίες, οι εταιρείες μπορούν να επικεντρωθούν στην ανάπτυξη πρόσθετου λογισμικού που απαιτείται για τη στήριξη της χρήσης τυποποιημένων containers σε περιβάλλον επιχειρήσεων ή cloud. Το είδος του απαιτούμενου λογισμικού περιλαμβάνει συστήματα ενορχήστρωσης και διαχείρισης και ασφάλειας των containers. Το πιο γνωστό δωρεάν και ανοιχτού κώδικα σύστημα διαχείρισης των containers είναι το Kubernetes, το οποίο είναι ένα έργο λογισμικού που προέρχεται από τη Google. Το Kubernetes παρέχει μηχανισμούς για ανάπτυξη, συντήρηση και κλιμάκωση εφαρμογών σε containers. Οι Linux Containers και ιδιαίτερα το Docker αποτελούν ένα ουσιαστικό μονοπάτι για την εικονοποίηση σε ενσωματωμένες συσκευές, καθώς μπορούν να φέρουν εξελιγμένες δυνατότητες ανάπτυξης λογισμικού στις συσκευές αυτές. Τα VMs έχουν βοηθήσει σημαντικά στην αυτοματοποίηση και παραγωγικότητα πάνω στην ανάπτυξη στο cloud τα τελευταία χρόνια, αλλά λόγω των αναπόφευκτων μειονεκτημάτων τους δεν ενδείκνυνται για χρήση σε ενσωματωμένα συστήματα. Με την εικονοποίηση, όμως, στο επίπεδο του λειτουργικού συστήματος που παρέχει το Docker (και οι Linux Containers) δεν εμφανίζονται αντίστοιχα μειονεκτήματα και έτσι είναι δυνατή η χρήση τους για συνδεδεμένες συσκευές στο IoT. Στο γίνεται μία σύγκριση του Docker και του KVM (ένα feature του Linux που του δίνει τη δυνατότητα να φερθεί σαν Hypervisor και έτσι να τρέξει πολλαπλές εικονικές μηχανές πάνω από αυτό) σε διαφορετικές μελέτες περίπτωσης σε θέματα CPU, I/O και απόδοση μνήμης, και φαίνεται ότι το Docker είναι παρόμοιο ή και καλύτερο από το KVM σε απόδοση. Ένας Docker container έχει δικό του μοναδικό Namespace (PID, UID, filesystems, network) και έτσι οι διεργασίες που τρέχουν μέσα σε αυτό μπορούν να απομονωθούν πλήρως από το λειτουργικό σύστημα πάνω στο οποίο τρέχει, και να μοιραστούν τους πόρους που τους έχουν γίνει διαθέσιμοι, χωρίς να έχουν επαφή με τους διαθέσιμους πόρους άλλων containers. Ουσιαστικά, το Docker είναι ένα λογισμικό ή εργαλείο που προσφέρει containerization [34].

### 3.6.2. Αρχιτεκτονική του Docker

Το παρακάτω σχήμα παρουσιάζει την αρχιτεκτονική του Docker [35] καθώς και τα βασικά στοιχεία που παρουσιάζει




#### 3.6.2.1. Docker Engine

Γενικά, ένας container engine είναι ένα πρόγραμμα που ελέγχει τους containers και είναι υπεύθυνο για να διανέμει όλους τους απαραίτητους πόρους από το σύστημα στους containers που εκτελούν κάποια διεργασία μέσω του kernel. Συγκεκριμένα, εκτελείται σαν διεργασία αυτόνομη χωρίς να βρίσκεται υπό τον άμεσο έλεγχο ενός δια δραστικού χρήστη, στο βασικό λειτουργικό σύστημα, γνωστή και ως ‘daemon. Ο Docker Engine παρέχει πρόσβαση στους Docker containers αντίστοιχη, ώστε οι τελευταίοι να έχουν όλους τους απαραίτητους πόρους για να λειτουργούν ορθά και ομαλά. Αποτελείται από έναν Docker ‘daemon που εκτελείται αυτόνομα, και δημιουργεί, διαχειρίζεται και καταστρέφει τους containers.

3.6.2.2. Docker Client

Το Docker client είναι ο κύριος τρόπος για να αλληλοεπιδράσουν με το docker. Όταν χρησιμοποιούμε εντολές όπως η docker run, το client στέλνει αυτές τις εντολές στο daemon, το οποίο τις εκτελεί. Οι εντολές που ξεκινούν με “docker” χρησιμοποιούν το docker API. Ουσιαστικά το Docker Client (CLI) είναι ένα σύνολο από ποικίλες command line διαθέσιμες εντολές για τη διαχείριση των Docker containers. Ο χρήστης μέσω του Docker Client στέλνει εντολές στον Docker daemon μέσω ενός REST API για να διαχειριστεί τους containers.

3.6.2.3. Docker Image

Ένα Docker Image [36] είναι μία μονάδα η οποία μπορεί να αποτελείται από τα πολύ βασικά binaries και libraries, μέχρι μια πολύπλοκη σύνθεση εντολών για μία εκτελέσιμη εφαρμογή. Ουσιαστικά, αποτελείται από εντολές, κάθε μία από τις οποίες προσθέτει ένα επιπλέον επίπεδο (layer) στο image αυτό. Τα layers στα περισσότερα Docker images ξεκινάνε με το λεγόμενο base image, ένα επίσημο image το οποίο παρέχεται από το Docker στο cloud-based αποθετήριο ανοικτού κώδικα, Docker Hub, στο οποίο μπορούν να δημοσιευτούν container images από μικρές κοινότητες ή αυτόνομοι developers που επιθυμούν να προσφέρουν μέχρι και πολύ μεγάλες ομάδες λογισμικού. Ένα Docker image μπορεί να δημιουργηθεί μέσω εντολών, ή μέσω του Dockerfile. Το Dockerfile είναι ένα προσαρμοσμένο αρχείο που αποτελείται από διάφορες εντολές και παραμέτρους, των οποίων η σωστή αλληλουχία θα οδηγήσει σε ένα νέο image, το οποίο θα δημιουργήσει ο Docker Engine. Για παράδειγμα, κάποιες από τις εντολές αυτές μπορεί να είναι το base image, ποια αρχεία θα πρέπει να αντιγραφούν μέσα στον container, ποια p θα πρέπει να φαίνονται εξωτερικά από τον container, ποια dependencies και σε ποιες εκδόσεις θα πρέπει να προστεθούν και να γίνουν λειτουργικά στον container κ.ά. . Μία Docker εικόνα (Docker image) είναι ένα read only πρότυπο που περιέχει οδηγίες για την δημιουργία ενός Docker container. Συνήθως, κάθε εικόνα βασίζεται σε κάποια άλλη εικόνα με μερικές επιπρόσθετες αλλαγές. Για παράδειγμα, θα μπορούσαμε να βασιστούμε σε μια εικόνα των Ubuntu για να δημιουργήσουμε μια νέα εικόνα η οποία θα εγκαθιστά το PostgreSQL καθώς και θα δημιουργεί τους πίνακες που θέλουμε σε αυτό. Μπορούμε είτε να δημιουργήσουμε δικές μας εικόνες (δημιουργώντας τες με την χρήση ενός Dockerfile) είτε να χρησιμοποιήσουμε εικόνες που έχουν δημιουργήσει άλλοι χρήστες και έχουν ανεβάσει σε κάποιο registry.

3.6.2.4. Docker Container

Τα Docker images γίνονται containers κατά το χρόνο εκτέλεσης τους στο Docker Engine. Συγκεκριμένα, αποτελούνται από τον εκτελέσιμο κώδικα της εφαρμογής και διάφορες άλλες εξαρτήσεις (βιβλιοθήκες και άλλα αρχεία για να λειτουργεί η συγκεκριμένη εφαρμογή), και είναι instances των Docker images από τα οποία δημιουργούνται. Κατά τη διαδικασία δημιουργίας ενός Docker container, αντιγράφονται όλα τα layers που είναι μόνο για ανάγνωση (image layers), και προστίθεται ένα νέο layer πάνω από αυτά το οποίο είναι δυνατό για ανάγνωση και εγγραφή (container layer). Docker Registry Το αποθετήριο Docker συμβάλλει στην αποθήκευση και κατανομή των Docker images. Μπορεί να είναι είτε δημόσιο, δηλαδή το Docker Hub που [37]αναφέρθηκε προηγουμένως, είτε ιδιωτικό, καθώς πολλές εταιρίες μπορεί να έχουν το δικό τους αποθετήριο για τα δικά τους Docker images. Μπορούμε όμως να ελέγξουμε πόσο απομονωμένο θα είναι το container. Ένα container ορίζεται από την docker εικόνα του καθώς και ότι ρυθμίσεις του παρέχουμε κατά την δημιουργία του. Μπορούμε να δημιουργήσουμε ένα container από μια εικόνα με την εντολή “docker run [OPTIONS] IMAGE-NAME [COMMAND]” π.χ. “docker run -i -t ubuntu /bin/bash”.

3.6.2.5. Docker Registry

Ένα Docker Registry περιέχει docker εικόνες. Ένα registry μπορεί να είτε ιδιωτικό είτε δημόσιο. Όταν χρησιμοποιούμε την εντολή docker run ή docker pull, το docker θα τραβήξει τις απαιτούμενες εικόνες από το προεπιλεγμένο registry. Με την εντολή docker push μπορούμε να ανεβάσουμε μια εικόνα στο registry. Το προεπιλεγμένο απομακρυσμένο registry είναι το docker hub (https://hub.docker.com/) το οποίο είναι ένα δημόσιο registry που διαχειρίζεται η Docker Inc. και το οποίο μπορεί να χρησιμοποιήσει ο οποιοσδήποτε. Στο docker hub μπορούμε να βρούμε εικόνες για πολλές δημοφιλείς εφαρμογές [38].

3.6.3. Επιπλέον Docker Λογισμικά

3.6.3.1. Dockerfile

Ένα Dockerfile είναι ένα αρχείο κειμένου που περιέχει οδηγίες για την δημιουργία μιας docker εικόνας. Το Dockerfile περιέχει όλες τις εντολές που θα τρέχαμε στην γραμμή εντολών προκειμένου

να σχηματίσουμε την εικόνα. Με την εντολή docker build δημιουργούμε μια εικόνα από ένα Dockerfile.

3.6.3.2. Docker Service

Μια υπηρεσία (service) μας επιτρέπει να ορίσουμε το πως θέλουμε να τρέξουμε τα container των εφαρμογών μας σε ένα σμήνος (swarm). Μια υπηρεσία μας επιτρέπει να ορίσουμε την επιθυμητή κατάσταση, όπως το πόσα αντίγραφα (replicas) της εφαρμογής θέλουμε να τρέξουμε συγχρόνως. Από προεπιλογή γίνεται εξισορρόπηση φορτίου (load balancing) ανάμεσα σε όλους τους κόμβους-εργάτες (worker nodes) του σμήνους. Στο χρήστη η υπηρεσία φαίνεται σαν να είναι μία μόνο εφαρμογή. Συχνά μια υπηρεσία είναι μια micro service στο πλαίσιο μιας μεγαλύτερης εφαρμογής. Παραδείγματα υπηρεσιών ίσως περιλαμβάνουν έναν HTTP server, μια βάση δεδομένων ή οποιοδήποτε άλλο είδος εκτελέσιμου προγράμματος θα επιθυμούσαμε να τρέξουμε σε ένα κατανεμημένο περιβάλλον [39].

3.6.3.3. Docker Compose

Σε περιπτώσεις που μια σύνθετη Docker εφαρμογή περιλαμβάνει αρκετούς containers ή χρειάζεται για κάποιο λόγο τμήματα αυτής να διασπαστούν σε περισσότερους από έναν, το Docker compose διευκολύνει σημαντικά τη δημιουργία, εκτέλεση και σύνδεση των επιμέρους containers, καθένα από τα οποία θα έχει το δικό του Dockerfile. Πιο αναλυτικά, η δυνατότητα αυτή δίνεται χρησιμοποιώντας ένα YAML αρχείο στο οποίο ορίζονται όλες οι εφαρμογές σαν υπηρεσίες (services), και ορίζεται ο τρόπος εκτέλεσης και σύνδεσης τους. Εν κατακλείδι, ο χρήστης θα έχει τη δυνατότητα μέσω μιας εντολής να κάνει build, και run όλων των containers μαζικά, εξοικονομώντας έτσι σημαντικό χρόνο [40].

3.6.3.4. Docker Swarm

Το Docker διαθέτει δική της εγγενής λύση για clustering εφαρμογές, το Docker Swarm, το οποίο Docker μία πλατφόρμα ενορχήστρωσης (orchestration) των containers ανοικτού κώδικα, και χρησιμοποιεί το δικό του API. Ουσιαστικά, κάθε εφαρμογή ή υπηρεσία που μπορεί να εκτελεστεί μέσω Docker containers μπορεί να εκτελεστεί με παρόμοιο τρόπο μέσω Docker Swarm , μετατρέποντας έτσι ένα σύνολο από πραγματικούς servers ή hosts σε ένα εικονικό host. Βασικό χαρακτηριστικό του είναι ότι δημιουργείται το λεγόμενο ‘swarm (σμήνος) που περιέχει δύο τύπους ρόλων, τους workers και τους managers. Ένας manager μπορεί να τερματίσει, να ξεκινήσει, να ενημερώσει και να διαχειριστεί τους containers, ενώ ο worker μπορεί μόνο να χρησιμοποιηθεί για τη φιλοξενία των containers κατά την εκτέλεσή τους. Αντίστοιχο εργαλείο που έχει δυνατότητες ενορχήστρωσης των containers είναι το Kubernetes [41].

3.6.4. Βασικές Εντολές Docker

Από τους developers που συνεισφέρουν στην κοινότητα του Docker έχει δημιουργηθεί μία ποικιλία εντολών μέσω των οποίων είναι εύχρηστη και εύκολη η δημιουργία, διαχείριση, και καταστροφή των Docker containers. Κάποιες από τις βασικές εντολές που χρησιμοποιούνται συχνά είναι:

- **docker build**: Η δημιουργία ενός Docker image από ένα Dockerfile.
- **docker run**: Η εκτέλεση ενός Docker image, δημιουργώντας έναν Docker container μέσα στον οποίο θα εκτελεστεί απομονωμένα κάποια εφαρμογή η διεργασία.
- **docker-compose build /up**: Οι διαδικασίες δημιουργίας και εκτέλεσης πολλών Docker images στο ίδιο node, με τα χαρακτηριστικά τους να βρίσκονται συγκεντρωμένα σε ένα YAML αρχείο.
- **docker stats**: Η εντολή αυτή δείχνει σε πραγματικό χρόνο την κατανάλωση μνήμης και CPU από κάθε Docker container ξεχωριστά.
- **docker history**: Με αυτόν τον τρόπο εμφανίζονται όλα τα ξεχωριστά layers που αποτελούν ένα image.
- **docker ps**: Η εντολή αυτή χρησιμοποιείται για την εμφάνιση της κατάστασης όλων των Docker containers που εκτελούνται εκείνη τη στιγμή, μαζί με τα IDs τους.
- **docker container ls -a**: Μέσω της εντολής αυτής παρουσιάζεται μία λίστα με όλους τους containers, είτε εκτελούνται εκείνη τη στιγμή είτε όχι.

3.6.5. Πλεονεκτήματα Docker

Τα containers υπερτερούν της παραδοσιακής εικονοποίησης σε πολλούς τομείς και ειδικά στον τομέα της απόδοσης και της επεκτασιμότητας. Το Docker αποτελεί καλή επιλογή για τα περισσότερα συστήματα υπολογιστικού νέφους, τα οποία χρειάζονται μια εύκολη επιλογή επεκτασιμότητας. Αντί να χρησιμοποιούνται πολλαπλές εικονικές μηχανές, οι οποίες σπαταλούν πολύτιμους πόρους, το

Docker μπορεί να ξεκινήσει πολύ γρήγορα πολλά περισσότερα container και με πολύ μικρότερη επιβάρυνση πόρων. Ένα πρόβλημα που μπορεί να επιλύσει το Docker είναι η λεγόμενη «κόλαση εξαρτήσεων» (dependency hello) δηλαδή όταν δύο διαφορετικές εφαρμογές έχουν ως εξάρτηση διαφορετικές εκδόσεις του ίδιου πακέτου λογισμικού ή βιβλιοθήκης, οι οποίες είναι ασύμβατες μεταξύ τους και πιθανόν να μην μπορούν να εγκατασταθούν παράλληλα. Στο Docker κάθε container περιέχει όλες τις εξαρτήσεις της εφαρμογής και επειδή τα container είναι απομονωμένα το ένα από το άλλο, δεν υπάρχει πρόβλημα αν σε διαφορετικά container έχουν εγκατασταθεί διαφορετικές εκδόσεις της ίδιας βιβλιοθήκης. Μερικά από τα πιο βασικά πλεονεκτήματα του Docker είναι τα ακόλουθα [42]:

- **Φορητές Υλοποιήσεις**: Καθώς τα container είναι φορητά, οι εφαρμογές μπορούν να ενσωματωθούν σε μια ενιαία μονάδα και μπορούν να αναπτυχθούν σε διάφορα περιβάλλοντα χωρίς να χρειαστεί να γίνουν αλλαγές στο container. Τα container μπορούν να τρέχουν σε οποιοδήποτε σύστημα Linux, Windows ή Mac, καθώς και σε συστήματα υπολογιστικού νέφους, επιτραπέζιους υπολογιστές, φυσικούς διακομιστές και ούτω καθεξής. Μπορούμε να μεταφέρετε container από το περιβάλλον της επιφάνειας εργασίας μας στο νέφος και πάλι πίσω εύκολα και γρήγορα.
- **Ταχεία παράδοση**: Η ροή εργασίας των container καθιστά εύκολη τη συνεργασία ανάμεσα σε προγραμματιστές, διαχειριστές συστημάτων και ομάδων QA. Λόγω της τυπικής μορφής του container, μόνο οι προγραμματιστές χρειάζονται να ανησυχούν για τις εφαρμογές που τρέχουν μέσα σε αυτό ενώ οι διαχειριστές του συστήματος χρειάζονται μόνο να ανησυχούν για την τοποθέτηση του container στους διακομιστές. Αυτός ο καλά διαχωρισμένος τρόπος διαχείρισης των container οδηγεί σε γρηγορότερη ανάπτυξη και παράδοση της εφαρμογής. Επιπλέον η δημιουργία νέων container είναι πολύ γρήγορη επειδή τα container είναι πολύ ελαφριά, Αυτό με τη σειρά του μειώνει τον χρόνο για τη δοκιμή και την ανάπτυξη της εφαρμογής.
- **Κλιμάκωση**: Η κλιμάκωση των container προς τα πάνω και κάτω είναι απίστευτα γρήγορη. Μπορούμε εύκολα και γρήγορα να κλιμακώσουμε τα container από ένα σε μερικές εκατοντάδες και να τα ξανά μειώσουμε αργότερα όταν πια δεν τα χρειαζόμαστε. Έτσι τα container είναι ιδανικά για εφαρμογές έχουν σχεδιαστεί και κατασκευαστεί για πλατφόρμες υπολογιστικού νέφους.
- **Καλύτερη απόδοση**: Μπορούμε να έχουμε πολλά περισσότερα container σε σύγκριση με τις εικονικές μηχανές. Αφού τα container δεν χρειάζονται να χρησιμοποιήσουν κάποιο hypervisor, μπορούν να αξιοποιήσουν καλύτερα τους διαθέσιμους πόρους. Επειδή τα container δεν χρησιμοποιούν ένα πλήρες λειτουργικό σύστημα, οι απαιτήσεις σε πόρους είναι μικρότερες σε σύγκριση με τις εικονικές μηχανές.

# 4. Εγκατάσταση Βασικών Εργαλείων Διπλωματικής

## 4.1. Εισαγωγή

Σε αυτό το κεφάλαιο παρουσιάζονται εκτενώς οι τρόποι και οι τεχνικές με τις οποίες εγκαθίστανται τα εργαλεία που χρησιμοποιούνται στην παρούσα διπλωματική. Πιο συγκεκριμένα κάθε υπό κεφάλαιο ασχολείται με ένα εργαλείο και τα βήματα που απαιτούνται για την εγκατάσταση αυτού του εργαλείου στα λογισμικά Ubuntu και Windows 10.

## 4.2. Εγκατάσταση Docker και docker-compose

### 4.2.1. Εγκατάσταση Docker σε Windows

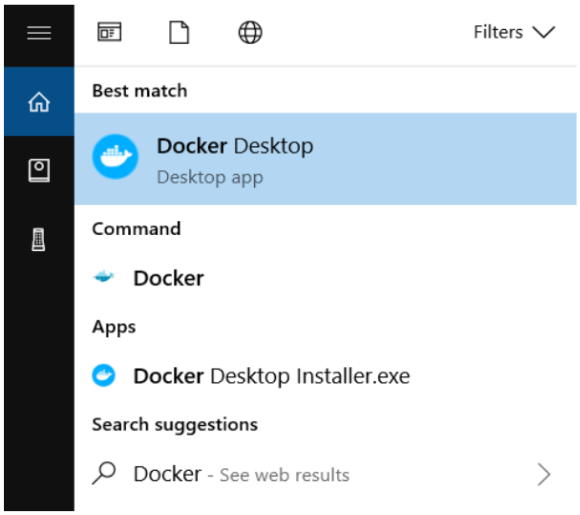
Για την εγκατάσταση του Docker στο λειτουργικό σύστημα Windows θα χρησιμοποιηθεί η έκδοση Docker Desktop η οποία αποτελεί την Community Edition του Docker για το λειτουργικό σύστημα Windows. Το πρώτο βήμα είναι η λήψη του Docker Desktop for Windows από αυτόν τον σύνδεσμο Download from Docker Hub. Με την λήψη του λογισμικού αυτού συμφωνείτε στους όρους χρήσης του Docker [43]. Πριν την εγκατάσταση ο χρήστης του συστήματος πρέπει να γνωρίζει τα παρακάτω:

System Requirements

- Windows 10 64-bit: Pro, Enterprise, or Education (Build 15063 or later)
- Πρέπει να είναι ενεργοποιημένα τα Hyper-V Windows features
- Το μηχάνημα στο οποίο θα εκτελείται ο Docker πρέπει να έχει τα ακόλουθα χαρακτηριστικά:
    - 64 bit processor with Second Level Address Translation (SLAT)
    - 4GB system RAM
    - BIOS-level hardware virtualization support must be enabled in the BIOS settings
- Docker Windows Installer
    - Ο installer για την έκδοση Docker Desktop περιέχει εκτελέσιμα για Docker Engine, Docker CLI client, Docker Compose, Docker Machine, Kinematic
    - Οι Containers και Images που δημιουργούνται χρησιμοποιώντας το Docker Desktop μοιράζονται μεταξύ όλων των χρηστών στα μηχανήματα που εγκαθίστανται, αυτό οφείλεται στο γεγονός ότι όλοι οι λογαριασμοί των Windows χρησιμοποιούν το ίδιο VM για να χτίσουν και να τρέξουν τους docker containers.

Τα βήματα που απαιτούνται για την εκτέλεση του Docker Desktop στα Windows είναι τα ακόλουθα:

1. Double-click Docker Desktop Installer.exe
2. Ακολουθήστε τις οδηγίες του Installation Wizard και στην συνέχεια αποδεχτείτε τις άδειες χρήσης, εξουσιοδοτείται τον Docker Installer.
3. Πατήστε Finish για την ολοκλήρωση της εγκατάστασης.




Αναζήτηση Docker Desktop

Αφού ολοκληρωθούν τα παραπάνω βήματα πρέπει να γίνει εκκίνηση του Docker Desktop. Για την εκκίνηση, ψάξτε μέσω της αναζήτησης των Windows το λεκτικό **Docker Desktop** και επιλέξτε. Αφού επιλεχθεί το whale icon του Docker στο status bar των Windows παραμείνει σταθερό τότε ο Docker Desktop είναι έτοιμο για χρήση και προσβάσιμο από οποιαδήποτε terminal του συστήματος

Docker Whale Icon

Μετά την επιτυχημένη εγκατάσταση του Docker για Windows εμφανίζεται ένα success pop-up message με προτεινόμενα βήματα και σύνδεσμο για το Docker documentation.

4.2.2. Εγκατάσταση Docker σε Ubuntu

Για την εγκατάσταση του Docker στο λογισμικό Ubuntu ακολουθήθηκαν τα παρακάτω βήματα [44].

- Ανανέωση της λίστας των πακέτων του συστήματος

```
$ sudo apt update
```

- Εγκατάσταση των απαραίτητων πακέτων που βοηθούν τον Ubuntu Package Manager να λειτουργήσει πάνω από το HTTPS πρωτόκολλο.

```
$ sudo apt install apt-transport-https ca-certificates curl
software-properties-common
```

- Add GCP key for the official Docker Installation to your machine

```
$ curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo
apt-key add -
```

- Πρόσθεση των Docker repositories στα APT sources του Ubuntu συστήματος

```
$ sudo add-apt-repository "deb [arch=amd64]
https://download.docker.com/linux/ubuntu bionic stable"
```

- Στην συνέχεια πρέπει να γίνει πάλι ανανέωση των πακέτων του συστήματος, έπειτα από την πρόσθεση των docker repositories

```
$ sudo apt update
```

- Για την τελική εγκατάσταση του Docker επιπλέον χρειάζονται οι παρακάτω εντολές

```
$ apt-cache policy docker-ce
        $ sudo apt install docker-ce
        $ sudo systemctl status docker
```

Η εγκατάσταση είναι επιτυχημένη εάν η έξοδος του συστήματος είναι η ακόλουθη:

```
Output
● docker.service - Docker Application Container Engine
   Loaded: loaded (/lib/systemd/system/docker.service; enabled; vendor preset: enabled)
   Active: active (running) since Thu 2018-07-05 15:08:39 UTC; 2min 55s ago
     Docs: https://docs.docker.com
 Main PID: 10096 (dockerd)
    Tasks: 16
   CGroup: /system.slice/docker.service
           ├─10096 /usr/bin/dockerd -H fd://
           └─10113 docker-containerd --config /var/run/docker/containerd/containerd.toml
```

4.3. Εγκατάσταση της Γλώσσας Προγραμματισμού Python

Σε αυτό το σημείο παρουσιάζονται τα βήματα που απαιτούνται για την εγκατάσταση της γλώσσας προγραμματισμού Python στα λειτουργικά συστήματα Windows και Ubuntu.

4.3.1. Εγκατάσταση Python σε Windows

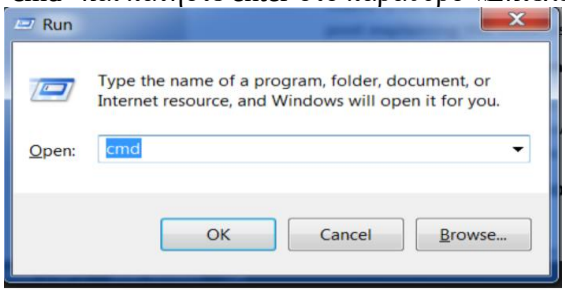
Για την εγκατάσταση της Python πρέπει να ελεγχθεί αρχικά αν το λειτουργικό σύστημα Windows είναι 32-bit ή 64-bit. Επιλέξτε "System type" από τη σελίδα System Info. Για να φτάσετε σε αυτή τη σελίδα με μια από τις ακόλουθες μεθόδους [45]:

- Πιέστε το κουμπί των Windows (δίπλα από το αριστερό Alt) και το Pause/Break ταυτόχρονα.

- Ανοίξτε τον πίνακα ελέγχου (Control Panel) από το Windows μενού και περιηγηθείτε στο System & Security, έπειτα επιλέξτε System
- Πιέστε το πλήκτρο των Windows και έπειτα περιηγηθείτε στο Settings > System > About

Σε επόμενο στάδιο πρέπει να γίνει λήψη των εκτελέσιμων αρχείων της Python, τα αρχεία αυτά βρίσκονται στο επίσημο Website της Python . Κάντε click στο σύνδεσμο "Latest Python 3 Release - Python x.x.x". Αν ο υπολογιστής σας τρέχει **64-bit** Windows, τότε κατεβάστε το αρχείο **Windows x86-64 executable installer**. Ειδάλλως, κατεβάστε το αρχείο **Windows x86 executable installer**. Αφού κατεβάσετε το αρχείο εγκατάστασης, κάντε διπλό κλικ πάνω του και ακολουθήστε τις οδηγίες. Κατά τη διάρκεια της εγκατάστασης, θα παρατηρήσετε ένα παράθυρο με το όνομα "Setup". Σιγουρευτείτε ότι είναι επιλεγμένο το κουτάκι "Add Python 3.6 to PATH" ή Add Python to your environment variables" και έπειτα κάνετε click στην επιλογή "Install Now", όπως φαίνεται στην παρακάτω εικόνα (ίσως να φαίνεται διαφορετικά σε εσάς αν εγκαθιστάτε διαφορετική έκδοση):


Όταν ολοκληρωθεί η εγκατάσταση, ενδέχεται να εμφανιστεί ένα παράθυρο διαλόγου με μια σύνδεση που είναι παρέχει επιπλέον πληροφορία σχετικά με την Python ή σχετικά με την έκδοση που έχει εγκατασταθεί. Στα επερχόμενα βήματα, θα χρησιμοποιείτε τη γραμμή εντολών των Windows . Για τώρα, εάν χρειάζεται να πληκτρολογήσετε μερικές εντολές, πηγαίνετε στο μενού Έναρξη και πληκτρολογήστε στο πεδίο Αναζήτηση «Command Prompt» (Γραμμή Εντολών). (Σε παλαιότερες εκδόσεις των Windows, μπορείτε να ξεκινήσετε τη γραμμή εντολών με το Start menu → Windows System → Command Prompt.) Μπορείτε να επίσης κρατήστε πατημένο το πλήκτρο των Windows και το πλήκτρο «R» μέχρι να εμφανιστεί το παράθυρο «Εκτέλεση» (Run). Για να ανοίξετε τη γραμμή εντολών, πληκτρολογήστε "cmd" και πατήστε enter στο παράθυρο «Εκτέλεση» (Run).




Σημείωση: Αν χρησιμοποιείτε μια παλαιότερη έκδοση των Windows (7, Vista ή οποιαδήποτε παλαιότερη έκδοση) και το πρόγραμμα εγκατάστασης της Python 3.6.x αποτύχει με σφάλμα, τότε μπορείτε να δοκιμάσετε τα εξής:

1. Εγκαταστήστε όλες τις ενημερώσεις των Windows και προσπαθήστε ξανά να εγκαταστήσετε Python ή
2. εγκαταστήστε μια παλαιότερη έκδοση της Python, π.χ. 3.4.6.

Εάν εγκαταστήσετε μια παλαιότερη έκδοση της Python, η οθόνη εγκατάστασης μπορεί να φαίνεται λίγο διαφορετική από ότι φαίνεται παραπάνω. Βεβαιωθείτε ότι μπορείτε να μετακινηθείτε προς τα κάτω για να δείτε το «Add python.exe to Path». Όταν το δείτε τότε κάντε κλικ στο κουμπί στην αριστερή πλευρά και διαλέξτε το "Will be installed on local hard drive". Όπως φαίνεται και στην παρακάτω εικόνα. Για την επιβεβαίωση της εγκατάστασης της γλώσσας προγραμματισμού Python σε μηχάνημα που φέρει το λειτουργικό Windows πρέπει να ανοιχτεί μια Windows Console και ο χρήστης να πληκτρολογήσει την ακόλουθη εντολή.
Σε περίπτωση λανθασμένης εγκατάστασης θα πρέπει να ακολουθηθούν τα παραπάνω βήματα από την αρχή.

```
$ python --version
```

4.3.2. Εγκατάσταση Python σε Ubuntu
Τα περισσότερα συστήματα Unix έχουν εγκατεστημένη την γλώσσα προγραμματισμού Python για να γίνει ο σχετικός έλεγχος ανοίξτε μια κονσόλα και πληκτρολογήστε τα ακόλουθα:

```
$ python --version
```

Στην περίπτωση που δεν αναγνωρίζεται η συγκεκριμένη εντολή τότε για την εγκατάσταση της Python στο λειτουργικό σύστημα Ubuntu πρέπει να εκτελεστεί Η επόμενη εντολή:

```
$ sudo apt install python3
```

4.4. Εγκατάσταση Elasticsearch και Kibana
Για την εγκατάσταση των Elasticsearch και Kibana είναι απαραίτητο να έχει ολοκληρωθεί πρώτα η εγκατάσταση του εργαλείου Docker και της γλώσσας προγραμματισμού Python, καθώς για την εγκατάσταση τους, ανεξάρτητα πλατφόρμας δηλαδή είτε είναι Windows είτε είναι Ubuntu θα χρησιμοποιηθούν docker compose scripts τα οποία αυτοματοποιούν την όλη διαδικασία [46].
Συνεπώς αρχικά ελέγχουμε αν έχουν εγκατασταθεί σωστά το Docker, docker-compose και Python στο μηχάνημα μας. Αυτο πραγματοποιείται εκτελώντας τις παρακάτω εντολές:

```
$ docker ps
$ docker-compose
$ python --version
```

Αφού εκτελέσουμε τις παραπάνω εντολές και διαπιστώσουμε ότι αποκρίνονται σωστά, όπως περιγράφεται και στα παραπάνω κεφάλαια πρέπει να εκτελέσουμε το παρακάτω script:

```yaml
version: '3.4'

services:

  elasticsearch:
    image: docker.elastic.co/elasticsearch/elasticsearch:6.6.2
    hostname: elasticsearch
    container_name: elasticsearch
    restart: always
    environment:
      - "ES_JAVA_OPTS: -Xms750m -Xmx750m"
    networks:
      docker-elk:
    ports:
      - "9200:9200"

  kibana:
    image: docker.elastic.co/kibana/kibana:6.6.2
    container_name: kibana
    restart: always
    environment:
      - "ELASTICSEARCH_URL=http://elasticsearch:9200"
    networks:
      - docker-elk
    ports:
      - "5601:5601"
    depends_on:
      - elasticsearch


volumes:
  db:

networks:
  docker-elk:
    driver: bridge
    driver_opts:
      com.docker.network.driver.mtu: 1400
```

Σε αυτό το σημείο θα αναλύσουμε τι σημαίνουν οι παραπάνω γραμμές του αρχείου που παρατίθεται. Για λόγους συμβολισμού αναφέρουμε ότι η παρακάτω αρίθμηση αναφέρεται στις αντίστοιχες γραμμές του παραπάνω εικονιζόμενου αρχείου.
Οι γραμμές 1 -3 χρησιμοποιούνται για να αναφέρουν την έκδοση του docker-compose Που χρησιμοποιείται καθώς και για να καθορίσουν το είδος των εγκαθιστομένων λογισμικών.

- Η γραμμή 1 "version: '3.4'": αναφέρει την έκδοση του docker-compose που χρησιμοποιείται.
- Η γραμμή 3 "services:": αναφέρει ότι από αυτή την γραμμή και έπειτα περιγράφονται σε μορφή YAML οι υπηρεσίες (Elasticsearch, Kibana) που πρόκειται να εγκατασταθούν.

Οι γραμμές 5 - 15 του παραπάνω YAML αρχείου χρησιμοποιούνται για την εγκατάσταση του Elasticsearch στο μηχάνημα που κάνει host την εφαρμογή.

- Η γραμμή 5 "Elasticsearch:": αποτελεί το όνομα του λογισμικού/ υπηρεσίας προς εγκατάσταση, σε αυτή την περίπτωση ονοματίζουμε το Elasticsearch
- Η γραμμή 6 "image: docker.elastic.co/Elasticsearch/Elasticsearch:6.6.2": αναφέρει το όνομα της εικόνας που πρέπει να ληφθεί από το Docker Hub και να εγκατασταθεί στο μηχάνημα που θα κάνει Host την εφαρμογή.
- Η γραμμή 7 "hostname: Elasticsearch": θέτει ποιο θα είναι το Hostname της υπηρεσίας
- Η γραμμή 8 "container_name: Elasticsearch": θέτει ποιο θα είναι το όνομα του docker container για το Elasticsearch που θα δημιουργηθεί μετά την εκτέλεση του παραπάνω script.
- Η γραμμή 9 "restart: always": σημαίνει ότι κάθε φορά που γίνεται επανεκκίνηση το μηχάνημα θα επανεκκινούνται και οι υπηρεσίες που περιγράφονται σε αυτό το YAML script.
- Η γραμμή 10 "environment:": αναφέρει ότι από αυτό το σημείο και μετά θέτονται οι μεταβλητές περιβάλλοντος του Elasticsearch.

- Η γραμμή 11 "- "ES_JAVA_OPTS: -Xms750m -Xmx750m"": θέτει την μεταβλητή **ES_JAVA_OPTS** του Elasticsearch
- Η γραμμή 13 "docker-elk:": θέτει σε ποιο Docker Network θα ανήκει η υπηρεσία Elasticsearch
- Η γραμμή 15 "- "9200:9200"": αναφέρει ότι η εσωτερική πόρτα του Elasticsearch 9200 γίνεται populate στην εξωτερική πόρτα 9200 του συστήματος

Οι γραμμές 17 - 28 του παραπάνω εικονιζόμενου αρχείου χρησιμοποιούνται για την εγκατάσταση του Kibana.

- Η γραμμή 17 "kibana:": ονοματίζει την εγκαθιστάμενη υπηρεσία / λογισμικό
- Η γραμμή 18 "image: docker.elastic.co/kibana/kibana:6.6.2": αναφέρει ποια εικόνα από το Docker Hub σχετική με το Kibana θα ληφθεί και θα εγκατασταθεί
- Η γραμμή 19 "container_name: kibana": χρησιμοποιείται για να ονομαστεί ο Docker container του Kibana
- Η γραμμή 20 "restart: always": σημαίνει ότι κάθε φορά που επανεκκινείται το μηχάνημα θα επανεκκινουνται και οι υπηρεσίες που περιγράφονται σε αυτό το YAML script.
- Η γραμμή 21- 22 χρησιμοποιούνται για τον καθορισμό των μεταβλητών περιβάλλοντος του Kibana, πιο συγκεκριμένα θέτουμε το hostname (γραμμή 7) του Elasticsearch "- "ELASTICSEARCH_URL=http://Elasticsearch:9200""
- Οι γραμμές 23-24 χρησιμοποιούνται για τον καθορισμό του Docker Network του Kibana Container "- docker-elk"
- Οι γραμμές 25-26 "- "5601:5601"" αναφέρει ότι η εσωτερική πόρτα του Kibana 5601 γίνεται populate στην εξωτερική πόρτα 5601 του συστήματος
- Η γραμμή 27- 28 "- Elasticsearch": αναφέρει ότι για να εγκατασταθεί το Kibana πρέπει πρώτα να εγκατασταθεί το Elasticsearch

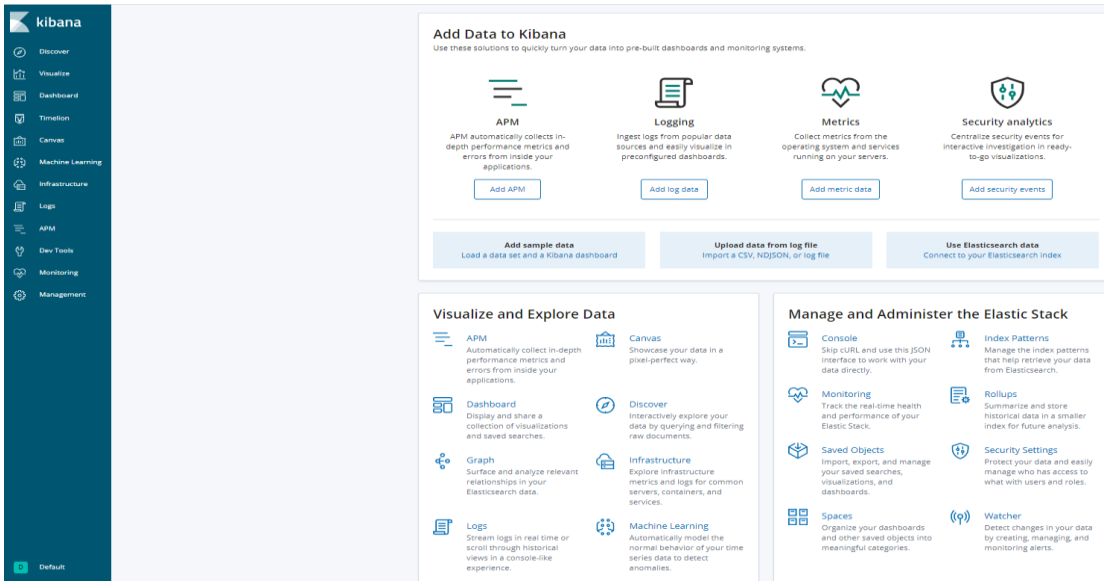
Οι γραμμές 34-38 χρησιμοποιούνται για τον καθορισμό των Docker Networks του Docker Compose, πιο συγκεκριμένα ορίζεται το Docker Network **docker-elk** στο οποίο υπάρχουν το Elasticsearch και το Kibana και μέσω αυτού είναι εύκολο να προσπεραστούν αυτά τα components μέσω των hostnames τους.

- Γραμμή 34-36 "docker-elk:" Καθορισμός Docker Network
- Γραμμή 37-38 "com.docker.network.driver.mtu: 1400": Καθορισμός αριθμού πακέτων που θα διέρχονται στο Docker Network σε κάθε κλήση.

Σε τελευταίο στάδιο για την εγκατάσταση αυτών των εργαλείων πρέπει να βρεθούμε στον φάκελο που βρίσκεται το παραπάνω docker-compose script και να εκτελέσουμε την παρακάτω εντολή στο terminal του μηχανήματός μας:

```
$ docker-compose up -d
```

Για την επαλήθευση της σωστής εγκατάστασης του εργαλείου Kibana πρέπει να σε αυτή την διεύθυνση του μηχανήματος **http://127.0.0.1:5601/app/kibana** και αφού μεταβούμε πρέπει να δούμε το dashboard του Kibana:




Ομοίως για να επαληθεύσουμε ότι το εργαλείο Elasticsearch έχει εγκατασταθεί επιτυχημένα στο μηχάνημα μας πρέπει να μεταβούμε σε αυτόν εδώ τον σύνδεσμο **http://127.0.0.1:9200/** και πρέπει να δούμε στον browser το παρακάτω μήνυμα:


```
{
  "name" : "kKGJ23o",
  "cluster_name" : "docker-cluster",
  "cluster_uuid" : "2gGtfH8VQ8abNpQNcV12jw",
  "version" : {
    "number" : "6.6.2",
    "build_flavor" : "default",
    "build_type" : "tar",
    "build_hash" : "3bd3e59",
    "build_date" : "2019-03-06T15:16:26.864148Z",
    "build_snapshot" : false,
    "lucene_version" : "7.6.0",
    "minimum_wire_compatibility_version" : "5.6.0",
    "minimum_index_compatibility_version" : "5.0.0"
  },
  "tagline" : "You Know, for Search"
}
```


Για την εγκατάσταση της Python βιβλιοθήκης του Elasticsearch εκτελούμε την παρακάτω εντολή [47]:

```
$ pip install Elasticsearch
```

και για να επαληθεύσουμε την σωστή χρήση της βιβλιοθήκης εκτελούμε το παράδειγμα κώδικα που βρίσκεται στην επίσημη σελίδα της βιβλιοθήκης:

```
from datetime import datetime
from Elasticsearch import Elasticsearch
es = Elasticsearch()

doc = {
    'author': 'kimchy',
    'text': 'Elasticsearch: cool. bonsai cool.',
    'timestamp': datetime.now(),
```

```
}
res = es.index(index="test-index", doc_type='tweet', id=1, body=doc)
print(res['result'])

res = es.get(index="test-index", doc_type='tweet', id=1)
print(res['_source'])

es.indices.refresh(index="test-index")

res = es.search(index="test-index", body={"query": {"match_all": {}}})
print("Got %d Hits:" % res['hits']['total']['value'])
for hit in res['hits']['hits']:
    print("%(timestamp)s %(author)s: %(text)s" % hit["_source"])
```

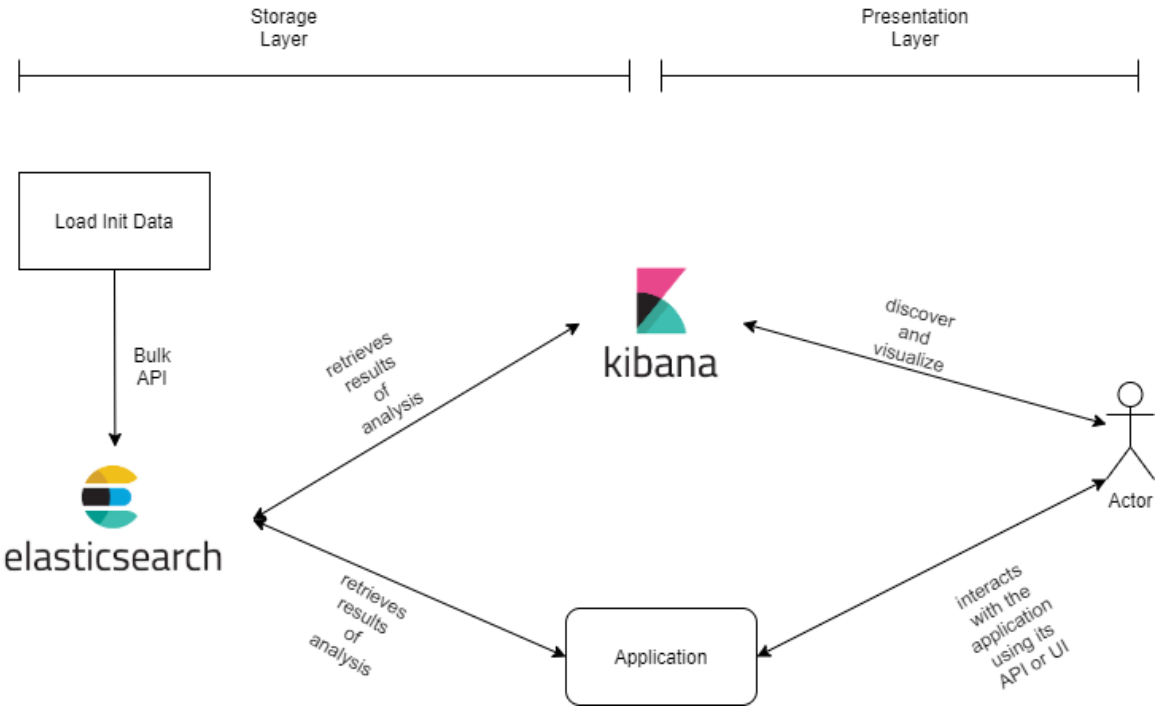
5. Αρχιτεκτονική Συστήματος

5.1. Εισαγωγή

Σε αυτό το σημείο θα παρατεθεί η αρχιτεκτονική του συστήματος που παρουσιάζεται σε αυτή την διπλωματική εργασία. Πιο συγκεκριμένα θα παρουσιαστεί το γενικό σχήμα της αρχιτεκτονικής και σε επόμενο στάδιο θα σχολιαστούν οι αλληλεπιδράσεις μεταξύ των επιμέρους υποσυστημάτων.

5.2. Σχήμα Αρχιτεκτονικής

Το παρακάτω σχήμα παραθέτει την αρχιτεκτονική του συστήματος της διπλωματικής εργασίας:



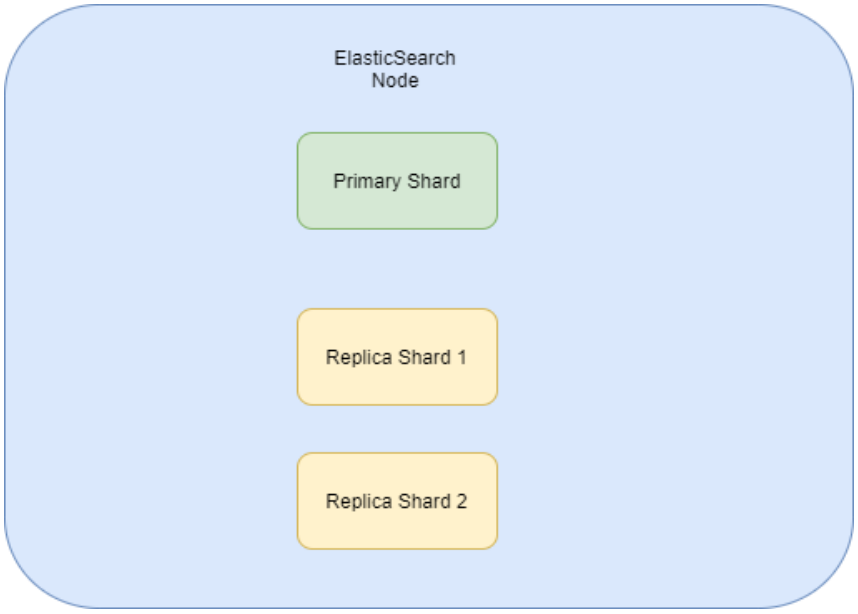

Όπως παρατηρείται από το σχήμα της αρχιτεκτονικής η εφαρμογή αποτελείται από 2 layers το Storage Layer και το Presentation Layer. Τα δεδομένα που είναι σχετικά με βιβλία εισάγονται στο Elasticsearch με την χρήση του Bulk API, το εργαλείο Kibana στέλνει requests για ανάλυση, αναζήτηση, εισαγωγή, διαγραφή και ανανέωση δεδομένων στο Elasticsearch και λαμβάνει τα αντίστοιχα responses διαθέτει και User Interface με σκοπό την παροχή στον τελικό χρήση δυνατοτήτων εξερεύνησης των λειτουργιών του Elasticsearch. Το κυριότερο μέρος αποτελεί η εφαρμογή η οποία έχει υλοποιηθεί με την χρήση της γλώσσας προγραμματισμού Python και της βιβλιοθήκης Elasticsearch η οποία αλληλοεπιδρά με το εργαλείο Elasticsearch. Με την εφαρμογή μπορεί να αλληλοεπιδράσει ο τελικός χρήστης κάνοντας χρήση των APIs καθώς και με την χρήση ενός User Interface, το οποίο καλεί τα APIs της εφαρμογής χρησιμοποιώντας την γλώσσα προγραμματισμού JavaScript. Στις επόμενες παραγράφους παρουσιάζονται τα 2 layers της αρχιτεκτονικής καθώς και τα υποσυστήματα που υπάγονται σε αυτά τα layers.

5.2.1. Storage Layer

Το storage layersαποτελεί την κύρια υποδομή της αρχιτεκτονικής καθώς σε αυτό υπάγεται το εργαλείο Elasticsearch καθώς και το σύστημα που αρχικοποιεί τον δείκτη book_index στον οποίο αποθηκεύονται τα δεδομένα των βιβλίων και καθορίζει ποια θα είναι τα properties, δηλαδή τα πεδία των βιβλίων που θα αποθηκεύονται στον δείκτη book_index, καθώς και αποθηκεύει δεδομένα βιβλίων.

5.2.1.1. Application Elasticsearch Cluster

Το Elasticsearch cluster που χρησιμοποιείται είναι single node, λόγω υπολογιστικών πόρων, και δομείται με τον εξής τρόπο έχει ένα Primary shard δηλαδή, σαν shard ορίζεται το πλήθος των δεδομένων. Επίσης έχει και άλλα 2 replicas αυτού του shard κυρίως για λόγους κλιμάκωσης και περιορισμού των λαθών. Χρησιμοποιεί την πόρτα 9200 για να λαμβάνει requests για διαδικασίες ανάλυσης, αναζήτησης, εισαγωγής, διαγραφής, ανανέωσης και aggregations.




Κατά την αρχικοποίηση του δείκτη που αποθηκεύει τα δεδομένα των βιβλίων γίνεται και η διαδικασία του mapping κατά την οποία καθορίζεται πως ένα βιβλίο και γενικότερα ένα document θα αποθηκεύεται στον δείκτη book_index. Αυτό εκτελείται με την λειτουργία mapping του Elasticsearch και είναι η παρακάτω κλήση.

```
PUT /book_index
{
  "mappings": {
    "book_doc": {
       "properties": {
         "title": {
           "type": "text"
         },
         "authors": {
           "type": "text"
         },
         "summary": {
           "type": "text"
         },
         "publish_date": {
           "type": "text"
         },
         "num_reviews": {
           "type": "integer"
         },
         "publisher": {
           "type": "text"
         }
       }
}
}
}
```

Όπως παρατηρείται ορίζεται από την διαδικασία mapping ότι τα documents που θα αποθηκεύονται στον δείκτη book_index θα είναι του τύπου book_doc και θα περιλαμβάνουν τίτλο που θα είναι σε μορφή text, authors που θα είναι σε μορφή text, summary στην μορφή text, publish_date στην μορφή

date και publisher στη μορφή text. Το Python script initializer.py εκτελεί την παραπάνω αρχικοποίηση καθώς και την εισαγωγή δεδομένων με χρήση του Bulk API. Συνεπώς τα δεδομένα που εισάγονται στο σύστημα πρέπει να είναι της μορφής:

{"title": "Elasticsearch: The Definitive Guide", "authors": ["clinton gormley", "zachary tong"],"summary": "A distibuted real-time search and analytics engine", "publish_date": "2015-02-07", "num_reviews": 20,"publisher": "oreilly"},

Το αρχείο **initializer.py** αρχικά δημιουργεί τον δείκτη book_index και στην συνέχεια εφαρμόζει τα παραπάνω mappings έτσι ώστε τα δεδομένα των βιβλίων να εισάγονται με συγκεκριμένα μορφή. Στα πλαίσια της εφαρμογής έχει υλοποιηθεί η διαδικασία αρχικοποίησης του book_index στην γλώσσα προγραμματισμού Python.

```
        body = {
            "mappings": {
                "book_doc": {
                    "properties": {
                        "title": {
                            "type": "text"
                        },
                        "authors": {
                            "type": "text"
                        },
                        "summary": {
                            "type": "text"
                        },
                        "publish_date": {
                            "type": "date"
                        },
                        "num_reviews": {
                            "type": "integer"
                        },
                        "publisher": {
                            "type": "text"
                        }
                    }
                }
            }
        }
        self.es.indices.create(index=self.book_index, body=body, ignore=400)
```

Και σε επόμενο βήμα εκτελείται η εισαγωγή δεδομένων με την χρήση του Bulk API που και αυτό έχει υλοποιηθεί στην γλώσσα προγραμματισμού Python στα πλαίσια της εφαρμογής.

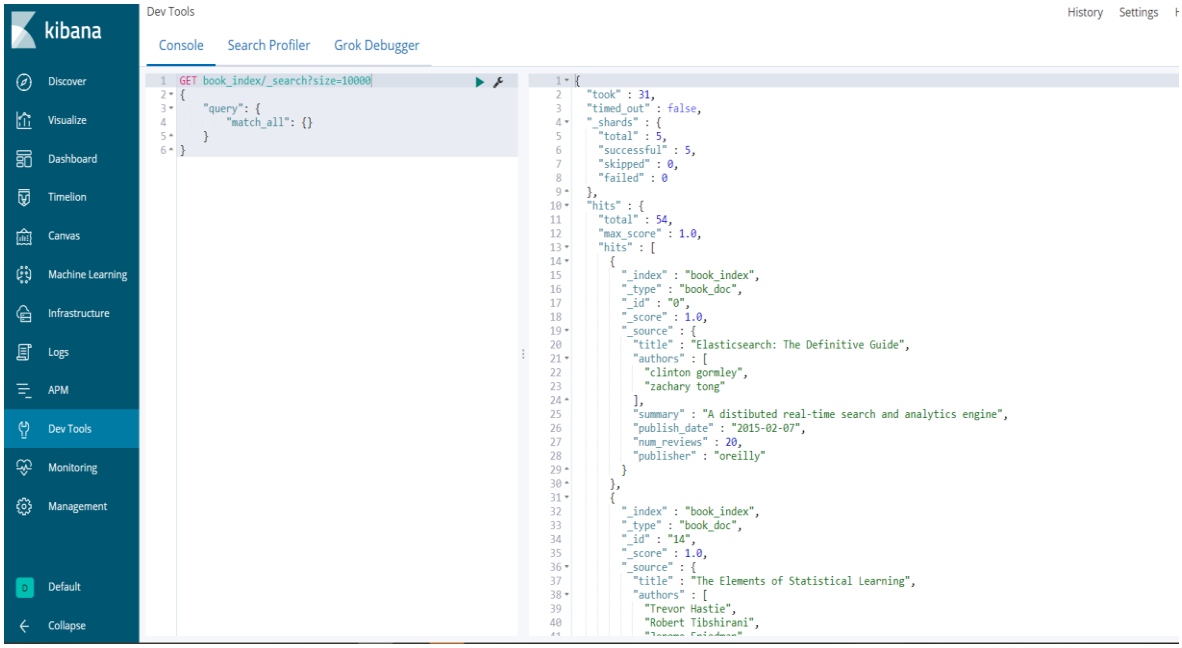
5.2.2. Presentation Layer

Το συγκεκριμένο layer ονομάστηκε έτσι καθώς αποτελεί το στρώμα της αρχιτεκτονικής στο οποίο τα επιμέρους υποσυστήματα είτε λόγω της ίδιας τους της λειτουργίας (Kibana) είτε λόγω της υλοποίησης τους (η εφαρμογή) χρησιμοποιούνται για εξερεύνηση στα δεδομένα που είναι καταχωρημένα και καλούν τις επιμέρους λειτουργίες του Elasticsearch για αναζήτηση ανάκτηση και aggregations πάνω στα δεδομένα βιβλίων.

5.2.2.1. Υποσύστημα Kibana

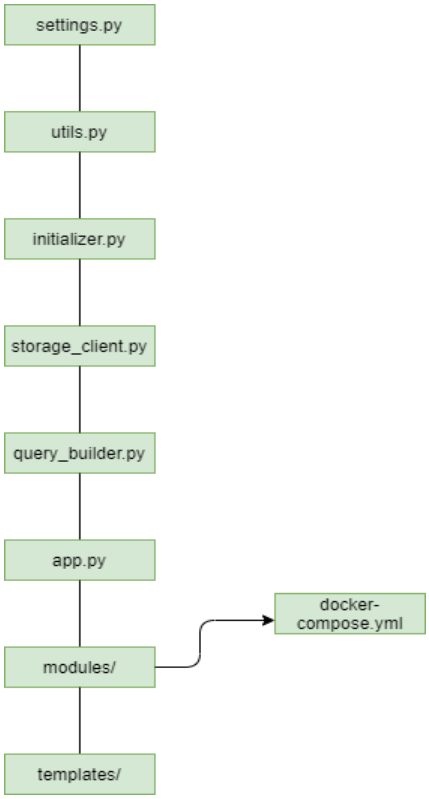
Το εργαλείο Kibana αποτελεί ένα από τα θεμελιώδη component του ELK stack και η γενικότερη χρήση του αποτελεί η κλήση των Elasticsearch λειτουργιών και για εξερεύνηση των δεδομένων. Όλες οι λειτουργίες που περιγράφονται στο κεφάλαιο 6 καλούνται μέσω του Kibana και μέσω της

βιβλιοθήκης Elasticsearch από την πλευρά της εφαρμογής. Το Kibana πρέπει να γνωρίζει ποια είναι η IP και η πόρτα του Elasticsearch, στην περίπτωση της εφαρμογής είναι http://localhost:9200. Επιπλέον το εργαλείο Kibana προσφέρει ένα User Interface το οποίο είναι προσβάσιμο στην πόρτα 5601, πιο συγκεκριμένα αν κάποιος προσπεράσει αυτή την διεύθυνση http://localhost:5601 και στην συνέχεια επιλέξει από την αριστερή στήλη το tab “Dev Tools” έχει την δυνατότητα να δοκιμάσει τις λειτουργίες του Elasticsearch μέσω του Kibana. Στην παρακάτω εικόνα πραγματοποιείται κλήση στο Kibana για λήψη όλων των βιβλίων - documents που είναι αποθηκευμένα στον δείκτη book_index.




5.2.2.2. Εφαρμογή

Σε αυτό το σημείο θα γίνει ανάλυση του υποσυστήματος της εφαρμογής που υλοποιήθηκε στα πλαίσια της διπλωματικής εργασίας. Η εφαρμογή έχει υλοποιηθεί στην γλώσσα προγραμματισμού Python 3.7 και χρησιμοποιεί την αντίστοιχη βιβλιοθήκη του εργαλείου Elasticsearch για να καλέσει τις αντίστοιχες λειτουργίες του εργαλείου. Πιο συγκεκριμένα η δομή των αρχείων της εφαρμογής είναι η παρακάτω:




- **settings.py**: Αυτό το αρχείο περιέχει όλες τις στατικές μεταβλητές της εφαρμογής όπως για παράδειγμα την πόρτα 9200 από την οποία είναι προσβαάσιμη το Elasticsearch.
- **utils.py**: Το αρχείο utils.py περιέχει κάποιες βοηθητικές συναρτήσεις που χρησιμοποιούνται από τα υπόλοιπα αρχεία της εφαρμογής όπως για παράδειγμα διαχείριση JSON δεδομένων.
- **initializer.py**: Το συγκεκριμένο Python Script χρησιμοποιείται για την αρχικοποίηση του δείκτη book_index όπου θα αποθηκεύονται τα βιβλία καθώς και για την εισαγωγή δεδομένων στο Elasticsearch.
- **storage_client.py:** Το συγκεκριμένο αρχείο χρησιμοποιεί την Python βιβλιοθήκη του Elasticsearch που ονομάζεται "Elasticsearch" με σκοπό να εκτελέσει λειτουργίες σχετικές με εισαγωγή, αναζήτηση, ανανέωση, διαγραφή δεδομένων. Το μόνο που χρειάζεται η συγκεκριμένη βιβλιοθήκη για να λειτουργήσει είναι την πόρτα του Elasticsearch η οποία είναι η 9200 καθώς και τον δείκτη στον οποίο βρίσκονται τα δεδομένα των βιβλίων, δηλαδή τον δείκτη book_index. Το αρχείο αυτό περιέχει υλοποιημένες τις λειτουργίες του Elasticsearch που περιγράφονται στο κεφάλαιο της υλοποίησης.
- **query_builder.py**: Όπως αναφέρθηκε πριν το αρχείο storage_client.py περιέχει συναρτήσεις οι οποίες επιτελούν λειτουργίες του συστήματος Elasticsearch. Στα πλαίσια της εφαρμογής χρειάστηκε η ύπαρξη ενός επιμέρους αρχείου το οποίο χρησιμοποιεί αυτές τις συναρτήσεις. Αυτό το αρχείο είναι το query_builder.py και πιο συγκεκριμένα καλεί τις συναρτήσεις του storage_client.py βάσει των παραμέτρων που δέχεται.
- **app.py**: Το συγκεκριμένο Python αρχείο χρησιμοποιείται για την δημιουργία ενός API για την χρήση των λειτουργιών που υλοποιήθηκαν από τα αρχεία query_builder και storage_client. Το API αυτό δέχεται POST requests από τον τελικό χρήστη, όπου ο χρήστης δίνει σαν παραμέτρους ποια λειτουργία στο Elasticsearch θέλει να επιτελέσει καθώς και της επιμέρους παραμέτρους και η εφαρμογή του απαντάει πίσω με τα δεδομένα που ζήτησε. Αυτό το API είναι προσβάσιμο από το URL /ask/storage/ και η πόρτα που δέχεται requests είναι η 5000. Πιο συγκεκριμένα στο επόμενο κεφάλαιο στην παράγραφο που αναφέρεται ο Query Builder υπάρχουν παραδείγματα χρήσης του API (βλέπε 6.7).
- **modules/**: Ο συγκεκριμένος φάκελος περιέχει το αρχείο docker-compose.yml το οποίο χρησιμοποιείται για την εγκατάσταση των εργαλείων Elasticsearch και Kibana.
- **templates/**: Ο συγκεκριμένος φάκελος περιέχει HTML αρχεία τα οποία με την χρήση JavaScript καλούν το API της εφαρμογής με σκοπό την δημιουργία ενός User Interface για

την χρήση των λειτουργιών του Elasticsearch, το User Interface της εφαρμογής αναλύεται πιο λεπτομερώς στο επόμενο κεφάλαιο (βλέπε 6.8)

Συνεπώς στα πλαίσια χρήσης της εφαρμογής ο τελικός χρήστης έχει την δυνατότητα να αλληλοεπιδράσει με το Elasticsearch κάνοντας χρήση το API της εφαρμογής είτε κάνοντας χρήση το User Interface της εφαρμογής με σκοπό την εισαγωγή δεδομένων βιβλίων, την ανανέωση είτε την διαγραφή υπαρχόντων δεδομένων, την αναζήτηση βιβλίων βάσει των παραμέτρων τους καθώς και την λήψη aggregations σχετικών με τα δεδομένα των βιβλίων.

Το Dataset (πήραμε ένα υποσύνολο 60 βιβλίων και όχι όλο το dataset) της διπλωματικής εργασίας το πήραμε από το Kaggle dataset repository[72]. Επιπλέον αξίζει να σημειώσουμε ότι έχουμε πάρει το αρχείο και κρατήσαμε ένα υποσύνολο των στηλών. Πιο συγκεκριμένα κρατήσαμε τα πεδία του dataset που αναφέρουν τα παρακάτω:

1. Τίτλος βιβλίου
2. Περιγραφή βιβλίου
3. Συγγραφείς του βιβλίου
4. Ημερομηνία έκδοσης
5. Αριθμός reviews για το βιβλίο

# 6. Υλοποίηση

## 6.1. Εισαγωγή

Σε αυτό το κεφάλαιο παρουσιάζεται η προγραμματιστική υλοποίηση της διπλωματικής. Πιο συγκεκριμένα έχει εγκατασταθεί το Elasticsearch, το εργαλείο Kibana με την χρήση docker και docker-compose καθώς και η γλώσσα προγραμματισμού Python. Στην συνέχεια με την χρήση των

κατάλληλων βιβλιοθηκών την Python υλοποιούνται κάποιες κύριες λειτουργίες όσον αφορά την αναζήτηση σε δεδομένα κειμένου και στην προκειμένη περίπτωση να γίνει χρήση των κατάλληλων queries του Elasticsearch για να επιτευχθεί αναζήτηση πάνω σε πληροφορία σχετική με βιβλία (τίτλος βιβλίου, συγγραφείς, summary, εκδότης κλπ.). Πιο συγκεκριμένα περιγράφεται η ανάπτυξη λογισμικού καθώς και η μεθοδολογία που ακολουθήθηκε για την χρήση του εργαλείου Elasticsearch πάνω σε δεδομένα βιβλίων. Αυτά τα δεδομένα είναι τα εξής:

- **title**: Ο τίτλος του βιβλίου, π.χ "Elasticsearch: The Definitive Guide"
- **authors**: Οι συγγραφείς του βιβλίου, π.χ. "[glinton gormley]",
- **summary**: Περίληψη του βιβλίου, π.χ "A distributed real-time search and analytics engine"
- **publish_date**: Ημερομηνία έκδοσης του βιβλίου, π.χ "2015-02-07"
- **num_reviews**: Αριθμός ανασκοπήσεων/κριτικών του βιβλίου, π.χ. "22",
- **publisher**: Εκδοτικός Οίκος βιβλίου, π.χ. "oreilly"

Όλες οι παρακάτω λειτουργίες έχουν υλοποιηθεί στην γλώσσα προγραμματισμού Python και πιο συγκεκριμένα ικανοποιούν το μοντέλο αντικειμενοστραφούς προγραμματισμού. Το Python script **storage_client.py** περιέχει την κλάση **ElasticBookStorage** η οποία στον constructor της αρχικοποιεί την σύνδεση με το Elasticsearch και οι λειτουργίες του Elasticsearch που περιγράφονται στις παρακάτω παραγράφους αποτελούν συναρτήσεις μέλη της κλάσης **ElasticBookStorage.**
Για την σύνδεση με το Elasticsearch μέσω της γλώσσας προγραμματισμού Python χρησιμοποιείται η παρακάτω κλάση, όπου στον constructor της υλοποιείται η σύνδεση (host=127.0.0.1, port=9200)

```
class ElasticBookStorage(object):

    def __init__(self):
        self.book_index = "book_index"
        self.book_doc = "book_doc"

        self.es = Elasticsearch([{
            'host': "127.0.0.1",
            'port': 9200
        }])
```

Οι παρακάτω παράγραφοι περιγράφουν βασικά queries του εργαλείου Elasticsearch. Σημειώνεται ότι τα queries και τα APIs εκτελούνται μέσω του Interface Dev Tools του Kibana.

6.2. Cluster APIs

Σε αυτή την παράγραφο γίνεται αναφορά στις λειτουργίες του Elasticsearch οι οποίες δεν έχουν να κάνουν με την εύρεση, εισαγωγή, διαγραφή δεδομένων αλλά χρησιμοποιούνται για την λήψη πληροφοριών και δεδομένων σχετικά με στατιστικά χρήσης των συστημάτων που φέρουν το Elasticsearch δίνοντας με αυτόν τον τρόπο στον administrator χρήστη του συστήματος την ευχέρεια να γνωρίζει πότε θα χρειαστεί να προσθέσει νέους κόμβους στο cluster. Επίσης με την χρήση τους επιτυγχάνεται η επικοινωνία μεταξύ των κόμβων χωρίς να υπάρχει πρόβλημα καθώς τα δεδομένα αυξάνονται. Η χρήση των Cluster APIs βοηθάει στην πρόληψη καταστάσεων "split brain" όπου συνήθως λόγω networking issues σταματάει να υπάρχει επικοινωνία μεταξύ των κόμβων και ο Masternode χάνει οποιαδήποτε πληροφορία σχετική με το που βρίσκονται τα δεδομένα π.χ. χάνεται η πληροφορία σε ποιον datanode μπορεί να υπάρχουν δεδομένα αναζήτησης χρηστών [48].

6.2.1. Cluster Health API

Το συγκεκριμένο API επιστρέφει πληροφορία σχετική με την υγεία/κατάσταση ενός Elasticsearch cluster, το συγκεκριμένο API μπορεί να κληθεί με την χρήση συγκεκριμένων indices με σκοπό να προσδιοριστεί η κατάσταση των δεικτών που δόθηκαν σαν παράμετροι. Σύμφωνα με το επίσημο documentation του Elasticsearch η κατάσταση ενός cluster μπορεί να προσδιορίζεται σύμφωνα με τις παρακάτω τάξεις [49]:

- **green**: Σημαίνει ότι η κατάσταση του συστήματος είναι πλήρως λειτουργική και ότι τα δεδομένα έχουν κατανεμηθεί ή κατακερματιστεί σε όλους τους κόμβους του cluster. Επίσης υποδεικνύει την ύπαρξη και άλλων κόμβων με αποτέλεσμα να γίνεται εφικτή η λειτουργία replication των δεδομένων έτσι ώστε αν καταρρεύσει ένας από τους κόμβους τα δεδομένα που υπήρχαν σε αυτόν να βρίσκονται και σε άλλους. Με αυτόν τον τρόπο δεν υπάρχει απώλεια δεδομένων.
- **yellow**: Σημαίνει ότι η κατάσταση του συστήματος είναι πλήρως λειτουργική και ότι τα δεδομένα θα αποθηκευτούν σε έναν κόμβο ο οποίος λειτουργεί σαν masternode και σαν datanode, πιο συγκεκριμένα σε αυτή την περίπτωση δεν υπάρχει δυνατότητα replication (αντιγραφής) των δεδομένων μιας και υπάρχει μόνο ένας κόμβος.
- **red**: Η κατάσταση αυτή υποδεικνύει ότι το cluster δεν είναι υγιές και είναι αναγκαίο να προστεθεί ένας η περισσότεροι κόμβοι στο σύστημα, αυτό μπορεί να συμβεί όταν υπάρχει μεγάλος όγκος δεδομένων και τα όρια αποθήκευσης των κόμβων έχουν φτάσει το ανώτερο.

Ένα από τα πλεονεκτήματα της χρήσης του Elasticsearch είναι ότι το cluster επιτρέπει λειτουργίες εγγραφής, διαγραφής, εισαγωγής μόνο όταν η κατάσταση του βρίσκεται στην κόκκινη ή κίτρινη τάξη. Με το παρακάτω κομμάτι κώδικα επιτυγχάνεται η κλήση του Cluster Health API.

```
GET /_cluster/health
```

Στα πλαίσια της εφαρμογής υλοποιήθηκε συνάρτηση στην γλώσσα προγραμματισμού που υλοποιεί την λειτουργία του συγκεκριμένου API.

```
def get_cluster_health(self):
    """This function is used to retrieve cluster health info"""
    try:
        cluster_health = self.es.cluster.health()
        return cluster_health
    except Exception as ex:
        print(ex, flush=True)
```

Η έξοδος της παραπάνω λειτουργίας είναι η εξής:

```
{
  "cluster_name" : "docker-cluster",
  "status" : "yellow",
  "timed_out" : false,
  "number_of_nodes" : 1,
  "number_of_data_nodes" : 1,
  "active_primary_shards" : 6,
  "active_shards" : 6,
  "relocating_shards" : 0,
  "initializing_shards" : 0,
  "unassigned_shards" : 5,
  "delayed_unassigned_shards" : 0,
  "number_of_pending_tasks" : 0,
  "number_of_in_flight_fetch" : 0,
  "task_max_waiting_in_queue_millis" : 0,
  "active_shards_percent_as_number" : 54.54545454545454
}
```

6.2.2. Cluster Stats API

Το συγκεκριμένο API αποτελεί μια πιο λεπτομερή επέκταση του cluster health API, το οποίο περιεγράφηκε στην προηγούμενη παράγραφο. Επιστρέφει πιο λεπτομερή στατιστικά σχετικά με μετρικές δεικτών στου Elasticsearch, όπως που βρίσκονται αντίγραφα δεδομένων των συγκεκριμένων δεικτών, χώρος που καταναλώνεται για την αποθήκευση αυτών των δεδομένων, CPU usage. Επίσης επιστρέφεται και πληροφορία πιο σχετική με το cluster, για παράδειγμα, ο αριθμός των κόμβων ενός cluster, οι ρόλοι κάθε κόμβου (datanode/masternode), έκδοση Java που χρησιμοποιείται από το σύστημα και εγκατεστημένα Plugins του συστήματος. Το συγκεκριμένο API καλείται στο Kibana με το παρακάτω [50]:

```
GET /_cluster/stats
```

Και στα πλαίσια της εφαρμογής υλοποιήθηκε η παρακάτω συνάρτηση στην γλώσσα προγραμματισμού Python η οποία συνδέεται με το Elasticsearch cluster και επιστρέφει τα στατιστικά του. Μετά την παρακάτω συνάρτηση περιλαμβάνεται και μέρος του αποτελέσματος της συγκεκριμένης λειτουργίας.

```python
def get_cluster_stats(self):
    """This function is used to retrieve cluster stats info"""
    try:
        cluster_stats = self.es.cluster.stats()
        return cluster_stats
    except Exception as ex:
        print(ex, flush=True)
```

```
"indices" : {
  "count" : 2,
  "shards" : {
    "total" : 6,
    "primaries" : 6,
    "replication" : 0.0,
    "index" : {
      "shards" : {
        "min" : 1,
        "max" : 5,
        "avg" : 3.0
      },
      "primaries" : {
        "min" : 1,
        "max" : 5,
        "avg" : 3.0
      },
      "replication" : {
        "min" : 0.0,
        "max" : 0.0,
        "avg" : 0.0
      }
    }
  },
  "docs" : {
    "count" : 57,
    "deleted" : 0
  },
```

Η επόμενη παράγραφος περιγράφει την δημιουργία κατάλληλων δεικτών, indices, για την αποθήκευση δεδομένων βιβλίων.

6.3. Δημιουργία Book Index

Σε αυτό το σημείο καθώς και για τους λόγους της εφαρμογής πρέπει να δημιουργηθεί ένας δείκτης (index). Ένας δείκτης είναι σαν μια βάση δεδομένων, σε αυτή την περίπτωση χρησιμοποιείται για την αποθήκευση δεδομένων βιβλίων, συνεπώς σύμφωνα με τα δεδομένα που θα αποθηκευτούν πρέπει να οριστούν και τα αντίστοιχα properties του συγκεκριμένου δείκτη. Αυτά θα είναι το όνομα του τύπου των εγγραφών που θα αποθηκεύονται, εδώ πρέπει να σημειωθεί ότι οι εγγραφές στο Elasticsearch ονομάζονται documents, αλλά και τι δεδομένα πρέπει να περιέχει ένα document [51].
Συνοψίζοντας στα πλαίσια της εφαρμογής ο δείκτης που θα χρησιμοποιηθεί ονομάζεται **book_index,** το είδος των documents αποθηκεύονται θα είναι βιβλία και το όνομα του τύπου ορίστηκε σαν **book_doc** και τα αντίστοιχα properties θα είναι ο τίτλος του βιβλίου, τα ονόματα των συγγραφέων, η περίληψη του βιβλίου, η ημερομηνία έκδοσής του, ο αριθμός κριτικών του και το όνομα του εκδοτικού οίκου.

```
PUT /book_index
{
  "mappings": {
    "book_doc": {
      "properties": {
        "title": {
          "type": "text"
        },
        "authors": {
          "type": "text"
        },
        "summary": {
          "type": "text"
        },
        "publish_date": {
          "type": "date"
        },
        "num_reviews": {
          "type": "integer"
        },
        "publisher": {
          "type": "text"
        }
      }
}
}
}
```

Η αντίστοιχη συνάρτηση σε Python είναι στο παρακάτω χωρίο κώδικα

```
    def create_book_index(self):
        """
        The following function is used to create the book index
        :return: pass
        :Examples:
            >>> elk = ElasticBookStorage()
```

```
        >>> elk.create_book_index()
    """
    try:
        body = {
            "mappings": {
                "book_doc": {
                    "properties": {
                        "title": {
                            "type": "text"
                        },
                        "authors": {
                            "type": "text"
                        },
                        "summary": {
                            "type": "text"
                        },
                        "publish_date": {
                            "type": "date"
                        },
                        "num_reviews": {
                            "type": "integer"
                        },
                        "publisher": {
                            "type": "text"
                        }
                    }
                }
            }
        }
        self.es.indices.create(index=self.book_index, body=body, ignore=400)
    except Exception as ex:
        print(ex, flush=True)
    pass
```

Με την εκτέλεση των παραπάνω δημιουργείται ο δείκτης book_index που αποθηκεύει δεδομένα βιβλίων στα αντίστοιχα book_doc documents. Εν αντιστοιχία ορίστηκαν σαν properties των documents title, authors, summary, publish_date, num_reviews, publisher.

6.4. Διαχείριση Documents

Η διαχείριση των documents, δηλαδή των βιβλίων στην περίπτωση της εφαρμογής, γίνεται με την χρήση των λειτουργιών εισαγωγής δεδομένων, ανανέωσης των δεδομένων, διαγραφής των δεδομένων.

6.4.1. Εισαγωγή βιβλίου στο Elasticsearch

Για την εισαγωγή των δεδομένων βιβλίων στο Elasticsearch θα χρειαστούν να εισαχθούν οι πληροφορία που έχει να κάνει με τον τίτλο του βιβλίου, τα ονόματα των συγγραφέων, την περίληψη του βιβλίου, την ημερομηνία έκδοσης του βιβλίου, τον αριθμό των reviews του αλλά και το όνομα του εκδοτικού οίκου [52].

Η αντίστοιχη λειτουργία του Elasticsearch που χρησιμοποιείται ονομάζεται **create document** και τα δεδομένα εισάγονται στο σώμα κλήσης της λειτουργίας αυτής. Για παράδειγμα το παρακάτω αποτελεί εκτέλεση του create document, όπως παρατηρείται για την δημιουργία του νέου document το οποίο θα αποθηκευτεί στον δείκτη book_index πρέπει να περιλαμβάνει στο σώμα της κλήσης του τα properties που ορίστηκαν κατά την δημιουργία του δείκτη book_index, αυτά τα properties θα είναι τίτλος, συγγραφείς, περίληψη, ημερομηνία έκδοσης, αριθμός reviews, εκδοτικός οίκος.

```
POST /book_index/book_doc
{
  "title": "Using Elasticsearch",
  "authors": ["George kibana"],
  "summary": "This is a guide how to use Elasticsearch",
  "num_reviews":20,
  "publish_date":"2016-07-05",
  "publisher": "wiley"
}
```

Στην γλώσσα προγραμματισμού Python, στα πλαίσια υλοποίησης της εφαρμογής υλοποιήθηκε η παρακάτω συνάρτηση η οποία δέχεται σαν ορίσματα τον τίτλο του βιβλίου, τους συγγραφείς την περιγραφή τον αριθμό των reviews την ημερομηνία έκδοσης καθώς και τον εκδοτικό οίκο και αποθηκεύει αυτή την πληροφορία στον δείκτη book_index και στο document book_doc.

```
    def create_book_doc(self, title, authors, summary, publisher, num_reviews, publish_date):
        """
        The following function is used to create a book entry to Elasticsearch using the provided info
        :param title: book title
        :param authors: book authors
        :param summary: book summary
        :param publisher: book publisher
        :param num_reviews: book number of reviews
        :param publish_date: book publish date
        :return: pass

        :Example:
            >>> title="Some book"
            >>> authors=["Author1", "Author2"]
            >>> summary = "this is a book written by Author1 and Author2"
            >>> publisher = "Publisher SA"
            >>> num_reviews = 20
            >>> publish_date = "2014-04-05"
            >>> elk = ElasticBookStorage()
            >>> elk.create_book_doc(title, authors, summary, publisher, num_reviews, publish_date)
        """
        try:
            body = {
                "title": title,
                "authors": authors,
                "summary": summary,
                "publisher": publisher,
                "num_reviews": num_reviews,
                "publish_date": publish_date
            }
            self.es.index(index=self.book_index, doc_type=self.book_doc, body=body)
        except Exception as e:
            print(e, flush=True)
```

Η παραπάνω συνάρτηση περιέχει και παράδειγμα κλήσης της για την σωστή της χρήση καθώς και την επιτυχή εισαγωγή ενός βιβλίου στο Elasticsearch.

6.4.2. Εισαγωγή πολλών βιβλίων - Bulk Insert

Για την εισαγωγή παραπάνω από ένα βιβλίων στο Elasticsearch υπάρχουν 2 τρόποι. Ο πρώτος είναι η χρήση της προηγούμενης λειτουργίας, πιο συγκεκριμένα η επαναληπτική εισαγωγή δεδομένων βιβλίων στο Elasticsearch. Ο συγκεκριμένος όμως έχει μειονεκτήματα τα πιο σημαντικά είναι ότι σε περίπτωση όπου πρέπει να εισαχθεί μεγάλος όγκος βιβλίων θα δημιουργηθεί τεράστιο overhead καθώς όλοι οι υπολογιστικοί πόροι του συστήματος θα είναι απασχολημένοι σε αυτή την διαδικασία, πράγμα που είναι χρονοβόρο, ακριβό σε πράξεις και δεν επιτρέπεται η παραλληλοποίηση διαδικασιών. Για αυτό τον λόγο ο δεύτερο τρόπος που χρησιμοποιείται ονομάζεται **bulk insertion** η οποία όταν χρησιμοποιείται η εισαγωγή πολλών δεδομένων γίνεται αποδοτικά έτσι μειώνεται σημαντικά το overhead καθώς και ο χρόνος εισαγωγής των δεδομένων [53].
Στην περίπτωση χρήσης της λειτουργία bulk insert εισάγονται τα δεδομένα μαζί με μια κλήση της συνάρτησης. Το παρακάτω αποτελεί ένα παράδειγμα της λειτουργίας στο Kibana.

```
POST /bookdb_index/book_doc/_bulk
  { "index": { "_id": 1 }}
  { "title": "Elasticsearch: The Definitive Guide", "authors": ["clinton gormley", "zachary tong"],
"summary" : "A distibuted real-time search and analytics engine", "publish_date" : "2015-02-
07", "num_reviews": 20, "publisher": "oreilly" }

 { "index": { "_id": 2 }}
 { "title": "Taming Text: How to Find, Organize, and Manipulate It", "authors": ["grant
ingersoll", "thomas morton", "drew farris"], "summary" : "organize text using approaches such
as full-text search, proper name recognition, clustering, tagging, information extraction, and
summarization", "publish_date" : "2013-01-24", "num_reviews": 12, "publisher": "manning" }
```

Όπως παρατηρείται από την χρήση της παραπάνω λειτουργίας ορίζονται για κάθε βιβλίο - document που θα εισαχθεί στο Elasticsearch ο τίτλος, οι συγγραφείς, η περίληψη, η ημερομηνία έκδοσης, ο αριθμός των reviews το όνομα του εκδοτικού οίκου.
Στα πλαίσια της εφαρμογής δημιουργήθηκε στην γλώσσα προγραμματισμού Python η αντίστοιχη συνάρτηση που υλοποιεί την λειτουργία bulk insertion. Όπως παρατηρείται η παρακάτω συνάρτηση δέχεται σαν είσοδο την λίστα με τα δεδομένα σε μορφή ευρετηρίων με αυτό τον τρόπο γίνεται πιο αποδοτικά η εισαγωγή μέσω την λειτουργία των ευρετηρίων στην γλώσσα προγραμματισμού Python.

```
    def bulk_insert(self, data):
        """
        The following function is used to insert bulk data to Elasticsearch

        :param data: list of dict
        :return:

        Example:
            >>> data = [{ "title": "Solr in Action", "authors": ["trey grainger", "timothy potter"],
"summary" : "Comprehensive guide","publish_date" : "2015-12-03", "num_reviews": 18,
"publisher": "manning" }]
            >>> bulk_insert(data)
        """
        try:
            actions = [
                {
                    "_index": self.book_index,
                    "_type": self.book_doc,
                    "_id": i,
                    "_source": data[i]
                }
                for i in range(len(data))
```

```
            ]
            helpers.bulk(self.es, actions=actions)
        except Exception as e:
            print(e, flush=True)
```

6.4.3. Ανάκτηση βιβλίου - Get book document

Με την χρήση των παραπάνω λειτουργιών εισάγονται δεδομένα βιβλίων στο Elasticsearch. Σε αυτό το σημείο πρέπει να αναφερθεί πως γίνεται ανάκτηση των δεδομένων από το Elasticsearch. Η πιο βασική λειτουργία είναι η ανάκτηση ενός βιβλίου χρησιμοποιώντας το ID του. Η συγκεκριμένη λειτουργία ονομάζεται **get document** και επιτυγχάνεται με την χρήση του παρακάτω (στο παράδειγμα γίνεται κλήση για την ανάκτηση του βιβλίου με ID=2) [54]:

```
GET /bookdb_index/book_doc/2
```

Η αντίστοιχη συνάρτηση στην Python που επιστρέφει ένα βιβλίο δίνοντας σαν παράμετρο της συνάρτησης το ID του βιβλίου είναι η παρακάτω:

```
    def retrieve_book_by_id(self, book_id):
        """
        The following function is used to retrieve a book document from the elastic search using is ID
        :param book_id: book id
        :return: result document

        :Example:
            >>> elk = ElasticBookStorage()
            >>> book = elk.retrieve_book_by_id(book_id=2)
        """
        try:
            results = self.es.get_source(index=self.book_index, doc_type=self.book_doc,
id=str(book_id))
            return results
        except Exception as ex:
            print(ex, flush=True)
```

Και τα δεδομένα που επιστρέφονται από το Elasticsearch για το βιβλίο με ID=2 είναι τα παρακάτω:

```
{
  "_index" : "bookdb_index",
  "_type" : "book_doc",
  "_id" : "2",
  "_version" : 1,
  "_seq_no" : 0,
  "_primary_term" : 1,
  "found" : true,
  "_source" : {
    "title" : "Taming Text: How to Find, Organize, and Manipulate It",
    "authors" : [
      "grant ingersoll",
      "thomas morton",
      "drew farris"
    ],
    "summary" : "organize text using approaches such as full-text search, proper name recognition,
clustering, tagging, information extraction, and summarization",
```

```
    "publish_date" : "2013-01-24",
    "num_reviews" : 12,
    "publisher" : "manning"
  }
}
```

Παρατηρείται ότι τα επιστρεφόμενα δεδομένα είναι για το βιβλίο με ID=2.

6.4.4. Ανάκτηση πολλών βιβλίων - Multi Get book documents

Με την χρήση αυτής της λειτουργίας του Elasticsearch (**multi get documents**) δίνεται η δυνατότητα ανάκτησης παραπάνω από ενός βιβλίου με μία μόνο κλήση, με αποτέλεσμα να επιτυγχάνεται μικρότερος χρόνος απόκρισης του συστήματος και δεύτερον να μειώνεται το σχετικό overhead σε σύγκριση με το να υπάρχει επαναληπτική κλήση στο σύστημα Elasticsearch για καθένα από τα βιβλία που υπάρχει ανάγκη για ανάκτηση [55]. Για την χρήση της συγκεκριμένης λειτουργίας απαιτείται η χρήση των απαιτούμενων IDs στο σώμα κλήσης της λειτουργίας multi get, όπως φαίνεται και στο παρακάτω χωρίο όπου επιστρέφονται οι πληροφορίες για τα βιβλία με IDs 1 και 2.

```
GET /book_index/_mget
{
   "docs" : [
      {
         "_id" : "1"
      },
      {
         "_id" : "2"
      }
   ]
}
```

Στα πλαίσια της προγραμματιστικής υλοποίησης της εφαρμογής δημιουργήθηκε η παρακάτω συνάρτηση η οποία δέχεται σαν όρισμα μια λίστα με τα ζητούμενα IDs. Το παρακάτω χωρίο περιλαμβάνει την συνάρτηση καθώς και παράδειγμα κλήσης της συνάρτησης αυτής.

```
    def multi_get_books(self, **kwargs):
        """
        Examples:
            >>> elk = ElasticBookStorage()
            >>> results = elk.multi_get_books(ids=[19,11])
        """
            book_ids = kwargs['ids']
            book_results = self.es.mget(
                index=self.book_index,
                doc_type=self.book_doc,
            body={'ids': book_ids},
            )
            book_docs = book_results['docs']
            results = {'docs': [book['_source'] for book in book_docs]}
            return results
```

6.4.5. Λίστα Βιβλίων

Η χρήση της συγκεκριμένης λειτουργίας επιστρέφει όλα τα βιβλία που είναι καταχωρημένα στο σύστημα Elasticsearch. Η συγκεκριμένη λειτουργία ονομάζεται **match all query** και επιστρέφει όλα

τα βιβλία. Το παρακάτω αποτελεί κλήση για ανάκτηση όλων των βιβλίων που είναι καταχωρημένα [56].

```
GET book_index/_search
{
  "query": {
    "match_all": {}
  }
}
```

Η αντίστοιχη συνάρτηση στην γλώσσα προγραμματισμού Python η οποία υλοποιήθηκε στα πλαίσια της εφαρμογής και επιστρέφει όλα τα καταχωρημένα βιβλία στο Elasticsearch είναι η παρακάτω. Αξίζει να σημειωθεί ότι η συνάρτηση αυτή χρησιμοποιείται χωρίς παραμέτρους.

```
def fetch_all_docs(self):
    """This function is used to returned all stored books"""
    try:
        body = {
            "query": {
                "match_all": {}
            }
        }
        results = self.es.search(
            index=self.book_index,
            body=body,
            size=HITS_SIZE)["hits"]["hits"]
        return results
    except Exception as ex:
        print(ex, flush=True)
```

6.4.6. Διαγραφή βιβλίου - Delete document

Κάνοντας χρήση της λειτουργίας **delete document** είναι δυνατή η διαγραφή ενός βιβλίου από το Elasticsearch δηλώνοντας το ID του βιβλίου το οποίο είναι για διαγραφή. Πιο συγκεκριμένα πρέπει να δηλωθεί ο δείκτης που χρησιμοποιείται, στην περίπτωση αυτή **book_index**, καθώς και το ID του βιβλίου. Το παρακάτω αποτελεί χρήση της λειτουργίας delete document, όπου χρησιμοποιείται για την διαγραφή από το Elasticsearch του βιβλίου με ID=1 [57].

```
DELETE book_index/book_doc/1
```

Στα πλαίσια της εφαρμογής υλοποιήθηκε και η αντίστοιχη συνάρτηση στην γλώσσα προγραμματισμού Python όπου δέχεται σαν όρισμα το ID του βιβλίου προς διαγραφή από το Elasticsearch.

```
def remove_book_doc(self, book_id):
    """
    The following function is used to remove a book entry from elastic search
    using its ID
    :param book_id: book ID
    :return:

    :Example:
        >>> elk = ElasticBookStorage()
```

```
        >>> elk.remove_book_doc(book_id=2)
    """
    try:
        self.es.delete(index=self.book_index, doc_type=self.book_doc, id=str(book_id))
    except Exception as e:
        print(e, flush=True)
```

6.4.7. Διαγραφή βιβλίου βάσει όρων - Delete By Query

Πολλές φορές μπορεί να εμφανιστεί η ανάγκη για μια πιο γενικευμένη διαγραφή. Πιο συγκεκριμένα σε περιπτώσεις όπου δεν είναι γνωστό το ID ενός βιβλίου αλλά μπορεί να είναι επιθυμητή η διαγραφή βιβλίων όπου των οποίων τα δεδομένα μπορεί να πληρούν ένα συγκεκριμένο μοτίβο. Ένα παράδειγμα είναι όταν πρέπει να διαγραφούν βιβλία των οποίων ο εκδοτικός οίκος είναι ο “Wiley” ή να διαγραφούν τα βιβλία των οποίων ο τίτλος περιέχει την λέξη “python”. Η συγκεκριμένα λειτουργία ονομάζεται **delete by query** [58]. Το παρακάτω αποτελεί παράδειγμα χρήσης της λειτουργία delete by query στο εργαλείο Kibana για διαγραφή όλων των βιβλίων που στον τίτλο τους υπάρχει η λέξη “python”.

```
POST /book_index/_delete_by_query
{"query": {
   "match": {
     "title": "python"
   }}}
```

Η αντίστοιχη συνάρτηση στην γλώσσα προγραμματισμού Python δέχεται σαν παραμέτρους το πεδίο καθώς και το query για τα οποία θα βρεθούν τα προς διαγραφή βιβλία. Με αυτόν τον τρόπο η συνάρτηση αποκτά έναν πιο γενικευμένο χαρακτήρα καθώς υπάρχει δυνατότητα να αλλάξει το πεδίο καθώς και ο όρος αναζήτησης των βιβλίων.

```
def delete_by_query(self, query, fields):
    """
    This function is used to delete by query

    :param query: provided query
    :param fields: provided fields
    :return:

    Example:
        >>> delete_by_query(query="python", fields=['title'])
    """
    try:
        client = Elasticsearch(
            hosts=ELASTIC_HOSTNAME
        )
        s = Search(using=client, index=self.book_index)

        retrieved_items = s.query(
            Q("multi_match", query=query, fields=fields)
        )
        retrieved_items.delete()

        results = {'count': retrieved_items.count()}
        return results
    except Exception as ex:
```

```
        print(ex, flush=True)
```

Αφού εκτελεστούν η συνάρτηση στην Python καθώς και η λειτουργία του Kibana εμφανίζεται το παρακάτω, όπου παρατηρείται ότι τα βιβλία που διαγράφηκαν ήταν 4.

```
{
  "took" : 82,
  "timed_out" : false,
  "total" : 4,
  "deleted" : 4,
  "batches" : 1,
  "version_conflicts" : 0
}
```

6.4.8. Ανανέωση βιβλίου βάσει όρων - Update By Query

Πολλές φορές μπορεί να εμφανιστεί η ανάγκη για μια πιο γενικευμένη ανανέωση των δεδομένων των βιβλίων. Πιο συγκεκριμένα σε περιπτώσεις όπου δεν είναι γνωστό το ID ενός βιβλίου αλλά μπορεί να είναι επιθυμητή η ανανέωση των βιβλίων όπου των οποίων τα δεδομένα μπορεί να πληρούν ένα συγκεκριμένο μοτίβο [59].

Ένα παράδειγμα είναι όταν πρέπει να αλλάξουν τα δεδομένα σε βιβλία των οποίων ο εκδοτικός οίκος είναι ο “Wiley” ή στα βιβλία των οποίων ο τίτλος περιέχει την λέξη “python”. Η συγκεκριμένα λειτουργία ονομάζεται **update by query**. Το παρακάτω αποτελεί παράδειγμα χρήσης της λειτουργία update by query στο εργαλείο Kibana για ανανέωση των δεδομένων σε όλα τα βιβλία όπου ο εκδοτικός οίκος είναι “wiley” και θα αλλάξει ο εκδοτικός οίκος σε “Wiley Press”.

```
POST book_index/_update_by_query
{
  "query": {
    "multi_match": {
      "query": "wiley",
      "fields": ["publisher"]
    }
  },
  "script": {"source": "ctx._source.publisher='Wiley Press'"}}
```

Στα πλαίσια της προγραμματιστικής υλοποίησης της διπλωματικής εργασίας υλοποιήθηκε στην γλώσσα προγραμματισμού Python και δέχεται σαν ορίσματα τα παρακάτω:

- το πεδίο για το οποίο γίνεται η αναζήτηση,
- η τιμή του πεδίου για το οποίο πραγματοποιείται η αναζήτηση,
- το πεδίο για το οποίο θέλουμε να αλλάξουμε
- την τιμή και την νέα τιμή του πεδίου.

Το παρακάτω χωρίο είναι ένα παράδειγμα κλήσης της συνάρτησης με τις δοθείσες παραμέτρους και ακολουθεί και ο κώδικας της συνάρτησης.

```
update_by_query(fields="publisher", query="oreilly",field_to_update="publisher",
new_value="OnMedia")
```

```
    def update_by_query(self, **kwargs):
        """
        This function is used to update Elasticsearch entries by record

        :param kwargs: provided kwargs
```

```
        :return:

        Example:
          >>> update_by_query(fields="publisher", query="oreilly", field_to_update="publisher",
new_value="OnMedia")
        """
        try:
            client = Elasticsearch(
                hosts=ELASTIC_HOSTNAME
            )
            ubq = UpdateByQuery(
                using=client,
                index=self.book_index
            )

            search_fields = kwargs["fields"]
            query = kwargs["query"]
            field_to_update = kwargs["field_to_update"]
            new_value = kwargs["new_value"]

            update_query = ubq.query(
                "multi_match",
                query=query,
                fields=search_fields
            ).script(
                source="ctx._source.{}='{}'".format(field_to_update, new_value)
            )
            results = update_query.execute()._d_
            return results
        except Exception as ex:
            print(ex, flush=True)
```

6.5. Αναζήτηση Βιβλίων

Σε αυτή την παράγραφο αναλύεται και παρουσιάζεται η διαδικασία αναζήτησης του Elasticsearch καθώς και οι αντίστοιχες λειτουργίες οι οποίες έχουν να κάνουν με:

- την αναζήτηση ολόκληρου κειμένου,
- με αναζήτηση λέξεων σε συγκεκριμένα πεδία των documents - βιβλίων,
- με την χρήση bool queries τα οποία χρησιμοποιούν συνδυαστική αναζήτηση,
- ερωτήματα στο Elasticsearch που χρησιμοποιούν regular expressions
- fuzzy queries για συγκεκριμένα πεδία, όπου υποστηρίζεται ταίριασμα του όρου ερωτήματος στο αντίστοιχο πεδίο
- Αναζήτηση βάσει ημερομηνιών
- Αναζήτηση όρων στα αντίστοιχα πεδία
- Ακριβές ταίριασμα όρου στο αντίστοιχο πεδίο

6.5.1. Αναζήτηση πλήρους κειμένου - Full Text Query

Σε αυτή την περίπτωση η λειτουργία **full text query** χρησιμοποιείται για αναζήτηση όρου σε όλες τις ιδιότητες properties κάθε βιβλίου - document. Για παράδειγμα αν δοθεί είσοδος στην λειτουργία αυτή το ερώτημα (query) “guide” το Elasticsearch θα αναζητήσει βιβλία που είναι καταχωρημένα σε αυτό τα οποία περιέχουν την λέξη “guide ” στα properties του. Πιο συγκεκριμένα ο όρος αναζήτησης πρέπει να βρίσκεται είστε στον τίτλο, είτε στα ονόματα των συγγραφέων, είτε στην περίληψη είστε στο όνομα του εκδοτικού οίκου. Παράδειγμα κλήσης της συγκεκριμένης συνάρτησης αποτελεί το παρακάτω πεδίο. Όπως παρατηρείται δίνεται σαν παράμετρος στην συγκεκριμένη λειτουργία το ερώτημα, γίνεται αναζήτηση για τα βιβλία που φέρουν στα properties του την λέξη “guides”.

```
GET /book_index/book_doc/_search?q=guide
```

Στα πλαίσια της προγραμματιστικής υλοποίησης της εφαρμογής υλοποιήθηκε η παρακάτω συνάρτηση στην γλώσσα προγραμματισμού Python.

```
def multi_match_query(self, query):
    """
        >>> elk = ElasticBookStorage()
        >>> results = elk.basic_match_query(query="guide")
    """
    results = self.es.search(index=self.book_index, q=query, size=HITS_SIZE)["hits"]["hits"]
    return results
```

6.5.2. Αναζήτηση Βιβλίου βάσει ταιριάσματος ιδιότητας - Match Query

Κάνοντας χρήση αυτής της λειτουργίας δίνεται η δυνατότητα για αναζήτηση βιβλίων - documents ταιριάζοντας το ερώτημα που δίνεται σαν όρισμα στην λειτουργία με το αντίστοιχο property του βιβλίου που και αυτό δίνεται σαν όρισμα. Στο παρακάτω παράδειγμα χρήσης της λειτουργίας **match query** γίνεται αναζήτηση για βιβλία τα οποία περιέχουν το string “in action” στον τίτλο τους [60].

```
POST /book_index/book_doc/_search?size=1000
{
   "query": {
      "match" : {
         "title" : "in action"
      }
   }
}
```

Η αντίστοιχη συνάρτηση στην γλώσσα προγραμματισμού Python που υλοποιήθηκε δέχεται 2 παραμέτρους. Η πρώτη παράμετρος είναι το πεδίο (property) πάνω στο οποίο θέλουμε να γίνει η αναζήτηση και η δεύτερη παράμετρος είναι το query. Η παρακάτω συνάρτηση καλείται με αυτόν τον τρόπο (1ο όρισμα πεδίο, 2ο όρισμα query):

```
search_book_by_title("title", "in action")
```

```
def search_book_by_param(self, *args, _source=[]):
    """
    :Examples:
        >>> query = "in action"
        >>> elk = ElasticBookStorage()
        >>> results = elk.search_book_by_title("title", "in action")
    """
        term = args[0]
        query = args[1]

        body = {
            "query": {
                "match": {
                    "{}".format(term): query
                }
            },
```

```
            "_source": _source
        }
        results = self.es.search(index=self.book_index, body=body,
size=HITS_SIZE)["hits"]["hits"]
        return results
```

6.5.3. Book Fuzzy Matching - Fuzzy Query

Η λειτουργία **fuzzy matching / fuzzy query** εφαρμόζεται πάνω στις λειτουργίες match query και multi match query με σκοπό να καλυφθούν spelling errors. Το Elasticsearch χρησιμοποιεί τον αλγόριθμο Levenstein distance, ο οποίος υπολογίζει την διαφορά μεταξύ δύο αλφαριθμητικών βάσει το πόσοι είναι οι χαρακτήρες των 2 αυτών strings. Συνεπώς ο βαθμός ασάφειας (degree of fuzziness) στο Elasticsearch έχει να κάνει με τον υπολογισμό της διαφοράς του query που δίνεται σαν παράμετρος στην λειτουργία fuzzy matching. Παρατηρείται ότι δίνεται στο σώμα της κλήσης της ιδιότητας το ερώτημα, τα πεδία για τα οποία γίνεται η αναζήτηση σε αυτή την περίπτωση είναι ο τίτλος και η περίληψη και η παράμετρος **fuzziness: AUTO** σημαίνει ότι ενεργοποιείται ο βαθμός ασάφειας.

```
POST /book_index/book_doc/_search?size=1000
{
   "query": {
      "multi_match" : {
         "query" : "comprihensiv guide",
         "fields": ["title", "summary"],
         "fuzziness": "AUTO"
      }
   }
}
```

Το αποτέλεσμα αυτής της λειτουργίας θα επιστρέψει όλα τα βιβλία των οποίων τα πεδία τίτλος και περίληψη περιέχουν ένα αλφαριθμητικό το οποίο σύμφωνα με την απόσταση Levenstein βρίσκεται κοντά στο query που δόθηκε σαν όρισμα στην λειτουργία αυτή. Η αντίστοιχη συνάρτηση σε Python είναι η παρακάτω και σαν ορίσματα το ερώτημα και τα πεδία για τα οποία θα γίνει η fuzzy αναζήτηση.

```
    def fuzzy_queries(self, query, _source=[], **kwargs):
        """

        :Examples:
            >>> query="comprihensiv guide"
            >>> elk = ElasticBookStorage()
            >>> elk.fuzzy_queries(query, fields=["title", "summary"])
        """
        try:
            fields = kwargs["fields"]
            body = {
                "query": {
                    "multi_match": {
                        "query": query,
                        "fields": fields,
                        "fuzziness": "AUTO"
                    }
                },
                "_source": _source,
```

```
            "size": 1
        }

        results = self.es.search(index=self.book_index, body=body,
size=HITS_SIZE)["hits"]["hits"]
        return results
    except Exception as ee:
        print(ee, flush=True)
```

6.5.4. Αναζήτηση χρησιμοποιώντας κανονικές εκφράσεις - Regex Query

Υπάρχουν περιπτώσεις όπου, για την αναζήτηση βιβλίων - documents στο Elasticsearch, προκύπτει η ανάγκη χρήσης κανονικών εκφράσεων στους όρους αναζήτησης των βιβλίων. Η λειτουργία αυτή στο Elasticsearch ονομάζεται **regex query** και δίνεται η ευελιξία χρήσης κανονικών εκφράσεων οι οποίοι προσπαθούν να ταιριάξουν τα μοτίβα με τα properties των βιβλίων [61].

Πιο συγκεκριμένα το παρακάτω παράδειγμα χρήσης του regex query αναζητά βιβλία των οποίων το όνομα των συγγραφέων ξεκινά με το χαρακτήρα “t”. Παρατηρείται ότι δίνεται σαν είσοδος στο σώμα της λειτουργίας το property authors για το οποίο διεξάγεται η αναζήτηση καθώς και το regular expression pattern “t*” το οποίο προσπαθεί να ταιριάξει authors των οποίων το όνομα ξεκινά με “t”.

```
POST /book_index/book_doc/_search
{
  "query": {
    "wildcard" : {
      "authors" : "t*"
    }
  }
}
```

Η αντίστοιχη συνάρτηση στην γλώσσα προγραμματισμού Python είναι η παρακάτω και δέχεται σαν παραμέτρους εισόδου το property για το οποίο επιθυμείτε αναζήτηση καθώς και το pattern το οποίο οποίο προσπαθεί να ταιριάξει τα αποθηκευμένα βιβλία στο Elasticsearch. Η συνάρτηση καλείται ως εξής:

```
        wild_card_query(field="authors", query="t*")
```

```
def wild_card_query(self, _source=[], **kwargs):
    """
    This function is used to perform Elasticsearch wild card queries

    :param args: given arguments
    :param query: given query
    :return:

    Example:
        >>> wild_card_query(field="authors", query="t*")
    """
    try:
        field = kwargs['field']
        query = kwargs['query']

        body = {
            "query": {
                "wildcard": {
```

```
                    "{}".format(field): query
                }
            },
            "_source": _source
        }
        results = self.es.search(index=self.book_index, body=body,
size=HITS_SIZE)["hits"]["hits"]
        return results
    except Exception as ex:
        print(ex, flush=True)
```

6.5.5. Ταίριασμα Έκφρασης - Match Phrase Query

Σε αυτή την παράγραφο αναλύεται η λειτουργία ταιριάσματος έκφρασης στο σύστημα Elasticsearch η αλλιώς **match phrase query** η οποία προϋποθέτει την ύπαρξη των παραμέτρων εισόδου στις ιδιότητες των βιβλίων - documents. Σύμφωνα με την λειτουργία αυτή οι παράμετροι που δίνονται πρέπει να είναι ο ένας μετά τον άλλον αλλά υπάρχει και η παράμετρος slop η οποία υποδεικνύει "πόσο μακριά" μπορούν να βρίσκονται οι παράμετροι ταιριάσματος [62].

Πιο συγκεκριμένα στο παρακάτω παράδειγμα χρήσης της λειτουργίας match phrase query στο εργαλείο Kibana αναζητούνται βιβλία τα οποία περιέχουν στους τίτλους τους ή στο summary τους την έκφραση "search engine'. Επιπλέον στο παράδειγμα παρέχεται και η παράμετρος slop και δίνεται σε αυτή η τιμή 3 που σημαίνει ότι το αποτέλεσμα της αναζήτησης θα επιστρέψει και βιβλία στα οποία στους τίτλους τους και στο summary τους υπάρχει η λέξη "search" και μετά από 3 λέξεις από αυτήν θα υπάρχει η λέξη "engine".

```
POST /book_index/book_doc/_search
{
  "query": {
    "multi_match" : {
      "query": "search engine",
      "fields": ["title", "summary"],
      "type": "phrase",
      "slop": 3
    }
  }
}
```

Η αντίστοιχη συνάρτηση στην γλώσσα προγραμματισμού Python που υλοποιεί την λειτουργία του Elasticsearch match phrase query δέχεται σαν όρισμα την έκφραση για την οποία επιθυμείτε το ταίριασμα τα properties των βιβλίων για τα οποία θα γίνει η αναζήτηση και η τιμή της παραμέτρου slop.

```
def match_phrase_query(self, query, **kwargs):
    """
    Example:
        >>> match_phrase_query(query="search engine", fields=["title", "summary"], slop=3,
_source=[])
    """
    try:
        fields = kwargs["fields"]
        slop = kwargs["slop"]

        body = {
```

```
            "query": {
              "multi_match": {
                 "query": query,
                 "fields": fields,
                 "type": "phrase",
                 "slop": slop
              }
            },
            "_source": []
          }
          results = self.es.search(index=self.book_index, body=body,
size=HITS_SIZE)["hits"]["hits"]
          return results
        except Exception as ex:
          print(ex, flush=True)
```

6.5.6. Match Phrase Prefix Query

Σε αυτή την περίπτωση επεκτείνεται η λειτουργία της προηγούμενης λειτουργίας και πιο συγκεκριμένα χρησιμοποιώντας την λειτουργία **match phrase prefix** [63] γίνεται ταίριασμα έκφρασης στα αποθηκευμένα βιβλία με την χρήση auto complete αν κάποιος δώσει σαν παράμετρο αναζήτησης την έκφραση “search en” για το property summary των αποθηκευμένων βιβλίων η λειτουργία θα προσπαθήσει να ταιριάξει την έκφραση εισόδου με όλα τα βιβλία που ενδέχεται να την περιέχουν και επιπλέον με την χρήση της ιδιότητας auto complete αυτής της λειτουργίας ενδέχεται να επιστραφούν και βιβλία που μπορεί να περιέχουν εκφράσεις που συμπληρώνουν το query εισόδου, για παράδειγμα να επιστραφούν κείμενα που περιέχουν την έκφραση “search engine”. Επιπλέον και σε αυτή την λειτουργία υπάρχει η παράμετρος slop η οποία προσδιορίζει την σειρά των λέξεων της έκφρασης αλλά καθορίζεται μια επιπλέον παράμετρος η οποία ονομάζεται max_expansions και καθορίζει πόσες εκφράσεις θα ταιριάσουν με την χρήση της ιδιότητας auto complete στην έκφραση εισόδου. Το παρακάτω παράδειγμα αναζητά αποθηκευμένα βιβλία τα οποία ενδέχεται να περιέχουν την έκφραση “search en” στο summary τους η παράμετρος slop παίρνει τιμή 3 και η παράμετρος max_expansions παίρνει την τιμή 10.

```
POST /book_index/book_doc/_search
{
  "query": {
    "match_phrase_prefix" : {
      "summary": {
        "query": "search en",
        "slop": 3,
        "max_expansions": 10
      }
    }
  }
}
```

Κάνοντας χρήση της παραπάνω λειτουργίας στο εργαλείο Kibana παρατηρούμε ότι επιστρέφονται τα παρακάτω βιβλία τα οποία περιέχουν τις λέξεις “search” και “engine” οι οποίες προέκυψαν από το slop =3 (απόσταση 3 μεταξύ των λέξεων της έκφρασης) και την ιδιότητα auto complete που ταίριαξε την έκφραση “search en” με την έκφραση “search engine” .

```
[{
      "summary": "Comprehensive guide to implementing a scalable search engine using Apache
Solr"}},
   { {
      "summary": "A distibuted real-time search and analytics engine",
    }]
```

Η αντίστοιχη συνάρτηση στην γλώσσα προγραμματισμού Python είναι η παρακάτω και δέχεται σαν παράμετρο την έκφραση αναζήτησης καθώς και το slop, το οποίο παίρνει default τιμή 3.

```
    def match_phrase_prefix(self, query, slop, max_expansions=10, _source=[]):
        """
        Example:
            >>> match_phrase_prefix(query="search en", slop=3)
        """
        try:
            body = {
                "query": {
                    "match_phrase_prefix": {
                        "summary": {
                            "query": query,
                            "slop": slop,
                            "max_expansions": max_expansions
                        }
                    }
                },
                "_source": _source
            }
            results = self.es.search(index=self.book_index, body=body,
size=HITS_SIZE)["hits"]["hits"]
            return results
        except Exception as ex:
            print(ex, flush=True)
```

6.5.7. Αναζήτηση όρου - Term Query

Στην περίπτωση της λειτουργίας αναζήτησης όρου, **term query** functionality [64] του Elasticsearch, είναι επιθυμητή η αναζήτηση βιβλίων - documents όπου υπάρχει ακριβές ταίριασμα του όρου αναζήτησης στο πεδίο αναζήτησης. Για παράδειγμα term queries μπορούν να χρησιμοποιηθούν σε περιπτώσεις όπου θέλουμε το αποτέλεσμα της αναζήτησης να επιστρέψει όλα τα βιβλία - documents όπου ο εκδοτικός τους οίκος είναι “manning”. Σε αυτή την διαδικασία απαιτείται η χρήση λειτουργίας ακριβούς ταιριάσματος του όρου αναζήτησης με το πεδίο αναζήτησης. Το παρακάτω αποτελεί παράδειγμα χρήσης αυτής της λειτουργίας όπου δίνονται σαν είσοδος το πεδίο (publisher) αναζήτησης καθώς και ο όρος προς αναζήτησης (manning).

```
POST /book_index/book_doc/_search
{
  "query": {
    "term": {
      "publisher": "manning"
    }
  }
}
```

Η αντίστοιχη συνάρτηση στην γλώσσα προγραμματισμού Python είναι η παρακάτω, η οποία δέχεται σαν είσοδο το πεδίο αναζήτησης και το όρο αναζήτησης. Παράδειγμα χρήσης της συνάρτησης είναι το παρακάτω:

```
            term_query(field="publisher", term="manning")
```

```
    def term_query(self, _source=[], **kwargs):
```

```
        """
        Example:
            >>> term_query(field="publisher", term="manning")
        """
        try:
            field = kwargs["field"]
            term = kwargs["term"]

            if isinstance(term, list):
                term_or_terms = "terms"
            else:
                term_or_terms = "term"

            body = {
                "query": {
                    "{}".format(term_or_terms): {
                        "{}".format(field): term
                    }
                },
                "_source": _source
            }
            results = self.es.search(index=self.book_index, body=body,
size=HITS_SIZE)["hits"]["hits"]
            return results
        except Exception as ex:
            print(ex, flush=True)
```

6.5.8. Αναζήτηση όρων - Terms Query

Η λειτουργία αναζήτησης όρων, **terms query** [65] σύμφωνα με την ορολογία του Elasticsearch αποτελεί μια επέκταση της προηγούμενη λειτουργίας καθώς αναζητά βιβλία - documents που έχουν στο πεδίο τους που καθορίζεται σαν παράμετρος της λειτουργίας περισσότερους από έναν όρους. Πιο συγκεκριμένα γίνεται αναζήτηση όλων των βιβλίων όπου οι εκδοτικοί οίκοι είναι είτε “oreilly” είτε “manning”. Όπως παρατηρείται η αναζήτηση στην λειτουργία terms query δέχεται σαν είσοδο παραπάνω από έναν όρους, εν αντιθέσει με την λειτουργία term query. Το παρακάτω αποτελεί παράδειγμα χρήσης της λειτουργίας αυτής όπου αναζητούνται βιβλία των οποίων ο εκδοτικός οίκος είναι “wiley” η “oreilly”.

```
POST /book_index/book_doc/_search
{
   "query": {
      "terms" : {
         "publisher": ["oreilly", "manning"]
      }
   }
}
```

Η αντίστοιχη συνάρτηση στην γλώσσα προγραμματισμού Python είναι η ίδια με την συνάρτηση που χρησιμοποιείται για τα term queries, το μόνο που αλλάζει είναι η κλήση όπου καλείται ως εξής για την περίπτωση των term queries

```
    term_query(field="publisher", term=["manning","oreilly"])
```

6.5.9. Range Query

Η συγκεκριμένη λειτουργία χρησιμοποιείται για την επιστροφή αποτελεσμάτων τα οποία βρίσκονται εντός προκαθορισμένου εύρους το οποίο δίνεται σαν παράμετρο της συνάρτησης. Η λειτουργία αυτή χρησιμοποιείται για δεδομένα τα οποία μπορεί να είναι ημερομηνίες οι αριθμοί. Στην περίπτωση της εφαρμογής τα δεδομένα τα οποία μπορούν να δοθούν σαν είσοδος στην εφαρμογή είναι ο αριθμός κριτικών του βιβλίου αλλά και οι ημερομηνίες έκδοσης κάθε βιβλίου. Το παρακάτω παράδειγμα επιστρέφει όλα τα βιβλία που εκδόθηκαν μεταξύ “2016-01-01” και “2016-12-31”. Παρατηρούμε ότι δίνεται σαν είσοδος το πεδίο πάνω στο οποίο πραγματοποιείται η αναζήτηση καθώς και το εύρος των χρονικών διαστημάτων, μεγαλύτερο ή ίσο του (gte) “2016-01-01” και μικρότερο ή ίσο του (lte) “2016-12-31”

```
POST /book_index/book_doc/_search
{
  "query": {
    "range" : {
      "publish_date": {
        "gte": "2016-01-01",
        "lte": "2016-12-31"
      }
    }
  }
}
```

Η αντίστοιχη συνάρτηση στην γλώσσα προγραμματισμού Python δέχεται σαν είσοδο το εύρος των διαστημάτων για τα οποία το Elasticsearch θα επιστρέψει τα αποθηκευμένα βιβλία - documents τα οποία έχουν δημοσιευτεί εντός του εύρους που δίνεται σαν παράμετρος.

```
    def range_query(self, **kwargs):
        """
        The following function is used to fetch elastic search records based on range queries

        :param kwargs: provided kwargs
        :return: elastic search response

        Examples:
            >>> range_query(field='publish_date', range={"gte": "2015-01-01","lte": "2015-12-31"})
        """
        field = kwargs['field']
        ranges = kwargs['range']

        try:
            body = {
                "query": {
                    "range": {
                        "{}".format(field): ranges
                    }
                },
            }
            results = self.es.search(index=self.book_index, body=body,
size=HITS_SIZE)["hits"]["hits"]
            return results
        except Exception as ex:
            print(ex, flush=True)
```

6.5.10. Bool Query

Η συγκεκριμένη λειτουργία, bool query [66] σύμφωνα με την ορολογία του Elasticsearch χρησιμοποιείται για την αναζήτηση βιβλίων των οποίων τα πεδία ικανοποιούν μια σειρά από bool συνδυασμούς άλλων queries. Τα bool queries βασίζονται σε έναν οι περισσότερους δεσμευμένους τύπους, οι τύποι αυτοί είναι οι παρακάτω:

- **must**: Ο όρος που χρησιμοποιείται με τον τύπο must πρέπει υποχρεωτικά να βρίσκεται στα αποθηκευμένα βιβλία. Είναι κάτι αντίστοιχο με το AND.
- **should**: Ο όρος που χρησιμοποιείται με τον τύπο should θα έπρεπε να υπάρχει στα αποθηκευμένα βιβλία, πιο συγκεκριμένα το should υποδηλώνει την μη υποχρεωτική ύπαρξη του όρου. Είναι κάτι αντίστοιχο με το OR.
- **must_not**: Ο όρος που χρησιμοποιείται με τον τύπο must_not υποδηλώνει ότι ο όρος που χρησιμοποιείται με το must_not δεν πρέπει να υπάρχει στα επιστρεφόμενα βιβλία - documents. Είναι κάτι αντίστοιχο με το NOT.

Η συγκεκριμένα λειτουργία συνεπώς χρησιμοποιείται για πιο σύνθετες αναζητήσεις όπου τα επιστρεφόμενα βιβλία ικανοποιούν τους όρους που έχουν τεθεί με την χρήση των must, should, must_not. Για παράδειγμα επιθυμούμε τα βιβλία τα οποία ο τίτλος τους περιέχει την λέξη “Elasticsearch” ή την λέξη “Solr”, στους συγγραφείς πρέπει να είναι ο “clinton gormely” και να μην βρίσκεται στους συγγραφείς ο “radu george”. Αυτό είναι ίσο με το παρακάτω:

```
POST /book_index/book_doc/_search
{
  "query": {
    "bool": {
      "must": {
        "bool" : {
          "should": [
            { "match": { "title": "Elasticsearch" }},
            { "match": { "title": "Solr" }}
          ],
          "must": { "match": { "authors": "clinton gormely" }}
        }
      },
      "must_not": { "match": {"authors": "radu gheorge" }}
    }
  }
}
```

Παρατηρούμε ότι στο should πηγαίνει ο περιορισμός του τίτλου στο must ότι ο συγγραφέας πρέπει να είναι ο “clinton gormely” και στο must_not ο περιορισμός που αποκλείει από τους συγγραφείς τον “radu george”.
Η αντίστοιχη συνάρτηση στην γλώσσα προγραμματισμού Python, δέχεται σαν όρισμα τις λίστες που θα συμπληρώσουν το should, must, must_not. Οπότε το παραπάνω υλοποιείται με την χρήση της παρακάτω συνάρτησης και κλίνεται όπως ακολουθείται.

```
>>> should=[["title", "Elasticsearch"], ["title", "Solr"]],
>>> must=[["authors", "clinton gormely"]],
>>> must_not=[["authors", "radu george"]]
>>> query_combination(should, must, must_not)
```

```
def query_combination(self, **kwargs):
    """
    This function performs Combined bool queries
```

```
        :param kwargs: provided kwargs
        :return:

        Example:
            >>> should=[["title", "Elasticsearch"], ["title", "Solr"]],
            >>> must=[["authors", "clinton gormely"]],
            >>> must_not=[["authors", "radu george"]]
            >>> query_combination(should, must, must_not)

        """
        try:
            client = Elasticsearch(
                hosts=ELASTIC_HOSTNAME
            )

            s = Search(using=client, index=self.book_index)

            q = Q('bool',
                  must=[
                      Q({"multi_match": {"query": "{}".format(m[1]), "fields": ["{}".format(m[0])]}})
                      for m in kwargs["must"]] if "must" in kwargs.keys() else [],

                  should=[Q({"multi_match": {"query": "{}".format(m[1]), "fields":
["{}".format(m[0])]}})
                          for m in kwargs["should"]] if "should" in kwargs.keys() else [],

                  must_not=[Q({"multi_match": {"query": "{}".format(m[1]), "fields":
["{}".format(m[0])]}})
                            for m in kwargs["must_not"]] if "must_not" in kwargs.keys() else [],
                  minimum_should_match=1
                  )
            response = s.query(q).execute()["hits"]["hits"]
            return response.__dict__['_l_']
```

6.6. Aggregations

Το Aggregation Framework του εργαλείου Elasticsearch δίνει την δυνατότητα παροχής aggregations σε δεδομένα τα οποία επιστρέφουν οι λειτουργίες του Elasticsearch. Οι παρακάτω παράγραφοι αναλύουν τα metric aggregations πάνω στα κριτικές των βιβλίων, range aggregations που επιστρέφουν τα εύρη των κριτικών των βιβλίων καθώς και τα filter aggregations τα οποία ομαδοποιούν δεδομένα που προκύπτουν από διαδικασία αναζήτησης σύμφωνα με τον όρο αναζήτησης που δόθηκε.

6.6.1. Metric Aggregations

Η λειτουργία **metric aggregations** [67] του Elasticsearch εφαρμόζεται επί των πλείστων σε αριθμητικά δεδομένα, στην περίπτωση των βιβλίων - documents η δοκιμή αυτής της λειτουργίας θα γίνει πάνω στις κριτικές του βιβλίου καθώς αποτελεί το μοναδικό πεδίο που είναι αριθμητικό. Η λειτουργία metric aggregations βρίσκει των μέσο όρο των κριτικών του βιβλίου, την μέγιστη τιμή των κριτικών του βιβλίου, την ελάχιστη τιμή των reviews καθώς και το άθροισμα των τιμών αυτώ. Το παρακάτω αποτελεί χρήση της λειτουργίας αυτής.

```
POST /book_index/book_doc/_search?size=0
{
   "aggs" : {
      "stats_reviews" : { "stats" : { "field" : "num_reviews" } }
   }
```

```
}
```

Το αποτέλεσμα της παραπάνω λειτουργίας είναι τα στατιστικά για τα reviews των αποθηκευμένων βιβλίων:

```
{
 "aggregations" : {
  "stats_reviews" : {
   "count" : 54,
   "min" : 12.0,
   "max" : 562.0,
   "avg" : 125.37037037037037,
   "sum" : 6770.0
  }
}
}
```

Η παρακάτω συνάρτηση υλοποιεί την λειτουργία metric aggregations και έχει υλοποιηθεί στην γλώσσα προγραμματισμού Python. Στην συνάρτηση δίνεται σαν όρισμα το πεδίο το οποίο είναι συνέχεια το num_reviews, σε αυτή την περίπτωση είναι το πεδίο στο οποίο θα εφαρμοστεί η λειτουργία καθώς και η μετρική υπάρχει η δυνατότητα η συνάρτηση να επιστρέψει μία μόνο μετρική αν οριστεί σε περίπτωση που θέλουμε να επιστραφούν όλες δίνεται σαν μετρική το stats.

```
def metric_aggregations(self, **kwargs):
    """
    The following function is used to return metric aggregations

    :param kwargs: provided kwargs
    :return: aggregated results

    Examples:
        >>> results = metric_aggregations(field='num_reviews', metric='min')
    """

    field = kwargs['field']
    metric = kwargs['metric']

    aggregation_name = "{}_{}".format(metric, field)

    try:
        body = {
            "aggs": {
                "{}".format(aggregation_name): {
                    "{}".format(metric): {"field": field}
                }
            }
        }

        results = self.es.search(index=self.book_index, body=body, size=0)["aggregations"]
        return results
    except Exception as ex:
        print(ex, flush=True)
```

6.6.2. Filter Aggregations

Η λειτουργία metric aggregations εφαρμόζει τα στατιστικά πάνω σε όλα τα αποθηκευμένα βιβλία - documents που υπάρχουν στο Elasticsearch, στην περίπτωση της λειτουργίας **filter aggregations** [68] οι στατιστικές αυτές, δηλαδή average/max/min/sum των αριθμών των reviews των βιβλίων, εφαρμόζονται μόνο στα καταχωρημένα documents τα οποία ικανοποιούν τον όρο που δίνεται σαν παράμετρος στην λειτουργία αυτήν. Πιο συγκεκριμένα το παρακάτω παράδειγμα εφαρμόζει τις στατιστικές αυτές στα βιβλία των οποίων ο εκδοτικός οίκος είναι “wiley”. Παρατηρούμε ότι δίνεται σαν είσοδος της λειτουργίας το πεδίο στο οποίο θα εφαρμοστεί η αναζήτηση καθώς και ο όρος αναζήτησης, σε αυτή την περίπτωση ότι ο publisher είναι “wiley”.

```
POST /book_index/book_doc/_search?size=0
{
   "aggs" : {
      "aggregated_data" : {
         "filter" : { "term": { "publisher": "wiley" } },
         "aggs" : {
            "stats_num_reviews" : { "stats": { "field" : "num_reviews" } }
         }
      }
   }
}
```

Το αποτέλεσμα της λειτουργίας filter aggregations είναι το ακόλουθο, παρατηρούμε ότι επιστρέφει και τον αριθμό των βιβλίων που ικανοποιούν την συνθήκη “publisher=wiley” καθώς και τα αντίστοιχα στατιστικά.

```
{
 "aggregations" : {
  "aggregated_data" : {
   "doc_count" : 7,
   "stats_num_reviews" : {
    "count" : 7,
    "min" : 12.0,
    "max" : 88.0,
    "avg" : 54.57142857142857,
    "sum" : 382.0
   }
  }
 }
}
```

Η αντίστοιχη συνάρτηση στην γλώσσα προγραμματισμού Python είναι η παρακάτω και δέχεται σαν όρισμα τον όρο αναζήτησης καθώς και το πεδίο για το οποίο θα γίνει η αναζήτηση το πεδίο για το οποίο θα υπολογιστούν τα στατιστικά, αυτό το πεδίο είναι πάντα το num_reviews, καθώς και η μετρική η οποία παίρνει τιμή stats για τον υπολογισμό όλων των στατιστικών. Παράδειγμα χρήσης της συνάρτησης αυτής είναι το παρακάτω χωρίο

```
filter_aggregations(term='publisher', query='wiley',field='num_reviews',metric='stats')
```

```
    def filter_aggregations(self, **kwargs):
        """
        This function is used to aggregate data from elastic search using term query
```

```
        :param kwargs: provided kwargs
        :return: aggregated bucket data
        """

        term = kwargs['term']
        query = kwargs['query']
        field = kwargs['field']
        metric = kwargs['metric']

        try:

            body = {
                "aggs": {
                    "aggregated_data": {
                        "filter": {"term": {"{}".format(term): query}},
                        "aggs": {
                            "{}_{}".format(metric, field): {"{}".format(metric): {"field": field}}
                        }
                    }
                }
            }

            results = self.es.search(index=self.book_index, body=body, size=0)["aggregations"]
            return results
        except Exception as ex:
            print(ex, flush=True)
```

6.6.3. Range Aggregations

Η λειτουργία **range aggregation** [69] είναι εντελώς διαφορετική από τις υπόλοιπες λειτουργίες που αναφέρθηκαν πριν, metric & filter aggregation. Στόχος αυτής της λειτουργίας είναι ο καθορισμός groups που περιέχουν documents - βιβλία που βρίσκονται ενός συγκεκριμένου εύρους το οποίο δίνεται σαν παράμετρος στην λειτουργία range aggregation. Το παρακάτω παράδειγμα ομαδοποιεί τα documents σε 3 groups. Το πρώτο γκρουπ περιέχει τα βιβλία με αριθμό κριτικών λιγότερο από 100, το δεύτερο γκρουπ περιέχει τα βιβλία των οποίων ο αριθμός κριτικών είναι μεγαλύτερος από 100 αλλά μικρότερος από 200 και το τρίτο γκρουπ περιέχει τα βιβλία των οποίων ο αριθμός κριτικών είναι μεγαλύτερος από 200. Παράδειγμα χρήσης της λειτουργίας αυτής είναι το παρακάτω:

```
POST /book_index/book_doc/_search?size=0
{
   "aggs" : {
      "review_ranges" : {
         "range" : {
            "field" : "num_reviews",
            "ranges" : [
               { "to" : 100.0 },
               { "from" : 100.0, "to" : 200.0 },
               { "from" : 200.0 }
            ]
         }
      }
   }
}
```

Το αποτέλεσμα της συγκεκριμένης λειτουργίας είναι το παρακάτω παρατηρούμε την δημιουργία των 3 γκρουπ καθώς και το πλήθος των βιβλίων ανά γκρουπ.


```
{
  "aggregations" : {
    "review_ranges" : {
      "buckets" : [
        {
          "key" : "*-100.0",
          "to" : 100.0,
          "doc_count" : 40
        },
        {
          "key" : "100.0-200.0",
          "from" : 100.0,
          "to" : 200.0,
          "doc_count" : 3
        },
        {
          "key" : "200.0-*",
          "from" : 200.0,
          "doc_count" : 11
        }
      ]
    }
  }
}
```


Η αντίστοιχη συνάρτηση στην γλώσσα προγραμματισμού Python είναι η παρακάτω δέχεται σαν είσοδο τα εύρη στα οποία θα δημιουργηθούν τα γκρουπ.

```
reviews_range_aggregation(ranges=[{ "to" : 100 },{ "from" : 200 },{ "from" : 100, "to" : 200 }])
```

```
  def reviews_range_aggregation(self, **kwargs):
      """
      Examples:
          >>> elk = ElasticBookStorage()
          >>> res = elk.reviews_range_aggregation(ranges=[{ "to" : 100 },{ "from" : 200 },{ "from"
: 100, "to" : 200 }])
      """
      try:
```

```
        ranges = kwargs['ranges']

        body = {
          "aggs": {
            "review_ranges": {
              "range": {
                "field": "num_reviews",
                "ranges": ranges
              }
            }
          }
        }
        results = self.es.search(index=self.book_index, body=body, size=0)["aggregations"]
        return results
      except Exception as ex:
        print(ex, flush=True)
```

6.7. Υλοποίηση Query Builder

Αφού έχουν δημιουργηθεί οι παραπάνω συναρτήσεις στην γλώσσα προγραμματισμού Python οι οποίες υλοποιούν τις λειτουργίες του συστήματος Elasticsearch, οι οποίες πραγματοποιούν την δημιουργία του αντίστοιχου δείκτη που θα αποθηκεύονται τα βιβλία, που λαμβάνουν ενδείξεις από το Elasticsearch cluster σχετικά με monitoring δεδομένα από αυτό, λειτουργίες αναφορικά με την αναζήτηση βιβλίων βάσει όρων, εκφράσεων, IDs καθώς και aggregations των αποθηκευμένων βιβλίων, ήρθε η ώρα για την δημιουργία ενός πιο γενικευμένου scripts το οποίο δέχεται παραμέτρους από ένα API για το ποια από αυτές τις λειτουργίες θα πραγματοποιήσει και ποιες θα είναι η παραμέτρους εισόδου. Δηλαδή αυτό το Python script αποτελεί μια “all in one” συνάρτηση η οποία ανάλογα με τις παραμέτρους που θα της δώσουν θα καλέσει την αντίστοιχη λειτουργία του συστήματος Elasticsearch.

Η πιο βασική παράμετρος της συνάρτησης αυτής, η οποία ονομάζεται **QueryBuilder** είναι η παράμετρος action η οποία ανάλογα με την τιμή της θα καλέσει και την αντίστοιχη συνάρτηση που υλοποιεί μια συγκεκριμένη λειτουργία του Elasticsearch, επίσης δέχεται και άλλες παραμέτρους οι οποίες έχουν να κάνουν με τις επιμέρους παραμέτρους που δέχεται η αντίστοιχη λειτουργία του Elasticsearch.

Οι τιμές που μπορεί να πάρει η παράμετρος **action** αυτής είναι οι παρακάτω:

- **append_book**: Όταν η παράμετρος action ισούται με την τιμή “append_book” αυτό σημαίνει την εισαγωγή ενός καινούργιου βιβλίου στο Elasticsearch. Οι επιμέρους παράμετροι που δέχεται η συνάρτηση QueryBuilder και έχουν να κάνουν με την εισαγωγή του βιβλίου είναι οι εξής:
  - title: τίτλος του βιβλίου
  - authors: οι συγγραφείς του βιβλίου
  - summary: summary του βιβλίου
  - publisher: ο εκδότης του βιβλίου
  - num_reviews: πλήθος κριτικών για το βιβλίο
  - publish_date: ημερομηνία έκδοσης
- **retrieve_book_by_id**: Όταν η παράμετρος action ισούται με την τιμή “retrieve_book_by_id” τότε καλείται η λειτουργία η οποία επιστρέφει ένα βιβλίου βάσει του book_id που δίνεται σαν παράμετρος. Οι επιμέρους παράμετροι όταν action=”retrieve_book_by_id”:
  - book_id: Το ID του βιβλίου το οποίο θέλουμε.
- **remove_book_by_id**: Διαγραφή βιβλίου όταν action=”remove_book_by_id”, η επιμέρους παράμετρος είναι η παρακάτω:
  - book_id: Το ID του βιβλίου προς διαγραφή
- **search_book_by_parameter**: Καλείται η λειτουργία για αναζήτηση βιβλίου βάσει ταιριάσματος ιδιότητας. Οι επιμέρους παράμετροι είναι:
  - field: Η ιδιότητα του βιβλίου (property) για την οποία θα γίνει αναζήτηση

    - query: Ο όρος αναζήτησης.
- **fuzzy_queries**: Καλείται η λειτουργία fuzzy query του Elasticsearch και οι επιμέρους παράμετροι είναι:
    - query: Ο όρος αναζήτησης.
    - fields: τα πεδία, παραπάνω από ένα, για τα οποία θα διεξαχθεί αναζήτηση
- **wild_card_query**: Καλείται η λειτουργία regex query του Elasticsearch και οι επιμέρους παράμετροι είναι:
    - query: Το regular expression pattern που θα διερευνηθεί στα αποθηκευμένα βιβλία.
    - field: Το attribute του βιβλίου για το οποίο θα διεξαχθεί αναζήτηση.
- **match_phrase_query**: Καλείται η λειτουργία match_phrase_query και οι επιμέρους παράμετροι είναι:
    - query: Η έκφραση η οποία δίνεται σαν είσοδος
    - slop: Τιμή παραμέτρου slop (default value slop=3)
    - fields: Τα πεδία για τα οποία προσπαθεί να ταιριαστεί η έκφραση η οποία δίνεται σαν παράμετρος
- **match_phrase_prefix**: Καλείται η λειτουργία match_phrase_prefix, σε σχέση με την παραπάνω ιδιότητα είναι ακριβώς η ίδια το μόνο επιπλέον είναι ότι εφαρμόζεται και auto complete για ταίριασμα με επιπλέον βιβλία - documents, και οι επιμέρους παράμετροι είναι οι παρακάτω:
    - query: η έκφραση η οποία δίνεται ως είσοδος
    - slop: Τιμή παραμέτρου slop (default value slop=3)
    - fields: Τα πεδία για τα οποία προσπαθεί να ταιριαστεί η έκφραση η οποία δίνεται σαν παράμετρος
- **term_query**: Καλείται η λειτουργία term_query για ταίριασμα όρου/όρων και οι επιμέρους παράμετροι είναι οι παρακάτω:
    - term: ο όρος για τον οποίο πραγματοποιείται η αναζήτηση βιβλίων.
    - field: το πεδίο του βιβλίου που εφαρμόζεται ο παραπάνω όρος
- **delete_by_query:** Καλείται η λειτουργία delete_by_query για διαγραφή βιβλίων που ταιριάζουν στον όρο που δίνεται στις επιμέρους παραμέτρους, οι οποίες είναι:
    - query: ο όρος αναζήτησης
    - fields: τα πεδία του βιβλία στα οποία εφαρμόζεται ο όρος αναζήτησης
- **update_by_query**: Καλείται η λειτουργία update_by_query για ανανέωση των δεδομένων βιβλίων των οποίων τα πεδία ταιριάζουν στον όρο που δίνεται στις επιμέρους παραμέτρους, οι οποίες είναι οι παρακάτω:
    - query: ο όρος αναζήτησης
    - fields: τα πεδία του βιβλία στα οποία εφαρμόζεται ο όρος αναζήτησης
    - field_to_update: το πεδίο του βιβλίου προς ανανέωση
    - new_value: η νέα τιμή του πεδίου προς ανανέωση
- **bool_query**: Κλήση της λειτουργίας boolean_query του Elasticsearch και οι επιμέρους παράμετροι είναι οι παρακάτω:
    - should: λίστα όρων που η παρουσία τους στα αποθηκευμένα βιβλία δεν είναι αναγκαστική, συνθήκη OR
    - must: λίστα όρων των οποίων η παρουσία στα αποθηκευμένα βιβλία είναι αναγκαστική
    - must_not: λίστα όρων που δεν πρέπει να υπάρχουν στα αποθηκευμένα βιβλία
- **range_query**: Κλήση της λειτουργίας range_query του Elasticsearch για αναζήτηση βιβλίων που βρίσκονται εντός των ευρών που δίνονται σαν επιμέρους παράμετροι:
    - field: πεδίο στο οποίο θα εφαρμοστεί το εύρος (publish_date),
    - range: εύρος ημερομηνιών
- **metric_aggregations**: Κλήση της λειτουργίας metric aggregations όπου υπολογίζονται στατιστικά για όλα τα documents - βιβλία. Επιμέρους παράμετροι:
    - field: Πεδίο που θα εφαρμοστούν τα στατιστικά, αυτό το πεδίο θα είναι πάντα το num_reviews καθώς είναι το μοναδικό αριθμητικό πεδίο των βιβλίων

    - metric: Οι μετρικές που θα εφαρμοστούν, οι τιμές είναι min, max, avg, sum, stats το οποίο είναι όλες οι στατιστικές
- **filter_aggregations**: Κλήση της λειτουργίας filter_aggregations όπου υπολογίζονται τα στατιστικά για τα βιβλία που πληρούν τον όρο που δίνεται σαν επιμέρους παράμετροι:
    - term: το πεδίο που πρέπει να πληροί τον όρο που δίνεται σαν παράμετρος
    - query: το query που δίνεται σαν παράμετρος
    - field: Πεδίο που θα εφαρμοστούν τα στατιστικά, αυτό το πεδίο θα είναι πάντα το num_reviews καθώς είναι το μοναδικό αριθμητικό πεδίο των βιβλίων
    - metric: Οι μετρικές που θα εφαρμοστούν, οι τιμές είναι min, max, avg, sum, stats το οποίο είναι όλες οι στατιστικές
- **reviews_range_aggregations** Κλήση της λειτουργίας reviews range aggregations, όπου υπολογίζονται τα γκρουπ των αποθηκευμένων βιβλίων ανάλογα με τα εύρη που δίνονται σαν επιμέρους παράμετροι:
    - ranges: εύρος τιμών num_reviews
- **get_cluster_health**: Κλήση της λειτουργίας get_cluster_health.
- **get_cluster_Stats:** Κλήση της λειτουργίας get_cluster_stats.
- **multi_get:** Κλήση της λειτουργίας multi_get_query για την λήψη παραπάνω από ενός βιβλίου χρησιμοποιώντας τα IDs τους. Οι επιμέρους παράμετροι είναι:
    - ids: λίστα με τα book IDs για αναζήτηση.

Το παρακάτω καλεί την λειτουργία για metric aggregations του Elasticsearch (action=metric_aggregation), παρατηρούμε ότι δηλώνεται το πεδίο για το οποίο θα υπολογιστούν τα στατιστικά και η τιμή της μετρικής ισούται με την τιμή stats έτσι ώστε να υπολογιστούν όλα τα στατιστικά (avg, min, max, count, sum):

```
POST /ask/storage/ HTTP/1.1
Host: 127.0.0.1:5000
Content-Type: application/json

{
        "action": "metric_aggregations",
        "field": "num_reviews",
        "metric": "stats"
}
```

Άλλο ένα παράδειγμα της χρήσης του Query Builder είναι η παρακάτω. Σε αυτό το παράδειγμα δηλώνεται η τιμή της παραμέτρου action ίση με “term_query”, πράγμα που σημαίνει την κλήση της λειτουργίας term_query του Elasticsearch η οποία προσπαθεί να εντοπίσει τα αποθηκευμένα βιβλία στο Elasticsearch στα οποία εμφανίζεται ο όρος που δίνεται σαν παράμετρος στο σύστημα. Οι επιμέρους παράμετροι που δίνονται είναι το όνομα του πεδίου του βιβλίου για το οποίο πραγματοποιείται η αναζήτηση καθώς και ο όρος ο οποίος πρέπει να υπάρχει αυτούσιος στο πεδίο του βιβλίου.

```
POST /ask/storage/ HTTP/1.1
Host: 127.0.0.1:5000
Content-Type: application/json

{
        "action": "term_query",
        "field": "publisher",
        "term": "manning"
}
```

Άλλο ένα παράδειγμα είναι η κλήση του multi get query του Elasticsearch, για την αναζήτηση παραπάνω από ένα βιβλίων δίνοντας τα IDs τους σαν λίστα παραμέτρων.

```
POST /ask/storage/ HTTP/1.1
Host: 127.0.0.1:5000
Content-Type: application/json

{
        "action":"multi_get",
        "ids":[11,19]
}
```

Η κλήση της λειτουργίας delete_by_query γίνεται μέσω QueryBuilder με τον παρακάτω τρόπο, παρατηρείται ότι οι επιμέρους παράμετροι που θέτονται είναι το πεδίο των βιβλίων για τα οποία θα πραγματοποιηθεί η αναζήτηση και το query είναι ο όρος για τον οποίο θα διεξαχθεί η αναζήτηση.

```
POST /ask/storage/ HTTP/1.1
Host: 127.0.0.1:5000
Content-Type: application/json

{
        "action": "delete_by_query",
        "fields": ["title"],
        "query": "python"
}
```

Η λειτουργία για την λήψη δεδομένων σχετικών για την “υγεία” του Elasticsearch cluster γίνεται με τον παρακάτω τρόπο μέσω του QueryBuilder.

```
POST /ask/storage/ HTTP/1.1
Host: 127.0.0.1:5000
Content-Type: application/json

{
        "action":"get_cluster_health"
}
```

Η λειτουργία filter_aggregations γίνεται με τον παρακάτω τρόπο, όπως παρατηρείται οι επιπλέον παράμετροι που χρησιμοποιούνται για την λειτουργία αυτή είναι ο όρος για τον οποία θα πραγματοποιηθεί η αναζήτηση, το πεδίο των βιβλίων το οποίο πρέπει να ταιριάξει, το πεδίο στο οποίο θα εφαρμοστούν τα στατιστικά καθώς και το είδος των στατιστικών, επιλέγεται η χρήση της τιμής stats για εξαγωγή όλων των στατιστικών.

```
POST /ask/storage/ HTTP/1.1
Host: 127.0.0.1:5000
Content-Type: application/json

{
        "action": "filter_aggregations",
        "term": "publisher",
        "query": "oreilly",
        "metric": "stats",
        "field": "num_reviews"
}
```

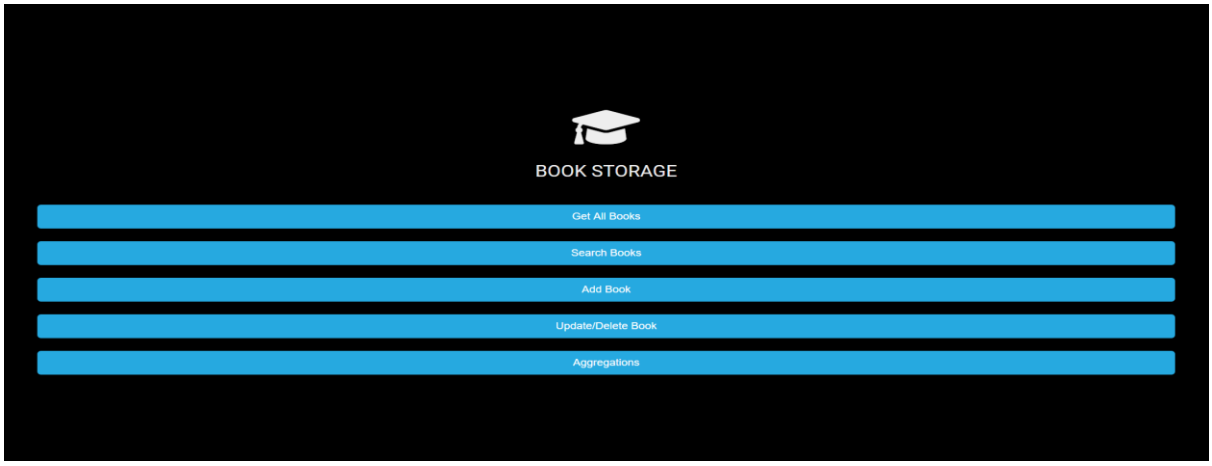
6.8. User Interface

Στα πλαίσια της διπλωματικής εργασίας υλοποιήθηκε και ένα User Interface το οποίο καλεί το API του Query Builder, το συγκεκριμένο User Interface υλοποιήθηκε με την χρήση της γλώσσας προγραμματισμού JavaScript και HTML. Πιο συγκεκριμένα έγινε χρήση της εργαλειοθήκης JQuery για την ασύγχρονη κλήση του Query Builder API και το response που είναι η έξοδος των λειτουργιών του Elasticsearch, δηλαδή τα βιβλία που πληρούν τις παραμέτρους εισόδου, παρουσιάζεται σε HTML tables. Οι παρακάτω οθόνες παρουσιάζουν τις λειτουργίες του User Interface με τις οποίες μπορεί να αλληλοεπιδράσει ο χρήστης πιο εύκολα.

Πιο συγκεκριμένα επιλέχθηκαν συγκεκριμένες οθόνες του front end συστήματος (User Interface), οι οποίες θα παρουσιάζουν την αρχική οθόνη, τις επιμέρους οθόνες οι οποίες διαχωρίζουν τις λειτουργίες καθώς σε άλλη οθόνη βρίσκονται οι λειτουργίες οι οποίες είναι σχετικές με αναζήτηση, σε άλλη οθόνη βρίσκονται οι λειτουργίες οι οποίες είναι σχετικές με updates και διαγραφές και σε άλλη οθόνη βρίσκονται οι aggregation λειτουργίες. Συνεπώς επιλέχθηκαν από κάθε οθόνη να παρουσιαστεί μια βασική λειτουργία.

6.8.1. Οθόνη Εισόδου

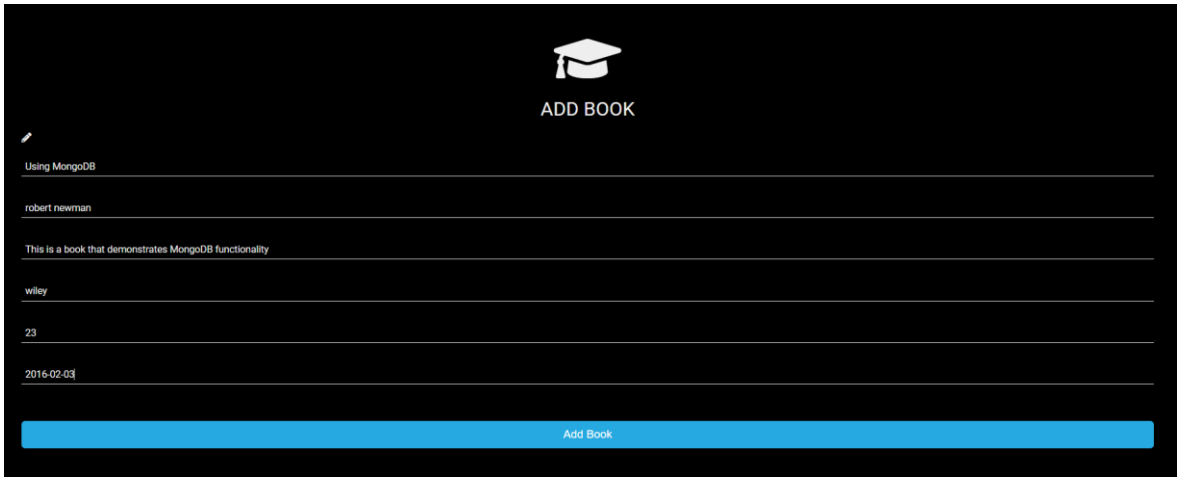
Η οθόνη εισόδου αποτελεί την εναρκτήρια σελίδα του User Interface και περιέχει buttons τα οποία εξυπηρετούν στην πλοήγηση σε άλλες οθόνες του front end συστήματος που καλούν συγκεκριμένες λειτουργίες. Η παρακάτω οθόνη δίνει την δυνατότητα πλοήγησης για την δημιουργία, αναζήτηση, update / remove βιβλίου καθώς και την λειτουργία aggregations.




6.8.2. Εισαγωγή βιβλίου

Για την εισαγωγή βιβλίου στο σύστημα πρέπει να επιλεχθεί το button “Add Book” της αρχικής οθόνης με την επιλογή του συγκεκριμένου ο χρήστης μεταβαίνει στην παρακάτω οθόνη, όπου πρέπει να εισαχθούν ο τίτλος, οι συγγραφείς, η περίληψη ο εκδότης ο αριθμός των reviews καθώς και η

ημερομηνία έκδοσης του βιβλίου και με το πάτημα του κουμπιού “ Add Book ” πραγματοποιείται η καταχώρηση. Η παρακάτω οθόνη παρουσιάζει αυτή την λειτουργία του User Interface.

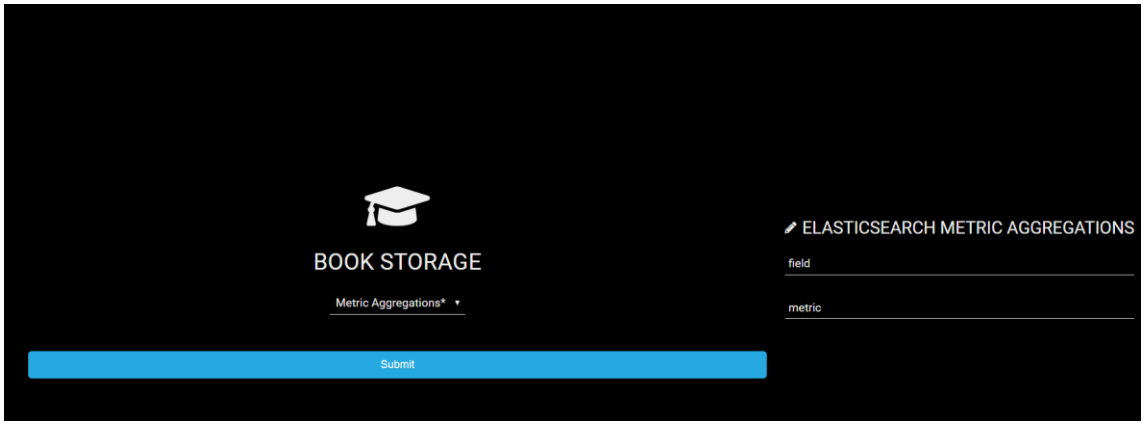



6.8.3. Λίστα Καταχωρημένων Βιβλίων.

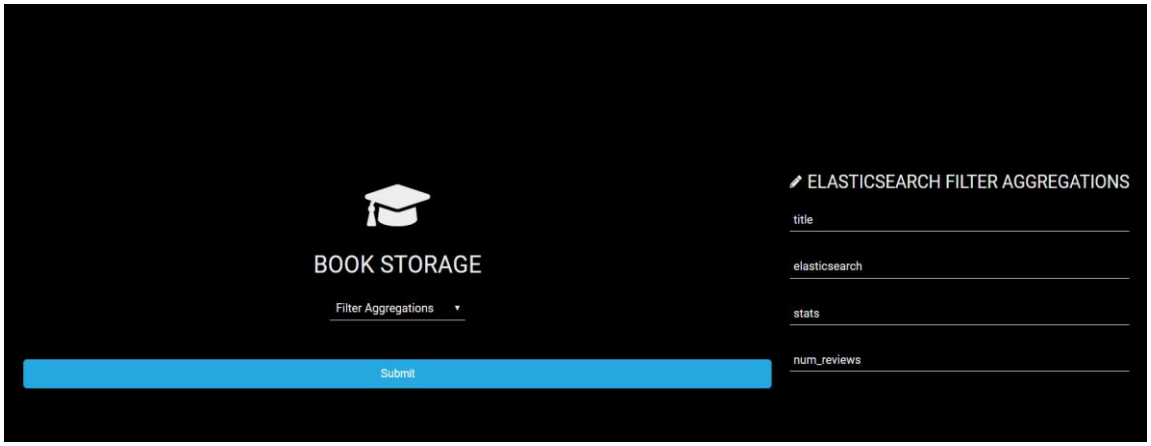
Με την επιλογή του button “Get All books” ο χρήστης μπορεί να μεταβεί στην οθόνη με τα καταχωρημένα βιβλία στο σύστημα. Η οθόνη αυτή αποτελείται από paginated HTML πίνακες που περιέχουν την πληροφορία των βιβλίων επίσης οι παρακάτω πίνακες δίνουν την δυνατότητα αναζήτησης. Η παρακάτω εικόνα είναι αντιπροσωπευτική της λειτουργίας αυτής:


6.8.4. Aggregations Page

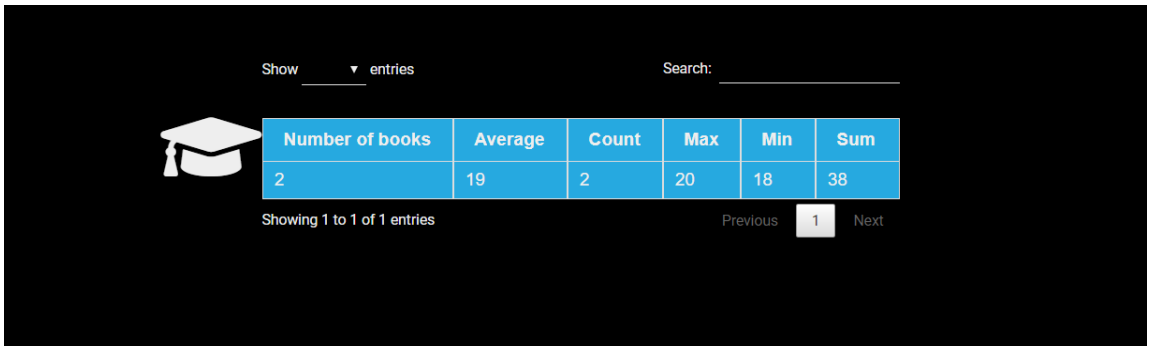
Αυτή η οθόνη περιέχει την λειτουργία των aggregations, όπου ο χρήστης αφού μεταβεί σε αυτή έχει την δυνατότητα να επιλέξει την κλήση των λειτουργιών metric aggregations, filter aggregations και range aggregations.




Από το drop down menu ο χρήστης μπορεί να επιλέξει το είδος του aggregation, έστω ότι ο χρήστης επιλέγει το filter aggregations, για στατιστικά σχετικά με τον αριθμό των reviews για τα βιβλία που στο τίτλο τους περιέχουν την λέξη “Elasticsearch”.




Με το πάτημα του submit button τα παρακάτω αποτελέσματα επιστρέφουν στην οθόνη αποτελεσμάτων:

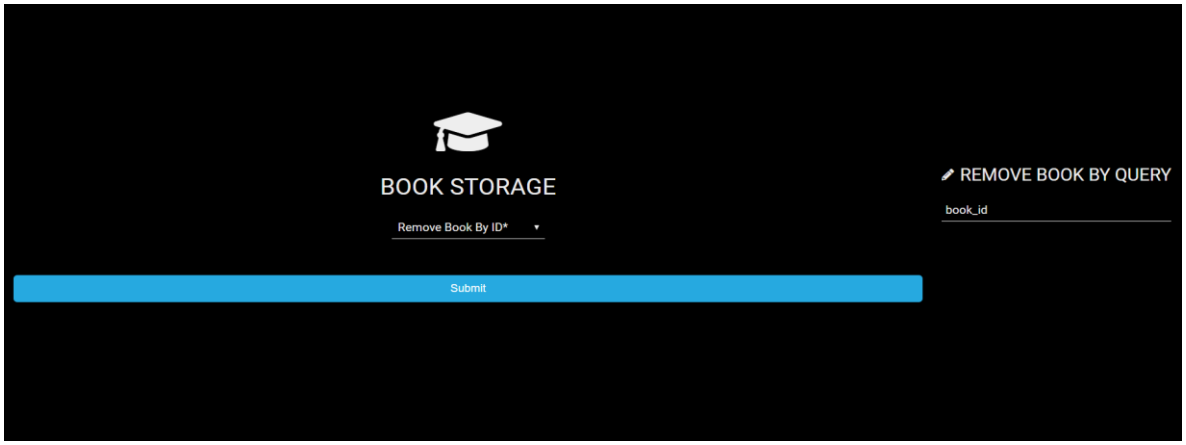



Όπως παρατηρείται βρέθηκαν 2 βιβλία που πληρούν τον περιορισμό να βρίσκεται η λέξη “Elasticsearch” στον τίτλο τους και ο μέσος όρος των reviews τους είναι 19, ο μέγιστος είναι 20 ο μικρότερος 18 και το άθροισμα των reviews τους είναι 38.

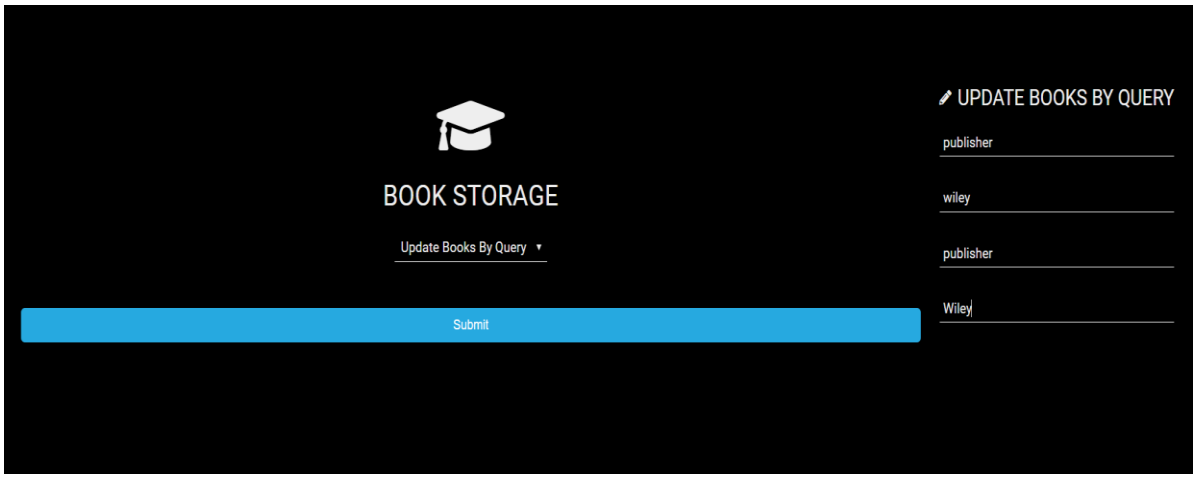
6.8.5. Update - Delete Page

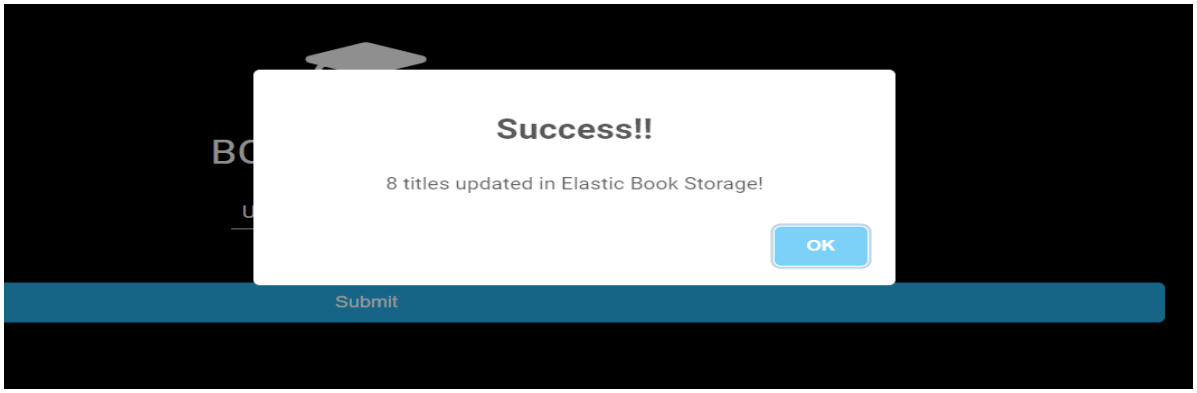
Σε αυτό το σημείο αναφέρεται η λειτουργία του User Interface ανανέωσης και διαγραφής των βιβλίων που είναι καταχωρημένων στο σύστημα. Η παρακάτω οθόνη περιέχει ένα drop down menu από το οποίο κάποιος μπορεί να επιλέξει να διαγράψει ένα βιβλίο χρησιμοποιώντας το ID του, την διαγραφή βιβλίων βάσει συγκεκριμένων όρων καθώς και την ανανέωση δεδομένων των βιβλίων

βάσει συγκεκριμένων όρων παραμέτρων που θέτει ο χρήστης στην σελίδα που παρουσιάζεται παρακάτω.




Έστω ότι ο χρήστης επιλέγει την λειτουργία update books by query και στόχος είναι η ανανέωση των βιβλίων που ικανοποιούν την συνθήκη publisher=”wiley” και η νέα τιμή της παραμέτρου αυτών των βιβλίων να είναι “Wiley”. Πιο συγκεκριμένα η παρακάτω οθόνη παρουσιάζει αυτή την λειτουργία του User Interface, και με το πάτημα του Submit εκτελείται:



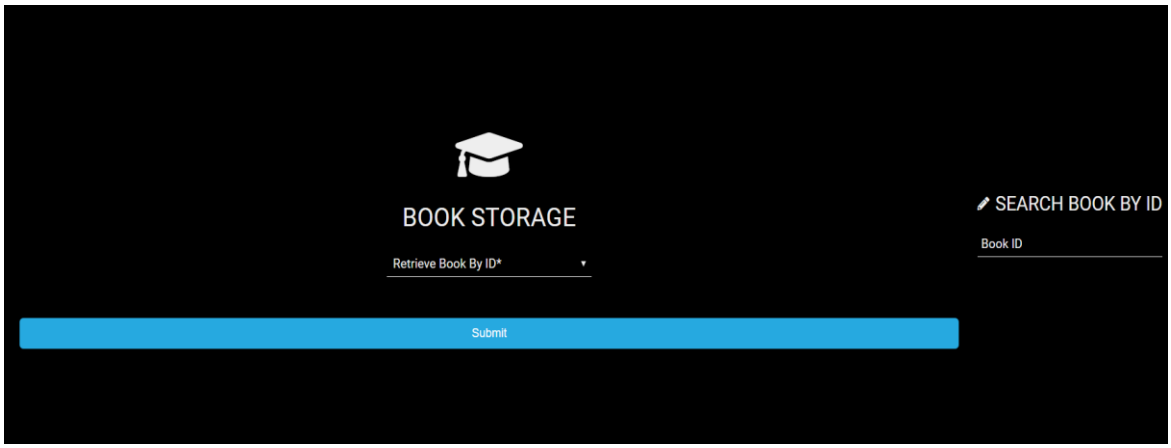




6.8.6. Οθόνη Αναζήτησης Βιβλίων

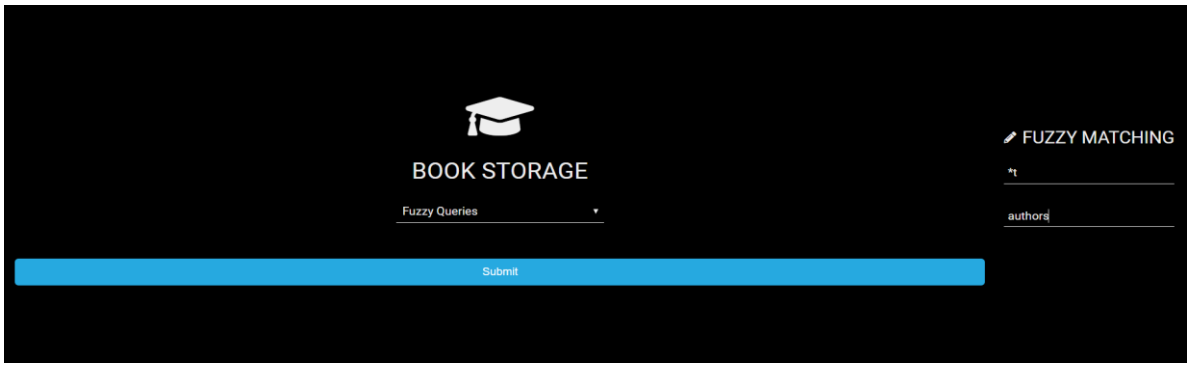
Σε αυτό το σημείο θα αναφερθεί η οθόνη αναζήτησης βιβλίων, ο χρήστης μπορεί να μεταβεί σε αυτήν την οθόνη ακολουθώντας το “Search book” button και ο χρήστης οδηγείται σε ένα drop down menu όπου μπορεί να επιλέξει να εκτελέσει συγκεκριμένες λειτουργίες αναζήτησης όπως για παράδειγμα:

- Retrieve Book By ID
- Retrieve Multiple Books Using their IDs

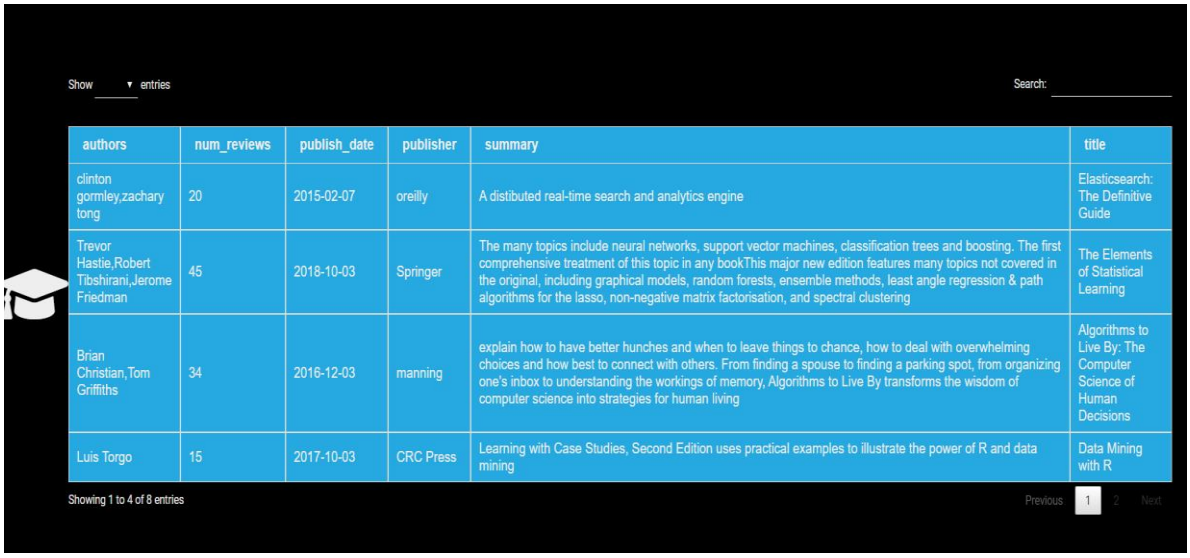
- Search Books By Parameter
- Fuzzy Queries
- WildCard / Regex Query
- Match Phrase Query
- Match Phrase Prefix Query
- Term Query
- Range Query
- Bool Query




Έστω ότι ο χρήστης επιλέξει την αναζήτηση των καταχωρημένων βιβλίων με την χρήση του widlcard query, δηλαδή queries χρησιμοποιώντας regular expressions. Για παράδειγμα επιθυμείται η αναζήτηση βιβλίων των οποίων τα ονόματα των συγγραφέων αρχίζουν από “t”, οπότε δίνεται σαν είσοδο στην αντίστοιχη φόρμα το regular expression “t*” καθώς δηλώνεται σαν attribute αναζήτησης το πεδίο authors.




Με το πάτημα του Submit button ο χρήστης μεταφέρεται στην ακόλουθη σελίδα:




7. Επίλογος

Κατά την διάρκεια αυτής της διπλωματικής εξετάστηκε το εργαλείο Elasticsearch σαν μια εναλλακτική NoSQL βάσης δεδομένων. Πιο συγκεκριμένα το Elasticsearch επιλέχθηκε λόγω της φύσης των δεδομένων που πραγματεύεται η εφαρμογή που ήταν αποτέλεσμα της διπλωματικής. Αυτά

τα δεδομένα είναι κειμενικά συνεπώς η χρήση μιας NoSQL βάσης δεδομένων η οποία εκτός από την αποθήκευση documents - βιβλίων επιτρέπει την λειτουργίες όπως αναζήτησης όρων με μη δομημένο αλλά και με δομημένο τρόπο, την χρήση aggregation καθώς και πολλών τρόπων αναζήτησης, οι οποίο παρουσιάστηκαν εκτενώς στο κεφάλαιο 6.

Επιπλέον πλεονεκτήματα του Elasticsearch είναι ότι αποτελεί ένα εργαλείο κατανεμημένων συστημάτων κάτι που σημαίνει ότι παρέχει δυνατότητες διαβάθμισης με σκοπό την υποστήριξη όλο και περισσότερων δεδομένων. Το εργαλείο Kibana το οποίο αποτελεί κομμάτι του ELK stack αποτελεί ένα open source front end σύστημα το οποία χρησιμοποιείται για την αλληλεπίδραση με το Elasticsearch και την χρήση των λειτουργιών του με σκοπό την ανάκτηση δεδομένων από αυτό. Η εφαρμογή η οποία υλοποιήθηκε στο πλαίσιο της διπλωματικής κάνοντας χρήση της γλώσσας προγραμματισμού Python και της αντίστοιχης βιβλιοθήκης του Elasticsearch Μας επέτρεψε την υλοποίηση ενός επιπλέον συστήματος για την διαχείριση βιβλίων και των επιμέρους δεδομένων του, βασισμένο στις κύριες λειτουργίες του Elasticsearch. Η εφαρμογή περιέχει 2 επίπεδα (layers) το storage layer και το presentation layer. Όπου στο storage layer αποθηκεύονται οι πληροφορίες των βιβλίων και χρησιμοποιούνται οι αποθηκευτικές λειτουργίες του Elasticsearch και το presentation layer της εφαρμογής χρησιμοποιείται για την χρήση των λειτουργιών του Elasticsearch με σκοπό την λήψη αναζητήσεων και αναλύσεων από τον τελικό χρήστη μέσω των APIs της εφαρμογής καθώς και από το User Interface.

Οι μελλοντικοί στόχοι της εργασίας αυτής είναι η γενίκευση της εφαρμογής για παραπάνω από ένα είδος documents. Αυτή την στιγμή υποστηρίζονται μόνο documents βιβλίων, οπότε στόχος είναι η δημιουργία και άλλων δεικτών οι οποίοι υποστηρίζουν και άλλα είδη δεδομένων και διαφορετικών δεδομένων. Επίσης ένα από τα επόμενα βήματα της παρούσας εργασίας είναι η σύνδεση της εφαρμογής, με την χρήση connectors και με άλλες βάσεις δεδομένων οι οποίες μπορεί να είναι σχεσιακές βάσεις δεδομένων αλλά και NoSQL και οι connectors που αναφέρθηκαν πριν να μεταφέρουν αυτόματα την αποθηκευμένη λειτουργία από τις βάσεις αυτές στο Elasticsearch. Με αυτόν τον τρόπο η εφαρμογή θα υποστηρίζει την μεταφορά δεδομένων από δομημένες πηγές. Επιπλέον θα μπορούσε να γίνει και η χρήση του εργαλείου LogStash, αποτελεί κομμάτι του ELK stack. Πιο συγκεκριμένα αποτελεί έναν ingestion μηχανισμό δεδομένων στο Elasticsearch και μπορεί να χρησιμοποιηθεί για την διοχέτευση αδόμητης πληροφορίας στο Elasticsearch, όπου αδόμητη πληροφορία μπορεί να αναφερθούν πολλά αρχεία τα οποία δεν εμφανίζουν σταθερή δομή αναμεταξύ τους.

Βιβλιογραφία

[1] C. Brooks, "What is SQL?," Business New Daily, [Online]. Available: https://www.businessnewsdaily.com/5804-what-is-sql.html.

[2] Wikipedia, "Relational Databases," wikipedia.org, [Online]. Available: https://en.wikipedia.org/wiki/Relational_database.

[3] C. Taylor, "Structured vs Unstructured Data," Datamation, [Online]. Available: https://www.datamation.com/big-data/structured-vs-unstructured-data.html.
[4] M. Taylor, "NoSQL Databases - Key value Store," Core View Systems, [Online]. Available: https://coreviewsystems.com/nosql-databases-key-value-store/.
[5] R. A. Team, "NoSQL Column Family Databas," [Online]. Available: https://www.rcvacademy.com/hbase/nosql-column-family-database/.
[6] R. Covington, "Document Databases," [Online]. Available: https://www.alachisoft.com/nosdb/document-databases.html.
[7] F. Vazquez, "Graph Databases. What's the Big Deal?," Medium, [Online]. Available: https://towardsdatascience.com/graph-databases-whats-the-big-deal-ec310b1bc0ed.
[8] G. Vaish, Getting started with NoSQL, Packt Publishing Ltd , 2013.
[9] M. R. Rafal Kuc, ElasticSearch Server, Packt, 2014.
[10] D. O. M. d. R. Ork de Rooij, "ThemeStreams: visualizing the stream of themes discussed in politics," *SIGIR '13: Proceedings of the 36th international ACM SIGIR conference on Research and development in information retrieval*, p. 1077–1078, 2013.
[11] A. B. M. B. Bouke Huurnink, "AVResearcher: Exploring Audiovisual Metadata," 2013.
[12] J. K. A. S. Douglas Grosvenor, "Kongress: a search and data mining application for U.S. congressional voting and Twitter data," *SIGSPATIAL'13: Proceedings of the 21st ACM SIGSPATIAL International Conference on Advances in Geographic Information Systems*, pp. 560 - 563, 2013.
[13] M. Chan, "Big Data in the Cloud: Why Cloud Computing is the Answer to Your Big Data Initiatives," 2018. [Online]. Available: https://www.thorntech.com/2018/09/big-data-in-the-cloud/.
[14] Wikipedia, "Schema-agnostic databases," [Online]. Available: https://en.wikipedia.org/wiki/Schema-agnostic_databases.
[15] M. Hernan, "NoSQL Tutorial: Learn NoSQL Features, Types, What is, Advantages," 2019. [Online]. Available: https://www.guru99.com/nosql-tutorial.html.
[16] "NoSQL Definition," 2013. [Online]. Available: https://techterms.com/definition/nosql.
[17] ElasticSearch.org, "What is Elasticsearch?," [Online]. Available: https://www.elastic.co/what-is/elasticsearch.
[18] Elastic.co, " An Introduction to the ELK stack," [Online]. Available: http://www.elastic.co/webinars/introduction-elk-stack.
[19] "DB-Engines. Db-engines ranking of search engines," [Online]. Available: http://db--engines.com/en/ranking/search+engine.
[20] L. Harris, "Introduction to Apache Lucene," 2018. [Online]. Available: https://www.baeldung.com/lucene.
[21] . . Kurt Hurtadoand, "Going beyond the needle in a haystack: Elasticsearch and the elk stack," [Online]. Available: https://www.speakerd.s3.amazonaws.com/presentations/70d4390b68504e02b4a40a1aa3754532/strata15-elk-stack-needle- haystack.pdf. [Accessed 2019-07-02].
[22] B. Hundley, "Optimizing Elasticsearch: How Many Shards per Index?," 2015. [Online]. Available: https://qbox.io/blog/optimizing-elasticsearch-how-many-shards-per-index.
[23] Wikipedia, "Representational state transfer," [Online]. Available: https://en.wikipedia.org/wiki/Representational_state_transfer.
[24] Wikipedia, "Application programming interface," [Online]. Available: https://en.wikipedia.org/wiki/Application_programming_interface.
[25] A. Mohan, "Introduction to Analysis and analyzers in Elasticsearch," 2017. [Online]. Available: https://medium.com/elasticsearch/introduction-to-analysis-and-analyzers-in-elasticsearch-4cf24d49ddab.
[26] Elastic.co, "Text Analysis," [Online]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/current/analysis.html.

[27] Elastic.co, "ElasticSearch Search APIs," [Online]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/current/search.html.
[28] Wikipedia, "Model–view–controller," [Online]. Available: https://en.wikipedia.org/wiki/Model%E2%80%93view%E2%80%93controller.
[29] T. Points, "MVC Framework - Controllers," [Online]. Available: https://www.tutorialspoint.com/mvc_framework/mvc_framework_controllers.htm.
[30] T. Points, "MVC Framework - Models," [Online]. Available: https://www.tutorialspoint.com/mvc_framework/mvc_framework_models.htm.
[31] T. Points, "MVC Framework - Views," [Online]. Available: tutorialspoint.com/mvc_framework/mvc_framework_views.htm.
[32] Wikipedia, "Python (programming language)," [Online]. Available: https://en.wikipedia.org/wiki/Python_(programming_language).
[33] Docker, "Docker for Begineers," [Online]. Available: https://docker-curriculum.com/.
[34] Docker, "Docker Use Cases," [Online]. Available: https://www.docker.com/use-cases.
[35] Docker, "Docker Architecture," [Online]. Available: https://docs.docker.com/get-started/overview/.
[36] Docker, "Docker Images," [Online]. Available: https://docs.docker.com/engine/reference/commandline/images/.
[37] Docker, "What is Docker Container," [Online]. Available: https://www.docker.com/resources/what-container.
[38] Docker, "About Docker Registry," [Online]. Available: https://docs.docker.com/registry/introduction/.
[39] Docker, "How Docker Services Work," [Online]. Available: https://docs.docker.com/engine/swarm/how-swarm-mode-works/services/.
[40] Docker, "Overview of docker compose," [Online]. Available: https://docs.docker.com/compose/.
[41] Docker, "Docker Swarm Overview," [Online]. Available: https://docs.docker.com/engine/swarm/.
[42] E. Novoseltseva, "Top 10 Benefits of Docker," [Online]. Available: https://dzone.com/articles/top-10-benefits-of-using-docker.
[43] Docker, "Install Docker on Windows," [Online]. Available: https://docs.docker.com/docker-for-windows/install/.
[44] B. Hogan, "How To Install and Use Docker on Ubuntu 18.04," [Online]. Available: https://www.digitalocean.com/community/tutorials/how-to-install-and-use-docker-on-ubuntu-18-04.
[45] J. Fitzpatric, "How to Install Python on Windows," [Online]. Available: https://www.howtogeek.com/197947/how-to-install-python-on-windows/.
[46] A. Sivji, "Elasticsearch + Kibana using Docker Compose," [Online]. Available: https://alysivji.github.io/elasticsearch-kibana-with-docker-compose.html.
[47] Python, "Python Elasticsearch Client," [Online]. Available: https://elasticsearch-py.readthedocs.io/en/master/.
[48] Elastic.co, "Cluster APIs," [Online]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/current/cluster.html.
[49] Elastic.co, "Cluster Health API," [Online]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/current/cluster-health.html.
[50] Elastic.co, "Cluster Stats API," [Online]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/current/cluster-stats.html.
[51] Elastic.co, "Create Index API," [Online]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/current/indices-create-index.html.
[52] Elastic.co, "Insert document to elasticsearch," [Online]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/current/docs-index_.html.

[53] Elastic.co, "Bulk API," [Online]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/current/docs-bulk.html.
[54] Elastic.co, "GET API," [Online]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/current/docs-get.html.
[55] Elastic.co, "Multi get api," [Online]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/current/docs-multi-get.html.
[56] Elastic.co, "Match All Query," [Online]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-match-all-query.html.
[57] Elastic.co, "Delete API," [Online]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/current/docs-delete.html.
[58] Elastic.co, "Delete by Query API," [Online]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/current/docs-delete-by-query.html.
[59] Elastic.co, "Update By Query," [Online]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/current/docs-update-by-query.html.
[60] Elastic.co, "Match Query," [Online]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-match-query.html.
[61] Elastic.co, "Regex Query," [Online]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-regexp-query.html.
[62] Elastic.co, "Match Phrase Query," [Online]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-match-query-phrase.html.
[63] Elastic.co, "Match Phrase Prefix Query," [Online]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-match-query-phrase-prefix.html.
[64] Elastic.co, "Term Query," [Online]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-term-query.html.
[65] Elastic.co, "Terms Query," [Online]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-terms-query.html.
[66] Elastic.co, "Bool Query," [Online]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/current/query-dsl-bool-query.html.
[67] Elastic.co, "Metric Aggregations," [Online]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-metrics.html.
[68] Elastic.co, "Filter Aggregations," [Online]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-bucket-filter-aggregation.html.
[69] Elastic.co, "Range Aggregation," [Online]. Available: https://www.elastic.co/guide/en/elasticsearch/reference/current/search-aggregations-bucket-range-aggregation.html.
[70] "NoSQL," En.wikipedia.org, [Online]. Available: https://en.wikipedia.org/wiki/NoSQL.
[71] K. D. Foote, "A Brief History of Non-Relational Databases," Dataversity, [Online]. Available: https://www.dataversity.net/a-brief-history-of-non-relational-databases/.
[72] Kaggle, "CMU Book Summary Dataset," [Online]. Available: https://www.kaggle.com/ymaricar/cmu-book-summary-dataset.