ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ

ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Ανάλυση και αναζήτηση γεωχωρικής πληροφορίας από πηγές του διαδικτύου

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΣΤΑΜΑΤΟΥΚΟΥ ΚΩΝΣΤΑΝΤΙΝΟΥ

Επιβλέπων : Ιωάννης Βασιλείου
Καθηγητής Ε.Μ.Π.

Αθήνα, Ιούνιος 2015

Η σελίδα αυτή είναι σκόπιμα λευκή.

ΕΘΝΙΚΟ ΜΕΤΣΟΒΙΟ ΠΟΛΥΤΕΧΝΕΙΟ
ΣΧΟΛΗ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ
ΚΑΙ ΜΗΧΑΝΙΚΩΝ ΥΠΟΛΟΓΙΣΤΩΝ
ΤΟΜΕΑΣ ΤΕΧΝΟΛΟΓΙΑΣ ΠΛΗΡΟΦΟΡΙΚΗΣ
ΚΑΙ ΥΠΟΛΟΓΙΣΤΩΝ

Ανάλυση και αναζήτηση γεωχωρικής πληροφορίας από πηγές του διαδικτύου

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

του

ΣΤΑΜΑΤΟΥΚΟΥ ΚΩΝΣΤΑΝΤΙΝΟΥ

Επιβλέπων : Ιωάννης Βασιλείου
Καθηγητής Ε.Μ.Π.

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την .....

.....	.....	.....
Ιωάννης Βασιλείου	Νεκτάριος Κοζύρης	Θοδωρής Δαλαμάγκας
Καθηγητής Ε.Μ.Π.	Καθηγητής Ε.Μ.Π.	Ερευνητής Β

Αθήνα, Ιούνιος 2015

.....

ΚΩΝΣΤΑΝΤΙΝΟΣ ΣΤΑΜΑΤΟΥΚΟΣ

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

Περίληψη

Το πλήθος των γεωχωρικών δεδομένων στο Διαδίκτυο που προέρχονται από χρήστες αυξάνεται με ραγδαίους ρυθμούς, καταλήγοντας να αποτελεί μία πολύτιμη πηγή πληροφορίας για τη δημιουργία, τη βελτίωση και τον εμπλουτισμό γεωχωρικών εφαρμογών και υπηρεσιών. Η αξιοποίηση αυτής της πληροφορίας, ωστόσο, είναι κάθε άλλο παρά τετριμμένη και απαιτεί αφενός τη δημιουργία εργαλείων συλλογής και ενοποίησης της και αφετέρου τη δημιουργία εφαρμογών που μπορούν να την επεξεργάζονται και να την παρουσιάζουν στο χρήστη με τέτοιο τρόπο ώστε να καθιστούν δυνατή την εξαγωγή χρήσιμων συμπερασμάτων.

Η παρούσα διπλωματική εργασία επικεντρώνεται κυρίως στο δεύτερο κομμάτι, την ανάπτυξη δηλαδή μιας web εφαρμογής που, χρησιμοποιώντας ως βάση υπάρχουσα ενοποιημένη πληροφορία που έχει ήδη εξαχθεί από διάφορες πηγές, υπολογίζει και παρουσιάζει γραφικά στο χρήστη διάφορα αποτελέσματα. Συγκεκριμένα, τα γεωχωρικά δεδομένα που έχουμε στη διάθεσή μας αφορούν Σημεία Ενδιαφέροντος, Φωτογραφίες και Εκδηλώσεις. Σκοπός της εφαρμογής είναι να διευκολύνει τον χρήστη να αναζητήσει τα δεδομένα αυτά ανά περιοχές και να τα φιλτράρει χρησιμοποιώντας λέξεις κλειδιά ή με βάση την κατηγοριοποίηση τους. Επίσης, να μπορεί να συσχετίσει δεδομένα διαφορετικού τύπου μεταξύ τους τοπικά και εννοιολογικά (πχ Φωτογραφίες με Σημεία ενδιαφέροντος). Τέλος θέλουμε να μπορεί να αντλήσει πληροφορίες για τη συγκέντρωση και την κατανομή των δεδομένων σε μια πόλη. Ειδικότερα, επικεντρωνόμαστε στην εξαγωγή Περιοχών Ενδιαφέροντος και Οδών Ενδιαφέροντος, και κατ' επέκταση στο πώς αυτά μπορούν να παρουσιαστούν στον χρήστη με ένα μικρό αλλά αντιπροσωπευτικό σύνολο αποτελεσμάτων.

Η ανάπτυξη της εφαρμογής αυτής έγινε σε δύο σκέλη. Αρχικά, αναπτύχθηκε η διεπαφή υπηρεσιών κατά REST που δίνουν πρόσβαση στα δεδομένα και εκτελούν τους αλγόριθμους φιλτραρίσματος και συσχετισμού των δεδομένων. Έπειτα, αναπτύχθηκε το γραφικό περιβάλλον χρήστη, το οποίο χρησιμοποιώντας την προαναφερθείσα διεπαφή φέρνει τα δεδομένα στην επιφάνεια και τα απεικονίζει γραφικά.

Για την υλοποίηση της διεπαφής υπηρεσιών επιλέξαμε να χρησιμοποιήσουμε Java. Οι αλγόριθμοι που γράψαμε για τις ανάγκες της εφαρμογής εκτελούν πολλές αριθμητικές πράξεις στα δεδομένα και η Java προσφέρει πολύ καλή επίδοση σε τέτοιου είδους επεξεργασία σε σύγκριση με τη JavaScript που ήταν η επόμενη επιλογή μας. Ταυτόχρονα, υπάρχουν πολλές βιβλιοθήκες που καθιστούν εύκολη την ανάπτυξη εφαρμογών διεπαφής υπηρεσιών ενώ το Αντικειμενοστραφές Μοντέλο προγραμματισμού ταιριάζει απόλυτα στη φιλοσοφία της εφαρμογής.

Από την άλλη, για την υλοποίηση του γραφικού περιβάλλοντος, επιλέξαμε να χρησιμοποιήσουμε HTML5, CSS3 και JavaScript πάνω σε ένα Node.JS εξυπηρετητή γραμμένο σε JavaScript. Ο εξυπηρετητής απλά προωθεί τα αιτήματα από το χρήστη στη διεπαφή υπηρεσιών και επιστρέφει τα αποτελέσματα. Η διεπαφή καθεαυτή υλοποιήθηκε με χρήση της βιβλιοθήκης AngularJS καθώς και πολλά άλλα εργαλεία που θα παρουσιαστούν στη συνέχεια.

Τέλος, ιδιαίτερη έμφαση δόθηκε στη μεθοδολογία ανάπτυξης και σχεδιασμού της εφαρμογής. Συγκεκριμένα ακολουθήσαμε την προσέγγιση ανάπτυξης με αφετηρία τα Τεστ (Test Driven Development). Επίσης, για την επιτάχυνση της διαδικασίας θέσαμε σε λειτουργία σύστημα αυτόματης εγκατάστασης της εφαρμογής (CI) στα περιβάλλοντα.

Λέξεις Κλειδιά:

γεωχωρικά δεδομένα, διεπαφή υπηρεσιών, γραφικό περιβάλλον, σημεία ενδιαφέροντος, φωτογραφίες, εκδηλώσεις, περιοχές ενδιαφέροντος, οδοί ενδιαφέροντος, Java, REST, Node.JS, JavaScript, AngularJS, HTML5, CSS3, CI, Test Driven Development

Η σελίδα αυτή είναι σκόπιμα λευκή.

Abstract

The amount of user-generated geospatial content on the Web is constantly increasing, making it a valuable source of information for enabling, enriching and enhancing geospatial applications and services. Using this information, however, is far from trivial and it basically requires implementing processing workflows that address two basic aspects: (a) collecting, cleaning and integrating the data, and (b) analyzing the collected information and presenting it to the users in a meaningful way, thus helping them to extract useful insights and conclusions.

This diploma thesis focuses mainly on the second part, namely, the implementation of a web application that, using existing data that has been collected and integrated from various Web sources, presents it to the user in a graphical and meaningful way. The geospatial data in our disposal come in three different types: Points of Interest, Photos and Events. Our goal is to enable the user to search for this data in certain locations and filter it using keywords or based on the data categorization. In addition, we want to enable the user to correlate different types of geospatial data based on their location and their similarity (i.e. Photos and Points of Interest). The most important feature of the application, though, is its ability to show the distribution of the data in a given area. Specifically, it can group the data in Regions of Interest or Streets of Interest. Furthermore, it generates short descriptions for these by selecting a small but representative set of results to display to the user.

The development of the system was split into two distinct parts. The first comprised a RESTful API that gives access to the data and executes all the algorithms that perform the filtering, clustering and correlation of the data; the second, comprised a Web user interface that graphically displays the data coming from the aforementioned API.

For the implementation of the RESTful API we chose to use Java. This is because the algorithms that solve the problems at hand execute numerous arithmetic and floating point operations on the data, and Java offers a great performance for this type of operations, compared to JavaScript which was the second candidate. At the same time, there are numerous libraries implemented in Java that greatly help the implementation of RESTful APIs and the Object Oriented Paradigm fits perfectly to the data of this application.

On the other hand, for the implementation of the user interface we chose to use a Node.js server, hosting a plain HTML5, CSS3, JavaScript Web application. The role of the Node.js server is to merely proxy the requests of the client to the REST API and serve the static files for the application. We implemented the UI using the AngularJS framework in addition to other state of the art tools that will be presented later on.

Finally, special attention was given to the design and development methodology used for implementing the system. In particular, we followed the Test Driven Development approach. Also, to improve the development cycle and guarantee continuous delivery, we put in place a continuous integration system that automatically deploys the parts of the application on the Virtual Machines.

Keywords:

geospatial data, API, user interface, points of interest, photos, events, areas of interest, streets of interest, Java, REST, Node.JS, JavaScript, AngularJS, HTML5, CSS3, CI, Test Driven Development

Η σελίδα αυτή είναι σκόπιμα λευκή.

Περιεχόμενα

1 Εισαγωγή 4

1.1 Γενικό πλαίσιο 4

1.2 Σκοπός και περιγραφή 5

1.3 Οργάνωση κειμένου 6

2 Επισκόπηση της αρχιτεκτονικής του συστήματος 8

2.1 Πηγές και επεξεργασία δεδομένων 9

2.2 Διεπαφή Υπηρεσιών 9

2.3 Διεπαφή Χρήστη / Γραφικό περιβάλλον 10

3 Δεδομένα 11

3.1 Προϋπάρχοντα Δεδομένα 11

3.1.1 Βάση Δεδομένων 11

3.1.2 Πλατφόρμα αναζήτησης Solr 13

3.1.3 Εξαγωγή και ενοποίηση γεωχωρικής πληροφορίας από πηγές του διαδικτύου 14

3.2 Δεδομένα που προστέθηκαν 16

3.2.1 Δομή των νέων δεδομένων 16

3.2.2 Διαδικασία εισαγωγής οδών 16

3.3 Εκτέλεση ερωτήσεων 20

3.3.1 Ερωτήσεις στη βάση δεδομένων 20

3.3.2 Παραδείγματα αναζητήσεων που πραγματοποιούμε με χρήση του Solr 23

3.3.3 Solr και Java 24

4 Αλγόριθμοι επεξεργασίας και ανάλυσης δεδομένων 25

4.1 Δομές δεδομένων 25

4.1.1 Point Grid 25

4.1.2 Vantage Point Tree (VP) 27

4.2 Ο αλγόριθμος ομαδοποίησης σημείων σε Περιοχές Ενδιαφέροντος 30

4.2.1 Περιγραφή του προβλήματος 30

4.2.2 Τρόπος λειτουργίας του αλγορίθμου 30

4.2.3 Βελτιστοποίηση του αλγορίθμου και πολυπλοκότητα 31

4.3 Ο αλγόριθμος εύρεσης Οδών Ενδιαφέροντος ή Γραφικών Οδών 32

4.3.1 Περιγραφή του προβλήματος 32

4.3.2 Τρόπος λειτουργίας του αλγορίθμου 33

4.3.3 Bresenham-based αλγόριθμος για τον υπολογισμό του Supercover Line ... 34

4.3.4 Συνολική πολυπλοκότητα εύρεσης οδών ενδιαφέροντος ... 35
4.4 Ο αλγόριθμος ποικίλων φωτογραφιών ... 36
4.4.1 Περιγραφή του προβλήματος ... 36
4.4.2 Ορισμός απόστασης ... 36
4.4.3 Τρόπος λειτουργίας του αλγορίθμου ... 38
4.4.4 Πολυπλοκότητα ... 39
4.5 Διασύνδεση Σημείων Ενδιαφέροντος με Φωτογραφίες και Εκδηλώσεις. 40
4.6 Συσχέτιση Περιοχών και Οδών με Σημειακές Οντότητες ... 41
4.6.1 Ο αλγόριθμος PNPOLY σε Java ... 41
4.6.2 Συσχέτιση Οδών με Σημεία Ενδιαφέροντος και Φωτογραφίες ... 42
5 Τεχνική περιγραφή της διεπαφής υπηρεσιών ... 43
5.1 Τεχνολογίες και εργαλεία ... 43
5.1.1 Maven για διαχείριση εξαρτήσεων και "χτίσιμο" της εφαρμογής ... 43
5.1.2 Spring και SpringMVC για την υλοποίηση της REST διεπαφής υπηρεσιών. 44
5.1.3 TestNG και Test Driven Development ... 48
5.1.4 Swagger για την αυτόματη περιγραφή της διεπαφής ... 49
5.1.5 Χρήση Git για την αποθήκευση του κώδικα και τη διατήρηση του ιστορικού αλλαγών ... 52
5.1.6 Χρήση Jenkins για την αυτόματη εγκατάσταση της εφαρμογής ... 53
5.2 Εγκατάσταση και παραμετροποίηση της διεπαφής υπηρεσιών ... 54
5.2.1 Διαπιστευτήρια σύνδεσης στη Βάση Δεδομένων και το Solr ... 54
5.2.2 Προετοιμασία μηχανήματος εξυπηρετητή για την αυτόματη εγκατάσταση 55
5.2.3 Αυτόματη εγκατάσταση ... 56
6 Τεχνική περιγραφή του γραφικού περιβάλλοντος ... 57
6.1 Τεχνολογίες και εργαλεία ... 57
6.1.1 Node.js ... 58
6.1.2 NPM και Bower για τη διαχείριση εξαρτήσεων ... 58
6.1.3 Εισαγωγή εξαρτήσεων στον κώδικα σύμφωνα με το CommonJS ... 60
6.1.4 Test Driven Development χρησιμοποιώντας Karma, Mocha και Chai ... 61
6.1.5 LESS και CSS για το σχεδιασμό της διεπαφής χρήστη ... 63
6.1.6 Angular.js για την υλοποίηση του γραφικού περιβάλλοντος ... 64
6.1.7 Gulp για το χτίσιμο της διεπαφής ... 66
6.2 JavaScript στον Εξυπηρετητή ... 68

6.2.1 Μηχανή Απεικόνισης ... 68
6.2.2 Φάκελος στατικών αρχείων ... 69
6.2.3 Αντιστοίχηση του μονοπατιού ρίζας με την εισαγωγική σελίδα ... 69
6.2.4 Προώθηση των αιτημάτων κάτω από το μονοπάτι /api προς τη διεπαφή υπηρεσιών ... 69
6.2.5 Εκτέλεση του Εξυπηρετητή ... 70
6.3 Εγκατάσταση και παραμετροποίηση της εφαρμογής ... 71
6.3.1 Έναρξη της εφαρμογής με χρήση του Forever.js ... 71
6.3.2 Ορισμός του εξυπηρετητή web ως υπηρεσία συστήματος του Ubuntu ... 72
6.3.3 Αυτόματη εγκατάσταση του εξυπηρετητή web από το CI ... 73
6.3.4 Εκτέλεση υπηρεσίας συστήματος για τον εξυπηρετητή web για πρώτη φορά 74
7 Πειραματική αξιολόγηση ... 75
7.1 Αλγόριθμος εύρεσης Περιοχών Ενδιαφέροντος ... 75
7.2 Αλγόριθμος εύρεσης οδών ενδιαφέροντος ... 78
7.3 Αλγόριθμος Ποικίλων Φωτογραφιών ... 80
8 Περιγραφή της εφαρμογής ... 82
8.1 Περιγραφή της διεπαφής υπηρεσιών ... 82
8.1.1 Ο Πόρος «Εκτάσεις» ... 83
8.1.2 Ο Πόρος «Κατηγορίες» ... 83
8.1.3 Οι Πόροι «Σημεία Ενδιαφέροντος», «Φωτογραφίες» και «Εκδηλώσεις» ... 83
8.1.4 Ο πόρος «Περιοχές» ... 84
8.1.5 Ο πόρος «Οδοί» ... 85
8.2 Περιγραφή του γραφικού περιβάλλοντος ... 86
8.2.1 Περιγραφή δομής της σελίδας έναρξης της εφαρμογής ... 86
8.2.2 Εκτέλεση βασικής αναζήτησης για διάφορους τύπους ... 88
8.2.3 Διαφορετικοί τρόποι επιλογής Έκτασης/Τοποθεσίας Αναζήτησης ... 89
8.2.4 Παραμετροποίηση αναζήτησης ... 90
8.2.5 Περιγραφή της δομής της σελίδας έπειτα από αναζήτηση ... 91
8.2.6 Εφαρμογή φίλτρων στα αποτελέσματα ... 94
8.2.7 Παράθυρο πληροφοριών και ενέργειες ... 95
9 Επίλογος ... 99
10 Βιβλιογραφία ... 101

1

Εισαγωγή

1.1 Γενικό πλαίσιο

Τα τελευταία χρόνια, ο αριθμός των χρηστών του διαδικτύου αυξάνεται με ταχύτατους ρυθμούς, μετρώντας το 2014 περίπου 2.92 δισεκατομμύρια χρήστες (7% περισσότεροι σε σχέση με το 2013), ενώ υπολογίζεται ότι το 2015 ο αριθμός αυτός θα ξεπεράσει τα 3 δισεκατομμύρια. Η εκρηκτική αυτή ανάπτυξη του διαδικτύου συνοδεύεται, όπως είναι αναμενόμενο, από ανάλογη αύξηση της πληροφορίας που είναι διαθέσιμη ανά πάσα στιγμή στο διαδίκτυο.

Επιπλέον, η ευρεία χρήση κινητών τηλεφώνων και σύγχρονων φυλλομετρητών, σε συνδυασμό με τη μεγάλη επιτυχία εφαρμογών όπως το Foursquare, το Google Maps, το Open Street Maps ή το Wikimapia, έχουν οδηγήσει στην παραγωγή μεγάλου όγκου γεωχωρικής πληροφορίας. Πληροφορίας, δηλαδή, που πέρα από τα υπόλοιπα δεδομένα της (φωτογραφίες, κείμενο κλπ), συνοδεύεται από μία τοποθεσία. Μάλιστα, η τελευταία συνήθως λαμβάνεται αυτόματα με βάση την τρέχουσα τοποθεσία του χρήστη (για παράδειγμα στο Foursquare) αλλά μπορεί και να τίθεται χειροκίνητα (για παράδειγμα στο Open Street Maps).

Και ενώ ως σήμερα οι ανωτέρω εφαρμογές είναι ιδιαίτερα δημοφιλείς για την αναζήτηση μεμονωμένων οντοτήτων στο χώρο, καμία δε μελετά τις χωρικές οντότητες μιας περιοχής ως ένα σύνολο. Παράλληλα, οι χωρικές οντότητες που ανήκουν στα διαφορετικά συστήματα δε συσχετίζονται μεταξύ τους με αποτέλεσμα κανένα σύστημα να μην εκμεταλλεύεται το σύνολο της γεωχωρικής πληροφορίας που υπάρχει διαθέσιμη στο διαδίκτυο.

Συνεπώς, γίνεται εύκολα αντιληπτή η ανάγκη της συλλογής και της ενοποίησης αυτής της πληροφορίας σε ένα σύστημα που, προβαίνοντας σε κατάλληλη επεξεργασία της τελευταίας, θα μπορεί να αναλύει την κατανομή της, να την εξετάζει στο σύνολό της, αλλά και να ανιχνεύει συσχετίσεις ανάμεσα στις επιμέρους οντότητες που αυτή περιέχει.

1.2 Σκοπός και περιγραφή

Σκοπός της παρούσας διπλωματικής είναι η ανάπτυξη ενός συστήματος το οποίο θα εκμεταλλεύεται τη διαθέσιμη γεωχωρική πληροφορία που περιγράψαμε προηγουμένως και θα επιτρέπει στους χρήστες την εξαγωγή χρήσιμων συμπερασμάτων. Μέχρι σήμερα, οι πληροφορίες παρουσιάζονται από τις υπάρχουσες εφαρμογές ως ανεξάρτητες μονάδες. Καμία εφαρμογή δεν τις εξετάζει ως ένα σύνολο ώστε να παρέχει πληροφορίες για το χώρο στον οποίο ανήκουν. Αυτό είναι το αποτέλεσμα που θέλουμε να πετύχουμε μέσα από αυτή τη διπλωματική.

Ως αφετηρία για την υλοποίηση του συστήματός θα χρησιμοποιήσουμε υπάρχουσα ενοποιημένη πληροφορία που έχει ήδη εξαχθεί από διάφορες πηγές του διαδικτύου. Τα δεδομένα που έχουμε στη διάθεσή μας κατατάσσονται σε τρία είδη: (α) Σημεία Ενδιαφέροντος, (β) Φωτογραφίες και (γ) Εκδηλώσεις. Προέρχονται δε, από ένα σύνολο διαφορετικών πηγών όπως για παράδειγμα το Foursquare, το Wikimapia, το Google Places και το Panoramio.

Το σύστημα που θα υλοποιήσουμε, θέλουμε να παρέχει τις ακόλουθες δυνατότητες στους χρήστες του:

- Να μπορούν να αναζητήσουν την πληροφορία με βάση ένα σύνολο κριτηρίων όπως η τοποθεσία τους, μια λέξη κλειδί και η κατηγορία τους.
- Να μπορούν, εύκολα και γρήγορα, να συσχετίσουν διαφορετικές οντότητες μεταξύ τους, σύμφωνα με την τοποθεσία τους αλλά και με βάση το σημασιολογικό τους περιεχόμενο.
- Να μπορούν να εντοπίσουν περιοχές μέσα σε μια έκταση στο χάρτη με μεγάλη συγκέντρωση σε Σημεία Ενδιαφέροντος που πληρούν κάποια κριτήρια. Τις περιοχές αυτές τις ονομάζουμε Περιοχές Ενδιαφέροντος.
- Να μπορούν να εντοπίσουν οδούς μέσα σε μια έκταση στο χάρτη που περιέχουν πολλά Σημεία Ενδιαφέροντος με βάση ένα σύνολο κριτηρίων. Οι οδοί αυτές ονομάζονται Οδοί Ενδιαφέροντος.

- Να μπορούν να εντοπίσουν οδούς μέσα σε μια έκταση στο χάρτη που περιέχουν πολλές φωτογραφίες. Οι οδοί αυτές ονομάζονται Γραφικές Οδοί.
- Να μπορούν να εξάγουν συνοπτικές πληροφορίες για τις Περιοχές Ενδιαφέροντος, τις Οδούς Ενδιαφέροντος και τις Γραφικές Οδούς με βάση το περιεχόμενό τους.

Η εφαρμογή που αναπτύξαμε αποτελείται από δύο υποσυστήματα ώστε να διαχωρίσουμε καλύτερα την ευθύνη του καθενός και να μπορέσουμε να επιλέξουμε τις τεχνολογίες που ταιριάζουν καλύτερα σε κάθε ανάγκη. Συγκεκριμένα, αναπτύξαμε αφενός μία διεπαφή υπηρεσιών η οποία εκτελεί τους διάφορους αλγόριθμους στα δεδομένα και επιστρέφει τα αποτελέσματά τους και αφετέρου ένα γραφικό περιβάλλον το οποίο χρησιμοποιεί την έξοδο της διεπαφής υπηρεσιών και την παρουσιάζει στο χρήστη. Η διεπαφή υπηρεσιών υλοποιήθηκε σε Java και φιλοξενείται σε έναν εξυπηρετητή Tomcat, ενώ το γραφικό περιβάλλον υλοποιήθηκε σε JavaScript και φιλοξενείται από έναν εξυπηρετητή Node.js.

1.3 Οργάνωση κειμένου

Στο Κεφάλαιο 2 θα δοθεί μία επισκόπηση της εφαρμογής που σχεδιάσαμε. Συγκεκριμένα, θα δείξουμε τα επιμέρους τμήματα από τα οποία αποτελείται και θα περιγράψουμε συνοπτικά το καθένα από αυτά. Επιπλέον, θα δείξουμε πώς κάθε επιμέρους τμήμα του συστήματος συνδέεται με τα υπόλοιπα για τη δημιουργία ενός εύρυθμου συνόλου με καλά ορισμένες ευθύνες των μελών του.

Στο Κεφάλαιο 3 θα παρουσιάσουμε με λεπτομέρεια τις πηγές δεδομένων που χρησιμοποιεί η εφαρμογή μας για να αντλήσει την πληροφορία που χρειάζεται. Στο πρώτο κομμάτι του κεφαλαίου θα περιγράψουμε συνοπτικά την πληροφορία που προϋπήρχε της διπλωματικής και την οποία χρησιμοποιήσαμε ως είχε. Στο δεύτερο μισό του κεφαλαίου θα αναλύσουμε τα δεδομένα που προστέθηκαν στο σύστημα στα πλαίσια αυτής της εργασίας καθώς και θα δώσουμε λεπτομέρειες για την προέλευσή τους και τον τρόπο ενσωμάτωσής τους με την υπάρχουσα πληροφορία. Τέλος θα δούμε πως μπορούμε να εκτελέσουμε ερωτήματα για την άντληση της πληροφορίας αυτής από τις πηγές δεδομένων.

Στο Κεφάλαιο 4 θα αναλύσουμε με λεπτομέρεια τις αλγοριθμικές διαδικασίες που χρησιμοποιεί η εφαρμογή μας για την εξαγωγή συμπερασμάτων από τα δεδομένα. Θα δείξουμε πως υπολογίζονται οι Περιοχές Ενδιαφέροντος και οι Οδοί Ενδιαφέροντος αλλά και πως υπολογίζονται συνοπτικές πληροφορίες για αυτές. Επίσης θα δούμε πως αντιστοιχίζουμε οντότητες μεταξύ τους. Για κάθε αλγόριθμο ή δομή δεδομένων που παρουσιάζεται στο κεφάλαιο, θα αναλύσουμε και την πολυπλοκότητά της.

Στο Κεφάλαιο 5 θα παρουσιάσουμε τεχνικές λεπτομέρειες σχετικές με την υλοποίηση της διεπαφής υπηρεσιών. Συγκεκριμένα, θα περιγράψουμε τα εργαλεία και τις βιβλιοθήκες που χρησιμοποιήσαμε για την υλοποίησή της αλλά και πως χρησιμοποιήσαμε το κάθε ένα από αυτά. Τέλος θα δείξουμε πως εγκαθίσταται η εφαρμογή σε έναν εξυπηρετητή Tomcat αλλά και πως αυτοματοποιήσαμε την εγκατάσταση νέων εκδόσεων με τη χρήση κατάλληλων εργαλείων.

Στο Κεφάλαιο 6 περιγράφουμε τις τεχνικές λεπτομέρειες υλοποίησης του γραφικού περιβάλλοντος της εφαρμογής. Όμοια με το προηγούμενο κεφάλαιο, στο πρώτο τμήμα του κεφαλαίου, θα περιγράψουμε τις διάφορες τεχνολογίες και εργαλεία που χρησιμοποιήσαμε για την υλοποίησή της. Στη συνέχεια, θα αναφερθούμε σύντομα στον εξυπηρετητή Node.js και στο πως αυτός υλοποιήθηκε. Τέλος, θα δείξουμε πως μπορούμε να εγκαταστήσουμε αυτή την εφαρμογή σε ένα μηχάνημα ώστε να αρχίσει να εξυπηρετεί αιτήματα αλλά και πώς, όμοια με τη διεπαφή υπηρεσιών, αυτοματοποιήσαμε τη διαδικασία εγκατάστασης νέων εκδόσεων του γραφικού περιβάλλοντος έπειτα από κάθε αλλαγή του κώδικα.

Στο Κεφάλαιο 7 θα αξιολογήσουμε τους αλγόριθμους που περιγράφηκαν στο Κεφάλαιο 4 εκτελώντας ένα σύνολο από μετρήσεις. Θα δείξουμε επίσης, πως επαληθεύεται η πολυπλοκότητα που υπολογίσαμε θεωρητικά στο Κεφάλαιο 4 και θα εξηγήσουμε την επίδραση που είχαν στους χρόνους εκτέλεσης οι διάφορες βελτιστοποιήσεις των αλγορίθμων, όπως η χρήση χωρικών ευρετηρίων.

Στο Κεφάλαιο 8 θα περιγράψουμε αναλυτικότερα τις λειτουργικότητες που υλοποιήθηκαν στα πλαίσια της εφαρμογής, τόσο για τη διεπαφή υπηρεσιών όσο και για το γραφικό περιβάλλον. Ουσιαστικά το κεφάλαιο αυτό αποτελεί το εγχειρίδιο χρήσης της εφαρμογής μας. Στο πρώτο μισό του κεφαλαίου περιγράφεται η διεπαφή υπηρεσιών. Στο δεύτερο μισό παρουσιάζεται με χρήση εικόνων το γραφικό περιβάλλον και η λειτουργικότητες που παρέχει στους χρήστες του.

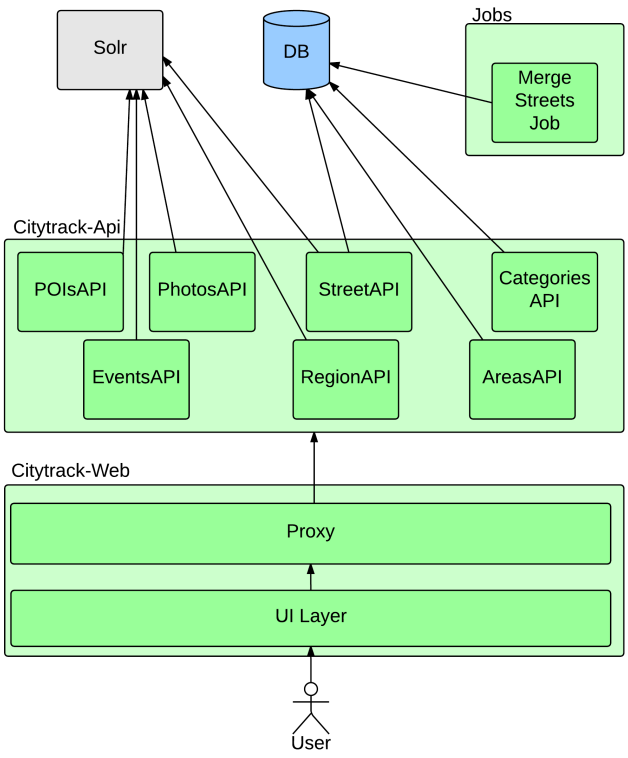
Τέλος, στο Κεφάλαιο 9 συνοψίζουμε τα αποτελέσματα της διπλωματικής εργασίας και περιγράφουμε πιθανές επεκτάσεις της ενώ στο Κεφάλαιο 10 παρατίθενται οι αναφορές που χρησιμοποιήθηκαν κατά την ανάπτυξη της εφαρμογής και τη συγγραφή της διπλωματικής.

2

Επισκόπηση της αρχιτεκτονικής του συστήματος

Η εφαρμογή που σχεδιάσαμε, όπως έχει αναφερθεί ήδη σε προηγούμενο κεφάλαιο, αποτελείται από πολλά επιμέρους υποσυστήματα τα οποία διασυνδέονται με κατάλληλο τρόπο.

Τα επιμέρους τμήματα από τα οποία αποτελείται η εφαρμογή μας παρουσιάζονται συνοπτικά στο ακόλουθο σχεδιάγραμμα:



Εικόνα 2.1 Σχηματική αναπαράσταση αρχιτεκτονικής του συστήματος

Συγκεκριμένα, η εφαρμογή μας χωρίζεται στα εξής τμήματα:

1. Τις πηγές δεδομένων (Solr και PostgreSQL) και τις διεργασίες για την εισαγωγή νέων δεδομένων.
2. Τη διεπαφή υπηρεσιών «Citytrack-API» που προσφέρει πρόσβαση στα δεδομένα.
3. Τη γραφική διεπαφή «Citytrack-Web» μέσα από την οποία οι χρήστες αποκτούν πρόσβαση στην εφαρμογή.

Στο σχεδιάγραμμα, ο χρωματικός κώδικας ακολουθεί τις εξής συμβάσεις:

- Με πράσινο, επισημαίνουμε όλα τα νέα τμήματα που προστέθηκαν στην εφαρμογή στα πλαίσια της διπλωματικής
- Με μπλε, επισημαίνουμε τα τμήματα που υπήρχαν ήδη, αλλά τα αλλάξαμε ή τους προσθέσαμε επιπλέον λειτουργικότητα/δεδομένα
- Με γκρι, παρουσιάζονται όλα τα μέρη που προϋπήρχαν της διπλωματικής και δεν τα αλλάξαμε με κάποιο τρόπο

2.1 Πηγές και επεξεργασία δεδομένων

Τα δεδομένα που χρησιμοποιεί η εφαρμογή είναι αποθηκευμένα σε δύο πηγές: (α) μία αντικειμενοστραφή/σχεσιακή βάση δεδομένων (PostgreSQL) και (β) μία πλατφόρμα ευρετηριοποίησης και αναζήτησης (Solr). Και οι δύο θα αναλυθούν λεπτομερώς στο κεφάλαιο 3. Στο ίδιο κεφάλαιο, θα περιγράψουμε και τη διαδικασία εισαγωγής και επεξεργασίας νέων δεδομένων για τις Οδούς στη βάση δεδομένων.

2.2 Διεπαφή Υπηρεσιών

Η διεπαφή υπηρεσιών της εφαρμογής είναι το κομμάτι που είναι υπεύθυνο για την άντληση των δεδομένων από τις πηγές δεδομένων, την επεξεργασία και ανάλυσή τους με τη χρήση διάφορων αλγορίθμων που θα περιγράψουμε σε επόμενο κεφάλαιο και, τέλος, την προώθηση των αποτελεσμάτων στο χρήστη.

Η διεπαφή υπηρεσιών υλοποιήθηκε σε Java με τη βοήθεια μιας πληθώρας βιβλιοθηκών που θα αναφερθούν με περισσότερη λεπτομέρεια στο κεφάλαιο 5.

Περισσότερες πληροφορίες για τη δομή της διεπαφής μπορούν να βρεθούν στην ενότητα 8.1

2.3 Διεπαφή Χρήστη / Γραφικό περιβάλλον

Το γραφικό περιβάλλον είναι το σημείο με το οποίο ο χρήστης έρχεται σε επαφή με την εφαρμογή μας και κατ' επέκταση με όλο το σύστημα που αναπτύξαμε.

Η διεπαφή χρήστη ουσιαστικά αποτελείται από δύο επιμέρους τμήματα. Αφενός, από τον κώδικα ο οποίος εκτελείται στον περιηγητή του χρήστη και αφετέρου από τον εξυπηρετητή ο οποίος «σερβίρει» τον κώδικα αυτό στο χρήστη και προωθεί τα αιτήματα που πηγάζουν από τον περιηγητή στη διεπαφή υπηρεσιών, ώστε να εξυπηρετηθούν.

Η ανάπτυξη του κώδικα που εκτελείται στον περιηγητή (Web Layer) έγινε χρησιμοποιώντας τις τεχνολογίες JavaScript, HTML5 και CSS3 καθώς και μία σωρεία από εργαλεία και βιβλιοθήκες γραμμένα σε JavaScript.

Ο κώδικας που υλοποιεί τον εξυπηρετητή του γραφικού περιβάλλοντος είναι επίσης γραμμένος σε JavaScript. Συγκεκριμένα, πρόκειται για έναν εξυπηρετητή Node.js με λειτουργικότητα που περιορίζεται όπως αναφέραμε στο «σερβίρισμα» των αρχείων και την προώθηση των αιτημάτων που πηγάζουν από τους περιηγητές των χρηστών προς τη διεπαφή υπηρεσιών.

Περισσότερες λεπτομέρειες σχετικά με την υλοποίηση του γραφικού περιβάλλοντος τόσο σε σχεδιαστικό όσο και σε τεχνικό επίπεδο μπορούν να βρεθούν στο κεφάλαιο 4. Επιπλέον, περισσότερες πληροφορίες σχετικά με τη σχεδίαση, τη χρηστικότητα και τις λειτουργίες της διεπαφής χρήστη μπορούν να αναζητηθούν στο κεφάλαιο 8.2.

3

Δεδομένα

Στο κεφάλαιο αυτό, αναλύουμε τις πηγές από τις οποίες αντλούμε τα δεδομένα για τις ανάγκες της εφαρμογής. Αρχικά, θα περιγράψουμε τα δεδομένα που προϋπήρχαν της διπλωματικής, τη δομή τους, και συνοπτικά, τον τρόπο συλλογής τους. Στη συνέχεια, θα αναφερθούμε στα δεδομένα που προσθέσαμε για τις ανάγκες της εφαρμογής. Τέλος, θα δούμε πως μπορούμε να εκτελέσουμε ερωτήσεις στα διάφορα συστήματα που περιέχουν τα δεδομένα μας.

3.1 Προϋπάρχοντα Δεδομένα

3.1.1 Βάση Δεδομένων

Η βάση δεδομένων ήταν ήδη υλοποιημένη στο Σύστημα Διαχείρισης Βάσεων Δεδομένων (ΣΔΒΔ) PostgreSQL (PostgreSQL 2015), με την επέκταση PostGIS (PostGIS 2015). Συγκεκριμένα, η PostgreSQL είναι ένα αντικειμενοστραφές-σχεσιακό ΣΔΒΔ. Είναι ένα από τα πιο δημοφιλή και αποδοτικά σχεσιακά συστήματα ανοικτού λογισμικού (Open Source) με κοινότητα που μεγαλώνει συνεχώς. Η επέκταση PostGIS προσθέτει στο ΣΔΒΔ χωρικές δομές δεδομένων, ευρετήρια και συναρτήσεις για την αποθήκευση, επεξεργασία και ερώτηση χωρικών δεδομένων, όπως για παράδειγμα σημεία, γραμμές, πολύγωνα κλπ.

Το μεγαλύτερο τμήμα της βάσης, όπως ήδη αναφέραμε, υλοποιήθηκε έξω από τα πλαίσια αυτής της διπλωματικής εργασίας, παρόλα αυτά θα περιγράψουμε για λόγους πληρότητας, τη δομή των βασικών οντοτήτων που χρησιμοποιούμε.

Οι ενδιαφέρουσες δομές δεδομένων που προϋπήρχαν στη βάση δεδομένων παρουσιάζονται στον ακόλουθο πίνακα:

Περιγραφή	**Όνομα Πίνακα**
Σημείο Ενδιαφέροντος	venues
Φωτογραφία	photos
Εκδήλωση	events
Έκταση	areas
Κατηγορία	categories
Αντιστοιχίσεις Κατηγοριών	category_mappings

Οι οντότητες «Σημείο Ενδιαφέροντος», «Φωτογραφία» και «Εκδήλωση» έχουν παρόμοια δομή. Οι πληροφορίες των οντοτήτων αυτών που μας ενδιαφέρουν είναι οι ακόλουθες:

Ιδιότητα	**Όνομα στη Βάση**	**Περιγραφή**
Χαρακτηριστικό (ID)	id	Τιμή που ορίζει μονοσήμαντα την οντότητα
Λεζάντα	label	Συνοπτικός τίτλος όπως αυτός εξάγεται από τις διάφορες πηγές
Περιγραφή	events	Αναλυτικότερη περιγραφή της οντότητας (αν υπάρχει).
Πηγή	areas	Η προέλευση της οντότητας (πχ Foursquare)
Συντεταγμένες	coordinates	Το σημείο στο χάρτη σε μορφή <longitude,latitude> στο οποίο βρέθηκε η οντότητα
Ετικέτες (Μόνο για την Φωτογραφία)	tags	Σύνολο από ετικέτες που περιγράφουν τη Φωτογραφία

Πίνακας 3.1 Το Βασικές ιδιότητες των οντοτήτων Σημείο Ενδιαφέροντος, Φωτογραφία και Εκδήλωση

Η οντότητα «Έκταση» φέρει τις εξής πληροφορίες:

Ιδιότητα	**Όνομα στη Βάση**	**Περιγραφή**
Χαρακτηριστικό (ID)	id	Τιμή που ορίζει μονοσήμαντα την περιοχή
Όνομα	name	Το όνομα της περιοχής (πχ. Αθήνα)
Ορθογώνιο	bbox	Ορθογώνια οντότητα που ορίζει χωρικά η περιοχή αυτή
Κέντρο	center	Το σημείο που αποτελεί το κέντρο της περιοχή (όχι απαραίτητα το κέντρο της προηγούμενης περιοχής

Πίνακας 3.2 Βασικές ιδιότητες της οντότητας Περιοχή

Τέλος, οι οντότητες «Κατηγορία» και «Αντιστοιχίσεις Κατηγοριών» περιέχουν τις κατηγορίες που λαμβάνονται από τις διάφορες πηγές δεδομένων και την αντιστοίχισή τους με τις εσωτερικές κατηγορίες (βλ Ενότητα 3.1.3.2). Η μόνη πληροφορία που χρησιμοποιεί η εφαρμογή μας βρίσκεται στην οντότητα «Αντιστοιχίσεις Κατηγοριών» και είναι η ακόλουθη:

Ιδιότητα	Όνομα στη Βάση	Περιγραφή
Κατηγορία Πρώτου Επιπέδου	gs_cat_1	Το όνομα της κατηγορίας πρώτου επιπέδου (πχ Food, Services κλπ)

Πίνακας 3.3 Βασική Ιδιότητα της οντότητας Αντιστοίχιση Κατηγοριών

3.1.2 Πλατφόρμα αναζήτησης Solr

Το Solr (Solr 2015) είναι μια πλατφόρμα αναζήτησης που υποστηρίζεται από την κοινότητα ανοικτού λογισμικού (Open Source) και χαρακτηρίζεται από υψηλή ταχύτητα εκτέλεσης αναζητήσεων. Επιπλέον, χαρακτηρίζεται από υψηλή αξιοπιστία και επεκτασιμότητα και χρήζει ιδιαίτερης αναγνώρισης στην κοινότητα ανάπτυξης διαδικτυακών εφαρμογών με ανάγκες αναζήτησης.

Κάθε οντότητα στο Solr είναι ένα «έγγραφο» (document όπως αναφέρεται στην ορολογία του Solr). Κάθε «έγγραφο», αποτελείται από ένα ή περισσότερα πεδία που περιέχουν πληροφορίες γι' αυτό και μπορούν να είναι διαφόρων τύπων, όπως για παράδειγμα κείμενο, αριθμοί, ημερομηνίες ή ακόμη και γεωχωρικές οντότητες. Όλα τα «έγγραφα» ανήκουν σε μία «συλλογή» (collection κατά την ορολογία του Solr) και κάθε «συλλογή» χαρακτηρίζεται από ένα «σχήμα» που ορίζει τη δομή των «εγγράφων» που περιέχονται σε αυτή.

Ορίζοντας ένα «σχήμα», ουσιαστικά περιγράφουμε στο Solr, πέρα από τη δομή των εγγράφων, και το πως πρέπει να διαχειρίζεται τα «έγγραφα» αυτά και τα πεδία τους. Η μεγάλη δύναμη του Solr, εντοπίζεται στις δυνατότητες που έχει να ευρετηριοποιεί αποδοτικά τα δεδομένα των πεδίων ανάλογα με τον τύπο τους και στη συνέχεια να εκτελεί ταχύτατες αναζητήσεις στα πεδία αυτά. Χρησιμοποιώντας το «σχήμα» μπορούμε να ορίσουμε ποια πεδία χρησιμοποιούνται για αναζήτηση και πρέπει επομένως να ευρετηριοποιηθούν.

Μάλιστα, το Solr μπορεί να εκτελεί ταχύτατες αναζητήσεις σε περισσότερα του ενός πεδία ταυτόχρονα για κάθε έγγραφο, με περισσότερα του ενός κριτήρια και στη συνέχεια να ενοποιεί τα δεδομένα. Τέλος μπορεί να ταξινομεί τα αποτελέσματα είτε με βάση την επιθυμία του χρήστη ή υπολογίζοντας ένα σκορ που εκτιμά την σχετικότητα μιας οντότητας με την αναζήτηση που πραγματοποίησε ο χρήστης.

Οι οντότητες που υπάρχουν ως έγγραφα στην πλατφόρμα Solr του συστήματός μας είναι οι ακόλουθες:

- Σημεία Ενδιαφέροντος (POIs)
- Φωτογραφίες (Photos)
- Εκδηλώσεις (Events)

Όλες οι οντότητες εισάγονται ως «έγγραφα» που ακολουθούν το ίδιο «σχήμα» και διαφέρουν μόνο ως προς το πεδίο «τύπος» (type). Συγκεκριμένα, ο τύπος για τα Σημεία Ενδιαφέροντος είναι «poi», για τις Φωτογραφίες είναι «photo» και τέλος για τις Εκδηλώσεις είναι «event».

3.1.3 Εξαγωγή και ενοποίηση γεωχωρικής πληροφορίας από πηγές του διαδικτύου

Σε αυτήν την υπό-ενότητα θα δούμε συνοπτικά τη διαδικασία με την οποία δημιουργήθηκαν τα δεδομένα που προϋπήρχαν στη βάση δεδομένων (Lamprianidis, et al. 2014). Θα δούμε, δηλαδή, πως συλλέγεται η γεωχωρική πληροφορία από διάφορες πηγές αλλά και πως ενοποιείται στη βάση δεδομένων.

Ουσιαστικά η διαδικασία εξαγωγής και ενοποίησης της εν λόγω πληροφορίας χωρίζεται στα εξής τμήματα:

1. Συλλογή Σημείων Ενδιαφέροντος, Φωτογραφιών και Εκδηλώσεων από πηγές του διαδικτύου συμπεριλαμβανομένων των DBpedia, OpenStreetMap, Wikimapia, Google Places, Foursquare, και Eventful.
2. Ταξινόμηση της συλλεγμένης πληροφορίας με βάση μια κοινή δομή κατηγοριών, και υπολογισμός αντιστοιχίσεων των κατηγοριών από κάθε πηγή σε μία από τις εσωτερικές κατηγορίες του συστήματος μας.
3. Εντοπισμός και ενοποίηση διπλοτύπων, οντοτήτων δηλαδή που μπορεί να προέρχονται από την ίδια ή διαφορετικές πηγές και αναφέρονται στην ίδια λογική οντότητα.

3.1.3.1 Συλλογή Σημείων Ενδιαφέροντος, Φωτογραφιών και Εκδηλώσεων

Η μεγαλύτερη πρόκληση σε αυτό το κομμάτι είναι η υλοποίηση ενός συνόλου «Εξορυκτών Δεδομένων», ένα για κάθε πηγή από την οποία ήταν απαραίτητη η άντληση δεδομένων. Τα δεδομένα που συλλέγονται με αυτή τη διαδικασία από όλες τις πηγές και αποθηκεύονται σε μία χωρική βάση δεδομένων PostgreSQL (PostgreSQL 2015) με ομοιόμορφη δομή ώστε να γίνει ευκολότερη η μετέπειτα επεξεργασία τους.

3.1.3.2 Κατηγοριοποίηση της πληροφορίας με βάση μια κοινή δομή κατηγοριών

Ένα μεγάλο μέρος της επεξεργασίας κυρίως των σημείων ενδιαφέροντος είναι η κατηγοριοποίηση τους με βάση ένα προαποφασισμένο δέντρο κατηγοριών. Κάθε πηγή από

την οποία συλλέγουμε δεδομένα ακολουθεί και μια διαφορετική κατηγοριοποίηση. Για να μπορούν λοιπόν τα δεδομένα μας να είναι συγκρίσιμα ήταν απαραίτητη η εκ νέου κατηγοριοποίησή τους.

Έτσι, κάθε κατηγορία από κάποια εξωτερική πηγή αντιστοιχίζεται με μια από τις κατηγορίες που ορίστηκαν για τις ανάγκες της εφαρμογής μας. Μάλιστα, η κατηγοριοποίηση αυτή μπορεί να γίνει τόσο χειροκίνητα όσο και αλγοριθμικά, αφού το σύστημα είναι σε θέση να προτείνει αντιστοιχίσεις ανάμεσα σε «εξωτερικές» κατηγορίες και «εσωτερικές» κατηγορίες με βάση την ομοιότητά τους.

Αποτέλεσμα αυτού του βήματος είναι ότι όλα τα δεδομένα που συλλέγονται καταλήγουν να εντάσσονται σε μια από τις εσωτερικές κατηγορίες του συστήματός μας.

3.1.3.3 Εντοπισμός και ενοποίηση διπλοτύπων

Όπως αναφέραμε συνοπτικά σε προηγούμενη ενότητα, επειδή οι οντότητες εισάγονται από πολλές διαφορετικές πηγές του διαδικτύου, είναι φυσικό πολλές από αυτές να αναφέρονται στο ίδιο Σημείο Ενδιαφέροντος. Για παράδειγμα, εισάγοντας δεδομένα για την Ακρόπολη βρίσκονται Σημεία Ενδιαφέροντος για τον Παρθενώνα σε πολλές από της πηγές δεδομένων (Google, Wikimapia, κλπ)

Για να αποφευχθεί λοιπόν η ύπαρξη τέτοιον διπλοτύπων ήταν απαραίτητη η υλοποίηση αλγορίθμων που τα εντοπίζουν χρησιμοποιώντας χωρικά και σημασιολογικά κριτήρια και να ενοποιούν την πληροφορία τους σε μία μοναδική υπέρ-ενότητα.

Ο αλγόριθμος προσπαθεί να εντοπίσει οντότητες που πιθανότατα αναφέρονται στην ίδιο Σημείο Ενδιαφέροντος και τις παρουσιάζει με γραφικό τρόπο στο διαχειριστή του συστήματος. Ο διαχειριστής μπορεί στη συνέχεια να αποφασίσει αν η αντιστοίχιση ήταν ορθή ή όχι. Στην περίπτωση που αποφασίσει ότι η αντιστοίχιση ήταν σωστή, οι οντότητες ενοποιούνται αυτόματα.

3.2 Δεδομένα που προστέθηκαν

3.2.1 Δομή των νέων δεδομένων

Η προσθήκη που κάναμε στη βάση δεδομένων στα πλαίσια αυτής της διπλωματικής εργασίας ήταν οι οδοί για διάφορες πόλεις, ώστε να καταστεί εφικτή η εξαγωγή συμπερασμάτων σχετικά με τις Οδούς Ενδιαφέροντος ή τις Γραφικές Οδούς.

Η οντότητα Οδός ορίζει τις ακόλουθες πληροφορίες:

Ιδιότητα	**Όνομα στη Βάση**	**Περιγραφή**
Χαρακτηριστικό (ID)	id	Τιμή που ορίζει μονοσήμαντα την οδό
Όνομα	street_name	Το όνομα της οδού
Μήκος	street_length	Ορθογώνια οντότητα που ορίζει χωρικά η περιοχή αυτή
Τύπος Οδού	street_type	Το είδος της οδού, π.χ.: «pedestrian», «highway» κλπ
Γραμμή Οδού	street_line	Το σημείο που αποτελεί το κέντρο της περιοχή (όχι απαραίτητα το κέντρο της προηγούμενης περιοχής

Πίνακας 3.4 Ιδιότητες της οντότητας Οδός

3.2.2 Διαδικασία εισαγωγής οδών

Για την υλοποίηση των Οδών Ενδιαφέροντος είναι απαραίτητο να έχουμε δεδομένα για τις Οδούς που υπάρχουν σε μία πόλη.

Ο πλέον εύκολος τρόπος να εξασφαλίσουμε την οδική πληροφορία για μία περιοχή είναι να την «κατεβάσουμε» από το OpenStreetMap (OpenStreetMap 2015) σε ένα σύνολο αρχείων και στη συνέχεια να την εισάγουμε στη βάση δεδομένων.

3.2.2.1 Εξαγωγή πληροφορίας Οδών από το OpenStreetMap

Το OpenStreetMap είναι μία πρωτοβουλία για τη συγκέντρωση χωρικών δεδομένων ανά τον κόσμο και την ανοιχτή διάθεσή τους μέσα από ένα σύνολο ανοικτών διεπαφών υπηρεσιών. Για τη συγκέντρωση χωρικών δεδομένων βασίζεται στην κοινότητα των χρηστών του, οι οποίοι, μέσα από μια σειρά από εργαλεία, μπορούν να εισάγουν πληροφορίες οι οποίες επικυρώνονται και ενσωματώνονται στο σύνολο της πληροφορίας που διατίθεται μέσα από το OpenStreetMap.

Τα δεδομένα του OpenStreetMap διατίθενται μέσα από ένα σύνολο από σημεία πρόσβασης κάθε ένα από τα οποία εξυπηρετεί διαφορετικές ανάγκες. Στην περίπτωσή μας, χρειαζόμαστε δεδομένα για αρκετά μεγάλες περιοχές καθιστώντας ιδανική την περίπτωση του σημείου πρόσβασης «OSM Extended API» (xAPI κατά συντομογραφία).

Το xAPI προσφέρει πρόσβαση στα δεδομένα με διαφορετικούς τρόπους. Στην ενότητα αυτή θα παρουσιάσουμε μόνο εκείνον που χρησιμοποιήσαμε για την εξαγωγή όλων των οδών.

Για παράδειγμα, για την εξαγωγή όλων των οδών για την περιοχή της Αθήνας χρησιμοποιούμε το ακόλουθο URL:

```
http://open.mapquestapi.com/xapi/api/0.6/way[highway=*][bbox=23.5,37.8,24,38.2]
```

Ουσιαστικά ζητούμε την εξαγωγή όλων των οντοτήτων τύπου «way» οι οποίες έχουν ένα πεδίο που ονομάζεται «highway» και εντοπίζονται στην ορθογώνια περιοχή που ορίζεται από τα σημεία $P_{southWest}(23.4, 37.8)$ και $P_{northEast}(23.4, 37.8)$

Αντίστοιχα για την περιοχή του Βερολίνου το αντίστοιχο URL είναι

```
http://open.mapquestapi.com/xapi/api/0.6/way[highway=*][bbox=13.09,52.34,13.76,52.68]
```

Το μόνο δηλαδή που αλλάζει είναι οι συντεταγμένες της περιοχής που τίθενται στην παράμετρο bbox.

Κάθε ένα από τα URL αυτά επιστρέφει ένα XML αρχείο με κατάληξη *.osm το οποίο περιέχει όλες τις οντότητες που πληρούν τα κριτήρια με βάση τα οποία πραγματοποιήσαμε την αναζήτηση.

3.2.2.2 Εισαγωγή των δεδομένων για τις Οδούς στη βάση

Το επόμενο βήμα μετά την απόκτηση της πληροφορίας των Οδών ως ένα σύνολο «osm» αρχείων, είναι να την εισάγουμε στη βάση δεδομένων μας.

Ο γρηγορότερος τρόπος να το πετύχουμε αυτό είναι κάνοντας χρήση του εργαλείου «osm2pgsql». Το «osm2pgsql» είναι ένα εργαλείο γραμμής εντολών, που κάνει ακριβώς αυτό που λέει το όνομά του. Εισάγει δηλαδή την γεωχωρική πληροφορία που υπάρχει σε ένα αρχείο τύπου «osm» σε μία βάση δεδομένων PostgreSQL.

Στη συνέχεια παραθέτουμε την εντολή που εισάγει όλα τα δεδομένα που περιέχονται στο αρχείο «file.osm» σε μία βάση δεδομένων και επεξηγούμε όλες τις παραμέτρους:

```
osm2pgsql --append --latlong --cache=2000 --database=DBNAME
          --username=USERNAME file.osm --host=HOST
```

Όπου:

- DBNAME: είναι το όνομα της βάσης δεδομένων
- USERNAME: το όνομα χρήστη που έχει πρόσβαση στη βάση
- HOST: Η διεύθυνση του εξυπηρετητή στον οποίο είναι εγκατεστημένη η βάση
- --cache=2000: Ορίζει το μέγεθος της διαθέσιμης RAM μνήμης σε Mb και χρησιμοποιείται για την επιτάχυνση της διαδικασίας εισαγωγής
- --append: Δηλώνει στο πρόγραμμα ότι οι απαιτούμενες δομές είναι ήδη στη βάση δεδομένων και ενδεχομένως περιέχουν δεδομένα που δε θέλουμε να σβηστούν.
- --latlong: Δηλώνει στο πρόγραμμα τον τρόπο με τον οποίο επιθυμούμε να εισαχθεί η χωρική πληροφορία. Εδώ ζητάμε να εισαχθεί ως με βάση το γεωγραφικό πλάτος και μήκος της, ώστε να συμβαδίζει με τη δομή όλων των υπόλοιπων οντοτήτων.

Η παραπάνω εντολή πρέπει να εκτελεστεί για κάθε αρχείο *.osm που θέλουμε να εισάγουμε στη βάση δεδομένων.

Τέλος, να σημειωθεί ότι την πρώτη φορά που θα εκτελέσουμε την ανωτέρω εντολή στη βάση δεδομένων, πρέπει να παραλείψουμε την παράμετρο --append για να επιτρέψουμε στο πρόγραμμα να δημιουργήσει τις δομές που χρειάζεται ώστε να λειτουργήσει σωστά.

3.2.2.3 Επεξεργασία των οδών και αποθήκευσή τους σε κατάλληλη δομή

Τα δεδομένα που εισάγονται στη βάση από το εργαλείο osm2psql, περιέχουν αρκετή περιττή πληροφορία. Επίσης, οι Οδοί εισάγονται κατά τμήματα και όχι ως μία ενιαία γραμμή από την αρχή ως το τέλος της οδού.

Για να «καθαρίσουμε» λοιπόν τα εν λόγω δεδομένα, δημιουργήσαμε μία διεργασία που εκτελείται αφού έχουμε εισάγει τις οδούς στη βάση δεδομένων, και ενοποιεί τα διάφορα τμήματα κάθε οδού σε ένα. Στη συνέχεια, το αποθηκεύει μαζί με το συνολικό μήκος του σε ένα νέο πίνακα με όνομα «street». Η δομή του πίνακα περιγράφηκε στην ενότητα 3.2.1.

Έστω Π η περιοχή για την οποία θέλουμε να «καθαρίσουμε» τα δεδομένα που εισήχθηκαν από το osm2psql και να τα μεταφέρουμε στον πίνακα «street». Η διαδικασία ενοποίησης έχει ως εξής.

Αρχικά, ο αλγόριθμος φορτώνει στη μνήμη όλα τα τμήματα οδών που υπάρχουν στους ενδιάμεσους πίνακες που δημιουργεί το osm2psql. Κάθε τμήμα οδού αποθηκεύεται σε μία λίστα που εντοπίζεται σε ένα πίνακα κατακερματισμού (HashTable) με βάση το όνομα του τμήματος. Σκοπός μας είναι όλες οι Οδοί με το ίδιο όνομα να αποθηκευθούν στην ίδια λίστα. Οδοί χωρίς όνομα απλά αγνοούνται.

Στη συνέχεια, ο αλγόριθμος περνά από κάθε λίστα που αποθηκεύτηκε στον πίνακα κατακερματισμού και προσπαθεί να κατασκευάσει όλες τις οδούς που αποτελούνται από τα επιμέρους αυτά τμήματα. Προφανώς, σε κάθε λίστα μπορεί να αντιστοιχούν περισσότερες από μία Οδοί, αφού σε μια περιοχή Π υπάρχουν πιθανότατα περισσότερες από μία οδοί με το ίδιο όνομα. Ο τρόπος με τον οποίο προσπαθεί να ενώσει τα τμήματα αυτά, είναι εξετάζοντας ποια από αυτά αρχίζουν πολύ κοντά από εκεί που τελειώνει κάποιο άλλο. Κάθε φορά που βρίσκει μια τέτοια αντιστοίχιση ενώνει τα δύο τμήματα σε ένα, τα αφαιρεί από τη λίστα, και προσθέτει πίσω στη λίστα το αποτέλεσμα της ένωσής τους, καθώς μπορεί να υπάρχουν και άλλες αντιστοιχίσεις με το νέο τμήμα. Όταν περάσει από όλα τα τμήματα μιας λίστας, αποθηκεύει τα αποτελέσματα στον πίνακα «street».

```
result ← {};
while (candidates.hasMore()):
   candidate ← poll(candidates)
   performedMerge = mergeConsequentParts(candidate, candidates);
   if (performedMerge && candidates.hasMore()) {
      candidates.addFirst(candidate);
   } else {
      result.add(candidate);
   }
}
return result;
```

Κώδικας 3.1 Ψευδοκώδικας εξαγωγής ενοποιημένων οδών από οδικά τμήματα

3.3 Εκτέλεση ερωτήσεων

3.3.1 Ερωτήσεις στη βάση δεδομένων

Στην ενότητα αυτή θα δούμε τις βασικές ερωτήσεις που εκτελούμε στη βάση δεδομένων μέσω της εφαρμογής μας. Όπως αναφέραμε συνοπτικά στην εισαγωγή αυτού του κεφαλαίου, οι μόνες περιπτώσεις στις οποίες εκτελούμε ερωτήσεις απευθείας στη βάση δεδομένων είναι για την εύρεση των διαθέσιμων περιοχών, των διαθέσιμων κατηγοριών και τέλος των οδών που περνούν από μία ορθογώνια περιοχή.

Αρχικά, για τη σύνδεση στη βάση δεδομένων από την εφαρμογή μας χρησιμοποιούμε τον «οδηγό» που υλοποιεί τη διεπαφή JDBC για τη βάση PostgreSQL. Η διεπαφή JDBC ορίζει ένα σύνολο από κλάσεις, διεπαφές και εντολές, με τη χρήση των οποίων μπορεί κανείς να συνδεθεί και να εκτελέσει ερωτήσεις σε σχεσιακά ΣΔΒΔ. Ο «οδηγός» για την PostgreSQL υλοποιεί τη διεπαφή που ορίζει το JDBC συγκεκριμένα για μία βάση δεδομένων PostgreSQL.

3.3.1.1 Εύρεση διαθέσιμων περιοχών

Το SQL ερώτημα για την εύρεση όλων των διαθέσιμων περιοχών φαίνεται στη συνέχεια:

```
SELECT * FROM areas
```

Κώδικας 3.2 Ερώτημα για την εύρεση όλων των περιοχών

3.3.1.2 Εύρεση όλων των κατηγοριών πρώτου επιπέδου

Παρομοίως, απλή είναι και η ερώτηση για την εύρεση όλων των κατηγοριών πρώτου επιπέδου που έχουν οριστεί στο σύστημα. Το εν λόγω ερώτημα φαίνεται στη συνέχεια:

```
SELECT DISTINCT gs_cat_1
FROM
    category_mappings
WHERE
    gs_cat_1 IS NOT NULL AND
    gs_cat_1 <> '';
```

Κώδικας 3.3 Ερώτημα για την εύρεση των κατηγοριών πρώτου επιπέδου

Όπως βλέπουμε στο εν λόγω ερώτημα αποκλείουμε τις μη έγκυρες κατηγορίες. Εκείνες δηλαδή που είτε δεν περιέχουν κείμενο, ή εμφανίζονται ως «μη ορισμένες» (NULL).

3.3.1.3 Εύρεση όλων των οδών που περνούν από μία ορθογώνια περιοχή

Με το ερώτημα αυτό θέλουμε να βρούμε όλες τις οδούς που περνούν (έστω και ένα τμήμα τους) από μία ορθογώνια περιοχή που δίδεται ως είσοδος στο ερώτημα.

Έστω ότι η ορθογώνια περιοχή ορίζεται από δύο σημεία, $P_1(minLng, minLat)$ και $P_2(maxLng, maxLat)$.

Στη συνέχεια παρατίθεται το ερώτημα. Ας σημειωθεί ότι για το εν λόγω ερώτημα χρησιμοποιούμε τη «γραμματική» SQL που ορίζει η PostgreSQL.

```
SELECT
   id,
   street_name,
   street_type,
   street_length,
   ST_AsText(street_line) as street_line
FROM
   street
WHERE
   ST_Intersects(street_line,
                  ST_MakeEnvelope(minLng, minLat, maxLng, maxLat, 4326))
```

Κώδικας 3.4 Ερώτημα εύρεσης οδών που περνούν από μια περιοχή

Ουσιαστικά αυτό το ερώτημα χρησιμοποιεί τις «χωρικές» συναρτήσεις που υλοποιεί η επέκταση PostGIS, για να κατασκευάσει μια ορθογώνια περιοχή (ST_MakeEnvelope) με βάση τα δυο σημεία που παρέχονται ως είσοδος, και στη συνέχεια για να ελέγξει ποιες οδοί τέμνουν την περιοχή αυτή (ST_Intersects). Τέλος, μετατρέπει το πεδίο street_line από τη δυαδική μορφή στην οποία αποθηκεύεται στην αντίστοιχη αλφαριθμητική της αναπαράσταση(ST_AsText) ώστε να είναι εύκολη η μετέπειτα εξαγωγή της στην Java εφαρμογή.

3.3.1.4 Εύρεση οδού με βάση το χαρακτηριστικό (id) της

Σε πολλές περιπτώσεις στην εφαρμογή, παρέχεται στην εφαρμογή το χαρακτηριστικό μιας Οδού και ζητείται η εύρεση των σημείων που συσχετίζονται με αυτή, ή κάτι σχετικό με την οδό αυτή γενικότερα.

Ο SQL κώδικας που πραγματοποιεί αυτή την αναζήτηση παρατίθεται στη συνέχεια. Έστω *ID*, το χαρακτηριστικό της οδού για την οποία εκτελείται η αναζήτηση:

```
SELECT
   id,
   street_name,
   street_type,
   street_length,
   ST_AsText(street_line) as street_line
FROM
   street
WHERE
   id=ID
```

Κώδικας 3.5 Ερώτημα αναζήτησης μιας οδού με βάση το χαρακτηριστικό της (ID)

3.3.2 Παραδείγματα αναζητήσεων που πραγματοποιούμε με χρήση του Solr

Ο Solr παρέχει πρόσβαση στα δεδομένα του τόσο με γραφικό τρόπο αλλά κυρίως μέσω μιας διεπαφής υπηρεσιών πάνω από το πρωτόκολλο HTTP. Ουσιαστικά για κάθε συλλογή κοινοποιεί έναν «Ενιαίο Εντοπιστή Πόρων» (URL) μέσω του οποίου επιτρέπει στους χρήστες ή σε άλλες εφαρμογές να εκτελέσουν αναζητήσεις χρησιμοποιώντας την Γλώσσα Ειδικού Σκοπού (Domain Specific Language) που ορίζει ο Solr.

Για παράδειγμα μπορούμε να αναζητήσουμε όλα τα έγγραφα που φέρουν το κλειδί «Ακρόπολη» μέσα από ένα αίτημα HTTP με την ακόλουθη μορφή:

```
http://SOLR_HOST:PORT/solr/COLLECTION_NAME/select?q=Ακρόπολη
```

Όπου:

- SOLR_HOST: Η διεύθυνση(IP ή HostName) στην οποία είναι εγκατεστημένος ο Solr
- PORT: Η θύρα στην οποία «ακούει» ο Solr για αιτήματα. Για παράδειγμα 8983
- COLLECTION_NAME: Το όνομα της «συλλογής» στην οποία εκτελούμε την αναζήτηση

Για τις υπόλοιπες αναζητήσεις αγνοούμε το πρώτο κομμάτι του URL.

Πέρα από αναζητήσεις με μια λέξη ή πρόταση κλειδί, μπορούμε να ορίσουμε και πιο περίπλοκα φίλτρα αναζήτησης ανά πεδίο. Για παράδειγμα με το ακόλουθο query αναζητούμε Φωτογραφίες που ταιριάζουν με το κλειδί «Ακρόπολη»:

```
/geostream_test/select?fq=type:photo&q=Ακρόπολη
```

Τέλος μπορούμε να πραγματοποιήσουμε αναζήτηση με χωρικό χαρακτήρα για «έγγραφα» που εντοπίζονται εντός μιας ορισμένης περιοχής που ορίζεται από δύο σημεία, $P_1(minLng, minLat)$ και $P_2(maxLng, maxLat)$.

```
/geostream_test/select?q=Ακρόπολη&fq=type:photo AND
                        location:[minLat,minLng TO maxLat,maxLng]
```

Αλλά και να αναζητήσουμε «έγγραφα» με βάση την απόστασή τους, έστω R (σε km), από ένα σημείο $P(lng, lat)$

```
/geostream_test/select?pt=lat,lng&d=R&q=Ακρόπολη&fq=type:photo AND
{!geofilt sfield=location}
```

3.3.3 Solr και Java

Για να εκτελέσουμε αναζητήσεις στην πλατφόρμα Solr από την εφαρμογή μας που υλοποιήθηκε σε Java χρησιμοποιούμε τη διεπαφή πελάτη SolrJ (SolrJ 2015).

Η εν λόγω διεπαφή παρέχει ένα σύνολο από κλάσεις οι οποίες βοηθούν στην κατασκευή ερωτημάτων για την αναζήτηση στο Solr. Ουσιαστικά δημιουργεί μια αφαίρεση υψηλότερου επιπέδου ώστε να μη χρειάζεται ο προγραμματιστής να γράφει ολόκληρα τα URL για να εκτελεί αναζητήσεις.

4

Αλγόριθμοι επεξεργασίας και ανάλυσης δεδομένων

4.1 Δομές δεδομένων

Στο σημείο αυτό θα περιγραφούν οι δομές δεδομένων που χρησιμοποιήσαμε για τις ανάγκες των αλγορίθμων που θα περιγραφούν στη συνέχεια και ξεφεύγουν από τις συνηθισμένες δομές που παρέχονται εγγενώς από τη Java.

4.1.1 Point Grid

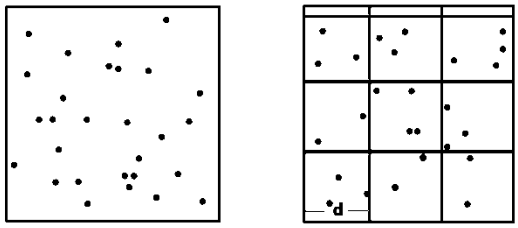
4.1.1.1 Γενική περιγραφή

Πρόκειται για μία χωρική δομή δεδομένων η οποία τεμαχίζει μία δοθείσα ορθογώνια περιοχή και αντίστοιχα το σύνολο των οντοτήτων που ανήκουν σε αυτή (π.χ. τα Σημεία Ενδιαφέροντος) σε κελιά με συγκεκριμένο μήκος πλευράς d. Έπειτα καθιστά αποδοτική την απάντηση στα εξής ερωτήματα:

- Σε ποιο κελί ανήκει μια χωρική οντότητα
- Ποιες οντότητες ανήκουν σε ένα κελί

- Δεδομένης μιας οντότητας, βρες όλες τις γειτονικές οντότητες (οντότητες που ανήκουν στο ίδιο αλλά και σε γειτονικά κελιά)

Παρακάτω φαίνεται μια εικονική αναπαράσταση της δομής:



Εικόνα 4.1 Γραφική αναπαράσταση λειτουργίας του Grid

4.1.1.2 Σκοπός χρήσης

Η δομή αυτή χρησιμοποιείται για να επιταχύνουμε αλγόριθμους που εξετάζουν γειτονικές οντότητες σε ένα χάρτη ώστε να εξάγουν πληροφορίες για την πυκνότητα μιας περιοχής ή για να τις ομαδοποιήσουν με βάση την απόστασή τους.

4.1.1.3 Λεπτομέρειες υλοποίησης

Η βασική δομή πίσω από τη δομή Grid είναι ένας πίνακας κατακερματισμού (HashMap) κλειδί του οποίου είναι η αλφαριθμική σύζευξη της τετμημένης και της τεταγμένης του εκάστοτε κελιού. Για παράδειγμα το κλειδί για το κελί (0,0) είναι το «0;0», για το κελί (10,5) είναι το «10,5» κοκ.

4.1.1.4 Απόδοση δομής δεδομένων

Μας ενδιαφέρει η απόδοση της δομής αυτής για τις 4 περιπτώσεις που αναφέραμε πιο πάνω. Στη συνέχεια θα εξετάσουμε κάθε μια από αυτές ξεχωριστά

Α. Σε ποιο κελί ανήκει μια οντότητα;

Για την κατάταξη μιας σημειακής οντότητας e(lng,lat) σε ένα κελί, εφαρμόζουμε τον ακόλουθο κώδικα:

```
X = (int) ceil((e.lng – area.minLng) / d);
Y = (int) ceil((e.lat - area.minLat) / d);
```

Κώδικας 4.2 Εύρεση κελιού X,Y στο Grid για μια οντότητα

Ουσιαστικά μετατρέπουμε την τοποθεσία ενός σημείου από ένα ζεύγος πραγματικών αριθμών σε ένα ζεύγος ακεραίων που αντιστοιχούν στο κελί στο οποίο ανήκει η εν λόγω οντότητα. Η αντιστοίχιση δηλαδή γίνεται πολύ γρήγορα O(1).

Β. Ποιες οντότητες ανήκουν σε ένα κελί

Η εύρεση των οντοτήτων που ανήκουν σε ένα κελί γίνεται επίσης σε O(1). Δεδομένου ενός κελιού (X,Y) αρκεί να κατασκευάσουμε το κλειδί «X;Y» και να το αναζητήσουμε στον πίνακα κατακερματισμού (O(1)).

Γ. Δεδομένης μιας οντότητας, βρες όλες τις γειτονικές σε αυτή οντότητες

Η εύρεση των γειτόνων ενός κελιού επίσης γίνεται πολύ γρήγορα με τα εξής βήματα:

1. Βρίσκουμε σε ποιο κελί ανήκει
2. Βρίσκουμε τα γειτονικά σε αυτό κελιά
3. Βρίσκουμε τις οντότητες που ανήκουν στα κελιά αυτά

Ουσιαστικά τα 1 και 2 περιγράφηκαν προηγουμένως. Ενώ η εύρεση των γειτόνων ενός κελιού είναι τετριμμένη και γίνεται επίσης σε O(1).

Δ. Κατασκευή της δομής Grid

Τέλος θα μελετήσουμε την απόδοση κατασκευής μιας τέτοιας δομής και του κατακερματισμού των οντοτήτων σε κελιά.

Για την κατασκευή του Grid λοιπόν πρέπει, για κάθε οντότητα να υπολογίσουμε το κελί (O(1)) στο οποίο ανήκει και στη συνέχεια να το προσθέσουμε στην λίστα που αντιστοιχεί στο κελί αυτό (O(1)). Άρα συνολικά δεδομένου ότι υπάρχουν N οντότητες στην περιοχή του Grid, ο συνολικός χρόνος κατασκευής του είναι O(N).

4.1.2 Vantage Point Tree (VP)

Το VP δένδρο (Yianilos 1993) είναι μια δομή που χρησιμοποιείται για την επιτάχυνση της αναζήτησης των γειτόνων μιας οντότητας με βάση ένα Μετρικό Χώρο (Metric Space) όταν είναι γνωστή μόνο η απόσταση μεταξύ των οντοτήτων χωρίς να γνωρίζουμε ωστόσο τις ακριβείς συντεταγμένες τους στο χώρο αυτό.

Για να πετύχει την επιτάχυνση της αναζήτησης των γειτόνων μιας οντότητας εκμεταλλεύεται την τριγωνική ανισότητα την οποία πρέπει να ακολουθούν οι αποστάσεις ανάμεσα στα

σημεία ενός μετρικού χώρου. Δηλαδή, αν p_1, p_2 και p_3 είναι 3 σημεία του χώρου αυτού, και d_{12}, d_{13}, d_{23} οι μεταξύ τους αποστάσεις, τότε για ένα Μετρικό χώρο πρέπει πάντα να ισχύει ότι:

$$d_{12} + d_{23} \geq d_{13}$$

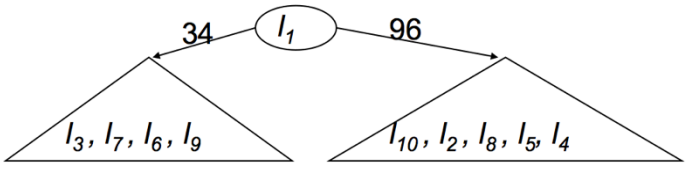
4.1.2.1 Κατασκευή Δέντρου Vantage Point

Για να δούμε πώς το VP δένδρο εκμεταλλεύεται αυτή την ανισότητα ας δούμε πρώτα πως αυτό κατασκευάζεται:

Έστω $L = \{l_1, l_2, \dots, l_3\}$ μία λίστα οντοτήτων και $d = (l_i, l_j \rightarrow d_{ij} \in R)$ μία συνάρτηση η οποία επιστρέφει μια μετρική απόσταση ανάμεσα σε οποιεσδήποτε δυο οντότητες της λίστας αυτής. Η κατασκευή του VP δένδρου ξεκινά με την επιλογή ενός τυχαίου l_i από τη λίστα L. Στη συνέχεια, ο αλγόριθμος ταξινομεί τις υπόλοιπες οντότητες, με βάση την απόστασή τους από την οντότητα l_i και χωρίζει την ταξινομημένη λίστα σε δυο υπό-λίστες. Έπειτα κατασκευάζει έναν κόμβο ο οποίος θα έχει στα αριστερά του τις πρώτες μισές οντότητες και στα δεξιά του τις υπόλοιπες. Επίσης ορίζει για τον κόμβο δύο ιδιότητες. Την «Μέγιστη Απόσταση Αριστερά» (maxDL) και την «Ελάχιστη Απόσταση Δεξιά» (minDR). Τέλος για κάθε υπό-λίστα εκτελεί την ίδια διαδικασία αναδρομικά. Έτσι στο τέλος της εκτέλεσής του έχει δημιουργήσει ένα δυαδικό δέντρο με βάση τις αποστάσεις των οντοτήτων της λίστας L.

Μια γραφική αναπαράσταση ενός βήματος του αλγορίθμου κατασκευής του VP δένδρου φαίνεται στη συνέχεια:

l_3	l_7	l_6	l_9	l_{10}	l_2	l_8	l_5	l_4
5	6	18	34	96	102	111	300	401



4.1.2.2 Αναζήτηση των γειτόνων μιας οντότητας

Στη συνέχεια θα εξετάσουμε πως με βάση αυτό το δέντρο μπορούμε να απορρίψουμε ένα μεγάλο κομμάτι από το δέντρο αυτό, δεδομένου ότι η απόσταση μέσα στην οποία αναζητάμε τους γείτονες μιας οντότητας είναι επαρκώς μικρή.

Έστω λοιπόν ότι θέλουμε να βρούμε τους γείτονες μιας οντότητας q με βάση μια μέγιστη απόσταση d. Ξεκινάμε από τι ρίζα r του δέντρου και υπολογίζουμε την απόσταση της οντότητας που αυτή αντιπροσωπεύει με την q. Έστω ότι η απόσταση αυτή είναι $d(q,r)$

Μπορούμε ασφαλώς να αγνοήσουμε το αριστερό υπό-δέντρο αν για κάθε οντότητα o_i ισχύει ότι $d(o_i,q) \geq d$

Με βάση τώρα την τριγωνική ανισότητα του αριστερού υπό-δέντρου θα ισχύει ότι:

$$d(q,o_i) + d(o_i,r) \geq d(q,r) \rightarrow d(q,o_i) \geq d(q,r) - d(o_i,r)$$

Από την κατασκευή του δένδρου όμως γνωρίζουμε ότι η μέγιστη δυνατή τιμή του $d(o_i,r)$ ισούται με $maxDL$. Άρα αν

$$d(q,r) - maxDL \geq d \quad (1)$$

τότε θα ισχύει πάντα ότι $d(o_i,r) \geq d$. Που σημαίνει ότι μπορούμε να αγνοήσουμε όλο το αριστερό υπό-δέντρο. Με παρόμοιο τρόπο αποδεικνύεται ότι μπορούμε να αγνοήσουμε το δεξί υπό-δέντρο αν ισχύει ότι

$$minDR - d(q,r) \geq d \quad (2)$$

Σε κάθε βήμα λοιπόν του αλγόριθμου για την εύρεση των γειτόνων μιας οντότητας σε ένα VP δένδρο, υπολογίζουμε σε κάθε βήμα την απόσταση $d(q,r)$, και ανάλογα με το αν ισχύουν οι ανισότητες (1) ή (2) «κλαδεύουμε» την αναζήτηση και προχωράμε στα υπό-δένδρα μόνο αν χρειάζεται. Ο αλγόριθμος σταματά αν όλοι οι κόμβοι έχουν εξερευνηθεί ή «κλαδευτεί».

Όπως είναι φανερό το ευρετήριο αυτό προσφέρει βελτίωση στο χρόνο εκτέλεσης μόνο αν η απόσταση d είναι αρκετά μικρή ώστε να κλαδεύονται τα υπό-δένδρα με μεγάλη πιθανότητα.

Αν ωστόσο η απόσταση d είναι αρκετά μεγάλη τότε ενδεχομένως να εξεταστούν όλοι οι κόμβοι.

Η ιδανική απόσταση d ορίζεται ανάλογα με τη συνάρτηση απόσταση και την εμπειρική παρατήρηση των δεδομένων.

4.2 Ο αλγόριθμος ομαδοποίησης σημείων σε Περιοχές Ενδιαφέροντος

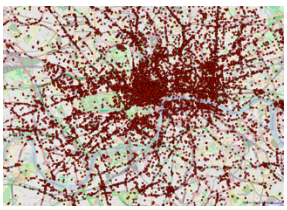
4.2.1 Περιγραφή του προβλήματος

Έστω ένα σύνολο χωρικών οντοτήτων P που βρίσκονται εντός μίας ορθογώνιας περιοχής Π, και d η μέγιστη απόσταση που μπορούν να απέχουν δυο οντότητες για να θεωρηθούν γειτονικές. Ζητούμενο είναι να ομαδοποιήσουμε της οντότητες αυτές σε ομάδες τέτοιες ώστε κάθε οντότητα που ανήκει σε μία ομάδα να έχει τουλάχιστον K γείτονες.

Σκοπός αυτής της ενότητας δεν είναι να εξετάσει πώς βρίσκουμε το σύνολο οντοτήτων P με βάση κάποια κριτήρια, αλλά να εμβαθύνει στον τρόπο με τον οποίο δεδομένου ενός τέτοιου συνόλου ομαδοποιούμε τις οντότητές του σε ικανά-πυκνές ομάδες οντοτήτων.

Ο αλγόριθμος δέχεται ως είσοδο μία λίστα από οντότητες, την μέγιστη απόσταση d και έναν αριθμό K, και επιστρέφει μία λίστα από σύνολα P1, P2, …, PN. Κάθε ένα από τα P_i ορίζει μια ομάδα οντοτήτων που πληροί τις προϋποθέσεις, όπως αυτές ορίζονται από το d και το K.

Ένα παράδειγμα εκτέλεσης του αλγορίθμου για την περιοχή του Λονδίνου φαίνεται στις παρακάτω εικόνες. Είναι φανερό ότι η ομαδοποίηση των δεδομένων βοηθά σημαντικά στην εξαγωγή συμπερασμάτων για μια περιοχή.



Εικόνα 4.2 Χωρικές οντότητες χωρίς ομαδοποίηση

Εικόνα 4.2 Ομάδες χωρικών οντοτήτων για συγκεκριμένο K και d

4.2.2 Τρόπος λειτουργίας του αλγορίθμου

Ο αλγόριθμος clustering βασίζεται στον DBSCAN (M. Ester 1996). Ο DBSCAN είναι ένας αλγόριθμος που χρησιμοποιείται για την εύρεση πυκνών ομάδων από χωρικές οντότητες.

Σε κάθε βήμα επιλέγει τυχαία μια οντότητα p_i από τη λίστα εισόδου P, την οποία δεν έχει ήδη χαρακτηρίσει ως «ΘΟΡΥΒΟ» ή «ΜΕΛΟΣ ΟΜΑΔΑΣ», και προχωρά βρίσκοντας όλες τις γειτονικές σε αυτό οντότητες που απέχουν το πολύ απόσταση d. Στη συνέχεια εξετάζει αν το πλήθος των γειτονικών οντοτήτων του p_i, έστω Γ_{pi}, είναι μικρότερο από K, και αν ναι, χαρακτηρίζει το p_i ως «ΘΟΡΥΒΟ» και προχωρά στην επόμενη οντότητα από το P που δεν έχει ήδη επισκεφθεί. Αν, ωστόσο, το Γ_{pi} είναι μεγαλύτερο από K τότε θεωρεί ότι το p_i είναι ικανοποιητικά-πυκνό και χαρακτηρίζει το p_i και τους γείτονές του ως «ΜΕΛΟΣ ΟΜΑΔΑΣ». Στη συνέχεια επιχειρεί να επεκτείνει την ομάδα προσθέτοντας σε αυτή όλες τις ικανοποιητικά-πυκνές οντότητες που θα βρει εκτελώντας την ίδια διαδικασία σε κάθε ικανοποιητικά-πυκνό γείτονα του p_i. Στο σημείο αυτό, όταν η περαιτέρω επέκταση τις ομάδας δεν είναι εφικτή, προσθέτει την ομάδα αυτή στη λίστα που θα επιστρέψει ως έξοδο. Η εκτέλεση σταματά όταν ο αλγόριθμος έχει χαρακτηρίσει όλες τις οντότητες που ανήκουν στο P ως «ΘΟΡΥΒΟ» ή «ΜΕΛΟΣ ΟΜΑΔΑΣ».

Η πολυπλοκότητα του DBSCAN χωρίς την ύπαρξη κάποιας ειδικής δομής ή χωρικού ευρετηρίου για τη γρήγορη αναζήτηση γειτόνων είναι $O(N^2)$. Αυτό γιατί, ο αλγόριθμος επισκέπτεται κάθε οντότητα μόνο μια φορά, ωστόσο, για να βρει τους γείτονές της χρειάζεται να επισκεφθεί όλες της οντότητες ξανά.

4.2.3 Βελτιστοποίηση του αλγορίθμου και πολυπλοκότητα

Μπορούμε να βελτιστοποιήσουμε σε μεγάλο βαθμό τον αλγόριθμο χρησιμοποιώντας τη δομή Point Grid που περιγράψαμε στο κεφάλαιο 1.1.1. Χρησιμοποιώντας το Point Grid με μέγεθος πλευράς d, μπορούμε πολύ γρήγορα να βρούμε τους γείτονες μιας χωρικής οντότητας της λίστας P. Αρκεί να βρούμε τις οντότητες που ανήκουν στο ίδιο ή σε γειτονικά κελιά, έστω Γ_{pi}', του Grid και να εξετάσουμε μόνο αυτές σε σχέση με την απόστασή τους. Επίσης, μπορούμε πολύ γρήγορα να αποκλείσουμε από την εξέταση οντότητες για τις οποίες ισχύει ότι $|\Gamma_{pi}'| < K$.

Στη γενική περίπτωση όπου τα δεδομένα είναι διεσπαρμένα σε πολλά κελιά του Point Grid, μπορούμε να πετύχουμε μια αρκετά καλή επιτάχυνση του αλγορίθμου. Εντούτοις, στη χειρότερη περίπτωση, που όλες οι οντότητες βρίσκονται στο ίδιο κελί έχουμε μηδενικό κέρδος στην ταχύτητα εκτέλεσης του αλγορίθμου αφού πάλι θα πρέπει να εξεταστούν όλες ως προς απόστασή τους από την οντότητα που εξετάζουμε.

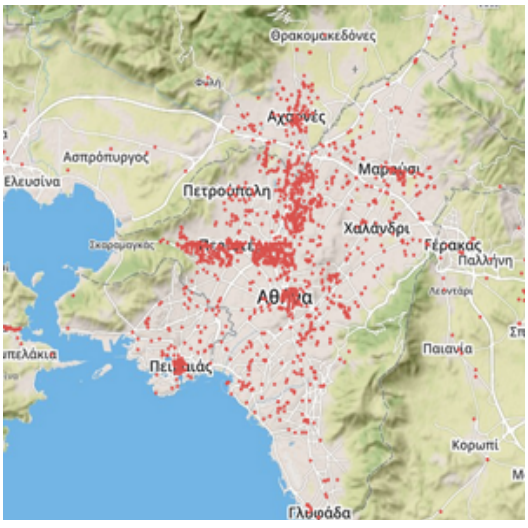
4.3 Ο αλγόριθμος εύρεσης Οδών Ενδιαφέροντος ή Γραφικών Οδών

4.3.1 Περιγραφή του προβλήματος

Στο προηγούμενο κεφάλαιο μελετήσαμε τον τρόπο με τον οποίο εξάγουμε πυκνές «Περιοχές Ενδιαφέροντος» από σημεία ενδιαφέροντος. Στο κεφάλαιο αυτό θα μελετήσουμε τον αλγόριθμο που αντιστοιχίζει «Σημεία Ενδιαφέροντος» ή «Φωτογραφίες» με «Οδούς» και εξάγει από μία περιοχή τις «Οδούς» με τη μεγαλύτερη πυκνότητα σε «Σημεία Ενδιαφέροντος» ή «Φωτογραφίες». Στη συνέχεια, χρησιμοποιούμε τον όρο «Σημεία Ενδιαφέροντος» ή απλά «Σημεία» για να περιγράψουμε, τόσο «Σημεία Ενδιαφέροντος» όσο και «Φωτογραφίες».

Έστω λοιπόν Π μια περιοχή στο χάρτη και P ένα σύνολο Σημείων Ενδιαφέροντος που ανήκουν σε αυτή την περιοχή και πληρούν κάποιες προϋποθέσεις (όπως για παράδειγμα Σημεία Ενδιαφέροντος για την κατηγορία «Καταστήματα»). Έστω επίσης S το σύνολο των Οδών που περνούν από αυτή την περιοχή. Ζητούμενο του αλγορίθμου είναι η νεύρεση των Οδών εκείνων από το S, οι οποίες αντιστοιχίζονται με τουλάχιστον K Σημεία Ενδιαφέροντος τα οποία να απέχουν το πολύ απόσταση d από την οδό.

Στη συνέχεια φαίνεται μια γραφική αναπαράσταση του αποτελέσματος της εκτέλεσης του αλγορίθμου για καταστήματα στην περιοχή της Αθήνας:



Εικόνα 4.3 Σημεία Ενδιαφέροντος για την κατηγορία Καταστήματα στην Αθήνα

Εικόνα 4.5 Οδοί Ενδιαφέροντος για την κατηγορία Καταστήματα στην Αθήνα

4.3.2 Τρόπος λειτουργίας του αλγορίθμου

Βασικά δομικά συστατικά του αλγορίθμου αυτού είναι η δομή Point Grid που περιγράψαμε στην ενότητα 1.1.1 σε συνδυασμό με τον Bresenham-based Supercover Line Algorithm (Dedu 2001), έναν αλγόριθμο που δεδομένης μιας γραμμικής οντότητας και ενός συστήματος συντεταγμένων, επιστρέφει όλα τα ακέραια κελιά που αυτή ακουμπά. Ο αλγόριθμος αυτός θα εξηγηθεί με λεπτομέρεια στη συνέχεια. Σε αυτή την ενότητα θα περιγράψουμε την υπόλοιπη διαδικασία υπολογισμού των Οδών Ενδιαφέροντος.

Στο επόμενο απόσπασμα βλέπουμε σε ψευδοκώδικα τον εν λόγω αλγόριθμο. Π είναι η περιοχή για την οποία εκτελείται, P το σύνολο των Σημείων Ενδιαφέροντος που εντοπίστηκαν στην περιοχή Π, και τέλος, d η μέγιστη απόσταση που μπορεί να απέχει ένα σημείο από μία οδό για να θεωρηθεί ότι «ανήκει» στην οδό αυτή. Σημειώνεται, ότι κάθε Σημείο Ενδιαφέροντος είναι δυνατόν να αντιστοιχιστεί με περισσότερες από μία Οδούς.

```
Result ← {}

grid ← new Grid(P, Π, d);

for each (street in S) {
   cellsForStreet ← grid.getCellsForStreet(street);
   points ← getEntitiesIn(cellsForStreet)
   if (points.size() >= K)  points ← filterBasedOnDistance(points, street, d);
```

Κώδικας 4.3 Ψευδοκώδικας εύρεσης Οδών Ενδιαφέροντος

Αρχικά, για λόγους απόδοσης, όπως και με τις περιοχές ενδιαφέροντος, ο αλγόριθμος προσθέτει όλα τα Σημεία Ενδιαφέροντος σε ένα Point Grid όπως αυτό που περιγράψαμε στην ενότητα 1.1.1 με μέγεθος πλευράς ίσο με d. Στη συνέχεια, παίρνει κάθε οδό από το σύνολο S, την κανονικοποιεί στις διαστάσεις του Point Grid και βρίσκει τα κελιά εκείνα του Grid από τα οποία περνά καθώς και τα γειτονικά τους κελιά. Αν το πλήθος των οντοτήτων που περιέχονται σε αυτά είναι μικρότερο από K τότε απλά αγνοεί την οδό αυτή. Αν το πλήθος των οντοτήτων που περιέχονται στα τελευταία είναι μεγαλύτερο από K τότε βρίσκει εκείνες τις οντότητες που πράγματι απέχουν το πολύ d από την οδό και συγκρίνει ξανά το πλήθος τους με το K. Μόνο αν βρει ότι περισσότερες από K οντότητες βρίσκονται κοντά στην οδό θα την προσθέσει στο αποτέλεσμα. Ο αλγόριθμος τερματίζει όταν έχει εξετάσει όλες τις οδούς του συνόλου S.

Όπως αναφέραμε στην αρχή θέλουμε τις οδούς με τη μεγαλύτερη πυκνότητα σε σημεία ενδιαφέροντος. Ως πυκνότητα ορίζουμε τον αριθμό των Σημείων Ενδιαφέροντος που αντιστοιχίστηκαν με την οδό ανά μέτρο.

Η ταξινόμηση γίνεται με βάση το ακόλουθο μέγεθος:

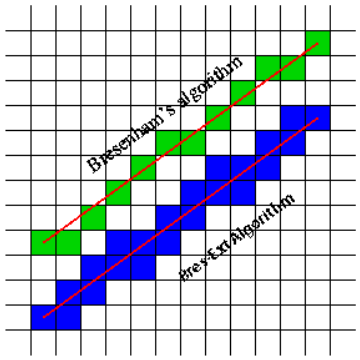
$$score = \frac{Αριθμός\ Σημείων}{\max(Μήκος\ Οδού, 100)}$$

Ο λόγος που επιλέξαμε αυτή την τιμή για το score είναι για να «υποβαθμίσουμε» τις οδούς εκείνες που σκαρφαλώνουν ψηλά στα αποτελέσματα λόγω του ότι είναι πολύ μικρές σε μήκος. Για παράδειγμα μάλλον δε μας ενδιαφέρει μία οδός η οποία έχει μήκος 2 μέτρα και τυχαίνει να έχει τριγύρω της 10 Σημεία Ενδιαφέροντος τα οποία κατά πάσα πιθανότητα δεν ανήκουν καν σε αυτή.

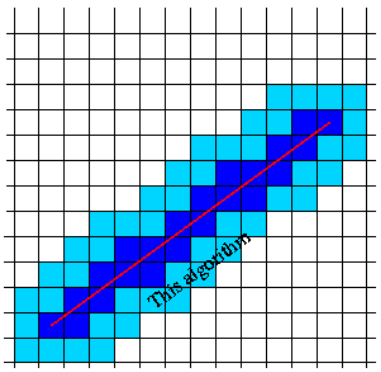
4.3.3 Bresenham-based αλγόριθμος για τον υπολογισμό του Supercover Line

Όπως περιγράψαμε στην προηγούμενη ενότητα, για να βρούμε τις οδούς ενδιαφέροντος χρησιμοποιήσαμε έναν Bresenham-based αλγόριθμο για τον υπολογισμό του Supercover Line (Dedu 2001) τον οποίο τροποποιήσαμε για τις δικές μας ανάγκες.

Στη συνέχεια παρουσιάζουμε γραφικά τη λειτουργία των διαφορετικών εκδοχών του αλγόριθμου που αναφέραμε προηγουμένως.



Εικόνα 4.5 Γραφική αναπαράσταση των κελιών που επιστρέφει ο αλγόριθμος του Bresenham και ο αλγόριθμος Supercover



Εικόνα 4.5 Γραφική αναπαράσταση των κελιών που επιστρέφει η δική μας εκδοχή του Supercover αλγόριθμου.

Στην περίπτωσή μας, κάθε οδός S αποτελείται από δύο ή περισσότερα σημεία έστω $P_1, P_2, \dots, P_N$. Δεδομένου λοιπόν ενός Point Grid πλευράς d σκοπός μας είναι να υπολογίσουμε τα κελιά του Grid από τα οποία περνά η οδός. Έτσι για κάθε γειτονικό ζεύγος σημείων P_i, P_{i+1}, αρχικά βρίσκουμε τα κελιά του Grid στα οποία ανήκουν, και στη συνέχεια

εκτελούμε τον αλγόριθμο Supercover (Dedu 2001) για να βρούμε τα κελιά του Grid από τα οποία περνά το ευθύγραμμο τμήμα που ορίζουν τα δύο αυτά σημεία. Έπειτα, κάνοντας χρήση του Grid βρίσκουμε όλα τα γειτονικά κελιά του Supercover και τα προσθέτουμε και αυτά στη λύση. Η διαδικασία επαναλαμβάνεται για κάθε ζεύγος σημείων $P_i, P_{i+1}, i = 1 \dots (N-1)$.

Η πολυπλοκότητα εκτέλεσης του αλγορίθμου για μία οδό μήκους L σε ένα Grid με μήκος πλευράς d δίνεται από τον ακόλουθο τύπο:

$$O(\frac{L}{d})$$

4.3.4 Συνολική πολυπλοκότητα εύρεσης οδών ενδιαφέροντος

Αν το σύνολο P έχει μέγεθος N, το σύνολο S έχει μέγεθος M και το μέσο μήκος των οδών είναι L τότε η πολυπλοκότητα του αλγορίθμου στη χειρότερη περίπτωση είναι ίση με:

$$O(N + MN\frac{L}{d})$$

Αυτό γιατί αν όλα τα σημεία και οι οδοί είναι στο ίδιο κελί του Grid τότε για κάθε οδό πρέπει να εξεταστούν M σημεία. Ωστόσο δεδομένου του ότι το Grid χωρίζει αποτελεσματικά το χώρο σε κελιά η απόδοση του αλγορίθμου βελτιώνεται δραστικά.

4.4 Ο αλγόριθμος ποικίλων φωτογραφιών

4.4.1 Περιγραφή του προβλήματος

Έστω μια οδός s ή μία περιοχή r η οποία έχει στη γειτονιά της ένα σύνολο φωτογραφιών P. Σκοπός του προβλήματος είναι να εξάγει ένα υποσύνολο αυτών των φωτογραφιών μεγέθους K το οποίο να είναι όσο το δυνατόν πιο σχετικό με το r ή το s, και ταυτόχρονα ποικίλο (Sreenivas Gollapudi 2009). Θέλουμε δηλαδή να περιγράφει ικανοποιητικά την περιοχή ή την οδό.

4.4.2 Ορισμός απόστασης

Ουσιαστικά θέλουμε το σύνολο που επιστρέφεται από τον αλγόριθμο να είναι σχετικό με τη δομή στην οποία ανήκει και ταυτόχρονα τα μέλη του να είναι όσο το δυνατόν πιο διαφορετικά μεταξύ τους. Επίσης, θέλουμε η απόσταση να ορίζει ένα Μετρικό Χώρο ώστε να μπορεί να χρησιμοποιηθεί το VP δένδρο που περιγράψαμε στην ενότητα 4.1.2 για τη βελτίωση της απόδοσης του αλγορίθμου.

Για να ορίσουμε τη σχετικότητα ή τη διαφορετικότητα ενός συνόλου με τη δομή, χρησιμοποιούμε ένα σύνολο από tags που χαρακτηρίζουν τις φωτογραφίες καθώς και τη χωρική απόστασή τους.

4.4.2.1 Χωρική απόσταση

Η χωρική απόσταση ανάμεσα σε δύο σημεία ορίζεται ως η ευκλείδεια απόστασή τους. Δίνεται δηλαδή από τον τύπο:

$$spatial\_distance = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$$

Για την ακρίβεια επειδή θέλουμε να κανονικοποιήσουμε τη χωρική απόσταση ανάμεσα στο 0 και το 1 ώστε να είναι συγκρίσιμη με τη σημασιολογική απόσταση, τη διαιρούμε με τη μέγιστη δυνατή απόσταση που είναι εφικτή για τη δομή δεδομένων που εξετάζουμε.

Για παράδειγμα η μέγιστη δυνατή απόσταση η οποία είναι εφικτή για μία οδό είναι το μήκος της οδού ενώ για μία περιοχή ενδιαφέροντος είναι η μέγιστη διάμετρος του πολυγώνου που την ορίζει.

Για τον υπολογισμό της μέγιστης διαμέτρου για ένα πολύγωνο χρησιμοποιούμε τον αλγόριθμο που βρίσκει τη μέγιστη απόσταση ανάμεσα στα αντικριστά σημεία του πολυγώνου (Preparata 1985).

Τελικά ως χωρική απόσταση χρησιμοποιούμε τη σχετική χωρική απόσταση:

$rel\_spatial\_distance = \frac{\sqrt{(x_1-x_2)^2+(y_1-y_2)^2}}{\text{maxAreaD}istance}$ η οποία κυμαίνεται ανάμεσα σε 0 και 1.

4.4.2.2 Σημασιολογική απόσταση

Για τη σημασιολογική απόσταση μεταξύ δυο φωτογραφικών ή μιας φωτογραφίας από ένα σύνολο φωτογραφιών χρησιμοποιούμε το cosine similarity ανάμεσα στους πίνακες που ορίζουν τα tags των φωτογραφιών.

Έστω A το σύνολο των tags της μιας φωτογραφίας α, B το σύνολο των tags της φωτογραφίας β και $\Gamma = A \cup B$ το σύνολο όλων των πιθανών tags για τις δύο φωτογραφίες.

Ορίζουμε ως πίνακα tags της φωτογραφίας α και β ως εξής:

$V_A = [V_{A_1}, V_{A_2}, \dots, V_{A_N}]$ όπου $V_{A_i} = \begin{cases} 1 \; \alpha\nu \; \Gamma_i \in A \\ 0 \; \alpha\nu \; \Gamma_i \notin A \end{cases}$

$V_B = [V_{B_1}, V_{B_2}, \dots, V_{B_N}]$ όπου $V_{B_i} = \begin{cases} 1 \; \alpha\nu \; \Gamma_i \in B \\ 0 \; \alpha\nu \; \Gamma_i \notin B \end{cases}$

Δεδομένων αυτών των ορισμών το cosine similarity μεταξύ των δύο Vectors δίδεται από τον ακόλουθο τύπο: $R = \frac{V_A \cdot V_B}{\|V_A\| \|V_B\|} = \frac{\Sigma V_{A_i} \times V_{B_i}}{\sqrt{\Sigma \left(V_{A_i}\right)^2} \times \sqrt{\Sigma \left(V_{B_i}\right)^2}}$

Με βάση το cosine similarity μπορούμε να ορίσουμε το cosine distance ως $D_{nm} = 1 - R$

Ωστόσο, αυτός ο ορισμός της απόστασης δεν ορίζει ένα «μετρικό» χώρο για τις οντότητες, και αυτό γιατί δεν ισχύει πάντα η τριγωνική ανισότητα για αυτή την απόσταση.

Ένας ορισμός απόστασης που ορίζει ένα μετρικό χώρο για τις οντότητες με βάση το cosine similarity που υπολογίσαμε προηγουμένως είναι ο ακόλουθος (Van Dongen 2012)

$$D_{metric} = \sqrt{\frac{(1-R)}{2}}$$

4.4.2.3 Συνολική απόσταση

Τέλος, ως συνολική απόσταση (σημασιολογική και χωρική) ορίζουμε το μέσο όρο των δύο αποστάσεων η οποία είναι επίσης «μετρική» ως γραμμικός συνδυασμός «μετρικών» αποστάσεων.

$$totalDistance = \frac{rel\_spatial\_distance + semantic\_distance}{2}$$

4.4.3 Τρόπος λειτουργίας του αλγορίθμου

Ο αλγόριθμος diversification δέχεται ως είσοδο ένα σύνολο από φωτογραφίες έστω P και έναν ακέραιο αριθμό K, και προσπαθεί να βρει ένα ικανοποιητικά ποικίλο και σχετικό υποσύνολο του P.

Βασική ιδέα του αλγορίθμου είναι ο διαχωρισμός των οντοτήτων σε όμοιες ομάδες. Ομάδες δηλαδή, που περιέχουν οντότητες που δεν απέχουν (σημασιολογικά και χωρικά) περισσότερο από μία σταθερά. Ουσιαστικά προσπαθεί να τις χωρίσει σε υποπεριοχές ακτίνας d.

Έχοντας βρει όλες τις όμοιες ομάδες, τις ταξινομεί με βάση το μέγεθός τους θεωρώντας ότι όσο περισσότερες οντότητες περιέχει μια ομάδα τόσο περισσότερο αντιπροσωπεύει την οδό ή την περιοχή που εξετάζει.

Έπειτα, για κάθε ομάδα βρίσκει την «κεντροειδή» (centroid) οντότητα, και την προσθέτει στο αποτέλεσμα. Συνεχίζει μέχρι να έχει πάρει K οντότητες ή να εξαντλήσει τα αποτελέσματα.

Στη συνέχεια παρατίθεται ο ψευδοκώδικας του εν λόγω αλγορίθμου. Θεωρούμε ότι P είναι το σύνολο των φωτογραφιών και K το πλήθος των φωτογραφιών που θέλουμε να βρούμε.

```
clusters ← findSimilarClusters(P);
clusters ← sortBySize(clusters);
result ← newArrayList();
resultSize ← min(K, P.size());
while (result.size() < resultSize) {
   for each cluster in clusters:
      centroid ← findCentroid(cluster);
      result.add(centroid);
      cluster.remove(centroid);
   }
}
return result;
```

Κώδικας 4.4 Επιλογή ποικίλων φωτογραφιών

4.4.3.1 Υπολογισμός όμοιων ομάδων

Στην ενότητα αυτή θα παρουσιάσουμε τον αλγόριθμο επιλογής όμοιων ομάδων από μία λίστα φωτογραφιών P. Ως όμοια ομάδα, ορίζουμε ένα σύνολο φωτογραφιών οι οποίες ανήκουν σε ένα κύκλο με κέντρο ένα τυχαίο p_i και ακτίνα d.

Ο αλγόριθμος δέχεται ως είσοδο μία λίστα P και μία ακτίνα d και επιστρέφει μία λίστα από όμοιες ομάδες που περιέχουν 1 ή περισσότερα στοιχεία.

Σε κάθε βήμα επιλέγει μία οντότητα p_i που δεν έχει ήδη επισκεφθεί και βρίσκει το σύνολο των γειτόνων της, έστω N_{p_i}, που δεν ανήκουν σε κανένα άλλο cluster και δεν «απέχουν» από το p_i απόσταση μεγαλύτερη από d. Στη συνέχεια προσθέτει κάθε οντότητα στην ίδια ομάδα και χαρακτηρίζει κάθε μια από αυτές ως «ΜΕΛΟΣ-ΟΜΑΔΑΣ». Έπειτα επιλέγει την επόμενη οντότητα από τη λίστα P που δεν έχει ήδη επισκεφθεί και επαναλαμβάνει τη διαδικασία.

4.4.4 Πολυπλοκότητα

Η πολυπλοκότητα του αλγορίθμου καθορίζεται κατά βάση από την ομαδοποίηση των σημείων και των υπολογισμό του «κεντροειδούς» κάθε ομάδας. Και τα δύο βήματα χαρακτηρίζονται από πολυπλοκότητα χειρότερης περίπτωσης $O(N^2)$. Έτσι η πολυπλοκότητα του αλγορίθμου συνολικά, είναι στη χειρότερη περίπτωση:

$$O(N^2)$$

Ωστόσο, αν η απόσταση d είναι μεγαλύτερη από ένα κατώφλι $d_{min} = \min(d(p_i, p_j)), p_i, p_j \in P$, τότε η απόδοση του αλγόριθμου είναι σημαντικά καλύτερη προσεγγίζοντας το O(N). Αυτό γιατί σε κάθε βήμα, όσοι γείτονες εντοπίζονται αποκλείονται από μελλοντικές αναζητήσεις.

Αν δε χρησιμοποιήσουμε ένα VP δένδρο τότε μπορούμε να βελτιώσουμε σημαντικά την απόδοση του αλγορίθμου και για μικρά d, αφού όπως περιγράψαμε στην ενότητα 4.1.2 το VP δέντρο είναι πιο αποδοτικό όταν το d είναι μικρό. Φυσικά, με την εισαγωγή του δέντρου «πληρώνουμε» το χρόνο που χρειαζόμαστε για τη κατασκευή του.

4.5 Διασύνδεση Σημείων Ενδιαφέροντος με Φωτογραφίες και Εκδηλώσεις.

Στην ενότητα αυτή εξετάζουμε πώς έχοντας ως αφετηρία μια οντότητα, μπορούμε να βρούμε σχετικές οντότητες ίδιου ή διαφορετικού τύπου. Για παράδειγμα πώς δεδομένου ενός Σημείου ενδιαφέροντος μπορούμε να βρούμε σχετικές Φωτογραφίες, Γεγονότα ή και αντίστοιχα σημεία ενδιαφέροντος τριγύρω.

Ουσιαστικά, για να το πετύχουμε αυτό εκμεταλλευόμαστε χωρική/σημασιολογική λειτουργικότητα του Solr (Solr 2015). Έτσι κάθε φορά που ο χρήστης ζητά να βρει σχετικές οντότητες ζητάμε από το Solr να μας επιστρέψει όλες τις οντότητες του τύπου που ενδιαφέρει το χρήστη (Φωτογραφίες, Σημεία Ενδιαφέροντος ή Γεγονότα) σε μία ακτίνα γύρω από το αρχικό σημείο και που «ταιριάζουν» με τη λεζάντα του αρχικού σημείου.

Για παράδειγμα, έστω ότι έχουμε ένα σημείο με τίτλο «Ακρόπολη» και θέλουμε να βρούμε σχετικές φωτογραφίες. Τότε, ζητάμε από το Solr να μας βρει όλες τις φωτογραφίες που απέχουν το πολύ d από το αρχικό σημείο και περιέχουν κάπου στην περιγραφή τους τη λέξη «Ακρόπολη».

4.6 Συσχέτιση Περιοχών και Οδών με Σημειακές Οντότητες

Στην ενότητα αυτή θα εξετάσουμε πως βρίσκουμε οντότητες που περιέχονται χωρικά σε μία «Περιοχή Ενδιαφέροντος».

Μια «Περιοχή Ενδιαφέροντος» στην εφαρμογή μας ορίζεται, χωρικά, από ένα σύνολο σημείων που κατασκευάζουν ένα convex πολύγωνο. Ωστόσο, το Solr, χωρίς κάποια επιπλέον προσθήκη, υποστηρίζει μόνο ερωτήματα για κυκλικές ή ορθογώνιες περιοχές.

Για να φέρουμε λοιπόν όλες τις οντότητες που εντοπίζονται σε ένα πολύγωνο ακολουθούμε τα εξής βήματα.

1. «Ζητάμε» από το Solr να μας φέρει τις οντότητες που ανήκουν στην ορθογώνια περιοχή που περικλείει το πολύγωνο
2. Απορρίπτουμε τις φωτογραφίες που βρίσκονται εκτός του πολυγώνου

Για να εξετάσουμε αν ένα σημείο ανήκει η όχι στο πολύγωνο που ορίζει ένα σύνολο από σημεία χρησιμοποιούμε τον PNPOLY αλγόριθμο. (Franklin 2014)

4.6.1 Ο αλγόριθμος PNPOLY σε Java

Ουσιαστικά χρησιμοποιήσαμε τον αλγόριθμο PNPOLY με μια μικρή παραλλαγή ώστε να συμπεριλαμβάνουμε σημεία που βρίσκονται στα σύνορα με το πολύγωνο και να συμπεριλάβουμε σημεία που θα αγνοούνταν λόγω σφάλματος κινητής υποδιαστολής.

```
int size = polygon.size(), i, j;
boolean c = false;
double error = 0.00000001, x = p.getLng(), y = p.getLat();
for (i = 0, j = size - 1; i < size; j = i++) {
   MapPoint p1 = polygon.get(j), p2 = polygon.get(i);
   if (p1.equals(p) || p2.equals(p)) {
      return true;
   }
   double x1 = p1.getLng(), y1 = p1.getLat(), x2 = p2.getLng(), y2 = p2.getLat();
   //return true for borders
   if (x1 == x2 && x == x1) {
      return true;
   }
   if (x1 != x2 && abs(y - y1 - (x - x1) * (y2 - y1) / (x2 - x1)) <= error) {
      return true;
   }
   if (((x1 > x) != (x2 > x)) &&
         (y - y1 < (x - x1) * (y2 - y1) / (x2 - x1))) {
      c = !c;
   }
}
return c;
```

Κώδικας 4.5 Αλγόριθμος ελέγχου αν ένα σημείο ανήκει σε ένα πολύγωνο

4.6.2 Συσχέτιση Οδών με Σημεία Ενδιαφέροντος και Φωτογραφίες

Στην ενότητα αυτή θα εξετάσουμε πως βρίσκουμε οντότητες που βρίσκονται κοντά σε μία οδό, απέχουν δηλαδή από την οδό το πολύ d.

Όπως και με τα πολύγωνα, το Solr, χωρίς κάποια επιπλέον προσθήκη, δεν υποστηρίζει την εύρεση οντοτήτων τριγύρω από μια γραμμική οντότητα όπως μία οδό.

Για να φέρουμε λοιπόν τις οντότητες που βρίσκονται κοντά σε μια οδό, ακολουθούμε παρόμοια διαδικασία με αυτή για τις περιοχές. Δηλαδή:

1. Υπολογίζουμε ένα ορθογώνιο που περικλείει κάθε σημείο που ορίζει την οδό και προσθέτουμε σε αυτό ένα buffer μεγέθους d.
2. «Ζητάμε» από το Solr να μας φέρει όλες τις οντότητες μέσα στο πολύγωνο αυτό
3. Φιλτράρουμε τις οντότητες με βάση την απόστασή τους από την οδό

Για να βρούμε την απόσταση μιας οντότητας από την οδό, χρησιμοποιούμε την έτοιμη βιβλιοθήκη JTS που παρέχει έτοιμες χωρικές συναρτήσεις (JTS 2015). Συγκεκριμένα, η βιβλιοθήκη ορίζει μια συνάρτηση που επιστρέφει την απόσταση ενός σημείου από ένα ευθύγραμμο τμήμα. Επειδή ωστόσο μια οδός είναι ένα σύνολο από ευθύγραμμα τμήματα, θα ισχύει ότι η απόσταση ενός σημείου από την οδό ισούται με την ελάχιστη απόσταση του σημείου από τα ευθύγραμμα τμήματα που ορίζουν την οδό.

Λεπτομέρειες υλοποίησης της απόστασης σημείου από ευθύγραμμο τμήμα μπορούν να αναζητηθούν στην ιστοσελίδα της βιβλιοθήκης. Η βασική ιδέα στην οποία βασίζεται ωστόσο ο αλγόριθμος είναι η εξής:

Έστω AB το ευθύγραμμο τμήμα και C το σημείο που εξετάζουμε.

1. Αν το C βρίσκεται εντός της επιφάνειας που ορίζουν οι παράλληλες ευθείες ε1, ε2 που είναι κάθετες στο AB και περνούν από τα A και B αντίστοιχα τότε η απόσταση του C ισούται με την κάθετη απόσταση του C από την ευθεία που ορίζει το AB
2. Αν το C βρίσκεται εκτός της επιφάνειας που ορίστηκε πρωτύτερα, τότε:
    a. Αν το C βρίσκεται από την πλευρά του A, η απόσταση του είναι η απόσταση μεταξύ των A και C
    b. Αν το C βρίσκεται από την πλευρά του B, η απόστασή του είναι η απόσταση μεταξύ των B και C

5

Τεχνική περιγραφή της διεπαφής υπηρεσιών

5.1 Τεχνολογίες και εργαλεία

Όπως έχει ήδη αναφερθεί, για την υλοποίηση της διεπαφής υπηρεσιών επιλέξαμε να χρησιμοποιήσουμε τη γλώσσα προγραμματισμού Java.

Σε αυτή την ενότητα θα αναφερθούμε στα βασικά εργαλεία που χρησιμοποιήσαμε για την υλοποίηση της εφαρμογής.

5.1.1 Maven για διαχείριση εξαρτήσεων και "χτίσιμο" της εφαρμογής

Το Maven (Maven 2015) είναι ένα εργαλείο αυτοματοποίησης της διαδικασίας του «χτισίματος» εφαρμογών που είναι γραμμένες σε Java. Ο ρόλος που εξυπηρετεί το Maven για μία εφαρμογή είναι διττός. Συγκεκριμένα, περιγράφει αφενός τον τρόπο με τον οποίο χτίζεται η εφαρμογή και αφετέρου ορίζει και διαχειρίζεται βιβλιοθήκες, εργαλεία ή υπάρχων κώδικα που χρησιμοποιεί η εφαρμογή και από τα οποία εξαρτάται.

Η λειτουργία του βασίζεται σε ένα XML αρχείο με όνομα «pom.xml» το οποίο περιγράφει τα παραπάνω και τροποποιεί ενδεχομένως τον τρόπο με τον οποίο θα χτιστεί η εφαρμογή. Αυτό το XML αρχείο ουσιαστικά «τροποποιεί» ένα σύνολο από προεπιλεγμένες ρυθμίσεις που καθορίζει το Maven, ή προσθέτει επιπλέον πληροφορία, όπως για παράδειγμα εξαρτήσεις.

Ο λόγος που διαλέξαμε να χρησιμοποιήσουμε Maven είναι γιατί καθιστά ταχύτατη την ανάπτυξη και εγκατάσταση μιας Java εφαρμογής αλλά και την προσθήκη νέων βιβλιοθηκών που θα θέλαμε να χρησιμοποιήσουμε. Πλέον δε χρειάζεται να τοποθετούμε τις εξαρτήσεις στον φάκελο «libraries» της εφαρμογής χειροκίνητα. Απλά αναφέρουμε μία εξάρτηση στο XML αρχείο και το Maven αναλαμβάνει να την «κατεβάσει» από μία κεντρική συλλογή βιβλιοθηκών και να την τοποθετήσει στο κατάλληλο σημείο. Έπειτα, κατά το «χτίσιμο» της εφαρμογής, αναλαμβάνει να συγκεντρώσει όλες τις εξαρτήσεις, να κάνει compile τον κωδικά μας και να «πακετάρει» όλα τα αρχεία που χρειάζονται σε ένα αρχείο τύπου WAR το οποίο είναι έτοιμο να εγκατασταθεί σε κάποιον εξυπηρετητή (Tomcat στην περίπτωσή μας).

```
<dependency>
   <groupId>org.springframework</groupId>
   <artifactId>spring-webmvc</artifactId>
   <version>${spring.version}</version>
</dependency>
```

Κώδικας 5.1 Παράδειγμα δήλωσης εξάρτησης στο pom.xml αρχείο του Maven

Επιπρόσθετα, ένα μεγάλο πλεονέκτημα του Maven είναι ότι είναι σχεδιασμένο να δέχεται προσθήκες που επεκτείνουν τη βασική λειτουργία του. Μία από τις βασικότερες προσθήκες που χρησιμοποιήσαμε κατά κόρον είναι η προσθήκη «maven-surefire», η οποία κατά το χτίσιμο της εφαρμογής μας δίνει τη δυνατότητα να εκτελούμε ένα σύνολο από τεστ που επικυρώνουν την ορθή λειτουργία της εφαρμογής. Αν κάποια από τα τεστ αποτύχουν τότε η διαδικασία του χτισίματος ματαιώνεται και δημιουργείται μία κατάλληλη αναφορά για τον προγραμματιστή.

5.1.2 Spring και SpringMVC για την υλοποίηση της REST διεπαφής υπηρεσιών

Το Spring είναι μια σουίτα βιβλιοθηκών που φέρουν στον πυρήνα τους τη λογική της Εισαγωγής Εξαρτήσεων με βάση τη γνωστή πρακτική (pattern) ανάπτυξης λογισμικού με το όνομα «Inversion of Control». Η σουίτα περιέχει πολλές επιμέρους βιβλιοθήκες που είναι χρήσιμες κατά την ανάπτυξη εφαρμογών, ωστόσο εμείς θα επικεντρωθούμε στην επέκταση με όνομα SpringMVC.

Το SpringMVC είναι μία πλήρης σουίτα σχεδίασης εφαρμογών Web και βασίζεται στην πρακτική (pattern) Model View Controller (MVC). Βασική ιδέα της πρακτικής αυτής είναι ο διαχωρισμός των δεδομένων (Model) από τη λογική (Controller) και το πως παρουσιάζονται (View). Ουσιαστικά, το κομμάτι της λογικής δημιουργεί τα δεδομένα (αντικείμενα που ορίζουν το μοντέλο της εφαρμογής) και στη συνέχεια τα τροφοδοτεί στο στρώμα της παρουσίασης (View). Η παρουσίαση των δεδομένων δε χρειάζεται να είναι απαραίτητα

γραφική (η οποία στο spring γίνεται μέσω σελίδων jsp) αλλά μπορεί να είναι και ένα αρχείο με συγκεκριμένη μορφή όπως για παράδειγμα XML ή JSON.

Ο λόγος που διαλέξαμε τη χρήση του SpringMVC για το σχεδιασμό του API, είναι ότι διαθέτει πολύ ισχυρά εργαλεία σειριοποίησης αντικειμένων Java σε JSON και αντίστροφα αποσειριοποίησης τους. Αυτό σημαίνει ότι μπορούμε να επικεντρωθούμε στη δημιουργία των δεδομένων υπό τη μορφή απλών αντικειμένων σε Java, και η σειριοποίησή τους θα γίνει αυτόματα από το SpringMVC.

5.1.2.1 Προσθήκη και παραμετροποίηση του SpringMVC

Για τη σωστή εκκίνηση της web εφαρμογής μας ακολουθούμε την περιγραφή έκδοσης 3 του Servlet API (Oracle 2009) η οποία περιγράφει πώς μπορεί να γίνει η παραμετροποίηση ενός Servlet όπως η εφαρμογή μας χωρίς τη χρήση ενός web.xml αρχείου (που ήταν αναγκαίο πριν την έκδοση 3).

Το εισαγωγικό σημείο κατά την εκκίνηση της εφαρμογής παρουσιάζεται στο ακόλουθο παράρτημα:

```
public class AppInitializer implements WebApplicationInitializer {
    @Override
    public void onStartup(ServletContext servletContext) throws ServletException {
        WebApplicationContext context = getContext();
        servletContext.addListener(new ContextLoaderListener(context));
        ServletRegistration.Dynamic dispatcher = servletContext.addServlet("DispatcherServlet",
                                                 new DispatcherServlet(context));
        dispatcher.setLoadOnStartup(1);
        dispatcher.addMapping("/");
    }
    private AnnotationConfigWebApplicationContext getContext() {
        AnnotationConfigWebApplicationContext context = new AnnotationConfigWebApplicationContext();
        context.setConfigLocation("gr.athena_innovation.imis.citytrack.config");
        return context;
    }
}
```

Κώδικας 5.2 Ρύθμιση ενός spring context

Στο σημείο αυτό δεν έχουμε ενεργοποιήσει ακόμη το *SpringMVC*. Αυτός ο κώδικας δημιουργεί έναν "Dispacher" ο οποίος «ακούει» για αιτήματα στο μονοπάτι «/» και θα προσπαθήσει να τα εξυπηρετήσει (ακόμη δεν έχουμε δείξει πως). Ένα σημαντικό κομμάτι του κώδικα αυτού είναι το σημείο όπου θέτουμε ένα *ConfigLocation* στο *Context* του *Spring*. Αυτό είναι το σημείο από το οποίο το *Spring* θα προσπαθήσει να βρει κλάσεις που εξυπηρετούν συγκεκριμένα μονοπάτια (**gr.athena_innovation.imis.citytrack.config**) και θα τις αρχικοποιήσει.

Για να ενεργοποιήσουμε το *SpringMVC* και να το παραμετροποιήσουμε πρέπει να τοποθετήσουμε τον ακόλουθο κώδικα στην τοποθεσία που θέσαμε στο *ConfigLocation*

```
@EnableWebMvc
@Configuration
public class WebMvcConfig extends WebMvcConfigurerAdapter {
   @Override
   public void addResourceHandlers(ResourceHandlerRegistry registry) {
      registry.addResourceHandler("/css/**").addResourceLocations("/css/").setCachePeriod(31556926);
      registry.addResourceHandler("/img/**").addResourceLocations("/img/").setCachePeriod(31556926);
      registry.addResourceHandler("/js/**").addResourceLocations("/js/").setCachePeriod(31556926);
   }

   @Override
   public void configureDefaultServletHandling(DefaultServletHandlerConfigurer configurer) {
      configurer.enable();
   }

   @Bean
   public InternalResourceViewResolver getInternalResourceViewResolver() {
      InternalResourceViewResolver resolver = new InternalResourceViewResolver();
      resolver.setPrefix("/WEB-INF/pages/");
      resolver.setSuffix(".jsp");
      return resolver;
   }
}
```

Κώδικας 5.3 Ενεργοποίηση και παραμετροποίηση του Spring MVC

Το μόνο σημαντικό κομμάτι που πρέπει να προσέξει κανείς εδώ είναι οι επισημάνσεις *@Configuration και @EnableWebMvc* στην κορυφή του κώδικα. Η πρώτη ενημερώνει το Spring ότι η κλάση αυτή ρυθμίζει κάτι και πρέπει να ερευνηθεί, και η άλλη ενεργοποιεί το MVC κομμάτι του Spring για αυτή την εφαρμογή. Οι υπόλοιπες μέθοδοι ορίζουν αντιστοιχίσεις URL μονοπατιών με τοποθεσίες στις οποίες πρέπει να αναζητηθούν τα εκάστοτε αρχεία.

Τέλος πρέπει να υποδείξουμε στο *Spring* πού μπορεί να εντοπίσει κλάσεις που εξυπηρετούν κάποιο URL. Οι κλάσεις αυτές στο Spring ονομάζονται Controllers και είναι τα σημεία εισόδου όσον αφορά στην εξυπηρέτηση αιτημάτων όπως αυτά προκύπτουν κατά τη διάρκεια της εφαρμογής. Η εν λόγω υπόδειξη στο Spring γίνεται με τον ακόλουθο τρόπο:

```
@Configuration
@ComponentScan(basePackages = {
      "gr.athena_innovation.imis.citytrack.rest"
})
public class RootConfig {}
```

Κώδικας 5.4 Υπόδειξη Εντοπισμού Controllers

Ουσιαστικά λέμε στο Spring να ψάξει για controllers στο package:

(gr.athena_innovation.imis.citytrack.rest)

5.1.2.2 Ορισμός ενός Spring Controller

Τέλος ας δούμε ένα απλό παράδειγμα ενός controller που βρίσκεται στο *package* που ορίσαμε προηγουμένως.

```
@RequestMapping("/")
@Component
public class AreaResourceController {
   private final AreaRepo areaRepo;

   @Autowired
   public AreaResourceController(AreaRepo areaRepo) {
      this.areaRepo = areaRepo;
   }

   @RequestMapping(value = "/areas", produces =APPLICATION_JSON_VALUE, method = GET)
   @ResponseBody
   public List<Area> getAvailableAreas() throws DataAccessException {
      return areaRepo.getAreas();
   }

   @ExceptionHandler({ DataAccessException.class, Exception.class })
   public ResponseEntity<String> internalServerErrorHandler(Exception e) {
      HttpHeaders headers = new HttpHeaders();
      e.printStackTrace();
      return new ResponseEntity<>(e.getMessage(), headers, HttpStatus.INTERNAL_SERVER_ERROR);
   }
}
```

Κώδικας 5.5 Παράδειγμα ενός Controller κατά SpringMVC

Οι ενδείξεις @Component και @RequestMapping("/") σηματοδοτούν στο Spring ότι αυτή η κλάση είναι ένας Controller και ότι όλα τα μονοπάτια τα οποία θα αναφερθούν στη συνέχεια ακολουθούν αυτό που ορίστηκε από την τιμή της ένδειξης @RequestMapping, δηλαδή «/».

Επιπρόσθετα, ορίζεται ένα μονοπάτι «/areas» το οποίο εξυπηρετείται από τη μέθοδο getAvailableAreas και παράγει JSON, ενώ ανταποκρίνεται μόνο σε αιτήματα τύπου GET. Η ένδειξη @ResponseBody, «ζητά» από το Spring να σειριοποιήσει το αντικείμενο που επιστρέφει η μέθοδος και να προωθήσει τη σειριοποιημένη έκδοση του αντικειμένου αυτού στον αιτούντα. Η μορφή στην οποία θα σειριοποιηθούν τα αντικείμενα καθορίζεται με βάση το πρότυπο που αναφέρθηκε στην ένδειξη @RequestMapping, δηλαδή στην περίπτωση αυτή, JSON.

Τέλος η ένδειξη @ExceptionHandler σηματοδοτεί μια μέθοδο που εκτελείται κάθε φορά που υπάρχει κάποιο Exception με βάση τις κλάσεις που αναφέρονται (DataAccessException.**class**, Exception.**class**).

5.1.3 TestNG και Test Driven Development

Για το σχεδιασμό και την ανάπτυξη της εφαρμογής, χρησιμοποιήθηκε η τεχνοτροπία ανάπτυξης λογισμικού που ονομάζεται Test Driven Development. Η τεχνοτροπία αυτή ορίζει ένα τρόπο προσέγγισης της ανάπτυξης λογισμικού κατά τον οποίο τα Τεστ είναι αυτά που καθοδηγούν την ανάπτυξη και κατ' επέκταση το σχεδιασμό της εφαρμογής. Σύμφωνα με την τεχνοτροπία αυτή, καμία γραμμή κώδικα δεν πρέπει να προστίθεται αν δεν υπάρχει κάποιο Test που την καθιστά αναγκαία. Μόνη εξαίρεση στο παραπάνω είναι η αλλαγή να στοχεύει στη βελτιστοποίηση και όχι στην προσθήκη λειτουργικότητας.

Το πλεονέκτημα αυτής της μεθοδολογίας είναι ότι σε κάθε βήμα υπάρχει μία πλήρης σουίτα από τεστ τα οποία περιγράφουν τον κώδικα και είναι σε θέση να επικυρώσουν τη λειτουργία του. Έτσι, όταν κανείς κάνει αλλαγές ή προσθήκες, νιώθει την ασφάλεια ότι αν μέσω αυτών των αλλαγών κάτι που δούλευε δε δουλεύει πια, θα φανεί από τα τεστ τα οποία θα αποτύχουν. Επίσης, επειδή καμία γραμμή κώδικα δε γράφεται χωρίς να είναι αναγκαίο, ξεφεύγουμε από την παγίδα της υλοποίησης περισσότερης λειτουργικότητας από όση είναι πραγματικά αναγκαία.

Για να μπορέσουμε να εφαρμόσουμε την τεχνοτροπία αυτή χρειαζόμαστε κάποια εργαλεία που να μας επιτρέπουν τόσο τη συγγραφή όσο και την εκτέλεση των τεστ. Μάλιστα, η εκτέλεση πρέπει να είναι εύκολη και γρήγορη τόσο κατά τη συγγραφή ώστε να έχουμε άμεση ανατροφοδότηση για τις αλλαγές που κάνουμε, όσο και κατά την εγκατάσταση του κώδικα ώστε να ειδοποιούμαστε έγκαιρα αν ξεχάσαμε κάποια λειτουργικότητα ή κάτι πήγε στραβά.

5.1.3.1 TestNG

Το TestNG είναι ένα πολύ καλό εργαλείο για αυτές τις ανάγκες. Το TestNG στην ουσία ορίζει δύο πράγματα. Δηλαδή, ένα γρήγορο και ευέλικτο τρόπο ορισμού και εκτέλεσης των τεστ και στους δύο χρόνους που αναφέραμε προηγουμένως (ανάπτυξη, εγκατάσταση), και μία βιβλιοθήκη επικύρωσης που μας επιτρέπει εύκολα να ελέγξουμε προτάσεις ως προς την αλήθεια τους.

Μάλιστα, το TestNG ενσωματώνεται άψογα στα περισσότερα IDEs όπως το Eclipse, το NetBeans ή το IntelliJ, καθιστώντας την άμεση εκτέλεση μεμονωμένων τεστ διαθέσιμη με ένα μόνο κλικ.

Ένα παράδειγμα ενός Test κατά TestNG είναι το ακόλουθο:

```
@Test
public void testAdditionWorks() {
   int a = 1;
   int b = 2;
   assertEquals(a + b, 3);
}
```

Κώδικας 5.6 Παράδειγμα ενός Test κατά TestNG

Παρότι το τεστ αυτό είναι τετριμμένο, δείχνει δύο πράγματα. Ότι δηλαδή (1) ο ορισμός ενός test είναι πολύ εύκολος και (2) η επικύρωση του αποτελέσματος μπορεί να γίνει συνήθως σε μία γραμμή.

Αν για κάποιο λόγο η επικύρωση αποτύχει, το τεστ αποτυγχάνει και ο προγραμματιστής ειδοποιείται αναλόγως.

5.1.4 Swagger για την αυτόματη περιγραφή της διεπαφής

Η περιγραφή της διεπαφής υπηρεσιών μέσω κάποιας σελίδας βοήθειας είναι ίσως μία από τις πιο ανιαρές ενέργειες κατά την ανάπτυξη της. Ιδιαίτερα αν αυτή αλλάζει συχνά. Αποτελεί μάλιστα σύνηθες φαινόμενο η περιγραφή μιας διεπαφής να μην ανανεώνεται μαζί με τη διεπαφή που περιγράφει, με αποτέλεσμα να είναι λανθασμένη για μεγάλα χρονικά διαστήματα.

Για να αποφύγουμε κάτι τέτοιο, χρησιμοποιήσαμε ένα εργαλείο που ενσωματώνεται πολύ εύκολα με το Spring και δημιουργεί αυτόματα ένα UI που περιγράφει τη διεπαφή υπηρεσιών που υλοποιείται από τον κώδικα. Το όνομα του εργαλείου αυτού είναι Swagger (Swagger 2015), και η επέκταση που υλοποιεί το UI ονομάζεται SwaggerUI.

Η παραμετροποίηση του Swagger είναι πολύ εύκολη. Συγκεκριμένα, αρκεί να τοποθετήσουμε την ακόλουθη κλάση σε μια τοποθεσία που είναι αναζητήσιμη από το Spring (βλέπε 5.1.2.1)

```
@Configuration
@EnableSwagger2
public class SwaggerConfig {}
```

Στη συνέχεια μπορούμε να δίνουμε οδηγίες στο Swagger σε σχέση με το πώς να περιγράψει καλύτερα τη διεπαφή μας.

Για παράδειγμα, στον ακόλουθο κώδικα βλέπουμε πως μπορούμε να ενσωματώσουμε οδηγίες του Swagger στον Controller της ενότητας 5.1.2.2. Οι γραμμές κώδικα που παρατίθενται σε γκρίζο χρώμα παρατίθενται απλά για να επισημάνουν τη σχετική τοποθεσία τον οδηγιών.

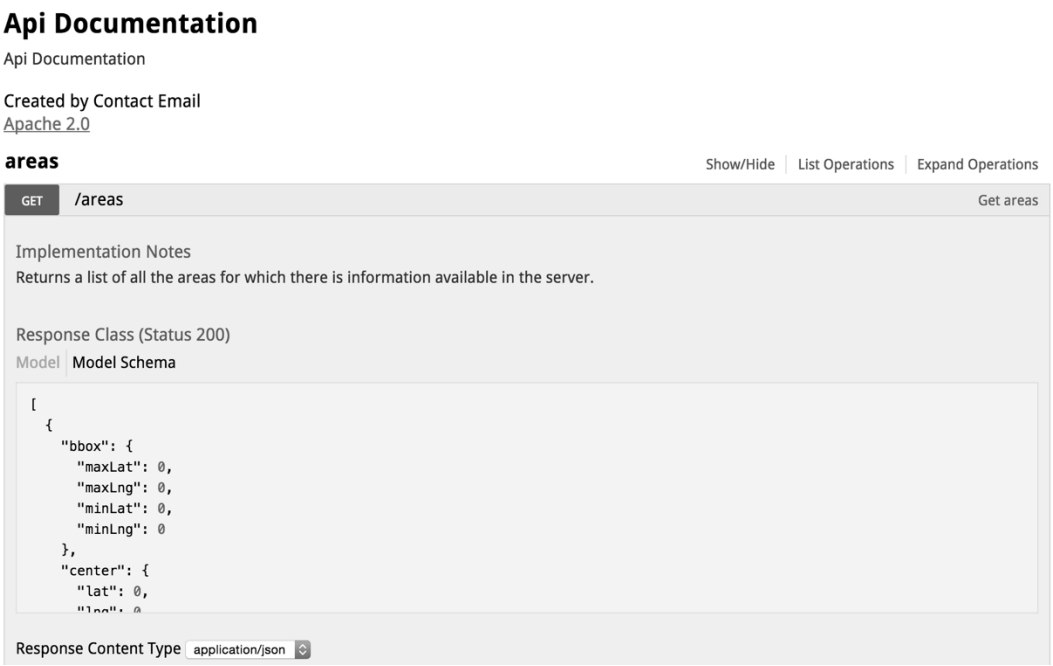
```
@RequestMapping("/")
@Component
@Api(value = "Areas")
public class AreaResourceController {
   private final AreaRepo areaRepo;

   @Autowired
   public AreaResourceController(AreaRepo areaRepo) {
      this.areaRepo = areaRepo;
   }

   @RequestMapping(value = "/areas", produces =APPLICATION_JSON_VALUE, method = GET)
   @ResponseBody
   @ApiOperation(value = "Get areas",
      notes = "Returns a list of all the areas for which there is information available in the server.")
   public List<Area> getAvailableAreas() throws DataAccessException {
      return areaRepo.getAreas();
   }
...
}
```

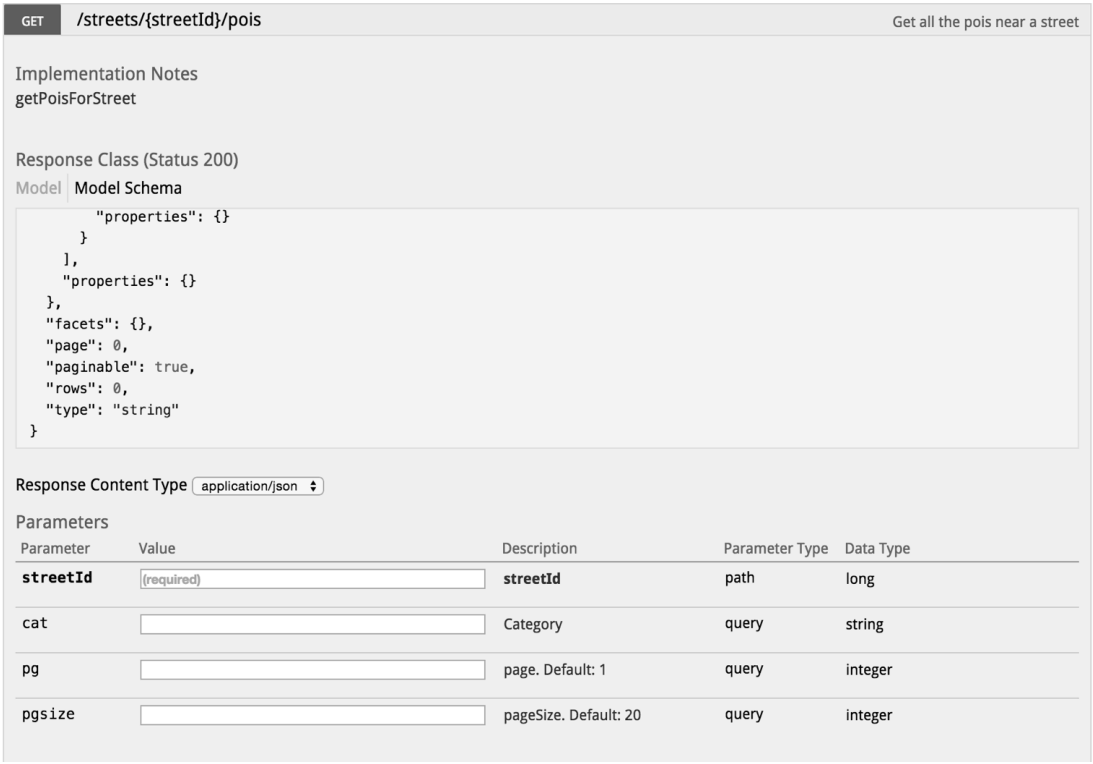
Κώδικας 5.7 Controller με οδηγίες περιγραφής κατά Swagger

Στην εικόνα που ακολουθεί παραθέτουμε το αποτέλεσμα που προκύπτει από την περιγραφή αυτή στο Swagger UI.



Εικόνα 5.1 Αποτέλεσμα της περιγραφής της διεπαφής με Swagger και Swagger UI

Στη συνέχεια δείχνουμε ένα παράδειγμα ενός σημείου εισόδου της διεπαφής το οποίο έχει περισσότερα στοιχεία στην περιγραφή του και παρατίθεται εδώ ως αναφορά σε σχέση με το τι μπορεί να περιέχει η περιγραφή στο SwaggerUI:



Εικόνα 5.2 Ακόμη μια περιγραφή διεπαφής από το SwaggerUI. Εδώ βλέπουμε και περιγραφή σχετικά με τις παραμέτρους εισόδου

Όπως βλέπουμε, μέσω του Swagger, μπορούμε να περιγράψουμε το σημείο εισόδου, αλλά και να παρέχουμε περισσότερες λεπτομέρειες για αυτό.

Αναφορικά, μπορούμε να περιγράψουμε,

- Το σημείο εισόδου γενικότερα
- Τις παραμέτρους που απαιτούνται ως είσοδος στη διεπαφή
- Τη μορφή που έχουν τα δεδομένα που επιστρέφονται (πχ JSON, XML κλπ)
- Τη δομή των δεδομένων που επιστρέφονται
- Τα διάφορα είδη των κωδικών των απαντήσεων HTTP που επιστρέφονται (400, 404 κλπ

5.1.5 Χρήση Git για την αποθήκευση του κώδικα και τη διατήρηση του ιστορικού αλλαγών

Ένα από τα μεγαλύτερα προβλήματα που αντιμετωπίζουν οι ομάδες ανάπτυξης λογισμικού είναι η ανεύρεση ενός τρόπου να δουλεύουν παράλληλα στην ίδια βάση κώδικα (ιδανικά σε διαφορετικά σημεία αυτού) και να μπορούν εύκολα στο τέλος να ενοποιούν τις αλλαγές τους. Ευτυχώς πλέον, υπάρχει μία πληθώρα εργαλείων που λύνουν ακριβώς αυτό το πρόβλημα, με πιο γνωστό αυτή τη στιγμή να είναι το Git (Git 2015) .

Στο Git, υπάρχουν δύο είδη «αποθηκών» κώδικα (repository): (1) οι τοπικές (local repositories), οι οποίες βρίσκονται στον υπολογιστή του κάθε προγραμματιστή, και (2) οι απομακρυσμένες (remote repositories) οι οποίες βρίσκονται σε κάποιον εξυπηρετητή Web. Μία συνήθης ροή εργασίας με Git για μία ομάδα, περιλαμβάνει μία απομακρυσμένη αποθήκη κώδικα και μία ή περισσότερες τοπικές αποθήκες κώδικα, αποθηκευμένες στους προσωπικούς υπολογιστές των προγραμματιστών. Η απομακρυσμένη αποθήκη είναι το σημείο στο οποίο ενώνονται οι αλλαγές που κάνουν οι προγραμματιστές στον κωδικά που έχουν αποθηκευμένο στα τοπικά τους αντίγραφα.

Το Git, λειτουργεί με βάση σύνολα αλλαγών που ονομάζονται «commits» και «ζουν» στο τοπικό μηχάνημα του προγραμματιστή. Ένα commit περιγράφει πώς από μια δεδομένη μορφή του κώδικα μπορούμε να μεταβούμε στην επόμενη χρησιμοποιώντας τη λογική των «δέλτα» διαφορών. Έτσι, όποτε ο προγραμματιστής επιθυμεί, μπορεί να προωθήσει ένα σύνολο από commits από το τοπικό του αντίγραφο στην απομακρυσμένη αποθήκη κώδικα.

Ένα τέτοιο εργαλείο, βέβαια, δεν είναι χρήσιμο μόνο σε ομάδες αλλά και όταν δουλεύει κανείς σε μία βάση κώδικα μόνος του. Μερικά από τα πλεονεκτήματα που προσφέρει είναι τα ακόλουθα:

- Διατηρεί το ιστορικό των αλλαγών που έχουν γίνει στον κώδικα
- Ο κώδικας δε «ζει» μόνο στον προσωπικό υπολογιστή του προγραμματιστή
- Επειδή ο κώδικας αποθηκεύεται σε κάποια υπηρεσία, είναι ταυτόχρονα διαθέσιμος από πολλά άτομα που έχουν πρόσβαση σε αυτόν
- Καθιστά δυνατή τη χρήση συστημάτων που ανιχνεύουν αλλαγές στον κώδικα και εγκαθιστούν αυτόματα τη νέα έκδοση του όπου χρειάζεται.

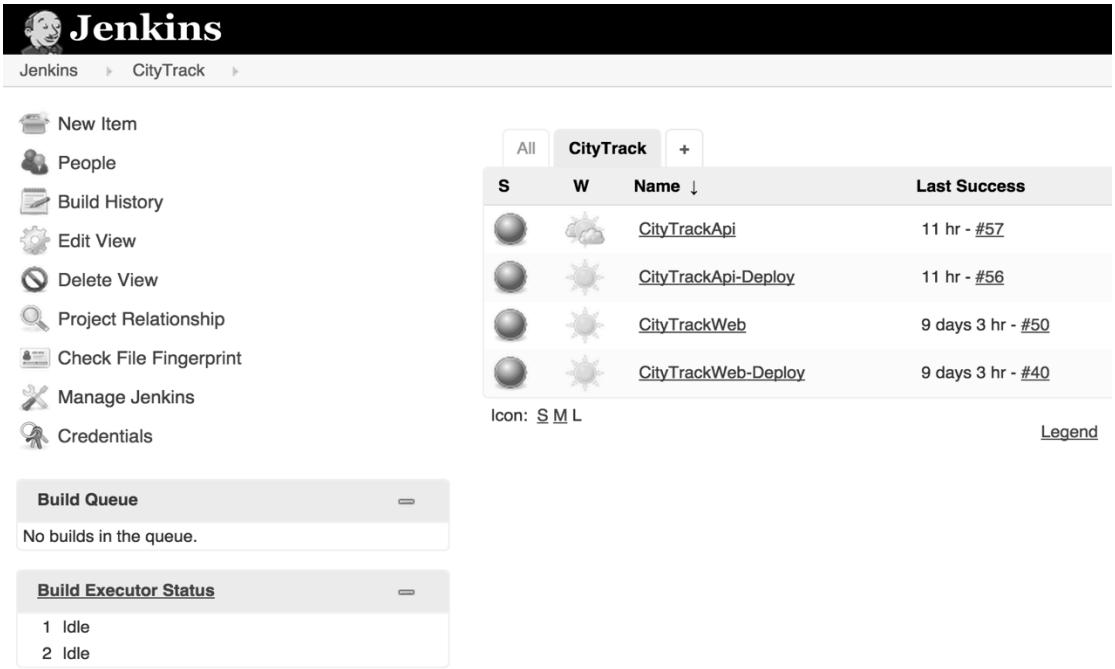
5.1.6 Χρήση Jenkins για την αυτόματη εγκατάσταση της εφαρμογής

Κατά την ανάπτυξη μιας εφαρμογής μια από τις πιο χρονοβόρες διαδικασίες είναι η συνεχής εγκατάσταση της νέας έκδοσης της εφαρμογής για να ελεγχθεί αν όντως λειτουργεί. Για την αντιμετώπιση αυτού του προβλήματος υπάρχει σήμερα μία πληθώρα από συστήματα που μπορούν χρησιμοποιώντας τον κώδικα της εφαρμογής και ένα σύνολο από ρυθμίσεις να την εγκαθιστούν αυτόματα όπου είναι αναγκαίο. Με αυτόν τον τρόπο, μπορούμε να ρυθμίσουμε το σύστημα αυτό μονάχα μια φορά και στη συνέχεια να αναλαμβάνει αυτό να εγκαθιστά την εφαρμογή μας στην κατάλληλη τοποθεσία κάθε φορά που υπάρχει μια νέα έκδοση.

Τα συστήματα αυτά ονομάζονται Συστήματα Συνεχούς Ενοποίησης (Continuous Integration Systems) ή απλά CIs. Το CI που επιλέξαμε ονομάζεται Jenkins καθώς φέρει πολλές λειτουργίες είναι αξιόπιστο και χρησιμοποιείται ευρέως σήμερα στη βιομηχανία.

Ουσιαστικά το σύστημα Jenkins, μας δίνει τη δυνατότητα να ορίσουμε ένα σύνολο από «εργασίες» οι οποίες εκτελούνται με βάση κάποια κριτήρια ή ανά τακτά χρονικά διαστήματα. Στην περίπτωσή μας, οι διεργασίες αυτές πυροδοτούνται κάθε φορά που ο Jenkins ανιχνεύει κάποια αλλαγή στον κώδικά που βρίσκεται στην «απομακρυσμένη αποθήκη κώδικα» (remote code repository). Όταν ανιχνευθεί κάποια αλλαγή, ο Jenkins ακολουθώντας ένα σύνολο από οδηγίες, οι οποίες ρυθμίζονται από εμάς για κάθε εργασία ξεχωριστά, θα «χτίσει» και θα εγκαταστήσει την εφαρμογή.

Στη συνέχεια παρατίθεται μια εικόνα που παρουσιάζει το γραφικό περιβάλλον του Jenkins:



5.2 Εγκατάσταση και παραμετροποίηση της διεπαφής υπηρεσιών

Όπως εξηγήσαμε στην ενότητα 5.1.6, χρησιμοποιήσαμε το σύστημα CI Jenkins για την εγκατάσταση της εφαρμογής, αποφεύγοντας έτσι τη «χειροκίνητη» διαδικασία. Ωστόσο, για να είναι εφικτή η αυτοματοποίηση της εγκατάστασης χρειάστηκε να γίνουν κάποια βήματα από ρυθμίσεις τόσο στα υπολογιστικά συστήματα στα οποία εγκαθίστανται οι εφαρμογές όσο και στο ίδιο το CI.

5.2.1 Διαπιστευτήρια σύνδεσης στη Βάση Δεδομένων και το Solr

Για να μπορεί να συνδεθεί η εφαρμογή μας να συνδεθεί στη βάση δεδομένων και τον Solr χρειάζεται με κάποιον τρόπο να της «δοθούν» τα διαπιστευτήρια που χρειάζονται για την εκάστοτε σύνδεση. Συγκεκριμένα, για να μπορεί να συνδεθεί στα ανωτέρω συστήματα χρειάζεται τις ακόλουθες πληροφορίες για κάθε ένα από αυτά:

- Διεύθυνση Εξυπηρετητή (Server Host)
- Όνομα Χρήστη
- Κωδικός
- Αριθμός Θήρας Σύνδεσης
- Όνομα Βάσης (Συλλογής για τον Solr)

Μόνο οι διαχειριστές του συστήματος πρέπει να γνωρίζουν τα πιστοποιητικά αυτά, και μια κοινή πρακτική με την οποία πραγματοποιείται αυτό, είναι με το να τα παρέχουν στην εφαρμογή είτε μέσω μιας υπηρεσίας ή μέσω ενός αρχείου που τοποθετείται στο μηχάνημα του εξυπηρετητή με περιορισμένη πρόσβαση. Για την εν λόγω εφαρμογή επιλέξαμε την προσέγγιση με τη χρήση ενός αρχείου. Υπάρχει δηλαδή μία ευρέως γνωστή τοποθεσία όπου η εφαρμογή μας αναζητά το αρχείο αυτό, το διαβάζει και εξάγει τις ρυθμίσεις που χρειάζεται.

Το αρχείο αναζητείται στην ακόλουθη τοποθεσία:

/var/citytrack-api/data/config/connection.properties

Η ευθύνη για την τοποθέτηση αυτού του αρχείου στη σωστή τοποθεσία ανήκει στους διαχειριστές. Έτσι αποφεύγουμε να τοποθετήσουμε τα διαπιστευτήρια αυτά μαζί με τον κώδικα.

Το format του αρχείου πρέπει να είναι επίσης μέρος του συμβολαίου και πρέπει να είναι το ακόλουθο:

```
 db.host=83.212.123.13
db.database=geostream_test
db.port=5432
db.user=****
db.password=*****

solr.host=http:///83.212.123.13
solr.port=8983
solr.collection=geostream_test
solr.user=*****
solr.password=*****
```

5.2.2 Προετοιμασία μηχανήματος εξυπηρετητή για την αυτόματη εγκατάσταση

Σκοπός μας για την αυτόματη εγκατάσταση της διεπαφής χρήστη είναι να μπορεί το CI μας να αντιγράφει το αρχείο WAR που περιέχει την εφαρμογή στον κατάλληλο φάκελο του Tomcat, ώστε αυτός με τη σειρά του να εκκινήσει τη διαδικασία για την εγκατάσταση της νέας έκδοσης.

Για να το πετύχουμε αυτό, πρέπει αρχικά να μπορεί το CI να συνδεθεί στο μηχάνημα εξυπηρετητή. Για το σκοπό αυτό, δημιουργήσαμε ένα ζεύγος από public/private κλειδιά SSH στο μηχάνημα του Jenkins και τα εγκαταστήσαμε ως έμπιστα κλειδιά στο μηχάνημα του εξυπηρετητή.

Έπειτα, δε μπορούμε να γράψουμε στο φάκελο του Tomcat που περιέχει της εφαρμογές χωρίς προνόμια διαχειριστή, και συνάμα δε θέλουμε το CI συνδέεται με προνόμια διαχειριστή. Για το λόγο αυτό δημιουργήσαμε ένα τρίτο φάκελο στο οποίο έχει πρόσβαση το CI, και δημιουργήσαμε ένα συμβολικό σύνδεσμο (symlink) στο φάκελο του Tomcat που «δείχνει» στο WAR αρχείο που θα αντιγράφει το CI. Η σειρά εντολών που πραγματοποιεί αυτά τα βήματα ρυθμίσεων για την εφαρμογή μας είναι η ακόλουθη:

```
#!/bin/bash
 # Create directories and permissions
 sudo mkdir -p /var/www/citytrack-api
 sudo chown -R kstam /var/www/citytrack-api
 sudo chown -R kstam:kstam /var/www/citytrack-api
 # Create symbolic link from tomcat webapps to the warfile
  sudo ln -s /var/www/citytrack-api/citytrack-api.war /var/lib/tomcat7/webapps/citytrack-api.war
```

Τέλος, για να δέχεται ο Tomcat συμβολικούς συνδέσμους πρέπει να αλλάξουμε το ακόλουθο αρχείο:

```
/var/lib/tomcat7/conf/context.xml
```

Και να προσθέσουμε την ακόλουθη ιδιότητα στο στοιχείο Context:

```
<Context allowLinking="true">
```

Για να ληφθούν οι ρυθμίσεις αυτές, χρειάζεται επανεκκίνηση του Tomcat.

5.2.3 Αυτόματη εγκατάσταση

Έχοντας ρυθμίσει τα παραπάνω, μπορούμε πλέον να ρυθμίσουμε το CI μας ώστε να εγκαθιστά αυτόματα την εφαρμογή κάθε φορά που αλλάζουμε τον κώδικα στην απομακρυσμένη αποθήκη κώδικα.

Ο Jenkins (CI), παρέχει από προεπιλογή τη δυνατότητα να ανιχνεύει αλλαγές σε μία αποθήκη κώδικα Git. Χρησιμοποιώντας τη δυνατότητα του αυτή, τον ρυθμίσαμε ώστε κάθε φορά που ανιχνεύει κάποια αλλαγή, να «χτίζει» την εφαρμογή με χρήση του Maven και να τρέχει όλα τα Τεστ τα οποία βρίσκονται μαζί με τον κώδικα. Τότε, και μόνο αν ο κώδικας περάσει όλα τα τεστ, ξεκινά τη διαδικασία εγκατάστασης στο μηχάνημα εξυπηρετητή.

Συγκεκριμένα με χρήση του maven, «χτίζει» ένα αρχείο WAR το οποίο στη συνέχεια αντιγράφει στην σωστή τοποθεσία του εξυπηρετητή. Η διαδικασία που ακολουθείται για την εγκατάσταση φαίνεται στον παρακάτω κώδικα:

```
#!/bin/bash
#Create the WAR file
/usr/lib/maven/bin/mvn clean package -DskipTests

#Check if the WAR file was created properly
[ -f "target/citytrack-api.war" ] || { echo "WAR file NOT FOUND"; exit 1; }

#Upload the WAR FILE to the VM
scp -P 10000 target/citytrack-api.war kstam@83.212.114.165:/var/www/citytrack-api/
```

6

Τεχνική περιγραφή του γραφικού περιβάλλοντος

6.1 Τεχνολογίες και εργαλεία

Αρχικά, όπως έχουμε ήδη αναφέρει, για την υλοποίηση του γραφικού περιβάλλοντος χρησιμοποιήσαμε κατά κύριο λόγο τη γλώσσα προγραμματισμού JavaScript. Επίσης, βασική επιλογή της σχεδίασης και υλοποίησης του γραφικού περιβάλλοντος ήταν να φιλοξενηθεί το τελευταίο από έναν εξυπηρετητή Node.js (Node.js 2015) ο οποίος είναι επίσης βασισμένος στη γλώσσα JavaScript.

Όπως αναφέραμε στο κεφάλαιο 2.3 ο κώδικας που αποτελεί την εφαρμογή γραφικού περιβάλλοντος χωρίζεται σε δύο τμήματα. (1) Στον κώδικα που ορίζει τη λειτουργία του εξυπηρετητή όπως τις θήρες στις οποίες ακούει, τα URL τα οποία εξυπηρετεί και τις τοποθεσίες από τις οποίες «σερβίρει» τα στατικά αρχεία της εφαρμογής, και (2) στον κώδικα που υλοποιεί το γραφικό περιβάλλον χρησιμοποιώντας τις ευρέως χρησιμοποιούμενες τεχνολογίες JavaScript, CSS, HTML5.

Συγκεκριμένα, επιλέξαμε να φιλοξενήσουμε την εφαρμογή μας σε έναν Node.js εξυπηρετητή, διαφορετικό από αυτό στον οποίο φιλοξενείται η διεπαφή υπηρεσιών για τους εξής λόγους:

1. Ανεξαρτησία του γραφικού περιβάλλοντος από τη διεπαφή υπηρεσιών. Μπορούμε να αλλάζουμε το ένα ανεξάρτητα από το άλλο εφόσον διατηρούμε το «συμβόλαιο» επικοινωνίας των δύο. Αυτό μας προσφέρει αρκετή ευελιξία κατά την ανάπτυξη.
2. Ομοιομορφία στη γλώσσα προγραμματισμού που χρησιμοποιείται. Κάθε υποσύστημα χρησιμοποιεί κατά βάση μία γλώσσα προγραμματισμού.

3. Για το Node.js προσφέρονται πληθώρα εργαλείων που διευκολύνουν δραστικά την ανάπτυξη μιας εφαρμογής γραφικού περιβάλλοντος. Ακόμη και αν δεν επιλέγαμε το Node.js για την εξυπηρέτηση του γραφικού περιβάλλοντος, είναι βέβαιο ότι θα αποτελούσε ούτως ή άλλως τμήμα του «χτισίματος» της εφαρμογής.
4. Το «χτίσιμο» μιας HTML5 εφαρμογής έχει εντελώς διαφορετικές απαιτήσεις από ότι μια διεπαφή υπηρεσιών. Χωρίζοντας τις δύο εφαρμογές κερδίζουμε τόσο ευελιξία κατά το χτίσιμο, όσο και καθαρότητα ως προς την παραμετροποίησή του.

6.1.1 Node.js

Το Node.js είναι μια πλατφόρμα που μας επιτρέπει να γράψουμε και να εκτελέσουμε JavaScript προγράμματα χωρίς την ανάγκη ενός φυλλομετρητή ιστού (web browser). Για να το πετύχει αυτό, μεταγλωττίζει σε γλώσσα μηχανής και «τρέχει» τα προγράμματα, κάνοντας χρήση της V8, μιας εικονικής μηχανής που αναπτύχθηκε από τη Google για τις ανάγκες του Chrome και στη συνέχεια εκδόθηκε ανεξάρτητα.

Βασική φιλοσοφία του Node.js είναι ότι ο κώδικας που δεν πραγματοποιεί Είσοδο/Εξοδο δεδομένων εκτελείται ταχύτατα και σύγχρονα, ωστόσο η Είσοδος/Εξοδος εκτελείται ασύγχρονα. Κάθε φορά δηλαδή που το πρόγραμμά μας προσπαθεί να «διαβάσει» κάτι από κάποιο αρχείο ή πηγή στο διαδίκτυο, η εφαρμογή δεν παγώνει περιμένοντας, αλλά θέτει μία συνάρτηση που θα εκτελεστεί όταν η είσοδος διαβαστεί και συνεχίζει κανονικά την εκτέλεσή του χωρίς να σταματά για ΕΕ.

Ένα από τα δυνατά σημεία του Node.js είναι, εκτός των άλλων, η τεράστια κοινότητα ανοικτού λογισμικού που το υποστηρίζει. Σχεδόν καθημερινά προστίθενται νέα εργαλεία, επεκτάσεις και βιβλιοθήκες που είναι γραμμένα για την πλατφόρμα Node.js. Οι επεκτάσεις αυτές δημοσιεύονται σε μία δημόσια αποθήκη πακέτων Node στο διαδίκτυο, και μπορούν να εγκατασταθούν πολύ εύκολα σε μία εφαρμογή με χρήση του διαχειριστή πακέτων Node.js, NPM.

6.1.2 NPM και Bower για τη διαχείριση εξαρτήσεων

Το NPM (NPM 2015), όπως αναφέρθηκε στην προηγούμενη ενότητα είναι ένα εργαλείο διαχείρισης πακέτων του Node.js (Node Package Manager). Η λειτουργία του είναι διττή. Αφενός επιτρέπει στους προγραμματιστές να δημοσιεύουν τον κώδικά τους στην απομακρυσμένη αποθήκη πακέτων του Node.js και αφετέρου επιτρέπει την εγκατάσταση βιβλιοθηκών που έχουν δημοσιευτεί στην αποθήκη αυτή για να χρησιμοποιηθούν από το

πρόγραμμα. Με ένα τρόπο όμοιο με το Maven, κάνοντας χρήση του NPM, δε χρειάζεται πλέον να αντιγράφουμε κώδικα βιβλιοθηκών χειροκίνητα στην εφαρμογή, αλλά αφήνουμε το NPM να το κάνει αυτό για μας.

Η περιγραφή των απαιτήσεων στο NPM γίνεται με ένα JSON αρχείο που πρέπει να φέρει το όνομα «*package.json*». Το αρχείο αυτό μοιάζει αρκετά με το XML αρχείο ρυθμίσεων του Maven ως προς το σκοπό του, διαφέρει ωστόσο ως προς τη μορφή και τη σύνταξη του. Συγκεκριμένα, ορίζει δύο ήδη εξαρτήσεων: (1) εκείνες οι οποίες αποτελούν κομμάτι της εφαρμογής όταν αυτή δημοσιεύεται (dependencies) και (2) εκείνες οι οποίες χρησιμεύουν μόνο κατά την ανάπτυξη της εφαρμογής (devDependencies). Οι δεύτερες αγνοούνται κατά το «χτίσιμο» της εφαρμογής αλλά μπορεί να είναι εργαλεία που βοηθούν στη διαδικασία του «χτισίματος».

Στο NPM μπορεί να βρει κανείς μία πληθώρα βιβλιοθηκών για χρήση σε έναν εξυπηρετητή (διαχείριση αιτήσεων, ταυτοποίηση, ασφάλεια, κλπ), αλλά και εργαλείων που σκοπεύουν στη διευκόλυνση της ανάπτυξης εφαρμογών (αυτόματο «χτίσιμο», βιβλιοθήκες για τεστ κλπ). Επίσης, τελευταία, δημοσιεύονται στο NPM πολλές βιβλιοθήκες για χρήση στον φυλλομετρητή του χρήστη (jQuery, Angular.js κλπ).

Ωστόσο, για τη διαχείριση εξαρτήσεων φυλλομετρητή, υπάρχει ένα καταλληλότερο εργαλείο, το Bower (Bower 2015). Το Bower είναι επίσης γραμμένο σε JavaScript και εξειδικεύεται στη διαχείριση εξαρτήσεων για γραφικά περιβάλλοντα. Οι εξαρτήσεις αυτές μπορούν να είναι JavaScript κώδικας, HTML, ή ακόμη και CSS. Η φιλοσοφία ρύθμισής του είναι ακριβώς όμοια με αυτή του NPM, με τη διαφορά ότι το αντίστοιχο αρχείο για το bower πρέπει να έχει το όνομα «*bower.json*».

Για την εφαρμογή μας χρησιμοποιήσαμε και τα δύο. Για εξαρτήσεις σχετικές με εργαλεία ή βιβλιοθήκες που χρησιμοποιούνται στο κομμάτι του εξυπηρετητή χρησιμοποιούμε NPM. Για εξαρτήσεις που χρησιμοποιούνται αποκλειστικά στην εφαρμογή που εκτελείται στο φυλλομετρητή ιστού χρησιμοποιούμε bower. Αν κάποια εξάρτηση χρησιμοποιείται και από τα δύο κομμάτια τότε χρησιμοποιούμε NPM.

package.json

```
{
 "name": "citytrack-web",
 "version": "0.0.1",
 "scripts": {},
 "dependencies": {
  "express": "~4.9.0"
 },
 "devDependencies": {
  "bower": "^1.3.12"
 }
}
```

bower.json

```
{
 "name": "citytrack-web",
 "version": "0.0.1",
 "description": "City Track Web Client",
 "dependencies": {
  "jquery": "~2.1.1"
 },
 "devDependencies": {
  "angular-mocks": "~1.3.13"
 },
}
```

Κώδικας 6.1 Παράδειγμα σύνταξης bower.json και package.json

6.1.3 Εισαγωγή εξαρτήσεων στον κώδικα σύμφωνα με το CommonJS

Το CommonJS είναι η προσέγγιση που ακολουθεί η πλατφόρμα Node.js για την εισαγωγή εξαρτήσεων στον κώδικα. Με τον όρο εισαγωγή εξαρτήσεων, εννοούμε τη δυνατότητα χρήσης κώδικα που έχει αποθηκευθεί σε ένα αρχείο από ένα άλλο αρχείο, χωρίς να χρειάζεται να αντιγράψουμε τον κώδικα. Ανάλογη νοοτροπία ακολουθούν οι εισαγωγές κατά Java.

Στον κόσμο του JavaScript υπάρχουν γενικά τρεις προσεγγίσεις αυτή τη στιγμή όσων αφορά στην εισαγωγή εξαρτήσεων από διαφορετικά αρχεία. (1) Η ασύγχρονη φόρτωση ενοτήτων (Asynchronous Module Loading ή AMD) (2) Η προσέγγιση φόρτωσης κατά Node.js και (3) Το ανερχόμενο πρότυπο εισαγωγής εξαρτήσεων με βάση το πρότυπο EcmaScript6.

Σύντομα, όταν θα εκδοθεί το πρότυπο EcmaScript6 όλα τα εργαλεία θα αρχίσουν να χρησιμοποιούν τον τρόπο που ορίζεται εκεί. Ωστόσο σε αυτό το σημείο δεν υποστηρίζεται από τους περισσότερους φυλλομετρητής, ούτε από την πλατφόρμα Node.js. Για το λόγο αυτό, και για λόγους ομοιομορφίας, διαλέξαμε να ακολουθήσουμε το πρότυπο εισαγωγής CommonJS τόσο για τον κώδικα εξυπηρετητή όσο και για τον κώδικα πελάτη.

Σύμφωνα με το πρότυπο CommonJS κάθε αρχείο JavaScript, όταν εκτελεστεί «εξάγει» ένα αντικείμενο ή μία συνάρτηση ως τιμή της μεταβλητής «module.exports». Όταν ένα άλλο αρχείο «εισάγει» το προηγούμενο αρχείο χρησιμοποιώντας τη συνάρτηση «require(FILE_NAME)», τότε του επιστρέφεται το αντικείμενο που ορίστηκε ως τιμή του "module.exports" στο αρχείο αυτό. Για παράδειγμα έστω ότι το αρχείο A.js περιέχει τον ακόλουθο κώδικα:

```
module.exports = 3;
```

Τότε η εκτέλεση του ακόλουθου κώδικα,

```
var x = require('A');
console.log(x);
```

Θα εκτυπώσει στην κονσόλα τον αριθμό «3».

Στην πλευρά του εξυπηρετητή η εκτέλεση αυτού του κώδικα είναι τετριμμένη γιατί ο κώδικας μεταγλωττίζεται πριν εκτελεστεί, που σημαίνει ότι όλες οι εξαρτήσεις επιλύονται κατά το χρόνο μεταγλώττισης. Από την άλλη, στον κώδικα της πλευράς πελάτη πρέπει να βρούμε κάποιο τρόπο αυτές οι εξαρτήσεις να ενσωματώνονται σε ένα ή περισσότερα αρχεία τα οποία να γίνονται διαθέσιμα στον φυλλομετρητή κατά τη φόρτωση της σελίδας. Αυτό ακριβώς το πρόβλημα επιλύει το εργαλείο με όνομα Browserify (Browserify 2015).

Το Broswerify (Browserify 2015) είναι ένα εργαλείο γραμμένο σε JavaScript που επιτρέπει τη χρήση του προτύπου CommonJS για τη διαχείριση εξαρτήσεων στην πλευρά πελάτη. Το Browserify ουσιαστικά εκτελείται κατά το «χτίσιμο» της εφαρμογής, δέχεται ως είσοδο ένα αρχείο εισόδου, και με βάση αυτό επιλύει όλες τις εξαρτήσεις και δημιουργεί ένα αρχείο στο οποίο τις ενοποιεί. Αυτό είναι το αρχείο το οποίο στη συνέχεια θα φορτώσουμε στον φυλλομετρητή κατά το άνοιγμα της εφαρμογής.

Τέλος, επειδή κάποιες βιβλιοθήκες, ειδικότερα αυτές που εγκαθίστανται από το Bower δεν είναι γραμμένες σε CommonJS μορφή, χρειάζεται να υποστούν κάποια τροποποίηση πριν την εκτέλεση του Browserify. Την τροποποίηση αυτή αναλαμβάνει αυτόματα το debowerify.

6.1.4 Test Driven Development χρησιμοποιώντας Karma, Mocha και Chai

Όπως και με τον κώδικα της διεπαφής υπηρεσιών που γράφτηκε σε Java έτσι και με τον κώδικα του γραφικού περιβάλλοντος ακολουθήσαμε τη φιλοσοφία Test Driven Development που περιγράψαμε στην ενότητα 5.1.3.

Βέβαια, για τον κώδικα σε JavaScript υπάρχει ένα διαφορετικό σετ από εργαλεία για τη συγγραφή και εκτέλεση τεστ. Συγκεκριμένα, θα αναφερθούμε στα εργαλεία Karma και Mocha που χρησιμοποιήσαμε κατά την ανάπτυξη της εφαρμογής Citytrack.

Το Mocha είναι μια βιβλιοθήκη που επιτρέπει τον ορισμό και την εκτέλεση test σε ένα περιβάλλον Node.js χωρίς την ανάγκη ύπαρξης ενός φυλλομετρητή. Ορίζει, δε, περισσότερα από ένα στυλ συγγραφής τεστ και μας επιτρέπει να επιλέξουμε εκείνο που μας ταιριάζει περισσότερο. Στην περίπτωση που θέλουμε τα test μας να εκτελούνται σε φυλλομετρητή, όπως για παράδειγμα για τις ανάγκες του Citytrack, το Mocha μας επιτρέπει να φορτώσουμε τα test μας σε μία HTML σελίδα και να τα εκτελέσουμε στο φυλλομετρητή.

Το Chai είναι μία βιβλιοθήκη επικυρώσεων (Assertion Framework), η οποία δουλεύει πολύ καλά σε συνεργασία με τη βιβλιοθήκη Mocha και μας δίνει τη δυνατότητα να ορίζουμε τις προσδοκίες μας από ένα test και να τις ελέγχουμε κατά την εκτέλεσή του. Επίσης, όπως και το Mocha, μας επιτρέπει να διαλέξουμε από ένα σύνολο διαφορετικών στυλ για τον ορισμό των προσδοκιών.

```
describe('JavaScript', function() {
   it('should perform addition properly', function() {
      expect(2 + 0).to.equal(2);
      expect(2 + 5).to.equal(7);
      expect(2 + 3).not.to.equal(6);
   });
});
```

Κώδικας 6.2 Παράδειγμα ενός τεστ γραμμένου με χρήση mocha και chai

Τέλος, το Karma είναι ένα εργαλείο το οποίο ενώνει τις δυνατότητες των ανωτέρω βιβλιοθηκών σε μία σελίδα HTML, εκτελεί όλα τα τεστ και επιστρέφει τα αποτελέσματα από την εκτέλεσή τους. Το Karma, βασίζεται σε ένα αρχείο ρυθμίσεων με βάση το οποίο γνωρίζει ποιες βιβλιοθήκες να χρησιμοποιήσει, που να βρει τα τεστ, σε τι browser να τα εκτελέσει, πώς να δημοσιεύσει τα αποτελέσματα και ενδεχομένως επιπλέον πληροφορίες σχετικές με την εκτέλεση των τεστ. Για παράδειγμα, μπορούμε να ορίσουμε κάποιο είδος προ-επεξεργασίας πριν εκτελεστούν τα τεστ. Για να εκτελέσουμε τα τεστ με Karma τρέχουμε την εντολή «karma start *karma.config.js*» όπου karma.config.js είναι το αρχείο ρυθμίσεων.

```
// Karma configuration
module.exports = function(config) {
  config.set({
    // base path that will be used to resolve all patterns
    basePath: './',
    // frameworks to use
    frameworks: ['browserify', 'mocha', 'chai'],
    // list of files / patterns to load in the browser
    files: [
      'bower_components/es5-shim/es5-shim.js',
      'test/**/*Spec.js',
      'src/**/*.html'
    ],
    // list of files to exclude
    exclude: ['public/js/build/*.js'],
    // preprocess matching files before serving them to the browser
    preprocessors: {
      'test/client/**/*.js': [ 'browserify' ],
      'test/common/*.js': [ 'browserify' ],
      'test/model/*.js': [ 'browserify' ],
      'src/**/*.html': ['ng-html2js']
    },
    ngHtml2JsPreprocessor: {
      stripPrefix: 'src/'
    },
    browserify: {
      transform: ['debowerify'],
      paths: ['./node_modules', './src/js', './views']
    },
    // test results reporter to use
    reporters: ['progress'],
    // web server port
    port: 9876,
    // enable / disable colors in the output (reporters and logs)
    colors: true,
    // level of logging
    logLevel: config.LOG_INFO,
    // watch file and execute tests whenever any file changes
    autoWatch: true,
    // start these browser
    browsers: ['PhantomJS'],
    // Continuous Integration mode
    singleRun: false
  });
};
```

Αριστερά, βλέπουμε το αρχείο ρυθμίσεων που χρησιμοποιήσαμε για να εκτελέσουμε τα τεστ μας με χρήση karma.

Χρησιμοποιώντας το αρχείο αυτό, το μόνο που έχουμε να κάνουμε είναι να εκτελέσουμε την εντολή «karma start *karma.config.js*» και μπορούμε να αρχίσουμε να ορίζουμε τεστ. Κάθε φορά που ορίζουμε ένα νέο τεστ, το karma το «βλέπει» και τρέχει ξανά όλη τη σουίτα τεστ. Ομοίως όταν πραγματοποιούμε κάποια αλλαγή στον κωδικά μας.

Κώδικας 6.3 Αρχείο ρυθμίσεων Karma για την εκτέλεση των τεστ στο Citytrack Web

6.1.5 LESS και CSS για το σχεδιασμό της διεπαφής χρήστη

Για το σχεδιασμό διεπαφών στον ιστό, η σύγχρονη προσέγγιση εστιάζεται κυρίως στις τεχνολογίες HTML5, JavaScript και CSS. Η συγγραφή ωστόσο κώδικα CSS αν και εύκολη καταλήγει να είναι ιδιαίτερα επίπονη για μεγάλες εφαρμογές. Για το λόγο αυτό υπάρχουν διάφορες επεκτάσεις ή για την ακρίβεια καινούργιες γλώσσες που μοιάζουν με την CSS και επιχειρούν να επιλύσουν προβλήματα όπως η διαχείριση πολλών αρχείων, η εισαγωγή συναρτήσεων στο συντακτικό της γλώσσας, ο εμφωλιασμός διάφορων στυλ ώστε ο κώδικας να ακολουθεί τη δομή του HTML κοκ.

Μία από αυτές τις νέες γλώσσες είναι και η LESS. Η LESS ορίζει ένα συντακτικό αρκετά όμοιο με αυτό της CSS επεκτείνοντάς το όπου χρειάζεται για την προσθήκη έξτρα λειτουργικότητας. Η LESS δεν επεκτείνει τις δυνατότητες της CSS αφού ο LESS κώδικας στο τέλος πάντα μετατρέπεται στον αντίστοιχο CSS, αλλά καθιστά τη συγγραφή κώδικα για το στυλ της εφαρμογής πολύ πιο ευέλικτη. Τα αρχεία LESS μπορούν να μετατρέπονται σε CSS είτε απευθείας στον φυλλομετρητή ή κατά το χτίσιμο της εφαρμογής. Το τελευταίο συνήθως προτιμάται για λόγους ταχύτητας ιδιαίτερα για μεγάλες εφαρμογές με μεγάλες απαιτήσεις σε επεξεργαστική ισχύ.

Στη συνέχεια βλέπουμε ένα παράδειγμα σε LESS και τον αντίστοιχο κώδικα σε CSS που παράγεται ως αποτέλεσμα του LESS κώδικα:

```
@padding: 5px;
@height: 30px;
.logo {
   float: left;
   padding-top: @padding + 5;
   .logo-icon {
      height: @height + 10;
   }
}
```

```
.logo {
   float: left;
   padding-top: 10px;
}
.logo .logo-icon {
   height: 40px;
}
```

Κώδικας 6.4 Παράδειγμα LESS κώδικα και του αντίστοιχου CSS που αυτός παράγει

Μερικά από τα πλεονεκτήματα του LESS κώδικα που φαίνονται παραπάνω είναι ο ορισμός μεταβλητών, η εκτέλεση εκφράσεων με βάση τις μεταβλητές καθώς και το αυτόματο «ξεδίπλωμα» του κώδικα ώστε να είναι συμβατός με CSS.

6.1.6 Angular.js για την υλοποίηση του γραφικού περιβάλλοντος

Το Angular.js είναι μία πλατφόρμα ανάπτυξης για γραφικά περιβάλλοντα και συγκεκριμένα για εφαρμογές ιστού μιας σελίδας (single page web applications). Το Angular.js βασίζεται στη γνωστή πρακτική MVC που αναλύσαμε στην ενότητα 5.1.2 για το Spring. Διαχωρίζει δηλαδή την ευθύνη για τα επιμέρους τμήματα της εφαρμογής σε τρία μέρη: (1) τους Χειριστές (Controllers), (2) το Μοντέλο (Model) και (3) την Απεικόνιση (View).

Το Angular.js επικεντρώνει την προσοχή του στα εξής τρία χαρακτηριστικά:

- Στην ανεξαρτητοποίηση της λογικής μιας εφαρμογής από την διαχείριση του DOM.
- Στην ανεξαρτητοποίηση του κώδικα που ορίζει τη λειτουργικότητα στην εφαρμογή από τον κώδικα που είναι υπεύθυνος για την αλληλεπίδραση με τον εξυπηρετητή και την ανάκτηση δεδομένων.
- Στην παροχή μιας δομημένης ακολουθίας βημάτων για το σχεδιασμό και την ανάπτυξη έως και τον έλεγχο της εφαρμογής.

Το Angular, όταν ενσωματώνεται σε μια εφαρμογή, ξεκινά διαβάζοντας το DOM της σελίδας και αναγνωρίζοντας ένα σύνολο από προκαθορισμένες ιδιότητες που ενσωματώνονται στους κόμβους του DOM (DOM Custom Attributes). Έπειτα, αποτιμά τις εκφράσεις που αυτές οι ιδιότητες περιέχουν και τις χρησιμοποιεί ως είσοδο για την έναρξη των διαφόρων τμημάτων της εφαρμογής.

Ένα από τα βασικά χαρακτηριστικά του Angular.js είναι η «δέσμευση δεδομένων διπλής κατεύθυνσης» ή «Two-way Data Binding». Σύμφωνα με αυτό, οι αλλαγές που γίνονται (προγραμματιστικά) στο μοντέλο της εφαρμογής απεικονίζονται αυτόματα από το Angular στον φυλλομετρητή του χρήστη αλλάζοντας το HTML της σελίδας κατάλληλα. Παράλληλα, αλλαγές που γίνονται από το χρήστη στην απεικόνιση της εφαρμογής (πχ αλλαγή της τιμής ενός πεδίου κειμένου) μεταφέρονται αυτόματα στο μοντέλο αλλάζοντας την τιμή του. Με αυτό τον τρόπο οι προγραμματιστές δε χρειάζεται να ανησυχούν για την απευθείας αλλαγή του HTML της εφαρμογής.

Για την εκκίνηση του Angular.js απαιτείται αφενός η φόρτωση της βιβλιοθήκης Angular.js στη σελίδα, και αφετέρου η τοποθέτηση της ιδιότητας «ng-app» όπως φαίνεται στον κώδικα που ακολουθεί.

```
<div class="view-container" ng-app="citytrack">
  …
</div>
```

Δεδομένης αυτής της ιδιότητας το Angular θα προσπαθήσει να βρει αν έχει οριστεί κάποιο module με το όνομα που δίδεται ως τιμή της ιδιότητας ng-app και στη συνέχεια θα χρησιμοποιήσει τον ορισμό αυτής της εφαρμογής για να ερμηνεύσει όλες τις ιδιότητες που περιέχονται στο HTML στοιχείο.

Για την ανάπτυξη της εφαρμογής, το Angular.js ορίζει 4 διαφορετικά ήδη ενοτήτων τα οποία εξυπηρετούν διαφορετικούς σκοπούς:

1. Χειριστές (Controllers)
2. Υπηρεσίες (Services)
3. Οδηγίες (Directives)
4. Σταθερές (Constants)

Οι Χειριστές είναι αυτοί που «ενώνουν» τα δεδομένα και τη λογική της εφαρμογής με την απεικόνισή τους μέσω του Μοντέλου. Στο Angular.js το μοντέλο παρέχεται στους Χειριστές μέσω της Υπηρεσίας «`$scope`». Ό,τι προστίθεται στο αντικείμενο `$scope` ενός Χειριστή παρέχεται στη συνέχεια στην απεικόνιση και μπορεί να χρησιμοποιηθεί στα HTML αρχεία που την ορίζουν.

Οι Υπηρεσίες είναι συνήθως τμήματα κώδικα που «ζητούν» τα δεδομένα από τον εξυπηρετητή αλλά δεν περιορίζονται μόνο σε αυτό. Μπορούν επίσης να ορίζουν πολύπλοκα κομμάτια λογικής ή αλγορίθμων, να αποτελούν μέσο μεταφοράς δεδομένων μέσω των διαφόρων Χειριστών ή τέλος να παράγουν Γεγονότα με βάση τα οποία οι Χειριστές να ενημερώνονται για συγκεκριμένες ενέργειες που έγιναν ή πρέπει να εκτελεστούν.

Οι Οδηγίες ορίζουν επαναχρησιμοποιούμενες οντότητες που «εκθέτουν» μία διεπαφή, παράγουν κάποια δεδομένα και ορίζουν τη δική τους απεικόνιση. Ένα παράδειγμα οντότητας μπορεί για παράδειγμα να είναι ένα πεδίο επιλογής ημερομηνίας το οποίο να προστίθεται στο HTML απλά με μία γραμμή, και όταν συναντάται από το Angular.js να αντικαθίσταται από ένα σύνολο από HTML οντότητες οι οποίες συνεργάζονται μεταξύ τους και υλοποιούν ένα ημερολόγιο για να επιλέξει ο χρήστης μια ημερομηνία. Η ημερομηνία είναι το δεδομένο που επιστρέφεται, το ημερολόγιο είναι η απεικόνιση και τέλος ο τρόπος ενσωμάτωσής της Οδηγίας στο HTML είναι η διεπαφή που ορίζει.

Τέλος οι Σταθερές είναι μια υποπερίπτωση των υπηρεσιών, που απλά ορίζουν ένα σύνολο από τιμές που ανταποκρίνονται σε συγκεκριμένα κλειδιά και μπορούν να μοιράζονται στα διάφορα επιμέρους τμήματα της εφαρμογής.

6.1.7 Gulp για το χτίσιμο της διεπαφής

Όπως αναφέραμε ήδη, ένα από τα βασικότερα μελήματά μας από την αρχή της υλοποίησης του γραφικού περιβάλλοντος ήταν η αυτοματοποίηση της διαδικασίας του χτισίματος όπου αυτό είναι δυνατόν, και ταυτόχρονα η επιτάχυνση του κύκλου ανάπτυξης/ελέγχου της εφαρμογής.

Ταυτόχρονα, για να πετύχουμε την γρήγορη φόρτωση της εφαρμογής μας από τους φυλλομετρητές των χρηστών χρειάζεται να προβούμε σε βελτιστοποιήσεις και σε κατάλληλη οργάνωση και εισαγωγή του κώδικα στην HTML σελίδα. Αυτό είναι σημαντικό τόσο για τον κώδικα JavaScript όσο και για τον κώδικα CSS ή και τμηματικά HTML τα οποία φορτώνονται ασύγχρονα από την εφαρμογή.

Ένα πολύ ισχυρό εργαλείο που επικεντρώνεται στα παραπάνω είναι το Gulp.js (Gulp.js 2015) ή gulp. Το gulp είναι ένα εργαλείο ανοικτού λογισμικού γραμμένο σε Node.js το οποίο διαθέτει ήδη μια τεράστια κοινότητα χρηστών και πληθώρα επεκτάσεων που εξειδικεύονται σε συγκεκριμένες λειτουργίες. Στο gulp, το χτίσιμο γίνεται μέσα από ένα σύνολο από διεργασίες οι οποίες εκτελούνται σύμφωνα με μια σειρά που ορίζεται στις διεργασίες αυτές. Σε αντίθεση με παρόμοια εργαλεία, οι διεργασίες του gulp γράφονται επίσης ως κώδικας JavaScript, γεγονός που τις καθιστά ιδιαίτερα ευανάγνωστες.

Σημείο εισόδου στο gulp είναι ένα αρχείο με όνομα gulpfile.js. Μέσα σε αυτό ορίζονται ένα σύνολο από διεργασίες (tasks). Κάθε διεργασία χαρακτηρίζεται από ένα μοναδικό όνομα και μπορεί να εκτελείται από τη γραμμή εντολών με "gulp NAME" όπου NAME, το όνομα της διεργασίας.

Εάν υπάρχουν πολλές διεργασίες, αυτές μπορούν να χωρίζονται σε επιμέρους αρχεία και στη συνέχεια να «εισάγονται» ως απαιτήσεις στο gulpfile.js χρησιμοποιώντας την ίδια μέθοδο απαιτήσεων που περιγράφηκε στην ενότητα 6.1.3.

Για την εφαρμογή μας ορίσαμε αυτόματες διεργασίες για τις ακόλουθες ενέργειες:

1. Μετάφραση του LESS σε CSS
2. Κατέβασμα όλων των εξαρτήσεων bower
3. Μετακίνηση όλων των εικόνων που έρχονται από εξωτερικές εξαρτήσεις σε κατάλληλη τοποθεσία
4. Συνένωση και συμπίεση όλου του «εξωτερικού» CSS κώδικα σε ένα αρχείο
5. Συνένωση και συμπίεση όλου του κώδικα JavaScript σε ένα αρχείο
6. Συνένωση και συμπίεση όλων των εξωτερικών βιβλιοθηκών JavaScript σε ένα αρχείο

7. Εκτέλεση όλων των Test με χρήση του Karma
8. Έλεγχος του JavaScript για συμβατότητα με το πρότυπο JSHint
9. Δημιουργία ενός αρχείου που περιέχει όλα τα επιμέρους HTML αρχεία που χρησιμοποιούνται από το Angular.js για την αποφυγή επιπλέον ασύγχρονων κλήσεων δικτύου για τη φόρτωσή τους
10. Εκτέλεση της εφαρμογής σε development mode ώστε να παρακολουθούνται αλλαγές στον κώδικα και να ανανεώνεται αυτόματα η εφαρμογή με την τεχνολογία livereload.

Τέλος ορίσαμε μία διεργασία με όνομα "build" η οποία μπορεί να εκτελεί όλες τις ανωτέρω διεργασίες με κατάλληλη σειρά. Όταν η build διεργασία τελειώσει μπορούμε να εκκινήσουμε τον Node.js εξυπηρετητή και όλα τα απαιτούμενα αρχεία θα είναι στην κατάλληλη θέση.

Για παράδειγμα, η ακόλουθη διεργασία χρησιμοποιεί ένα σύνολο από επεκτάσεις για να μεταφράσει τον κώδικα LESS σε CSS, να τον συμπιέσει, και τέλος να τον τοποθετήσει στην τοποθεσία *«./public/css»* όπου και θα αναζητηθεί από την εφαρμογή.

```
var gulp = require('gulp');
var less = require('gulp-less');
var path = require('path');
var minify = require('gulp-minify-css');

gulp.task('less', function() {
  return gulp.src('./src/less/main.less')
    .pipe(less())
    .pipe(minify())
    .pipe(gulp.dest('./public/css'));
});
```

Κώδικας 6.5 Ορισμός διεργασίας «less» στο Gulp

Όπως βλέπουμε, το gulp βασίζεται στα node streams τα οποία ορίζονται με τη λέξη κλειδί «pipe». Αυτό σημαίνει ότι το αποτέλεσμα του ενός βήματος αποστέλλεται ως είσοδος στο επόμενο ως "stream" χωρίς να αποθηκεύεται το αποτέλεσμά του στο δίσκο, επιταχύνοντας σημαντικά τις διεργασίες. Βλέπουμε επίσης πώς ορίζεται μια διεργασία με το όνομά της, και πως εισάγονται οι απαιτήσεις από άλλες διεργασίες ή εργαλεία σύμφωνα με το πρότυπο Common.js.

Τέλος, για τον ορισμό εξαρτήσεων μπορούμε να χρησιμοποιήσουμε τον ακόλουθο τρόπο ορισμού διεργασιών:

```
var gulp = require('gulp');
gulp.task('theProcess', ['theDependency'], function() {
  // do stuff
});
```

Κώδικας 6.6 Ορισμός εξαρτήσεων μεταξύ διαφορετικών διεργασιών στο Gulp. Εδώ η διεργασία «theProcess» δε θα εκτελεστεί πριν τη διεργασία «theDependency»

6.2 JavaScript στον Εξυπηρετητή

Για να δημιουργήσουμε έναν εξυπηρετητή με τη χρήση του Node.js χρησιμοποιήσαμε κατά βάση τη βιβλιοθήκη-πλαίσιο express.js (Express.js 2015).

Το express.js καθιστά την υλοποίηση ενός εξυπηρετητή με χρήση Node.js υπόθεση μερικών δεκάδων γραμμών κώδικα. Συγκεκριμένα για τον ορισμό του εξυπηρετητή τα πιο σημαντικά τμήματα που χρειάστηκε να ρυθμίσουμε ήταν τα εξής:

1. Μία μηχανή απεικόνισης (view engine)
2. Το σημείο από το οποίο «σερβίρονται» τα στατικά αρχεία
3. Αντιστοίχηση του μονοπατιού ρίζας με την εισαγωγική σελίδα
4. Προώθηση των αιτημάτων κάτω από το μονοπάτι /api προς τη διεπαφή υπηρεσιών

Στη συνέχεια παρουσιάζονται οι εν λόγω ρυθμίσεις αναλυτικότερα.

6.2.1 Μηχανή Απεικόνισης

Ως μηχανή απεικόνισης χρησιμοποιήσαμε το Handlebars. Το Handlebars αποτελεί μία πολύ εύκολη στη χρήση μηχανή απεικόνισης η οποία επιτρέπει την απεικόνιση μέσω συγκεκριμένων templates. Για την εφαρμογή μας θα αρκούσε απλά το «σερβίρισμα» στατικών αρχείων. Ο μόνος λόγος που χρησιμοποιήσαμε τη μηχανή απεικόνισης είναι για να μπορούμε να χρησιμοποιήσουμε το μονοπάτι ρίζας για να ανοίξει η βασική σελίδα της εφαρμογής.

Η μηχανή απεικόνισης στο express.js ορίστηκε σύμφωνα με τον ακόλουθο κώδικα:

```
var app = express();
var express = require('express');
var path = require('path');
var exphbs = require('express-handlebars');
app.engine('.hbs', exphbs({
   defaultLayout: 'main',
   extname: '.hbs',
   layoutsDir: 'views/layouts/',
   partialsDir: 'views/partials/'
}));
app.set('views', path.join(__dirname, 'views'));
app.set('view engine', '.hbs');
```

Ουσιαστικά ορίζουμε πού βρίσκονται τα αρχεία που ορίζουν τη σελίδα εισόδου της εφαρμογής.

6.2.2 Φάκελος στατικών αρχείων

Ο φάκελος στατικών αρχείων είναι το σημείο στο οποίο ο εξυπηρετητής ψάχνει να βρει τα αρχεία που ορίζονται στην HTML σελίδα. Για παράδειγμα αν ο φάκελος στατικών αρχείων είναι ο φάκελος /public και στο HTML φορτώνεται η εικόνα /img/logo.png τότε αυτή θα αναζητηθεί στο φάκελο /public/img/log.png.

Ο ορισμός του φακέλου στατικών αρχείων στο express.js γίνεται με τον ακόλουθο τρόπο:

```
app.use(express.static(path.join(__dirname, 'public')));
```

6.2.3 Αντιστοίχηση του μονοπατιού ρίζας με την εισαγωγική σελίδα

Έπειτα πρέπει όταν κανείς επισκέπτεται το μονοπάτι ρίζα του εξυπηρετητή («/»), αυτός να του «σερβίρει» το αρχείο εισόδου της εφαρμογής. Αυτό γίνεται με τον ακόλουθο κώδικα:

```
var express = require('express');
var router = express.Router();
router.get('/', function(req, res) {
  res.render('index', { title: 'Citytrack' });
});
var app = express();
app.use('/', router);
```

Ουσιαστικά το render θα εξυπηρετηθεί από τη μηχανή απεικόνισης που ορίσαμε στην ενότητα 6.2.1 η οποία θα αναζητήσει ένα αρχείο με αυτό το όνομα “index.hbs” στο φάκελο που ορίστηκε ότι περιέχει τα views.

6.2.4 Προώθηση των αιτημάτων κάτω από το μονοπάτι /api προς τη διεπαφή υπηρεσιών

Μία εφαρμογή HTML που εκτελείται σε ένα φυλλομετρητή μπορεί να προσκομίζει πληροφορία από πολλές διαφορετικές διευθύνσεις. Για παράδειγμα, εικόνες ή άλλο κώδικα. Ωστόσο, όσον αφορά στην πραγματοποίηση ασύγχρονων απαιτήσεων AJAX, οι περισσότεροι φυλλομετρητές θέτουν διάφορους περιορισμούς για λόγους ασφαλείας. Ένας βασικός περιορισμός είναι ότι δεν επιτρέπουν την πρόσβαση υπηρεσιών που φιλοξενούνται σε εξυπηρετητές με διαφορετικό HostName.

Δε μπορούμε δηλαδή να καλέσουμε από το φυλλομετρητή τη διεπαφή υπηρεσιών αν αυτή είναι εγκατεστημένη σε διαφορετικό μηχάνημα ή εκτίθεται με URL που δεν ξεκινά με το ίδιο hostname όπως αυτό στο οποίο φιλοξενείται η σελίδα.

Για να ξεπεράσουμε το πρόβλημα αυτό πρέπει να κάνουμε διαθέσιμες τις υπηρεσίες της διεπαφής υπηρεσιών κάτω από το ίδιο hostname και ο ευκολότερος τρόπος να το πετύχουμε αυτό είναι κάνοντας χρήση της τεχνικής της προώθησης (proxy).

Η προώθηση στο Node.js είναι πολύ εύκολη. Στη συνέχεια παρουσιάζεται πως υλοποιήσαμε την προώθηση όλων των αιτημάτων σε URLs που ξεκινούν με το μονοπάτι /api προς τη διεπαφή υπηρεσιών:

```
var app = express();
var config = require('./config');
var request = require('request');
var TIMEOUT = 10 * 60 * 1000; //10min
var api = function(req, res) {
   var url = REMOTE_SERVICES_URL + require('url').parse(req.url).path;
   req.pipe(request({
      url: url,
      timeout: TIMEOUT
   }, function(error){
      if (error && error.code === 'ECONNREFUSED'){
         res.status(500).send('Could\'t reach remote server. Proxy failed');
      }
   })).pipe(res);
};
app.use('/api', api);
```

Κώδικας 6.7 Προώθηση αιτημάτων στη διεπαφή υπηρεσιών

6.2.5 Εκτέλεση του Εξυπηρετητή

Όλες οι ανωτέρω ρυθμίσεις γίνονται πάνω σε ένα αντικείμενο app το οποίο ορίζεται από τη βιβλιοθήκη express.js. Το αντικείμενο app στη συνέχεια μπορεί να χρησιμοποιηθεί για να «ακούσει» σε μία θύρα εξυπηρετώντας έτσι αιτήματα προς αυτή τη θύρα.

Η εκτέλεση γίνεται τρέχοντας ένα αρχείο με όνομα server.js το οποίο φέρει τον ακόλουθο κώδικα. Η εκτέλεση του αρχείου γίνεται με την εντολή «*node server.js*»

```
var app = require('../app.js');
var server;
app.set('port', process.env.PORT || 3000);
server = app.listen(app.get('port'), function() {
   console.log('Express server listening on port ' + server.address().port);
});
```

6.3 Εγκατάσταση και παραμετροποίηση της εφαρμογής

6.3.1 Έναρξη της εφαρμογής με χρήση του Forever.js

Βασική προϋπόθεση για να είναι βιώσιμη η αυτόματη εγκατάσταση της εφαρμογής είναι η μηδενική παύση εξυπηρέτησης αιτημάτων (downtime). Αυτό μπορεί να συμβεί για παράδειγμα κατά τη διάρκεια της εγκατάστασης μιας νέας έκδοσης αν σταματάμε την εφαρμογή για να εγκαταστήσουμε τη νέα έκδοση. Επίσης μπορεί να συμβεί στην περίπτωση που προκύψει κάποιο μη αναμενόμενο σφάλμα το οποίο δε διαχειριζόμαστε, οπότε η εκτέλεση του εξυπηρετητή θα σταματήσει.

Για το λόγο αυτό, χρησιμοποιούμε το εργαλείο Forever.js (Forever.js 2015) για να εκκινήσουμε τον εξυπηρετητή Node.js. Η χρήση του Forever.js είναι πολύ απλή και φαίνεται παρακάτω:

```
process.env.PORT = 8090;
process.env.NODE_ENV = process.env.NODE_ENV || 'production';
var forever = require('forever');
var child = new (forever.Monitor)('bin/start.js', {
    'silent': false,
    'pidFile': 'pids/app.pid',
    'watch': true,
    'watchDirectory': '.',      // Top-level directory to watch from.
    'watchIgnoreDotFiles': true, // whether to ignore dot files
    'watchIgnorePatterns': [], // array of glob patterns to ignore, merged with contents of
    'logFile': 'logs/forever.log', // Path to log output from forever process (when daemonized)
    'outFile': 'logs/forever.out', // Path to log output from child stdout
    'errFile': 'logs/forever.err'
});
child.start();
forever.startServer(child);
```

Κώδικας 6.8 Ορισμός του αρχείου εκκίνησης του server με χρήση του Forever.js

Όπως φαίνεται στον παραπάνω κώδικα, αρχικά ορίζουμε μια διεργασία παιδί η οποία φέρει κάποιες ρυθμίσεις. Συγκεκριμένα, ποιο JavaScript αρχείο θέλουμε να τρέξουμε καθώς και μια σειρά από παραμέτρους όπως την παρακολούθηση αρχείων για αλλαγές, τα αρχεία εξόδου κοκ. Στη συνέχεια, εκκινούμε αυτή τη διεργασία καθώς και ένα εξυπηρετητή forever ο οποίος λαμβάνει τη διεργασία ως είσοδο. Έτσι, αν για κάποιο λόγο η διαδικασία παιδί τερματίσει ή κάποιο από τα αρχεία υπό παρακολούθηση αλλάξει, η διεργασία επανεκκινείται αυτόματα από τον εξυπηρετητή forever. Πετυχαίνουμε έτσι το στόχο της ελαχιστοποίησης του downtime αλλά και της αυτόματης επανεκκίνησης της εφαρμογής όταν τα αρχεία υπό παρακολούθηση αλλάξουν.

6.3.2 Ορισμός του εξυπηρετητή web ως υπηρεσία συστήματος του Ubuntu

Όπως περιγράψαμε στην προηγούμενη ενότητα, με χρήση του Forever πετυχαίνουμε την αδιάκοπη λειτουργία της εφαρμογής για όσο το σύστημα εκτελείται κανονικά. Ποιος όμως εκτελεί αυτό το forever αρχείο και τι θα γίνει στην περίπτωση που το σύστημα επανεκκινήσει;

Ουσιαστικά, θέλουμε να αποφύγουμε την ανάγκη να ανοίξουμε το σύστημα και να εκτελέσουμε ξανά τον εξυπηρετητή με τη χρήση του forever σε περίπτωση που αυτό επανεκκινήσει από λάθος ή λόγω τεχνικού προβλήματος. Ένας καλός τρόπος να το πετύχουμε αυτό είναι να δημιουργήσουμε μία υπηρεσία συστήματος στο Ubuntu η οποία να ξεκινά τον server αυτό κατά την εκκίνησή της, και η οποία θα ξεκινά μαζί με το λειτουργικό σύστημα.

Το Upstart είναι ένα εργαλείο του Ubuntu το οποίο εκκινεί υπηρεσίες συστήματος των Ubuntu. Οι υπηρεσίες που εκκινεί ορίζονται σε αρχεία με την κατάληξη `.conf` τα οποία βρίσκονται στο φάκελο `/etc/init/`. Κατά την εκκίνηση του συστήματος δηλαδή, το Upstart φορτώνει όλα τα αρχεία με κατάληξη `.conf` από αυτό το φάκελο και ορίζει υπηρεσίες σύμφωνα με τις ρυθμίσεις που αυτά περιέχουν. Για να δημιουργήσουμε μια τέτοια υπηρεσία συστήματος, δημιουργήσαμε ένα αρχείο με όνομα `citytrack-web.conf` και το τοποθετήσαμε στο φάκελο `/etc/init/`. Τα περιεχόμενα του αρχείου φαίνονται παρακάτω:

```
description "node.js forever server for citytrack-web"
author      "Kostas Stamatoukos <stamatoukosks@gmail.com>"
version     "1.0"
expect fork
# Define start and stop
start on started mountall
stop on shutdown
# Automatically Respawn:
respawn
respawn limit 99 5
env HOME=/var/www/citytrack-web
# Script to be executed during startup
script
  export HOME=$HOME
  chdir $HOME
  exec /usr/local/bin/node server.js > logs/node.log &
end script
```

Κώδικας 6.9 Ορισμός υπηρεσίας Upstart για τον εξυπηρετητή

Ως μέρος της υπηρεσίας δηλαδή, εκτελούμε το αρχείο που εκκινεί τον εξυπηρετητή με χρήση του Forever.js όπως το ορίσαμε στην προηγούμενη ενότητα (6.3.1).

6.3.3 Αυτόματη εγκατάσταση του εξυπηρετητή web από το CI

Για να εγκαταστήσουμε τον εξυπηρετητή web χρησιμοποιούμε το CI που περιγράψαμε λεπτομερώς στην ενότητα 5.1.6. Δημιουργήσαμε δηλαδή μία διεργασία στον Jenkins η οποία όταν ανιχνεύσει αλλαγές στον κώδικα που βρίσκεται στην απομακρυσμένη αποθήκη κώδικα, θα εγκαταστήσει τη νέα έκδοση στο μηχάνημα στόχο έστω TARGET.

Για την εγκατάσταση πραγματοποιεί τα ακόλουθα βήματα:

1. Συμπιέζει τον νέο κώδικα και τον αντιγράφει στο μηχάνημα στόχο σε προκαθορισμένο φάκελο
2. Συνδέεται στο μηχάνημα στόχο και
    a. Αποσυμπιέζει το αρχείο
    b. Εγκαθιστά τυχόν εξαρτήσεις από το NPM
    c. Χτίζει την εφαρμογή μέσω Gulp

Το πρόγραμμα που εκτελεί τα ανωτέρω βήματα παρατίθεται στη συνέχεια:

```
#!/bin/bash
mkdir -p build
echo "Compressing and copying code to server"
tar --exclude='./node_modules' --exclude='./bower_components' \
   --exclude='./build' --exclude='./.git'\
   --exclude='./.idea' -czf build/web.tar.gz .
scp build/web.tar.gz USER@TARGET:/var/www/citytrack-web/

echo "Connecting to server to deploy"
ssh  USER@TARGET << EOF
      ## These steps are executed in the target machine
      echo “Unzipping content”
      cd /var/www/citytrack-web
      gunzip –f web.tar.gz
      tar –xf web.tar
      rm –rf web.tar
      echo “Installing and building”
      npm install
      gulp
EOF
```

Κώδικας 6.10 Πρόγραμμα γραμμής εντολών που εκτελεί την εγκατάσταση νέας έκδοσης εφαρμογής

Το παραπάνω πρόγραμμα προϋποθέτει βέβαια ότι ο φάκελος `/var/www/citytrack-web` υπάρχει, και ότι ο χρήστης USER έχει προνόμια εγγραφής στο φάκελο αυτό.

Ο λόγος που το script αυτό θα οδηγήσει σε αναβάθμιση της τρέχουσας εφαρμογής είναι η χρήση του Forever.js όπως περιγράψαμε στην ενότητα 6.3.1

6.3.4 Εκτέλεση υπηρεσίας συστήματος για τον εξυπηρετητή web για πρώτη φορά

Την πρώτη φορά που το CI θα αντιγράψει τα αρχεία στο φάκελο `/var/www/citytrack-web` δε θα γίνει τίποτα αν η υπηρεσία του εξυπηρετητή δεν έχει ξεκινήσει.

Για να εκκινήσουμε την υπηρεσία αυτή όπως την ορίσαμε στην ενότητα 6.3.2 πρέπει να εκτελέσουμε τον ακόλουθο κώδικα:

```
sudo su
start citytrack-web
```

Όπως βλέπουμε, η υπηρεσία αυτή εκκινείται με δικαιώματα superuser. Είναι δυνατόν να ξεκινήσουμε την εφαρμογή και ως non-super-user μέσω του Upstart αλλά για τις ανάγκες της διπλωματικής εργασίας δεν συνεχίσαμε ως εκεί.

7

Πειραματική αξιολόγηση

7.1 Αλγόριθμος εύρεσης Περιοχών Ενδιαφέροντος

Με βάση την ανάλυση που έγινε στην ενότητα 4.2 η πολυπλοκότητα του αλγορίθμου στη γενική περίπτωση είναι $O(N^2)$. Ωστόσο, πετυχαίνουμε μεγάλη βελτίωση της απόδοσης του αλγορίθμου με τη χρήση ενός Grid όπως αυτό που περιγράψαμε στην ενότητα 4.1.1.

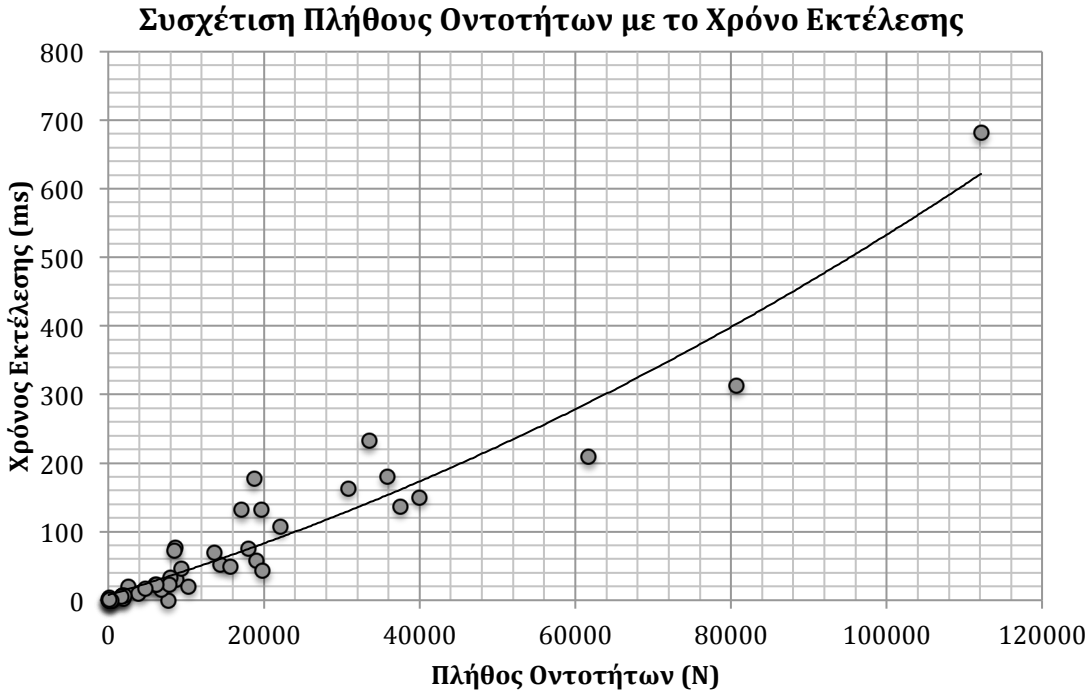
Ο αλγόριθμος εύρεσης περιοχών δέχεται ως είσοδό του μία περιοχή στο χάρτη και το όνομα μιας κατηγορίας, και επιστρέφει ένα σύνολο από περιοχές ενδιαφέροντος. Στον πίνακα που ακολουθεί παραθέτουμε αντιπροσωπευτικά το χρόνο εξυπηρέτησης τέτοιων αιτημάτων συμπεριλαμβανομένου του χρόνου Εισόδου/Εξόδου (ΕΕ) που χρειάζεται σε κάθε περίπτωση αλλά και χωρίς αυτόν.

Πόλη	Κατηγορία	Πλήθος Σημείων	Χρόνος με ΕΕ (ms)	Χρόνος χωρίς ΕΕ (ms)
Αθήνα	Φαγητό	17110	2865	131
Αθήνα	Πολιτισμός	1823	484	6
Βερολίνο	Φαγητό	37583	7180	137
Βερολίνο	Πολιτισμός	3945	671	10
Λονδίνο	Φαγητό	80747	15282	313
Λονδίνο	Πολιτισμός	6028	1323	22
Βιέννη	Φαγητό	18072	2533	75
Βιέννη	Πολιτισμός	1987	655	7

Πίνακας 7.1 Χρόνοι υπολογισμού των Περιοχών Ενδιαφέροντος για διάφορες περιοχές και κατηγορίες.

Με βάση τον Πίνακας 7.1 Χρόνοι υπολογισμού των Περιοχών Ενδιαφέροντος για διάφορες περιοχές και κατηγορίες., γίνεται αμέσως αντιληπτό ότι ο περισσότερος χρόνος κατά την εξυπηρέτηση ενός αιτήματος «σπαταλάται» στην αναμονή για τη λήψη των δεδομένων από τις πηγές δεδομένων και την αποθήκευσή τους στη μνήμη. Αντίθετα, η εκτέλεση της λογικής του αλγορίθμου είναι ταχύτατη. Αυτό σημαίνει ότι θα μπορούσαμε να βελτιώσουμε σημαντικά την απόδοση του αλγορίθμου αν επιτυγχάναμε γρηγορότερη ΕΕ. Για παράδειγμα με καλύτερη διαχείριση/partitioning της βάσης δεδομένων και του Solr, εκμετάλλευση του παραλληλισμού για την απόκτηση των αποτελεσμάτων ή εξασφάλιση ταχύτερων μηχανημάτων για τη βάση δεδομένων και το σύστημα Solr. Ωστόσο, αυτό είναι κάτι που ξεφεύγει από τα πλαίσια της διπλωματικής και αφήνεται ως μελλοντική βελτίωση.

Στη συνέχεια, στο γράφημα που ακολουθεί, παρουσιάζουμε ένα ευρύτερο σύνολο μετρήσεων. Συγκεκριμένα, δείχνουμε τη συσχέτιση του χρόνου εκτέλεσης του αλγορίθμου με το πλήθος των οντοτήτων που δίδεται ως είσοδος.



Γράφημα 7.1 Γραφική αναπαράσταση του χρόνου εκτέλεσης του αλγορίθμου εύρεσης Περιοχών Ενδιαφέροντος σε σχέση με το πλήθος των οντοτήτων που δίδονται ως είσοδος. Στο χρόνο δε συμπεριλαμβάνεται ο χρόνος εισόδου/εξόδου

Από το γράφημα, διαπιστώνουμε ότι ο χρόνος εκτέλεσης σε σχέση με το πλήθος των οντοτήτων που δίνονται ως είσοδος αυξάνει γραμμικά για τα μεγέθη (N) που μας

ενδιαφέρουν. Οι διακυμάνσεις που παρατηρούνται έχουν να κάνουν με την κατανομή των δεδομένων η οποία μπορεί να επηρεάσει σημαντικά την απόδοση του ευρετηρίου Grid.

7.2 Αλγόριθμος εύρεσης οδών ενδιαφέροντος

Όπως αναφέραμε στην ενότητα 4.3 η πολυπλοκότητα του αλγορίθμου αυτού δίδεται από τον τύπο: $O(N + MN\frac{L}{d})$ όπου N είναι το πλήθος των σημείων, M το πλήθος των οδών, L το μέσο μήκος των οδών και d το μήκος του κελιού με το οποίο κατασκευάζουμε το grid. Για τις οδούς ενδιαφέροντος το d είναι πάντα ίσο με τη μέγιστη απόσταση από την οποία μπορεί να απέχει ένα σημείο από μία οδό για να θεωρείται ότι είναι στη γειτονιά της.

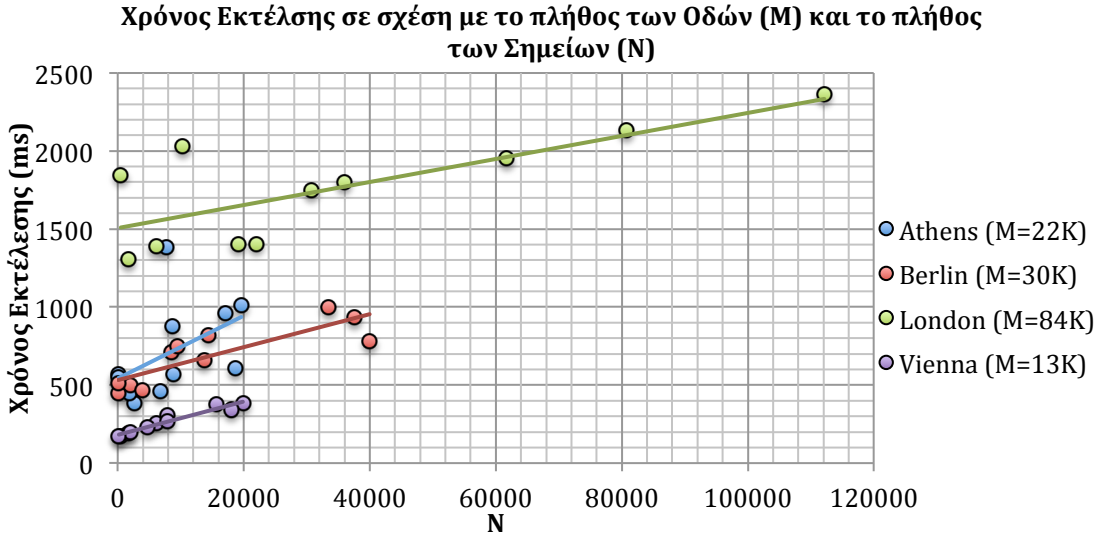
Στον πίνακα που ακολουθεί δείχνουμε το χρόνο εκτέλεσης του αλγορίθμου εύρεσης των Οδών Ενδιαφέροντος σε μία περιοχή, συμπεριλαμβανομένου του χρόνου Εισόδου/Εξόδου (ΕΕ) αλλά και χωρίς αυτόν.

Πόλη	Κατηγορία	Πλήθος Σημείων	Πλήθος Οδών	Χρόνος με ΕΕ (ms)	Χρόνος χωρίς ΕΕ (ms)
Αθήνα	Φαγητό	17110	21793	6776	956
Αθήνα	Πολιτισμός	1823	21793	5179	446
Βερολίνο	Φαγητό	37583	29908	10788	931
Βερολίνο	Πολιτισμός	3945	29908	5729	464
Λονδίνο	Φαγητό	80747	83932	26839	2133
Λονδίνο	Πολιτισμός	6028	83932	13243	1389
Βιέννη	Φαγητό	18072	12982	4975	341
Βιέννη	Πολιτισμός	1987	12982	2896	198

Πίνακας 7.2 Χρόνοι υπολογισμού των Οδών Ενδιαφέροντος για διάφορες περιοχές και κατηγορίες

Όπως και στην προηγούμενη ενότητα (7.1), παρατηρούμε ότι ο χρόνος εισόδου/εξόδου που φαίνεται στον Πίνακας 7.2 αποτελεί το μεγαλύτερο τμήμα του χρόνου που χρειάζεται ο αλγόριθμος για να επιστρέψει το αποτέλεσμά του. Σε αντίθεση, ο αλγόριθμος της αντιστοίχισης εκτελείται ταχύτατα, αφού στη χειρότερη περίπτωση χρειάζεται λιγότερο από 2.5 δευτερόλεπτα.

Στη συνέχεια, παρουσιάζουμε το αντίστοιχο γράφημα το οποίο περιέχει ένα ευρύτερο σύνολο μετρήσεων:



Γράφημα 7.2 Γραφική αναπαράσταση του χρόνου υπολογισμού Οδών ενδιαφέροντος σε σχέση με το πλήθος των οδών M και το πλήθος των σημείων N

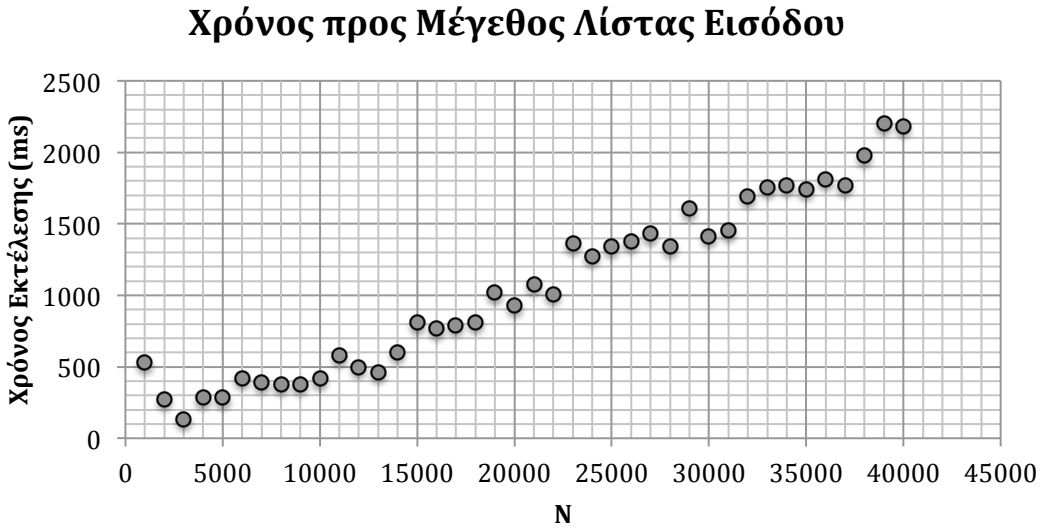
Με βάση τις παραπάνω τιμές εξάγουμε τα εξής συμπεράσματα:

Τόσο το N όσο και το M επηρεάζουν σημαντικά το χρόνο εκτέλεσης. Για παράδειγμα στο Λονδίνο ακόμη και αναζητήσεις σε κατηγορίες με μικρό αριθμό σημείων ο χρόνος είναι αρκετά υψηλότερος από τον αντίστοιχο χρόνο σε περιοχές με αρκετά μικρότερο αριθμό οδών. Επίσης, βλέπουμε πως με την αύξηση του N ο χρόνος τείνει να αυξάνεται σχεδόν γραμμικά με αρκετά μικρή λύση ωστόσο χάρη στο ευρετήριο Grid.

7.3 Αλγόριθμος Ποικίλων Φωτογραφιών

Όπως περιγράψαμε στην ενότητα 4.4.4 η πολυπλοκότητα του αλγορίθμου εύρεσης ποικίλων φωτογραφιών εντοπίζεται κατά κύριο λόγο στον υπολογισμό των όμοιων ομάδων με βάση τη λίστα των φωτογραφιών που παρέχονται στον αλγόριθμο ως είσοδος. Όπως αναφέραμε λοιπόν η πολυπλοκότητα του αλγορίθμου στη γενική περίπτωση είναι $O(N^2)$ και γι αυτό το λόγο χρησιμοποιήσαμε ένα δέντρο Vantage Point με σκοπό να επιταχύνουμε τον αλγόριθμο.

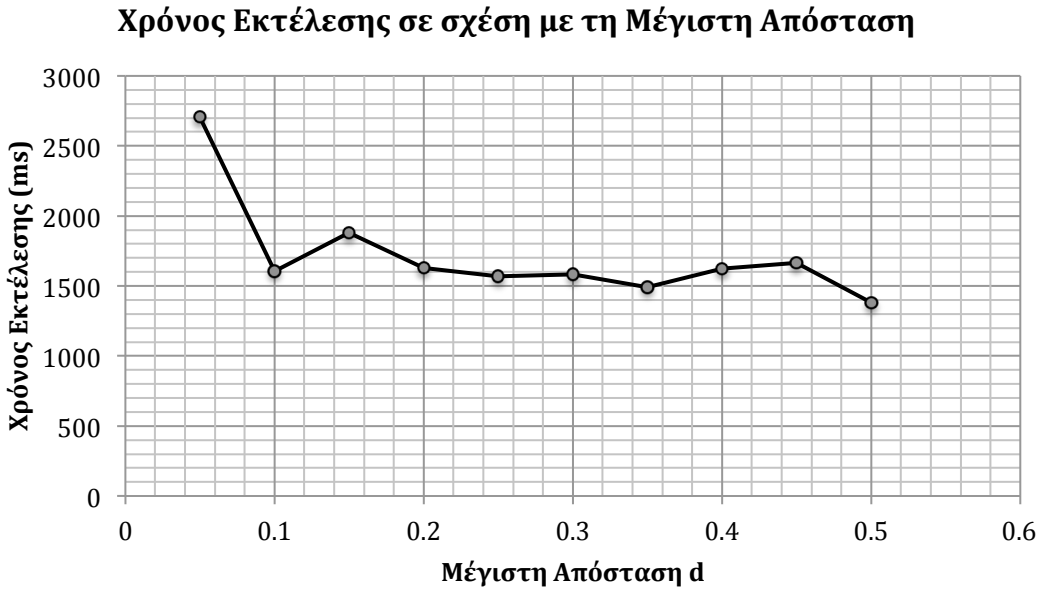
Στη συνέχεια παρουσιάζουμε την καμπύλη απόδοσης του αλγορίθμου σε σχέση με το μέγεθος της εισόδου για μία συγκεκριμένη «ακτίνα» (απόσταση οντοτήτων από την κεντρική οντότητα.



Γράφημα 7.3 Χρόνος(ms) σε σχέση με το μέγεθος της λίστας εισόδου

Όπως βλέπουμε στο Γράφημα 7.3 η αύξηση του χρόνου εκτέλεσης ως προς το μέγεθος της εισόδου αυξάνεται γραμμικά σε σχέση με το μέγεθος της εισόδου. Ακόμη και σε ακραίες περιπτώσεις όπου ο αλγόριθμος δέχεται είσοδο μεγέθους 40,000 απαντά σε 2 περίπου δευτερόλεπτα. Αναφορικά να σημειώσουμε εδώ ότι, μέχρι στιγμής, ποτέ δε συναντήσαμε στην εφαρμογή μέγεθος εισόδου μεγαλύτερο από 20,000 φωτογραφίες.

Ένα άλλο γράφημα που παρουσιάζει ενδιαφέρον για τον αλγόριθμο αυτό είναι εκείνο που συσχετίζει το χρόνο εκτέλεσης με την ακτίνα αναζήτησης γειτόνων, δηλαδή τη μέγιστη απόσταση d με βάση την οποία δύο οντότητες θεωρούνται γειτονικές.



Γράφημα 7.4 Χρόνος εκτέλεσης σε σχέση με τη μέγιστη απόσταση d για σταθερό μέγεθος εισόδου N

Όπως βλέπουμε, στην περίπτωση αυτή ο χρόνος εκτέλεσης παραμένει πρακτικά σταθερός όσο αυξάνεται η μέγιστη απόσταση d ένα κατώφλι όπως το ορίσαμε στην ενότητα 4.4.4. Ενώ για πολύ μικρές αποστάσεις έχουμε μια μικρή επιβράδυνση η οποία θα ήταν πολύ μεγαλύτερη χωρίς ένα VP δένδρο. Αυτό σημαίνει ότι η επιλογή της μέγιστης απόστασης d μπορεί να γίνει με βάση των αριθμό των clusters που θέλουμε να επιστρέφονται, χωρίς να μας ανησυχεί ο χρόνος επίδοσης.

8

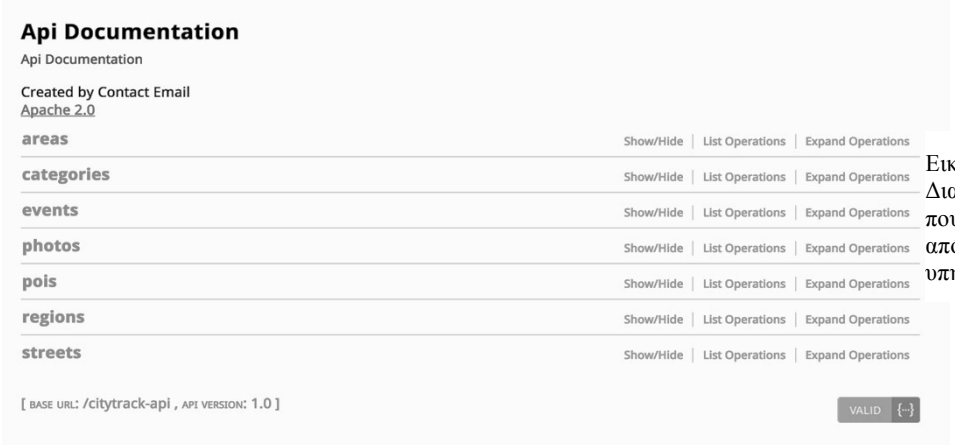
Περιγραφή της εφαρμογής

8.1 Περιγραφή της διεπαφής υπηρεσιών

Η διεπαφή υπηρεσιών την οποία περιγράψαμε συνοπτικά στην ενότητα 2.3 είναι υπεύθυνη για την άντληση των δεδομένων από τις πηγές δεδομένων, την επεξεργασία και ανάλυσή τους με τη χρήση διάφορων αλγορίθμων που περιγράψαμε στα προηγούμενα κεφάλαια και, τέλος, την προώθηση των αποτελεσμάτων στο χρήστη.

Στο κεφάλαιο αυτό περιγράφουμε αναλυτικά όλες τις διαφορετικές υπηρεσίες που εκτίθενται μέσα από τη διεπαφή υπηρεσιών. Για το σκοπό αυτό θα χρησιμοποιήσουμε το γραφικό περιβάλλον που παρέχεται από το SwaggerUI για την περιγραφή της διεπαφής υπηρεσιών.

Αρχικά, οι υπηρεσίες είναι χωρισμένες με βάση το είδος τους σε διαφορετικούς πόρους, όπως φαίνεται στην εικόνα που ακολουθεί:



Εικόνα 8.1 Διαφορετική πόροι που παρέχονται από τη διεπαφή υπηρεσιών

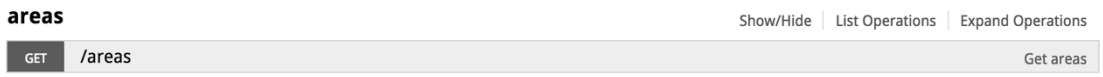
Οι πόροι αυτοί είναι, ονομαστικά, οι ακόλουθοι:

- Εκτάσεις (areas)
- Κατηγορίες (categories)
- Εκδηλώσεις (events)
- Φωτογραφίες (photos)
- Σημεία Ενδιαφέροντος (pois)
- Περιοχές (regions)
- Οδοί (streets)

Στη συνέχεια θα δείξουμε τα σημεία πρόσβασης που προσφέρει η διεπαφή υπηρεσιών για κάθε πόρο και θα περιγράψουμε σύντομα κάθε ένα από αυτά. Δε θα τα περιγράψουμε ωστόσο με μεγάλη λεπτομέρεια αφού η περιγραφή αυτή είναι ανά πάσα στιγμή διαθέσιμη (και ενημερωμένη) στο SwaggerUI της εφαρμογής όπως περιγράψαμε αναλυτικότερα στην ενότητα 5.1.4.

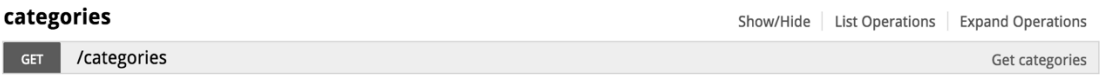
8.1.1 Ο Πόρος «Εκτάσεις»

Στον πόρο αυτό ορίζεται μόνο ένα σημείο πρόσβασης, το οποίο επιστρέφει όλες τις εκτάσεις που είναι ορισμένες στη βάση δεδομένων.



8.1.2 Ο Πόρος «Κατηγορίες»

Στον πόρο αυτό ορίζεται επίσης μόνο ένα σημείο πρόσβασης, το οποίο επιστρέφει όλες τις κατηγορίες πρώτου επιπέδου που υπάρχουν στη βάση δεδομένων.



8.1.3 Οι Πόροι «Σημεία Ενδιαφέροντος», «Φωτογραφίες» και «Εκδηλώσεις»

Θα περιγράψουμε τους τρεις αυτούς πόρους στην ίδια ενότητα, καθώς παρουσιάζουν πολλές ομοιότητες στη δομή τόσο των παραμέτρων εισόδου όσο και των αποτελεσμάτων που επιστρέφουν. Η μόνη διαφορά τους έγκειται στον τύπο των οντοτήτων που επιστρέφουν.

Συνοπτικά, και οι τρεις πόροι μπορούν να αναζητούν οντότητες που περιέχονται: (α) σε ορθογώνιες εκτάσεις, (β) γύρω από ένα σημείο με βάση μία απόσταση από αυτό και (γ) οντότητες που περιέχονται σε ένα κυρτό πολύγωνο που ορίζεται ως μία σειρά σημείων.

Επίσης, και οι τρεις πόροι μπορούν να φιλτράρουν τις οντότητες με βάση τις ακόλουθες πληροφορίες

- Λέξη κλειδί
- Πηγή (πχ Foursquare, Wikimapia κλπ)
- Κατηγορία (Μόνο για Σημεία Ενδιαφέροντος)

Τέλος, και οι τρεις πόροι μπορούν να σελιδοποιούν τις οντότητες καθώς και να τις ταξινομούν με βάση τις επιλογές του χρήστη.

Στη συνέχεια φαίνονται οι εν λόγω πόροι και τα σημεία εισόδου τους.

8.1.4 Ο πόρος «Περιοχές»

Ο πόρος «Περιοχές» ορίζει σημεία εισόδου που επιτρέπουν τον υπολογισμό και την αλληλεπίδραση με Περιοχές Ενδιαφέροντος.

Συγκεκριμένα όπως φαίνεται στην εικόνα που ακολουθεί εκθέτει τρία διαφορετικά σημεία εισόδου:

Αναφορικά:

- Σημείο εισόδου για τον υπολογισμό των Περιοχών Ενδιαφέροντος σε μία Έκταση (Τετράγωνη ή Κυκλική)

- Σημείο εισόδου για τον υπολογισμό ποικίλων φωτογραφιών μιας Περιοχής Ενδιαφέροντος
- Σημείο εισόδου για τον υπολογισμό ενός νέφους λέξεων που περιγράφουν μία Περιοχή Ενδιαφέροντος

8.1.5 Ο πόρος «Οδοί»

Ο πόρος «Οδοί» ορίζει σημεία εισόδου σχετικά με τον υπολογισμό και την αλληλεπίδραση με Οδούς Ενδιαφέροντος.

Συγκεκριμένα όπως φαίνεται στην εικόνα που ακολουθεί εκθέτει πέντε διαφορετικά σημεία εισόδου:

Το σημείο εισόδου `/streets` υπολογίζει τις Οδούς Ενδιαφέροντος που περιέχονται σε μία τετράγωνη Έκταση με βάση μια επιλεγμένη Κατηγορία.

Το σημείο εισόδου `/scenicStreets` υπολογίζει τις Γραφικές Οδούς που περιέχονται σε μία τετράγωνη Έκταση. Οι φωτογραφίες δεν έχουν κάποια κατηγορία, οπότε η έκταση αποτελεί τη μοναδική παράμετρο που δέχεται αυτό το σημείο εισόδου.

Το σημείο εισόδου `/streets/{streetId}/diversePhotos` υπολογίζει ένα σύνολο από ποικίλες και σχετικές φωτογραφίες για την οδό τις οποίας το χαρακτηριστικό (streetId) παρέχεται ως είσοδος.

Το σημείο εισόδου `/streets/{streetId}/photos` επιστρέφει όλες τις Φωτογραφίες που αντιστοιχίζονται με την Οδό τις οποίας το χαρακτηριστικό (streetId) παρέχεται ως είσοδος. Οι φωτογραφίες επιστρέφονται σελιδοποιημένες.

Το σημείο εισόδου `/streets/{streetId}/pois` επιστρέφει όλα τα Σημεία Ενδιαφέροντος που αντιστοιχίζονται με την Οδό τις οποίας το χαρακτηριστικό (streetId) παρέχεται ως είσοδος. Τα Σημεία Ενδιαφέροντος επιστρέφονται σελιδοποιημένα.

Τέλος, το σημείο εισόδου `/streets/{streetId}/tagCloud` υπολογίζει ένα νέφος λέξεων που περιγράφουν ικανοποιητικά την οδό της οποίας το χαρακτηριστικό (streetId) παρέχεται ως είσοδος.

8.2 Περιγραφή του γραφικού περιβάλλοντος

Το γραφικό περιβάλλον της εφαρμογής επιτρέπει στους χρήστες να περιηγηθούν στα δεδομένα που παρέχει η διεπαφή υπηρεσιών με ένα γραφικό τρόπο. Στο κεφάλαιο αυτό θα δείξουμε τον τρόπο με τον οποίο μπορούν να εκτελεστούν οι διάφορες αναζητήσεις καθώς και τί δυνατότητες προσφέρει η εφαρμογή μας στους χρήστες της.

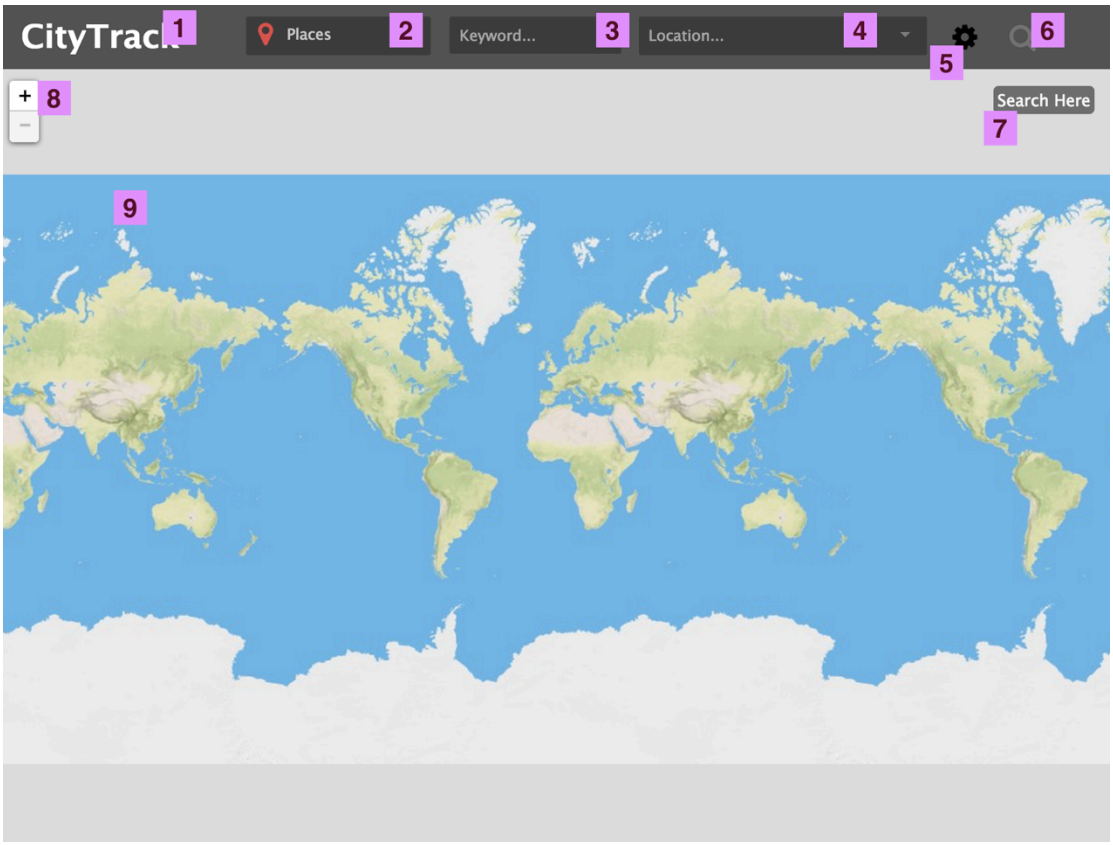
Η λειτουργικότητα που παρέχει το γραφικό περιβάλλον συνοψίζεται στα εξής σημεία:

- Αναζήτηση και φιλτράρισμα Σημείων Ενδιαφέροντος, Φωτογραφιών ή Εκδηλώσεων οι οποίες ανήκουν σε μία έκταση που ορίζεται μέσω ενός χάρτη από το χρήστη.
- Αναζήτηση Σημείων Ενδιαφέροντος, Φωτογραφιών ή Εκδηλώσεων που είναι σχετικά με άλλα Σημεία Ενδιαφέροντος, Φωτογραφίες ή Εκδηλώσεις. Συσχετισμός δηλαδή των οντοτήτων μεταξύ τους.
- Εύρεση Περιοχών Ενδιαφέροντος σε μία Έκταση του χάρτη και με βάση μια Κατηγορία.
- Εύρεση Οδών Ενδιαφέροντος σε μία Έκταση του χάρτη και με βάση μια Κατηγορία.
- Συνόψιση των Περιοχών Ενδιαφέροντος και των Οδών Ενδιαφέροντος με Σχετικές και Ποικίλες Φωτογραφίες καθώς και με ένα Νέφος Λέξεων.
- Εύρεση Οντοτήτων όπως «Σημεία Ενδιαφέροντος» ή «Φωτογραφίες» μέσα σε μια «Περιοχή Ενδιαφέροντος» ή κοντά σε μία «Οδό Ενδιαφέροντος».

Βέβαια η λειτουργικότητα της εφαρμογής δεν περιορίζεται αυστηρά σε αυτά τα έξι σημεία, αφού ως τμήμα της λειτουργικότητας της εφαρμογής θα πρέπει να θεωρηθούν και ένα σύνολο από γραφικές ή λειτουργικές βελτιστοποιήσεις που διευκολύνουν το χρήστη να περιηγηθεί στα δεδομένα και να τροποποιήσει την αναζήτησή του.

8.2.1 Περιγραφή δομής της σελίδας έναρξης της εφαρμογής

Όταν ανοίγει κανείς την εφαρμογή για πρώτη φορά βλέπει μπροστά του το γραφικό περιβάλλον που παρατίθεται στην Εικόνα 8.1. Στην εικόνα αυτή έχουμε σημειώσει με αριθμούς τα διάφορα σημεία με τα οποία μπορούν να αλληλεπιδράσουν οι χρήστες.



Εικόνα 8.2 Σελίδα έναρξης της εφαρμογής

Ακολουθεί η περιγραφή κάθε αριθμημένου σημείου:

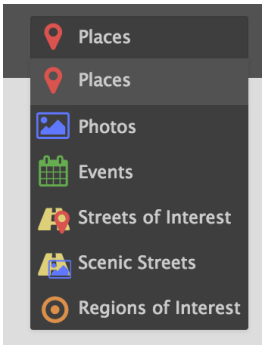
1. Το λογότυπο της εφαρμογής.
2. Η λίστα επιλογής του τύπου των οντοτήτων που θέλουμε να αναζητήσουμε. Μπορεί να είναι «Σημεία Ενδιαφέροντος», «Φωτογραφίες», «Εκδηλώσεις», «Οδοί Ενδιαφέροντος», «Γραφικές Οδοί» και «Περιοχές Ενδιαφέροντος»
3. Η λέξη κλειδί της αναζήτησης. Το πεδίο αυτό είναι διαθέσιμο μόνο στην περίπτωση που ο επιλεγμένος τύπος οντοτήτων είναι «Σημεία Ενδιαφέροντος», «Φωτογραφίες» ή «Εκδηλώσεις».
4. Λίστα επιλογής της έκτασης στην οποία θέλουμε να γίνει η αναζήτηση. Περιέχει τις εκτάσεις που επιστρέφονται από τη βάση καθώς και τις τελευταίες εκτάσεις αναζήτησης του χρήση από τη στιγμή που άνοιξε την εφαρμογή. Η λίστα αυτή δεν είναι ο μόνος τρόπος επιλογής της έκτασης αναζήτησης όπως θα δούμε στη συνέχεια.
5. Το κουμπί αυτό ανοίγει το παράθυρο με τις τροποποιήσεις που μπορούν να γίνουν ανάλογα με τον τύπο αναζήτησης. Αν το παράθυρο είναι άδειο τότε δεν υπάρχουν τροποποιήσεις.

6. Το κουμπί αναζήτησης. Όταν ο χρήστης έχει επιλέξει όλα τα απαιτούμενα δεδομένα για τον επιλεγμένο τύπο το κουμπί αυτό είναι ενεργό. Αλλιώς παραμένει απενεργοποιημένο
7. Κουμπί επιλογής της τρέχουσας «θέας» στο χάρτη ως έκταση αναζήτησης. Αν το κουμπί αυτό πατηθεί τότε η έκταση αναζήτησης τίθεται στην έκταση που φαίνεται στο χάρτη
8. Κουμπιά εστίασης/από-εστίασης από το χάρτη
9. Ο χάρτης στον οποίο απεικονίζονται όλα τα αποτελέσματα. Ο χρήστης μπορεί επίσης να αλληλεπιδρά με το χάρτη για να θέτει την περιοχή αναζήτησης με ένα σύνολο τρόπων που θα δούμε στη συνέχεια.

8.2.2 Εκτέλεση βασικής αναζήτησης για διάφορους τύπους

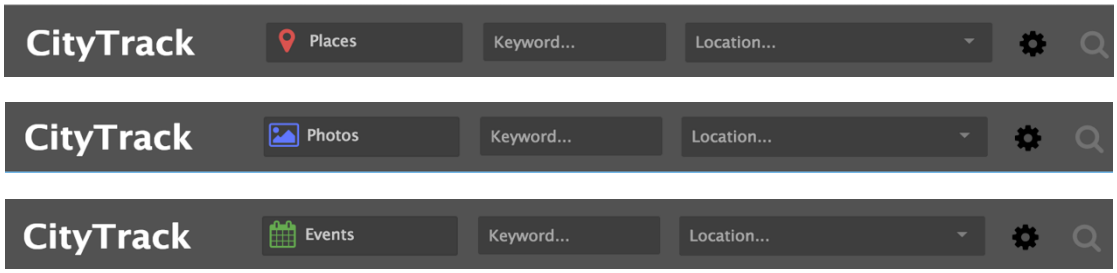
Όταν ο χρήστης ανοίξει την εφαρμογή, το επόμενο βήμα είναι να πραγματοποιήσει κάποια αναζήτηση για έναν από τους παρεχόμενους τύπους οντοτήτων. Στην υπό-ενότητα αυτή θα περιγράψουμε τον τρόπο βασικής αναζήτησης, όπου συμπληρώνουμε μόνο τα απαραίτητα πεδία για τους διάφορους τύπους οντοτήτων. Οι επιπλέον δυνατότητες παραμετροποίησης της αναζήτησης θα παρουσιαστούν σε επόμενη ενότητα.

Αρχικά, η επιλογή τύπου οντότητας από τη λίστα επιλογής τύπου φαίνεται στην Εικόνα 8.4. Κάθε ένας από τους τύπους αυτούς, όπως είπαμε πιο πάνω, απαιτεί ένα σύνολο από διαφορετικές παραμέτρους ώστε να μπορέσει η εφαρμογή να πραγματοποιήσει τη σχετική αναζήτηση. Η γραμμή αναζήτησης προσαρμόζεται αυτόματα, ώστε να περιέχει όλες τις απαιτούμενες παραμέτρους ανάλογα με τον επιλεγμένο τύπο. Στη συνέχεια θα παρουσιάσουμε αυτές τις παραμέτρους για κάθε έναν από τους τύπους που φαίνονται στη διπλανή εικόνα.



Εικόνα 8.4 Λίστα επιλογής "Τύπου"

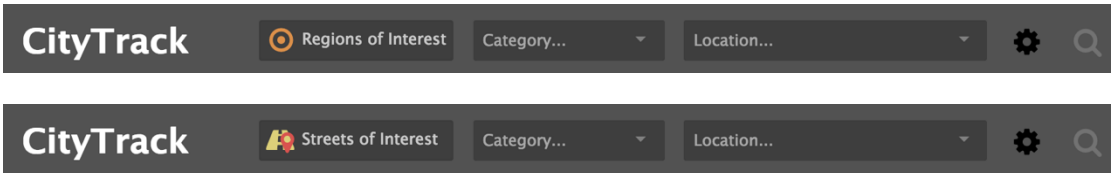
Οι τύποι «Σημεία Ενδιαφέροντος», «Φωτογραφίες» και «Εκδηλώσεις απαιτούν ακριβώς τις



Εικόνα 8.3 Αναζήτηση για τους τύπους Σημεία Ενδιαφέροντος, Φωτογραφίες και Εκδηλώσεις

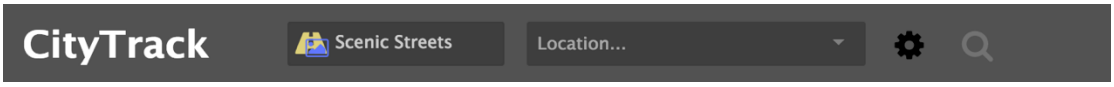
ίδιες παραμέτρους εισόδου, όπως βλέπουμε στην Εικόνα 8.3 που ακολουθεί. Συγκεκριμένα απαιτούν μόνο μία λέξη κλειδί και μία τοποθεσία/έκταση και έπειτα είναι σε θέση να εκτελέσουν μια αναζήτηση.

Έπειτα, οι «Οδοί Ενδιαφέροντος» και οι «Περιοχές Ενδιαφέροντος» απαιτούν επίσης μια τοποθεσία, αλλά στη θέση της λέξης κλειδί, απαιτούν μία κατηγορία. Η γραμμή αναζήτησης για αυτούς τους τύπους παρατίθεται στην Εικόνα 8.6



Εικόνα 8.6 Γραμμή αναζήτησης για τους τύπους Περιοχές Ενδιαφέροντος και Οδοί Ενδιαφέροντος

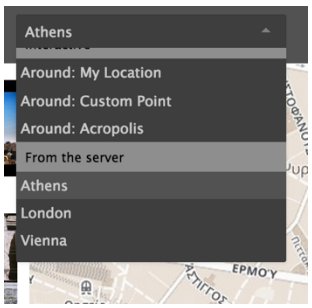
Τέλος, για τον τύπο «Γραφικές Οδοί» απαιτείται μόνο μια τοποθεσία. Αυτό γιατί οι Γραφικές Οδοί ορίζονται με βάση την πυκνότητά τους σε φωτογραφίες, και οι φωτογραφίες δεν ανήκουν σε κάποια κατηγορία. Η Εικόνα 8.5 δείχνει τη γραμμή αναζήτησης για τον τύπο αυτό.



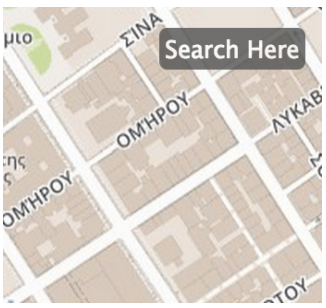
Εικόνα 8.5 Η γραμμή αναζήτησης για τον τύπο "Γραφικές Οδοί"

8.2.3 Διαφορετικοί τρόποι επιλογής Έκτασης/Τοποθεσίας Αναζήτησης

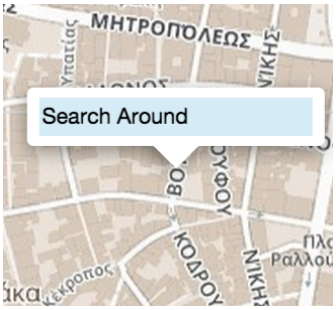
Όπως συνοπτικά αναφέρθηκε σε προηγούμενη ενότητα η εφαρμογή δίνει τη δυνατότητα στο χρήστη να επιλέξει την έκταση αναζήτησης με τρεις διαφορετικούς τρόπους. Παρακάτω φαίνονται σε τρεις εικόνες οι διαφορετικοί τρόποι επιλογής έκτασης αναζήτησης.



Εικόνα 8.8 Επιλογή έκτασης από τη λίστα (α)



Εικόνα 8.5 Επιλογή έκτασης από το χάρτη (β)



Εικόνα 8.8 Επιλογή έκτασης τριγύρω από σημείο στο χάρτη (γ)

Όπως φαίνεται στις εικόνες, ο χρήστης μπορεί να επιλέξει μία έκταση αναζήτησης (α) χρησιμοποιώντας τη λίστα επιλογής τοποθεσίας, (β) χρησιμοποιώντας το κουμπί «αναζήτηση

εδώ» στο χάρτη και (γ) κάνοντας δεξί κλικ οπουδήποτε πάνω στο χάρτη και επιλέγοντας «Αναζήτηση Τριγύρω».

Η λίστα τοποθεσίας περιέχει τόσο στατικές εκτάσεις που προέρχονται από τη βάση δεδομένων όσο και δυναμικές εκτάσεις. Δυναμικές εκτάσεις μπορεί να είναι:

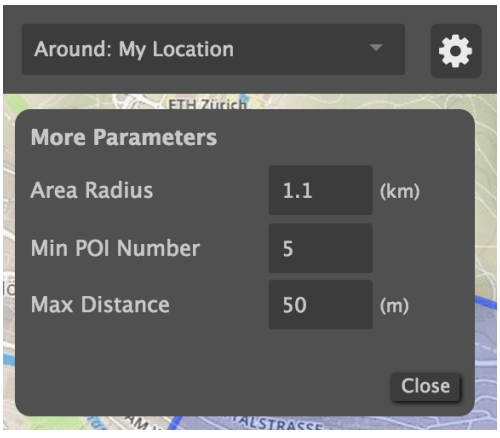
- Η τελευταία θέα του χάρτη που επιλέχθηκε κάνοντας κλικ στο κουμπί «Αναζήτηση Εδώ»
- Η τελευταία κυκλική έκταση που επιλέχθηκε κάνοντας δεξί κλικ στο χάρτη και επιλέγοντας «Αναζήτηση Τριγύρω»
- Οποιαδήποτε περιοχή που προέρχεται από «Ενέργειες» του παραθύρου λεπτομερειών κάποιας οντότητας
- Η έκταση γύρω από την τρέχουσα τοποθεσία του χρήστη

Εδώ ας σημειώσουμε ότι οι κυκλικές περιοχές από προεπιλογή ορίζονται με ακτίνα 1km. Ωστόσο το μέγεθος μπορεί να αλλάξει εύκολα όπως θα δείξουμε στην επόμενη ενότητα, που είναι σχετική με την περαιτέρω παραμετροποίηση της αναζήτησης.

8.2.4 Παραμετροποίηση αναζήτησης

Στις προηγούμενες ενότητες είδαμε πώς μπορεί ο χρήστης να εκτελέσει αναζητήσεις συμπληρώνοντας τα βασικά πεδία που απαιτούνται σε κάθε αναζήτησης και τα οποία δεν έχουν κάποιες προεπιλεγμένες τιμές. Στην ενότητα αυτή θα δούμε πώς αυτές οι αναζητήσεις μπορούν να παραμετροποιηθούν περαιτέρω.

Όπως είδαμε στην ενότητα 8.2.1, δίπλα στο κουμπί αναζήτησης υπάρχει το κουμπί παραμετροποίησης. Όταν το κουμπί αυτό πατηθεί εμφανίζει στο χρήστη όλες τις παραμέτρους που μπορεί να αλλάξει ανάλογα με τον τύπο και την περιοχή που έχει επιλέξει. Στην Εικόνα 8.9 που φαίνεται δίπλα, βλέπουμε όλες τις πιθανές παραμέτρους που μπορούμε να ρυθμίσουμε.



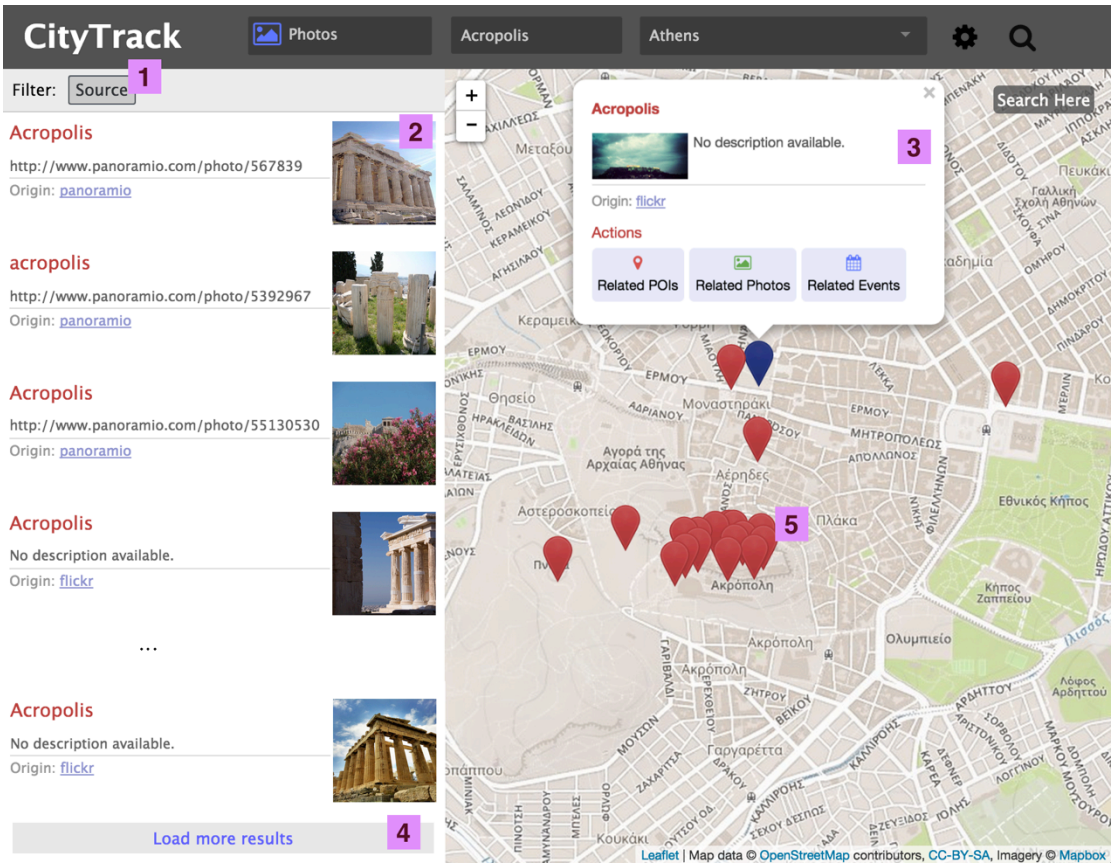
Εικόνα 8.9 Παράθυρο παραμετροποιήσεων αναζήτησης

Συγκεκριμένα οι παράμετροι αυτές μπορεί να είναι οι ακόλουθες:

- Ακτίνα Έκτασης Αναζήτησης: Εμφανίζεται μόνο στην περίπτωση που επιλεγμένη έκταση είναι κυκλική (έχει επιλεχθεί με δεξί κλικ στο χάρτη ή με βάση την τρέχουσα τοποθεσία ή τέλος με βάση την απόσταση από κάποια οντότητα).
- Ελάχιστος Αριθμός Σημείων Ενδιαφέροντος: Εμφανίζεται μόνο για τους τύπους «Οδοί Ενδιαφέροντος», «Γραφικές Οδοί» και «Περιοχές Ενδιαφέροντος». Μπορεί να αναφέρεται τόσο σε Σημεία Ενδιαφέροντος όσο και σε Φωτογραφίες. Ορίζει τον ελάχιστο αριθμό οντοτήτων που μπορεί να φέρει μία περιοχή ή οδός για να επιστραφεί στο αποτέλεσμα.
- Μέγιστη Απόσταση: Εμφανίζεται μόνο για τον τύπο «Περιοχές Ενδιαφέροντος». Ορίζει την μέγιστη απόσταση ανάμεσα σε δύο σημεία ώστε αυτά να θεωρηθούν γειτονικά. Για περισσότερες λεπτομέρειες σχετικά με το πως χρησιμοποιείται η απόσταση αυτή μπορεί να ανατρέξει κανείς στην ενότητα 4.2 όπου περιγράφεται ο αλγόριθμος της ομαδοποίησης.

8.2.5 Περιγραφή της δομής της σελίδας έπειτα από αναζήτηση

Μέχρι στιγμής, είδαμε πώς μπορεί ο χρήστης να πραγματοποιήσει μια αναζήτηση. Στην ενότητα αυτή, θα περιγράψουμε τη δομή του γραφικού περιβάλλοντος της εφαρμογής, αφού ο χρήστης πραγματοποιήσει κάποια αναζήτηση και επιστραφούν τα αποτελέσματα. Όπως



Εικόνα 8.10 Αποτέλεσμα εκτέλεσης μια αναζήτησης για τον Τύπο "Εικόνες"

φαίνεται στην εικόνα 8.10, το γραφικό περιβάλλον της εφαρμογής αλλάζει για να φιλοξενήσει τα αποτελέσματα.

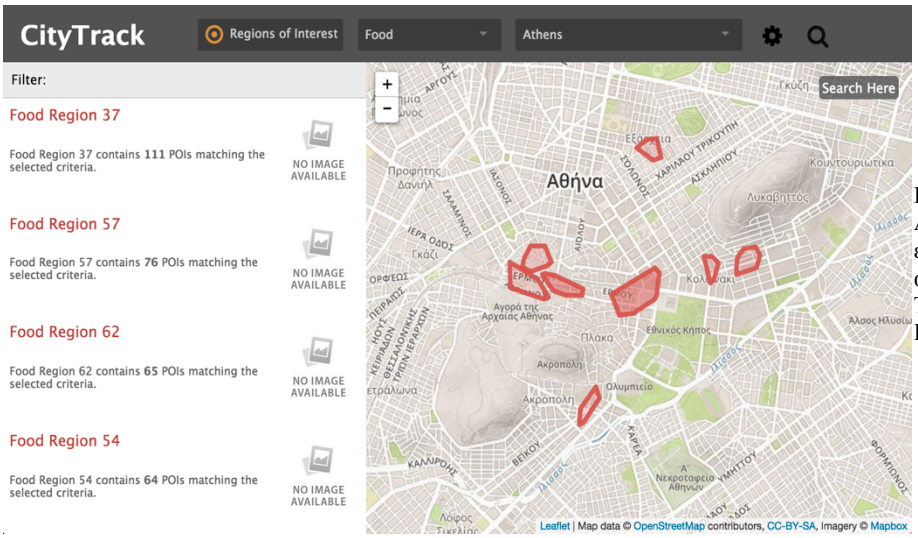
Αρχικά, βλέπουμε ότι μετά από την αναζήτηση το κύριο τμήμα της εφαρμογής χωρίζεται σε δύο στήλες. Στη δεξιά πλευρά παραμένει ο χάρτης ενώ στην αριστερή δημιουργείται μία λίστα με τα αποτελέσματα που επέστρεψε η αναζήτηση. Επίσης, ο χάρτης επικεντρώνεται αυτόματα στα σημεία που επέστρεψε η αναζήτηση έτσι ώστε να τα περιέχει όλα.

Στη συνέχεια θα αναλύσουμε διάφορα σημεία με ιδιαίτερο ενδιαφέρον στην παραπάνω εικόνα τα οποία έχουν σημειωθεί με αριθμούς:

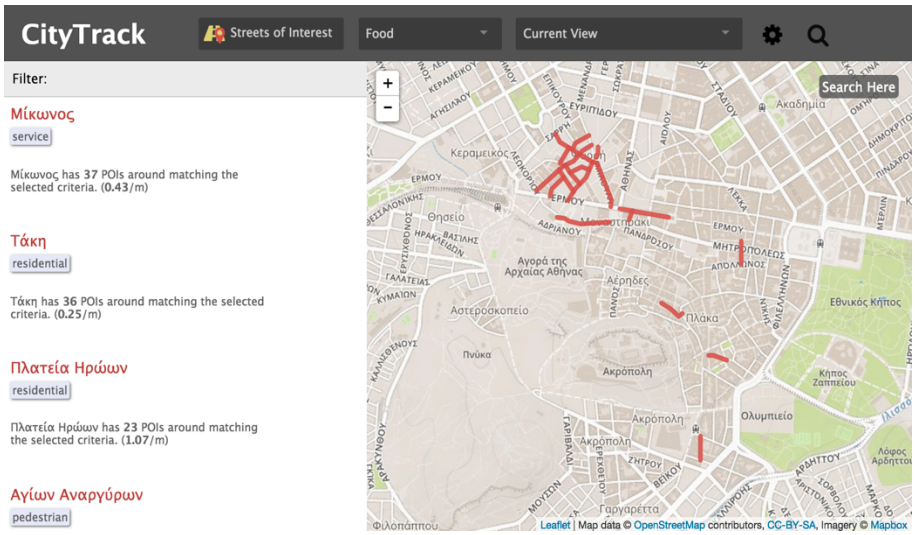
1. Η «επικεφαλίδα» της λίστας των αποτελεσμάτων. Αυτή περιέχει ένα σύνολο από φίλτρα των οποίων τον τρόπο χρήσης θα παρουσιάσουμε σε επόμενη υπό-ενότητα. Όταν δεν υπάρχουν διαθέσιμα φίλτρα η επικεφαλίδα φίλτρων είναι άδεια.
2. Λίστα με τα αποτελέσματα που επέστρεψε η αναζήτηση. Κάθε οντότητα στη λίστα αντιστοιχεί σε ένα σημείο στο χάρτη. Ο χρήστης μπορεί να κάνει κλικ πάνω σε μία οντότητα της λίστας και η οντότητα αυτή θα επιλεχθεί και στο χάρτη, ενώ θα ανοίξει ένα παράθυρο στο χάρτη με πληροφορίες για την οντότητα αυτή.
3. Παράθυρο με περισσότερες πληροφορίες και ενέργειες για την οντότητα που είναι επιλεγμένη. Το παράθυρο αυτό θα περιγραφεί με περισσότερη λεπτομέρεια για όλους τους τύπους οντοτήτων σε επόμενη υπό-ενότητα.
4. Κουμπί φόρτωσης περισσότερων αποτελεσμάτων. Το κουμπί αυτό είναι διαθέσιμο μόνο για τα αποτελέσματα που είναι σελιδοποιημένα, και μόνο αν υπάρχουν περαιτέρω δεδομένα να επιστραφούν. Αν τα δεδομένα δεν είναι σελιδοποιημένα ή δεν υπάρχουν περαιτέρω δεδομένα τότε το κουμπί αυτό είναι απενεργοποιημένο.
5. Σύνολο από σημεία που αντιστοιχούν στις οντότητες που επιστράφηκαν. Ο χρήστης μπορεί να επιλέξει κάποιο σημείο κάνοντας κλικ απάνω του ώστε να ανοίξει το παράθυρο πληροφοριών για το σημείο αυτό.

Στην προηγούμενη εικόνα (Εικόνα 8.10) είδαμε το αποτέλεσμα αναζήτησης για τον τύπο «Φωτογραφίες». Για τους τύπους «Σημεία Ενδιαφέροντος» και «Εκδηλώσεις» τα αποτελέσματα παρουσιάζονται με ακριβώς όμοιο τρόπο. Ωστόσο, για τους τύπους «Οδοί Ενδιαφέροντος», «Γραφικές Οδοί» και «Περιοχές Ενδιαφέροντος» τα αποτελέσματα διαφέρουν ως προς τον τρόπο με τον οποίο παρουσιάζονται στο χάρτη. Οι εικόνες που ακολουθούν στην επόμενη σελίδα παρουσιάζουν τις διαφορές ανάμεσα στους προηγούμενους τύπους. Οι διαφορές αυτές εντοπίζονται κυρίως στον τρόπο αναπαράστασης της κάθε οντότητας στο χάρτη (οι Περιοχές με πολύγωνα και οι Οδοί με γραμμές) αλλά και στης πληροφορίες που παρέχονται για κάθε οντότητα στη λίστα αποτελεσμάτων. Για

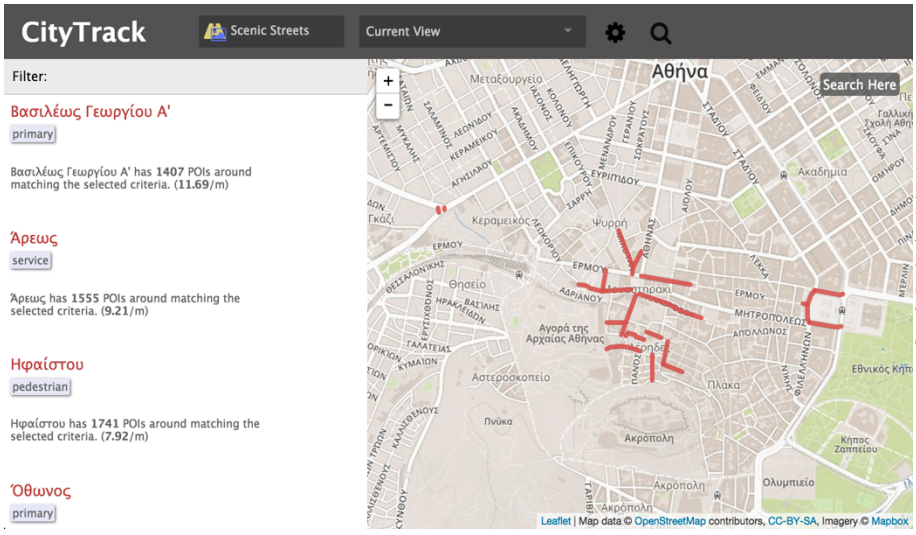
παράδειγμα, αναφέρεται η πυκνότητα των οδών σε Οντότητες, ο αριθμός των Οντοτήτων σε μια περιοχή, ο τύπος μιας οδού κλπ. Τέλος οι οδοί τιτλοφορούνται με το όνομά τους, ενώ οι περιοχές με βάση την κατηγορία αναζήτησης και έναν τυχαίο αύξοντα αριθμό.



Εικόνα 8.11 Αποτέλεσμα εκτέλεσης μια αναζήτησης για τον Τύπο "Περιοχές Ενδιαφέροντος"



Εικόνα 8.12 Αποτέλεσμα εκτέλεσης μια αναζήτησης για τον Τύπο "Οδοί Ενδιαφέροντος"



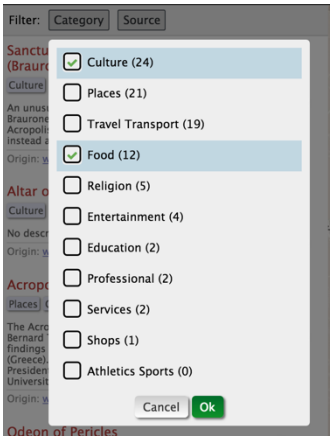
Εικόνα 8.13 Αποτέλεσμα εκτέλεσης μια αναζήτησης για τον Τύπο "Γραφικές Οδοί"

8.2.6 Εφαρμογή φίλτρων στα αποτελέσματα

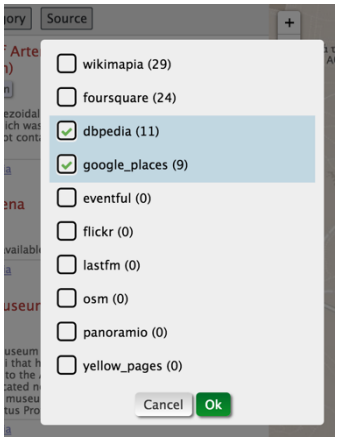
Κάποιοι από τους τύπους αναζητήσεων επιτρέπουν περαιτέρω φιλτράρισμα των αποτελεσμάτων με βάση την πηγή από την οποία τα δεδομένα προέρχονται ή την κατηγορία τους. Συγκεκριμένα, ο τύπος «Σημεία Ενδιαφέροντος» επιτρέπει το φιλτράρισμα με βάση τόσο την Κατηγορία των αποτελεσμάτων, όσο και με βάση την πηγή τους. Από την άλλη, οι τύποι «Φωτογραφίες» και «Εκδηλώσεις» που δεν είναι κατηγοριοποιημένες επιτρέπουν το φιλτράρισμα μόνο με βάση την πηγή προέλευσή τους. Οι υπόλοιπες οντότητες δεν επιτρέπουν κάποιο είδος περαιτέρω φιλτραρίσματος.

Όπως βλέπουμε στις παρακάτω εικόνες, ο χρήστης μπορεί να φιλτράρει το σύνολο των αποτελεσμάτων χρησιμοποιώντας τα κουμπιά που βρίσκονται στην κορυφή της λίστας των αποτελεσμάτων. Τα κουμπιά αυτά εμφανίζονται μόνο αν υπάρχει διαθέσιμο φιλτράρισμα για κάποιο πεδίο. Όταν ο χρήστης πατήσει κάποιο από αυτά τα κουμπιά, εμφανίζεται ένα παράθυρο επιλογής φίλτρων. Ο χρήστης μπορεί να επιλέξει περισσότερα του ενός φίλτρα. Πατώντας OK η εφαρμογή θα πραγματοποιήσει μια νέα αναζήτηση λαμβάνοντας υπ' όψιν τα επιλεγμένα φίλτρα.

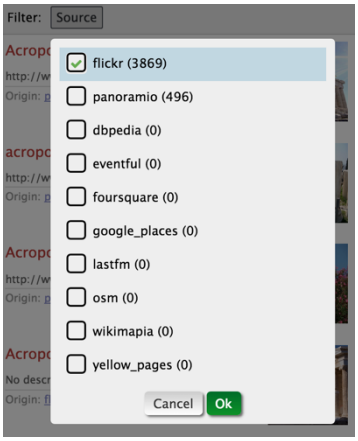
Τα φίλτρα μπορούν να απενεργοποιηθούν ανά πάσα στιγμή ακολουθώντας την ίδια διαδικασία που περιγράφηκε παραπάνω.



Εικόνα 8.16 Φιλτράρισμα Σημείων Ενδιαφέροντος με βάση την κατηγορία τους



Εικόνα 8.16 Φιλτράρισμα Σημείων Ενδιαφέροντος με βάση την Πηγή τους



Εικόνα 8.16 Φιλτράρισμα Φωτογραφιών με βάση την Πηγή προέλευσής τους

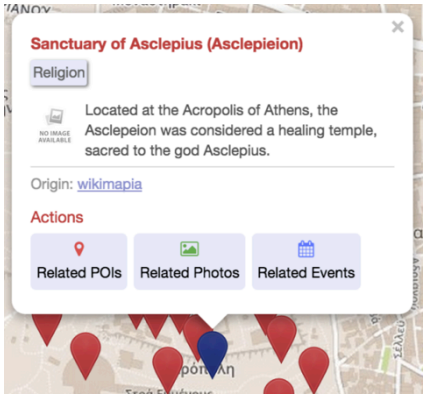
8.2.7 Παράθυρο πληροφοριών και ενέργειες

Μία από τις πιο σημαντικές λειτουργικότητες της εφαρμογής έπειτα από την πραγματοποίηση μιας αναζήτησης, είναι η αλληλεπίδραση με τις οντότητες που επιστρέφονται. Η αλληλεπίδραση αυτή είναι εφικτή μέσω του παραθύρου λεπτομερειών, που παρουσιάζεται στο χάρτη όταν επιλέξει μια οντότητα από τη λίστα ή απευθείας από το χάρτη.

Η δομή του παραθύρου είναι διαφορετική για κάθε τύπο οντότητας που παρουσιάζεται στο χάρτη. Συγκεκριμένα, υπάρχουν δύο τύποι παραθύρων: (α) αυτά που αφορούν σημειακές οντότητες, όπως για παράδειγμα Σημεία Ενδιαφέροντος και (β) αυτά που αφορούν Περιοχές ή Οδούς. Στη συνέχεια θα παρουσιάσουμε την κάθε μια από τις δυο ομάδες ξεχωριστά.

8.2.7.1 Παράθυρο Πληροφοριών Σημειακών Οντοτήτων

Το παράθυρο πληροφοριών για τις σημειακές οντότητες φαίνεται στην ακόλουθη εικόνα:



Εικόνα 8.17 Παράθυρο πληροφοριών Σημειακής οντότητας

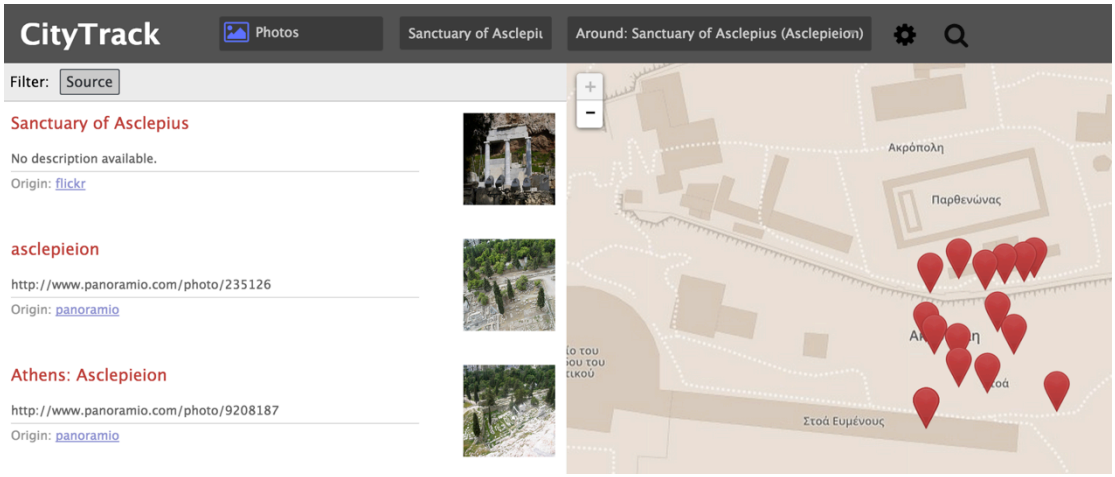
Όπως βλέπουμε το παράθυρο αυτό παρουσιάζει βασικές πληροφορίες για τις σημειακές οντότητες. Αναφορικά, παρουσιάζει:

- Τον τίτλο της οντότητας
- Την κατηγορία της οντότητας (αν υπάρχει)
- Την περιγραφή της οντότητας
- Μια φωτογραφία για την οντότητας (αν υπάρχει)
- Την πηγή προέλευσης της οντότητας
- Λίστα με περισσότερες φωτογραφίες (αν υπάρχουν)

Η σημαντικότερη λειτουργία του παραθύρου ωστόσο όπως περιγράψαμε είναι οι ενέργειές που μπορεί ο χρήστης να πραγματοποιήσει με βάση αυτή την οντότητα. Αναφορικά ο χρήστης μπορεί:

- Να αναζητήσει Σημεία Ενδιαφέροντος σχετικά με την ενεργή οντότητα στη γειτονιά της
- Να αναζητήσει Φωτογραφίες σχετικές με την ενεργή οντότητα στη γειτονιά της
- Να αναζητήσει Εκδηλώσεις σχετικές με την ενεργή οντότητα στη γειτονιά της

Για παράδειγμα, αν για την προηγούμενη οντότητα της εικόνας Εικόνα 8.17 κάνουμε κλικ στην ενέργεια «Σχετικές Φωτογραφίες» θα επιστρέψει μια λίστα με κοντινές φωτογραφίες που σχετικές με τον τίτλο της προηγούμενες οντότητες που βρίσκονται στη γειτονιά της. Το αποτέλεσμα της ενέργειας φαίνεται στην Εικόνα 8.18.

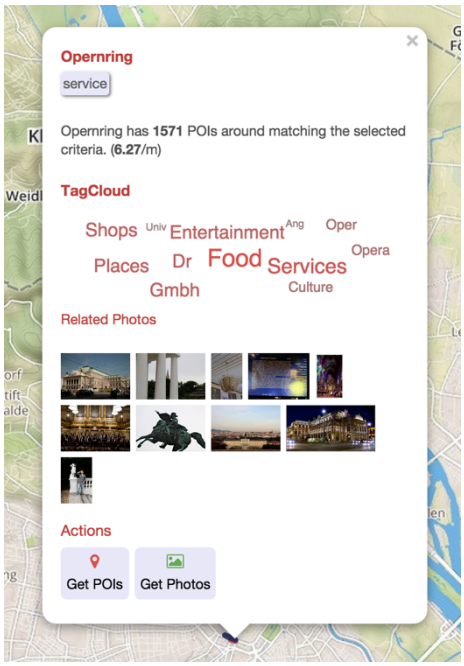


Εικόνα 8.18 Αποτέλεσμα αναζήτησης σχετικών φωτογραφιών για την οντότητα της προηγούμενης εικόνας

Με παρόμοιο τρόπο λειτουργούν και οι υπόλοιπες ενέργειες των σημειακών οντοτήτων. Για περισσότερες πληροφορίες σχετικά με τον τρόπο λειτουργίας του ανωτέρω μπορεί κανείς να ανατρέξει στην ενότητα 4.5

8.2.7.2 Παράθυρο πληροφοριών Οδών και Περιοχών

Όπως οι σημειακές οντότητες, έτσι και οι Οδοί και οι Περιοχές, όταν επιλεγούν από το χρήστη, εμφανίζουν ένα παράθυρο πληροφοριών στο χάρτη. Τα παράθυρα πληροφοριών των οντοτήτων αυτών φαίνονται στις εικόνες (Εικόνα 8.20 και Εικόνα 8.20) που ακολουθούν:



Εικόνα 8.20 Παράθυρο πληροφοριών για Οδούς

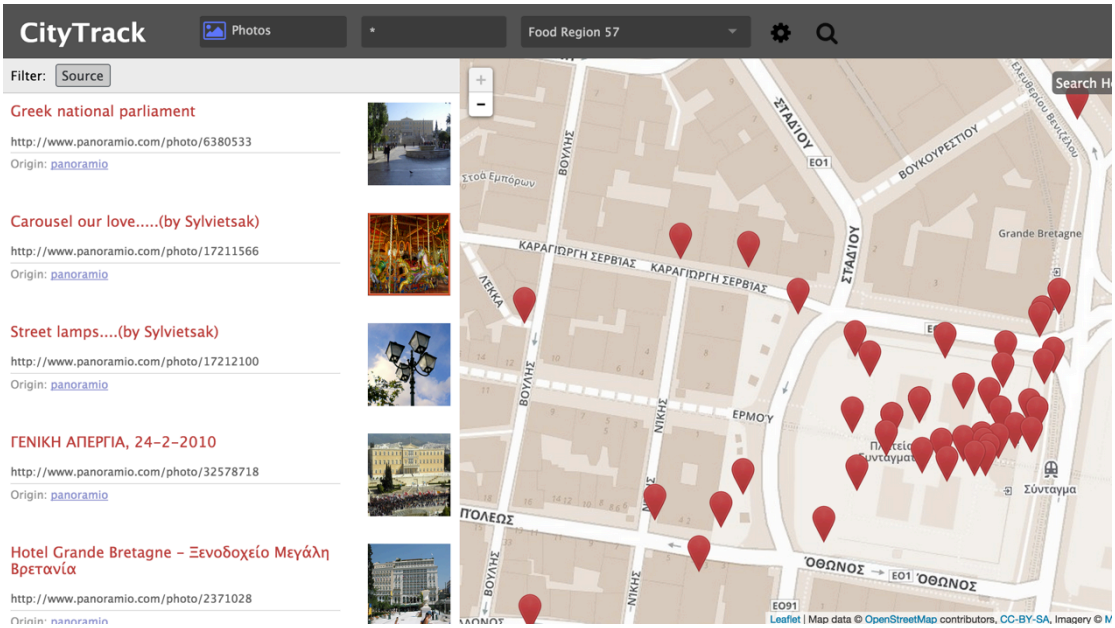
Εικόνα 8.20 Παράθυρο πληροφοριών για Περιοχές

Όπως βλέπουμε, το παράθυρο αυτό παρουσιάζει περισσότερες πληροφορίες για τις οντότητες αυτές σε σχέση με εκείνο των σημειακών οντοτήτων. Συγκεκριμένα για κάθε οντότητα παραθέτει:

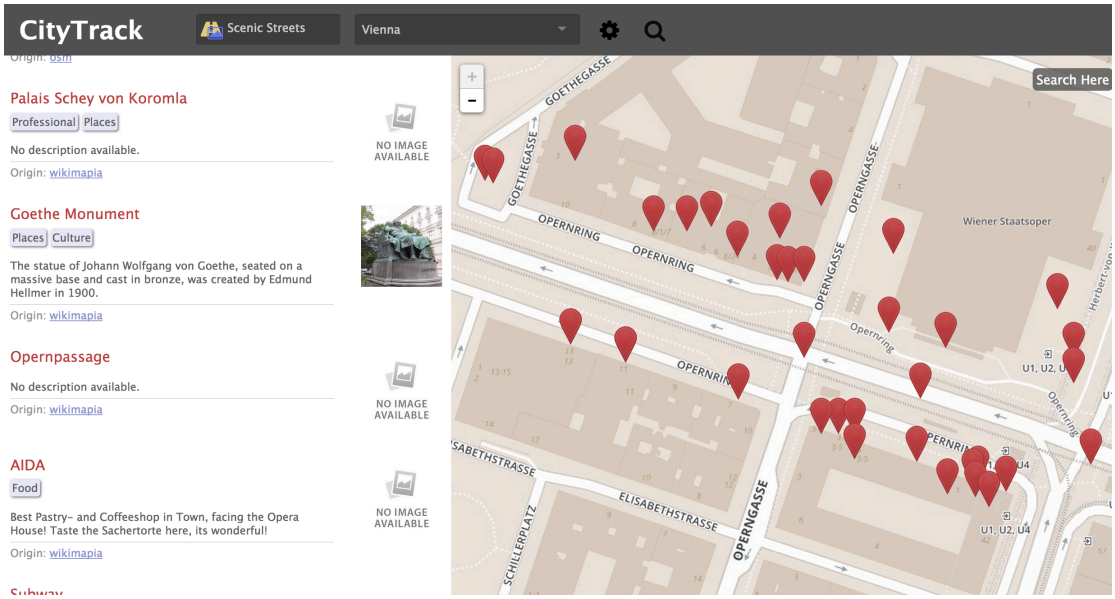
- Τον τίτλο της
- Τον αριθμό των οντοτήτων που περιέχονται στην οντότητα
- Ένα νέφος λέξεων που περιγράφουν τα Σημεία Ενδιαφέροντος που ανήκουν στην οντότητα αυτή.
- Ένα σύνολο από σχετικές και ποικίλες φωτογραφίες που περιγράφουν την οντότητα (περισσότερες πληροφορίες στην ενότητα 4.4)
- Το είδος της οντότητας (Μόνο για τις οδούς το είδος του δρόμου)

Όπως με τις σημειακές οντότητες έτσι και με τις οδούς και τις περιοχές, παρέχονται μέσω του παραθύρου πληροφοριών ένα σύνολο ενεργειών για να αποκτήσουμε σημειακές οντότητες που αντιστοιχούν στις οδούς και τις περιοχές. Συγκεκριμένα μπορούμε να αποκτήσουμε Σημεία Ενδιαφέροντος ή Φωτογραφίες που «ανήκουν» στις τελευταίες.

Στην Εικόνα 8.22 βλέπουμε το αποτέλεσμα της ενέργειας εύρεσης Φωτογραφιών για την Περιοχή Ενδιαφέροντος της Εικόνα 8.20 Παράθυρο πληροφοριών για Περιοχές. Αντίστοιχα, στην Εικόνα 8.21 βλέπουμε το αποτέλεσμα της ενέργειας εύρεσης Σημείων Ενδιαφέροντος για τη Γραφική Οδό της Εικόνα 8.20.



Εικόνα 8.22 Αποτέλεσμα ενέργειας εύρεσης φωτογραφιών για την Περιοχή Ενδιαφέροντος της Εικόνα 8.20 Παράθυρο πληροφοριών για Περιοχές



Εικόνα 8.21 Αποτέλεσμα της ενέργειας εύρεσης Σημείων Ενδιαφέροντος για την Οδό της Εικόνα 8.20

Βλέπουμε λοιπόν ότι μέσα από το παράθυρο πληροφοριών και τις ενέργειες που αυτό παρέχει μπορούμε, όχι μόνο να αποκτήσουμε επιπλέον πληροφορία για την οντότητα στην οποία το παράθυρο αντιστοιχεί, αλλά και να περιηγηθούμε ανάμεσα σε οντότητες που είναι σχετικές μεταξύ τους.

9

Επίλογος

Το πλήθος της γεωχωρική πληροφορίας που είναι διαθέσιμη σήμερα στο διαδίκτυο καθιστά περισσότερο αναγκαία από ποτέ την ύπαρξη εργαλείων που είναι σε θέση να την αναλύουν και να παρουσιάζουν τα αποτελέσματα της ανάλυσης αυτής. Για το λόγο αυτό, στην εργασία μας, σχεδιάσαμε ένα σύστημα που μετά από τη συλλογή πληθώρας γεωχωρικών δεδομένων τα αναλύει και παρουσιάζει τα αποτελέσματα σε ένα λειτουργικό γραφικό περιβάλλον.

Αναλυτικότερα, είδαμε πώς με τη χρήση διαφόρων αλγορίθμων ομαδοποίησης και αντιστοίχισης μπορούμε να εξάγουμε χρήσιμα συμπεράσματα για τα Σημεία Ενδιαφέροντος και τις Φωτογραφίες που περιέχονται σε μια πόλη. Πιο συγκεκριμένα, με τη χρήση του συστήματός μας μπορούμε να εντοπίσουμε «Περιοχές» και «Οδούς» που παρουσιάζουν ιδιαίτερο ενδιαφέρον σε μία περιοχή, με βάση τα Σημεία Ενδιαφέροντος ή τις Φωτογραφίες που περιέχουν. Επίσης, μελετήσαμε πώς μπορούμε να αντιστοιχίσουμε σχετικές οντότητες μεταξύ τους σε πραγματικό χρόνο, χρησιμοποιώντας τη γεωχωρική και σημασιολογική πληροφορία που αυτές περιέχουν. Δίδουμε έτσι στο χρήστη τη δυνατότητα να περιηγηθεί στα δεδομένα, με βάση τη σχέση που έχουν αυτά μεταξύ τους, και όχι μόνο χρησιμοποιώντας αποτελέσματα αναζητήσεων. Και όλα αυτά, μέσα από ένα λειτουργικό και εύκολο στη χρήση γραφικό περιβάλλον, που δεν προϋποθέτει κάποια ιδιαίτερη γνώση από την πλευρά του χρήστη.

Ένα άλλο σημείο στο οποίο δώσαμε ιδιαίτερη βάση ήταν η ποιότητα της τελικής εφαρμογής χωρίς αυτό να επηρεάζει την ταχύτητα του κύκλου ανάπτυξης της τελευταίας. Για να το πετύχουμε αυτό, χρησιμοποιήσαμε τη σύγχρονη προσέγγιση της ανάπτυξης κώδικας με βάση τα τεστ. Επίσης, χρησιμοποιήσαμε μια πληθώρα βιβλιοθηκών και εργαλείων που μας

βοήθησαν, τόσο στο γρήγορο έλεγχο του παραγόμενου κώδικα, όσο και στην αυτοματοποίηση της όλης διαδικασίας ελέγχου και την εγκατάστασης της εφαρμογής.

Ένα σημείο το οποίο μπορεί να βελτιωθεί σημαντικά είναι ο χρόνος ανάκτησης των δεδομένων από τις διάφορες πηγές δεδομένων. Τόσο το σύστημα ευρετηριοποίησης όσο και η Βάση Δεδομένων είναι εγκατεστημένα σε πολύ αργά μηχανήματα με αποτέλεσμα να χρειάζονται πολύ χρόνο για να επιστρέψουν ακόμη και σχετικά μικρή ποσότητα πληροφορίας. Επίσης λόγο της φύσης της πληροφορίας, μεγάλο κέρδος σε ταχύτητα μπορεί να επιτευχθεί αν αυτή διαχωριστεί σε διαφορετικά μηχανήματα με την τεχνική του Partitioning. Τέλος η χρήση κάποιου μηχανισμού Caching θα μπορούσε να βελτιώσει σημαντικά την ταχύτητα ανάκτησης πληροφορίας για όμοιες αναζητήσεις.

Επίσης θα μπορούσαν να υλοποιηθούν μία σειρά από λειτουργικές βελτιώσεις στο γραφικό περιβάλλον για τη βελτίωση της εμπειρίας του χρήστη. Η πιο σημαντική, θα ήταν η υλοποίηση ενός κουμπιού πλοήγησης σε προηγούμενες αναζητήσεις που πραγματοποίησε ο χρήστης στην τρέχουσα επίσκεψή του, με τη μορφή ίσως ενός κουμπιού «Πίσω». Επιπλέον, θα μπορούσε να υλοποιηθεί ένα σύστημα προτάσεων λέξης κλειδιού, όσο πληκτρολογεί ο χρήστης, με βάση τις οντότητες που υπάρχουν στη βάση ή με βάση συχνές αναζητήσεις που γίνονται από το σύνολο των χρηστών.

Μία πολύ χρήσιμη επέκταση της εφαρμογής που σχεδιάσαμε θα ήταν να χρησιμοποιεί την πληροφορία που διαθέτει για να καταστρώνει μονοπάτια με βάση το ενδιαφέρον του χρήστη. Να δίνει, δηλαδή, τη δυνατότητα στο χρήστη να επιλέξει ένα σημείο προορισμού, ένα σημείο προέλευσης και ένα κριτήριο και να επιστρέφει ένα μονοπάτι που μεγιστοποιεί το δοθέν κριτήριο. Για παράδειγμα, ο χρήστης θα μπορούσε να ζητήσει ένα μονοπάτι που περνά από δρόμους στους οποίους βγαίνουν πολλές φωτογραφίες, ή από δρόμους που περνούν από πολλά εστιατόρια κοκ.

10

Βιβλιογραφία

Bower. *Bower.* 2015. http://bower.io/.

Browserify. *Browserify.* 2015. http://browserify.org/.

Dedu, Eugen. «Bresenham-based Supercover Line Algorithm.» 2001. http://lifc.univ-fcomte.fr/~dedu/projects/bresenham/index.html.

Development, Test Driven. «Test Driven Development.» *Wikipedia.* http://en.wikipedia.org/wiki/Test-driven_development.

Express.js. *Express.js.* 2015. http://expressjs.com/.

Forever.js. *Forever.js.* 2015. https://github.com/foreverjs/forever.

Franklin, W. Randolph. «PNPOLY - Point Inclusion in Polygon Test.» 2014. http://www.ecse.rpi.edu/~wrf/Research/Short_Notes/pnpoly.html (πρόσβαση 2015).

Git. *Git-SCM.* 2015. http://git-scm.com/.

Gulp.js. *Gulp.js.* 2015. http://gulpjs.com/.

JTS. *Vivid Solutions.* 2015. http://www.vividsolutions.com/jts/JTSHome.htm (πρόσβαση 2015).

Lamprianidis, George, Dimitrios Skoutas, George Papatheodorou, και Dieter Pfoser. «Extraction, Integration and Analysis of Crowdsourced Points of Interest from Multiple Web Sources.» Paper, 2014.

M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. *A density-based algorithm for discovering clusters in large spatial databases with noise.* 1996.

Maven. *Apache Software Foundation.* 2015. https://maven.apache.org/.

Node.js. *Joyent, Inc.* 2015. https://nodejs.org/.

NPM. *NPM, Inc.* 2015. https://www.npmjs.com/.

OpenStreetMap. *OpenStreetMap Contributors.* 2015. https://www.openstreetmap.org/about.

Oracle. «Java™ Servlet Specification Version 3.» *Java™ Servlet Specification Version 3.* 2009. http://download.oracle.com/otn-pub/jcp/servlet-3.0-fr-eval-oth-JSpec/servlet-3_0-final-spec.pdf (πρόσβαση 2015).

PostGIS. *OSGeo.* 2015. http://postgis.net/ (πρόσβαση 2015).

PostgreSQL. *Postgres Global Development Group.* 2015. http://www.postgresql.org/ (πρόσβαση 2015).

Preparata, Franco P., Michael Ian Shamos, and Franco P. Preparata. *Computational geometry: an introduction.* Vol. 5. New York: Springer-Verlag, 1985.

Solr. *Apache Software Foundation.* 2015. http://lucene.apache.org/solr/.

SolrJ. *https://www.openstreetmap.org/about.* 2015. https://wiki.apache.org/solr/Solrj.

Sreenivas Gollapudi, Aneesh Sharma. *An axiomatic approach for result diversification.* WWW, 2009, 381-390.

Swagger. *Swagger.* 2015. http://swagger.io/.

Van Dongen, Stijn, and Anton J. Enright. *Metric distances derived from cosine similarity and pearson and spearman correlations.* arXiv preprint, 2012, 1208.3145.

Yianilos, Peter N. *Data Structures and Algorithms for Nearest Neighbor Search in General Metric Spaces.* Paper, SODA, Vol. 93. No. 194, 1993.