Ατομική Διπλωματική Εργασία

ΜΕΛΕΤΗ ΚΑΙ ΥΛΟΠΟΙΗΣΗ ΤΟΥ ΥΠΟΣΥΣΤΗΜΑΤΟΣ ΔΕΔΟΜΕΝΩΝ ΤΗΣ ΠΛΑΤΦΟΡΜΑΣ ΓΕΩΤΟΠΟΘΕΤΗΣΗΣ A4IoT

Νικόλας Νεοφύτου

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Μάιος 2021

ΠΑΝΕΠΙΣΤΗΜΙΟ ΚΥΠΡΟΥ
ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΜΕΛΕΤΗ ΚΑΙ ΥΛΟΠΟΙΗΣΗ ΤΟΥ ΥΠΟΣΥΣΤΗΜΑΤΟΣ ΔΕΔΟΜΕΝΩΝ ΤΗΣ ΠΛΑΤΦΟΡΜΑΣ ΓΕΩΤΟΠΟΘΕΤΗΣΗΣ A4IoT

Νικόλας Νεοφύτου

Επιβλέπων Καθηγητής
Δημήτρης Ζεϊναλιπούρ

Η Ατομική Διπλωματική Εργασία υποβλήθηκε προς μερική εκπλήρωση των απαιτήσεων απόκτησης του πτυχίου Πληροφορικής του Τμήματος Πληροφορικής του Πανεπιστημίου Κύπρου

Μάιος 2021

Ευχαριστίες

Αρχικά θα ήθελα να ευχαριστήσω τον Δρ. Δημήτρη Ζεϊναλιπούρ για την τεράστια ευκαιρία που μου έδωσε να συνεργαστώ μαζί του σε ένα σύστημα σαν το Anyplace αφού μέσω αυτού απέκτησα πάρα πολλές γνώσεις και μια καλή εικόνα για το πώς αναπτύσσεται ένα project σε επαγγελματικό επίπεδο.

Επιπλέον θα ήθελα να ευχαριστήσω και να αποδώσω ένα σημαντικό μέρος της προσπάθειας που έγινε στον διδακτορικό φοιτητή Πασχάλη Μπέη, αφού ο ίδιος βρισκόταν πάντα στο πλάι μου. Ανεξαρτήτως ώρας και ημέρας ήταν παρόν για να με βοηθήσει σε κάθε πρόβλημα που αντιμετώπιζα, ακόμη και στην κατανόηση του υπάρχοντα κώδικα του Anyplace. Επίσης, θα ήθελα να τον ευχαριστήσω για την υπομονή και κατανόηση που έδειχνε προς εμένα κατά την διάρκεια της διπλωματικής εργασίας, ειδικά στα αρχικά στάδια προσαρμογής.

Τέλος, θα ήθελα να ευχαριστήσω την οικογένεια μου για την οικονομική και ψυχολογική υποστήριξη που μου παρείχε κατά την διάρκεια των σπουδών μου ανεξαρτήτως των δυσκολιών που αντιμετώπιζαν οι ίδιοι και ελπίζω ότι μια μέρα θα βρίσκομαι σε θέση να ανταποδώσω.

Περίληψη

Στην σύγχρονη εποχή που ζούμε ξοδεύουμε το 90% του χρόνου μας σε εσωτερικούς χώρους. Κατά την διάρκεια αυτού του χρόνου δουλεύουμε, κοινωνικοποιούμαστε, ψωνίζουμε, διασκεδάζουμε. Παράλληλα, κάνοντας αυτές τις δραστηριότητες ερχόμαστε σε επαφή με το διαδίκτυο και την τεχνολογία. Επομένως, με βάση τα πιο πάνω είναι απαραίτητο να εντάξουμε στην καθημερινότητα μας ένα σύστημα γεωτοποθέτησης, χαρτογράφησης, πλοήγησης και πληροφόρησης εσωτερικών χώρων και εάν το καταφέρουμε αυτό η διακίνηση μας στους εσωτερικούς χώρους θα γίνει πιο ευχάριστη, πιο εύκολη και πιο γρήγορη.

Σε αυτή τη διπλωματική εργασία παρουσιάζονται διάφορες αλλαγές και επεκτάσεις που έγιναν στο υπάρχον σύστημα Anyplace. Το Anyplace αποτελεί ένα ολοκληρωμένο σύστημα πληροφόρησης για εσωτερικούς χώρους. Υποστηρίζει λειτουργίες όπως: χαρτογράφηση, γεωπλοήγηση, πληροφόρηση εσωτερικών χώρων. Αναπτύχθηκε από το «Data Management Systems Laboratory» του τμήματος της Πληροφορικής του Πανεπιστημίου Κύπρου.

Αρχικά θα γίνει παρουσίαση του Anyplace Architecture όπου αναλύει τα υποσυστήματα που απαρτίζουν το Anyplace. Έπειτα, ακολουθεί μελέτη που έγινε στα δεδομένα που αποθηκεύονταν στην υπάρχουσα βάση δεδομένων Couchbase Server, η οποία αξιοποιήθηκε για να σχηματιστεί δομή (σχήμα). Μετά, θα περιγραφεί λεπτομερώς η διαδικασία αλλαγής της βάσης δεδομένων. Κατά την αλλαγή αυτή εντοπίστηκαν και διορθώθηκαν διάφορα bugs και malfunctions του συστήματος όπου και περιγράφονται στην συνέχεια.

Τέλος, θα παρουσιαστούν διάφορα πειράματα που έγιναν για να συγκριθεί η νέα έκδοση του συστήματος με την προηγούμενη. Τα πειράματα αυτά εστιάζουν στην ταχύτητα και στον όγκο δεδομένων που καταναλώνει η νέα εκδοχή του συστήματος Anyplace. Όσο αφορά την ταχύτητα έγιναν συγκρίσεις χρόνου εκτέλεσης από API calls του Anyplace, ενώ όσο αφορά τον όγκο δεδομένων έγιναν συγκρίσεις στο πλήθος των αντικειμένων που αποθηκεύει η βάση δεδομένων και στον χώρο αποθήκευσης που καταναλώνει το υποσύστημα δεδομένων.

Περιεχόμενα

Ευχαριστίες ..... **iii**

Περίληψη ..... **iv**

Περιεχόμενα ..... **v**

Κεφάλαιο 1 Εισαγωγή **1**

1.1 Υποκίνηση εργασίας **1**
1.2 Ανασκόπηση συστήματος Anyplace **3**
1.3 Πρόβλημα **4**
1.4 Συνεισφορές **5**
1.5 Περίγραμμα εργασίας **6**

Κεφάλαιο 2 Υπόβαθρο και επισκόπηση τεχνολογίας **7**

2.1 MongoDB **8**
2.1.1 Εισαγωγή 8
2.1.2 CRUD 9
2.1.3 Aggregation 10
2.1.4 Υπολογιζόμενο μοτίβο 10
2.1.5 MongoDB Compass 11
2.2 CouchbaseDB **11**
2.2.1 Εισαγωγή 11
2.2.2 Ευρετήρια & προβολές 12
2.3 Γλώσσες προγραμματισμού και εργαλεία **12**
2.3.1 Scala και SBT 12
2.3.2 Play Framework 13
2.3.3 Python 13
2.3.4 Unit Testing 13
2.3.5 Postman 14
2.3.6 Unix και προγραμματισμός κελύφους 14

Κεφάλαιο 3 Αρχιτεκτονική του Anyplace **15**

3.1 Χρονική αναδρομή στο Anyplace **16**
3.2 Anyplace Server & NoSQL αποθήκη δεδομένων **17**
3.3 Anyplace Architect **18**
3.4 Anyplace Viewer & Viewer Campus **22**
3.5 Anyplace Navigator & Logger **25**

Κεφάλαιο 4 Νέο υποσύστημα δεδομένων 28

4.1 Εισαγωγή 28
4.2 Πρόβλημα 29
4.3 Δημιουργία και εγκατάσταση νέας βάσης δεδομένων 30
4.3.1 Εξαγωγή δεδομένων 30
4.3.2 Μελέτη δεδομένων 31
4.3.3 Δημιουργία σχήματος 32
4.3.4 Δημιουργία βάσης MongoDB 36
4.3.5 Εισαγωγή δεδομένων 36
4.3.6 Επικοινωνία βάσης δεδομένων – κώδικα Anyplace 39

Κεφάλαιο 5 Αλλαγές στην Διεπαφή Προγραμματισμού Εφαρμογών (API) 45

5.1 Εισαγωγή 45
5.2 Πρόβλημα 45
5.3 Υλοποίηση 46

Κεφάλαιο 6 Πειραματική αποτίμηση 51

6.1 Εισαγωγή 51
6.2 Αποτίμηση χρόνου 52
6.3 Μέγεθος δεδομένων 59
6.3.1. Αντικείμενα 59
6.3.2. Αποθηκευτικός χώρος 61

Κεφάλαιο 7 Συμπεράσματα και μελλοντικές επεκτάσεις 63

7.1 Συμπεράσματα 63
7.2 Μελλοντικές επεκτάσεις 64
7.2.1 Επεκτάσεις στα endpoints 64
7.2.2 Επεκτάσεις στην βάση δεδομένων 64
7.2.3 Επεκτάσεις στο Anyplace 65

Βιβλιογραφία 66

Παράρτημα Α – Κώδικας καινούριας κλάσης MongodbDatasource.Scala 1

Κεφάλαιο 1

Εισαγωγή

1.1 Υποκίνηση εργασίας 1

1.2 Ανασκόπηση συστήματος Anyplace 3

1.3 Πρόβλημα 4

1.4 Συνεισφορές 5

1.5 Περίγραμμα εργασίας 6

1.1 Υποκίνηση εργασίας

Έρευνες της Strategy Analytics [1] δείχνουν ότι κατά μέσο όρο ο άνθρωπος ξοδεύει το 80-90% του χρόνου του σε εσωτερικούς χώρους. Ο χρόνος αυτός κατανέμεται σε εσωτερικούς χώρους όπως σχολεία, πανεπιστήμια, χώρους εργασίας, εμπορικά κέντρα, αεροδρόμια, νοσοκομεία κ.α. Η διακίνηση σε αυτούς τους χώρους αρκετές φορές είναι πολύπλοκη και χρονοβόρα, ιδίως για άτομα που εισέρχονται σε διάφορους χώρους για πρώτη φορά. Σε συνδυασμό της δυσκολίας αυτής και της αυξημένης χρήσης έξυπνων συσκευών δημιουργήθηκε η ανάγκη ανάπτυξης συστημάτων πλοήγησης εσωτερικών χώρων. Σημαντικότεροι παράγοντες για συστήματα τέτοιου είδους είναι η ζωντανή και ακριβής γνώση της τοποθεσίας του χρήστη καθώς διατηρείται το κόστος χαμηλό αλλά και η δυνατότητα χρήσης σε συσκευές με περιορισμένους πόρους. Επιπρόσθετα, η χρήση αυτού του συστήματος μπορεί να επιφέρει τεράστιες αλλαγές στην καθημερινότητα μας, αφού σε ένα εμπορικό περιβάλλον υπάρχει η δυνατότητα παρακολούθησης του εργατικού προσωπικού και άλλων ατόμων για θέματα ασφάλειας. Παράλληλα δίνεται η ευκαιρία σε καταστηματάρχες να καταγράφουν σημαντικά στατιστικά με βάση τον τρόπο που κινείται ένας πελάτης στο κατάστημά τους.

Ακόμη η γεωπλοήγηση μπορεί να αποτελέσει σημαντικό παράγοντα και σε θέματα υγείας. Έχουν περάσει σχεδόν δύο χρόνια από όταν πρωτοεμφανίστηκε η πανδημία της νόσου του κορονοϊού (COVID-19). Μία από τις μεγαλύτερες προκλήσεις που κληθήκαμε να αντιμετωπίσουμε είναι η ιχνηλάτηση του ιού καθώς ήταν δύσκολο να ξέρει κάποιος με ποια άτομα και πού είχε έρθει σε επαφή.

Για να είναι εφικτή η ανάπτυξη ενός συστήματος σαν κι αυτό χρειάστηκε να επινοηθούν καινούριοι τρόποι παροχής αξιόπιστης ένδειξης της τοποθεσίας ενός χρήστη σε ένα εσωτερικό χώρο. Τεχνολογίες όπως το GPS (Global Positioning System) δεν είναι δυνατό να χρησιμοποιηθούν σε εσωτερικούς χώρους λόγω της εξασθένησης των σημάτων από παρεμβολές και εμπόδια που υπάρχουν σε ένα κτήριο, επομένως θα υπήρχε μειωμένη απόδοση.

Η ραγδαία ανάπτυξη της τεχνολογίας μας παρέχει τα απαραίτητα για να υλοποιήσουμε τέτοια συστήματα, αφού μαζί της έφερε αφθονία συσκευών τις οποίες χρησιμοποιούμε καθημερινώς και μέσα τους βρίσκονται διάφοροι αισθητήρες. Επιπλέον, σε κάθε κτήριο υπάρχουν πλέον δεκάδες σημεία πρόσβασης Wi-Fi τα οποία μπορούμε να εκμεταλλευτούμε. Κατά μέσο όρο υπάρχουν 5-10 τέτοια σημεία σε κάθε σπίτι όπου και αναμένεται να αυξηθούν στα 500 μέχρι το 2022. Μια έξυπνη κινητή συσκευή λαμβάνοντας σήματα από αυτά τα σημεία έχει την δυνατότητα να καθορίζει πιο σήμα ανήκει σε κάθε σημείο πρόσβασης με βάση ένα μοναδικό χαρακτηριστικό, την διεύθυνση MAC. Έτσι με βάση την ένταση που λαμβάνει από κάθε σημείο μπορεί να υπολογίσει την ακριβή τοποθεσία του χρήστη στον χώρο, αφού διασταυρώσει αυτή την πληροφορία με μια βάση δεδομένων που διατηρεί στοιχεία εσωτερικής γεωτοποθέτησης.

Ακόμη, η χαρτογράφηση των εσωτερικών χώρων σε επίπεδο μοντέλων (κτιρίων, ορόφων, σημεία ενδιαφέροντος) και σε επίπεδο σημάτων μπορεί να γίνει με την βοήθεια οποιουδήποτε χρήστη που έχει στην κατοχή του μια έξυπνη συσκευή. Επομένως, η πλοήγηση στον χώρο είναι μια προέκταση της χαρτογράφηση, η οποία ολοκληρώνει τον ορισμό του Συστήματος Πληροφόρησης για Εσωτερικούς Χώρους (Indoor Information System – IIS).

1.2 Ανασκόπηση συστήματος Anyplace

Στην παρούσα διπλωματική εργασία θα παρουσιαστεί η αλλαγή που έγινε στο υποσύστημα δεδομένων του Anyplace [2]. Η αλλαγή αυτή ήταν ριζική αφού ολόκληρη η βάση δεδομένων αντικαταστάθηκε. Στην αλλαγή αυτή έγιναν διορθώσεις στο API (Application Protocol Interface) που αφορούσαν ελέγχους και bugs. Επίσης, έγιναν αλλαγές στην δομή κάποιων από τα αντικείμενα που αποθηκεύει η βάση. Η αιτία της αλλαγής προήλθε από την ανάγκη μείωσης των ελάχιστων απαιτήσεων υλικού (hardware) και γενικότερα για να αναδιοργανωθεί η δομή του υποσυστήματος.

Τα δεδομένα που συλλέγονται από το σύστημα του Anyplace είναι μοντέλα εσωτερικών χώρων όπως κτήρια, όροφοι, σημεία ενδιαφέροντος (π.χ. ένα δωμάτιο) καθώς και αποτυπώματα σημάτων πρόσβασης Wi-Fi. Τα δεδομένα αυτά εισάγονται κυρίως από τον Anyplace Architect και τον Anyplace Logger από χρήστες που επιθυμούν να συνεισφέρουν στην διαμόρφωση των κτηρίων με την μέθοδο του πληθοπορισμού (crowdsourcing). Παράλληλα, εξάγονται κυρίως από τον Anyplace Architect, Anyplace Viewer, Anyplace Viewer Campus, Anyplace Logger και Anyplace Navigator έτσι ώστε να παρέχονται τα μοντέλα εσωτερικών χώρων, οι διαδρομές για πλοήγηση και άλλες λειτουργείες του συστήματος στους χρήστες. Είναι προφανές ότι υπάρχουν πολύ περισσότερα reads από writes οπότε η ταχύτητα ανάκτησης των δεδομένων αποτελεί μεγάλο παράγοντα στο σύστημα αυτό.

Όλα τα δεδομένα αποθηκεύονταν σε μια αποθήκη δεδομένων, CouchbaseDB. Επομένως, για καλύτερη απόδοση υπήρχαν διάφορα views. Ο όγκος δεδομένων ξεκίνησε να μεγαλώνει με αποτέλεσμα κάποια από τα views να αδυνατούν να λειτουργήσουν. Επίσης αρκετά αντικείμενα είχαν περιττές πληροφορίες με αποτέλεσμα ο όγκος να μεγαλώνει ακόμη περισσότερο. Αυτό είχε σαν συνέπεια αρκετές καθυστερήσεις σε συγκεκριμένα endpoints του API.

Όσο αφορά τον νέο τρόπο διαχείρισης δεδομένων αποφασίστηκε να χρησιμοποιηθεί παρόμοιου τύπου βάση δεδομένων, η MongoDB. Η βάση αυτή δεν αποτελεί αποθήκη δεδομένων αφού διατηρεί ένα σχήμα όπου ομαδοποιεί αντικείμενα παρόμοιας κατηγορίας σε collections. Επίσης, αφού η αρχική ανάγκη ήταν η μείωση του απαιτούμενου υλικού έγιναν αρκετές αλλαγές στον τρόπο που αποθηκεύονται τα

δεδομένα, κυρίως στα αποτυπώματα σημείων Wi-Fi αφού το ποσοστό αυτών είναι 99%. Πιο αναλυτικά η περιγραφή του νέου υποσυστήματος θα γίνει στο *Κεφάλαιο* 4.

1.3 Πρόβλημα

Η ομάδα του Anyplace και το εργαστήριο DMSL έχουν ως στόχο να βελτιώνουν συνεχώς το σύστημα έτσι ώστε να είναι πιο αποδοτικό, πιο γρήγορο, πιο εύκολο στην χρήση με νέες λειτουργίες και καινούρια χαρακτηριστικά. Για να είναι αυτό εφικτό το σύστημα χρειάζεται συντήρηση και συστηματική αναβάθμιση. Στην παρούσα διπλωματική εργασία έγινε μεγάλη προσπάθεια έτσι ώστε να μειωθεί ο χώρος αποθήκευσης που καταναλώνει το σύστημα με κύριο στόχο να μπορεί να ενσωματωθεί και να λειτουργεί σε Raspberry PI. Επίσης, λειτουργίες που ήταν προβληματικές, οι οποίες χρησιμοποιούσαν αποτυπώματα χρόνου ή γεω-τοποθεσίες διορθώθηκαν και λειτουργούν κανονικά. Παράλληλα, κάποια αντικείμενα αναδιαμορφώθηκαν όσο αφορά την δομή τους μαζί με ολόκληρη την δομή της βάσης δεομένων. Τα πιο πάνω επιτεύχθηκαν με την αντικατάσταση της βάσης δεδομένων CouchbaseDB σε MongoDB.

Στην προηγούμενη έκδοση υπήρχαν προβληματικά endpoints που είχαν ως αποτέλεσμα την δυσλειτουργία ορισμένων χαρακτηριστικών του συστήματος. Τα endpoints αυτά αδυνατούσαν να εκτελεστούν λόγο της βάσης δεδομένων. Η βάση δεδομένων προσπαθούσε να κτίσει materialized views για να μπορεί να απαντά γρήγορα σε περίπτωση που εκτελούταν ένα αίτημα αλλά στα συγκεκριμένα αιτήματα το view δεν κτιζόταν ποτέ είτε επειδή γέμιζε η μνήμη ram του server είτε επειδή έκανε time-out η εντολή που θα έκτιζε το view από τον πολλή χρόνο. Έπειτα, το γεγονός ότι κτίζονταν και αποθηκεύονταν τα materialized views, σε συνδυασμό με κακό τρόπο αποθήκευσης των αντικειμένων που παράγει το σύστημα, ο απαιτούμενος αποθηκευτικός χώρος ήταν πολύ μεγάλος.

Για να αντιμετωπίσουμε τις ποιο πάνω προκλήσεις έπρεπε να γίνουν αλλαγές στο υποσύστημα δεδομένων. Επομένως, αποφασίστηκε να αντικατασταθεί η προηγούμενη βάση δεδομένων, CouchbaseDB, με άλλη βάση δεδομένων, την MongoDB. Οι δύο βάσεις έχουν αρκετά κοινά αφού ανήκουν στην ίδια κατηγορία, No-SQL. Πριν την μετάβαση έγινε μελέτη έτσι ώστε ο απαραίτητος αποθηκευτικός χώρος να μειωθεί. Επίσης, η MongoDB δεν κτίζει materialized views επομένως η μείωση ήταν τεράστια. Παράλληλα, με την εισαγωγή της MongoDB μπορεί να αφαιρεθεί και η δεύτερη βάση

δεδομένων που χρησιμοποιεί το σύστημα, την InfluxDB, αφού οι απαιτήσεις που καλύπτει μπορούν να καλυφθούν από την MongoDB.

1.4 Συνεισφορές

Στα πλαίσια της διπλωματικής εργασίας έκανα τις πιο κάτω σημαντικές αλλαγές στο σύστημα Anyplace:

1. Αντικατέστησα την υφιστάμενη βάση δεδομένων, CouchbaseDB, με καινούρια βάση, την MongoDB. Πλέον όλες οι λειτουργίες του Anyplace που αλληλοεπιδρούν με την βάση δεδομένων διεκπεραιώνονται από την MongoDB. Η αλλαγή αυτή δεν ήταν απλή μεταφορά των δεδομένων. Αρχικά ακολούθησε μελέτη και κατηγοριοποίηση των αντικειμένων που αποθήκευε η βάση σε Collections (buildings, campuses, edges, fingerprintsWifi, floorplans, pois, users). Στην συνέχεια έγινε εισαγωγή των δεδομένων στην MongoDB (διαδικασία όπου αυτοματοποιήθηκε). Μετά ακολούθησε η δημιουργία κλάσης (MongodbDatasource.Scala) όπου υλοποιήθηκαν όλα τα queries και άλλες συνάρτησης που υπήρχαν και χρησιμοποιούνταν από την υπάρχων κλάση (CouchbaseDatasource.Scala). Τέλος, κατά την αλλαγή της βάσης δεδομένων, συγκεκριμένα στο στάδιο όπου μελετήθηκαν τα δεδομένα παρουσιάστηκαν ευκαιρίες βελτιστοποίησης. Για να γίνουν εφικτές, έκανα ορισμένες αλλαγές σε κάποια Json αντικείμενα, κυρίως του Collection fingerprintsWifi.
2. Αυτοματοποίησα την διαδικασία μεταφοράς από CouchbaseDB σε MongoDB. Συνδύασα διάφορα bash και python scripts τα οποία έγραψα. Αρχικά όλα τα αποθηκευμένα δεδομένα της CouchbaseDB γίνονται export σε ένα αρχείο. Στην συνέχεια διαμοιράζονται σε άλλα αρχεία με βάση το Collection που ανήκουν. Τέλος, διαβάζοντας τα αρχεία αυτά ένα-ένα γίνονται οι αλλαγές που αποφάσισα να κάνω σε συγκεκριμένα Collections και ταυτόχρονα εισάγονται στην MongoDB.
3. Κατά την συγγραφή της καινούριας κλάσης MongodbDatasource.Scala έκανα αρκετές διορθώσεις σε μερικά endpoints, διόρθωσα διάφορα bugs που εντόπισα και παράλληλα πρόσθεσα αρκετούς λογικούς ελέγχους.

1.5 Περίγραμμα εργασίας

Στο πρώτο κεφάλαιο παρουσιάζεται συνοπτικά η υποκίνηση της παρούσας διπλωματικής εργασία. Στην συνέχεια γίνεται μια σύντομη ανασκόπηση του συστήματος Anyplace και τέλος αναφέρονται οι συνεισφορές που έγιναν με την εκπλήρωση της πτυχιακής εργασίας.

Στο δεύτερο κεφάλαιο γίνεται περιγραφή όλων των εργαλείων που χρησιμοποιήθηκαν κατά την υλοποίηση της διπλωματικής εργασίας. Αναφέρονται διάφορες έννοιες, ορισμοί, τεχνικές και γενικότερα πληροφορίες που αφορούν τα εργαλεία αυτά. Περισσότερη βαρύτητα δίνεται στα εργαλεία που αφορούν την καινούρια βάση δεδομένων, MongoDB και MongoDB Compass.

Στο τρίτο κεφάλαιο αναλύεται η αρχιτεκτονική του Anyplace έτσι ώστε να παρουσιαστούν τα στοιχεία που το αποτελούν. Αρχικά βλέπουμε τα διάφορα στάδια που πέρασε το σύστημα Anyplace για να φτάσει στην τωρινή εκδοχή, και μετά ακολουθεί αναλυτική περιγραφή ιστοσελίδων και εφαρμογών που το αποτελούν με διάφορα παραδείγματα και φωτογραφίες για να κατανοήσουμε τι είναι και πως λειτουργεί.

Στο τέταρτο κεφάλαιο γίνεται εκτενής περιγραφή της μελέτης και υλοποίησης του καινούριου υποσυστήματος της πλατφόρμας Anyplace, MongoDB. Στην περιγραφή αυτή αναλύεται όλη η διαδικασία και όλες οι αποφάσεις που έγιναν κατά την διάρκεια της αλλαγής και εισαγωγής της νέας βάσης δεδομένων.

Στο πέμπτο κεφάλαιο παρουσιάζεται το καινούριο API που προέκυψε από την διπλωματική εργασία. Αναλύονται όλες οι αλλαγές που έγιναν όπου κυρίως σχετίζονται με διορθώσεις bug και ελέγχους λογικής που προστέθηκαν έτσι ώστε να βελτιώσουν την εμπειρία χρήσης.

Στο έκτο κεφάλαιο γίνεται πειραματική αξιολόγηση της νέας βάσης δεδομένων με διάφορες μετρικές (χρόνος, χωρητικότητα). Γράφτηκαν αυτοματοποιημένοι έλεγχοι οι οποίοι συγκρίνουν την επίδοση της νέας βάσης δεδομένων με την προηγούμενη (CouchbaseDB).

Στο έβδομο και τελευταίο κεφάλαιο αναφέρονται κάποια συμπεράσματα και κάποιες προτάσεις για μελλοντικές επεκτάσεις προκειμένου η αλληλεπίδραση με το σύστημα Anyplace να γίνει ακόμη πιο ευχάριστη, αποτελεσματική και αποδοτική.

Κεφάλαιο 2

Υπόβαθρο και επισκόπηση τεχνολογίας

2.1 MongoDB 8

2.1.1 Εισαγωγή 8

2.1.2 CRUD 9

2.1.3 Aggregation 10

2.1.4 Computed patterns 10

2.1.5 MongoDB Compass 11

2.2 CouchbaseDB 11

2.2.1 Εισαγωγή 11

2.2.2 Ευρετήρια και Προβολές 12

2.3 Γλώσσες προγραμματισμού και εργαλεία 12

2.3.1 Scala και SBT 12

2.3.2 Play Framework 13

2.3.3 Python 13

2.3.4 Unit Testing 13

2.3.5 Postman 14

2.3.6 Unix 14

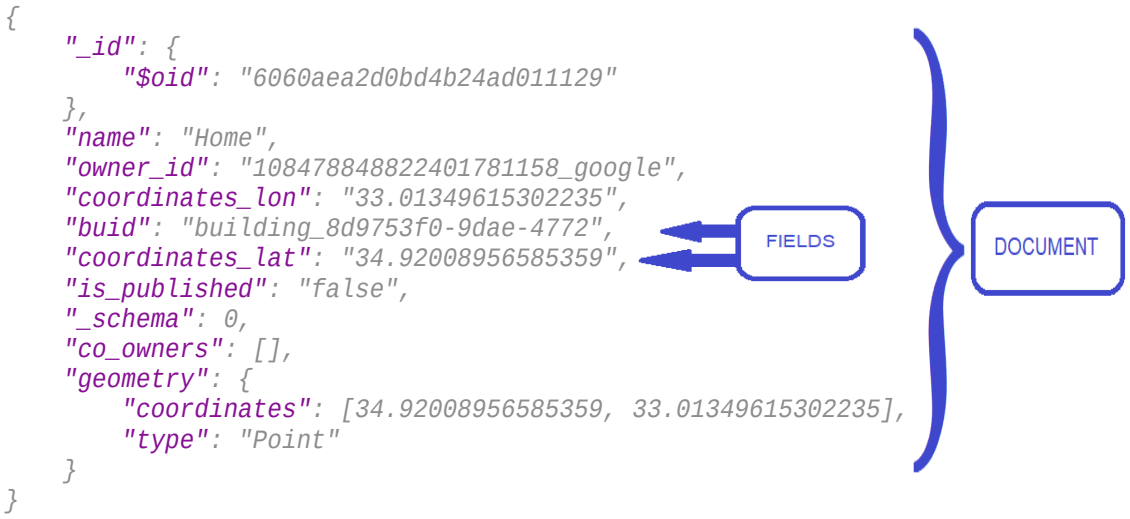
2.1 MongoDB

2.1.1 Εισαγωγή

Η mongoDB [3] είναι μια source-available document-oriented βάση δεδομένων. Κατατάσσετε στις NoSQL βάσεις δεδομένων αφού είναι σχεδιασμένη για να είναι schema-less και χρησιμοποιεί JSON-like documents. Το όνομα της προέρχεται από την αγγλική λέξη "humongous", επειδή μπορεί να επεξεργαστεί τεράστιους όγκους δεδομένων και κάνει scale-up με ευκολία. Η πρώτη έκδοση της mongoDB κυκλοφόρησε το 2009 (version 1), ενώ τώρα η βρισκόμαστε στην έκδοση 4.4 που κυκλοφόρησε το 2020.

Για να δούμε αναλυτικά την δομή δεδομένων που έχει μια NoSQL (non-relational) βάση μπορούμε να την συγκρίνουμε με μια SQL. Στον πιο κάτω πίνακα βλέπουμε την αντιστοιχία όρων μεταξύ SQL Server και MongoDB. Η κύρια διαφορά μεταξύ των δύο κατηγοριών είναι ότι οι No-SQL δεν έχουν κάποιο schema και μπορούν να αποθηκεύουν δεδομένα με περισσότερη ελευθερία.

SQL Server	MongoDB
Database	Database
Table	Collection
Index	Index
Row	Document
Column	Field
Joining	Linking & Embedding
Partition	Sharding
Replication	ReplSet



Σχήμα 2.1: JSON αντικείμενο

2.1.2 CRUD

Στην συνέχεια έχουμε τα CRUD operations, τα οποία είναι create, read, update και delete.

Create:

Για το create υπάρχουν δύο μέθοδοι, insertOne και insertMany.

```
db.users.insertOne( { name: "john", age: 26 } )
```

Read:

Μέθοδος find.

```
db.users.find( { name: ”john”  } )
```

Update:

Για το operation αυτό υπάρχουν 3 επιλογές, updateOne, updateMany και replaceOne.

```
db.users.updateMany(
       { age: 40 },
       { $set: { "Status" : true } }
    )
```

Delete:

Το delete έχει 2 επιλογές, deleteOne, deleteMany.

```
db.users.deleteMany( { "name" : "john" } )
```

Όλες οι λειτουργίες CRUD παίρνουν παράμετρο ένα json το οποίο μπορεί να περιέχει πολλά query criteria. Μερικά από αυτά είναι $gt, $lt, $in, $regex, $or. Με την χρήση αυτών υπάρχει περισσότερη ευελιξία και αποτελεσματικότητα.

2.1.3 Aggregation

Η συσσωμάτωση, γνωστό ως Aggregation, είναι μια λειτουργεία όπου επεξεργάζεται τα δεδομένα και επιστρέφει υπολογιζόμενα αποτελέσματα. Στην ουσία, ομαδοποιεί δεδομένα από διάφορα αντικείμενα (documents) και τα επεξεργάζεται με πολλούς τρόπους έτσι ώστε να υπολογίζει και να επιστρέψει ένα υπολογιζόμενο αποτέλεσμα. Οι διάφορες πράξης που γίνονται στα δεδομένα μπορούν να θεωρηθούν ως μία διασωλήνωση πράξεων.

Τα aggregations μπορούν να παρέχουν μια σειρά από query πράξης στα αντικείμενα ενός collection. Η χρήση αυτών θυμίζει συμπεριφορές query που εκτελούνται από σχεσιακές βάσης δεδομένων, αφού ακολουθούνται κάποια στάδια κατά την εκτέλεση ενός aggregation. Αυτά είναι το $match και το $group. Αρχικά, οποιοδήποτε αντικείμενο πληροί το match μεταβιβάζεται στο επόμενο στάδιο, group όπου και γίνεται ο υπολογισμός.

π.χ.

```
db.orders.aggregate([
   { $match: { status: "A" } },
   { $group: { _id: "$cust_id", total: { $sum: "$amount" } } }
])
```

2.1.4 Υπολογιζόμενο μοτίβο

Συχνά χρειάζεται να αντλήσουμε μια τιμή που είναι αποθηκευμένη στην βάση. Ο υπολογισμός μιας τέτοιας τιμής απαιτεί σημαντικούς πόρους από τον CPU. Εάν αυτή η τιμή απαιτείτε συχνά, θα ήταν πιο αποδοτικό να την αποθηκεύαμε παρά να την υπολογίζαμε κάθε φορά. Η τεχνική αυτή ονομάζεται υπολογιζόμενο μοτίβο (computed pattern).

Εξυπηρετεί καλύτερα ένα σύστημα όπου υπάρχουν περισσότερα reads από writes. Η τιμή αυτή ενημερώνεται με κάθε διαγραφή ή εισαγωγή νέου αντικειμένου. Γενικά τα υπολογιζόμενα μοτίβα ακολουθούν την φιλοσοφία “never recompute when you can precompute”.

2.1.5 MongoDB Compass

Το MongoDB Compass [4] είναι το επίσημο γραφικό περιβάλλον διεπαφής χρήστη της MongoDB. Κύριος στόχος του MongoDB Compass είναι να βοηθήσει τους χρήστες να πάρουν καλύτερες αποφάσεις όσο αφορά την δομή των δεδομένων, των queries και άλλες λειτουργείες που μπορούν να γίνουν από μια βάση δεδομένων.

Μια εναλλακτική του MongoDB Compass είναι το mongo shell αλλά το Compass υπερτερεί του shell για τους εξής λόγους. Υπάρχει η δυνατότητα προβολής και εξερεύνησης των δεδομένων σε διάφορες μορφές (πίνακας, json μορφή, λίστα). Υπάρχουν διάφορα στατιστικά σε πραγματικό χρόνο. Μπορούν να κατανοηθούν πιο εύκολα προβλήματα που αφορούν την επίδοση με visual plans. Η σύνταξη πολύπλοκων query είναι εύκολη αφού υπάρχουν διάφορα πεδία όπου μπορεί ο χρήστης να συμπληρώσει, όπως filter, project, sort, collation, skip, max time και limit. Επίσης υπάρχει κατηγορία Aggregation όπου βοηθά στην σύνταξη query που κάνουν aggregate. Τέλος, ένα πολύ χρήσιμο χαρακτηριστικό του εργαλείου είναι η εξαγωγή query σε γλώσσα προγραμματισμού , όπου υπάρχουν τέσσερις επιλογές, python3, java, node και C sharp. Στην λειτουργεία αυτή υπάρχει η επιλογή πρόσθεσης των ανάλογων import βιβλιοθηκών, αφού κάποια queries απαιτούν συγκεκριμένες βιβλιοθήκες για να εκτελεστούν.

2.2 CouchbaseDB

Παρόλο που η CouchbaseDB δεν χρησιμοποιήθηκε αλλά αντικαταστάθηκε σε αυτή τη διπλωματική εργασία θα ήταν καλό να γίνει μια σύντομη ανασκόπηση για να εξηγηθούν διάφοροι όροι και τεχνικές για καλύτερη κατανόηση.

2.2.1 Εισαγωγή

Η CouchbaseDB είναι open-source κατανεμημένη NoSQL document-oriented βάση δεδομένων και έχει βελτιστοποιημένη απόδοση σε εφαρμογές. Αυτές οι εφαρμογές έχουν πολλούς χρήστες συνεχώς οι οποίοι εισάγουν, εξάγουν και αλληλοεπιδρούν με τα δεδομένα. Λαμβάνοντας υπόψη αυτό, η Couchbase σχεδιάστηκε με τέτοιο τρόπο έτσι ώστε να κλιμακώνεται εύκολα (easy-to-scale) και να έχει γρήγορη πρόσβαση στα δεδομένα. Επίσης, είναι σχεδιασμένη για να είναι ομαδοποιημένη (clustered) από ένα

μηχάνημα μέχρι και πολύ μεγάλης κλίμακας εφαρμογές που χρησιμοποιούν πολλά μηχανήματα.

2.2.2 Ευρετήρια & προβολές

Η CouchbaseDB παρέχει στον χρήστη την δυνατότητα να δημιουργεί ευρετήρια και προβολές (indexes και views). Τα index βασίζονται σε ένα μοναδικό κλειδί που περιέχει ένα JSON αντικείμενο. Ένα index συνήθως χρησιμοποιείται για την εκτέλεση απλών query χωρίς παραμέτρους ή φίλτρα.

Ένα view δημιουργεί ένα index βασισμένο σε προκαθορισμένη δομή και μορφή. Το view αποτελείται από συγκεκριμένα πεδία και πληροφορίες που εξάγονται από αντικείμενα της βάσης. Τα views χρησιμοποιούνται για αρκετές λειτουργίες. Μερικές από αυτές είναι: αναζήτηση δεδομένων (query) από αποθηκευμένα αντικείμενα, εξαγωγή πληροφοριών από την βάση δεδομένων, μείωση των πληροφοριών (reduce) σχετικά με ένα σύνολο αποθηκευμένων δεδομένων κτλ. Επίσης υπάρχουν και τα materialized views τα οποία αποθηκεύουν το αποτέλεσμα ενός query το οποίο περιέχει διάφορους υπολογισμούς ή χρειάζεται να κάνει aggregate τα δεδομένα.

2.3 Γλώσσες προγραμματισμού και εργαλεία

2.3.1 Scala και SBT

Η Scala [5] είναι γλώσσα προγραμματισμού η οποία υποστηρίζει object-oriented και functional προγραμματισμό με δυναμικά χαρακτηριστικά. Το όνομα της προέρχεται από δύο αγγλικές λέξης, Scalable Language υποδείχνοντας ότι έχει σχεδιαστεί για να κλιμακώνεται με βάση τις ανάγκες των χρηστών. Έχει πολλά κοινά χαρακτηριστικά με την Java. Κατά τον σχεδιασμό της γλώσσας πολλές αποφάσεις πάρθηκαν με βάση κριτικές που έχει η Java. Ο κώδικας μεταγλωττίζεται σε Java bytecode επομένως το εκτελέσιμο τρέχει στο Java Virtual Machine (JVM). Ένα μεγάλο πλεονέκτημα της Scala είναι η συμβατότητα που υπάρχει με την Java, αφού μπορεί να χρησιμοποιεί όλες τις βιβλιοθήκες της Java. Ακόμη με την βοήθεια άλλων εργαλείων όπως το SBT (Scala Build Tool) [6] είναι εφικτό Java και Scala κώδικας να συνυπάρχουν στο ίδιο project. Το SBT είναι γραμμένο σε Scala αλλά αυτό δεν σημαίνει ότι είναι αποκλειστικά δημιουργημένο για την Scala, είναι παρόμοιο εργαλείο του maven.

2.3.2 Play Framework

Το Play Framework [7] είναι open-source web application framework, δηλαδή είναι framework όπου σχεδιάστηκε για να υποστηρίζει την ανάπτυξη εφαρμογών web (web services, web resources και web APIs). Έχει παρόμοια αρχιτεκτονική με το MVC (model view controller). Είναι γραμμένο σε Scala και μπορεί να χρησιμοποιηθεί από γλώσσες που μεταγλωττίζονται σε JVM bytecode. Κύριος στόχος είναι η βελτιστοποίηση της παραγωγικότητας του προγραμματιστή χρησιμοποιώντας διάφορες τεχνικές.

2.3.3 Python

Η Python [8] είναι interpreted γλώσσα προγραμματισμού υψηλού επιπέδου, δηλαδή δεν χρειάζεται να μεταγλωττίσει τον κώδικα σε γλώσσα μηχανής. Είναι σχεδιασμένη έτσι ώστε να δίνει έμφαση στην υποστήριξη κοινά χρησιμοποιημένων μεθοδολογιών προγραμματισμού όπως: σχεδιασμού data structure, object-oriented προγραμματισμού. Παράλληλα ενθαρρύνει τους προγραμματιστές να γράφουν καθαρό κώδικα και επομένως εύκολα διατηρήσιμο παρέχοντας μια κομψή σημειογραφία.

Η Python κατατάσσεται στις πιο δημοφιλείς γλώσσες προγραμματισμού. Είναι αρκετά εύκολη η χρήση της αφού η σύνταξη της είναι απλή. Επιπλέον, η πληθώρα παραδειγμάτων καθιστά την εκμάθηση της ακόμη ευκολότερη. Γενικά, υπάρχουν πάρα πολλές βιβλιοθήκες οι οποίες παρέχουν απεριόριστες δυνατότητες στον προγραμματιστή. Τέλος, ένα μεγάλο πλεονέκτημα είναι η συμβατότητα της γλώσσας με πολλές πλατφόρμες και συστήματα.

2.3.4 Unit Testing

Το Unit Testing είναι μέθοδος όπου γίνονται δοκιμές σε ένα λογισμικό. Η μέθοδος αυτή εστιάζει σε μεμονωμένες μονάδες του πηγαίου κώδικα. Οι μονάδες αυτές είναι σύνολα ενός ή περισσότερων ενοτήτων του προγράμματος μαζί με τα σχετικά δεδομένα ελέγχου όπου και ελέγχονται για να διαπιστωθεί ότι είναι κατάλληλες για χρήση. Τα unit tests συνήθως αυτοματοποιούνται για σκοπούς ευκολίας.

Η χρήση του unit testing βρίσκει προβλήματα στα αρχικά στάδια του κύκλου ανάπτυξης ενός συστήματος. Τα προβλήματα αυτά μπορεί να είναι bugs που δημιουργήθηκαν από την υλοποίηση του προγραμματιστή ή ελλείποντα μέρη της υλοποίησης. Κατά την διάρκεια συγγραφής των unit tests ο προγραμματιστής καλείται να σκεφτεί διάφορα σενάρια εκτέλεσης του κώδικα, έτσι αναγκάζεται να σκεφτεί τις εισόδους (inputs), τα

αποτελέσματα (outputs) και τις συνθήκες σφάλματος (error conditions). Έτσι μπορεί να ορίσει πιο προσεκτικά την επιθυμητή συμπεριφορά του κώδικα. Επιπλέον, κακογραμμένος κώδικας είναι σχεδόν αδύνατο να δοκιμαστεί με unit testing. Αυτό ωθεί τον προγραμματιστή να γράφει δομημένα με χρήση συναρτήσεων και αντικειμένων. Τέλος, η σωστή εφαρμογή των ποιο πάνω μειώνει το κόστος εύρεσης προβλημάτων.

2.3.5 Postman

Το Postman είναι μια πλατφόρμα όπου κάποιος χρήστης μπορεί να κάνει δοκιμές σε RESTful APIs. Είναι πολύ βοηθητικό κατά την ανάπτυξη ενός API αφού επιτρέπει στον χρήση να στείλει διάφορα αιτήματα (requests) και να πάρει απάντηση (response) χωρίς να σπαταλά πολύτιμο χρόνο για να γράψει διάφορες δοκιμές. Για να στείλει ένα τέτοιο αίτημα ο χρήστης αρχικά επιλέγει το είδος του αιτήματος (GET, POST, DELETE κτλ.) και στην συνέχεια δίνει την διεύθυνση (URL) του αιτήματος στο address bar. Εάν το αίτημα που προσπαθεί να στείλει χρειάζεται κάποιο body (π.χ. API-KEY για σκοπούς επαλήθευσης) το Postman παρέχει αυτή την δυνατότητα. Επίσης υπάρχουν και άλλες κατηγορίες πέραν του body, όπως Authorization, Headers όπου επιτρέπουν στον χρήστη να κτίζει πολύπλοκα αιτήματα.

2.3.6 Unix και προγραμματισμός κελύφους

Το Unix είναι μια οικογένεια από λειτουργικά συστήματα τα οποία υποστηρίζουν πολλαπλές εργασίες (multi-tasking) και πολλαπλούς χρήστες (multi-user). Τα συστήματα Unix χαρακτηρίζονται από τον σπονδυλωτό σχεδιασμό (modular design) που έχουν, γνωστό και ως φιλοσοφία Unix. Η φιλοσοφία αυτή πρέπει να παρέχει ένα σύνολο από εργαλεία. Αυτά είναι, ένα σύστημα αρχείων (file system), έναν μηχανισμό για επικοινωνία μεταξύ διεργασιών (γνωστό ως pipe) και μια γλώσσα scripting κελύφους που σε συνδυασμό με μια γλώσσα εντολών απαρτίζουν τον κέλυφος (Unix shell).

Το bash είναι Unix κέλυφος και γλώσσα εντολών (command language) το οποίο τρέχει διάφορες εντολές, συνήθως σε παράθυρο, τις οποίες δίνει χρήστης. Επίσης, το bash μπορεί να διαβάσει και να τρέξει εντολές από αρχεία, τα λεγόμενα shell scripts. Το bash, όπως και όλα τα Unix κελύφη, παρέχει διάφορες τεχνικές προγραμματισμού. Μερικές από αυτές είναι διασωλήνωση (piping), χρήση μεταβλητών, επαναλήψεις (iterations), συνθήκες κτλ. Χρησιμοποιώντας τις τεχνικές αυτές το bash μπορεί να χρησιμοποιηθεί και για unit testing σε APIs, αφού μπορούν να γραφτούν διάφορα scripts τα οποία καλούν κάποιο endpoint (με την βοήθεια της εντολής curl) και επεξεργάζονται το αποτέλεσμα.

Κεφάλαιο 3

Αρχιτεκτονική του Anyplace

3.1 Χρονική αναδρομή στο Anyplace 16

3.2 Anyplace Server & NoSQL αποθήκη δεδομένων 17

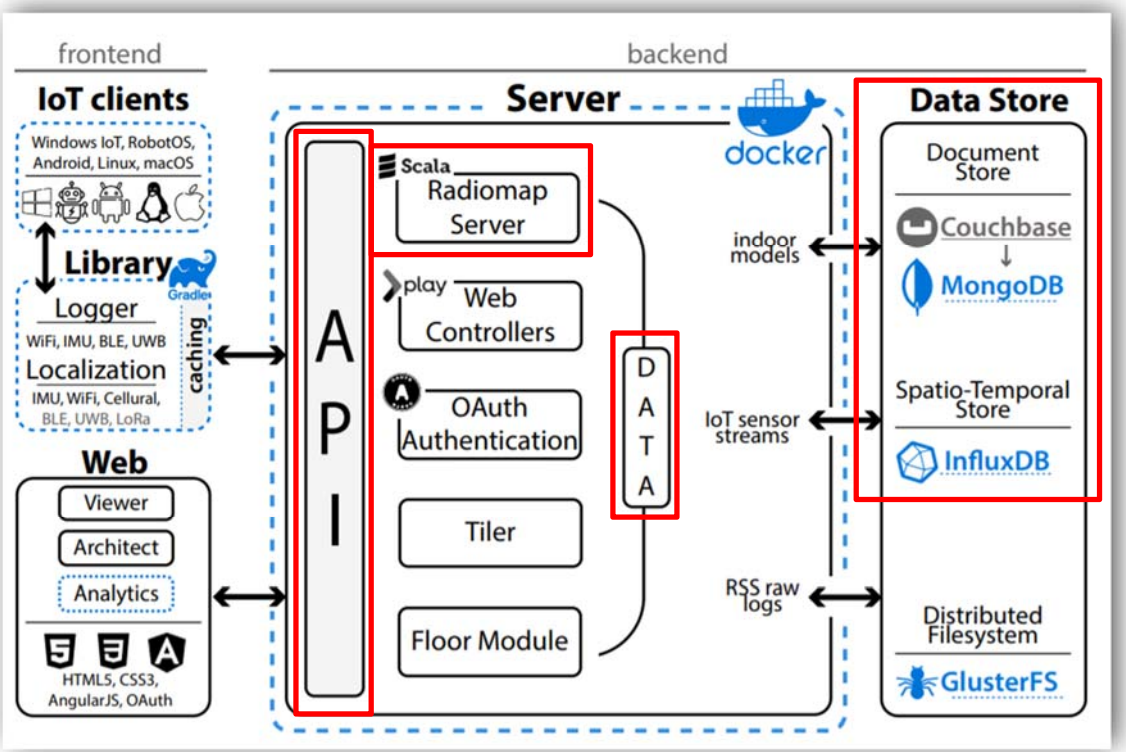
3.3 Anyplace Architect 18

3.4 Anyplace Viewer 22

3.5 Anyplace Navigator & Logger 25

Η πλατφόρμα του Anyplace αποτελείται από πέντε βασικά συστατικά, αυτά είναι ο Anyplace Server, μία NoSQL αποθήκη δεδομένων, ο Anyplace Viewer & Viewer Campus (εφαρμογή ιστού), ο Anyplace Architect (εφαρμογή ιστού) και τέλος ο Anyplace Logger και Anyplace Navigator όπου μαζί απαρτίζουν την Android εφαρμογή του Anyplace.

Σε αυτό το κεφάλαιο θα αναλύσουμε τα πιο πάνω ως σύνολο. Το σύνολο αυτό παρουσιάζεται με περισσότερες λεπτομέρειες στο *Σχήμα 3.1*. Στην ανάλυση αυτή θα γίνει σύντομη περιγραφή κάθε συστατικού αλλά και του συστήματος γενικότερα.



Σχήμα 3.1: Αρχιτεκτονική του συστήματος Anyplace. Σε κόκκινο πλαίσιο τα υποσυστήματα που αφορούν την εν λόγω διπλωματική εργασία. [15]

3.1 Χρονική αναδρομή στο Anyplace

Το 2012 έγιναν τα πρώτα βήματα του Anyplace, αφού βγήκε η πρώτη εκδοχή του, με το όνομα Airplace [9]. Το Airplace είχε ως στόχο την εξακρίβωση τοποθεσίας του χρήστη μέσα στον χώρο με την χρήση αποτυπωμάτων Wi-Fi. Το 2013-2014 εισάγεται η τεχνική πληθοπορισμού (crowdsourcing) [10] όπου πολλά άτομα συμβάλλουν για την επίλυση κάποιου προβλήματος. Το πρόβλημα αυτό ήταν η χαρτογράφηση κτηρίων και συλλογή αποτυπωμάτων Wi-Fi έτσι ώστε το σύστημα, που πλέον ονομάζεται Anyplace [11], να μπορεί να υποστηρίζει πλοήγηση σε περισσότερους εσωτερικούς χώρους. Το 2015 το Anyplace βελτιώνεται αφού δουλεύει πλέον με εγγυήσεις ανωνυμίας [12]. Το 2017 η ζήτηση για localization οδηγεί στην ανάπτυξη τεχνικών που δημιουργούν χάρτες από αποτυπώματα Wi-Fi [13]. Το 2019 εξακολουθεί να υπάρχει περιθώριο βελτίωσης στο localization, έτσι με νέους μεθόδους γίνεται πιο ομαλό και πιο ακριβής [14] . Το 2020 βρισκόμαστε στην έκδοση A4IoT [15] όπου εστιάζει στην εγκατάσταση και χρήση του Anyplace από Raspberry Pi χρησιμοποιώντας time-series βάση δεδομένων (InfluxDB

[16]). Σε αυτό το σημείο αναδεικνύεται η ανάγκη βελτίωσης του υποσυστήματος δεδομένων η οποία επιτεύχθηκε σε αυτή την διπλωματική εργασία.

3.2 Anyplace Server & NoSQL αποθήκη δεδομένων

Ο Anyplace server ακολουθεί μια αρχιτεκτονική μεγάλων δεδομένων όπου παρέχει ένα Web 2.0 API, με την χρήση του Play Framework. Το API μεταφέρει τις πληροφορίες σε μορφή JSON για τις λειτουργείες χαρτογράφησης, πλοήγησης και γεωτοποθέτησης. Όσο αφορά το back-end, χρησιμοποιείται η βάση CouchbaseDB όπου αποθηκεύει τα δεδομένα σε μορφή JSON. Η CouchbaseDB κατατάσσεται στις NoSQL βάσεις δεδομένων και ο κύριος λόγος που χρησιμοποιείται από το σύστημα είναι η εύκολη επεκτασιμότητα και γρήγορη ανάκτηση δεδομένων που παρέχει. Στο σύστημα εισέρχονται, εξέρχονται και αποθηκεύονται δεδομένα τα οποία χειρίζεται ο Server. Τα δεδομένα αυτά μπορούν να χωριστούν σε δύο κατηγορίες:

1. Δεδομένα JSON (μοντέλα κτηρίων, ορόφων , σημείων ενδιαφέροντος, RSS fingerprints) τα οποία αποθηκεύονται στην CouchbaseDB.

2. Δεδομένα binary (π.χ. αρχιτεκτονικά σχέδια ορόφων) τα οποία αποθηκεύονται στον Server.

Εικόνα 3.1: Παράδειγμα αρχείου JSON ενός κτηρίου

Εικόνα 3.2: Παράδειγμα αρχείου JSON ενός RSS fingerprint

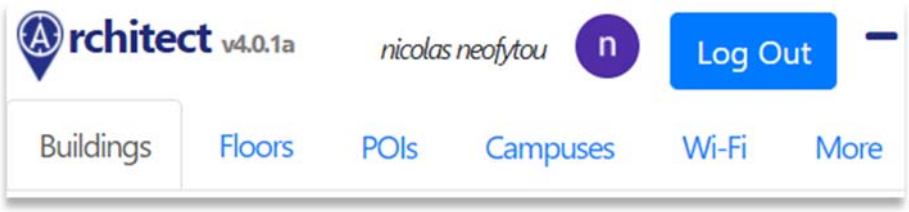
Τα πιο πάνω δεδομένα μπορούν εύκολα να εξαχθούν ή να εισαχθούν σε μορφή JSON (export / import).

3.3 Anyplace Architect

Ο Anyplace Architect είναι μία εφαρμογή ιστού, η οποία είναι φιλική προς τον χρήστη με πολλές λειτουργείες που εξυπηρετούν στην διαχείριση μοντέλων εσωτερικού χώρου. Αυτά είναι κτήρια, όροφοι, σημεία ενδιαφέροντος, ομάδες κτηρίων και σημεία Wi-Fi. Η εφαρμογή αποτελείται από συνδυασμό αρκετών τεχνολογιών ιστού (HTML5, CSS3, JavaScript) που παρέχουν γρήγορη παρουσίαση των μοντέλων που ανήκουν στο Anyplace. Είναι κτισμένη στην πλατφόρμα AngularJS και χρησιμοποιεί κάποιες υπηρεσίες από την Google όπως Maps, Heat-maps, Directions, URL shortener. Για το κτίσιμο της γραφικές διεπαφής χρησιμοποιούνται βιβλιοθήκες και εργαλεία όπως Bootstrap, Font-Awesome και jQuery UI.

Αρχικά, όταν ο χρήστης επισκεφθεί την σελίδα Anyplace Architect βλέπει τον παγκόσμιο χάρτη ο οποίος παρέχεται από το Google Maps. Υπάρχει δυνατότητα παρουσίασης Satellite, OSM, Carto Light και Map. Εάν ο χρήστης επιθυμεί να χρησιμοποιήσει τις υπηρεσίες που παρέχει ο Anyplace Architect πρέπει να συνδεθεί με google Account. Την πρώτη φορά που επιχειρεί να συνδεθεί δημιουργείται ένα μοναδικό id το οποίο αποθηκεύεται στην βάση δεδομένων μαζί με τα στοιχεία του χρήστη. Όταν προσπαθήσει να ξανασυνδεθεί, το σύστημα βρίσκει τον λογαριασμό του με βάση το id αυτό και του επιτρέπει να προχωρήσει. Καθώς αλληλοεπιδρά με το σύστημα κάποιες λειτουργίες όπως

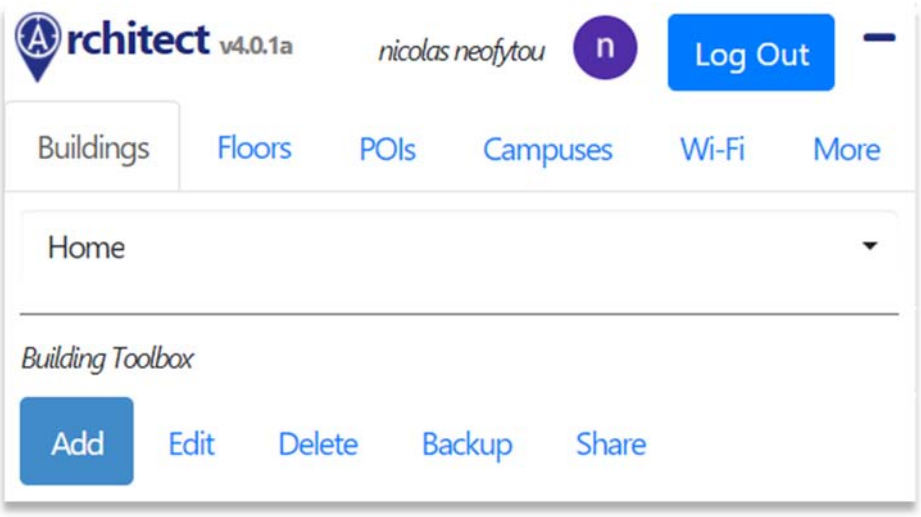
διαγραφή κτηρίου απαιτούν ταυτοποίηση. Η ταυτοποίηση αυτή, αλλά και η λειτουργεία σύνδεσης γίνεται με βάση ένα API-KEY το οποίο παρέχεται από το OAuth 2.0 protocol [17]. Αφού συνδεθεί επιτυχώς εμφανίζεται ένα menu, όπου μέσω αυτού αλληλοεπιδρά με το σύστημα, *Εικόνα* 3.3.



Εικόνα 3.3: Anyplace Architect Menu

Όταν ο χρήστης επιθυμεί να επεξεργαστεί κτήρια επιλέγει την κατηγορία Buildings από το menu. Η κατηγορία Buildings παρέχει ένα Building Toolbox *Εικόνα* 3.4 όπου έχει τις εξής επιλογές: Add, Edit, Delete, Backup, Share. Ο χρήστης μπορεί:

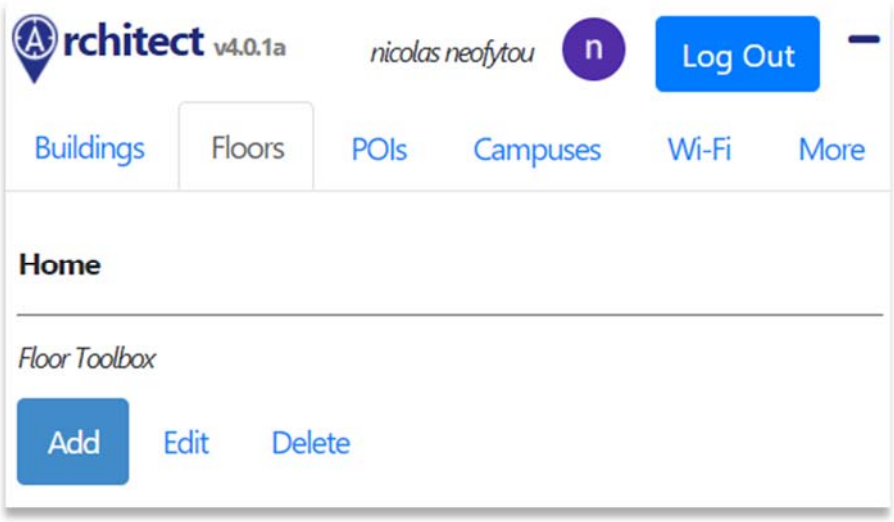
- να προσθέσει καινούριο κτήριο πατώντας το Add. Με drag & drop επιλέγει την τοποθεσία που επιθυμεί να τοποθετήσει το κτήριο και στην συνέχεια συμπληρώνει building code, building name, building description, make building public to view (check box).
- να επεξεργαστεί κάποιο κτήριο πατώντας Edit. Με drag & drop μπορεί να αλλάξει τις συντεταγμένες του κτηρίου και παράλληλα μπορεί να δώσει νέες τιμές στα πεδία building code, building name, building description, make building public to view.
- να διαγράψει κάποιο κτήριο πατώντας Delete, όπου μαζί του διαγράφονται οι όροφοι, τα σημεία ενδιαφέροντος και τα fingerprints Wi-Fi.
- να αποθηκεύσει το κτήριο πατώντας Backup, όπου αποθηκεύει στον υπολογιστή ένα zip-file με όλες τις πληροφορίες που υπάρχουν για το κτήριο (όροφοι, σημεία ενδιαφέροντος κτλ.).
- να μοιραστεί κάποιο κτήριο πατώντας το Share. Αυτό μπορεί να γίνει είτε με URL είτε με διάφορα μέσα κοινωνικής δικτύωσης (Facebook, twitter κτλ.).



Εικόνα 3.4: Anyplace Architect Building Toolbox

Όταν ο χρήστης επιθυμεί να επεξεργαστεί τους ορόφους των κτηρίων επιλέγει την κατηγορία Floors από το menu. Η κατηγορία Floors παρέχει ένα Floor Toolbox *Εικόνα* 3.5 όπου έχει τις εξής επιλογές: Add, Edit, Delete. Ο χρήστης μπορεί:

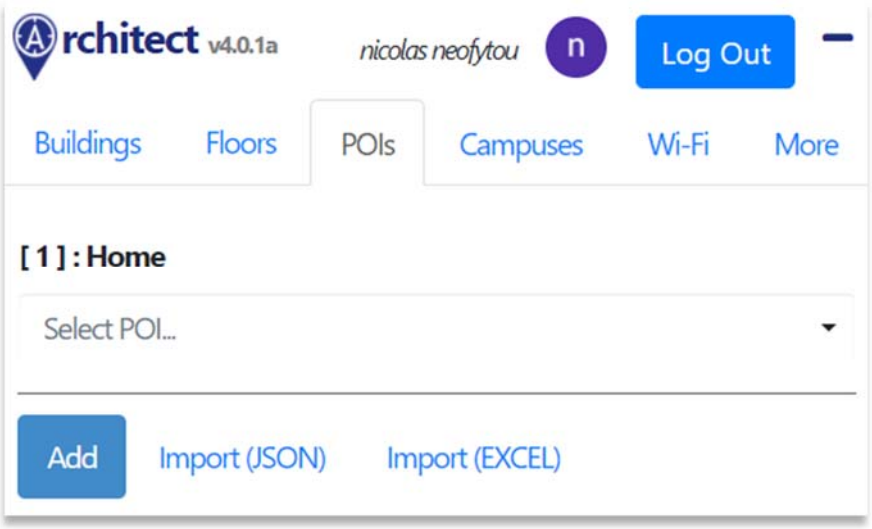
- να προσθέσει ένα όροφο πατώντας το Add. Αφού επιλέξει την κατηγορία αυτή και καθορίσει το κτήριο και τον αριθμό του ορόφου μπορεί να πατήσει στο choose file και να επιλέξει την εικόνα της κάτοψης.
- να αλλάξει την κάτοψη του κτηρίου με μια καινούρια, πατώντας το Edit. Ο κατάλογος αυτός παραπέμπει τον χρήστη την πρώτη κατηγορία, Add, αφού επαναλαμβάνοντας την διαδικασία γίνετε overwrite ο όροφος.
- να διαγράψει τον όροφο πατώντας στο Delete. Με την επιτυχημένη διαγραφή του κτηρίου επίσης διαγράφονται όλα τα σημεία ενδιαφέροντος του ορόφου και τα fingerprints Wi-Fi.



Εικόνα 3.5: Anyplace Architect Floor Toolbox

Όταν ο χρήστης επιθυμεί να επεξεργαστεί τα σημεία ενδιαφέροντος επιλέγει την κατηγορία POIs από το menu. Η κατηγορία POIs παρέχει ένα POI Toolbox *Εικόνα* 3.6 όπου έχει τις εξής επιλογές: Add, Import(JSON), Import(EXCEL). Ο χρήστης μπορεί:

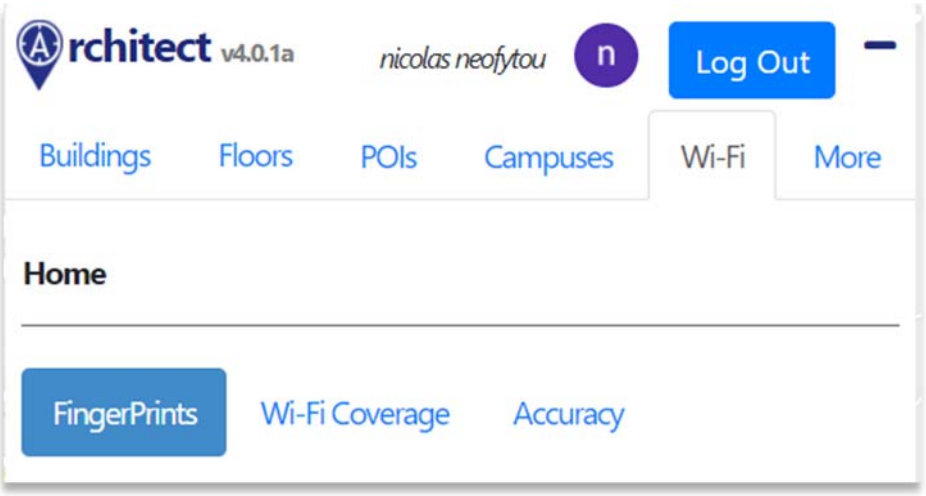
- να προσθέσει σημεία ενδιαφέροντος με drag & drop πατώντας το Add. Υπάρχουν τριών ειδών σημεία: POI, connectors και edges.
- να εξάγει ή να εισάγει δεδομένα σε μορφή JSON πατώντας το Import (JSON).
- να εισάγει δεδομένα από excel αρχείο πατώντας το Import (EXCEL). Στην κατηγορία αυτή υπάρχει σύντομη περιγραφή για την δομή που πρέπει να έχει το excel αρχείο.



Εικόνα 3.6: Anyplace Architect Poi Toolbox

Τέλος, όταν ο χρήστης επιθυμεί να επεξεργαστεί τα σημεία Wi-Fi επιλέγει την κατηγορία Wi-Fi από το menu. Η κατηγορία Wi-Fi παρέχει ένα Wi-Fi Toolbox *Εικόνα* 3.7 όπου έχει τις εξής επιλογές: FingerPrints, Wi-Fi Coverage και Accuracy. Ο χρήστης μπορεί:

- να εμφανίσει και να αποκρύψει τα Wi-Fi fingerprints για σκοπούς καλύτερης ορατότητας, να δει τα fingerprints σε συνάρτηση με τον χρόνο. Αυτές οι λειτουργείες παρέχονται από την κατηγορία Fingerprints.
- να δει το Wi-Fi map και το υπολογιζόμενο Wi-Fi AP position με βάση δύο επιλογές (MAC ή manufacturer). Αυτές οι λειτουργείες παρέχονται από την κατηγορία Wi-Fi coverage.
- να δει το access map, να διαγράψει το radio map. Αυτές οι λειτουργείες παρέχονται από την κατηγορία Accuracy.



Εικόνα 3.7: Anyplace Architect Wi-Fi Toolbox

3.4 Anyplace Viewer & Viewer Campus

Ο Anyplace Viewer είναι επίσης μια εφαρμογή ιστού η οποία εστιάζει στην γρήγορη πρόσβαση των κτηρίων του Anyplace. Δεν χρειάζεται η εγκατάσταση κάποιας εφαρμογής αλλά ούτε και η σύνδεση/εγγραφή του χρήστη στο σύστημα. Το μόνο προαπαιτούμενο είναι ένας περιηγητής και ο χρήστης μπορεί να επισκεφτεί την σελίδα. Για να καταλάβουμε τον λόγο υλοποίησης του Viewer αρκεί να ακολουθήσουμε το πιο κάτω σενάριο.

Ένας νέος υπάλληλος μιας εταιρίας με μεγάλες εγκαταστάσεις στην προσπάθεια του να μεταβεί σε ένα κτήριο έχει χαθεί. Καθώς κινείται στον χώρο βλέπει ένα ενημερωτικό

φυλλάδιο που λέει ότι πρόσφατα τα κτήρια της εταιρίας αυτής υποστηρίζονται από την υπηρεσία του Anyplace. Έτσι, επισκέπτεται την ιστοσελίδα του Anyplace Viewer όπου αμέσως βλέπει τα κοντινότερα κτήρια από το σημείο που βρίσκεται αλλά και δυνατότητες αναζήτησης και πλοήγησης προς αυτά.

Εικόνα 3.8: Anyplace Viewer

Επίσης, υπάρχει και ο Viewer Campus. Ο Viewer Campus ομαδοποιεί ένα αριθμό κτηρίων και δημιουργεί ένα μοναδικό Viewer με τα κτήρια αυτά. Η *Εικόνα* 3.9 είναι ένα Campus και όπως φαίνεται δείχνει μόνο κάποια κτήρια.

Εικόνα 3.9: Anyplace Viewer Campus. UCY Campus

Ο Anyplace Viewer δεν παρέχει μόνο οδηγίες πλοήγησης από την τρέχων τοποθεσία του χρήστη προς ένα σημείο ενδιαφέροντος αλλά και οδηγίες πλοήγησης μεταξύ δύο σημείων ενδιαφέροντος που επιλέγει ο χρήστης με την προϋπόθεση να βρίσκονται στο ίδιο κτήριο. Ακόμη ο χρήστης μπορεί να μοιραστεί ένα σημείο ενδιαφέροντος είτε με URL είτε με HTML5 κώδικα iframe. Επίσης μπορεί να δει την περιγραφή ενός σημείου. Οι λειτουργείες αυτές παρέχονται από ένα πολύ απλό menu, *Εικόνα* 3.10.

Εικόνα 3.10: Anyplace Viewer POIs menu

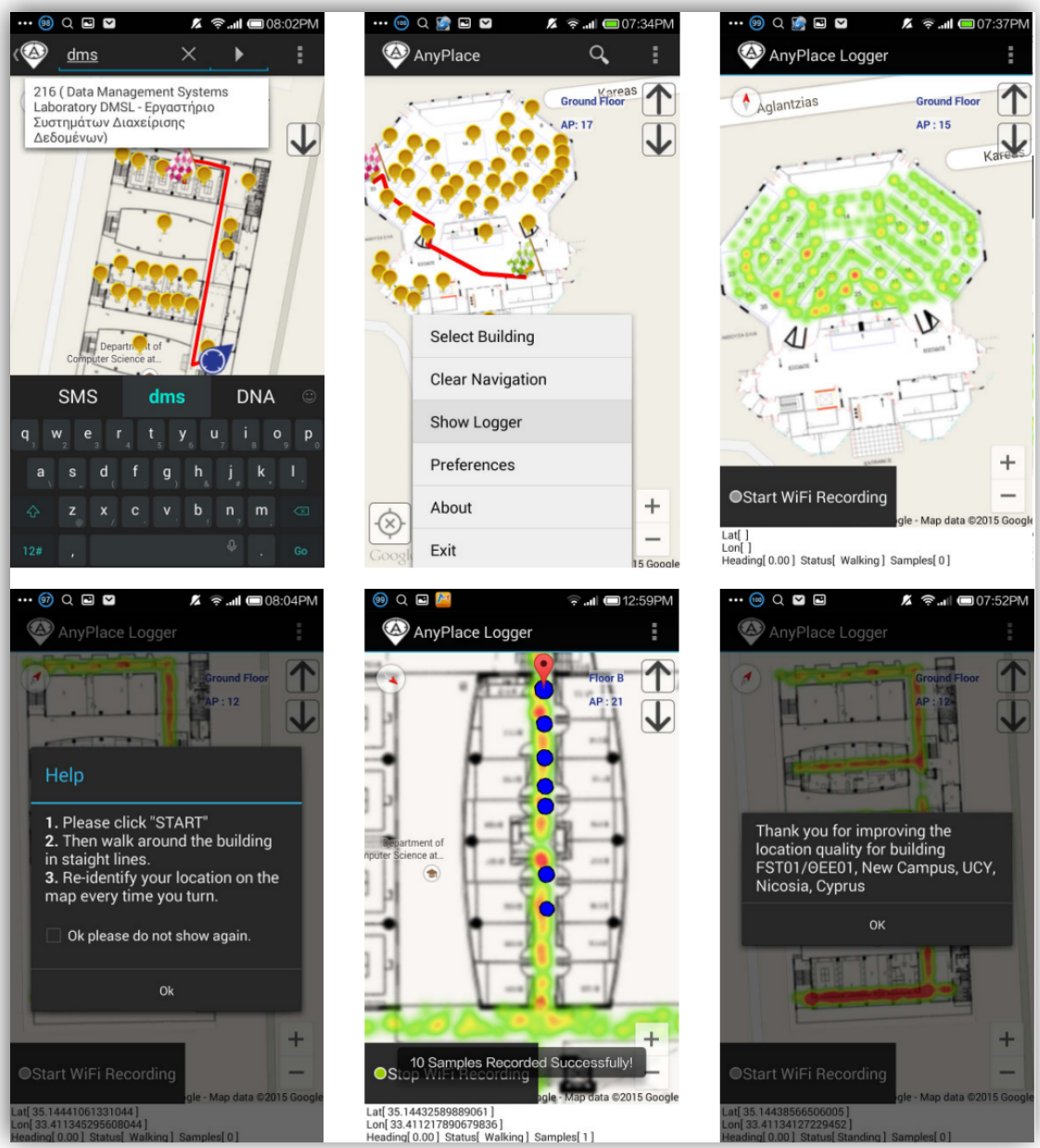
Γενικά, ο Anyplace Viewer είναι ιδανικός για καινούριους χρήστες που δεν έχουν εμπειρία με το σύστημα Anyplace. Εάν κάποιος χρήστης που επιθυμεί περισσότερες δυνατότητες μπορεί να εγκαταστήσει την εφαρμογή Android.

3.5 Anyplace Navigator & Logger

Η εφαρμογή του Anyplace αποτελείται από δύο μέρη, τον Anyplace Navigator και Anyplace Logger. Είναι για χρήστες Android και είναι διατεθειμένη στο Google Play. Η εφαρμογή εκμεταλλεύεται την ύπαρξη δακτυλικών αποτυπωμάτων από σήματα Wi-Fi και έτσι παρέχει την λειτουργία γεωτοποθέτησης. Επιπλέον, ο χρήστης μπορεί να δει την τρέχουσα τοποθεσία του στον χάρτη και να περιηγηθεί σε αυτόν με παρόμοια διαπροσωπία του Anyplace Viewer. Το διαφορετικό που παρέχει η εφαρμογή σε σύγκριση με την εφαρμογή ιστού είναι η εξαιρετική ακρίβεια των 1,96 μέτρων και η δυνατότητα καταγραφής RSS.

Όταν ο χρήστης ανοίξει την εφαρμογή του Anyplace βλέπει τον Anyplace Navigator και μπορεί να μεταφερθεί στον Anyplace Logger από το menu (πατώντας Show Logger *Εικόνα* 3.11 στιγμιότυπο 2). Ο Navigator παρέχει την θέση του χρήστη μέσα στο κτήριο αλλά και την δυνατότητα πλοήγησης από ένα σημείο ενδιαφέροντος σε ένα άλλο. Κατά την εκκίνηση της εφαρμογής ο χρήστης μπορεί να δει τα κτήρια που βρίσκονται γύρο του, στην πλατφόρμα Google Maps. Εάν επιθυμεί να δει περισσότερες πληροφορίες για ένα κτήριο τότε μπορεί να το επιλέξει και τότε εμφανίζονται οι όροφοι, τα σημεία ενδιαφέροντος μαζί με τους σχετικούς ραδιοχάρτες. Σε αυτή τη φάση ο χρήστης μπορεί να επιλέξει τον προορισμό του.

Όπως προανέφερα, η εφαρμογή έχει υψηλή ακρίβεια μέσα στον χώρο. Ο τρόπος που υπάρχει αυτή η ακρίβεια είναι με την συλλογή αποτυπωμάτων εντάσεως σημάτων Wi-Fi από τον Anyplace Logger. Ο Logger, όπως και ο Architect, είναι ανοικτός προς το κοινό και επιτρέπει σε όποιο χρήστη επιθυμεί να συνεισφέρει στο σύστημα συλλέγοντας αποτυπώματα (*Εικόνα* 3.11 στιγμιότυπο 3, “Start WiFi Recording”). Αφού ο χρήστης επιλέξει το κτήριο, τον όροφο και την τοποθεσία που βρίσκεται τότε μπορεί να ξεκινήσει την καταγραφή αποτυπωμάτων. Τότε ο Logger καταγράφει διάφορες μετρήσεις από τα σημεία πρόσβασης Wi-Fi και τα συσχετίζει με την τοποθεσία που υπέδειξε ο χρήστης. Σε αυτό το σενάριο ο χρήστης έχει επίσης την δυνατότητα να κινείται στον χώρο σε ευθείες αφού υποδείξει την αρχή και το τέλος της διαδρομής που θα κάνει. Έτσι ο Logger υπολογίζει και διαμοιράζει ομοιόμορφα τις μετρήσεις που πάρθηκαν καθώς ο χρήστης περπατούσε. Στην συνέχεια υπάρχει δυνατότητα προβολής heat-map των σημείων προτού σταλθούν στον Anyplace Server για αποθήκευση. Η διαδικασία αυτή παρουσιάζεται στην *Εικόνα* 3.11 με τα 3 τελευταία στιγμιότυπα.



Εικόνα 3.11: Anyplace Navigator [11]

Κεφάλαιο 4

Νέο υποσύστημα δεδομένων

4.1 Εισαγωγή 28

4.2 Πρόβλημα 29

4.3 Δημιουργία και εγκατάσταση νέας βάσης δεδομένων 30

4.4.1 Εξαγωγή δεδομένων 30

4.4.2 Μελέτη δεδομένων 31

4.4.3 Δημιουργία σχήματος 32

4.4.4 Δημιουργία βάσης MongoDB 35

4.4.5 Εισαγωγή δεδομένων 35

4.4.6 Επικοινωνία βάσης δεδομένων – κώδικα Anyplace 38

4.1 Εισαγωγή

Σε αυτό το κεφάλαιο θα παρουσιάσουμε τη διαδικασία με την οποία αναπτύχθηκε η νέα βάση δεδομένων και πως εντάχθηκε στο σύστημα Anyplace. Παράλληλα θα δείξουμε γιατί ήταν αναγκαία αυτή η αλλαγή. Αρχικά έγινε μελέτη στα δεδομένα που υπήρχαν στο προηγούμενο υποσύστημα δεδομένών. Στην συνέχεια δημιουργήθηκε ένα σχήμα για την καινούρια βάση δεδομένων με βάση την μελέτη που έγινε. Αφού δημιουργήθηκε μια βάση τότε γράφτηκαν κάποια scripts όπου εξάγουν όλα τα δεδομένα από την CouchbaseDB, τα επεξεργάζονται, τα ομαδοποιούν και τέλος τα εισάγουν στην βάση αυτή. Τότε άρχισε η μετάβαση από CouchbaseDB σε MongoDB αφού έπρεπε να αλλαχτεί ο κώδικας έτσι ώστε να εισάγει και να εξάγει δεδομένα από την καινούρια βάση.

4.2 Πρόβλημα

Τα προβλήματα που ώθησαν στην αλλαγή του υποσυστήματος δεδομένων του συστήματος μπορούν να χωριστούν σε δύο κατηγορίες. Η μία κατηγορία αφορά τους υπολογιστικούς πόρους που απαιτεί το σύστημα και η άλλη αφορά την κακή διαχείριση των δεδομένων.

Η αρχική έκδοση του Anyplace χρησιμοποιούσε την CouchbaseDB αφού επέτρεπε εύκολη και γρήγορη ανάκτηση των δεδομένων. Ωστόσο, οι υπολογιστικές απαιτήσεις αυτής της προσέγγισης καθιστούν το σύστημα μη-αποτελεσματικό σε συσκευές αιχμής, όπως το δημοφιλές ARMv7 Raspberry PI. Άρα προέκυψε η ανάγκη για μία πιο ελαφρά βάση δεδομένων.

Ένα άλλο εξίσου σημαντικό πρόβλημα ήταν η κακή διαχείριση αποθήκευσης των δεδομένων. Δεν υπήρχε ξεχωριστή διαχείριση για τα διαφορετικά μοντέλα που έχει το σύστημα αλλά αποθηκεύονταν όλα μαζί, που συνολικά ήταν 11,250,308. Αυτά τα 11 εκατομμύρια αντικείμενα μπορούσαν να χωριστούν σε διάφορες κατηγορίες όπως θα δούμε πιο κάτω.

Το σύστημα για να μπορεί να δουλεύει γρήγορα και αποτελεσματικά χρειαζόταν indexes και views. Στην ουσία ήταν το αποτέλεσμα γραμμένο σε ένα αρχείο και όταν το σύστημα δεχόταν κάποιο αίτημα για ένα endpoint τότε επέστρεφε το αποτέλεσμα γρήγορα. Όμως κάποια views δεν τέλειωναν ποτέ λόγο του μεγάλου όγκου των δεδομένων. Έτσι έτρεχαν μέχρι να γίνουν expire ή να τελειώσει ο δίσκος και μετά ξεκινούσαν από την αρχή. Αυτό εξαντλούσε τους υπολογιστικούς πόρους έτσι έπρεπε να απενεργοποιηθούν με αποτέλεσμα κάποιες λειτουργίες του συστήματος να μην δουλεύουν.

Ένα άλλο πρόβλημα που προέκυψε από την κακή αποθήκευση των δεδομένων είναι ότι εάν θέλαμε να ψάξουμε ένα αντικείμενο έπρεπε να ψάχνουμε σε ένα πολύ μεγαλύτερο σύνολο παρά στο σύνολο του αντικειμένου αυτού. Για παράδειγμα εάν θέλαμε να βρούμε όλα τα κτήρια, τα οποία είναι συνολικά 4439, έπρεπε να ελέγχουμε μέσα στα 11 εκατομμύρια αντικείμενα που είχε η αποθήκη δεδομένων, ενώ εάν ήταν κατηγοριοποιημένα θα επιστρέφαμε την κατηγορία κτήρια.

Πιο κάτω μπορούμε να δούμε το view που ελέγχει εάν το αντικείμενο έχει κάποια συγκεκριμένα πεδία και ταυτόχρονα δεν έχει κάποια άλλα για να ταυτοποίηση ότι είναι

κτήριο. Αυτό είναι κακή τεχνική αφού με την παραμικρή αλλαγή το view πρέπει να ξανατρέξει έτσι ώστε να ενημερώσει την απάντηση.

```
"building_all": {
   "map": "function (doc, meta) {
      if( !doc.puid && !doc.floor_number && !doc.edge_type && doc.buid
            && doc.is_published == \"true\")
         emit(meta.id, doc);
         emit(meta.id, null);
      }"
}
```

4.3 Δημιουργία και εγκατάσταση νέας βάσης δεδομένων

4.3.1 Εξαγωγή δεδομένων

Το πρώτο βήμα της υλοποίησης ήταν η εξαγωγή των δεδομένων του Anyplace από την CouchbaseDB. Η διαδικασία αυτή ήταν σχετικά απλή αφού η Couchbase προσφέρει δύο εντολές, την cbexport και την cbexportjson, οι οποίες μπορούν να εκτελεστούν σε Unix κέλυφος. Χρησιμοποίησα την cbexportjson αφού τα δεδομένα ήταν αποθηκευμένα σε JSON μορφή. Η εντολή αυτή παρέχει στον χρήστη δύο επιλογές, να εξάγει τα δεδομένα σε λίστα ή σε πολλαπλές γραμμές όπου μία γραμμή αντιστοιχεί σε ένα αντικείμενο JSON. Επέλεξα να εξάγω τα δεδομένα σε πολλαπλές γραμμές αφού θα ήταν πιο εύκολο να τα επεξεργαστώ.

Παρόλο που η εντολή εκτελείται σε κέλυφος επέλεξα να γράψω το script σε Python (pull_couchbase.py) αφού σε επόμενο στάδιο θα επεξεργαζόμουν τα JSON αντικείμενα, και η διαχείριση των αντικειμένων θα ήταν πολύ πιο εύκολη με Python σε σύγκριση με το κέλυφος. Έτσι με την βοήθεια της βιβλιοθήκης subprocess εκτέλεσα την εντολή cbexportjson. Η εντολή αυτή δέχεται διάφορες μεταβλητές. Αυτές είναι -c url, -u username, -p password -b bucket -f format και -o output. Το url είναι η διεύθυνση του cluster όπου βρίσκονται τα δεδομένα, το username και password χρειάζονται για πρόσβαση στα δεδομένα, το bucket είναι το όνομα του bucket που είναι αποθηκευμένα τα δεδομένα, το format αναφέρεται στην μορφή που θα εξαχθούν τα δεδομένα και τέλος το output είναι το μονοπάτι που επιθυμεί ο χρήστης να αποθηκεύσει τα δεδομένα.

```
subprocess.run(["/opt/couchbase/bin/cbexport", "json", "-c", URL, "-u",
CDB_USERNAME, "-p", CDB_PASSWORD, "-b", CDB_BUCKET, "-f", "lines", "-o",
allDocs])
```

4.3.2 Μελέτη δεδομένων

Αφού εξήγαγα τα δεδομένα τότε μπορούσα να κάνω διάφορες μετρήσεις και δοκιμές με αυτά. Η προσέγγιση που ακολούθησα ήταν να ομαδοποιήσω αντικείμενα που είχαν κοινά key - values. Το τελικό αποτέλεσμα ήταν η ομαδοποίηση των αντικειμένων σε επτά διαφορετικές κατηγορίες. Οι κατηγορίες αυτές έχουν τα εξής πεδία (με πράσινο χρώμα τα πεδία που υπήρχαν σε όλα τα αντικείμενα, ενώ με κόκκινο είναι πεδία που δεν υπήρχαν σε όλα τα αντικείμενα):

Κατηγορία 1: κτήρια (buildings)

Πεδία σε όλα τα αντικείμενα: address, buid, coordinates_lat, coordinates_lon, description, geometry, coordinates, is_published, name, url
Πεδία σε κάποια αντικείμενα: bucode, geometry, username_creator, co_owners, owner_id

Κατηγορία 2: ομάδες κτηρίων (campuses)

Πεδία σε όλα τα αντικείμενα: buids, cuid, description, name, owner_id
Πεδία σε κάποια αντικείμενα: greeklish

Κατηγορία 3: ακμές (edges)

Πεδία σε όλα τα αντικείμενα: buid, buid_a, buid_b, cuid, edge_type, floor_a, floor_b, is_published, pois_a, pois_b, weight

Κατηγορία 4: αποτυπώματα Wi-Fi (fingerprintsWifi)

Πεδία σε όλα τα αντικείμενα: MAC, buid, floor, geometry, heading, rss, timestamp, x, y
Πεδία σε κάποια αντικείμενα: strongestWifi

Κατηγορία 5: όροφοι (floorplans)

Πεδία σε όλα τα αντικείμενα: buid, description, floor_name, floor_number, fuid, is_published
Πεδία σε κάποια αντικείμενα: username_creator, bottom_left_lat, bottom_left_lng, top_right_lat, top_right_lng, height, width, zoom

Κατηγορία 6: σημεία ενδιαφέροντος (pois)

Πεδία σε όλα τα αντικείμενα: buid, coordinates_lat, coordinates_lon, description, floor_name, floor_number, geometry, is_building_entrance, is_door, is_published, name, pois_type, puid, url
Πεδία σε κάποια αντικείμενα: username_creator, image

Κατηγορία 7: χρήστες (users)

Πεδία σε όλα τα αντικείμενα: doc_type, owner_id, type
Πεδία σε κάποια αντικείμενα: name, auid, clients, doctype, email, isadmin, nickname, password, scope, username

Η κατηγορίες αυτές είναι στην ουσία όλα τα μοντέλα εσωτερικών χώρων που αποθηκεύει το Anyplace.

4.3.3 Δημιουργία σχήματος

Με το τέλος της μελέτης έπρεπε να γραφτεί κώδικας όπου διαβάζει όλα τα αντικείμενα που έγιναν export, καθορίζει σε ποια κατηγορία ανήκει το κάθε αντικείμενο και στην συνέχεια να το αποθηκεύει σε ένα αρχείο έτσι ώστε να γίνει εισαγωγή των δεδομένων. Ο κώδικας αυτός, που επίσης γράφτηκε σε python (defineCollections.py), καλείται αυτόματα από το script που εξάγει τα δεδομένα. Κάθε αντικείμενο όμοιας κατηγορίας αποθηκεύεται στο ίδιο αρχείο έτσι ώστε να υπάρχει ένα αρχείο για κάθε κατηγορία. Υπάρχει εξαίρεση στα αντικείμενα κατηγορίας αποτυπωμάτων Wi-Fi, *Εικόνα* 4.1. Αποφάσισα να αποθηκεύω στο ίδιο αρχείο αποτυπώματα που ανήκουν στο ίδιο κτήριο. Η αιτία της ξεχωριστής διαχείρισης των αντικειμένων αυτών ήταν για να γίνει μείωση στον χρόνο εισαγωγής των αντικειμένων στην βάση, αφού τα αντικείμενα αποτυπωμάτων ήταν πάρα πολλά και κατά την εισαγωγή τους γίνονταν κάποιες αλλαγές, *Εικόνα* 4.4.

```
if isBuilding(obj):
    fixed_obj = fixBUILDING(obj)
    b.write(json.dumps(fixed_obj))
elif isCampus(obj):
    fixed_obj = fixCAMPUS(obj)
    c.write(json.dumps(fixed_obj))
elif isEdge(obj):
    fixed_obj = fixEDGES(obj)
    e.write(json.dumps(fixed_obj))
elif isFingerprint(obj):
    fixed_obj = fixFINGERPRINT(obj)
    splitToBuid(fixed_obj, fingPath)
    fingerprints += 1
elif isFloorPlan(obj):
    fixed_obj = fixFLOORPLAN(obj)
    fl.write(json.dumps(fixed_obj))
elif isPois(obj):
    fixed_obj = fixPOIS(obj)
    p.write(json.dumps(fixed_obj))
elif isUser(obj):
    fixed_obj = fixUSER(obj)
    u.write(json.dumps(fixed_obj))
```

Πιο πάνω φαίνεται ένα κομμάτι του κώδικα που χωρίζει τα αντικείμενα σε κατηγορίες. Μπορεί να παρατηρηθεί ότι όλα τα αντικείμενα τυγχάνουν επεξεργασία από την

αντίστοιχη μέθοδο με βάση την κατηγορία τους (π.χ. fixBUILDING για τα κτήρια). Η επεξεργασία αυτή κάνει κάποιες αλλαγές στα αντικείμενα πριν εισαχθούν στη νέα βάση δεδομένων. Η αλλαγές αυτές ήταν πρόσθεση νέων πεδίων, αφαίρεση πεδίων και μετονομασία πεδίων. Συγκεκριμένα:

- Προστέθηκε το πεδίο _schema όπου υποδεικνύει την εκδοχή του σχήματος. Δόθηκε η τιμή 0. Σε μεταγενέστερες αλλαγές του σχήματος η τιμή θα αλλάξει σε 1 κτλ.
- Τα πεδία address, url, description, name, bucode, username_creator .τα οποία εμφανίζονται σε διάφορες κατηγορίες, πολλές φορές ήταν κενές συμβολοσειρές. Όσα από αυτά ήταν κενά αφαιρέθηκαν.
- Χρήστες: Το πεδίο doc_type αφαιρέθηκε. Το πεδίο type μετονομάστηκε σε external. Προστέθηκε το πεδίο type όπου υποδηλώνει την ιδιότητα του χρήστη. Όλοι οι χρήστες, είναι type user εκτός και αν το owner_id τους βρίσκεται σε μια προκαθορισμένη λίστα (admins[]) η οποία υποδηλώνει ότι το type τους θα είναι admin.

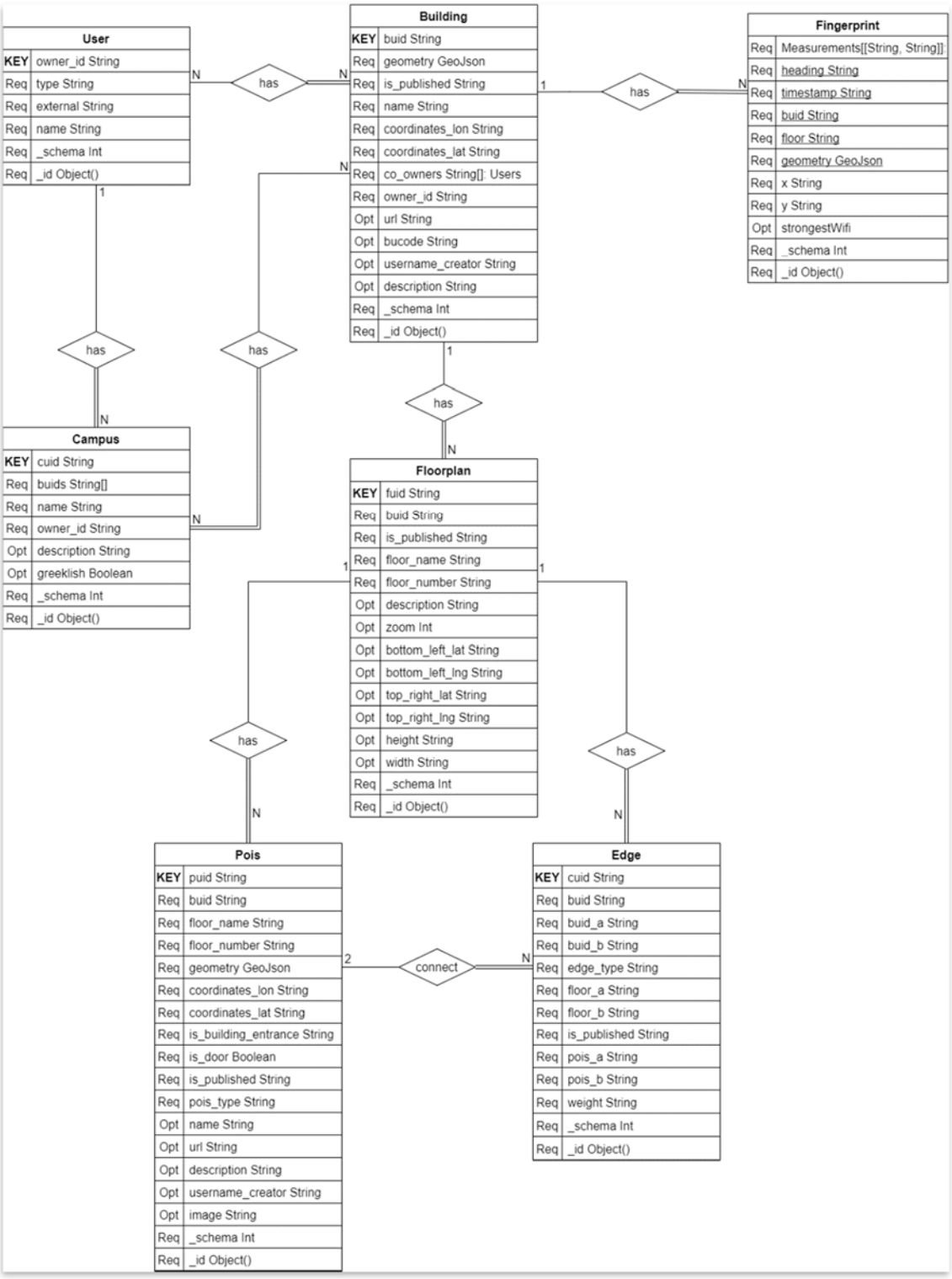
Όταν ο κώδικας εκτελεστεί τυπώνει μία αναφορά όπου δείχνει πόσα αντικείμενα βρέθηκαν ανά κατηγορία. Στην *Εικόνα* 4.1 μπορούμε να δούμε την αναφορά αυτή. Επίσης, τα δύο αντικείμενα που ήταν undefined δεν ταίριαζαν με κάποια άλλα αφού είχαν μοναδική μορφή και μάλλον δεν εισάχθηκαν από το σύστημα, αλλά από κάποιο άτομο ίσως για δοκιμαστικούς σκοπούς.

```
Buildinds:  4439
Campus: 258
Edge:  45269
Fingerprints:  11142975
Floorplans:  3975
Pois:  49037
Users:  4353
Undefined:  2
```

Εικόνα 4.1: Αποτέλεσμα εκτέλεσης defineCollections.py

Πιο κάτω φαίνεται το σχεσιακό διάγραμμα (ER) που προκύπτει. Το διάγραμμα αυτό δείχνει τα πεδία της κάθε κατηγορίας μετά την επεξεργασία που έτυχαν. Η δομή των αντικειμένων fingerprintsWifi εξηγείται αναλυτικά πιο κάτω. Σημαντικό να αναφέρουμε

ότι στο παρελθόν δεν υπήρχε κανένα σχήμα αλλά τα αντικείμενα αποθηκεύονταν όλα μαζί σε ένα bucket. Επίσης αυτό δεν είναι το τελικό σχήμα αφού κατά την διάρκεια της μετάβασης από Couchbase σε MongoDB δημιουργήθηκαν νέα collections όπου και αναλύονται στην συνέχεια (υποκεφάλαιο 4.3.5).



Εικόνα 4.2: Σχήμα της βάσης MongoDB

4.3.4 Δημιουργία βάσης MongoDB

Στην συνέχεια, αφού χωρίστηκαν και αποθηκευτήκαν τα αντικείμενα σε κατηγορίες το επόμενο βήμα ήταν η εισαγωγή τους στην βάση δεδομένων. Άρα έπρεπε να δημιουργηθεί η βάση. Αρχικά έγινε εγκατάσταση της MongoDB και προσαρμογή κάποιων ρυθμίσεων, π.χ. εισαγωγή διεύθυνσης του server στα conf αρχεία για να μπορώ να έχω πρόσβαση στην βάση από απομακρυσμένη σύνδεση. Στην συνέχεια, μέσω του Mongo Shell με την χρήση της εντολής createUser μετέφερα την βάση admin (βάση που παράγεται αυτόματα από την MongoDB) στον χρήστη "admin" με συγκεκριμένα στοιχεία σύνδεσης. Αυτό έγινε για σκοπούς ασφαλείας.

```
db.createUser({
   user: "admin",
   pwd: "password",
   roles: [
   { role: "userAdminAnyDatabase", db: "admin" },
   "readWriteAnyDatabase",
   ],
 });
```

Τότε, ενώθηκα με τα στοιχεία του χρήστη admin και δημιούργησα μια καινούρια βάση για το σύστημα Anyplace, την οποία ονόμασα anyplace. Έτσι, υπάρχει ο χρήστης "admin", ο οποίος έχει πρόσβαση σε όλες της βάσεις δεδομένων, αλλά και ο χρήστης "user" που έχει πρόσβαση μόνο στην βάση anyplace. Αυτό έγινε για σκοπούς ασφαλείας. Όλες οι επαφές με την βάση θα γίνονται με τον χρήστη "user".

```
db.createUser({
  user: "username",
  pwd: "password",
  roles: [{ role: "dbOwner", db: "anyplace" }],
});
```

Η διαδικασία αυτή καταγράφηκε από την αρχή μέχρι το τέλος σε αρχείο README έτσι ώστε χρήστες του Anyplace να μπορούν με ευκολία να ακολουθήσουν αυτό τον οδηγό.

4.3.5 Εισαγωγή δεδομένων

Αφού δημιούργησα την βάση τότε έπρεπε να γράψω ένα πρόγραμμα το οποίο εισάγει τα δεδομένα. Το πρόγραμμα αυτό (migration_0.py), αρχικά ενώνεται στην βάση anyplace με τον χρήστη "user" και ελέγχει πόσα collection υπάρχουν. Εάν αυτά είναι επτά (7) τότε κοιτάζει τα ονόματα τους και αν είναι αυτά που πρέπει (buildings, campuses, edges, fingerprintsWifi, floorplans, pois, users) ενημερώνει τον χρήστη ότι βρέθηκαν όλα και του δίνει την επιλογή να τα διαγράψει όλα για να εισαχθούν ξανά. Εάν ένα από αυτά δεν

ταιριάζει με κάποιο από τα ονόματα που πρέπει τότε ενημερώνει τον χρήστη με το ανάλογο μήνυμα. Τα ονόματα είναι σημαντικά αφού όταν εκτελείται ένα query ψάχνει σε συγκεκριμένο collection (π.χ. anyplace.*users*.find("name": "user")). Εάν τα collections δεν είναι επτά τότε το σύστημα αναγνωρίζει ποια δεν υπάρχουν και τότε ανοίγει τα ανάλογα αρχεία και τα προσθέτει. Το ίδιο γίνεται εάν δεν υπάρχει κανένα collection.

Το πρόγραμμα διαβάζει τα αρχεία ένα – ένα και αναλόγως το αρχείο προσθέτει τα αντικείμενα που περιέχει σε ένα collection. Αυτό δεν ισχύει για τα αρχεία που περιέχουν αποτυπώματα Wi-Fi αφού πριν εισαχθούν τυγχάνουν ειδική επεξεργασίας για τον λόγο ότι κατανάλωναν άσκοπα μεγάλη ποσότητα μνήμης.

Συγκεκριμένα, όταν το υποσύστημα δεδομένων του Anyplace λάμβανε κάποιες μετρήσεις αποτυπωμάτων Wi-Fi δημιουργούσε ένα αντικείμενο για την κάθε μία. Τα αντικείμενα αυτά είχαν πολλά κοινά πεδία. Αυτά ήταν το buid, το timestamp, το heading, το floor, το x, το y, το geometry και strongestWifi. Επομένως αποθηκεύονταν πολλά αντικείμενα που είχαν αυτά τα πεδία ίδια και διέφεραν μόνο στο MAC και rss. Αφού είχα διαχωρίσει τα αποτυπώματα Wi-Fi ανά κτήριο (εσκεμμένα στο πρόγραμμα defineCollections.py) μπορούσα να ελέγχω ποια από τα αυτά είχαν τα συγκεκριμένα πεδία ίδια και να τα αντικαθιστούσα με ένα νέο αντικείμενο όπου έχει το πεδίο measurements. Το πεδίο αυτό θα περιέχει key – value τιμές, MAC – rss. Με αυτό τον τρόπο πληροφορία που αναπαριστανόταν με πολλαπλά αντικείμενα (συνήθως 8 με 10, εξαρτάτε πόσα Wi-Fi σημεία υπήρχαν στο σημείο που έπαιρνε μετρήσεις ο χρήστης) μειώθηκε σε ένα αντικείμενο που περιέχει έναν πίνακα measurements. Στις δύο πιο κάτω εικόνες μπορούμε να δούμε την παλιά και νέα δομή των αντικειμένων.

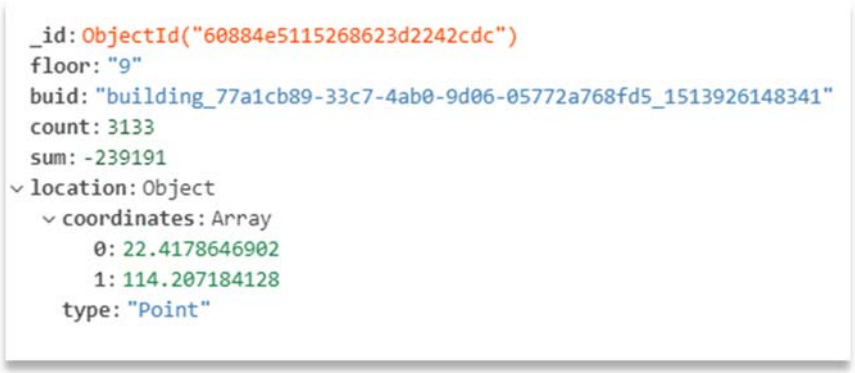
```
{"buid": "building_fdf82513-296d-4b20-b272-77f7cae951a9_1571293503151", "heading": "278.92798", "x":
"12.84841993070293", "y": "77.66729246824978", "geometry": {"coordinates": [12.84841993070293, 77.66729246824978],
"type": "Point"}, "floor": "0", "strongestWifi": "b8:27:eb:91:c6:35", "timestamp": "1571305932708",
"MAC": "58:f3:9c:aa:66:d9", "rss": "-81"}

{"buid": "building_fdf82513-296d-4b20-b272-77f7cae951a9_1571293503151", "heading": "278.92798", "x":
"12.84841993070293", "y": "77.66729246824978", "geometry": {"coordinates": [12.84841993070293, 77.66729246824978],
"type": "Point"}, "floor": "0", "strongestWifi": "b8:27:eb:91:c6:35", "timestamp": "1571305932708",
"MAC": "58:f3:9c:aa:66:df", "rss": "-81"}

{"buid": "building_fdf82513-296d-4b20-b272-77f7cae951a9_1571293503151", "heading": "278.92798", "x":
"12.84841993070293", "y": "77.66729246824978", "geometry": {"coordinates": [12.84841993070293, 77.66729246824978],
"type": "Point"}, "floor": "0", "strongestWifi": "b8:27:eb:91:c6:35", "timestamp": "1571305932708",
"MAC": "04:79:70:94:0a:f0", "rss": "-80"}

{"buid": "building_fdf82513-296d-4b20-b272-77f7cae951a9_1571293503151", "heading": "278.92798", "x":
"12.84841993070293", "y": "77.66729246824978", "geometry": {"coordinates": [12.84841993070293, 77.66729246824978],
"type": "Point"}, "floor": "0", "strongestWifi": "b8:27:eb:91:c6:35", "timestamp": "1571305932708",
"MAC": "58:f3:9c:aa:66:d8", "rss": "-82"}

{"buid": "building_fdf82513-296d-4b20-b272-77f7cae951a9_1571293503151", "heading": "278.92798", "x":
"12.84841993070293", "y": "77.66729246824978", "geometry": {"coordinates": [12.84841993070293, 77.66729246824978],
"type": "Point"}, "floor": "0", "strongestWifi": "b8:27:eb:91:c6:35", "timestamp": "1571305932708",
"MAC": "58:f3:9c:aa:66:de", "rss": "-81"}

{"buid": "building_fdf82513-296d-4b20-b272-77f7cae951a9_1571293503151", "heading": "278.92798", "x":
"12.84841993070293", "y": "77.66729246824978", "geometry": {"coordinates": [12.84841993070293, 77.66729246824978],
"type": "Point"}, "floor": "0", "strongestWifi": "b8:27:eb:91:c6:35", "timestamp": "1571305932708",
"MAC": "a4:f0:5e:25:4d:3d", "rss": "-86"}

{"buid": "building_fdf82513-296d-4b20-b272-77f7cae951a9_1571293503151", "heading": "278.92798", "x":
"12.84841993070293", "y": "77.66729246824978", "geometry": {"coordinates": [12.84841993070293, 77.66729246824978],
"type": "Point"}, "floor": "0", "strongestWifi": "b8:27:eb:91:c6:35", "timestamp": "1571305932708",
"MAC": "b8:27:eb:91:c6:35", "rss": "-74"}

{"buid": "building_fdf82513-296d-4b20-b272-77f7cae951a9_1571293503151", "heading": "278.92798", "x":
"12.84841993070293", "y": "77.66729246824978", "geometry": {"coordinates": [12.84841993070293, 77.66729246824978],
"type": "Point"}, "floor": "0", "strongestWifi": "b8:27:eb:91:c6:35", "timestamp": "1571305932708",
"MAC": "58:f3:9c:aa:66:dc", "rss": "-81"}

{"buid": "building_fdf82513-296d-4b20-b272-77f7cae951a9_1571293503151", "heading": "278.92798", "x":
"12.84841993070293", "y": "77.66729246824978", "geometry": {"coordinates": [12.84841993070293, 77.66729246824978],
"type": "Point"}, "floor": "0", "strongestWifi": "b8:27:eb:91:c6:35", "timestamp": "1571305932708",
"MAC": "b8:c7:4a:c4:8f:c3", "rss": "-79"}
```


Εικόνα 4.3: JSON αντικείμενα αποτυπωμάτων Wi-Fi αποθηκευμένα στην CouchbaseDB


```
{{
  "_id": {
  "$oid": "604937c3538a9e844a195e2a"
},
  "buid": "building_fdf82513-296d-4b20-b272-77f7cae951a9_1571293503151",
  "heading": "278.92798",
  "x": "12.84841993070293",
  "y": "77.66729246824978",
  "geometry": {
  "coordinates": [12.84841993070293, 77.66729246824978],
  "type": "Point"
},
  "floor": "0",
  "strongestWifi": "b8:27:eb:91:c6:35",
  "timestamp": "1571305932708",
  "_schema": 0,
  "measurements": [
  ["a4:f0:5e:25:4d:3d", "-86"],
  ["04:79:70:94:0a:f0", "-80"],
  ["58:f3:9c:aa:66:dc", "-81"],
  ["58:f3:9c:aa:66:d9", "-81"],
  ["58:f3:9c:aa:66:d8", "-82"],
  ["58:f3:9c:aa:66:de", "-81"],
  ["b8:27:eb:91:c6:35", "-74"],
  ["b8:c7:4a:c4:8f:c3", "-79"],
  ["58:f3:9c:aa:66:df", "-81"]
  ]
}
```


Εικόνα 4.4: Νέα δομή JSON αντικειμένου για αποτυπώματα Wi-Fi στην MongoDB

Επίσης, με το τέλος της δημιουργίας του collection fingerprintsWifi δημιουργούνται και κάποια άλλα collections που βασίζονται σε αυτό. Ο λόγος δημιουργίας τους είναι για την ταχύτερη διεκπεραίωση κάποιων endpoints τα οποία ασχολούνται με τα αποτυπώματα WiFi για την δημιουργία των heatmaps. Τα collections αυτά ονομάζονται accessPointWifi, heatmapWifi1, heatmapWifi2, heatmapWifi3, heatmapWifiTimestamp1, heatmapWifiTimestamp2, heatmapWifiTimestamp3. Βασίζονται στην φιλοσοφία "never recompute when you can precompute" που αναφέραμε στα υπολογιζόμενα μοτίβα όπου λειτουργούν σαν cache. Δηλαδή, όταν ένας χρήστης ζητήσει τα δεδομένα αυτά, την πρώτη φορά, δημιουργούνται και αποθηκεύονται στο ανάλογο collection ενώ για κάθε επόμενη φορά που τα ζητά επιστρέφονται από την βάση. Εάν ο χρήστης κάνει αλλαγές στα αποτυπώματα τότε αυτόματα τα αντικείμενα που δημιουργήθηκαν στα άλλα collections με βάση τα αποτυπώματα αυτά θα διαγράφονται, έτσι εξοικονομείται αρκετός αποθηκευτικός χώρος.



Εικόνα 4.5: JSON αντικείμενο του Collection heatmapWifi1

4.3.6 Επικοινωνία βάσης δεδομένων – κώδικα Anyplace

Καθώς ένας χρήστης αλληλοεπιδρά με το σύστημα Anyplace από τον Architect, Viewer, Viewer Campus, Navigator ή Logger εκτελούνται διάφορα endpoints (π.χ. GET και POST requests). Τα endpoints αυτά, πολλές φορές χρησιμοποιούσαν δεδομένα από την βάση δεδομένων CouchbaseDB, άρα έπρεπε να αλλαχτούν και να βασίζονται στην καινούρια βάση, MongoDB.

Συνολικά υπήρχαν 108 endpoints εκ των οποίων τα 71 αλληλοεπιδρούσαν με την βάση δεδομένων. Από τα 71 τα 21 δεν χρησιμοποιούνταν ενώ 7 δεν λειτουργούσαν επειδή δεν ήταν υλοποιημένα ή ήταν απενεργοποιημένα ή είχαν κάποιο bug. Τελικά, 59 από τα 71 endpoints (όλα όσα χρησιμοποιούνταν και λειτουργούσαν δηλ. 44 από τα 44, κάποια που

δεν χρησιμοποιούνταν δηλ. 8 από τα 21 και όλα που δεν λειτουργούσαν δηλ. 7 από τα 7) υλοποιήθηκαν σε MongoDB και ελέγχθηκαν ότι δουλεύουν όπως αναμένεται.

Ακολουθεί ένας κατάλογος με τα 71 endpoints. Με κίτρινο φόντο αυτά που δεν λειτουργούσαν και υλοποιήθηκαν, με πράσινο φόντο αυτά που δεν χρησιμοποιούνταν και υλοποιήθηκαν ενώ με κόκκινο φόντο αυτά που δεν χρησιμοποιούνταν και δεν υλοποιήθηκαν.

1	/anyplace/mapping/accounts/sign
2	/anyplace/mapping/building/add
3	/anyplace/mapping/building/update
4	/anyplace/mapping/building/delete
5	/anyplace/mapping/building/all
6	/anyplace/mapping/building/all_owner
7	/anyplace/mapping/building/coordinates
8	/anyplace/mapping/building/get
9	/anyplace/mapping/pois/add
10	/anyplace/mapping/pois/delete
11	/anyplace/mapping/pois/update
12	/anyplace/mapping/pois/all_floor
13	/anyplace/mapping/pois/all_building
14	/anyplace/mapping/pois/all_pois
15	/anyplace/mapping/connection/add
16	/anyplace/mapping/connection/delete
17	/anyplace/mapping/connection/all_floor
18	/anyplace/mapping/connection/all_floors
19	/anyplace/mapping/floor/add
20	/anyplace/mapping/floor/delete
21	/anyplace/mapping/floor/uploadWithZoom
22	/anyplace/mapping/floor/all
23	/anyplace/mapping/campus/add
24	/anyplace/mapping/campus/update
25	/anyplace/mapping/campus/delete
26	/anyplace/mapping/campus/all_cucode
27	/anyplace/mapping/campus/all_owner
28	/anyplace/position/radio_upload

29	/anyplace/position/radio/delete
30	/anyplace/position/radio/delete/time
31	/anyplace/position/radio_download_floor
32	/anyplace/position/radio_by_floor_bbox
33	/anyplace/position/radio/time
34	/anyplace/position/radio_by_building_floor
35	/anyplace/position/radio_by_building_floor_all
36	/anyplace/position/radio/APs_building_floor
37	/anyplace/mapping/radio/heatmap_building_floor
38	/anyplace/position/radio/heatmap_building_floor_average_1
39	/anyplace/position/radio/heatmap_building_floor_average_2
40	/anyplace/position/radio/heatmap_building_floor_average_3
41	/anyplace/position/radio/heatmap_building_floor_average_3_tiles
42	/anyplace/position/radio/heatmap_building_floor_timestamp
43	/anyplace/position/radio/heatmap_building_floor_timestamp_average_1
44	/anyplace/position/radio/heatmap_building_floor_timestamp_average_2
45	/anyplace/position/radio/heatmap_building_floor_timestamp_average_3
46	/anyplace/position/radio/heatmap_building_floor_timestamp_tiles
47	/anyplace/navigation/building/id
48	/anyplace/navigation/pois/id
49	/anyplace/navigation/route
50	/anyplace/navigation/route_xy
51	/anyplace/debug/accounts_all
52	/anyplace/mapping/building/coowners
53	/anyplace/mapping/building/all_bucode
54	/anyplace/mapping/building/newowner
55	/anyplace/mapping/pois/all_pois_nconnectors
56	/anyplace/mapping/floor/upload
57	/anyplace/mapping/connection/update
58	/anyplace/mapping/floor/update
59	/anyplace/position/path_add
60	/anyplace/position/path_delete
61	/anyplace/position/paths_by_floor
62	/anyplace/position/paths_by_buid
63	/anyplace/position/milestones_add
64	/anyplace/position/milestones_by_floor
65	/anyplace/position/estimate_position
66	/anyplace/mapping/radio/radio_buid_floor
67	/anyplace/mapping/maintenance

68	/accounts/oauth2/token
69	/login
70	/logout
71	/authenticate

Πίνακας 4.1: Endpoints που αλληλοεπιδρούν με την βάση δεδομένων

Εάν ο χρήστης εκτελέσει ένα endpoint τότε ξεκινά μία διαδικασία μέχρι να γίνει η εκτέλεση του αιτήματος που έκανε. Για παράδειγμα αν ο χρήστης συνδεθεί στον λογαριασμό του στον Anyplace Architect αυτόματα καλείται το endpoint /anyplace/mapping/building/all_owner για να του εμφανίσει όλα τα κτήρια που του ανήκουν. Η διαδικασία που ακολουθείτε για το συγκεκριμένο endpoint είναι η εξής:

- το endpoint καλεί την μέθοδο controllers.AnyplaceMapping.buildingAllByOwner() (μπορούμε να δούμε ότι βρίσκεται στον controller mapping αφού το endpoint ανήκει στην κατηγορία /anyplace/mapping)
- στην συνέχεια ελέγχονται τα πεδία του JSON που στέλνει το request (εάν υπάρχουν, στο παράδειγμα αυτό γίνεται ταυτοποίηση με το κλειδί API) και μετά καλείται η μέθοδος ProxyDataSource.getIDatasource.getAllBuildings(). Η μέθοδος αυτή κληρονομείτε από την κλάση IDatasource.Scala στην κλάση ProxyDataSource.Scala και στην συνέχεια κληρονομείται στην κλάση MongodbDatasource.Scala. Πριν την μετάβαση σε MongoDB την ProxyDataSource κληρονομούσε η CouchbaseDB όπου υπήρχε και το ανάλογο view.
- Αφού φτάσαμε στην κλάση MongodbDatasource εκτελούμε το ανάλογο query το οποίο επιστρέφει όλα τα κτήρια που ανήκουν στον χρήστη.

```
override def getAllBuildingsByOwner(oid: String): List[JsValue] = {
  val collection = mdb.getCollection("buildings")
  var buildingLookUp = collection.find(or(equal("owner_id", oid),
    equal("co_owners", oid)))
  if (admins.contains(oid)) {
    buildingLookUp = collection.find()
  }
  val awaited = Await.result(buildingLookUp.toFuture, Duration.Inf)
  val res = awaited.toList
  val listJson = convertJson(res)
  val buildings = new java.util.ArrayList[JsValue]()
  for (building <- listJson) {
      buildings.add(building.as[JsObject] - "co_owners" - __geometry
        - "owner_id" - "_id" - "_schema")
  }
  buildings.toList
}
```

Παράδειγμα 1: endpoint /anyplace/mapping/building/all_owner

```
override def replaceJsonDocument(col: String, key: String, value: String,
                                 document: String): Boolean = {
  val collection = mdb.getCollection(col)
  val query = BsonDocument(key -> value)
  val update = BsonDocument(document)
  val replaceJson = collection.replaceOne(query, update)
  val awaited = Await.result(replaceJson.toFuture, Duration.Inf)
  val res = awaited
  if (res.getModifiedCount == 0)
    false
  else
    true
}
```

Παράδειγμα 2: query που αντικαθιστά ένα JSON. Χρησιμοποιείται από τα endpoints που κάνουν update (π.χ. /anyplace/mapping/pois/update)

```
override def getFingerPrintsBBox(buid: String, floor: String, lat1: String,
                                 lon1: String, lat2: String,
                                 lon2: String): List[JsValue] = {
  val collection = mdb.getCollection("fingerprintsWifi")
  val fingerprints = collection.find(and(
    geoWithinBox("geometry", lat1.toDouble, lon1.toDouble,
      lat2.toDouble, lon2.toDouble),
    equal("buid", buid),
    equal("floor", floor)))
  val awaited = Await.result(fingerprints.toFuture, Duration.Inf)
  val res = awaited.toList
  val listJson = convertJson(res)
  val newList = new util.ArrayList[JsValue]()
  for (f <- listJson) {
    if ((f \ "buid").as[String] == buid &&
      (f \ "floor").as[String] == floor) {
      newList.add(f)
    }
  }
  newList.toList
}
```

Παράδειγμα 3: spatial query που βρίσκει αποτυπώματα Wi-Fi σε ένα bounding box

Κεφάλαιο 5

Αλλαγές στην Διεπαφή Προγραμματισμού Εφαρμογών (API)

5.1 Εισαγωγή 44

5.2 Πρόβλημα 44

5.3 Υλοποίηση 45

5.1 Εισαγωγή

Μια διεπαφή προγραμματισμού εφαρμογών, γνωστή και ως API[18], είναι μια διεπαφή κάποιου λογισμικού έτσι ώστε να επιτρέπει να γίνονται διάφορες αιτήσεις από άλλα προγράμματα σε αυτό. Καθορίζει το είδος του αιτήματος, τον τρόπο με τον οποίο γίνεται, την μορφή των δεδομένων που δέχεται και διάφορες συμβάσεις που πρέπει να τηρούνται. Ένα μεγάλο προτέρημα των διεπαφών αυτών είναι ότι παρέχονται σε ένα χρήστη πλήρες λειτουργίες μιας εφαρμογής χωρίς να του δοθεί ο πηγαίος κώδικας.

Κατά την αλλαγή του υποσυστήματος δεδομένων έγιναν διάφορες αλλαγές στο API του Anyplace [19] όπου κυρίως αφορούν ελέγχους των δεδομένων που παίρνουν τα αιτήματα και διορθώσεις σε αιτήματα που λειτουργούσαν λανθασμένα σε κάποια σενάρια με προκαθορισμένη είσοδο.

5.2 Πρόβλημα

Μπορούμε να χωρίσουμε τα προβλήματα που είχε το API σε τρεις κατηγορίες. Οι κατηγορίες αυτές αφορούν ελέγχους των τύπων δεδομένων, ελέγχους ακεραιότητας και τέλος ελλιπές ή λανθασμένες υλοποιήσεις.

Οι έλεγχοι των τύπων δεδομένων δεν αποτελούσαν μεγάλο πρόβλημα στο σύστημα αλλά εάν κάποιος χρήστης έστελνε κάποιο αίτημα με λάθος τύπο δεδομένων τότε έπαιρνε απάντηση «500 Internal Server Error». Η απάντηση αυτή είναι παραπλανητική και δεν αντιπροσωπεύει την πραγματική αιτία του σφάλματος. Η σωστή προσέγγιση για την ένδειξη λάθους θα ήταν «400 Bad Request» και το ανάλογο μήνυμα λάθους π.χ. «coordinate_lat must be Float»

Οι έλεγχοι ακεραιότητας είναι πάρα πολύ σημαντικοί αφού μπορεί να οδηγήσουν σε λάθος αποτελέσματα ή δυσλειτουργίες του συστήματος. Για παράδειγμα εάν ο χρήστης στείλει ένα αίτημα όπου προσθέτει ένα καινούριο όροφο σε ένα κτήριο αλλά το κτήριο δεν υπάρχει και το σύστημα το επιτρέψει τότε έχουμε ένα όροφο που δεν ανήκει πουθενά. Σε μεταγενέστερο στάδιο μπορεί κάποιος να προσθέσει ένα σημείο ενδιαφέροντος στον όροφο αυτό και να ζητά πλοήγηση σε αυτό ενώ ο όροφος αυτός να μην υπάρχει στην πραγματικότητα.

Οι λανθασμένες υλοποιήσεις ίσως αποτελούν στο σημαντικότερο πρόβλημα αφού τις πλείστες φορές δίνουν λάθος αποτελέσματα τα οποία μπορούν να προκαλέσουν πολλά προβλήματα σε αυτόν που εκτελεί το αίτημα. Τα προβλήματα αυτά συνδέονται άμεσα με τους ελέγχους ακεραιότητας.

5.3 Υλοποίηση

Στον *Πίνακα 5.1* μπορούμε να δούμε όλα τα αιτήματα – endpoints που έτυχαν αλλαγής αφού εντοπίστηκε κάποιο πρόβλημα. Οι στήλες «buid», «puid», «cuid», «floor», «coordinates» και «timestamp» αναπαριστούν τιμές που χρειάζονται κάποια από τα αιτήματα για να εκτελεστούν. Σε κάθε κουτί που υπάρχει το σύμβολο «✓» σημαίνει ότι για το συγκεκριμένο αίτημα, το συγκεκριμένο πεδίο πλέον ελέγχεται εάν έχει τον σωστό τύπο δεδομένων. Κάποια αιτήματα έχουν κόκκινο φόντο, αυτό υποδεικνύει ότι υπήρχαν και άλλα προβλήματα πέραν από τους τύπους δεδομένων τα οποία αναλύονται στην συνέχεια.

Endpoint	buid	puid	cuid	floor	coordinates	timestamp
/mapping/building/update	✓					
/mapping/building/delete	✓					
/mapping/building/coowners	✓					
/mapping/building/newowner	✓					
/mapping/building/coordinates						
/mapping/floor/delete	✓			✓		
/mapping/floor/add	✓			✓		
/mapping/floor/uploadWithZoom	✓					
/mapping/floor/all	✓					
/mapping/pois/all_pois_nconnectors	✓					
/mapping/pois/delete	✓	✓				
/mapping/pois/update	✓	✓				
/mapping/pois/all_floor	✓			✓		
/mapping/pois/all_building	✓					
/mapping/pois/add	✓					
/mapping/connection/add	✓	✓		✓		
/mapping/connection/delete	✓	✓				
/mapping/connection/all_floor	✓			✓		
/mapping/connection/all_floors	✓					
/mapping/campus/update			✓			
/mapping/campus/delete			✓			
/mapping/campus/all_cucode						
/navigation/building/id	✓					
/navigation/pois/id		✓				
/navigation/route_xy		✓		✓	✓	
/navigation/route		✓				
/position/radio/delete	✓			✓	✓	
/position/radio/delete/time	✓			✓	✓	✓
/position/radio_upload						
/position/radio/time	✓			✓		
/position/radio_download_floor				✓	✓	
/position/radio_by_floor_bbox				✓	✓	
/position/radio_by_building_floor	✓			✓		
/position/radio_by_building_floor_all	✓			✓		
/position/radio/APs_building_floor	✓			✓		
/mapping/radio/heatmap_building_floor	✓			✓		
/position/radio/heatmap_building_floor_average_1	✓			✓		
/position/radio/heatmap_building_floor_average_2	✓			✓		
/position/radio/heatmap_building_floor_average_3	✓			✓		
/position/radio/heatmap_building_floor_average_3_tiles	✓			✓		
/position/radio/heatmap_building_floor_timestamp	✓			✓		✓
/position/radio/heatmap_building_floor_timestamp_average_3	✓			✓		✓
/position/radio/heatmap_building_floor_timestamp_average_1	✓			✓		✓
/position/radio/heatmap_building_floor_timestamp_average_2	✓			✓		✓
/position/radio/heatmap_building_floor_timestamp_tiles	✓			✓		✓

Πίνακας 5.1: Endpoints που αλλάχτηκαν

Το endpoint /mapping/pois/all_floor ψάχνει και επιστρέφει όλα τα σημεία ενδιαφέροντος κάποιου ορόφου σε ένα κτήριο. Προστέθηκε έλεγχος που διαβεβαιώνει ότι το κτήριο στο οποίο ψάχνουμε υπάρχει. Παρόμοιος έλεγχος γίνεται και στα endpoints /mapping/pois/all_building, /mapping/connection/all_floor.

Το endpoint /mapping/connection/add προσθέτει μία σύνδεση μεταξύ δύο σημείων ενδιαφέροντος. Δέχεται δύο κτήρια, δύο ορόφους, δύο σημεία ενδιαφέροντος, κατηγορία, ένα API κλειδί και κατά πόσο θα είναι δημοσιευμένο σε όλους. Με βάση αυτά το σύστημα ενώνει τα δύο σημεία ενδιαφέροντος. Όταν ένας απλός χρήστης προσθέσει μία ένωση τότε το σύστημα λειτουργεί όπως αναμένεται αφού εκτελεί το αίτημα και τα πεδία συμπληρώνονται αυτόματα από την βάση. Εάν όμως κάποιος προχωρημένος χρήστης χρησιμοποιήσει το endpoint με POST αίτημα μπορεί να προσθέσει μία ένωση σε ένα κτήριο που δεν υπάρχει. Επομένως πρόσθεσα ελέγχους όπου διασφαλίζουν ότι τα κτήρια, οι όροφοι και τα σημεία ενδιαφέροντος υπάρχουν.

Το endpoint /mapping/campus/all_cucode ψάχνει και επιστρέφει ένα campus. Εάν δεν έβρισκε το campus τότε ενημέρωνε τον χρήστη με το μήνυμα «Something went wrong while fetching buildings.», ενώ τώρα το μήνυμα είναι «Campus 'campus name' not found!». Επίσης εάν βρεθούν δύο campuses με το ίδιο όνομα τότε ο χρήστης ενημερώνεται με το ανάλογο μήνυμα.

Όλα τα endpoints της κατηγορίας navigation (/navigation/building/id, /navigation/pois/id, /navigation/route, /navigation/route_xy) δεν επέστρεφαν μηνύματα λάθους και σε περίπτωση που τύγχανε κάποιο το σύστημα απαντούσε πάντα με το μήνυμα «500 Internal Server Error». Για παράδειγμα στο αίτημα route εάν έλειπαν παράμετροι από το αίτημα, εάν δεν έβρισκε κάποια διαδρομή, εάν τα σημεία ενδιαφέροντος που υπήρχαν στο αίτημα δεν υπήρχαν στην βάση δεδομένων ή τύγχανε οτιδήποτε άλλο ο χρήστης δεν έπαιρνε το ανάλογο μήνυμα λάθους.

Επίσης στα /navigation/route και /navigation/route_xy έλειπαν σημαντικοί έλεγχοι. Το αίτημα route δέχεται δύο σημεία ενδιαφέροντος του ίδιου κτηρίου και επιστρέφει την πιο σύντομη διαδρομή. Σε περίπτωση που τα σημεία ενδιαφέροντος δεν υπήρχαν το σύστημα δεν διαχειριζόταν το πρόβλημα ενώ τώρα ενημερώνει τον χρήστη με το ανάλογο μήνυμα. Το αίτημα route_xy δέχεται ένα σημείο ενδιαφέροντος και μια τοποθεσία σε μορφή συντεταγμένων, όπου αρχικά ψάχνει το κοντινότερο σημείο ενδιαφέροντος στις συντεταγμένες αυτές και στην συνέχεια παρουσιάζει πλοήγηση στο σημείο του

αιτήματος. Σημαντικό να σημειώσουμε ότι δεν υπήρχε κάποιος περιορισμός στην απόσταση του κοντινότερου σημείου ενδιαφέροντος. Σε συγκεκριμένες δοκιμές το σύστημα έδωσε απάντηση παρόλο που το κοντινότερο σημείο ήταν 5000 χιλιόμετρα μακριά. Πλέον υπάρχει περιορισμός στην απόσταση του κοντινότερου σημείο ο οποίος είναι τα 500 μέτρα.

Το endpoint /position/radio_upload χρησιμοποιείται από τον Anyplace Logger και προσθέτει αποτυπώματα Wi-Fi στον server και στην βάση δεδομένων, τα οποία δέχεται από το αίτημα σε μορφή αρχείου μαζί με το κλειδί API. Για κάποιο λόγο, πιθανόν λανθασμένης υλοποίησης επαλήθευσης του API κλειδιού, ο κώδικας android δημιουργούσε και εμφύτευε ένα άδειο JSON στο αίτημα πριν το αποστείλει. Στην συνέχεια το σύστημα έψαχνε το API κλειδί στο αντικείμενο αυτό αλλά δεν το έβρισκε αφού ήταν άδειο. Στα πειράματα που έγιναν για να εκτελεστεί σωστά το αίτημα έπρεπε να περιέχει το αρχείο με τα αποτυπώματα Wi-Fi, το API κλειδί και ένα άδειο JSON. Έτσι, το αίτημα αλλάχτηκε και πλέον δέχεται μόνο ένα αρχείο και ένα API κλειδί.

Τα endpoints /position/radio_download_floor και /position/radio_by_floor_bbox είναι παρόμοια αφού και τα δύο επιστρέφουν ένα σύνδεσμο ο οποίος παραπέμπει σε ένα αρχείο με μη επεξεργασμένα αποτυπώματα Wi-Fi. Στην προηγούμενη έκδοση τα δύο αιτήματα δέχονταν συντεταγμένες και όροφο όπου σε συνδυασμό με μία εμβέλεια, έκτιζαν ένα bounding box και αποθηκεύαν τα αποτυπώματα που υπήρχαν μέσα σε αυτό σε ένα αρχείο. Η εμβέλεια για το πρώτο αίτημα ήταν πάντα 500 μέτρα ενώ για το δεύτερο δεν υπήρχε περιορισμός αφού ήταν παράμετρος. Το αρχείο που περιείχε τις τιμές αποθηκευόταν στον server και το όνομα που έπαιρνε ήταν τυχαιοποιημένο. Αυτή η τεχνική ήταν πολύ λανθασμένη αφού εάν εκτελούσα 10 φορές το ίδιο αίτημα θα δημιουργούνταν 10 ίδια αρχεία με διαφορετικό όνομα. Στην νέα έκδοση, αρχικά προστέθηκε περιορισμός για την εμβέλεια όπου το μέγιστο είναι τα 500 μέτρα. Στην συνέχεια, εάν γινόταν το πρώτο αίτημα ή το δεύτερο με εμβέλεια 500 τότε αντί να ξαναδημιουργείται το αρχείο επιστρέφεται απευθείας από τον server (τα αρχεία για ολόκληρο τον όροφο υπάρχουν αφού όταν ένας χρήστης εισάγει αποτυπώματα αποθηκεύονται ακατέργαστα στον server). Εάν το αίτημα έχει εμβέλεια μικρότερη των 500 μέτρων τότε ακολουθείται η ίδια διαδικασία με την προηγούμενη έκδοση αλλά το αρχείο ονομάζεται διαφορετικά. Το όνομα δημιουργείται από έναν τυχαίο αριθμό που δημιουργείται με βάση ένα κλειδί (συντεταγμένες + όροφος) και μετά επικολλάτε η

εμβέλεια, έτσι σε περίπτωση που εκτελεστεί το ίδιο αίτημα δύο φορές ο τυχαίος αριθμός θα είναι ο ίδιος και κατ' επέκταση το όνομα του αρχείο. Επομένως, δεν αποθηκεύεται άσκοπα η ίδια πληροφορία πολλές φορές και σε περίπτωση ίδιου αιτήματος το σύστημα απαντά πολύ πιο γρήγορα αφού η απάντηση υπάρχει ήδη.

Το endpoint /mapping/building/coordinates δεχόταν ως είσοδο συντεταγμένες και ένα κλειδί API. Με βάση τις συντεταγμένες αυτές επέστρεφε όλα τα κτήρια που ανήκουν στον χρήστη σε ακτίνα 50 μέτρων. Έγιναν κάποιες μετατροπές και πλέον το endpoint επιστρέφει όλα τα published κτήρια που βρίσκονται γύρω από τον χρήστη και όλα τα κτήρια που του ανήκουν ή είναι συνιδιοκτήτης (ακόμη και αν δεν είναι published). Επίσης προστέθηκε μία προαιρετική τιμή range όπου καθορίζει την εμβέλεια που ψάχνει το σύστημα. Η μέγιστη τιμή που μπορεί να πάρει είναι τα 500 μέτρα(αν δοθεί μεγαλύτερη τιμή αυτόματα μειώνεται στα 500 μέτρα) και εάν δεν δοθεί κάποια τότε αυτόματα το σύστημα ψάχνει για 50 μέτρα.

Τα endpoints /position/radio/heatmap_building_floor_average_ 1, 2, 3 και 3_tiles επέστρεφαν ένα πίνακα με αντικείμενα radiopoints. Η δομή των αντικειμένων αποτελούταν από συντεταγμένες δύο μεταβλητών x, y και μιας ακόμη μεταβλητής w η οποία περιείχε ένα JSON αντικείμενο σαν συμβολοσειρά με "count", "average", "total" ("{\"count\":19,\"average\":-90.63157894736842,\"total\":-1722}"). Πλέον τα αντικείμενα αυτά έχουν διαφορετική μορφή, η οποία είναι ένα GeoJSON αντικείμενο "location" το οποίο περιέχει την τοποθεσία και τρεις μεταβλητές "count", "sum", "average".

Κεφάλαιο 6

Πειραματική αποτίμηση

6.1 Εισαγωγή 50

6.2 Αποτίμηση χρόνου 51

6.3 Μέγεθος δεδομένων 58

6.3.1 Αντικείμενα 58

6.3.2 Μνήμη 60

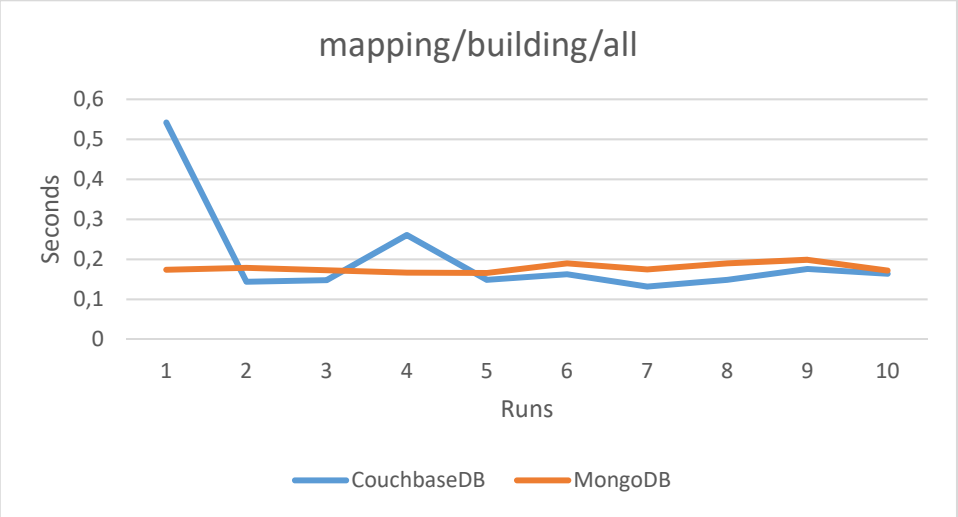
6.1 Εισαγωγή

Με το τέλος της υλοποίησης τρέξαμε διάφορα πειράματα για να συγκρίνουμε την νέα εκδοχή του συστήματος με την παλιά. Τα πειράματα αυτά αφορούν τον χρόνο που χρειάζεται το σύστημα για να επεξεργαστεί και να εκτελέσει ένα αίτημα. Επίσης, έγιναν συγκρίσεις και στα δεδομένα, τόσο στον αριθμό αντικειμένων όσο και στον αποθηκευτικό χώρο που χρησιμοποιούν.

Συγκεκριμένα για τους ελέγχους ταχύτητας έγιναν πειράματα για ένα τυχαίο υποσύνολο των endpoints, όπου για κάθε ένα από αυτά παίρναμε 10 χρόνους εκτέλεσης με MongoDB και 10 χρόνους εκτέλεσης με CouchbaseDB. Για να γίνει αυτό αποτελεσματικά γράφτηκαν διάφορα προγράμματα σε κέλυφος, όπου αυτόματα τρέχουν το endpoint 10 φορές και αποθηκεύουν τον χρόνο εκτέλεσης. Παράλληλα, ελέγχουν ότι το αποτελέσματα που επιστρέφουν τα αιτήματα είναι ίδια και στις δύο υλοποιήσεις για σκοπούς επαλήθευσης. Όσο αφορά την σύγκριση των δεδομένων έγινε με την βοήθεια του MongoDB Compass και του couchbase web UI αφού και τα δύο προσφέρουν επαρκή περιγραφή των δεδομένων.

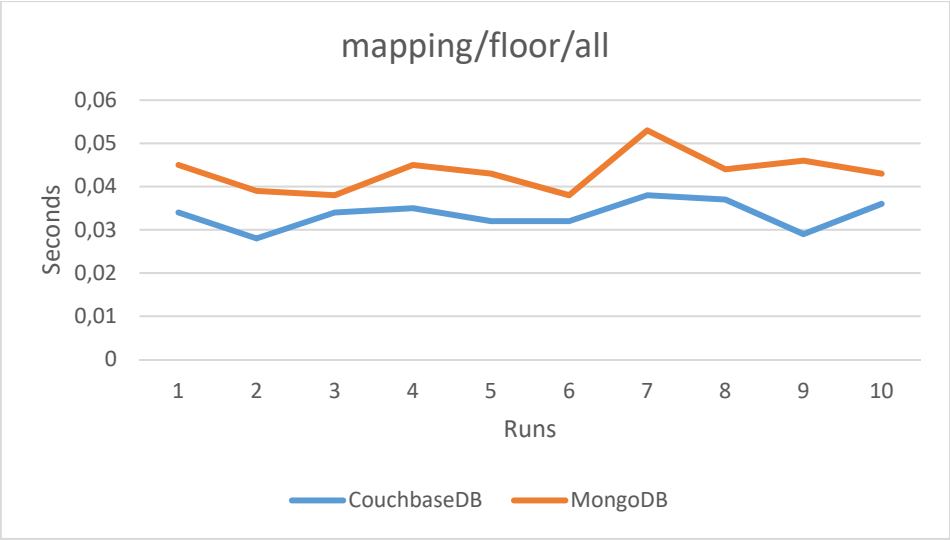
6.2 Αποτίμηση χρόνου

Το endpoint mapping/building/all επιστρέφει όλα τα δημοσιοποιημένα κτήρια του Anyplace. Στην πιο κάτω γραφική παράσταση ο συνολικός αριθμός αντικειμένων που επιστρέφονται είναι 2880. Βλέπουμε ότι ο χρόνος εκτέλεσης με MongoDB είναι σταθερός και κυμαίνεται στα 180ms, ενώ με Couchbase η πρώτη εκτέλεση είναι 540ms και οι επόμενοι χρόνοι κυμαίνονται στα 165ms.



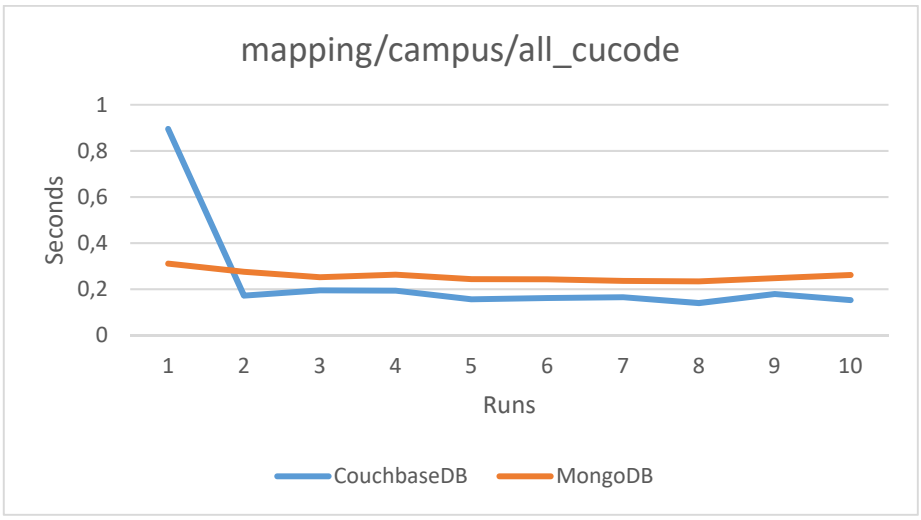
Γραφική παράσταση 6.1: Σύγκριση χρόνου εκτέλεσης του endpoint mapping/building/all μεταξύ MongoDB και CouchbaseDB

Στην πιο κάτω γραφική παράσταση βλέπουμε τον χρόνο εκτέλεσης του endpoint mapping/floor/all, το οποίο επιστρέφει όλους τους ορόφους ενός κτηρίου. Για το συγκεκριμένο παράδειγμα οι όροφοι ήταν 4. Μπορούμε να δούμε ότι η CouchbaseDB είναι πιο γρήγορη αλλά η διαφορά είναι μηδαμινή αφού κατά μέσο όρο διαφέρουν μόνο 10 milliseconds.



Γραφική παράσταση 6.2: Σύγκριση χρόνου εκτέλεσης του endpoint mapping/floor/all μεταξύ MongoDB και CouchbaseDB

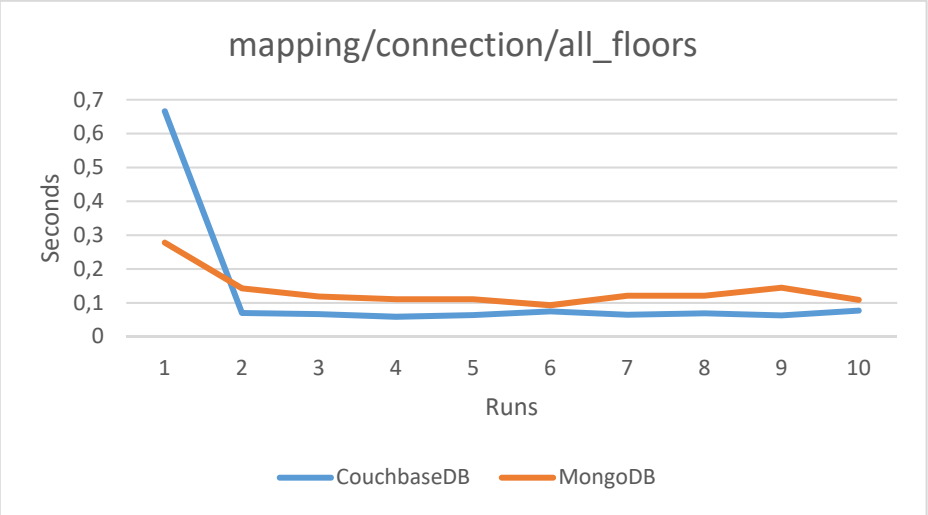
Το endpoint mapping/campus/all_cucode επιστρέφει όλα τα κτήρια που ανήκουν σε κάποιο campus και σε αυτή την γραφική παράσταση βλέπουμε τον χρόνο που χρειάζεται το σύστημα για να βρει τα κτήρια του campus “ucy”, τα οποία είναι 67. Για ακόμη μία φορά μπορούμε να δούμε ότι η πρώτη εκτέλεση της CouchbaseDB είναι αρκετά αργή (περίπου 3 φορές πιο αργό από MongoDB) ενώ η MongoDB διατηρεί μία σταθερότητα.



Γραφική παράσταση 6.3: Σύγκριση χρόνου εκτέλεσης του endpoint mapping/campus/all_cucode μεταξύ MongoDB και CouchbaseDB

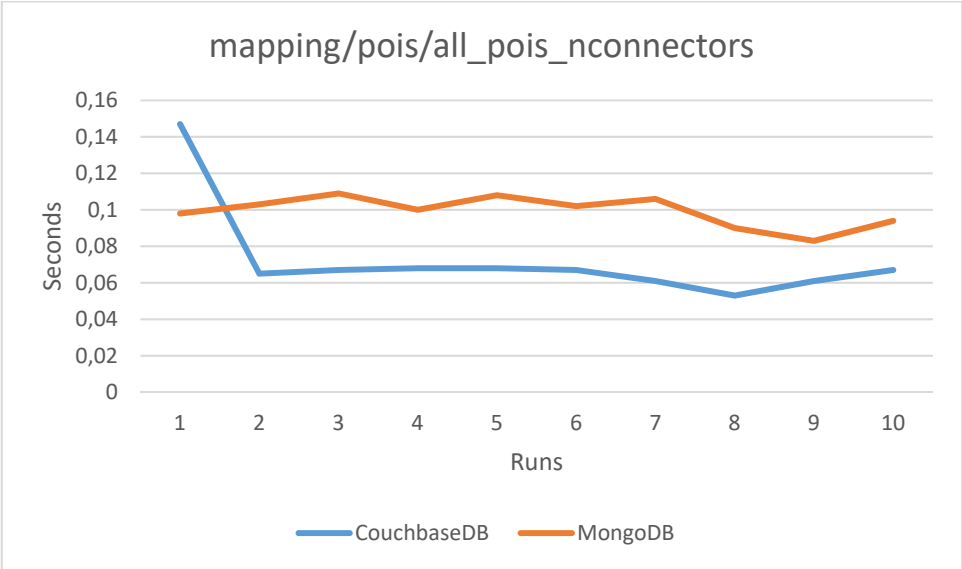
Στην πιο κάτω γραφική παράσταση βλέπουμε τους χρόνους εκτέλεσης του endpoint mapping/connection/all_floors το οποίο βρίσκει και επιστρέφει όλες τις ενώσεις μεταξύ δύο σημείων ενδιαφέροντος σε ένα κτήριο. Το συγκεκριμένο σενάριο, που ψάχνει στο

κτήριο του τμήματος Πληροφορικής έχει συνολικά 443 ενώσεις. Για ακόμη μία φορά έχουμε την ίδια συμπεριφορά όπου η Couchbase μετά την πρώτη εκτέλεση γίνεται πιο γρήγορη.



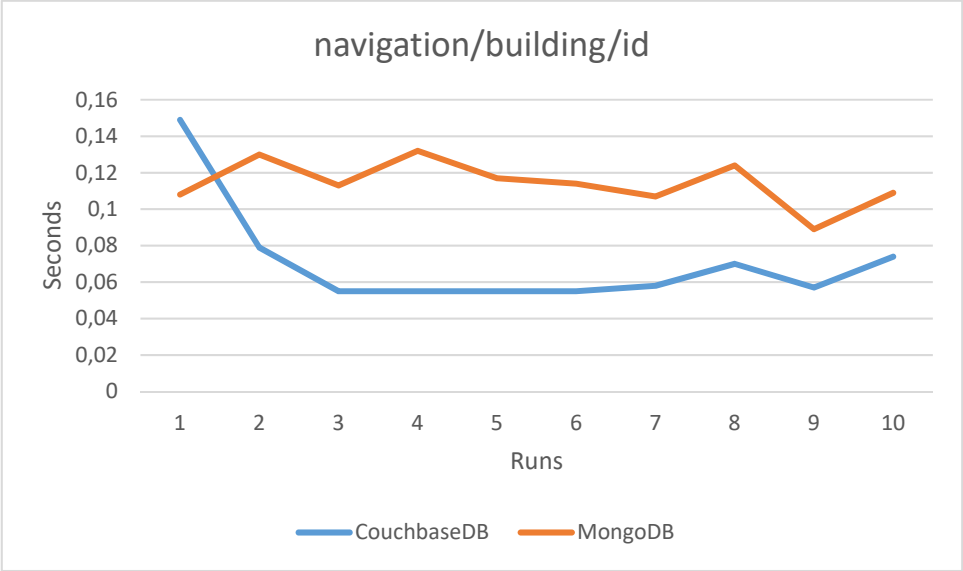
Γραφική παράσταση 6.4: Σύγκριση χρόνου εκτέλεσης του endpoint mapping/connection/all_floors μεταξύ MongoDB και CouchbaseDB

Το endpoint mapping/pois/all_pois_nconnectors είναι παρόμοιο με το προηγούμενο αφού επιστρέφει τα σημεία ενδιαφέροντος ενός κτηρίου. Όπως στα προηγούμενα πειράματα το κτήριο που χρησιμοποιήθηκε ήταν του τμήματος της Πληροφορικής το οποίο έχει 393 σημεία ενδιαφέροντος, και για ακόμη μια φορά η Couchbase έχει καλύτερη επίδοση μετά την πρώτη εκτέλεση.



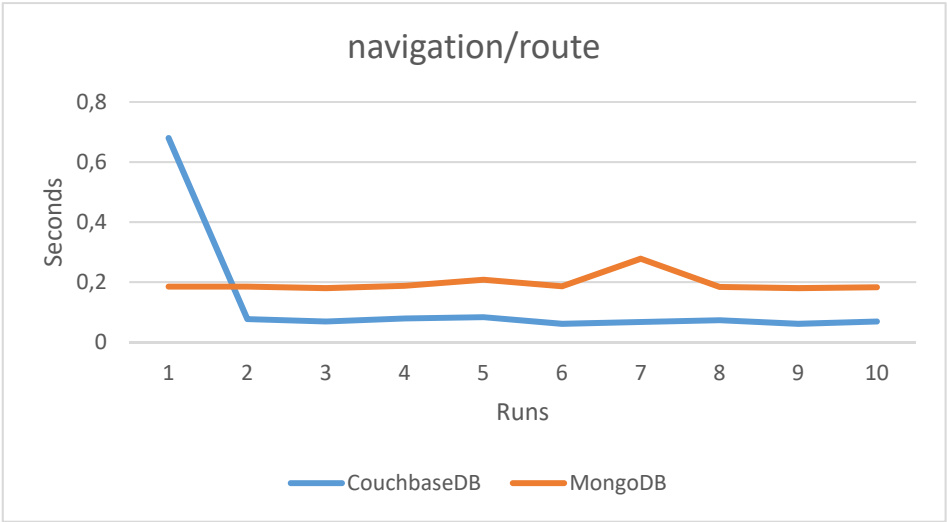
Γραφική παράσταση 6.5: Σύγκριση χρόνου εκτέλεσης του endpoint mapping/pois/all_pois_nconnectors μεταξύ MongoDB και CouchbaseDB

Στην πιο κάτω γραφική παράσταση έχουμε τους χρόνους εκτέλεσης του endpoint navigation/building/id το οποίο κοιτάζει αν υπάρχει ένα κτήριο. Όπως και στα προηγούμενα παραδείγματα η Couchbase είναι πιο γρήγορη μετά την πρώτη εκτέλεση.



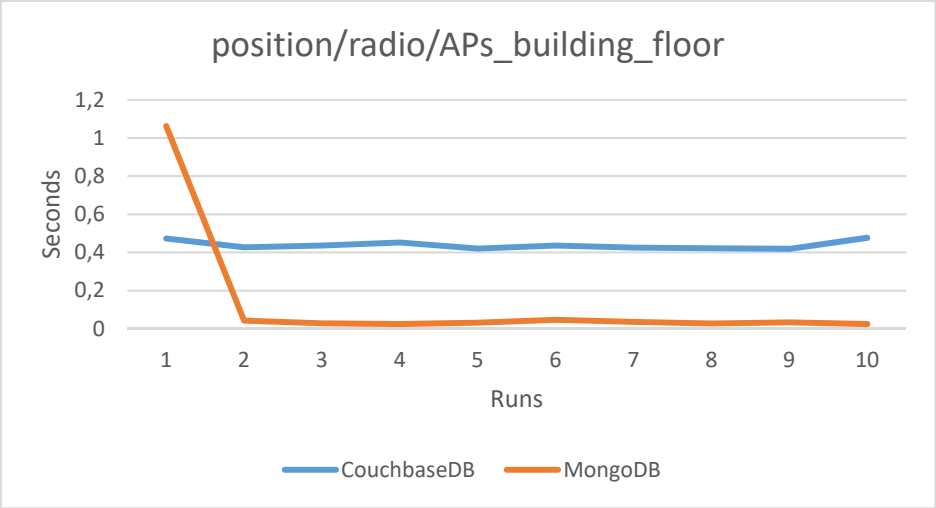
Γραφική παράσταση 6.6: Σύγκριση χρόνου εκτέλεσης του endpoint navigation/building/id μεταξύ MongoDB και CouchbaseDB

Το endpoint navigation/route βρίσκει την γρηγορότερη διαδρομή από ένα σημείο ενδιαφέροντος σε ένα άλλο. Για ακόμη μία φορά έχουμε την ίδια συμπεριφορά στην συμπεριφορά του χρόνου.



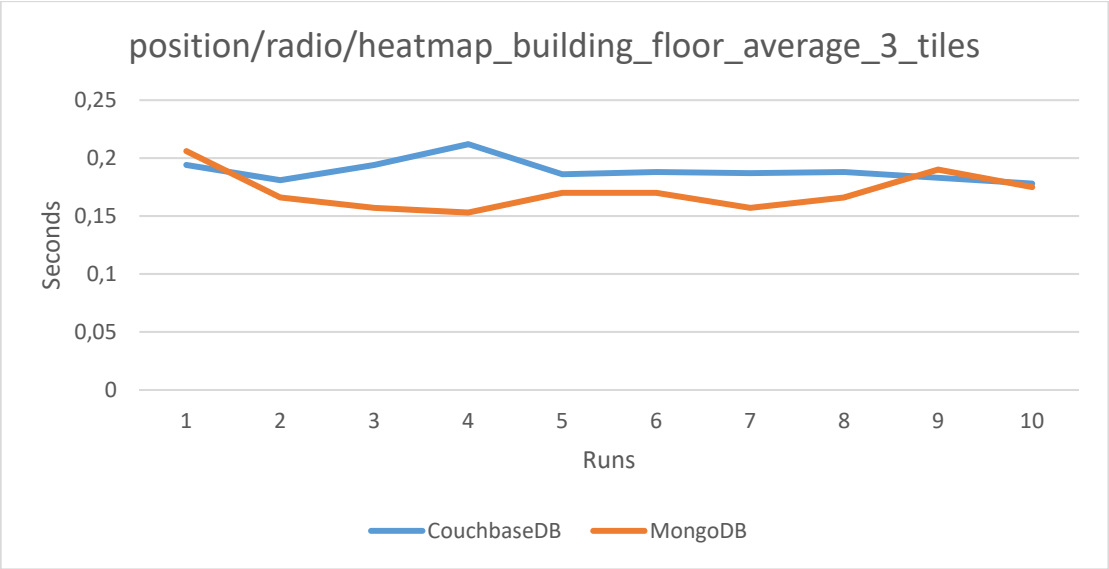
Γραφική παράσταση 6.7: Σύγκριση χρόνου εκτέλεσης του endpoint navigation/route μεταξύ MongoDB και CouchbaseDB

Στην πιο κάτω γραφική παράσταση βλέπουμε τον χρόνο εκτέλεσης του position/radio/Aps_building_floor το οποίο υπολογίζει την τοποθεσία των routers σε ένα κτήριο. Η MongoDB την πρώτη φορά χρειάζεται περίπου ένα δευτερόλεπτο για να υπολογίσει τις τοποθεσίες των router όπου και τις αποθηκεύει σε ένα collection, έτσι τις επόμενες φορές χρειάζεται γύρω στα 50ms για να υπολογίσει την τοποθεσία τους ενώ η CouchbaseDB χρειάζεται περίπου 450ms. Η τεχνική αυτή παρόλο που θυμίζει την συμπεριφορά των χρόνων εκτέλεσης της Couchbase στα προηγούμενα πειράματα διαφέρει και θα εξηγηθεί στην συνέχεια γιατί.



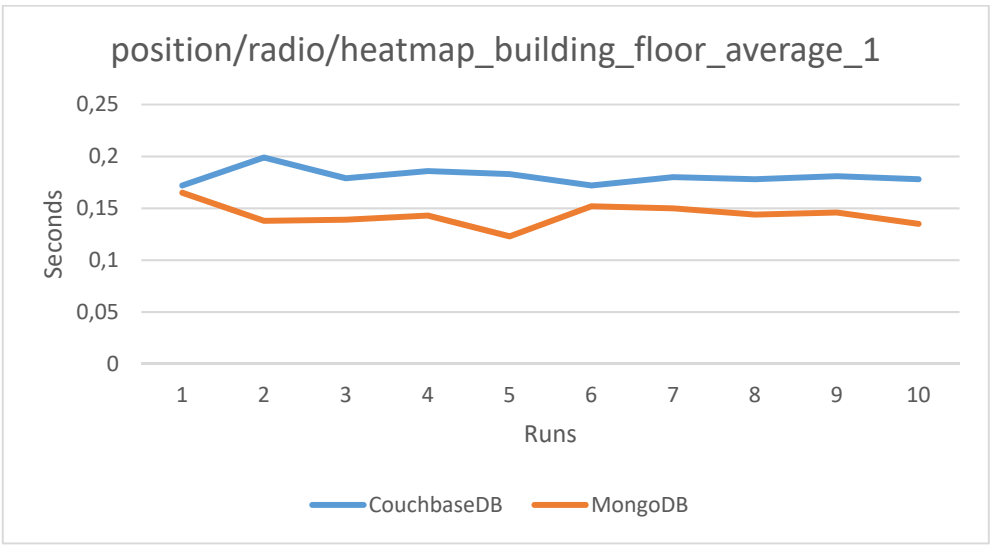
Γραφική παράσταση 6.8: Σύγκριση χρόνου εκτέλεσης του endpoint position/radio/APs_building_floor μεταξύ MongoDB και CouchbaseDB

Το πιο κάτω endpoint, position/radio/heatmap_building_floor_average_3_tiles, βρίσκει αποτυπώματα Wi-Fi τα οποία ανήκουν σε κάποιο πλακίδιο του χάρτη σε επίπεδο zoom 3. Στο συγκεκριμένο πείραμα τα αποτυπώματα που βρίσκονταν στο πλακίδιο ήταν 686. Η MongoDB έχει καλύτερη επίδοση από την Couchbase αφού τα αποτυπώματα βρίσκονται σε ένα από τα collections που προστέθηκαν και περιέχουν computed patterns για σκοπούς καλύτερης απόδοσης.



Γραφική παράσταση 6.9: Σύγκριση χρόνου εκτέλεσης του endpoint position/radio/heatmap_building_floor_average_3_tiles μεταξύ MongoDB και CouchbaseDB

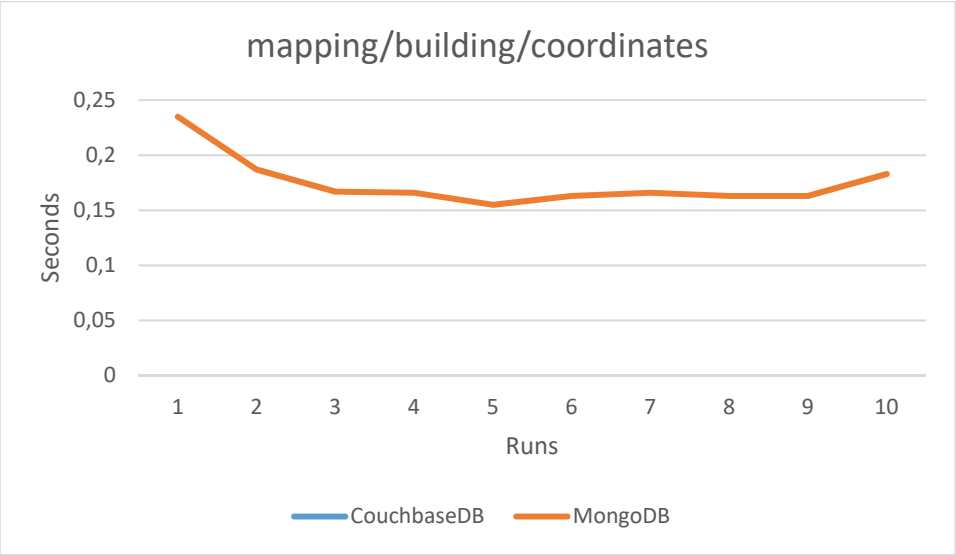
Η πιο κάτω γραφική παράσταση δείχνει τον χρόνο εκτέλεσης του endpoint position/radio/heatmap_building_floor_average_1 το οποίο είναι παρόμοιο με το προηγούμενο με την διαφορά ότι τα αποτυπώματα δεν πρέπει να ανήκουν σε κάποιο πλακίδιο αλλά ομαδοποιούνται με επίπεδο zoom 1.



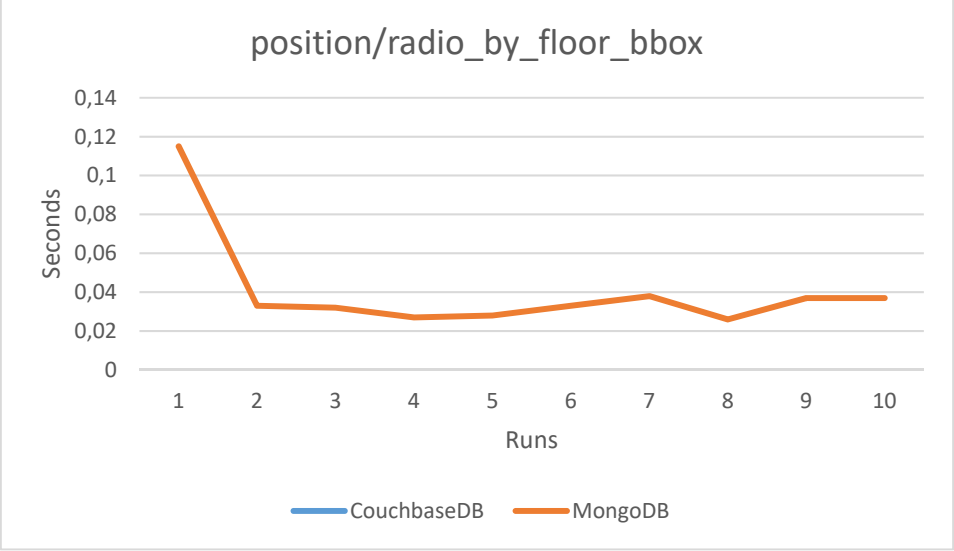
Γραφική παράσταση 6.10: Σύγκριση χρόνου εκτέλεσης του endpoint position/radio/heatmap_building_floor_average_1 μεταξύ MongoDB και CouchbaseDB

Οι δύο γραφικές παραστάσεις που ακολουθούν δείχνουν τον χρόνο εκτέλεσης δύο χωρικών endpoints. Μπορούμε να δούμε ότι δεν αναγράφονται χρόνοι εκτέλεσης για την Couchbase, αφού τα views για τα αιτήματα αυτά είναι απενεργοποιημένα. Ο λόγος που

απενεργοποιήθηκαν ήταν επειδή η Couchbase αδυνατούσε να τρέξει τα views αφού έπαιρναν πολύ χρόνο να κτιστούν σε σημείο που γίνονταν time-out ή γέμιζαν την ram και προκαλούσαν πρόβλημα στο σύστημα. Με την νέα βάση δεδομένων τα endpoints αυτά εκτελούνται όπως αναμένεται σε πολύ μικρό χρονικό διάστημα αφού το ένα χρειάζεται 250ms ενώ το άλλο 40ms αφού γίνει cached.



Γραφική παράσταση 6.11: Σύγκριση χρόνου εκτέλεσης του endpoint mapping/building/coordinates μεταξύ MongoDB και CouchbaseDB



Γραφική παράσταση 6.12: Σύγκριση χρόνου εκτέλεσης του endpoint position/radio_by_floor_bbox μεταξύ MongoDB και CouchbaseDB

Στα πιο πάνω πειράματα τις πλείστες φορές η CouchbaseDB είναι πιο γρήγορη από την MongoDB. Αυτό οφείλεται στα materialized views που κτίζονται από την CouchbaseDB

τα οποία αποθηκεύουν την απάντηση με αποτέλεσμα τα endpoints να εκτελούνται πολύ γρήγορα. Αυτό δεν είναι πλεονέκτημα αφού καθώς ένας χρήστης αλληλοεπιδρά με το Anyplace σχεδόν πάντα εκτελεί το κάθε αίτημα μόνο μία φορά. Για παράδειγμα όταν ο χρήστης θέλει να δει το περιεχόμενο ενός κτηρίου θα κάνει κλικ μία φορά σε αυτό και θα εμφανιστούν όλα τα σημεία ενδιαφέροντος και οι ενώσεις τους. Επίσης, τα views δεν διατηρούν την βελτίωση του χρόνου για αρκετό χρονικό διάστημα αφού ανανεώνονται συνέχεια, δηλαδή εάν ο χρήστης εκτελέσει κάποιο αίτημα ο χρόνος εκτέλεσης θα είναι 700ms, εάν το εκτελέσει ξανά το αμέσως επόμενο δευτερόλεπτο ο χρόνος εκτέλεσης θα μειωθεί στα 200ms αλλά εάν εκτελεστεί μετά από αρκετή ώρα θα είναι και πάλι 700ms. Η MongoDB σε κάποια endpoints (*Γραφική Παράσταση 6.8*) χρησιμοποιεί παρόμοια τεχνική όπου αποθηκεύει το αποτέλεσμα ενός αιτήματος για σκοπούς επιτάχυνσης. Η διαφορά είναι ότι ο χρόνος εκτέλεσης μειώνεται μόνιμα εκτός εάν μεταβληθούν τα δεδομένα στα οποία βασίστηκε και κτίστηκε η αποθηκευμένη απάντηση (για παράδειγμα εάν διαγραφτούν 300 αποτυπώματα Wi-Fi τότε το αίτημα που βρίσκει πόσα routers υπάρχουν στο κτήριο χρειάζεται ανανέωση). Αυτό δεν αποτελεί πρόβλημα αφού το Anyplace εκ φύσεως έχει πολύ περισσότερα reads παρά writes. Επίσης, η αποθηκευμένη απάντηση δεν ανανεώνεται την ίδια στιγμή, αλλά διαγράφεται και ξανά-κτίζεται σε περίπτωση που ζητηθεί. Παρόλα αυτά, το τίμημα για την γρήγορη εκτέλεση των endpoints με χρήση materialized views από την Couchbase πληρώνεται με μεγάλη κατανάλωση μνήμης σε σημείο που το σύστημα έχει δυσλειτουργίες.

Επίσης, δεν παρουσιάστηκαν γραφικές παραστάσεις για όλα τα endpoints αλλά ένα αντιπροσωπευτικό υποσύνολο αυτών. Υπάρχουν αιτήματα από όλες τις κατηγορίες (mapping, position, navigation, spatial queries) για διάφορα μεγέθη απάντησης (1, 4, 443, 686, 2880), έτσι η επιδόσεις που είδαμε δεν εστιάζουν μόνο σε κάποια κατηγορία αιτημάτων.

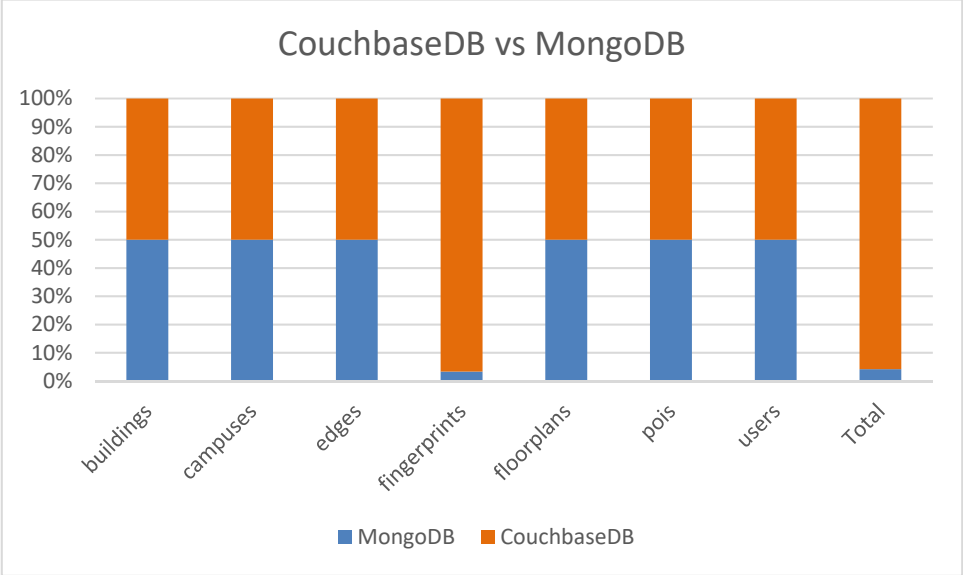
6.3 Μέγεθος δεδομένων

6.3.1. Αντικείμενα

Στον *Πίνακα 6.1* βλέπουμε αναλυτικά πόσα αντικείμενα έχε η CouchbaseDB και η MongoDB. Όπως προανάφερα στο Κεφάλαιο 4, τα δεδομένα στην CouchbaseDB δεν ήταν κατηγοριοποιημένα αλλά όλα μαζί σε ένα bucket, δηλαδή δεν είναι εμφανής πόσα αντικείμενα υπάρχει για κάθε κατηγορία, αλλά αν ήταν θα ήταν όπως πιο κάτω.

	CouchbaseDB	MongoDB
buildings	4,439	4,439
campuses	258	258
edges	45,269	45,269
fingerprints	11,142,975	387,611
floorplans	3,975	3,975
pois	49,037	49,037
users	4,353	4,353
Total	11,250,306	494,942

Πίνακας 6.1: Σύγκριση αντικειμένων ανά κατηγορία



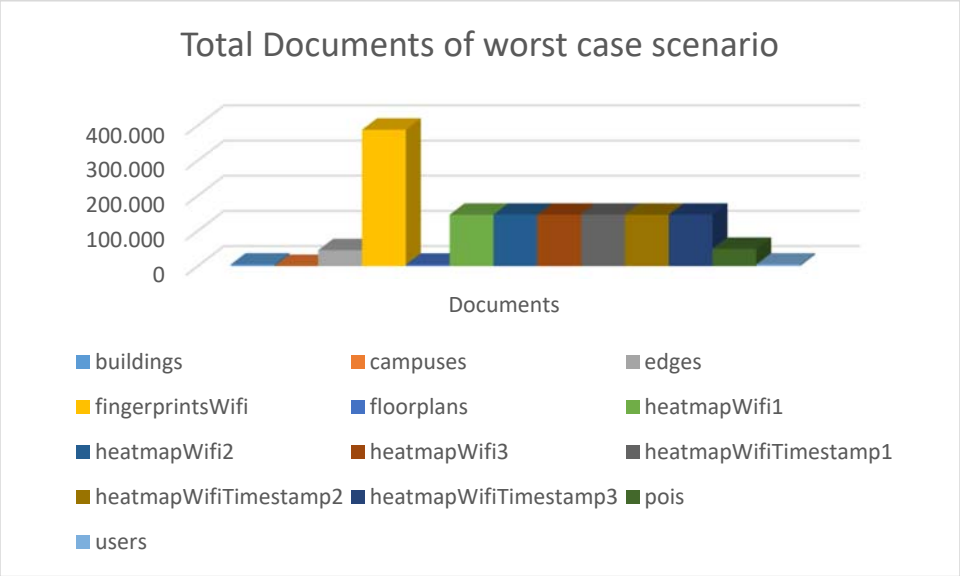
Γραφική παράσταση 6.13: Σύγκριση αντικειμένων ανά κατηγορία

Είναι εμφανής η τεράστια μείωση του συνόλου των αντικειμένων, η οποία επιτεύχθηκε με την ομαδοποίηση των αποτυπωμάτων Wi-Fi. Το ποσοστό της κατηγορίας αποτυπώματα Wi-Fi σε ολόκληρη την βάση ήταν 99% ενώ τώρα είναι 78%. Η μείωση αυτή είναι πολύ σημαντική αφού συνολικά μειώθηκε ο αριθμός των αποτυπωμάτων σχεδόν 29 φορές ενώ ο αριθμός αντικειμένων ολόκληρης της βάσης μειώθηκε περίπου 23 φορές.

Σε μεταγενέστερο στάδιο όταν υλοποιήθηκαν κάποια endpoints χρειάστηκαν καινούρια collections έτσι ο αριθμός των αντικειμένων αυξήθηκε. Με μια πρώτη ματιά αυτό φαίνεται κακή τεχνική αλλά εάν το δούμε συγκριτικά με την προηγούμενη έκδοση ουσιαστικά όλα views που υλοποιούσαν τα endpoints δημιουργούσαν νέους πίνακες όπου αποθήκευαν την απάντηση. Επιπλέον, τα collections αυτά δεν διατηρούν τα δεδομένα που έχουν για πάντα αφού όταν γίνονται διάφορες αλλαγές διαγράφονται και

εισάγονται ξανά ανανεωμένα όταν κάποιος τα ζητήσει κάποιος χρήστης. Επίσης, η καινούρια βάση δεδομένων είναι σχεδιασμένη να αποθηκεύει τα δεδομένα σε κατηγορίες, επομένως δεν επηρεάζεται η μία κατηγορία από την άλλη. Ακόμη, αφού ο αριθμός των αντικειμένων μειώθηκε σχεδόν 23 φορές η εισαγωγή καινούριων αντικειμένων για βελτίωση της ταχύτητας ήταν εφικτή και δεν προκάλεσε καθυστέρηση σε άλλα endpoints. Τέλος, θα δούμε ότι το πραγματικό πρόβλημα ήταν ο αποθηκευτικός χώρος που κατανάλωναν τα views και ο χρόνος εκτέλεσης τους.

Στην πιο κάτω γραφική παράσταση βλέπουμε την χείριστη περίπτωση που μπορεί να βρεθεί η βάση δεδομένων MongoDB. Αυτό προκύπτει όταν για κάθε αποτύπωμα δεδομένων δημιουργήθηκαν τα ανάλογα αντικείμενα για διάφορα zoom levels (1,2 και 3).



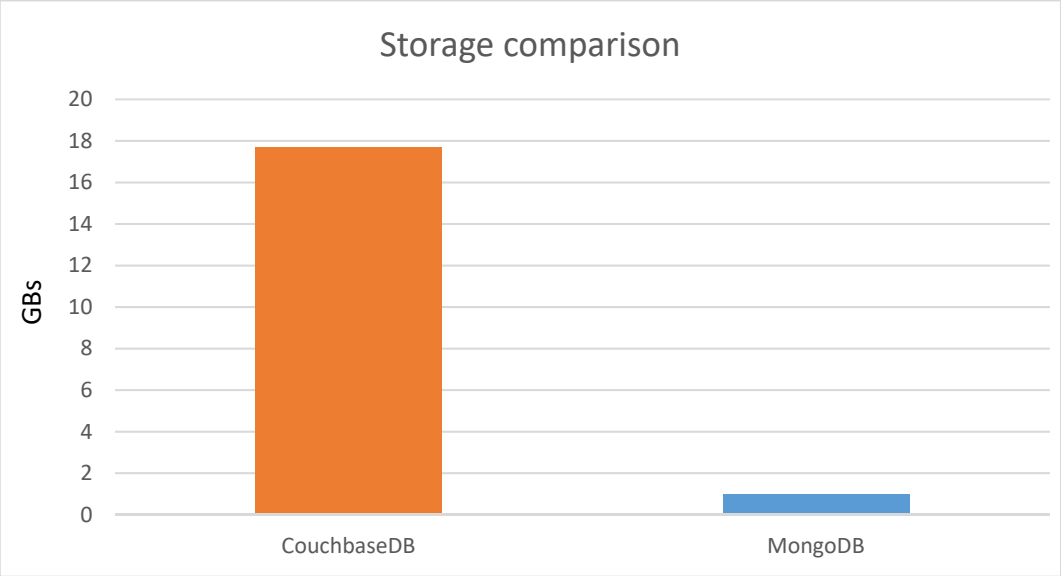
Πίνακας 6.14: Χείριστη μορφή τελικού σχήματος MongoDB

6.3.2. Αποθηκευτικός χώρος

Η αφορμή για την αλλαγή της βάσης δεδομένων ήταν το μεγάλο μέγεθος της μνήμης που κατανάλωνε το σύστημα, έτσι δόθηκε μεγάλη βαρύτητα στην μείωση αυτής κατά την αλλαγή. Ο στόχος αυτός επιτεύχθηκε με μεγάλη επιτυχία αφού η μείωση του απαιτούμενου αποθηκευτικού χώρου είναι τεράστια.

Το συνολικό μέγεθος του αποθηκευτικού χώρου που καταναλωνόταν από την CouchbaseDB είναι 17.7GB, το οποίο οφείλεται στα materialized views αφού

καταναλώνουν περίπου 14GB, ενώ η MongoDB στην χείριστη περίπτωση αναμένεται να καταναλώνει περίπου 1GB. Με την μεταφορά των αντικειμένων στην MongoDB αρχικά χρειάζεται περίπου 850MB, και εάν υποθέσουμε ότι τα collections που λειτουργούν σαν caches δημιούργησαν τα μισά αντικείμενα επειδή μόνο αυτά ζητήθηκαν τότε αναμένεται να χρειάζονται 900-920MB. Επομένως ο συνολικός απαιτούμενος αποθηκευτικός χώρος μειώθηκε περίπου 18 φορές.



Γραφική παράσταση 6.15: Σύγκριση αποθηκευτικού χώρου μεταξύ CouchbaseDB και MongoDB

Κεφάλαιο 7

Συμπεράσματα και μελλοντικές επεκτάσεις

7.1 Συμπεράσματα 62

7.2 Μελλοντικές Επεκτάσεις 63

7.2.1 Επεκτάσεις στα endpoints 63

7.2.2 Επεκτάσεις στην βάση δεδομένων 63

7.2.3 Επεκτάσεις στο Anyplace 64

7.1 Συμπεράσματα

Το Anyplace είναι ένα ολοκληρωμένο σύστημα εσωτερικής πληροφορίας όπου μπορεί εύκολα να επεκταθεί και να αναπτυχθεί. Σε αυτή τη διπλωματική εργασία αναδεικνύονται οι ιδιότητες αυτές αφού μέσα από αυτήν έγιναν διάφορες βελτιώσεις και αλλαγές οι οποίες οδήγησαν στην νέα έκδοση που παρουσιάζεται.

Με την σύγχρονη και πιεστική καθημερινότητα που βιώνουμε ο χρόνος μας είναι πολύτιμος. Σε συνδυασμό με την αυξημένη χρήση της τεχνολογίας από την πλειοψηφία των ανθρώπων και το γεγονός ότι ξοδεύουν το 80-90% σε εσωτερικούς χώρους προκύπτει η ανάγκη ενίσχυσης της ποιότητας περιήγησης σε αυτούς. Ως αποτέλεσμα η καλύτερη διαχείριση χρόνου και ευκολότερη διακίνηση στους εσωτερικούς χώρους.

Σημαντικό ρόλο σε εφαρμογές εσωτερικών χώρων είναι η ακριβής διατύπωση της τοποθεσίας του χρήστη μέσα στο κτήριο. Σε αντίθεση με τους εξωτερικούς χώρους, όπου η πληροφορία αυτή μπορεί να ληφθεί με τεχνολογίες όπως το GPS και χάρτες από το Google Maps, η γεωτοποθέτηση σε εσωτερικούς χώρους είναι κάτι πρωτόγνωρο το οποίο απαιτεί οικονομικές λύσεις με όσο το δυνατό μεγαλύτερη ακρίβεια. Η πληθώρα σημείων

πρόσβασης Wi-Fi σε ένα κτήριο είναι μία από τις επιλογές για τον προσδιορισμό της θέσης των χρηστών.

Ακόμα με το πέρας της παρούσας διπλωματικής εργασίας μπορούμε να βγάλουμε το συμπέρασμα ότι το Anyplace, αν και ολοκληρωμένο σύστημα, έχει περιθώρια βελτίωσης αφού στην προκειμένου περίπτωση η αλλαγή του υποσυστήματος δεδομένων μείωσε τον απαιτούμενο αποθηκευτικό χώρο και παράλληλα επέφερε βελτίωση στον χρόνο εκτέλεσης σε κάποια endpoints ενώ σε κάποια άλλα έγιναν διορθώσεις και προστέθηκαν έλεγχοι.

7.2 Μελλοντικές επεκτάσεις

Το Anyplace βρίσκεται σε ένα ικανοποιητικό στάδιο αφού χρησιμοποιείται από όλες τις ομάδες χρηστών (από άπειρους χρήστες μέχρι και ερευνητικές ομάδες, σε παγκόσμια κλίμακα). Τα τελευταία χρόνια εξελίσσεται συνεχώς έτσι ώστε να βελτιώνει την εμπειρία χρήσης των χρηστών και να καλύπτει της ανάγκες τους με κάθε τρόπο. Παρόλα αυτά, πάντα υπάρχουν περιθώρια βελτίωσης με την ομαλότερη λειτουργία του συστήματος ως έχει και με την εισαγωγή νέων χαρακτηριστικών και λειτουργιών.

7.2.1 Επεκτάσεις στα endpoints

Όπως είδαμε υπάρχουν 71 endpoints τα οποία αλληλοεπιδρούν με την βάση δεδομένων εκ των οποίων τα 59 υλοποιήθηκαν σε MongoDB. Σε μεταγενέστερο θα υλοποιηθούν τα endpoints που έμειναν παρόλο που δεν χρησιμοποιούνται. Παράλληλα, αυτά που είναι υλοποιημένα μπορούν να βελτιωθούν όσο αφορά χρόνο εκτέλεσης με διάφορες τεχνικές όπως indexes.

7.2.2 Επεκτάσεις στην βάση δεδομένων

Όσο αφορά την βάση δεδομένων υπάρχουν περιθώρια βελτίωσης στους τύπους δεδομένων. Υπάρχουν αρκετές μεταβλητές που είναι τύπου integer, double, boolean οι οποίες αποθηκεύονται σαν συμβολοσειρές. Η αλλαγή αυτή δεν έγινε στην αρχική μεταφορά αφού είναι χρονοβόρα διότι απαιτούνται οι ανάλογες αλλαγές στην εφαρμογή και στο JavaScript (π.χ. σε μια μεταβλητή που θα έπρεπε να είναι double και διαβάζετε σαν string με την μέθοδο getString πρέπει να αλλαχτεί σε getDouble). Με την αλλαγή αυτή πλέον η βάση δεδομένων θα μεταβεί στο _schema 1.

7.2.3 Επεκτάσεις στο Anyplace

Επίσης, κάποιες άλλες πιθανές επεκτάσεις στο σύστημα Anyplace είναι νέο σύστημα εγγραφής και chat. Το νέο σύστημα εγγραφής θα προσφέρει την δυνατότητα στους χρήστες να συνδέονται χωρίς Gmail αλλά με τον παραδοσιακό τρόπο σύνδεσης σε κάποιο σύστημα (username και password). Στο μεταξύ, έγιναν ήδη κάποιες προσπάθειες για υλοποίηση chat όπου οι χρήστες θα μπορούν να επικοινωνούν μεταξύ τους. Με την επιτυχή εισαγωγή μιας λειτουργίας σαν αυτή οι χρήστες έρχονται πιο κοντά με αποτέλεσμα να μπορούν να συνεργαστούν για χαρτογράφηση κτηρίων ή και να ζητούν βοήθεια όταν αντιμετωπίζουν δυσκολίες.

Βιβλιογραφία

[1] “Strategy Analytics” http://www.strategyanalytics.com.

[2] “Anyplace” https://anyplace.cs.ucy.ac.cy/

[3] “MongoDB” https://www.mongodb.com/

[4] “MongoDB Compass” https://www.mongodb.com/products/compass

[5] “Scala” https://www.scala-lang.org/

[6] “SBT (Scala Build Tool)” https://www.scala-sbt.org/

[7] “Play Framework” https://www.playframework.com/

[8] “Python” https://www.python.org/

[9] "The Airplace Indoor Positioning Platform for Android Smartphones", Christos Laoudias, Georgios Constantinou, Marios Constantinides, Silouanos Nicolaou, Demetrios Zeinalipour-Yazti, Christos G. Panayiotou "Proceedings of the 13th IEEE International Conference on Mobile Data Management" (MDM '12), IEEE Computer Society, Pages: 312-315, Bangalore, India, https://www.cs.ucy.ac.cy/~dzeina/papers/mdm12-airplace-demo.pdf

[10] "Crowdsourced Indoor Localization for Diverse Devices through Radiomap Fusion", Christos Laoudias, Demetrios Zeinalipour-Yazti and Christos G. Panayiotou "Proceedings of the 4th Intl. Conference on Indoor Positioning and Indoor Navigation" (IPIN '13), Montbeliard-Belfort, France Pages: 1-7, 2013, https://www.cs.ucy.ac.cy/~dzeina/papers/ipin13-crowdsourcing.pdf

[11] "Anyplace: A Crowdsourced Indoor Information Service", Kyriakos Georgiou, Timotheos Constambeys, Christos Laoudias, Lambros Petrou, Georgios Chatzimilioudis and Demetrios Zeinalipour-Yazti "Proceedings of the 16th IEEE International Conference on Mobile Data Management" (MDM '15), IEEE Press, Volume 2, Pages: 291-294, 2015, https://www.cs.ucy.ac.cy/~dzeina/papers/mdm15-anyplace-demo.pdf

[12] "Privacy-Preserving Indoor Localization on Smartphones (Extended Abstract)", Andreas Konstantinidis, Georgios Chatzimilioudis, Demetrios Zeinalipour-Yazti,

Paschalis Mpeis, Nikos Pelekis and Yannis Theodoridis "Proceedings of the IEEE 32nd International Conference on Data Engineering" (ICDE '16), IEEE Computer Society, Pages: 1470--1471, Helsinki, Finland, ISBN: 978-1-5090-2020-1, 2016, https://www.cs.ucy.ac.cy/~dzeina/papers/tkde15-tvm.pdf

[13] “"Indoor Localization Accuracy Estimation from Fingerprint Data", Artyom Nikitin, Christos Laoudias, Georgios Chatzimilioudis, Panagiotis Karras and Demetrios Zeinalipour-Yazti "Proceedings of the 18th IEEE International Conference on Mobile Data Management" (MDM '17), IEEE Computer Society, Pages: 196--205, May 29 - June 1, 2017, Daejeon, South Korea, 2017”, https://www.cs.ucy.ac.cy/~dzeina/papers/mdm17-acces.pdf

[14] "Towards Robust Methods for Indoor Localization using Interval Data", Nacim Ramdani, Demetrios Zeinalipour-Yazti, Michalis Karamousadakis, Andreas Panayides "Proceedings of the 1st IEEE Intl. Workshop on ALgorithms for Indoor Architectures and Systems" (ALIAS '19), IEEE Press, Pages: 403--408, June 10, 2019, Hong Kong, 2019, https://www.cs.ucy.ac.cy/~dzeina/papers/alias19-interval.pdf

[15] “The Anyplace 4.0 IoT Localization Architecture", Paschalis Mpeis, Thierry Roussel, Manish Kumar, Constantinos Costa, Christos Laoudias, Denis Capot-Ray Demetrios Zeinalipour-Yazti, Proceedings of the 21st IEEE International Conference on Mobile Data Management (MDM '20), IEEE Computer Society, ISBN:, pp. 8, June 30 - July 3, 2020, Versailles, France, 2020

[16] “InfluxDB” https://www.influxdata.com/

[17] “OAuth 2.0 protocol” https://developers.google.com/identity/protocols/oauth2

[18] “API” https://en.wikipedia.org/wiki/API

[19] “Anyplace API” https://ap.cs.ucy.ac.cy/developers/

[20] “A realistic evaluation and comparison of indoor location technologies: Experiences and lessons learned”, D. Lymberopoulos, J. Liu, X. Yang, R. R. Choudhury, V. Handziski, and S. Sen, in Proceedings of the 14th international conference on information processing in sensor networks. ACM, 2015, pp. 178–189.

[21] “Indoor positioning techniques based on wireless LAN”, C. Rizos, A. G. Dempster, B. Li, and J. Salter, in 1st IEEE Intl. Conf. on Wireless Broadband and Ultra Wideband Communications. IEEE, 2007, pp. 13–16.

[22] “A survey of indoor positioning systems for wireless personal networks”, Y. Gu, A. Lo, and I. Niemegeers, IEEE Communications surveys & tutorials, vol. 11, no. 1, pp. 13–32, 2009.

[23] “Redpin-adaptive, zero-configuration indoor localization through user collaboration”, P. Bolliger, in Proceedings of the first ACM international workshop on Mobile entity localization and tracking in GPS-less environments, 2008, pp. 55–60

[24] “The anatomy of the anyplace indoor navigation service”, D. Zeinalipour-Yazti and C. Laoudias, SIGSPATIAL Special, vol. 9, no. 2, pp. 3– 10, 2017

[25] “Iot data prefetching in indoor navigation SOAs”, A. Konstantinidis, P. Irakleous, Z. Georgiou, D. Zeinalipour-Yazti, and P. K. Chrysanthis, ACM Transactions on Internet Technology (TOIT), vol. 19, no. 1, pp. 1–21, 2018.

Παράρτημα Α – Κώδικας καινούριας κλάσης MongodbDatasource.Scala

```
/*
 * AnyPlace: A free and open Indoor Navigation Service with superb
accuracy!
 *
 * Anyplace is a first-of-a-kind indoor information service offering
GPS-less
 * localization, navigation and search inside buildings using ordinary
smartphones.
 *
 * Author(s): Nikolas Neofytou
 *
 * Co-Supervisor: Paschalis Mpeis
 * Supervisor: Demetrios Zeinalipour-Yazti
 *
 * URL: https://anyplace.cs.ucy.ac.cy
 * Contact: anyplace@cs.ucy.ac.cy
 *
 * Copyright (c) 2016, Data Management Systems Lab (DMSL), University
of Cyprus.
 * All rights reserved.
 *
 * Permission is hereby granted, free of charge, to any person
obtaining a copy of
 * this software and associated documentation files (the “Software”),
to deal in the
 * Software without restriction, including without limitation the
rights to use, copy,
 * modify, merge, publish, distribute, sublicense, and/or sell copies
of the Software,
 * and to permit persons to whom the Software is furnished to do so,
subject to the
 * following conditions:
 *
 * The above copyright notice and this permission notice shall be
included in all
 * copies or substantial portions of the Software.
 *
 * THE SOFTWARE IS PROVIDED “AS IS”, WITHOUT WARRANTY OF ANY KIND,
EXPRESS
 * OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY,
 * FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT. IN NO EVENT
SHALL THE
 * AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY CLAIM, DAMAGES OR
OTHER
 * LIABILITY, WHETHER IN AN ACTION OF CONTRACT, TORT OR OTHERWISE,
ARISING
 * FROM, OUT OF OR IN CONNECTION WITH THE SOFTWARE OR THE USE OR OTHER
 * DEALINGS IN THE SOFTWARE.
 *
 */
package datasources
```

```
import java.io.{FileOutputStream, IOException, PrintWriter}
import java.util

import com.couchbase.client.java.document.json.JsonObject
import datasources.MongodbDatasource.{__geometry, admins, convertJson,
mdb}
import datasources.SCHEMA._
import db_models.RadioMapRaw.unrollFingerprint
import db_models.{Connection, Poi, RadioMapRaw}
import floor_module.IAlgo
import org.mongodb.scala._
import org.mongodb.scala.bson.BsonDocument
import org.mongodb.scala.model.Aggregates
import org.mongodb.scala.model.Aggregates.project
import org.mongodb.scala.model.Filters._
import play.Play
import play.api.libs.json.{JsNumber, JsObject, JsValue, Json}
import play.twirl.api.TemplateMagic.javaCollectionToScala
import utils.JsonUtils.cleanupMongoJson
import utils.{GeoJSONPoint, GeoPoint, JsonUtils, LPLogger}

import scala.collection.JavaConversions.mapAsScalaMap
import scala.collection.mutable.ListBuffer
import scala.concurrent.Await
import scala.concurrent.duration.Duration
import scala.text.Document.break
import scala.util.control.Breaks


object MongodbDatasource {
  private var sInstance: MongodbDatasource = null
  private var mdb: MongoDatabase = null
  private var admins: List[String] = List[String]()
  val __SCHEMA: Int = 0
  val __USERS = "users"
  val __geometry = "geometry"

  def getMDB: MongoDatabase = mdb

  def getStaticInstance: MongodbDatasource = {
    val conf = Play.application().configuration()
    val username = conf.getString("mongodb.app.username")
    val password = conf.getString("mongodb.app.password")
    val hostname = conf.getString("mongodb.hostname")
    val port = conf.getString("mongodb.port")
    val database = conf.getString("mongodb.database")
    sInstance = createInstance(hostname, database, username, password,
port)
    sInstance
  }

  def createInstance(hostname: String, database: String, username:
String, password: String, port: String): MongodbDatasource = {
    val uri: String = "mongodb://" + username + ":" + password + "@" +
hostname + ":" + port
    val mongoClient: MongoClient = MongoClient(uri)
    mdb = mongoClient.getDatabase(database)
    val collections = mdb.listCollectionNames()
    val awaited = Await.result(collections.toFuture, Duration.Inf)
    val res = awaited.toList
```

```
    LPLogger.info("MongoDB: Connected to: " + hostname + ":" + port)
    LPLogger.debug("Collections = " + res)
    admins = loadAdmins()
    new MongodbDatasource()
  }

  /**
   * Cache admins on MongoDB initialization.
   *
   * @return a list with admins.
   */
  def loadAdmins(): List[String] = {
    val collection = mdb.getCollection(__USERS)
    val query = BsonDocument("type" -> "admin")
    val adm = collection.find(query)
    val awaited = Await.result(adm.toFuture, Duration.Inf)
    val res = awaited.toList
    val ret = new util.ArrayList[String]
    for (admin <- res) {
      val temp = Json.parse(admin.toJson())
      ret.add((temp \ "owner_id").as[String])
    }
    ret.toList
  }

  def convertJson(list: List[Document]): List[JsValue] = {
    val jsList = ListBuffer[JsValue]()
    for (doc <- list) {
      if (doc != null)
        jsList.append(convertJson(doc))
    }
    jsList.toList
  }

  def convertJson(doc: Document) =
cleanupMongoJson(Json.parse(doc.toJson()))
}

class MongodbDatasource() extends IDatasource {

  override def poisByBuildingIDAsJson(buid: String): List[JsValue] = {
    val collection = mdb.getCollection("pois")
    val query = BsonDocument("buid" -> buid)
    val pois = collection.find(query)
    val awaited = Await.result(pois.toFuture, Duration.Inf)
    val res = awaited.toList
    val listJson = convertJson(res)
    val poisArray = new java.util.ArrayList[JsValue]()
    for (poi <- listJson)
      poisArray.add(poi.as[JsObject] - "owner_id" - "geometry" - "_id"
- "_schema")
    poisArray.toList
  }

  def greeklishTogreekList(greeklish: String) = {
    val words = new java.util.ArrayList[String]
    words.add("")
    val myChars = greeklish.toCharArray
    var i = 0
    for (i <- 0 until greeklish.length) {
```

```
      val size = words.size
      var j = 0
      for (j <- 0 until size) {
        var myword = ""
        myword = words.get(j)
        if (myChars(i) == 'a') words.add(myword + "α")
        else if (myChars(i) == 'b') {
          words.add(myword + "β")
          words.add(myword + "μπ")
        }
        else if (myChars(i) == 'c') {
          if (i < greeklish.length - 1) if (myChars(i + 1) == 'h')
words.add(myword + "χ")
          words.add(myword + "γ")
        }
        else if (myChars(i) == 'd') {
          words.add(myword + "δ")
          words.add(myword + "ντ")
        }
        else if (myChars(i) == 'e') {
          words.add(myword + "ε")
          words.add(myword + "αι")
          words.add(myword + "ι")
          words.add(myword + "η")
        }
        else if (myChars(i) == 'f') words.add(myword + "φ")
        else if (myChars(i) == 'g') {
          words.add(myword + "γ")
          words.add(myword + "γγ")
          words.add(myword + "γκ")
        }
        else if (myChars(i) == 'h') {
          if (myword.length > 0 && myword.charAt(myword.length - 1) ==
'θ') {
            words.add(myword)

          }
          else if (myword.length > 0 && myword.charAt(myword.length -
1) == 'χ') {
            words.add(myword)

          } else {
            words.add(myword + "χ")
            words.add(myword + "η")
          }
        }
        else if (myChars(i) == 'i') {
          words.add(myword + "ι")
          words.add(myword + "η")
          words.add(myword + "υ")
          words.add(myword + "οι")
          words.add(myword + "ει")
        }
        else if (myChars(i) == 'j') words.add(myword + "ξ")
        else if (myChars(i) == 'k') {
          if (i < greeklish.length - 1) if (myChars(i + 1) == 's')
words.add(myword + "ξ")
          words.add(myword + "κ")
        }
        else if (myChars(i) == 'l') words.add(myword + "λ")
```

```
        else if (myChars(i) == 'm') words.add(myword + "μ")
        else if (myChars(i) == 'n') words.add(myword + "ν")
        else if (myChars(i) == 'o') {
          words.add(myword + "ο")
          words.add(myword + "ω")
        }
        else if (myChars(i) == 'p') {
          if (i < greeklish.length - 1) if (myChars(i + 1) == 's')
words.add(myword + "ψ")
          words.add(myword + "π")
        }
        else if (myChars(i) == 'q') words.add(myword + ";")
        else if (myChars(i) == 'r') words.add(myword + "ρ")
        else if (myChars(i) == 's') {
          if (myword.length > 0 && myword.charAt(myword.length - 1) ==
'ξ') {
            words.add(myword)

          } else if (myword.length > 0 && myword.charAt(myword.length
- 1) == 'ψ') {
            words.add(myword)
          }
          else {
            words.add(myword + "σ")
            words.add(myword + "ς")
          }
        }
        else if (myChars(i) == 't') {
          if (i < greeklish.length - 1) if (myChars(i + 1) == 'h')
words.add(myword + "θ")
          words.add(myword + "τ")
        }
        else if (myChars(i) == 'u') {
          words.add(myword + "υ")
          words.add(myword + "ου")
        }
        else if (myChars(i) == 'v') words.add(myword + "β")
        else if (myChars(i) == 'w') words.add(myword + "ω")
        else if (myChars(i) == 'x') {
          words.add(myword + "χ")
          words.add(myword + "ξ")
        }
        else if (myChars(i) == 'y') words.add(myword + "υ")
        else if (myChars(i) == 'z') words.add(myword + "ζ")

      }

      for (k <- 0 until size) {
        words.remove(0)
      }
    }
    words
  }

  def greeklishTogreek(greeklish: String) = {
    val myChars = greeklish.toCharArray
    var i = 0
    while ( {
      i < greeklish.length
    }) {
```

```
myChars(i) match {
  case 'a' =>
    myChars(i) = 'α'

  case 'b' =>
    myChars(i) = 'β'

  case 'c' =>
    myChars(i) = 'ψ'

  case 'd' =>
    myChars(i) = 'δ'

  case 'e' =>
    myChars(i) = 'ε'

  case 'f' =>
    myChars(i) = 'φ'

  case 'g' =>
    myChars(i) = 'γ'

  case 'h' =>
    myChars(i) = 'η'

  case 'i' =>
    myChars(i) = 'ι'

  case 'j' =>
    myChars(i) = 'ξ'

  case 'k' =>
    myChars(i) = 'κ'

  case 'l' =>
    myChars(i) = 'λ'

  case 'm' =>
    myChars(i) = 'μ'

  case 'n' =>
    myChars(i) = 'ν'

  case 'o' =>
    myChars(i) = 'ο'

  case 'p' =>
    myChars(i) = 'π'

  case 'q' =>
    myChars(i) = ';'

  case 'r' =>
    myChars(i) = 'ρ'

  case 's' =>
    myChars(i) = 'σ'

  case 't' =>
    myChars(i) = 'τ'
```

```
          case 'u' =>
            myChars(i) = 'θ'

          case 'v' =>
            myChars(i) = 'ω'

          case 'w' =>
            myChars(i) = 'ς'

          case 'x' =>
            myChars(i) = 'χ'

          case 'y' =>
            myChars(i) = 'υ'

          case 'z' =>
            myChars(i) = 'ζ'

          case _ =>

        }

        {
          i += 1;
          i - 1
        }
      }
      String.valueOf(myChars)
    }

    var wordsELOT = new java.util.ArrayList[java.util.ArrayList[String]]
    var allPoisSide = new java.util.HashMap[String,
java.util.List[JsValue]]()
    var allPoisbycuid = new java.util.HashMap[String,
java.util.List[JsValue]]()
    var lastletters = ""


    override def poisByBuildingAsJson2(cuid: String, letters: String):
List[JsValue] = {
      var pois: java.util.List[JsValue] = null
      val pois2 = new java.util.ArrayList[JsValue]
      pois = allPoisbycuid.get(cuid)
      if (pois == null) {
        val buids = getBuildingSet(cuid)
        val tempPois = new java.util.ArrayList[JsValue]
        for (buid <- (buids(0) \ "buids").as[List[String]]) {
          for (poi <- poisByBuildingAsJson(buid)) {
            if (poi != null) {
              tempPois.add(poi)
            }
          }
        }
        pois = tempPois
      }
      val words = letters.split(" ")
      var flag = false
      for (json <- pois) {
        flag = true
```

```
      for (w <- words if flag) {
        var name = ""
        var description = ""
        if ((json \ "name").toOption.isDefined)
          name = (json \ "name").as[String].toLowerCase
        if ((json \ "description").toOption.isDefined)
          description = (json \ "description").as[String].toLowerCase
        if (!(name.contains(w.toLowerCase) ||
description.contains(w.toLowerCase)))
          flag = false
      }
      if (flag) pois2.add(json)
    }
    pois2.toList
  }

  override def poisByBuildingAsJson2GR(cuid: String, letters: String):
List[JsValue] = {
    var pois = allPoisbycuid.get(cuid)
    if (pois == null) {
      val buids = getBuildingSet(cuid)
      val tempPois = new java.util.ArrayList[JsValue]
      for (buid <- (buids(0) \ "buids").as[List[String]]) {
        for (poi <- poisByBuildingAsJson(buid)) {
          if (poi != null) {
            tempPois.add(poi)
          }
        }
      }
      pois = tempPois
    }
    val pois2 = new java.util.ArrayList[JsValue]
    val words = letters.split(" ")
    var flag = false
    var flag2 = false
    var flag3 = false

    if (letters.compareTo(lastletters) != 0) {
      lastletters = letters
      wordsELOT = new java.util.ArrayList[java.util.ArrayList[String]]
      var j = 0
      for (word <- words) {
        wordsELOT.add(greeklishTogreekList(word.toLowerCase))
      }
    }
    for (json <- pois) {
      flag = true
      flag2 = true
      flag3 = true
      var j = 0
      // create a Breaks object as follows
      val loop = new Breaks

      val ex_loop = new Breaks
      ex_loop.breakable {
        for (w <- words) {
          var name = ""
          var description = ""
          if ((json \ "name").toOption.isDefined)
            name = (json \ "name").as[String].toLowerCase
```

```
          if ((json \ "description").toOption.isDefined)
            description = (json \
"description").as[String].toLowerCase
          if (!(name.contains(w.toLowerCase) ||
description.contains(w.toLowerCase))) flag = false
          val greeklish = greeklishTogreek(w.toLowerCase)
          if (!(name.contains(greeklish) ||
description.contains(greeklish))) flag2 = false
          if (wordsELOT.size != 0) {
            var wordsELOT2 = new java.util.ArrayList[String]
            wordsELOT2 = wordsELOT.get({
              j += 1
              j - 1
            })
            if (wordsELOT2.size == 0) flag3 = false
            else {
              loop.breakable {
                for (greeklishELOT <- wordsELOT2) {
                  if (!(name.contains(greeklishELOT) ||
description.contains(greeklishELOT))) flag3 = false
                  else {
                    flag3 = true
                    loop.break()
                  }
                }
              }
            }
          }
          else flag3 = false
          if (!flag3 && !flag && !flag2) ex_loop.break()
          if (flag || flag2 || flag3) pois2.add(json)
        }
      }
    }

    pois2.toList
  }

  override def poisByBuildingAsJson3(buid: String, letters: String):
List[JsValue] = {
    LPLogger.info("poisByBuildingAsJson3")
    var pois: java.util.List[JsValue] = null
    val pois2 = new java.util.ArrayList[JsValue]
    if (allPoisSide.get(buid) != null)
      pois = allPoisSide.get(buid)
    else
      pois = poisByBuildingAsJson(buid)

    val words = letters.split(" ")

    var flag = false
    var flag2 = false
    var flag3 = false

    if (letters.compareTo(lastletters) != 0) {
      lastletters = letters
      wordsELOT = new java.util.ArrayList[java.util.ArrayList[String]]
      var j = 0
      for (word <- words) {
        wordsELOT.add(greeklishTogreekList(word.toLowerCase))
```

```
      }
    }
    for (json <- pois) {
      flag = true
      flag2 = true
      flag3 = true
      var j = 0
      // create a Breaks object as follows
      val loop = new Breaks

      val ex_loop = new Breaks
      ex_loop.breakable {
        for (w <- words) {
          var name = ""
          var description = ""
          if ((json \ "name").toOption.isDefined)
            name = (json \ "name").as[String].toLowerCase
          if ((json \ "description").toOption.isDefined)
            description = (json \
"description").as[String].toLowerCase
          if (!(name.contains(w.toLowerCase) ||
description.contains(w.toLowerCase))) flag = false
          val greeklish = greeklishTogreek(w.toLowerCase)
          if (!(name.contains(greeklish) ||
description.contains(greeklish))) flag2 = false
          if (wordsELOT.size != 0) {
            var wordsELOT2 = new java.util.ArrayList[String]
            wordsELOT2 = wordsELOT.get({
              j += 1
              j - 1
            })
            if (wordsELOT2.size == 0) flag3 = false
            else {
              loop.breakable {
                for (greeklishELOT <- wordsELOT2) {
                  if (!(name.contains(greeklishELOT) ||
description.contains(greeklishELOT))) flag3 = false
                  else {
                    flag3 = true
                    loop.break()
                  }
                }
              }
            }
          }
          else flag3 = false
          if (!flag3 && !flag && !flag2) ex_loop.break()
          if (flag || flag2 || flag3) pois2.add(json)
        }
      }
    }

    pois2.toList
  }

  override def init(): Boolean = ???

  def addJsonDocument(key: String, expiry: Int, document: String):
Boolean = ???
```

```
  override def addJsonDocument(col: String, document: String): Boolean
= {
    val collection = mdb.getCollection(col)
    val addJson = collection.insertOne(Document.apply(document))
    val awaited = Await.result(addJson.toFuture, Duration.Inf)
    val res = awaited
    if (res.toString() == "The operation completed successfully")
      true
    else
      false
  }

  override def replaceJsonDocument(key: String, expiry: Int, document:
String): Boolean = ???

  override def replaceJsonDocument(col: String, key: String, value:
String, document: String): Boolean = {
    val collection = mdb.getCollection(col)
    val query = BsonDocument(key -> value)
    val update = BsonDocument(document)
    val replaceJson = collection.replaceOne(query, update)
    val awaited = Await.result(replaceJson.toFuture, Duration.Inf)
    val res = awaited
    if (res.getModifiedCount == 0)
      false
    else
      true
  }

  override def deleteFromKey(key: String): Boolean = ???

  override def deleteFromKey(col: String, key: String, value: String):
Boolean = {
    val collection = mdb.getCollection(col)
    val query = BsonDocument(key -> value)
    val deleted = collection.deleteOne(query)
    val awaited = Await.result(deleted.toFuture, Duration.Inf)
    val res = awaited
    res.wasAcknowledged()
  }

  override def getFromKey(key: String) = ???

  override def getFromKey(col: String, key: String, value: String):
JsValue = {
    val collection = mdb.getCollection(col)
    val buildingLookUp = collection.find(equal(key, value)).first()
    val awaited = Await.result(buildingLookUp.toFuture, Duration.Inf)
    val res = awaited.asInstanceOf[Document]
    if (res != null)
      convertJson(res)
    else
      null
  }

  override def getFromKeyAsJson(key: String) = ???

  override def getFromKeyAsJson(collection: String, key: String,
value: String): JsValue = {
    if (key == null || key.trim().isEmpty) {
```

```
      throw new IllegalArgumentException("No null or empty string
allowed as key!")
    }
    val db_res = getFromKey(collection, key, value)
    if (db_res == null) {
      return null
    }
    try {
      db_res
    } catch {
      case e: IOException => {
        LPLogger.error("CouchbaseDatasource::getFromKeyAsJson()::
Could not convert document from Couchbase into JSON!")
        null
      }
    }
  }

  override def fingerprintExists(col: String, buid: String, floor:
String, x: String, y: String, heading: String): Boolean = {
    val collection = mdb.getCollection(col)
    val query = BsonDocument("buid" -> buid, "floor" -> floor, "x" ->
x, "y" -> y, "heading" -> heading)
    val fingerprintLookUp = collection.find(query)
    val awaited = Await.result(fingerprintLookUp.toFuture,
Duration.Inf)
    val res = awaited.asInstanceOf[List[Document]]
    if (res.size > 0)
      return true
    return false
  }

  override def buildingFromKeyAsJson(value: String): JsValue = {
    var building = getFromKeyAsJson("buildings", "buid", value)
    if (building == null) {
      return null
    }
    building = building.as[JsObject] - "_id" - "_schema"
    val floors = new java.util.ArrayList[JsValue]()
    for (f <- floorsByBuildingAsJson(value)) {
      floors.add(f)
    }

    // ignore pois with pois_type = "None"
    val pois = new java.util.ArrayList[JsValue]()
    for (p <- poisByBuildingAsJson(value)) {
      if (!(p \
"pois_type").toString.equalsIgnoreCase(Poi.POIS_TYPE_NONE))
        pois.add(p)
    }
    building.as[JsObject] + ("floors" -> Json.toJson(floors.toList)) +
      ("pois" -> Json.toJson(pois.toList))

  }

  override def poiFromKeyAsJson(collection: String, key: String,
value: String): JsValue = {
    getFromKeyAsJson(collection, key, value)
  }
```

```
  override def poisByBuildingFloorAsJson(buid: String, floor_number:
String): List[JsValue] = {
    val collection = mdb.getCollection("pois")
    val query = BsonDocument("buid" -> buid, "floor_number" ->
floor_number)
    val pois = collection.find(query)
    val awaited = Await.result(pois.toFuture, Duration.Inf)
    val res = awaited.toList
    val listJson = convertJson(res)
    val poisArray = new java.util.ArrayList[JsValue]()
    for (poi <- listJson)
      poisArray.add(poi.as[JsObject] - "owner_id" - __geometry - "_id"
- "_schema")
    poisArray.toList
  }

  override def poisByBuildingFloorAsMap(buid: String, floor_number:
String): java.util.List[java.util.HashMap[String, String]] = {
    val pois = poisByBuildingFloorAsJson(buid, floor_number)
    val result = new java.util.ArrayList[java.util.HashMap[String,
String]]()
    for (pois <- pois) {
      result.add(JsonUtils.getHashMapStrStr(pois))
    }
    result
  }

  override def poisByBuildingAsJson(buid: String):
java.util.List[JsValue] = {
    val collection = mdb.getCollection("pois")
    val poisLookUp = collection.find(equal("buid", buid))
    val awaited = Await.result(poisLookUp.toFuture, Duration.Inf)
    val res = awaited.toList
    val listJson = convertJson(res)
    val pois = new java.util.ArrayList[JsValue]()
    for (floor <- listJson) {
      pois.add(floor.as[JsObject] - "owner_id" - __geometry - "_id" -
"_schema")
    }
    pois
  }

  override def poiByBuidFloorPuid(buid: String, floor_number: String,
puid: String): Boolean = {
    val collection = mdb.getCollection("pois")
    val query = BsonDocument("buid" -> buid, "floor_number" ->
floor_number, "puid" -> puid)
    val poisLookUp = collection.find(query)
    val awaited = Await.result(poisLookUp.toFuture, Duration.Inf)
    val res = awaited.toList
    if (res.size > 0)
      return true
    false
  }

  override def poisByBuildingAsMap(buid: String):
java.util.List[java.util.HashMap[String, String]] = {
    val pois = poisByBuildingAsJson(buid)
    val result = new java.util.ArrayList[java.util.HashMap[String,
String]]()
```

```
    for (poi <- pois) {
      result.add(JsonUtils.getHashMapStrStr(poi))
    }
    result
  }

  override def floorsByBuildingAsJson(buid: String):
java.util.List[JsValue] = {
    val collection = mdb.getCollection("floorplans")
    val floorLookUp = collection.find(equal("buid", buid))
    val awaited = Await.result(floorLookUp.toFuture, Duration.Inf)
    val res = awaited.toList
    val listJson = convertJson(res)
    val floors = new java.util.ArrayList[JsValue]()
    for (floor <- listJson) {
      try {
        floors.add(floor.as[JsObject] - "owner_id" - "_id" -
"_schema")
      } catch {
        case e: IOException =>
      }
    }
    floors
  }

  override def connectionsByBuildingAsJson(buid: String):
List[JsValue] = {
    val collection = mdb.getCollection("edges")
    val query = BsonDocument("buid" -> buid)
    val edges = collection.find(query)
    val awaited = Await.result(edges.toFuture, Duration.Inf)
    val res = awaited.toList
    convertJson(res)
  }

  override def connectionsByBuildingAsMap(buid: String):
java.util.List[java.util.HashMap[String, String]] = {

    val edges = connectionsByBuildingAsJson(buid)
    var hm: util.HashMap[String, String] = null
    val conns = new util.ArrayList[util.HashMap[String, String]]()
    for (edge <- edges) {
      hm = JsonUtils.getHashMapStrStr(edge)
      if
(!hm.get("edge_type").equalsIgnoreCase(Connection.EDGE_TYPE_OUTDOOR))
        conns.add(hm)
    }
    conns
  }

  override def connectionsByBuildingFloorAsJson(buid: String,
floor_number: String): List[JsValue] = {
    val collection = mdb.getCollection("edges")
    val query = BsonDocument("buid" -> buid, "floor_a" ->
floor_number, "floor_b" -> floor_number)
    val edges = collection.find(query)
    val awaited = Await.result(edges.toFuture, Duration.Inf)
    val res = awaited.toList
    val listJson = convertJson(res)
    val edgesArray = new java.util.ArrayList[JsValue]()
```

```
    for (poi <- listJson)
      edgesArray.add(poi.as[JsObject] - "owner_id" - "_id" -
"_schema")
    edgesArray.toList
  }

  override def connectionsByBuildingAllFloorsAsJson(buid: String):
List[JsValue] = {
    val collection = mdb.getCollection("edges")
    val query = BsonDocument("buid" -> buid)
    val edges = collection.find(query)
    val awaited = Await.result(edges.toFuture, Duration.Inf)
    val res = awaited.toList
    val listJson = convertJson(res)
    val edgesArray = new java.util.ArrayList[JsValue]()
    for (poi <- listJson)
      edgesArray.add(poi.as[JsObject] - "owner_id" - "_id" -
"_schema")
    edgesArray.toList
  }


  override def deleteAllByBuilding(buid: String): Boolean = {
    LPLogger.debug("Cascading building " + buid)
    var ret = true

    // deleting floors along with pois, edges
    val floors = floorsByBuildingAsJson(buid)
    for (floor <- floors) {
      if ((floor \ "floor_number").toOption.isDefined) {
        val tempBool = deleteAllByFloor(buid, (floor \
"floor_number").as[String])
        ret = ret && tempBool
      }
    }

    // deleting building
    var query = BsonDocument("buid" -> buid)
    var collection = mdb.getCollection("buildings")
    val objects = collection.deleteOne(query)
    val awaited = Await.result(objects.toFuture, Duration.Inf)
    val res = awaited
    val bool = res.wasAcknowledged()

    // deleting buid from campus buids[]
    query = BsonDocument("buids" -> buid)
    collection = mdb.getCollection("campuses")
    val campuses = collection.find(query)
    var await = Await.result(campuses.toFuture, Duration.Inf)
    var re = await.toList
    val camp = convertJson(re)
    LPLogger.debug("camp = " + camp)
    for (c <- camp) {
      val newBuids: util.ArrayList[String] = new
util.ArrayList[String]()
      val buids = (c \ "buids").as[List[String]]
      for (buid2 <- buids) {
        if (buid2 != buid)
          newBuids.add(buid2)
      }
```

```
      //      LPLogger.debug("OldBuids = " + buids)
      //      LPLogger.debug("newBuids = " + newBuids)
      val newCamp: String = (c.as[JsObject] + ("buids" ->
Json.toJson(newBuids.toList))).toString()
      ret = ret && replaceJsonDocument("campuses", "cuid", (c \
"cuid").as[String], newCamp)
    }

    // deleting fingerprints
    // requires testing
    query = BsonDocument("buid" -> buid)
    collection = mdb.getCollection("fingerprintsWifi")
    val deleted = collection.deleteMany(query)
    val delAwaited = Await.result(deleted.toFuture, Duration.Inf)
    val delRes = delAwaited
    val bool1 = delRes.wasAcknowledged()
    LPLogger.debug("fingerprints with buid " + buid + " " +
delRes.toString)
    ret = ret && bool1

    (ret && bool)
  }

  override def deleteAllByFloor(buid: String, floor_number: String):
Boolean = {
    LPLogger.debug("deleteAllByFloor::")
    LPLogger.debug("Cascade deletion: edges reaching this floor, edges
of this floor," +
      " pois of this floor, floorplan(mongoDB), floorplan
image(server)")
    var collection = mdb.getCollection("edges")

    // queryBuidA deletes edges that start from building buid
(containing edges that have buid_b == buid_a)
    val queryBuidA = BsonDocument("buid_a" -> buid, "floor_a" ->
floor_number)
    var deleted = collection.deleteMany(queryBuidA)
    var awaited = Await.result(deleted.toFuture, Duration.Inf)
    var res = awaited
    val bool1 = res.wasAcknowledged()
    LPLogger.debug("edges from buid_a: " + buid + " " + res.toString)

    // queryBuidB deletes edges that point to building buid
    val queryBuidB = BsonDocument("buid_b" -> buid, "floor_b" ->
floor_number)
    deleted = collection.deleteMany(queryBuidB)
    awaited = Await.result(deleted.toFuture, Duration.Inf)
    res = awaited
    val bool2 = res.wasAcknowledged()
    LPLogger.debug("edges to buid_b: " + buid + " " + res.toString)

    // this query will delete the pois of the floor
    val queryFloor = BsonDocument("buid" -> buid, "floor_number" ->
floor_number)
    collection = mdb.getCollection("pois")
    deleted = collection.deleteMany(queryFloor)
    awaited = Await.result(deleted.toFuture, Duration.Inf)
    res = awaited
    val bool3 = res.wasAcknowledged()
    LPLogger.debug("pois in building with buid: " + buid + " " +
```

```
res.toString)

    // this query will delete the floor it self
    collection = mdb.getCollection("floorplans")
    deleted = collection.deleteMany(queryFloor)
    awaited = Await.result(deleted.toFuture, Duration.Inf)
    res = awaited
    val bool4 = res.wasAcknowledged()
    LPLogger.debug("floorplan with buid " + buid + " " + res.toString)
    bool1 && bool2 && bool3 && bool4
  }

  override def deleteAllByConnection(cuid: String):
java.util.List[String] = {
    val all_items_failed = new util.ArrayList[String]()
    if (!this.deleteFromKey("edges", "cuid", cuid)) {
      all_items_failed.add(cuid)
    }
    all_items_failed
  }

  // ASK:PM
  override def deleteAllByPoi(puid: String): java.util.List[String] =
{
    val all_items_failed = new util.ArrayList[String]()
    if (!this.deleteFromKey("edges", "pois_a", puid)) {
      all_items_failed.add(puid)
    }
    if (!this.deleteFromKey("edges", "pois_b", puid)) {
      all_items_failed.add(puid)
    }
    if (!this.deleteFromKey("pois", "puid", puid)) {
      all_items_failed.add(puid)
    }
    all_items_failed
  }

  override def getRadioHeatmap(): java.util.List[JsonObject] = ???

  override def getRadioHeatmapByBuildingFloor(buid: String, floor:
String): List[JsValue] = {
    val collection = mdb.getCollection("heatmapWifi3")
    val query = BsonDocument("buid" -> buid, "floor" -> floor)
    val radioPoints = collection.aggregate(Seq(
      Aggregates.filter(query),
      project(
        Document("location" -> "$location", "w" -> "$count")
      )))
    val awaited = Await.result(radioPoints.toFuture, Duration.Inf)
    val res = awaited.toList
    return convertJson(res)
  }

  override def getRadioHeatmapByBuildingFloorAverage1(buid: String,
floor: String): List[JsValue] = {
    val collection = mdb.getCollection("heatmapWifi1")
    val query = BsonDocument("buid" -> buid, "floor" -> floor)
    val radioPoints = collection.aggregate(Seq(
      Aggregates.filter(query),
      project(
```

```
        Document("location" -> "$location", "sum" -> "$sum", "count" -
> "$count")
      )))
    val awaited = Await.result(radioPoints.toFuture, Duration.Inf)
    val res = awaited.toList

    val heatmaps = new util.ArrayList[JsValue]()
    for (heatmap <- convertJson(res)) {
      val count = (heatmap \ "count").as[Int]
      val sum = (heatmap \ "sum").as[Int]
      val average = sum.toDouble / count.toDouble
      heatmaps.add(Json.obj("location" -> (heatmap \ "location" \
"coordinates").as[List[Double]],
        "count" -> count, "sum" -> sum, "average" -> average))
    }
    return heatmaps.toList
  }

  override def getRadioHeatmapByBuildingFloorAverage2(buid: String,
floor: String): List[JsValue] = {
    val collection = mdb.getCollection("heatmapWifi2")
    val query = BsonDocument("buid" -> buid, "floor" -> floor)
    val radioPoints = collection.aggregate(Seq(
      Aggregates.filter(query),
      project(
        Document("location" -> "$location", "sum" -> "$sum", "count" -
> "$count", "average" -> "$average")
      )))
    val awaited = Await.result(radioPoints.toFuture, Duration.Inf)
    val res = awaited.toList

    val heatmaps = new util.ArrayList[JsValue]()
    for (heatmap <- convertJson(res)) {
      val count = (heatmap \ "count").as[Int]
      val sum = (heatmap \ "sum").as[Int]
      val average = sum.toDouble / count.toDouble
      heatmaps.add(Json.obj("location" -> (heatmap \ "location" \
"coordinates").as[List[Double]],
        "count" -> count, "sum" -> sum, "average" -> average))
    }
    return heatmaps.toList
  }

  override def getRadioHeatmapByBuildingFloorAverage3(buid: String,
floor: String): List[JsValue] = {
    val collection = mdb.getCollection("heatmapWifi3")
    val query = BsonDocument("buid" -> buid, "floor" -> floor)
    val radioPoints = collection.aggregate(Seq(
      Aggregates.filter(query),
      project(
        Document("location" -> "$location", "sum" -> "$sum", "count" -
> "$count", "average" -> "$average")
      )))
    val awaited = Await.result(radioPoints.toFuture, Duration.Inf)
    val res = awaited.toList

    val heatmaps = new util.ArrayList[JsValue]()
    for (heatmap <- convertJson(res)) {
      val count = (heatmap \ "count").as[Int]
      val sum = (heatmap \ "sum").as[Int]
```

```
      val average = sum.toDouble / count.toDouble
      heatmaps.add(Json.obj("location" -> (heatmap \ "location" \
"coordinates").as[List[Double]],
        "count" -> count, "sum" -> sum, "average" -> average))
    }
    return heatmaps.toList
  }

  override def getRadioHeatmapByBuildingFloorTimestamp(buid: String,
floor: String, timestampX: String, timestampY: String): List[JsValue]
= {
    val collection = mdb.getCollection("heatmapWifiTimestamp3")
    val radioPoints = collection.aggregate(Seq(
      Aggregates.filter(and(gt("timestamp", timestampX),
lt("timestamp", timestampY),
        equal("buid", buid), equal("floor", floor))),
      project(
        Document("location" -> "$location.coordinates", "w" ->
"$count")
      )))
    val awaited = Await.result(radioPoints.toFuture, Duration.Inf)
    val res = awaited.toList
    return convertJson(res)
  }

  override def getRadioHeatmapByBuildingFloorTimestampAverage1(buid:
String, floor: String, timestampX: String, timestampY: String):
List[JsValue] = {
    val collection = mdb.getCollection("heatmapWifiTimestamp1")
    val radioPoints = collection.aggregate(Seq(
      Aggregates.filter(and(gt("timestamp", timestampX),
lt("timestamp", timestampY),
        equal("buid", buid), equal("floor", floor))),
      project(
        Document("location" -> "$location.coordinates", "w" ->
"$count")
      )))
    val awaited = Await.result(radioPoints.toFuture, Duration.Inf)
    val res = awaited.toList
    return convertJson(res)
  }

  override def getRadioHeatmapByBuildingFloorTimestampAverage2(buid:
String, floor: String, timestampX: String, timestampY: String):
List[JsValue] = {
    val collection = mdb.getCollection("heatmapWifiTimestamp2")
    val radioPoints = collection.aggregate(Seq(
      Aggregates.filter(and(gt("timestamp", timestampX),
lt("timestamp", timestampY),
        equal("buid", buid), equal("floor", floor))),
      project(
        Document("location" -> "$location.coordinates", "w" ->
"$count")
      )))
    val awaited = Await.result(radioPoints.toFuture, Duration.Inf)
    val res = awaited.toList
    return convertJson(res)
  }

  override def getAPsByBuildingFloorcdb(buid: String, floor: String):
```

```
util.List[JsonObject] = ???

  override def getAPsByBuildingFloor(buid: String, floor: String):
List[JsValue] = {
    val collection = mdb.getCollection("fingerprintsWifi")
    val query = BsonDocument("buid" -> buid, "floor" -> floor)
    val fingerprints = collection.find(query)
    val awaited = Await.result(fingerprints.toFuture, Duration.Inf)
    val res = awaited.toList
    val listJson = convertJson(res)
    val hm = new util.HashMap[JsValue, Array[Double]]()
    for (f <- listJson) {
      val measurements = (f \ "measurements").as[List[List[String]]]
      for (measurement <- measurements) {
        val json = unrollFingerprint(f, measurement)
        val key = Json.obj("x" -> (json \ "x").as[String], "y" ->
(json \ "y").as[String],
          "AP" -> (json \ "MAC").as[String])
        if (!hm.containsKey(key)) {
          // count / average / total
          val tArray = new Array[Double](3)
          tArray(0) = 1
          tArray(1) = (json \ "rss").as[String].toDouble
          tArray(2) = (json \ "rss").as[String].toDouble
          hm.put(key, tArray)
        } else {
          val tArray = hm.get(key)
          tArray(0) += 1
          tArray(1) = (tArray(1) / tArray(0))
          tArray(2) += hm.get(key)(2)
          hm.put(key, tArray)
        }
      }
    }
    val points = new util.ArrayList[JsValue]()
    for (h <- hm.toMap.toList) {
      var tempJson: JsValue = h._1
      val rss = Json.obj("count" -> JsNumber(h._2(0)),
        "average" -> JsNumber(h._2(2)),
        "total" -> JsNumber(h._2(1)))
      tempJson = tempJson.as[JsObject] + ("RSS" -> rss)
      if ((rss \ "average").as[Double] >= -70)
        points.add(tempJson.as[JsObject])
    }
    points.toList
  }

  override def getCachedAPsByBuildingFloor(buid: String, floor:
String): JsValue = {
    val collection = mdb.getCollection("accessPointsWifi")
    val query = BsonDocument("buid" -> buid, "floor" -> floor)
    val accessPointsLookup = collection.find(query)
    val awaited = Await.result(accessPointsLookup.toFuture,
Duration.Inf)
    val res = awaited.toList
    if (res.size == 0)
      return null
    val json = convertJson(res(0))
    return json
  }
```

```
  override def deleteAllByXsYs(buid: String, floor: String, x: String,
y: String): java.util.List[String] = ???

  override def getFingerPrintsBBox(buid: String, floor: String, lat1:
String, lon1: String, lat2: String, lon2: String): List[JsValue] = {
    val collection = mdb.getCollection("fingerprintsWifi")
    val fingerprints = collection.find(and(
      geoWithinBox("geometry", lat1.toDouble, lon1.toDouble,
lat2.toDouble, lon2.toDouble),
      equal("buid", buid),
      equal("floor", floor)))
    val awaited = Await.result(fingerprints.toFuture, Duration.Inf)
    val res = awaited.toList
    val listJson = convertJson(res)
    val newList = new util.ArrayList[JsValue]()
    for (f <- listJson) {
      if ((f \ "buid").as[String] == buid && (f \ "floor").as[String]
== floor) {
        newList.add(f)
      }
    }
    newList.toList
  }

  override def getFingerPrintsTimestampBBox(buid: String, floor:
String, lat1: String, lon1: String, lat2: String,
                                            lon2: String, timestampX:
String, timestampY: String): List[JsValue] = {
    val collection = mdb.getCollection("fingerprintsWifi")
    val fingerprints = collection.find(and(geoWithinBox("geometry",
lat1.toDouble, lon1.toDouble,
      lat2.toDouble, lon2.toDouble), and(gt("timestamp", timestampX),
lt("timestamp", timestampY))))
    val awaited = Await.result(fingerprints.toFuture, Duration.Inf)
    val res = awaited.toList
    val listJson = convertJson(res)
    val newList = new util.ArrayList[JsValue]()
    for (f <- listJson) {
      if ((f \ "buid").as[String] == buid && (f \ "floor").as[String]
== floor) {
        newList.add(f)
      }
    }
    newList.toList
  }

  override def getFingerPrintsTime(buid: String, floor: String):
List[JsValue] = {
    val collection = mdb.getCollection("fingerprintsWifi")
    val fingerprints = collection.find(and(and(gt("timestamp",
"000000000000000"),
      lt("timestamp", "999999999999999")), and(equal("buid", buid)),
equal("floor", floor)))
    val awaited = Await.result(fingerprints.toFuture, Duration.Inf)
    val res = awaited.toList
    LPLogger.debug("res = " + res.size)
    val listJson = convertJson(res)
    val points = new util.ArrayList[JsValue]()
    for (f <- listJson) {
```

```
      points.add(Json.obj("date" -> (f \ "timestamp").as[String],
        "count" -> (f \ "count").as[Int]))
    }
    points.toList
  }

  override def getRadioHeatmapByBuildingFloor2(lat: String, lon:
String, buid: String, floor: String, range: Int):
java.util.List[JsonObject] = ???

  override def getRadioHeatmapBBox(lat: String, lon: String, buid:
String, floor: String, range: Int): java.util.List[JsonObject] = ???

  override def getRadioHeatmapBBox2(lat: String, lon: String, buid:
String, floor: String, range: Int): java.util.List[JsonObject] = ???


  override def getAllBuildings(): List[JsValue] = {
    val collection = mdb.getCollection("buildings")
    val query = BsonDocument("is_published" -> "true")
    val buildings = collection.find(query)
    val awaited = Await.result(buildings.toFuture, Duration.Inf)
    val res = awaited.toList
    LPLogger.debug(s"Res on complete Length:${res.length}")
    val listJson = convertJson(res)
    var newJson: JsValue = null
    val newList = new util.ArrayList[JsValue]()
    for (x <- listJson) {
      newJson = x.as[JsObject] - __geometry - "owner_id" - "co_owners"
- "_id"
      newList.add(newJson)
    }
    newList.toList
  }


  override def getAllBuildingsByOwner(oid: String): List[JsValue] = {
    val collection = mdb.getCollection("buildings")
    var buildingLookUp = collection.find(or(equal("owner_id", oid),
      equal("co_owners", oid)))
    if (admins.contains(oid)) {
      buildingLookUp = collection.find()
    }
    val awaited = Await.result(buildingLookUp.toFuture, Duration.Inf)
    val res = awaited.toList
    val listJson = convertJson(res)
    val buildings = new java.util.ArrayList[JsValue]()
    for (building <- listJson) {
      buildings.add(building.as[JsObject] - "co_owners" - __geometry
        - "owner_id" - "_id" - "_schema")
    }
    buildings.toList
  }


  override def getAllBuildingsByBucode(bucode: String): List[JsValue]
= {
    val collection = mdb.getCollection("buildings")
    val buildingLookUp = collection.find(equal("bucode", bucode))
    val awaited = Await.result(buildingLookUp.toFuture, Duration.Inf)
```

```
    val res = awaited.toList
    val listJson = convertJson(res)
    val buildings = new java.util.ArrayList[JsValue]()
    for (building <- listJson) {
      try {
        buildings.add(building.as[JsObject] - "co_owners" - __geometry
- "owner_id" - "_id" - "_schema")
      } catch {
        case e: IOException =>
      }
    }
    buildings.toList
  }

  override def getBuildingByAlias(alias: String): JsonObject = ???

  override def getAllBuildingsNearMe(lat: Double, lng: Double, range:
Int, owner_id: String): List[JsValue] = {
    val bbox = GeoPoint.getGeoBoundingBox(lat, lng, range)
    val collection = mdb.getCollection("buildings")
    val buildingLookUp = collection.find(and(geoWithinBox("geometry",
bbox(0).dlat, bbox(0).dlon, bbox(1).dlat,
      bbox(1).dlon),
      or(equal("is_published", "true"),
        and(equal("is_published", "false"), equal("owner_id",
owner_id)))))
    val awaited = Await.result(buildingLookUp.toFuture, Duration.Inf)
    val res = awaited.toList
    LPLogger.debug("getAllBuildingsNearMe: fetched " + res.size + "
building(s) within a range of: " + range)
    val listJson = convertJson(res)
    val buildings = new java.util.ArrayList[JsValue]()
    for (building <- listJson) {
      try {
        buildings.add(building.as[JsObject] - "co_owners" - __geometry
- "owner_id" - "_id")
      } catch {
        case e: IOException =>
      }
    }
    buildings.toList
  }

  override def dumpRssLogEntriesSpatial(outFile: FileOutputStream,
bbox: Array[GeoPoint], floor_number: String): Long = {
    val writer = new PrintWriter(outFile)
    val floorLimit = 100000
    val queryLimit = 5000
    var totalFetched = 0
    var floorFetched = 0
    val collection = mdb.getCollection("fingerprintsWifi")
    val fingerprints = collection.find(geoWithinBox("geometry",
bbox(0).dlat, bbox(0).dlon, bbox(1).dlat,
      bbox(1).dlon)).limit(queryLimit)
    val awaited = Await.result(fingerprints.toFuture, Duration.Inf)
    val res = awaited.toList
    val listJson = convertJson(res)
    for (rss <- listJson) {
      if (floorFetched > floorLimit)
        break
```

```
      totalFetched += 1
      if ((rss \ "floor").as[String] == floor_number) {
        floorFetched += 1
        val measurements = (rss \
"measurements").as[List[List[String]]]
        for (measurement <- measurements) {

writer.println(RadioMapRaw.toRawRadioMapRecord(unrollFingerprint(rss,
measurement)))
        }
      }
    }
    LPLogger.info("total fetched: " + totalFetched)

    writer.flush()
    writer.close()

    floorFetched
  }

  override def dumpRssLogEntriesWithCoordinates(floor_number: String,
lat: Double, lon: Double): String = {
    val collection = mdb.getCollection("floorplans")
    val query = BsonDocument("floor_number" -> floor_number)
    val floorLookUp = collection.find(query)
    val awaited = Await.result(floorLookUp.toFuture, Duration.Inf)
    val res = awaited.asInstanceOf[List[Document]]
    val floorplans = convertJson(res)
    var unique = 0
    var uniqBuid = ""
    for (floorplan <- floorplans) {
      val lat1 = (floorplan \ fLatBottomLeft)
      val lon1 = (floorplan \ "bottom_left_lng")
      val lat2 = (floorplan \ "top_right_lat")
      val lon2 = (floorplan \ "top_right_lng")
      if (lat1.toOption.isDefined && lon1.toOption.isDefined &&
lat2.toOption.isDefined && lon2.toOption.isDefined) {
        if (lat1.as[String].toDouble <= lat && lat <=
lat2.as[String].toDouble &&
          lon1.as[String].toDouble <= lon && lon <=
lon2.as[String].toDouble) {
          unique += 1
          uniqBuid = (floorplan \ "buid").as[String]
        }
      }
    }
    if (unique == 1)
      return uniqBuid
    LPLogger.error("Coordinates were found in two floors.")
    return null
  }

  override def dumpRssLogEntriesByBuildingFloor(outFile:
FileOutputStream, buid: String, floor_number: String): Long = {
    val writer = new PrintWriter(outFile)
    var totalFetched = 0

    val collection = mdb.getCollection("fingerprintsWifi")
    val query = BsonDocument("buid" -> buid, "floor" -> floor_number)
    val fingerprintLookUp = collection.find(query)
```

```
    val awaited = Await.result(fingerprintLookUp.toFuture,
Duration.Inf)
    val res = awaited.asInstanceOf[List[Document]]
    val rssLog = convertJson(res)
    // splitting Measurements[MAC, rss] to buid, floor, .., MAC, rss,
... (old form)
    if (rssLog.size > 0) {
      for (rss <- rssLog) {
        val measurements = (rss \
"measurements").as[List[List[String]]]
        for (measurement <- measurements) {

writer.println(RadioMapRaw.toRawRadioMapRecord(unrollFingerprint(rss,
measurement)))
          totalFetched += 1
          if (totalFetched == 10000) {
            writer.flush()
            writer.close()
            return totalFetched
          }
        }
      }
    }
    writer.flush()
    writer.close()
    totalFetched
  }

  override def dumpRssLogEntriesByBuildingACCESFloor(outFile:
FileOutputStream, buid: String, floor_number: String): Long = ???

  override def getAllAccounts(): List[JsValue] = {
    val collection = mdb.getCollection("users")
    val users = collection.find()
    val awaited = Await.result(users.toFuture, Duration.Inf)
    val res = awaited.toList
    LPLogger.debug(s"Res on complete Length:${res.length}")
    val usersList = convertJson(res)
    val ret = new util.ArrayList[JsValue]()
    for (user <- usersList) {
      ret.add(cleanupMongoJson(user))
    }
    ret.toList
  }

  override def predictFloor(algo: IAlgo, bbox: Array[GeoPoint],
strongestMACs: Array[String]): Boolean = ???

  override def deleteRadiosInBox(): Boolean = ???

  override def magneticPathsByBuildingFloorAsJson(buid: String,
floor_number: String): java.util.List[JsonObject] = ???

  override def magneticPathsByBuildingAsJson(buid: String):
java.util.List[JsonObject] = ???

  override def magneticMilestonesByBuildingFloorAsJson(buid: String,
floor_number: String): java.util.List[JsonObject] = ???

  override def BuildingSetsCuids(cuid: String): Boolean = {
```

```
    if (getBuildingSet(cuid).size > 1)
      return true
    false
  }

  override def getBuildingSet(cuid: String): List[JsValue] = {
    val collection = mdb.getCollection("campuses")
    val query: BsonDocument = BsonDocument("cuid" -> cuid)
    val campus = collection.find(query)
    val awaited = Await.result(campus.toFuture, Duration.Inf)
    val res = awaited.toList
    convertJson(res)
  }

  override def getAllBuildingsetsByOwner(owner_id: String):
List[JsValue] = {
    val collection = mdb.getCollection("campuses")
    val query: BsonDocument = BsonDocument("owner_id" -> owner_id)
    val campus = collection.find(query)
    val awaited = Await.result(campus.toFuture, Duration.Inf)
    val res = awaited.toList
    convertJson(res)
  }

  def trimCoordinates(fingerprint: JsValue, level: Int):
util.ArrayList[Double] = {
    if ((fingerprint \ "geometry" \ "coordinates").toOption.isEmpty) {
      LPLogger.debug("trimCoordinates: " + fingerprint + ".No field
coordinates.")
      return null
    }
    val location = new util.ArrayList[Double]
    try {
      val lat = (fingerprint \ "geometry" \
"coordinates").get(0).as[Double]
      val lon = (fingerprint \ "geometry" \
"coordinates").get(1).as[Double]
      if (level == 1) {
        location.add(lat.toString.dropRight(5).toDouble)
        location.add(lon.toString.dropRight(5).toDouble)
      } else if (level == 2) {
        location.add(lat.toString.dropRight(4).toDouble)
        location.add(lon.toString.dropRight(4).toDouble)
      } else if (level == 3) {
        location.add(lat)
        location.add(lon)
      }
      return location
    } catch {
      case e: Exception => LPLogger.debug("fingerprint" + fingerprint)
    }
    null
  }

  def createHeatmap(fingerprint: JsValue, level: Int, timestamp:
Boolean): JsValue = {
    var heatmap: JsValue = fingerprint.as[JsObject] - "_id" -
"strongestWifi" - "heading" - "x" - "y" - __geometry -
      "measurements"
    if (!timestamp)
```

```
      heatmap = heatmap.as[JsObject] - "timestamp"
    val location = trimCoordinates(fingerprint, level)
    if (location == null) return null
    heatmap = heatmap.as[JsObject] + ("location" -> Json.toJson(new
GeoJSONPoint(location.get(0),
      location.get(1)).toGeoJSON()))
    return heatmap
  }

  def fetchStoredHeatmap(collection: String, fingerprint: JsValue,
level: Int, timestamp: Boolean): JsValue = {
    val heatmap = mdb.getCollection(collection)
    val location = trimCoordinates(fingerprint, level)
    if (location == null) return null
    var query: BsonDocument = null
    if (!timestamp) {
      query = BsonDocument("buid" -> (fingerprint \
"buid").as[String], "floor" -> (fingerprint \ "floor").as[String],
        "location.coordinates" -> location.toList)
    } else {
      query = BsonDocument("buid" -> (fingerprint \
"buid").as[String], "floor" -> (fingerprint \ "floor").as[String],
        "location.coordinates" -> location.toList, "timestamp" ->
(fingerprint \ "timestamp").as[String])
    }
    val heatmapLookup = heatmap.find(query).first()
    val awaited = Await.result(heatmapLookup.toFuture, Duration.Inf)
    val res = awaited.asInstanceOf[Document]
    if (res != null)
      return convertJson(res)
    null
  }

  def replaceStoredHeatmap(collection: String, newHeatmap: JsValue,
fingerprint: JsValue, level: Int, timestamp: Boolean): Boolean = {
    val heatmap = mdb.getCollection(collection)
    val location = trimCoordinates(fingerprint, level)
    if (location == null) return false
    var query: BsonDocument = null
    if (!timestamp) {
      query = BsonDocument("buid" -> (fingerprint \
"buid").as[String], "floor" -> (fingerprint \ "floor").as[String],
        "location.coordinates" -> location.toList)
    } else {
      query = BsonDocument("buid" -> (fingerprint \
"buid").as[String], "floor" -> (fingerprint \ "floor").as[String],
        "location.coordinates" -> location.toList, "timestamp" ->
(fingerprint \ "timestamp").as[String])
    }
    val update = BsonDocument(newHeatmap.toString())
    val heatmapReplace = heatmap.replaceOne(query, update)
    val awaited = Await.result(heatmapReplace.toFuture, Duration.Inf)
    val res = awaited
    if (res.getModifiedCount == 0)
      false
    else
      true
  }

  def confirmNegativity(number: Int): Int = {
```

```
    if (number > 0)
      return number * (-1)
    return number
  }

  def updateHeatmap(fingerprint: JsValue, level: Int, timestamp:
Boolean): Boolean = {
    var collectionName = "heatmapWifi"
    if (timestamp)
      collectionName = collectionName + "Timestamp"
    collectionName = collectionName + level
    val storedHeatmap = fetchStoredHeatmap(collectionName,
fingerprint, level, timestamp)
    if (storedHeatmap == null) {
      val heatmap = createHeatmap(fingerprint, level, timestamp)
      ProxyDataSource.getIDatasource().addJsonDocument(collectionName,
heatmap.toString())
    } else {
      val newSum = confirmNegativity((fingerprint \ "sum").as[Int]) +
confirmNegativity((storedHeatmap \ "sum").as[Int])
      val newCount = (fingerprint \ "count").as[Int] + (storedHeatmap
\ "count").as[Int]
      val newHeatmap: JsValue = storedHeatmap.as[JsObject] + ("sum" ->
JsNumber(newSum)) + ("count" -> JsNumber(newCount))
      replaceStoredHeatmap(collectionName, newHeatmap, fingerprint,
level, timestamp)
    }
  }

  override def generateHeatmaps(): Boolean = {
    LPLogger.debug("generateHeatmaps: Generating Heatmaps")
    val bCollection = mdb.getCollection("buildings")
    val flCollection = mdb.getCollection("floorplans")
    val fiCollection = mdb.getCollection("fingerprintsWifi")
    //val tempQuery: BsonDocument = BsonDocument("buid" ->
"username_1373876832005")
    val buildingsLookup = bCollection.find()
    val awaitedB = Await.result(buildingsLookup.toFuture,
Duration.Inf)
    val resB = awaitedB.toList
    val buildings = convertJson(resB)
    for (building <- buildings) {
      val floorsLookup = flCollection.find(equal("buid", (building \
"buid").as[String]))
      val awaitedFl = Await.result(floorsLookup.toFuture,
Duration.Inf)
      val resFl = awaitedFl.toList
      val floors = convertJson(resFl)
      for (floor <- floors) {
        val query: BsonDocument = BsonDocument("buid" -> (building \
"buid").as[String], "floor" -> (floor \ "floor_number").as[String])
        val fingerprintsLookup = fiCollection.find(query)
        val awaitedFi = Await.result(fingerprintsLookup.toFuture,
Duration.Inf)
        val resFi = awaitedFi.toList
        val fingerprints = convertJson(resFi)
        for (fingerprint <- fingerprints) {
          if (!updateHeatmap(fingerprint, 1, false)) {
            LPLogger.debug("error at level 1")
            return false
```

```
            }
            if (!updateHeatmap(fingerprint, 2, false)) {
              LPLogger.debug("error at level 2")
              return false
            }
            if (!updateHeatmap(fingerprint, 3,false)) {
              LPLogger.debug("error at level 3")
              return false
            }
            if (!updateHeatmap(fingerprint, 1, true)) {
              LPLogger.debug("error at level 1 timestamp")
              return false
            }
            if (!updateHeatmap(fingerprint, 2, true)) {
              LPLogger.debug("error at level 2 timestamp")
              return false
            }
            if (!updateHeatmap(fingerprint, 3, true)) {
              LPLogger.debug("error at level 3 timestamp")
              return false
            }
          }
        }
      }
      LPLogger.debug("generateHeatmaps: Generated Heatmaps")
      true
    }

    override def deleteNotValidDocuments(): Boolean = ???
}
```