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

Εφαρμογή «Έξυπνης» Αναζήτησης στο Twitter

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ιωάννης Ιάσων Κ. Σκουρογιάννης

Επιβλέπων : Θεοδώρα Βαρβαρίγου
Καθηγήτρια Ε.Μ.Π.

Αθήνα, Μάρτιος 2020

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

Εφαρμογή «Έξυπνης» Αναζήτησης στο Twitter

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Ιωάννης Ιάσων Κ. Σκουρογιάννης

Επιβλέπων : Θεοδώρα Βαρβαρίγου
Καθηγήτρια Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την 13η Μαρτίου 2020.

.....	.....	.....
Θ. Βαρβαρίγου	Σ. Παπαβασιλείου	Ε. Βαρβαρίγος
Καθηγήτρια Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.

Αθήνα, Μάρτιος 2020

.....
Ιωάννης Ιάσων Κ. Σκουρογιάννης

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

Περίληψη

Τα μέσα κοινωνικής δικτύωσης προσφέρουν στους χρήστες τους τη δυνατότητα να εκφράζονται και να ανταλλάζουν δημόσια απόψεις, αναρτώντας περιεχόμενο ποικίλης μορφής. Το Twitter αποτελεί ένα από τα πιο δημοφιλή μέσα, που επιτρέπει την ανάρτηση σύντομων κειμένων, εικόνων και βίντεο. Λόγω της ευρείας χρήσης του από το κοινό, το Twitter μπορεί να χρησιμοποιηθεί ως μια μεγάλη πηγή πληροφοριών, σχετικών με θέματα επιστημονικού ενδιαφέροντος. Για την ανάκτηση των πληροφοριών αυτών, είναι αναγκαία η δημιουργία μιας εφαρμογής για σύνθετη αναζήτηση στο Twitter με βάση τους όρους που μας ενδιαφέρουν.

Σε μία υπηρεσιοστρεφή αρχιτεκτονική, μία ολοκληρωμένη διαδικτυακή εφαρμογή υλοποιείται ως ένα σύνολο αυτόνομων υπηρεσιών (web services) που βρίσκονται στον ίδιο ή διαφορετικό διακομιστή, εκτελώντας ξεχωριστές λειτουργίες η καθεμία. Οι επιμέρους υπηρεσίες επικοινωνούν μεταξύ τους ανταλλάσσοντας μηνύματα μέσω του πρωτοκόλλου HTTP. Η αλληλεπίδραση των χρηστών με την εφαρμογή πραγματοποιείται μέσω της διεπαφής χρήστη, η οποία παρουσιάζεται με τη μορφή ιστοσελίδας μέσω ενός προγράμματος περιήγησης.

Στόχος της παρούσας εργασίας είναι η δημιουργία μιας διαδικτυακής εφαρμογής για σύνθετη αναζήτηση δημοσιεύσεων από το Twitter με βάση λέξεις-κλειδιά, που είτε εισάγονται από τους χρήστες από μια γραμμή αναζήτησης, είτε επιλέγονται από έναν κατάλογο με προκαθορισμένες έννοιες που διαβάζονται από ένα αρχείο οντολογίας και στη συνέχεια επαυξάνονται αυτόματα με συνώνυμους όρους. Επίσης, μπορεί να χρησιμοποιηθεί κάποιος συνδυασμός αυτών των δύο μεθόδων. Η αναζήτηση και η συλλογή των δημοσιεύσεων πραγματοποιείται από τις υπηρεσίες που εκτελούνται στον διακομιστή.

Λέξεις – Κλειδιά: Μέσα κοινωνικής δικτύωσης, Twitter, σύνθετη αναζήτηση, διαδικτυακή υπηρεσία, διεπαφή χρήστη, αναζήτηση δημοσιεύσεων, οντολογία, νήματα, προγραμματισμός, υπηρεσιοστρεφής αρχιτεκτονική, σημασιολογικός ιστός

Abstract

Social media give their users the opportunity to express themselves and discuss publicly, by posting a wide range of content. Twitter is one of the most popular social media, because it enables users to post short texts, images and videos. Because of its wide public use, Twitter can be utilised as a great source of information about issues of scientific interest. To retrieve this information, it is necessary to create an application for advanced search on Twitter, based on the terms that we are interested in.

Within a service-oriented architecture, an integrated web application is implemented as several independent web services running on the same or different server, each one of which performs a special task. The services communicate with each other by sending and receiving messages through the HTTP protocol. Users interact with the application through the user interface, which is displayed as a normal site on a web browser.

This diploma thesis aims to create a web application for advanced keyword-based search of Twitter posts. These keywords are either entered in a search box, or selected from a list of predefined terms, read from an ontology file, and are then automatically extended with synonymous terms. A combination of these two methods can also be used. Search and retrieval of tweets is performed by the services running on the web server.

Keywords: Social media, Twitter, advanced search, web service, user interface, tweet search, ontology, threads, programming, service-oriented architecture, semantic web

Ευχαριστίες

Πρώτα από όλα, θα ήθελα να ευχαριστήσω την επιβλέπουσα καθηγήτρια Θεοδώρα Βαρβαρίγου, που μου ανέθεσε αυτή την εργασία, δίνοντάς μου τη δυνατότητα να ασχοληθώ με ένα πολύ ενδιαφέρον αντικείμενο, τις εφαρμογές διαδικτύου.

Επίσης, ευχαριστώ θερμά τους συναδέλφους Στάθη Καραναστάση και Θύμιο Χονδρογιάννη, για τον χρόνο που μου αφιέρωσαν, τη βοήθεια, τις οδηγίες και κατευθύνσεις που μου έδιναν σε όλα τα στάδια εκπόνησης της εργασίας.

Ευχαριστώ πολύ την Κωνσταντίνα για τη βοήθειά της, καθώς και για τα υπέροχα χρόνια που μου χάρισε.

Το μεγαλύτερο ευχαριστώ, όμως, το οφείλω προπαντός στην οικογένειά μου, για την υπομονή και τη στήριξή τους κατά τη διάρκεια των σπουδών μου.

Ιωάννης Ιάσων Κ. Σκουρογιάννης
Αθήνα, Μάρτιος 2020

Πίνακας περιεχομένων

1 Εισαγωγή 15
1.1 Μέσα κοινωνικής δικτύωσης 15
1.1.1 Twitter 15
1.2 Γενικά για την εργασία 16
1.2.1 Δομή της εργασίας 16
2 Σχετικές τεχνολογίες 19
2.1 Περιβάλλον προγραμματισμού 19
2.1.1 Το μοντέλο Πελάτη-Διακομιστή 19
2.1.2 Το πρωτόκολλο HTTP 20
2.1.3 Υπηρεσιοστρεφής Αρχιτεκτονική 21
2.1.3.1 Διαδικτυακές Υπηρεσίες 21
2.1.3.2 RESTful Διαδικτυακές Υπηρεσίες 21
2.1.4 Διακομιστές ιστού 22
2.1.4.1 Ο διακομιστής Apache Tomcat 22
2.2 Τμήματα λογισμικού 22
2.2.1 Λογισμικό πελάτη – client-side 22
2.2.1.1 Η γλώσσα JavaScript και η τεχνολογία AJAX 22
2.2.1.2 Η βιβλιοθήκη JQuery και το πρόσθετο jsTree 23
2.2.1.3 Το πρότυπο JSON 25
2.2.2 Λογισμικό διακομιστή – server-side 25
2.2.2.1 Η γλώσσα Java 26
2.2.2.2 Java και server-side κώδικας – Java Servlets 26
2.2.2.3 Βιβλιοθήκες Java 27
2.3 Σημασιολογικός ιστός 27
2.3.1 Ο σημερινός Παγκόσμιος Ιστός 28
2.3.2 Η μετάβαση στο Σημασιολογικό Ιστό 28
2.3.3 Οντολογίες 29
3 Δομή της εφαρμογής 31
3.1 Αρχιτεκτονική του συστήματος 31
3.1.1 Διεπαφή χρήστη 32
3.1.2 Υπηρεσίες στον server 32
3.2 Τρόπος λειτουργίας του συστήματος 33
3.2.1 Αρχικοποίηση του συστήματος 33
3.2.2 Κανονική λειτουργία της εφαρμογής 34
3.2.2.1 Επικοινωνία της διεπαφής χρήστη με τον server 34

3.2.2.2 Service επεξεργασίας δεδομένων 35
3.2.2.3 Service αναζήτησης στο Twitter 36
3.3 Ανέβασμα της εφαρμογής στον server 36
4 Αλληλεπίδραση χρήστη – εφαρμογής 39
4.1 Εκκίνηση της εφαρμογής 39
4.1.1 Οργάνωση της διεπαφής χρήστη 39
4.1.2 Ανάγνωση της οντολογίας 41
4.2 Πραγματοποίηση αναζήτησης 43
4.2.1 Λειτουργικότητες της διεπαφής χρήστη 43
4.2.2 Επιλογή όρων και εκκίνηση της αναζήτησης 44
4.3 Παρουσίαση αποτελεσμάτων της αναζήτησης 45
5 Λειτουργικότητα αναζήτησης και βελτίωση αποτελεσμάτων 47
5.1 Οι δύο φάσεις της διαδικασίας 47
5.2 Πρώτη φάση: Συλλογή δημοσιεύσεων από το Twitter 48
5.2.1 Αποφυγή εξαιρέσεων 49
5.3 Δεύτερη φάση: Αναζήτηση δημοσιεύσεων με λέξεις-κλειδιά 50
5.3.1 Προετοιμασία της αναζήτησης 50
5.3.1.1 Εντοπισμός των επιλεγμένων keywords 50
5.3.1.2 Βελτιστοποίηση της διαδικασίας 51
5.3.1.3 Εκκίνηση της αναζήτησης 51
5.3.2 Αναζήτηση δημοσιεύσεων 52
5.3.2.1 Βασική λειτουργικότητα του service 52
5.3.2.2 Τροποποιήσεις και βελτιώσεις 54
5.3.2.2.1 Αρχική φάση αναζήτησης 54
5.3.2.2.2 Διάσπαση σε επιμέρους λέξεις 55
5.3.2.2.3 Ο αλγόριθμος Porter Stemmer [6] 56
5.3.2.3 Αξιολόγηση προηγούμενης και νέας υλοποίησης 57
5.4 Επιστροφή αποτελεσμάτων 58
5.4.1 Συλλογή δημοσιεύσεων 58
6 Σύνοψη 61
6.1 Συμπεράσματα 61
6.2 Μελλοντικές επεκτάσεις 62
Παράρτημα 63
Π.1 Ανάγνωση της οντολογίας 63
Π.2 Εκκίνηση αναζήτησης από τον χρήστη 64
Π.3 Ανάκτηση των keywords και λήψη αποτελεσμάτων 66
Βιβλιογραφία 71

Κατάλογος εικόνων

Εικόνα 2.1: Σύστημα Πελάτη-Διακομιστή [19] 19
Εικόνα 2.2: Επικοινωνία client-server μέσω πρωτοκόλλου HTTP [15] 20
Εικόνα 2.3: JavaScript vs JQuery 24
Εικόνα 2.4: Χαρακτηριστική δομή jsTree 24
Εικόνα 2.5: Αντικείμενο JSON 25
Εικόνα 2.6: Η λειτουργία ενός servlet [15] 27
Εικόνα 3.1: Επικοινωνία μεταξύ των τμημάτων της εφαρμογής 32
Εικόνα 3.2: Σώμα του POST Request για αρχικοποίηση του Search Service 33
Εικόνα 4.1: Διεπαφή χρήστη πριν την εκτέλεση αναζήτησης 41
Εικόνα 4.2: Κλάση της οντολογίας που χρησιμοποιήθηκε στην εργασία [18] 41
Εικόνα 4.3: Η διεπαφή χρήστη με επιλεγμένες τις έννοιες Lifestyle και Interventions και την λέξη pain στο πεδίο κειμένου 44
Εικόνα 4.4: Αποτελέσματα της αναζήτησης 46
Εικόνα 5.1: Οι λειτουργίες της εφαρμογής σε διάγραμμα UML 48
Εικόνα 5.2: JSON Body του POST HTTP Request προς το Search Service 51
Εικόνα 5.3: Λίστα με ενδεικτικά αποτελέσματα που επιστρέφει το service 53
Εικόνα 5.4: Precision – Recall στην αρχική και τελική μορφή του service 58

1

Εισαγωγή

1.1 Μέσα κοινωνικής δικτύωσης

Η σημερινή εποχή χαρακτηρίζεται από την ραγδαία ανάπτυξη και την ευρεία χρήση των μέσων κοινωνικής δικτύωσης. Ο όρος μέσα κοινωνικής δικτύωσης (ή αλλιώς social media) αναφέρεται στα μέσα αλληλεπίδρασης και επικοινωνίας μεταξύ ανθρώπων που δημιουργούν, μοιράζονται ή ανταλλάσσουν πληροφορίες και ιδέες μέσω διαδικτυακών εικονικών κοινοτήτων. Χαρακτηριστικά παραδείγματα τέτοιων κοινωνικών δικτύων αποτελούν το Facebook, το Twitter, το Instagram, καθώς και τα blog, τα forum κλπ.

Τα social media αποτελούν πλέον αναπόσπαστο κομμάτι της καθημερινότητας εκατομμυρίων ανθρώπων παγκοσμίως. Και αυτό διότι διαθέτουν τα παρακάτω χαρακτηριστικά:

- Υποστηρίζουν ποικίλες μορφές περιεχομένου, όπως κείμενο, βίντεο, φωτογραφίες και ήχο.
- Αυξάνουν την ταχύτητα και το εύρος διάδοσης των πληροφοριών.
- Επιτρέπουν την επικοινωνία μεταξύ των χρηστών σε πραγματικό χρόνο.
- Είναι ανεξάρτητα της συσκευής, δηλαδή μπορεί κανείς να συνδεθεί σε αυτά μέσω υπολογιστή, κινητού ή tablet.

Από την πλευρά των επιχειρήσεων, πολλές γνωστές εταιρείες αξιοποιούν τις δυνατότητες διασύνδεσης που προσφέρουν οι πλατφόρμες κοινωνικής δικτύωσης για να ενισχύσουν την παραγωγικότητα, την καινοτομία, τη φήμη, τη συνεργασία και τη δέσμευση των εργαζομένων τους με την εταιρεία. Οι επικεφαλής αναζητούν τρόπους αξιοποίησης της δημοτικότητας και της αξίας που μπορούν να τους προσθέσουν τα social media, ενισχύοντας έτσι την απόδοση των οργανισμών τους και προάγοντας τους εταιρικούς στόχους.

1.1.1 Twitter

Στο πλαίσιο της παρούσας εργασίας, χρησιμοποιήσαμε το Twitter για να αντλήσουμε τα δεδομένα που μας ενδιαφέρουν, συνεπώς θα πρέπει να γίνει μια σύντομη αναφορά στη συγκεκριμένη πλατφόρμα κοινωνικής δικτύωσης.

Το Twitter είναι ένας ιστόχωρος κοινωνικής δικτύωσης που επιτρέπει στους χρήστες του να στέλνουν και να διαβάζουν σύντομα μηνύματα, μέχρι 280 χαρακτήρες, τα οποία ονομάζονται tweets. Τα μηνύματα μπορούν να αναγνωστούν και από μη συνδεδεμένους χρήστες, αλλά μόνο οι συνδεδεμένοι μπορούν να δημοσιεύσουν κείμενα. Δημιουργήθηκε στις 21 Μαρτίου του 2006 από τον Τζακ Ντόρσεϊ και έγινε διαθέσιμο στους χρήστες τον Ιούλιο του ίδιου χρόνου. Η υπηρεσία έγινε γρήγορα δημοφιλής και σήμερα έχει 321 εκατομμύρια ενεργούς μηνιαίους χρήστες (2019), όντας ένας από τους δέκα πιο δημοφιλείς ιστοτόπους του διαδικτύου [14].

1.2 Γενικά για την εργασία

Σκοπός της εργασίας ήταν η δημιουργία μιας διαδικτυακής εφαρμογής (web application) για την παρουσίαση δημοσιεύσεων από το Twitter, με βάση ορισμένες λέξεις-κλειδιά που πληκτρολογούνται ή επιλέγονται από τον χρήστη της εφαρμογής, σε συνδυασμό με κάποιες επιπλέον προκαθορισμένες έννοιες που δίνονται ξεχωριστά κατά την αρχικοποίηση της υπηρεσίας συλλογής των δημοσιεύσεων. Ο χρήστης επιλέγει μία ή περισσότερες από τις διαθέσιμες έννοιες που του παρουσιάζονται σε μορφή καταλόγου μέσω της διαδικτυακής διεπαφής (web interface) της εφαρμογής και στη συνέχεια βλέπει τις δημοσιεύσεις από το Twitter που περιέχουν τις επιλεγμένες λέξεις-κλειδιά, καθώς και τις προεπιλεγμένες έννοιες που δόθηκαν κατά την αρχικοποίηση της εφαρμογής.

Ζητούμενο ήταν η αξιοποίηση του Twitter ως μια τεράστια πηγή πληροφοριών, από όπου μπορούμε να αντλήσουμε δεδομένα με επιστημονικό και όχι μόνο ενδιαφέρον, τα οποία δύνανται να χρησιμοποιηθούν για την εξαγωγή σημαντικών συμπερασμάτων, τη συλλογή στατιστικών στοιχείων κλπ. Αυτό επιτυγχάνεται χάρη στη δυνατότητα για σύνθετη αναζήτηση που δίνει η εφαρμογή μας, καθώς οι χρήστες μπορούν να βρίσκουν πολύ πιο εξειδικευμένες πληροφορίες σε σημαντικά μικρότερο χρόνο από ό,τι στην συμβατική αναζήτηση του Twitter, επιλέγοντας τα κατάλληλα keywords. Μέσω της εφαρμογής, αναδεικνύεται η δυνατότητα του συγκεκριμένου μέσου κοινωνικής δικτύωσης να αποτελέσει χρήσιμο εργαλείο για τις ανάγκες και τους σκοπούς της επιστημονικής έρευνας.

Παράλληλα, μέσα από αυτήν την εργασία μας δόθηκε η ευκαιρία να έρθουμε σε επαφή με τις πλέον διαδεδομένες τεχνολογίες του διαδικτύου. Ενδεικτικά, ασχοληθήκαμε με το σύστημα πελάτη-διακομιστή (client-server) και το πρωτόκολλο HTTP που χρησιμοποιείται για την επικοινωνία των επιμέρους τμημάτων του συστήματος αυτού, καθώς επίσης και με τεχνολογίες δικτυακού προγραμματισμού, τόσο στην πλευρά του διακομιστή, όπως οι Java Servlets, όσο και στην πλευρά του πελάτη, όπως η γλώσσα JavaScript και η βιβλιοθήκη της, jQuery. Επιπλέον, για την παρουσίαση των keywords στους χρήστες καθώς και για την βελτίωση της αναζήτησης γίνεται ανάγνωση ενός αρχείου οντολογίας (αρχείο .owl), στο οποίο διατηρούνται όλες οι απαραίτητες πληροφορίες για κάθε λέξη-κλειδί. Τέλος, για την ανάπτυξη της εφαρμογής χρησιμοποιήθηκε λογική προσανατολισμένη στις υπηρεσίες (service-oriented), η οποία αποτελεί σήμερα την κυρίαρχη τάση στη δημιουργία εφαρμογών διαδικτύου.

1.2.1 Δομή της εργασίας

Στο κεφάλαιο 2 παρουσιάζονται οι τεχνολογίες που χρησιμοποιήθηκαν για την υλοποίηση της εφαρμογής. Γίνεται αναφορά στο μοντέλο αρχιτεκτονικής λογισμικού «πελάτη-διακομιστή», στο πρωτόκολλο επικοινωνίας HTTP και στον Apache Tomcat που αποτέλεσε τον server της εφαρμογής. Περιγράφονται η γλώσσα προγραμματισμού Java και οι Java Servlets, καθώς και η γλώσσα JavaScript, ενώ τέλος παρουσιάζονται οι οντολογίες και ο σημασιολογικός ιστός.

Στο κεφάλαιο 3 εξετάζεται η δομή της εφαρμογής. Περιγράφεται η αρχιτεκτονική του συστήματος, τα βασικά του τμήματα και ο ρόλος του καθενός. Έπειτα, περιγράφεται ο τρόπος αρχικοποίησης του συστήματος και αναλύεται η λειτουργία των επιμέρους τμημάτων του και η επικοινωνία μεταξύ τους.

Στο κεφάλαιο 4 παρατίθενται τα χαρακτηριστικά της διεπαφής χρήστη. Αναλύεται η δομή της και περιγράφονται τόσο οι λειτουργικότητές της όσο και οι επιλογές που έχουν οι χρήστες όταν αλληλεπιδρούν μέσω αυτής με την εφαρμογή. Επίσης, γίνεται εκτενής περιγραφή της μορφής της οντολογίας που χρησιμοποιείται για τη δημιουργία του καταλόγου με τα keywords, καθώς και της διαδικασίας ανάγνωσης της οντολογίας για την κατασκευή του καταλόγου αυτού.

Στο κεφάλαιο 5 παρουσιάζονται οι δύο ξεχωριστές φάσεις της διαδικασίας αναζήτησης δημοσιεύσεων στο Twitter. Περιγράφεται πρώτα η αναζήτηση και συλλογή δημοσιεύσεων από το Twitter με βάση κάποιες προεπιλεγμένες έννοιες που έχουν δοθεί κατά την αρχικοποίηση της εφαρμογής, ενώ στη συνέχεια εξετάζονται τα διαφορετικά στάδια της αναζήτησης (ή

φιλτραρίσματος) που πραγματοποιείται κατόπιν επιλογής ή και εισαγωγής κάποιων όρων από τον χρήστη μέσω του user interface. Γίνεται επίσης αξιολόγηση της ακολουθούμενης μεθοδολογίας αναζήτησης και φιλτραρίσματος με βάση τις μετρικές precision και recall.

Τέλος, στο κεφάλαιο 6 παρατίθενται τα συμπεράσματα που προέκυψαν μετά την ολοκλήρωση της υλοποίησης της εφαρμογής και αφού αυτή τέθηκε σε λειτουργία. Προτείνονται επίσης ενδεικτικά κάποιες πιθανές μελλοντικές βελτιώσεις της εφαρμογής.

2

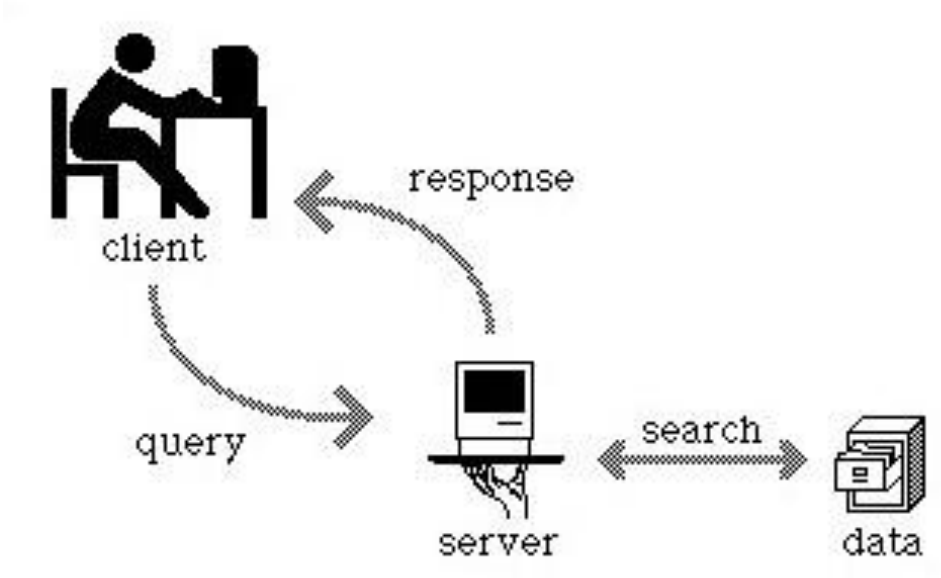
Σχετικές τεχνολογίες

2.1 Περιβάλλον προγραμματισμού

Η εφαρμογή δομήθηκε με βάση το σύστημα πελάτη-διακομιστή, για την επικοινωνία του οποίου χρησιμοποιήθηκε το πρωτόκολλο μεταφοράς υπερκειμένου (Hypertext Transfer Protocol, HTTP), ενώ ως web server επιλέχθηκε ο Apache Tomcat.

2.1.1 Το μοντέλο Πελάτη-Διακομιστή

Η εφαρμογή αναπτύχθηκε ως web application προκειμένου να καταστεί δυνατή η επικοινωνία και η ανταλλαγή δεδομένων με τρίτες εφαρμογές μέσω του διαδικτύου. Χρησιμοποιήθηκε το μοντέλο πελάτη-διακομιστή (client-server), το οποίο αποτελεί την πιο συνήθη μέθοδο ανάπτυξης λογισμικού για εφαρμογές διαδικτύου. Σε ένα σύστημα client-server ο πελάτης (client), ένα τμήμα λογισμικού, ζητά κάτι (π.χ. έναν πόρο, τα αποτελέσματα ενός υπολογισμού κλπ) και ένα άλλο τμήμα λογισμικού, ο διακομιστής (server) του το επιστρέφει. Κάθε διακομιστής μπορεί να εξυπηρετεί πολλαπλούς πελάτες [8].



Εικόνα 2.1: Σύστημα Πελάτη-Διακομιστή [19]

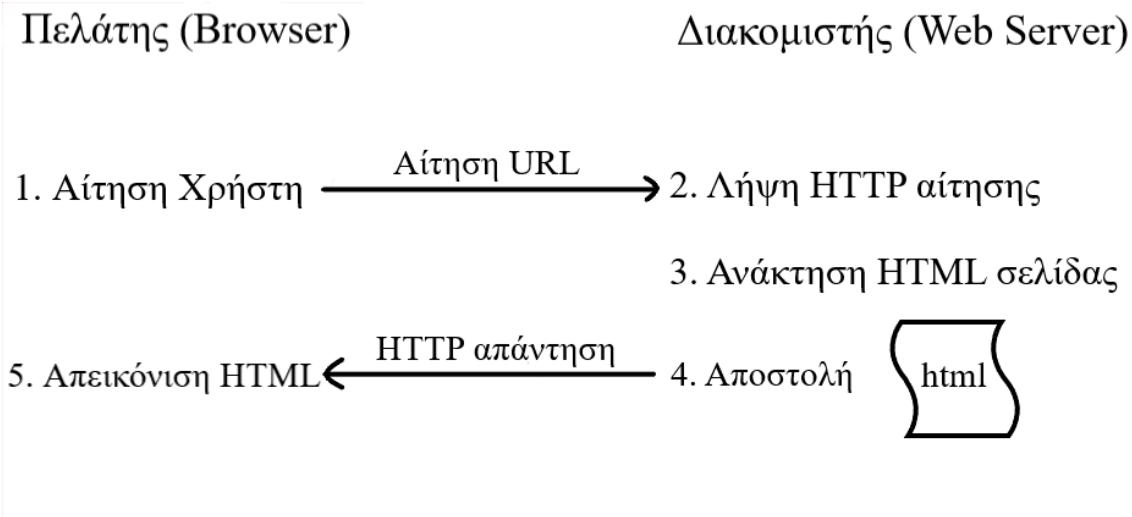
2.1.2 Το πρωτόκολλο HTTP

Το πρωτόκολλο HTTP (Hypertext Transfer Protocol) είναι ένα πρωτόκολλο δικτύου που χρησιμοποιείται από τα προγράμματα περιήγησης ή φυλλομετρητές του Παγκόσμιου Ιστού (web browsers) και τους διακομιστές ιστού (web servers) για την μεταξύ τους επικοινωνία. Είναι εύκολο να το αναγνωρίσουμε κατά την επίσκεψη σε έναν ιστότοπο, επειδή είναι γραμμένο στη διεύθυνση URL (π.χ. http://www.google.com). Αποτελεί το κύριο πρωτόκολλο επικοινωνίας που μπορεί να χρησιμοποιηθεί σε ένα σύστημα client-server για την μεταφορά δεδομένων. Συγκεκριμένα, ένα πρόγραμμα περιήγησης ιστού, που έχει το ρόλο του πελάτη, ζητά αρχεία HTML από έναν διακομιστή ιστού, τα οποία στη συνέχεια εμφανίζονται στο πρόγραμμα περιήγησης με κείμενο, εικόνες, υπερσυνδέσεις κλπ [9].

Οι υπολογιστές-πελάτες και οι διακομιστές HTTP επικοινωνούν μέσω μηνυμάτων αίτησης και απόκρισης HTTP. Οι τρεις κύριοι τύποι μηνυμάτων HTTP είναι GET, POST και HEAD:

- HTTP GET: τα μηνύματα που αποστέλλονται σε ένα διακομιστή περιέχουν μόνο μια διεύθυνση URL. Στο τέλος της διεύθυνσης URL μπορούν να επισυναφθούν προαιρετικές παράμετροι δεδομένων. Ο διακομιστής επεξεργάζεται το προαιρετικό τμήμα δεδομένων της διεύθυνσης URL, εάν υπάρχει, και επιστρέφει το αποτέλεσμα (μια ιστοσελίδα ή ένα στοιχείο μιας ιστοσελίδας) στο πρόγραμμα περιήγησης.
- HTTP POST: τα μηνύματα τοποθετούν τις προαιρετικές παραμέτρους δεδομένων στο σώμα του μηνύματος αίτησης αντί να τις προσθέτουν στο τέλος της διεύθυνσης URL.
- HTTP HEAD: η αίτηση λειτουργεί το ίδιο με τα αιτήματα GET. Αντί να απαντά με το πλήρες περιεχόμενο της διεύθυνσης URL, ο διακομιστής στέλνει πίσω μόνο τις πληροφορίες κεφαλίδας (που περιέχονται στο τμήμα HTML) [3].

Το πρόγραμμα περιήγησης εκκινεί την επικοινωνία με ένα διακομιστή HTTP εκκινώντας μια περίοδο σύνδεσης στο διακομιστή. Μόλις αυτή δημιουργηθεί, ο χρήστης ενεργοποιεί την αποστολή και λήψη μηνυμάτων HTTP με την επίσκεψη στην ιστοσελίδα.



Εικόνα 2.2: Επικοινωνία client-server μέσω πρωτοκόλλου HTTP [15]

2.1.3 Υπηρεσιοστρεφής Αρχιτεκτονική

Η Υπηρεσιοστρεφής Αρχιτεκτονική (Service-Oriented Architecture) είναι μια προσέγγιση που χρησιμοποιείται στην κατασκευή έργων λογισμικού, στην οποία μια εφαρμογή αποτελείται από ένα σύνολο από αυτόνομες, χαλαρά συνδεδεμένες υπηρεσίες. Οι υπηρεσίες αυτές επικοινωνούν μεταξύ τους για την επίτευξη ενός κοινού στόχου, με την καθεμία να εκτελεί μια ανεξάρτητη λειτουργία, χωρίς να επηρεάζεται από την κατάσταση των υπόλοιπων υπηρεσιών του συστήματος. Η ανταλλαγή μηνυμάτων μεταξύ των υπηρεσιών γίνεται με βάση κάποιο πρότυπο για τη διασφάλιση της αξιοπιστίας του συστήματος, αλλά και για μεγαλύτερη αποτελεσματικότητα στην ανταλλαγή της πληροφορίας. Η τεχνολογία των διαδικτυακών υπηρεσιών είναι η πλέον διαδεδομένη τεχνολογία υλοποίησης μιας υπηρεσιοστρεφούς αρχιτεκτονικής.

2.1.3.1 Διαδικτυακές Υπηρεσίες

Πρόκειται για τον πιο ευρέως χρησιμοποιούμενο τύπο υπηρεσιών στις υπηρεσιοστρεφείς αρχιτεκτονικές. Οι διαδικτυακές υπηρεσίες (web services) αποτελούν αυτόνομες, κατανεμημένες δυναμικές εφαρμογές, οι οποίες είναι διαθέσιμες μέσω του διαδικτύου ή μέσω ενός ιδιωτικού δικτύου, και μπορούν να κληθούν από κάποιον web browser. Οι υπηρεσίες αυτές μοιράζονται δεδομένα μέσω του δικτύου, επιτρέποντας την επικοινωνία μεταξύ εφαρμογών από διαφορετικές πηγές, χωρίς χρονοβόρες διαδικασίες [17]. Παράλληλα, δεν εξαρτώνται από το λειτουργικό σύστημα ή τη γλώσσα προγραμματισμού, γεγονός που καθιστά εύκολη την επαναχρησιμοποίηση και την επέκτασή τους. Θα πρέπει να σημειώσουμε ότι μια διαδικτυακή υπηρεσία δεν χρειάζεται να έχει υποχρεωτικά κάποια γραφική διεπαφή (GUI), ενώ ακόμα κι αν δημιουργήσουμε μια τέτοια για την παροχή επιπλέον λειτουργικότητας στους χρήστες, η υπηρεσία δεν θα εξαρτάται από αυτή, αλλά θα συνεχίσει να αποτελεί μια ανεξάρτητη και πλήρως λειτουργική οντότητα.

2.1.3.2 RESTful Διαδικτυακές Υπηρεσίες

Ο όρος REST περιγράφει ένα σύνολο αρχιτεκτονικών αρχών, με βάση τις οποίες μπορούμε να σχεδιάσουμε υπηρεσίες με συγκεκριμένες προδιαγραφές. Το βασικότερο χαρακτηριστικό των RESTful υπηρεσιών είναι η αποκλειστική χρήση του πρωτοκόλλου HTTP για την επικοινωνία μεταξύ τους. Η ανταλλαγή μηνυμάτων πραγματοποιείται με την κλήση HTTP μεθόδων, όπως αυτές που αναφέρθηκαν στην παράγραφο 2.1.2. Για παράδειγμα, αν θέλουμε να δημιουργήσουμε έναν πόρο στον διακομιστή χρησιμοποιούμε τη μέθοδο POST, ενώ για να ανακτήσουμε έναν πόρο από τον διακομιστή χρησιμοποιούμε GET.

Ένα άλλο σημαντικό χαρακτηριστικό αυτού του τύπου υπηρεσιών είναι ότι κάθε αναφορά σε αυτές πραγματοποιείται μέσω ενός μοναδικού URI, το οποίο έχει δομή καταλόγου, ως ακολούθως:

```
http://localhost:8080/SMA_Adapter/TwitterServlet
```

Πρόκειται για μια ιεραρχική δομή που αναπαριστά μία διαδρομή στο διακομιστή, η οποία παρέχει πρόσβαση στην υπηρεσία. Όπως βλέπουμε, οι διαδοχικές περιοχές του διακομιστή που συναποτελούν αυτή τη διαδρομή, είναι διαχωρισμένες με κάθετο.

Τέλος, η μορφή των δεδομένων που ανταλλάσσουν οι επιμέρους RESTful υπηρεσίες μεταξύ τους είναι αυστηρά καθορισμένη και σύμφωνη με κάποιο πρότυπο, όπως είναι το XML, το JSON ή κάποιος συνδυασμός τους. Έτσι, τα μηνύματα καθίστανται ευανάγνωστα, ενώ η υπηρεσία είναι εφικτό να επικοινωνήσει με ένα πλήθος άλλων υπηρεσιών, οι οποίες μπορεί να είναι γραμμένες σε διαφορετικές γλώσσες και να τρέχουν σε διαφορετικές πλατφόρμες ή συσκευές [17].

2.1.4 Διακομιστές ιστού

Οι διακομιστές ιστού (web servers) είναι προγράμματα που δέχονται αιτήματα HTTP και τα εξυπηρετούν επιστρέφοντας ως απάντηση HTML αρχεία. Είναι το λογισμικό που συνήθως απαντά όταν καλούμε μια διεύθυνση σε ένα φυλλομετρητή (browser). Οι δημοφιλέστεροι είναι ο Apache Web Server της Apache Foundation και ο IIS της Microsoft [15].

2.1.4.1 Ο διακομιστής Apache Tomcat

Ο Apache Tomcat (που επίσης αναφέρεται ως Tomcat Server) είναι ένας διακομιστής ιστού που υλοποιεί διάφορες προδιαγραφές Java Enterprise Edition (Java EE), όπως το Java Servlet, οι σελίδες Java Server (JSP), το Java EL και το WebSocket και παρέχει ένα περιβάλλον διακομιστή HTTP στο οποίο μπορεί να εκτελεστεί κώδικας Java.

Το λογισμικό Tomcat αναπτύσσεται και συντηρείται από μια ανοιχτή κοινότητα προγραμματιστών υπό την αιγίδα του Apache Software Foundation, κυκλοφορεί υπό την άδεια Apache License 2.0 και είναι λογισμικό ανοιχτού κώδικα [2].

2.2 Τμήματα λογισμικού

Για την υλοποίηση κάθε web εφαρμογής απαιτείται η επικοινωνία και η συνεργασία δύο ξεχωριστών τμημάτων λογισμικού: του λογισμικού στην πλευρά του πελάτη (client-side η frontend) και του λογισμικού στην πλευρά του διακομιστή (server-side ή backend).

2.2.1 Λογισμικό πελάτη – client-side

Το client-side κομμάτι μιας διαδικτυακής εφαρμογής αποτελείται από μία ή περισσότερες σελίδες HTML. Υλοποιεί τη διεπαφή χρήστη (user interface), δηλαδή το τμήμα της ιστοσελίδας που βλέπουν οι χρήστες, μέσω του οποίου πραγματοποιείται η αλληλεπίδρασή τους με την εφαρμογή για την αποστολή δεδομένων στον server, την παρουσίαση αποτελεσμάτων κλπ.

Για την μορφοποίηση της εμφάνισης του περιεχομένου μιας HTML σελίδας χρησιμοποιείται η τεχνολογία των CSS (Cascading Style Sheets), ενώ η προσθήκη λειτουργικότητας στη σελίδα γίνεται μέσω της γλώσσας προγραμματισμού JavaScript.

2.2.1.1 Η γλώσσα JavaScript και η τεχνολογία AJAX

Η JavaScript (JS) είναι μια γλώσσα σεναρίων (script language) που βασίζεται στα πρωτότυπα (prototype-based). Αρχικά αποτέλεσε μέρος της υλοποίησης των web browsers, ώστε τα σενάρια από την πλευρά του πελάτη (client-side scripts) να μπορούν να επικοινωνούν με τον χρήστη, να ανταλλάσσουν δεδομένα ασύγχρονα και να αλλάζουν δυναμικά το περιεχόμενο του HTML εγγράφου που εμφανίζεται. Κάθε τμήμα του κώδικα JavaScript συνδέεται σε κάποιο στοιχείο της HTML σελίδας, όπως κάποιο κουμπί ή checkbox, και ενεργοποιείται με την εκτέλεση κάποιου γεγονότος, που μπορεί να είναι το πάτημα ενός κουμπιού, η υποβολή μιας φόρμας κλπ.

Η σύνταξή της είναι επηρεασμένη από τη C. Η JavaScript αντιγράφει πολλά ονόματα και συμβάσεις ονοματοδοσίας από τη Java, αλλά γενικά οι δύο αυτές γλώσσες δε σχετίζονται και έχουν

πολύ διαφορετική σημασιολογία. Είναι γλώσσα βασισμένη σε διαφορετικά προγραμματιστικά παραδείγματα (multi-paradigm), υποστηρίζοντας αντικειμενοστρεφές, προστακτικό και συναρτησιακό στυλ προγραμματισμού.

Η JavaScript χρησιμοποιείται και σε εφαρμογές εκτός ιστοσελίδων. Μερικά παραδείγματα είναι τα έγγραφα PDF, οι εξειδικευμένοι φυλλομετρητές (site-specific browsers) και οι μικρές εφαρμογές της επιφάνειας εργασίας (desktop widgets). Οι νεότερες εικονικές μηχανές και πλαίσια ανάπτυξης για JavaScript (όπως το Node.js) έχουν επίσης κάνει τη JavaScript πιο δημοφιλή για την ανάπτυξη εφαρμογών Ιστού στην πλευρά του διακομιστή (server-side) [11].

Η τεχνολογία AJAX (Asynchronous JavaScript and XML) είναι ένα σύνολο από τεχνικές web development που χρησιμοποιούν πολλές τεχνολογίες του διαδικτύου, προκειμένου να προσδώσουν διαδραστικές δυνατότητες σε ένα δυναμικό site, μετατρέποντάς το σε μια διαδικτυακή εφαρμογή. Με AJAX, οι web εφαρμογές μπορούν να στέλνουν και να ανακτούν δεδομένα από έναν server ασύγχρονα, τρέχοντας δηλαδή στο παρασκήνιο, χωρίς να παρεμβαίνουν στην εμφάνιση και τη συμπεριφορά της υπάρχουσας σελίδας. Χάρη στην αποσύνδεση του επιπέδου παρουσίασης της σελίδας από το επίπεδο των δεδομένων που έχουν την δυνατότητα αλλαγής, η AJAX επιτρέπει σε web σελίδες, και κατ' επέκταση σε web εφαρμογές, να αλλάζουν το περιεχόμενό τους δυναμικά, χωρίς να χρειάζεται να φορτωθεί εκ νέου ολόκληρη η σελίδα. Το ενσωματωμένο XMLHttpRequest αντικείμενο εντός της JavaScript χρησιμοποιείται συνήθως για να εκτελέσει την AJAX, επιτρέποντας σε ιστοσελίδες να φορτώσουν το περιεχόμενό τους, χωρίς να ανανεώσουν τη σελίδα [7].

2.2.1.2 Η βιβλιοθήκη JQuery και το πρόσθετο jsTree

Η JQuery είναι μια βιβλιοθήκη της JavaScript σχεδιασμένη να απλοποιήσει την υλοποίηση σεναρίων (scripting) στην πλευρά του πελάτη (client-side) και υποστηρίζει πολλαπλούς φυλλομετρητές Ιστού. Χρησιμοποιείται σε πάνω από το 65% των 10.000 ιστοτόπων με τη μεγαλύτερη επισκεψιμότητα.

Η JQuery είναι ελεύθερο λογισμικό και διανέμεται υπό την άδεια του MIT [12]. Πρόκειται για ένα αρχείο JavaScript, που περιέχει όλες τις λειτουργίες. Μπορεί να συμπεριληφθεί σε μια ιστοσελίδα παρέχοντας το αρχείο τοπικά:

```
<script type="text/javascript" src="jquery.js"></script>
```

ή έχοντας έναν σύνδεσμο σε έναν από τους πολλούς διακομιστές που τη φιλοξενούν, επιλογή η οποία προτιμήθηκε στην παρούσα εργασία:

```
<script src="http://AJAX.googleapis.com/AJAX/libs/jquery/1.9.1/jquery.min.js"></script>
```

Θα πρέπει να σημειωθεί ότι με τη χρήση της JQuery απλοποιείται σημαντικά το έργο της προσθήκης λειτουργικότητας σε σελίδες HTML, καθώς μας παρέχει μια ευέλικτη προγραμματιστική διεπαφή (API, Application Programming Interface) με την οποία μπορούμε να γράψουμε λιγότερες γραμμές κώδικα από ό,τι αν γράφαμε σε «καθαρή» JavaScript, ενώ ο κώδικάς μας είναι πιο ευανάγνωστος, άρα καθίσταται ευκολότερο να συντηρηθεί και να επεκταθεί.

```
// JavaScript code
var header = table.createTHead();
var row = header.insertRow(-1);
for (var i = 0; i < columnCount; i++) {
    var headerCell = document.createElement('th');
    headerCell.innerText = columnHeadings[i].toUpperCase();
    row.appendChild(headerCell);
}

// JQuery code
var header = $('<thead />').appendTo(table);
for (var i = 0; i < columnCount; i++) {
    $('<th />', { text: columnHeadings[i].toUpperCase() }).appendTo(header);
}
```

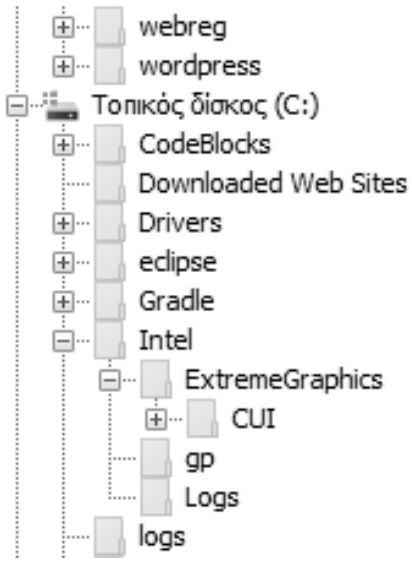
Εικόνα 2.3: JavaScript vs JQuery

Το jsTree είναι ένα πρόσθετο (plugin) της βιβλιοθήκης JQuery. Είναι δηλαδή ένα συστατικό λογισμικού (software component) που σχεδιάστηκε για να επεκτείνει τη λειτουργικότητα της JQuery, προκειμένου να μπορούμε να κατασκευάσουμε και να εμφανίσουμε στη σελίδα μας έναν κατάλογο με τη μορφή διαδραστικού δέντρου, παρόμοιου με τη δομή ενός συστήματος αρχείων στον υπολογιστή μας, όπως φαίνεται στην εικόνα 2.4. Για τις ενέργειες που γίνονται σε κόμβους του δέντρου χρησιμοποιείται το σύστημα διαχείρισης γεγονότων της JQuery.

Και εδώ μπορεί να συμπεριληφθεί το plugin στην εφαρμογή μας είτε παρέχοντας κάποιο αρχείο τοπικά είτε δίνοντας κάποιον εξωτερικό σύνδεσμο. Προτιμήθηκε ξανά η δεύτερη επιλογή:

```
<link rel="stylesheet" href="https://cdnjs.cloudflare.com/AJAX/libs/jstree/3.2.1/themes/default/style.min.css"/>
```

Το jsTree είναι ελεύθερο λογισμικό ανοιχτού κώδικα και διανέμεται υπό την άδεια του MIT όπως και η JQuery [4].



Εικόνα 2.4: Χαρακτηριστική δομή jsTree

2.2.1.3 *Το πρότυπο JSON*

Στην πράξη, οι σύγχρονες εφαρμογές συνήθως χρησιμοποιούν JSON (JavaScript Object Notation) αντί για XML, λόγω των εγγενών πλεονεκτημάτων του πρώτου στην JavaScript. Το JSON είναι ένα ανοιχτό πρότυπο ανταλλαγής δεδομένων, που χρησιμοποιεί κείμενο αναγνώσιμο από τον άνθρωπο για να μεταδώσει πληροφοριακά αντικείμενα δεδομένων. Αποτελείται από ζεύγη τύπου χαρακτηριστικό-τιμή (key-value) και χρησιμοποιείται κυρίως για τη μετάδοση δεδομένων μεταξύ του εξυπηρετητή και των διαδικτυακών εφαρμογών, ως εναλλακτική λύση της XML [13]. Η τεχνολογία JSON είναι ανεξάρτητη της χρησιμοποιούμενης γλώσσας, ενώ παρέχει μια αυτοπεριγραφική και εύκολη στην κατανόηση μορφή αναπαράστασης της πληροφορίας.

```
{
    "arguments" : { "number" : 10 },
    "url" : "http://localhost:8080/restty-tester/collection",
    "method" : "POST",
    "header" : {
        "Content-Type" : "application/json"
    },
    "body" : [
        {
            "id" : 0,
            "name": "name 0",
            "description" : "description 0"
        },
        {
            "id" : 1,
            "name": "name 1",
            "description" : "description 1"
        }
    ],
    "output" : "json"
}
```

Εικόνα 2.5: Αντικείμενο JSON

2.2.2 *Λογισμικό διακομιστή – server-side*

Το server-side κομμάτι μιας διαδικτυακής εφαρμογής αποτελείται από τον κώδικα που εκτελείται μόλις ο server λάβει κάποιο HTTP request, όπως για παράδειγμα κατά την υποβολή ενός url από τη γραμμή διευθύνσεων του browser ή κατά την υποβολή των δεδομένων μιας φόρμας. Ο κώδικας του server ανάλογα με το είδος του αιτήματος (GET, POST, HEAD κλπ.) εκτελεί την αντίστοιχη λειτουργία και επιστρέφει τα αποτελέσματα στον client.

Συχνά, ένας διακομιστής αλληλεπιδρά με κάποια βάση δεδομένων, από την οποία μπορεί να ανακτά πληροφορίες αν έχουμε GET request, ή να προσθέτει σε αυτή νέες εγγραφές σε περίπτωση POST request. Στο πλαίσιο της παρούσας εργασίας δεν χρησιμοποιήθηκε κάποια βάση για την αποθήκευση και ανάκτηση δεδομένων, οπότε δεν θα ασχοληθούμε περαιτέρω με αυτή την περίπτωση.

Οι πλέον διαδεδομένες γλώσσες για server-side προγραμματισμό είναι η PHP, η Python και η Java. Για τη συγγραφή server-side κώδικα σε Java μπορούν να χρησιμοποιηθούν διάφορες τεχνολογίες που παρέχει η συγκεκριμένη γλώσσα, όπως είναι τα JSP (Java Server Pages), τα JavaBeans και τα Servlets.

2.2.2.1 Η γλώσσα Java

Η Java είναι μια αντικειμενοστρεφής γλώσσα προγραμματισμού που σχεδιάστηκε από την εταιρεία πληροφορικής Sun Microsystems. Ένα από τα βασικά πλεονεκτήματα της Java έναντι των περισσότερων άλλων γλωσσών είναι η ανεξαρτησία από το λειτουργικό σύστημα και την πλατφόρμα. Τα προγράμματα που είναι γραμμένα σε Java τρέχουνε ακριβώς το ίδιο σε Windows, Linux, Unix και Macintosh χωρίς να χρειαστεί να ξαναγίνει μεταγλώττιση (compiling) ή να αλλάξει ο πηγαίος κώδικας για κάθε διαφορετικό λειτουργικό σύστημα. Αυτό επιτυγχάνεται με τη χρήση της εικονικής μηχανής της Java (Java Virtual Machine, JVM). Κατά τη μεταγλώττιση ενός προγράμματος, ο πηγαίος κώδικας μετατρέπεται σε ακολουθίες από ψηφία 0 και 1, δηλαδή σε κώδικα μηχανής, ο οποίος διαφέρει ανάλογα με το είδος του επεξεργαστή (Intel x86, IBM, Sun SPARC, Motorola) και του λειτουργικού συστήματος. Η εικονική μηχανή μεταφράζει τον ήδη μεταγλωττισμένο κώδικα σε γλώσσα μηχανής που να υποστηρίζεται από το λειτουργικό σύστημα και τον επεξεργαστή, προκειμένου να εκτελεστεί.

Ως αντικειμενοστρεφής γλώσσα, η Java βασίζεται στη χρήση αντικειμένων. Τα αντικείμενα είναι συλλογές πεδίων πληροφορίας και μεθόδων επεξεργασίας και προβολής πληροφορίας. Τα διάφορα αντικείμενα ανήκουν σε ενότητες κώδικα που ονομάζονται κλάσεις, οι οποίες δηλώνουν τον τύπο ομοειδών αντικειμένων. Κάθε μέλος μίας κλάσης, είτε πεδίο είτε μέθοδος, συνοδεύεται από ένα προσδιοριστικό εμβέλειας. Υπάρχουν τρία τέτοια προσδιοριστικά: private, protected και public. Τα private μέλη είναι ορατά μόνο από την ίδια την κλάση, τα protected από κλάσεις του ίδιου πακέτου και από κλάσεις εκτός πακέτου που επεκτείνουν (extend) αυτήν την κλάση, ενώ τα public μέλη είναι ορατά από όλες τις κλάσεις της εφαρμογής.

Για να γράψει κάποιος κώδικα Java δε χρειάζεται τίποτα άλλο παρά έναν επεξεργαστή κειμένου, όπως το Σημειωματάριο (Notepad) των Windows ή ο vi, γνωστός στο χώρο του Unix. Παρ' όλ' αυτά, ένα ολοκληρωμένο περιβάλλον ανάπτυξης (integrated development environment, IDE) βοηθάει πολύ, ιδιαίτερα στον εντοπισμό σφαλμάτων (debugging). Υπάρχουν αρκετά διαθέσιμα, ενώ πολλά από αυτά παρέχονται δωρεάν. Ως IDE για την ανάπτυξη της εφαρμογής μας χρησιμοποιήθηκε το Eclipse [10].

2.2.2.2 Java και server-side κώδικας – Java Servlets

Τα Java Servlets είναι κλάσεις της Java, ο κώδικας των οποίων εκτελείται σε κάποιον web server δημιουργώντας και επιστρέφοντας δυναμικό περιεχόμενο, που συνήθως είναι σελίδες HTML ή αντικείμενα JSON. Τα Servlets χρησιμοποιούνται σε εφαρμογές διαδικτύου που απαιτούν προεπεξεργασία πριν την αποστολή της απόκρισης στον χρήστη. Κληρονομούν όλα τα πλεονεκτήματα που προσφέρει η Java και αποτελούν το server-side πρόσωπο της Java σε αντίθεση με τα Applets που αποτελούν το client-side πρόσωπο της Java.

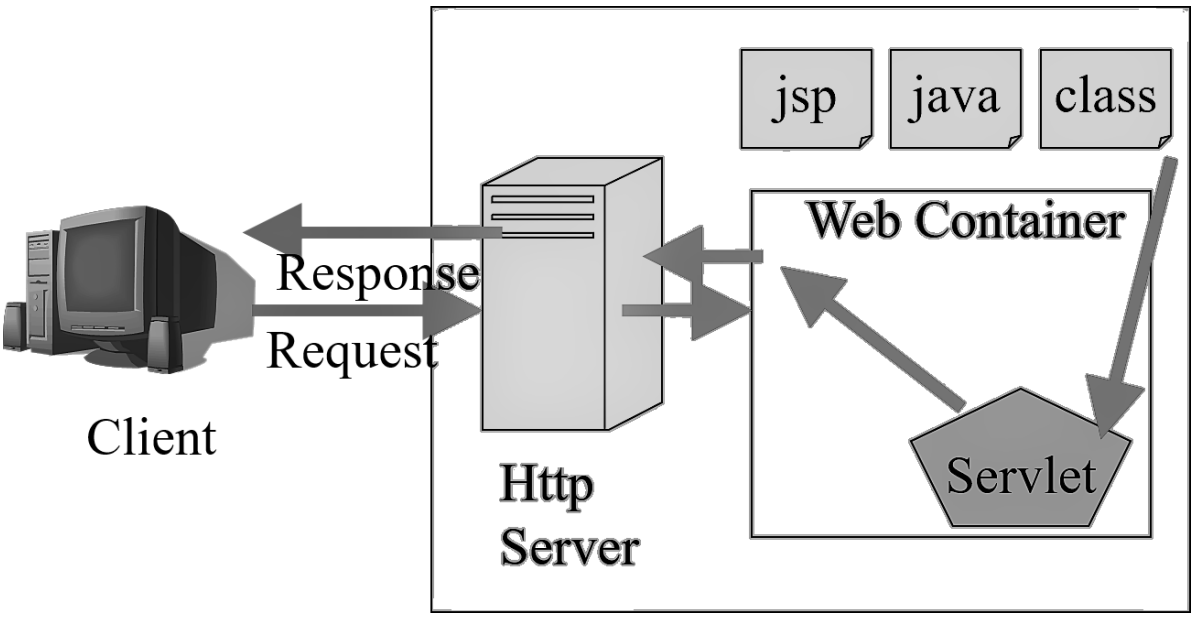
Τα βασικά πλεονεκτήματα των Servlets είναι:

- Ανεξαρτησία από την πλατφόρμα.
- Πλήρης αξιοποίηση του Java API
- Ασφάλεια κώδικα, χάρη στους μηχανισμούς συλλογής απορριμάτων (Garbage Collector) και χειρισμού εξαιρέσεων (Exception Handling)
- Κομψότητα, λόγω της αντικειμενοστρεφούς λογικής και της καθαρότητας και απλότητας του κώδικα
- Εύκολη ενσωμάτωση νέων υπηρεσιών

Ένα Servlet είναι μια κλάση που εφαρμόζει (implements) το javax.servlet.Servlet interface. Πρακτικά, τα περισσότερα servlets επεκτείνουν (extends) την κλάση javax.servlet.http.HttpServlet. Όταν ένα servlet δέχεται μια αίτηση από τον πελάτη, λαμβάνει δύο αντικείμενα (objects), ένα ServletRequest που εξασφαλίζει την κατεύθυνση επικοινωνίας από τον πελάτη προς το servlet, και ένα ServletResponse, που εξασφαλίζει την επικοινωνία με κατεύθυνση από το servlet πίσω στον πελάτη. Τα ServletRequest και ServletResponse είναι διεπαφές (interfaces) ορισμένες στο πακέτο

javax.servlet. Βασικές μέθοδοι του HttpServlet είναι οι doGet και doPost, για τον χειρισμό HTTP αιτημάτων GET και POST αντίστοιχα [15].

Ένας server (π.χ. Tomcat) δέχεται την αίτηση από τον πελάτη, και καλεί το αντίστοιχο servlet μέσω κάποιου url, περνώντας του και τις παραμέτρους του χρήστη. Το servlet αυτό εμπεριέχει την λογική που πρέπει να εκτελεστεί με είσοδο τις παραμέτρους αυτές, και απαντάει επιστρέφοντας μια HTML σελίδα. Η σελίδα αυτή είναι δυναμική καθώς παρήχθη μόλις ζητήθηκε και με βάση τις εκάστοτε παραμέτρους του χρήστη.



Εικόνα 2.6: Η λειτουργία ενός servlet [15]

2.2.2.3 Βιβλιοθήκες Java

Ένα σημαντικό μέρος της λειτουργίας της εφαρμογής αφορά στην ανάγνωση ενός αρχείου οντολογίας και στην επικοινωνία με το Twitter. Οι δύο αυτές ενέργειες διευκολύνονται από τη χρήση του OWL API και του Twitter API (Application Programming Interface). Ένα Java API αποτελεί ένα σύνολο μεθόδων υλοποιημένων σε Java, τις οποίες μπορούμε να χρησιμοποιήσουμε έτοιμες αντί να τις γράψουμε από την αρχή, εξοικονομώντας πολύτιμο χρόνο. Για την ενσωμάτωση και τη χρήση των δύο APIs στην ανάπτυξη της εφαρμογής, είναι απαραίτητο να συμπεριλάβουμε στις βιβλιοθήκες της Java που υπάρχουν ήδη στον server, ορισμένα αρχεία εξαρτήσεων με κατάληξη '.jar', τα οποία περιέχουν τις Java μεθόδους που μας παρέχονται από τα APIs. Μπορούμε με ευκολία να βρούμε τα αρχεία αυτά στο διαδίκτυο, να τα κατεβάσουμε και να τα ενσωματώσουμε στον server.

2.3 Σημασιολογικός ιστός

Ο Παγκόσμιος Ιστός (World Wide Web, www) έχει αλλάξει τον τρόπο επικοινωνίας των ανθρώπων, τον τρόπο διάδοσης και ανάκτησης των πληροφοριών, καθώς και τον τρόπο διεξαγωγής των επιχειρηματικών δραστηριοτήτων. Ο όρος Σημασιολογικός Ιστός (Semantic Web) περιλαμβάνει

τεχνικές που υπόσχονται να βελτιώσουν εντυπωσιακά τον υπάρχοντα Παγκόσμιο Ιστό και τη χρήση του.

2.3.1 Ο σημερινός Παγκόσμιος Ιστός

Οι τυπικές χρήσεις του Ιστού από τους ανθρώπους σήμερα περιλαμβάνουν την αναζήτηση και τη χρησιμοποίηση πληροφοριών, την αναζήτηση άλλων ατόμων και την επαφή μαζί τους, καθώς και την παραγγελία προϊόντων από ηλεκτρονικές πλατφόρμες καταστημάτων. Οι δραστηριότητες αυτές δεν υποστηρίζονται επαρκώς από εργαλεία λογισμικού. Εκτός από την ύπαρξη συνδέσμων που συνδέουν έγγραφα, τα βασικά πολύτιμα και πραγματικά απαραίτητα εργαλεία είναι οι μηχανές αναζήτησης (search engines) που βασίζονται σε λέξεις-κλειδιά, όπως η Yahoo και η Google, οι οποίες αποτελούν τα κύρια εργαλεία χρήσης του σύγχρονου Ιστού.

Υπάρχουν ωστόσο σοβαρά προβλήματα σχετικά με τη χρήση τους, όπως:

- Υψηλή ανάκληση με χαμηλή ακρίβεια. Ακόμα και αν ανακτηθούν οι βασικές σχετικές σελίδες, θα έχουν μικρή χρησιμότητα αν ανακτηθούν και πολλά ακόμα λίγο ή καθόλου σχετικά έγγραφα.
- Τα αποτελέσματα είναι ιδιαίτερα ευαίσθητα στο λεξιλόγιο. Όταν οι αρχικές λέξεις-κλειδιά δεν επιστρέφουν τα επιθυμητά αποτελέσματα, γίνεται χρήση διαφορετικής ορολογίας από αυτήν του αρχικού ερωτήματος. Αυτό δεν είναι ικανοποιητικό, επειδή τα σημασιολογικά παρόμοια ερωτήματα θα πρέπει να επιστρέφουν παρόμοια αποτελέσματα.
- Τα αποτελέσματα είναι μεμονωμένες σελίδες. Αν χρειαζόμαστε πληροφορίες που έχουν διασκορπιστεί σε διάφορα έγγραφα, θα πρέπει να υποβάλουμε πολλά ερωτήματα για να συλλέξουμε τα σχετικά έγγραφα, και έπειτα να εξάγουμε τις επιμέρους πληροφορίες με μη αυτόματο τρόπο και να τις συνθέσουμε [1].

Το βασικό εμπόδιο για την παροχή καλύτερης υποστήριξης στους χρήστες του Ιστού είναι ότι το νόημα του περιεχομένου του Ιστού δεν είναι προς το παρόν προσπελάσιμο από υπολογιστές (machine accessible). Κυρίως όσον αφορά την ερμηνεία προτάσεων και την εξαγωγή χρήσιμων πληροφοριών για τους χρήστες, οι δυνατότητες του υπάρχοντος λογισμικού είναι ακόμα πολύ περιορισμένες. Πώς θα μπορούσε λοιπόν να βελτιωθεί η παρούσα κατάσταση;

2.3.2 Η μετάβαση στο Σημασιολογικό Ιστό

Μια ιδιαίτερα ενδιαφέρουσα προσέγγιση είναι η αναπαράσταση του διαδικτυακού περιεχομένου σε μορφή που είναι ευκολότερα επεξεργάσιμη από υπολογιστές και η χρήση νοήμονων τεχνικών για την εκμετάλλευση αυτών των αναπαραστάσεων. Αυτό το εγχείρημα, που θα φέρει επανάσταση στον Ιστό, αναφέρεται ως πρωτοβουλία για τον Σημασιολογικό Ιστό (Web 3.0). Πρόκειται για μια επέκταση του σημερινού Ιστού, που θα φέρει δομή στο σημασιολογικό περιεχόμενο των ιστοσελίδων. Ο Σημασιολογικός Ιστός βασίζεται σε τεχνολογίες που ήδη υπάρχουν (URI και XML) αλλά και σε νέες τεχνολογίες (RDF, RDFS, OWL, κα.), οι οποίες ολοένα αναπτύσσονται. Δεδομένου ότι ο νέος Ιστός σκοπεύει να είναι μια μεγάλη βάση όπου δεδομένα από διαφορετικά πεδία θα συνδέονται μεταξύ τους, αναμένεται να παίξει μεγάλο ρόλο στη ζωή μας.

Μερικά από τα πεδία στα οποία αναμένεται να έχει την μεγαλύτερη επίδραση είναι στην υγεία, στην παιδεία και στις επιχειρήσεις. Υπάρχουν ήδη πολλές προσπάθειες από εταιρίες, ερευνητές και μη κερδοσκοπικές οργανώσεις για να παραγάγουν πρότυπα οντολογιών, κυρίως για τα παραπάνω πεδία, για να υπάρχουν κοινές γλώσσες και περισσότερα δεδομένα τα οποία να μπορούν να συνδυαστούν για καλύτερα αποτελέσματα. Στην υγεία, γίνεται προσπάθεια για τη δημιουργία ενοποιημένων γλωσσών ιατρικής ορολογίας και υπηρεσίες που θα βοηθούν το ιατρικό προσωπικό και θα κατευθύνουν τους καταναλωτές σε αξιόπιστες πληροφορίες υγείας σχετικά με την κατάστασή τους. Στην εκπαίδευση, ο Σημασιολογικός Ιστός θα συμβάλει σημαντικά στην

μάθηση, κυρίως στον τρόπο αναζήτησης πληροφοριών, στην οργάνωση των αποτελεσμάτων και στη δημιουργία ενός προγράμματος μάθησης ειδικό για τον καθένα. Στον επιχειρηματικό τομέα, θα υπάρχει καλύτερη οργάνωση των εταιριών, καλύτερες εμπειρίες για τους χρήστες στις διαδικτυακές αγορές και καλύτερος συντονισμός μεταξύ διαφορετικών εταιριών [1].

2.3.3 Οντολογίες

Η οντολογία είναι μια τυπική, κατηγορηματική προδιαγραφή μιας διαμοιρασμένης εννοιολογικής αναπαράστασης. Ο όρος «εννοιολογική αναπαράσταση» (conceptualization) αναφέρεται σε ένα αφηρημένο μοντέλο φαινομένων του κόσμου, στο οποίο έχουν προσδιοριστεί οι έννοιες που σχετίζονται με τα φαινόμενα αυτά. Ο όρος «κατηγορηματική» (explicit) σημαίνει ότι το είδος των εννοιών που χρησιμοποιούνται και οι περιορισμοί που αφορούν την χρήση αυτών των εννοιών είναι προσδιορισμένα με σαφήνεια. Με τον όρο «αυστηρή» (formal) εννοούμε ότι η οντολογία πρέπει να είναι μηχανικά αναγνώσιμη. Τέλος, ο όρος «διαμοιρασμένη» (shared) αναφέρεται στο ότι η οντολογία πρέπει να αποτυπώνει γνώση κοινής αποδοχής στα πλαίσια μιας κοινότητας. Μια οντολογία μπορεί να πάρει διάφορες μορφές αλλά οπωσδήποτε θα περιλαμβάνει ένα λεξιλόγιο όρων και κάποιας μορφής προδιαγραφές για τη σημασία τους.

Διακρίνουμε τέσσερις κατηγορίες βασικών συστατικών μιας οντολογίας:

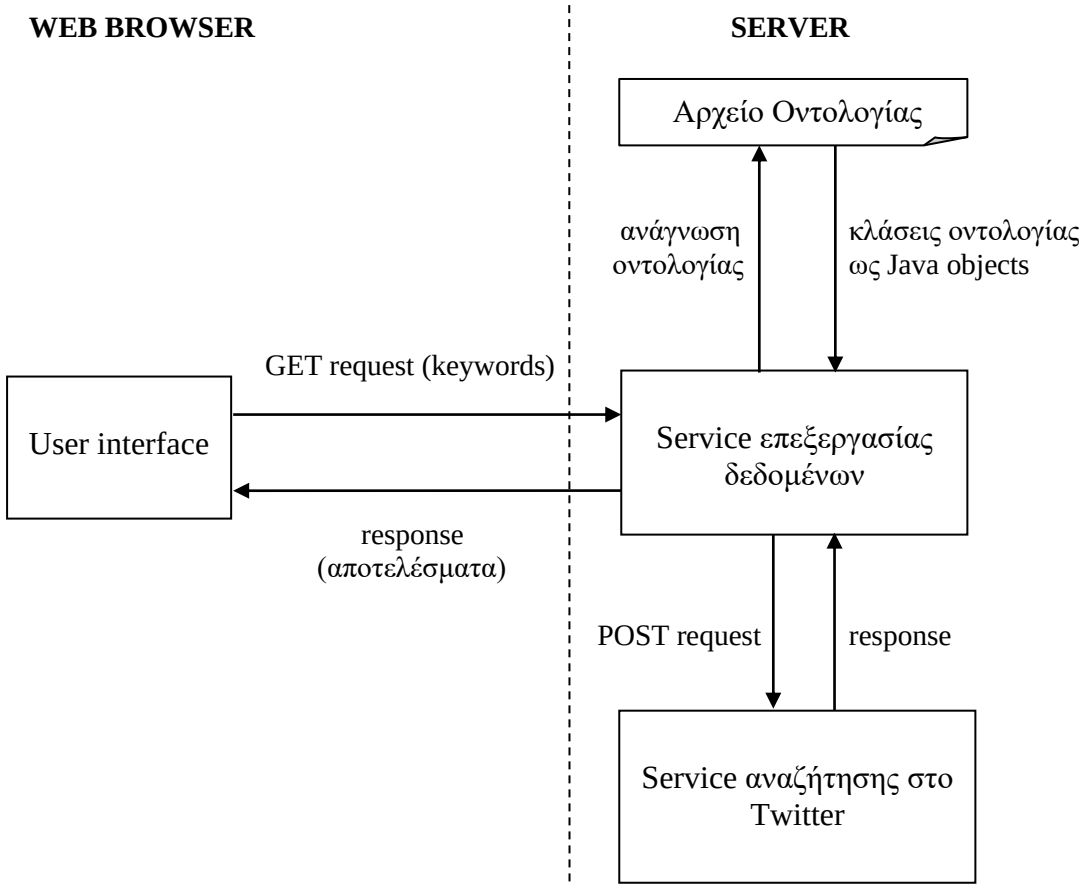
- Κλάσεις (classes): Έννοιες που σχετίζονται με ένα πεδίο ή κάποιες εργασίες, οι οποίες είναι συνήθως οργανωμένες σε κάποιο ταξινομικό σύστημα. Σε μια οντολογία που αφορά το πανεπιστήμιο, ο «φοιτητής» και ο «καθηγητής» αποτελούν δύο κλάσεις.
- Σχέσεις (relations): Ένας τύπος αλληλεπίδρασης μεταξύ εννοιών ενός πεδίου, όπως: subclass-of, is-a
- Αξιώματα (axioms): αναπαριστούν προτάσεις που είναι πάντα αληθείς. Για παράδειγμα, αν ο Φ είναι δευτεροετής φοιτητής τότε μπορεί να εγγραφεί στο επιλεγόμενο μάθημα Μ.
- Στιγμιότυπα (instances): αναπαριστούν συγκεκριμένα στοιχεία, πχ ο φοιτητής με το όνομα Νίκος είναι ένα στιγμιότυπο της κλάσης «φοιτητής» [16].

3

Δομή της εφαρμογής

3.1 Αρχιτεκτονική του συστήματος

Η διαδικτυακή μας εφαρμογή δομήθηκε με βάση τις προδιαγραφές ενός συστήματος client-server. Αναπτύχθηκαν, δηλαδή, ξεχωριστά τμήματα λογισμικού στην πλευρά του πελάτη και στην πλευρά του server. Όπως έχει ήδη αναφερθεί στο εισαγωγικό κεφάλαιο, η εφαρμογή που δημιουργήσαμε είναι υπηρεσιοστρεφής, ή καλύτερα service-oriented. Αυτό σημαίνει ότι δεν αποτελείται από ένα ενιαίο πρόγραμμα που εκτελεί όλες τις διαφορετικές λειτουργίες της. Κάτι τέτοιο θα δημιουργούσε σοβαρό πρόβλημα σε περίπτωση τροποποίησης κάποιας λειτουργίας, αφού μπορεί να επηρεάζονταν και άλλες, με αποτέλεσμα η εφαρμογή να μην λειτουργούσε σωστά ή να σταματούσε να λειτουργεί. Επιπλέον, δεν θα μπορούσαμε να χρησιμοποιήσουμε εύκολα σε κάποιο άλλο project τμήματα κώδικα της εφαρμογής που εκτελούν επιμέρους λειτουργίες, καθώς θα ήταν δύσκολο και χρονοβόρο να απομονωθούν από το υπόλοιπο πρόγραμμα. Οι σύγχρονες εφαρμογές διαδικτύου, προκειμένου να αντιμετωπίσουν το παραπάνω πρόβλημα που καθιστά δύσκολη τη διόρθωση, τη συντήρηση και την επέκταση του κώδικά τους, είναι δομημένες με τέτοιο τρόπο ώστε να αποτελούνται από περισσότερα μικρότερα προγράμματα που ονομάζονται services. Κάθε service είναι πλήρως ανεξάρτητο από τα υπόλοιπα και, παρόλο που επικοινωνούν μεταξύ τους, η σωστή λειτουργία καθενός δεν εξαρτάται πλέον από τη λειτουργία των υπολοίπων. Σε αυτή την ενότητα παρατίθενται τα βασικά χαρακτηριστικά κάθε τμήματος και εξηγείται συνοπτικά ο ρόλος του καθενός στο σύστημα, όπως φαίνεται και στην εικόνα 3.1.



Εικόνα 3.1: Επικοινωνία μεταξύ των τμημάτων της εφαρμογής

3.1.1 Διεπαφή χρήστη

Η διεπαφή χρήστη διαμορφώνεται από το τμήμα του λογισμικού στην πλευρά του πελάτη, τον κώδικα δηλαδή που «τρέχει» μόλις ανοίξουμε την εφαρμογή μας με έναν web browser, και δίνει στους χρήστες τη δυνατότητα να αλληλεπιδρούν με τις υπηρεσίες της εφαρμογής. Συγκεκριμένα, μέσω της διεπαφής ο χρήστης έχει τη δυνατότητα είτε να πληκτρολογήσει τους όρους της αναζήτησης, είτε να τους επιλέξει από έναν κατάλογο με προκαθορισμένους όρους, καθώς και να δει τα αποτελέσματα που επιστρέφει το σύστημα.

3.1.2 Υπηρεσίες στον server

Η ανάκτηση των δημοσιεύσεων από το Twitter και το φιλτράρισμά τους σύμφωνα με τις επιλογές των χρηστών πραγματοποιείται από τα web services της εφαρμογής. Το backend κομμάτι της εφαρμογής αποτελείται από δύο τέτοια services, αυτό που πραγματοποιεί την αναζήτηση στο twitter και εκείνο που επεξεργάζεται τα δεδομένα και επικοινωνεί με το frontend. Τα αποτελέσματα της αναζήτησης που επιστρέφουν τα services παρουσιάζονται σε κατανοητή για τους χρήστες μορφή μέσω του user interface.

3.2 Τρόπος λειτουργίας του συστήματος

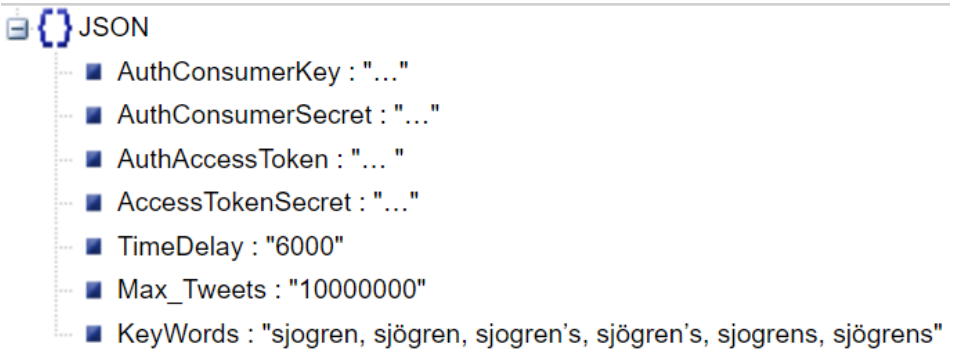
Η λειτουργία της εφαρμογής αποτελείται από δύο φάσεις: την αρχικοποίηση του συστήματος, όπου πρέπει να δώσουμε στο service αναζήτησης στο Twitter τις κατάλληλες παραμέτρους για την εκκίνηση της αναζήτησης, και την κανονική λειτουργία, κατά την οποία οι χρήστες επιλέγουν ή εισάγουν κάποιες λέξεις-κλειδιά και το σύστημα φιλτράρει τις δημοσιεύσεις που έχει συλλέξει και τους επιστρέφει τα αποτελέσματα.

3.2.1 Αρχικοποίηση του συστήματος

Για να ξεκινήσει η λειτουργία της εφαρμογής, απαιτείται πρώτα μια αρχικοποίηση της υπηρεσίας αναζήτησης στο Twitter. Χρειάζεται να στείλουμε ένα POST request στο service, με το σώμα του αιτήματος να έχει την μορφή που φαίνεται στην εικόνα 3.2. Για την παρουσίαση του JSON σε πιο κατανοητή μορφή, χρησιμοποιήθηκε το online εργαλείο JSON Viewer [5]. Υποθέτοντας ότι το domain του διακομιστή είναι το localhost, και η θύρα στην οποία δέχεται HTTP αιτήματα είναι η 8080, το url στο οποίο θα αποστείλουμε το POST request είναι:

```
http://localhost:8080/SMA_Adapter/TwitterServlet
```

Περισσότερα σχετικά με τη σημασία του url αυτού θα αναφερθούν στην παράγραφο 3.2.2.2. Καθεμία από τις τιμές των παραμέτρων που υπάρχουν στο JSON Body του αιτήματος αποθηκεύεται σε μια μεταβλητή της μνήμης του προγράμματος. Πρώτα από όλα, πρέπει να έχουμε τα απαραίτητα δικαιώματα σε κάποιον λογαριασμό, ώστε να πραγματοποιήσουμε την αναζήτηση στο Twitter μέσω του λογαριασμού αυτού. Οι τέσσερις πρώτες παράμετροι του JSON περιέχουν τις πληροφορίες που χρειαζόμαστε για να έχουμε δικαίωμα να χρησιμοποιήσουμε έναν λογαριασμό Twitter για αυτόν τον σκοπό, με τις τιμές τους να αποκρύπτονται για ευνόητους λόγους. Η παράμετρος 'TimeDelay' περιέχει τον αριθμό των millisecond που θα περιμένει το πρόγραμμα πριν ξεκινήσει μια νέα αναζήτηση. Η 'Max_Tweets' ορίζει τον μέγιστο αριθμό tweets που μπορεί να συγκεντρώσει το service, ενώ τέλος στην παράμετρο 'KeyWords' διατηρούνται οι προκαθορισμένες λέξεις-κλειδιά με βάση τις οποίες πραγματοποιείται η αναζήτηση στο Twitter.



Εικόνα 3.2: Σώμα του POST Request για αρχικοποίηση του Search Service

Αφού στείλουμε το παραπάνω POST request για να αρχικοποιήσουμε την υπηρεσία, στη συνέχεια θα πρέπει να στείλουμε και ένα GET request στο ίδιο url, ώστε να ξεκινήσει τελικά η αναζήτηση των δημοσιεύσεων που περιέχουν τα επιλεγμένα keywords στο Twitter. Ο πιο εύκολος τρόπος για την αποστολή των δύο παραπάνω αιτημάτων είναι να χρησιμοποιήσουμε κάποιο

πρόγραμμα για χειροκίνητη απευθείας αποστολή HTTP requests, όπως είναι το Postman. Η διαδικασία αυτή πρέπει να επαναλαμβάνεται κάθε φορά που επανεκκινούμε τον server, καθώς τα αποτελέσματα της αναζήτησης βρίσκονται μόνο στη μνήμη του προγράμματος και όχι σε κάποιο άλλο αρχείο ή αποθηκευτικό μέσο, με αποτέλεσμα να σβήνονται αν για κάποιο λόγο κλείσει ο server και η αναζήτησή τους να πρέπει να ξεκινήσει ξανά από την αρχή.

3.2.2 Κανονική λειτουργία της εφαρμογής

Σε αυτή την ενότητα θα περιγράψουμε τη λειτουργία της διεπαφής χρήστη και των services του συστήματος, παραθέτοντας τα βασικά χαρακτηριστικά κάθε τμήματος. Ταυτόχρονα, θα αναλύσουμε και τον τρόπο επικοινωνίας μεταξύ αυτών των πλήρως διαχωρισμένων τμημάτων, ο οποίος παρουσιάζεται σχηματικά στην εικόνα 3.1. Τα επιμέρους τμήματα της εφαρμογής απεικονίζονται ως ορθογώνια παραλληλόγραμμα ενώ τα requests και τα responses που ανταλλάζουν στο πλαίσιο της μεταξύ τους επικοινωνίας αναπαρίστανται από βέλη με κατάλληλη φορά. Η επικοινωνία πραγματοποιείται μέσω του πρωτοκόλλου HTTP, με την ανταλλαγή δεδομένων σε μορφή αντικειμένων JSON. Περαιτέρω τεχνικές λεπτομέρειες και λειτουργικότητες για κάθε τμήμα περιγράφονται πιο αναλυτικά στα κεφάλαια 4 και 5.

3.2.2.1 Επικοινωνία της διεπαφής χρήστη με τον server

Η διεπαφή χρήστη αποτελείται από ένα και μόνο html αρχείο, με όνομα home.html, το οποίο περιέχει τη δομή και τη λειτουργικότητα της σελίδας home, που αποτελεί και τη μοναδική σελίδα της εφαρμογής. Η ύπαρξη περισσότερων σελίδων κρίθηκε μη αναγκαία, καθώς ο πίνακας με τα αποτελέσματα της αναζήτησης παρουσιάζεται δίπλα στον κατάλογο με τις λέξεις-κλειδιά σε μια ενιαία σελίδα, η οποία είναι χωρισμένη σε δύο στήλες. Οι διάφορες λειτουργικότητες και τα τεχνικά χαρακτηριστικά της διεπαφής χρήστη θα αναλυθούν στο επόμενο κεφάλαιο.

Ταυτόχρονα, εκτός από την εμφάνιση του user interface, στο ίδιο html αρχείο ορίζεται και ο τρόπος επικοινωνίας του με τον διακομιστή, με τη βοήθεια των συναρτήσεων της JavaScript. Αρχικά, ορίζουμε μια συνάρτηση με όνομα HttpClient, η οποία δημιουργεί ένα αντικείμενο της JavaScript που ονομάζεται XMLHttpRequest. Ο ρόλος αυτού του αντικειμένου είναι να ανοίξει έναν δίαυλο επικοινωνίας μεταξύ του client και του server, ώστε να καταστεί δυνατή η ανταλλαγή δεδομένων μέσω του πρωτοκόλλου HTTP. Στη συνέχεια, δημιουργούμε το url που είναι απαραίτητο για την αποστολή GET HTTP requests στον server:

```
http://localhost:8080/OntService/OntServlet?items={items}&mykeyword
s={mykeywords}&logic={logic}&akalogic={akalogic}
```

Θα μπορούσαμε να στείλουμε ένα GET request αν απλώς πληκτρολογούσαμε το συγκεκριμένο url στη γραμμή διευθύνσεων του browser και πατούσαμε ‘Enter’. Εδώ πραγματοποιούμε την ίδια διαδικασία προγραμματιστικά, δηλαδή μέσω του κώδικα της εφαρμογής. Το νόημα του παραπάνω url είναι ότι στέλνοντας request στη θύρα 8080 του server, καλούμε την υπηρεσία (service) που βρίσκεται στον server, και συγκεκριμένα στη διαδρομή (path) “/OntService/OntServlet”. Ο κώδικας του service βρίσκεται στο αρχείο java servlet με το όνομα OntServlet.java. Περνάμε επίσης στο service μέσω του url τις παραμέτρους items, mykeywords, logic και akalogic, των οποίων οι τιμές δίνονται από τον χρήστη μέσω του interface. Ο πλήρης κώδικας JavaScript για την παραπάνω λειτουργία παρατίθεται στο Παράρτημα.

3.2.2.2 Service επεξεργασίας δεδομένων

Θα ασχοληθούμε πρώτα με την περιγραφή της υπηρεσίας που επεξεργάζεται τα δεδομένα που στέλνονται από και προς το frontend τμήμα της εφαρμογής. Το service αυτό αναπτύχθηκε εξ ολοκλήρου στα πλαίσια της παρούσας εργασίας, με σκοπό την επικοινωνία με το user interface, αλλά και την επεξεργασία των αποτελεσμάτων της αναζήτησης στο Twitter που στέλνονται από το άλλο service.

Το συγκεκριμένο service καλείται με την εκτέλεση ενός GET request από το frontend. Όπως αναφέρθηκε στην παράγραφο 3.2.2.1, για την κατασκευή της υπηρεσίας χρησιμοποιήθηκε ένα java servlet αρχείο. Μέσω του παραπάνω αιτήματος GET καλείται η μέθοδος doGet που διαθέτει ο servlet για τον χειρισμό αιτημάτων αυτού του είδους. Καθώς αυτό είναι και το μοναδικό request που πρόκειται να σταλεί στο service επεξεργασίας, η μέθοδος doPost που επίσης υπάρχει στον servlet για τον χειρισμό POST αιτημάτων αφήνεται κενή. Φυσικά, μπορεί να υλοποιηθεί ώστε να παρέχει την απαραίτητη λειτουργικότητα σε ενδεχόμενη μελλοντική επέκταση της εφαρμογής. Επομένως, όλος ο κώδικας του service, οι λειτουργίες του οποίου θα περιγραφούν εδώ αλλά και πιο αναλυτικά στα επόμενα κεφάλαια, βρίσκεται στο σώμα της doGet.

Αρχικά, πραγματοποιείται ανάγνωση του αρχείου κειμένου infos.txt. Το αρχείο αυτό περιέχει τις εξής πληροφορίες: το όνομα του αρχείου οντολογίας, το domain του server καθώς και τη θύρα (port) από την οποία δέχεται HTTP requests. Ο λόγος που τα δεδομένα αυτά διαβάζονται από αρχείο αντί να είναι γραμμένα κατευθείαν στον κώδικα, είναι για να καταστεί η εφαρμογή μας παραμετροποιήσιμη. Δηλαδή σε περίπτωση που θέλουμε να τη μεταφέρουμε σε διαφορετικό server ή να της δώσουμε ως είσοδο κάποιο άλλο αρχείο οντολογίας, να μην χρειάζεται να επέμβουμε απευθείας στον κώδικα της εφαρμογής, καθώς κάτι τέτοιο θα μπορούσε να δημιουργήσει προβλήματα στην ομαλή της λειτουργία, αν κατά λάθος πειράζαμε κάποιο τμήμα του κώδικα που δεν θα έπρεπε.

Έπειτα, με βάση το όνομα του αρχείου οντολογίας που διαβάστηκε από το infos.txt, γίνεται ανάγνωση του αρχείου, το οποίο βρίσκεται αποθηκευμένο σε γνωστή διαδρομή (path) στον δίσκο, και φορτώνονται οι όροι της οντολογίας στη μνήμη του προγράμματος. Στη συνέχεια, διαβάζει τις τιμές των παραμέτρων που έχουν σταλεί μέσω του url και αναζητά τα επιλεγμένα keywords στην οντολογία, βάζοντας τα σε ένα αντικείμενο JSON. Τέλος, προστίθεται στο αντικείμενο JSON και η παράμετρος logic που διαβάστηκε από το url. Μόλις δημιουργήσουμε το JSON και συμπληρώσουμε όλες τις παραμέτρους του, εκτελούμε POST HTTP request στο service αναζήτησης στο Twitter, χρησιμοποιώντας το κατάλληλο url, με domain και port αυτά που διαβάστηκαν από το αρχείο infos.txt. Λαμβάνοντας υπόψη όσα υποθέσαμε για το url στην παράγραφο 3.2.1, το POST request αποστέλλεται στο url:

```
http://localhost:8080/SMA_Adapter/TwitterServlet
```

Βλέπουμε ότι πρόκειται για το ίδιο url όπου στείλαμε τα requests κατά την αρχικοποίηση του συστήματος. Αυτό συμβαίνει καθώς για όλα τα αιτήματα που στέλνονται στο service αναζήτησης είναι αναγκαίο να χρησιμοποιείται το ίδιο url Σημειώνεται ότι το domain και το port του παραπάνω url μπορεί να είναι ίδια ή διαφορετικά από αυτά που φαίνονται στο url της παραγράφου 3.2.2.1, καθώς τα δύο services μπορούν να βρίσκονται στον ίδιο ή σε διαφορετικό server. Στο σώμα του αιτήματος τοποθετείται το παραπάνω JSON object, μέσω του οποίου στέλνουμε στο service τόσο τα keywords όσο και τη λογική με την οποία πρέπει να πραγματοποιηθεί η αναζήτηση.

Μετά την αναζήτηση στο Twitter, το service επεξεργασίας διαβάζει την απάντηση (response) που έστειλε στο αίτημά του το service αναζήτησης. Η απάντηση αυτή περιλαμβάνει όλα τα tweets που βρέθηκαν τα οποία περιέχουν στο κείμενό τους τις λέξεις-κλειδιά. Τα tweets επιστρέφουν ως JSON objects τοποθετημένα σε έναν πίνακα από JSON, ο οποίος με τη σειρά του βρίσκεται σε ένα μεγαλύτερο JSON αντικείμενο, που αποτελεί το response που αναφέρθηκε προηγουμένως. Ο πίνακας αυτός με τα αποτελέσματα της αναζήτησης τοποθετείται σε άλλο JSON, το οποίο αποστέλλεται ως response στο frontend, για την παρουσίαση των αποτελεσμάτων μέσω του user interface. Ο κώδικας Java για την αποστολή POST request και τη λήψη και επεξεργασία του αντίστοιχου response δίνεται στο Παράρτημα.

3.2.2.3 Service αναζήτησης στο Twitter

Ακολουθεί η συνοπτική περιγραφή της λειτουργίας και των απαιτήσεων του service που πραγματοποιεί την αναζήτηση στο Twitter και επιστρέφει τα ζητούμενα tweets στο service επεξεργασίας των δεδομένων. Η συγκεκριμένη υπηρεσία προϋπήρχε ως ανεξάρτητο και πλήρως λειτουργικό service, το οποίο σχεδιάστηκε και αναπτύχθηκε από τον συνάδελφο Γιάννη Βιόλο, δεν δημιουργήθηκε δηλαδή από μηδενική βάση στο πλαίσιο της παρούσας διπλωματικής εργασίας. Παρόλα αυτά υπέστη σημαντικές τροποποιήσεις, με σκοπό την βελτίωση της ακρίβειας της αναζήτησης και την αύξηση του πλήθους των αποτελεσμάτων.

Όπως το service επεξεργασίας, έτσι και αυτό της αναζήτησης δομήθηκε με τη χρήση ενός servlet, ο οποίος αποτελεί και τον πυρήνα του service. Εδώ χρησιμοποιούνται και οι δύο βασικές μέθοδοι που διαθέτει ένας servlet, η doPost και η doGet. Η doPost αποτελείται από δύο τμήματα, εκ των οποίων το πρώτο χρησιμοποιείται μόνο κατά την εκκίνηση και αρχικοποίηση του service, ενώ το δεύτερο, που περιέχει και τη βασική λειτουργικότητα της υπηρεσίας, καλείται κάθε φορά που ο χρήστης πραγματοποιεί μια καινούρια αναζήτηση. Η doGet καλείται μόνο κατά την αρχικοποίηση του service, αφού κληθεί πρώτα η doPost. Έτσι, γίνεται αντιληπτό ότι η αρχικοποίηση απαιτεί την αποστολή με τη σειρά ενός POST και ενός GET request, όπως αναφέραμε και στην παράγραφο 3.2.1, ενώ κάθε φορά που κάποιος χρήστης πραγματοποιεί μια νέα αναζήτηση, αποστέλλεται στο service αναζήτησης ένα POST request.

Είναι σημαντικό να τονίσουμε ότι η αναζήτηση στο Twitter δεν πραγματοποιείται τη στιγμή που στέλνεται το POST request από το service επεξεργασίας. Κάτι τέτοιο θα ήταν υπερβολικά χρονοβόρο και ο χρήστης θα περίμενε πάρα πολλή ώρα μέχρι να βρεθούν και να εμφανιστούν όλα τα αποτελέσματα της αναζήτησης. Για το λόγο αυτό, η αναζήτηση πραγματοποιείται στο παρασκήνιο, χωρίς να επηρεάζει δηλαδή τη λειτουργία του υπόλοιπου προγράμματος, και επαναλαμβάνεται ανά τακτά χρονικά διαστήματα, ώστε να ξεπεραστεί και ο περιορισμός που θέτει το Twitter για μέγιστο όριο 100 αποτελεσμάτων σε κάθε αναζήτηση. Η αναζήτηση γίνεται με βάση μια ή περισσότερες προκαθορισμένες λέξεις-κλειδιά και τα ζητούμενα tweets αποθηκεύονται στη μνήμη του προγράμματος. Μόλις δοθούν κάποια keywords από τον χρήστη, το service αναζητά μεταξύ των tweets που έχει συλλέξει και αποθηκεύσει, εκείνα των οποίων το κείμενο περιέχει μία τουλάχιστον ή και όλες τις επιλεγμένες από τον χρήστη λέξεις-κλειδιά.

Αφού σταλεί το POST request που περιέχει στο σώμα του τα επιλεγμένα από τον χρήστη keywords, το service αναζήτησης ενεργοποιείται και αναζητά τα keywords αυτά στα tweets που έχει συλλέξει, σύμφωνα και με τη λογική της αναζήτησης που έχει ορίσει ο χρήστης, όπως θα δούμε στο επόμενο κεφάλαιο. Κάθε tweet που ταιριάζει με τους όρους της αναζήτησης, τοποθετείται σε έναν πίνακα. Αφού ολοκληρωθεί η αναζήτηση, ο πίνακας με τα αποτελέσματα εισάγεται σε ένα JSON, το οποίο στέλνεται ως απάντηση στο service επεξεργασίας δεδομένων.

3.3 Ανέβασμα της εφαρμογής στον server

Προκειμένου να χρησιμοποιήσουμε την εφαρμογή μας, είναι απαραίτητο να την ανεβάσουμε σε κάποιον διακομιστή (deployment), αφού πρόκειται για διαδικτυακή εφαρμογή. Αρχικά, χρειάζεται να δημιουργήσουμε ένα αρχείο war (Web Archive) της εφαρμογής μας. Ένα αρχείο αυτού του τύπου περιέχει μια διαδικτυακή εφαρμογή σε συμπιεσμένη μορφή, προκειμένου να μεταφερθεί σε κάποιον web server. Η εξαγωγή αυτού του αρχείου μπορεί να γίνει αυτόματα μέσω του Eclipse IDE που χρησιμοποιήθηκε για την ανάπτυξη της εφαρμογής. Χρησιμοποιώντας ως διακομιστή τον Apache Tomcat, θα πρέπει να τοποθετήσουμε το war αρχείο στον φάκελο webapps του και στη συνέχεια να τον εκκινήσουμε. Αν θέλουμε να μεταφερθούμε στη σελίδα της εφαρμογής, αρκεί να πληκτρολογήσουμε στη γραμμή διευθύνσεων του browser το url:

```
http://localhost:8080/OntService/home.html
```

Φυσικά, αν ο Tomcat server δεν είναι εγκατεστημένος στον υπολογιστή μας, θα χρειαστεί να αντικαταστήσουμε το localhost με τη διεύθυνση IP του server, καθώς και το 8080 με την αντίστοιχη θύρα στην οποία ο server δέχεται HTTP αιτήματα. Τα δυο services της εφαρμογής μπορούν να κληθούν απευθείας μέσω των αντίστοιχων url. Πριν εκτελέσουμε κάποια αναζήτηση, είτε μέσω της διεπαφής ή καλώντας απευθείας τα services, θα πρέπει να ακολουθήσουμε τη διαδικασία αρχικοποίησης που περιγράφεται στην ενότητα 3.2.1.

4

Αλληλεπίδραση χρήστη – εφαρμογής

4.1 Εκκίνηση της εφαρμογής

Κατά την εκκίνηση της εφαρμογής, εμφανίζεται η διεπαφή χρήστη, ή αλλιώς user interface, η οποία αποτελεί το μέσο που έχουν οι χρήστες για να επικοινωνούν με την εφαρμογή, να εισάγουν δηλαδή δεδομένα, όπως είναι οι όροι και οι συνθήκες της αναζήτησης, και να βλέπουν τα αποτελέσματα που επιστρέφουν τα services. Συγχρόνως, στέλνεται αυτόματα ένα GET request στον server, με χρήση του url που αναφέρθηκε στο κεφάλαιο 3:

```
http://localhost:8080/OntService/OntServlet?items=null&mykeywords=n
ull&logic=and&akalogic=no
```

ώστε να ξεκινήσει η ανάγνωση της οντολογίας, από όπου θα αντληθούν οι απαραίτητες πληροφορίες για να εμφανιστεί στη συνέχεια ο κατάλογος των keywords στο user interface. Οι τιμές των παραμέτρων items και mykeywords είναι “null”, καθώς δεν έχει επιλεγεί καμία λέξη-κλειδί ούτε έχει εισαχθεί κάποια στο ειδικό πεδίο κειμένου, ενώ οι παράμετροι logic και akalogic, που έχουν ως προεπιλεγμένες τιμές τις “and” και “no” αντίστοιχα, δηλώνουν αν η αναζήτηση θα πραγματοποιηθεί με λογική and ή or και αν θα ληφθούν υπόψη κατά την αναζήτηση ή όχι τυχόν συνώνυμα ή ακρωνύμια των επιλεγμένων keywords.

4.1.1 Οργάνωση της διεπαφής χρήστη

Όπως αναφέρθηκε και στο προηγούμενο κεφάλαιο, το interface είναι διαχωρισμένο σε δύο στήλες, από τις οποίες η αριστερή περιέχει τις επιλογές αναζήτησης και τον κατάλογο με τις λέξεις-κλειδιά και έχει πλάτος όσο το 30% της οθόνης, ενώ στη δεξιά, που καταλαμβάνει το υπόλοιπο 70%, παρουσιάζεται ο πίνακας με τα αποτελέσματα.

Στην αριστερή πλευρά της οθόνης, στην κορυφή βλέπουμε το κουμπί Clear. Το κουμπί αυτό χρησιμοποιείται κάθε φορά που θέλουμε να «καθαρίσουμε» τη σελίδα από τις επιλογές και τα αποτελέσματα της αναζήτησης, κάτι που επιτυγχάνεται φορτώνοντας απλώς τη σελίδα ξανά από την αρχή, όπως αν πατούσαμε ανανέωση στον browser. Η επιλογή “Clear” πρέπει να δίνεται στον χρήστη με ξεκάθαρο τρόπο, αφενός γιατί δεν είναι σωστή πρακτική για τις διαδικτυακές εφαρμογές να εξαρτάται η λειτουργικότητά τους από τα κουμπιά του browser, αφετέρου γιατί ο χρήστης δεν μπορεί να γνωρίζει ότι αν κάνει ανανέωση το αποτέλεσμα θα είναι το ίδιο. Στην πραγματικότητα, με το πάτημα του Clear πραγματοποιείται η ίδια διαδικασία όπως και στην εκκίνηση της εφαρμογής.

Κάτω από το Clear, βλέπουμε τη φράση “Search logic?” και μετά δύο κουμπιά με τις επιλογές “AND” και “OR”, με προεπιλεγμένο το κουμπί AND. Πρόκειται για μια συνθήκη που επιλέγεται από το χρήστη, η οποία καθορίζει τη λογική της αναζήτησης. Αν ο χρήστης επιλέξει το

OR, το σύστημα θα πρέπει να βρει και να επιστρέψει όλα τα tweets που περιέχουν τουλάχιστον μία από τις επιλεγμένες λέξεις-κλειδιά. Διαφορετικά, η αναζήτηση θα πραγματοποιηθεί με λογική AND, δηλαδή θα αναζητηθούν και θα επιστραφούν μόνο τα tweets που περιέχουν όλες τις λέξεις κλειδιά που επέλεξε ο χρήστης.

Αμέσως μετά την επιλογή για τη λογική της αναζήτησης, εμφανίζεται η ερώτηση "Search with akas and acronyms?", καθώς και δύο κουμπιά με τις επιλογές "YES" και "NO", με προεπιλεγμένο το NO. Και αυτή η συνθήκη καθορίζεται από το χρήστη και υποδεικνύει αν θα επιστραφούν, μαζί με τα υπόλοιπα αποτελέσματα της αναζήτησης, και εκείνα τα tweets που, αντί για ένα επιλεγμένο keyword, περιέχουν κάποιο συνώνυμο ή ακρωνύμιο του συγκεκριμένου keyword. Όπως αναφέραμε, έχουμε ορίσει ως προεπιλογή να μην συμπεριλαμβάνονται στα αποτελέσματα της αναζήτησης τα tweets με τα συνώνυμα και τα ακρωνύμια των λέξεων-κλειδιών που επιλέχθηκαν.

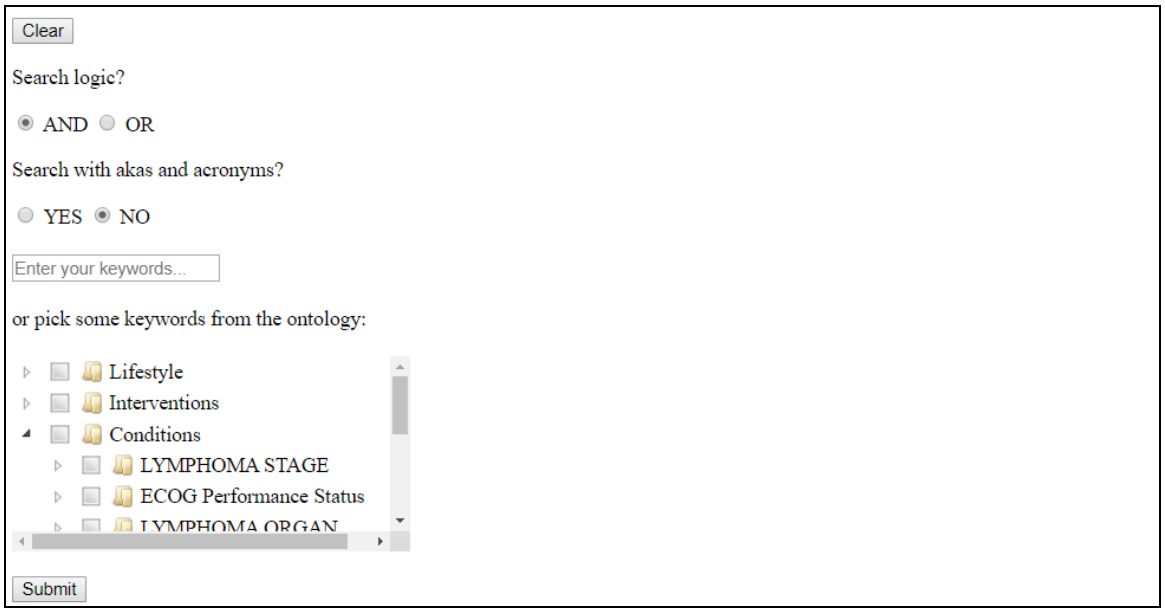
Στη συνέχεια, έχει τοποθετηθεί ένα πεδίο κειμένου, στο οποίο ο χρήστης έχει τη δυνατότητα να εισάγει τα δικά του keywords, που μπορεί να βρίσκονται ή να μη βρίσκονται στην οντολογία. Αν επιθυμεί να εισάγει περισσότερες από μία λέξεις-κλειδιά, θα πρέπει να τις διαχωρίσει μεταξύ τους με κόμμα, ενώ αν πρέπει το keyword να αποτελείται από περισσότερες από μία λέξεις, τότε αρκεί να πληκτρολογήσει όλες τις λέξεις με κενό ανάμεσά τους, και αυτές θα θεωρηθούν ως μια ενιαία φράση-κλειδί. Θα πρέπει να σημειωθεί ότι σε περίπτωση που ο χρήστης εισάγει κατά λάθος ένα ή περισσότερα κενά πριν ή μετά το κόμμα διαχωρισμού δύο keywords, δεν θα δημιουργηθεί κανένα πρόβλημα, καθώς το service επεξεργασίας όπου αποστέλλονται τα δεδομένα εξαλείφει τυχόν κενά που υπάρχουν στην αρχή ή στο τέλος των λέξεων.

Κάτω από τις προηγούμενες επιλογές εμφανίζεται ο κατάλογος με τις λέξεις-κλειδιά. Για την παρουσίαση των keywords στον χρήστη χρησιμοποιήθηκε το plugin jsTree της βιβλιοθήκης jQuery. Με τη χρήση του jsTree τα keywords παρουσιάζονται σε μορφή καταλόγου, ο οποίος αποτελείται από επιμέρους λίστες σε μορφή αντεστραμμένου δέντρου. Κάθε keyword είναι και ένας κόμβος της λίστας. Ρίζα του δέντρου αποτελεί ο αρχικός κόμβος, δηλαδή κάποιο keyword που ορίζει μια ευρεία κατηγορία εννοιών, ενώ κάθε λέξη-κλειδί με πιο εξειδικευμένη έννοια που εντάσσεται νοηματικά στην κατηγορία του keyword της ρίζας, αποτελεί έναν επόμενο κόμβο της ίδιας επιμέρους λίστας, ή αλλιώς ένα φύλλο του δέντρου. Δίπλα σε κάθε keyword παρέχεται και ένα checkbox για την επιλογή του από τον χρήστη.

Τέλος, πρέπει να σημειωθεί ότι κάτω από τον κατάλογο εμφανίζεται το κουμπί "Submit", το οποίο χρησιμοποιείται όταν έχει ολοκληρωθεί η επιλογή όλων των keywords και θέλουμε να ξεκινήσει η αναζήτηση. Για να είναι το Submit πάντα ορατό, πρέπει να μένει στην ίδια θέση στην οθόνη όσους κόμβους του δέντρου των keywords κι αν ανοίξουμε. Για να εξασφαλιστεί αυτό το χαρακτηριστικό, ορίσαμε ως μέγιστο ύψος του πλαισίου του δέντρου τα 150 pixels. Αν το δέντρο φτάσει σε μεγαλύτερο ύψος κατά το άνοιγμα ενός ή περισσότερων κόμβων του, το τμήμα του που υπερβαίνει το ύψος αυτό πλέον δεν είναι ορατό, ενώ εμφανίζεται στα δεξιά του καταλόγου μια ράβδος κύλισης (scroll bar). Για να αποκαλυφθεί το κρυμμένο τμήμα του δέντρου, αρκεί να κάνουμε scroll down.

Ταυτόχρονα, το πλάτος του καταλόγου δεν θα πρέπει να υπερβαίνει το προκαθορισμένο πλάτος της αριστερής στήλης, το οποίο ισούται με το 25% της οθόνης. Αν μία ή περισσότερες λέξεις-κλειδιά υπερβαίνουν το πλάτος αυτό, τότε το αντίστοιχο τμήμα τους αποκρύπτεται και στο κάτω μέρος του καταλόγου εμφανίζεται μια οριζόντια ράβδος κύλισης. Προκειμένου να μην υπάρχουν keywords που υπερβαίνουν κατά πολύ το προκαθορισμένο πλάτος, αν κάποιο keyword με περισσότερες από μία λέξεις έχει κάποια λέξη που ξεκινάει μετά το όριο, έχει οριστεί να εμφανίζεται στην επόμενη γραμμή ολόκληρο το τμήμα του keyword που ξεκινά από αυτή τη λέξη.

Στην εικόνα 4.1 φαίνεται η δομή του user interface. Ο άδειος χώρος στα δεξιά της οθόνης προορίζεται για την εμφάνιση των αποτελεσμάτων της αναζήτησης. Θα πρέπει να σημειωθεί ότι, σε αντίθεση με τα υπόλοιπα χαρακτηριστικά της διεπαφής, τα οποία εμφανίζονται αμέσως μόλις ανοίξουμε την εφαρμογή, ο κατάλογος με τις λέξεις-κλειδιά δεν γίνεται κατευθείαν ορατός. Αυτό συμβαίνει διότι οι λέξεις-κλειδιά διαβάζονται από το αρχείο της οντολογίας, μια διαδικασία που ξεκινά αμέσως μετά την εκκίνηση της εφαρμογής και για να ολοκληρωθεί απαιτείται ένα χρονικό διάστημα μερικών δευτερολέπτων, κατά το οποίο στη θέση των keywords εμφανίζεται το μήνυμα "Loading…". Η διαδικασία εξαγωγής των keywords από την οντολογία περιγράφεται αναλυτικά στη συνέχεια.



Εικόνα 4.1: Διεπαφή χρήστη πριν την εκτέλεση αναζήτησης

4.1.2 Ανάγνωση της οντολογίας

Κατά την εκκίνηση της εφαρμογής, πραγματοποιείται ανάγνωση της οντολογίας και εξάγονται από αυτήν οι λέξεις-κλειδιά. Η οντολογία βρίσκεται σε ένα αρχείο τύπου .owl, στο οποίο τα δεδομένα αναπαρίστανται στη μορφή που ορίζεται από το πρότυπο OWL (Ontology Web Language), μια μορφή παρόμοια με αυτή που χρησιμοποιείται από τη γλώσσα XML, εμπλουτισμένη με πολλά στοιχεία που προσδίδουν στα δεδομένα νέες ιδιότητες και χαρακτηριστικά. Πιο συγκεκριμένα, κάθε keyword αναπαρίσταται στο αρχείο αυτό ως μια κλάση που μπορεί να έχει ένα σύνολο από ιδιότητες. Παρακάτω βλέπουμε μια από τις κλάσεις της οντολογίας, στη μορφή που είναι αποθηκευμένη στο αρχείο owl.

```
<owl:Class rdf:about="http://www.semanticweb.org/ntua/iccs/harmonicss/terminology/vocabulary#SYMPT-120">
    <rdfs:subClassOf rdf:resource="http://www.semanticweb.org/ntua/iccs/harmonicss/terminology/vocabulary#SYMPT-CAT-B"/>
    <rdfs:comment xml:lang="en">A dry cough is a cough where no phlegm or mucus is produced (known as non-productive).</rdfs:comment>
    <rdfs:label xml:lang="en">Dry Cough</rdfs:label>
</owl:Class>
```

Εικόνα 4.2: Κλάση της οντολογίας που χρησιμοποιήθηκε στην εργασία [18]

Η οντολογία πάνω στην οποία δουλέψαμε περιέχει αποκλειστικά ιατρικούς όρους, όπως διάφορες ασθένειες και κατηγορίες ασθενειών, συμπτώματα, μεθόδους θεραπείας και πρόληψης, είδη ιατρικών εξετάσεων κλπ., και προέκυψε στα πλαίσια του Ευρωπαϊκού ερευνητικού προγράμματος HarmonicSS [18]. Φυσικά, η λειτουργία της εφαρμογής δεν επηρεάζεται καθόλου από το περιεχόμενο της οντολογίας, παρά μόνο από τη μορφή και τις ιδιότητες των κλάσεών της. Συνεπώς, η εφαρμογή μας θα λειτουργούσε χωρίς κανένα πρόβλημα για οποιαδήποτε οντολογία της ίδιας μορφής με αυτήν που μας δόθηκε. Στη συνέχεια για λόγους ευκολίας θα αναφερόμαστε σε έννοιες και ιδιότητες της οντολογίας με την οποία εργαστήκαμε.

Ορισμένες από τις ιδιότητες κάθε κλάσης περικλείονται μέσα σε tags, ενώ άλλες είναι καταγεγραμμένες μέσα στο ίδιο το tag, δηλαδή μεταξύ των συμβόλων ‘<’ και ‘>’. Οι ιδιότητες που

έχει τόσο η κλάση που φαίνεται στην προηγούμενη εικόνα, όσο και όλες οι υπόλοιπες κλάσεις της οντολογίας είναι οι εξής:

- Αναγνωριστικό: Είναι το όνομα της κλάσης και βρίσκεται μέσα στο tag "<owl:Class …>". Δεν θα πρέπει να ξεχνάμε ότι κάθε κλάση της οντολογίας αναφέρεται σε μια οντότητα του σημασιολογικού ιστού, όπου κάθε οντότητα αναπαρίσταται από ένα uri. Στην εικόνα 4.2, όνομα της κλάσης αποτελεί το uri: "http://www.semanticweb.org/ntua/iccs/harmonicss/terminology/vocabulary#SYMPT-120"
- Κωδικός (id): Πρόκειται για ένα λεκτικό που βρίσκεται στο τέλος του uri του ονόματος της κλάσης και διαχωρίζεται από το υπόλοιπο uri με μια δίεση (#). Επομένως, το id της παραπάνω κλάσης θα είναι "SYMPT-120".
- Ετικέτα (label): Αποτελεί τη βασική πληροφορία που θέλουμε να εξάγουμε, καθώς είναι το keyword που περιέχεται στην κλάση. Περικλείεται από τα tags "<rdfs:label xml:lang="en">…</rdfs:label>", και στην κλάση που χρησιμοποιούμε ως παράδειγμα είναι το "Dry cough".

Τα παραπάνω χαρακτηριστικά συναντώνται όπως είπαμε σε όλες τις κλάσεις. Στην οντολογία υπάρχουν έξι βασικές κλάσεις, με ετικέτες "Lifestyle", "Interventions", "Conditions", "Demographics", "Pregnancy" και "Examinations", ενώ καθεμία από τις υπόλοιπες είναι υποκλάση κάποιας από αυτές τις έξι. Η ιδιότητα της υποκλάσης δηλώνεται μέσω της σχέσης "subClassOf", η οποία αναγράφεται με το tag "<rdfs:subClassOf…>", ενώ μέσα στο ίδιο το tag δηλώνεται μέσω ενός uri και το όνομα της υπερκλάσης της. Για παράδειγμα, μπορούμε να καταλάβουμε ότι η κλάση του παραδείγματος είναι υποκλάση της:
"http://www.semanticweb.org/ntua/iccs/harmonicss/terminology/vocabulary#SYMPT-CAT-B"

Επιπλέον, σε πολλές από τις κλάσεις της οντολογίας συναντάμε μία ή περισσότερες από τις παρακάτω ιδιότητες:

- Σχόλια (comments): Συνιστούν μια συνοπτική ή αναλυτική περιγραφή της έννοιας που περιέχεται στην κλάση, ενώ πολλά από τα σχόλια παραθέτουν και μια εξωτερική παραπομπή προς κάποιο άρθρο σχετικό με την έννοια αυτή. Τα σχόλια περικλείονται μέσα στα tags "<rdfs:comment xml:lang="en">…</rdfs:comment>", ενώ στην κλάση που εξετάζουμε, σχόλιο αποτελεί το κείμενο "A dry cough is a cough where no phlegm or mucus is produced (known as non-productive)."
- Συνώνυμα (akas): Παρέχουν μια εναλλακτική ονομασία για την έννοια της κλάσης, διαφορετική από αυτή στο πεδίο "label", που μπορεί να είναι πιο ευρέως διαδεδομένη και όχι αυστηρά επιστημονική, ή να χρησιμοποιείται στην καθομιλουμένη μεταξύ των ειδικών. Βρίσκεται μεταξύ των tags "<aka>…</aka>".
- Ακρωνύμια (acronyms): Αποτελούν συντομογραφίες της έννοιας που είναι γραμμένη στο "label", και συνήθως αποτελούνται μόνο από κεφαλαία γράμματα. Περιβάλλονται από τα tags "<acronym>…</acronym>".

Τέλος, σε ορισμένες κλάσεις συναντάμε κι άλλες ιδιότητες, όμως οι κλάσεις αυτές είναι λίγες και οι συγκεκριμένες ιδιότητες δεν είναι σημαντικές για την εφαρμογή, οπότε δεν θα αναλυθούν περαιτέρω.

Επομένως, γίνεται εξαγωγή από την οντολογία τόσο των keywords, όσο και των πληροφοριών που τις συνοδεύουν. Η διαδικασία αυτή πραγματοποιείται στο backend μετά από κάθε GET request που στέλνεται από το frontend, δηλαδή αμέσως μόλις εκκινήσουμε την εφαρμογή ή μετά το πάτημα των κουμπιών "Clear" και "Submit". Τα δεδομένα διαβάζονται από την οντολογία και εισάγονται σε ένα JSON object με τον ακόλουθο τρόπο:

- Έχουμε κατασκευάσει μια Java κλάση με το όνομα "aClass", η οποία περιέχει τα ακόλουθα πεδία: "name", "id", "label", "comment", "aka", "acronym", "subClasses" και "isSubClass". Η μεταβλητή subClasses περιέχει μια λίστα από αντικείμενα της κλάσης aClass, τα οποία

αποτελούν τις υποκλάσεις της συγκεκριμένης κλάσης, ενώ η isSubClass είναι μια λογική μεταβλητή που υποδεικνύει αν μια κλάση είναι υποκλάση κάποιας άλλης ή όχι, με προκαθορισμένη τιμή “false”.

- Διαβάζουμε μία προς μία τις κλάσεις της οντολογίας και καθεμία την αποθηκεύουμε ως ένα αντικείμενο της Java κλάσης aClass, στις μεταβλητές του οποίου καταχωρίζονται οι τιμές των αντίστοιχων στοιχείων της κλάσης στην οντολογία.
- Αφού αποθηκεύσουμε στην παραπάνω λίστα όλες τις κλάσεις της οντολογίας και τις ιδιότητές τους, αναζητάμε ξανά στην οντολογία όλες τις σχέσεις subClassOf. Κάθε φορά που βρίσκουμε τη σχέση αυτή, διατρέχουμε την Java λίστα με τις κλάσεις, μέχρι να βρούμε την κλάση εκείνη με όνομα ίδιο με εκείνο της υπερκλάσης της σχέσης subClassOf που εξετάζουμε. Μόλις την εντοπίσουμε, διατρέχουμε ξανά τη λίστα, αναζητώντας αυτή τη φορά την κλάση εκείνη που αποτελεί την υποκλάση στην ίδια σχέση subClassOf. Έπειτα, προσθέτουμε την υποκλάση στη λίστα των υποκλάσεων που είναι αποθηκευμένη στη μεταβλητή subClasses της υπερκλάσης. Ταυτόχρονα, στην υποκλάση η τιμή του πεδίου isSubClass γίνεται “true”. Έτσι, για κάθε κλάση της οντολογίας κατασκευάζουμε μια λίστα με όλες τις υποκλάσεις της.
- Απομένει να τοποθετήσουμε τη λίστα των κλάσεων στο αντικείμενο JSON που θα σταλεί ως response στο frontend. Στο JSON αυτό θα εισαχθεί μια λίστα από JSON, καθένα από τα οποία θα αντιστοιχεί σε μία κλάση και θα περιέχει τις μεταβλητές με τις απαραίτητες πληροφορίες της κλάσης, καθώς και έναν πίνακα από JSON, που καθένα θα περιέχει για κάθε υποκλάση τις πληροφορίες της καθώς και έναν πίνακα με τα JSON των υποκλάσεων της κ.ο.κ.. Παρατηρούμε ότι στη δομή της λίστας των JSON εμφανίζεται ένα επαναλαμβανόμενο μοτίβο, καθώς κάθε αντικείμενο της λίστας περιέχει μια άλλη λίστα με αντικείμενα, που και αυτά με τη σειρά τους περιέχουν άλλες λίστες αντικειμένων. Όπως γίνεται αντιληπτό, ο τρόπος αυτός οργάνωσης της λίστας αναπαριστά την ιεραρχία των κλάσεων στην οντολογία και ο μόνος τρόπος για να υλοποιηθεί είναι η χρήση της αναδρομής. Η τεχνική αυτή βασίζεται στην κατασκευή μίας μεθόδου της Java, η οποία καλεί συνεχώς τον εαυτό της, προκειμένου να κατασκευάσει τη λίστα ξεκινώντας από τα κατώτερα επίπεδα ιεραρχίας και συνεχίζοντας προς τα ανώτερα.
-

Για όσο χρόνο απαιτείται μέχρι να ολοκληρωθεί η παραπάνω διαδικασία, ο χρήστης βλέπει στην αριστερή πλευρά της οθόνης, στη θέση όπου πρόκειται να εμφανιστεί ο κατάλογος των keywords, το μήνυμα “Loading…”, το οποίο εξαφανίζεται αμέσως μόλις ολοκληρωθεί η φόρτωση της οντολογίας, δίνοντας τη θέση του στις λέξεις-κλειδιά. Ο κώδικας Java για την ανάγνωση της οντολογίας παρατίθεται στο Παράρτημα.

4.2 Πραγματοποίηση αναζήτησης

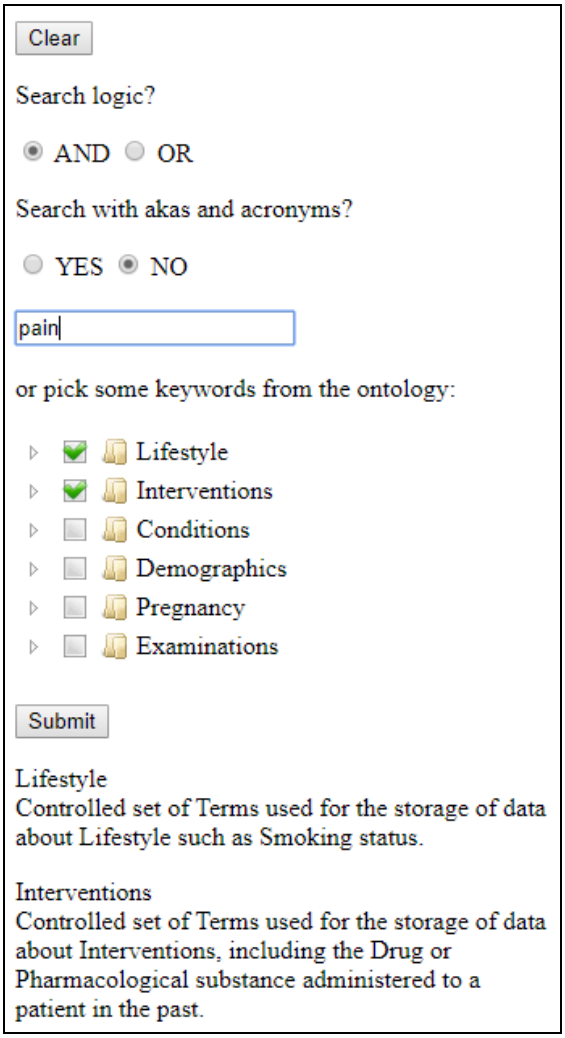
Τώρα θα εξετάσουμε τι συμβαίνει καθώς οι χρήστες αλληλεπιδρούν με την εφαρμογή. Θα περιγράψουμε τις ενέργειες που λαμβάνουν χώρα όταν κάποιος επιλέγει ένα ή περισσότερα keywords ή όταν ορίζει τις συνθήκες της αναζήτησης, αναλύοντας τις διάφορες λειτουργικότητες που έχουν προστεθεί στο διεπαφή μέσω της γλώσσας JavaScript και της βιβλιοθήκης JQuery.

4.2.1 Λειτουργικότητες της διεπαφής χρήστη

Πρώτα από όλα, θα πρέπει να τονίσουμε ότι τα ονόματα των keywords που περιέχονται στον κατάλογο είναι οι λέξεις ή φράσεις που περιέχονται στο tag “rdfs:label” κάθε κλάσης της οντολογίας, αποτελούν δηλαδή την ετικέτα της κλάσης που αντιστοιχεί στο keyword αυτό. Οι ετικέτες επιστρέφονται στο frontend μαζί με τα υπόλοιπα πεδία των κλάσεων μέσω του JSON response, όπως αναλύθηκε διεξοδικά στην παράγραφο 4.1.2.

Για κάθε λέξη-κλειδί που επιλέγει ο χρήστης, εμφανίζεται κάτω από το Submit μία συνοπτική περιγραφή του επιλεγμένου keyword, με το ίδιο το keyword στην πρώτη γραμμή της. Πρόκειται για τα πεδία label και comment της κλάσης του συγκεκριμένου keyword. Καθώς ο χρήστης επιλέγει κι άλλες λέξεις-κλειδιά, η περιγραφή της καθεμίας εμφανίζεται πάντα κάτω από την προηγούμενη. Δηλαδή, η σειρά εμφάνισης των περιγραφών δεν εξαρτάται από τη σειρά των keywords στον κατάλογο, αλλά από τη σειρά με την οποία επιλέχθηκαν.

Φυσικά, αν ο χρήστης το θελήσει, μπορεί να αποεπιλέξει ένα ή περισσότερα keywords. Για κάθε λέξη-κλειδί που αποεπιλέγει, το αντίστοιχο κείμενο με το όνομα και την περιγραφή της εξαφανίζεται. Στην επόμενη εικόνα βλέπουμε το user interface με κάποιες λέξεις-κλειδιά να έχουν επιλεγεί από τον χρήστη.



Εικόνα 4.3: Η διεπαφή χρήστη με επιλεγμένες τις έννοιες Lifestyle και Interventions και την λέξη pain στο πεδίο κειμένου

4.2.2 Επιλογή όρων και εκκίνηση της αναζήτησης

Όλες οι επιλογές του χρήστη σχετικά με τις λέξεις-κλειδιά, καθώς και με το αν η αναζήτηση θα γίνει με λογική and ή or και αν θα περιλαμβάνει και τα συνώνυμα και ακρωνύμια των επιλεγμένων keywords, θα πρέπει κάπου να αποθηκεύονται ώστε να είναι άμεσα διαθέσιμες κατά

την έναρξη της αναζήτησης. Για το σκοπό αυτό χρησιμοποιήθηκε το αντικείμενο sessionStorage της JavaScript, το οποίο παρέχει έναν προσωρινό αποθηκευτικό χώρο όπου μπορούμε να αποθηκεύσουμε τιμές σε ζεύγη τύπου key-value με τον ίδιο τρόπο όπως σε ένα JSON object. Σε αντίθεση όμως με το JSON, το sessionStorage δεν μπορεί να σταλεί στον server, αλλά είναι «ορατό» και προσβάσιμο μόνο από τον browser. Μάλιστα, στα ζεύγη key-value που περιέχονται στο sessionStorage έχει πρόσβαση μόνο η εφαρμογή που τα αποθήκευσε εκεί. Αυτό σημαίνει ότι εφαρμογές που τυχόν υπάρχουν σε άλλες καρτέλες του browser δεν μπορούν να τροποποιήσουν ή να διαγράψουν τα δεδομένα που αποθηκεύονται μέσω της εφαρμογής μας στο sessionStorage, ενώ μόλις κλείσουμε την καρτέλα της εφαρμογής, όλες οι τιμές που έχουν τοποθετηθεί από αυτήν στο sessionStorage χάνονται.

Έχουμε ορίσει, κάθε φορά που φορτώνεται η σελίδα της εφαρμογής, δηλαδή είτε κατά την εκκίνηση, είτε όταν πατάμε "Clear", να πραγματοποιείται αρχικοποίηση των τιμών που αποθηκεύονται στο sessionStorage. Συγκεκριμένα, οι παράμετροι items και mykeywords, οι οποίες περιέχουν τα id των επιλεγμένων keywords και τα keywords που πληκτρολογήσαμε στο text field αντίστοιχα, αρχικοποιούνται στην τιμή "null", η παράμετρος logic που αναφέρεται στη λογική της αναζήτησης αρχικοποιείται στην τιμή "and", ενώ η παράμετρος akalogic που αφορά τα συνώνυμα παίρνει αρχική τιμή "no". Θα πρέπει να επισημάνουμε ότι η παράμετρος items περιλαμβάνει τη λίστα με τα id των επιλεγμένων keywords, όπως αυτά διαβάστηκαν από την οντολογία, και όχι τα ίδια τα ονόματά τους, αφού σε περίπτωση που επιλέγονταν δύο keywords με το ίδιο όνομα, που όμως ανήκαν σε διαφορετικές κατηγορίες εννοιών, θα οδηγούμασταν ενδεχομένως σε λάθος αποτέλεσμα. Οι τιμές των παραμέτρων αυτών που υπάρχουν στο sessionStorage, ενημερώνονται άμεσα μετά από κάθε ανάλογη ενέργεια του χρήστη.

Η αναζήτηση ξεκινάει μόλις επιλέξουμε ή γράψουμε κάποιες λέξεις-κλειδιά και πατήσουμε Submit. Πρώτα διαβάζονται από το πεδίο κειμένου οι λέξεις-κλειδιά ακριβώς όπως τις έχουμε γράψει, και στη συνέχεια διαβάζονται οι τιμές των παραμέτρων από το sessionStorage όπως έχουν διαμορφωθεί μετά τις επιλογές του χρήστη και τοποθετούνται στο url για την αποστολή GET HTTP αιτήματος στον server. Για παράδειγμα, αν ο χρήστης έχει επιλέξει τις λέξεις-κλειδιά "Ophthalmologist" και "Dry Eyes", ενώ έχει γράψει στο πεδίο κειμένου τις λέξεις "pain, disease" και ταυτόχρονα έχει ορίσει να γίνεται η αναζήτηση με λογική and και να περιλαμβάνονται σε αυτήν συνώνυμα και ακρωνύμια, το url θα είναι:

```
http://localhost:8080/OntService/OntServlet?items=SPEC-01,SYMPT-030
&mykeywords= pain,disease&logic=and&akalogic=yes
```

όπου "SPEC-01" και "SYMPT-030" είναι τα id των "Ophthalmologist" και "Dry Eyes" αντίστοιχα. Η διαδικασία αποστολής του αιτήματος μέσω κώδικα JavaScript περιγράφηκε πιο αναλυτικά στην παράγραφο 3.1.1.

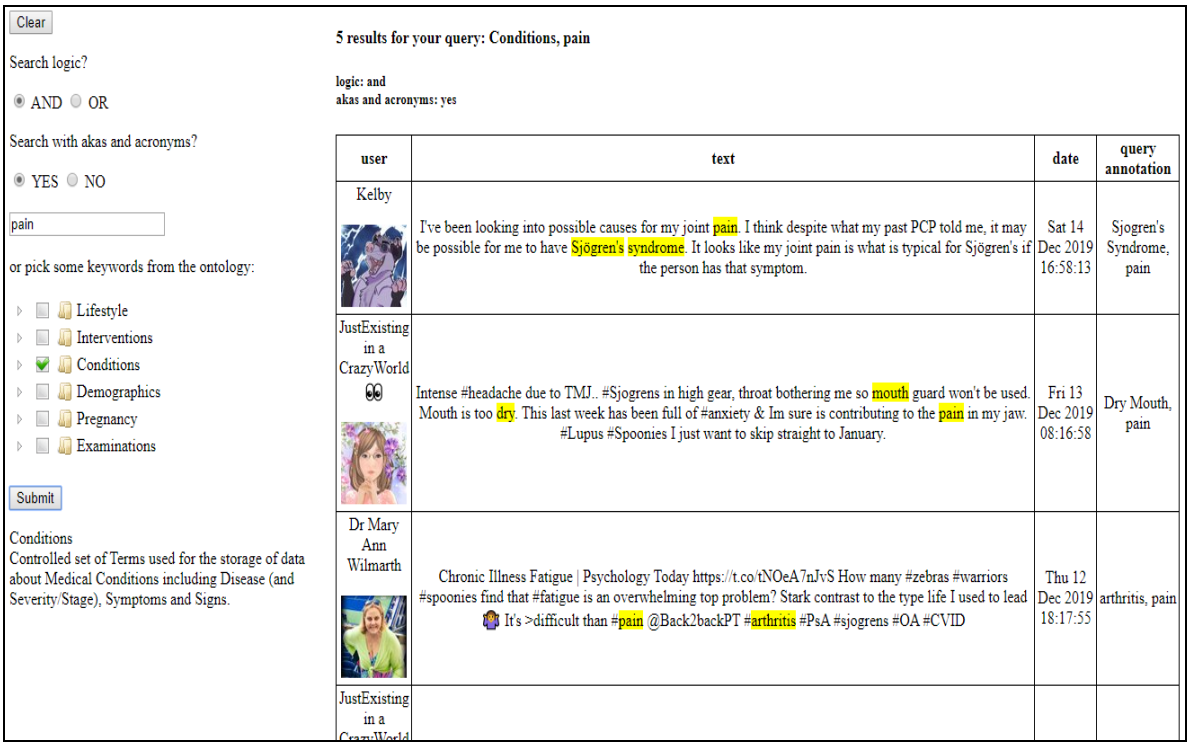
Τέλος, θα πρέπει να σημειώσουμε ότι με το πάτημα του Submit διενεργείται έλεγχος σχετικά με το αν έχουν επιλεγεί από τον κατάλογο ή έχουν εισαχθεί στο πεδίο κειμένου κάποιες λέξεις-κλειδιά. Αν δεν έχει συμβεί τίποτα από τα δύο, εμφανίζεται μέσω ενός alert box από τον Browser το μήνυμα "Please enter or select some keywords", ενώ η αναζήτηση δεν πραγματοποιείται, καθώς δεν αποστέλλεται το request στον server. Ο έλεγχος αυτός είναι απαραίτητος προκειμένου να εξασφαλιστεί ότι δεν θα πραγματοποιηθεί αναζήτηση χωρίς καμία λέξη-κλειδί.

4.3 Παρουσίαση αποτελεσμάτων της αναζήτησης

Τις πληροφορίες που περιλαμβάνει το response που στέλνεται από το service επεξεργασίας στο frontend, τις διαχειριζόμαστε μέσω κώδικα JavaScript. Αφού εξάγουμε από το JSON του response τα δεδομένα που αναφέρθηκαν στην προηγούμενη παράγραφο, τα παρουσιάζουμε στον χρήστη της εφαρμογής μας μέσω της διεπαφής. Όλες οι πληροφορίες που επιστρέφει η αναζήτηση τοποθετούνται στη δεξιά πλευρά της οθόνης. Πρώτα εμφανίζονται τα στοιχεία σχετικά με την

αναζήτησή του, δηλαδή το πλήθος των αποτελεσμάτων, οι λέξεις-κλειδιά που επέλεξε, η λογική της αναζήτησης και το αν αυτή περιλαμβάνει συνώνυμα και ακρωνύμια. Ακριβώς από κάτω εμφανίζεται ο πίνακας με τα αποτελέσματα της αναζήτησης, ο οποίος αποτελείται από τέσσερις στήλες. Η πιο αριστερή στήλη περιέχει το όνομα και τη φωτογραφία προφίλ του χρήστη, στην επόμενη στήλη εμφανίζεται το κείμενο της δημοσίευσης με υπογραμμισμένες τις λέξεις-κλειδιά που βρέθηκαν, καθώς και κάποια εικόνα που ίσως περιλαμβάνει η δημοσίευση, στην προτελευταία στήλη αναγράφεται η ημερομηνία της δημοσίευσης, ενώ στην τελευταία στήλη παρουσιάζονται τα keywords που περιέχει το κείμενο διαχωρισμένα με κόμμα.

Στην εικόνα 4.4 απεικονίζεται ένα στιγμιότυπο της οθόνης, αμέσως αφού πραγματοποιήσουμε αναζήτηση με τις λέξεις-κλειδιά “Conditions” και “pain”, με λογική and και συμπεριλαμβάνοντας και τα συνώνυμα και τα ακρωνύμια των λέξεων. Παρατηρούμε ότι μετά την επιστροφή και εμφάνιση των αποτελεσμάτων, διατηρούνται στα αριστερά της οθόνης οι τελευταίες επιλογές του χρήστη. Αυτό οφείλεται στο ότι δεν φορτώνει ξανά ολόκληρη η σελίδα για να παρουσιαστούν τα αποτελέσματα, αφού αυτά επιστρέφονται με χρήση της τεχνικής AJAX. Όπως αναφέρθηκε και στο κεφάλαιο 2, χρησιμοποιώντας την AJAX μπορούμε να ανανεώσουμε δυναμικά το περιεχόμενο ενός μόνο μέρους της σελίδας, διατηρώντας την υπόλοιπη σελίδα ως έχει. Έτσι, κάθε φορά που ολοκληρώνεται μια αναζήτηση, οι επιλογές του χρήστη παραμένουν ανέπαφες, καθώς ανανεώνεται μόνο το περιεχόμενο στο δεξί μέρος της σελίδας, και άρα δεν επηρεάζονται με κανέναν τρόπο από την εμφάνιση των αποτελεσμάτων.



Εικόνα 4.4: Αποτελέσματα της αναζήτησης

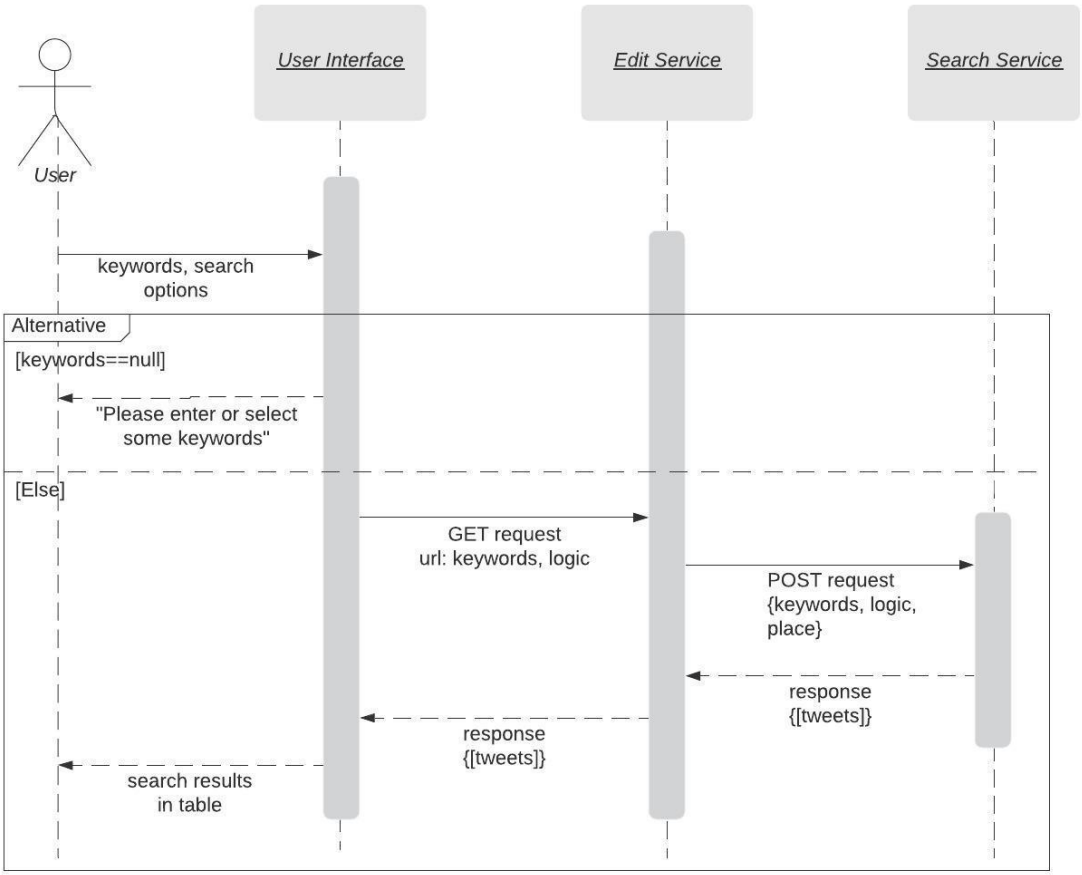
5

Λειτουργικότητα αναζήτησης και βελτίωση αποτελεσμάτων

5.1 Οι δύο φάσεις της διαδικασίας

Η διαδικασία της αναζήτησης δημοσιεύσεων θα μπορούσε να χωριστεί σε δύο φάσεις. Κατά την πρώτη φάση, πραγματοποιείται αναζήτηση στο Twitter, με τη χρήση των μεθόδων και των κλάσεων του Twitter API, με βάση τις προκαθορισμένες λέξεις-κλειδιά που στέλνουμε στο service κατά την αρχικοποίησή του, η οποία αναλύθηκε στην παράγραφο 3.2.1. Η δεύτερη φάση αποτελείται από το φιλτράρισμα των αποτελεσμάτων της πρώτης φάσης, με βάση τις λέξεις-κλειδιά που επιλέχθηκαν μέσω της διεπαφής χρήστη. Η λειτουργία κάθε φάσης πραγματοποιείται από διαφορετικό τμήμα λογισμικού του service αναζήτησης. Όπως ειπώθηκε στην παράγραφο 3.2.2.3, το service της αναζήτησης υπήρχε ως αυτοτελής υπηρεσία πριν ξεκινήσει η παρούσα εργασία, που μας δόθηκε αρχικά για απλή χρήση, ενώ στη συνέχεια υπέστη επεξεργασία προκειμένου να προστεθούν νέες λειτουργικότητες. Τόσο οι προϋπάρχουσες λειτουργίες, όσο και εκείνες που προστέθηκαν αργότερα αναλύονται παρακάτω.

Όλη η διαδικασία που ακολουθείται από τη στιγμή της εισαγωγής των δεδομένων αναζήτησης μέχρι και την επιστροφή των αποτελεσμάτων, παρουσιάζεται σχηματικά μέσω του διαγράμματος UML (Unified Modeling Language), που φαίνεται στην εικόνα 5.1. Τα διαγράμματα UML χρησιμοποιούνται για να περιγράψουν τις λειτουργίες που εκτελούνται από μια εφαρμογή και την αλληλεπίδρασή της με τους χρήστες, και διακρίνονται σε κατηγορίες ανάλογα με το αν απεικονίζουν μια περίπτωση χρήσης της εφαρμογής, την ακολουθία των λειτουργιών που εκτελούνται, τις κλάσεις που αποτελούν την εφαρμογή κλπ. Το διάγραμμα της εικόνας παρουσιάζει την ακολουθία ενεργειών που πραγματοποιούνται κατά την εκτέλεση της εφαρμογής και ονομάζεται ακολουθιακό διάγραμμα UML (sequence UML diagram).



Εικόνα 5.1: Οι λειτουργίες της εφαρμογής σε διάγραμμα UML

5.2 Πρώτη φάση: Συλλογή δημοσιεύσεων από το Twitter

Όπως ειπώθηκε συνοπτικά στο κεφάλαιο 3, η συλλογή δημοσιεύσεων από το Twitter δεν πραγματοποιείται με βάση τις λέξεις-κλειδιά που επιλέγει ο χρήστης, αλλά με βάση κάποια άλλα keywords, τα οποία τα έχουμε προσδιορίσει στο body του post request που στέλνουμε για να ξεκινήσει η συλλογή. Γίνεται αντιληπτό, λοιπόν, ότι πρώτα συλλέγονται τα πιο πρόσφατα tweets που περιέχουν τις προκαθορισμένες λέξεις-κλειδιά, και στη συνέχεια σταχυολογούνται με βάση τις λέξεις-κλειδιά που δόθηκαν από τον χρήστη. Έτσι, μας δίνεται η δυνατότητα να συγκεντρώσουμε πληροφορίες για το πώς μια έννοια που επιλέξαμε μέσω του interface μπορεί να σχετίζεται με τις προκαθορισμένες λέξεις ενδιαφέροντος που βρίσκονται στο post request. Στην περίπτωση της οντολογίας ιατρικών όρων με την οποία εργαζόμαστε, έχουμε ως προκαθορισμένη έννοια στο post request την αυτοάνοση ασθένεια Sjogren's Syndrome, γραμμένη σε έξι διαφορετικές μορφές, επομένως αναζητάμε πρώτα τις δημοσιεύσεις που περιέχουν αυτή τη λέξη γραμμένη με έναν από αυτούς τους τρόπους. Στη συνέχεια, οι δημοσιεύσεις αυτές ελέγχονται μία προς μία, προκειμένου να διαπιστωθεί αν περιέχουν, ανάλογα με τη λογική της αναζήτησης, μια τουλάχιστον ή περισσότερες από τις έννοιες που έδωσε ο χρήστης ή υποέννοιές τους. Έτσι, από τις φιλτραρισμένες δημοσιεύσεις μπορεί να εξαχθεί πολύτιμη γνώση σχετικά με τα συμπτώματα της ασθένειας, τους τρόπους θεραπείας ή άλλες ασθένειες που σχετίζονται με το σύνδρομο Sjogren.

Αναφέρθηκε, επίσης, ότι η αναζήτηση των tweets γίνεται ασύγχρονα, δηλαδή δεν ξεκινά μόλις ο χρήστης πατήσει Submit, αλλά πραγματοποιείται στο παρασκήνιο αφήνοντας τις άλλες λειτουργίες της εφαρμογής ανεπηρέαστες. Επιπλέον, ανά τακτά χρονικά διαστήματα ξεκινά αυτόματα μια αναζήτηση, με σκοπό να εντοπιστούν νεότερα tweets και να προστεθούν σε αυτά που ήδη έχουν βρεθεί από τις προηγούμενες αναζητήσεις. Η διαδικασία αυτή υλοποιείται με τη βοήθεια

των νημάτων (Threads) και της διεπαφής Runnable που παρέχει η Java. Οι διεπαφές (interfaces) της Java αποτελούν συλλογές μεθόδων που έχουν κάποια σταθερά χαρακτηριστικά, ενώ οι υπόλοιπες λειτουργίες τους υλοποιούνται στην κλάση που εφαρμόζει την διεπαφή στην οποία ανήκουν. Τα στιγμιότυπα των κλάσεων εκείνων της Java που εφαρμόζουν την συγκεκριμένη διεπαφή ονομάζονται νήματα, ή αλλιώς Threads, και έχουν τη δυνατότητα να εκτελούνται στο παρασκήνιο, ανεξάρτητα από τα υπόλοιπα τμήματα του προγράμματος, χάρη στη μέθοδο run() που παρέχεται από την Runnable και υλοποιείται από την κλάση που την εφαρμόζει. Δημιουργείται αρχικά ένα Thread, που αποτελεί αντικείμενο της κλάσης ConfigNRetrieve, η οποία εφαρμόζει τη διεπαφή Runnable. Μέσα στην κλάση υλοποιείται, όπως είπαμε, η μέθοδος run(), η οποία εκτελεί παρασκηνιακά την αναζήτηση στο Twitter.

Αρχικά, διαβάζονται και αποθηκεύονται οι τιμές των παραμέτρων του JSON body του request που στάλθηκε στο service κατά την αρχικοποίησή του, κάποιες από τις οποίες μπορούμε να δούμε αν ανατρέξουμε στην εικόνα 3.2. Ανάμεσα σε αυτές περιέχονται και οι λέξεις κλειδιά. Στη συνέχεια, η αναζήτηση στο Twitter διακρίνεται με τη σειρά της σε δύο στάδια, την αναζήτηση παλαιών και νέων δημοσιεύσεων. Οι δύο αυτές λειτουργίες εκτελούνται μέσα σε δυο ξεχωριστούς βρόχους επανάληψης. Όταν αναζητούνται οι παλιότερες δημοσιεύσεις, τίθεται αρχικά ένα όριο στο πλήθος των αποτελεσμάτων που θα επιστρέφονται σε κάθε αναζήτηση, το οποίο βρίσκεται στις 100 δημοσιεύσεις, όσο και το μέγιστο όριο που επιτρέπει το Twitter. Αυτό σημαίνει ότι στην αναζήτηση που εκτελείται σε κάθε επανάληψη θα επιστρέφονται από το Twitter 100 ή και λιγότερες δημοσιεύσεις. Για την εκτέλεση της αναζήτησης κάθε φορά, δημιουργείται ένα νέο αντικείμενο της κλάσης Query, στο οποίο αποθηκεύεται η λέξη-κλειδί της αναζήτησης, και μέσω αυτού εκτελείται η επιθυμητή λειτουργία με τη χρήση της μεθόδου search(). Τα tweets επιστρέφονται ως αντικείμενα της κλάσης Status, και αποθηκεύονται σε μια λίστα με αντικείμενα αυτού του τύπου. Τόσο οι κλάσεις Query και Status, όσο και η μέθοδος search() παρέχονται υλοποιημένες από το Twitter API. Μετά από κάθε αναζήτηση, το πρόγραμμα σταματά να εκτελείται για 6 δευτερόλεπτα, ή αλλιώς 6000 ms, χρόνος που επίσης δίνεται από το σώμα του αιτήματος αρχικοποίησης (“TimeDelay”: 6000). Καθώς το Twitter θέτει αυστηρούς περιορισμούς σχετικά με την μαζική εξαγωγή δεδομένων (rate limits), η παρέλευση του παραπάνω χρονικού διαστήματος είναι απαραίτητη ώστε η εφαρμογή μας να μην αποκλειστεί αυτόματα από τη δυνατότητα περαιτέρω αναζητήσεων. Μόλις η αναζήτηση σταματήσει να επιστρέφει αποτελέσματα, οι επαναλήψεις σταματούν και μπαίνουμε στον δεύτερο βρόχο, εκείνο της αναζήτησης νέων δημοσιεύσεων.

5.2.1 Αποφυγή εξαιρέσεων

Η λειτουργία της αναζήτησης δημοσιεύσεων στο Twitter που περιγράφηκε προηγουμένως, χρησιμοποιήθηκε σχεδόν αυτούσια όπως μας δόθηκε στην αρχή της εργασίας, καθώς κρίθηκε ότι δεν χρειάζεται σε αυτή τη φάση της αναζήτησης να προστεθούν επιπλέον λειτουργικότητες. Η μόνη επεξεργασία που υπέστη αυτό το τμήμα του service αφορά στην αποφυγή μιας εξαίρεσης (exception) της Java που εμφανιζόταν όταν τα δύο τμήματα του service λειτουργούσαν παράλληλα.

Ας θεωρήσουμε το εξής σενάριο: ο χρήστης της εφαρμογής πατάει submit, οπότε πραγματοποιείται ανάκτηση των δημοσιεύσεων που έχουν βρεθεί κατά την αναζήτηση στο Twitter και φιλτράρισμά τους με βάση τις επιλεγμένες λέξεις-κλειδιά. Υπενθυμίζουμε ότι στο παρασκήνιο συνεχίζει να πραγματοποιείται αναζήτηση στο Twitter, τα αποτελέσματα της οποίας προστίθενται στην λίστα με τα υπόλοιπα tweets, όπου όμως έχει αποκτήσει πλέον πρόσβαση και το τμήμα του service που είναι υπεύθυνο για την ανάκτησή τους κατόπιν αιτήματος του χρήστη. Στην περίπτωση που επιχειρηθεί ανάγνωση της λίστας ταυτόχρονα με την εγγραφή των νέων αποτελεσμάτων της αναζήτησης σε αυτή, δημιουργείται μια εξαίρεση της Java του τύπου “Concurrent Modification Exception”. Οι εξαιρέσεις αποτελούν και αυτές κλάσεις της Java, τα αντικείμενα των οποίων δημιουργούνται όταν παρουσιάζεται κάποιο σφάλμα κατά την εκτέλεση του προγράμματος και έχουν ως αποτέλεσμα τον απευθείας τερματισμό του χωρίς την επιστροφή των προσδοκώμενων αποτελεσμάτων.

Η εμφάνιση εξαίρεσης αυτού του τύπου οδηγούσε στην επιστροφή μιας κενής λίστας, με αποτέλεσμα ο χρήστης να βλέπει ότι δεν βρέθηκε κανένα αποτέλεσμα για την αναζήτησή του. Για την αντιμετώπιση αυτής της εξαίρεσης, το πρόγραμμα τροποποιήθηκε έτσι ώστε οι δημοσιεύσεις που ανακτώνται από το Twitter να αποθηκεύονται σε μια λίστα η οποία αποτελεί αντικείμενο της κλάσης "CopyOnWriteArrayList". Οι λίστες αυτής της κλάσης έχουν την ιδιότητα να δημιουργούν αντίγραφα του εαυτού τους κάθε φορά που το πρόγραμμα επιχειρεί να τις τροποποιήσει, να τους προσθέσει δηλαδή νέες εγγραφές. Έτσι, αν τα δυο τμήματα του service αποκτήσουν ταυτόχρονα πρόσβαση στη λίστα, η ανάκτηση των tweets που έχουν βρεθεί μέχρι στιγμής θα πραγματοποιηθεί από την αρχική λίστα, ενώ η εγγραφή των νέων tweets θα δημιουργήσει ένα ακριβές αντίγραφό της, το οποίο στη συνέχεια θα τροποποιήσει και θα αποθηκεύσει στη θέση της αρχικής λίστας. Γενικά, οι λίστες της Java είναι αντικείμενα της κλάσης ArrayList, παραλλαγή της οποίας αποτελεί η κλάση CopyOnWriteArrayList, που χρησιμοποιείται προκειμένου να εξαλειφθούν τα exceptions σε περιπτώσεις πολυνηματικών (multi-threading) εφαρμογών όπως η δική μας.

5.3 Δεύτερη φάση: Αναζήτηση δημοσιεύσεων με λέξεις-κλειδιά

Η φάση αυτή της αναζήτησης αποτελεί και τον πυρήνα της εφαρμογής. Πρόκειται για τη λειτουργία που πραγματοποιείται από τη στιγμή που ο χρήστης πατάει Submit μέχρι την επιστροφή των αποτελεσμάτων στο frontend. Η λειτουργία αυτή περιλαμβάνει την αποστολή των απαραίτητων GET και POST requests προς τα δύο services, καθώς και τα αντίστοιχα responses, την αναζήτηση στην οντολογία των keywords που επέλεξε ο χρήστης, καθώς και την ανάκτηση των αποθηκευμένων δημοσιεύσεων που περιέχουν τα keywords αυτά.

5.3.1 Προετοιμασία της αναζήτησης

Με την αποστολή του αιτήματος GET από τον browser καλείται αρχικά το service επεξεργασίας των δεδομένων. Σκοπός του service σε αυτό το σημείο της διαδικασίας είναι να προετοιμάσει το JSON body του POST request που θα σταλεί στην συνέχεια στο service της αναζήτησης. Το JSON αυτό θα περιέχει, μεταξύ άλλων, τις λέξεις-κλειδιά που επέλεξε ο χρήστης μέσω του user interface.

5.3.1.1 Εντοπισμός των επιλεγμένων keywords

Αρχικά, διαβάζονται οι τιμές των παραμέτρων που στάλθηκαν μέσω του url. Όπως αναφέραμε στο κεφάλαιο 4, μεταφέρονται στο service μέσω της παραμέτρου items τα id των keywords που έχει επιλέξει ο χρήστης, διαχωρισμένα με κόμμα. Πρέπει πρώτα να γίνουν γνωστά τα ίδια τα keywords για να σταλούν έπειτα στο service της αναζήτησης. Αυτό επιτυγχάνεται μέσω της αναζήτησης των id τους στην οντολογία. Πρώτα διαβάζεται η οντολογία με την ίδια ακριβώς διαδικασία που περιγράψαμε στην παράγραφο 4.1.2, και στη συνέχεια για κάθε id της παραμέτρου items, ελέγχονται τα id όλων των κλάσεων της οντολογίας. Μόλις βρεθεί μία αντιστοίχιση, η τιμή του πεδίου label της κλάσης, δηλαδή η ίδια η λέξη-κλειδί, αποθηκεύεται σε έναν πίνακα από Strings.

5.3.1.2 Βελτιστοποίηση της διαδικασίας

Προκειμένου να καταστεί η αναζήτηση των δημοσιεύσεων από το Twitter πιο αποτελεσματική, κρίθηκε σκόπιμη η αναζήτηση όχι μόνο των tweets που περιέχουν τα επιλεγμένα keywords, αλλά και εκείνων που περιέχουν κάποια από τις υποέννοιές τους. Έτσι, τα αποτελέσματα της αναζήτησης εμπλουτίζονται, καθώς πλέον περιλαμβάνουν περισσότερες δημοσιεύσεις, πολλές από τις οποίες αφορούν μεν πιο εξειδικευμένα θέματα, εντάσσονται δε στην ευρύτερη κατηγορία των θεμάτων που ενδιαφέρουν τον χρήστη.

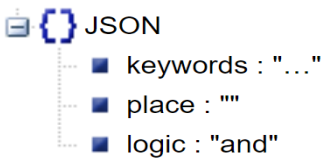
Επομένως, η αναζήτηση στην οντολογία διαφοροποιείται ως εξής: μόλις βρεθεί μία κλάση με το id ενός keyword που έχει επιλέξει ο χρήστης, κρατάμε το label της κλάσης και συνεχίζουμε την αναζήτηση στις υποκλάσεις της, από τις οποίες κρατάμε επίσης το label και συνεχίζουμε το ίδιο με τις δικές τους υποκλάσεις κ.ο.κ.. Παρατηρούμε ότι πρόκειται και πάλι για μία αναδρομική διαδικασία, σαν αυτή που χρησιμοποιήσαμε κατά την ανάγνωση της οντολογίας για να φορτώσουμε τις πληροφορίες στη μνήμη του προγράμματος και ταυτόχρονα να διατηρήσουμε την ιεραρχική δομή της. Η μόνη διαφορά είναι ότι εδώ απλώς διατρέχουμε την οντολογία αντί να την κατασκευάσουμε.

Όπως ειπώθηκε στην παράγραφο 5.3.1.1, κάθε λέξη-κλειδί αποθηκεύεται σε έναν πίνακα από Strings. Με την παραπάνω τροποποίηση στην αναζήτηση των keywords, στον πίνακα αυτόν θα πρέπει να εισαχθούν τώρα και όλες οι υποέννοιες κάθε επιλεγμένης λέξης-κλειδί. Έτσι, κάθε στοιχείο του πίνακα θα αποτελείται πλέον από ένα σύνολο keywords, διαχωρισμένων με τον χαρακτήρα '|', όπου το πρώτο keyword θα είναι αυτό που επέλεξε ο χρήστης και θα ακολουθούν οι υποέννοιές του. Για παράδειγμα, ας θεωρήσουμε ότι ένας χρήστης επιλέγει μόνο την έννοια Lifestyle. Αν δούμε τον κατάλογο των keywords στο interface, θα διαπιστώσουμε ότι το Lifestyle έχει ως υποέννοια το SMOKING STATUS, το οποίο με τη σειρά του έχει υποέννοιες τα Ex-Smoker, Never Smoker και Active Smoker. Επομένως, ο πίνακας θα έχει μόνο ένα στοιχείο, άρα θα είναι της μορφής: ["Lifestyle|SMOKING STATUS|Ex-Smoker|Never Smoker|Active Smoker"]. Αν πάλι θεωρήσουμε ότι επιλέγονται τα keywords SMOKING STATUS και Conditions, τότε ο πίνακας θα είναι:
["SMOKING STATUS|Ex-Smoker|Never Smoker|Active Smoker", "Conditions|LYMPHOMA STAGE|…"].

Ο λόγος που επιλέχθηκε ο χαρακτήρας '|' αντί κάποιου άλλου χαρακτήρα διαχωρισμού, είναι επειδή χαρακτήρες όπως το κόμμα (,), η παύλα (-) και η κάθετος (/) βρίσκονται ήδη σε ορισμένες λέξεις-κλειδιά. Άρα, αν για παράδειγμα επιλέγαμε ως χαρακτήρα διαχωρισμού την παύλα, η έννοια Ex-Smoker θα εξαγόταν από τον πίνακα στο service αναζήτησης ως δυο ξεχωριστές έννοιες, "Ex" και "Smoker", για καθεμία από τις οποίες η υπηρεσία θα πραγματοποιούσε ξεχωριστή αναζήτηση και προφανώς θα επέστρεφε αποτελέσματα που δεν μας ενδιαφέρουν.

5.3.1.3 Εκκίνηση της αναζήτησης

Η αναζήτηση ξεκινά με την αποστολή ενός POST αιτήματος από το service επεξεργασίας στο service αναζήτησης. Το αίτημα αυτό έχει στο JSON body του την παράμετρο "keywords", η οποία περιέχει τον πίνακα με τις επιλεγμένες λέξεις-κλειδιά και τις υποέννοιές τους. Επίσης, περιέχει την παράμετρο "logic", της οποίας η τιμή μπορεί να είναι "and" ή "or", ανάλογα με την επιλογή του χρήστη από το user interface.



Εικόνα 5.2: JSON Body του POST HTTP Request προς το Search Service

Στην προηγούμενη εικόνα βλέπουμε ότι υπάρχει μια επιπλέον παράμετρος place. Η παράμετρος αυτή χρησιμοποιείται για να «φιλτράρουμε» τα αποτελέσματα της αναζήτησης και να βρούμε μόνο τα tweets που δημοσιεύτηκαν από μια συγκεκριμένη χώρα. Στην παρούσα εργασία δεν δίνουμε στον χρήστη τη δυνατότητα για τέτοιου είδους φιλτράρισμα, επομένως η συγκεκριμένη παράμετρος αφήνεται κενή. Θα μπορούσαμε, βέβαια, να ορίσουμε μέσω κώδικα μια προκαθορισμένη τιμή για την παράμετρο place, ενώ σε κάποια ενδεχόμενη βελτιστοποίηση της εφαρμογής, θα μπορούσε να υπάρχει δυνατότητα επιλογής μιας χώρας από τον χρήστη, οπότε θα διαβάζαμε και την τιμή της place από το url, όπως και των υπόλοιπων παραμέτρων.

5.3.2 Αναζήτηση δημοσιεύσεων

Τώρα θα εξετάσουμε τον τρόπο με τον οποίο το service αναζητά ανάμεσα στις δημοσιεύσεις που έχει συλλέξει από το Twitter, εκείνες που περιέχουν τις επιλεγμένες λέξεις-κλειδιά. Θα πρέπει να σημειώσουμε ότι αν και το συγκεκριμένο τμήμα του service προϋπήρχε της εφαρμογής μας, στην πορεία υπέστη σημαντικές τροποποιήσεις προκειμένου η αναζήτηση των keywords να καταστεί πιο ευέλικτη και αποτελεσματική. Παρακάτω θα δούμε τη βασική λειτουργικότητα που είχε η προϋπάρχουσα υπηρεσία καθώς και ποια προβλήματα ανέκυπταν, ενώ στη συνέχεια θα περιγραφούν όλες οι προσθήκες που πραγματοποιήθηκαν στο πλαίσιο της διπλωματικής εργασίας.

5.3.2.1 Βασική λειτουργικότητα του service

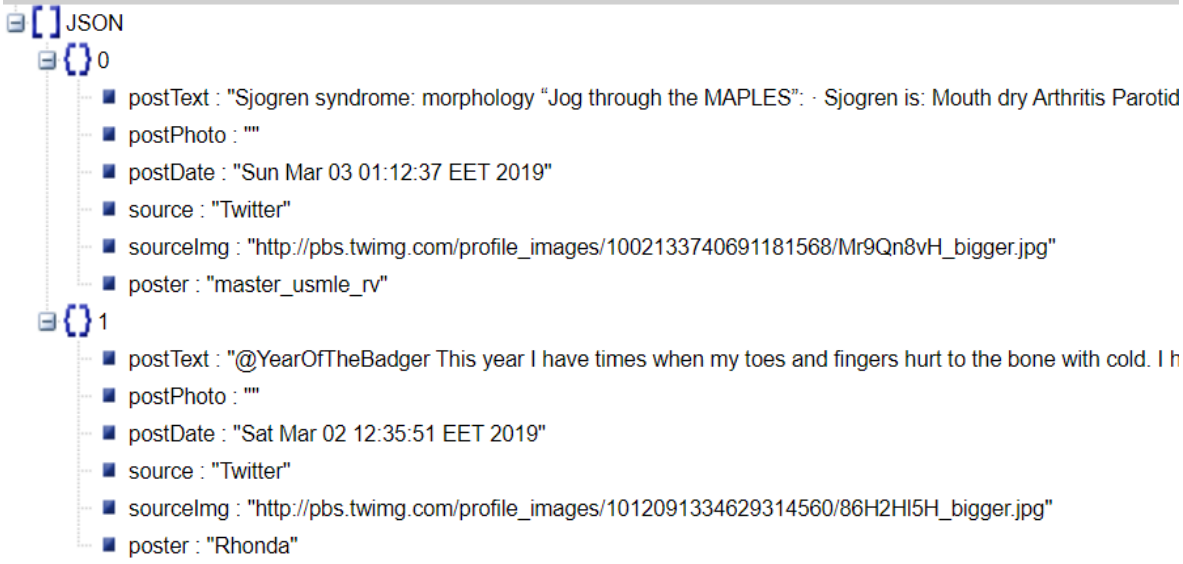
Το αρχικό service για να ξεκινήσει την αναζήτηση δέχεται ένα POST request, στο σώμα του οποίου υπάρχει ένα JSON αντικείμενο, παρόμοιας μορφής με αυτήν της εικόνας 5.2, χωρίς όμως την παράμετρο logic. Δεν υπάρχει επομένως η δυνατότητα να ορίσουμε με κάποιον τρόπο αν η αναζήτηση θα πραγματοποιηθεί με λογική “and” ή “or”, καθώς είναι προκαθορισμένο οι δημοσιεύσεις να αναζητούνται με λογική and. Αυτό σημαίνει πως για να συμπεριληφθεί ένα tweet στα επιστρεφόμενα αποτελέσματα πρέπει να περιέχει όλες τις λέξεις-κλειδιά που του στείλαμε μέσω του request.

Τα keywords διαβάζονται από την παράμετρο keywords του JSON Body. Πρόκειται ουσιαστικά για μια ενιαία συμβολοσειρά (String), στην οποία περιέχονται οι λέξεις-κλειδιά, διαχωρισμένες μεταξύ τους με κόμμα. Το πρόγραμμα ξεχωρίζει το κάθε keyword, κάνοντας χρήση της μεθόδου split(‘,’) που παρέχει η Java, μέσω της οποίας ένα String μπορεί να χωριστεί με βάση κάποιον χαρακτήρα διαχωρισμού που δίνεται ως όρισμα, όπως στην περίπτωσή μας το κόμμα, σε επιμέρους μικρότερα Strings, τα οποία επιστρέφονται αποθηκευμένα σε έναν πίνακα (Array). Έτσι, τα keywords αποθηκεύονται στον πίνακα keywordsList.

Έπειτα, το πρόγραμμα εισέρχεται σε έναν βρόχο επανάληψης. Σε κάθε επανάληψη εξετάζεται κι ένα από τα αποθηκευμένα tweets, ώστε να διαπιστωθεί αν περιέχει όλες τις λέξεις-κλειδιά. Ο έλεγχος για κάθε keyword πραγματοποιείται μέσα σε έναν εμφωλευμένο βρόχο, ο οποίος διατρέχει τον πίνακα keywordsList. Για να ελέγξουμε αν το κείμενο μιας δημοσίευσης περιέχει το keyword, χρησιμοποιούμε τη μέθοδο contains(), η οποία δέχεται ως όρισμα το keyword κι επιστρέφει τιμή true αν αυτό περιέχεται στο κείμενο της δημοσίευσης που ελέγχουμε, αλλιώς επιστρέφει false. Για να επιστραφεί τιμή true, πρέπει η λέξη να περιέχεται αυτούσια στο κείμενο, να είναι δηλαδή ίδιοι όλοι οι χαρακτήρες ένας προς έναν, καθώς γίνεται ακόμα και διάκριση μεταξύ πεζών και κεφαλαίων γραμμάτων. Για την επιστροφή περισσότερων σωστών αποτελεσμάτων, πριν την κλήση της μεθόδου contains() μετατρέπονται όλοι οι χαρακτήρες τόσο του κειμένου του εξεταζόμενου tweet όσο και του keyword σε πεζούς, με κλήση της κατάλληλης μεθόδου toLowerCase().

Παράλληλα, διατηρούμε μια λογική μεταβλητή με όνομα TweetContainAllKeywords, την οποία αρχικοποιούμε στην τιμή true κάθε φορά που ελέγχουμε ένα καινούριο tweet. Αν έστω και

μία λέξη-κλειδί δεν βρεθεί στο κείμενο, καταγράφουμε την πληροφορία αυτή δίνοντάς στην παραπάνω μεταβλητή την τιμή false. Αν η μεταβλητή αυτή έχει τιμή true αφού μετά τον έλεγχο όλων των keywords, δηλαδή αν στο κείμενο περιέχονται αυτούσιες όλες οι λέξεις-κλειδιά, δημιουργείται ένα αντικείμενο JSON, στο οποίο προστίθενται οι βασικές πληροφορίες του συγκεκριμένου tweet, όπως το όνομα και η φωτογραφία προφίλ του συντάκτη-χρήστη του Twitter, το κείμενο της δημοσίευσης, η ημερομηνία δημοσίευσης καθώς και κάποια εικόνα που τυχόν έχει το tweet αυτό. Το JSON με τις πληροφορίες αυτές προστίθεται σε μια μεγαλύτερη λίστα από JSON αυτής της μορφής, η οποία περιέχει τα αποτελέσματα που έχουν βρεθεί μέχρι τώρα. Όταν ελεγχθούν όλα τα tweets, η διαδικασία ολοκληρώνεται και το service στέλνει ως response αυτή τη λίστα με όλα τα αποτελέσματα, ένα παράδειγμα της οποίας φαίνεται στην παρακάτω εικόνα.



Εικόνα 5.3: Λίστα με ενδεικτικά αποτελέσματα που επιστρέφει το service

Από τα προηγούμενα διακρίνουμε αμέσως ένα σημαντικό πρόβλημα που είχε το αρχικό service. Αυτό αφορά στο ακριβές ταίριασμα των επιλεγμένων λέξεων-κλειδιών με τις λέξεις στο κείμενο ενός tweet, χωρίς το οποίο θα χάσουμε αρκετά tweets από το σύνολο όλων των επιθυμητών αποτελεσμάτων. Ακόμα και στην πιο απλή περίπτωση που η λέξη-κλειδί περιέχεται αυτούσια μέσα στο κείμενο, αν την πληκτρολογήσουμε αφήνοντας κατά λάθος ένα κενό πριν τη λέξη, το πρόγραμμα θα κρατήσει αυστηρά μόνο τα tweets εκείνα που την περιέχουν με ένα κενό πριν τον πρώτο της χαρακτήρα. Έτσι, αν για παράδειγμα το keyword που στέλνουμε στο service μέσω του POST request είναι “ pain”, με κενό δηλαδή στην αρχή, θα επιστραφούν μεν τα tweets που περιέχουν τα “ pain” ή “ painful”, αλλά θα απορριφθούν τα tweets που περιέχουν τα “chronicpain” ή “#pain”.

Ένα δεύτερο, λιγότερο σημαντικό και πιο εύκολα επιλύσιμο πρόβλημα είναι ότι ακόμα κι αν το κείμενο της δημοσίευσης που ελέγχεται δεν περιέχει τη ζητούμενη λέξη, οι έλεγχοι συνεχίζονται και για τα υπόλοιπα keywords χωρίς να υπάρχει λόγος, αφού έτσι κι αλλιώς η δημοσίευση δεν θα συμπεριληφθεί στα αποτελέσματα. Αυτό προσθέτει επιπλέον υπολογιστικό κόστος στο πρόγραμμα, αφού στην περίπτωση που πρέπει να ελεγχθούν m tweets και έχουμε στείλει n λέξεις-κλειδιά, πάντα θα πραγματοποιούνται m x n έλεγχοι, ακόμα και αν το πρώτο keyword της λίστας δεν περιέχεται σε κανένα tweet, οπότε θα αρκούσαν ακριβώς m έλεγχοι. Μια γρήγορη λύση που δόθηκε σε αυτό το πρόβλημα, όπως θα δούμε και στη συνέχεια, είναι ο έλεγχος του προγράμματος να βγαίνει αμέσως από τον εσωτερικό βρόχο αν κάποια λέξη-κλειδί δεν βρεθεί, και να συνεχίζει ελέγχοντας το επόμενο tweet.

5.3.2.2 Τροποποιήσεις και βελτιώσεις

Η πρώτη σημαντική αλλαγή που έγινε στο service είναι η δυνατότητα να καθορίζει ο χρήστης της εφαρμογής αν η αναζήτηση θα πραγματοποιηθεί με λογική and ή or. Για το σκοπό αυτό, στο JSON που στέλνουμε κάθε φορά για την εκκίνηση της διαδικασίας συμπεριλαμβάνεται και η παράμετρος logic, με δυνατές τιμές and ή or, οι οποίες καθορίζουν αν σε κάθε tweet πρέπει να βρεθούν όλες οι επιλεγμένες λέξεις-κλειδιά ή αρκεί να βρεθεί τουλάχιστον μία αντίστοιχα. Υπενθυμίζουμε ότι αν το service δεν χρησιμοποιείται αυτοτελώς, αλλά στα πλαίσια της εφαρμογής μας, η τιμή της παραμέτρου logic ορίζεται από τον χρήστη μέσω του interface, με προκαθορισμένη τιμή and. Καθώς η υλοποίηση της αναζήτησης διαφέρει στις δύο περιπτώσεις, ο κώδικας της κάθε μίας περιλαμβάνεται σε μπλοκ if-else.

Μια άλλη διαφοροποίηση σε σχέση με το αρχικό service είναι ότι στην περίπτωση επιλογής των λέξεων από τον κατάλογο στο interface, στέλνονται στο service μέσω της παραμέτρου keywords τόσο οι επιλεγμένες έννοιες όσο και οι υποέννοιές τους, σε έναν πίνακα με τη μορφή που βλέπουμε στην παράγραφο 5.3.1.2. Τυχόν keywords που δίνονται μέσω του πεδίου κειμένου προστίθενται ως νέα στοιχεία στο τέλος του πίνακα. Το πρόγραμμα, για να αποθηκεύσει τα keywords, δημιουργεί αρχικά μια ενιαία λίστα με το όνομα keywordsList. Έπειτα, διαβάζει ξεχωριστά κάθε στοιχείο του πίνακα, καλώντας κάθε φορά τη μέθοδο split(), με χαρακτήρα διαχωρισμού αυτή τη φορά τον '|', προκειμένου να ξεχωρίσει την επιλεγμένη έννοια και τις υποέννοιές της και να τις αποθηκεύσει όλες σε έναν πίνακα. Τέλος, προσθέτει τον πίνακα αυτό στην αρχική λίστα. Έτσι, η αναζήτηση πλέον γίνεται σαφώς πιο περίπλοκη, αφού για κάθε tweet καλούμαστε να διατρέξουμε κάθε πίνακα της λίστας, αντί για έναν μόνο πίνακα όπως στην αρχική έκδοση της υπηρεσίας.

Για το σκοπό αυτό, μέσα στον εξωτερικό βρόχο, σε κάθε επανάληψη του οποίου πραγματοποιείται έλεγχος ενός νέου tweet, έχουμε πλέον δύο εσωτερικούς βρόχους. Ο πρώτος προσπελαύνει τα στοιχεία της εξωτερικής λίστας, δηλαδή τους πίνακες με τα keywords, ενώ ο δεύτερος, ο οποίος είναι εμφωλευμένος στον προηγούμενο, ελέγχει αν τα στοιχεία ενός πίνακα περιέχονται στο υπό εξέταση tweet. Θα πρέπει να σημειωθεί ότι στην περίπτωση που έχουμε λογική and δεν χρειάζεται να βρεθούν όλα τα keywords από όλους τους πίνακες της λίστας για να προστεθεί μια δημοσίευση στα αποτελέσματα. Κάτι τέτοιο άλλωστε θα ήταν απίθανο, καθώς κάθε επιλεγμένο keyword μπορεί να αναφέρεται σε ένα ευρύ φάσμα θεμάτων, και άρα να έχει πάρα πολλές υποέννοιες συχνά άσχετες μεταξύ τους, που δεν θα ήταν λογικό να βρίσκονται όλες στο ίδιο tweet. Αρκεί, λοιπόν, να βρεθεί στο κείμενο της δημοσίευσης μία τουλάχιστον λέξη από κάθε πίνακα της λίστας. Σε περίπτωση που έχουμε λογική or, εξακολουθούμε να αναζητούμε μία τουλάχιστον λέξη-κλειδί από όλες αυτές που περιέχονται γενικά στη λίστα.

Εκτός από τις παραπάνω τροποποιήσεις, καθοριστικές βελτιώσεις έχουν γίνει και στον τρόπο που αναζητείται ένα keyword στο κείμενο. Οι λειτουργίες που θα περιγραφούν παρακάτω πραγματοποιούνται σε κάθε επανάληψη του πιο εσωτερικού βρόχου, στον οποίο υπενθυμίζουμε ότι γίνεται προσπέλαση κάθε στοιχείου ενός πίνακα της λίστας των keywords. Υπογραμμίζουμε ότι τόσο στην περίπτωση που έχουμε λογική and όσο και σε αυτήν που έχουμε λογική or, η ακολουθία ενεργειών κατά την αναζήτηση είναι ακριβώς η ίδια, με τη μόνη διαφορά να έγκειται στη συνθήκη τερματισμού της αναζήτησης, δηλαδή στα keywords που είναι απαραίτητο να βρεθούν σε κάθε περίπτωση. Μόλις βρούμε τη λέξη-κλειδί που αναζητείται, αν έχουμε λογική or τερματίζουμε την αναζήτηση, ενώ αν πρόκειται για λογική and συνεχίζουμε διαβάζοντας τον επόμενο πίνακα των keywords από την λίστα keywordsList. Αυτός ο τρόπος τερματισμού ακολουθείται σε όλα τα στάδια της αναζήτησης.

5.3.2.2.1 Αρχική φάση αναζήτησης

Κατά την αναζήτηση μίας λέξης στο κείμενο, ελέγχεται αρχικά αν η συγκεκριμένη λέξη-κλειδί ανήκει σε μια ευρύτερη ομάδα λέξεων που αποκαλούνται "stop words", και οι οποίες πρέπει να εξαιρεθούν από την αναζήτηση. Στην ομάδα αυτή των λέξεων περιλαμβάνονται ενδεικτικά οι προσωπικές και κτητικές αντωνυμίες, όπως you, we, me, himself, her κλπ, δεικτικές και

ερωτηματικές αντωνυμίες, όπως για παράδειγμα οι this, that, what, which κλπ, επιρρήματα όπως up, down, now, below, between κλπ, καθώς και τα ρήματα be, do, have και οι διαφορετικές μορφές τους, π.χ. is, are, done, has. Είναι φανερό ότι η εύρεση κάποιας από αυτές τις λέξεις στο κείμενο δεν παίζει ρόλο για την αποδοχή ή την απόρριψη της υπό εξέταση δημοσίευσης, καθώς αποτελούν λέξεις γενικής χρήσης που δεν περιέχουν κάποια σημαντική πληροφορία. Έτσι, η αναζήτησή τους είναι περιττή και δεν πραγματοποιείται, ώστε να εξοικονομηθεί πολύτιμος χρόνος και ταυτόχρονα να μην απορριφθούν tweets που τα χρειαζόμαστε, ή να μην συμπεριληφθούν στα αποτελέσματα μη σχετικά tweets.

Επίσης, για να μας επιστρέφει το service πιο ακριβή αποτελέσματα, έχουμε κάνει την εξής παραδοχή: οι λέξεις-κλειδιά που αποτελούν συνώνυμα ή ακρωνύμια άλλων λέξεων, αν περιέχουν το πολύ ένα πεζό γράμμα και όλοι οι υπόλοιποι χαρακτήρες τους είναι κεφαλαία γράμματα ή νούμερα, σύμβολα κλπ, τότε η λέξη θα πρέπει να αναζητηθεί στις δημοσιεύσεις όπως ακριβώς τη διαβάζουμε από τη λίστα. Πραγματοποιείται επομένως ο κατάλληλος έλεγχος, με την κλήση της μεθόδου minDistance() που υλοποιήσαμε προκειμένου να ελέγξουμε αν δύο Strings με το ίδιο πλήθος χαρακτήρων, διαφέρουν κατά έναν το πολύ χαρακτήρα. Η μέθοδος αυτή καλείται στην προκειμένη περίπτωση με δύο ορίσματα, από τα οποία το ένα είναι αυτούσια η λέξη, ενώ το άλλο είναι και πάλι η ίδια λέξη, με όλους τους χαρακτήρες της να έχουν μετατραπεί σε κεφαλαίους. Αν τα δύο ορίσματα διαφέρουν κατά έναν το πολύ χαρακτήρα και συγχρόνως η λέξη που εξετάζουμε αποτελείται από περισσότερους από 2 χαρακτήρες, σημαίνει ότι η λέξη αυτή έχει όλα τα γράμματά της κεφαλαία, εκτός ίσως από ένα. Σε αυτή την περίπτωση, ελέγχουμε αν περιέχεται αυτούσια στη δημοσίευση καλώντας τη μέθοδο contains() της Java, με όρισμα τη λέξη που ψάχνουμε. Αν η λέξη-κλειδί περιέχει το πολύ έναν πεζό χαρακτήρα και ταυτόχρονα αποτελεί συνώνυμο ή ακρωνύμιο, τότε αν δεν περιέχεται στο κείμενο της δημοσίευσης θα πρέπει να προχωρήσουμε στην ανάγνωση και αναζήτηση του επόμενου keyword του πίνακα. Στις περιπτώσεις των λέξεων που έχουν γραφτεί με κεφαλαία αλλά δεν αποτελούν συνώνυμο ή ακρωνύμιο, μετατρέπουμε όλους τους χαρακτήρες τους σε πεζούς μέσω της μεθόδου toLowerCase() και συνεχίζουμε την αναζήτησή τους καλώντας και πάλι τη μέθοδο contains(). Φυσικά, αν μια λέξη βρεθεί στο κείμενο, τότε την προσθέτουμε στον πίνακα με τις λέξεις-κλειδιά που έχουν βρεθεί και ελέγχουμε τη συνθήκη τερματισμού που περιγράφηκε παραπάνω. Αν όχι, προχωράμε στο επόμενο στάδιο της αναζήτησης.

5.3.2.2.2 Διάσπαση σε επιμέρους λέξεις

Σε επόμενη φάση, γίνεται εξαντλητική αναζήτηση του keyword. Συγκεκριμένα, αν αυτό αποτελείται από περισσότερες από μία λέξεις, το χωρίζουμε στις επιμέρους λέξεις τους καλώντας τη μέθοδο split(‘ ‘), δηλαδή χρησιμοποιούμε ως χαρακτήρα διαχωρισμού το κενό. Ταυτόχρονα, χωρίζουμε το κείμενο της δημοσίευσης στις επιμέρους λέξεις του και τις αποθηκεύουμε σε έναν πίνακα. Έπειτα, συγκρίνουμε την κάθε λέξη του keyword με την κάθε λέξη του tweet. Αυτή η διαδικασία πραγματοποιείται εντός δύο βρόχων επανάληψης, από τους οποίους ο ένας είναι εμφωλευμένος μέσα στον άλλο. Στον εξωτερικό βρόχο προσπελαύνονται οι επιμέρους λέξεις του keyword, ενώ ο εσωτερικός διατρέχει τον πίνακα με τις λέξεις του κειμένου της δημοσίευσης. Πριν τη σύγκριση αυτή, που επίσης πραγματοποιείται σε περισσότερα από ένα στάδια, ελέγχουμε αν η λέξη αποτελεί stop word, όπως κάναμε και με το ίδιο το keyword νωρίτερα. Αν είναι όντως stop word, προχωράμε απευθείας στην επόμενη λέξη του ίδιου keyword. Αν όχι, εκτελούμε με τη σειρά τις συγκρίσεις που περιγράφονται παρακάτω.

Πρώτα από όλα, εξετάζουμε με τη χρήση της μεθόδου minDistance(), αν η λέξη της δημοσίευσης και η λέξη-κλειδί είναι απόλυτα ίδιες, εφόσον αποτελούνται από λιγότερους από 6 χαρακτήρες, ή αν διαφέρουν κατά έναν το πολύ χαρακτήρα, σε περίπτωση που οι χαρακτήρες τους είναι περισσότεροι. Αν κάτι τέτοιο δεν συμβαίνει, ελέγχουμε αν η λέξη-κλειδί περιέχεται στη λέξη του tweet, καλώντας τη μέθοδο contains(). Και στους δύο προηγούμενους ελέγχους, αν οι δύο λέξεις διαφέρουν κατά έναν το πολύ χαρακτήρα ή είναι ίδιες, αποθηκεύουμε την συγκεκριμένη λέξη του tweet στον πίνακα actualWords. Αν στο τέλος της αναζήτησης, όλες οι επιμέρους λέξεις του keyword έχει βρεθεί ότι πληρούν τα κριτήρια ώστε να θεωρήσουμε πως βρέθηκαν στο κείμενο, τότε ο πίνακας actualWords θα περιέχει τις λέξεις εκείνες του tweet που έχουν αντιστοιχιστεί στις λέξεις

του keyword που αναζητούμε. Αν από τους δύο ελέγχους δεν προκύψει αντιστοίχιση των συγκρινόμενων λέξεων, πρέπει να προχωρήσουμε στο επόμενο στάδιο συγκρίσεων.

Σε αυτό το στάδιο, αφαιρούμε από τις προς σύγκριση λέξεις τους χαρακτήρες '-' και '/', αν τυχόν υπάρχουν, ώστε να βεβαιωθούμε ότι το γεγονός πως δεν προέκυψε αντιστοίχιση στις προηγούμενες συγκρίσεις δεν οφείλεται στο ότι δύο ίδιες έννοιες είναι γραμμένες με διαφορετικό τρόπο, όπως για παράδειγμα θα συνέβαινε με τις λέξεις "auto-immune" και "autoimmune". Αφού αφαιρέσουμε τους χαρακτήρες αυτούς, στη συνέχεια πραγματοποιούμε τους ίδιους ακριβώς ελέγχους με πριν, καλώντας και πάλι με τη σειρά τις μεθόδους minDistance() και contains(). Αν και πάλι οι έλεγχοι αποτύχουν, περνάμε στο τελευταίο στάδιο, όπου οι καταλήξεις των λέξεων αφαιρούνται με τη χρήση του αλγορίθμου Porter Stemmer.

5.3.2.2.3 Ο αλγόριθμος Porter Stemmer [6]

Χρησιμοποιήθηκε ο συγκεκριμένος αλγόριθμος για τη βελτιστοποίηση της αναζήτησης. Η λειτουργία του είναι να αφαιρεί από τις λέξεις τις καταλήξεις τους, διατηρώντας μόνο το κύριο θέμα της λέξης. Με τον τρόπο αυτό, δύο λέξεις που αποτελούνται από το ίδιο θέμα αλλά οι καταλήξεις τους διαφέρουν, θα αντιστοιχιστούν μεταξύ τους. Θα πρέπει να επισημάνουμε ότι ο αλγόριθμος εφαρμόζεται μόνο στις επιμέρους λέξεις του keyword και όχι στις λέξεις της δημοσίευσης, καθώς οι τελευταίες είναι πολύ περισσότερες και, λόγω και της σημαντικής πολυπλοκότητας του Porter Stemmer, η εκτέλεση του προγράμματος θα απαιτούσε πολύ περισσότερη ώρα για να ολοκληρωθεί. Ταυτόχρονα, επειδή ελέγχουμε μόνο αν η λέξη-κλειδί περιέχεται σε κάποια λέξη του tweet και όχι το αντίστροφο, αρκεί να ελέγξουμε αντίστοιχα αν στο κείμενο περιέχεται μόνο το θέμα της συγκεκριμένης λέξης. Για παράδειγμα, αν έχουμε ως λέξη-κλειδί την λέξη "eyes", ενώ στο κείμενο του tweet υπάρχει η λέξη "eye", ο αλγόριθμος θα θεωρήσει ως κατάληξη της λέξης τα δύο τελευταία γράμματα "-es", και θα τα αφαιρέσει, κάτι που είναι αρκετό για να προκύψει αντιστοίχιση μεταξύ των δύο λέξεων. Κάτι αντίστοιχο θα συμβεί και στην περίπτωση που έχουμε ως λέξη-κλειδί το "painful", ενώ το κείμενο περιέχει τη λέξη "pain". Στην αντίστροφη περίπτωση, όπου δηλαδή είχαμε ως λέξη-κλειδί το "pain" και στο κείμενο τη λέξη "painful", δεν θα χρειαζόταν να εφαρμόσουμε τον αλγόριθμο, αφού η αντιστοίχιση θα είχε βρεθεί σε προηγούμενο στάδιο με την κλήση της μεθόδου contains("pain").

Στη συνέχεια, παρουσιάζονται συνοπτικά τα βήματα που εκτελεί ο αλγόριθμος Porter Stemmer και τις καταλήξεις που αφαιρεί στο καθένα, προκειμένου τελικά να απομονώσει το θέμα της συγκεκριμένης λέξης του keyword:

1. Έλεγχει αν η λέξη τελειώνει σε: '-s', '-es', '-ed' ή '-ing' και απομακρύνει τις καταλήξεις αυτές.
2. Εξετάζεται αν η λέξη που προέκυψε από το προηγούμενο βήμα έχει κατάληξη '-y' και ταυτόχρονα πριν την κατάληξη προηγείται φωνήεν, οπότε ο συγκεκριμένος χαρακτήρας αντικαθίσταται από '-i'.
3. Ελέγχεται το προτελευταίο γράμμα της λέξης που προέκυψε. Ανάλογα με το γράμμα που θα βρεθεί, εξετάζονται τα τελευταία γράμματα της λέξης, με βάση ένα σύνολο από προκαθορισμένες καταλήξεις, και γίνονται οι ανάλογες αντικαταστάσεις. Για παράδειγμα, αν το προτελευταίο γράμμα της λέξης είναι το 'a', τότε αν η λέξη τελειώνει σε '-ational', ολόκληρη η κατάληξη αυτή αντικαθίσταται από την κατάληξη '-ate', ενώ αν το προτελευταίο γράμμα είναι το 't' και η κατάληξη είναι το '-ivity', τότε αντικαθίσταται από την κατάληξη '-ive'.
4. Χρησιμοποιείται η ίδια λογική με το προηγούμενο βήμα, με τη μόνη διαφορά ότι ελέγχεται το τελευταίο αντί για το προτελευταίο γράμμα.
5. Ίδια λογική με το βήμα (3), με διαφορετικές καταλήξεις να ελέγχονται και να αφαιρούνται, χωρίς να αντικαθίστανται από άλλες.

6. Στο τελευταίο βήμα, ελέγχεται αν η λέξη έχει κατάληξη ‘-e’. Αν το ‘e’ δεν είναι ο μοναδικός χαρακτήρας που έχει απομείνει στη λέξη μετά τα προηγούμενα βήματα, αφαιρείται και απομένει μόνο το θέμα της λέξης το οποίο θα αναζητηθεί.

Μετά την εφαρμογή του αλγορίθμου Porter Stemmer, ακολουθούν ακριβώς οι ίδιες συγκρίσεις που κάναμε μετά τη διάσπαση της λέξης-κλειδί σε επιμέρους λέξεις.

5.3.2.3 Αξιολόγηση προηγούμενης και νέας υλοποίησης

Παρά τη σημαντική απουσία ευελιξίας, η οποία οδηγεί σε σαφώς λιγότερο αποδοτική αναζήτηση με σχετικά λίγα επιστρεφόμενα αποτελέσματα, μπορούμε να εντοπίσουμε και ορισμένα πλεονεκτήματα στην αρχική έκδοση της υπηρεσίας. Ως τέτοιο θα μπορούσαμε να θεωρήσουμε την πολύ μεγάλη ταχύτητα με την οποία διενεργείται η αναζήτηση, η οποία και ολοκληρώνεται εντός ενός χρονικού διαστήματος μερικών millisecond. Ένα άλλο πλεονέκτημα αποτελεί η υψηλή ακρίβεια της αναζήτησης, ή αλλιώς precision, δηλαδή ο λόγος των ορθών αποτελεσμάτων ως προς τα συνολικά αποτελέσματα που επιστράφηκαν, καθώς το ακριβές ταίριασμα που αναζητείται, εγγυάται στο 83.3% των περιπτώσεων την ορθότητα των αποτελεσμάτων. Παρ’ όλα αυτά, με μια σύντομη επισκόπηση των διαθέσιμων προς εξέταση tweets, εκτιμάται ότι το πλήθος των ορθών αποτελεσμάτων που επιστράφηκαν, σε σχέση με αυτά που ιδανικά θα έπρεπε να επιστραφούν, στη μέση περίπτωση δεν ξεπερνά το 50%. Το ποσοστό αυτό αποκαλείται recall.

Στο πλαίσιο της παρούσας διπλωματικής ζητούμενο ήταν η ανάκτηση όσο το δυνατόν περισσότερων και ταυτόχρονα ορθών αποτελεσμάτων, με την προσθήκη επιπλέον λειτουργιών για πιο σύνθετη αναζήτηση. Έτσι, κρίθηκε αναγκαία η σημαντική αύξηση της πολυπλοκότητας της αναζήτησης, με το ανάλογο βέβαια τίμημα κυρίως στον χρόνο ολοκλήρωσης της διαδικασίας, ο οποίος πλέον κυμαίνεται στα 5-6 δευτερόλεπτα. Αντίθετα, το precision αυξήθηκε λόγω της μεγαλύτερης ακρίβειας των αλγορίθμων, και βρίσκεται κατά μέσο όρο στο 85.1%. Επίσης, το ποσοστό του recall παρουσιάζει σημαντική βελτίωση λόγω της επιστροφής περισσότερων αποτελεσμάτων, και κυμαίνεται πλέον κοντά στο 96.6% στη μέση περίπτωση. Για την σύγκριση των ποσοστών του precision και του recall δημιουργήθηκαν οι πίνακες της εικόνας 5.4, στους οποίους φαίνονται τα παραπάνω μεγέθη κατά την αρχική και τελική μορφή του service για κάποιες επιλεγμένες λέξεις-κλειδιά. Οι αριθμοί αυτοί προέκυψαν από μετρήσεις σε ένα σύνολο 741 δημοσιεύσεων που είχε συλλέξει το service ως τη στιγμή που πραγματοποιήθηκαν οι μετρήσεις αυτές. Φυσικά τα ποσοστά αυτά μπορεί να μεταβληθούν ελαφρά αν αλλάξουν τα keywords.

Keywords	Αρχικό service				
	Συνολικά Αποτελέσματα	Ορθά αποτελέσματα	Ιδανικά αποτελέσματα	Precision	Recall
fibromyalgia	5	5	7	100%	71%
Auto-immune disease	2	2	22	100%	9.1%
Dry Eyes	4	4	21	100%	19%
Liver	6	2	2	33.3%	100%
M.O.				83.3%	49.8%

Keywords	Τελικό service				
	Συνολικά Αποτελέσματα	Ορθά αποτελέσματα	Ιδανικά αποτελέσματα	Precision	Recall
fibromyalgia	7	7	7	100%	100%
Auto-immune disease	20	19	22	95%	86.4%
Dry Eyes	22	21	21	95.5%	100%
Liver	4	2	2	50%	100%
M.O.				85.1%	96.6%

Εικόνα 5.4: Precision – Recall στην αρχική και τελική μορφή του service

5.4 Επιστροφή αποτελεσμάτων

Η επιστροφή των αποτελεσμάτων είναι μια διαδικασία που περνάει και αυτή με τη σειρά της από διαδοχικά στάδια. Ξεκινά από το service αναζήτησης, όπου μετά από κάθε επιτυχημένη αναζήτηση των επιλεγμένων keyword στο κείμενο ενός tweet, το tweet αυτό προστίθεται στη λίστα με τα αποτελέσματα. Μετά τον έλεγχο όλων των διαθέσιμων δημοσιεύσεων, η λίστα αυτή στέλνεται ως JSON body του μηνύματος απάντησης του service αναζήτησης στο service επεξεργασίας, και στη συνέχεια στο frontend, όπου παρουσιάζονται στον χρήστη με τη μορφή πίνακα μέσω του user interface, όπως θα δούμε και παρακάτω.

5.4.1 Συλλογή δημοσιεύσεων

Αν μία ή περισσότερες επιλεγμένες λέξεις-κλειδιά, ανάλογα και με τη λογική της αναζήτησης, βρεθούν στο κείμενο του tweet που εξετάζεται, το συγκεκριμένο tweet θα προστεθεί στα αποτελέσματα. Συγκεκριμένα, τα στοιχεία του tweet που πρέπει να προσθέσουμε είναι το όνομα και η φωτογραφία προφίλ του συντάκτη-χρήστη του Twitter, το κείμενο της δημοσίευσης, κάποια εικόνα που τυχόν περιέχεται στη δημοσίευση, καθώς και οι λέξεις-κλειδιά που αναζητήθηκαν και βρέθηκαν σε αυτή.

Τονίζουμε ότι πριν προσθέσουμε το κείμενο της δημοσίευσης στη λίστα των αποτελεσμάτων, εντοπίζουμε ξανά στο κείμενο όλες τις λέξεις-κλειδιά που βρήκαμε και αποθηκεύσαμε προσωρινά σε έναν πίνακα. Έπειτα, τις περιβάλλουμε με το ειδικό tag της html <mark>…</mark>, και τέλος προσθέτουμε το κείμενο μαζί με τα υπόλοιπα στοιχεία της δημοσίευσης στη λίστα. Το κείμενο που περικλείεται από αυτό το tag, εμφανίζεται στον browser με μια κίτρινη υπογράμμιση. Με αυτόν τον τρόπο, ο χρήστης της εφαρμογής διαπιστώνει κατευθείαν

ότι μια δημοσίευση περιέχει όντως τις λέξεις που αναζήτησε, ενώ βλέπει και σε ποιο σημείο του κειμένου αυτές βρίσκονται. Με άλλα λόγια, η υπογράμμιση των keywords αποτελεί έναν τρόπο για να αξιολογήσει κανείς άμεσα την ορθότητα των επιστρεφόμενων αποτελεσμάτων και κατ' επέκταση το αν η εφαρμογή μας λειτουργεί πράγματι σωστά.

Μόλις η αναζήτηση ολοκληρωθεί, με τον έλεγχο όλων των δημοσιεύσεων, η λίστα με τα αποτελέσματα στέλνεται ως response στο service επεξεργασίας των δεδομένων, και από εκεί στέλνεται απευθείας στο frontend, δίχως να υποστεί κάποια επιπλέον επεξεργασία. Μαζί με τη λίστα στέλνουμε στο frontend και ορισμένα ακόμα στοιχεία, όπως τα keywords της αναζήτησης, τη λογική της αναζήτησης, το αν συμπεριλάβαμε και τα συνώνυμα ή τα ακρωνύμια, όπως επίσης και το πλήθος των αποτελεσμάτων.

6

Σύνοψη

6.1 Συμπεράσματα

Στην παρούσα διπλωματική εργασία μελετήθηκε και υλοποιήθηκε μία ολοκληρωμένη διαδικτυακή εφαρμογή αναζήτησης στο Twitter. Βασικό αντικείμενο της εργασίας αποτέλεσε η εφαρμογή των πλέον αποτελεσματικών μεθόδων και τεχνικών αναζήτησης λέξεων μέσα σε κείμενο, όπως επίσης και η εξεύρεση του καλύτερου δυνατού συνδυασμού των τεχνικών αυτών, με σκοπό την ανάκτηση όσο το δυνατόν περισσότερων και σωστότερων αποτελεσμάτων. Μετά από διαφορετικούς συνδυασμούς που δοκιμάστηκαν, προέκυψε ότι είναι προτιμότερο η αναζήτηση των λέξεων να πραγματοποιείται σε διαδοχικά στάδια, καθένα από τα οποία έχει μεγαλύτερη πολυπλοκότητα από το προηγούμενο και μικρότερη από το επόμενο. Αυτός ο συνδυασμός φαίνεται να δίνει τα καλύτερα αποτελέσματα στον ελάχιστο δυνατό χρόνο στη μέση περίπτωση, δηλαδή όταν κατά μέσο όρο χρειάζεται να φτάσουμε μέχρι τα μεσαία στάδια της αναζήτησης για να βρούμε τη λέξη-κλειδί.

Παράλληλα, μελετήθηκε σε βάθος ο τρόπος λειτουργίας και οι δυνατότητες μιας διαδικτυακής εφαρμογής προσανατολισμένης στις υπηρεσίες. Πυρήνα της εφαρμογής αποτέλεσαν τα επιμέρους services, η σωστή και αποδοτική λειτουργία των οποίων ήταν και ο βασικός στόχος της εργασίας. Ιδιαίτερο ενδιαφέρον παρουσιάζει η επικοινωνία μεταξύ των services και η ανταλλαγή μηνυμάτων μέσω GET και POST HTTP requests, και κατ' επέκταση η συγγραφή κώδικα στις γλώσσες Java και JavaScript για την αποστολή, τη λήψη και την επεξεργασία των αιτημάτων αυτών. Μας απασχόλησε, επίσης, και η υλοποίηση άλλων δευτερευουσών λειτουργιών. Τέτοιες ήταν η δημιουργία ενός λειτουργικού και φιλικού προς το χρήστη user interface, ο συγχρονισμός των διαφορετικών Threads που λειτουργούν ταυτόχρονα προκειμένου να αποφύγουμε τις εξαιρέσεις, καθώς και η ανάγνωση αρχείων σε έναν Servlet και η επεξεργασία μιας οντολογίας, με σκοπό την κατασκευή μιας λίστας αντικειμένων με συγκεκριμένη ιεραρχία.

Η εφαρμογή στην τελική της μορφή καθιστά εύκολη και γρήγορη τη σύνθετη αναζήτηση στο Twitter, δηλαδή την αναζήτηση που περιλαμβάνει εισαγωγή λέξεων από μια κλασική γραμμή αναζήτησης, αλλά συγχρόνως και επιλογή από ένα σύνολο προκαθορισμένων όρων που δίνονται στο χρήστη με τη μορφή καταλόγου, παρέχοντας επιπλέον στον χρήστη τη δυνατότητα να αποφασίσει αν η αναζήτηση θα πραγματοποιηθεί με λογική and ή or. Φυσικά, τόσο τα services όσο και το user interface, λόγω της αυτονομίας με την οποία έχουν σχεδιαστεί να λειτουργούν, μπορούν να τροποποιηθούν ώστε να χρησιμοποιηθούν μαζί ή ξεχωριστά σε κάποια άλλη εφαρμογή, η οποία μπορεί να περιλαμβάνει αναζήτηση είτε σε άλλα social media, όπως το Facebook και το YouTube με τη βοήθεια των αντίστοιχων APIs, είτε άμεσα σε μια βάση δεδομένων κάποιας βιβλιοθήκης, εταιρίας κλπ. Δημιουργήθηκε έτσι ένα εργαλείο χρήσιμο σε πολλές περιπτώσεις όπου απαιτείται εξειδικευμένη αναζήτηση με τη χρήση οντολογιών.

6.2 Μελλοντικές επεκτάσεις

Είναι σαφές πως η εφαρμογή μας επιδέχεται πολλών βελτιώσεων και επεκτάσεων. Μια σημαντική βελτίωση θα μπορούσε να αφορά στον τρόπο που γίνεται η αναζήτηση. Σε αυτήν την κατεύθυνση θα συνέβαλε πολύ η χρήση ενός ολοκληρωμένου λεξικού, από το οποίο θα συλλέγονται όχι μόνο οι λέξεις που έχουν το ίδιο θέμα με το keyword που αναζητείται, αλλά και συγγενικές ετυμολογικά έννοιες, δηλαδή έννοιες που προέρχονται από την ίδια αρχική ρίζα. Θα βοηθούσε επίσης ιδιαίτερα αν οι έννοιες περιέχονταν στο λεξικό αυτό μεταφρασμένες σε περισσότερες από μία γλώσσες. Έτσι, σε tweets γραμμένα σε διαφορετική γλώσσα από το keyword, θα υπήρχε η δυνατότητα αυτό να αναζητηθεί μεταφρασμένο στην αντίστοιχη γλώσσα, οπότε θα παίρναμε ακόμα περισσότερα και πιο ακριβή αποτελέσματα.

Ακόμα, τροποποιήσεις μπορούν να γίνουν και σε ό,τι αφορά τη διεπαφή χρήστη, και συγκεκριμένα τον πίνακα των αποτελεσμάτων. Για παράδειγμα, θα ήταν καλή ιδέα να δίνεται στους χρήστες η δυνατότητα να ταξινομήσουν τα αποτελέσματα ως προς κάποια στήλη του πίνακα. Με τον τρόπο αυτό, θα είναι δυνατό να έχουμε αποτελέσματα ταξινομημένα αλφαβητικά με βάση το όνομα του συντάκτη της δημοσίευσης ή του keyword που βρέθηκε σε αυτή, ή χρονολογικά με βάση την ημερομηνία της δημοσίευσης, σε φθίνουσα ή αύξουσα σειρά. Οι παραπάνω λειτουργικότητες δεν ενσωματώθηκαν στην εφαρμογή, καθώς κρίθηκε ότι υπερβαίνουν τις ανάγκες και τις απαιτήσεις αυτής της εργασίας. Παρ' όλα αυτά, θα μπορούσαν να προστεθούν σε ενδεχόμενη αναβάθμιση της εφαρμογής ή στο πλαίσιο κάποιας άλλης διπλωματικής εργασίας στο μέλλον.

Παράρτημα

Εδώ παρατίθεται ο κώδικας των πιο βασικών λειτουργιών της εφαρμογής, καθώς και ορισμένων Java μεθόδων που υλοποιήθηκαν στο πλαίσιο της εργασίας.

Π.1 Ανάγνωση της οντολογίας

Η ανάγνωση της οντολογίας και η εξαγωγή των keywords και των πληροφοριών που τις συνοδεύουν, πραγματοποιείται από τη μέθοδο doGet της κλάσης OntServlet και ορισμένες βοηθητικές μεθόδους στο service επεξεργασίας των δεδομένων. Παρατίθεται στη συνέχεια ο κώδικας Java:

```
private static void findClasses(){
      Set<OWLClass> ontClasses = new HashSet<OWLClass>();
      ontClasses = ontology.getClassesInSignature();
      allClasses = new ArrayList<aClass>();

      for (Iterator<OWLClass> it = ontClasses.iterator(); it.hasNext();){
            aClass f = new aClass();
            f.name = it.next();
            f.id = f.name.getIRI().getFragment();
            for(OWLAnnotationAssertionAxiom a:
            ontology.getAnnotationAssertionAxioms(f.name.getIRI())) {
                  if(a.getProperty().isLabel()) {
                        if(a.getValue() instanceof OWLLiteral) {
                              OWLLiteral val = (OWLLiteral) a.getValue();
                              f.label = val.getLiteral();
                        }
                  }
                  else if(a.getProperty().isComment()) {
                        if(a.getValue() instanceof OWLLiteral) {
                              OWLLiteral val = (OWLLiteral) a.getValue();
                              f.comment = val.getLiteral();
                        }
                  }
                  else{
                        if(a.getProperty().getIRI().getFragment().equals("
                        aka"){
                              if(a.getValue() instanceof OWLLiteral) {
                                    OWLLiteral val = (OWLLiteral)
                                    a.getValue();
                                    f.aka = val.getLiteral();
                              }
                        }
                        else if
                        (a.getProperty().getIRI().getFragment().equals("ac
                        ronym")){
                              if(a.getValue() instanceof OWLLiteral) {
                                    OWLLiteral val = (OWLLiteral)
                                    a.getValue();
                                    f.acronym = val.getLiteral();
                              }
                        }
                  }
```

```
            }
            allClasses.add(f);
      }
}

private static void findSubclasses(){
      for (final OWLSubClassOfAxiom subClasse :
      ontology.getAxioms(AxiomType.SUBCLASS_OF)){
            OWLClass sup = (OWLClass) subClasse.getSuperClass();
            OWLClass sub = (OWLClass) subClasse.getSubClass();

            if (sup instanceof OWLClass && sub instanceof OWLClass){
                  int i;
                  for(i=0; i<allClasses.size(); i++){
                        if (sup.equals(allClasses.get(i).name)) break;
                  }
                  int j;
                  for(j=0; j<allClasses.size(); j++){
                        if (sub.equals(allClasses.get(j).name)){
                              allClasses.get(i).subClasses.add(allClasses.
                              get(j));
                              allClasses.get(j).isSubClass = true;
                              break;
                        }
                  }
            }
      }
}
protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException{
      // TODO Auto-generated method stub

      manager = OWLManager.createOWLOntologyManager();
      Scanner s = new Scanner(new BufferedReader(new
      FileReader(getServletContext().getRealPath("/WEB-INF/infos.txt"))));
      String[] line1 = s.nextLine().split(":");
      System.out.println(line1[1].trim());
      documentIRI = IRI.create(getServletContext().getResource("/WEB-
      INF/"+line1[1].trim()));
      try{
            ontology =
            manager.loadOntologyFromOntologyDocument(documentIRI);
            findClasses();
            findSubclasses();
      }
      catch (OWLOntologyCreationException e) {
            e.printStackTrace();
      }
      …
}
```

Π.2 Εκκίνηση αναζήτησης από τον χρήστη

Ακολουθεί η συνάρτηση getResults() της JavaScript, η οποία περιέχει τις λειτουργίες που εκτελούνται στο frontend, από τη στιγμή που ο χρήστης ξεκινάει μια αναζήτηση ως την επιστροφή και εμφάνιση των αποτελεσμάτων. Ο κώδικας Java των ενδιάμεσων λειτουργιών στο backend παρατίθεται στην επόμενη ενότητα:

```
function getResults() {
      var HttpClient = function() {
            this.get = function(aUrl, aCallback) {
                  var anHttpRequest = new XMLHttpRequest();
                  anHttpRequest.onreadystatechange = function() {
                        if (anHttpRequest.readyState == 4 &&
                        anHttpRequest.status == 200)
                              aCallback(anHttpRequest.responseText);
                  }
                  anHttpRequest.open( "GET", aUrl, true );
                  anHttpRequest.send( null );
            }
      }
      var theurl;
      var myItems = sessionStorage.getItem("items");
      var myKeywords = document.getElementById('mykeywords');
      if (!(myKeywords && myKeywords.value.trim()) && !myItems){
            alert("Please enter or select some keywords");
            return false;
      }
      var logicType = sessionStorage.getItem("logic");
      var akaLogicType = sessionStorage.getItem("akalogic");
      if(myKeywords && myKeywords.value.trim())
            theurl='OntServlet?items='+myItems+'&mykeywords='+myKeywords.v
            alue.trim()+'&logic='+logicType+'&akalogic='+akaLogicType;
      else
            theurl='OntServlet?items='+myItems+'&mykeywords=null&logic='+l
            ogicType+'&akalogic='+akaLogicType;
      var client = new HttpClient();
      client.get(theurl, function(response) {
            var response1 = JSON.parse(response);
            var ont = response1.ontology;
            var twitter = response1.tweets;
            var query = response1.query;
            var total = response1.total;
            var logic = response1.logic;
            var akalogic = response1.akalogic;
            if((myKeywords && myKeywords.value.trim()) || myItems){
                  var myResults = '';
                  myResults += "<h4 align='left'>"+total+" results for
                  your query: "+query+"<h4><h5 align='left'>logic:
                  "+logic+"<br>akas and acronyms: "+akalogic+"<h5>";
                  $('#myResults').html(myResults);
            }
            var tweets = '';
            if (twitter.length>0){

                  tweets += '<table table-layout="auto" width="100%"><tr
                  padding="2px"><th>user</th><th>text</th><th>date</th><th
                  >query annotation</th></tr>';
                  for(i=0; i<twitter.length; i++){
                        if (twitter[i].postPhoto==="")
                              tweets += '<tr
                              align="center"><td>'+twitter[i].poster+'<br>
                              <br><img
                              src="'+twitter[i].sourceImg+'"></td><td>'+tw
                              itter[i].postText+'</td><td>'+twitter[i].pos
                              tDate+'</td><td>'+twitter[i].keyword+'</td><
                              /tr>';
                        else
```

```
                              tweets += '<tr
                              align="center"><td>'+twitter[i].poster+'<br>
                              <br><img
                              src="'+twitter[i].sourceImg+'"></td><td>'+tw
                              itter[i].postText+'<br><img
                              src="'+twitter[i].postPhoto+'"
                              style="width:150px;height:150px;"></td><td>'
                              +twitter[i].postDate+'</td><td>'+twitter[i].
                              keyword+'</td></tr>';
                        }
                  }
                  $('#tweets').html(tweets);
            });
}
```

Π.3 Ανάκτηση των keywords και λήψη αποτελεσμάτων

Τέλος, στην κλάση OntServlet του service επεξεργασίας περιέχονται οι μέθοδοι της Java για την αναζήτηση των επιλεγμένων keywords με βάση τα id τους στην οντολογία και την εξαγωγή τους από αυτή, καθώς και για την λήψη των αποτελεσμάτων της αναζήτησης που στέλνονται από το service αναζήτησης:

```
private static void findSubJSON(List<JSONObject> subs, aClass ac){
      for(int i=0; i<ac.subClasses.size(); i++){
            JSONObject ac1 = new JSONObject();
            List<JSONObject> subs1 = new ArrayList<JSONObject>();
            findSubJSON(subs1, ac.subClasses.get(i));
            try {
                  String content="";
                  ac1.put("id", ac.subClasses.get(i).id);
                  ac1.put("text", ac.subClasses.get(i).label);
                  JSONObject cont = new JSONObject();
                  content+=ac.subClasses.get(i).label;
                  if(!(ac.subClasses.get(i).comment.equals("")))
                        content+="<br/>"+ac.subClasses.get(i).comment;
                  if(!(ac.subClasses.get(i).aka.equals("")))
                        content+="<br/>aka: "+ac.subClasses.get(i).aka;
                  if(!(ac.subClasses.get(i).acronym.equals("")))
                        content+="<br/>acronym:
                        "+ac.subClasses.get(i).acronym;
                  cont.put("content", content);
                  ac1.put("li_attr",cont);
                  ac1.put("children", subs1);

                  subs.add(ac1);
            } catch (JSONException e) {
                  // TODO Auto-generated catch block
                  e.printStackTrace();
            }
      }
}

private static String getSubKeywords(String keywords, aClass checked){
      if(keywords.equals("")) keywords += checked.label;
      else keywords += "|"+checked.label;
      if(akalogic.equals("yes")){
            if(!(checked.aka.equals(""))){
```

```
                List<String> akaList =
                Arrays.asList(checked.aka.split(",[ ]*"));
                for(int i=0; i<akaList.size(); i++){
                      keywords += "|"+akaList.get(i)+" (aka of
                      "+checked.label+")";
                }
          }
          if(!(checked.acronym.equals(""))){
                List<String> acronymList =
                Arrays.asList(checked.acronym.split(",[ ]*"));
                for(int i=0; i<acronymList.size(); i++){
                      keywords += "|"+acronymList.get(i)+" (acronym of
                      "+checked.label+")";
                }
          }
    }
    for(int i=0; i<checked.subClasses.size(); i++)
          keywords = getSubKeywords(keywords,checked.subClasses.get(i));
    return keywords;
}

private static List<JSONObject> fetchOntology(List<JSONObject> ontology)
{
    for(int i=0; i<allClasses.size(); i++){
          if(!allClasses.get(i).isSubClass){
                try {
                      JSONObject ac = new JSONObject();
                      JSONObject cont = new JSONObject();
                      String content="";
                      List<JSONObject> subs = new
                      ArrayList<JSONObject>();
                      findSubJSON(subs, allClasses.get(i));
                      ac.put("id", allClasses.get(i).id);
                      ac.put("text", allClasses.get(i).label);
                      content+=allClasses.get(i).label;
                      if(!(allClasses.get(i).comment.equals("")))
                            content+="<br/>"+allClasses.get(i).comment;
                      if(!(allClasses.get(i).aka.equals("")))
                            content+="<br/>aka: "+allClasses.get(i).aka;
                      if(!(allClasses.get(i).acronym.equals("")))
                            content+="<br/>acronym:
                            "+allClasses.get(i).acronym;
                      cont.put("content", content);

                      ac.put("li_attr",cont);
                      ac.put("children", subs);

                      ontology.add(ac);
                } catch (JSONException e) {
                      // TODO Auto-generated catch block
                      e.printStackTrace();
                }
          }
    }
    return ontology;
}

protected void doGet(HttpServletRequest request, HttpServletResponse
response) throws ServletException, IOException{
    …
    JSONObject all = new JSONObject();
```

```
List<JSONObject> ontology = new ArrayList<JSONObject>();
List<JSONObject> tweets = new ArrayList<JSONObject>();
String querykeywords="";
String checked = request.getParameter("items");
String myKeywords = request.getParameter("mykeywords");
String logic = request.getParameter("logic");
akalogic = request.getParameter("akalogic");
if(checked.equals("null") && myKeywords.equals("null")) ontology =
fetchOntology(ontology);
else{
      String line2[] = s.nextLine().split(":");
      String line3[] = s.nextLine().split(":");
      URL url = new
      URL("http://"+line2[1].trim()+":"+line3[1].trim()+"/SMA_Adapte
      r/TwitterServlet");
      List<String> keywordsList = new ArrayList<String>();
      if(!checked.equals("null")){
            List<String> checkedList =
            Arrays.asList(checked.split(","));
            ontology = fetchOntology(ontology);
            for(int j=0; j<checkedList.size(); j++){
                  for(int i=0; i<allClasses.size(); i++){
                  if(checkedList.get(j).equals(allClasses.get(i).id)
                  ){
                        try{
                              String keywords = getSubKeywords("",
                              allClasses.get(i));
                              keywordsList.add(keywords);
                              if(querykeywords.equals(""))
                              querykeywords =
                              allClasses.get(i).label;
                              else querykeywords += ",
                              "+allClasses.get(i).label;
                              }
                        catch (Exception e) {
                              System.out.println(e);
                        }
                        break;
                  }
            }
      }
}
if(!myKeywords.equals("null")){
      List<String> mykeywords=Arrays.asList(myKeywords.split(","));
      for(int j=0; j<mykeywords.size(); j++){
            keywordsList.add(mykeywords.get(j).trim());
            if(querykeywords.equals("")) querykeywords =
            mykeywords.get(j).trim();
            else querykeywords += ", "+mykeywords.get(j).trim();
      }
}
try{
      JSONObject params = new JSONObject();
      params.put("keywords", keywordsList);
      params.put("place", "");
      params.put("logic", logic);
      String postData = params.toString();
      byte[] postDataBytes = postData.getBytes("UTF-8");
      HttpURLConnection conn =
      (HttpURLConnection)url.openConnection();
      conn.setRequestMethod("POST");
```

```
            conn.setRequestProperty("Content-Type", "application/JSON");
            conn.setRequestProperty("Content-Length",
            String.valueOf(postDataBytes.length));
            conn.setDoOutput(true);
            conn.getOutputStream().write(postDataBytes);
            Reader in = new BufferedReader(new
            InputStreamReader(conn.getInputStream(), "UTF-8"));
            StringBuilder sb = new StringBuilder();
            for (int c; (c = in.read()) >= 0;) sb.append((char)c);
            String resp = sb.toString();
            List<String> myresp = Arrays.asList(resp.split("\\[",2));
            String JSONarr = "[" + myresp.get(1);
            JSONArray JSONarray = new JSONArray(JSONarr);
            for (int k = 0; k < JSONarray.length(); k++) {
                  tweets.add(JSONarray.getJSONObject(k));
            }
      }
      catch (Exception e) {
            System.out.println(e);
      }
}
try {
      all.put("ontology", ontology);
      all.put("tweets", tweets);
      all.put("query", querykeywords);
      all.put("total", tweets.size());
      all.put("logic", logic);
      all.put("akalogic", akalogic);
} catch (JSONException e) {
      // TODO Auto-generated catch block
      e.printStackTrace();
}
response.setContentType("text/html; charset=UTF-8");
response.setCharacterEncoding("UTF-8");
PrintWriter pw = response.getWriter();
pw.print(all.toString());
pw.close();
}
```

Βιβλιογραφία

[1] Antoniou, G., van Harmelen, F. (2009). *Εισαγωγή στο Σημασιολογικό Ιστό*, 2η αμερικανική έκδοση, Αθήνα: Εκδόσεις Κλειδάριθμος, σσ. 19-26.

[2] Apache Tomcat. Ανάκτηση από: http://tomcat.apache.org/

[3] Connolly, D. (2014, Σεπτέμβριος 01). *Hypertext Transfer Protocol -- HTTP/1.1*. Ανάκτηση Αύγουστος 28, 2019, από: https://www.w3.org/Protocols/rfc2616/rfc2616-sec9.html

[4] jsTree. Ανάκτηση από: https://www.jstree.com/

[5] Online JSON Viewer. Ανάκτηση Μάρτιος 05, 2020, από: http://jsonviewer.stack.hu/

[6] Porter Stemmer Algorithm in Java. Ανάκτηση Νοέμβριος 27, 2019, από: https://tartarus.org/martin/PorterStemmer/java.txt

[7] Wikipedia. (2019, Ιούλιος 10). Ανάκτηση Σεπτέμβριος 04, 2019, από: https://en.wikipedia.org/wiki/AJAX_(programming)

[8] Wikipedia. (2019, Αύγουστος 20). Ανάκτηση Αύγουστος 28, 2019, από: https://en.wikipedia.org/wiki/Client-server_model

[9] Wikipedia. (2019, Αύγουστος 23). Ανάκτηση Αύγουστος 28, 2019, από: https://en.wikipedia.org/wiki/Hypertext_Transfer_Protocol

[10] Wikipedia. (2019, Σεπτέμβριος 06). Ανάκτηση Σεπτέμβριος 08, 2019, από: https://en.wikipedia.org/wiki/Java_(programming_language)

[11] Wikipedia. (2019, Αύγουστος 26). Ανάκτηση Αύγουστος 30, 2019, από: https://en.wikipedia.org/wiki/JavaScript

[12] Wikipedia. (2019, Αύγουστος 29). Ανάκτηση Σεπτέμβριος 02, 2019, από: https://el.wikipedia.org/wiki/JQuery

[13] Wikipedia. (2019, Σεπτέμβριος 03). Ανάκτηση Σεπτέμβριος 04, 2019, από: https://el.wikipedia.org/wiki/JSON

[14] Wikipedia. (2019, Αύγουστος 19). Ανάκτηση Αύγουστος 20, 2019, από: https://en.wikipedia.org/wiki/Twitter

[15] Βαρβαρίγου, Θ. (2016, Απρίλιος 12). *JSPs – Servlets*. Ανάκτηση Αύγουστος 29, 2019, από: http://ecourses.dbnet.ntua.gr/el/diktyakos_programmatismos/dialejeis___ergastiria/jsp___servlets.html

[16] Γεργατσούλης Μ., Παπαθεοδώρου Χ. (n.d.). *Εισαγωγή στις Οντολογίες και το Σημασιολογικό Ιστό*. Ανάκτηση Σεπτέμβριος 15, 2019, από: https://eclass.aueb.gr/modules/document/file.php/INF180/lectures/kos-5.2-ontology.pdf

[17] Γκότσε, Μ. (2015, Ιούλιος). *Έξυπνες ροές εργασίας RESTful διαδικτυακών υπηρεσιών με τη βοήθεια του εργαλείου WebHookIt,* Διπλωματική Εργασία, Εθνικό Μετσόβιο Πολυτεχνείο, Αθήνα, σσ. 19-20, 31-43.

[18] Η οντολογία που χρησιμοποιήθηκε, αναπτύχθηκε στα πλαίσια του Ευρωπαϊκού ερευνητικού έργου *HARMONIzation and integrative analysis of regional, national and international Cohorts on primary Sjögren's Syndrome (pSS) towards improved stratification, treatment and health policy making (HarmonicSS).* Ανάκτηση Σεπτέμβριος 15, 2019, από: https://cordis.europa.eu/project/id/731944 και https://www.harmonicss.eu/

[19] *Το μοντέλο πελάτη εξυπηρετητή.* Ανάκτηση Αύγουστος 28, 2019, από: https://ipt2013a.wordpress.com/2012/11/15/το-μοντέλο-πελάτη-εξυπηρετητή/