ΕΛΛΗΝΙΚΟ ΜΕΣΟΓΕΙΑΚΟ ΠΑΝΕΠΙΣΤΗΜΙΟ

Τμήμα Ηλεκτρολόγων Μηχανικών και Μηχανικών Η/Υ

Πρόγραμμα Σπουδών: Μηχανικών Πληροφορικής

Πτυχιακή Εργασία

Τίτλος: Διαδικτυακή υπηρεσία συλλογής και απεικόνισης γεωγραφικών δεδομένων από πολλαπλούς χρήστες (Crowdsourcing Mapping Web Service)

Τσουκαλάς Εμμανουήλ (ΑΜ: ΤΠ 3440)

Επιβλέπων Εκπαιδευτικός : Παναγιωτάκης Σπυρίδων

Επιτροπή Αξιολόγησης : Παναγιωτάκης Σπυρίδων
Μαρκάκης Βαγγέλης
Παχουλάκης Ιωάννης

Ημερομηνία Παρουσίασης : 24/09/2021

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον Κ. Παναγιωτάκη για τη βοήθεια που μου έδωσε κατά τη διάρκεια της συγγραφής της πτυχιακής μου εργασίας και κυρίως για την πάντα άμεση επίλυση αποριών που μου δημιουργούνταν. Επιπλέον, θα ήθελα να ευχαριστήσω τον Βασίλη Παπαβασιλείου για τις τεχνικές συμβουλές που μου πρόσφερε στην ανάπτυξη της εφαρμογής. Τέλος, τον Ορέστη Δραπανιώτη για τις δοκιμές και προτάσεις που πρόσφερε πάνω στην ανάπτυξη της πλατφόρμας.

Abstract

Crowdsourcing Mapping Web Service

Crowdsourcing refers to a distributed problem-solving model in which a crowd of undefined size is engaged in the task of solving a complex problem through an open call. This novel problem-solving model found its way into numerous applications on the web for voting, fund-raising, micro-works and wisdom-of-the-crowd scenarios. On the other hand, the shift of desktop users to mobile platforms in the post-PC era, along with the unique multi-sensing capabilities of modern mobile devices are expected to eventually unfold the full potential of Crowdsourcing. Smartphones offer a great platform for extending and diversifying web-based crowdsourcing applications to a larger contributing crowd, making contribution easier and omni-present. This thesis presents the fundamental concepts behind building a web platform for collecting and displaying crowdsource data.

The purpose of this thesis is to develop a platform in which users after creating an account will be able to create their own maps, their own data categories and will be able to post map pins with data.

To develop such platform, we need a spatial database, a web framework and libraries which can work with spatial data, a mobile friendly frontend and last but not least the infrastructure in which the platform will work on.

We successfully met our goal as we were able to develop a platform where the user can register and authenticate, use a mobile friendly UI utilizing the Progressive Web App (PWA) technology to make the application feeling more like a native mobile application. The user through his smartphone can make a post on a map using the three main media types (photo, video and audio), can make comments on other users posts and can like them and finally has the ability to make his own maps with its own data categories. We present below the steps we took to develop and support our application in the theoretical part of our thesis.

1. The technologies and libraries we used.
2. Code analysis.
3. The observations and conclusions we derived during the development of our thesis.

Σύνοψη

Crowdsourcing Mapping Web Service

Crowdsourcing ή πληθοπορισμός αναφέρεται σε ένα κατανεμημένο μοντέλο επίλυσης προβλημάτων, στο οποίο ένα πλήθος απροσδιόριστου μεγέθους ασχολείται με την επίλυση ενός σύνθετου προβλήματος μέσω ανοιχτής πρόσκλησης. Αυτό το νέο μοντέλο επίλυσης προβλημάτων βρήκε το δρόμο του σε πολλές διαδικτυακές εφαρμογές, από ψηφοφορίες, σε φιλανθρωπικούς σκοπούς για τη συγκέντρωση χρημάτων, σε ανάθεση μικρών κομματιών ενός μεγαλύτερου έργου και σε σενάρια που χρησιμοποιείτε η σοφία του πλήθους. Από την άλλη η αλλαγή των χρηστών από προσωπικούς υπολογιστές σε κινητά τηλέφωνα smartphones, μαζί με τις δυνατότητες που δίνουν οι πολλαπλοί αισθητήρες των smartphone αναμένετε να ξεδιπλώσουν τις πλήρες δυνατότητες του crowdsourcing. Τα smartphones προσφέρουν μια εξαιρετική πλατφόρμα για την επέκταση και διαφοροποίηση των διαδικτυακών crowdsourcing εφαρμογών, κάνοντας τη συνεισφορά ποιο εύκολη και συνεχής. Αυτή η πτυχιακή παρουσιάζει τις βασικές έννοιες για τη δημιουργία μιας διαδικτυακής πλατφόρμας που έχει ως σκοπό την περισυλλογή και την απεικόνιση crowdsource δεδομένων πάνω σε χάρτη.

Σκοπός της πτυχιακής εργασίας είναι η δημιουργία μιας πλατφόρμας στην οποία οι χρήστες αφού έχουν φτιάξει ένα λογαριασμό θα μπορούν να δημιουργήσουν τους δικούς τους χάρτες, τις δικές τους κατηγορίες δεδομένων και θα μπορούν να κάνουν δημοσίευση με τη μορφή πινέζας στο χάρτη πληροφορίες.

Για τη δημιουργία της πλατφόρμας χρειαζόμαστε μια βάση δεδομένων που να μπορεί να χειριστεί γεωγραφικά δεδομένα, ένα web framework που να υποστηρίζει και γεωγραφικά δεδομένα, το γραφικό περιβάλλον να είναι φιλικό ως προς τον χρήστη και να μπορεί να λειτουργήσει και σε smartphones και τέλος την υποδομή των μηχανημάτων για να λειτουργήσει η πλατφόρμα στο διαδίκτυο.

Ο σκοπός ολοκληρώθηκε με επιτυχία καθώς καταφέραμε να δημιουργήσουμε μία πλατφόρμα που ο χρήστης μπορεί να κάνει εγγραφή και authentication, να χρησιμοποιήσει με το smartphone του ένα mobile friendly frontend που χρησιμοποιήθηκε η τεχνολογία Progressive Web App (PWA) για να δώσουμε ένα feeling native mobile εφαρμογής. Ο χρήστης μέσω του smartphone του μπορεί να κάνει δημοσίευση σε κάποιον χάρτη χρησιμοποιώντας τρεις τύπους media (φωτογραφία, βίντεο και ήχο), μπορεί να κάνει σχόλια σε άλλες δημοσιεύσεις και να τις βαθμολογήσει ως χρήσιμες και τέλος έχει τη δυνατότητα να φτιάξει τους δικούς του χάρτες με τα δικά του φίλτρα.

Στο θεωρητικό μέρος της εργασίας αναλύουμε τα εξής βήματα που ακολουθήσαμε για να αναπτύξουμε την πλατφόρμα μας:

1. Τις τεχνολογίες και τις βιβλιοθήκες που χρησιμοποιούμε.
2. Αναλύσεις του κώδικα που αναπτύξαμε και τέλος
3. Τα συμπεράσματα που βγάλαμε κατά τη διάρκεια της ανάπτυξης και συγγραφής της πτυχιακής μας εργασίας.

Πίνακας Περιεχομένων

Πίνακας Εικόνων

1. Crowdsourcing

1.1. Τι είναι το crowdsourcing

Crowdsourcing ή πληθοπορισμός αναφέρεται σε ένα κατανεμημένο μοντέλο επίλυσης προβλημάτων, στο οποίο ένα πλήθος απροσδιόριστου μεγέθους ασχολείται με την επίλυση ενός σύνθετου προβλήματος μέσω ανοιχτής πρόσκλησης. Ο όρος "crowdsourcing" χρησιμοποιήθηκε πρώτη φορά από τον Jeff Howe εκδότη του περιοδικού Wired. Χρησιμοποίησε τον όρο για να περιγράψει, πως οι εταιρίες αναθέτουν εξωτερικές εργασίες στον κόσμο μέσω του διαδικτύου.

Αυτό το νέο μοντέλο επίλυσης προβλημάτων βρήκε το δρόμο του σε πολλές διαδικτυακές εφαρμογές, από διαδικτυακές ψηφοφορίες, σε φιλανθρωπικούς σκοπούς για τη συγκέντρωση χρημάτων, στο τομέα της υγείας και σε μεγάλα και περίπλοκα έργα. Η αλλαγή των χρηστών από προσωπικούς υπολογιστές σε κινητά τηλέφωνα smartphones, μαζί με τις δυνατότητες που δίνουν οι πολλαπλοί αισθητήρες των smartphone αναμένετε να ξεδιπλώσουν τις πλήρες δυνατότητες του crowdsourcing. Τα smartphones προσφέρουν μια εξαιρετική πλατφόρμα για την επέκταση και διαφοροποίηση των διαδικτυακών crowdsourcing εφαρμογών, κάνοντας τη συνεισφορά ποιο εύκολη και συνεχής.

Εικόνα 1 - η εικόνα αυτή συμβολίζει τη χρήση ιδεών από ένα ευρύ φάσμα ατόμων που χρησιμοποιείται στο crowdsourcing

1.2. Γιατί είναι σημαντικό

Το crowdsourcing υιοθετεί τον τρόπο σκέψεις "περισσότερα κεφάλια είναι καλύτερα από ένα". Επομένως, έχουμε περισσότερες ιδέες και δεξιότητες λόγω της συμμετοχής ενός μεγαλύτερου πλήθους κόσμου. Έχοντας περισσότερους ανθρώπους να προσφέρουν τις καλύτερες ιδέες, δεξιότητες και την υποστήριξη τους, μπορούμε να πετύχουμε ένα ποιοτικό αποτέλεσμα. Όσο αφορά το κομμάτι της πληροφορίας, θα έχουμε ποιο ολοκληρωμένη πληροφορία και θα είναι ποιο καλά δομημένη. Το crowdsourcing μας επιτρέπει να επιλέξουμε το καλύτερο αποτέλεσμα όχι από ένα πάροχο αλλά από ένα μεγάλο πλήθος ταλέντων. Επίσης, τα αποτελέσματα μπορούν να παραδοθούν ποιο γρήγορα από τις παραδοσιακές ή τις συμβατικές μεθόδους, δεδομένου ότι είναι μια μορφή freelancing.

Το crowdsourcing επίσης μπορεί να βελτιώσει τη δημιουργικότητα και την παραγωγικότητα. Τα έξοδα που θα μπορούσαμε να χρησιμοποιήσουμε για έρευνα και προσωπικό μειώνονται. Εφόσον η διαδικασία βασίζεται στο διαδίκτυο, η περισυλλογή δεδομένων μέσω στοχευμένων ομάδων και έρευνας μειώνεται δραματικά, επειδή μία κοινότητα παθιασμένων ατόμων που τους ενδιαφέρει το θέμα, είναι πιο πρόθυμοι να συνεισφέρουν τις γνώσεις τους σε μια εργασία crowdsourcing. Οι περισσότεροι συμμετέχοντες το κάνουν για κάποια μικρή αναγνώριση ή για να προσφέρουν στην κοινότητα και σπάνια για κάποιο οικονομικό αντίκτυπο.

1.3. Πού χρησιμοποιείται

Το crowdsourcing τα τελευταία χρόνια έχει ανέβει σε δημοτικότητα. Ενδεχομένως λόγω συνειδητοποίησης των εταιρειών, ότι η σχέση μεταξύ εταιρείας και πελάτη έχει αλλάξει και τώρα ποια η καταναλωτές κατέχουν μεγάλο μέρος του λόγου. Πολλές εταιρείες διάλεξαν αντί να εναντιωθούν να το χρησιμοποιήσουν προς όφελος τους. Παρακάτω θα δούμε μερικούς τρόπους να χρησιμοποιήσουμε το crowdsourcing.

1.4. Περιληπτικά παραδείγματα χρήσης του crowdsource

- **Crowdfunding** είναι ένας από τους ποιο γνωστούς τρόπους crowdsourcing που μπορεί να χρησιμοποιήσει ένα άτομο ή μια εταιρεία. Μπορείτε να μετρήσετε το ενδιαφέρων των πελατών και να προωθήσετε τις προπαραγγελίες προτού ξεκινήσει η ανάπτυξη του προϊόντος.
- **Στον προγραμματισμό** παρόμοια με διαγωνισμούς σχεδίασης, μέσω του crowdsourcing επιτρέπεται σε πολλά άτομα να ανταγωνιστούν για να προσπαθήσουν να λύσουν ένα πρόβλημα, η να σχεδιάσουν μια εφαρμογή για μια εταιρεία.
- **Στη συλλογή δεδομένων** μπορούμε να χρησιμοποιήσουμε το crowdsourcing για να συλλέξουμε πολλά δεδομένα με μικρό κόστος. Αντί να χρησιμοποιούμε ολόκληρα τμήματα για να συλλέξουν πληροφορίες, χρησιμοποιούμε τον κόσμο με μικρά ποσά για να συλλέξουν την πληροφορία.
- **Αναζήτηση Γνώμης** ένας κλασικός τρόπος να χρησιμοποιήσουμε το crowdsourcing για να ρωτήσουμε τη γνώμη του κόσμου ώστε να βελτιώσουμε το προϊόν ή μια υπηρεσία. Μέσω αυτής της προσέγγισης μπορούμε να φτιάξουμε προϊόντα και προσφορές που θα έχουν καλύτερη κατανάλωση από το κόσμο.

1.5. Στόχος πτυχιακής εργασίας

Ο στόχος της πτυχιακής εργασίας είναι η δημιουργία μιας πλατφόρμας συλλογής και απεικόνισης γεωγραφικών δεδομένων από πολλαπλούς χρήστες. Ποιο αναλυτικά ο χρήστης αφού φτιάξει ένα λογαριασμό θα έχει τη δυνατότητα να φτιάξει τους δικούς του χάρτες που ο κάθε χάρτης θα αναπαριστά μια ομαδοποίηση δεδομένων και θα μπορεί να δημοσιεύσει πληροφορίες πάνω στο χάρτη με τη μορφή πινέζας. Η πλατφόρμα αυτή θα είναι διαδικτυακή και ο χρήστης θα μπορεί να τη χρησιμοποιήσει μέσω του web browser.

1.6. Δομή Πτυχιακής Εργασίας

Στην αναφορά της πτυχιακής εργασίας ο αναγνώστης θα συναντήσει μια πληθώρα κεφαλαίων. Πιο συγκεκριμένα ξεκινώντας από το κεφάλαιο 2 θα αναλύσουμε τα βασικά χαρακτηριστικά μιας διαδικτυακής εφαρμογής. Θα μιλήσουμε για τις βάσεις δεδομένων και ποιο συγκεκριμένα για βάσεις δεδομένων που διαχειρίζονται γεωγραφικά δεδομένα, το web framework, με ποιο τρόπο θα μεταφέρουμε την πληροφορία και τον τρόπο που ο χρήστης θα χρησιμοποιεί την πλατφόρμα και τέλος ποιες τεχνικές υπάρχουν για να λειτουργήσουμε μια εφαρμογή στο διαδίκτυο.

Προχωρώντας στο κεφάλαιο 3 θα αναλύσουμε το GIS θα μιλήσουμε για το τί είναι το GIS και το WebGIS, θα αναλύσουμε τεχνολογίες Geolocation, για τρόπους αναπαράστασης γεωγραφικών δεδομένων όπως το GeoJSON και για γεωγραφικές βάσης δεδομένων.

Στο κεφάλαιο 4 γίνεται λεπτομερής αναφορά στις τεχνολογίες που χρησιμοποιήθηκαν για την υλοποίηση της πλατφόρμας οι οποίες αποτελούνται, από την PostgreSQL μαζί με το extension PostGIS, το Django Web Framework μαζί με το Django Rest Framework και τέλος θα μιλήσουμε αναλυτικά για το deployment της εφαρμογής στο διαδίκτυο.

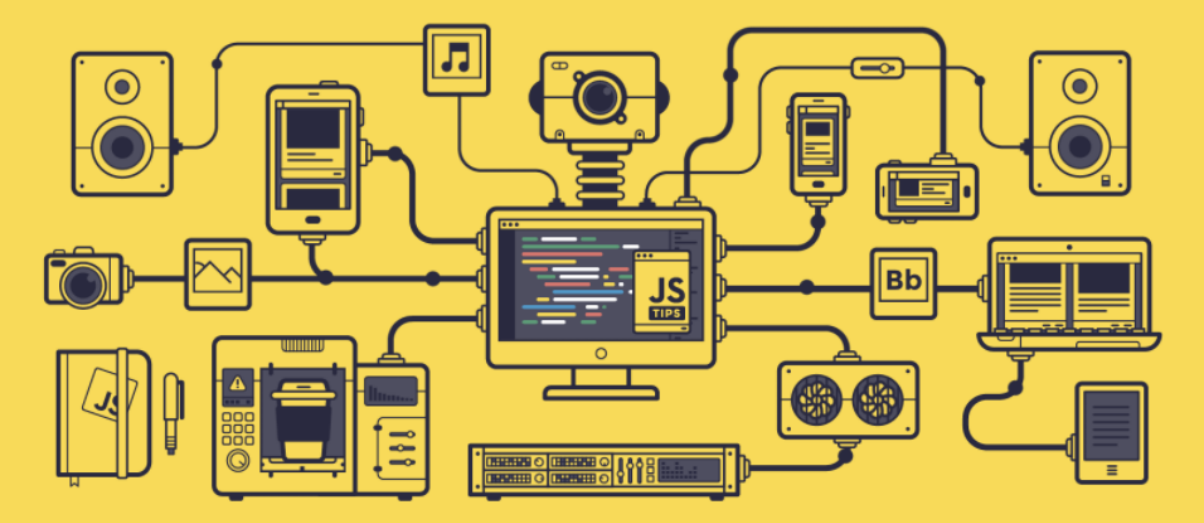
Στο κεφάλαιο 5 θα δούμε σε επίπεδο κώδικα πως όλες αυτές οι τεχνολογίες που έχουν προαναφερθεί στο προηγούμενο κεφάλαιο συνδυάζονται μεταξύ τους με σκοπό την ανάπτυξη της πλατφόρμας.

Αφότου έχουμε πλέον μιλήσει τόσο για την αρχιτεκτονική, έχουμε αναλύσει τις τεχνολογίες και έχουμε δει τον κώδικα της πλατφόρμας φτάνουμε στο κεφάλαιο 6 όπου μπορούμε να μάθουμε περισσότερα για τα σενάρια λειτουργίας τα οποία μπορεί να πραγματοποιήσει πλέον το ολοκληρωμένο πλέον σύστημα μας.

Τέλος, στο κεφάλαιο 7 γίνεται η ανακεφαλαίωση όπου και παραθέτονται τα συμπεράσματα τα οποία βγήκαν από τη δημιουργία της πλατφόρμας καθώς και μελλοντικές επεκτάσεις οι οποίες θα μπορούσαν να βελτιώσουν τη χρησιμότητα και τις δυνατότητες της πλατφόρμας.

2. Ανάπτυξη διαδικτυακών εφαρμογών

2.1 Web framework



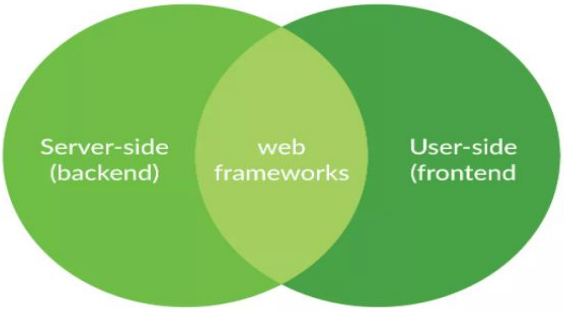
Εικόνα 2 - Web framework

2.1.1 Τι είναι τα web application frameworks

Ένα web framework (WF) ή αλλιώς web application framework (WAF) είναι ένα software framework που έχει σχεδιαστεί για να υποστηρίζει την ανάπτυξη διαδικτυακών εφαρμογών, συμπεριλαμβανομένων web services, web resources και web APIs. Τα web frameworks παρέχουν έναν τυπικό τρόπο δημιουργίας και ανάπτυξης διαδικτυακών εφαρμογών. Έχουν ως σκοπό την αυτοματοποίηση βημάτων που σχετίζονται με κοινές δραστηριότητες που εκτελούνται κατά τη διάρκεια της ανάπτυξης αυτών. Για παράδειγμα, πολλά web frameworks παρέχουν βιβλιοθήκες για πρόσβαση στη βάση δεδομένων, templating frameworks, session management και προωθούν την επαναχρησιμοποίηση κώδικα. Παρόλο που στοχεύουν συχνά στην ανάπτυξη δυναμικών ιστότοπων, ισχύουν επίσης και για στατικούς.

2.1.2 Τύποι web framework

Υπάρχουν δύο κύριες λειτουργίες των web frameworks, η λειτουργία στην πλευρά του server (backend) και η λειτουργία στην πλευρά του χρήστη (frontend) αναλόγως τον τύπο. Τα frontend frameworks ασχολούνται κυρίως με τις εξωτερικές λειτουργίες ενός web application, δηλαδή το τι βλέπει ένας χρήστης όταν ανοίγει την εφαρμογή και το backend είναι για τις εσωτερικές λειτουργίες του web framework.



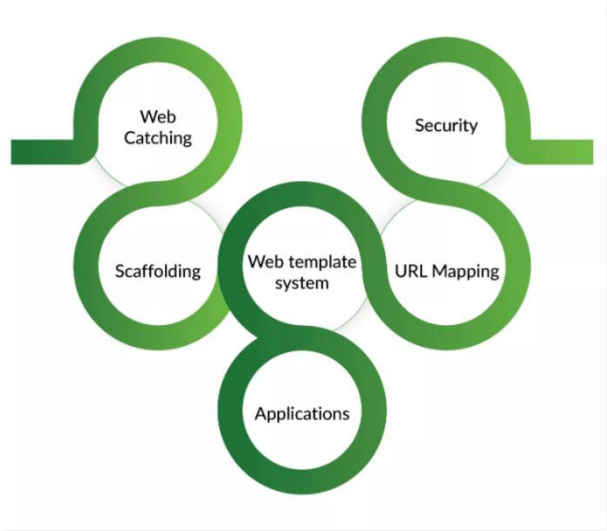
Εικόνα 3 - Τύποι web frameworks

- Server-side frameworks. Οι κανόνες και η αρχιτεκτονική αυτών των framework μας επιτρέπει να δημιουργήσουμε απλές σελίδες και φόρμες συλλογής στοιχείων διαφόρων τύπου. Ωστόσο, για να δημιουργήσουμε μια διαδικτυακή εφαρμογή με μία καλά αναπτυγμένη διεπαφή, θα πρέπει να έχουμε ένα μεγαλύτερο εύρος από λειτουργικότητες. Τα server-side frameworks διαμορφώνουν τα δεδομένα που βγαίνουν προς τα έξω και βελτιώνουν την ασφάλεια σε περίπτωση επιθέσεων. Οι παραπάνω επιλογές μας δίνουν τη δυνατότητα να απλοποιήσουμε τη διαδικασία ανάπτυξης μιας διαδικτυακής εφαρμογής. Παρακάτω μερικά από τα πιο δημοφιλής backend frameworks με τη γλώσσα που έχουν υλοποιηθεί:
    - Django – Python
    - Laravel – PHP
    - Express.js – Javascript
    - Ruby on Rails – Ruby
- Client-side frameworks. Σε αντίθεση με τα server-side frameworks τα client-side δεν έχουν καμία σχέση με το business logic. Όλοι η δουλειά που προσφέρουν πραγματοποιείται μέσα στο browser. Έτσι, μπορεί κανείς να βελτιώσει και να εφαρμόσει νέες διεπαφές χρήστη. Έχουμε τη δυνατότητα να δημιουργήσουμε διάφορα animations, σχέδια και SPA (single-page applications). Κάθε ένα από τα client-side frameworks διαφέρουν ως προς τη λειτουργία και τη χρήση. Παρακάτω θα δούμε μερικά από τα πιο δημοφιλή:
    - Backbone
    - Angular
    - Ember.js
    - Vue.js

Τα παραπάνω frameworks είναι υλοποιημένα στη Javascript.

2.1.3 Χαρακτηριστικά

Παρακάτω θα δούμε μερικά από τα πιο κοινά χαρακτηριστικά που περιέχουν τα web frameworks ώστε να βοηθήσουν τους προγραμματιστές στην ανάπτυξη νέων εφαρμογών.



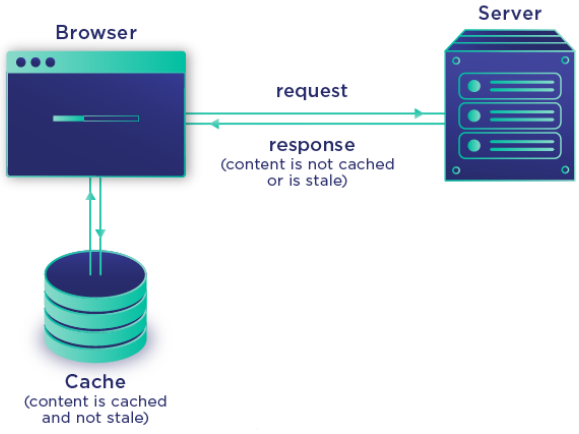
Εικόνα 4 - Web framework features

- Web Caching:

  Η προσωρινή μνήμη είναι ένα στοιχείο λογισμικού ή υλικού που χρησιμοποιείται για την προσωρινή αποθήκευση τιμών για ταχύτερη μελλοντική πρόσβαση. Είναι μια μικρή βάση δεδομένων αρχείων που περιέχει ληφθέντες πόρους, όπως εικόνες, βίντεο, CSS και Javascript. Η βασική ιδέα πίσω από αυτό είναι η ακόλουθη.

  Το πρόγραμμα περιήγησης ζητά κάποιο περιεχόμενο από τον web server. Εάν το περιεχόμενο δε βρίσκεται στην προσωρινή μνήμη του προγράμματος περιήγησης, ανακτάται απευθείας από τον διακομιστή ιστού. Εάν το περιεχόμενο είχε προσωρινά αποθηκευτεί, το πρόγραμμα περιήγησης παρακάμπτει τον διακομιστή και φορτώνει το περιεχόμενο απευθείας από την προσωρινή μνήμη του.

  Το περιεχόμενο θεωρείται παλιό, ανάλογα με το αν το προσωρινά αποθηκευμένο περιεχόμενο έχει λήξει ή όχι. Το φρέσκο, από την πλευρά, σημαίνει ότι το περιεχόμενο δεν έχει περάσει από την ημερομηνία λήξης του και μπορεί να προβληθεί απευθείας από την προσωρινή μνήμη του προγράμματος περιήγησης χωρίς τη συμμετοχή του διακομιστή.



Εικόνα 5 - Web Caching

- Scaffolding:

  Αυτή είναι μια άλλη σημαντική τεχνική που πρέπει να γνωρίζετε και να χρησιμοποιήσετε η οποία υποστηρίζεται από ορισμένα MVC frameworks. Τυπικά μέρη μιας εφαρμογής ή ολόκληρη η δομή του έργου μπορούν να δημιουργηθούν αυτόματα από το framework. Αυτή η προσέγγιση αυξάνει την ταχύτητα του κύκλου ανάπτυξης και τυποποιεί το codebase.

- Web template system:

  Ένα web template σύστημα επιτρέπει στους σχεδιαστές ιστοσελίδων και στους προγραμματιστές να συνεργάζονται με web templates για να δημιουργούν αυτόματα προσαρμοσμένες ιστοσελίδες, όπως τα αποτελέσματα μιας αναζήτησης. Αυτό μας επιτρέπει να επαναχρησιμοποιούμε τα στατικά στοιχεία μιας ιστοσελίδας ενώ μπορούμε να καθορίσουμε δυναμικά στοιχεία βάσει των παραμέτρων ενός web request.

- Security

  Η διαδικτυακή ασφάλεια έχει πολλά κριτήρια για τον προσδιορισμό και την άδεια ή την απόρριψη της πρόσβασης σε διαφορετικές λειτουργίες σε ένα web framework. Βοηθά επίσης στην αναγνώριση των προφίλ που χρησιμοποιούν την εφαρμογή για να αποφευχθούν τα clickjacking. Ως αποτέλεσμα, το ίδιο το web framework είναι αυθεντικό και εξουσιοδοτημένο.

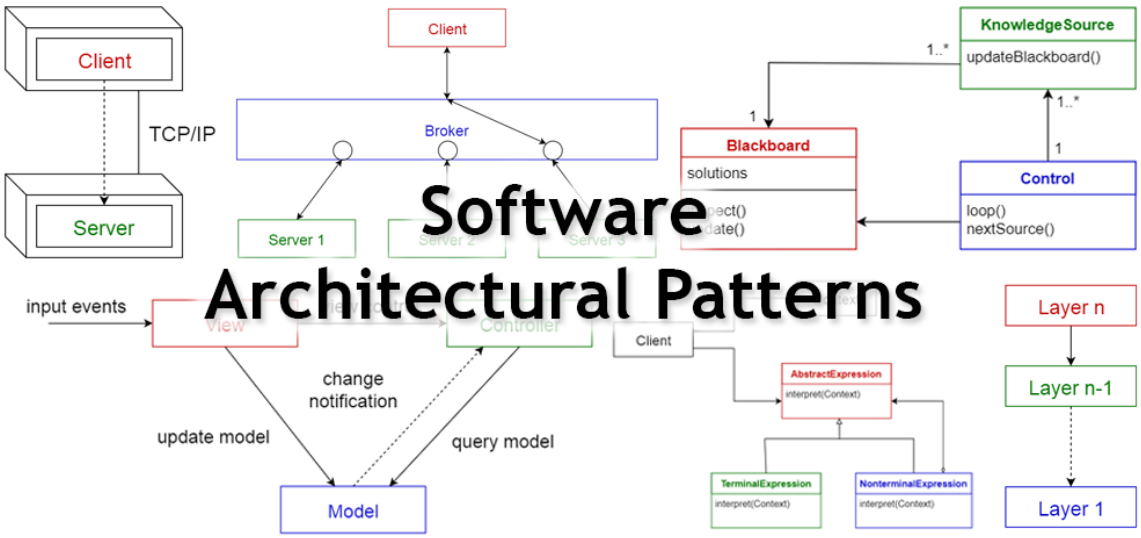
- URL Mapping

  Ένα web framework μας προσφέρει τη δυνατότητα να δημιουργούμε urls με εύκολο τρόπο ώστε να απλοποιήσουμε την ευρετηρίαση του ιστότοπου μας από τις μηχανές αναζήτησης.

- Applications

  Πολλοί τύποι εφαρμογών web υποστηρίζονται από τα web frameworks. Τα πιο συνηθισμένα και καλύτερα για την ανάπτυξη εφαρμογών υποστηρίζουν τη δημιουργία blog, forum, ιστότοπων γενικού σκοπού και συστήματα διαχείρισης περιεχομένου (CMS).

2.2 Αρχιτεκτονικό μοτίβο



Εικόνα 6 - Αρχιτεκτονικά μοτίβα

2.2.1 Τι είναι το αρχιτεκτονικό μοτίβο

Ακριβώς όπως η αρχιτεκτονική ενός κτιρίου, η αρχιτεκτονική λογισμικού περιγράφει τον σχεδιασμό και τη συλλογή εξαρτημάτων σε συστήματα που αποτελούν τα δομικά στοιχεία του λογισμικού. Η αρχιτεκτονική του λογισμικού εξηγεί τη δομική σύνθεση του λογισμικού και τις αλληλεπιδράσεις μεταξύ των στοιχείων. Η αρχή που καθορίζει το σχήμα οργάνωσης λογισμικού για αυτά τα συστήματα ονομάζεται αρχιτεκτονικό μοτίβο.

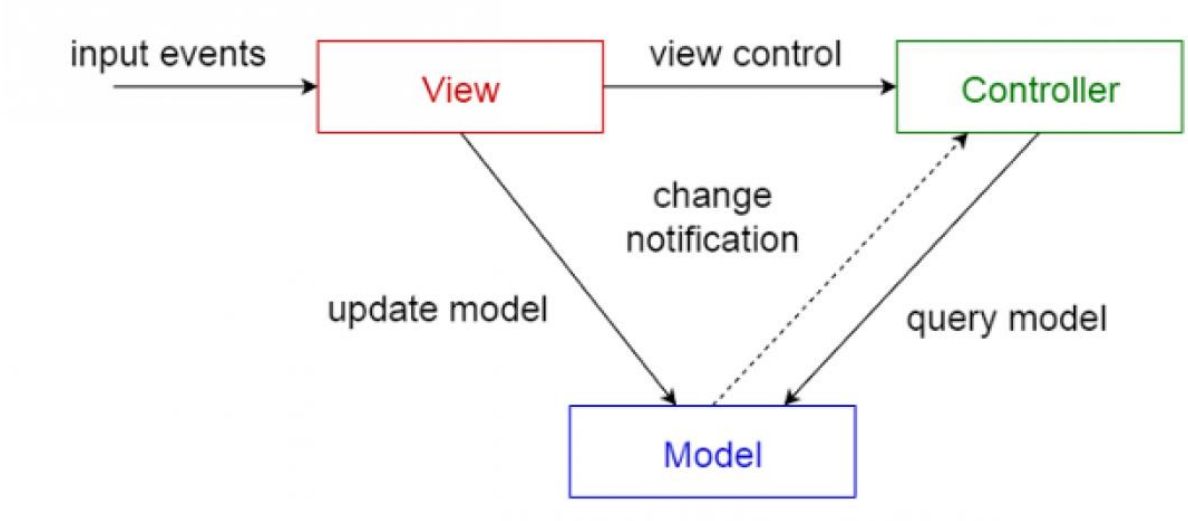
Το αρχιτεκτονικό μοτίβο αποτυπώνει τις δομές σχεδιασμού διαφόρων συστημάτων και στοιχείων λογισμικού έτσι ώστε να μπορούν να επαναχρησιμοποιηθούν. Κατά τη διαδικασία σύνταξης κώδικα, οι προγραμματιστές αντιμετωπίζουν παρόμοια προβλήματα πολλές φορές εντός ενός έργου, εντός της εταιρείας και εντός της σταδιοδρομίας τους. Ένας τρόπος για να αντιμετωπιστεί αυτό είναι η δημιουργία πρότυπων σχεδιασμού που δίνουν στους μηχανικούς έναν επαναχρησιμοποιήσιμο τρόπο επίλυσης αυτών των προβλημάτων, επιτρέποντας στους μηχανικούς λογισμικού να επιτύχουν την ίδια απόδοση δομικά για ένα δεδομένο έργο.

Από την άποψη ενός μηχανικού, τα πρότυπα αρχιτεκτονικής λογισμικού είναι σημαντικά επειδή οδηγούν στην αποδοτικότητα και την παραγωγικότητα. Οι προγραμματιστές μπορούν να συμμετάσχουν σε ένα υπάρχον έργο σε οποιοδήποτε σημείο χωρίς να χρειάζεται χρόνος προσαρμογής, καθώς κατανοούν ήδη το μοτίβο αρχιτεκτονικής που χρησιμοποιείται στο έργο. Νέα χαρακτηριστικά μπορούν επίσης να προστεθούν στο έργο χωρίς καμία δυσκολία και τα κοινά προβλήματα εφαρμογής μπορούν να επιλυθούν εύκολα.

Από την πλευρά του πελάτη, τα πρότυπα αρχιτεκτονικής βελτιστοποιούν το κόστος ανάπτυξης, επιταχύνουν το χρονοδιάγραμμα του έργου και επιτρέπουν στον μηχανικό να παραδώσει ένα προϊόν υψηλής ποιότητας. Οι εκτιμήσεις κόστους βασίζονται στην κατανόηση των αρχιτεκτονικών συστημάτων, οπότε οι διαχειριστές προϊόντων έχουν πιο ακριβές κόστος έργου, επιτρέποντας τον έγκαιρο προγραμματισμό και τον προϋπολογισμό. Επιπλέον, μια σαφώς καθορισμένη αρχιτεκτονική σημαίνει ότι το σύστημα έχει επικυρωθεί και έχει

εφαρμοστεί. Αυτό βοηθά τους μηχανικούς να επικεντρωθούν στα βασικά του προϊόντος και επιτρέπει επίσης στους διαχειριστές να σχεδιάσουν κατάλληλα για την ολοκλήρωση του έργου.

2.2.2 Model-view-controller pattern



Εικόνα 7 - model-view-controller pattern

Το μοτίβο model-view-controller (MVC) χωρίζει μια εφαρμογή σε τρία στοιχεία το model, view και ο controller.

Το μοντέλο, το οποίο είναι το κεντρικό συστατικό του μοτίβου, περιέχει τα δεδομένα της εφαρμογής και τις βασικές λειτουργίες. Είναι η δυναμική δομή δεδομένων της εφαρμογής λογισμικού και ελέγχει τα δεδομένα και τη λογική της εφαρμογής. Ωστόσο, δεν περιέχει τη λογική που περιγράφει πως παρουσιάζονται τα δεδομένα σε έναν χρήστη.

Το view εμφανίζει δεδομένα και αλληλοεπιδρά με τον χρήστη. Μπορεί να έχει πρόσβαση στα δεδομένα του μοντέλου αλλά δεν μπορεί να καταλάβει τα δεδομένα, ούτε καταλαβαίνει πως μπορούν να χειριστούν τα δεδομένα.

Ο controller χειρίζεται την είσοδο από τον χρήστη και μεσολαβεί μεταξύ του μοντέλου και του view. Ακούει εξωτερικές εισόδους από το view ή από έναν χρήστη και δημιουργεί κατάλληλες εξόδους. Ο controller αλληλοεπιδρά με το μοντέλο καλώντας μια μέθοδο σε αυτό για να δημιουργήσει τις κατάλληλες αποκρίσεις.

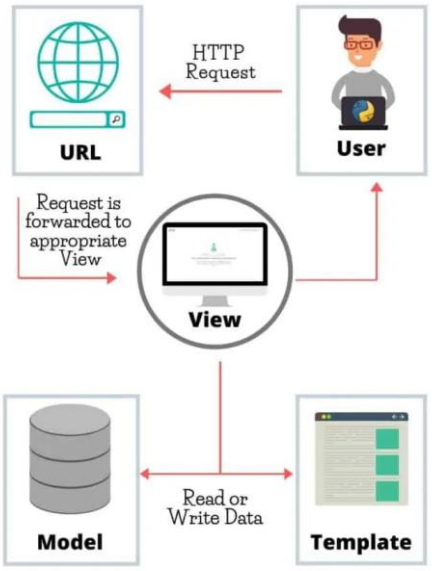
Αυτά τα τρία στοιχεία αλληλεπιδρούν μέσω κάποιας μορφής ειδοποίησης, όπως ένα συμβάν η ένα callback. Αυτές οι ειδοποιήσεις περιέχουν πληροφορίες της κατάστασης, όπως αλλαγές, οι οποίες κοινοποιούνται για την ενημέρωση αυτών των στοιχείων. Για παράδειγμα, ένα εξωτερικό συμβάν από τον χρήστη μπορεί να μεταδοθεί από τον controller για να ενημερώσει το view. Το μοτίβο MVC, επομένως, αποσυνδέει τα στοιχεία του λογισμικού και επιτρέπει την εύκολη επαναχρησιμοποίηση του κώδικα.

Δημοφιλές γλώσσες προγραμματισμού όπως η JavaScript, Python, Java και Swift διαθέτουν MVC frameworks για την ανάπτυξη εφαρμογών στο διαδίκτυο και εφαρμογές για το κινητό. Web application frameworks όπως το Ruby on Rails και το Laravel (για PHP) βασίζονται σε αυτήν την αρχιτεκτονική.

Ένα πλεονέκτημα του MVC είναι ότι πολλοί μηχανικοί μπορούν να εργαστούν και στα τρία εξαρτήματα ταυτόχρονα χωρίς σύγκρουση. Επιπλέον, το MVC επιτρέπει τη λογική

ομαδοποίηση των σχετικών εξόδων για τη δημιουργία πολυάριθμων view από το μοντέλο. Ωστόσο, ένα μειονέκτημα είναι ότι η πλοήγηση στο framework θα μπορούσε να είναι περίπλοκη καθώς εισάγει διάφορα επίπεδα abstraction.

2.2.3 Model-view-template pattern



Εικόνα 8 - model-view-template pattern

Το model -view-template (MVT) διαφέρει ελαφρώς από το MVC. Είναι μια συλλογή τριών βασικών στοιχείων το model, view και template. Αυτά τα τρία στρώματα είναι υπεύθυνα για διαφορετικά πράγματα και χρησιμοποιούμε ανεξάρτητα. Το Django είναι ένα δημοφιλές πλαίσιο ανάπτυξης ιστοσελίδων που χρησιμοποιεί το μοτίβο σχεδίασης MVT.

Η κύρια διαφορά μεταξύ των δύο πρότυπων είναι ότι το ίδιο το Django φροντίζει για το controller (κώδικας λογισμικού που ελέγχει τις αλληλεπιδράσεις μεταξύ του μοντέλου και του view), αφήνοντας μας το template. Το template είναι ένα αρχείο HTML αναμεμειγμένο με το Django template language (DTL). Παρακάτω είναι ένα απλό διάγραμμα που δείχνει την αρχιτεκτονική MVT στο Django.

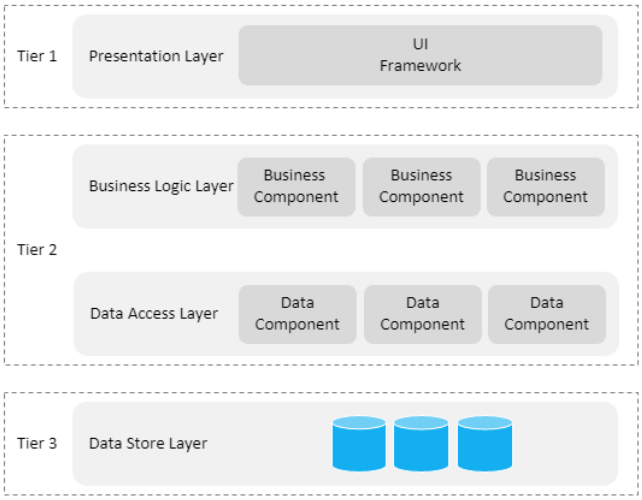
Το μοντέλο βοηθά στον χειρισμό της βάσης δεδομένων. Είναι ένα επίπεδο πρόσβασης δεδομένων, το οποίο παρέχει τα απαιτούμενα πεδία και συμπεριφορές των δεδομένων που αποθηκεύετε. Δεν υπάρχει σχεδόν καμία εφαρμογή χωρίς βάση δεδομένων. Ένα μοντέλο είναι μια κλάση Python και δε γνωρίζει τίποτα για άλλα επίπεδα του Django. Τα μοντέλα βοηθούν τους προγραμματιστές να δημιουργούν, να διαβάζουν, να ενημερώνουν και να διαγράφουν αντικείμενα (λειτουργίες CRUD) στην αρχική βάση δεδομένων. Επίσης, έχουν το business logic, προσαρμοσμένες μεθόδους, ιδιότητες και άλλα πράγματα που σχετίζονται με τον χειρισμό των δεδομένων.

Το view χρησιμοποιείται για την εκτέλεση του business logic και την αλληλεπίδραση με ένα μοντέλο για τη μεταφορά δεδομένων και την εμφανίσει ενός template. Το view λαμβάνει δεδομένα από ένα μοντέλο. Στη συνέχεια, είτε δίνει σε κάθε template πρόσβαση σε συγκεκριμένα δεδομένα που πρέπει να εμφανίζονται, είτε επεξεργάζεται δεδομένα εκ των

προτέρων. Αποδέχεται αιτήματα HTTP, εφαρμόζει το business logic που παρέχεται από κλάσεις και μεθόδους και παρέχει απαντήσεις HTTP στα client requests.

Το template είναι ένα επίπεδο παρουσίασης που χειρίζεται πλήρως το τμήμα της διεπαφής χρήστη. Αυτά είναι αρχεία με κώδικα HTML, ο οποίος χρησιμοποιείται για την προβολή δεδομένων. Το περιεχόμενο αυτών των αρχείων μπορεί να είναι στατικό ή δυναμικό. Ένα template χρησιμοποιείται μόνο για την παρουσίαση δεδομένων, καθώς δεν υπάρχει business logic σε αυτό.

2.2.4 Layered architecture pattern



Εικόνα 9 - layered architecture pattern

Η layered αρχιτεκτονική λογισμικού είναι το πιο συχνά χρησιμοποιούμενο πρότυπο αρχιτεκτονικής στη μηχανική λογισμικού. Αυτό το αρχιτεκτονικό μοτίβο είναι επίσης γνωστό ως στυλ αρχιτεκτονικής n-tier ή στυλ αρχιτεκτονικής πολλαπλών στρωμάτων. Ο σκοπός μιας πολύ επίπεδης αρχιτεκτονικής είναι να οργανώσει τα συστατικά μιας εφαρμογής σε οριζόντια λογικά επίπεδα και φυσικές βαθμίδες.

Ένα επίπεδο είναι μια λογική μονάδα που διαχωρίζει έναν συγκεκριμένο ρόλο και ευθύνη μέσα σε μια εφαρμογή. Κάθε επίπεδο διαχειρίζεται τις δικές του εξαρτήσεις λογισμικού. Ένα υψηλότερο επίπεδο μπορεί να χρησιμοποιήσει υπηρεσίες σε χαμηλότερο επίπεδο, αλλά όχι το αντίστροφο. Μια βαθμίδα είναι μια φυσική μονάδα όπου εκτελείται ο κώδικας, για παράδειγμα, ένας διακομιστής ή μια βάση δεδομένων.

Κάθε επίπεδο μπορεί να φιλοξενηθεί στη δική του βαθμίδα, ωστόσο δεν απαιτείται. Πολλά επίπεδα μπορούν να φιλοξενηθούν στην ίδια βαθμίδα. Η επεκτασιμότητα, η διατηρησιμότητα και η ανθεκτικότητα αυξάνονται κατά τον φυσικό διαχωρισμό των επιπέδων, ωστόσο η καθυστέρηση αυξάνεται λόγω της πρόσθετης επικοινωνίας του δικτύου.

Τα τέσσερα τυποποιημένα στρώματα περιγράφονται ως εξής.

- Presentation layer

  Αυτό το επίπεδο περιέχει όλες τις διεπαφές χρήστη που εκτίθενται σε έναν χρήστη. Μπορεί να παρέχει διαφορετικούς τύπους διεπαφών χρήστη, συγκεκριμένα web, desktop και native εφαρμογές για κινητά.

- Business logic layer

  Αυτό το επίπεδο χειρίζεται όλο το business logic, τα validations και τις διαδικασίες.

- Data access layer

  Αυτό το επίπεδο είναι υπεύθυνο για την αλληλεπίδραση με μια βάση δεδομένων. Είναι επίσης γνωστό ως το επίπεδο επιμονής.

- Data store layer

  Αυτό το επίπεδο είναι η πραγματική αποθήκευση δεδομένων για την εφαρμογή.

Με την εισαγωγή διαφορετικών επιπέδων για τα στοιχεία του λογισμικού, το separation of concern (SoC) αυξάνεται δραστικά, βελτιώνοντας έτσι την απλότητα, τη διατηρησιμότητα και τη δοκιμασία των συστατικών.

Οι βαθμίδες περιγράφονται ως εξής.

- Βαθμίδα 1: The presentation tier

  Αυτή η βαθμίδα φιλοξενεί το front-end κώδικα, δηλαδή το επίπεδο παρουσίασης. Αυτό είναι το κορυφαίο επίπεδο της εφαρμογής και είναι ουσιαστικά ένα επίπεδο στο οποίο οι χρήστες μπορούν να έχουν άμεση πρόσβαση.

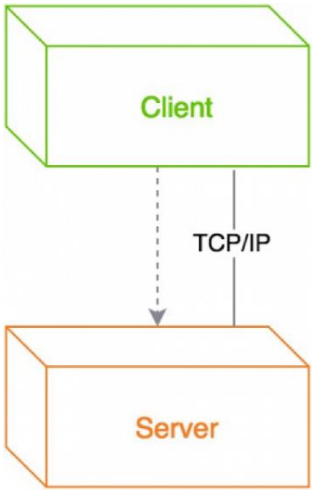
- Βαθμίδα 2: The application tier

  Αυτή η βαθμίδα φιλοξενεί το back-end κώδικα, δηλαδή το επίπεδο του business logic και το επίπεδο πρόσβασης δεδομένων. Είναι επίσης γνωστό ως μεσαίο επίπεδο.

- Βαθμίδα 3: The data tier

  Αυτή η βαθμίδα φιλοξενεί το data store, δηλαδή το επίπεδο αποθήκευσης δεδομένων. Βάσεις δεδομένων, σύστημα αρχείων, αποθήκευση blob, document database είναι παραδείγματα πόρων που βρίσκονται σε ένα data store.

2.2.5 Client-server pattern



Εικόνα 10 - client-server pattern

Στο μοτίβο αρχιτεκτονικής client-server, υπάρχουν δύο κύρια συστατικά, ο client ο οποίος είναι ο αιτών υπηρεσιών και ο server, ο οποίος είναι ο πάροχος υπηρεσιών. Παρόλο που και ο client και ο server ενδέχεται να βρίσκονται στο ίδιο σύστημα, συχνά επικοινωνούν μέσω ενός δικτύου σε ξεχωριστό υλικό.

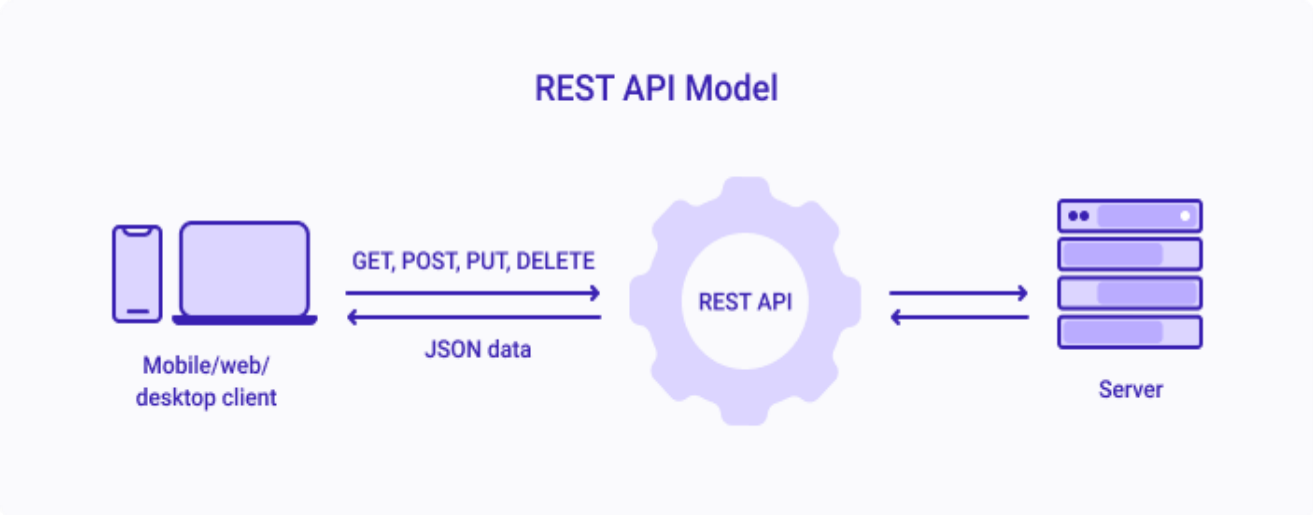
Το στοιχείο client ξεκινά ορισμένες αλληλεπιδράσεις με τον server για τη δημιουργία των απαιτούμενων υπηρεσιών. Ενώ τα στοιχεία του client έχουν θύρες που περιγράφουν τις απαραίτητες υπηρεσίες, οι server έχουν θύρες που περιγράφουν τις υπηρεσίες που παρέχουν. Και τα δύο στοιχεία συνδέονται με συνδέσμους request/reply. Ένα κλασικό παράδειγμα αυτού του μοτίβου αρχιτεκτονικής είναι ο παγκόσμιος ιστός. Το μοτίβο client-server χρησιμοποιείται επίσης για διαδικτυακές εφαρμογές, όπως file sharing και email.

Ένα απλό παράδειγμα είναι οι διαδικτυακές τραπεζικές υπηρεσίες. Όταν ένας πελάτης της τράπεζας έχει πρόσβαση σε διαδικτυακές υπηρεσίες χρησιμοποιώντας τον browser του, ο πελάτης υποβάλλει ένα αίτημα στον web server της τράπεζας. Σε αυτήν την περίπτωση, ο web browser είναι ο πελάτης, ο οποίος αποκτά πρόσβαση στον web server της τράπεζας για δεδομένα χρησιμοποιώντας τα στοιχεία σύνδεσης του πελάτη. Ο application server ερμηνεύει αυτά τα δεδομένα χρησιμοποιώντας το business logic της τράπεζας και στη συνέχεια παρέχει την κατάλληλη έξοδο στον web server.

Ένα σημαντικό πλεονέκτημα αυτού του μοτίβου αρχιτεκτονικής είναι ο κεντρικός υπολογισμός δεδομένων. Όλα τα αρχεία αποθηκεύονται σε κεντρική τοποθεσία για αυτό το δίκτυο. Επομένως, τα δεδομένα καθώς και τα περιφερειακά του δικτύου, ελέγχονται κεντρικά. Ένα μειονέκτημα, ωστόσο είναι ότι η αγορά και διαχείριση του server είναι ακριβή.

Το μοντέλο client-server σχετίζεται με το μοτίβο αρχιτεκτονικής peer-to-peer και συχνά περιγράφεται ως υποκατηγορία αυτού του μοτίβου. Το τελευταίο χρησιμοποιεί ένα αποκεντρωμένο σύστημα στο οποίο οι ομότιμοι επικοινωνούν μεταξύ τους άμεσα.

2.3 Representational State Transfer (REST)



Εικόνα 11 - representation state transfer

2.3.1 Τι είναι ένα REST API

Ένα API ή application programming interface, είναι ένα σύνολο κανόνων που καθορίζουν τον τρόπο με τον οποίο οι εφαρμογές ή οι συσκευές μπορούν να συνδέονται μεταξύ τους και να επικοινωνούν μεταξύ τους. Το REST API είναι ένα API που συμμορφώνεται με τις αρχές σχεδιασμού του REST, ή του representational state transfer. Για το λόγο αυτό, τα REST API αναφέρονται μερικές φορές σε RESTful APIs.

Το REST πρωτοεμφανίστηκε το 2000 από τον επιστήμονα υπολογιστών Dr. Roy Fielding στη διδακτορική του διατριβή, το REST παρέχει σχετικά υψηλό επίπεδο ευελιξίας και ελευθερίας στους προγραμματιστές. Αυτή η ευελιξία είναι ένας από τους λόγους για τον οποίο τα REST API εμφανίστηκαν ως μια κοινή μέθοδος για τη σύνδεση στοιχείων και εφαρμογών σε μια αρχιτεκτονική μικροϋπηρεσιών.

2.3.2 Αρχές σχεδιασμού REST

Στο πιο βασικό επίπεδο, ένα API είναι ένας μηχανισμός που επιτρέπει σε μια εφαρμογή ή υπηρεσία να έχει πρόσβαση σε έναν πόρο σε μια άλλη εφαρμογή ή υπηρεσία. Η εφαρμογή που πραγματοποιεί την πρόσβαση ονομάζεται πελάτης και η εφαρμογή που περιέχει τον πόρο ονομάζεται διακομιστής.

Ορισμένα API, όπως το SOAP ή το XML-RPC, επιβάλουν ένα αυστηρό πλαίσιο στους προγραμματιστές. Αλλά τα REST API μπορούν να αναπτυχθούν χρησιμοποιώντας σχεδόν οποιαδήποτε γλώσσα προγραμματισμού και να υποστηρίξουν μια ποικιλία μορφών δεδομένων. Η μόνη απαίτηση είναι να ευθυγραμμιστούν με τις ακόλουθες έξι αρχές σχεδιασμού REST γνωστές και ως αρχιτεκτονικοί περιορισμοί.

- Uniform interface

  Όλα τα αιτήματα API για τον ίδιο πόρο πρέπει να φαίνονται ίδια, ανεξάρτητα από που προέρχεται το αίτημα. Το REST API θα πρέπει να διασφαλίζει ότι το ίδιο κομμάτι δεδομένων, όπως το όνομα ή διεύθυνση ηλεκτρονικού ταχυδρομείου ενός χρήστη, ανήκει μόνο σε ένα ενιαίο αναγνωριστικό πόρου (URI). Οι πόροι δεν πρέπει να είναι πολύ μεγάλοι αλλά πρέπει να περιέχουν κάθε πληροφορία που μπορεί να χρειαστεί ο πελάτης.

- Client server decoupling

  Στον σχεδιασμό REST API, οι εφαρμογές πελάτη και διακομιστή πρέπει να είναι εντελώς ανεξάρτητες μεταξύ τους. Οι μόνες πληροφορίες που πρέπει να γνωρίζει η εφαρμογή πελάτη είναι το URI του ζητούμενου πόρου, δεν μπορεί να αλληλεπίδραση με τη εφαρμογή διακομιστή με άλλους τρόπους. Ομοίως, μια εφαρμογή διακομιστή δεν πρέπει να τροποποιεί την εφαρμογή πελάτη εκτός από τη μεταφορά της στα ζητούμενα δεδομένα μέσω HTTP.

- Statelessness

  Τα REST API είναι stateless, πράγμα που σημαίνει ότι κάθε αίτημα πρέπει να περιλαμβάνει όλες τις απαραίτητες πληροφορίες για την επεξεργασία του. Με άλλα λόγια, τα REST API δεν απαιτούν server-side sessions. Οι εφαρμογές διακομιστή δεν επιτρέπεται να αποθηκεύουν δεδομένα που σχετίζονται με client request.

- Cacheability

Όταν είναι δυνατόν, οι πόροι πρέπει να αποθηκεύονται με προσωρινή μνήμη cache από την πλευρά του πελάτη ή του διακομιστή. Οι απαντήσεις διακομιστή πρέπει επίσης να περιέχουν πληροφορίες σχετικά με το εάν επιτρέπεται η προσωρινή αποθήκευση για τον παραδιδόμενο πόρο. Ο στόχος είναι να βελτιωθεί η απόδοση από την πλευρά του πελάτη, αυξάνοντας παράλληλα την επεκτασιμότητα από την πλευρά του διακομιστή.

- Layered system architecture

  Στα REST API, οι κλήσεις και οι απαντήσεις περνούν από διαφορετικά επίπεδα. Κατά κανόνα μην υποθέσετε ότι οι εφαρμογές προγράμματος πελάτη και διακομιστή συνδέονται απευθείας μεταξύ τους. Μπορεί να υπάρχουν πολλοί διαφορετικοί ενδιάμεσοι στον βρόχο επικοινωνίας. Τα REST API πρέπει να σχεδιαστούν έτσι ώστε ούτε ο πελάτης ούτε ο διακομιστής να μπορούν να γνωρίζουν εάν επικοινωνεί με την τελική εφαρμογή ή με ενδιάμεσο.

- Code on demand (optional)

  Τα REST API συνήθως στέλνουν στατικούς πόρους, αλλά σε ορισμένες περιπτώσεις, οι απαντήσεις μπορούν επίσης να περιέχουν εκτελέσιμο κώδικα (όπως εφαρμογές Java). Σε αυτές τις περιπτώσεις, ο κώδικας πρέπει να λειτουργεί μόνο κατόπιν αιτήματος.

2.3.3 Πώς λειτουργούν τα REST API

Τα REST API επικοινωνούν μέσω αιτημάτων HTTP για την εκτέλεση τυπικών λειτουργιών στη βάση δεδομένων, όπως η δημιουργία, η ανάγνωση και η διαγραφή εγγραφών (επίσης γνωστών ως CRUD) μέσα σε ένα πόρο. Για παράδειγμα, ένα REST API θα χρησιμοποιούσε ένα αίτημα GET για ανάκτηση, ένα αίτημα POST για δημιουργία, ένα αίτημα PUT για ενημέρωση και ένα αίτημα DELETE για διαγραφή μιας εγγραφής. Όλες οι μέθοδοι HTTP μπορούν να χρησιμοποιηθούν σε κλήσεις API.

Η κατάσταση ενός πόρου σε οποιαδήποτε συγκεκριμένη στιγμή ή χρονική σήμανση, είναι γνωστή ως αναπαράσταση πόρων. Αυτές οι πληροφορίες μπορούν να παραδοθούν σε έναν πελάτη σε σχεδόν οποιαδήποτε μορφή, συμπεριλαμβανομένης της Javascript Object Notation (JSON), HTML, XML, Python, PHP ή απλού κειμένου. Το JSON είναι δημοφιλές επειδή είναι ευανάγνωστο τόσο από ανθρώπους όσο και από μηχανές.

Οι headers και οι παράμετροι ενός αιτήματος είναι επίσης σημαντικά σε μια κλήση REST επειδή περιλαμβάνουν σημαντικές πληροφορίες αναγνώρισης, όπως μεταδεδομένα, εξουσιοδοτήσεις, ενιαία αναγνωριστικά πόρων (URI), caching και cookie. Οι headers αιτήματος και οι headers απάντησης, μαζί με τους συμβατικούς HTTP status codes, χρησιμοποιούνται σε καλά σχεδιασμένα REST API.

2.3.4 Βέλτιστες πρακτικές REST API

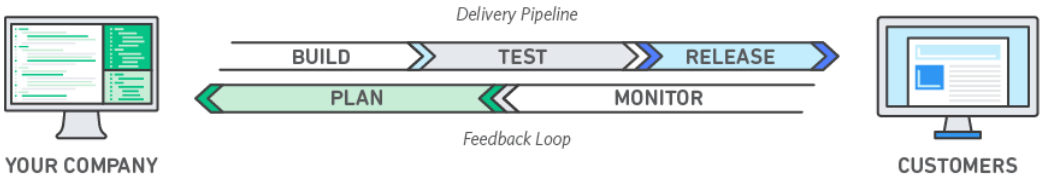
Παρόλο που η ευελιξία είναι ένα μεγάλο πλεονέκτημα του σχεδιασμού ενός REST API, η ίδια ευελιξία μπορεί να δημιουργήσει ένα API που είναι σπασμένο ή έχει κακή απόδοση. Για το λόγο αυτό, οι προγραμματιστές μοιράζονται τις βέλτιστες πρακτικές στις προδιαγραφές ενός REST API.

Η προδιαγραφή OpenAPI (OAS) δημιουργεί μια διεπαφή για την περιγραφή ενός API με τρόπο που επιτρέπει σε οποιονδήποτε προγραμματιστή ή εφαρμογή να το διαβάσει και να

κατανοήσει πλήρως τις παραμέτρους και τις δυνατότητές του όπως διαθέσιμα endpoints, επιτρεπόμενες λειτουργίες σε κάθε endpoint, παραμέτρους λειτουργίας, μεθόδους authentication και άλλες πληροφορίες. Η τελευταία έκδοση OAS3, περιλαμβάνει πρακτικά εργαλεία, όπως το OpenAPI generator.

Η εξασφάλιση ενός REST API ξεκινά επίσης με τις βέλτιστες πρακτικές του κλάδου, όπως η χρήση αλγορίθμων κατακερματισμού για την ασφάλεια των κωδικών πρόσβασης και το HTTPS για ασφαλή μετάδοση δεδομένων. Ένα πλαίσιο εξουσιοδότησης όπως το OAuth 2.0 μπορεί να βοηθήσει στον περιορισμό των προνομίων που έχουν εφαρμογές τρίτων. Χρησιμοποιώντας μια χρονική σήμανση στο HTTP header, ένα API μπορεί επίσης να απορρίψει οποιοδήποτε αίτημα έρχεται μετά από μια ορισμένη χρονική περίοδο. Η επικύρωση παραμέτρων και τα JSON Web Tokens είναι άλλοι τρόποι για να διασφαλίσετε ότι μόνο εξουσιοδοτημένοι πελάτες μπορούν να έχουν πρόσβαση στο API.

2.4 DevOps



Εικόνα 12 - DevOps delivery pipeline

2.4.1 Τι είναι το DevOps

Η λέξη DevOps είναι ένας συνδυασμός των όρων ανάπτυξης και λειτουργιών, που σημαίνει ότι αντιπροσωπεύουν μια συνεργατική ή κοινή προσέγγιση στα καθήκοντα που εκτελούνται από τις ομάδες ανάπτυξης εφαρμογών και μηχανικών μιας εταιρείας. Με το ευρύτερο νόημα του, το DevOps είναι μια φιλοσοφία που προωθεί την καλύτερη επικοινωνία και συνεργασία μεταξύ αυτών των ομάδων και άλλων σε έναν οργανισμό. Στην πιο συγκεκριμένη ερμηνεία του, το DevOps περιγράφει την υιοθέτηση επαναληπτικής ανάπτυξης λογισμικού, αυτοματοποίηση και προγραμματιζόμενη ανάπτυξη και συντήρηση της υποδομής. Ο όρος καλύπτει επίσης αλλαγές την κουλτούρα, όπως η οικοδόμηση εμπιστοσύνης και συνοχής μεταξύ προγραμματιστών και διαχειριστών συστημάτων και η ευθυγράμμιση των τεχνολογικών έργων με τις επιχειρηματικές απαιτήσεις. Το DevOps μπορεί να αλλάξει την αλυσίδα παράδοσης λογισμικού, τις υπηρεσίες, τους ρόλους, τα εργαλεία και τις βέλτιστες πρακτικές.

Ενώ το DevOps δεν είναι τεχνολογία, τα περιβάλλοντα DevOps έχουν γενικά κοινές μεθοδολογίες. Αυτά περιλαμβάνουν τα ακόλουθα.

- Εργαλεία συνεχής ενσωμάτωσης και συνεχής παράδοσης ή συνεχούς ανάπτυξης (CI / CD), με έμφαση στην αυτοματοποίηση εργασιών.
- Προϊόντα που υποστηρίζουν την υιοθέτηση του DevOps, συμπεριλαμβανομένων συστημάτων παρακολούθησης και διαχείρισης συμβάντων σε πραγματικό χρόνο, πλατφόρμες διαχείρισης διαμόρφωσης και συνεργασίας.
- Cloud computing, μικροϋπηρεσίες και containers που υλοποιούνται ταυτόχρονα με τις μεθοδολογίες DevOps.

2.4.2 Πως λειτουργεί το DevOps

Το DevOps είναι μια μεθοδολογία που προορίζεται για τη βελτίωση της εργασίας καθ' όλη τη διάρκεια ζωής του λογισμικού. Μπορείτε να απεικονίσετε μια διαδικασία DevOps ως έναν ατέρμονο βρόχο, που περιλαμβάνει αυτά τα βήματα, το σχέδιο, τη δημιουργία κώδικα, την κατασκευή, τη δοκιμή, την κυκλοφορία, την ανάπτυξη, τη λειτουργικότητα και την παρακολούθηση μέσω feedback και τέλος το σχέδιο που επαναφέρει τον βρόχο.

Στην ιδανική περίπτωση, το DevOps σημαίνει ότι μια ομάδα γράφει λογισμικό που ανταποκρίνεται απόλυτα στις απαιτήσεις των χρηστών, αναπτύσσεται χωρίς σπατάλη χρόνου και λειτουργεί άριστα με την πρώτη δοκιμή. Οι οργανισμοί χρησιμοποιούν συνδυασμό κουλτούρας και τεχνολογίας για να επιδιώξουν αυτόν τον στόχο.

Για να ευθυγραμμιστεί το λογισμικό με τις προσδοκίες, οι προγραμματιστές και τα ενδιαφερόμενα μέρη επικοινωνούν για το έργο και οι προγραμματιστές εργάζονται σε μικρές ενημερώσεις που λειτουργούν ανεξάρτητα ο ένας από τον άλλο.

Για να αποφύγουν τους χρόνους αναμονής, οι ομάδες χρησιμοποιούν CI/CD pipelines και άλλους αυτοματισμούς για να μεταφέρουν τον κώδικα από το ένα βήμα ανάπτυξης στο άλλο. Οι ομάδες αναθεωρούν τις αλλαγές αμέσως και μπορούν να επιβάλλουν πολιτικές για να διασφαλίσουν ότι οι κυκλοφορίες πληρούν τα πρότυπα.

2.4.3 Τι προβλήματα λύνει το DevOps

Κάθε εταιρεία αντιμετωπίζει τις δικές της προκλήσεις, αλλά τα κοινά προβλήματα περιλαμβάνουν κυκλοφορίες που παίρνουν πολύ χρόνο, λογισμικό που δεν ανταποκρίνεται στις προσδοκίες και ομάδες πληροφορικής που περιορίζουν την ανάπτυξη των επιχειρήσεων.

Χωρίς χρόνους αναμονής, χειροκίνητες διαδικασίες και μακρές αναθεωρήσεις, ένα έργο DevOps μετακινείται από τις απαιτήσεις σε ετοιμοπαράδοτο λογισμικό γρηγορότερα. Οι μικρότεροι κύκλοι ανάπτυξης μπορούν να εμποδίσουν τις απαιτήσεις να αλλάξουν, έτσι ώστε το προϊόν να προσφέρει αυτό που θέλουν οι πελάτες.

Το DevOps λύνει προβλήματα επικοινωνίας και προτεραιότητας μεταξύ ειδικοτήτων πληροφορικής. Για να δημιουργήσουν βιώσιμο λογισμικό, οι ομάδες ανάπτυξης πρέπει να κατανοήσουν το περιβάλλον παραγωγής και να δοκιμάσουν τον κώδικα τους σε ρεαλιστικές συνθήκες. Μια παραδοσιακή δομή βάζει ομάδες ανάπτυξης και λειτουργίας σε κουτάκια. Αυτό σημαίνει ότι οι προγραμματιστές είναι ικανοποιημένοι όταν ο κώδικας τους παρέχει λειτουργικότητα. Αν η κυκλοφορία σπάσει στην παραγωγή, εναπόκειται στην ομάδα επιχειρήσεων να κάνει τις διορθώσεις.

Με μια κουλτούρα DevOps, οι προγραμματιστές δεν θα έχουν την απάντηση "Δούλεψε στο μηχάνημα μου" όταν προκύψει πρόβλημα. Οι αλλαγές στην παραγωγή είναι μικρές και αναστρέψιμες. Επιπλέον, όλη η ομάδα καταλαβαίνει τις αλλαγές, οπότε η διαχείριση περιστατικών απλοποιείται σε μεγάλο βαθμό.

Με μια ταχύτερη διαδικασία από την ιδέα στην ολοκλήρωση του λογισμικού, οι εταιρείες μπορούν να αξιοποιήσουν τις ευκαιρίες της αγοράς. Με αυτό τον τρόπο, το DevOps παρέχει ανταγωνιστικό πλεονέκτημα για τις επιχειρήσεις.

2.4.4 Οφέλη από το DevOps

- Ταχύτητα

  Κινηθείτε με μεγάλη ταχύτητα, ώστε να μπορείτε να καινοτομείτε για τους πελάτες γρηγορότερα, να προσαρμόζεστε καλύτερα στις μεταβαλλόμενες αγορές και να γίνετε πιο αποδοτικοί στο να καθοδηγείτε την επιχείρηση σας σε ένα καλύτερο αποτέλεσμα. Το μοντέλο DevOps επιτρέπει στους προγραμματιστές και τις ομάδες επιχειρήσεων σας να επιτύχουν αυτά τα αποτελέσματα. Για παράδειγμα, οι μικροϋπηρεσίες και η συνεχής παράδοση επιτρέπουν στις ομάδες να αναλαμβάνουν την ιδιοκτησία των υπηρεσιών και στη συνέχεια, να εκδίδουν πιο γρήγορα ενημερώσεις.

- Γρήγορη παράδοση

  Αυξήστε τη συχνότητα και τον ρυθμό των εκδόσεων, ώστε να μπορείτε να καινοτομήσετε και να βελτιώσετε το προϊόν σας γρηγορότερα. Όσο πιο γρήγορα μπορείτε να κυκλοφορήσετε νέες δυνατότητες και να διορθώσετε σφάλματα, τόσο πιο γρήγορα μπορείτε να ανταποκριθείτε στις ανάγκες των πελατών σας και να δημιουργήσετε ανταγωνιστικό πλεονέκτημα. Η συνεχής ενσωμάτωση και η συνεχής παράδοση είναι πρακτικές που αυτοματοποιούν τη διαδικασία έκδοσης λογισμικού, από την κατασκευή έως την έκδοση.

- Αξιοπιστία

  Εξασφαλίστε την ποιότητα των ενημερώσεων των εφαρμογών και των αλλαγών στην υποδομή, ώστε να μπορείτε να προβάλλετε αξιόπιστα με πιο γρήγορο ρυθμό διατηρώντας παράλληλα μια θετική εμπειρία για τους τελικούς χρήστες. Χρησιμοποιήστε πρακτικές όπως η συνεχής ενσωμάτωση και η συνεχής παράδοση για να ελέγξετε ότι κάθε αλλαγή είναι λειτουργική και ασφαλής. Οι πρακτικές παρακολούθησης και καταγραφής σας βοηθούν να είστε ενημερωμένοι για την απόδοση σε πραγματικό χρόνο.

- Ανάπτυξη

  Λειτουργήστε και διαχειριστείτε τις διαδικασίες υποδομής και ανάπτυξης σας σε μεγάλη κλίμακα. Ο αυτοματισμός και η συνέπεια σας βοηθούν να διαχειρίζεστε πολύπλοκα ή μεταβαλλόμενα συστήματα αποτελεσματικά και με μειωμένο κίνδυνο. Για παράδειγμα, η υποδομή ως κώδικας σας βοηθά να διαχειρίζεστε τα περιβάλλοντα ανάπτυξης, δοκιμών και παραγωγής σας με επαναλαμβανόμενο και πιο αποτελεσματικό τρόπο.

- Βελτιωμένη συνεργασία

  Δημιουργήστε πιο αποτελεσματικές ομάδες κάτω από μια κουλτούρα DevOps, η οποία δίνει έμφαση σε αξίες όπως η ιδιοκτησία και ευθύνη. Οι προγραμματιστές και οι ομάδες επιχειρήσεων συνεργάζονται στενά, μοιράζονται πολλές ευθύνες και συνδυάζουν τις ροές εργασίας τους. Αυτό μειώνει την αναποτελεσματικότητα και εξοικονομεί χρόνο.

- Ασφάλεια

Κινηθείτε γρήγορα διατηρώντας τον έλεγχο και διατηρώντας τη συμμόρφωση. Μπορείτε να υιοθετήσετε ένα μοντέλο DevOps χωρίς να θυσιάσετε την ασφάλεια χρησιμοποιώντας αυτοματοποιημένες πολιτικές συμμόρφωσης, λεπτομερείς ελέγχους και τεχνικές διαχείρισης διαμόρφωσης. Για παράδειγμα, χρησιμοποιώντας την υποδομή ως κώδικα και την πολιτική ως κώδικα, μπορείτε να ορίσετε και στη συνέχεια να παρακολουθήσετε τη συμμόρφωση σε κλίμακα.

2.4.5 Γιατί το DevOps έχει σημασία

Το λογισμικό και το διαδίκτυο έχουν μεταμορφώσει τον κόσμο και τις βιομηχανίες του, από ψώνια σε ψυχαγωγία σε τραπεζικές συναλλαγές. Το λογισμικό δεν υποστηρίζει πλέον απλώς μια επιχείρηση, αλλά γίνεται αναπόσπαστο συστατικό κάθε μέρους μια επιχείρησης. Οι εταιρείες αλληλεπιδρούν με τους πελάτες τους μέσω λογισμικού που παρέχεται ως διαδικτυακές υπηρεσίες ή εφαρμογές που λειτουργούν και σε κάθε είδους συσκευές. Χρησιμοποιούν επίσης το λογισμικό για να αυξήσουν τη λειτουργική αποτελεσματικότητα μετατρέποντας κάθε τμήμα της αλυσίδας, όπως logistics, επικοινωνίες και λειτουργίες. Με παρόμοιο τρόπο που οι εταιρείες φυσικών αγαθών μεταμόρφωσαν τον τρόπο σχεδιασμού, κατασκευής και παράδοσης προϊόντων με τη χρήση βιομηχανικού αυτοματισμού καθ’ όλη τη διάρκεια του 20ού αιώνα, οι εταιρείες στον σημερινό κόσμο πρέπει να μεταμορφώσουν τον τρόπο κατασκευής και παράδοσης λογισμικού.

3. Geographic information system

3.1 Τι είναι το GIS

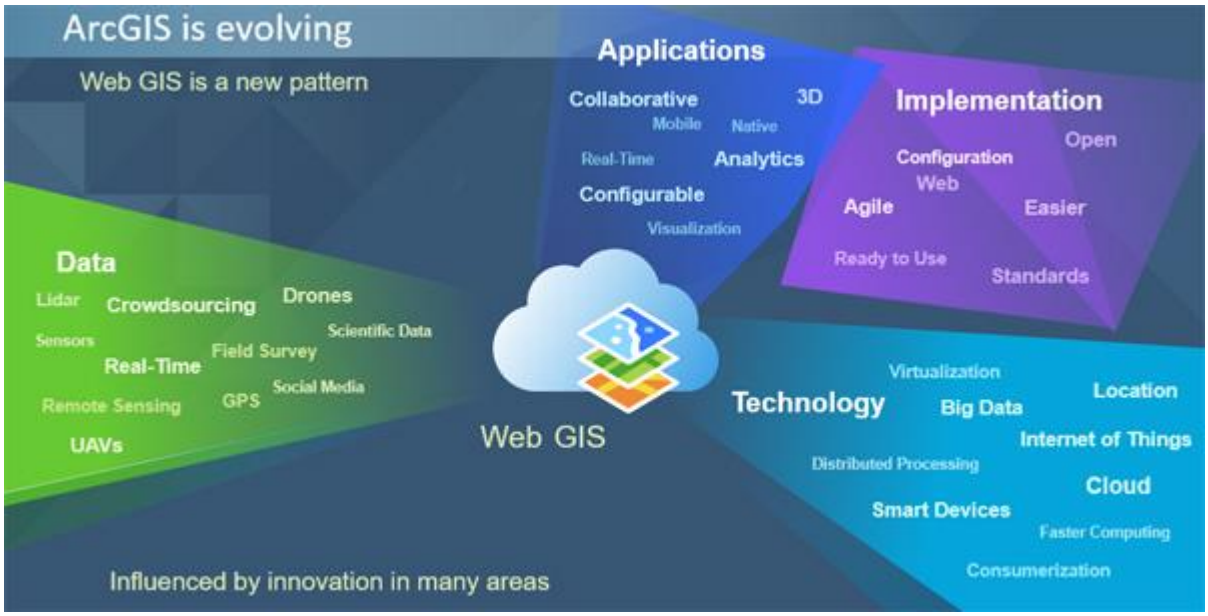
Εικόνα 13 - Geographic information system (GIS)

Ο ποιο κοινότυπος τρόπος επεξεργασίας και ανάλυσης χωρικών δεδομένων γίνεται με τη χρήση του GIS η αλλιώς geographic information system. Το GIS είναι μια συλλογή από προγράμματα ή ένα συνδυασμό προγραμμάτων που συνεργάζονται για να βοηθήσουν τους χρήστες να κατανοήσουν τα χωρικά δεδομένα τους. Αυτό περιλαμβάνει τη διαχείριση, παραμετροποίηση, ανάλυση και τη δημιουργία απεικονίσεων. Ένας χρήστης συνήθως χρησιμοποιεί πολλά χωρικά σύνολα δεδομένων ταυτόχρονα και θα τα συγκρίνει ή θα τα συνδυάζει μεταξύ τους. Κάθε χωρικό σύνολο δεδομένων μπορεί να αναφέρεται ως επίπεδο.

Εάν χρησιμοποιούσαμε το GIS για ένα πρόγραμμα για το δήμο, ενδεχομένως να έχουμε διανυσματικά δεδομένα όπως δεδομένα για δρόμους και οδούς (γραμμές), δεδομένα για τα όρια μιας γειτονιάς (πολύγωνα) και δεδομένα για τοποθεσίες όπως ένα δημοτικό σχολείο (σημεία). Κάθε σύνολο δεδομένων θα υπήρχε ως το δικό του επίπεδο στο σύστημα GIS. Η τοποθέτηση των επιπέδων είναι σημαντική για οπτικούς λόγους, καθώς θα μας βοηθήσει να κατανοήσουμε τους διάφορους τύπους δεδομένων και να παρουσιάσουμε τα ευρήματα μας με έναν εύκολα κατανοητό τρόπο.

Το πεδίο και η μελέτη του GIS εκτείνεται πολύ περισσότερο από την ψηφιακή χαρτογράφηση και τη χαρτογραφία. Αποτελείται από μια ποικιλία κατηγοριών, όπως χωρική ανάλυση, τηλεπισκόπηση και γεω-οπτικοποίηση. Σε αυτά τα πεδία του GIS, τα χωρικά δεδομένα γίνονται πολύ πιο περίπλοκα και δύσκολα στη χρήση.

3.2 WebGIS



Εικόνα 14 - WebGIS

3.2.1 Τι είναι το WebGIS

Ο παγκόσμιος ιστός (WWW) έχει αλλάξει τα πάντα και το GIS δεν αποτελεί εξαίρεση. Το WebGIS είναι μια προηγμένη μορφή γεωγραφικών συστημάτων πληροφοριών που διατίθεται σε διαδικτυακές πλατφόρμες. Ξεκίνησε ως GIS που λειτουργεί σε προγράμματα περιήγησης και εξελίχθηκε σε WebGIS που εξυπηρετεί υπολογιστές και εφαρμογές σε κινητά.

Είναι οποιοδήποτε GIS που χρησιμοποιεί τεχνολογία web για επικοινωνία μεταξύ ορισμένων στοιχείων, ενός διακομιστή GIS (προσδιορίζεται από μια διεύθυνση URL) και ενός προγράμματος-πελάτη (ένα πρόγραμμα περιήγησης, μια εφαρμογή desktop ή μια εφαρμογή για κινητά). Η επικοινωνία γίνεται μέσω πρωτοκόλλου μεταφοράς υπερκειμένου (HTTP/HTTPS) και η μορφή απόκρισης μπορεί να είναι HTML, binary εικόνα, XML (Extensible Markup Language), GML (Geography Markup Language) ή JSON (Javascript Object Notation).

Ο κύριος στόχος αυτής της τεχνολογίας είναι να επιτρέψει στους χρήστες να έχουν δυναμική πρόσβαση, κοινή χρήση και χειρισμό γεωχωρικών δεδομένων στο web, ανεξάρτητα από την πλατφόρμα ή το πρωτόκολλο.

Με απλά λόγια, το WebGIS είναι ένα πρότυπο, ή μια αρχιτεκτονική προσέγγιση, για την εφαρμογή ενός σύγχρονου GIS. Τροφοδοτείται από υπηρεσίες ιστού – τυπικές υπηρεσίες που παρέχουν δεδομένα, δυνατότητες και συνδέουν στοιχεία. Το WebGIS μπορεί να υλοποιηθεί στο cloud (χρησιμοποιώντας το ArcGIS Online), σε εσωτερικούς χώρους (χρησιμοποιώντας ArcGIS server) ή πιο τυπικά ως υβριδικό συνδυασμό, αξιοποιώντας το καλύτερο και των δύο κόσμων.

Το WebGIS δεν είναι νέο, στην πραγματικότητα εξελίσσεται εδώ και καιρό. Περάσαμε σε ένα σημείο καμπής όπου η καινοτομία στα GIS και τις σχετικές τεχνολογίες κατέστησε το WebGIS όχι μόνο δυνατό, αλλά απαραίτητο.

3.2.2 GIS server και OGC web services

Ο GIS server είναι ένα λογισμικό που ακούει ενεργά συγκεκριμένα αιτήματα που αποστέλλονται από έναν πελάτη. Αυτά τα αιτήματα θα μπορούσαν να αφορούν διαφορετικές συμβατές με OGC υπηρεσίες, για παράδειγμα αιτήματα GetMap στο Web Map Service (WMS), αίτημα GetFeature στην περίπτωση του Web Feature Service (WFS) ή λήψη αιτημάτων κάλυψης στις Web Coverage Services (WCS).

Ο GIS server φορτώνει το σύνολο δεδομένων που ζητήθηκε (για παράδειγμα ένα αρχείο shapefile ή ένα raster) το αποδίδει, χωρίζει την εικόνα σε πλακίδια και το στέλνει στον αιτούντα πελάτη. Κάθε φορά που ένας πελάτης αλληλοεπιδρά με το χάρτη, ο GIS server λαμβάνει αιτήματα και στέλνει πλακίδια εικόνας ως απαντήσεις με πολύ γρήγορη ταχύτητα.

Ο Geoserver είναι ένας από τους πιο συχνά χρησιμοποιούμενος GIS server. Διαθέτει διεπαφή διαχείρισης που βασίζεται στο web και αναπτύσσει σύνολα δεδομένων σε πρωτόκολλα συμβατά με OGC, γεγονός που το καθιστά ιδανική επιλογή.

Οι υπηρεσίες OGC καθορίζονται από την Open Geospatial Consortium (OGC) που επιτρέπει όλα τα είδη γεωχωρικών λειτουργιών. Επιτρέπουν την ανταλλαγή γεωγραφικών δεδομένων στο web. Για την ανταλλαγή πληροφοριών χρησιμοποιείται η Extensible Markup Language (XML) με βάση τη Geography Markup Language (GML).

Μερικά παραδείγματα.

- Web map services (WMS)

  Είναι ένα πρότυπο OGC που επιτρέπει στους χρήστες να αποκτήσουν απομακρυσμένη πρόσβαση σε γεωγραφικά αναφερόμενες εικόνες χάρτη μέσω αιτημάτων πρωτοκόλλου μεταφοράς υπερκειμένου (HTTPS).

- Web feature service (WFS)

  Είναι μια διεπαφή που καθορίζεται από την Open GIS Consortium (OGC) που επιτρέπει την ανταλλαγή γεωγραφικών δεδομένων στον ιστό. Οι χρήστες μπορούν να δημιουργήσουν, να διαγράψουν, να ενημερώσουν ή να κλειδώσουν ένα feature instance.

- Web coverage service (WCS)

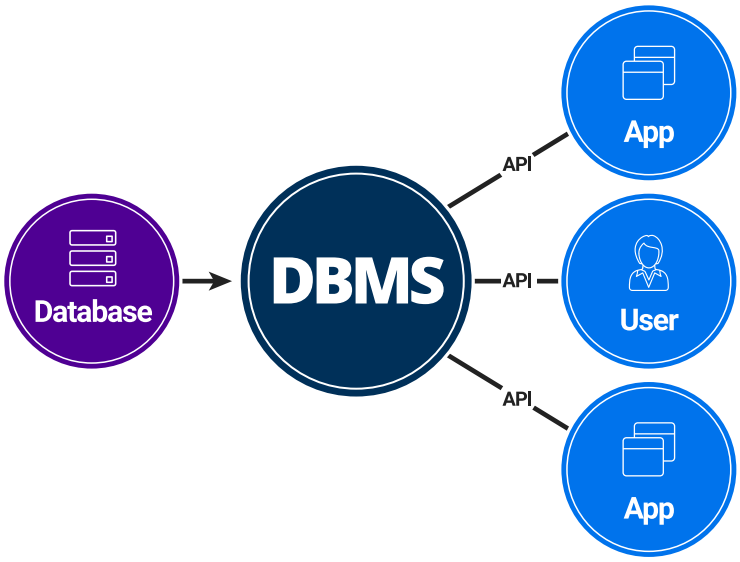
  Προσφέρει πολυδιάστατα δεδομένα κάλυψης για πρόσβαση μέσω διαδικτύου, για παράδειγμα εικόνες raster.

3.2.3 WebGIS client

Οι clients ενός WebGIS μπορούν να περιλαμβάνουν desktop εφαρμογές, εφαρμογές για κινητά ή εφαρμογές που υποστηρίζουν προγράμματα περιήγησης. Αρκετές open-source και ιδιόκτητες εφαρμογές (API) χρησιμοποιούνται για τη δημιουργία διαδικτυακών ή κινητών εφαρμογών που χρησιμοποιούν συμβατές υπηρεσίες διαδικτύου OGC. Μερικές από αυτές είναι το Open layers, Leaflet, Mapbox GL, ArcGIS API.

Για να παρέχουν μια διαισθητική εμπειρία χρήστη, αυτές οι εφαρμογές client και τα API συχνά αφαιρούν τις λεπτομέρειες όλων των web service request που εμφανίζονται στο παρασκήνιο.

3.3 Spatial data and databases



Εικόνα 15 - Database management system (DBMS)

3.3.1 Τι είναι η βάση δεδομένων

Μια βάση δεδομένων είναι το hardware που αποθηκεύει πληροφορίες, είτε πρόκειται για ένα τηλεφωνικό κατάλογο, μια λίστα προϊόντων ή άλλου τύπου αλληλένδετων δεδομένων. Ο όρος βάση δεδομένων μπορεί επίσης να αναφέρεται στα ίδια τα δεδομένα. Επιπλέων η βάση δεδομένων μπορεί να σημαίνει τον συνδυασμό των δεδομένων, του hardware και το σύστημα διαχείρισης βάσεων δεδομένων (DBMS), το οποίο είναι και το λογισμικό που επιτρέπει στους χρήστες να έχουν πρόσβαση και να επεξεργάζονται τα αποθηκευμένα δεδομένα.

3.3.2 Σύστημα διαχείρισης βάσης δεδομένων

Το σύστημα διαχείρισης βάσης δεδομένων λειτουργεί πρωτίστως ως διεπαφή μεταξύ του χρήστη και της βάσης δεδομένων, διαχειρίζοντας ταυτόχρονα τα δεδομένα, τη μηχανή της βάσης δεδομένων και το σχήμα προκειμένου να διευκολυνθεί η οργάνωση και ο χειρισμός των δεδομένων.

Η διαχείριση βάσης δεδομένων αναφέρεται στις ενέργειες που κάνουν οι επιχειρήσεις για να επεξεργαστούν τα δεδομένα τους, όπως η δημιουργία, επικύρωση, ανάκτηση, συντήρηση, αρχειοθέτηση και διαγραφή δεδομένων όταν πλέον δε χρειάζονται.

Το χρονικό διάστημα από το οποίο ένας οργανισμός δημιουργεί δεδομένα έως ότου αρχειοθετήσει ή διαγράψει είναι γνωστό ως data life cycle.

3.3.3 Σύστημα διαχείρισης χωρικών βάσεων δεδομένων

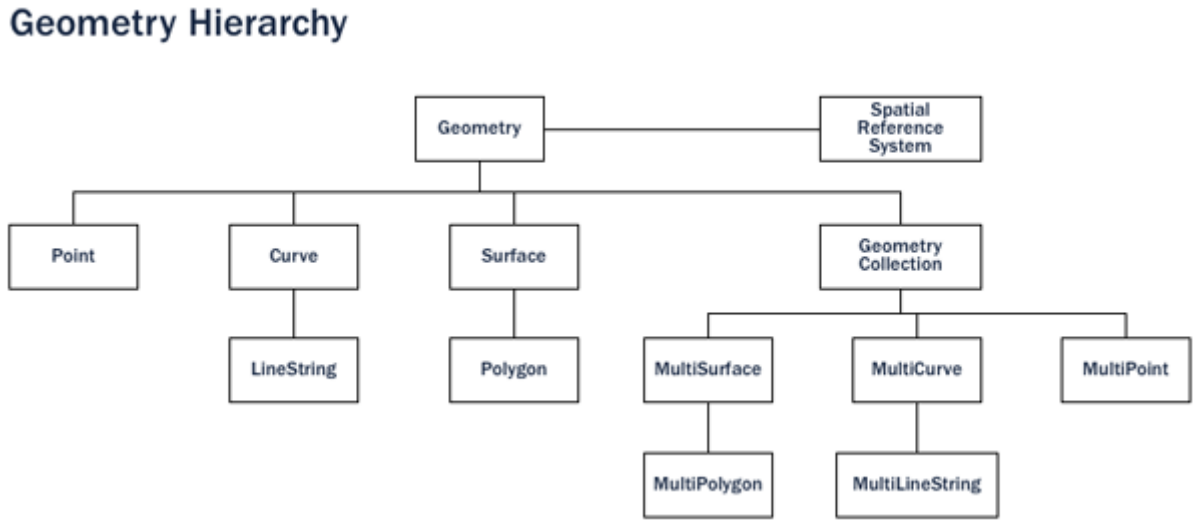
Μια χωρική βάση δεδομένων είναι βελτιστοποιημένη για δεδομένα που αντιπροσωπεύουν αντικείμενα που ορίζονται σε γεωμετρικό χώρο. Αυτά τα αντικείμενα μπορεί να είναι απλά όπως γραμμές και πολύγονα μέχρι πιο σύνθετα όπως αντικείμενα 3D. Τα

χωρικά δεδομένα είναι πολύτιμα για τη μοντελοποίηση πόλεων στον πολεοδομικό σχεδιασμό και τη χαρτογράφηση. Για τη διαχείριση χωρικών βάσεων δεδομένων χρειάζεται τα ίδια προσόντα με ένα κανονικό σύστημα διαχείρισης, καθώς εξοικείωση με τη γεωμετρία και τη γεωγραφία.

3.3.4 Χωρικά δεδομένα

Τα χωρικά δεδομένα περιλαμβάνουν τις σχετικές γεωγραφικές πληροφορίες για τη γη και τα χαρακτηριστικά της. Ένα ζεύγος συντεταγμένων γεωγραφικού πλάτους και μήκους καθορίζει μια συγκεκριμένη τοποθεσία στη γη. Τα χωρικά δεδομένα είναι δύο τύπων σύμφωνα με την τεχνική αποθήκευσης, δηλαδή έχουμε τα δεδομένα τύπου raster και vector.

- Raster data: Τα raster data είναι δεδομένα που παρουσιάζονται σε ένα πλέγμα από pixel. Κάθε εικονοστοιχείο εντός ενός raster έχει μια τιμή, είτε πρόκειται για χρώμα είτε για μονάδα μέτρησης. Τα raster συνήθως αναφέρονται σε εικόνες. Ωστόσο, στον χωρικό κόσμο, αυτό μπορεί να αναφέρεται σε φωτογραφίες από δορυφόρους (orthoimagery) ή άλλες εναέριες συσκευές.
- Vector data: Τα vector data περιγράφονται καλύτερα ως γραφικές παραστάσεις του πραγματικού κόσμου. Υπάρχουν τρεις κύριοι τύποι διανυσματικών δεδομένων, σημεία, γραμμές και πολύγωνα. Δυο σημεία σύνδεσης δημιουργούν γραμμές, τα σημεία σύνδεσης που σχηματίζουν μια κλειστή περιοχή δημιουργούν πολύγωνα. Τα vector data χρησιμοποιούνται καλύτερα για να παρουσιάσουν γενικεύσεις αντικειμένων ή χαρακτηριστικών πάνω στην επιφάνεια της γης.



Εικόνα 16 - Γεωμετρική ιεραρχία

3.3.5 GeoJSON

Το GeoJSON είναι μια μορφή για την κωδικοποίηση μια ποικιλίας δομών γεωγραφικών δεδομένων.

```
{
  "type": "Feature",
  "geometry": {
    "type": "Point",
    "coordinates": [125.6, 10.1]
  },
  "properties": {
    "name": "Dinagat Islands"
  }
}
```

Εικόνα 17 - παράδειγμα GeoJSON

Το GeoJSON υποστηρίζει τους ακόλουθους τύπους γεωμετρίας, Point, LineString, Polygon, MultiPoint, MultiLineString και MultiPolygon. Τα γεωμετρικά αντικείμενα με πρόσθετες ιδιότητες είναι αντικείμενα Feature. Τα σύνολα λειτουργιών περιέχονται σε αντικείμενα FeatureCollection.

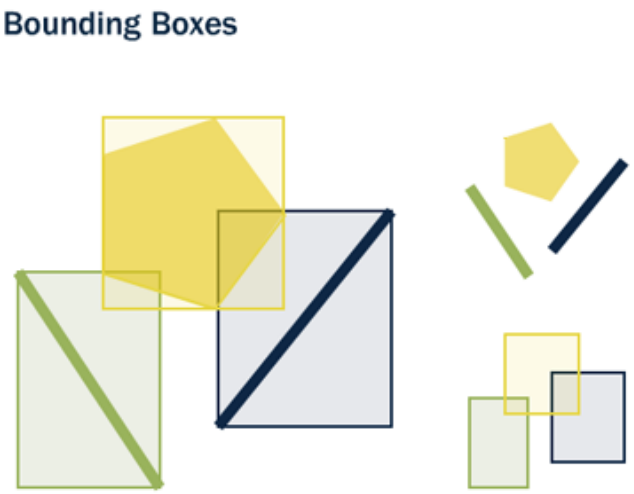
3.3.6 Spatial indexes and bounding boxes

Μια συνηθισμένη βάση δεδομένων παρέχει δείκτες που επιτρέπουν γρήγορη και τυχαία πρόσβαση σε υποσύνολα δεδομένων. Η ευρετηρίαση για τυπικούς τύπους (αριθμοί, συμβολοσειρές, ημερομηνίες) γίνεται συνήθως με ευρετήρια B-tree.

Ένα B-tree χωρίζει τα δεδομένα χρησιμοποιώντας τη φυσική σειρά ταξινόμησης για να τοποθετήσει τα δεδομένα σε ένα ιεραρχικό δέντρο. Η φυσική σειρά ταξινόμησης αριθμών, συμβολοσειρών και ημερομηνιών είναι απλή στον προσδιορισμό, κάθε τιμή είναι μικρότερη από, μεγαλύτερη ή ίση με κάθε άλλη τιμή.

Επειδή όμως τα πολύγωνα μπορούν να επικαλύπτονται, να περιέχονται το ένα στο άλλο και να τοποθετούνται σε έναν δισδιάστατο (ή περισσότερο) χώρο, ένα δέντρο B δεν μπορεί να χρησιμοποιηθεί για να τα ευρετηριάσει αποτελεσματικά. Οι πραγματικές χωρικές βάσεις δεδομένων παρέχουν έναν «χωρικό δείκτη» που απαντά στην ερώτηση «ποια αντικείμενα βρίσκονται μέσα στο συγκεκριμένο πλαίσιο οριοθέτησης?».

Ένα πλαίσιο οριοθέτησης είναι το μικρότερο ορθογώνιο παράλληλο με τους άξονες συντεταγμένων, ικανό να περιέχει ένα δεδομένο χαρακτηριστικό.



Εικόνα 18 - Bounding boxes

Τα πλαίσια οριοθέτησης χρησιμοποιούνται επειδή απαντώντας στην ερώτηση «είναι το Α μέσα στο Β?» είναι πολύ δύσκολη υπολογιστικά για πολύγωνα αλλά πολύ γρήγορη στην περίπτωση των ορθογώνιων. Ακόμα και τα πιο πολύπλοκα πολύγωνα και γραμμές μπορούν να αναπαρασταθούν με ένα απλό πλαίσιο.

Οι δείκτες πρέπει να λειτουργούν γρήγορα για να είναι χρήσιμη. Έτσι, αντί να παρέχουν ακριβή αποτελέσματα, όπως κάνουν τα B-tree, οι χωροταξικοί δείκτες παρέχουν κατά προσέγγιση αποτελέσματα. Το ερώτημα «ποιες γραμμές υπάρχουν μέσα σε αυτό το πολύγωνο?» θα ερμηνευτεί από έναν χωρικό δείκτη ως «ποιες γραμμές έχουν οριακά πλαίσια που περιέχονται μέσα στο πλαίσιο οριοθέτησης αυτού του πολύγωνου?».

Οι πραγματικοί χωροταξικοί δείκτες που εφαρμόζονται από διάφορες βάσεις δεδομένων διαφέρουν πολύ. Οι πιο συνηθισμένες εφαρμογές είναι οι R-tree και Quadtree (που χρησιμοποιούνται στο PostGIS), αλλά υπάρχουν επίσης δείκτες που βασίζονται σε πλέγματα και ευρετήρια GeoHash που εφαρμόζονται σε άλλες χωρικές βάσεις δεδομένων.

3.3.7 Spatial functions

Για τη χειραγώγηση δεδομένων κατά τη διάρκεια ενός ερωτήματος, μια συνηθισμένη βάση δεδομένων παρέχει λειτουργίες όπως η σύνδεση των συμβολοσειρών, η εκτέλεση λειτουργιών κατακερματισμού σε συμβολοσειρές, η εκτέλεση μαθηματικών σε αριθμούς και η εξαγωγή πληροφοριών από ημερομηνίες.

Μια χωρική βάση δεδομένων παρέχει ένα πλήρες σύνολο συναρτήσεων για την ανάλυση γεωμετρικών στοιχείων, τον προσδιορισμό χωρικών σχέσεων και τον χειρισμό γεωμετριών. Αυτές οι χωρικές λειτουργίες χρησιμεύουν ως δομικό στοιχείο για κάθε χωρικό έργο.

Η πλειοψηφία όλων των χωρικών συναρτήσεων μπορεί να ομαδοποιηθεί σε μία από τις ακόλουθες πέντε κατηγορίες.

- Conversion: Συναρτήσεις που μετατρέπουν γεωμετρίες και εξωτερικές μορφές δεδομένων.
- Management: Συναρτήσεις που διαχειρίζονται πληροφορίες σχετικά με τους χωρικούς πίνακες και τη διαχείριση PostGIS.

- Retrieval: Συναρτήσεις που ανακτούν ιδιότητες και μετρήσεις μιας γεωμετρίας.
- Comparison: Συναρτήσεις που συγκρίνουν δύο γεωμετρίες ως προς τη χωρική τους σχέση.
- Generation: Συναρτήσεις που δημιουργούν νέες γεωμετρίες από άλλες.

3.3.8 PostGIS

Εικόνα 19 - PostGIS logo

Το PostGIS είναι ένα λογισμικό συμβατό με την Open Geospatial Consortium (OGC) που χρησιμοποιείται ως πρόγραμμα επέκτασης για το PostgreSQL, το οποίο είναι μια μορφή βάσης δεδομένων. Ενώ το PostGIS είναι δωρεάν και ανοιχτού κώδικα, χρησιμοποιείται τόσο σε εμπορικά (π.χ. ArcGIS) όσο και λογισμικό ανοιχτού κώδικα (π.χ. QGIS). Το PostGIS επεκτείνει τις δυνατότητες του PostgreSQL για να αυξήσει τις δυνατότητες διαχείρισης του προσθέτοντας γεωχωρικούς τύπους και λειτουργίες για την ενίσχυση των χωρικών δεδομένων που διαχειρίζονται μέσα σε μια σχεσιακή βάση δεδομένων. Η γλώσσα του PostGIS είναι παρόμοια με τη SQL και επιτρέπει τη χωρική ανάλυση και τυπικά ερωτήματα να εκτελούνται σε χωρικά δεδομένα με σχετική ευκολία. Αυτό το καθιστά ένα σχετικά ισχυρό backend για βάσεις δεδομένων σε μεγάλα έργα, βοηθώντας τα έργα να χρησιμοποιούν λειτουργίες που μοιάζουν με SQL για να κάνουν πιο πολύπλοκες χωρικές αναλύσεις και ερωτήματα.

Πλεονεκτήματα για το PostGIS είναι ότι είναι σχετικά εύκολο στη διαχείριση και τη χρήση, σε σύγκριση με την τυπική αποθήκευση δεδομένων σε λογισμικό GIS, καθώς συχνά τα δεδομένα βρίσκονται μέσα σε μια δομή βάσης δεδομένων. Εκτός από τη διευκόλυνση των χωρικών ερωτημάτων, τα δεδομένα μπορούν επίσης να είναι πιο εύκολα προσβάσιμα χρησιμοποιώντας λογισμικό τρίτων ή άλλα προγράμματα, συμπεριλαμβανόμενης της λειτουργικότητας διακομιστή ιστού. Αυτό συμβαίνει επειδή χρησιμοποιεί τυπική απλή μορφοποίηση χαρακτηριστικών που επιτρέπει σε άλλους τύπους χωρικού λογισμικού να χρησιμοποιούν δυνητικά τα ίδια αποθηκευμένα δεδομένα. Ένα άλλο πλεονέκτημα είναι ότι η ανάλυση σε λογισμικό χρησιμοποιώντας τα ίδια αποθηκευμένα δεδομένα είναι επίσης σχετικά ευκολότερη. Για χρήστες που έχουν συνηθίσει να χρησιμοποιούν SQL, τον πιο συνηθισμένο τύπο γλώσσας ερωτήματος βάσης δεδομένων, το PostGIS είναι σχετικά εύκολο στη χρήση, καθώς χρησιμοποιεί την ίδια προσέγγιση για τη διεξαγωγή των ερωτημάτων του. Οι λειτουργίες ανάλυσης και επεξεργασίας μπορούν επίσης να πραγματοποιηθούν στο PostGIS

για δεδομένα ράστερ και διανύσματα, επιτρέποντας την εύκολη δημιουργία χαρτών που έχουν το επιθυμητό αναλυτικό αποτέλεσμα. Συχνά χειρίζονται τυπικές διανυσματικές μορφές Esri, όπως τα αρχεία σχήματος και τύποι ράστερ, όπως το GeoTiffs, αν και οι χρήστες έχουν μια σειρά από μορφές στις οποίες μπορούν να αποθηκεύσουν τα δεδομένα τους.

Ενώ το PostGIS έχει επιτρέψει σε διάφορα έργα να επωφεληθούν από το ισχυρό backend του, υπάρχει συζήτηση εάν το PostGIS είναι η καλύτερη προσέγγιση για διαφορετικά χωρικά προβλήματα. Οι πρόσφατες μέθοδοι NoSQL, που συχνά χρησιμοποιούν ένα πιο αντικειμενοστραφές στυλ στη δομή δεδομένων ή επιτρέπουν στους χρήστες να δημιουργούν τη δική τους δομή αποθήκευσης, έχει αποδειχθεί ότι είναι δυνητικά ταχύτερη στην ανάκτηση δεδομένων για πιο πολύπλοκα ερωτήματα, όπως ταυτόχρονα ερωτήματα πολλαπλών χρηστών. Το σχεσιακό μοντέλο που χρησιμοποιεί το PostGIS μπορεί να μην ανακτά πάντα πολύπλοκα ερωτήματα τόσο γρήγορα με βάση τον τρόπο αποθήκευσης και πρόσβασης των δεδομένων, ενώ πιο ευέλικτες προσεγγίσεις, όπου οι χρήστες GIS μπορούν να κατασκευάσουν τη δική τους μορφή αποθήκευσης δεδομένων, μπορεί να αποδειχθούν γρηγορότερες. Άλλοι έχουν επίσης διαπιστώσει ότι το PostGIS μπορεί να μην κλιμακώνεται τόσο καλά όσο και σε μεγαλύτερα προβλήματα χωρικής βάσης δεδομένων, ενώ το NoSQL μπορεί να είναι ευκολότερο να διανεμηθεί ή να κλιμακωθεί ευκολότερα σε πολλούς υπολογιστές. Αυτό θα μπορούσε να είναι ιδιαίτερα σημαντικό για web servers που έχουν μεγάλες ανάγκες αποθήκευσης χωρικών δεδομένων.

Για να παρακάμψουν τους πιθανούς περιορισμούς και προτιμήσεις των χρηστών, το λογισμικό ανοιχτού κώδικα όπως το QGIS έχει πλέον ενσωματώσει πρόσθετα και για τους δύο τύπους προσέγγισης της βάσης δεδομένων. Δημοφιλή εργαλεία όπως το MongoDB, το οποίο είναι μια προσέγγιση NoSQL, είναι ένα παράδειγμα τύπου βάσης δεδομένων που χρησιμοποιείται πλέον και με το PostGIS. Σε μεγάλο βαθμό, η χρήση διαφορετικών προσεγγίσεων NoSQL έναντι PostGIS (ή SQL) εξακολουθεί να είναι μια μορφή προτίμησης, καθώς αυτοί οι δύο τύποι προσεγγίσεων έχουν διαφορετικές μορφές ερωτημάτων ανάκτησης. Οι χρήστες γενικά γίνονται πιο άνετοι με μία προσέγγιση, όπου οι περιορισμοί του λογισμικού μπορεί να μην αποτελούν μείζον ζήτημα για το μεγαλύτερο μέρος της εργασίας που έχει γίνει. Για τους περισσότερους χρήστες, συχνά θα υπάρχουν αμελητέες διαφορές μεταξύ των δύο προσεγγίσεων όσον αφορά την απόδοση.

3.3.9 PostgreSQL

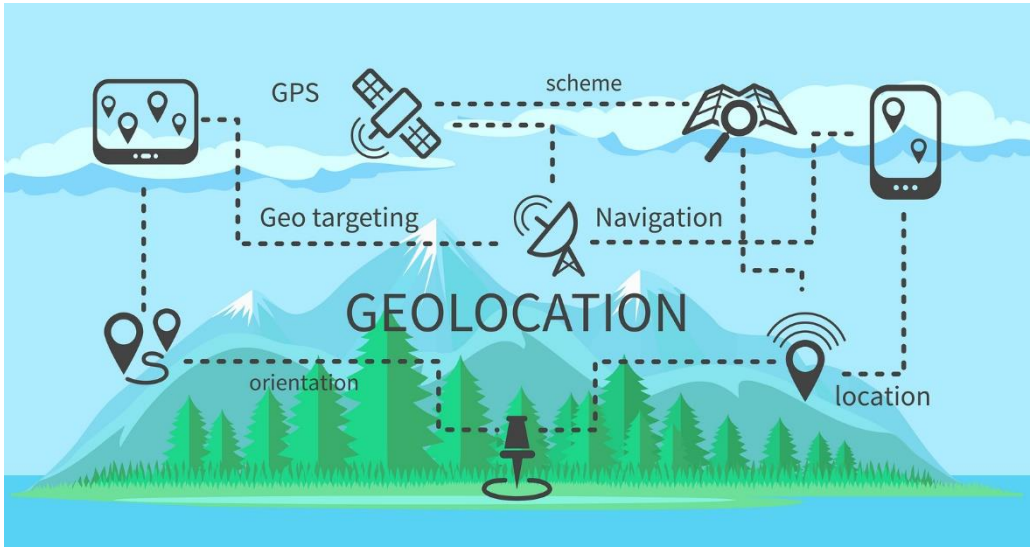
Εικόνα 20 - PostgreSQL logo

Η PostgreSQL είναι ένα ισχυρό, ανοιχτού κώδικα object-rational database που χρησιμοποιεί και επεκτείνει τη γλώσσα SQL σε συνδυασμό με πολλές δυνατότητες που αποθηκεύουν και κλιμακώνουν με ασφάλεια τον πιο περίπλοκο φόρτο εργασίας δεδομένων. Η

προέλευση της PostgreSQL χρονολογείτε από το 1986 ως μέρος του έργου POSTGRES στο πανεπιστήμιο της Καλιφόρνια στο Μπέρκλεϊ και έχει περισσότερα από 30 χρόνια ενεργής ανάπτυξης στην κεντρική πλατφόρμα.

Η PostgreSQL έχει κερδίσει μια ισχυρή φήμη για την αποδεδειγμένη αρχιτεκτονική, την αξιοπιστία, την ακεραιότητα των δεδομένων, το ισχυρό σύνολο δυνατοτήτων, την επεκτασιμότητα και την αφοσίωση της open source κοινότητας πίσω από το λογισμικό. Η PostgreSQL τρέχει σε όλα τα μεγάλα λειτουργικά συστήματα, είναι συμβατή με το ACID από το 2001 και διαθέτει ισχυρά πρόσθετα όπως το δημοφιλές PostGIS. Δεν αποτελεί έκπληξη το γεγονός ότι η PostgreSQL έχει γίνει η open source σχεσιακή βάση δεδομένων για πολλούς ανθρώπους και οργανισμούς.

3.4 Geolocation API



Εικόνα 21 - Geolocation API

3.4.1 Τι είναι το Geolocation API

Ένα Geolocation API είναι μια διεπαφή επικοινωνίας μεταξύ μιας συσκευής ή εφαρμογής από την πλευρά του πελάτη και μιας υπηρεσίας ή εφαρμογής από την πλευρά του διακομιστή που προσδιορίζει και επιστρέφει πληροφορίες σχετικά με τη γεωγραφική θέση του πελάτη. Οι παράμετροι εξόδου μπορεί να περιλαμβάνουν λεπτομέρειες σχετικά με το νόμισμα που χρησιμοποιείται στη συγκεκριμένη τοποθεσία, τη ζώνη ώρας και ακόμη και αν η χώρα είναι μέλος της ΕΕ.

Καλείται όταν υπάρχει ανάγκη για προγραμματιστική διαμόρφωση μιας διαδικτυακής ή κινητής εφαρμογής, με βάση την τοποθεσία του χρήστη και οι συνήθεις περιπτώσεις χρήσης περιλαμβάνουν πλοήγηση και γεωγραφική παρακολούθηση χάρτη σε πραγματικό χρόνο, προβολή διαφημίσεων με βάση τη τοποθεσία, εφαρμογή ελέγχων ασφαλείας χρηστών και προσαρμογή περιεχομένου σε συγκεκριμένες περιοχές.

Το Geolocation API δεν αποθηκεύει καμία πληροφορία από μόνο του, αλλά παρέχει τη δυνατότητα συλλογής υπαρχουσών πληροφοριών από τη συσκευή που ερωτάται. Μπορεί να συλλέξει πληροφορίες τοποθεσίας από πηγές όπως διεύθυνση IP, GPS, WIFI, Bluetooth,

RFID, GSM/CDMA ID ή είσοδο από τον χρήστη. Μόλις ολοκληρωθεί ένα ερώτημα API, μια τοποθεσία με συντεταγμένες επιστρέφεται στην εφαρμογή σε μορφή JSON.

3.4.2 Πώς λειτουργεί ένα Geolocation API

Συνήθως, ένα αίτημα για πληροφορίες τοποθεσίας σε ένα Geolocation API προέρχεται από μια συσκευή, για παράδειγμα, υπολογιστή ή κινητό τηλέφωνο, το οποίο παρέχει μία ή περισσότερες από τις ακόλουθες παραμέτρους εισόδου, διεύθυνση IP, διεύθυνση MAC, RFID, συντεταγμένες GPS, πύργο κυψελών (π.χ. GSM, WCDMA ή CDMA) ID ή θέση WiFi. Ένας χρήστης μπορεί επίσης να ζητήσει πληροφορίες τοποθεσίας για ένα μέρος ή έναν οργανισμό κατά όνομα.

Το Geolocation API κάνει ένα ερώτημα στον server πληροφοριών τοποθεσίας, ο οποίος απαντά στο API με διάφορες ιδιότητες τοποθεσίας, συμπεριλαμβανομένης της ακρίβειας των πληροφοριών που μεταδίδει. Στη συνέχεια, το API επιστρέφει αυτές τις πληροφορίες στον πελάτη. Από που προέρχονται οι πληροφορίες είναι διαφανείς για το API και το API το ίδιο δεν αποθηκεύει δεδομένα.

Οι πληροφορίες που παρέχει ένα τυπικό Geolocation API ποικίλλουν, αλλά μπορεί να περιλαμβάνουν τα ακόλουθα

- Φυσική διεύθυνση: Για παράδειγμα χώρα, πόλη, ήπειρος, ταχυδρομικός κώδικας, γεωγραφικό πλάτος και γεωγραφικό μήκος και αν η χώρα είναι μέλος της ΕΕ.
- Διεύθυνση IP και πληροφορίες ISP: Συμπεριλαμβανομένων πληροφοριών σχετικά με το εάν η IP είναι διεύθυνση proxy.
- Πάροχος: Όνομα παρόχου κινητής τηλεφωνίας, κωδικός περιοχής και κωδικός δικτύου.
- Τοπικά δεδομένα: Δεδομένα ζώνης ώρας (συμπεριλαμβανομένης της θερινής ώρας), της γλώσσας (συμπεριλαμβανομένων πολλών επίσημων γλωσσών) και του νομίσματος (συμπεριλαμβανομένου του συμβόλου και της περιγραφής νομίσματος).
- Πληροφορίες κινδύνου: Αν η διεύθυνση IP αποτέλεσε πηγή κακόβουλης δραστηριότητας.

3.4.3 Geolocation API use cases

Το Geolocation χρησιμοποιείται κυρίως για τη γεωγραφική προσαρμογή της ψηφιακής εμπειρίας ενός χρήστη.

- Browser context

  Όταν πραγματοποιείται αναζήτηση στο διαδίκτυο, το πρόγραμμα περιήγησης σας μπορεί να θέλει να ρωτήσει την τοποθεσία σας προκειμένου να παράσχει πληροφορίες που σχετίζονται με το ερώτημά σας για συγκεκριμένες τοπικές ρυθμίσεις, για παράδειγμα επιχειρήσεις που βρίσκονται στην περιοχή σας ή τοπικές ειδήσεις.

- Weather reports

  Εάν θέλετε να κοιτάξετε τον καιρό στο τηλέφωνο σας, η συσκευή θα σας ζητήσει να ενεργοποιήσετε τη ρύθμιση location του τηλεφώνου σας.

- Social media updates

  Εφαρμογές όπως το Facebook χρησιμοποιούν Geolocation API, ώστε οι χρήστες να μπορούν να προσθέσουν ετικέτες στις τοποθεσίες τους στις ενημερώσεις, για παράδειγμα να δείξουν στους φίλους τους που κάνουν διακοπές.

- Employment και business networking

  Websites αναζήτησης εργασίας χρησιμοποιούν geolocation για να ταιριάξουν τους αναζητητές εργασίας με τις διαθέσιμες ευκαιρίες στην περιοχή τους.

- Digital maps

  Χρησιμοποιήστε το τηλέφωνο σας για να εντοπίσετε την τοποθεσία σας και να βρείτε την ταχύτερη διαδρομή προς ένα σημείο χρησιμοποιώντας τους χάρτες της Google.

- Marketing και customer engagement

  Οι εφαρμογές web χρησιμοποιούν ένα Geolocation API για να παρακολουθούν τη θέση ενός χρήστη όταν βρίσκεται εκτός ή εντός πόλεως. Για παράδειγμα, όταν κάποιος πηγαίνει για ψώνια, μια εφαρμογή μπορεί να τον ειδοποίηση για τις τρέχουσες ειδικές προσφορές σε ένα συγκεκριμένο κατάστημα ή να προτείνει εστιατόρια στην περιοχή με βάση τις γαστρονομικές προτιμήσεις του χρήστη.

- Tracking IoT devices

  Έξυπνες οικιακές συσκευές, ή φορητά αξεσουάρ και οχήματα μπορούν εύκολα να εντοπιστούν ή να εντοπίσουν με γεωγραφική τοποθεσία.

- Cybersecurity

  Η γεωγραφική τοποθεσία μπορεί να χρησιμοποιηθεί για την παρακολούθηση εισβολέων δικτύου ή ύποπτων συνδέσεων συστήματος. Προσδιορίζει πότε και που πραγματοποιούνται ύποπτες διαδικτυακές συναλλαγές και ανακαλύψτε ποιος κρύβετε πίσω από μια διεύθυνση IP.

3.5 Frontend βιβλιοθήκες

3.5.1 Leaflet

Εικόνα 22 - leaflet.js logo

Το leaflet είναι η κορυφαία βιβλιοθήκη ανοιχτού κώδικα γραμμένη σε Javascript για mobile-friendly διαδραστικούς χάρτες. Έχει μέγεθος μόνο 39 KB και έχει όλες τις δυνατότητες χαρτογράφησης που χρειάζονται οι περισσότεροι προγραμματιστές.

Το leaflet έχει σχεδιαστεί με γνώμονα την απλότητα, την απόδοση και τη χρηστικότητα. Λειτουργεί αποτελεσματικά σε όλες τις μεγάλες πλατφόρμες για υπολογιστές και κινητές συσκευές, μπορεί να επεκταθεί με πολλά πρόσθετα, έχει ένα όμορφο, ευχάριστο και καλά τεκμηριωμένο API και έναν απλό, αναγνώσιμο πηγαίο κώδικα.

Εικόνα 23 - Παράδειγμα leaflet.js

Παρακάτω θα δούμε τον κώδικα για να δημιουργήσουμε έναν χάρτη και στη συνέχεια, προσθέτουμε έναν marker με κάποιο κείμενο σε ένα αναδυόμενο παράθυρο.

```
var map = L.map('map').setView([51.505, -0.09], 13);

L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {
    attribution: '&copy; <a
href="https://www.openstreetmap.org/copyright">OpenStreetMap</a> contributors'
}).addTo(map);

L.marker([51.5, -0.09]).addTo(map)
    .bindPopup('A pretty CSS3 popup.<br> Easily customizable.')
    .openPopup();
```

3.5.2 Mapbox GL

Εικόνα 24 - Mapbox GL JS logo

Το Mapbox GL JS είναι μια client-side βιβλιοθήκη Javascript για τη δημιουργία χαρτών web και εφαρμογών web με τη σύγχρονη τεχνολογία χαρτογράφησης του Mapbox. Μπορείτε να χρησιμοποιήσετε το Mapbox GL JS για να εμφανίσετε χάρτες του Mapbox σε web browser ή σε client, να προσθέσετε διαδραστικότητα χρηστών και να προσαρμόσετε την εμπειρία του χάρτη στην εφαρμογή σας.

Το Mapbox GL JS βασίζεται στο client side rendering. Οι χάρτες Mapbox GL JS αποδίδονται δυναμικά συνδυάζοντας διανυσματικά tiles με κανόνες στυλ στο πρόγραμμα περιήγησης και όχι σε server, γεγονός που καθιστά δυνατή την αλλαγή του στυλ και των εμφανιζόμενων δεδομένων ως απάντηση στην αλληλεπίδραση των χρηστών.

Η κλάση mapboxgl.Map είναι η βάση κάθε έργου Mapbox GL JS. Το παρακάτω παράδειγμα δείχνει τον ελάχιστο κώδικα που χρειάζεται για να προστεθεί ένας χάρτης σε μια ιστοσελίδα.

```
mapboxgl.accessToken = '<your access token here>';
const map = new mapboxgl.Map({
    container: 'map', // container ID
    style: 'mapbox://styles/mapbox/streets-v11', // style URL
    center: [-74.5, 40], // starting position [lng, lat]
    zoom: 9 // starting zoom
});
```

Οι χάρτες Mapbox GL JS μπορούν να αποτελούνται από πολλά επίπεδα που παρέχουν οπτικά στοιχεία και δεδομένα χάρτη. Κάθε επίπεδο παρέχει κανόνες σχετικά με τον τρόπο με τον οποίο ο renderer πρέπει να σχεδιάσει ορισμένα δεδομένα στο πρόγραμμα περιήγησης και ο renderer χρησιμοποιεί αυτά τα επίπεδα για να σχεδιάσει τον χάρτη στην οθόνη.

4. Σχεδιασμός της δική μας εφαρμογής WebGIS με στοιχεία crowdsourcing

4.1 Ανάλυση απαιτήσεων

Ο βασικός σκοπός είναι η δημιουργία μιας εφαρμογής WebGIS που θα μπορούν οι χρήστες να δημιουργούν τους δικούς τους χάρτες και να δουν γεωγραφική πληροφορία πάνω σε αυτούς. Χρησιμοποιώντας το μοντέλο crowdsource για να συλλέξουν πληροφορία οι χρήστες θα μπορούν να κάνουν ανοιχτή πρόσκληση σε άλλους χρήστες ώστε να ανεβάσουν γεωγραφική πληροφορία σε αυτούς του χάρτες μέσω κάποιας συσκευής είτε είναι ο προσωπικός υπολογιστής, είτε είναι το κινητό τηλέφωνο (smartphone).

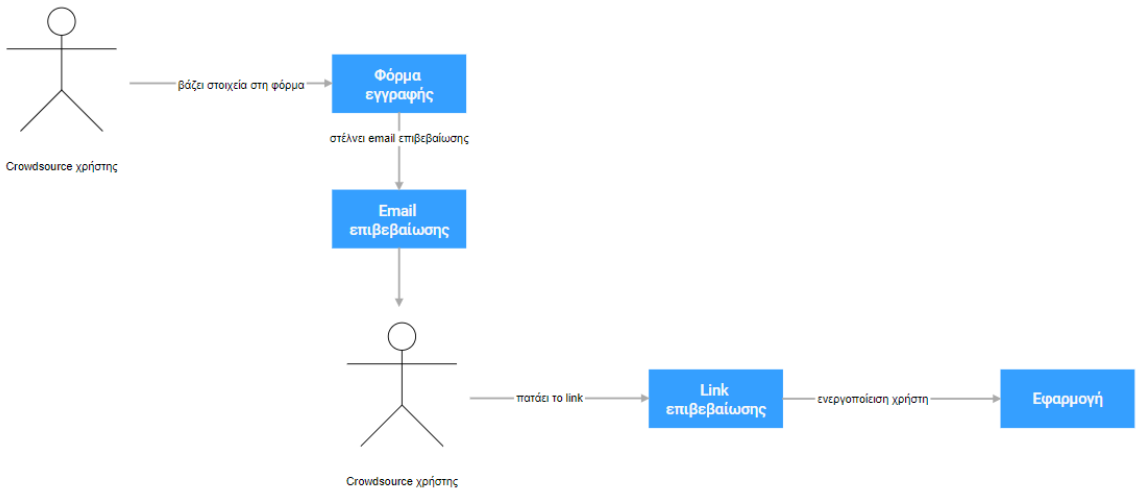
Από την ανάλυση προκύψαν οι παρακάτω απαιτήσεις που χρειάζεται η εφαρμογή για να μπορέσει να εξυπηρετήσει τον σκοπό της.

- Ο χρήστης πρέπει να μπορεί να φτιάξει λογαριασμό και να συνδεθεί στην εφαρμογή, όπως επίσης σε περίπτωση που ξεχάσει τον κωδικό του να μπορεί να τον ανακτήσει.
- Ένας εξουσιοδοτημένος χρήστης πρέπει να μπορεί να δημιουργεί χάρτες και να έχει τη δυνατότητα να τοποθετεί τα δικά του point of interest πάνω σε αυτούς.
- Πρέπει να υπάρχει ένα frontend που θα εξυπηρετεί και τον χρήστη που χρησιμοποιεί τον προσωπικό του υπολογιστή και τον χρήστη που χρησιμοποιεί το κινητό του τηλέφωνο.
- Η εφαρμογή πρέπει να έχει ένα well documented API ώστε να το χρησιμοποιήσει το frontend όπως και εφαρμογές τρίτων.
- Η βάση δεδομένων πρέπει να υποστηρίζει την αποθήκευση γεωγραφικών δεδομένων.
- Εφόσον χρησιμοποιούμε στην εφαρμογή το μοντέλο crowdsource θα περιμένουμε μεγάλο αριθμό από χρήστες να ανεβάζουν δεδομένα. Η εφαρμογή πρέπει να είναι scalable και να μπορεί να υποστηρίζει πολλούς ταυτόχρονους χρήστες.

4.2 Εννοιολογικός σχεδιασμός

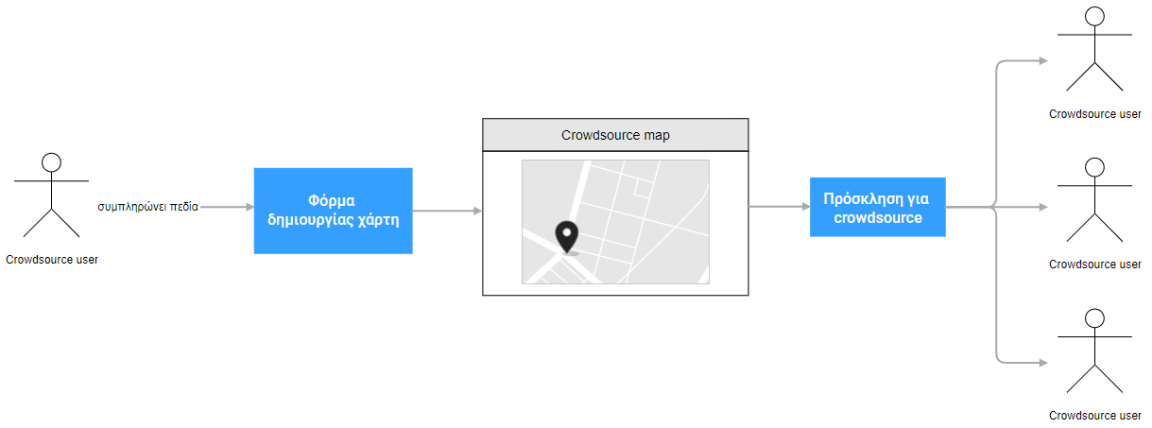
Έπειτα από την ανάλυση των απαιτήσεων της εφαρμογής, καταλήξαμε στις εξής έννοιες. Έχουμε τον χρήστη που αλληλοεπιδρά με το frontend είτε χρησιμοποιώντας τον υπολογιστή του είτε το κινητό του τηλέφωνο.

Ο χρήστης έχει τη δυνατότητα να φτιάξει το δικό του προσωπικό λογαριασμό. Η διαδικασία δημιουργίας λογαριασμού είναι απλή. Ο χρήστης δίνει τα στοιχεία του στη φόρμα εγγραφής, στη συνέχεια του έρχεται ένα email που μέσα έχει ένα link επιβεβαίωσης, πατώντας στο link ο χρήστης επιβεβαιώνει την εγγραφή του και τέλος μεταφέρεται πίσω στην εφαρμογή.



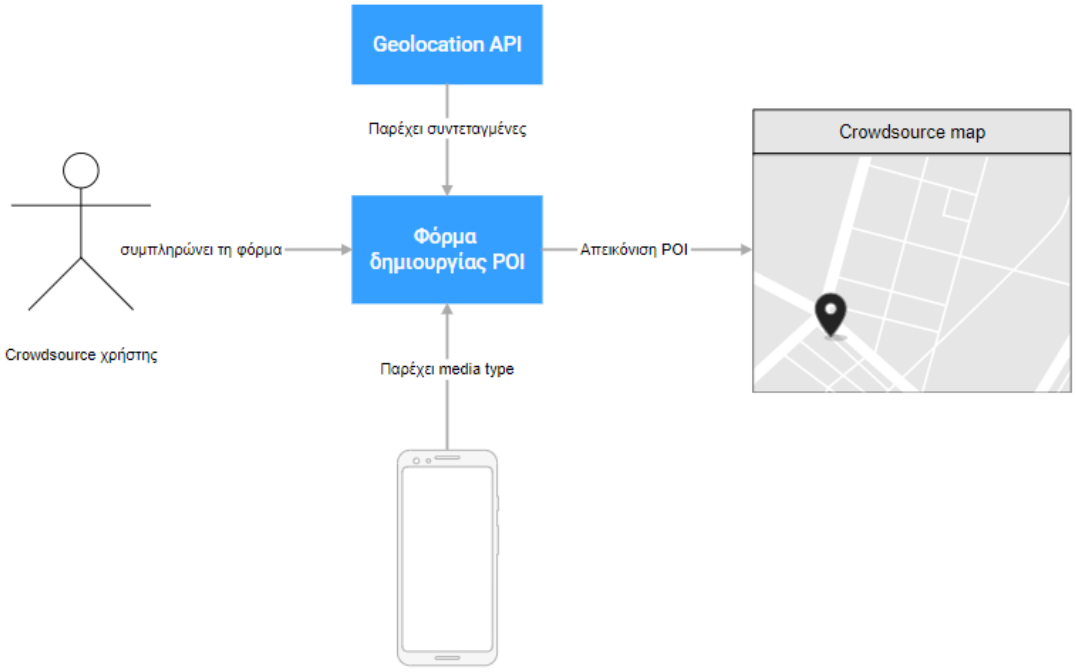
Εικόνα 25 - Διάγραμμα δημιουργίας χρήστη

Μόλις ο χρήστης έχει δημιουργήσει λογαριασμό έχει τη δυνατότητα της δημιουργίας χάρτη. Για να δημιουργήσει ένα χάρτη ο χρήστης συμπληρώνει τα πεδία του χάρτη όπως το όνομα και την περιγραφή του και φτιάχνει τις κατηγορίες που θέλει να περιέχει ο χάρτης. Μόλις δημιουργηθεί ο χάρτης μπορεί ο χρήστης με το μοναδικό URL που έχει να τον στείλει σε κόσμο και να ξεκινήσει να λειτουργεί ως crowdsourcing χάρτης.



Εικόνα 26 - Διάγραμμα δημιουργίας χάρτη

Ο χρήστης εκτός από χάρτη μπορεί να δημιουργήσει τα δικά του points of interest. Για να δημιουργήσει ο χρήστης ένα point of interest συμπληρώνει τα αντίστοιχα πεδία που είναι για poi. Επίσης, το poi χρησιμοποιεί γεωγραφικές συντεταγμένες και τύπο αναπαράστασης (media type) που μπορεί να είναι η φωτογραφία, το βίντεο ή ο ήχος. Γεωγραφικές συντεταγμένες μπορούμε να δώσουμε είτε από το Geolocation API ή να παρέχουμε εμείς. Αντίστοιχα media μπορούμε να δώσουμε από την κάμερα ή το μικρόφωνο του κινητού μας ή να παρέχουμε εμείς.

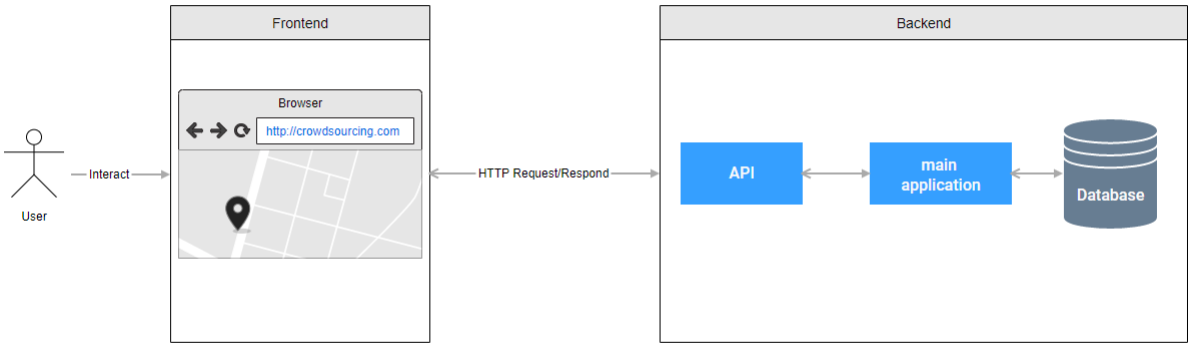


Εικόνα 27 - Διάγραμμα δημιουργίας POI

Το frontend για να μεταφέρει τα δεδομένα από τις παραπάνω λειτουργίες στο backend χρησιμοποιεί το API. Το API είναι υπεύθυνο να πάρει τα δεδομένα από το frontend να κάνει validation και sanitization και να τα μεταφέρει στο backend. Στο backend, η κεντρική εφαρμογή παίρνει τα data από το API εφαρμόζει το business logic και τέλος κάνει όποιες ενέργειες χρειάζονται στη βάση δεδομένων.

Εικόνα 28 - Data flow όταν έρχεται ένα http payload από το frontend

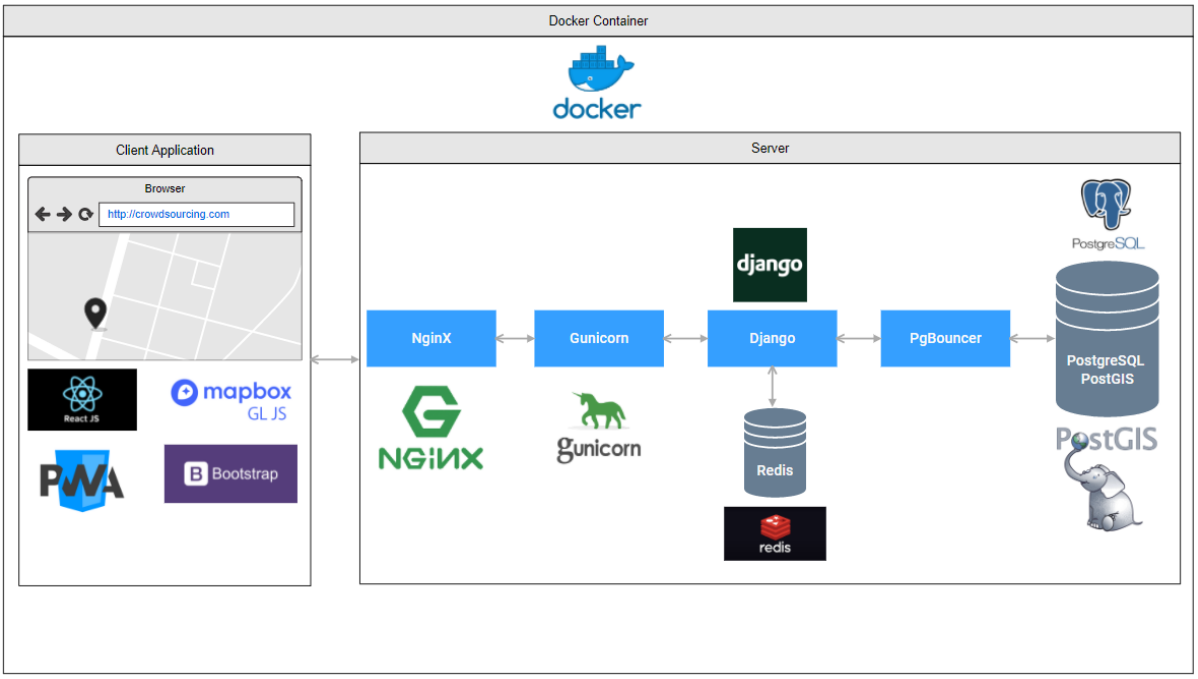
Παρακάτω βλέπουμε ένα σχεδιάγραμμα που απεικονίζει τις βασικές έννοιες που υλοποιεί η εφαρμογή.



Εικόνα 29 - Entities diagram

Ο αρχιτεκτονικός σχεδιασμός ολόκληρης της εφαρμογής θα είναι client-server. Ο λόγος είναι γιατί έχουμε πολλούς χρήστες που επικοινωνούν με ένα server. Για την κεντρική εφαρμογή θα χρησιμοποιήσουμε τον αρχιτεκτονικό σχεδιασμό Model-View-Template (MVT). Είναι ο αρχιτεκτονικός σχεδιασμός που υλοποιεί το Django web framework.

4.3 Αρχιτεκτονική της εφαρμογής



Εικόνα 30 - Αρχιτεκτονικό διάγραμμα

Μετά την ανάλυση των απαιτήσεων και έχοντας βρει την αρχιτεκτονική που θα εφαρμόσουμε είναι ώρα να δούμε ποια προγράμματα μπορούν να μας εξυπηρετήσουν στη δημιουργία μιας WebGIS εφαρμογής.

4.4 Τεχνολογίες

Ξεκινώντας από το backend για reverse proxy server και για σερβίρισμα στατικών αρχείων θα χρησιμοποιήσουμε το server Nginx. Ο Nginx μέσω web socket επικοινωνεί με το Gunicorn. Ο Gunicorn είναι ένας WSGI server που μέσω αυτού σερβίρετε η κεντρική εφαρμογή. Στη συνέχεια έχουμε την κεντρική εφαρμογή που είναι γραμμένη σε Django, εκεί γίνεται όλο το business logic και υπάρχει το API που είναι γραμμένο σε Django Rest Framework (DRF) για τη μεταφορά των δεδομένων μεταξύ του frontend και του backend. Η κεντρική εφαρμογή συνδέεται με δύο βάσης δεδομένων. Η μία είναι Redis και η άλλη είναι PostgreSQL. Η Redis χρησιμοποιείται ως cache και για message broker. Η PostgreSQL είναι η κεντρική βάση εκεί μέσω του extension PostGIS αποθηκεύονται τα γεωγραφικά δεδομένα του χρήστη. Για την επικοινωνία της βάσης δεδομένων με την κεντρική εφαρμογή έχουμε ένα connection pooler το PgBouncer.

Στο frontend έχουμε τη βιβλιοθήκη Javascript React.js. Η React είναι υπεύθυνη για όλη τη λειτουργεία που βλέπει ο χρήστης στο frontend. Για την απεικόνιση τον γεωγραφικών δεδομένων χρησιμοποιούμε το Mapbox GL JS. Για το στιλιστικό κομμάτι της εφαρμογής και για το responsiveness σε διάφορες συσκευές χρησιμοποιούμε τη CSS βιβλιοθήκη Bootstrap. Τέλος, για να δώσουμε ένα look and feel λες και είναι μια native εφαρμογή χρησιμοποιούμε την τεχνολογία PWA.

Το frontend και το backend βρίσκονται σε ένα docker container για να μπορούμε να κάνουμε γρήγορο deployment ανεξαρτήτως μηχανήματος.

4.4.1 Nginx

Εικόνα 31 - Nginx logo

Ο Nginx είναι ένα λογισμικό ανοιχτού κώδικα για web serving, reverse proxying, caching, load balancing, media streaming και άλλα. Ξεκίνησε ως web server σχεδιασμένος για μέγιστη απόδοση και σταθερότητα. Εκτός από τις δυνατότητες ως HTTP server ο Nginx μπορεί επίσης να λειτουργήσει ως proxy server για email (IMAP, POP3 και SMTP) και reverse proxy και load balancer για HTTP, TCP και UDP server.

4.4.2 Gunicorn

Εικόνα 32 - Gunicorn logo

Ο Gunicorn "Green Unicorn" είναι ένας Python WSGI HTTP server για UNIX. Είναι ένα pre-fork worker model. Ο Gunicorn server είναι σε γενικές γραμμές συμβατός με διάφορα web frameworks, είναι απλά υλοποιημένος, ελαφρύς στους πόρους του server και αρκετά γρήγορος.

4.4.3 Django

Εικόνα 33 - Django logo

Το Django είναι ένα high-level Python web framework που επιτρέπει την ταχεία ανάπτυξη ασφαλών και διατηρήσιμων ιστότοπων. Χτισμένο από έμπειρους προγραμματιστές, το Django φροντίζει για ένα μεγάλο μέρος της ταλαιπωρίας που έχει η ανάπτυξη ιστοσελίδων, ώστε να μπορείτε να εστιάσετε στη συγγραφή της εφαρμογής σας χωρίς να χρειάζεται να επανεφεύρετε τον τροχό. Είναι δωρεάν και ανοιχτού κώδικα, έχει μια ακμάζουσα και ενεργή κοινότητα, μεγάλη τεκμηρίωση και έχει πολλές επιλογές για δωρεάν και επί πληρωμή υποστήριξη.

Το Django ακολουθεί τη φιλοσοφία "Batteries included" και παρέχει σχεδόν όλα όσα θα ήθελαν να κάνουν οι προγραμματιστές "Out of the box". Επειδή όλα όσα χρειάζεστε είναι μέρος του προϊόντος, όλα λειτουργούν άψογα μαζί, ακολουθούν συνεπείς αρχές σχεδιασμού και έχουν εκτενή και ενημερωμένη τεκμηρίωση.

Το Django μπορεί (και έχει χρησιμοποιηθεί) για τη δημιουργία σχεδόν κάθε τύπου ιστότοπου. Από σύστημα διαχείρισης περιεχομένου και wikis, έως κοινωνικά δίκτυα και ειδησεογραφικούς ιστότοπους. Μπορεί να λειτουργήσει με οποιοδήποτε client-side framework και μπορεί να παραδώσει περιεχόμενο σε σχεδόν οποιαδήποτε μορφή (συμπεριλαμβανομένων HTML, RSS feeds, JSON, XML).

Το Django βοηθά τους προγραμματιστές να αποφύγουν πολλά κοινά λάθη ασφαλείας παρέχοντας ένα framework που έχει σχεδιαστεί για να κάνει τα σωστά πράγματα για την αυτόματη προστασία του ιστότοπου. Για παράδειγμα, το Django παρέχει έναν ασφαλή τρόπο διαχείρισης λογαριασμών χρηστών και κωδικών πρόσβασης, αποφεύγοντας συνηθισμένα λάθη, όπως η τοποθέτηση πληροφοριών session σε cookies όπου είναι ευάλωτα (αντί αυτού, τα cookies περιέχουν απλώς ένα κλειδί και τα πραγματικά δεδομένα αποθηκεύονται στη βάση δεδομένων) ή την απευθείας αποθήκευση κωδικών πρόσβασης αντί για password hash. Το Django επιτρέπει την προστασία από πολλές ευπάθειες από προεπιλογή, συμπεριλαμβανομένης SQL injection, cross-site scripting, cross-site request forgery και clickjacking.

4.4.4 Redis

Εικόνα 34 - Redis logo

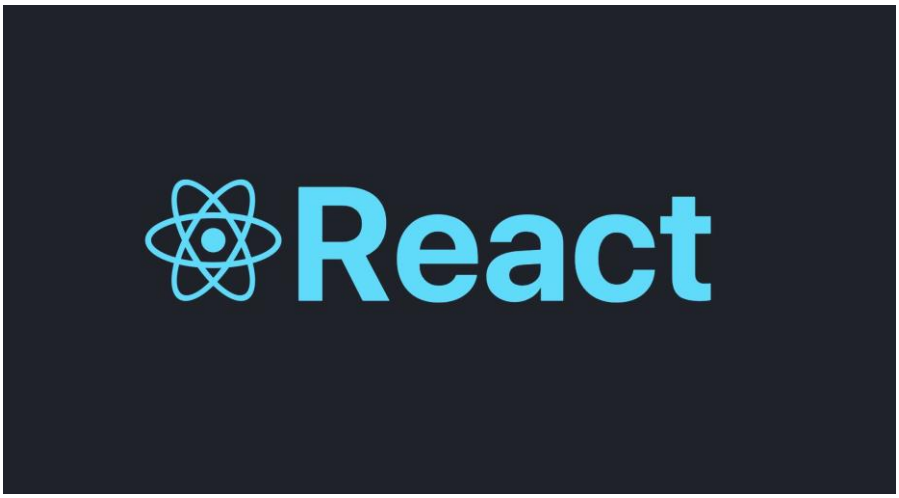
Η Redis είναι ενός ανοιχτού κώδικα (με άδεια BSD), data structure store στη μνήμη, που χρησιμοποιείται ως βάση δεδομένων, cache και message broker. Η Redis παρέχει δομές δεδομένων όπως συμβολοσειρές, hashes, λίστες, σύνολα, ταξινομημένα σύνολα, bitmaps, geospatial indexes και streams. Το Redis έχει ενσωματωμένο replication, Lua scripting, LRU eviction, transactions, διαφορετικά επίπεδα on-disk persistence, παρέχει υψηλή διαθεσιμότητα μέσω του Redis Sentinel και αυτόματο partitioning με το Redis Cluster.

4.4.5 PgBouncer

Το PgBouncer είναι ένας ανοιχτού κώδικα, ελαφρύς, single-binary connection pooler για PostgreSQL. Μπορεί να συγκεντρώσει συνδέσεις σε μία ή περισσότερες βάσεις δεδομένων (σε πιθανώς διαφορετικούς servers) και να εξυπηρετήσει πελάτες μέσω TCP και UNIX domain sockets.

Το PgBouncer διατηρεί μια ομάδα συνδέσεων για κάθε μοναδικό χρήστη. Συνήθως έχει διαμορφωθεί για να παραδώσει μία από αυτές τις συνδέσεις σε μια νέα εισερχόμενη σύνδεση πελάτη και να την επιστρέψει ξανά στην ομάδα όταν αποσυνδεθεί ο πελάτης. Μπορείτε να διαμορφώσετε το PgBouncer να συγκεντρώνει πιο επιθετικά, έτσι ώστε να μπορεί να παραλάβει και να επιστρέψει τη σύνδεση στην ομάδα σε όρια συναλλαγών ή δηλώσεων και όχι όρια σύνδεσης. Υπάρχουν, ωστόσο, δυνητικά ανεπιθύμητες συνέπειες από αυτές.

4.4.6 React



Εικόνα 35 - React logo

Η React (γνωστή και ως React.js ή ReactJS) είναι μια δωρεάν βιβλιοθήκη Javascript ανοιχτού κώδικα για τη δημιουργία διεπαφών χρήστη ή στοιχείων UI. Διατηρείται από το Facebook και μια κοινότητα μεμονωμένων προγραμματιστών και εταιρειών. Η React μπορεί να χρησιμοποιηθεί ως βάση στην ανάπτυξη εφαρμογών single-page ή κινητών. Ωστόσο, η React ασχολείται μόνο με τη διαχείριση της κατάστασης και την απόδοση αυτής στο DOM, οπότε η δημιουργία εφαρμογών React απαιτεί συνήθως τη χρήση πρόσθετων βιβλιοθηκών για δρομολόγηση, καθώς και ορισμένων λειτουργιών από την πλευρά του πελάτη.

4.4.7 Bootstrap

Εικόνα 36 - Bootstrap logo

Το Bootstrap είναι ένα δωρεάν και ανοιχτού κώδικα framework που απευθύνεται σε responsive, mobile-first, frontend web development. Περιέχει πρότυπα σχεδίασης που βασίζονται σε CSS και (προαιρετικά) Javascript για τυπογραφία, φόρμες, κουμπιά, πλοήγηση και άλλα στοιχεία διεπαφής.

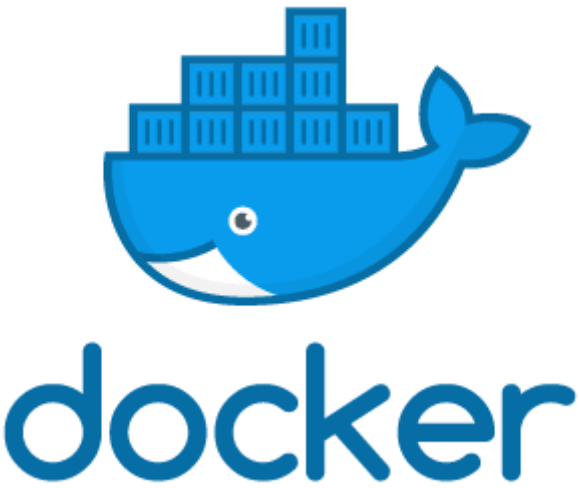
4.4.8 Progressive Web App

Εικόνα 37 - Progressive Web App logo

Progressive Web App (PWA) είναι ένα τύπος λογισμικού εφαρμογών που παραδίδεται μέσω του διαδικτύου, που έχει δημιουργηθεί χρησιμοποιώντας κοινές τεχνολογίες web, όπως HTML, CSS και Javascript. Προορίζεται να λειτουργήσει σε οποιαδήποτε πλατφόρμα που χρησιμοποιεί πρόγραμμα περιήγησης συμβατό με τα πρότυπα, συμπεριλαμβανομένων desktop και κινητών συσκευών.

Δεδομένου ότι ένα progressive web application είναι ένας τύπος ιστοσελίδας ή web application, δεν απαιτούν ξεχωριστή ομαδοποίηση ή διανομή. Οι προγραμματιστές μπορούν απλώς να δημοσιεύσουν τη διαδικτυακή εφαρμογή online, να διασφαλίσουν ότι πληροί τις βασικές απαιτήσεις εγκατάστασης και οι χρήστες θα μπορούν να προσθέσουν την εφαρμογή στην αρχική οθόνη τους. Η δημοσίευση της εφαρμογής σε ψηφιακά συστήματα διανομής όπως το Apple app store ή το Google Play είναι προαιρετική.

4.4.9 Docker



Εικόνα 38 - Docker logo

Το Docker είναι μια πλατφόρμα containerization ανοιχτού κώδικα. Επιτρέπει στους προγραμματιστές να συσκευάζουν εφαρμογές σε κοντέινερ, τυποποιημένα εκτελέσιμα

στοιχεία που συνδυάζουν πηγαίο κώδικα εφαρμογής με βιβλιοθήκες λειτουργικού συστήματος (OS) και εξαρτήσεις που απαιτούνται για την εκτέλεση αυτού του κώδικα σε οποιοδήποτε περιβάλλον. Τα containers απλοποιούν την παράδοση των διανεμημένων εφαρμογών και γίνονται όλο και πιο δημοφιλή καθώς οι οργανισμοί στρέφονται σε περιβάλλοντα που αναπτύσσονται στο cloud.

Οι προγραμματιστές μπορούν να δημιουργήσουν κοντέινερ χωρίς Docker, αλλά η πλατφόρμα καθιστά ευκολότερη, απλούστερη και ασφαλέστερη την κατασκευή, ανάπτυξη και διαχείριση κοντέινερ. Το Docker είναι ουσιαστικά μια εργαλειοθήκη που επιτρέπει στους προγραμματιστές να κατασκευάζουν, να αναπτύσσουν, να τρέχουν, να ενημερώνουν και να σταματούν κοντέινερ χρησιμοποιώντας απλές εντολές και αυτοματοποίηση εξοικονόμησης εργασίας μέσω ενός μόνο API.

5. Ανάπτυξη εφαρμογής WebGIS

5.1 Εγκατάσταση επιμέρους λογισμικού

Σε αυτό το κομμάτι της αναφοράς θα αναφερθώ στην εγκατάσταση του λογισμικού που χρησιμοποιήθηκε για την υλοποίηση της πτυχιακής. Το λογισμικό θα εγκατασταθεί σε μηχάνημα με λογισμικό Ubuntu.

5.1.1 Εγκατάσταση Gunicorn και Django

Για να εγκαταστήσουμε το Gunicorn, πρώτα θα φτιάξουμε ένα virtual environment ώστε να έχουμε τις βιβλιοθήκες που χρειάζεται η πτυχιακή οργανωμένες. Για να δημιουργήσουμε το περιβάλλον κάνουμε τα ακόλουθα βήματα.

- Πρώτα κάνουμε αναβάθμιση του pip.

  ```
  pip3 install --upgrade pip
  ```

- Στη συνέχεια κάνουμε εγκατάσταση το πακέτο virtualenv.

  ```
  pip3 install virtualenv
  ```

- Φτιάχνομε το περιβάλλον του project με τη παρακάτω εντολή.

  ```
  virtualenv crowdsource-env
  ```

- Έπειτα το ενεργοποιούμε.

  ```
  source crowdsource-env/bin/activate
  ```

- Τέλος κάνουμε εγκατάσταση τη βιβλιοθήκη Gunicorn.

  ```
  pip install Gunicorn Django
  ```

5.1.2 Εγκατάσταση Nginx και Certbot

Επειδή το Nginx είναι διαθέσιμο στα default repositories του Ubuntu, είναι δυνατό να το εγκαταστήσουμε από αυτά χρησιμοποιώντας το apt packaging system.

Δεδομένου ότι αυτή είναι η πρώτη μας αλληλεπίδραση με το apt packaging system σε αυτό το session, θα ενημερώσουμε το τοπικό ευρετήριο πακέτων, ώστε να έχουμε πρόσβαση στις πιο πρόσφατες λίστες πακέτων. Στη συνέχεια, μπορούμε να εγκαταστήσουμε το Nginx.

```
$ sudo apt update
$ sudo apt install nginx
```

Εικόνα 39 - Εγκατάσταση Nginx

Μετά την αποδοχή της διαδικασίας, το apt θα εγκαταστήσει το Nginx και τυχόν απαιτούμενες dependencies στον server μας.

Στη συνέχεια δημιουργούμε το nginx.conf αρχείο που χρειάζεται ο Nginx για να σερβίρει την εφαρμογή.

```
upstream crowdsourcing_app {
    server web:8000;
}

server {
    listen 80;

    server_name crowdsourcing.hopto.org;

    location / {
        proxy_pass http://crowdsourcing_app;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header Host $host;
        proxy_redirect off;
        client_max_body_size 100M;
    }

    location /static/ {
        autoindex on;
        alias /home/app/web/staticfiles/;
    }

    location /media/ {
        autoindex on;
        alias /home/app/web/mediafiles/;
    }

}
```

Το Upstream block είναι ο Gunicorn server που τρέχει στην πόρτα 8000. Το server block έχει τις ρυθμίσεις του Nginx. Οι ρυθμίσεις για το πως θα σερβίρει τον Gunicorn βρίσκονται στο location block, για τα στατικά αρχεία (js, css) είναι το static block και τέλος για τα media αρχεία (jpg, mp3, mp4) είναι το media block. Το server_name είναι το όνομα του domain που θα χρησιμοποιεί η εφαρμογή.

Για να ενεργοποιήσουμε το HTTPS πρωτόκολλο θα εγκαταστήσουμε τη βιβλιοθήκη certbot. Το HTTPS πρωτόκολλο χρειάζεται για να έχουμε μια ασφαλής εφαρμογή χωρίς να έχει φόβο ο χρήστης να του υποκλέψουν προσωπικά στοιχεία. Επίσης, με το HTTPS πρωτόκολλο μπορούμε να ενεργοποιήσουμε κάποιες λειτουργίες του Geolocation API και του PWA. Παρακάτω θα δούμε τις εντολές που χρειαζόμαστε για να εγκαταστήσουμε το certbot.

```
sudo apt update
sudo apt install certbot python3-certbot-nginx
sudo certbot --nginx -d crowdsourcing.hopto.org
```

Μετά από την εγκατάσταση και τη δημιουργία του πιστοποιητικού το nginx.conf αρχείο έχει την παρακάτω μορφή.

```
upstream crowdsourcing_app {
    server web:8080;
}

server {

    server_name crowdsourcing.hopto.org;

    location / {
        proxy_pass http://crowdsourcing_app;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header Host $host;
        proxy_redirect off;
        client_max_body_size 100M;
    }

    location /static/ {
        autoindex on;
        alias /home/app/web/staticfiles/;
    }

    location /media/ {
        autoindex on;
        alias /home/app/web/mediafiles/;
    }

    listen 443 ssl; # managed by Certbot
    ssl_certificate
/etc/letsencrypt/live/crowdsourcing.hopto.org/fullchain.pem; # managed by
Certbot
    ssl_certificate_key
/etc/letsencrypt/live/crowdsourcing.hopto.org/privkey.pem; # managed by
Certbot
    include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot

}

server {
    if ($host = crowdsourcing.hopto.org) {
        return 301 https://$host$request_uri;
    } # managed by Certbot


    listen 80;

    server_name crowdsourcing.hopto.org;
    return 404; # managed by Certbot
}
```

Οι γραμμές που προστέθηκαν είναι οι τοποθεσίες των κλειδιών για το HTTPS και ένα server block που κάνει redirect όταν ζητάμε τη σελίδα με HTTP και μας μεταφέρει στη σελίδα με HTTPS πρωτόκολλο με HTTP κωδικό 301 moved permanently.

5.1.3 Εγκατάσταση PostgreSQL και PostGIS

Η PostgreSQL και η PostGIS υπάρχουν στο default repository του ubuntu οπότε μπορούμε να τα κάνουμε install χρησιμοποιώντας το apt packaging system. Πρώτο βήμα είναι να κάνουμε ανανέωση το ευρετήριο τοπικών πακέτων του server μας.

```
sudo apt update
```

Στη συνέχεια, θα εγκαταστήσουμε το πακέτο PostgreSQL, PostGIS μαζί με τα πακέτα -contrib που προσθέτει κάποια επιπλέον βοηθητικά προγράμματα και λειτουργικότητα και postgresql-<version postgresql>-postgis-<version postgis> που δημιουργεί το extension για το version της postgresql μας.

```
sudo apt install postgresql postgresql-contrib postgresql-12-postgis-3
```

Από προεπιλογή, η PostgreSQL χρησιμοποιεί μια έννοια που ονομάζεται «ρόλοι» για να χειριστεί τον έλεγχο ταυτότητας και την εξουσιοδότηση. Αυτοί είναι, κατά κάποιο τρόπο, παρόμοιοι με τους κανονικούς λογαριασμούς τύπου Unix, αλλά η PostgreSQL δεν κάνει διάκριση μεταξύ χρηστών και ομάδων και προτιμά τον πιο ευέλικτο όρο «ρόλος».

Η διαδικασία εγκατάστασης δημιούργησε έναν λογαριασμό χρήστη που ονομάζεται postgres και σχετίζεται με τον προεπιλεγμένο ρόλο Postgres. Για να χρησιμοποιήσουμε το PostgreSQL, μπορούμε να συνδεθούμε σε αυτόν τον λογαριασμό.

5.1.4 Εγκατάσταση PgBouncer

Το PgBouncer βρίσκεται και αυτό στο default repository του ubuntu οπότε θα κάνουμε εγκατάσταση με το apt packaging system.

```
sudo apt-get install pgbouncer
```

Για να βεβαιωθούμε ότι το pgbouncer έχει εγκατασταθεί σωστά και ότι λειτουργεί τρέχουμε την παρακάτω εντολή.

```
sudo systemctl status pgbouncer
```

Θα πρέπει να μας απαντήσει με το παρακάτω μήνυμα.

```
● pgbouncer.service - LSB: start pgbouncer
   Loaded: loaded (/etc/init.d/pgbouncer; generated)
   Active: active (running) since Fri 2019-07-26 04:38:37 +06; 1
months 30 days ago
     Docs: man:systemd-sysv-generator(8)
    Tasks: 2 (limit: 4915)
   CGroup: /system.slice/pgbouncer.service
           └─28515 /usr/sbin/pgbouncer -d /etc/pgbouncer/pgbouncer.ini

Jul 26 04:38:37 postgres01-example-com systemd[1]: Starting LSB: start
pgbouncer...
Jul 26 04:38:37 postgres01-example-com pgbouncer[28493]:  * Starting
PgBouncer pgbouncer
Jul 26 04:38:37 postgres01-example-com pgbouncer[28493]:    ...done.
Jul 26 04:38:37 postgres01-example-com systemd[1]: Started LSB: start
pgbouncer.
```

5.1.5 Εγκατάσταση Docker και Docker Compose

Πριν εγκαταστήσουμε το Docker Engine για πρώτη φορά σε ένα νέο server, πρέπει να ρυθμίσουμε το Docker repository. Στη συνέχεια, μπορούμε να εγκαταστήσουμε και να ενημερώσουμε το Docker από το repository.

Ενημερώνουμε το repository του apt packaging system και κάνουμε εγκατάσταση τα πακέτα για να επιτρέψουμε στο apt να χρησιμοποιεί repository μέσω HTTPS.

```
sudo apt-get update


sudo apt-get install \
    apt-transport-https \
    ca-certificates \
    curl \
    gnupg \
    lsb-release
```

Προσθέτουμε το επίσημο κλειδί GPG του Docker.

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o
/usr/share/keyrings/docker-archive-keyring.gpg
```

Χρησιμοποιούμε την παρακάτω εντολή για να ρυθμίσουμε το σταθερό repository.

```
echo \

  "deb [arch=amd64 signed-by=/usr/share/keyrings/docker-archive-keyring.gpg]
https://download.docker.com/linux/ubuntu \

  $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list >
/dev/null
```

Ενημερώνουμε το apt repository και κάνουμε εγκατάσταση την πιο πρόσφατη έκδοση του Docker Engine.

```
sudo apt-get update

sudo apt-get install docker-ce docker-ce-cli containerd.io
```

5.2 Δημιουργία Docker Image και Docker Container

Παραπάνω εξηγήσαμε πως να κάνουμε εγκατάσταση τα επιμέρους συστήματα της εφαρμογής. Για να μην κάνουμε κάθε φορά την ίδια διαδικασία όταν αλλάζουμε server και για να μπορούμε να έχουμε καλύτερο έλεγχο στο τρόπο που θα δημιουργείτε το περιβάλλον της εφαρμογής θα φτιάξουμε μια σειρά από Docker Images για το κάθε ένα από τα επιμέρους συστήματα. Παρακάτω θα αναλύσουμε τα docker images που θα χρειαστούμε για να λειτουργήσει η εφαρμογή.

5.2.1 Nginx Docker Image

Για το Nginx θα χρησιμοποιήσουμε το official docker image που είναι βασισμένο στο Alpine linux. Το Alpine linux είναι ένα lightweight secure linux distro και χρησιμοποιείται σε πολλά container.

```
FROM nginx:1.21-alpine

# install certbot for nginx
RUN apk update && \
    apk add --no-cache certbot certbot-nginx

# delete default conf file
RUN rm /etc/nginx/conf.d/default.conf

# add our conf file
COPY nginx.conf /etc/nginx/conf.d
```

5.2.2 PostgreSQL Image

Το image της PostgreSQL θα το δημιουργήσουμε εμείς για να έχουμε περισσότερο έλεγχο στην εγκατάσταση της βάσης. Για βάση του image θα χρησιμοποιήσουμε το Alpine linux οπότε το image μας θα έχει μόνο τις βασικές βιβλιοθήκες του λογισμικού.

```
FROM alpine:3.12.4

# add enviromental variables
ENV POSTGRES_USER=postgres
ENV POSTGRES_DB=postgres

# update and install postgresql
RUN apk update && \
    apk add --no-cache postgresql postgis

# folder for .s.PGSQL.5432
RUN mkdir /run/postgresql && \
    chown postgres:postgres /run/postgresql/

# copy init user script
COPY init-user-db.sh /docker-entrypoint-initdb.d/init-user-db.sh

# give ownership to postgres and make it executable
RUN chown postgres:postgres /docker-entrypoint-initdb.d/init-user-db.sh && \
    chmod +x /docker-entrypoint-initdb.d/init-user-db.sh

# change to user postgres
USER postgres

# initialize database, start PostgreSQL, initialize user and database
RUN initdb -D /var/lib/postgresql/data && \
    pg_ctl start -D /var/lib/postgresql/data && \
    /bin/sh /docker-entrypoint-initdb.d/init-user-db.sh

# Adjust PostgreSQL configuration so that remote connections to the
# database are possible.
RUN echo "host all all 0.0.0.0/0 md5" >> /var/lib/postgresql/data/pg_hba.conf

# And add ``listen_addresses`` to
``/var/lib/postgresql/data/postgresql.conf``
RUN echo "listen_addresses='*'" >> /var/lib/postgresql/data/postgresql.conf

# Expose the PostgreSQL port
EXPOSE 5432/tcp

# Set the default command to run when starting the container
CMD ["postgres", "-D", "/var/lib/postgresql/data"]
```

Στη συνέχεια θα δημιουργήσουμε το init-user-db.sh script αρχείο που είναι υπεύθυνο να δημιουργήσει το χρήστη της βάσης και την ίδια τη βάση δεδομένων.

```
#!/bin/bash
set -e


psql -v ON_ERROR_STOP=1 --username "$POSTGRES_USER" --dbname "$POSTGRES_DB"
<<-EOSQL
    CREATE DATABASE crowdsource_app;
    CREATE USER crowdsource_admin WITH PASSWORD '12345678';
    ALTER USER crowdsource_admin WITH SUPERUSER;
    ALTER ROLE crowdsource_admin SET client_encoding TO 'utf8';
    ALTER ROLE crowdsource_admin SET default_transaction_isolation TO 'read
committed';
    ALTER ROLE crowdsource_admin SET timezone TO 'UTC';
    GRANT ALL PRIVILEGES ON DATABASE crowdsource_app TO crowdsource_admin;
EOSQL
```

Δημιουργούμε τη βάση με όνομα crowdsource_app, στη συνέχεια το χρήστη crowdsource_admin με password 12345678 (δεν είναι secure password αλλά θα τον χρησιμοποιήσουμε για λόγους παρουσίασης), έπειτα κάνουμε το χρήστη crowdsource_admin superuser κάνουμε κάποιες ρυθμίσεις που χρειάζεται το Django για να λειτουργήσει σωστά και τέλος δίνουμε όλα τα δικαιώματα της βάσις crowdsource_app στο χρήστη crowdsource_admin.

5.2.3 PgBouncer Image

Ακολουθούμε το ίδιο μοτίβο με την PostgreSQL φτιάχνουμε το image με βάση το Alpine linux.

```
FROM alpine:3.12.4

# enviromental variables with the project paths and user
ENV PG_LOG_FILE = /var/log/pgbouncer
ENV PG_PID_FILE = /var/log/pgbouncer
ENV PG_CONFIG_FILE = /etc/pgbouncer/pgbouncer.ini
ENV PG_USER_LIST = /etc/pgbouncer/userlist.txt
ENV PG_USER = postgres

# install pgbouncer
RUN apk update && \
    apk add --no-cache pgbouncer

# create user postgres pgbounce needs user postgres to work
RUN adduser -D postgres && \
    chmod -R 755 /var/log/pgbouncer && \
    chown -R postgres:postgres /var/log/pgbouncer

# copy entrypoint.sh
COPY entrypoint.sh ./

# make it executable
RUN chmod +x entrypoint.sh

# delete default pgbouncer.ini file
RUN rm /etc/pgbouncer/pgbouncer.ini

# copy our pgbouncer.ini file
COPY ./pgbouncer.ini /etc/pgbouncer/pgbouncer.ini

# add user list
COPY ./userlist.txt /etc/pgbouncer/userlist.txt

ENTRYPOINT ["./entrypoint.sh"]
```

Στη συνέχεια δημιουργούμε τα αρχεία entrypoint.sh, pgbouncer.ini και userlist.txt. Στο αρχείο entrypoint.sh έχουμε μόνο την εντολή για να τρέξει ο PgBouncer.

```
#!/bin/sh

exec /usr/bin/pgbouncer -u postgres /etc/pgbouncer/pgbouncer.ini
```

Το αρχείο pgbouncer.in έχει όλες τις ρυθμίσεις που χρειάζεται ο PgBouncer για να λειτουργήσει.

```
;;;
;;; PgBouncer configuration file
;;;

;; database name = connect string
;;
;; connect string params:
;;   dbname= host= port= user= password= auth_user=
;;   client_encoding= datestyle= timezone=
;;   pool_size= reserve_pool= max_db_connections=
;;   pool_mode= connect_query= application_name=
[databases]
crowdsource_app = dbname=crowdsource_app host=postgresql port=5432
auth_user=postgres

;; foodb over Unix socket
;foodb =

;; redirect bardb to bazdb on localhost
;bardb = host=localhost dbname=bazdb

;; access to dest database will go with single user
;forcedb = host=127.0.0.1 port=300 user=baz password=foo
client_encoding=UNICODE datestyle=ISO connect_query='SELECT 1'

;; use custom pool sizes
;nondefaultdb = pool_size=50 reserve_pool=10

;; use auth_user with auth_query if user not present in auth_file
;; auth_user must exist in auth_file
; foodb = auth_user=bar

;; fallback connect string
;* = host=testserver

;; User-specific configuration
[users]

;user1 = pool_mode=transaction max_user_connections=10

;; Configuration section
[pgbouncer]
auth_file = userlist.txt

;;;
;;; Administrative settings
;;;

logfile = /var/log/pgbouncer/pgbouncer.log
pidfile = /var/log/pgbouncer/pgbouncer.pid

;;;
;;; Where to wait for clients
;;;

;; IP address or * which means all IPs
listen_addr = *
```

```
listen_port = 6432

;; Unix socket is also used for -R.
;; On Debian it should be /var/run/postgresql
unix_socket_dir = /tmp
unix_socket_mode = 0777
unix_socket_group = postgres

;;;
;;; Authentication settings
;;;

;; any, trust, plain, md5, cert, hba, pam
auth_type = md5
auth_file = /etc/pgbouncer/userlist.txt

;; Query to use to fetch password from database.  Result
;; must have 2 columns - username and password hash.
auth_query = SELECT usename, passwd FROM pg_shadow WHERE usename=$1

;;;
;;; Pooler personality questions
;;;

;; When server connection is released back to pool:
;;   session      - after client disconnects (default)
;;   transaction  - after transaction finishes
;;   statement    - after statement finishes
pool_mode = session

;;;
;;; Connection limits
;;;

;; Total number of clients that can connect
max_client_conn = 100

;; Default pool size.  20 is good number when transaction pooling
;; is in use, in session pooling it needs to be the number of
;; max clients you want to handle at any moment
default_pool_size = 20
```

Ο PgBouncer για να λειτουργήσει χρειάζεται να συνδεθεί στη βάση που έχουμε δημιουργήσει. Για να συνδεθεί στη βάση πρέπει να του δώσουμε τα σωστά credentials. Τα credentials βρίσκονται στο userlist.txt.

```
"crowdsource_admin" "12345678"
```

5.2.4 Κεντρική εφαρμογή Image

Τέλος, μας έχει μείνει η κεντρική εφαρμογή το Django application. Για κεντρικό image χρησιμοποιούμε Alpine linux με εγκατεστημένη python 3.8.

```
# pull official base image
FROM python:3.8-alpine

# set environment variables
ENV PYTHONDONTWRITEBYTECODE 1
ENV PYTHONUNBUFFERED 1

# create directory for the app user
RUN mkdir -p /home/app

# create the app user
RUN addgroup -S app && adduser -S app -G app

# create the appropriate directories
ENV HOME=/home/app
ENV APP_HOME=/home/app/web
RUN mkdir $APP_HOME
RUN mkdir $APP_HOME/staticfiles
RUN mkdir $APP_HOME/mediafiles
WORKDIR $APP_HOME

# install dependencies
RUN apk update && \
    apk add postgresql-dev gcc python3-dev musl-dev libpq \
    tiff-dev jpeg-dev openjpeg-dev zlib-dev freetype-dev lcms2-dev \
    libwebp-dev tcl-dev tk-dev harfbuzz-dev fribidi-dev libimagequant-dev \
    libxcb-dev libpng-dev gdal gdal-dev geos-dev nodejs npm

# copy project requirments
COPY ./requirements.txt $APP_HOME

# install project requirements
RUN pip install -r requirements.txt

# copy project
COPY . $APP_HOME

# chown all the files to the app user
RUN chown -R app:app $APP_HOME

# change to the app user
USER app

RUN sed -i 's/\r$//g'  $APP_HOME/entrypoint.sh
RUN chmod +x  $APP_HOME/entrypoint.sh

# run entrypoint.prod.sh
ENTRYPOINT ["/home/app/web/entrypoint.sh"]
```

Μόλις φτιάξουμε το docker image φτιάχνουμε το entrypoint.sh.

```
#!/bin/sh

if [ "$DATABASE" = "postgres" ]
then
    echo "Waiting for postgres..."

    while ! nc -z $SQL_HOST $SQL_PORT; do
      sleep 0.1
    done

    echo "PostgreSQL started"
fi

cd crowdsourcing_map_service/frontend || exit
npm install && npm run build

cd ../..
python manage.py collectstatic --noinput
python manage.py migrate

exec "$@"
```

Το script όταν τρέχει περιμένει την postgresql να ενεργοποιηθεί στη συνέχεια μπαίνει στο φάκελο frontend του project και κάνει install όλα τα javascript dependencies. Έπειτα, κάνει build το javascript κώδικα, κάνει collect όλα τα στατικά αρχεία και τέλος τρέχει όλα τα migrations στη βάση δεδομένων.

5.2.5 Docker Container

Έχοντας δημιουργήσει τα βασικά images ήρθε η ώρα να τα ενώσουμε με το docker-compose. Για να το πετύχουμε φτιάχνουμε ένα αρχείο docker-compose.yml που περιγράφει το τρόπο που θα δημιουργηθούν τα images μέσα στο container.

```
version: "3"
services:
  web:
    image: web
    command: gunicorn config.wsgi:application --bind 0.0.0.0:8080
    expose:
      - 8080
    depends_on:
      - pgbouncer
    environment:
      - DJANGO_SETTINGS_MODULE=config.settings.prod
    volumes:
      - static_volume:/home/app/web/staticfiles
      - media_volume:/home/app/web/mediafiles

  celery:
    image: web
    command: celery -A config worker -l info
    depends_on:
      - web
      - redis
    environment:
      - DJANGO_SETTINGS_MODULE=config.settings.prod

  nginx:
    build: nginx/.
    ports:
      - 80:80
      - 443:443
    depends_on:
      - web
    volumes:
      - static_volume:/home/app/web/staticfiles
      - media_volume:/home/app/web/mediafiles

  postgresql:
    build: postgresql/.
    volumes:
      - db-data:/var/lib/postgresql/data
    ports:
      - "5432:5432"
    restart: unless-stopped

  pgbouncer:
    build: pgbouncer/.
    ports:
      - "6432:6432"
    depends_on:
      - postgresql
    restart: unless-stopped

  redis:
    image: redis:alpine
    hostname: redis
    volumes:
      - redis-data:/data
    ports:
```

```
      - 6379:6379
    restart: unless-stopped

volumes:
  db-data:
  redis-data:
  static_volume:
  media_volume:
```

Πρώτα έχουμε το version του docker-compose στο συγκεκριμένο θα χρησιμοποιήσουμε τη version 3. Στη συνέχεια έχουμε τα services που είναι τα images.

Πρώτο έχουμε το web που είναι το Django app. Το web κάνει expose εσωτερικά την πόρτα 8080, έχει dependency το pgbouncer που σημαίνει ότι πρώτα πρέπει να έχουμε δημιουργήσει το pgbouncer και ύστερα το web image, έχει μια environmental μεταβλητή που είναι με τι settings θέλουμε να τρέξουμε την εφαρμογή, δημιουργεί δύο volumes που είναι για τα στατικά αρχεία και για τα media αρχεία και τέλος την εντολή που θέλουμε να τρέξει το image όταν ολοκληρωθεί.

Δεύτερο έχουμε το celery που είναι το task queue μας. Τρέχει με τον ίδιο τρόπο που τρέξαμε το web app το μόνο που αλλάζει είναι το command που το συγκεκριμένο είναι για να τρέξει το celery.

Τρίτο έχουμε το ngnix. Το nginx κάνει expose δύο πόρτες, η μια είναι η πόρτα 80 που είναι το HTTP πρωτόκολλο και η άλλη είναι η 443 που είναι το HTTPS πρωτόκολλο. Μοιράζεται και αυτό τα volumes για τα στατικά και media αρχεία. Ο λόγος είναι ότι το web παράγει τα στατικά και τα media αρχεία και ο nginx τα σερβίρει.

Τέταρτο είναι η PostgreSQL κάνει expose τη πόρτα 5432 που είναι η default της postgres και δημιουργεί το δικό του volume που κρατάει όλα τα δεδομένα των βάσεων.

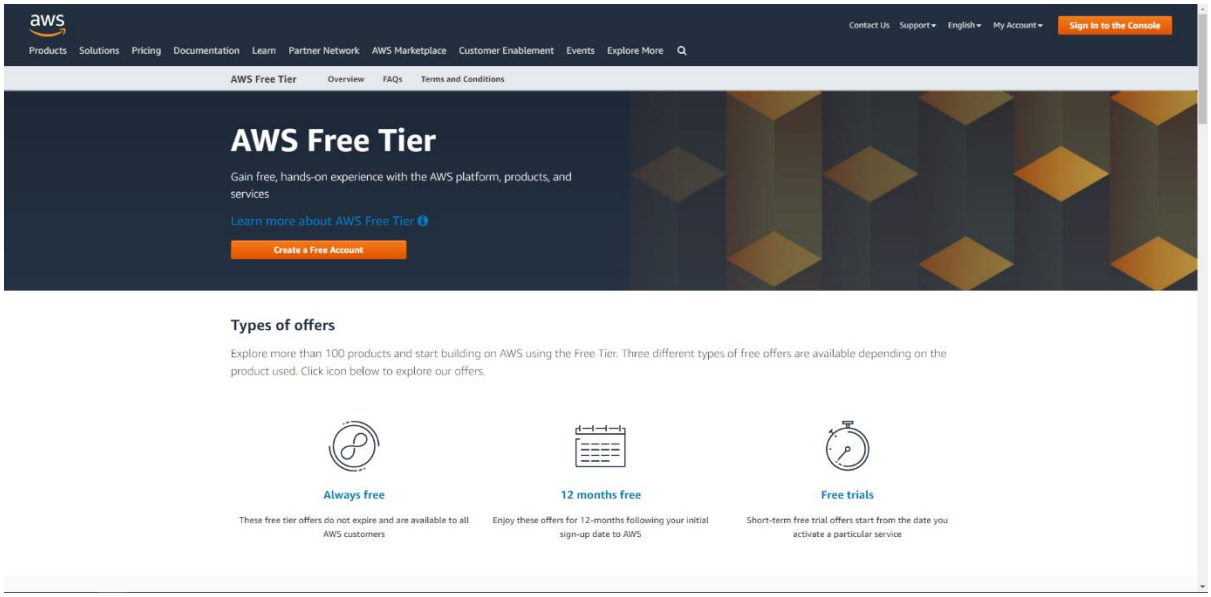
Πέμπτο είναι ο PgBouncer που κάνει expose την πόρτα 6432 και έχει ως dependency την postgres.

Τελευταίο έχουμε τη Redis που είναι η document βάση δεδομένων που τη χρησιμοποιούμε μαζί με το celery για να προγραμματίζουμε tasks και ως caching μηχανισμό.

5.3 Εγκατάσταση της εφαρμογής WebGIS

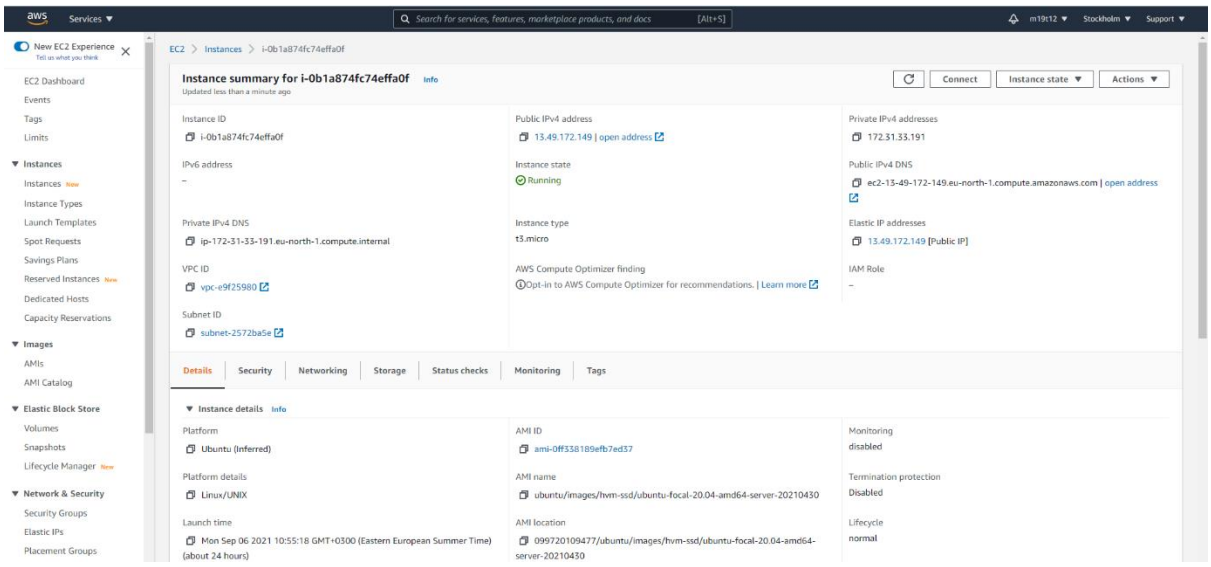
5.3.1 Δημιουργία Server

Για τη δημιουργία server χρησιμοποιήσαμε τις υπηρεσίες της Amazon το free tier. Πρώτα συνδεόμαστε στις υπηρεσίες AWS.



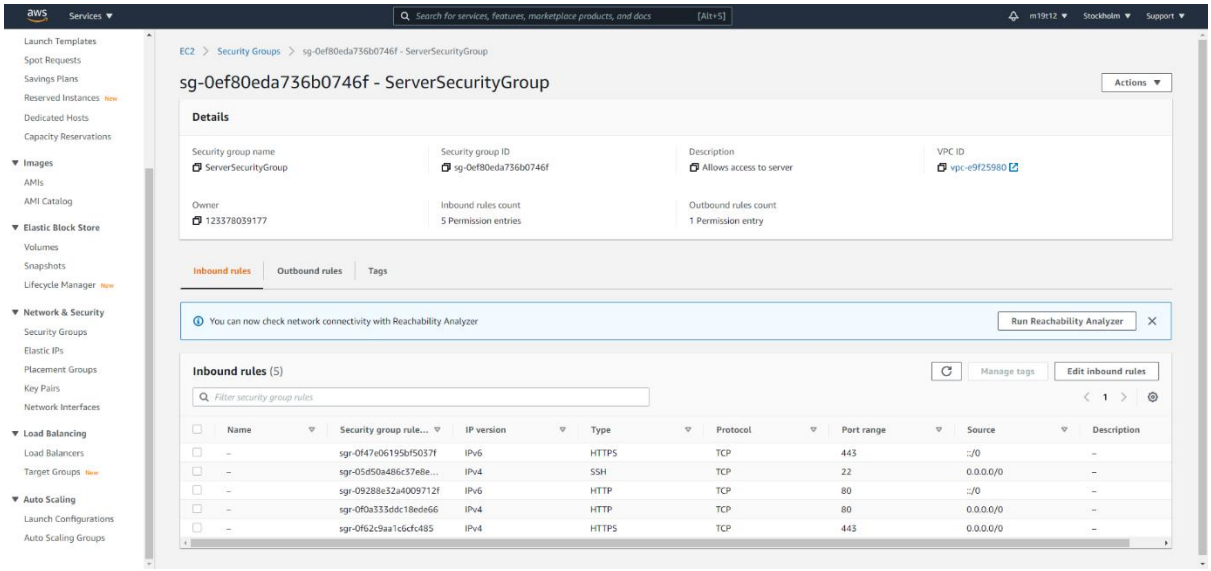
Εικόνα 40 - AWS free tier

Στη συνέχεια πηγαίνουμε στο EC2 και δημιουργούμε ένα μηχάνημα ubuntu 20.04.



Εικόνα 41 - AWS EC2 instance

Μόλις δημιουργήσουμε το EC2 Instance πηγαίνουμε στα security groups για να ανοίξουμε τα κατάλληλα ports για να ώστε να μπορεί ο server μας να επικοινωνεί με το δίκτυο.

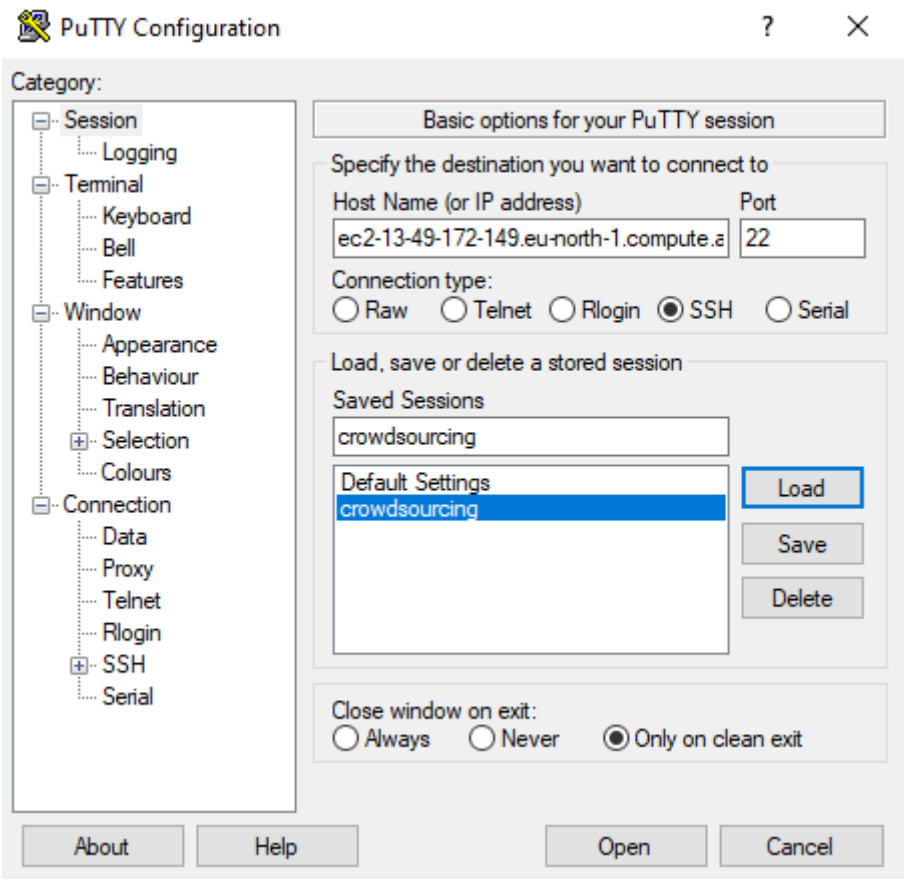


Εικόνα 42 - AWS security groups

Έχουμε ανοίξει τα ports 22 για να μπορούμε να συνδεόμαστε στο μηχάνημα με ssh. Επίσης έχουμε ανοίξει τα ports 80 που είναι το default HTTP και το port 443 που είναι το default HTTPS.

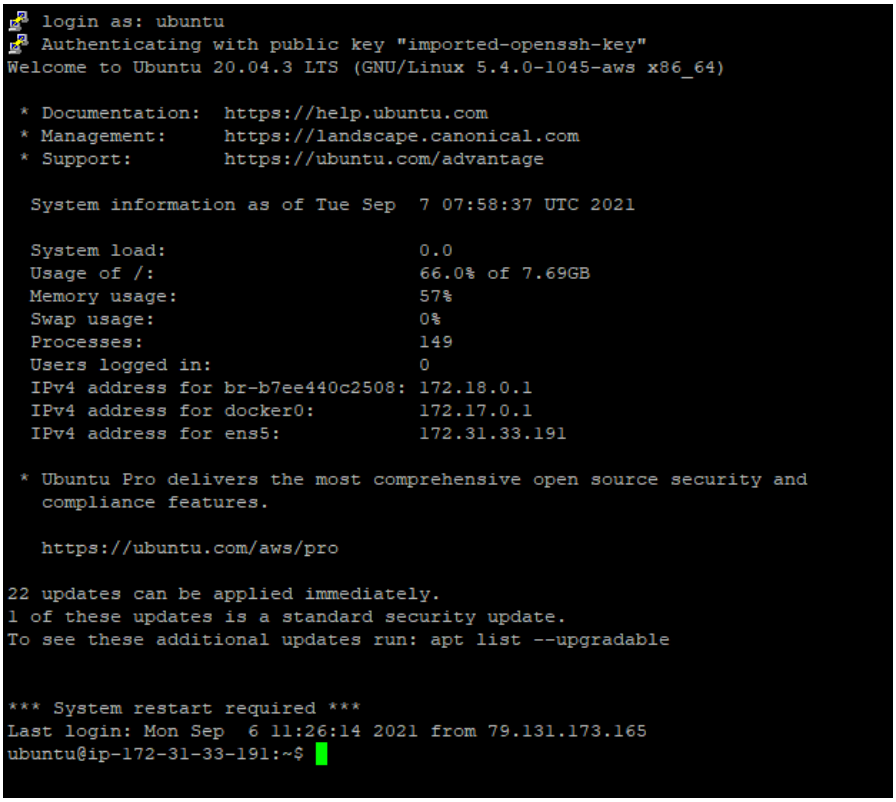
5.3.2 Εγκατάσταση της εφαρμογής στο server

Πρώτα συνδεόμαστε με ssh στο μηχάνημα που δημιουργήσαμε. Για να συνδεθούμε θα χρησιμοποιήσουμε την εφαρμογή PuTTY.



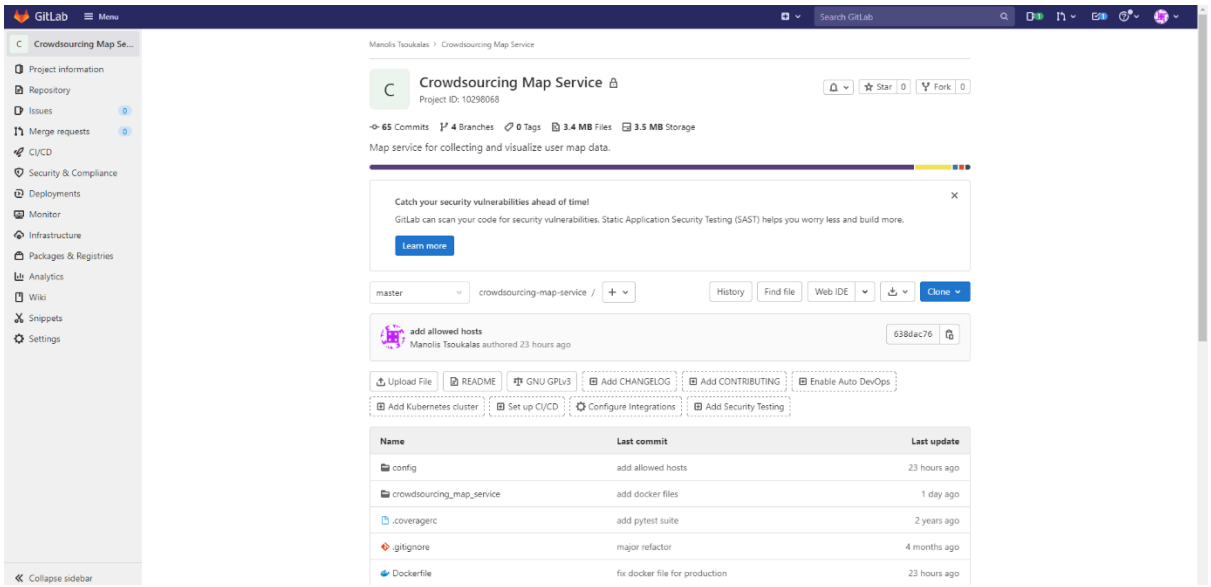
Εικόνα 43 - PuTTY software

Μόλις συνδεθούμε βλέπουμε την παρακάτω οθόνη.



Εικόνα 44 - Server terminal screen

Ο πηγαίος κώδικας της πτυχιακής εργασίας βρίσκεται στην εφαρμογή Gitlab. Μέσω της εφαρμογής μπορούμε να κάνουμε παρακολουθούμε τις αλλαγές του κώδικα και να δημιουργούμε tasks ώστε να έχουμε μια οργάνωση κατά το development της πτυχιακής.



Εικόνα 45 - Κεντρική σελίδα της πτυχιακής εργασίας στο Gitlab

Για να κατεβάσουμε την πτυχιακή εργασία στο server χρησιμοποιούμε την εντολή git pull.

```
git pull https://gitlab.com/m19t12/crowdsourcing-map-service.git
```

Μόλις κατεβάσουμε την εφαρμογή στο server μπαίνουμε στο φάκελο και δημιουργούμε το docker image της εφαρμογής.

```
cd crowdsourcing-map-service
Docker build . -t web
```

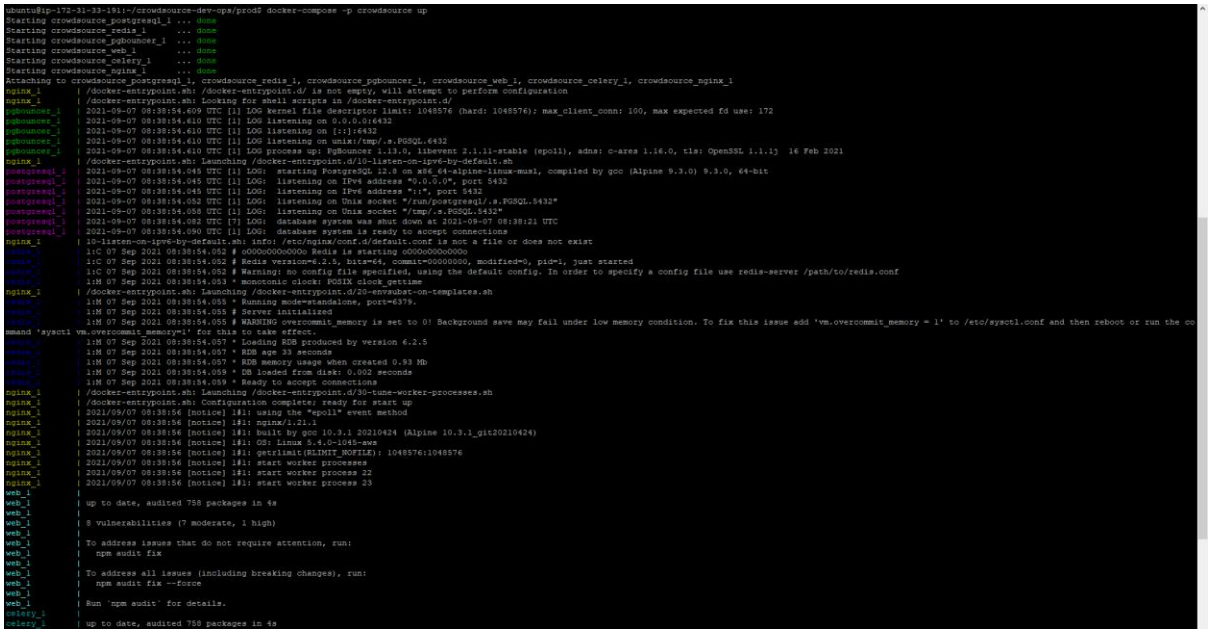
Έχοντας δημιουργήσει το docker image δε χρειαζόμαστε πια τον φάκελο του project οπότε βγαίνουμε από το φάκελο και τον διαγράφουμε. Στη συνέχεια κατεβάζουμε και τα docker αρχεία με όλα τα επιμέρους συστήματα.

```
git clone https://gitlab.com/m19t12/crowdsource-dev-ops.git
```

Μπαίνουμε στο φάκελο και δημιουργούμε το container της εφαρμογής.

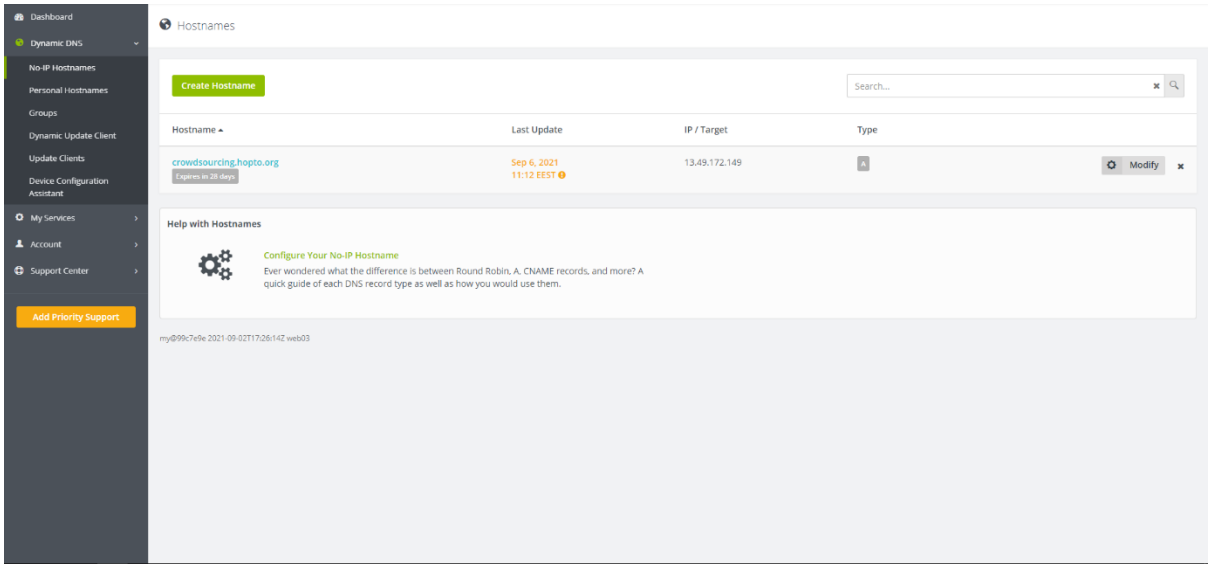
```
cd crowdsource-dev-ops/prod
docker-compose -p crowdsource up
```

Εκτελώντας την εντολή αν όλα τα συστήματα λειτουργήσουν σωστά βλέπουμε τα παρακάτω outputs από το κάθε image του container.



Εικόνα 46 - docker-compose output

Η εφαρμογή μετά από αυτό το βήμα λειτουργεί κανονικά το μόνο κομμάτι που μένει είναι να δημιουργήσουμε ένα domain ώστε οι χρήστες να μπορούν να το βρίσκουν πιο εύκολα. Για να δημιουργήσουμε ένα domain free θα φτιάξουμε λογαριασμό στην εφαρμογή No-Ip.



Εικόνα 47 - Κεντρική σελίδα No-IP

5.4 Ανάλυση κώδικα backend

5.4.1 Ανάλυση μοντέλων της βάσης δεδομένων

Το Django web framework μας δίνει τη δυνατότητα μέσω του συστήματος ORM (Object Relational Mapping) να δημιουργήσουμε πίνακες στη βάση χρησιμοποιώντας απλές python classes.

Το μοντέλο για να δημιουργήσουμε τον χρήστη στο σύστημα είναι το MapUser και το μοντέλο που θα έχει τις πληροφορίες του χρήστη είναι το UserProfile. Παρακάτω θα δούμε την υλοποίηση αυτών των δύο μοντέλων.

```
class MapUser(AbstractBaseUser, BaseModel, PermissionsMixin):
    email = models.EmailField(_('email address'), unique=True)
    first_name = models.CharField(_('first name'), max_length=30)
    last_name = models.CharField(_('last name'), max_length=150)
    is_staff = models.BooleanField(
        _('staff status'),
        default=False,
        help_text=_('Designates whether the user can log into this admin
site.'),
    )

    objects = MapUserManager()

    USERNAME_FIELD = 'email'
    REQUIRED_FIELDS = ['first_name', 'last_name']

    def __str__(self):
        return f'{self.email}'


class UserProfile(models.Model):
    user = models.OneToOneField(settings.AUTH_USER_MODEL,
on_delete=models.CASCADE, related_name='profile')
    profile_image = models.ImageField(upload_to=user_upload_path, blank=True,
null=True)
    address = models.CharField(max_length=256, blank=True, null=True)
    country = models.CharField(max_length=128, blank=True, null=True)
    city = models.CharField(max_length=128, blank=True, null=True)
    zip = models.CharField(max_length=16, blank=True, null=True)
```

Το μοντέλο MapUser κάνει override τον default manager και έχει τον δικό μας που είναι ο MapUserManager. Ο MapUserManager έχει μεθόδους δημιουργίας απλού χρήστη και superuser.

```
class MapUserManager(UserManager):
    def _create_user(self, email, password, **extra_fields):
        """
        Create and save a user with the given email and password.
        """
        if not email:
            raise ValueError('The given email must be set')

        email = self.normalize_email(email)
        user = self.model(email=email, **extra_fields)
        user.set_password(password)
        user.save(using=self._db)

        return user

    def create_user(self, email, password=None, **extra_fields):
        extra_fields.setdefault('is_staff', False)
        extra_fields.setdefault('is_superuser', False)
        return self._create_user(email, password, **extra_fields)

    def create_superuser(self, email, password=None, **extra_fields):
        extra_fields.setdefault('is_staff', True)
        extra_fields.setdefault('is_superuser', True)

        if extra_fields.get('is_staff') is not True:
            raise ValueError('Superuser must have is_staff=True.')
        if extra_fields.get('is_superuser') is not True:
            raise ValueError('Superuser must have is_superuser=True.')

        return self._create_user(email, password, **extra_fields)
```

Στη συνέχεια έχουμε το μοντέλο του χάρτη. Στο μοντέλο Map υπάρχει το πεδίο slug που εκεί ο χρήστης μπορεί να δώσει το δικό του αναγνωριστικό για το χάρτη που θα φτιάξει. Το πεδίο slug είναι unique ώστε δύο χρήστες να μην μπορούν να δώσουν το ίδιο όνομα.

```
class Map(BaseModel, MultiTenantEnable, DescriptionEnable):
    slug = models.SlugField(max_length=25, default='default-slug',
                            help_text='this is for accessing in url',
                            unique=True)

    def __str__(self):
        return f'{self.title}'

    class Meta:
        indexes = [
            models.Index(fields=['slug'], name='core_map_slug_idx')
        ]
```

Το μοντέλο MapPost είναι το μοντέλο που κρατάει τη βασική πληροφορία που θα έχουν οι χάρτες. Όταν ο χρήστης δημιουργήσει ένα Point Of Interest (POI) ουσιαστικά δημιουργεί μια εγγραφή MapPost στη βάση.

Τα πεδία που έχει το MapPost μοντέλο είναι το point που υποδηλώνει το σημείο στο χάρτη που θα βάλουμε το POI και είναι πεδίο με x, y συντεταγμένες. Έχουμε το media_type που είναι ο τύπος αναπαραστάσεις (video, image, audio), έχουμε το ίδιο το media που παίρνει ως όρισμα το φάκελο που θα το ανεβάσει ο χρήστης και τέλος το τύπο του POI και για ποιο χάρτη απευθύνεται. Στο μοντέλο έχουμε και δύο properties, το useful που μας επιστρέφει πόσοι χρήστες έχουν βρει αυτό το POI χρήσιμο και τα comments που μας επιστρέφει όλα τα comments που υπάρχουν για αυτό το POI.

```
class MapPost(BaseModel, MultiTenantEnable, DescriptionEnable):
    MEDIA_TYPE = (
        ('video', 'Video'),
        ('image', 'Image'),
        ('audio', 'Audio')
    )

    point = models.PointField()
    media_type = models.CharField(max_length=128, choices=MEDIA_TYPE,
default='image')
    media = models.FileField(upload_to=user_upload_path, blank=True,
null=True)
    map_post_type = models.ForeignKey('MapPostType',
on_delete=models.CASCADE)
    map = models.ForeignKey('Map', null=True, on_delete=models.CASCADE)

    @property
    def useful(self):
        return self.useful_set.all().count()

    @property
    def comments(self):
        return self.comment_set.all()

    def media_preview(self):
        if self.media_type == 'image':
            return mark_safe(f'<img src="{self.media.url}" />')

    media_preview.allow_tags = True

    def __str__(self):
        return f'{self.title}'
```

Έχουμε τη δυνατότητα όταν φτιάξουμε ένα χάρτη να δημιουργήσουμε τύπους που ομαδοποιούν τα POIs. Το μοντέλο αυτό ονομάζεται MapPostType και έχει δύο πεδία, το icon που είναι το εικονίδιο που θέλουμε να αναπαραστήσουμε τα POIs αυτού του τύπου πάνω στο χάρτη και το map που είναι για ποιο χάρτη αναφέρεται αυτός ο τύπος.

```
class MapPostType(BaseModel, DescriptionEnable):
    icon = models.CharField(blank=True, null=True, max_length=128)
    map = models.ManyToManyField(Map)

    def __str__(self):
        return f'{self.title}'
```

Επίσης, έχουμε τη δυνατότητα να κάνουμε comments πάνω σε ένα POI και να αναφέρουμε αν μας φάνηκε χρήσιμο. Τα μοντέλα που υλοποιούν αυτή τη δυνατότητα είναι το Comment και το Useful.

```
class Comment(BaseModel, MultiTenantEnable):
    body = models.TextField(max_length=2048)
    map_post = models.ForeignKey(MapPost, on_delete=models.CASCADE)

    class Meta:
        ordering = ['-created']


class Useful(BaseModel, MultiTenantEnable):
    map_post = models.ForeignKey(MapPost, on_delete=models.CASCADE,
null=True, blank=True)
    comment = models.ForeignKey(Comment, on_delete=models.CASCADE, null=True,
blank=True)

    def clean(self):
        if not self.map_post and not self.comment:
            raise ValidationError(_('Useful must point to a map_post or a '
                                    'comment cant be both null.'))

        if self.map_post and self.comment:
            raise ValidationError(_('You cant point to both '
                                    'map_post and comment.'))

        useful_post = Useful.objects.filter(map_post__useful__user=self.user)
        useful_comment =
Useful.objects.filter(comment__useful__user=self.user)

        if len(useful_post) != 0 and self.map_post:
            raise ValidationError(_('You have already rate this post.'))

        if len(useful_comment) != 0 and self.comment:
            raise ValidationError(_('You have already rate this comment'))
```

Μέσα στη μέθοδο clean μπορούμε να γράψουμε κάποιους ελέγχους πριν γίνει η αποθήκευση.

Τα παραπάνω μοντέλα κάνουν implement κάποια βασικά abstract μοντέλα. Έχουμε το BaseModel που εκεί έχουμε τα πεδία id που είναι το unique χαρακτηριστικό κάθε εγγραφής και είναι και το primary key μας, το created που είναι το πότε δημιουργήσαμε αυτή την εγγραφή στη βάση, το modified που είναι το πότε κάναμε τελευταία αλλαγή σε αυτή την εγγραφή και τέλος το is_active που μας υποδηλώνει αν είναι ενεργή μια εγγραφή. Στη συνέχεια έχουμε το μοντέλο MultiTenantEnable που έχει για πεδίο το χρήστη και Τέλος το μοντέλο DescriptionEnable που έχει για πεδία το title και το description.

```
class BaseModel(models.Model):
    """An abstract base class model that provides
    unique id across all tables,
    if is active and the time when created and modified.
    """
    id = fields.UIDField(primary_key=True)
    created = models.DateTimeField(auto_now_add=True)
    modified = models.DateTimeField(auto_now=True)
    is_active = models.BooleanField(default=True)

    class Meta:
        abstract = True


class MultiTenantEnable(models.Model):
    """An abstract base class model that provides the user.
    """
    user = models.ForeignKey(settings.AUTH_USER_MODEL,
on_delete=models.CASCADE, null=True)

    class Meta:
        abstract = True


class DescriptionEnable(models.Model):
    """An abstract base class model that provides
    title and description.
    """
    title = models.CharField(max_length=256)
    description = models.TextField(blank=True, null=True)

    class Meta:
        abstract = True
```

5.4.2 Ανάλυση API

Η εφαρμογή μας για να μπορέσει να επικοινωνήσει με το frontend ή με τρίτες εφαρμογές χρησιμοποιεί το API. Το API έχει δύο end-points το ένα είναι το private που το χρησιμοποιεί αποκλειστικά το frontend και είναι για εσωτερική χρήση και το V1 που το χρησιμοποιεί το frontend και τυχόν τρίτες εφαρμογές που θέλουν να επικοινωνήσουν με την εφαρμογή μας.

Το public api μας είναι versioned γι’ αυτό χρησιμοποιούμε τη λέξη V1. Σε περίπτωση που κάνουμε αλλαγές στο API μας για να μη σπάσουμε τις εφαρμογές που είδη το χρησιμοποιούν χρησιμοποιούμε το versioning για να γνωρίζουν για τυχών αλλαγές. Παρακάτω θα δούμε τη δήλωση των urls του private, v1 και του documentation του v1.

```
v1_api = routers.SimpleRouter()
v1_api.register(r'maps', MapViewSet)
v1_api.register(r'map-posts', MapPostViewSet)
v1_api.register(r'comments', CommentViewSet)
v1_api.register(r'map-posts-types', MapPostTypeViewSet)
v1_api.register(r'users', UserViewSet)


private_api = routers.SimpleRouter()
private_api.register(r'auth', AuthViewSet)

# Redoc OpenAPI documentation
schema_view = get_schema_view(
    title='Crowdsourcing map service API',
    url='https://www.example.org/api/',
    patterns=v1_api.urls,
    public=True,
    permission_classes=(permissions.AllowAny,),
    renderer_classes=[JSONOpenAPIRenderer]
)


urlpatterns = [
    path('v1/', include((v1_api.urls, 'crowdsourcing_map_service.api.v1'),
                        namespace='v1')),
    path('v1/schema/', schema_view, name='openapi-schema'),
    path('v1/redoc/', APIDocumentation.as_view(), name='redoc'),
    path('private/', include((private_api.urls,
                              'crowdsourcing_map_service.api.private'),
                             namespace='private'))
]
```

Στη συνέχεια θα δούμε τα views που χρησιμοποιεί το API. Ξεκινώντας από το private έχουμε το AuthViewSet. Τα ViewSet είναι ειδικές κλάσεις του Django για να δημιουργούμε ομαδοποιήσεις από endpoints. Κάθε μέθοδος του viewset είναι και ένα endpoint. Το AuthViewSet έχει τη μέθοδο login που καλή μια βοηθητική μέθοδο την login_user.

```
def login_user(request, serializer_class, email, password):
    user = authenticate(request, email=email, password=password)
    if user:
        # login saves the user's ID in the session
        # using Django's session framework
        login(request, user)

        token = jwt_encode_handler(jwt_payload_handler(user))

        data = {
            'token': token,
            'user': user
        }

        payload = serializer_class(instance=data)

        return Response(data=payload.data, status=status.HTTP_200_OK)
    return Response(data={"error": "Authentication error. "
                                   "Email or password is wrong."},
                    status=status.HTTP_401_UNAUTHORIZED)
```

Έχει την logout μέθοδο που κάνει logout το χρήστη, την user που μας δίνει τον logged in χρήστη και τέλος τη registration που καλή τη βοηθητική μέθοδο register_user.

```
def register_user(serializer_class, user_data):
    payload = serializer_class(data=user_data)
    payload.is_valid(raise_exception=True)

    password = payload.validated_data.pop('password')

    user = MapUser(**payload.validated_data)
    user.set_password(password)

    user.full_clean()
    user.save()

    activation_token = account_activation_token.make_token(user)

    send_registration_email.delay(user.pk, user.first_name, user.last_name,
                                  user.email, activation_token)
```

```
class AuthViewSet(ViewSet):
    queryset = MapUser.objects.all()
    permission_classes = (AllowAny,)

    @action(methods=['post'], detail=False)
    def login(self, request):
        return login_user(request=request,
                          serializer_class=TokenSerializer,
                          email=request.data.get('email', ''),
                          password=request.data.get('password', ''))

    @action(methods=['post'], detail=False)
    def logout(self, request):
        if getattr(settings, 'REST_SESSION_LOGIN', True):
            logout(request)

        return Response({"detail": _('Successfully logged out.')},
                        status=status.HTTP_200_OK)

    @action(methods=['get'], detail=False,
            permission_classes=[IsAuthenticated])
    def user(self, request):
        user = BaseUserSerializer(request.user).data
        return Response(user, status=status.HTTP_200_OK)

    @action(methods=['post'], detail=False)
    def registration(self, request):
        register_user(serializer_class=RegisterUserSerializer,
                      user_data=request.data)

        return Response(status.HTTP_201_CREATED)
```

Στη συνέχεια έχουμε τα views του v1. Το Django web framework μας δίνει τη δυνατότητα να δημιουργούμε views που βασίζονται πάνω σε ένα μοντέλο της βάσης δεδομένων. Τα views αυτά κάνουν implement όλο το CRUD functionality. Πάνω σε αυτά τα view μπορούμε να δηλώσουμε ποια πεδία θέλουμε να φιλτράρουμε να κάνουμε order ή να ψάξουμε. Παρακάτω θα δούμε τα views για το κάθε ένα από τα μοντέλα μας.

```
class MapViewSet(StaffBrowsableAPIMixin, ModelViewSet):
    queryset = Map.objects.all()
    permission_classes = (IsAuthenticatedOrReadOnly, IsOwnerOrReadOnly,)
    serializer_class = BaseMapSerializer
    filter_fields = ('id', 'is_active', 'slug',)
    search_fields = ('title', 'description', 'slug',)
    ordering_fields = ('id', 'created', 'modified', 'is_active',)
    ordering = ('created',)
```

```
class MapPostTypeViewSet(ModelViewSet):
    queryset = MapPostType.objects.all()
    permission_classes = (IsAuthenticatedOrReadOnly, IsOwnerOrReadOnly,)
    serializer_class = BaseMapPostTypeSerializer
    filter_fields = ('id', 'is_active', 'map',)
```

```
class MapPostViewSet(ModelViewSet):
    queryset = MapPost.objects.all()
    permission_classes = (IsAuthenticatedOrReadOnly, IsOwnerOrReadOnly,)
    filter_fields = ('id', 'is_active', 'media_type', 'map_post_type', 'map')
    search_fields = ('title', 'description',)
    ordering_fields = ('id', 'created', 'modified', 'is_active',)
    ordering = ('created',)
    bbox_filter_field = 'point'

    @action(methods=['post'], detail=True)
    def comment(self, request, pk=None):
        map_post = save_comment(serializer_class=BaseCommentSerializer,
                                body=request.data.get('body'),
                                user=request.user,
                                map_post_id=pk)

        return Response(data=MapPostDetailSerializer(map_post).data,
                        status=status.HTTP_201_CREATED)

    @action(methods=['post', 'delete'], detail=True)
    def useful(self, request, pk=None):
        if request.method == 'POST':
            map_post = save_useful(
                serializer_class=BaseUsefulSerializer,
                user=request.user,
                map_post_id=pk
            )

            map_post_serializer = MapPostDetailSerializer(map_post, context={
                'request': request})

            return Response(data=map_post_serializer.data,
                            status=status.HTTP_201_CREATED)

        if request.method == 'DELETE':
            map_post = delete_useful(
                serializer_class=BaseUsefulSerializer,
                user=request.user,
                map_post_id=pk
            )

            map_post_serializer = MapPostDetailSerializer(map_post, context={
                'request': request})

            return Response(data=map_post_serializer.data,
                            status=status.HTTP_201_CREATED)

    def get_serializer_class(self):
        if self.action == 'list':
            return MapPostListSerializer
        return MapPostDetailSerializer
```

```
class CommentViewSet(ModelViewSet):
    queryset = Comment.objects.all()
    permission_classes = (IsAuthenticatedOrReadOnly, IsOwnerOrReadOnly,)
    serializer_class = BaseCommentSerializer
    filter_fields = ('id', 'is_active', 'user', 'map_post',)
    search_fields = ('body',)
    ordering_fields = ('id', 'created', 'modified', 'is_active',
                       'map_post', 'user',)
    ordering = ('created',)


class UsefulViewSet(ModelViewSet):
    queryset = Useful.objects.all()
    permission_classes = (IsAuthenticatedOrReadOnly, IsOwnerOrReadOnly,)
    serializer_class = BaseUsefulSerializer
    filter_fields = ('id', 'is_active', 'user', 'map_post',)
    ordering_fields = ('id', 'created', 'modified', 'is_active',
                       'map_post', 'user',)
    ordering = ('created',)
```

Όλα τα views μας χρησιμοποιούνε τους serializers. Οι serializers παίρνουν τα fields από το μοντέλο της βάσεις και τα μεταφράζουν σε dictionary δεδομένα και από εκεί σε JSON για να μπορούμε να τα χρησιμοποιήσουμε στο frontend. Παρακάτω βλέπουμε τον serializer του MapPost.

```
class BaseMapPostSerializer(serializers.ModelSerializer):
    user = BaseUserSerializer(read_only=True)
    map = serializers.PrimaryKeyRelatedField(queryset=Map.objects.all())
    post_type = serializers.SerializerMethodField()
    point = GeometryField()

    def create(self, validated_data):
        validated_data['user'] = self.context['request'].user
        instance = MapPost.objects.create(**validated_data)
        return instance

    def get_post_type(self, instance):
        return BaseMapPostTypeSerializer(instance.map_post_type).data

    class Meta:
        model = MapPost
        fields = ('is_active', 'id', 'created', 'modified', 'title',
                  'description', 'post_type', 'point', 'media_type', 'media',
                  'user', 'useful', 'map_post_type', 'map',)
        extra_kwargs = {'map_post_type': {'write_only': True}}
```

5.4.3 Ανάλυση task queue

Η εφαρμογή μας χρησιμοποιεί τη βιβλιοθήκη celery για να δημιουργούμε tasks σε ουρά. Όταν ένας χρήστης μας δημιουργεί πρώτη φορά λογαριασμό του έρχεται ένα email για να επιβεβαιώσει την εγγραφή του. Για να μη δημιουργήσουμε καθυστέρηση στην εφαρμογή μέχρι να σταλθεί το email δημιουργούμε ένα task και το βάζουμε σε μια ουρά με tasks για να εκτελεστούν όταν μπορέσει το σύστημα. Παρακάτω θα δούμε το κώδικα που αρχικοποιούμε το celery και το κώδικα για το task που στέλνει το registration email.

```
if not settings.configured:
    # set the default Django settings module for the 'celery' program.
    os.environ.setdefault('DJANGO_SETTINGS_MODULE', 'config.settings.dev')

app = Celery('crowdsourcing')

# Using a string here means the worker doesn't have to serialize
# the configuration object to child processes.
# - namespace='CELERY' means all celery-related configuration keys
#   should have a `CELERY_` prefix.
app.config_from_object('django.conf:settings', namespace='CELERY')

# Load task modules from all registered Django app configs.
app.autodiscover_tasks()
```

```
@app.task
def send_registration_email(user_uid, first_name, last_name, email, token):
    subject = 'Crowdsourcing Map Service Account Activation'
    from_email = 'account@crowdsourcemapservice.com'
    to = email
    text_content = 'Thank you for registering, please click the link below '
\
                   'to activate your account.'
    html_content = render_to_string('email/registration-complete.html',
                                    {'uid': user_uid, 'first_name':
first_name,
                                     'last_name': last_name, 'token': token})
    message = EmailMultiAlternatives(subject, text_content, from_email, [to])
    message.attach_alternative(html_content, "text/html")
    message.send()
```

5.5 Ανάλυση κώδικα frontend

5.5.1 Templates

Το Django web framework μας δίνει τη δυνατότητα να δημιουργήσουμε επαναχρησιμοποιούμενα html αρχεία. Μπορούμε να δημιουργήσουμε base html αρχεία και να τα κληρονομούν διαφορετικά html αρχεία. Για base αρχείο έχουμε το base.html. Στο base.html δηλώνουμε το title της σελίδας που το παίρνουμε από μεταβλητή, δηλώνουμε τα extra που χρειάζεται για να δουλέψει το PWA και τα βασικά CSS και JavaScript αρχεία.

```
{% load webpack_static %}
{% load pwa_extras %}
```

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta http-equiv="content-type" content="text/html;charset=UTF-8"/>
    <meta charset="utf-8"/>
    <title>{% block page-title %}{% endblock %}</title>
    <meta name="viewport" content="width=device-width, initial-scale=1.0,
maximum-scale=1.0, user-scalable=no, shrink-to-fit=no"/>

    {% load_manifest %}

    <link rel="apple-touch-icon" href="{% webpack_static
'img/icons/favicon.png' %}">
    <link rel="apple-touch-icon" sizes="76x76" href="{% webpack_static
'img/icons/favicon.png' %}">
    <link rel="apple-touch-icon" sizes="120x120" href="{% webpack_static
'img/icons/favicon.png' %}">
    <link rel="apple-touch-icon" sizes="152x152" href="{% webpack_static
'img/icons/favicon.png' %}">
    <link rel="icon" type="image/x-icon" href="{% webpack_static
'img/icons/favicon.png' %}"/>

    <meta name="apple-mobile-web-app-capable" content="yes">
    <meta name="apple-touch-fullscreen" content="yes">
    <meta name="apple-mobile-web-app-status-bar-style" content="default">
    <meta content="CrowdSource Map Service" name="description"/>
    <meta content="Manolis Tsoukalas" name="author"/>

    <link rel="stylesheet" href="{% webpack_static
'bootstrap/css/bootstrap.min.css' %}" type="text/css">
    <link rel="stylesheet" href="{% webpack_static 'font-awesome/css/font-
awesome.min.css' %}" type="text/css">
    <link rel="stylesheet" href="{% webpack_static 'animate/animate.css' %}">
    <link rel="stylesheet" href="https://cdn.jsdelivr.net/npm/bootstrap-
select@1.13.14/dist/css/bootstrap-select.min.css">

    {% block css-libraries %}{% endblock %}

</head>
```

```
<body>

{% block page-content %}{% endblock %}

<script src="{% webpack_static 'jquery/jquery-3.3.1.min.js' %}"
type="text/javascript"></script>
<script src="{% webpack_static 'popper/popper.min.js' %}"
type="text/javascript"></script>
<script src="{% webpack_static 'bootstrap/js/bootstrap.min.js' %}"
type="text/javascript"></script>
<script src="https://cdn.jsdelivr.net/npm/bootstrap-
select@1.13.14/dist/js/bootstrap-select.min.js"></script>
{% load_worker %}
{% block server-context %}{% endblock %}
{% block js-libraries %}{% endblock %}
</body>
</html>
```

Έπειτα έχουμε την index.html που είναι το html αρχείο της κεντρικής σελίδας μας.

```
{% extends 'partials/base.html' %}
{% load webpack_static  %}

{% block page-title %}
    {% if object %}
        {{ object.title }}
    {% else %}
        Crowdsourcing Map Service
    {% endif %}
{% endblock %}

{% block css-libraries %}
    <link rel="stylesheet" href="{% webpack_static  'mapbox/mapbox-gl.css'
%}">
    <link rel="stylesheet" href="{% webpack_static  'mapbox-geocoder/mapbox-
gl-geocoder.css' %}">
    <link rel="stylesheet" href="{% webpack_static  'crowdsourcing-map-
service-theme/css/crowdsourcing.css' %}">
{% endblock %}

{% block page-content %}
    <div id="app"></div>
{% endblock %}

{% block server-context %}
    <script type="text/javascript">
        const map_title = "{{ object.title }}";
        const map_id = "{{ object.id }}";
    </script>
{% endblock %}

{% block js-libraries %}
    <script src="{% webpack_static  'mapbox-geocoder/mapbox-gl-
geocoder.min.js' %}"></script>
    <script type="text/javascript" src="{% webpack_static 'vendors.js'
%}"></script>
    <script type="text/javascript" src="{% webpack_static 'runtime.js'
%}"></script>
    <script type="text/javascript" src="{% webpack_static 'app.js'
%}"></script>
{% endblock %}
```

5.5.2 React

Για τη λειτουργικότητα του frontend έχουμε χρησιμοποιήσει τη βιβλιοθήκη React.js. Η React μας βοηθάει να δημιουργήσουμε reusable components σε JavaScript. Αρχιτεκτονικά έχουμε ένα κεντρικό container που έχει όλα τα stores. Μέσα από αυτό κάνουμε render τα components και μεταφέρουμε την πληροφορία από το πατερά στα παιδιά. Στη δικιά μας περίπτωση το κεντρικό container είναι το App. Το App έχει τις μεθόδους για να φορτώσουμε POIs στο χάρτη, να δείξουμε τις κατηγορίες που έχει ο συγκεκριμένος χάρτης και να δείξουμε τις λεπτομέρειες του POI που έχει διαλέξει ο χρήστης να δει.

Στη συνέχεια έχουμε το NavBar component που είναι υπεύθυνο για την μπάρα που βρίσκεται στο πάνω μέρος της εφαρμογής μας, το MapComponent που δείχνει το χάρτη, το Sidebar που είναι για την πλάγια μπάρα που περιέχει τα φίλτρα τη search μπάρα τους χάρτες και τα POIs, τα modals που χρησιμοποιούμε όταν φτιάχνουμε χάρτες, POIs και φίλτρα και

τέλος το MobileBanner που εμφανίζεται όταν επισκεπτόμαστε την εφαρμογή από κινητό και μας ρωτάει αν θέλουμε να την εγκαταστήσουμε.

```
const mapPostService = new CrowdSourceAPIService();

function App(){
    const [user, setUser] = useState(null);
    const [mapPosts, setMapPosts] = useState([]);
    const [detailPost, setDetailPost] = useState(null);
    const [location, setLocation] = useState(null);
    const [sideBarIsOpen, setSideBarIsOpen] = useState(false);
    const [mapPostTypes, setMapPostTypes] = useState([]);
```

```
function loadMapPosts() {
    mapPostService.getMapPosts().done(response => {
        setMapPosts(response.results);
    }).fail(error => {
        console.log(error);
    });
}

function loadMapPostTypes(all){
    mapPostService.getMapPostTypes(all).done(response => {
        setMapPostTypes(response.results);
    }).fail(error => console.log(error));
}

function showMapPost(id) {
    if (id) {
        mapPostService.getMapPost(id).done(response => {
            setDetailPost(response);
        });
        setSideBarIsOpen(true);
    } else {
        setDetailPost(null);
        mapPostService.setQuery('map_post_type', '');
        loadMapPosts();
    }
}

function toggleSideBar() {
    setSideBarIsOpen(!sideBarIsOpen);
}
```

```
return (
            <>
                <UserContext.Provider value={user}>
                    <NavBar setUser={setUser}/>
                    <MapComponent
                        loadMapPosts={loadMapPosts}
                        mapPosts={mapPosts}
                        showMapPost={showMapPost}
                        location={location}
                    />
                    <Sidebar
                        loadMapPosts={loadMapPosts}
                        showMapPost={showMapPost}
                        updateMapPost={setDetailPost}
                        showLocation={setLocation}
                        toggleSideBar={toggleSideBar}
                        mapPosts={mapPosts}
                        detailPost={detailPost}
                        sideBarIsOpen={sideBarIsOpen}
                    />
                    <MapModal
                        mapPostTypes={mapPostTypes}
                        loadMapPostTypes={loadMapPostTypes}/>
                    <MapPostTypeModal loadMapPostTypes={loadMapPostTypes}/>
                    <MapPostModal
                        toggleSideBar={toggleSideBar}
                        loadMapPosts={loadMapPosts}/>
                    <MobileBanner/>
                </UserContext.Provider>
            </>
        );

}


const wrapper = document.getElementById('app');
wrapper ? ReactDom.render(<App/>, wrapper) : false;
```

Από όλα τα components που έχουμε, το MapComponent έχει το πιο σημαντικό ρόλο. Μέσω του MapComponent δείχνουμε το χάρτη, παίρνουμε το location του χρήστη και φορτώνουμε και δείχνουμε τα POIs.

```
function createMarkerHTML(icon) {
    const markerElement = document.createElement('div');
    const marker_icon = document.createElement('i')
    markerElement.className = 'crowd-marker';
    marker_icon.className = `crowd-marker-icon ${icon}`;
    markerElement.appendChild(marker_icon)
    return markerElement
}

function initializeMap() {
    let map;

    if (!mapboxgl.supported()) {
        swal.fire('Error Loading Map', 'Your browser does not support Mapbox
GL', 'error');
    } else {
        mapboxgl.accessToken =
'pk.eyJ1IjoibTE5dDEyIiwiYSI6ImNqc2Fidnh2NTF2bHk0NHFwbGFkdGZneTAifQ.E_0U42_BRG
hTF9ier4YAog';

        map = new mapboxgl.Map({
            container: 'map',
            style: 'mapbox://styles/mapbox/streets-v11'
        })
    }

    return map;
}
```

```
export class MapComponent extends React.Component {
    constructor(props) {
        super(props);

        this.state = {
            userPosition: []
        };

        this.markers = [];

        this.mapPostService = new CrowdSourceAPIService();
    }

    componentDidMount() {
        //Initialize Map
        this.map = initializeMap();

        if (this.map) {
            //Get user Location
            getUserLocation().then((position) => {
                this.setState({
                    userPosition: [
                        position.coords.longitude,
                        position.coords.latitude
                    ]
                });
                this.flyMap(this.state.userPosition)
            }).catch((error) => {
                showLocationError(error);
            });

            this.map.on('moveend', () => {
                let mapBounds = this.map.getBounds().toArray();
                if (mapBounds.length) {
                    let southwest = this.map.getBounds().toArray()[0];
                    let northeast = this.map.getBounds().toArray()[1];
                    debounce(this.setMapPostBounds(southwest, northeast),
300)
                }
            });
        }
    }
```

```
componentWillUnmount(){
        this.map.remove();
    }

    componentDidUpdate(prevProps, prevState, snapshot) {
        if (prevProps.location !== this.props.location)
            this.flyMap(this.props.location.center);

        if (this.compareMapPosts(prevProps, this.props)) {
            this.loadMarkers();
        }
    }

    setMapPostBounds(southwest, northeast) {
        this.mapPostService.setQuery('in_bbox',
`${southwest.join()},${northeast.join()}`);
        this.props.loadMapPosts();
    }

    compareMapPosts(prevProps, props) {
        let prevMapPosts = prevProps.mapPosts.map(post => post.id).sort();
        let mapPosts = props.mapPosts.map(post => post.id).sort();

        return JSON.stringify(prevMapPosts) !== JSON.stringify(mapPosts);
    }

    loadMarkers() {
        this.clearMarkers();

        this.props.mapPosts.forEach((post) => {
            const {post_type, point} = post;

            let markerElement = createMarkerHTML(post_type['icon']);

            let marker = new mapboxgl.Marker({
                rotation: -45,
                element: markerElement
            }).setLngLat(point.coordinates).addTo(this.map);

            this.createPopup(post, marker);

            this.markers.push(marker);
        });
    }

    createPopup(post, marker) {
        const placeholder = document.createElement('div');
        ReactDOM.render(<MapPopup post={post}
showMapPost={this.props.showMapPost}/>, placeholder);

        let infoPopup = new mapboxgl.Popup({
            offset: [0, -20],
            className: 'map-service-popup'
        }).setLngLat(post.point.coordinates).setDOMContent(placeholder);

        marker.setPopup(infoPopup);
    }
```

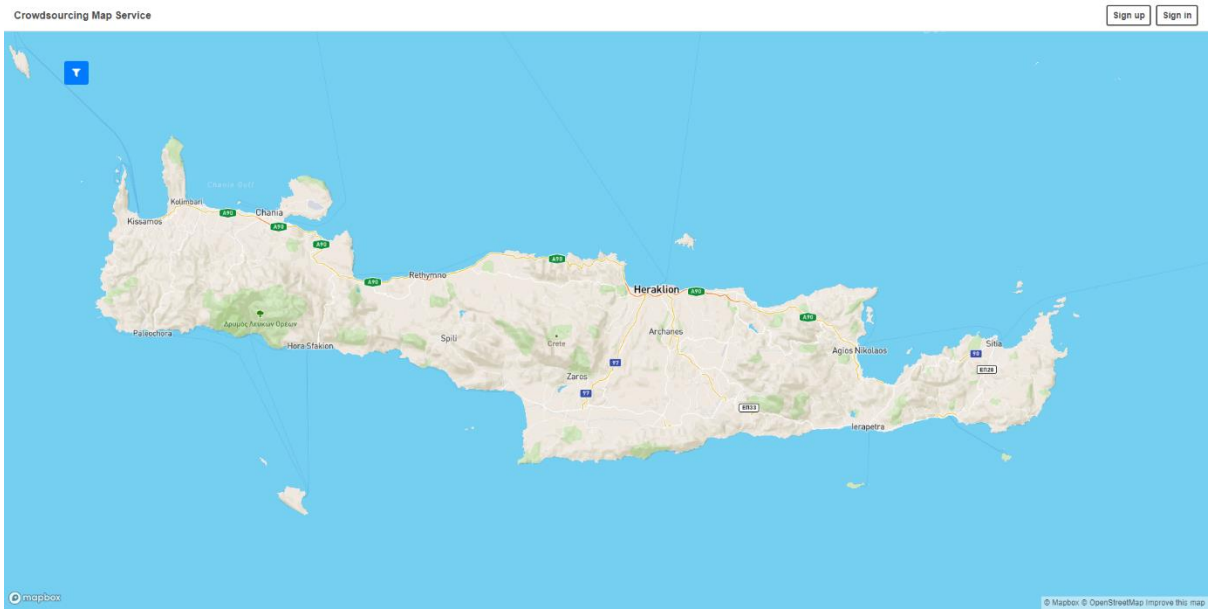
```
clearMarkers() {
        this.markers.forEach(marker => marker.remove());
        this.markers = [];
    }

    flyMap(position) {
        if (this.map) {
            this.map.flyTo({
                center: [
                    position[0],
                    position[1]
                ],
                zoom: 12
            })
        }
    }

    render() {
        return (
            <div id="map" style={style}>
            </div>
        );
    }
}
```

6. Παρουσίαση Frontend

Στο παρόν κεφάλαιο θα αναλύσουμε το frontend κομμάτι της εφαρμογής μας και επιπλέον θα δείξουμε μερικά παραδείγματα δημιουργίας χρήστη, δημιουργίας χάρτη και Points Of Interest. Frontend είναι ο όρος που χρησιμοποιείται στον προγραμματιστικό χώρο όταν θέλουμε να αναφερθούμε στο κομμάτι παρουσίασης και εξωτερικής εικόνας ενός προγράμματος, συστήματος, εφαρμογής κ.α.

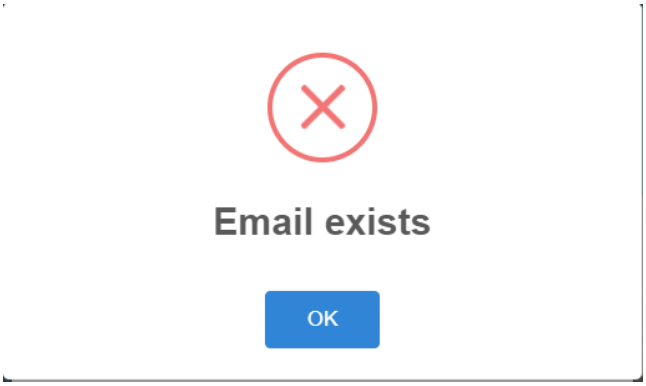
6.1 Παράδειγμα δημιουργίας χρήστη

Με το που ανοίξουμε την εφαρμογή η πρώτη εικόνα που θα δούμε είναι το χάρτη και το navigation bar.

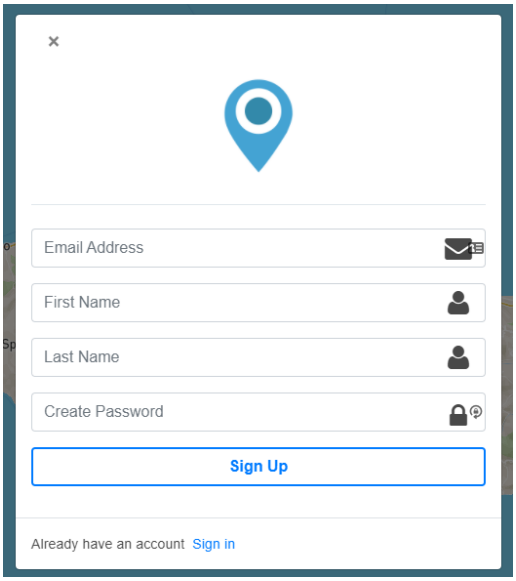


Εικόνα 48 - Εισαγωγική οθόνη

Κάνοντας κλικ στο κουμπί Sign up θα μας εμφανιστεί ένα modal για να δημιουργήσουμε λογαριασμό στην εφαρμογή μας. Μόλις συμπληρώσουμε τα πεδία, email address, first name, last name και τον κωδικό πατάμε το κουμπί Sign up. Αν το email που έχουμε συμπληρώσει υπάρχει είδη στη βάση δεδομένων τότε θα μας εμφανιστεί ένα προειδοποιητικό μήνυμα.

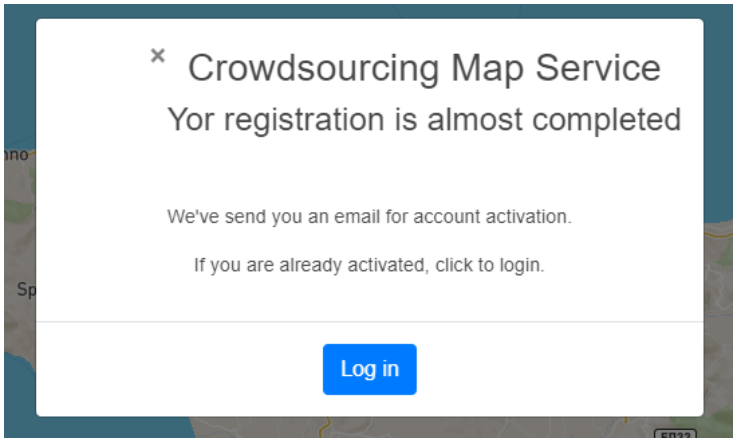


Εικόνα 49 - Αυτό το email υπάρχει μήνυμα



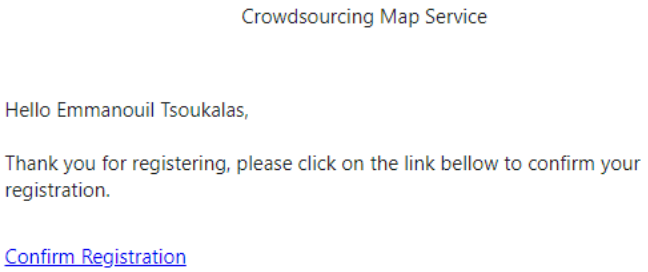
Εικόνα 50 - Modal εγγραφής χρήστη

Αν το email δεν υπάρχει τότε μόλις κάνουμε Sign up θα εμφανιστεί το παρακάτω μήνυμα.



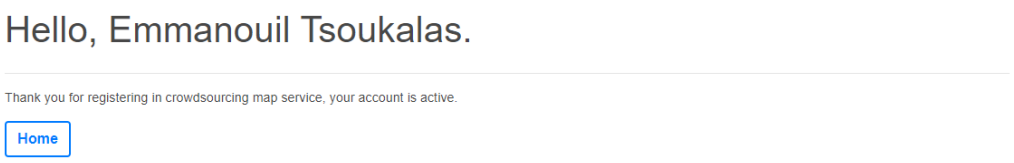
Εικόνα 51 - Η εγγραφή έγινε με επιτυχία

Στη συνέχεια θα μας έχει έρθει ένα email για να επιβεβαιώσουμε την εγγραφή μας στο σύστημα και να μπορούμε να συνδεθούμε.



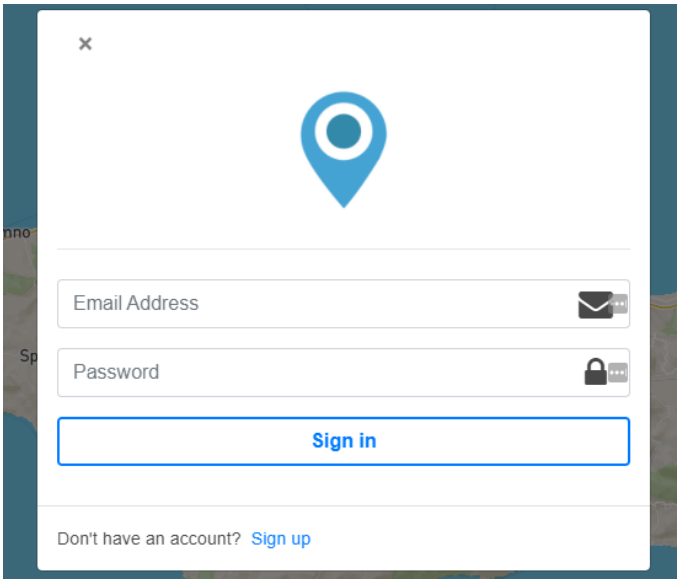
Εικόνα 52 - Email εγγραφής

Μόλις πατήσουμε το link θα μεταφερθούμε σε μια οθόνη επιβεβαίωσης ότι όλα έγιναν καλά.



Εικόνα 53 - Εγγραφή ολοκληρώθηκε

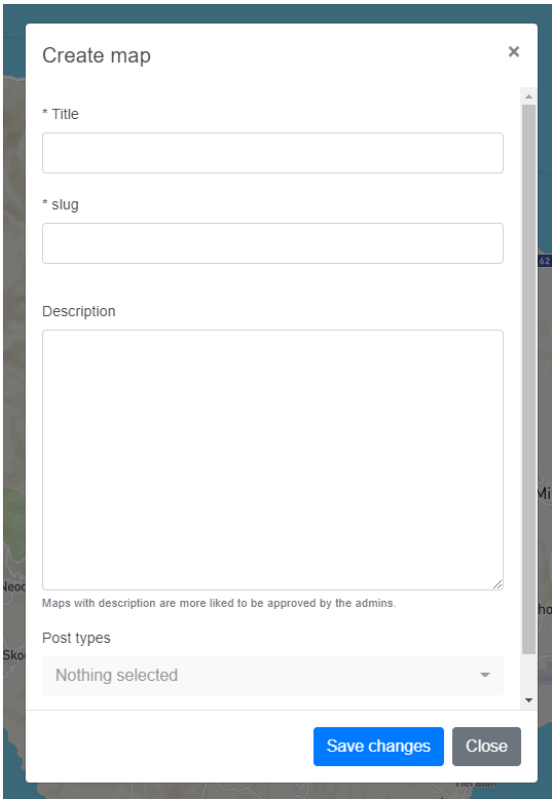
Πατώντας το κουμπί Home θα μεταφερθούμε στην κεντρική οθόνη. Τώρα μπορούμε να συνδεθούμε στο σύστημα πατώντας το κουμπί Sign in που βρίσκεται στη nav bar. Μόλις το πατήσουμε θα μας εμφανιστεί το modal για να συνδεθούμε. Συμπληρώνοντας τα πεδία email και password και πατώντας το κουμπί Sign in έχουμε συνδεθεί στο σύστημα.



Εικόνα 54 - Sign in modal

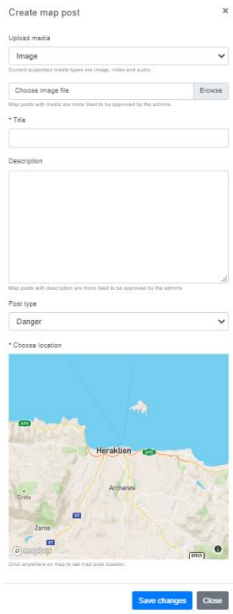
6.2 Παράδειγμα δημιουργίας χάρτη και point of interest

Μόλις έχουμε δημιουργήσει χρήστη και έχουμε κάνει σύνδεση στην εφαρμογή μπορούμε να ξεκινήσουμε να δημιουργούμε χάρτες και Points Of Interest. Για να δημιουργήσουμε ένα χάρτη πατάμε στην πλάγια μπάρα το κουμπί create map. Μόλις πατήσουμε το κουμπί μας εμφανίζεται ένα modal για να δημιουργήσουμε χάρτη. Τα βασικά πεδία είναι ο τίτλος και το μοναδικό χαρακτηριστικό slug για να μπορούν οι χρήστες να βρίσκουν το χάρτη μας και να μην υπάρχουν δύο ίδιοι χάρτες. Μπορούμε να δημιουργήσουμε φίλτρα ή να διαλέξουμε από είδη υπάρχοντα.



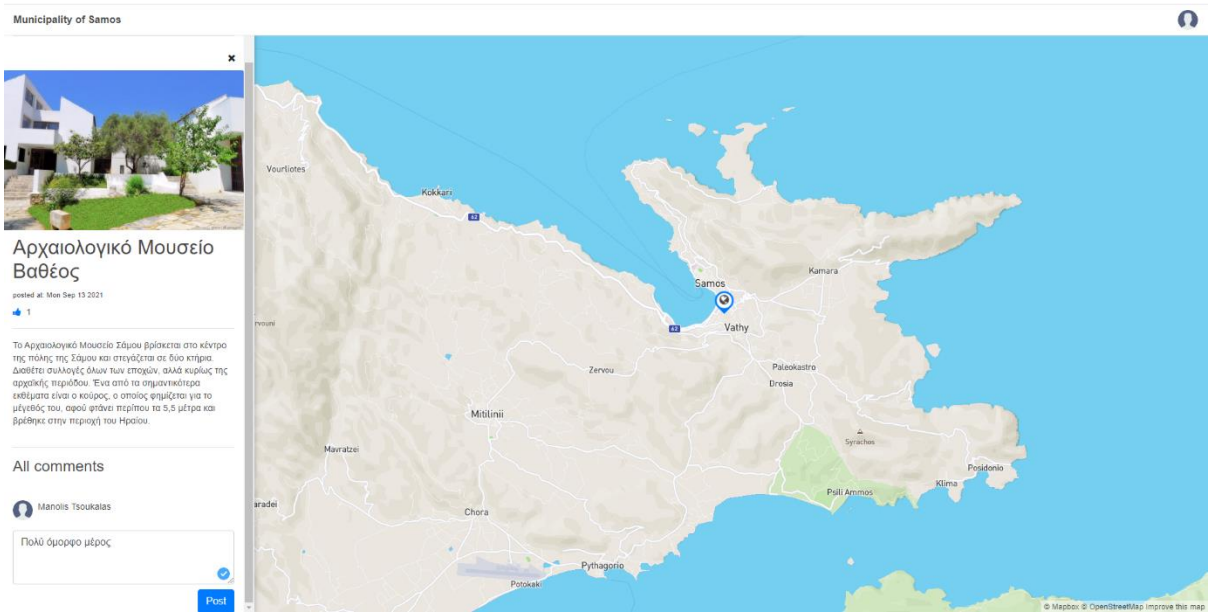
Εικόνα 55 - Δημιουργία χάρτη modal

Μόλις δημιουργήσουμε και το χάρτη μπορούμε να ξεκινήσουμε να βάζουμε points of interest ή όπως ονομάζονται στην εφαρμογή μας Map Posts. Πατώντας στο κουμπί create map post εμφανίζεται ένα modal δημιουργίας map post. Στο modal μας δίνεται η επιλογή να διαλέξουμε media type (image, video και audio) ανάλογα με το τι θα διαλέξουμε εμφανίζεται και το αντίστοιχο input field. Συμπληρώνουμε τίτλο, περιγραφή διαλέγουμε τι τύπος θα είναι και το μαρκάρουμε στο χάρτη.



Εικόνα 56 - Δημιουργία map post modal

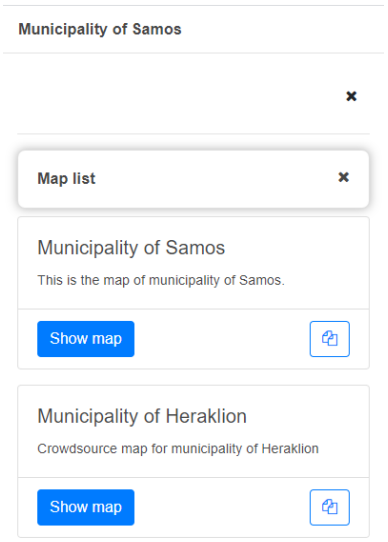
Όταν δημιουργήσουμε το point of interest βλέπουμε πάνω στο χάρτη μια πινέζα. Όταν κάνουμε κλικ πάνω της βλέπουμε το τίτλο και την περιγραφή του map post. Πατώντας το κουμπί details εμφανίζεται η πλάγια μπάρα με τις λεπτομέρειες του map post. Εκεί μπορούμε να το μαρκάρουμε ως χρήσιμο και να αφήσουμε κάποιο σχόλιο. Έτσι δημιουργούμε ένα crowdsource περιβάλλον που θα μπορούν χρήστες να δημιουργούν points of interest και ο κόσμος να τα μαρκάρει ως χρήσιμα και να αφήνει σχόλια.



Εικόνα 57 - Λεπτομέρειες map post

6.3 Παράδειγμα αποστολής Map URL

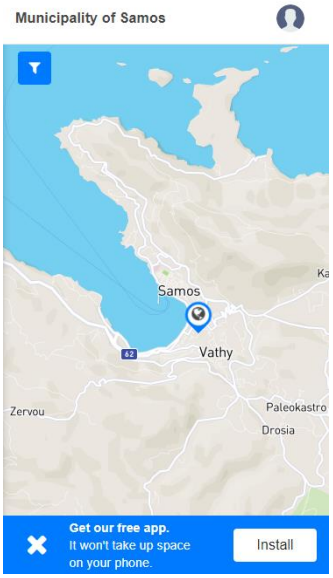
Μόλις έχουμε δημιουργήσει το χάρτη έχουμε τη δυνατότητα να στείλουμε τον χάρτη και σε άλλους χρήστες για να προσθέσουν πληροφορία. Από τη πλάγια μπάρα πατώντας το κουμπί «Select Map» μας εμφανίζεται μια λίστα από χάρτες που έχουν δημιουργήσει όλοι οι χρήστες. Μέσα στο κάθε χάρτη της λίστας υπάρχει το εικονίδιο της αντιγραφής πατώντας το έχουμε στο clipboard το URL του χάρτη για να το στείλουμε όπου θέλουμε.



Εικόνα 58 - Map list

6.4 Παράδειγμα mobile

Έχουμε τη δυνατότητα να χρησιμοποιήσουμε την εφαρμογή μέσω του κινητού μας τηλεφώνου. Αν συνδεθούμε μέσο του browser του κινητού μας τότε η εφαρμογή μας εμφανίζει ένα μήνυμα που γράφει αν θέλουμε να εγκαταστήσουμε την εφαρμογή στο κινητό μας. Αυτό γίνεται μέσω της τεχνολογίας PWA.



Εικόνα 59 - Install mobile banner

Ουσιαστικά δεν εγκαθιστά την εφαρμογή, όσο δημιουργεί ένα shortcut για τη σελίδα με ένα τρόπο ώστε να θυμίζει mobile εφαρμογή.

Εικόνα 60 - Mobile shortcut

7. Συμπεράσματα - Μελλοντικές επεκτάσεις

Το παρόν κεφάλαιο αποτελεί το τελευταίο της εργασίας μας στο οποίο θα αναλύσουμε τα διάφορα συμπεράσματα που βγάλαμε καθ' όλη τη διάρκεια της υλοποίησης της εφαρμογής αλλά και τις σκέψεις και ιδέες που έχουμε για την περαιτέρω ανάπτυξη και βελτίωση αυτής.

7.1 Αξιολόγηση υλοποίησης

Με την υλοποίηση της πτυχιακής εργασίας καταφέραμε να δημιουργήσουμε μια WebGIS εφαρμογή με στοιχεία crowdsourcing. Δημιουργήσαμε ένα σταθερό backend με δυνατότητες επέκτασης των μοντέλων έχοντας ένα επίπεδο abstraction ώστε να μπορούμε να υποστηρίξουμε διάφορα είδη γεωγραφικών δεδομένων. Δημιουργήσαμε ένα documented API ώστε να μπορούν εφαρμογές τρίτων να χρησιμοποιούν προγραμματιστικά την εφαρμογή μας. Η εφαρμογή υλοποιεί ένα frontend φιλικό προς το χρήστη με μοντέρνο σχεδιασμό και είναι mobile friendly ώστε οι χρήστες να μπορούν να ανεβάζουν δεδομένα και να κάνουν crowdsourcing όπου και να βρίσκονται.

Χρησιμοποιήσαμε τεχνολογίες όπως το PWA και τα Geolocation API ώστε να μπορεί ο χρήστης να κάνει crowdsourcing από το κινητό του εύκολα.

Η εφαρμογή υλοποιήθηκε με industry standard τεχνικές και αρχιτεκτονικά πρότυπα. Η βιβλιοθήκες που χρησιμοποιήσαμε μας βοήθησαν να δημιουργήσουμε μια ασφαλής, γρήγορη και μοντέρνα εφαρμογή.

7.2 Μελλοντικές επεκτάσεις εφαρμογής – Εμπορευματοποίηση

Με περισσότερο χρόνο στη διάθεση μας θα μπορούσαμε να προσθέσουμε μερικές ακόμα λειτουργίες που θα ολοκλήρωναν την εφαρμογή και θα την κάναν έτοιμη να βγει στην αγορά

- Hub page: Η εφαρμογή μας χρειάζεται μία κεντρική σελίδα που θα βλέπει ο χρήστης μόλις επισκέπτεται την εφαρμογή. Στη σελίδα αυτή ο χρήστης θα βλέπει όλους τους χάρτες που έχουν δημιουργηθεί από το community και διάφορα στατιστικά όπως πόσοι χρήστες βρίσκουν χρήσιμο ένα χάρτη.
- Σελίδα διαχείρισης για το χρήστη: Αυτή τη στιγμή η εφαρμογή έχει μια υποτυπώδες σελίδα διαχείρισης για τον διαχειριστή. Χρειαζόμαστε και μια σελίδα διαχείρισης για τον χρήστη ώστε να μπορεί να βλέπει τα map posts που έχει δημιουργήσει σε ποιους χάρτες τα έχει δημιουργήσει και διάφορα άλλα στατιστικά που θα τον βοηθούν στο engagement.
- Social media functionality: Πρέπει στην εφαρμογή μας να μπορούμε να κάνουμε Logged in και Sign up μέσο social media εφαρμογών. Επίσης, πρέπει να έχουμε κουμπιά ώστε να μπορούν οι χρήστες να κάνουν share στα social media τα map posts και τους χάρτες που έχουν δημιουργήσει.

Με τις παραπάνω βασικές προσθήκες μπορούμε να πούμε ότι η εφαρμογή είναι σε ένα πολύ καλό σημείο ώστε να βγει στην αγορά ως version 1.0.0. Σίγουρα θα χρειαστούμε βελτιώσεις στο κομμάτι του performance και διάφορα bug fixes, αλλά όσο ο κόσμος χρησιμοποιεί την

εφαρμογή μας τόσο θα μπορούμε να δούμε και εμείς τις ανάγκες που θα χρειαστούν να καλύψουμε.

Παρατηρούμε ότι οι περισσότερες εφαρμογές WebGIS δεν είναι τόσο φιλικές προς το χρήστη και απευθύνονται σε πιο εξειδικευμένο κοινό. Το καλό με τη δικιά μας εφαρμογή είναι ότι είχαμε στο μυαλό μας να γίνει όσο πιο φιλική στο χρήστη μπορεί να γίνει. Επίσης, πολλές από αυτές δεν υποστηρίζουν το mobile και ιδικά με μια τεχνολογία όπως το PWA.

7.3 Συμπεράσματα υλοποίησης – Επίλογος

Συμπερασματικά, η εφαρμογή είναι εύκολη προς τη χρήση είναι mobile first αυτό σημαίνει ότι λειτουργεί εύκολα με τα κινητά, μπορεί να λειτουργήσει ως μια εφαρμογή crowdsourse και με λίγο περισσότερο χρόνο ανάπτυξης θα μπορούσε να εισαχθεί στο χώρο του WebGIS και του crowdsource ως μια production ready εφαρμογή.

Σε προσωπικό επίπεδο παρά τις δυσκολίες που συναντήσαμε αποκτήσαμε πολλές γνώσεις που σίγουρα θα μας φανούν χρήσιμες στο επαγγελματικό μας μέλλον και ελπίζουμε πως θα εκτιμηθούν. Επιπλέον, παρά το γεγονός ότι υλοποιήσαμε την εφαρμογή με επιτυχία βρισκόμαστε σε ένα σημείο που θα θέλαμε να συνεχίσουμε στο μέλλον την περαιτέρω ανάπτυξη της εφαρμογής με την προσθήκη επιπλέον λειτουργιών και με βελτιωμένο frontend.

Βιβλιογραφία

1. Spyros Panagiotakis, Nancy Alonistioti, “Location-based Service Differentiation”, contribution to the collaborative book entitled: “The Handbook of Mobile Middleware”, edited by Paolo Bellavista, Antonio Corradi, published by Auerbach Publications (Taylor & Francis Group) in September 2006, chapter 30, pp. 787-818.
2. Spyridon Panagiotakis, Athanassia Alonistioti, Lazaros Merakos, "An advanced location information management scheme for supporting flexible service provisioning in reconfigurable mobile networks", IEEE Communications Magazine, February 2003, vol. 41, no. 2, pp. 88 - 98.
3. Spyros Panagiotakis, Athanassia Alonistioti, “Context-Aware Composition of Mobile Services”, IEEE IT Professional, July 2006, Volume 8, Number4, pp. 38-43.
4. Spyridon Panagiotakis, Maria Koutsopoulou, Athanasia Alonistioti, “Advanced Location Information Management Scheme for Supporting Flexible Service Provisioning in Reconfigurable Mobile Networks”, in proceedings of the IST Mobile Communication Summit, Thessaloniki, Greece, June 2002.
5. N. Alonistioti, Spyridon Panagiotakis, Alexandros Kaloxylos, “A Framework For Dynamic and Context-Aware Composition of Adaptable Mobile Services”, in proceedings of the 3rd International Conference On Computer Science, Software Engineering, Information Technology, e-Business and Applications (CSITeA 2004), Cairo, Egypt, December 2004.
6. Athanassia Alonistioti, Spyridon Panagiotakis, Nikos Houssos, Alexandros Kaloxylos, “Issues for the provision of Location-dependent services over 3G networks”, in proceedings of the 3rd generation infrastructure and services conference (3GIS), Athens, Greece, July 2001.
7. K. Molnar, Z. Nagy, Spyridon Panagiotakis, V. Gazis, N. Houssos, M. Koutsopoulou, “Location features in the MOBIVAS project”, in proceedings of the Mobile Location Workshop (MLW 2001), Espoo, Finland, June 2001.
8. Nancy Alonistioti, Spyros Panagiotakis, Maria Koutsopoulou, “Location and Profiling Issues for Personalized Mobile Communications”, in proceedings of the 9th International Symposium on Wireless Personal Multimedia Communications (WPMC), San Diego, CA, USA, September 2006.
9. https://www.wired.com/2006/06/crowds/
10. https://www.investopedia.com/terms/c/crowdsourcing.asp#:~:text=Crowdsourcing%20involves%20obtaining%20work%2C%20information,tasks%20on%20a%20voluntary%20basis.
11. https://www.nii.ac.jp/en/about/upload/NIIToday_en56.pdf
12. https://elk.adalidda.com/2018/03/futureinternet-10-00024.pdf
13. https://www.outsource-force.com/blog/what-is-crowdsourcing-and-how-important-is-it/#:~:text=Crowdsourcing%20allows%20you%20to%20select,creative%20process%20and%20enhance%20productivity.

14. https://scielo.conicyt.cl/scielo.php?script=sci_arttext&pid=S0718-18762015000100002
15. https://tweakyourbiz.com/marketing/9-great-examples-crowdsourcing-age-empowered-consumers
16. https://www.braineet.com/blog/crowdsourcing
17. https://djangostars.com/blog/what-is-a-web-framework/
18. https://djangostars.com/blog/python-frameworks-for-web-development/
19. https://pressidium.com/blog/2017/browser-cache-work/
20. https://www.mulesoft.com/resources/api/what-is-rest-api-design
21. https://www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm
22. https://restfulapi.net/
23. https://www.ibm.com/cloud/learn/rest-apis
24. https://aws.amazon.com/devops/what-is-devops/
25. https://searchitoperations.techtarget.com/definition/DevOps
26. https://newrelic.com/devops/what-is-devops
27. https://dzone.com/articles/5-major-software-architecture-patterns
28. https://www.redhat.com/architect/5-essential-patterns-software-architecture
29. https://thecodereaper.com/2020/08/22/layered-architecture-pattern-in-software-engineering/
30. https://www.pubnub.com/learn/glossary/what-is-a-geolocation-api/
31. https://www.isprs.org/proceedings/XXXIV/part4/pdfpapers/422.pdf
32. https://www.sciencedirect.com/topics/computer-science/spatial-database
33. https://link.springer.com/article/10.1007/BF01231602
34. https://www.researchgate.net/publication/313504781_Spatial_Databases_An_Overview
35. https://www.gislounge.com/difference-gis-geospatial/#:~:text=The%20word%20geospatial%20is%20used,a%20form%20of%20geospatial%20data.
36. https://www.gislounge.com/what-is-gis/
37. https://www.gislounge.com/webgis-section-2-overview-of-the-tools-and-technologies-for-webgis/
38. https://www.gislounge.com/webgis-section-2-overview-of-the-tools-and-technologies-for-webgis/
39. https://www.gislounge.com/webgis-section-2-overview-of-the-tools-and-technologies-for-webgis/

40. https://www.gislounge.com/webgis-section-2-overview-of-the-tools-and-technologies-for-webgis/
41. https://www.gislounge.com/what-is-postgis/
42. https://geojson.org/
43. https://www.pubnub.com/learn/glossary/what-is-a-geolocation-api/
44. https://www.gislounge.com/webgis-section-2-overview-of-the-tools-and-technologies-for-webgis/
45. https://www.nginx.com/resources/glossary/nginx/
46. https://gunicorn.org/
47. https://developer.mozilla.org/en-US/docs/Learn/Server-side/Django/Introduction
48. https://redis.io/topics/introduction
49. https://pgdash.io/blog/pgbouncer-connection-pool.html
50. https://en.wikipedia.org/wiki/React_(JavaScript_library)
51. https://en.wikipedia.org/wiki/Progressive_web_application
52. https://movilforum.com/en/what-is-pwa-and-what-is-it-used-for/