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

Συστήματα απεικόνισης γεγονότων σε real time περιβάλλοντα από ετερογενείς πηγές δεδομένων και κοινωνικών δικτύων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Λαμπρινός Ε. Χασάπης

Επιβλέπων : Θεοδώρα Βαρβαρίγου
Καθηγήτρια Ε.Μ.Π

Αθήνα, Μάιος 2017

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

Συστήματα απεικόνισης γεγονότων σε real time περιβάλλοντα από ετερογενείς πηγές δεδομένων και κοινωνικών δικτύων

ΔΙΠΛΩΜΑΤΙΚΗ ΕΡΓΑΣΙΑ

Λαμπρινός Ε. Χασάπης

Επιβλέπων : Θεοδώρα Βαρβαρίγου
Καθηγήτρια Ε.Μ.Π

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή

.....	.....	.....
Θεοδώρα Βαρβαρίγου	Συμεών Παπαβασιλείου	Δημήτριος Ασκούνης
Καθηγήτρια Ε.Μ.Π	Καθηγητής Ε.Μ.Π	Καθηγητής Ε.Μ.Π

Αθήνα, Μάιος 2017

.....
Λαμπρινός Ε. Χασάπης

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

Περίληψη

Η απεικόνιση γεγονότων και δεδομένων σε χάρτες αποτελεί έναν αποδοτικό τρόπο για την οργάνωση της πληροφορίας, την ευκολότερη ανάγνωσή της και την εξαγωγή συμπερασμάτων από αυτή. Ενώ σε έντυπη μορφή ο χάρτης είναι μια στατική συλλογή δεδομένων, μέσω του διαδικτύου τα γεωγραφικά δεδομένα μπορούν συνεχώς να εμπλουτίζονται σε πραγματικό χρόνο με πολυμεσικό περιεχόμενο και πρόσθετες πληροφορίες από πολλές διαφορετικές πηγές. Επιπλέον, με την αύξηση της δημοτικότητας των κοινωνικών δικτύων, παρέχεται πλέον στους χρήστες η δυνατότητα να προσθέτουν δικό τους περιεχόμενο, απόψεις και γνώσεις πάνω σε διάφορα θέματα. Προκειμένου να αξιοποιηθούν οι πληροφορίες που προκύπτουν από τις αλληλεπιδράσεις των χρηστών στα κοινωνικά δίκτυα, καθίσταται σημαντική η δημιουργία συστημάτων για την εξαγωγή, οργάνωση, αποθήκευση και αναπαράσταση των δεδομένων από αυτά.

Στην παρούσα διπλωματική εργασία αναπτύχθηκε μια εφαρμογή απεικόνισης τοποθεσιών, εκδηλώσεων και επιχειρήσεων (θα αναφέρονται γενικά ως venues) σε χάρτες πόλεων των ΗΠΑ. Η εφαρμογή συλλέγει γεωγραφικά δεδομένα για το χωρισμό μιας πόλης που θα αναζητήσει ο χρήστης σε επιμέρους περιοχές (θα αναφέρονται κι ως γειτονιές). Για κάθε μία από αυτές αναζητά στο Foursquare τα πιο δημοφιλή venues, τα αποθηκεύει σε συλλογές ανάλογα με την κατηγορία όπου ανήκουν (φαγητό, διασκέδαση, κλπ) και εμφανίζει στο χάρτη κάθε γειτονιά χρωματισμένη ανάλογα με τον αριθμό των venues της κατηγορίας που επιλέγει ο χρήστης. Παράλληλα, δίνεται η δυνατότητα στο χρήστη να δει περισσότερες πληροφορίες για κάθε venue σε μία γειτονιά, όπως όνομα, διεύθυνση, βαθμολογία, αριθμός επισκεπτών και τα σχόλιά τους στο Foursquare.

Εν κατακλείδι, η εργασία αυτή παρουσιάζει ένα σύστημα που εμπλουτίζει στατικά γεωγραφικά δεδομένα με πληροφορίες που εισάγονται από χρήστες σε κοινωνικά δίκτυα και επιτρέπει την παρακολούθησή τους σε πραγματικό χρόνο. Επιπλέον, έχει σχεδιαστεί με τρόπο που επιτρέπει την επέκτασή του μέσω σύνδεσης σε πρόσθετες πηγές δεδομένων, με σκοπό τη μελέτη τους για την εξαγωγή στατιστικών αποτελεσμάτων.

Λέξεις-Κλειδιά:

συστήματα απεικόνισης, γεωγραφικά δεδομένα, κοινωνικά δίκτυα, βάσεις δεδομένων, GeoJSON, Leaflet, Mapbox, NoSQL, MongoDB, REST, Node.JS, Daemon, Foursquare API, Javascript, jQuery, Java

Abstract

The depiction of events and data on maps is an efficient way to organise information, make it more readable and draw conclusions from it. While on paper a map is a static collection of data, on the Internet geographic data can be enriched in real-time with multimedia content and additional information from several different sources. Moreover, with the growing popularity of social networks, most users can add their own content, opinions and knowledge on various topics. In order to take advantage of the information resulting from user interactions on social networks, it is important to create systems for the extraction, management, storage and representation of data from them.

The purpose of this diploma thesis is to describe the development of an application that displays on an interactive map the location of events and companies (referred to as venues) in a US city. The application collects geographic data that divide a city of the user's choice into smaller regions (referred to as neighbourhoods). For each neighbourhood, it searches for the most popular venues in Foursquare, stores them into collections based on their category (food, entertainment, etc.) and displays on the map every neighbourhood coloured depending on the number of venues of a category chosen by the user. Furthermore, the user can view more information about each venue in a neighbourhood, such as name, address, rating, the number of visitors and their comments on Foursquare.

In conclusion, this paper presents a system that enriches static geographical data with information entered by users in social networks and allows their monitoring in real-time. Additionally, it is designed in a way that allows its expansion via connectivity to additional data sources for the purpose of studying them and extracting statistical results.

Keywords:

mapping systems, geographical data, social networks, databases, GeoJSON, Leaflet, Mapbox, NoSQL, MongoDB, REST, Node.JS, Daemon, Foursquare API, Javascript, jQuery, Java

Ευχαριστίες

Η διπλωματική αυτή εκπονήθηκε στο Εργαστήριο Distributed Knowledge and Media Systems Group του Τομέα Επικοινωνιών, Ηλεκτρονικής και Συστημάτων Πληροφορικής της σχολής Ηλεκτρολόγων Μηχανικών και Μηχανικών Υπολογιστών του Εθνικού Μετσόβιου Πολυτεχνείου.

Θα ήθελα να ευχαριστήσω θερμά την καθηγήτρια κα. Θεοδώρα Βαρβαρίγου για την υποστήριξη και καθοδήγηση που μου προσέφερε, καθώς και για την ευκαιρία που μου έδωσε να εκπονήσω την διπλωματική μου στο συγκεκριμένο θέμα.

Ιδιαίτερες ευχαριστίες απευθύνονται στον υποψήφιο διδάκτορα Βρεττό Μουλό, ο οποίος με την εμπειρία και τις συμβουλές του με βοήθησε να ορίσω την κατεύθυνση της έρευνάς μου και να την ολοκληρώσω με επιτυχία.

Καταλήγοντας, θα ήθελα να ευχαριστήσω την οικογένειά μου για τη συνεχή και κάθε είδους υποστήριξη σε όλα τα χρόνια των σπουδών μου.

Περιεχόμενα

Table of Contents

Κατάλογος Σχημάτων

Κεφάλαιο 1
Εισαγωγή

1.1 Η εξέλιξη των χαρτών

Οι χάρτες [1] αποτελούσαν πάντα ένα εύχρηστο μέσο για την αναπαράσταση πολλών τύπων πληροφορίας. Γεωφυσικοί χάρτες χρησιμοποιούνται για την απεικόνιση της μορφολογίας του εδάφους, πολιτικοί χάρτες δείχνουν τα διοικητικά όρια μιας χώρας, οδικοί χάρτες περιλαμβάνουν δρόμους και συγκοινωνίες και ταξιδιωτικοί χάρτες αποτελούν απαραίτητο εργαλείο κατά την επίσκεψη ενός προορισμού.

Στο παρελθόν, οι χάρτες ήταν σε έντυπη μορφή και παρουσίαζαν στατική πληροφορία. Με την ευρεία χρήση των ηλεκτρονικών συσκευών σήμερα, κυριαρχούν πλέον οι ηλεκτρονικοί χάρτες. Αυτοί έχουν το πλεονέκτημα ότι μπορούν να ανανεώνονται συχνότερα και ευκολότερα, προσφέρουν τη δυνατότητα αλληλεπίδρασης με το χρήστη για την εμφάνιση των δεδομένων που τον ενδιαφέρουν και αναπαριστούν γεγονότα σε πραγματικό χρόνο, όπως τη μεταβολή της θέσης ενός οχήματος.

Για τους παραπάνω λόγους, λοιπόν, παρατηρείται μια συνεχής αύξηση της χρήσης των χαρτών στο διαδίκτυο σε πολλές εφαρμογές. Για παράδειγμα, οι περισσότερες επιχειρήσεις έχουν πλέον ενσωματώσει στην ιστοσελίδα τους χάρτη με την τοποθεσία τους και οδηγίες για τη μετάβαση σε αυτή. Αντίστροφα, ιστοσελίδες όπως το Google Maps προσθέτουν στους χάρτες τους ολοένα και περισσότερες πληροφορίες, με σκοπό να βελτιώσουν τις υπηρεσίες τους και να διευκολύνουν τους χρήστες να αναζητήσουν το επιθυμητό γι' αυτούς περιεχόμενο.

Ένας βασικός λόγος της ραγδαίας αύξησης του όγκου των δεδομένων που ανταλάσσονται μέσω διαδικτύου είναι η δυνατότητα των χρηστών να δημιουργούν νέο περιεχόμενο. Η νέα γενιά του Παγκόσμιου Ιστού που βασίζεται σε αυτή τη δυνατότητα των χρηστών να προσθέτουν και να μοιράζονται πληροφορίες αναφέρεται συχνά με τον όρο Web 2.0. Σημαντική είναι η συμβολή και των κοινωνικών δικτύων σε αυτή την ανάπτυξη. Οι χρήστες τους ανταλλάσσουν καθημερινά αμέτρητα μηνύματα μεταξύ τους, δημιουργούν ή αναδημοσιεύουν άρθρα για ποικίλα θέματα και ασκούν κριτική σε προϊόντα και υπηρεσίες. Κρίνεται, επομένως, απαραίτητη η συστηματική εξόρυξη, αποθήκευση και αξιοποίηση των πληροφοριών από τα κοινωνικά δίκτυα, κάτι που αποτελεί αντικείμενο μελέτης ενός ολόκληρου κλάδου της επιστήμης των υπολογιστών, του λεγόμενου data mining ή στην προκειμένη περίπτωση, social media mining.

1.2 Κοινωνικά δίκτυα

Ένα κοινωνικό δίκτυο [2] ορίζεται ως ένα σύνολο από δράστες (άνθρωποι, οργανισμοί ή άλλες κοινωνικές ομάδες) και ένα σύνολο από τις μεταξύ τους σχέσεις (φιλίες, δεσμοί, χρηματικές συναλλαγές, κλπ). Στο διαδίκτυο, τα social networks αποτελούν υπηρεσίες που επιτρέπουν στα άτομα να δημιουργήσουν το δικό τους προφίλ, να επικοινωνήσουν με άλλους

χρήστες με τους οποίους μοιράζονται μια μορφή σύνδεσης, να μοιράζονται πολυμεσικό περιεχόμενο, να παρακολουθούν βίντεο εκδηλώσεων σε πραγματικό χρόνο, κλπ.

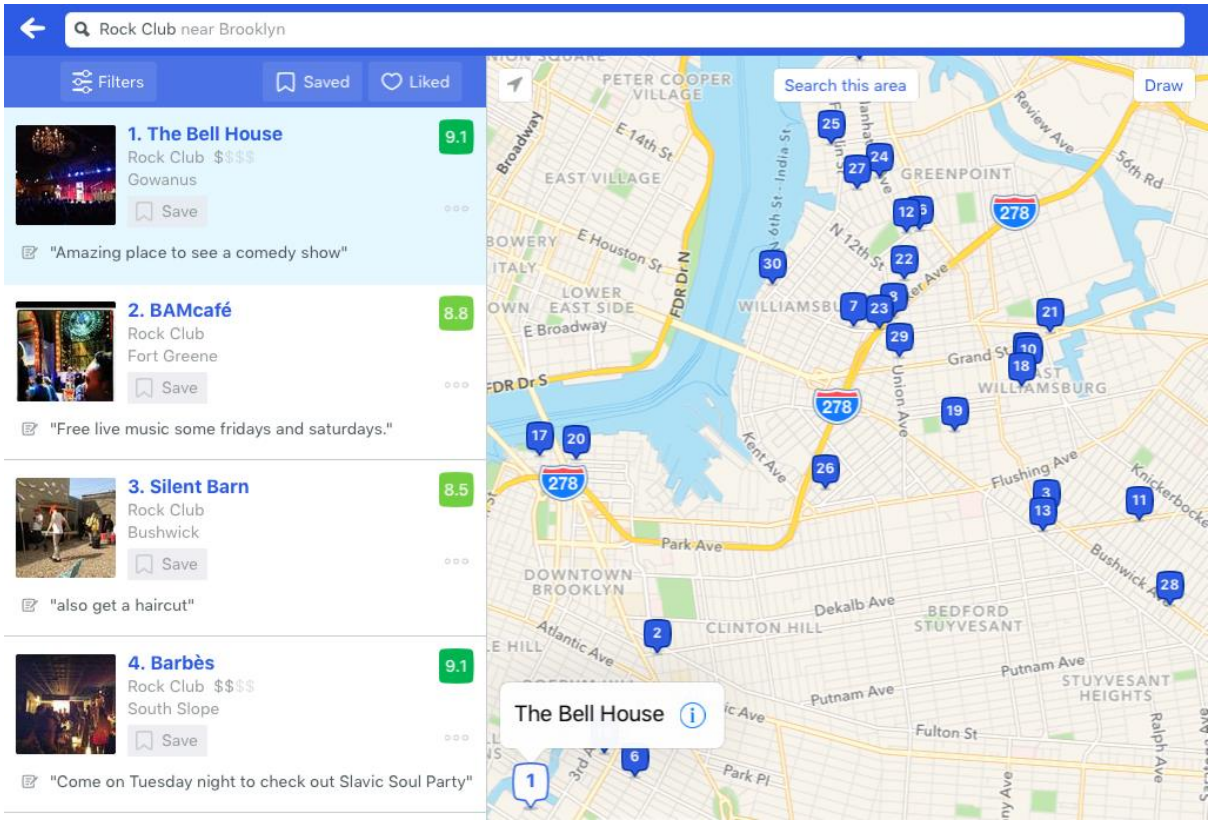
Η ενεργή συμμετοχή ενός τεράστιου αριθμού ατόμων σε κοινωνικά δίκτυα, έχει ως αποτέλεσμα τη συνεχή δημιουργία νέου περιεχομένου και τη μεταφορά μεγάλου όγκου δεδομένων. Πολλοί άνθρωποι πλέον χρησιμοποιούν τα κοινωνικά δίκτυα ως μέσα πληροφόρησης, ανταλλαγής απόψεων και δημοσίευσης φωτογραφιών και εντυπώσεων από τοποθεσίες και εκδηλώσεις που επισκέπτονται. Ως αποτέλεσμα, τα σχόλια και η κριτική που ασκούν οι χρήστες σε άρθρα, προϊόντα και υπηρεσίες στα κοινωνικά δίκτυα αντιπροσωπεύουν όλο και μεγαλύτερο ποσοστό της κοινής γνώμης και λαμβάνονται σοβαρά υπόψιν από εταιρείες για τη βελτίωση των υπηρεσιών τους.

Για παράδειγμα, μια επιχείρηση, προκειμένου να γνωστοποιήσει την ύπαρξή της, να βελτιώσει την εικόνα της και να προσελκύσει πελάτες, δίνει μεγάλη έμφαση στο προφίλ που διατηρεί σε κάποιο κοινωνικό δίκτυο. Στο προφίλ της περιλαμβάνει πάντα τηλέφωνο επικοινωνίας και χάρτη για πρόσβαση στην τοποθεσία της, προσπαθεί να κρατήσει ζωντανό το ενδιαφέρον των χρηστών με καθημερινές δημοσιεύσεις, φωτογραφίες και προσφορές και έχει τη δυνατότητα για άμεση ανταπόκριση σε ερωτήσεις και σχόλια χρηστών.

Μερικά δημοφιλή κοινωνικά δίκτυα είναι: Facebook, Twitter, Google+, Youtube, Tumblr, LinkedIn, Instagram, Pinterest, Foursquare, Flickr.

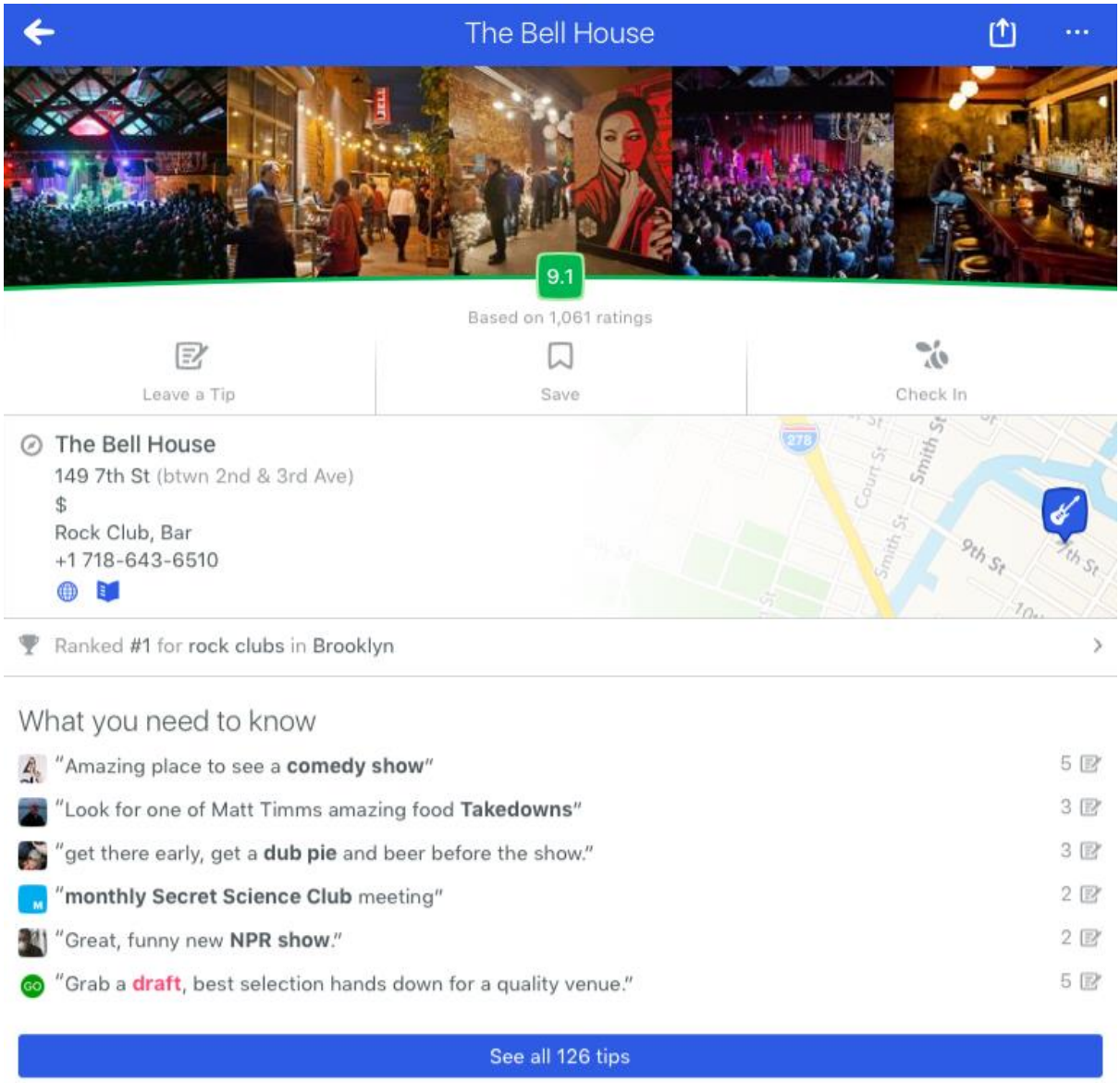
1.2.1 Foursquare

Το Foursquare [3] είναι μία διαδικτυακή πλατφόρμα που επιτρέπει στους χρήστες να αναζητήσουν τοποθεσίες, εκδηλώσεις, επιχειρήσεις και οργανισμούς και να φιλτράρουν τα αποτελέσματα βάσει κριτηρίων, όπως κατηγορία, απόσταση, λέξεις-κλειδιά και βαθμολογία επισκεπτών. Τα αποτελέσματα εμφανίζονται σε μία λίστα ταξινομημένα βάσει παραμέτρων, όπως η συνολική βαθμολογία χρηστών και η απόσταση από την τοποθεσία του χρήστη, ενώ παράλληλα απεικονίζονται οι τοποθεσίες τους σε ένα χάρτη.



Σχήμα 1.1: Αποτελέσματα αναζήτησης στο iOS app του Foursquare

Κάθε venue στο Foursquare περιλαμβάνει πολλές πληροφορίες, όπως όνομα, τύπος, διεύθυνση, επικοινωνία, φωτογραφίες, ωράριο, αριθμός χρηστών που το έχουν επισκευθεί, μέσος όρος βαθμολογίας, σχόλια και προτάσεις (tips) από επισκέπτες, κλπ. Επιπλέον, ένας χρήστης μπορεί να κάνει “check-in” σε μία τοποθεσία δηλώνοντας την παρουσία τους σε άλλους χρήστες και να αφήσει σχόλια για τις εντυπώσεις του, ώστε να βοηθήσει άλλους χρήστες στην επιλογή τους.

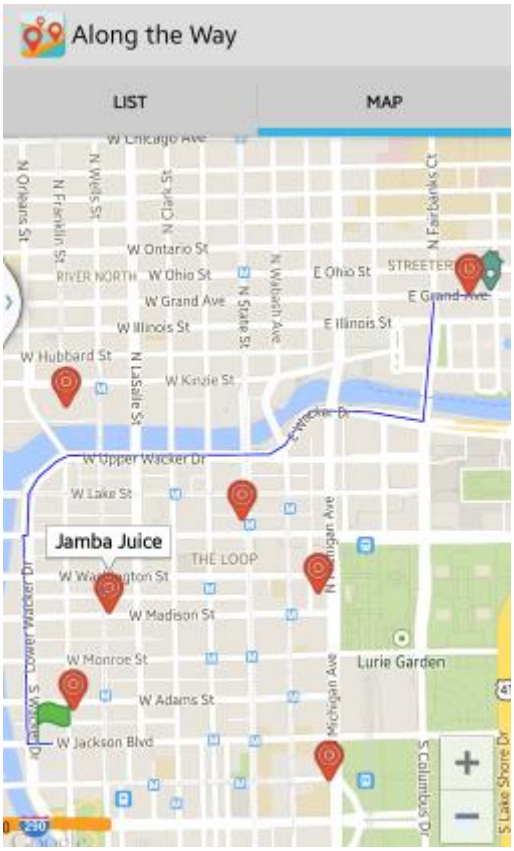


Σχήμα 1.2: Σελίδα ενός venue στο iOS app του Foursquare

Μέχρι το 2014, οι χρήστες είχαν τη δυνατότητα να μοιράζονται με φίλους τους τη δραστηριότητά τους, όπως την ώρα και το μέρος που επισκέπτονται και τις κριτικές τους, καθιστώντας το Foursquare ένα πολύ δημοφιλές κοινωνικό δίκτυο. Σήμερα, το social networking layer έχει ανεξαρτητοποιηθεί και αναπτύσσεται ως ξεχωριστή εφαρμογή με το όνομα Swarm. Παρ' όλα αυτά, το Foursquare διατηρεί πολλά στοιχεία ενός κοινωνικού δικτύου, αφού βασίζεται σε περιεχόμενο που έχουν προσθέσει χρήστες, όπως βαθμολογία και κριτικές. Άλλωστε, τα venues που προτείνονται σε κάθε χρήστη εξαρτώνται από το ιστορικό του, το προφίλ του, τις συνδέσεις του και τις προτιμήσεις του.

Το Foursquare, λοιπόν, περιλαμβάνει μια εκτενή συλλογή από venues για όλες τις πόλεις του κόσμου με την αναλυτική περιγραφή τους και ακριβείς πληροφορίες για την τοποθεσία τους. Παράλληλα, τα δεδομένα αυτά ανανεώνονται και εμπλουτίζονται συνεχώς από τους χρήστες της εφαρμογής. Γι' αυτούς τους λόγους, πολλές εφαρμογές που απεικονίζουν πληροφορίες σε χάρτες (Along The Way, Dine-O-Mat, Localscope, κλπ) χρησιμοποιούν

δεδομένα από το Foursquare, ενώ επιλέχθηκε ως πηγή δεδομένων και για το χάρτη της εφαρμογής που αναπτύχθηκε στην παρούσα εργασία.



Σχήμα 1.3: Η εφαρμογή Along The Way απεικονίζει σε χάρτη το δρόμο που πρέπει να ακολουθήσει ο χρήστης για να φτάσει στον προορισμό του, ενώ λαμβάνει από το Foursquare κάποια venues που βρίσκονται κοντά στη διαδρομή.

1.3 Αντικείμενο της διπλωματικής

Η αμεσότητα των ηλεκτρονικών χαρτών για την απεικόνιση πολλών τύπων δεδομένων και η πληθώρα πληροφοριών που μπορούν να εξαχθούν από τα κοινωνικά δίκτυα για πολλά θέματα αποτέλεσαν δύο βασικούς παράγοντες στη σύλληψη της ιδέας για την ανάπτυξη της παρούσας εργασίας.

Στόχος, λοιπόν, ήταν η συλλογή και αποθήκευση γεωγραφικών δεδομένων σε μορφή τέτοια, ώστε να είναι δυνατή η αξιοποίησή τους από πολλά εργαλεία και τεχνολογίες. Έπειτα, η δημιουργία ενός συστήματος που θα λαμβάνει από κοινωνικά δίκτυα πληροφορίες που αφορούν τοποθεσίες, επιχειρήσεις και εκδηλώσεις και θα τις αποθηκεύει σε μορφή συμβατή με αυτή των γεωγραφικών δεδομένων. Τέλος, η απεικόνιση όλων των παραπάνω δεδομένων σε χάρτη, δίνοντας τη δυνατότητα στο χρήστη να φιλτράρει τις πληροφορίες που τον ενδιαφέρουν.

Η εφαρμογή που αναπτύχθηκε, εκτελεί τις εξής λειτουργίες:

- Λαμβάνει από το διαδίκτυο γεωγραφικά δεδομένα για το χωρισμό των πόλεων στις ΗΠΑ σε πολύγωνα που αναπαριστούν επιμέρους γειτονιές και τα αποθηκεύει σε μια βάση δεδομένων.
- Για κάθε γειτονιά αναζητά στο Foursquare τις πιο δημοφιλείς τοποθεσίες, εκδηλώσεις και επιχειρήσεις (θα αναφέρονται γενικά ως venues) και τις αποθηκεύει στη βάση δεδομένων ανάλογα με τον τύπο τους και την θεματική τους κατηγορία.
- Εμφανίζει στο χάρτη μία πόλη χωρισμένη σε γειτονιές, όπου η κάθε μία είναι χρωματισμένη ανάλογα με τον αριθμό των venues κάποιας κατηγορίας (πχ εστιατόρια, μουσεία, κλπ).
- Επιτρέπει την απεικόνιση όλων των venues κάποιας κατηγορίας σε μια γειτονιά αναλυτικότερα, εμφανίζοντας πληροφορίες, όπως όνομα, τύπος και βαθμολογία χρηστών. Επίσης, δίνεται ο υπερσύνδεσμος για την επίσκεψη της αντίστοιχης σελίδας στο Foursquare, που παρέχει περισσότερες λεπτομέρειες.

Ο χρήστης έχει τη δυνατότητα να επιλέξει την πόλη που επιθυμεί και την κατηγορία των venues που τον ενδιαφέρει. Τα δεδομένα που συλλέγονται από το Foursquare αφορούν μία συγκεκριμένη χρονική περίοδο, οπότε ανανεώνονται συχνά από την εφαρμογή για την εμφάνιση πρόσφατων και αξιόπιστων αποτελεσμάτων.

Τέλος, η εφαρμογή μπορεί να συνδεθεί με μια εξωτερική βάση δεδομένων, σε περίπτωση που κάποιο άλλο σύστημα θέλει να αξιοποιήσει τα γεωγραφικά δεδομένα της. Για παράδειγμα, μπορεί να παρακολουθεί σε πραγματικό χρόνο αλλαγές σε μια βάση όπου αποθηκεύονται οι τοποθεσίες που παρατηρήθηκαν πυρκαγιές και να εμφανίσει στο χάρτη την τοποθεσία και περαιτέρω πληροφορίες για κάθε γεγονός.

1.4 Δομή εργασίας

Το κείμενο της διπλωματικής εργασίας αποτελείται από 6 κεφάλαια:

Στο κεφάλαιο 2 περιγράφεται η αρχιτεκτονική της εφαρμογής, ξεκινώντας με την παρουσίαση των συνιστωσών από τις οποίες απαρτίζεται σε ένα σχήμα. Στη συνέχεια, περιγράφονται η δομή της βάσης δεδομένων MongoDB, η λειτουργία του service που λαμβάνει τα γεωγραφικά δεδομένα και το service που αναζητά venues από το Foursquare API. Επιπλέον, παρουσιάζονται η λειτουργία ενός server που δέχεται requests από το χάρτη, εκτελεί ερωτήματα στη βάση δεδομένων και επιστρέφει τα κατάλληλα δεδομένα και οι λειτουργίες που πραγματοποιούνται για την εμφάνιση των δεδομένων στο χάρτη και την επιλογή της πόλης που θα απεικονιστεί. Τέλος, δίνεται κι ένα παράδειγμα σύνδεσης της εφαρμογής με μία άλλη βάση δεδομένων για την απεικόνιση στο χάρτη των τοποθεσιών όπου παρατηρήθηκε πυρκαγιά.

Στο κεφάλαιο 3 παρουσιάζονται τα εργαλεία με τα οποία αναπτύχθηκε η εφαρμογή. Πιο συγκεκριμένα, παρουσιάζονται τα βασικά χαρακτηριστικά της Java και του αντικειμενοστρεφούς προγραμματισμού, αναλύονται οι λόγοι επιλογής της MongoDB ως βάσης δεδομένων, τα πλεονεκτήματα της πλατφόρμας ανάπτυξης λογισμικού Node.JS και οι δυνατότητες του Mapbox ως mapping platform.

Το κεφάλαιο 4 περιέχει screenshots από όλες τις λειτουργίες της εφαρμογής με τη σύντομη επεξήγησή τους.

Στο κεφάλαιο 5 γίνεται η σύγκριση της εφαρμογής με το Tripadvisor και το Foursquare. Επιπλέον, περιγράφεται η επίδραση της ώρας αναζήτησης στον αριθμό των αποτελεσμάτων και παρουσιάζεται η δυνατότητα της εφαρμογής για τη μελέτη της δημοτικότητας των venues.

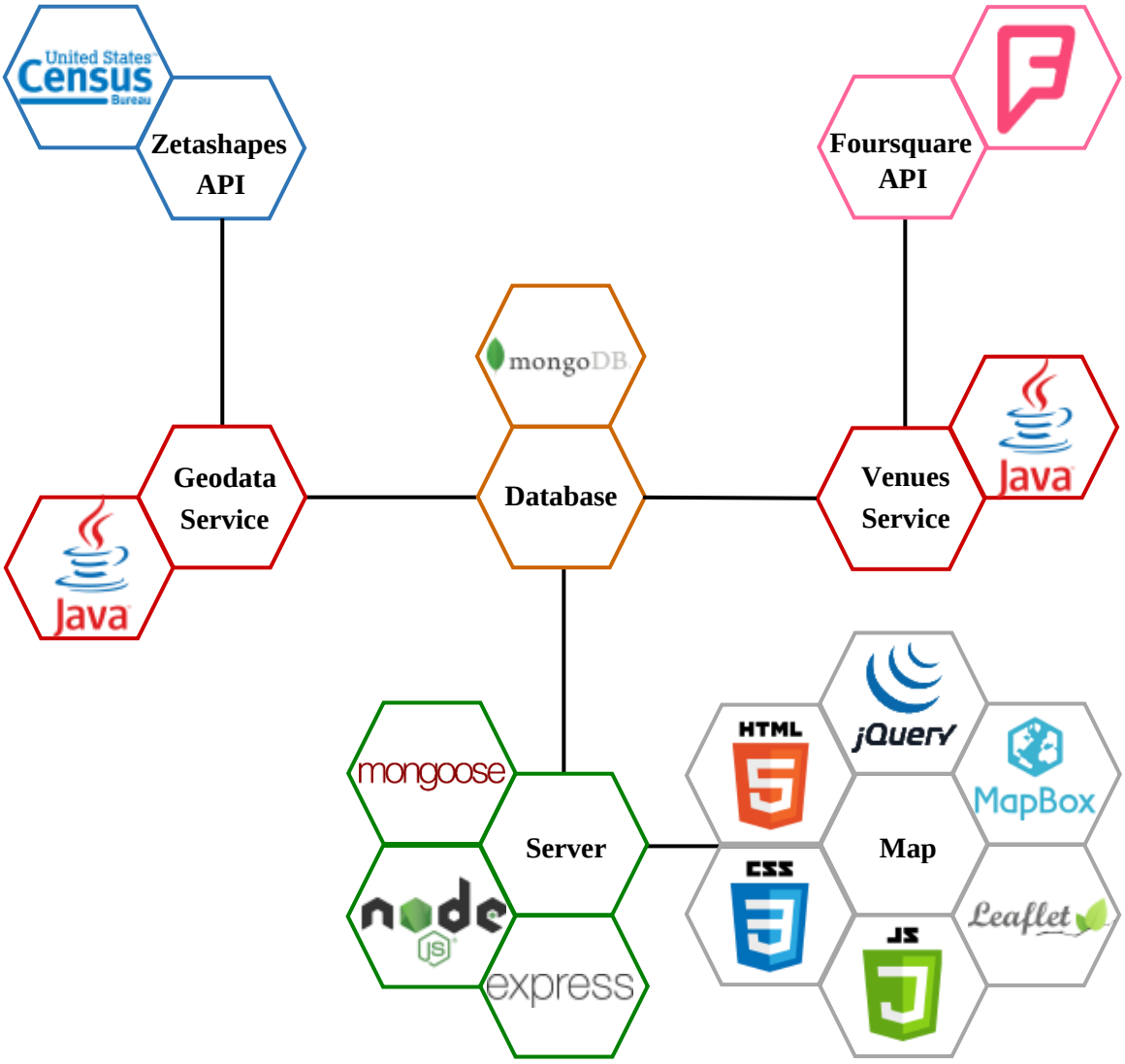
Τέλος, στο κεφάλαιο 6 προτείνονται μελλοντικές προσθήκες που θα μπορούσαν να βελτιώσουν την εφαρμογή.

Ο πηγαίος κώδικας όλων των τμημάτων της εφαρμογής παρατίθενται στο τέλος του εγγράφου, στα Παραρτήματα Α και Β, ενώ στο παράρτημα Γ δίνονται οδηγίες για την εγκατάσταση της εφαρμογής.

Κεφάλαιο 2
Αρχιτεκτονική Εφαρμογής

2.1 Γενική Εικόνα

Σε αυτό το κεφάλαιο θα αναλυθούν τα δομικά στοιχεία της εφαρμογής και οι λειτουργίες που επιτελούν. Μία υπηρεσία είναι υπεύθυνη για την ανάκτηση των γεωγραφικών δεδομένων και την αποθήκευσή τους σε μία βάση δεδομένων. Άλλη υπηρεσία αναζητά venues στο Foursquare και τα αποθηκεύει στη βάση δεδομένων. Ένας server είναι υπεύθυνος για το σωστό routing των requests του client, τη σύνδεση με τη βάση δεδομένων και την αποστολή του κατάλληλου response. Τέλος, ο χάρτης αποτελεί τη διεπαφή χρήστη για την απεικόνιση των επιθυμητών δεδομένων.

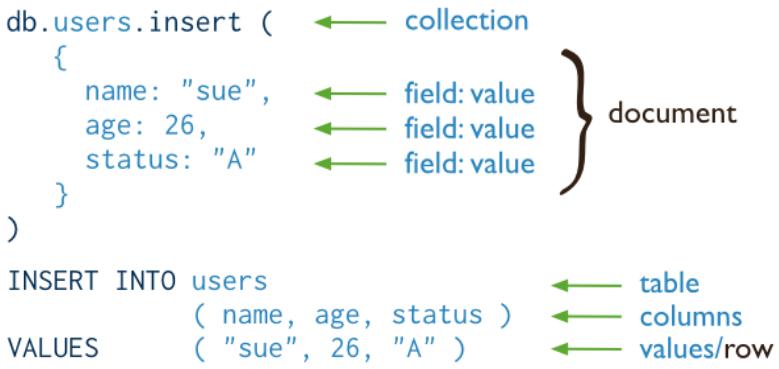


Σχήμα 2.1: Components της εφαρμογής και οι τεχνολογίες που χρησιμοποιήθηκαν για το καθένα

2.2 Βάση Δεδομένων MongoDB

Η βάση δεδομένων αποτελεί ένα σημαντικό στοιχείο της αρχιτεκτονικής της εφαρμογής, αφού σε αυτή αποθηκεύονται όλα τα δεδομένα που θα εμφανίζονται στο χάρτη. Η χρήση της κρίθηκε απαραίτητη, διότι βοηθάει στην καλύτερη οργάνωση των δεδομένων και στην ταχύτερη ανάκτηση οποιουδήποτε συνόλου τους από το front-end.

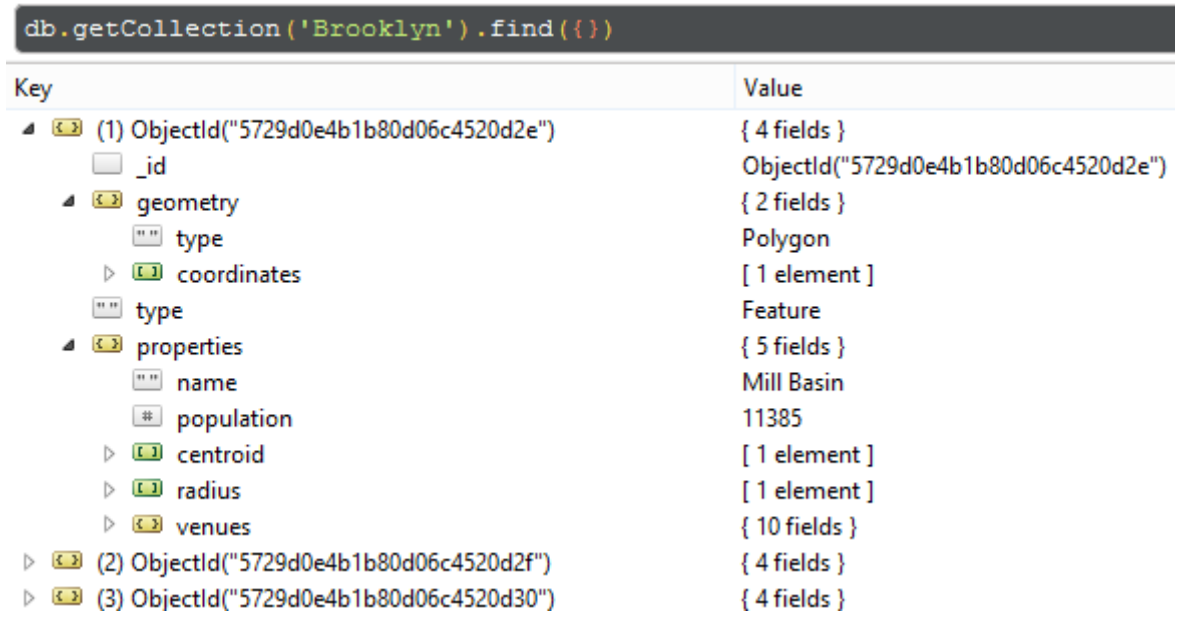
Μία βάση δεδομένων MongoDB [4] δομείται από συλλογές (collections), κάθε μία από τις οποίες περιέχει ένα σύνολο από έγγραφα (documents). Το έγγραφο είναι μία δομή δεδομένων που αποτελείται από ζεύγη πεδίων (fields) και των τιμών τους (values). Η δομή των εγγράφων μοιάζει με αυτή των JSON objects. Οι τιμές των πεδίων μπορεί να περιέχουν άλλα έγγραφα, πίνακες ή πίνακες εγγράφων.



Σχήμα 2.2: Η δομή ενός εγγράφου στη MongoDB (πάνω) συγκριτικά με έναν πίνακα σε SQL (κάτω)

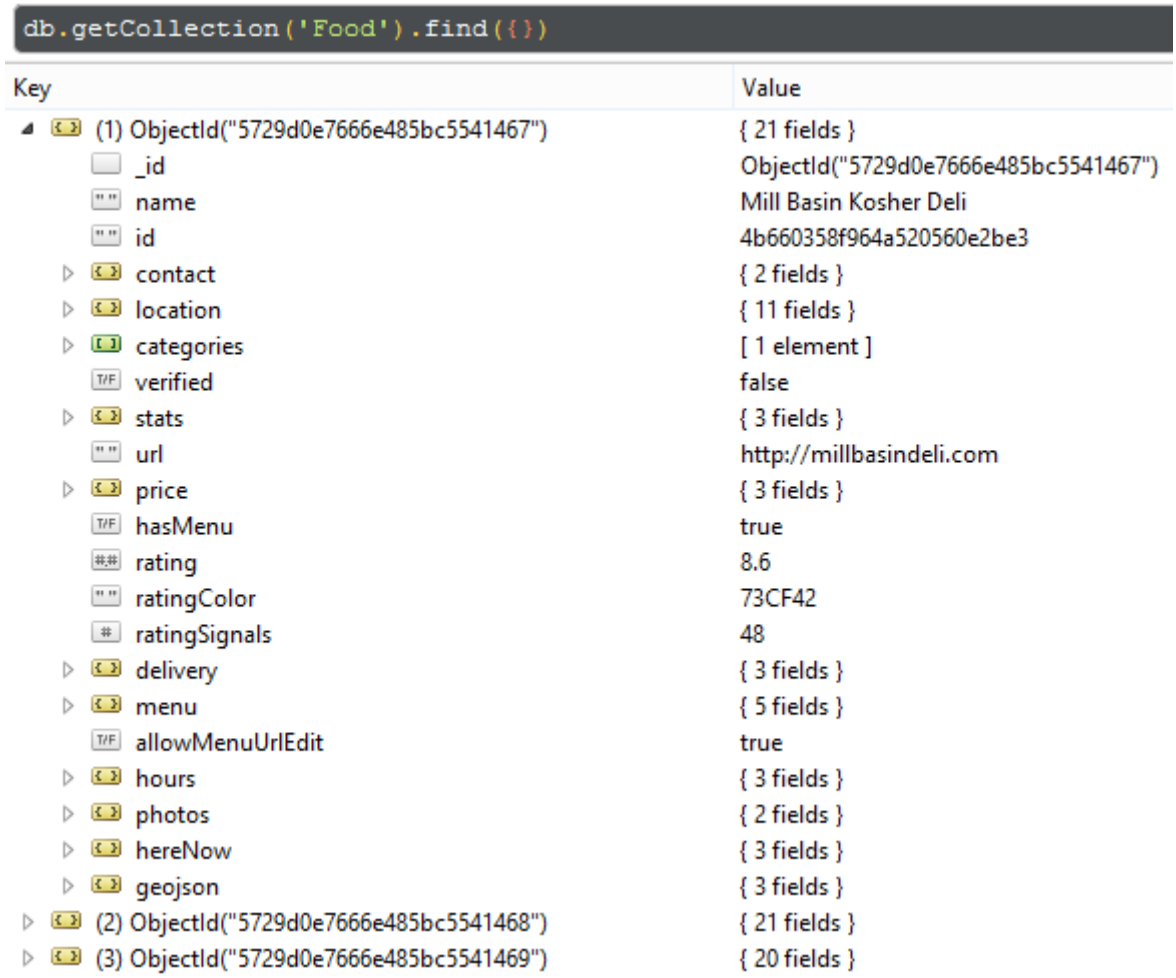
Η βάση δεδομένων της εφαρμογής που αναπτύχθηκε περιλαμβάνει δύο τύπους συλλογών, όσον αφορά τη δομή τους:

α) Τα collections που περιέχουν τα γεωγραφικά δεδομένα μια πόλης, όπως οι συντεταγμένες των κορυφών του πολυγώνου κάθε γειτονιάς. Κάθε collection περιγράφει μία πόλη και κάθε document μία γειτονιά. Η δομή ενός ολόκληρου εγγράφου (neighbourhood) παρουσιάζεται στην ενότητα 2.3.3.



Σχήμα 2.3: Η δομή ενός εγγράφου στο collection "Brooklyn" της MongoDB

β) Τα collections που περιέχουν όλα τα venues από το Foursquare (σε οποιαδήποτε γειτονιά) που ανήκουν στην ίδια κατηγορία. Κάθε collection αναπαριστά μία κατηγορία (food, shopping, κλπ) και κάθε document περιγράφει ένα venue με πληροφορίες, όπως όνομα, διεύθυνση, βαθμολογία, κλπ. Η δομή ενός ολόκληρου εγγράφου (venue) παρουσιάζεται στην ενότητα 2.4.3.



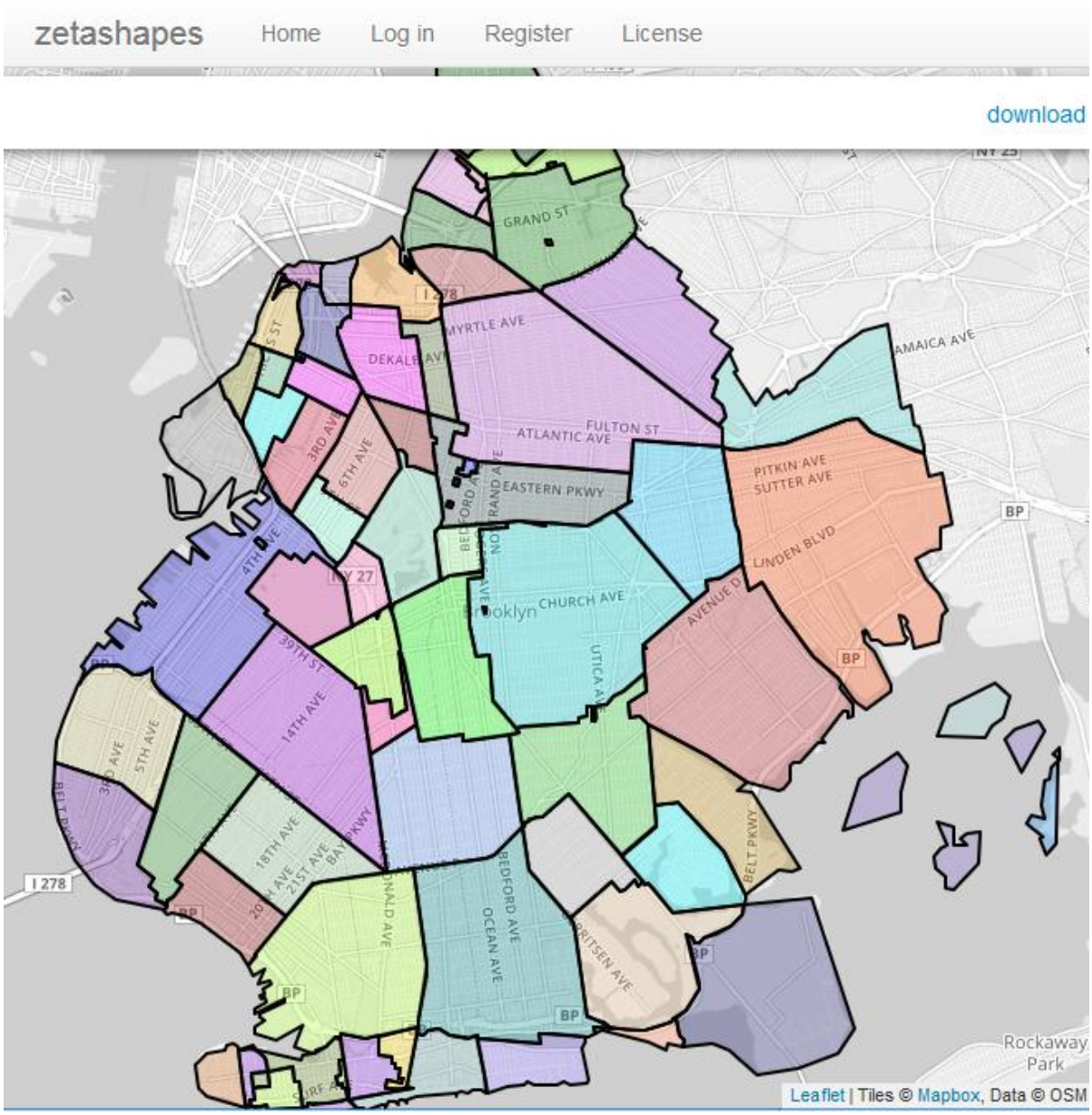
Σχήμα 2.4: Η δομή ενός εγγράφου στο collection "Food" της MongoDB

2.3 Service ανάκτησης και αποθήκευσης γεωγραφικών δεδομένων

Σε αυτή την ενότητα θα αναλυθούν οι λειτουργίες που υλοποιήθηκαν σε Java για την ανάκτηση των πολυγώνων που απαρτίζουν μια πόλη από το API του zetashapes.com, τον υπολογισμό του κύκλου που περιβάλλει κάθε πολύγωνο για την μετέπειτα αναζήτηση venues στο Foursquare και την αποθήκευση όλων των παραπάνω στη βάση δεδομένων σε GeoJSON format.

2.3.1 Neighbourhoods Project API

Τα γεωγραφικά δεδομένα για το χωρισμό μιας πόλης σε επιμέρους γειτονιές λαμβάνονται από το API του zetashapes.com. [5] Η αρχική πηγή τους είναι η βάση δεδομένων TIGER (**T**opologically **I**ntegrated **G**eographic **E**ncoding and **R**eferencing) του US Census Bureau [6] και ανήκουν στο public domain.



Σχήμα 2.5: Στιγμιότυπο από το zetashapes.com με το Brooklyn χωρισμένο σε γειτονιές

Για την ανάκτηση των γεωγραφικών δεδομένων μιας πόλης, το API του zetashapes.com δέχεται HTML GET requests με URL της μορφής:

http://zetashapes.com/api/neighborhoodsByArea?areaid=36061

Η παράμετρος areaid είναι ένα αναγνωριστικό μοναδικό για κάθε πόλη. Το API παρέχει δεδομένα για 30080 πόλεις των ΗΠΑ, τα ονόματα των οποίων είναι συγκεντρωμένα σε ένα έγγραφο κειμένου, όπου αντιστοιχίζονται με το αναγνωριστικό τους, πχ ["Los Angeles, CA", 06037]. Στα πλαίσια της εφαρμογής, λοιπόν, κατασκευάστηκε η μέθοδος `getAreaId(String area_name)` που δέχεται ως είσοδο το όνομα μίας πόλης, το αναζητά στο έγγραφο αυτό και επιστρέφει το areaid της.

Στη συνέχεια, στη μέθοδο `getNeighbourhoods(String areaname, String areaid)` κατασκευάζεται το κατάλληλο URL και στέλνεται το request στο API. Το response

είναι ένα έγγραφο μορφής GeoJSON που περιλαμβάνει πληροφορίες για όλες τις γειτονιές της πόλης που αναζητήθηκε.

Το GeoJSON [7] είναι ένα format για την κωδικοποίηση μιας ποικιλίας γεωγραφικών δομών δεδομένων. Ένα αντικείμενο (object) GeoJSON μπορεί να αντιπροσωπεύει μια γεωμετρία (geometry), ένα χαρακτηριστικό (feature) ή μια συλλογή χαρακτηριστικών (feature collection). Υποστηρίζονται οι ακόλουθοι τύποι γεωμετρίας: Point, LineString, Polygon, MultiPoint, MultiLineString, MultiPolygon και GeometryCollection. Ένα GeoJSON feature περιέχει ένα geometry object και πρόσθετες ιδιότητες (properties), ενώ ένα feature collection αντιπροσωπεύει μια λίστα από features. Ένα παράδειγμα φαίνεται παρακάτω:

```
{ "type": "FeatureCollection",
    "features": [
      { "type": "Feature",
        "geometry": {"type": "Point", "coordinates": [102.0, 0.5]},
        "properties": {"prop0": "value0"}
        },
      { "type": "Feature",
        "geometry": {
           "type": "Polygon",
           "coordinates": [
             [ [100.0, 0.0], [101.0, 0.0], [101.0, 1.0],
               [100.0, 1.0], [100.0, 0.0] ]
             ]
        },
        "properties": {
           "prop0": "value0",
           "prop1": {"this": "that"}
           }
        }
    ]
}
```

2.3.2 Γεωμετρικοί υπολογισμοί

Το response από το API περιλαμβάνει τις συντεταγμένες των κορυφών του πολυγώνου μιας γειτονιάς. Σκοπός της εφαρμογής είναι να αναζητήσει στο Foursquare venues σε αυτή την περιοχή. Όμως, η αναζήτηση στο Foursquare γίνεται σε κυκλικό δίσκο κι όχι σε πολύγωνο.

Γι' αυτό το λόγο κατασκευάστηκαν δύο μέθοδοι για τον υπολογισμό του κέντρου και της ακτίνας του κύκλου που περιβάλλει ένα πολύγωνο.

- Η μέθοδος `calCentroid(JsonArray coords)` δέχεται ως είσοδο τον πίνακα των συντεταγμένων των κορυφών ενός πολυγώνου και επιστρέφει τις συντεταγμένες του κέντρου βάρους του, που υπολογίζονται ως οι αριθμητικοί μέσοι των συντεταγμένων όλων των κορυφών.
- Η μέθοδος `calRadius(JsonArray coords, List<Double> centroid)` δέχεται ως είσοδο τον πίνακα με τις συντεταγμένες των κορυφών ενός πολυγώνου και μια λίστα με τις συντεταγμένες του κέντρου βάρους του. Επιστρέφει τη μέγιστη απόσταση μεταξύ κάποιας κορυφής και του κέντρου βάρους του πολυγώνου, η οποία υπολογίζεται με τη χρήση του τύπου Haversine [8]:

$\alpha = \sin^2(\Delta\varphi/2) + \cos\varphi_1 \cdot \cos\varphi_2 \cdot \sin^2(\Delta\lambda/2)$
$c = 2 \cdot \arctan(\sqrt{a} / \sqrt{1-a})$
$d = R \cdot c$
όπου: φ το γεωγραφικό πλάτος, λ το γεωγραφικό μήκος και R η ακτίνα της Γης

2.3.3 Αποθήκευση GeoJSON στη MongoDB

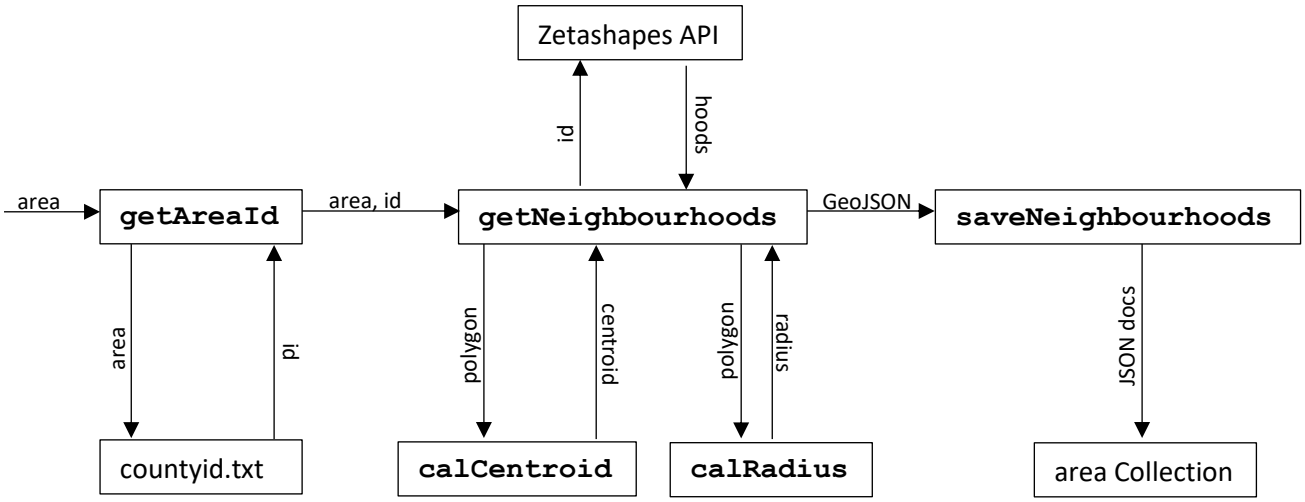
Τέλος, έχοντας όλα τα παραπάνω δεδομένα για μία πόλη, η εφαρμογή τα αποθηκεύει στο collection της MongoDB με όνομα αυτό της πόλης. Επειδή η MongoDB είναι document-oriented βάση δεδομένων, τα δεδομένα αποθηκεύονται ως έγγραφα μορφής GeoJSON με τη χρήση της μεθόδου `saveNeighbourhoods(String areaname, JsonObject hoods)`. Παρακάτω φαίνεται ένα παράδειγμα τέτοιου εγγράφου:

```
{
    "_id" : ObjectId("571339c1299f210a38c19e89"),
    "geometry" : {
        "type" : "Polygon",
        "coordinates" : [
            [
                [
                    -73.922498,
                    40.609809
                ],
               [
                    -73.92288,
                    40.610164
                ],

                ...
            ]
        ]
    },
    "type" : "Feature",
    "properties" : {
        "name" : "Mill Basin",
        "population" : 11385,
        "centroid" : [
            {
                "lng" : -73.9156251886793,
                "lat" : 40.6133409056604
            }
        ],
        "radius" : [
            1768
        ]
    }
}
```

2.3.4 Neighbourhoods Service – Block Διάγραμμα

Στο παρακάτω διάγραμμα παρουσιάζονται ως blocks οι βασικές μέθοδοι που υλοποιήθηκαν για το service και περιγράφηκαν στην προηγούμενη ενότητα, ενώ με βέλη υποδεικνύεται η ροή των δεδομένων μεταξύ τους.



Σχήμα 2.6: Neighbourhoods Service Block Diagram

2.4 Service αναζήτησης των venues στο Foursquare

Σε αυτή την ενότητα περιγράφονται οι λειτουργίες που υλοποιήθηκαν σε Java για την ανάκτηση των κατηγοριών των venues από το Foursquare API, την αναζήτηση των δημοφιλέστερων venues για κάθε γειτονιά μιας πόλης και την αποθήκευσή τους στο κατάλληλο collection της MongoDB.

2.4.1 Foursquare API & Categories Search

Το Foursquare διατηρεί μία εκτενή συλλογή από τοποθεσίες, επιχειρήσεις, εκδηλώσεις και οργανισμούς και επιτρέπει σε προγραμματιστές να τη χρησιμοποιούν ως βάση δεδομένων για τις εφαρμογές τους χωρίς να απαιτείται authentication. Το API [9] αποτελεί τη διεπαφή για την επικοινωνία μιας εφαρμογής με τη βάση δεδομένων του Foursquare, δέχεται HTTP GET Requests σε μορφή URL και επιστρέφει ένα response σε μορφή JSON.

Η εφαρμογή που αναπτύχθηκε χρειάζεται σε πρώτη φάση τις κατηγορίες των venues στο Foursquare και όλες τις υποκατηγορίες τους, με σκοπό τη μετέπειτα αποθήκευση των venues στο κατάλληλο collection της MongoDB. Δημιουργεί, λοιπόν, το ακόλουθο URL και μέσω της μεθόδου `getCategories(String url)` στέλνει ένα GET request:

https://api.foursquare.com/v2/venues/categories?oauth_token=DOZIUZ5HPYTZ2UWH2EFE0QBEAUFBVGINFET2F0GHKK0FTDKM&v=20160429

Το response από το API είναι της παρακάτω μορφής:


```
{
   "meta":{
      "code":200,
      "requestId":"57238aad498e5efb40efd98c"
   },
   "notifications":[
      {
         "type":"notificationTray",
         "item":{
            "unreadCount":0
         }
      }
   ],
   "response":{
      "categories":[
         {
            "id":"4d4b7104d754a06370d81259",
            "name":"Arts & Entertainment",
            "pluralName":"Arts & Entertainment",
            "shortName":"Arts & Entertainment",
            "icon":{

"prefix":"https:\/\/ss3.4sqi.net\/img\/categories_v2\/arts_entertain
ment\/default_",
               "suffix":".png"
            },
            "categories":[
               {
                  "id":"56aa371be4b08b9a8d5734db",
                  "name":"Amphitheater",
                  "pluralName":"Amphitheaters",
                  "shortName":"Amphitheater",
                  "icon":{

"prefix":"https:\/\/ss3.4sqi.net\/img\/categories_v2\/arts_entertain
ment\/default_",
                     "suffix":".png"
                  },
                  ...
```


Το response περιλαμβάνει ένα πίνακα με JSON objects, όπου το καθένα αναπαριστά μία κατηγορία (πχ Arts & Entertainment). Σε κάθε κατηγορία, υπάρχει άλλος ένας πίνακας με JSON objects, που περιλαμβάνει όλες τις υποκατηγορίες (πχ Amphitheater). Η εφαρμογή, λοιπόν, κρατάει σε μία λίστα από JSON objects τα ονόματα των κατηγοριών και τα ids των υποκατηγοριών της καθεμίας, πχ:

```
{
    "name": "Arts & Entertainment",
    "child_cat_ids": ["56aa371be4b08b9a8d5734db", ...]
}
```

2.4.2 Explore Foursquare Venues

Σε επόμενη φάση, γίνεται η αναζήτηση των Foursquare venues σε κάθε γειτονιά της πόλης. Για την προετοιμασία του URL για το request ακολουθούνται τα επόμενα βήματα:

- Ανακτώνται από τη MongoDB τα γεωγραφικά δεδομένα από το collection με όνομα αυτό της πόλης μέσω της μεθόδου `findHoods(String areaname)`. Η μέθοδος επιστρέφει έναν πίνακα με JSON objects, όπου το καθένα αναπαριστά μία γειτονιά.
- Από κάθε JSON object (γειτονιά) λαμβάνονται η ακτίνα (radius) και οι συντεταγμένες του κέντρου του κύκλου (centroid) που περιβάλλει τη γειτονιά.
- Δημιουργείται το URL με παραμέτρους τις συντεταγμένες του κέντρου, την ακτίνα αναζήτησης και το μέγιστο αριθμό αποτελεσμάτων, που είναι 50 για κάθε request. Η μορφή ενός τέτοιου URL φαίνεται παρακάτω:
  https://api.foursquare.com/v2/venues/explore?client_id=UR1CQXWS0U5Z5YYX0QHJUDBJ2GLIVSZNASYOL14OGMLVULFI&client_secret=0OCNNY3JZJ4FBDSDAWQQPHKGOQ425GAMZCHWBR5IJ0301240&v=20160429&ll=40.72610609210526,-73.9857985394737&limit=50&radius=1321.0

Στη μέθοδο `exploreVenues(String url)` στέλνεται στο API το GET request. Το response είναι ένα JSON document, το οποίο περιέχει έναν πίνακα με JSON objects, το καθένα από τα οποία περιγράφει ένα venue. Περιέχει πληροφορίες, όπως όνομα, κατηγορία, διεύθυνση, στοιχεία επικοινωνίας, βαθμολογία, σχόλια χρηστών, ωράριο, κλπ. Ένα παράδειγμα τέτοιου εγγράφου παρουσιάζεται στην ενότητα 2.4.3.

Τέλος, η μέθοδος `createGeoJson(JsonObject venue)` δημιουργεί ένα GeoJSON object με πληροφορίες για την τοποθεσία ενός venue, με σκοπό να γίνουν αργότερα κάποια geospatial queries στη βάση δεδομένων, τα οποία απαιτούν το συγκεκριμένο format. Ένα παράδειγμα είναι το εξής:

```
{
        "type" : "Feature",
        "geometry" : {
            "type" : "Point",
            "coordinates" : [
                -73.9099154220681,
                40.6168874075729
            ]
        },
        "properties" : {
            "name" : "La Villa Pizzeria",
            "category" : "Pizza Place",
            "id" : "4b193722f964a52056d923e3"
        }
}
```

2.4.3 Αποθήκευση των Venues στη MongoDB

Έχοντας, λοιπόν, όλα τα παραπάνω δεδομένα για τα venues σε μία γειτονιά, η εφαρμογή αποθηκεύει κάθε ένα από αυτά στο collection της MongoDB με όνομα αυτό της κατηγορίας που ανήκουν. Επειδή στο JSON object που περιγράφει ένα venue περιλαμβάνονται μόνο το

όνομα και το id της υποκατηγορίας του, είναι αναγκαίο να βρεθεί το όνομα της γενικότερης κατηγορίας που ανήκει, πχ η υποκατηγορία Burger Joint έχει parent category τη Food.

Γι' αυτό το λόγο κατασκευάστηκε η μέθοδος `getCategory(String cat_id)`, η οποία δέχεται ως είσοδο το id της child category, το αναζητά στη λίστα των κατηγοριών που αναφέρθηκε στην ενότητα 2.4.1 και επιστρέφει το όνομα της parent category.

Στη συνέχεια, στη μέθοδο `saveVenue(String catname, JsonObject venue, JsonObject geojson)` αποθηκεύεται το JSON document ενός venue στο collection με όνομα αυτό της κατηγορίας του. Ένα παράδειγμα τέτοιου εγγράφου φαίνεται παρακάτω:


```
{
    "_id" : ObjectId("571339c3bae05d5e03160143"),
    "name" : "Mill Basin Kosher Deli",
    "id" : "4b660358f964a520560e2be3",
    "contact" : {
        "phone" : "7182414910",
        "formattedPhone" : "(718) 241-4910"
    },
    "location" : {
        "address" : "5823 Avenue T",
        "crossStreet" : "E 59th St",
        "lat" : 40.6156116666667,
        "lng" : -73.91818,
        "distance" : 1450,
        "postalCode" : "11234",
        "cc" : "US",
        "city" : "Brooklyn",
        "state" : "NY",
        "country" : "United States",
        "formattedAddress" : [
            "5823 Avenue T (E 59th St)",
            "Brooklyn, NY 11234",
            "United States"
        ]
    },
    "categories" : [
        {
            "id" : "4bf58dd8d48988d146941735",
            "name" : "Deli / Bodega",
            "pluralName" : "Delis / Bodegas",
            "shortName" : "Deli / Bodega",
            "icon" : {
                "prefix" :
"https://ss3.4sqi.net/img/categories_v2/food/deli_",
                "suffix" : ".png"
            },
            "primary" : true
        }
    ],
    "verified" : false,
    "stats" : {
        "checkinsCount" : 1145,
        "usersCount" : 489,
        "tipCount" : 15
```

```
    },
    "url" : "http://millbasindeli.com",
    "price" : {
        "tier" : 3,
        "message" : "Expensive",
        "currency" : "$"
    },
    "hasMenu" : true,
    "rating" : 8.5,
    "ratingColor" : "73CF42",
    "ratingSignals" : 48,
    "delivery" : {
        "id" : "34966",
        "url" : "http://www.seamless.com/food-
delivery/restaurant.34966.r?a=1026&utm_source=Foursquare&utm_medium=
affiliate&utm_campaign=SeamlessOrderDeliveryLink",
        "provider" : {
            "name" : "seamless"
        }
    },
    "menu" : {
        "type" : "Menu",
        "label" : "Menu",
        "anchor" : "View Menu",
        "url" : "https://foursquare.com/v/mill-basin-kosher-
deli/4b660358f964a520560e2be3/menu",
        "mobileUrl" :
"https://foursquare.com/v/4b660358f964a520560e2be3/device_menu"
    },
    "allowMenuUrlEdit" : true,
    "hours" : {
        "status" : "Closed until 9:00 AM",
        "isOpen" : false,
        "isLocalHoliday" : false
    },
    "photos" : {
        "count" : 0,
        "groups" : []
    },
    "hereNow" : {
        "count" : 0,
        "summary" : "Nobody here",
        "groups" : []
    },
    "geojson" : {
        "type" : "Feature",
        "geometry" : {
            "type" : "Point",
            "coordinates" : [
                -73.91818,
                40.6156116666667
            ]
        },
        "properties" : {
            "name" : "Mill Basin Kosher Deli",
            "category" : "Deli / Bodega",
```

```
            "id" : "4b660358f964a520560e2be3"
        }
    }
}
```

2.4.4 Geospatial Queries στη MongoDB

Εκτελώντας τις παραπάνω λειτουργίες επαναληπτικά για κάθε γειτονιά μιας πόλης, υπάρχουν πλέον στη βάση δεδομένων όλα τα απαραίτητα δεδομένα. Μια ακόμη λειτουργία που κρίθηκε αναγκαία για την μετέπειτα εμφάνιση των δεδομένων στο χάρτη ήταν η ανανέωση του εγγράφου κάθε γειτονιάς με τον αριθμό των venues ανά κατηγορία που βρέθηκαν σε αυτή.

Υπενθυμίζεται πως τα αποτελέσματα της αναζήτησης στο Foursquare αφορούν τον κύκλο που περιβάλλει μία γειτονιά κι όχι το πολύγωνό της. Έτσι, λοιπόν, υλοποιήθηκε η μέθοδος `countVenues(JsonObject hood, String catname)` που μετράει τα venues της κατηγορίας catname που βρίσκονται μέσα στο πολύγωνο της γειτονιάς hood. Παράδειγμα ενός τέτοιου query στη MongoDB σε Javascript είναι το εξής:

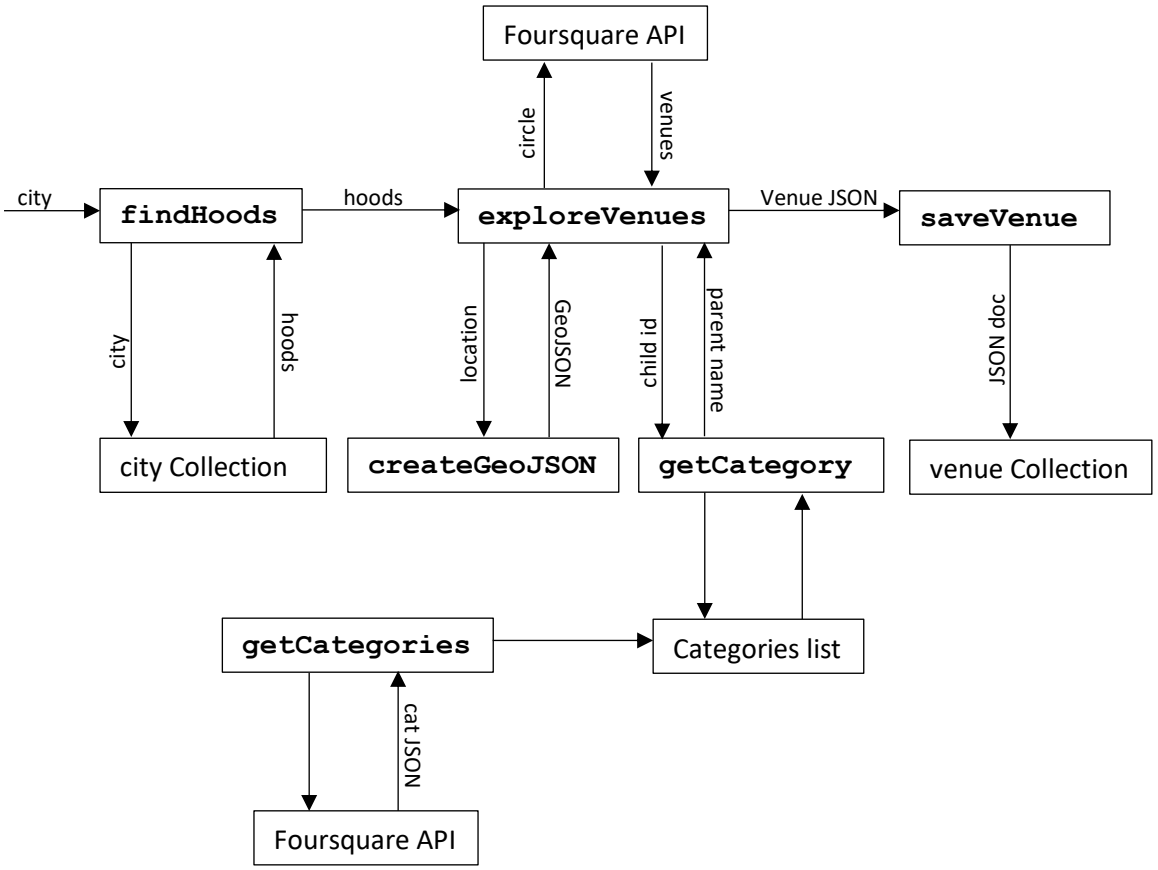
```
var neighbourhood =
db.getCollection('Manhattan').findOne({"properties.name":
"Gramercy"});
var venues =
db.getCollection('Food').count({"geojson.geometry":{$geoWithin:{$geo
metry:neighbourhood.geometry}}});
```

Τέλος, έχοντας μετρήσει τα venues όλων των κατηγοριών, η μέθοδος `updateHood(String areaname, JsonObject hood, String catname, long count)` προσθέτει στο έγγραφο της γειτονιάς ένα JSON object όπως το παρακάτω:

```
        "venues" : {
            "Arts_Entertainment" : NumberLong(0),
            "College_University" : NumberLong(0),
            "Event" : NumberLong(0),
            "Food" : NumberLong(8),
            "NightlifeSpot" : NumberLong(0),
            "Outdoors_Recreation" : NumberLong(0),
            "Professional_OtherPlaces" : NumberLong(0),
            "Residence" : NumberLong(0),
            "Shop_Service" : NumberLong(5),
            "Travel_Transport" : NumberLong(0)
        }
```

2.4.5 Venues Service – Block Διάγραμμα

Στο παρακάτω διάγραμμα παρουσιάζονται ως blocks οι βασικές μέθοδοι που υλοποιήθηκαν για το service και περιγράφηκαν στην προηγούμενη ενότητα, ενώ με βέλη υποδεικνύεται η ροή των δεδομένων μεταξύ τους.



Σχήμα 2.7: Venues Service Block Diagram

2.5 Server & Routing

Στις προηγούμενες ενότητες αναλύθηκαν οι λειτουργίες που υλοποιήθηκαν σε Java για την εύρεση και αποθήκευση των δεδομένων που θα απεικονιστούν στο χάρτη της εφαρμογής. Σε αυτή την ενότητα παρουσιάζεται η δομή ενός ενδιάμεσου layer, που μεσολαβεί στην επικοινωνία των επιμέρους τμημάτων του συστήματος. Συγκεκριμένα, περιγράφεται η λειτουργία ενός server που διαχειρίζεται τα requests που γίνονται από το χάρτη και πραγματοποιεί τα κατάλληλα queries στη βάση δεδομένων για την ανάκτηση των κατάλληλων δεδομένων.

Ο server της εφαρμογής υλοποιήθηκε με τη χρήση του Node.JS runtime environment [10] και του Express web framework [11] σε γλώσσα προγραμματισμού Javascript. Η εφαρμογή ξεκινάει τον server, που "ακούει" στην πύλη 3000 για συνδέσεις. Ο server δέχεται HTTP requests σε μορφή URL. Το routing καθορίζει τον τρόπο που ο server θα απαντήσει σε ένα request. Για κάθε έγκυρο request, έχει υλοποιηθεί μία handler function που ετοιμάζει το κατάλληλο response.

2.5.1 Mongoose & Handlers

Ο router, για κάθε request καλεί μία handler function, η οποία κάνει κάποια queries στη MongoDB και στέλνει ως response τα κατάλληλα δεδομένα. Ο driver της Node.JS για τη MongoDB που χρησιμοποιήθηκε στην εφαρμογή είναι ο Mongoose [12]. Η σύνδεση με τη βάση δεδομένων "foursquare" γίνεται ως εξής:

```
var mongoose = require('mongoose');
mongoose.connect('mongodb://localhost:27017/foursquare', function
(error) {
    if (error) {
        console.log(error);
    }
});
```

Στη συνέχεια, ορίζεται η δομή των collections, στα οποία θα γίνουν τα queries. Δηλώνονται δύο schemas, ένα για τα collections των πόλεων κι ένα για εκείνα των venues.

```
var Schema = mongoose.Schema;

var CitySchema = new Schema({
    type: String,
    geometry: Schema.Types.Mixed,
    properties: {name: String, centroid: Array, radius: Array}
});

var VenueSchema = new Schema({
  name: String,
  geojson: {geometry: Schema.Types.Mixed, properties:
Schema.Types.Mixed}
});
```

Παρακάτω παρουσιάζεται η handler function για requests που αφορούν το σύνολο των γειτονιών μιας πόλης. Το HTTP GET request γίνεται με ένα URL της μορφής `/hoods/:city`, όπου η παράμετρος "city" παίρνει ως τιμή το όνομα μιας πόλης. Μέσω του Mongoose, γίνεται ένα find query στο αντίστοιχο collection της MongoDB και το response είναι ένας πίνακας από JSON objects, όπου το καθένα αναπαριστά μία γειτονιά, με δομή αυτή που παρουσιάστηκε στην ενότητα 2.3.3.

```
router.get('/hoods/:city', function (req, res) {
    var City = mongoose.model('City', CitySchema, req.params.city);
    City.find({},{"_id": 0}, function (err, docs) {
        res.json(docs);
    });
});
```

Με τον ίδιο τρόπο λειτουργεί και η handler function για requests που αφορούν τα venues σε μία γειτονιά. Το URL του HTTP GET request είναι της μορφής `/venues/:city/:hood/:category` και περιέχει τρεις παραμέτρους: city, hood και category. Εκτελείται πρώτα το query για την εύρεση του πολυγώνου της γειτονιάς κι έπειτα το query που αναζητά στο collection τα venues της κατηγορίας που βρίσκονται μέσα στο

πολύγωνο. Το response είναι ένας πίνακας από JSON objects, που περιέχουν μόνο το field "geojson" του εγγράφου που παρουσιάστηκε στην ενότητα 2.4.3, δηλαδή την τοποθεσία ενός venue σε GeoJSON format.

```
router.get('/venues/:city/:hood/:category', function (req, res) {
    var City = mongoose.model('City', CitySchema, req.params.city);
    var Venue = mongoose.model('Venue', VenueSchema, req.params.category);
    City.findOne({"properties.name": req.params.hood},{"_id": 0}, function
(err, docs) {
        Venue.find({"geojson.geometry": {$geoWithin: {$geometry:
docs.geometry}}}, {"_id": 0, "geojson": 1}, function (err2, docs2) {
            res.json(docs2);
        });
    });
});
```

2.5.2 Rendering Views

Τέλος, ο router είναι υπεύθυνος και για την αποστολή της HTML σελίδας που απεικονίζει το χάρτη στον client. Για ένα HTTP GET request με URL της μορφής `/map/:city/:category`, το response θα είναι η HTML σελίδα του χάρτη, στην οποία θα περαστούν οι δύο παράμετροι city και category για την απεικόνιση των κατάλληλων δεδομένων.

Αυτό γίνεται δυνατό με τη χρήση ενός Jade template [13]. Η template engine του Express framework δίνει τη δυνατότητα διαχείρισης στατικών αρχείων σε μια εφαρμογή, όπως εικόνες, html, κλπ. Κατά την εκτέλεση, η template engine αντικαθιστά τις μεταβλητές σε ένα template με τις κατάλληλες τιμές και με τη μέθοδο "render" μετατρέπει το Jade template σε HTML αρχείο και το στέλνει στον client.

Στο παρακάτω παράδειγμα, γίνεται render το αρχείο map.jade, που είναι το template της html σελίδας που απεικονίζει το χάρτη και στο οποίο περνώνται οι τιμές των παραμέτρων city και category του URL του HTTP GET request.

```
router.get('/map/:city/:category', function(req,res) {
    res.render('map', {
        city: req.params.city,
        category: req.params.category,
    });
});
```

2.6 Leaflet Map

Η διεπαφή χρήστη της εφαρμογής είναι ένας διαδραστικός χάρτης, που απεικονίζει μία πόλη που επιλέγει ο χρήστης χωρισμένη σε γειτονιές. Κάθε γειτονιά είναι χρωματισμένη ανάλογα με το πλήθος των Foursquare venues μιας κατηγορίας που ενδιαφέρει το χρήστη.

Το base layer του χάρτη, που εμπλουτίζεται αργότερα με τα δεδομένα της εφαρμογής, λαμβάνεται μέσω του Mapbox API. Τα δεδομένα των χαρτών του Mapbox [14] έχουν πολλές

πηγές, όπως το OpenStreetMaps και τη NASA, ενώ στην ανανέωσή τους συμβάλλουν και οι χρήστες των εφαρμογών που χρησιμοποιούν το Mapbox, όπως το Strava και το Runkeeper.

Το customization της εμφάνισης του χάρτη και η υποστήριξη των λειτουργιών για την επαφή με το χρήστη γίνονται με τη βοήθεια της Javascript βιβλιοθήκης Leaflet [15]. Χρησιμοποιείται από την εφαρμογή για την προσθήκη των layers, markers, legend, popups, mouse events, τα οποία θα περιγραφούν στις επόμενες ενότητες.

2.6.1 Map Layers

Το χαμηλότερο layer του χάρτη είναι ένα tile layer και δημιουργείται ως εξής:

```
L.tileLayer('https://api.tiles.mapbox.com/v4/{id}/{z}/{x}/{y}.png?',
{
    maxZoom : 18,
    id : 'mapbox.light'
    }).addTo(map);
```

Ένα παράδειγμα του “mapbox light” tile layer φαίνεται παρακάτω:

Σχήμα 2.8: Το “mapbox light” tile layer

Πάνω από το tile layer προστίθενται τα layers που απεικονίζουν τα πολύγωνα των γειτονιών. Το Leaflet επιτρέπει την εισαγωγή δεδομένων για τα layers σε GeoJSON format, όπως δηλαδή είναι αποθηκευμένα στη βάση δεδομένων της εφαρμογής.

Με τη χρήση της jQuery, γίνεται ένα request για την ανάκτηση του JSON document που περιγράφει τα γεωγραφικά δεδομένα. Το URL του HTTP GET request έχει τη μορφή /hoods/:city κι όπως περιγράφηκε στην ενότητα 2.5.2, ο router αναλαμβάνει την ανάκτηση των δεδομένων. Το response είναι ένας πίνακας από features με τη μορφή JSON object και για κάθε feature, δηλαδή γειτονιά, δημιουργείται ένα layer. Η jQuery μέθοδος φαίνεται παρακάτω:

```
$.getJSON('/hoods/' + '#{city}', function(result){
    geojson = L.geoJson(result, {
        style : style,
        onEachFeature : onEachFeature
    }).addTo(map);
});
```

Για κάθε feature layer, εκτελούνται οι συναρτήσεις “style” και “getColor”, που καθορίζουν το χρώμα του πολυγώνου ανάλογα με τον αριθμό των venues μιας κατηγορίας σε αυτό. Όπως περιγράφηκε στην ενότητα 2.4.4, αυτή η πληροφορία βρίσκεται στο πεδίο “venues” του εγγράφου μιας γειτονιάς.

```
function style(feature) {
    return {
        weight : 2,
        opacity : 1,
        color : 'white',
        dashArray : '3',
        fillOpacity : 0.7,
        fillColor : getColor(feature.properties.venues.#{category})
    };
}
function getColor(d) {
    return d > 70 ? '#016450' : d > 60 ? '#02818a'
    : d > 50 ? '#3690c0' : d > 40 ? '#67a9cf'
    : d > 30 ? '#a6bddb' : d > 20 ? '#d0d1e6'
    : d > 10 ? '#ece2f0' : '#fff7fb';
}
```

Με παρόμοιο τρόπο, δημιουργούνται επιπλέον layers για τα markers που αναπαριστούν τα venues σε μία γειτονιά. Η jQuery μέθοδος που φαίνεται παρακάτω καλείται όταν ο χρήστης κάνει click πάνω στο πολύγωνο μιας γειτονιάς κι ως αποτέλεσμα δημιουργείται ένα layer για κάθε venue. Σε αυτό το σημείο γίνεται πλήρως κατανοητός κι ο λόγος για τον οποίο προστέθηκε το πεδίο “geojson” στο έγγραφο που περιγράφει ένα venue (ενότητα 2.4.2), αφού το Leaflet χρειάζεται αυτό το πεδίο για τη δημιουργία του GeoJSON layer.

```
$.getJSON('/venues/' + '#{city}/' + feature.properties.name +
'/#{category}', function(result){
    L.geoJson(result, {
        onEachFeature: onEachFeature2,
        pointToLayer: function (feature, latlng) {
            return L.marker(latlng);
        }
    }).addTo(map);
});
```

Για κάθε marker layer, εκτελείται η συνάρτηση “onEachFeature2”, όπου προστίθεται στον marker το περιεχόμενο του popup που εμφανίζεται όταν γίνει click στον marker. Το περιεχόμενο αυτό λαμβάνεται από το πεδίο “geojson ” του εγγράφου ενός venue.

```
function onEachFeature2(feature, layer) {
    var popupContent = '<a
href="https://foursquare.com/v/'+feature.properties.id+'"class="venu
e">'+feature.properties.name+'</a><br>'+feature.properties.category;
    layer.bindPopup(popupContent);
}
```

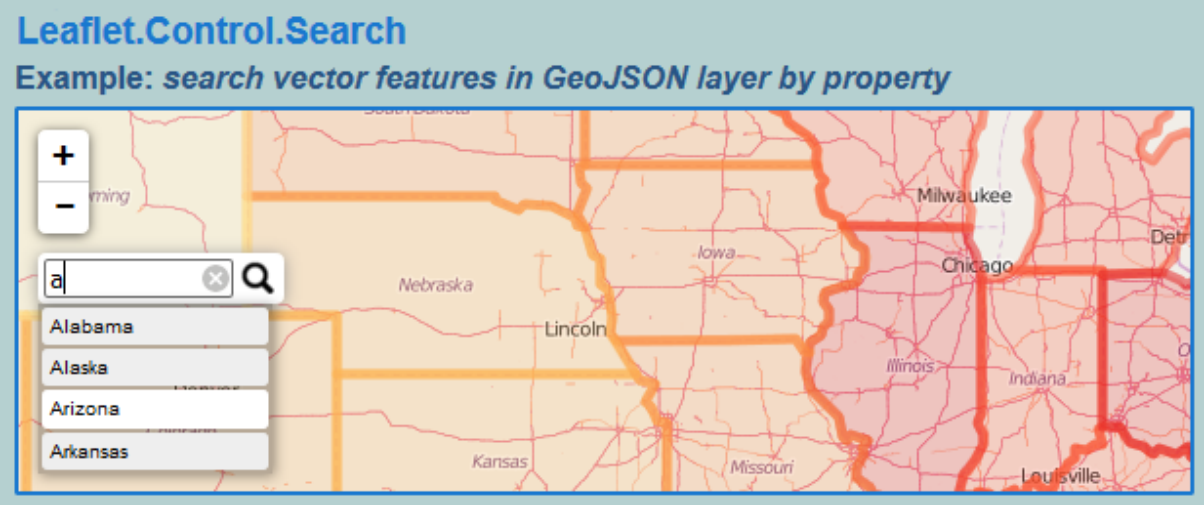
2.6.2 City Search

Ένα από τα πιο σημαντικά χαρακτηριστικά του χάρτη είναι η δυνατότητα του χρήστη να επιλέγει την πόλη και την κατηγορία των venues που τον ενδιαφέρουν. Υλοποιήθηκε, λοιπόν, η λειτουργία της αναζήτησης μιας πόλης γράφοντας το όνομά της σε ένα textfield με autocomplete και επιλέγοντας από μία λίστα διαθέσιμων πόλεων. Παράλληλα, υπάρχουν radio buttons για την επιλογή της κατηγορίας των venues που θα εμφανιστούν, τα οποία λειτουργούν σε συνδυασμό με το textfield για την πραγματοποίηση της αναζήτησης.

Η αναζήτηση υλοποιήθηκε με τη χρήση του plugin Leaflet.Control.Search [16]. Ο τρόπος λειτουργίας του μοιάζει με αυτόν για τη δημιουργία layers.

- Αρχικά, μέσω της jQuery μεθόδου `$.getJSON('/listCollections')` λαμβάνεται από τη βάση δεδομένων ένας πίνακας με τα ονόματα των collections που περιγράφουν πόλεις.
- Για κάθε στοιχείο του πίνακα, δημιουργείται ένα layer που περιέχει έναν marker με όνομα αυτό της πόλης. Ουσιαστικά, η λίστα του autocomplete στο textfield της αναζήτησης περιλαμβάνει τα ονόματα αυτών των markers.
- Τέλος, με παραμέτρους το όνομα της πόλης από το search textfield και το όνομα της κατηγορίας από το radio button, δημιουργείται το URL για την πραγματοποίηση του HTTP GET request και φορτώνεται με τη βοήθεια του router η νέα σελίδα, όπως περιγράφηκε στην ενότητα 2.5.2.

```
controlSearch.on('search_locationfound', function(e) {
    var cat = $('input[name="category"]:checked').val();
    location.href = 'http://localhost:3000/map/' +
this._input.value + '/' + cat;
});
```



Σχήμα 2.9: Παράδειγμα χρήσης του Leaflet.Control.Search plugin

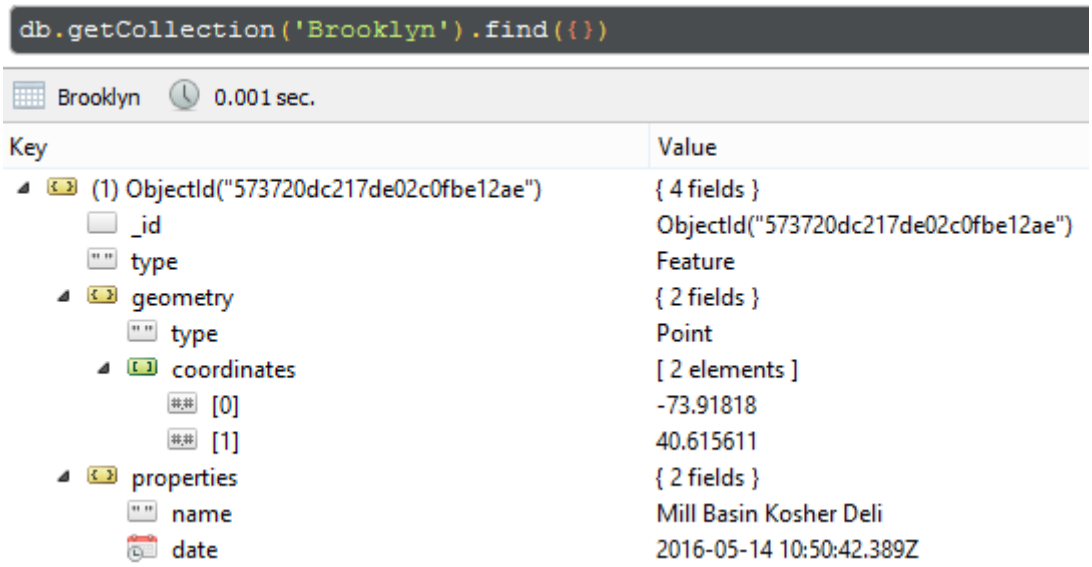
2.7 Σύνδεση με εξωτερική βάση δεδομένων

Η εφαρμογή έχει σχεδιαστεί με τέτοιο τρόπο, ώστε να επιτρέπει τη σύνδεση με μια άλλη βάση δεδομένων για την απεικόνιση των αποθηκευμένων πληροφοριών στο χάρτη. Για παράδειγμα, μπορεί να συνδεθεί με μια database με όνομα “fire”, όπου αποθηκεύονται σε μορφή GeoJSON οι τοποθεσίες όπου παρατηρήθηκαν πυρκαγιές και να εμφανίσει στο χάρτη την τοποθεσία και περαιτέρω πληροφορίες για κάθε γεγονός.

Η σύνδεση του server με τη βάση δεδομένων “fire” γίνεται ως εξής:

```
var conn2 =
mongoose.createConnection('mongodb://localhost:27017/fire', function
(error) {
    if (error) {
        console.log(error);
    }
});
```

Η βάση περιέχει συλλογές που η κάθε μία αντιπροσωπεύει μία πόλη, ενώ κάθε έγγραφο περιέχει πληροφορίες για την πυρκαγιά σε μία τοποθεσία σε GeoJSON format. Σε αυτό το παράδειγμα, κάθε τοποθεσία είναι ένα σημείο (Point) με συντεταγμένες (latitude, longitude) και περιλαμβάνει ένα όνομα και την ημερομηνία/ώρα που παρατηρήθηκε η πυρκαγιά.



Σχήμα 2.10: Ένα document στο collection ”Brooklyn” της database “fire”

Με τον ίδιο τρόπο που εμφανίζονται τα Foursquare venues σε μια περιοχή, σε αυτό το παράδειγμα εμφανίζονται markers στις τοποθεσίες που παρατηρήθηκε πυρκαγιά. Κάνοντας click σε μία περιοχή στο χάρτη, εκτελείται η εξής jQuery μέθοδος:

```
var str = '/fire/' + '#{city}' + '/' +
e.target.feature.properties.name;
$.getJSON(str,function(result){
```

```
    L.geoJson(result, {
        onEachFeature: onEachFeature2,
        pointToLayer: function (feature, latlng) {
            return L.marker(latlng);
        }
    }).addTo(map);
})
```

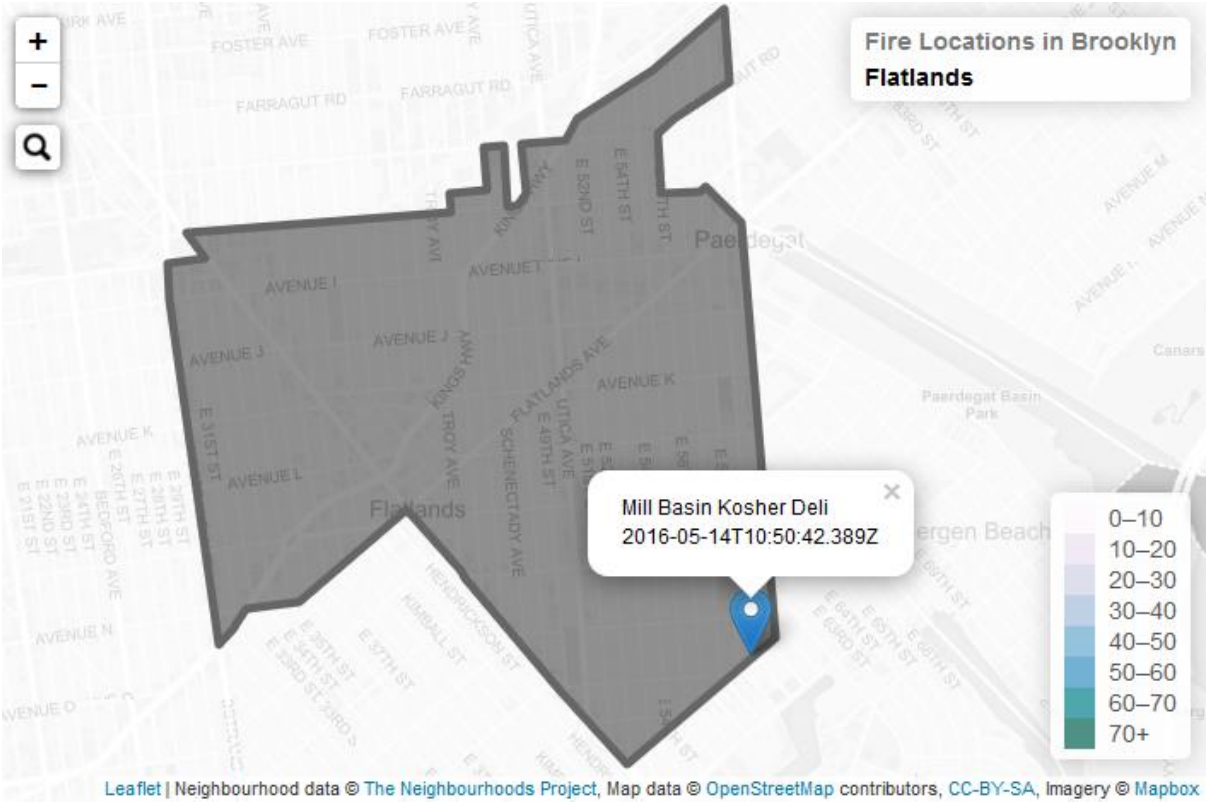
Μέσω του router του server, εκτελούνται ένα query στη βάση της εφαρμογής για την εύρεση των συντεταγμένων του πολυγώνου της περιοχής και ένα query στην fire database για την εύρεση των σημείων (πυρκαγιών) που περιέχονται στο πολύγωνο (γειτονιά).

```
router.get('/fire/:city/:hood', function(req,res) {
    var City = conn.model('City', CitySchema, req.params.city);
    City.findOne({"properties.name": req.params.hood},{"_id": 0},
function (err, docs) {
        var City2 = conn2.model('City', CitySchema,
req.params.city);
        City2.find({"geometry": {$geoWithin: {$geometry:
docs.geometry}}}, {"_id": 0}, function (err2, docs2) {
            res.json(docs2);
        });
    });
});
```

Ο router επιστρέφει JSON documents όπως το παρακάτω:

```
{
    "type" : "Feature",
    "geometry" : {
        "type" : "Point",
        "coordinates" : [
            -73.91818,
            40.615611
        ]
    },
    "properties" : {
        "name" : "Mill Basin Kosher Deli",
        "date" : ISODate("2016-05-14T10:50:42.389Z")
    }
}
```

Τα αποτελέσματα εμφανίζονται ως markers και κάνοντας click πάνω τους εμφανίζεται ένα popup με το όνομα της τοποθεσίας και την ημερομηνία/ώρα της φωτιάς:



Σχήμα 2.11: Παράδειγμα απεικόνισης της τοποθεσίας μιας πυργκαγιάς στο χάρτη

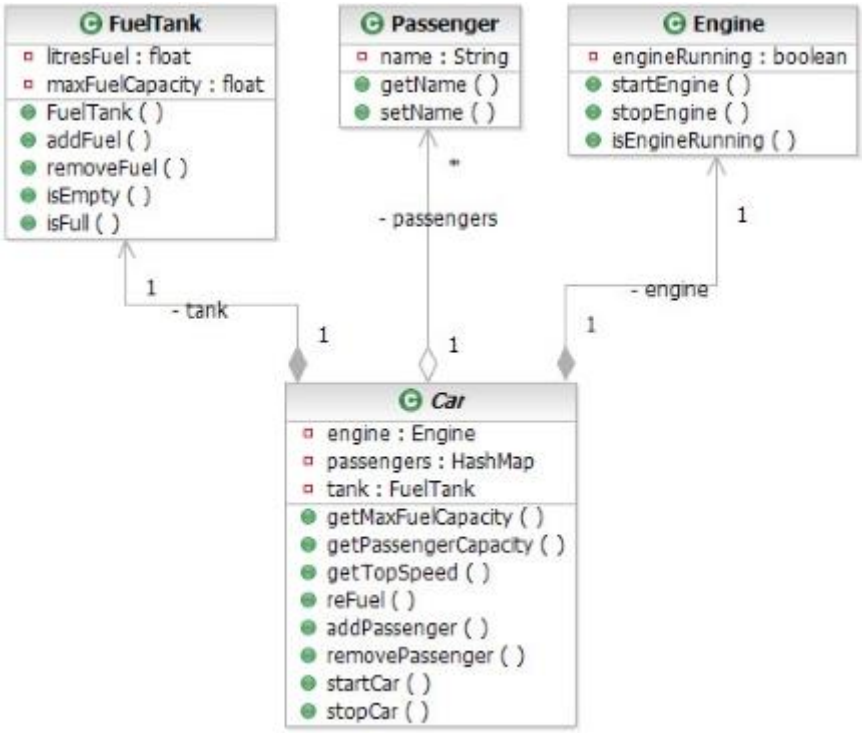
Κεφάλαιο 3
Εργαλεία Ανάπτυξης Λογισμικού

3.1 Εισαγωγή

Σε αυτό το κεφάλαιο περιγράφονται οι γλώσσες προγραμματισμού, οι πλατφόρμες ανάπτυξης λογισμικού και γενικότερα οι τεχνολογίες που χρησιμοποιήθηκαν για τον προγραμματισμό του κάθε μέρους της εφαρμογής. Επίσης, αναφέρονται οι λόγοι για τους οποίους επιλέχθηκαν ως καταλληλότερες σε σχέση με άλλα, παρόμοια εργαλεία. Παρουσιάζονται η γλώσσα προγραμματισμού Java, η NoSQL βάση δεδομένων MongoDB, η χρήση του Node.JS runtime environment για τη δημιουργία server και οι δυνατότητες του Mapbox ως mapping platform σε συνδυασμό με τη βιβλιοθήκη Leaflet για το styling του χάρτη.

3.2 Αντικειμενοστρεφής προγραμματισμός & Java

Η γλώσσα προγραμματισμού Java βασίζεται στο μοντέλο της αντικειμενοστρεφούς (object-oriented) ανάπτυξης λογισμικού [17]. Το μοντέλο αυτό αποσκοπεί στην οργάνωση του λογισμικού σε ένα σύνολο από αλληλεπιδρώντα αντικειμένα (objects). Η επιλογή αυτών των αντικειμένων υπαγορεύεται συνήθως από το φυσικό κόσμο και το πρόβλημα που πρόκειται να επιλυθεί. Κάθε αντικείμενο περιέχει κάποια δεδομένα, που χαρακτηρίζουν την κατάστασή του, και υλοποιεί κάποια συμπεριφορά. Αντικείμενα που έχουν παρόμοια μορφή μπορούν να οργανωθούν σε μία κλάση (class). Μια κλάση περιγράφει μια συλλογή δεδομένων και τις λειτουργίες που αυτά επιδέχονται, ενώ κάθε αντικείμενο δημιουργείται κατά τη διάρκεια εκτέλεσης ενός προγράμματος Java και υπάρχει μεχρι να καταστραφεί.



Σχήμα 3.1: Διάγραμμα κλάσεων μιας απλής Java εφαρμογής. Κάθε κλάση περιλαμβάνει ένα σύνολο μεθόδων, που επιτελούν κάποιες λειτουργίες

Η φιλοσοφία του αντικειμενοστρεφούς προγραμματισμού ακολουθήθηκε και στο σχεδιασμό της παρούσας εργασίας. Δημιουργήθηκε η κλάση AreaClass που περιλαμβάνει μεθόδους για την υλοποίηση των λειτουργιών του service που ανακτά τα γεωγραφικά δεδομένα από το API. Η κλάση VenueClass αφορά όλες τις λειτουργίες για την επικοινωνία με το Foursquare API, ενώ οι μέθοδοι της JsonClass καλούνται από τις υπόλοιπες για το parsing των JSON documents. Τέλος, η MongoClass χρησιμοποιεί τον Java Driver της MongoDB για την επικοινωνία με τη βάση δεδομένων.

Ένα άλλο χαρακτηστικό της Java είναι πως αποτελεί μία ερμηνευόμενη (interpreted) γλώσσα. Ο μεταγλωττιστής δεν παράγει τελικό κώδικα, για κάποιο συγκεκριμένο υπολογιστή, αλλά ενδιάμεσο κώδικα σε μορφή bytes, που ονομάζεται bytecode. Ο ενδιάμεσος κώδικας στη συνέχεια εκτελείται από ένα διερμηνέα (interpreter) της Java. Το πλεονέκτημα είναι ότι ο κώδικας μπορεί να εκτελεστεί σε πολλά διαφορετικά περιβάλλοντα υπολογιστών, αν φυσικά κάποιος διερμηνέας Java είναι διαθέσιμος σε αυτά. Ο διερμηνέας της γλώσσας μαζί με το σύστημα εκτέλεσης προγραμμάτων (run-time system) υλοποιεί μια εικονική μηχανή Java (Java Virtual Machine, JVM). Ο διερμηνέας έχει αναπτυχθεί για αρκετά περιβάλλοντα, συμβάλλοντας έτσι στη μεταφερσιμότητα του εκτελέσιμου κώδικα της γλώσσας. Προγράμματα Java μπορούν επίσης να εκτελεστούν από τους περισσότερους σύγχρονους web browsers, καθένας από τους οποίους έχει ενσωματωμένη μια εικονική μηχανή Java για την αναγνώριση και εκτέλεση του ενδιάμεσου κώδικα.

Στην εφαρμογή που υλοποιήθηκε, τα services που ανακτούν γεωγραφικά δεδομένα και Foursquare venues ώστε να τα αποθηκεύσουν στη βάση δεδομένων, έχουν πακεταριστεί σε εκτελέσιμα jar αρχεία, που περιλαμβάνουν τον πηγαίο κώδικα και τις βιβλιοθήκες που είναι απαραίτητες για τη λειτουργία τους. Έτσι, λοιπόν, μπορούν να εκτελεστούν σε οποιοδήποτε

υπολογιστή υπάρχει εγκατεστημένο το Java Runtime Environment και η MongoDB, ανεξάρτητα από τα υπόλοιπα τμήματα της εφαρμογής.

Τέλος, η Java υποστηρίζει πολλαπλά νήματα εκτέλεσης (multi-threading). Τα πολλαπλά νήματα εκτέλεσης αποτελούν ένα προγραμματιστικό μοντέλο για ανάπτυξη λογισμικού, που απαιτεί την "ταυτόχρονη" εκτέλεση πολλών διεργασιών. Οι διεργασίες μοιράζονται τον ίδιο χώρο μνήμης για την εκτέλεσή τους και τη διαχείριση δεδομένων. Τα πολλαπλά νήματα εκτέλεσης αποτελούν επίσης και μια αποδοτική λύση σε πολλά προβλήματα, αφού συνήθως η εναλλαγή εκτέλεσης ανάμεσα στα διάφορα νήματα δεν είναι ιδιαίτερα δαπανηρή. Εφαρμογές που χρησιμοποιούν πολλαπλά νήματα εκτέλεσης παρουσιάζουν βελτιωμένη απόκριση προς το χρήστη και, από προγραμματιστικής άποψης, είναι συνήθως ευκολότερες να αναπτυχθούν.

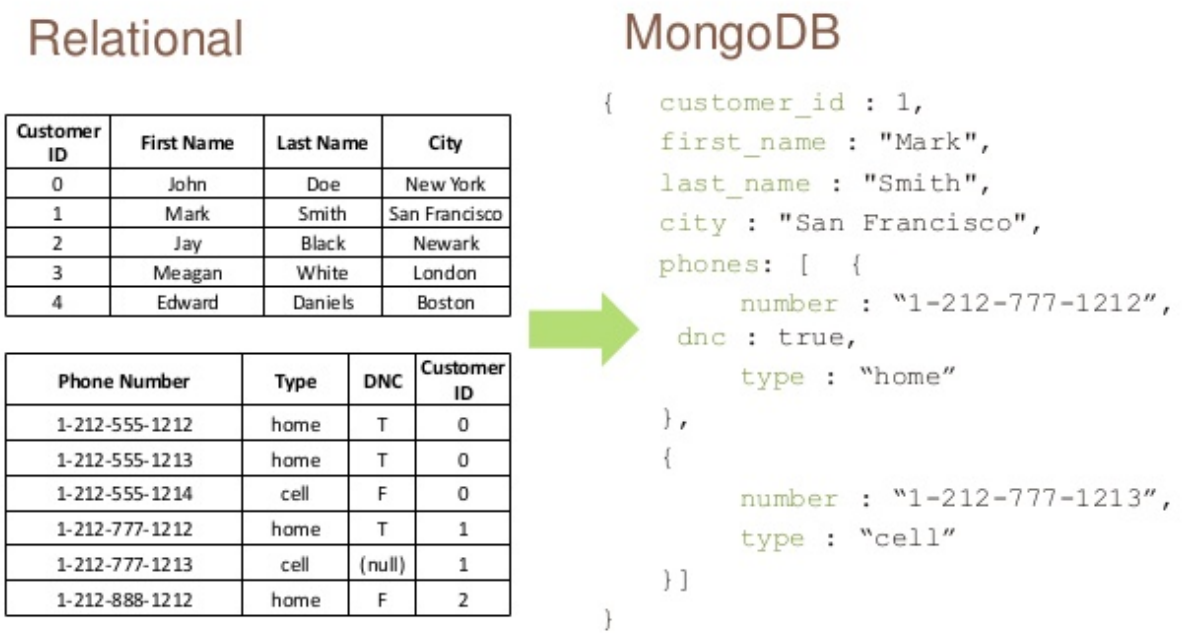
Στα services της συγκεκριμένης εφαρμογής δεν εκτελούνται πολλαπλά νήματα παράλληλα, διότι υπήρχαν μερικά προβλήματα συγχρονισμού των δεδομένων με τα τμήματα της εφαρμογής που βασίζονται σε ασύγχρονη επικοινωνία, που θα αναλυθεί στην ενότητα 3.4. Παρ' όλα αυτά, προσθέτοντας ένα ενδιάμεσο layer που θα κάνει συγχρονισμό των αποτελεσμάτων, μπορεί να αξιοποιηθεί το multi-threading της Java σε μελλοντικές εκδόσεις της εφαρμογής για τη βελτιώση της ταχύτητάς απόδοσής της.

3.3 NoSQL Βάσεις Δεδομένων & MongoDB

Η MongoDB είναι μία NoSQL βάση δεδομένων, που παρέχει ένα μηχανισμό προς αποθήκευση και ανάκτηση δεδομένων που είναι διαμορφωμένα με άλλο τρόπο από αυτό της μορφής πίνακα που χρησιμοποιείται στις σχεσιακές βάσεις δεδομένων.

Το σχεσιακό μοντέλο χωρίζει τα δεδομένα σε πολλούς αλληλένδετους πίνακες οι οποίοι αποτελούνται από γραμμές και στήλες. Κατά την αναζήτηση δεδομένων, οι επιθυμητές πληροφορίες πρέπει να συλλεχθούν από πολλούς πίνακες και να συνδυαστούν πριν να μπορούν να δοθούν στην εφαρμογή. Παρομοίως, κατά την εγγραφή δεδομένων, αυτή πρέπει να συντονιστεί και να γίνει σε πολλούς πίνακες.

Μια document-oriented NoSQL βάση δεδομένων, όπως η MongoDB, παίρνει τα προς αποθήκευση δεδομένα και τα συγκεντρώνει σε documents χρησιμοποιώντας τη μορφή JSON. Κάθε JSON document μπορεί να θεωρηθεί ως ένα αντικείμενο που θα χρησιμοποιηθεί από την εφαρμογή. Η ευελιξία του μοντέλου δεδομένων που προκύπτει, η ευκολία της αποδοτικής κατανομής των documents και οι βελτιώσεις απόδοσης των εγγραφών και αναγνώσεων την καθιστούν μια πολύ δημοφιλή επιλογή για τις εφαρμογές διαδικτύου.



Σχήμα 3.2: Σύγκριση του σχεσιασκού μοντέλου με αυτό της MongoDB

Για την εφαρμογή που αναπτύχθηκε, ζητούμενη ήταν μια ευέλικτη βάση δεδομένων, η οποία θα υποστηρίζει τους τύπους δεδομένων που χρησιμοποιούν οι περισσότερες υπηρεσίες στο διαδίκτυο. Τα δεδομένα που λαμβάνονται από τα APIs του Zetashapes και του Foursquare είναι σε JSON format. Η MongoDB αποθηκεύει τα δεδομένα ως έγγραφα σε δυαδική αναπαράσταση που ονομάζεται BSON (Binary JSON). Τα έγγραφα που τείνουν να έχουν παρόμοια δομή οργανώνονται ως συλλογές. Έτσι, λοιπόν, δε χρειάστηκε να αλλάξει η δομή των εγγράφων κατά την αποθήκευσή τους στη βάση δεδομένων, απλά προστέθηκαν επιπλέον πεδία.

Άλλο ένα πλεονέκτημα της MongoDB είναι ότι παρέχει εγγενείς οδηγούς για όλες τις δημοφιλείς γλώσσες προγραμματισμού και τα frameworks. Οι υποστηριζόμενοι οδηγοί περιλαμβάνουν Java, .NET, Ruby, PHP, JavaScript, Node.JS, Python, Perl, PHP, κλπ. Οι οδηγοί της MongoDB είναι σχεδιασμένοι να είναι ιδιωματικοί για την κάθε γλώσσα. Στην εργασία χρησιμοποιήθηκαν οι drivers της MongoDB για Java και Node.JS. Οι Java υπηρεσίες αποθηκεύουν στη βάση τα δεδομένα που λαμβάνουν από τα APIs και ο Node.JS server εκτελεί ερωτήματα στη βάση για την ανάκτηση των δεδομένων που θα εμφανιστούν στο χάρτη.

Τέλος, η MongoDB υποστηρίζει πολλούς τύπους ερωτημάτων. Ένα ερώτημα μπορεί να επιστρέψει ένα έγγραφο ή ένα υποσύνολο συγκεκριμένων πεδίων μέσα σε ένα έγγραφο. Οι τύποι ερωτημάτων που υποστηρίζονται από τη MongoDB είναι οι εξής:

- Τα ερωτήματα κλειδιού-ζεύγους επιστρέφουν αποτελέσματα βασισμένα σε οποιοδήποτε πεδίο εντός του εγγράφου.
- Τα ερωτήματα εύρους επιστρέφουν αποτελέσματα βασισμένα σε τιμές που ορίζονται ως ανισότητες (π.χ. μεγαλύτερο από, λιγότερο από ή ίσο με).
- Τα ερωτήματα αναζήτησης κειμένου επιστρέφουν αποτελέσματα βασισμένα σε ορίσματα κειμένου χρησιμοποιώντας λογικούς τελεστές (π.χ. AND, OR, NOT).

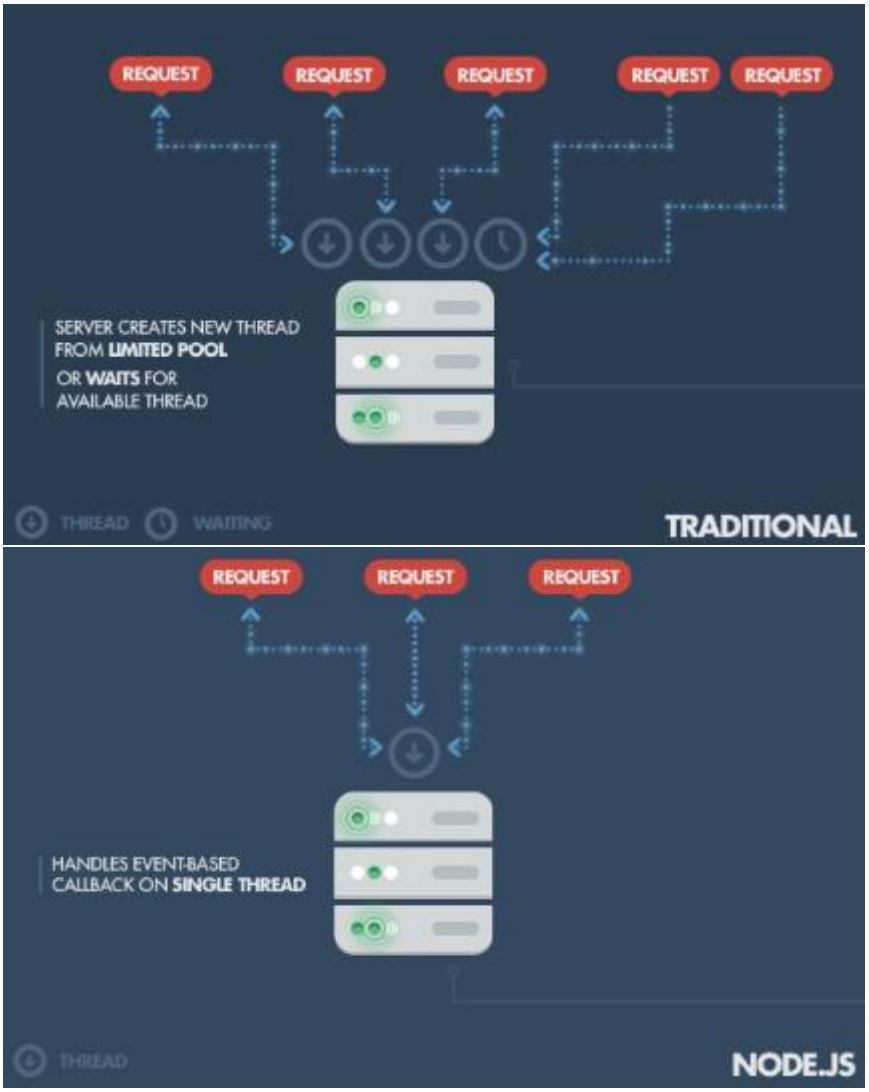
- Τα ερωτήματα Aggregation Framework επιστρέφουν αθροίσματα τιμών που επιστρέφονται από το ερώτημα (π.χ. count, min, max, average).
- Τα ερωτήματα MapReduce εκτελούν πολύπλοκες επεξεργασίες δεδομένων οι οποίες εκφράζονται σε JavaScript και εκτελούνται πάνω στα δεδομένα της βάσης.
- Τα γεωχωρικά ερωτήματα επιστρέφουν αποτελέσματα βασισμένα σε κριτήρια εγγύτητας, τομής και ένωσης όπως προσδιορίζονται από ένα σημείο, γραμμή, κύκλο ή πολύγωνο.

Η υποστήριξη γεωχωρικών ερωτημάτων (geospatial queries) αποδείκτηκε ιδιαίτερα χρήσιμη στην εφαρμογή που αναπτύχθηκε. Όπως παρουσιάστηκε στην ενότητα 2.4.4, μια βασική λειτουργία αποτελεί η εύρεση όλων των venues (σημεία) μέσα σε μία γειτονιά (πολύγωνο) και εκτελείται εύκολα και γρήγορα με τη χρήση των τελεστών $geometry και $geoWithin της MongoDB.

3.4 Node.JS & RESTful API

Το Node.JS είναι ένα interface για το V8 JavaScript runtime engine, το διερμηνέα της Javascript που χρησιμοποιείται στον Chrome web browser και χρησιμοποιείται για την εκτέλεση κώδικα σε Javascript σε έναν web server. Για τις βασικές λειτουργίες των εφαρμογών, όπως διάβασμα αρχείων και HTTP requests, το Node.JS διαθέτει ένα πλούσιο σετ βιβλιοθηκών με κομψή και απλή διεπαφή. Για άλλες λειτουργίες, η κοινότητα των χρηστών του Node.JS έχει δημιουργήσει ένα ολόκληρο οικοσύστημα από βιβλιοθήκες, που εγκαθιστώνται εύκολα με τη χρήση του Node Package Manager (NPM).

Σε έναν τυπικό LAMP server (Linux operating system, Apache HTTP Server, MySQL relational database management system, PHP programming language), κάθε νέα σύνδεση με το διακομιστή γεννά ένα νέο νήμα εκτέλεσης και ο μόνος τρόπος για να υποστηρίξει περισσότερους χρήστες είναι η προσθήκη περισσότερων servers. Το Node.JS δε χρησιμοποιεί πολυνηματικότητα (multi-threading) για τη διαχείριση πολλών συνδέσεων [18], καθώς στηρίζεται σε ένα νήμα εκτέλεσης με δυνατότητα ασύγχρονης εισόδου/εξόδου (non-blocking I/O). Με αυτό τον τρόπο, κάθε αίτημα πελάτη δε μπλοκάρει το νήμα και δεν είναι απαραίτητη η χρήση πολλών νημάτων για υποστήριξη πολλών χρηστών. Ο τρόπος που το Node.JS επιτυγχάνει ασύγχρονη είσοδο/έξοδο είναι μέσω των callback functions. Για κάθε λειτουργία ορίζεται μια callback function, η οποία θα κληθεί όταν τελειώσει η εν λόγω λειτουργία. Έτσι, το σύστημα εκκινεί μία λειτουργία, συνεχίζει στις επόμενες κι όταν η διαδικασία αυτή ολοκληρωθεί καλεί την callback function.



Σχήμα 3.3: Διαφορές μεταξύ Multi-threading και Ασύγχρονης επικοινωνίας

Ένα ακόμη πλεονέκτημα του Node.JS είναι η ευκολία επικοινωνίας με object-oriented βάσεις δεδομένων, όπως η MongoDB. Δεδομένα που είναι αποθηκευμένα σε μορφή JSON δε χρειάζονται κάποια μετατροπή για να διαβαστούν στο Node.JS και στη Javascript είναι πολύ εύκολος ο χειρισμός των JSON objects. Συνεπώς, αποφεύγονται οι μετατροπές χρησιμοποιώντας ένα κοινό format στον client, στον server και στη βάση δεδομένων, κάτι που έγινε και στην παρούσα εφαρμογή.

Το Node.JS χρησιμοποιήθηκε στην εφαρμογή για τη δημιουργία ενός RESTful API [19], που παίζει το ρόλο ενός ενδιάμεσου layer για τη μεταφορά δεδομένων μεταξύ χάρτη και βάσης δεδομένων. Ένα RESTful API πρέπει να υπακούει στους κανόνες της REST (Representational State Transfer) αρχιτεκτονικής, που είναι οι εξής:

- **Client-Server**: Πρέπει να υπάρχει διάκριση μεταξύ του server και του client. Οι clients δεν επιβαρύνονται με την αποθήκευση δεδομένων, ενώ οι servers δεν επιβαρύνονται από το user interface. Έτσι, οι servers και οι clients μπορούν να αναπτυχθούν και να αντικατασταθούν ανεξάρτητα με την προϋπόθεση ότι η διεπαφή μεταξύ τους δεν αλλάζει.

- **Stateless**: Ο server δεν αποθηκεύει στοιχεία για την κατάσταση του κάθε client. Αυτό συνεπάγεται ότι ο client πρέπει να κρατάει όλα τα στοιχεία που του είναι απαραίτητα για να ορίζει την παρούσα κατάστασή του και να τα στέλνει στον server, μαζί με κάθε request. Επίσης, θα πρέπει να τα αλλάζει ανάλογα με το response του server.
- **Cacheable**: Ο server πρέπει να υποδεικνύει στον client αν τα requests αποθηκεύονται στην cache ή όχι. Τα δεδομένα που χαρακτηρίζονται ως cacheable μπορούν να χρησιμοποιηθούν ως response σε ίδιο διαδοχικό request. Με αυτόν τον τρόπο αποφεύγονται κάποιες αλληλεπιδράσεις client-server, βελτιώνοντας τις επιδόσεις του συστήματος.
- **Layered System**: Η επικοινωνία μεταξύ ενός client και ενός server πρέπει να είναι τυποποιημένη με τέτοιο τρόπο, ώστε να επιτρέπει σε μεσάζοντες να απαντούν στα requests αντί του κεντρικού server χωρίς αυτό να γίνεται αντιληπτό από τον client.
- **Uniform Interface**: Τα resources αποστέλλονται σε μορφή HTML, JSON ή XML και κάθε μήνυμα περιέχει πληροφορίες για τον τρόπο επεξεργασίας τους από τον client.

Λαμβάνοντας υπόψιν τα παραπάνω, υλοποιήθηκε ένα web service που δέχεται HTTP GET requests σε μορφή URL, πχ http://localhost:3000/map/Brooklyn/Food, ανάλογα με τις ενέργειες του χρήστη πάνω στο χάρτη. Το Node.JS συνδέεται με τη MongoDB, όπου εκτελούνται queries για την ανάκτηση των κατάλληλων δεδομένων. Τέλος, το RESTful επιστρέφει τα κατάλληλα δεδομένα σε JSON format για την εμφάνισή τους στο χάρτη. Περισσότερες λεπτομέρειες της υλοποίησης παρουσιάζονται στην ενότητα 2.5.

3.5 Mapbox & Leaflet

Το Mapbox είναι ένας πάροχος online χαρτών με πολλές δυνατότητες για customization και χρησιμοποιείται σε πολλές δημοφιλείς εφαρμογές, όπως Foursquare, Pinterest, National Geographic, Roadtrippers, Evernote, Financial Times και Etsy. Επίσης, έχει συνεισφέρει σημαντικά στην ανάπτυξη πολλών open source βιβλιοθηκών και εφαρμογών, όπως MBTiles file format, TileMill cartography IDE, Leaflet JavaScript library και CartoCSS map styling language and parser.

Τα δεδομένα λαμβάνονται τόσο από open data πηγές, όπως OpenStreetMap και NASA, όσο και από proprietary data πηγές, όπως DigitalScope. Η τεχνολογία του βασίζεται σε Node.JS, CouchDB, Mapnik, GDAL και Leaflet. Επιπλέον, το Mapbox χρησιμοποιεί δεδομένα από τη δραστηριότητα των χρηστών που το χρησιμοποιούν, πχ από τις εφαρμογές Strava και Runkeeper, ώστε να αναγνωρίζει με αυτόματες μεθόδους ελλιπή δεδομένα και να τα διορθώνει ή να τα αναφέρει στους συντελεστές των OpenStreetMaps.

Το Mapbox διαθέτει software development kits (SDKs) για web και mobile εφαρμογές σε διάφορες γλώσσες προγραμματισμού, όπως Javascript και Python. Επίσης, προσφέρει τη δυνατότητα επικοινωνίας με πολλά REST APIs για directions, geocoding, uploads, distance, images, map matching, styles, κλπ.

Το γεγονός ότι τα δεδομένα του είναι κυρίως από open source πηγές, η συνεχής ανανέωση των γεωγραφικών πληροφοριών και οι αμέτρητες δυνατότητες για την παραμετροποίηση της εμφάνισης των χαρτών αποτέλεσαν τους βασικούς λόγους για τους οποίους επιλέχθηκε το Mapbox στην υλοποίηση της εφαρμογής. Από το Mapbox API λαμβάνεται το base layer του

χάρτη, στο οποίο προσθέτονται επιπλέον λειτουργίες με τη βοήθεια της Javascript βιβλιοθήκης Leaflet.

Τα χαρακτηριστικά της βιβλιοθήκης Leaflet που συντέλεσαν στην επιλογή της είναι η απλότητα, το μικρό μέγεθος, το καλό documentation, τα πολλά tutorials και η υποστήριξη πολλών μεθόδων για την εμφάνιση του χάρτη και των λειτουργιών της διεπαφής χρήστη.

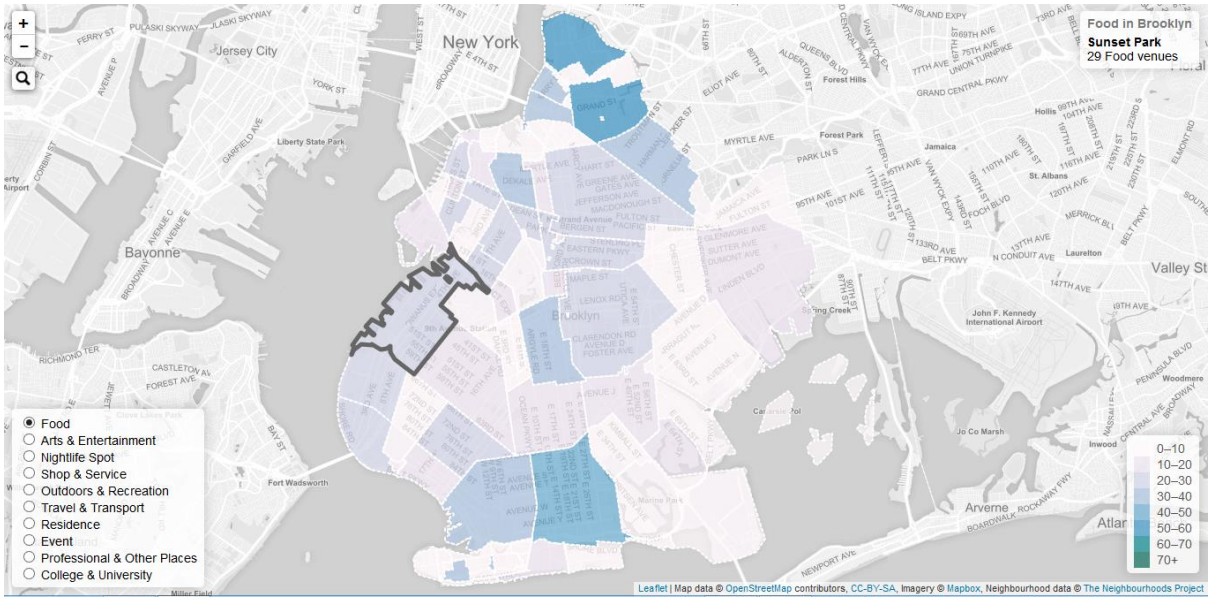
Η Leaflet είναι γραμμένη σε Javascript και υποστηρίζει HTML5 και CSS3. Επιπλέον, δέχεται γεωγραφικά δεδομένα σε GeoJSON format, όπως δηλαδή είναι αποθηκευμένα στη MongoDB της εφαρμογής. Περισσότερες λεπτομέρειες για τη λειτουργία της περιγράφονται στην ενότητα 2.6, ενώ στο κεφάλαιο 4 παρουσιάζονται screenshots με την εμφάνιση των χαρτών.

Σχήμα 3.4: Παράδειγμα χάρτη που χρησιμοποιεί Leaflet

Κεφάλαιο 4
Screenshots Εφαρμογής

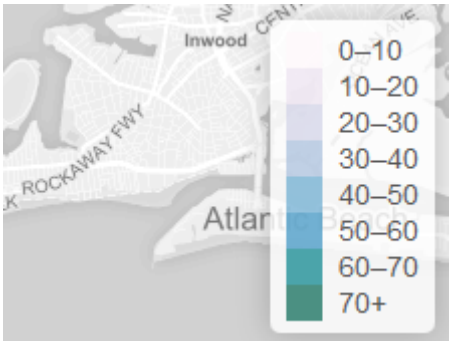
4.1 Απεικόνιση Περιοχών Πόλης

Στην παρακάτω εικόνα φαίνεται ο χάρτης κεντραρισμένος αυτόματα, ώστε να περιλαμβάνει ολόκληρο το Brooklyn, NY χωρισμένο σε περιοχές, όπου η κάθε μία είναι χρωματισμένη ανάλογα με τον αριθμό των Food venues που βρέθηκαν σε αυτή. Το base layer επιλέχθηκε grayscale (mapbox light) για να είναι πιο ευδιάκριτα τα χρώματα των περιοχών.



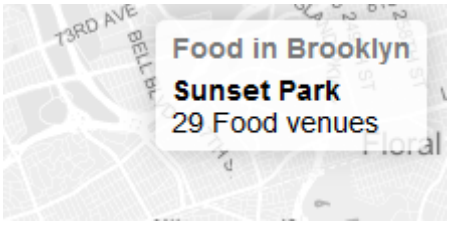
Σχήμα 4.1: Χάρτης του Brooklyn χωρισμένο σε γειτονιές

Κάτω δεξιά υπάρχει ένα υπόμνημα με τα διαστήματα των τιμών που αντιστοιχούν σε κάθε χρώμα. Τα διαστήματα είναι σταθερά και επιλέχθηκαν ως αντιπροσωπευτικά βάσει του πλήθους των venues ανά περιοχή στη βάση δεδομένων, ενώ τα χρώματα επιλέχθηκαν από το colorbrewer2.org, το οποίο περιέχει colour schemes ειδικά για χαρτογραφία.



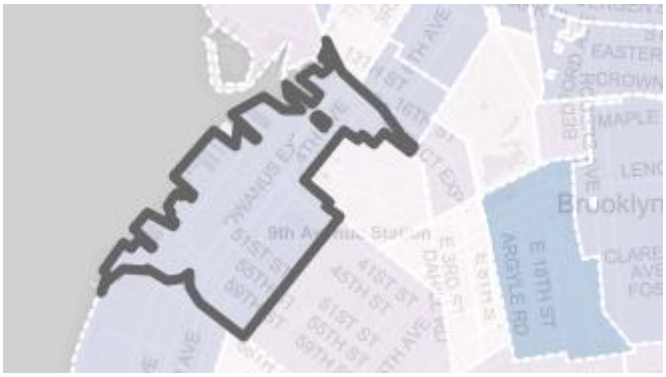
Σχήμα 4.2: Το υπόμνημα του χάρτη

Πάνω δεξιά βρίσκεται ένα panel, στο οποίο αναγράφονται το όνομα της κατηγορίας των venues και το όνομα της πόλης που απεικονίζεται, ενώ ανάλογα με τη θέση του δείκτη του ποντικιού του χρήστη ανανεώνεται το περιεχόμενο με το όνομα της περιοχής και τον αριθμό των venues σε αυτή:



Σχήμα 4.3: Το panel απεικόνισης πληροφοριών κάθε γειτονιάς

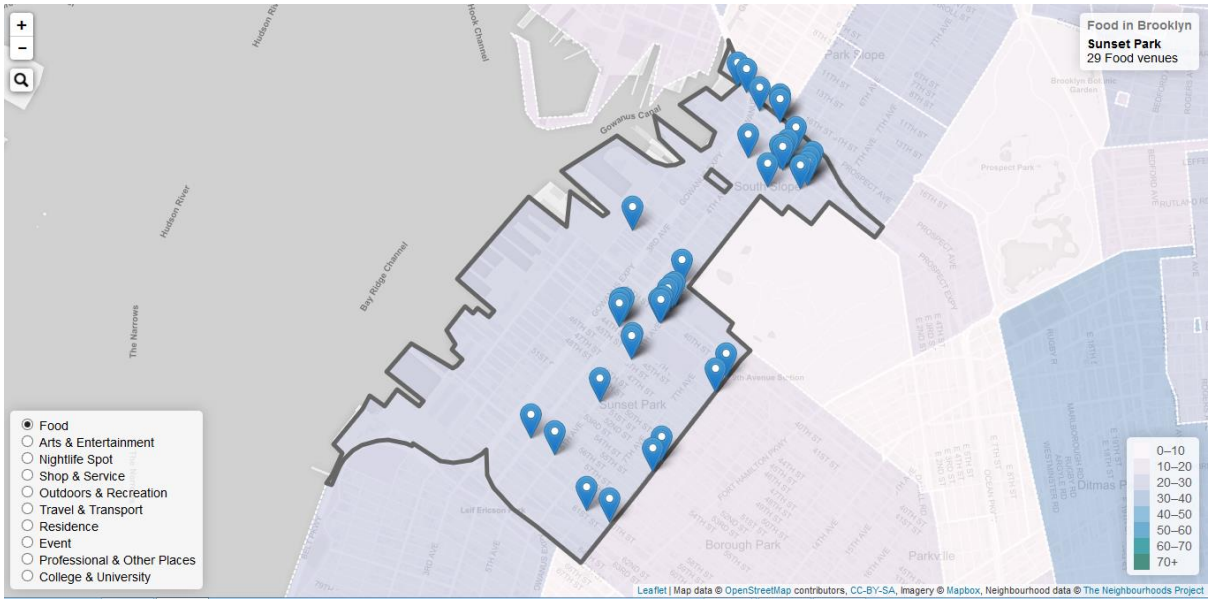
Παράλληλα, περνώντας το δείκτη του ποντικιού πάνω από ένα πολύγωνο, γίνεται εντονότερο το περίγραμμά του:



Σχήμα 4.4: Το περίγραμμα της επιλεγμένης γειτονιάς

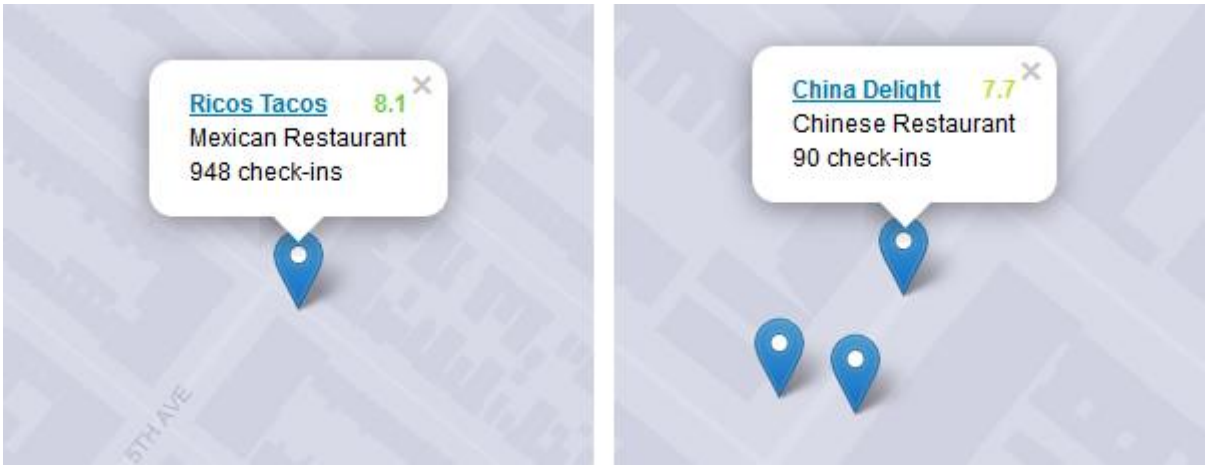
4.2 Απεικόνιση Venues Περιοχής

Κάνοντας click σε ένα πολύγωνο, γίνεται αυτόματη μεγέθυνση, ώστε να απεικονίζεται ολόκληρη η γειτονιά στην οθόνη. Επίσης, εμφανίζονται markers στις τοποθεσίες όλων των venues μέσα στην περιοχή.



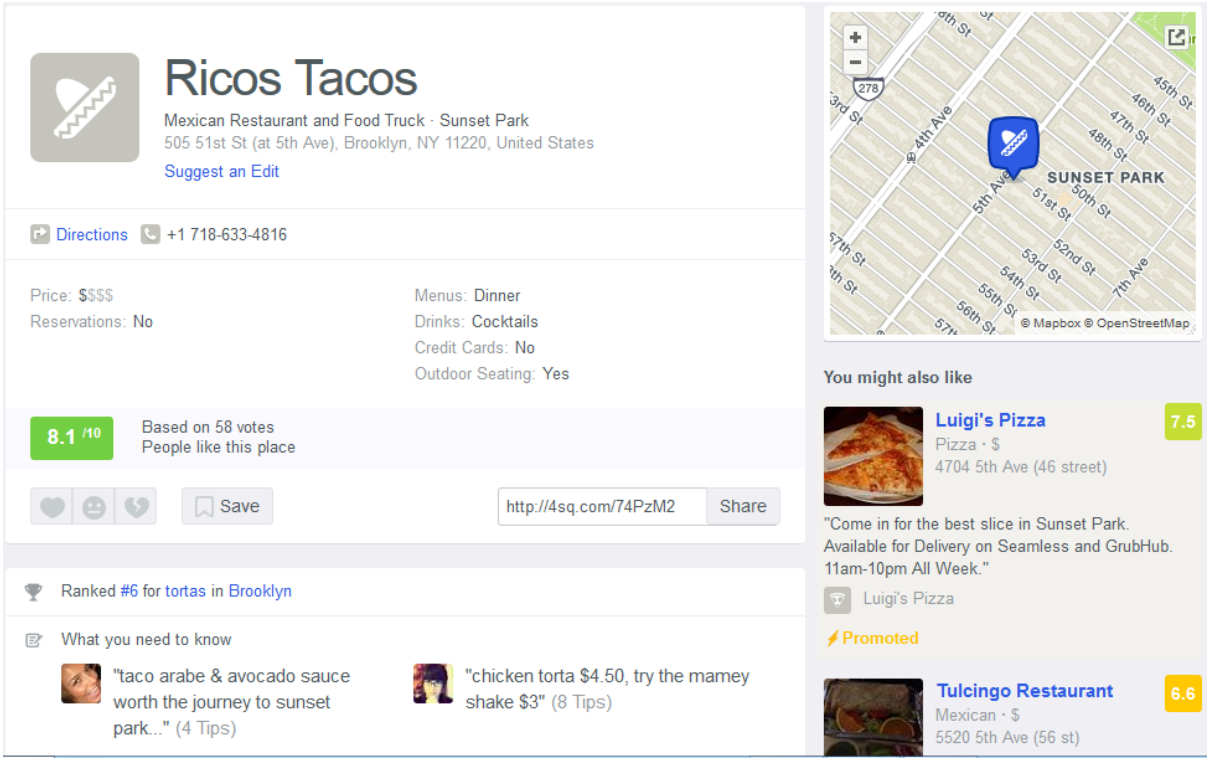
Σχήμα 4.5: Χάρτης μιας περιοχής με τα venues που βρέθηκαν σε αυτή

Κάνοντας click σε έναν marker, εμφανίζεται ένα popup με βασικές πληροφορίες που αφορούν το venue: όνομα, βαθμολογία επισκεπτών, κατηγορία και αριθμός των συνολικών επισκεπτών (check-ins). Το χρώμα του αριθμού της βαθμολογίας μεταβάλλεται ανάλογα με την τιμή της, από πράσινο για υψηλή βαθμολογία γίνεται πιο κίτρινο, μέχρι κόκκινο για χαμηλή.



Σχήμα 4.6: Popups με πληροφορίες για κάθε venue. Το εντονότερο πράσινο υποδεικνύει υψηλότερη βαθμολογία.

Κάνοντας click στο όνομα του venue, ανοίγει η αντίστοιχη σελίδα στο Foursquare, με περισσότερες πληροφορίες και σχόλια χρηστών:



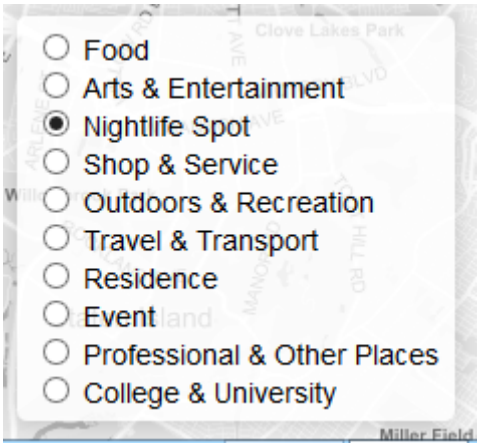
Σχήμα 4.7: Η σελίδα ενός venue στο Foursquare

4.3 Αναζήτηση Πόλης

Πάνω αριστερά υπάρχουν τα κουμπιά για zoom in/out στο χάρτη και κάτω από αυτά ένα κουμπί για την αναζήτηση. Πατώντας το κουμπί εμφανίζεται ένα πεδίο κειμένου για τον όρο της αναζήτησης, ενώ πληκτρολογώντας εμφανίζονται αυτόματα προτάσεις έγκυρων αναζητήσεων, βάσει των πόλεων που υπάρχουν ήδη στη βάση δεδομένων:

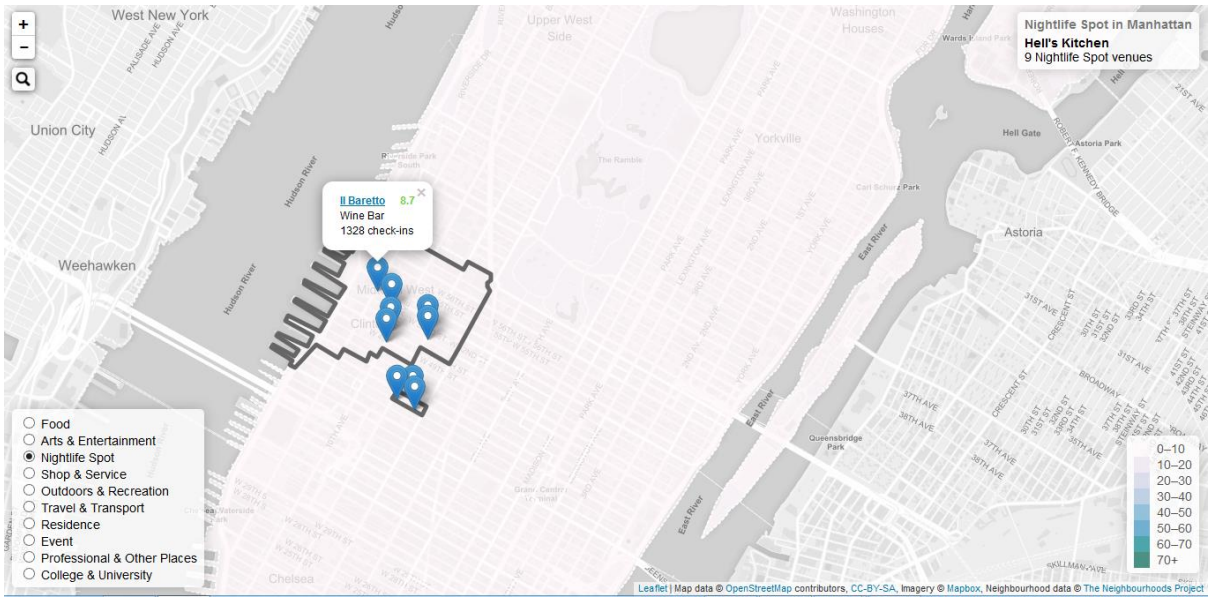
Σχήμα 4.8: Το πεδίο κειμένου αναζήτησης με auto-complete

Σε συνδυασμό με το πεδίο αναζήτησης ονόματος πόλης, λειτουργεί ένα μενού επιλογής της κατηγορίας των venues που θα εμφανιστούν. Το μενού βρίσκεται κάτω αριστερά και περιέχει radio buttons, όπου μόνο ένα είναι δυνατόν να επιλεγεί κάθε φορά.



Σχήμα 4.9: Τα radio buttons επιλογής μίας κατηγορίας venue

Επιλέγοντας κατηγορία με το κατάλληλο radio button και κάνοντας click σε μία πρόταση του πεδίου αναζήτησης, ανανεώνεται η σελίδα με το χάρτη της νέας πόλης και ακολουθείται ομοίως η παραπάνω διαδικασία για την απεικόνιση των νέων venues:



Σχήμα 4.10: Απεικόνιση των venues μιας γειτονιάς στο Manhattan

Κεφάλαιο 5
Μελέτη Περιπτώσεων

5.1 Ανάλυση Προβλήματος

Όπως αναφέρθηκε και σε προηγούμενα κεφάλαια, σκοπός της διπλωματικής εργασίας ήταν η ανάπτυξη ενός συστήματος για την απεικόνιση σε χάρτη μιας πόλης των ΗΠΑ χωρισμένη σε επιμέρους περιοχές, όπου η κάθε μία είναι χρωματισμένη ανάλογα με τον αριθμό των Foursquare venues που βρέθηκαν σε αυτή. Επιπλέον, δίνεται η δυνατότητα παρουσίασης των βασικών πληροφοριών για κάθε venue, όπως όνομα, κατηγορία, βαθμολογία και συνολικός αριθμός επισκεπτών.

Στην εφαρμογή χρησιμοποιήθηκε το Foursquare ως πηγή δεδομένων για τις εκδηλώσεις, επιχειρήσεις και τοποθεσίες που υπάρχουν σε μία πόλη, διότι περιλαμβάνει μια εκτενή συλλογή από venues για όλες τις πόλεις του κόσμου με την ακριβή τοποθεσία τους και πρόσθετες πληροφορίες, όπως κατηγορία, ωράριο, menu, επικοινωνία, κλπ. Επιπλέον, τα δεδομένα αυτά ανανεώνονται και εμπλουτίζονται συνεχώς από τους χρήστες της εφαρμογής με σχόλια και βαθμολογία, βάσει των εντυπώσεών τους από την επίσκεψή τους.

Στις περισσότερες εφαρμογές που προσφέρουν πληροφορίες για τοποθεσίες και εκδηλώσεις, ο αριθμός των αποτελεσμάτων μίας αναζήτησης έχει ένα άνω όριο, λόγω των περιορισμών που θέτει το αντίστοιχο API για τον όγκο των δεδομένων. Για παράδειγμα, μία αναζήτηση στο Foursquare για εστιατόρια στο Manhattan θα επιστρέψει μόνο τα 30 δημοφιλέστερα εστιατόρια σε όλο το Manhattan, διότι το Foursquare API επιστρέφει by default 30 αποτελέσματα για κάθε request. Η εφαρμογή της διπλωματικής δημιουργήθηκε με στόχο την εύρεση όσο το δυνατόν περισσότερων venues σε μία πόλη και τη δημιουργία μιας πληρέστερης εικόνας για την κατανομή τους σε κάθε περιοχή. Έτσι, λοιπόν, γίνεται αναζήτηση των venues για κάθε επιμέρους γειτονιά μιας πόλης με άνω όριο 50 αποτελεσμάτων για την κάθε μία (το ανώτερο που επιτρέπει το API) και αποθηκεύονται σε συλλογές ανάλογα με την κατηγορία τους, πχ φαγητό, διασκέδαση, κλπ. Επιπρόσθετα, ο τρόπος εμφάνισης των αποτελεσμάτων με χρωματισμό της κάθε περιοχής επιτρέπει την ευκολότερη εξαγωγή συμπερασμάτων για την κατανομή ανά περιοχή των venues της κάθε κατηγορίας.

Στις επόμενες ενότητες θα αναλυθούν οι διαφορές της εφαρμογής που υλοποιήθηκε με τις εφαρμογές του TripAdvisor και του Foursquare. Επιπλέον, μελετάται η επίδραση της επικαιρότητας στο σύστημα, δηλαδή η εξάρτηση των αποτελεσμάτων από την ώρα της αναζήτησης και η δυνατότητα χρήσης της εφαρμογής για τη μελέτη της δημοτικότητας των venues.

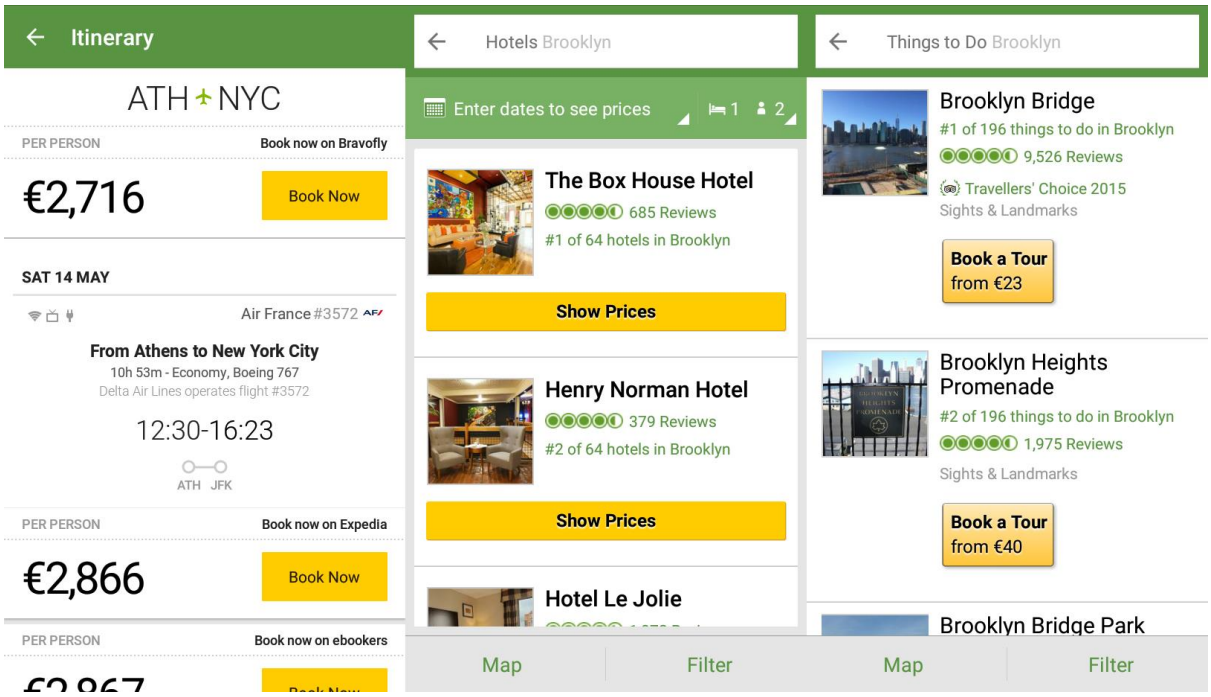
5.2 Σύγκριση εφαρμογής με άλλες εμπορικές

Σε αυτή την ενότητα παρουσιάζονται οι λειτουργίες που προσφέρουν οι εφαρμογές Tripadvisor και Foursquare και γίνεται η σύγκρισή τους με την εφαρμογή της διπλωματικής για την αναζήτηση εστιατορίων στο Brooklyn.

5.2.1 Tripadvisor

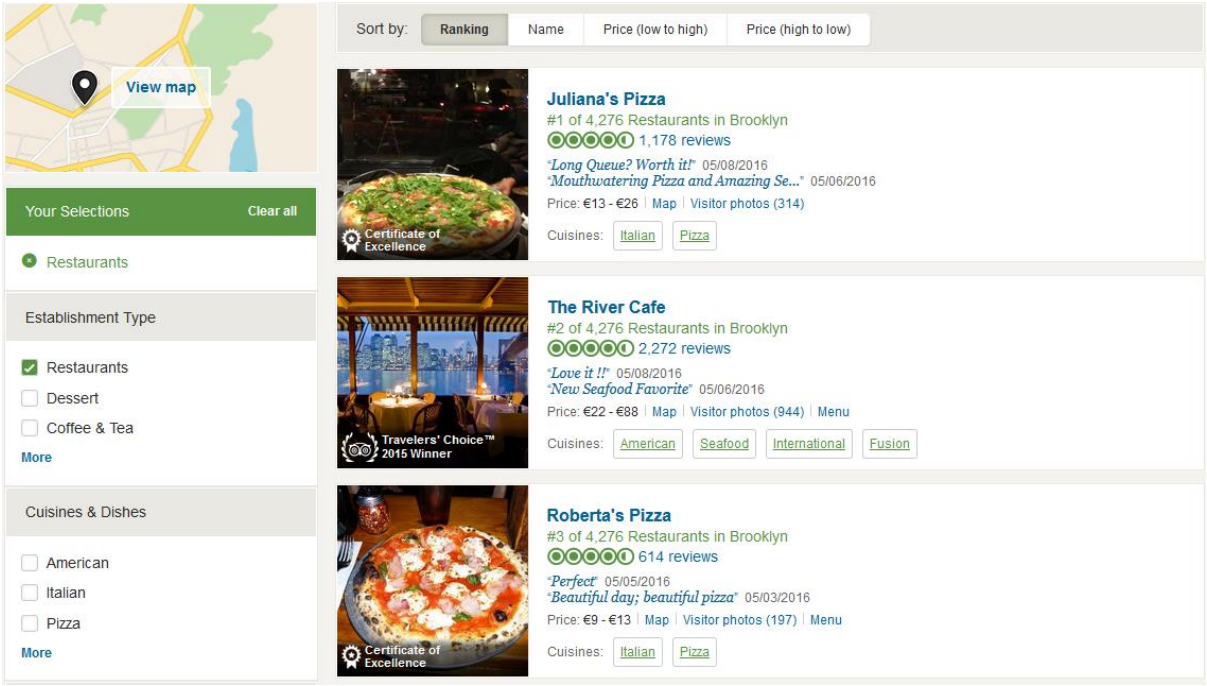
Το TripAdvisor [20] είναι μία εφαρμογή που παρέχει πληροφορίες για ταξιδιωτικούς προορισμούς, ξενοδοχεία, εστιατόρια, πτήσεις, αξιοθέατα και δραστηριότητες, είναι δηλαδή ένας διαδικτυακός τουριστικός οδηγός. Το περιεχόμενό του βασίζεται σε κριτικές χρηστών (user-generated content) που έχουν επισκεφτεί τους αντίστοιχους προορισμούς κι έχουν προσθέσει σχόλια με τις εντυπώσεις τους και βαθμολογία με άριστα το 5.

Μέσω της εφαρμογής παρέχεται η δυνατότητα στους χρήστες να επιλέξουν ένα προορισμό και να αγοράσουν αεροπορικά εισιτήρια. Επιπλέον, περιλαμβάνονται πολλά ξενοδοχεία χωρισμένα σε κατηγορίες με πλούσιο φωτογραφικό υλικό και κριτικές, ενώ υποστηρίζεται κι η λειτουργία για booking ενός δωματίου. Επίσης, πολύ χρήσιμη είναι η κατηγορία “Things To Do” που παρουσιάζει δημοφιλείς δραστηριότητες στην πόλη προορισμού, όπως αξιοθέατα, μουσεία, πάρκα, μαγαζιά, κλπ.



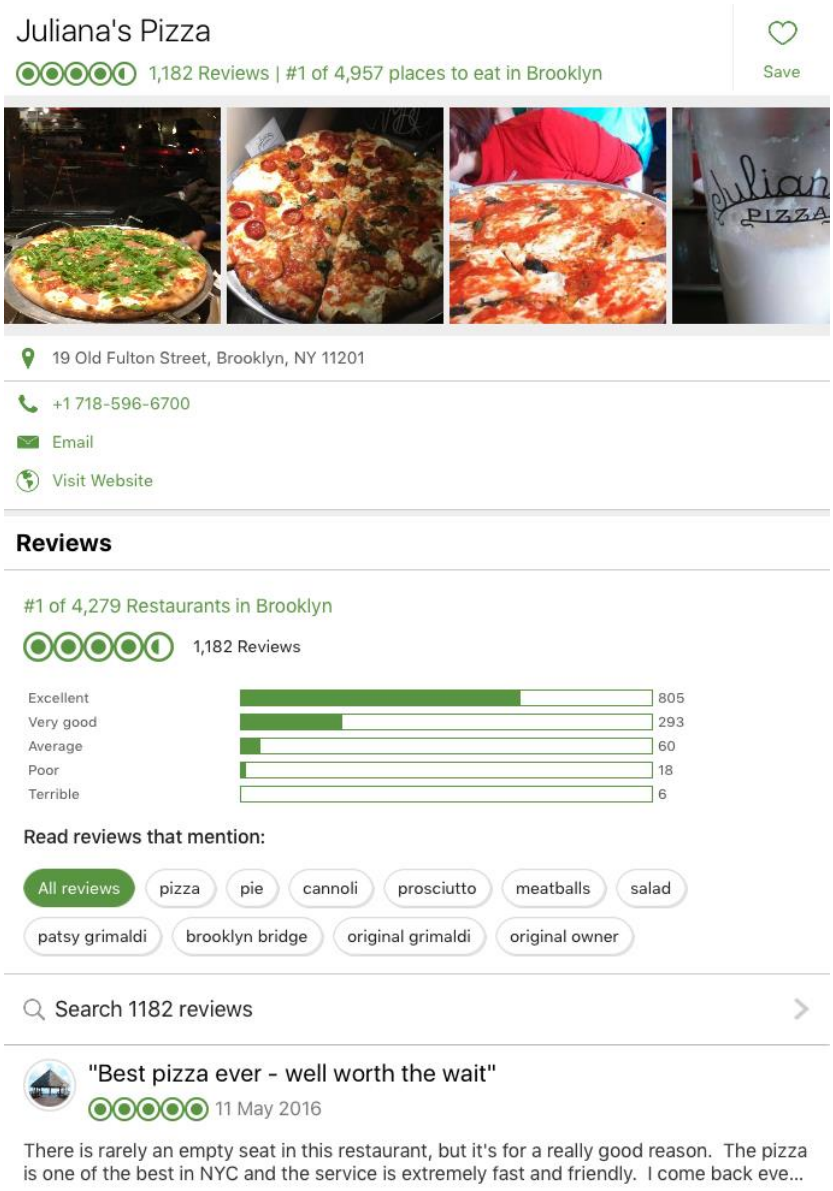
Σχήμα 5.1: Τρία στιγμιότυπα από το Android app του Tripadvisor για αναζήτηση πτήσεων, ξενοδοχείων και δραστηριοτήτων στο Brooklyn

Στα πλαίσια της εργασίας δε θα εξεταστούν αναλυτικά όλες οι λειτουργίες του TripAdvisor, παρά μόνο η αναζήτηση εστιατορίων στο Brooklyn, ώστε να συγκριθούν τα χαρακτηριστικά του με αυτά της εφαρμογής της διπλωματικής. Η αναζήτηση επέστρεψε 4279 εστιατόρια, συμπεριλαμβανομένων όμως και πολλών που βρίσκονται στο Manhattan, που μπορούν να ταξινομηθούν ανά βαθμολογία, όνομα ή τιμή και να φιλτραριστούν βάσει τύπου κουζίνας, γεύματος, κλπ.



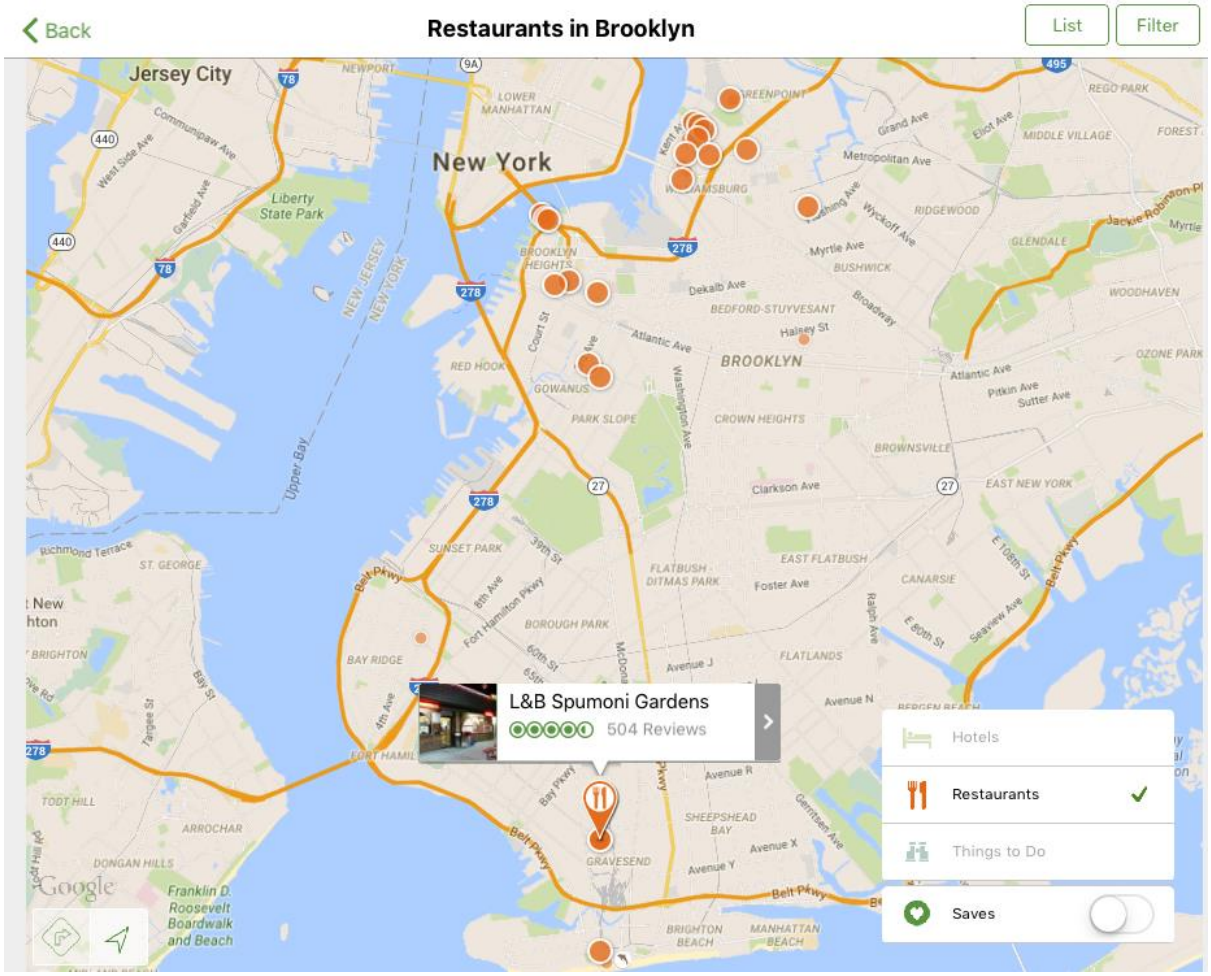
Σχήμα 5.2: Αναζήτηση εστιατορίων στη web εφαρμογή του Tripadvisor

Στη σελίδα ενός εστιατορίου εμφανίζονται πληροφορίες, όπως φωτογραφίες, διεύθυνση, επικοινωνία και κριτικές χρηστών, με τη δυνατότητα φιλτραρίσματός τους βάσει κάποιων keywords.



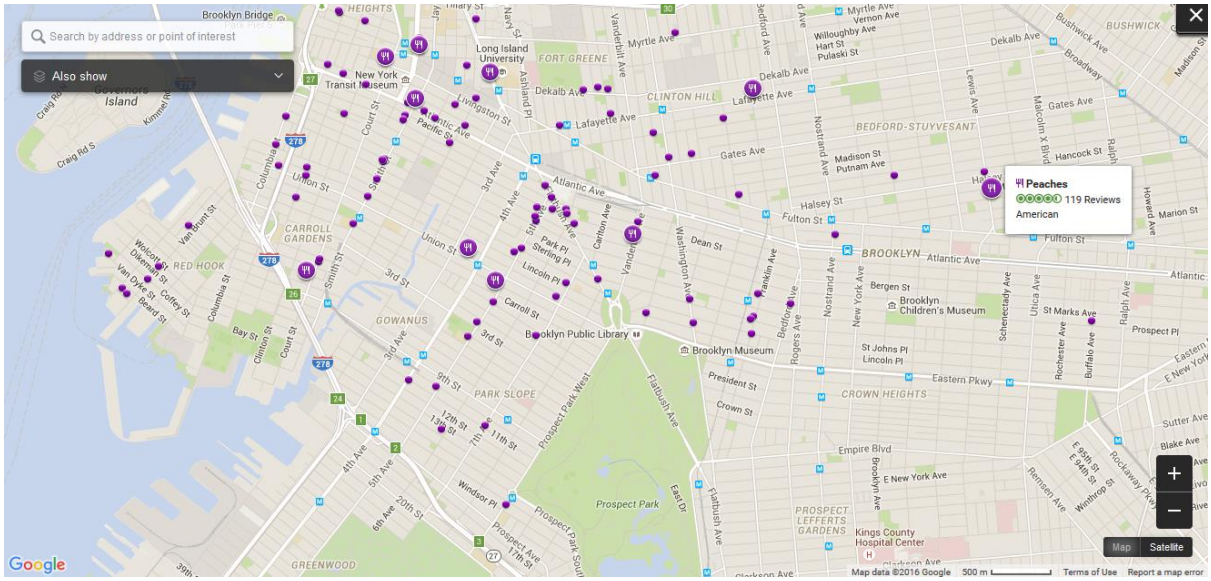
Σχήμα 5.3: Σελίδα εστιατορίου στο iOS app του Tripadvisor

Τέλος, υπάρχει η επιλογή προβολής των αποτελεσμάτων αναζήτησης σε χάρτη. Κάνοντας click σε έναν marker, εμφανίζεται το όνομα του εστιατορίου, η βαθμολογία και ο αριθμός των reviews.



Σχήμα 5.4: Χάρτης εστιατορίων στο Brooklyn στο iOS app του Tripadvisor

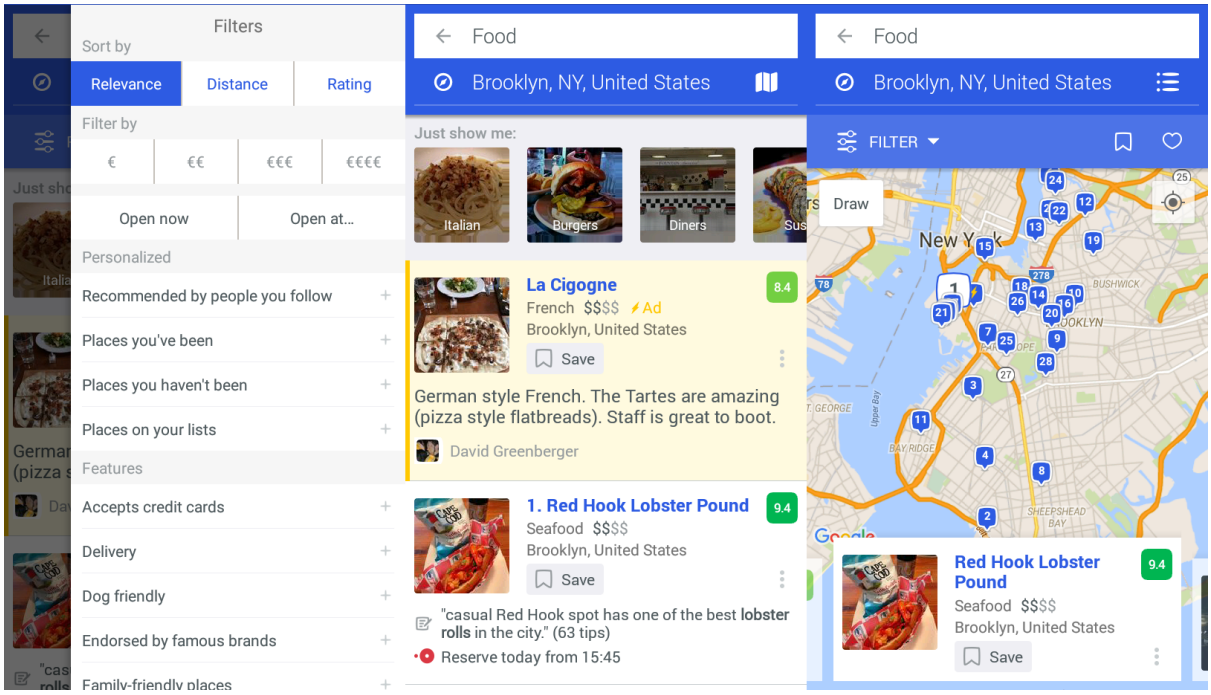
Από τα παραπάνω προκύπτει πως το Tripadvisor είναι μία πολύ πρακτική εφαρμογή για την οργάνωση ενός ταξιδιού, αφού παρέχει λειτουργίες για την εύρεση αεροπορικών εισιτηρίων, ξενοδοχείων, εστιατορίων και διάφορων δραστηριοτήτων. Το μεγάλο πλήθος θετικών και αρνητικών αξιολογήσεων από χρήστες της εφαρμογής εξασφαλίζει τη σφαιρική ενημέρωση του ενδιαφερόμενου πριν επισκεφτεί έναν προορισμό. Όσον αφορά την απεικόνιση σε χάρτη, όμως, δεν είναι δυνατή η άμεση εξαγωγή συμπερασμάτων για την κατανομή των αποτελεσμάτων σε ολόκληρη την πόλη, αφού προβάλλονται μόνο τα πιο δημοφιλή, ενώ για την απεικόνιση περισσότερων είναι απαραίτητη η μεγέθυνση σε μικρότερη περιοχή.



Σχήμα 5.5: Χάρτης αποτελεσμάτων μετά από μεγέθυνση στο web app του Tripadvisor

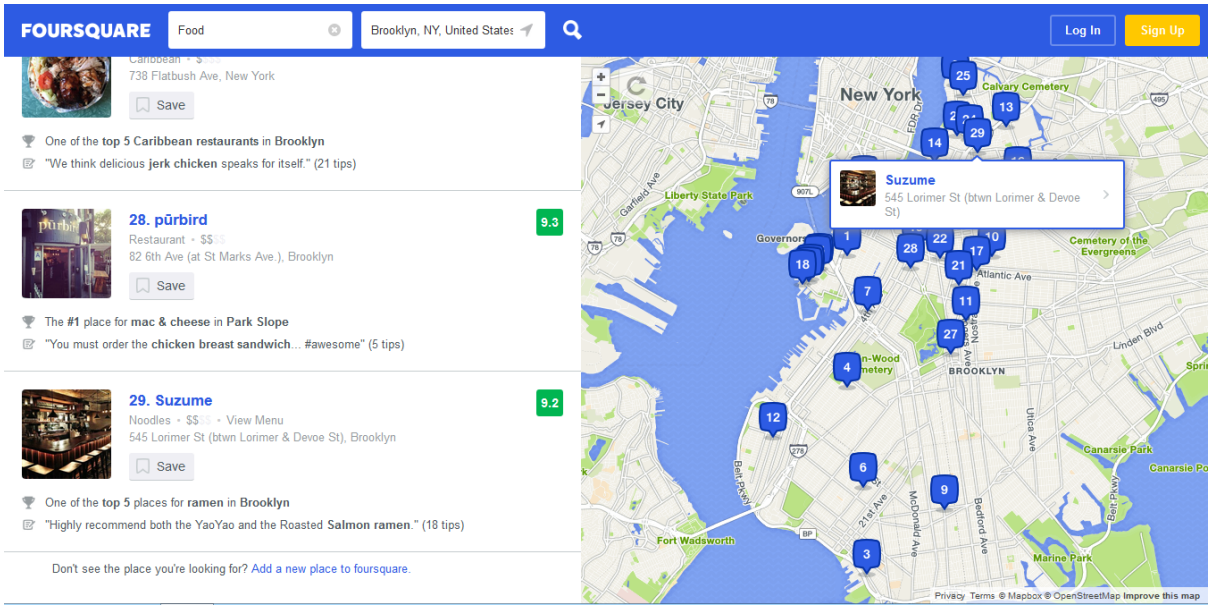
5.2.2 Foursquare

Η εφαρμογή του Foursquare έχει ως κύριο στόχο να προτείνει τα πιο δημοφιλή venues σε μία περιοχή βάσει κάποιων λέξεων-κλειδιών που εισάγει ο χρήστης σε ένα πεδίο κειμένου αναζήτησης. Λαμβάνοντας την τοποθεσία του χρήστη ή μια άλλη τοποθεσία της επιλογής του, επιστρέφει τα venues που περιλαμβάνουν κάποιο από τα keywords της αναζήτησης στο όνομά τους, στην περιγραφή τους ή στα σχόλια των χρηστών που τα έχουν επισκεφτεί. Τα αποτελέσματα μπορούν να ταξινομηθούν ανάλογα με τη συνολική βαθμολογία χρηστών ή την απόσταση από την τοποθεσία του χρήστη και εμφανίζονται σε χάρτη για την ευκολότερη εύρεση της τοποθεσίας τους. Παράλληλα, υπάρχουν και φίλτρα για την εισαγωγή περισσότερων κριτηρίων στην αναζήτηση, όπως εύρος τιμών των προϊόντων, ωράριο, αν ο χρήστης τα έχει ήδη επισκευτεί, κλπ.



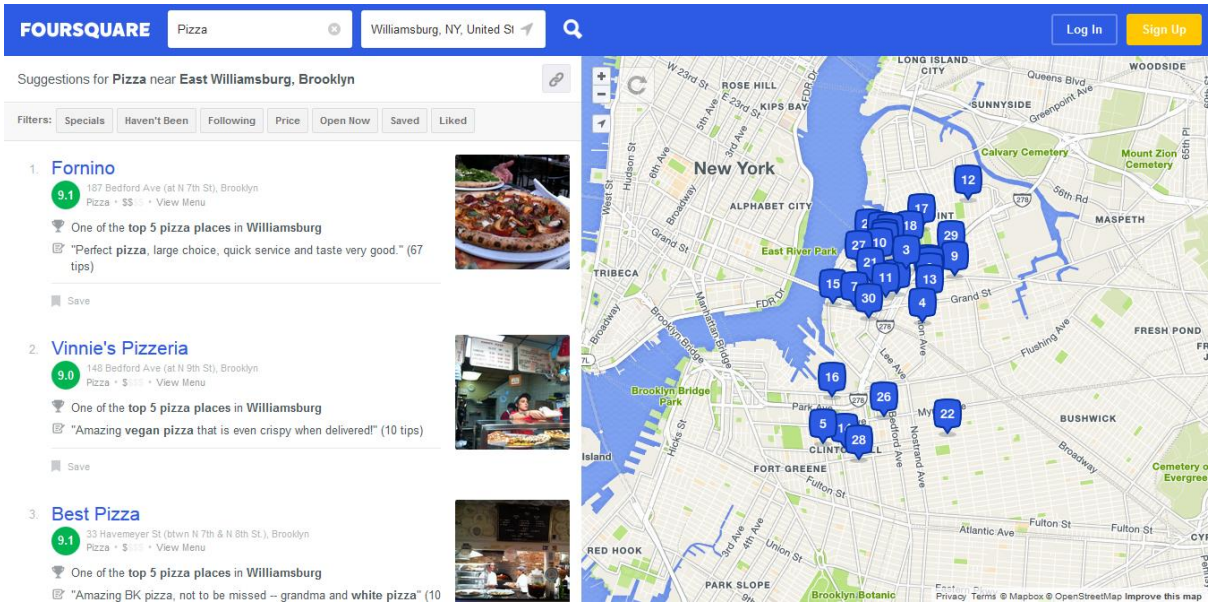
Σχήμα 5.6: Τρία στιγμιότυπα από την Android εφαρμογή του Foursquare

Στην παραπάνω εικόνα φαίνονται οι πολλές δυνατότητες του Foursquare για την εξατομίκευση των αποτελεσμάτων ανάλογα με τις προτιμήσεις του χρήστη, πχ εύρος τιμών, ώρες λειτουργίας, απόσταση από τη θέση του χρήστη, κλπ. Παρ' όλα αυτά, έχει το μειονέκτημα του περιορισμένου αριθμού αποτελεσμάτων, αφού σε κάθε αναζήτηση εμφανίζονται μόνο τα 30 καλύτερα αποτελέσματα που πληρούν τα κριτήρια που έθεσε ο χρήστης.



Σχήμα 5.7: Αποτελέσματα μιας αναζήτησης στη web εφαρμογή του Foursquare

Όπως είναι λογικό, υπάρχουν αρκετά venues που δεν εμφανίζονται στο χρήστη, κυρίως επειδή δεν είναι τόσο δημοφιλή από άποψη βαθμολογίας και επισκεψιμότητας. Για καλύτερα αποτελέσματα μία λύση είναι η αναζήτηση ενός πιο συγκεκριμένου keyword σε μια μικρότερη περιοχή. Για παράδειγμα, αντί για Food στο Brooklyn, να γίνει αναζήτηση για Pizza στο Williamsburg:



Σχήμα 5.8: Αποτελέσματα μιας πιο συγκεκριμένης αναζήτησης στο Foursquare

Συμπερασματικά, το Foursquare είναι ιδανικό για τη γρήγορη εύρεση των δημοφιλέστερων venues σε μία περιοχή μιας πόλης, αφού δίνει στο χρήστη τη δυνατότητα να προσαρμόσει την αναζήτησή του με πολλά κριτήρια. Παρ' όλα αυτά, δεν παρέχει πλήρη εικόνα για το συνολικό αριθμό των venues μίας κατηγορίας σε ολόκληρη την πόλη ή για την κατανομή τους σε διάφορες περιοχές. Τέτοιου είδους πληροφορίες παρέχει με μεγαλύτερη αμεσότητα η εφαρμογή που αναπτύχθηκε στην παρούσα διπλωματική εργασία.

5.2.3 Εφαρμογή της διπλωματικής

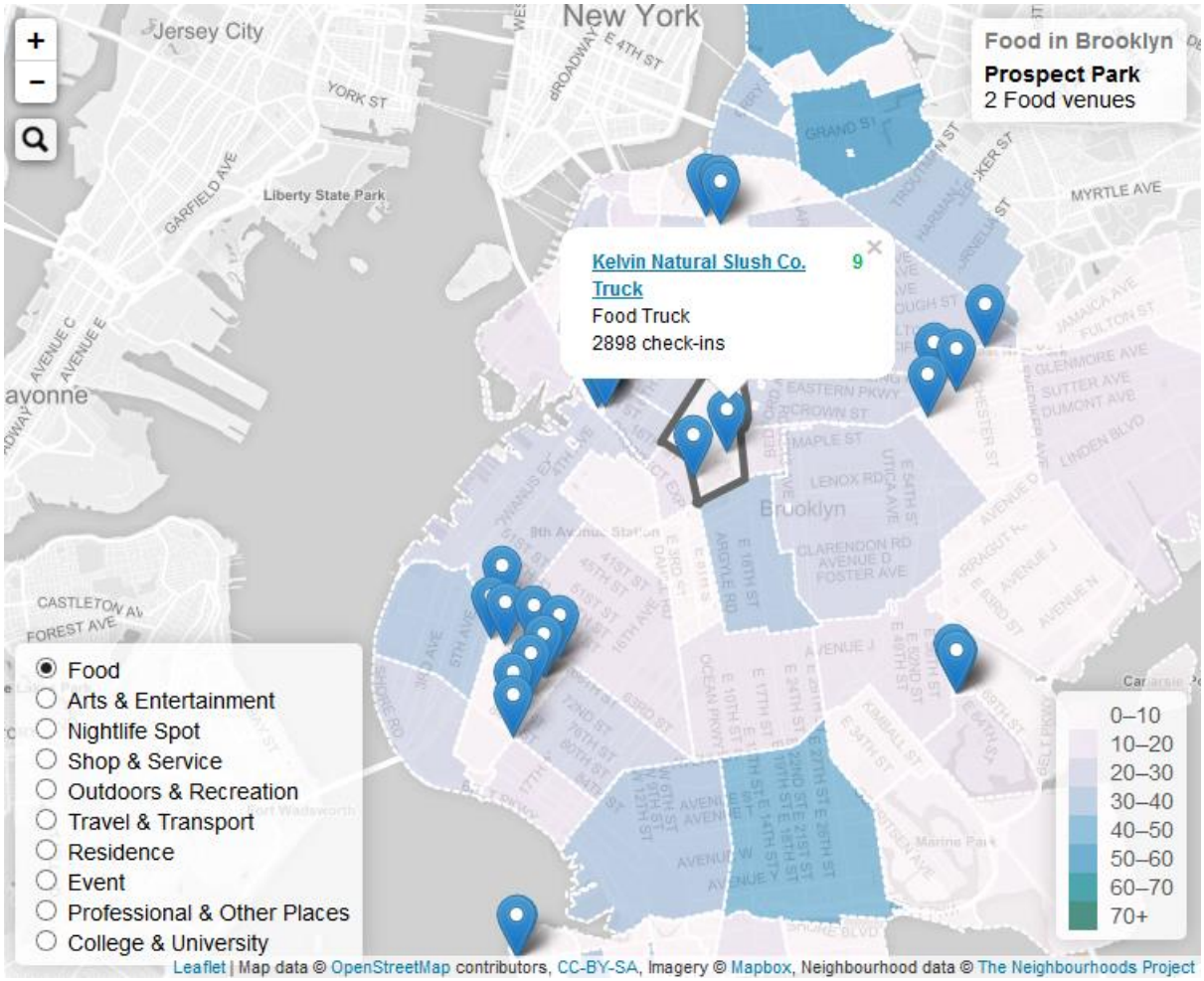
Η λειτουργία της εφαρμογής που υλοποιήθηκε βασίζεται στην επαναληπτική αναζήτηση στο Foursquare για κάθε επιμέρους περιοχή μιας πόλης. Για κάθε περιοχή λαμβάνονται έως και 50 venues και αποθηκεύονται σε συλλογές ανάλογα με την κατηγορία όπου ανήκουν. Οι κατηγορίες των venues είναι: Food, Arts & Entertainment, Nightlife Spot, Shop & Service, Outdoors & Recreation, Travel & Transport, Residence, Event, Professional & Other Place, College & University. Ως αποτέλεσμα, συλλέγεται ένας μεγάλος αριθμός venues για μία πόλη, πχ σε όλο το Brooklyn βρέθηκαν 1041 venues της κατηγορίας Food.

```
db.getCollection('Food').find({'location.city': {$eq: 'Brooklyn'}})
```

Key	Value
(1033) ObjectId("5731ded6f63f6bf00ec97e13")	{ 22 fields }
(1034) ObjectId("5731ded6f63f6bf00ec97e16")	{ 21 fields }
(1035) ObjectId("5731ded6f63f6bf00ec97e1a")	{ 22 fields }
(1036) ObjectId("5731ded6f63f6bf00ec97e1c")	{ 18 fields }
(1037) ObjectId("5731ded6f63f6bf00ec97e1d")	{ 17 fields }
(1038) ObjectId("5731ded6f63f6bf00ec97e1e")	{ 20 fields }
(1039) ObjectId("5731ded6f63f6bf00ec97e1f")	{ 20 fields }
(1040) ObjectId("5731ded6f63f6bf00ec97e20")	{ 20 fields }
_id	ObjectId("5731ded6f63f6bf00ec97e20")
name	Siam Orchid
id	4a8dcb4ff964a520e61020e3
contact	{ 2 fields }
location	{ 11 fields }
categories	[1 element]
verified	false
stats	{ 3 fields }
price	{ 3 fields }
hasMenu	true
rating	7.1
ratingColor	C5DE35
ratingSignals	22
delivery	{ 3 fields }
menu	{ 5 fields }
allowMenuUrlEdit	true
hours	{ 2 fields }
photos	{ 2 fields }
hereNow	{ 3 fields }
geojson	{ 3 fields }
(1041) ObjectId("5731ded6f63f6bf00ec97e21")	{ 17 fields }

Σχήμα 5.9: Query στη MongoDB με τα Food venues στο Brooklyn

Τα αποτελέσματα αυτά εμφανίζονται σε ένα χάρτη της πόλης χωρισμένης σε περιοχές, όπου η κάθε μία χρωματίζεται ανάλογα με τον αριθμό των venues μιας κατηγορίας που βρέθηκαν σε αυτή. Κάνοντας click σε μία περιοχή εμφανίζονται markers με τις τοποθεσίες των venues, ενώ κάνοντας click σε έναν marker εμφανίζεται popup με τις βασικές πληροφορίες του venue.



Σχήμα 5.10: Ο χάρτης των venues της κατηγορίας Food που βρέθηκαν στο Brooklyn

Έτσι, λοιπόν, παρουσιάζεται με άμεσο τρόπο η κατανομή των venues ανά περιοχή της πόλης, ενώ το μεγάλο πλήθος των αποτελεσμάτων παρέχει στο χρήστη περισσότερες επιλογές, πέρα από τις πιο δημοφιλείς. Παρ' όλα αυτά, στην αρχική έκδοση της εφαρμογής δεν παρέχεται η δυνατότητα για ταξινόμηση ή φιλτράρισμα των αποτελεσμάτων, καθιστώντας δύσκολη την εύρεση του ιδανικότερου venue για το χρήστη.

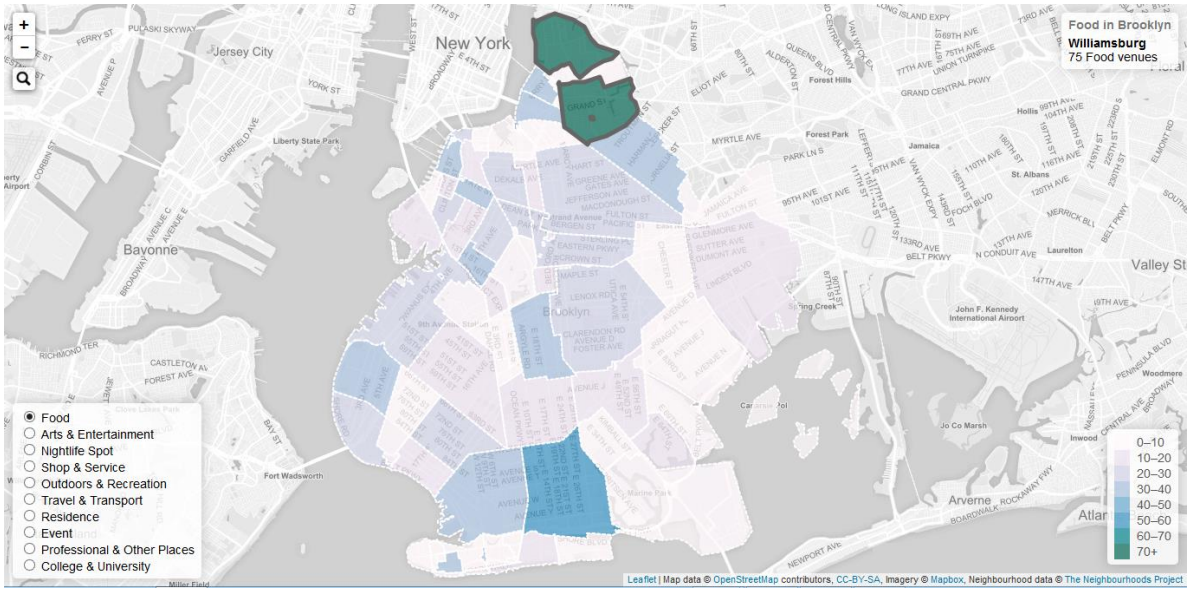
5.3 Επίδραση επικαιρότητας στο σύστημα

Τα δεδομένα που προβάλλονται στο χάρτη της εφαρμογής συλλέγονται από το Foursquare και ένα μέρος του περιεχομένου δημιουργείται και ανανεώνεται συνεχώς από τους χρήστες του Foursquare. Είναι, λοιπόν, απαραίτητη η συνεχής ανανέωση της βάσης δεδομένων για την εξασφάλιση των πιο πρόσφατων αποτελεσμάτων. Στην παρούσα ενότητα παρουσιάζονται η επίδραση της ώρας που γίνεται η αναζήτηση στον αριθμό των αποτελεσμάτων και η δυνατότητα χρήσης της εφαρμογής για τη μελέτη της δημοτικότητας των venues με την πάροδο του χρόνου.

5.3.1 Εξάρτηση αποτελεσμάτων από την ώρα αναζήτησης

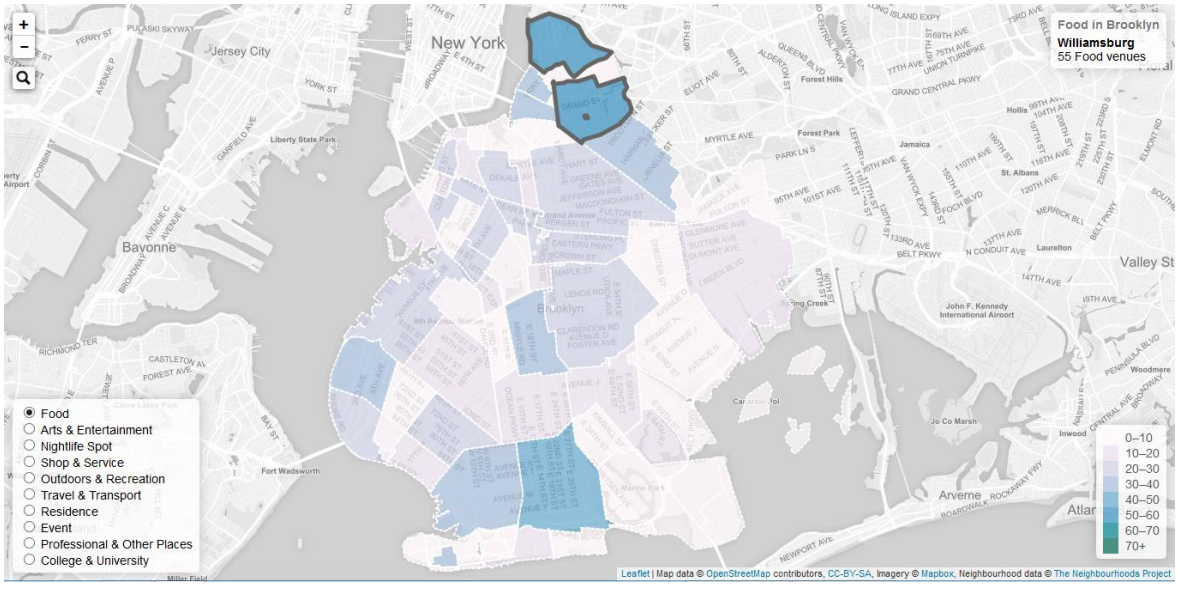
Με μερικές δοκιμές της εφαρμογής μπορεί να παρατηρηθούν κάποιες μικρές διαφορές στον αριθμό των αποτελεσμάτων της ίδιας αναζήτησης, κάτι που αρχικά φαίνεται περίεργο αν αναλογιστεί κανείς ότι τα δεδομένα αφορούν κυρίως επιχειρήσεις, οπότε ο ρυθμός προσθήκης ή διαγραφής των venues στο Foursquare δεν είναι τόσο γρήγορος. Αυτό οφείλεται στο γεγονός ότι το Foursquare API λαμβάνει υπόψιν και την ώρα της αναζήτησης. Άλλωστε, σε πολλά από τα venues περιλαμβάνεται και ωράριο λειτουργίας, οπότε ενδέχεται κάποια κλειστά μαγαζιά να παραλειφθούν από τα αποτελέσματα.

Για παράδειγμα, η αναζήτηση στο Brooklyn των venues της κατηγορίας Food επέστρεψε τα εξής αποτελέσματα στις 20:30 (13:30 ώρα Νέας Υόρκης).



Σχήμα 5.11: Αποτελέσματα αναζήτησης Food στο Brooklyn στις 13:30

Ενώ όταν η αναζήτηση έγινε στις 11:00 (4:00 ώρα Νέας Υόρκης):



Σχήμα 5.12: Αποτελέσματα αναζήτησης Food στο Brooklyn στις 4:00

Από τις παραπάνω εικόνες, φαίνεται η διαφορά στα χρώματα μερικών περιοχών, μάλιστα στο Williamsburg βρέθηκαν 20 αποτελέσματα λιγότερα.

Το φαινόμενο αυτό μπορεί να εξαλειφθεί αν κατά την αναζήτηση περαστούν στο URL του HTTP GET request η τιμή “any” στις παραμέτρους “time” και “day”, δηλαδή στην αναζήτηση να μη ληφθούν υπόψιν η ώρα και η μέρα.

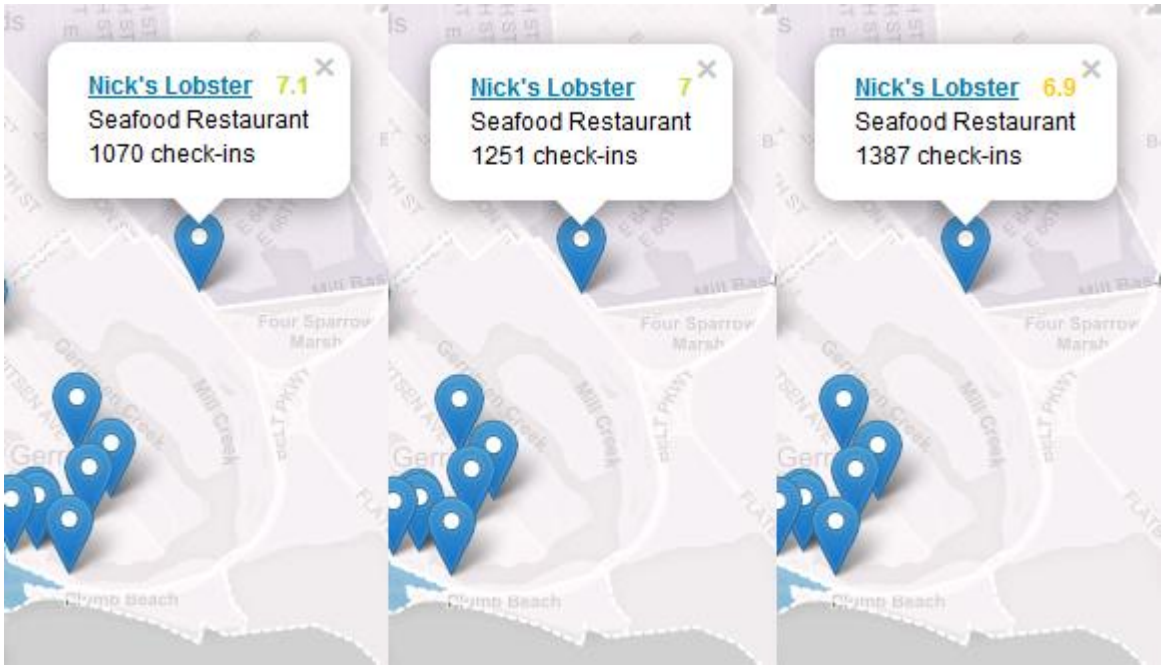
time	any	Pass `any` to retrieve results for any time of day. Omitting this parameter returns results targeted to the current time of day.
day	any	Pass `any` to retrieve results for any day of the week. Omitting this parameter returns results targeted to the current day of the week.

Σχήμα 5.13: Οι παράμετροι time και day στο endpoint “explore” του Foursquare API

5.3.2 Μελέτη της εξέλιξης της δημοτικότητας ενός venue

Όπως αναφέρθηκε στην ενότητα 5.2.3, ένα πλεονέκτημα της εφαρμογής είναι η εμφάνιση ενός μεγάλου πλήθους venues σε μία πόλη, που προκύπτει από την αναζήτηση στο Foursquare σε κάθε περιοχή ξεχωριστά. Ενώ στην εφαρμογή του Foursquare τα αποτελέσματα περιορίζονται μόνο στις πιο δημοφιλείς τοποθεσίες, στο χάρτη της εφαρμογής της διπλωματικής εμφανίζονται και venues με χαμηλή βαθμολογία ή κάποια καινούρια που δεν έχουν γίνει ακόμα τόσο δημοφιλή.

Με αυτό τον τρόπο, παρέχεται η δυνατότητα για παρατήρηση της επισκεψιμότητας ενός venue και των μεταβολών στη βαθμολογία του, ώστε να εξαχθούν συμπεράσματα για την εξέλιξη της δημοτικότητάς του. Έτσι, λοιπόν, μπορούν να αναγνωριστούν κάποια “ανερχόμενα” venues της πόλης πριν γίνουν αρκετά δημοφιλή ώστε να εμφανιστούν στα αποτελέσματα του Foursquare ή, αντίστροφα, να παρατηρηθεί η πτώση στη βαθμολογία κάποιου εστιατορίου με την πάροδο του χρόνου, όπως στο παρακάτω παράδειγμα:



Σχήμα 5.14: Η πτώση της βαθμολογίας ενός εστιατορίου σε διαστήματα 1 εβδομάδας

Τέλος, σημειώνεται πως το παραπάνω παράδειγμα βασίστηκε σε απλές παρατηρήσεις στο χάρτη της εφαρμογής. Στο επόμενο κεφάλαιο προτείνονται μελλοντικές επεκτάσεις της εφαρμογής για πιο συστηματική μελέτη της δραστηριότητας των venues για την εξαγωγή στατιστικών αποτελεσμάτων.

Κεφάλαιο 6
Μελλοντικές Επεκτάσεις

6.1 Εισαγωγή

Κατά την ανάπτυξη της εφαρμογής έγινε η προσπάθεια να συμπεριληφθούν όλα τα χαρακτηριστικά από την αρχική ανάλυση απαιτήσεων και δόθηκε ιδιαίτερη προσοχή στην αντιμετώπιση των πιθανών σφαλμάτων κατά την εκτέλεση. Το διαδίκτυο, όμως, εξελίσσεται ραγδαία, οι ανάγκες των χρηστών αλλάζουν και προκειμένου μια εφαρμογή να παραμείνει χρηστική και επίκαιρη πρέπει να έχει συνεχή εξέλιξη και ανάπτυξη. Στο παρόν κεφάλαιο παρουσιάζονται κάποιες αλλαγές και προσθήκες που μπορούν να γίνουν για τη βελτίωση και επέκταση της εφαρμογής.

6.2 Αναζήτηση Οποιαδήποτε Πόλης Σε Πραγματικό Χρόνο

Στην εφαρμογή δίνεται η δυνατότητα στο χρήστη να δει πληροφορίες για πόλεις που υπάρχουν ήδη στη βάση δεδομένων. Το Geodata Service της εφαρμογής μπορεί να αναζητά δεδομένα από το Zetashapes API για 30080 πόλεις των ΗΠΑ. Για πόλεις εκτός των ΗΠΑ, είναι απαραίτητη η σύνδεση με κάποιο άλλο API που προσφέρει γεωγραφικά δεδομένα σε GeoJSON format. Με αυτή την προϋπόθεση, η εφαρμογή λειτουργεί και για άλλες πηγές γεωγραφικών δεδομένων.

Επιπλέον, τα δεδομένα που εμφανίζονται στο χάρτη ανακτώνται από τη βάση δεδομένων, όπου υπάρχουν ήδη αποθηκευμένα, δηλαδή δε γίνεται εκείνη τη στιγμή η σύνδεση με το API. Με αυτό τον τρόπο εξασφαλίζεται η άμεση απόκριση της εφαρμογής σε κάθε ενέργεια του χρήστη. Δεν υποστηρίζεται η αναζήτηση δεδομένων σε πραγματικό χρόνο χρήστη, διότι υπήρχε πρόβλημα συγχρονισμού των δεδομένων που αποθηκεύουν τα services με το front-end, του οποίου η λειτουργία γίνεται ασύγχρονα. Αντί αυτού, σχεδιάστηκαν αυτόνομα services για την ανάκτηση και αποθήκευση των δεδομένων, τα οποία μπορούν να εκτελούνται και να ανανεώνουν τη βάση δεδομένων με οποιαδήποτε συχνότητα απαιτείται. Μια βελτίωση, λοιπόν, θα ήταν η αποδοτική υλοποίηση της αναζήτησης των δεδομένων που επιθυμεί ο χρήστης με τη δημιουργία ενός ενδιάμεσου layer για το συγχρονισμό των δεδομένων που ανταλλάσσονται μεταξύ των τμημάτων της εφαρμογής.

6.3 Φιλτράρισμα Κατηγοριών Venues Με Layers

Ο χάρτης της εφαρμογής εμφανίζει τα Foursquare venues μόνο της κατηγορίας που έχει επιλέξει ο χρήστης κατά την αναζήτηση. Για την εμφάνιση διαφορετικής κατηγορίας πρέπει να επαναληφθεί το αίτημα. Σε κάθε αίτημα, οι γειτονιές της πόλεις χρωματίζονται ανάλογα με το πλήθος των venues της εκάστοτε κατηγορίας, το οποίο εμφανίζεται όταν ο χρήστης περνάει το δείκτη του ποντικιού πάνω από το πολύγωνο μιας γειτονιάς.

Μια βελτίωση, λοιπόν, θα ήταν η φόρτωση των venues όλων των κατηγοριών και η δημιουργία ξεχωριστού layer στο χάρτη για κάθε κατηγορία. Έτσι, ο χρήστης, θα έχει τη δυνατότητα να φιλτράρει τα αποτελέσματα ανά κατηγορία, χωρίς να χρειάζεται η ανανέωση της σελίδας.

6.4 Μελέτη Δραστηριότητας Στα Venues

Στις πληροφορίες που συλλέγονται για κάθε venue περιλαμβάνεται κι ο αριθμός των συνολικών επισκεπτών (check-ins). Μια πιθανή επέκταση θα ήταν η δημιουργία ενός daemon, το οποίο θα εκτελείται ανά τακτά χρονικά διαστήματα παρακολουθώντας και καταγράφοντας τον αριθμό των check-ins στα venues μιας περιοχής ή μιας κατηγορίας. Συλλέγοντας συστηματικά τα δεδομένα υπάρχει έπειτα η δυνατότητα για τη στατιστική μελέτη τους και με αυτό τον τρόπο μπορούν να εξαχθούν συμπεράσματα όπως οι ώρες/μέρες αιχμής σε κάποια μαγαζιά, οι περιοχές της πόλης που προτιμώνται για βραδινή έξοδο, η εξέλιξη της δημοτικότητας ενός venue, η εύρεση ενός νέου, “ανερχόμενου” εστιατορίου και πολλά ακόμη.

Βιβλιογραφία

[1] "Map types and projections" Wikipedia https://en.wikipedia.org/wiki/Map

[2] "Social Network" Wikipedia https://en.wikipedia.org/wiki/Social_network

[3] "Foursquare" https://foursquare.com/

[4] "The MongoDB Manual" https://docs.mongodb.com/manual/

[5] "The Neighbourhoods Project" http://zetashapes.com/

[6] "TIGER Products" United States Census Bureau https://www.census.gov/geo/maps-data/data/tiger.html

[7] "The GeoJSON Specification" http://geojson.org/

[8] "Haversine formula" Wikipedia https://en.wikipedia.org/wiki/Haversine_formula

[9] "Foursquare API Endpoints" https://developer.foursquare.com/docs/

[10] "Node.js v7.10.0 Documentation" https://nodejs.org/dist/latest-v7.x/docs/api/

[11] "Express.js 4.x API Reference" https://expressjs.com/en/4x/api.html

[12] "Mongoose Guide" http://mongoosejs.com/docs/guide.html

[13] "Pug.js API Reference" https://pugjs.org/api/reference.html

[14] "Mapbox" Wikipedia https://en.wikipedia.org/wiki/Mapbox

[15] "Leaflet 1.0.3. API Reference" http://leafletjs.com/reference-1.0.3.html

[16] "Leaflet Control Search Library" http://labs.easyblog.it/maps/leaflet-search/

[17] "Sams Teach Yourself Java 6 in 21 Days" Rogers Cadenhead, Laura Lemay (2007)

[18] "Why Use Node.js?" Tomislav Capan https://www.toptal.com/nodejs/why-the-hell-would-i-use-node-js

[19] "Representational state transfer" Wikipedia https://en.wikipedia.org/wiki/Representational_state_transfer

[20] "Tripadvisor" https://www.tripadvisor.com.gr/

Παράρτημα Α
Πηγαίος Κώδικας Java

Α.1 Ανάκτηση Γεωγραφικών Δεδομένων – AreaClass.java

```
package foursquare;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.URL;
import java.util.ArrayList;
import java.util.List;

import javax.json.JsonArray;
import javax.json.JsonObject;

public class AreaClass {

	static String city_name;

	public static void main(String[] city) {
		city_name = city[0];
		try {
			new AreaClass();
		} catch (IOException e) {
			e.printStackTrace();
		}
	}

	public AreaClass() throws IOException {

		String area_name = city_name;
		String area_id = getAreaId(area_name);
		if (area_id == "") {
			System.out.println("City not found.\n");
		} else {

			new MongoClass();
			boolean exists = MongoClass.findArea(area_name);

			new JsonClass();
			if (!exists) {
				System.out.println("Searching for
neighbourhoods in " + area_name + ".\n");
				JsonObject hoods =
JsonClass.getNeighbourhoods(area_name, area_id);
				MongoClass.saveNeighbourhoods(area_name,
hoods);
```

```
                        System.out.println("Saved all neighbourhoods
in collection " + area_name + ".\n");
                }
                new VenueClass();
            }
        }

        /**
         * Returns a String of the ID of a US City according to the
Neighborhoods
         * Project (zetashapes.com) {@link String}. The name argument
is a String
         * that has to be the exact name of a city in zetashapes.com.
If the city is
         * not found, the method returns "".
         *
         * @param name
         *            the name of a US city
         * @return the ID of the city
         * @see String
         */
        public String getAreaId(String name) throws IOException {

            String id = "";
            URL url = new
URL("https://dl.dropboxusercontent.com/u/28290565/countyid.txt");
            BufferedReader in = new BufferedReader(new
InputStreamReader(url.openStream()));
            String line = "";
            // line example: ["Los Angeles, CA",06037],
            while ((line = in.readLine()) != null && id == "") {
                String[] tokens = line.split(",");
                if (tokens[0].equals("[\"" + name)) {
                    String[] ids = tokens[2].split("]");
                    id = ids[0];
                }
            }
            in.close();
            return id;
        }

        /**
         * Returns a List<Double> containing the coordinates of the
polygon's
         * centroid {@link List<E>}. The coords argument is a
JsonArray containing
         * the polygon's edges {@link JsonArray}.
         *
         * @param coords
         *            a JsonArray containing the polygon's edges
         * @return a list containing the coordinates of the polygon's
centroid
         * @see List<E>, JsonArray
         */
        public static List<Double> calCentroid(JsonArray coords) {
```

```
            List<Double> centroid = new ArrayList<Double>();
            double cx = 0, cy = 0;
            int s = coords.size();
            int i = 0;
            while (i < s) {
                  String[] point =
coords.get(i).toString().split(",");
                  String lng = point[0].substring(1);
                  double x = Double.parseDouble(lng);

                  String[] l = point[1].split("]");
                  String lat = l[0].toString();
                  double y = Double.parseDouble(lat);

                  cx = cx + x;
                  cy = cy + y;
                  i++;
            }
            cx = cx / s;
            cy = cy / s;
            centroid.add(cx);
            centroid.add(cy);

            return centroid;
      }

      /**
       * Returns the radius of the circle around a given polygon.
The coords
       * argument is a JsonArray containing the polygon's edges
{@link JsonArray}.
       * The centroid argument is a List<Double> containing the
coordinates of the
       * polygon's centroid {@link List<E>}.
       *
       * @param coords
       *            a JsonArray containing the polygon's edges
       * @param centroid
       *            a list of Doubles containing the coordinates of
the polygon's
       *            centroid
       * @return a long number representing the circle's radius in
meters
       * @see List<E>, JsonArray, long
       */
      public static long calRadius(JsonArray coords, List<Double>
centroid) {

            double cx = centroid.get(1), cy = centroid.get(0), radius
= 0;
            int s = coords.size();
            int i = 0;
            while (i < s) {

                  String[] point =
coords.get(i).toString().split(",");
```

```
                String lng = point[0].substring(1);
                double y = Double.parseDouble(lng);

                String[] l = point[1].split("]");
                String lat = l[0].toString();
                double x = Double.parseDouble(lat);

                double dlon = Math.toRadians(cy - y);
                double dlat = Math.toRadians(cx - x);
                double a = Math.pow(Math.sin(dlat / 2), 2)
                        + Math.cos(Math.toRadians(cx)) *
Math.cos(Math.toRadians(x)) * Math.pow(Math.sin(dlon / 2), 2);
                double c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1
- a));
                double R = 6371000; // the radius of the Earth in
meters
                double r = R * c;
                if (r > radius)
                      radius = r;
                i++;
            }
            long r = Math.round(radius);
            return r;
      }
}
```

A.2 Αναζήτηση Foursquare Venues - VenueClass.java

```
package foursquare;

import java.io.IOException;
import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Calendar;
import java.util.List;

import javax.json.JsonArray;
import javax.json.JsonNumber;
import javax.json.JsonObject;

public class VenueClass {

      String client_id =
"UR1CQXWS0U5Z5YYX0QHJUDBJ2GLIVSZNASYOL14OGMLVULFI";
      String client_sc =
"0OCNNY3JZJ4FBDSDAWQQPHKGOQ425GAMZCHWBR5IJ0301240";
      String date = new
SimpleDateFormat("yyyyMMdd").format(Calendar.getInstance().getTime()
);
      String venue_url = "https://api.foursquare.com/v2/venues/";

      String area_name = AreaClass.city_name;
      List<JsonObject> categories = new ArrayList<JsonObject>();
```

```
	public VenueClass() throws IOException {

		String cat_url = venue_url + "categories?client_id=" +
client_id + "&client_secret=" + client_sc + "&v=" + date;
		categories = JsonClass.getCategories(cat_url);

		JsonArray hoods = MongoClass.findHoods(area_name);
		getCircle(hoods);

	}

	/**
	 * Finds the centroid and radius for each neighbourhood and
calls the
	 * explore method.
	 *
	 * @param hoods
	 *            a JsonArray containing all neighbourhoods
	 * @return void
	 * @see JsonArray
	 */
	public void getCircle(JsonArray hoods) throws IOException {

		int i = 0;
		while (i < hoods.size()) {
			JsonObject hood = (JsonObject) hoods.get(i);
			String hoodname =
hood.getJsonObject("properties").getString("name");
			int j = 0;
			while (j <
hood.getJsonObject("properties").getJsonArray("radius").size()) {

				JsonObject c = (JsonObject)
hood.getJsonObject("properties").getJsonArray("centroid").get(j);
				double cen_lat =
c.getJsonNumber("lat").doubleValue();
				double cen_lng =
c.getJsonNumber("lng").doubleValue();

				JsonNumber r = (JsonNumber)
hood.getJsonObject("properties").getJsonArray("radius").get(j);
				double radius = r.doubleValue();

				explore(cen_lat, cen_lng, radius, hoodname);
				j++;
			}
			countVenues(hood, categories);
			i++;
		}
		System.out.println("All venues in " + area_name + " saved
into Database.\n");
	}

	/**
	 * Explores Foursquare venues given the coordinates of a point
and a radius
```

```
	 * in meters. Then, saves the venues in the appropriate
category/collection
	 * in the MongoDB.
	 *
	 * @param lat
	 *            a double representing the latitude
	 * @param lng
	 *            a double representing the longitude
	 * @param radius
	 *            a double representing the radius
	 * @param hoodname
	 *            a String representing the name of a
neighbourhood
	 * @return void
	 * @see double, String
	 */
	public void explore(double lat, double lng, double radius,
String hoodname) throws IOException {

		String ven = venue_url + "explore?client_id=" + client_id
+ "&client_secret=" + client_sc + "&v=" + date
				+ "&ll=" + lat + "," + lng +
"&limit=50&radius=" + radius;

		System.out.println("Exploring venues in " + hoodname +
".\n");
		JsonArray venues = JsonClass.exploreVenues(ven);

		for (JsonObject res :
venues.getValuesAs(JsonObject.class)) {

			JsonArray cat = res.getJsonArray("categories");
			String cat_id =
cat.getJsonObject(0).getString("id");
			String catname = getCategory(cat_id);

			JsonObject geojson = JsonClass.createGeoJson(res);

			if (catname != null)
				MongoClass.saveVenue(catname, res, geojson,
hoodname, area_name);
		}
	}

	/**
	 * Returns a String representing the name of the parent
category of a venue,
	 * given a String of the id of a child category {@link
String}.
	 *
	 * @param cat_id
	 *            the id of the child category
	 * @return a String of the name of the parent category
	 * @see String
	 */
	public String getCategory(String cat_id) throws IOException {
```

```
            String catname = null;
            int i = 0;
            while (i < categories.size()) {
                  JsonArray child =
categories.get(i).getJsonArray("child_cat_ids");
                  int j = 0;
                  while (j < child.size()) {
                        String s = child.getString(j);
                        if (s.equals(cat_id)) {
                              catname =
categories.get(i).getString("name");
                              j = child.size();
                              i = categories.size();
                        }
                        j++;
                  }
                  i++;
            }
            return catname;
      }

      /**
       * Calls the getVenues method that counts the venues in each
category, then
       * calls the updateHood method to update the neighbourhood
document in
       * MongoDB.
       *
       * @param hood
       *            a JsonObject containing neighbourhood geodata
       * @param categories
       *            a List of JsonObjects containing the category
names and ids
       * @return void
       * @see JsonObject
       */
      public void countVenues(JsonObject hood, List<JsonObject>
categories) throws IOException {

            int j = 0;
            while (j < categories.size()) {
                  JsonObject category = (JsonObject)
categories.get(j);
                  String catname = category.getString("name");
                  long count = MongoClass.getVenues(hood, catname);
                  MongoClass.updateHood(area_name, hood, catname,
count);
                  j++;
            }
      }
}
```

A.3 Επεξεργασία JSON - JsonClass.java

```
package foursquare;

import java.io.IOException;
import java.io.InputStream;
import java.net.URL;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import javax.json.Json;
import javax.json.JsonArray;
import javax.json.JsonArrayBuilder;
import javax.json.JsonBuilderFactory;
import javax.json.JsonNumber;
import javax.json.JsonObject;
import javax.json.JsonReader;
import javax.json.JsonString;

public class JsonClass {

	/**
	 * Returns a JsonObject containing all the neighbourhoods in a
US City
	 * according to the Neighborhoods Project (zetashapes.com)
	 * {@link JsonObject}. The areaname argument is a String of
the name of the
	 * city. The areaid argument is a String of the id of the city
according to
	 * zetashapes.com.
	 *
	 * @param areaname
	 *            the name of the city
	 * @param areaid
	 *            the id of the city
	 * @return JsonObject containing all the neighbourhoods
	 * @see String, JsonObject
	 */
	public static JsonObject getNeighbourhoods(String areaname,
String areaid) throws IOException {

		URL urltag = new
URL("http://zetashapes.com/api/neighborhoodsByArea?areaid=" +
areaid);

		Map<String, Object> config = new HashMap<String,
Object>();
		JsonBuilderFactory factory =
Json.createBuilderFactory(config);
		JsonArrayBuilder neighbourhoods =
factory.createArrayBuilder();
```

```
            InputStream is = urltag.openStream();
            JsonReader rdr = Json.createReader(is);
            JsonObject obj = rdr.readObject();
            JsonArray ftr = obj.getJsonArray("features");

            for (JsonObject res : ftr.getValuesAs(JsonObject.class))
{

                  JsonObject props = res.getJsonObject("properties");
                  JsonString hood_name = props.getJsonString("label");
                  JsonNumber hood_pop = props.getJsonNumber("pop10");

                  JsonArrayBuilder centroid =
factory.createArrayBuilder();
                  JsonArrayBuilder radius =
factory.createArrayBuilder();

                  String type =
res.getJsonObject("geometry").getString("type");
                  JsonArray temp =
res.getJsonObject("geometry").getJsonArray("coordinates");
                  JsonArrayBuilder coords =
factory.createArrayBuilder();
                  int i = 0;
                  while (i < temp.size()) {
                        if (type.equals("MultiPolygon")) {
                              JsonArray polyg = (JsonArray)
temp.get(i);
                              coords.add(polyg.get(0)); // Ignore
polygon's holes (if any)
                        } else {
                              coords.add(temp.get(0)); // Ignore
polygon's holes (if any)
                        }
                        JsonArray coods = coords.build();

                        List<Double> centr =
AreaClass.calCentroid((JsonArray) coods.get(i));
                        JsonObject c =
factory.createObjectBuilder().add("lng", centr.get(0)).add("lat",
centr.get(1)).build();
                        centroid.add(c);
                        long r = AreaClass.calRadius((JsonArray)
coods.get(i), centr);
                        radius.add(r);
                        i++;
                  }

                  JsonObject hood =
factory.createObjectBuilder().add("geometry",
res.getJsonObject("geometry"))
                              .add("type", "Feature")
                              .add("properties",
factory.createObjectBuilder().add("name",
hood_name).add("population", hood_pop)
```

```
                                        .add("centroid",
centroid.build()).add("radius", radius.build()).build())
                                .build();

                        neighbourhoods.add(hood);
                }
                JsonObject hoods =
factory.createObjectBuilder().add("type", "FeatureCollection")
                                .add("features", neighbourhoods).build();
                rdr.close();
                return hoods;
        }

        /**
         * Returns a List<JsonObject> containing all the categories of
Foursquare's
         * venues along with the ids of their children categories
{@link List<E>}.
         * The line argument is a String of the URL to Foursquare's
API
         * {@link String}.
         *
         * @param line
         *            the URL to Foursquare's API
         * @return a List<JsonObject> containing all the categories
         * @see String, List<E>
         */
        public static List<JsonObject> getCategories(String line)
throws IOException {

                Map<String, Object> config = new HashMap<String,
Object>();
                JsonBuilderFactory factory =
Json.createBuilderFactory(config);

                List<JsonObject> categories = new
ArrayList<JsonObject>();
                URL urltag = new URL(line);
                InputStream is = urltag.openStream();

                JsonReader rdr = Json.createReader(is);
                JsonObject obj = rdr.readObject();
                JsonArray cat =
obj.getJsonObject("response").getJsonArray("categories");

                for (JsonObject res : cat.getValuesAs(JsonObject.class))
{

                        JsonString cat_id = res.getJsonString("id");
                        JsonString cat_name = res.getJsonString("name");
                        JsonArray child_cats =
res.getJsonArray("categories");
                        JsonArrayBuilder ids = factory.createArrayBuilder();
                        for (JsonObject chd :
child_cats.getValuesAs(JsonObject.class)) {
                                JsonString id = chd.getJsonString("id");
```

```
                        ids.add(id);
                        // Some child categories have more child
categories
                        JsonArray child =
chd.getJsonArray("categories");
                        for (JsonObject chld :
child.getValuesAs(JsonObject.class)) {
                                JsonString id2 =
chld.getJsonString("id");
                                ids.add(id2);
                        }
                }
                JsonArray child_ids = ids.build();
                JsonObject category =
factory.createObjectBuilder().add("name", cat_name).add("id",
cat_id)
                                .add("child_cat_ids", child_ids).build();
                categories.add(category);
            }
            rdr.close();
            return categories;
        }

        /**
         * Returns a JsonArray containing all the venues in a
neighbourhood
         * {@link JsonArray}. The line argument is a String of the URL
to
         * Foursquare's API {@link String}.
         *
         * @param line
         *            the URL to Foursquare's API
         * @return a JsonArray containing all the venues in a
neighbourhood
         * @see String, JsonArray
         */
        public static JsonArray exploreVenues(String line) throws
IOException {

            URL urltag = new URL(line);
            InputStream is = urltag.openStream();

            JsonReader rdr = Json.createReader(is);
            JsonObject obj = rdr.readObject();
            JsonArray items =
obj.getJsonObject("response").getJsonArray("groups").getJsonObject(0
).getJsonArray("items");

            Map<String, Object> config = new HashMap<String,
Object>();
            JsonBuilderFactory factory =
Json.createBuilderFactory(config);
            JsonArrayBuilder venues = factory.createArrayBuilder();

            for (JsonObject res :
items.getValuesAs(JsonObject.class)) {
```

```
                JsonObject ven = res.getJsonObject("venue");
                venues.add(ven);
            }
            JsonArray v = venues.build();
            rdr.close();
            return v;
      }

      /**
       * Returns a JsonObject containing the location of a venue in
GeoJSON
       * format. {@link JsonObject}.
       *
       * @param venue
       *            a JsonObject containing the info of a venue
       * @return a JsonObject containing the GeoJSON data
       * @see JsonObject
       */
      public static JsonObject createGeoJson(JsonObject venue)
throws IOException {

            JsonObject loc = venue.getJsonObject("location");
            Map<String, Object> config = new HashMap<String,
Object>();
            JsonBuilderFactory factory =
Json.createBuilderFactory(config);

            JsonArray coords =
factory.createArrayBuilder().add(loc.getJsonNumber("lng")).add(loc.g
etJsonNumber("lat"))
                        .build();

            JsonObject cat = (JsonObject)
venue.getJsonArray("categories").get(0);

            double rate = 0;
            String colour = "";
            JsonNumber rating = venue.getJsonNumber("rating");
            if (rating != null) {
                rate = rating.doubleValue();
                colour = venue.getString("ratingColor");
            }

            JsonObject geojson =
factory.createObjectBuilder().add("type", "Feature")
                        .add("geometry",
factory.createObjectBuilder().add("type",
"Point").add("coordinates", coords).build())
                        .add("properties",

      factory.createObjectBuilder().add("name",
venue.getString("name"))
                                                .add("category",
cat.getString("name"))
                                                .add("rating", rate)
```

```
                                                .add("ratingColor",
colour)
                                                .add("checkinsCount",
venue.getJsonObject("stats").getJsonNumber("checkinsCount"))
                                                .add("id",
venue.getString("id")).build())
                        .build();

            return geojson;
      }
}
```

A.4 Queries στη MongoDB - MongoClass.java

```
package foursquare;

import java.io.StringReader;
import java.util.HashMap;
import java.util.Map;

import javax.json.Json;
import javax.json.JsonArray;
import javax.json.JsonArrayBuilder;
import javax.json.JsonBuilderFactory;
import javax.json.JsonObject;
import javax.json.JsonReader;

import org.bson.Document;

import com.mongodb.MongoClient;
import com.mongodb.client.FindIterable;
import com.mongodb.client.MongoCollection;
import com.mongodb.client.MongoDatabase;
import com.mongodb.client.model.UpdateOptions;
import com.mongodb.Block;

public class MongoClass {

      /**
       * Returns true if a city already exists in the MongoDB. The
name argument
       * is a String representing the name of a city/collection. If
the collection
       * is empty, the method returns false.
       *
       * @param name
       *            the name of a MongoDB collection
       * @return a boolean
       * @see String, boolean
       */
      public static boolean findArea(String name) {

            boolean exists = false;
            MongoClient mongo = new MongoClient("localhost", 27017);
```

```
            MongoDatabase db = mongo.getDatabase("foursquare");
            MongoCollection<Document> coll = db.getCollection(name);
            if (coll.count() != 0)
                  exists = true;
            mongo.close();
            return exists;
      }

      /**
       * Saves a JsonArray containing all the neighbourhoods of a
city in the
       * collection areaname of the MongoDB {@link JsonArray}. The
areaname
       * argument is a String representing the name of the city. The
hoods
       * argument is a JsonArray containing all the neighbourhoods.
       *
       * @param areaname
       *            the name of the city
       * @param hoods
       *            a JsonArray containing all the neighbourhoods
       * @return void
       * @see String, JsonArray
       */
      public static void saveNeighbourhoods(String areaname,
JsonObject hoods) {

            MongoClient mongo = new MongoClient("localhost", 27017);
            MongoDatabase db = mongo.getDatabase("foursquare");
            MongoCollection<Document> coll =
db.getCollection(areaname);
            JsonArray features = hoods.getJsonArray("features");
            int i = 0;
            while (i < features.size()) {
                  JsonObject hood = features.getJsonObject(i);
                  Document myDoc = Document.parse(hood.toString());
                  coll.insertOne(myDoc);
                  i++;
            }
            mongo.close();
      }

      /**
       * Returns a JsonArray containing all the neighbourhoods in a
US City
       * according to the Neighbouhoods Project {@link JsonArray}.
The areaname
       * argument is a String representing the name of the city
{@link String}.
       *
       * @param areaname
       *            the name of the city
       * @return a JsonArray containing all the neighbourhoods
       * @see String, JsonArray
       */
      public static JsonArray findHoods(String areaname) {
```

```
            Map<String, Object> config = new HashMap<String,
Object>();
            JsonBuilderFactory factory =
Json.createBuilderFactory(config);
            JsonArrayBuilder hoods = factory.createArrayBuilder();

            MongoClient mongo = new MongoClient("localhost", 27017);
            MongoDatabase db = mongo.getDatabase("foursquare");
            MongoCollection<Document> coll =
db.getCollection(areaname);

            FindIterable<Document> iterable = coll.find();
            iterable.forEach(new Block<Document>() {
                  @Override
                  public void apply(final Document document) {
                        String hood = document.toJson();

                        JsonReader jsonReader = Json.createReader(new
StringReader(hood));
                        JsonObject neighbourhood =
jsonReader.readObject();
                        jsonReader.close();

                        hoods.add(neighbourhood);
                  }
            });
            JsonArray neighbourhoods = hoods.build();
            mongo.close();
            return neighbourhoods;
      }

      /**
       * Saves the JsonObject containing a Foursquare venue in the
collection
       * catname of the MongoDB {@link JsonObject}. The catname
argument is a
       * String representing a venue category name and also a
collection in
       * MongoDB {@link String}. The venue argument is a JsonObject
containing a
       * Foursquare venue {@link JsonObject}. The geojson argument
is a JsonObject
       * containing the venue's location in GeoJSON format {@link
JsonObject}. The
       * hoodname argument is a String representing a
neighbourhood's name
       * {@link String}. The areaname argument is a String
representing the name
       * of a city {@link String}.
       *
       * @param catname
       *            a venue category name and also a collection in
MongoDB
       * @param venue
       *            a JsonObject containing a Foursquare venue
```

```
        * @param geojson
        *            a JsonObject containing the venue's location in
GeoJSON format
        * @param hoodname
        *            the neighbourhood's name
        * @param areaname
        *            the name of the city
        * @return void
        * @see String, JsonObject
        */
      public static void saveVenue(String catname, JsonObject venue,
JsonObject geojson, String hoodname,
                  String areaname) {

            MongoClient mongo = new MongoClient("localhost", 27017);
            MongoDatabase db = mongo.getDatabase("foursquare");
            MongoCollection<Document> coll =
db.getCollection(catname);

            Document myDoc = Document.parse(venue.toString());
            Document myDoc2 = Document.parse(geojson.toString());
            coll.updateOne(new Document("name",
venue.getString("name")),
                        new Document("$set", new
Document(myDoc).append("geojson", myDoc2)), new
UpdateOptions().upsert(true));
            mongo.close();
      }

      /**
       * Returns a long number of the venues found in a collection
{@link long}.
       *
       * @param hood
       *            a JsonObject representing a neighbourhood
       * @param catname
       *            the name of the category (collection)
       * @return a long number of the venues found
       * @see long, String, JsonObject
       */
      public static long getVenues(JsonObject hood, String catname)
{

            JsonObject geometry = hood.getJsonObject("geometry");
            Document myDoc = Document.parse(geometry.toString());
            MongoClient mongo = new MongoClient("localhost", 27017);
            MongoDatabase db = mongo.getDatabase("foursquare");
            MongoCollection<Document> coll =
db.getCollection(catname);

            long count = coll
                        .count(new Document("geojson.geometry", new
Document("$geoWithin", new Document("$geometry", myDoc))));
            mongo.close();
            return count;
      }
```

```
	/**
	 * Updates the Document of a neighbourhood with the number of
venues found
	 * in a category. {@link Document}
	 *
	 * @param areaname
	 *            the name of the city
	 * @param hood
	 *            a JsonObject representing a neighbourhood
	 * @param catname
	 *            the name of the category
	 * @return void
	 * @see String, JsonObject
	 */
	public static void updateHood(String areaname, JsonObject
hood, String catname, long count) {

		String cat = catname.replace(" ", "");
		String field = "properties.venues." + cat.replace("&",
"_");
		MongoClient mongo = new MongoClient("localhost", 27017);
		MongoDatabase db = mongo.getDatabase("foursquare");
		MongoCollection<Document> coll =
db.getCollection(areaname);

		coll.updateOne(new Document("properties.name",
hood.getJsonObject("properties").getString("name")),
				new Document("$set", new Document(field,
count)), new UpdateOptions().upsert(true));

		mongo.close();
	}
}
```

Παράρτημα Β
Πηγαίος Κώδικας Server

B.1 Routing & Queries - index.js

```
var express = require('express');
var router = express.Router();

// Mongoose import
var mongoose = require('mongoose');

// Mongoose connection to MongoDB
var conn =
mongoose.createConnection('mongodb://localhost:27017/foursquare',
function (error) {
    if (error) {
        console.log(error);
    }
});
var conn2 =
mongoose.createConnection('mongodb://localhost:27017/fire', function
(error) {
    if (error) {
        console.log(error);
    }
});

// Mongoose Schema definition
var Schema = mongoose.Schema;

var CitySchema = new Schema({
    type: String,
    geometry: Schema.Types.Mixed,
    properties: { name: String, population: Number, centroid: Array,
radius: Array }
});

var VenueSchema = new Schema({
  name: String,
  geojson: { name: String, geometry: Schema.Types.Mixed }
});

/* GET Collection names. */
router.get('/listCollections', function(req,res) {
    conn.db.listCollections().toArray(function(err, names) {
        if (err) {
            console.log(err);
        }
        else {
            res.json(names);
```

```
        };
    });
});

router.get('/fire/:city/:hood', function(req,res) {
    var City = conn.model('City', CitySchema, req.params.city);
    City.findOne({"properties.name": req.params.hood},{"_id": 0},
function (err, docs) {
        var City2 = conn2.model('City', CitySchema,
req.params.city);
        City2.find({"geometry": {$geoWithin: {$geometry:
docs.geometry}}}, {"_id": 0}, function (err2, docs2) {
            res.json(docs2);
        });
    });
});

/* GET City data. */
router.get('/hoods/:city', function (req, res) {
    var City = conn.model('City', CitySchema, req.params.city);
    City.find({},{"_id": 0}, function (err, docs) {
        if (docs == "") {
            var msg = runJava(req);
            res.send(msg);
        }
        else {
            res.json(docs);
        };
    });
});

/* GET Venue data. */
router.get('/venues/:city/:hood/:category', function (req, res) {
    var City = conn.model('City', CitySchema, req.params.city);
    var Venue = conn.model('Venue', VenueSchema,
req.params.category);
    City.findOne({"properties.name": req.params.hood},{"_id": 0},
function (err, docs) {
        Venue.find({"geojson.geometry": {$geoWithin: {$geometry:
docs.geometry}}}, {"_id": 0, "geojson": 1}, function (err2, docs2) {
            res.json(docs2);
        });
    });
});

/* GET Map page. */
router.get('/map/:city/:category', function(req,res) {
    var cat = req.params.category;
    var str = cat.replace(/ /g, "");
    var fld = str.replace(/&/g, "_");
    var City = conn.model('City', CitySchema, req.params.city);
    City.findOne({},{"_id": 0}, function (err, docs) {
        if (docs == null) {
            var msg = runJava(req);
            res.send(msg);
        } else {
```

```
            var c1 = docs.geometry.coordinates[0];
            var c2 = c1[0];
            res.render('map', {
                city: req.params.city,
                category: req.params.category,
                catfield: fld,
                lat : c2[1],
                lng : c2[0]
            });
        };
    });
});

/* GET Home page. */
router.get('/map', function(req,res) {
    var City = conn.model('City', CitySchema, 'Brooklyn');
    City.findOne({},{"_id": 0}, function (err, docs) {
        var c1 = docs.geometry.coordinates[0];
        var c2 = c1[0];
        res.render('map', {
            city: 'Brooklyn',
            category: 'Food',
            catfield: 'Food',
            lat : c2[1],
            lng : c2[0]
        });
    });
});

router.get('/fire', function(req,res) {
    var City = conn.model('City', CitySchema, 'Brooklyn');
    res.render('fire', {
        city: 'Brooklyn'
    });
});

function runJava(req) {
    var spawn = require('child_process').spawn;
    var frsq = spawn('java', ['-jar', './forsq.jar',
req.params.city]);
    frsq.stdout.on('data', (data) => {
        console.log(`stdout: ${data}`);
    });
    frsq.stderr.on('data', (data) => {
        console.log(`stderr: ${data}`);
    });
    frsq.on('close', (code) => {
        console.log(`child process exited with code ${code}`);
    });
    var msg = 'City not found in Database. Executing java
application. Refresh page after 2 mins.';
    return msg;
};

module.exports = router;
```

B.1 Map Template - map.jade

```
doctype html
html
  head
    title Foursquare Venues
    meta(charset='utf-8')
    meta(name='viewport', content='width=device-width, initial-
scale=1.0')
    link(rel='stylesheet',
href='http://cdn.leafletjs.com/leaflet/v0.7.7/leaflet.css')
    link(rel='stylesheet',
href='https://dl.dropboxusercontent.com/u/28290565/leaflet_search/le
aflet-search.css')
    style.
      body {
      margin: 0;
      padding: 0;
      }
      #map {
      position: absolute;
      top: 0;
      bottom: 0;
      width: 100%;
      }
      .info {
      padding: 6px 8px;
      font: 14px/16px Arial, Helvetica, sans-serif;
      background: white;
      background: rgba(255, 255, 255, 0.8);
      box-shadow: 0 0 15px rgba(0, 0, 0, 0.2);
      border-radius: 5px;
      }
      div#radio{
      padding: 6px 8px;
      font: 14px/16px Arial, Helvetica, sans-serif;
      background: white;
      background: rgba(255, 255, 255, 0.8);
      box-shadow: 0 0 15px rgba(0, 0, 0, 0.2);
      border-radius: 5px;
      }
      #nav {
      float: right;
      }
      .info h4 {
      margin: 0 0 5px;
      color: #777;
      }
      .legend {
      text-align: left;
      line-height: 18px;
      color: #555;
      }
```

```
      .legend i {
      width: 18px;
      height: 18px;
      float: left;
      margin-right: 8px;
      opacity: 0.7;
      }
  body
    #map
    script(src='http://cdn.leafletjs.com/leaflet/v0.7.7/leaflet.js')

script(src='https://dl.dropboxusercontent.com/u/28290565/leaflet_search/leaflet-search.js')

script(src='https://ajax.googleapis.com/ajax/libs/jquery/1.11.1/jquery.min.js')

script(src='https://ajax.googleapis.com/ajax/libs/jqueryui/1.10.4/jquery-ui.min.js')
    script(type='text/javascript').
      var map = L.map('map').setView([#{lat},#{lng}], 12);

L.tileLayer('https://api.tiles.mapbox.com/v4/{id}/{z}/{x}/{y}.png?access_token=pk.eyJ1IjoibGFtYmVyYmFzcyIsImEiOiJjaWh0amt1ZTAwMDJ6d3NrczliemFtbWk0In0.fDZtBff9qLcyNwJb4pYs7A',
      {
      maxZoom : 18,
      attribution : 'Map data &copy; <a href="http://openstreetmap.org">OpenStreetMap</a> contributors, '
      + '<a href="http://creativecommons.org/licenses/by-sa/2.0/">CC-BY-SA</a>, '
      + 'Imagery © <a href="http://mapbox.com">Mapbox</a>',
      id : 'mapbox.light'
      }).addTo(map);
      // control that shows state info on hover
      var info = L.control();
      info.onAdd = function(map) {
      this._div = L.DomUtil.create('div', 'info');
      this.update();
      return this._div;
      };
      info.update = function(props) {
      this._div.innerHTML = '<h4>#{category} in #{city}</h4>'
      + (props ? '<b>' + props.name + '</b><br />' +
props.venues.#{catfield}
      + ' #{category} venues ' : 'Hover over a neighbourhood');
      };
      info.addTo(map);
      // get color depending on number of venues
      function getColor(d) {
      return d > 70 ? '#016450' : d > 60 ? '#02818a'
      : d > 50 ? '#3690c0' : d > 40 ? '#67a9cf'
      : d > 30 ? '#a6bddb' : d > 20 ? '#d0d1e6'
      : d > 10 ? '#ece2f0' : '#fff7fb';
      }
      function style(feature) {
```

```
        return {
        weight : 2,
        opacity : 1,
        color : 'white',
        dashArray : '3',
        fillOpacity : 0.7,
        fillColor : getColor(feature.properties.venues.#{catfield})
        };
        }
        function highlightFeature(e) {
        var layer = e.target;
        layer.setStyle({
        weight : 5,
        color : '#666',
        dashArray : '',
        fillOpacity : 0.7
        });
        if (!L.Browser.ie && !L.Browser.opera) {
        layer.bringToFront();
        }
        info.update(layer.feature.properties);
        }
        var geojson;
        function resetHighlight(e) {
        geojson.resetStyle(e.target);
        info.update();
        }
        function zoomToFeature(e) {
        map.fitBounds(e.target.getBounds());
        var str = '/venues/' + '#{city}' + '/' +
e.target.feature.properties.name + '/' + '#{category}';
        var venues = str.replace(/&amp;/g, "&");
        $.getJSON(venues,function(result){
        var str = JSON.stringify(result);
        var r1 = str.replace(/}}}/g, "}}");
        var r2 = r1.replace(/geojson/g, "");
        var r3 = r2.replace(/"":{/g, "");
        var obj = JSON.parse(r3);
        L.geoJson(obj, {
        onEachFeature: onEachFeature2,
        pointToLayer: function (feature, latlng) {
        return L.marker(latlng);
        }
        }).addTo(map);
        });
        }
        function onEachFeature(feature, layer) {
        layer.on({
        mouseover : highlightFeature,
        mouseout : resetHighlight,
        click : zoomToFeature
        });
        }
        function onEachFeature2(feature, layer) {
        var popupContent = '<div id="nav"
style="color:#'+feature.properties.ratingColor+'"><b>'+feature.prope
```

```
rties.rating+'</b></div> <div id="title"> <a
href="https://foursquare.com/v/'+feature.properties.id+'"class="venu
e"><b>'+feature.properties.name+'</b></a></div> <div
id="venue_info">'+feature.properties.category+'<br>'+feature.propert
ies.checkinsCount+' check-ins </div>';
      layer.bindPopup(popupContent);
      }
      $.getJSON('/hoods/' + '#{city}', function(result){
      geojson = L.geoJson(result, {
      style : style,
      onEachFeature : onEachFeature
      }).addTo(map);
      map.fitBounds(geojson.getBounds());
      });
      map.attributionControl
      .addAttribution('Neighbourhood data &copy; <a
href="http://zetashapes.com/">The Neighbourhoods Project</a>');
      var legend = L.control({
      position : 'bottomright'
      });
      legend.onAdd = function(map) {
      var div = L.DomUtil.create('div', 'info legend'), grades = [
0,
      10, 20, 30, 40, 50, 60, 70 ], labels = [], from, to;
      for (var i = 0; i < grades.length; i++) {
      from = grades[i];
      to = grades[i + 1];
      labels.push('<i style="background:' + getColor(from + 1)
      + '"></i> ' + from + (to ? '&ndash;' + to : '+'));
      }
      div.innerHTML = labels.join('<br>');
      return div;
      };
      legend.addTo(map);
      var radioControl = L.control({
      position : 'bottomleft'
      });
      radioControl.onAdd = function (map) {
      var radio = L.DomUtil.create('radio', 'radio-control');
      radio.innerHTML += '<div id="radio"> <input type="radio"
name="category" value="Food" checked> Food<br>  <input type="radio"
name="category" value="Arts & Entertainment"> Arts &
Entertainment<br>  <input type="radio" name="category"
value="Nightlife Spot"> Nightlife Spot<br>  <input type="radio"
name="category" value="Shop & Service"> Shop & Service<br>  <input
type="radio" name="category" value="Outdoors & Recreation"> Outdoors
& Recreation<br>  <input type="radio" name="category" value="Travel
& Transport"> Travel & Transport<br>  <input type="radio"
name="category" value="Residence"> Residence<br>  <input
type="radio" name="category" value="Event"> Event<br>  <input
type="radio" name="category" value="Professional & Other Places">
Professional & Other Places<br>  <input type="radio" name="category"
value="College & University"> College & University </div>';
      return radio;
      };
      radioControl.addTo(map);
```

```
      var searchLayer = new L.LayerGroup();
      map.addLayer(searchLayer);
      var controlSearch = new L.Control.Search({layer: searchLayer,
initial: false, circleLocation: false, position:'topleft'});
      $.getJSON('/listCollections', function(result){
      $.each(result, function(index, value) {
      if (value.name != "Professional & Other Places" && value.name
!= "Nightlife Spot" && value.name != "College & University" &&
value.name != "Outdoors & Recreation" && value.name != "Arts &
Entertainment" && value.name != "Food" && value.name != "Travel &
Transport" && value.name != "Residence" && value.name != "Shop &
Service"  && value.name != "Event") {
      var title = value.name,
      loc = [41.239190,13.032145], //random loc
      marker = new L.Marker(new L.latLng(loc), {title: title} );
      searchLayer.addLayer(marker);};
      });
      });
      controlSearch.on('search_locationfound', function(e) {
      var cat = $('input[name="category"]:checked').val();
      location.href = 'http://localhost:3000/map/' +
this._input.value + '/' + cat;
      });
      map.addControl( controlSearch );
```

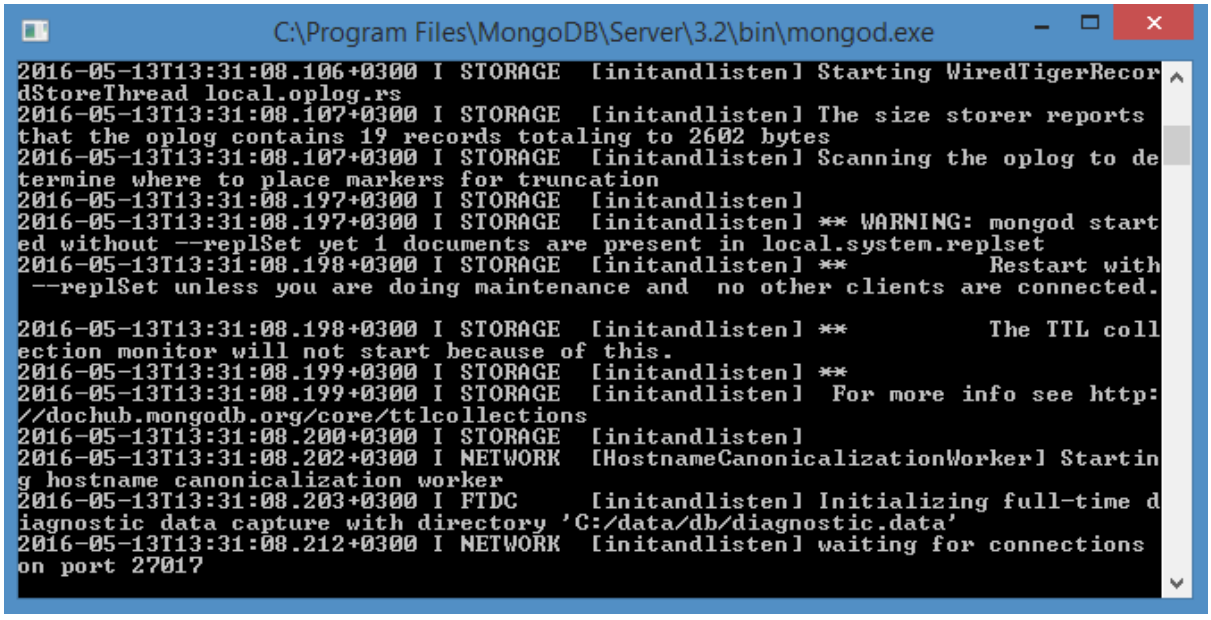
Παράρτημα Γ
Οδηγίες Εγκατάστασης Εφαρμογής

Γ.1 Εγκατάσταση MongoDB

Για τη λειτουργία της εφαρμογής είναι απαραίτητη η ύπαρξη μιας εγκατεστημένης έκδοσης της MongoDB στο σύστημα, ώστε το πρόγραμμα να αποθηκεύει και να ανακτά δεδομένα από τη βάση δεδομένων. Τα βήματα που πρέπει να ακολουθηθούν είναι τα εξής:

- Επίσκεψη της ιστοσελίδας www.mongodb.com, όπου υπάρχουν διαθέσιμα για δωρεάν download εκδόσεις της MongoDB για λειτουργικά συστήματα Windows, Linux, Mac OS X και Solaris.
- Download της επιθυμητής έκδοσης.
- Εκτέλεση του setup αρχείου και εγκατάσταση ακολουθώντας τις οδηγίες του installation wizard.
- Στο directory όπου έγινε η εγκατάσταση, εκτέλεση του προγράμματος mongod. Παράδειγμα ενός directory σε Windows είναι: C:\Program Files\MongoDB\Server\3.2\bin

Να σημειωθεί πως το mongod πρέπει να τρέχει συνεχώς τόσο κάτα την εκτέλεση των services που αποθηκεύουν δεδομένα, όσο και για τη λειτουργία του χάρτη που ανακτά δεδομένα από τη MongoDB. Χρησιμοποιείται το default port 27017 για τη MongoDB.

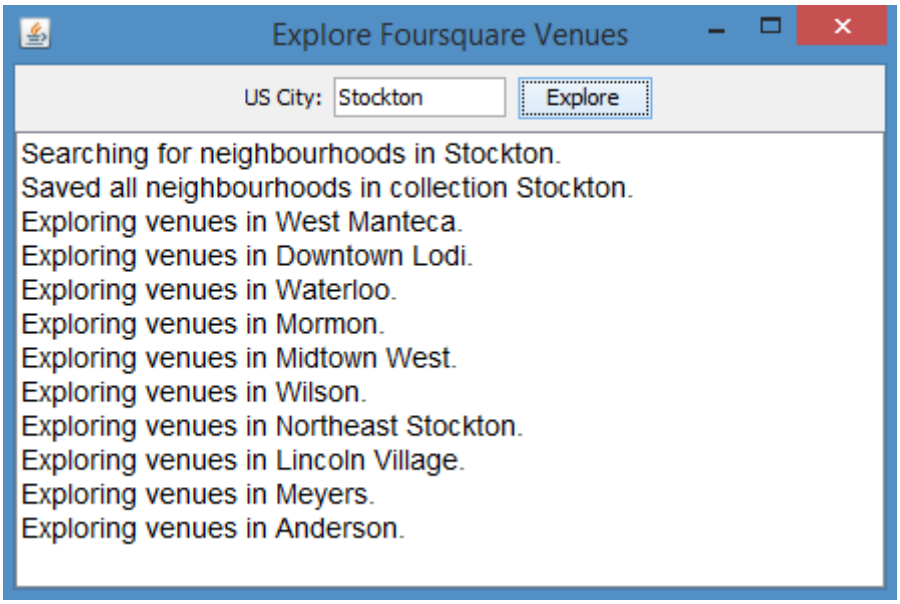


Σχήμα Γ1: Εκτέλεση του mongod.exe σε Windows 8.1 64bit

Γ.2 Εγκατάσταση Java

Τα services της εφαρμογής για την ανάκτηση των γεωγραφικών δεδομένων από το Zetashapes API και την αναζήτηση των venues από το Foursquare API είναι πακεταρισμένα στο εκτελέσιμο αρχείο foursquare.jar, που περιέχει τον κώδικα και τις απαραίτητες βιβλιοθήκες. Για την εκτέλεσή του είναι απαραίτητο να έχει εγκατασταθεί κάποιο Java Runtime Environment. Τα βήματα που πρέπει να ακολουθηθούν είναι τα εξής:

- Επίσκεψη της ιστοσελίδας www.oracle.com, όπου υπάρχουν διαθέσιμα για δωρεάν download εκδόσεις του Java RE για λειτουργικά συστήματα Windows, Linux, Mac OS X και Solaris.
- Download της επιθυμητής έκδοσης.
- Εκτέλεση του setup αρχείου και εγκατάσταση ακολουθώντας τις οδηγίες του installation wizard.
- Εκτέλεση του foursquare.jar. Στο παράθυρο που εμφανίζεται υπάρχει ένα πεδίο κειμένου όπου εισάγεται το όνομα της πόλης για την αναζήτηση γεωγραφικών δεδομένων και Foursquare Venues.



Σχήμα Γ2: Εκτέλεση του foursquare.jar σε Windows 8.1 64bit

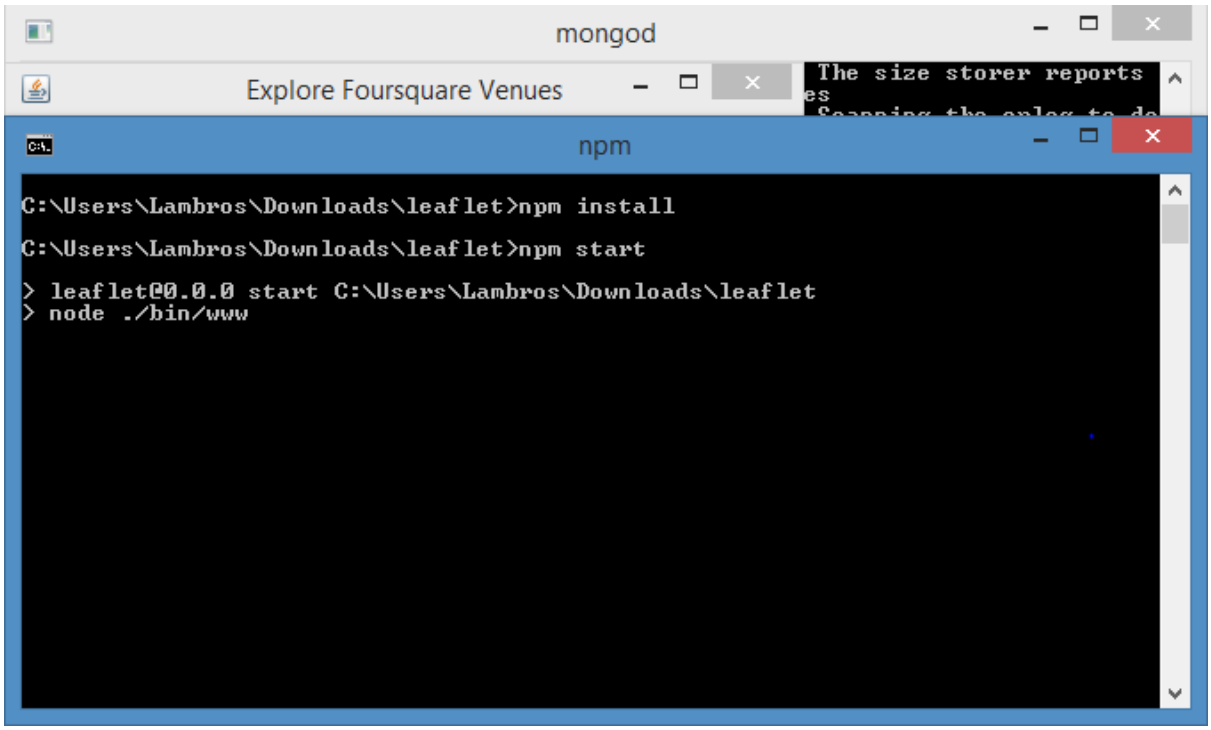
Γ.3 Εγκατάσταση Node.JS

Απαραίτητο για τη λειτουργία του χάρτη είναι η εγκατάσταση του Node.JS, στο οποίο βασίζεται ο server της εφαρμογής. Ο server λειτουργεί ως ένα ενδιάμεσο layer, που λαμβάνει τα requests που γίνονται από το χάρτη, κάνει τα απαραίτητα queries στη βάση δεδομένων και στέλνει τα κατάλληλα δεδομένα. Τα βήματα που πρέπει να ακολουθηθούν για την εγκατάσταση του Node.JS είναι τα εξής:

- Επίσκεψη της ιστοσελίδας www.nodejs.org, όπου υπάρχουν διαθέσιμα για δωρεάν download εκδόσεις του Node.JS για λειτουργικά συστήματα Windows, Linux, Mac OS X και SunOS.

- Download της επιθυμητής έκδοσης.
- Εκτέλεση του setup αρχείου και εγκατάσταση ακολουθώντας τις οδηγίες του installation wizard.
- Άνοιγμα ενός command prompt στο directory όπου είναι αποθηκευμένος ο φάκελος leaflet.
- Εκτέλεση των εντολών:
```
cd leaflet
npm install
npm start
```

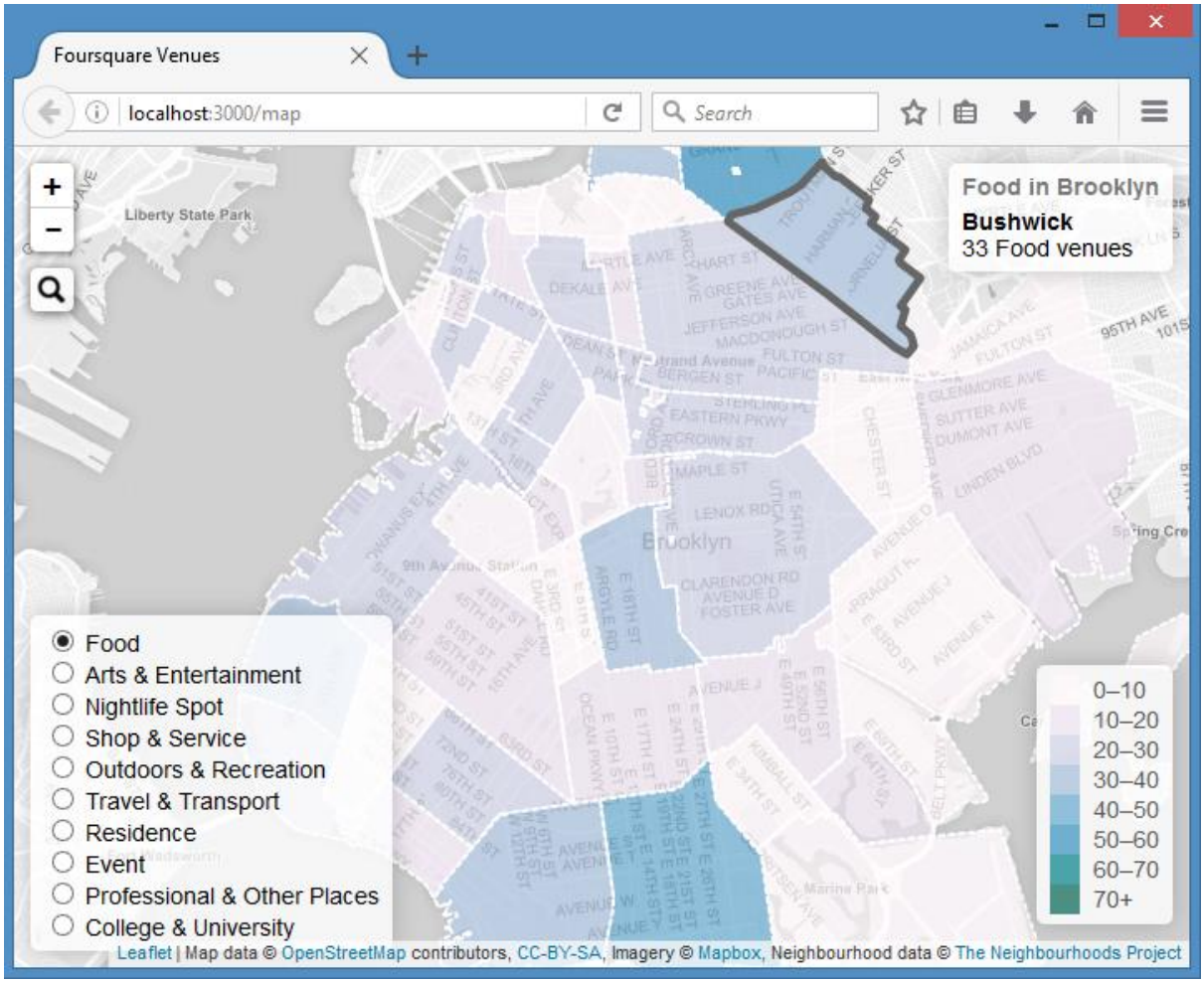
Ο server χρησιμοποεί το port 3000 και πρέπει να τρέχει συνεχώς για τη λειτουργία του χάρτη.



Σχήμα Γ3: Εκκίνηση του server σε Windows 8.1 64bit

Γ.4 Άνοιγμα χάρτη

Για την εμφάνιση του χάρτη σε ένα browser αρκεί η επίσκεψη του URL: http://localhost:3000/map. Σημειώνεται ότι για τη λειτουργία του πρέπει να εκτελούνται το mongod και ο server, όπως περιγράφηκε παραπάνω.



Σχήμα Γ4: Επίσκεψη της σελίδας localhost:3000/map σε Firefox web browser