Πανεπιστήμιο Πειραιώς - Τμήμα Πληροφορικής
Πρόγραμμα Μεταπτυχιακών Σπουδών
«Πληροφορική»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Android εφαρμογή για την εύρεση των κοντινότερων συσκευών με χρήση Bluetooth και Spring Boot backend** **Android application to locate nearby devices using Bluetooth and Spring Boot backend**
Ονοματεπώνυμο Φοιτητή	**Αυγέρης Χαράλαμπος**
Πατρώνυμο	**Δημήτριος**
Αριθμός Μητρώου	**ΜΠΠΛ18004**
Επιβλέπων	**Ευθύμιος Αλέπης, Αναπληρωτής καθηγητής**

Ημερομηνία Παράδοσης **Απρίλιος 2022**

Τριμελής Εξεταστική Επιτροπή

(υπογραφή)	(υπογραφή)	(υπογραφή)
Ευθύμιος Αλέπης Αναπληρωτής καθηγητής	Μαρία Βίρβου Καθηγήτρια	Κωνσταντίνος Πατσάκης Αναπληρωτής Καθηγητής

Περίληψη

Η παρούσα μεταπτυχιακή εργασία αποσκοπεί στην ανάπτυξη μιας android εφαρμογής, η οποία θα χρησιμοποιεί την τεχνολογία του Bluetooth για να εντοπίζει τις κοντινότερες κινητές συσκευές, ώστε να ενημερώνει και να ειδοποιεί το χρήστη για το αν βρίσκεται σε μέρος με συγκέντρωση πολλών ατόμων. Η ιδέα προήλθε από την ανάγκη περιορισμού όσο γίνεται περισσότερο των συνωστισμών του πληθυσμού εξαιτίας της πανδημίας του ιού Covid-19 και έχει σαν σκοπό να βοηθήσει στη μείωση της μεταδοτικότητας του ιού αυτού. Το Bluetooth λοιπόν αποτελεί την ενδεδειγμένη λύση για ένα πρόβλημα αυτού του είδους, εξαιτίας των τεχνικών χαρακτηριστικών του και κυρίως της εμβέλειάς του η οποία είναι στα 10 μέτρα. Έχει, επίσης, αναπτυχθεί και ένας σέρβερ, με τη χρήση του εργαλείου Spring Boot και ο οποίος θα χρησιμοποιεί μια βάση PostgreSQL. Η εφαρμογή λοιπόν θα αποστέλλει τα δεδομένα στο σέρβερ, τα οποία με τη σειρά τους θα αποθηκεύονται στη βάση και εν συνεχεία θα χρησιμοποιούνται για την εξαγωγή στατιστικών όπως για παράδειγμα σε ποιες χώρες παρατηρούνται οι περισσότερες συγκεντρώσεις ατόμων, σε ποιες πόλεις, μέρες κτλ.

Abstract

This master thesis aims to develop an android application, which will use Bluetooth technology to locate the nearest nearby devices, in order to inform and notify the user whether he is in a crowded place. The idea was inspired by the need to reduce population overcrowding as much as possible due to the Covid-19 virus pandemic and aims to reduce the transmissibility of the virus. Bluetooth is therefore the appropriate solution for a problem of this kind, due to its technical characteristics and especially its range which is at 10 meters. A web server has also been developed, using Spring Boot, which will use a PostgreSQL database. The application will send the data to the web server, which in turn will be stored in the database and then they will be used to export statistics such as in which countries the highest overcrowding is observed, in which cities, days etc.

Πίνακας περιεχομένων

1. Εισαγωγή

Το ξέσπασμα της πανδημίας του ιού Covid-19 είχε σαν αποτέλεσμα να αλλάξει σε σημαντικό βαθμό η καθημερινότητά μας και η κοινωνική μας δραστηριότητα. Σιγά σιγά όλοι οι άνθρωποι κληθήκαμε να είμαστε όλο και πιο προσεκτικοί όσον αφορά την κοινωνική μας συμπεριφορά και τις συναναστροφές μας με τους άλλους ανθρώπους. Αντίστοιχα μέτρα προς αυτήν την κατεύθυνση, όπως το να φοράμε μάσκες, να κρατάμε αποστάσεις, να μη συνωστιζόμαστε σε μέρη με πολύ κόσμο κτλ, πάρθηκαν από τις κυβερνήσεις με σκοπό να αναχαιτιστεί όσον το δυνατόν περισσότερο η έξαρση της πανδημίας του ιού αυτού.

Αντικείμενο της παρούσας εργασίας είναι η ανάπτυξη μιας android εφαρμογής, η οποία θα δίνει στο χρήστη δύο βασικές δυνατότητες: πρώτον με τη χρήση των λειτουργιών Bluetooth και GPS του κινητού του τηλεφώνου θα μπορεί ανά πάσα ώρα και στιγμή να βλέπει την ακριβή τοποθεσία στην οποία βρίσκεται, καθώς και αν στην τοποθεσία αυτή υπάρχει μεγάλος συνωστισμός ατόμων και δεύτερον θα έχει τη δυνατότητα να ενημερώνεται με τη μορφή στατιστικών για τα μέρη (ανά χώρα, ανά πόλη, ανά μέρα) στα οποία παρατηρούνται συχνά μεγάλες συγκεντρώσεις ατόμων. Το δεύτερο επιτυγχάνεται με τη χρήση ενός σέρβερ με τον οποίο θα επικοινωνεί η εν λόγω εφαρμογή και ο οποίος έχει υλοποιηθεί με το εργαλείο Spring Boot. Κάθε φορά λοιπόν που ένας χρήστης θα κάνει χρήση της εφαρμογής για να εντοπίσει αν βρίσκεται σε μέρος με συνωστισμό, μετά το πέρας της εκτέλεσης των λειτουργιών του Bluetooth και του GPS, τα αποτελέσματα θα αποστέλλονται στο σέρβερ και θα γίνεται η αποθήκευσή τους σε μια βάση PostgreSQL. Με τη συλλογή λοιπόν των αποτελεσμάτων αυτών από κάθε χρήστη, θα καθίσταται εφικτή η εξαγωγή των στατιστικών που θα αφορούν τα μέρη στα οποία παρατηρούνται οι περισσότερες συγκεντρώσεις πολλών ατόμων καθώς σε ποιες μέρες της εβδομάδας παρατηρούνται οι συγκεντρώσεις αυτές. Να σημειωθεί, στο σημείο αυτό, ότι κάθε χρήστης θα έχει ένα μοναδικό userId, το οποίο θα του δίνεται αυτόματα με την ολοκλήρωση της εγγραφής του στην εφαρμογή. Κατά την εγγραφή του θα καταχωρεί μόνο ένα όνομα χρήστη και έναν κωδικό της επιλογής του. Ουδέποτε θα ζητηθεί από το χρήστη να εισάγει προσωπικά στοιχεία όπως όνομα, επίθετο, διεύθυνση κτλ. Τα δεδομένα λοιπόν που θα αποθηκεύονται στη βάση θα αφορούν μόνο τον αριθμό των συσκευών που βρέθηκαν (από την εκτέλεση του Bluetooth), τις συντεταγμένες (latitude, longitude) της τοποθεσίας του χρήστη εκείνη τη στιγμή (από το GPS) καθώς και την ακριβή ημερομηνία και ώρα που πραγματοποιήθηκε η αποθήκευση των δεδομένων αυτών στη βάση του σέρβερ και τα οποία θα συνοδεύονται από το userId του εκάστοτε χρήστη. Η ανωνυμία λοιπόν όλων των χρηστών θα διατηρείται καθ’ όλη τη διάρκεια της λειτουργίας της εφαρμογής.

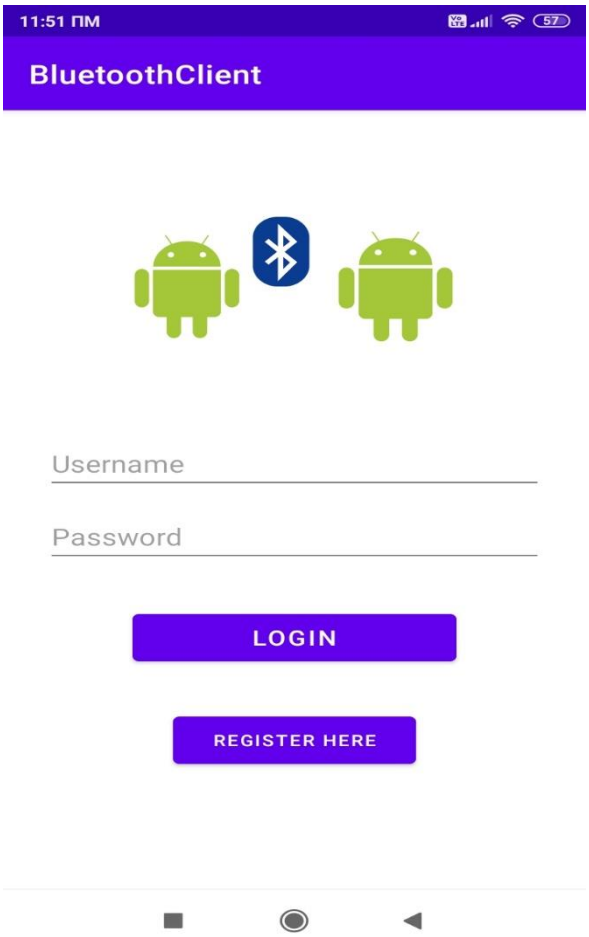
Με την ανάγκη λοιπόν, όπως είπαμε στην αρχή, της τήρησης των απαραίτητων αποστάσεων και της αποφυγής, όσον το δυνατόν περισσότερο, περιοχών με μεγάλο συνωστισμό, η εφαρμογή αυτή θα μπορούσε να αποτελέσει ένα χρήσιμο εργαλείο για τους χρήστες της με στόχο την αποτελεσματικότερη τήρηση των παραπάνω μέτρων.

2. Ανασκόπηση πεδίου

Είναι γεγονός ότι από την αρχή της πανδημίας του ιού Covid-19, έγινε αντιληπτό το πόσο σημαντική είναι η όσον το δυνατό μεγαλύτερη αναχαίτιση του ρυθμού μετάδοσης του ιού ανάμεσα στον πληθυσμό. Ένας κρίσιμος παράγοντας προς αυτήν την κατεύθυνση είναι ασφαλώς και η τήρηση κοινωνικών αποστάσεων και ένα κρίσιμο εργαλείο για την επίτευξή της αποτελούν τα κινητά μας τηλέφωνα (smartphones). Οι μέχρι τώρα υπάρχουσες εφαρμογές όμως δε διαθέτουν τις απαραίτητες δυνατότητες για την αντιμετώπιση των βασικών αυτών ζητημάτων εντοπισμού κοινωνικών αποστάσεων και πρόληψης μεγάλων συγκεντρωσεων και γι' αυτό το ενδιαφέρον στράφηκε γρήγορα στην τεχνολογία του Bluetooth. Το Bluetooth είναι μια ασύρματη τεχνολογία μικρής εμβέλειας που επιτρέπει την ασύρματη επικοινωνία δεδομένων μεταξύ ψηφιακών συσκευών. Η εμβέλειά του είναι περίπου 10 μέτρα. Γίνεται λοιπόν εύκολα αντιληπτό το πόσο χρήσιμο μπορεί να αποδειχτεί για την ανάπτυξη εφαρμογών με τις οποίες οι χρήστες θα μπορούν ενημερώνονται και ενδεχομένως να αποφεύγουν μέρη με συνωστισμό, γεγονός πολύ σημαντικό για την προστασία τους απ' τον ιό. Ο οργανισμός IEEE ανέπτυξε την εφαρμογή για android συσκευές "Social Distancing Alert System" η οποία χρησιμοποιεί το Bluetooth και ένα βελτιστοποιημένο αλγόριθμο για τη διατήρηση της κοινωνικής απόστασης. Το υπουργείο υγείας της Νέας Ζηλανδίας κυκλοφόρησε το Μάϊο του 2020 την εφαρμογή "NZ COVID Tracer" την οποία αργότερα ενημέρωσε προσθέτοντας τη λειτουργία του Bluetooth με τη χρήση του οποίου η εφαρμογή θα δημιουργούσε μια ανώνυμη εγγραφή κάθε ατόμου με το οποίο ο χρήστης είχε ποτέ βρεθεί κοντά. Αξιοσημείωτο είναι το γεγονός ότι τον Απρίλιο του 2020 η Google και η Apple ανακοίνωσαν την έναρξη μιας κοινής προσπάθειας ώστε με τη χρήση της τεχνολογίας του Bluetooth να βοηθήσουν τις κυβερνήσεις και τις υπηρεσίες υγείας να μειώσουν την εξάπλωση του ιού, με το απόρρητο και την ασφάλεια των χρηστών στο επίκεντρο του σχεδιασμού. Το υπουργείο υγείας της Αυστραλιανής Κυβέρνησης ανέπτυξε επίσης την εφαρμογή με όνομα "COVIDSafe" για να βοηθήσει στην προστασία της κοινότητάς της από τον Covid-19. Η COVIDSafe χρησιμοποιεί το Bluetooth του κινητού τηλεφώνου για να αναζητήσει άλλες συσκευές με εγκαταστημένη την εφαρμογή αυτή και αν υπάρχει επαφή τη σημειώνει καταγράφοντας με ασφάλεια τον κωδικό αναφοράς του άλλου χρήστη. Οι περισσότερες πλέον χώρες έχουν αναπτύξει τέτοιου είδους εφαρμογές με σκοπό την καταπολέμηση του ιού και ενδεχομένως να υπάρχουν και άλλα παραδείγματα εφαρμογών οι οποίες κάνουν χρήση του Bluetooth, με σκοπό την ασφάλεια και την υγεία των πολιτών.

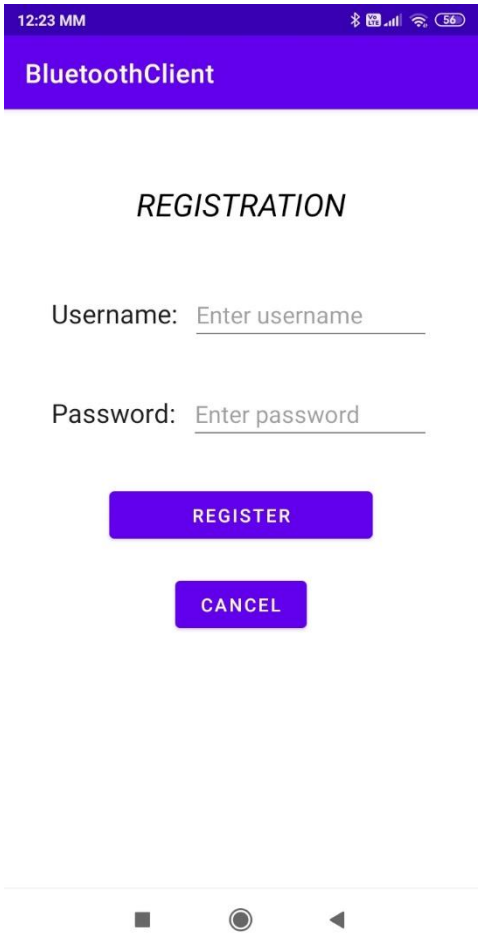
3. Παρουσίαση και χρήση της εφαρμογής

Ανοίγοντας την εφαρμογή εμφανίζεται αρχικά η φόρμα σύνδεσης (**Εικόνα 3.1**) . Εδώ ο χρήστης, εφόσον έχει ήδη πραγματοποιήσει εγγραφή, μπορεί να συνδεθεί εισάγοντας όνομα χρήστη και κωδικό στα αντίστοιχα πεδία και να πατήσει το κουμπί “LOGIN”. Αν είναι η πρώτη φορά που ανοίγει την εφαρμογή θα πρέπει πρώτα να κάνει εγγραφή πατώντας το κουμπί “REGISTER HERE”.

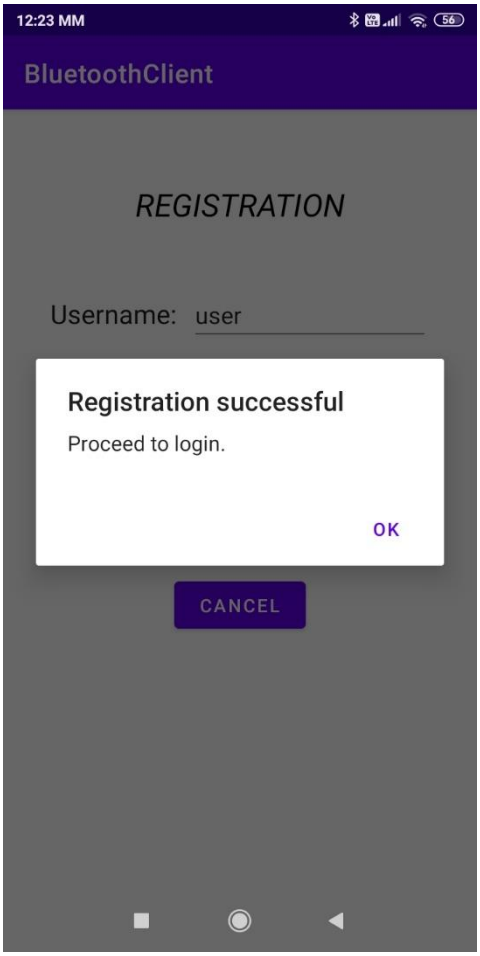


Εικόνα 3.1. Φόρμα σύνδεσης

Στη φόρμα εγγραφής (**Εικόνα 3.2**), ο χρήστης καλείται να εισάγει ένα όνομα χρήστη και έναν κωδικό της επιλογής του και στη συνέχεια πατώντας το κουμπί “REGISTER” να ολοκληρώσει την εγγραφή του. Εφόσον το όνομα χρήστη που πληκτρολόγησε δε συμπίπτει με αλλό που να υπάρχει ήδη στη βάση θα εμφανιστεί ένα μήνυμα που να τον ενημερώνει για την επιτυχία της εγγραφής του (**Εικόνα 3.3**). Φυσικά υπάρχει και το κουμπί “Cancel” το οποίο ακυρώνει τη διαδιακασία και στη συνέχεια οδηγεί ξανά στη φόρμα σύνδεσης.

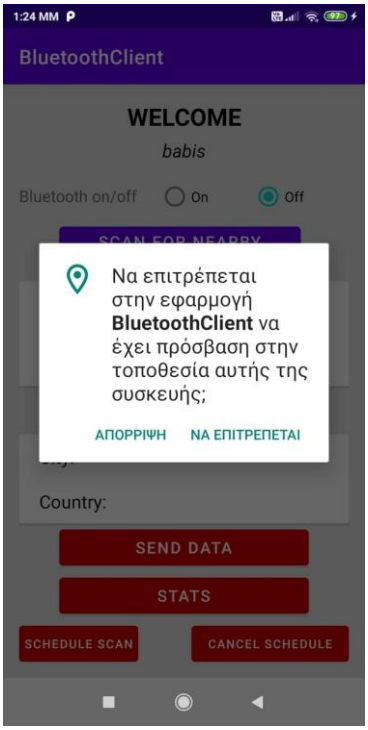


Εικόνα 3.2. Φόρμα εγγραφής

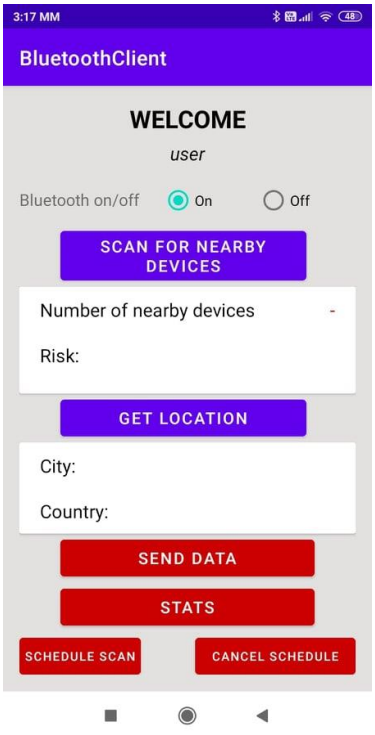


Εικόνα 3.3. Επιτυχής εγγραφή

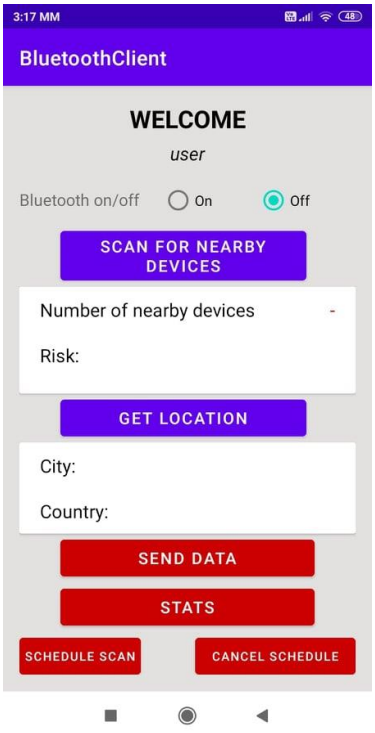
Μετά την επιτυχή εγγραφή και σύνδεση του χρήστη εμφανίζεται το κύριως μέρος της εφαρμογής. Την πρώτη φορά που θα πραγματοποιηθεί σύνδεση, θα ζητηθεί από την εφαρμογή να της χορηγηθεί η άδεια για την πρόσβασή της στην τοποθεσία της συσκευής (**Εικόνα 3.4**). Έπειτα ο χρήστης μπορεί, αρχικά, να δει στο πάνω μέρος της οθόνης και κάτω απ' το WELCOME το όνομα χρήστη που χρησιμοποιεί (στην προκειμένη περίπτωση "user"). Ακριβώς από κάτω έχει τη δυνατότητα να δει αν το Bluetooth της συσκευής του είναι ενεργοποιημένο ή όχι, ανάλογα με το ποιο radio button είναι τσεκαρισμένο (**Εικόνα 3.5** για Bluetooth ενεργό και **Εικόνα 3.6** για Bluetooth ανενεργό). Αν το Bluetooth αρχικά είναι ενεργό/ανενεργό ο χρήστης μπορεί να το ενεργοποιήσει/απενεργοποιήσει μέσα απ' την εφαρμογή, απλά επιλέγοντας το αντίστοιχο radio button on/off. Στην περίπτωση ενεργοποίησης του, θα ζητηθεί πρώτα από το χρήστη η άδεια ώστε να επιτραπεί στην εφαρμογή η ενεργοποίηση του Bluetooth (**Εικόνα 3.7**).



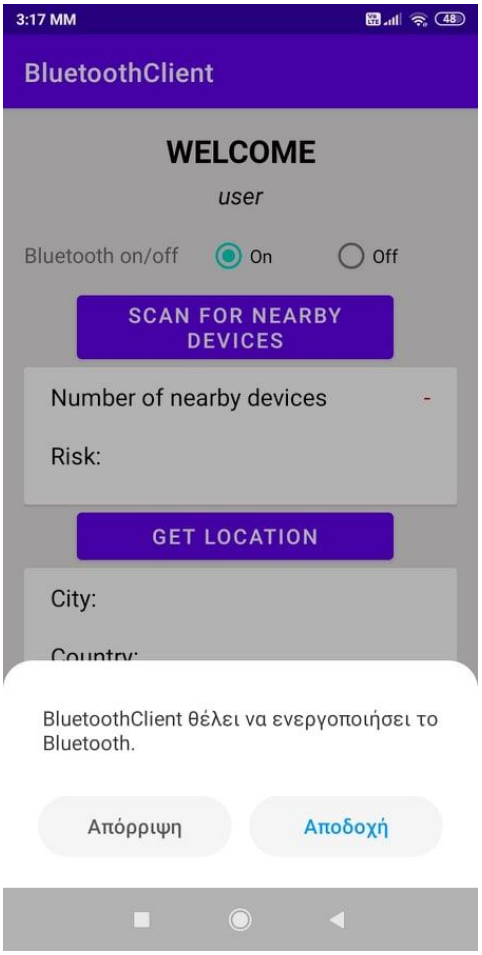
Εικόνα 3.4. Χορήγηση άδειας GPS



Εικόνα 3.5. Bluetooth ενεργό

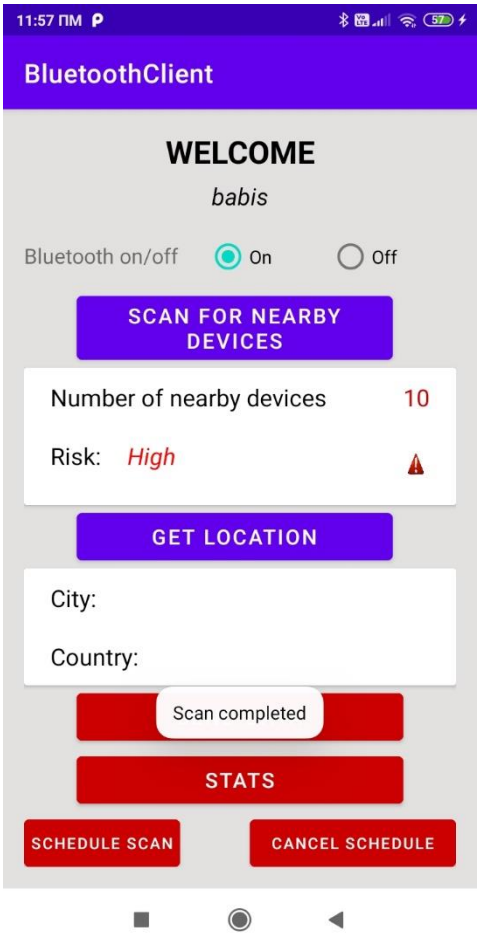


Εικόνα 3.6. Bluetooth ανενεργό

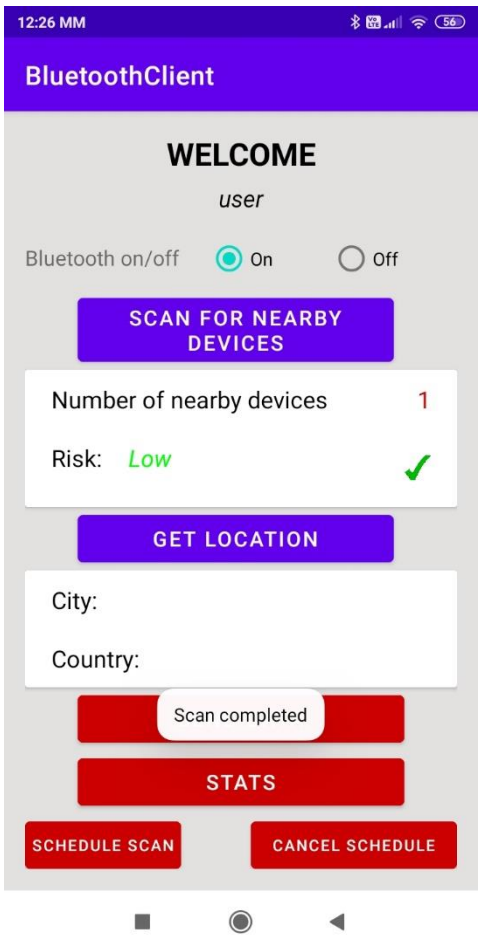


Εικόνα 3.7. Ενεργοποίηση του Bluetooth

Εφόσον το Bluetooth έχει ενεργοποιηθεί, ο χρήστης μπορεί να αναζητήσει κοντινές σε εκείνον συσκευές πατώντας το κουμπί “SCAN FOR NEARBY DEVICES”. Το Bluetooth θα ξεκινήσει την αναζήτηση και μόλις αυτή ολοκληρωθεί θα εμφανιστούν τα αποτελέσματα στο πλαίσιο κάτω από το προαναφερθέν κουμπί. Σε περίπτωση που οι συσκευές που θα έχουν βρεθεί ξεπερνούν τον αριθμό 9, τότε θα εμφανιστεί μια προειδοποίηση υψηλού κινδύνου στο χρήστη (**Εικόνα 3.8**), ενώ σε περίπτωση που ο αριθμός είναι μικρότερος του 9, θα δοθεί στο χρήστη μια ένδειξη χαμηλού κινδύνου (**Εικόνα 3.9**). Να σημειωθεί πως ο αριθμός των συσκευών που θα βρεθούν θα αφορά μόνο smartphones και όχι άλλες συσκευές (πχ headphones, laptop κτλ) και έτσι θα διασφαλίζεται όσον το δυνατόν καλύτερα ότι ο τελικός αριθμός θα αφορά τον πραγματικό αριθμό ατόμων που βρίσκονται σε κοντινή από το χρήστη απόσταση.

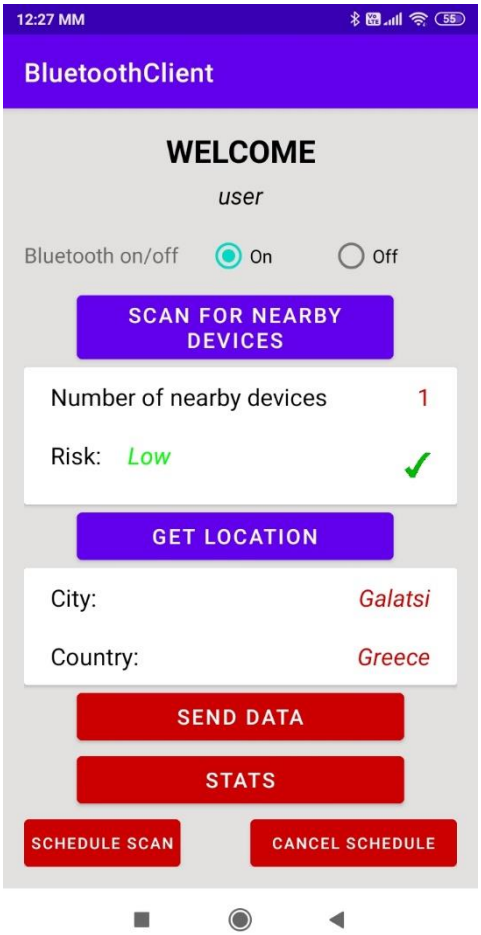


Εικόνα 3.8. Προειδοποίηση υψηλού κινδύνου



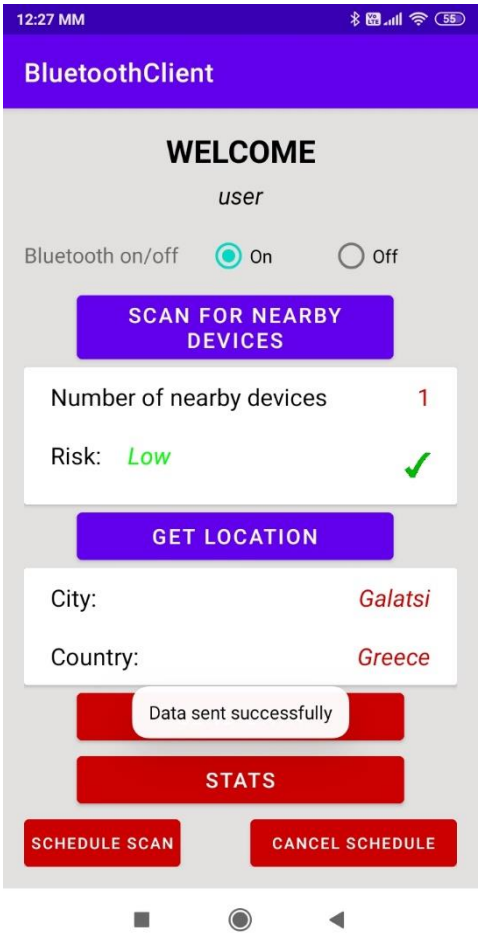
Εικόνα 3.9. Ένδειξη χαμηλού κινδύνου

Στη συνέχεια ο χρήστης μπορεί να κάνει αναζήτηση της ακριβούς τοποθεσίας του. Για να το επιτύχει αυτό, αρκεί να πατήσει του κουμπί “GET LOCATION”. Πατώντας λοιπόν το συγκεκριμένο κουμπί και εφόσον έχει ενεργοποιημένο το GPS της συσκευής του (σε περίπτωση που το GPS είναι ανενεργό θα του ζητηθεί να το ενεργοποιήσει) θα πραγματοποιηθεί αναζήτηση της τοποθεσίας του και τα αποτελέσματα θα εμφανιστούν στο αντίστοιχο πλαίσιο κάτω ακριβώς από το κουμπί αυτό (**Εικόνα 3.10**).



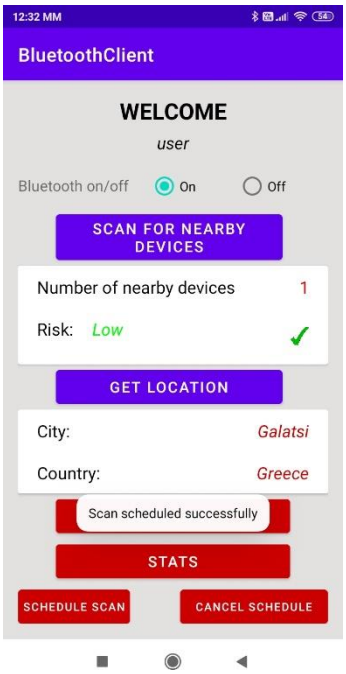
Εικόνα 3.10. Αποτελέσματα αναζήτησης τοποθεσίας

Αφού έχουν εκτελεστεί οι λειτουργίες των Bluetooth και GPS ο χρήστης μπορεί να πατήσει το κουμπί “SEND DATA” και τα δεδομένα από τις παραπάνω αυτές λειτουργίες θα αποσταλλούν στο σέρβερ (Spring Boot) όπου και θα αποθηκευτούν στη βάση PostgreSQL. Με την επιτυχή αποστολή τους θα εμφανιστεί και το αντίστοιχο μήνυμα στην οθόνη προς ενημέρωση του χρήστη (**Εικόνα 3.11)**.

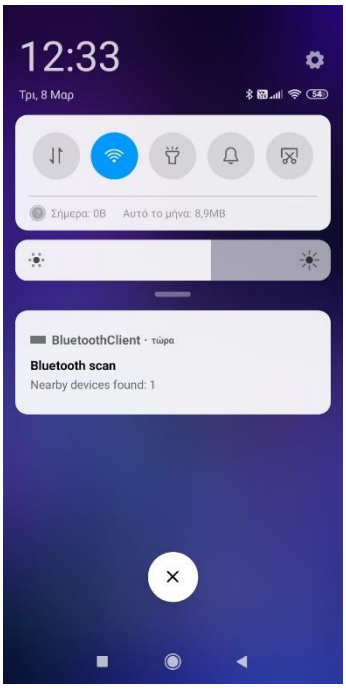


Εικόνα 3.11. Επιτυχής αποστολή δεδομένων

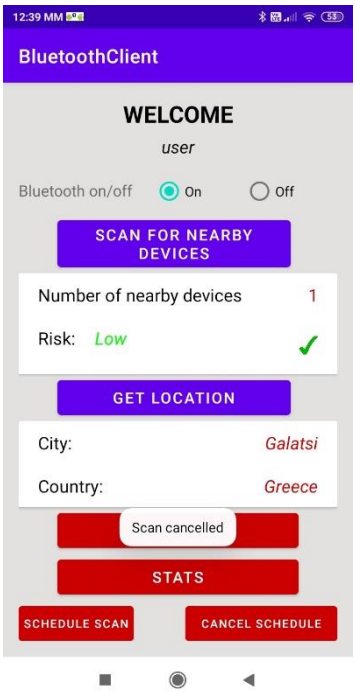
Ο χρήστης έχει τη δυνατότητα να προγραμματίσει τη διαδικασία ώστε να εκτελείται στο background της εφαρμογής χωρίς να απαιτείται αυτή να είναι ανοιχτή. Πατώντας το κουμπί "SCHEDULE SCAN" θα εμφανιστεί το μήνυμα της επιτυχούς έναρξης του χρονοδιαγράμματος (**Εικόνα 3.12**) το οποίο αφορά την ακόλουθη διαδικασία: αρχικά θα ξεκινήσει η αναζήτηση κοντινών συσκευών μέσω του Bluetooth όπου με την ολοκλήρωση της θα εμφανιστεί ένα notification στην αρχική οθόνη του κινητού του χρήστη με τον αριθμό των συσκευών που βρέθηκαν (**Εικόνα 3.13**), στη συνέχεια θα ακολουθήσει αναζήτηση της τοποθεσίας του χρήστη και μόλις ολοκληρωθεί και αυτή, θα γίνει αποστολή των δεδομένων στο σέρβερ και αποθήκευσή τους στη βάση. Η παραπάνω διαδικασία είναι αυτοματοποιημένη και θα επαναλαμβάνεται κάθε 15 λεπτά μέχρι ότου ο χρήστης πατήσει το κουμπί "CANCEL SCHEDULE" οπότε θα γίνει και η διακοπή της συνοδευόμενη από το αντίστοιχο μήνυμα (**Εικόνα 3.14**).



Εικόνα 3.12. Έναρξη χρονοδιαγράμματος

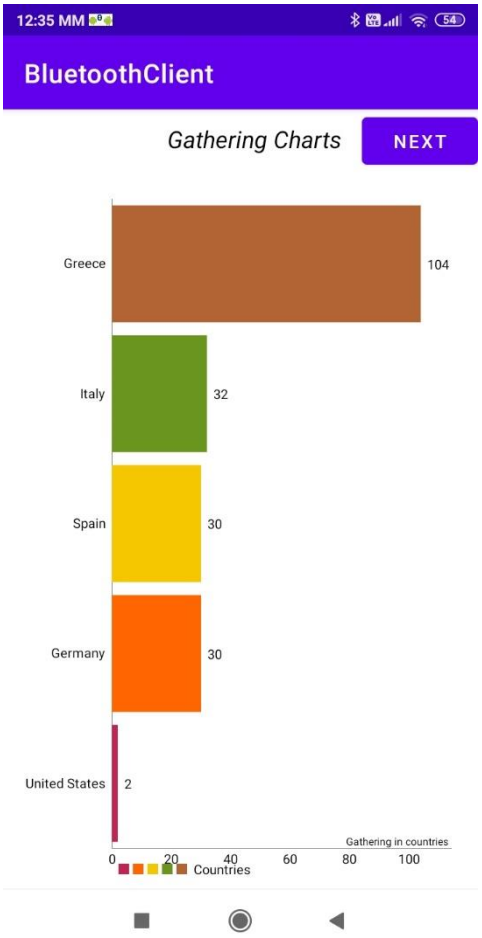


Εικόνα 3.13. Εμφάνιση notification



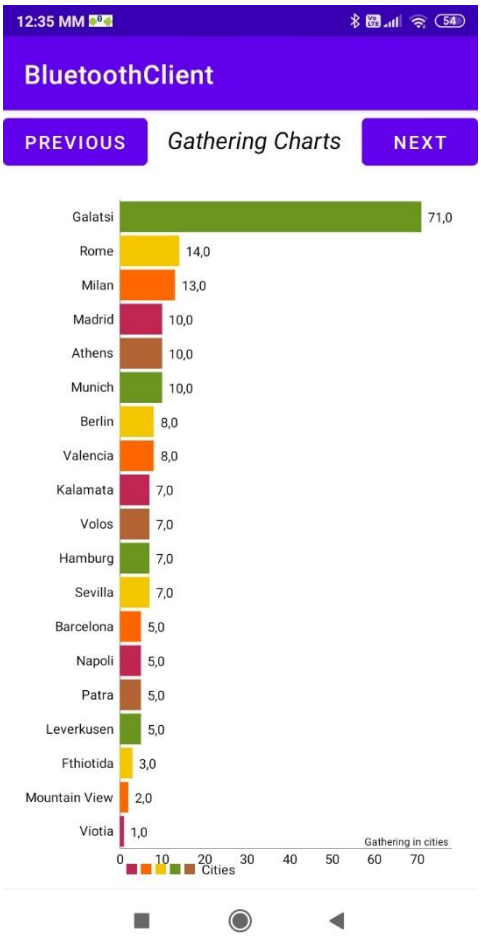
Εικόνα 3.14. Ακύρωση χρονοδιαγράμματος

Πατώντας το κουμπί “STATS” ανοίγει μια νέα activity στην οποία ο χρήστης αποκτά πρόσβαση σε διάφορα στατιστικά τα οποία εμφανίζονται με τη μορφή διαγραμμάτων. Η πλοήγηση μεταξύ των διαγραμμάτων μπορεί να γίνει με τη χρήση των κουμπιών “NEXT” και “PREVIOUS”. Για τις ανάγκες της παρουσίασης των διαγραμμάτων αυτών στο πλαίσιο της παρούσας εργασίας έχουν εισαχθεί χειροκίνητα κάποιες έγγραφες στη βάση του σέρβερ, ώστε να είναι ικανοποιητική η οπτική παρουσίασή τους. Ο χρήστης μπαίνοντας αρχικά στη νέα activity μπορεί να δει σε ποιες χώρες του κόσμου έχουν πραγματοποιηθεί μέχρι εκείνη τη στιμή οι περισσότερες συκεντρώσεις ατόμων (**Εικόνα 3.15**).



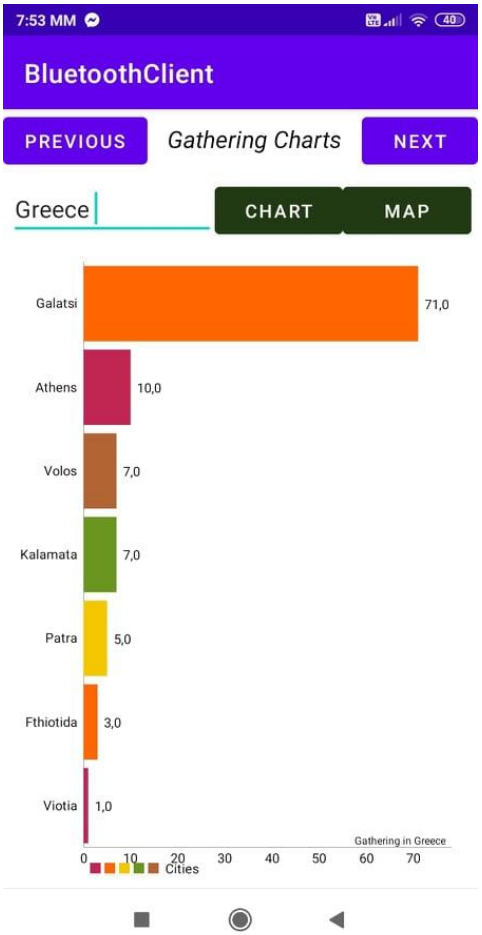
Εικόνα 3.15. Συγκεντρώσεις ανά χώρα

Ακολούθως πατώντας το κουμπί “NEXT” ο χρήστης μεταβαίνει στο επόμενο διάγραμμα στο οποίο μπορεί να ενημερωθεί αντίστοιχα για τις περισσότερες συγκεντρώσεις ανά πόλη παγκοσμίως (**Εικόνα 3.16**).

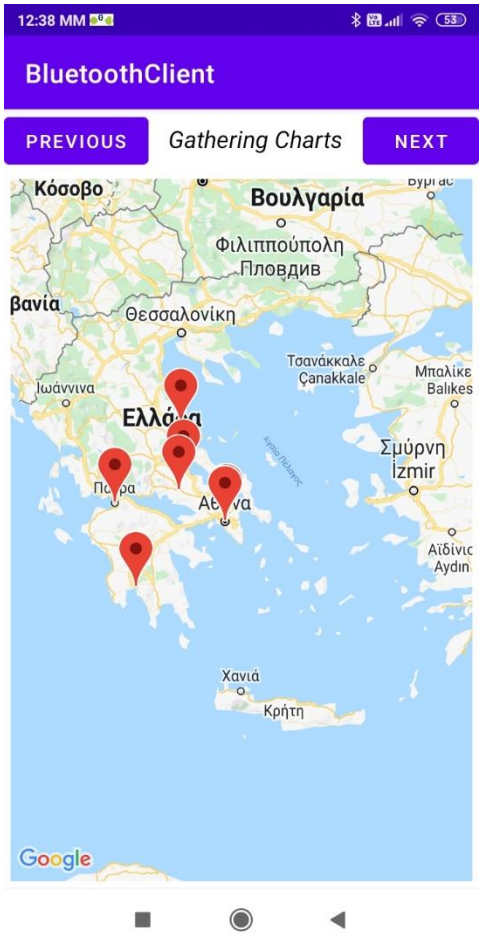


Εικόνα 3.16. Συγκεντρώσεις ανά πόλη

Στη συνέχεια ο χρήστης καλείται αρχικά να εισάγει μια χώρα της επιλογής του και στη συνέχεια έχει δύο επιλογές. Μπορεί να δει τις περισσότερες συγκεντρώσεις ανά πόλη για τη συγκεκριμένη χώρα την οποία εισήγαγε με τη μορφή ενός αντίστοιχου με τα προηγούμενα διαγράμματος πατώντας το κουμπί “CHART” (**Εικόνα 3.17**) ή έχει τη δυνατότητα χρησιμοποιώντας το κουμπί “MAP” να ανοίξει ένα χάρτη και να δει πάνω στο χάρτη αυτό, τα σημεία στα οποία παρατηρούνται οι περισσότερες συγκεντρώσεις για τη δεδομένη χώρα (**Εικόνα 3.18**).

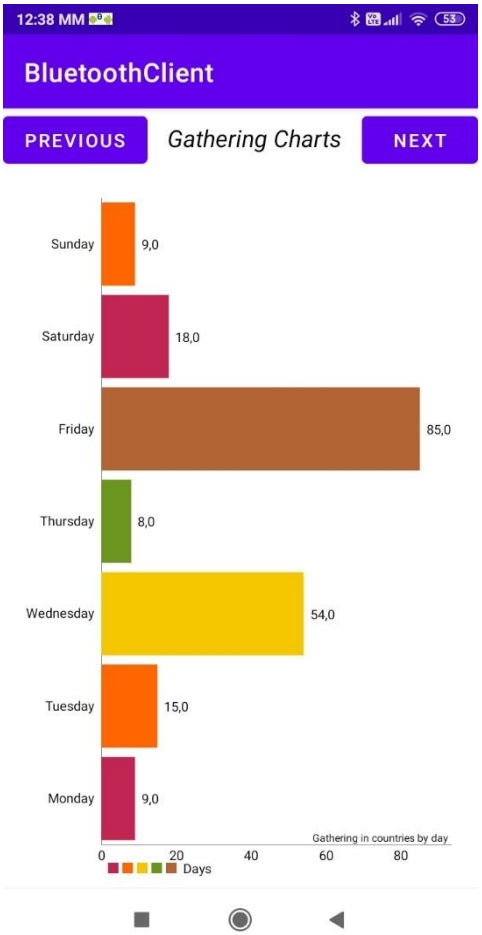


Εικόνα 3.17. Συγκεντρώσεις ανά πόλη για την Ελλάδα



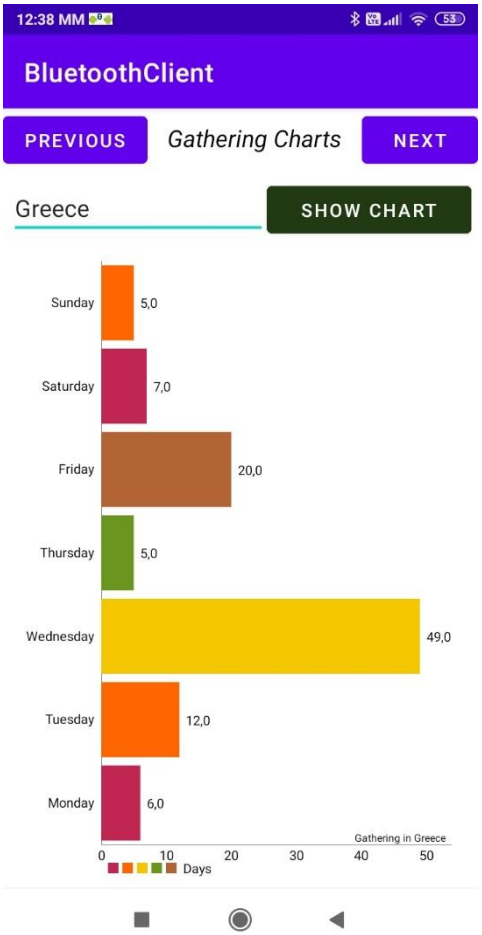
Εικόνα 3.18. Συγκεντρώσεις στην Ελλάδα χάρτης

Το επόμενο διάγραμμα αφορά τις περισσότερες συγκεντρώσεις πλυθησμού ανά μέρα παγκοσμίως (**Εικόνα 3.19**).



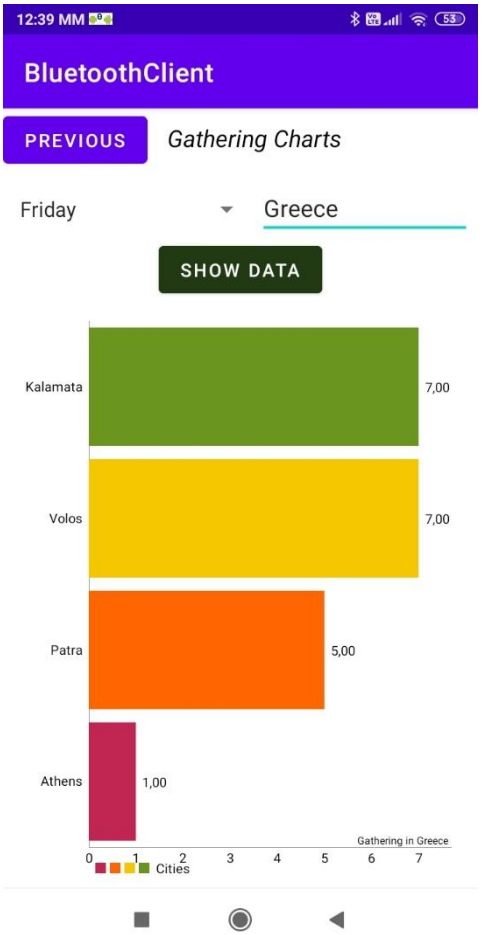
Εικόνα 3.19. Συγκεντρώσεις ανά μέρα παγκοσμίως

Για το επόμενο στατιστικό ο χρήστης καλείται ξανά να εισάγει μια χώρα της επιλογής του και πατώντας ακολούθως το κουμπί “CHART” εμφανίζεται το διάγραμμα με τις περισσότερες συγκεντρώσεις ανά μέρα για τη συγκεκριμένη χώρα (**Εικόνα 3.20**).



Εικόνα 3.20. Συγκεντρώσεις ανά μέρα για τη χώρα Ελλάδα

Τέλος, υπάρχει η δυνατότητα για το χρήστη να ενημερωθεί για τις πόλεις στις οποίες παρατηρούνται οι περισσότερες συγκεντρώσεις για κάθε χώρα, ξεχωριστά για κάθε μέρα της εβδομάδας. Ο χρήστης λοιπόν μπορεί να επιλέξει τη χώρα επιλογής του, καθώς και τη μέρα για την οποία θέλει να δει τα αποτελέσματα και στη συνέχεια με το κουμπί “SHOW DATA” να εμφανίσει το αντίστοιχο διάγραμμα (**Εικόνα 3.21**).



Εικόνα 3.21. Συγκεντρώσεις ανά πόλη για τη χώρα Ελλάδα την ημέρα Παρασκευή

4. Ανάλυση Σχεδιασμού

Για την υλοποίηση της εφαρμογής χρησιμοποιήθηκαν τα προγράμματα ανάπτυξης IntelliJ IDEA και Android Studio και η γλώσσα προγραμματισμού JAVA. Πιο συγκεκριμένα, στο IntelliJ IDEA αναπτύχθηκε το backend, το οποίο αποτελείται από το σέρβερ με τον οποίο θα επικοινωνεί η εφαρμογή και η υλοποίηση του οποίου πραγματοποιήθηκε με το εργαλείο Spring Boot. Για την αποθήκευση όλων των δεδομένων που θα στέλνονται στο σέρβερ χρησιμοποιήθηκε η βάση PostgreSQL. Αντίστοιχα στο Android Studio αναπτύχθηκε το frontend, δηλαδή η ίδια η android εφαρμογή.

Spring Framework

Το Spring Framework είναι ένα λογισμικό ανοιχτού κώδικα, οι λειτουργίες του οποίου μπορούν να χρησιμοποιηθούν για την ανάπτυξη JAVA εφαρμογών. Αν και το Framework αυτό δεν επιβάλλει κάποιο συγκεκριμένο μοντέλο προγραμματισμού, έχει γίνει δημοφιλές στην κοινότητα της JAVA ως μια προσθήκη στο μοντέλο Enterprise JavaBeans (EJB).

Η πρώτη έκδοση γράφτηκε από τον Rod Johnson, ο οποίος κυκλοφόρησε το Spring Framework με τη δημοσίευση του βιβλίου του “Expert One-On-One J2E Design and Development” τον Οκτώβριο του 2002. Το Spring Framework αρχικά κυκλοφόρησε με την άδεια Apache 2.0, τον Ιούνιο του 2003. Η πρώτη κυκλοφορία στην παραγωγή, 1.0, έλαβε χώρα το Μάρτιο του 2004. Το Spring 1.2.6 Framework κέρδισε το βραβείο παραγωγικότητας Jolt και το βραβείο JAX Innovation Award το 2006. Το Spring 2.0 κυκλοφόρησε τον Οκτώβριο του 2006, το Spring 2.5 το Νοέμβριο του 2007, το Spring 3.0 το Δεκέμβριο του 2009, το Spring 3.1 το Δεκέμβριο του 2011, και το Spring 3.2.5 το Νοέμβριο του 2013. Το Spring Framework 4.0 κυκλοφόρησε το Δεκέμβριο του 2013. Οι αξιοσημείωτες βελτιώσεις στο Spring 4.0 περιελάμβαναν την υποστήριξη για Java SE (Standard Edition) 8, Groovy 2, ορισμένες πτυχές της Java EE 7, και για WebSocket.

Το Spring Boot 1.0 κυκλοφόρησε τον Απρίλιο του 2014.

Το Spring Framework 4.2.0 κυκλοφόρησε στις 31 Ιουλίου 2015 και αναβαθμίστηκε αμέσως στην έκδοση 4.2.1, η οποία κυκλοφόρησε την 1η Σεπτεμβρίου 2015. Το Spring Framework 4.3 κυκλοφόρησε στις 10 Ιουνίου 2016 και θα υποστηρίζεται μέχρι το 2020. Το Spring 5 έχει ανακοινωθεί ότι θα κατασκευαστεί πάνω σε Reactive Streams και θα είναι συμβατό με Reactor Core.

Το Spring Framework περιλαμβάνει διάφορες ενότητες/μέρη (modules) που παρέχουν μια σειρά από υπηρεσίες:

- Spring Core Container: είναι το βασικό module του Spring και παρέχει spring containers (Bean Factory και ApplicationContext).
- Aspect-oriented programming: επιτρέπει την υλοποίηση “cross-cutting concerns”.
- Authentication and authorization: διαμορφωμένες διαδικασίες ασφαλείας που υποστηρίζουν μια σειρά προτύπων, πρωτοκόλλων, εργαλείων και πρακτικών μέσω του Spring Security sub-project (formerly Acegi Security System for Spring).

- Convention over configuration: μείωση του αριθμού των αποφάσεων που χρειάζεται να πάρει ένας προγραμματιστής που χρησιμοποιεί το framework χωρίς όμως να χάνεται η λειτουργικότητα και η ευελιξία.
- Data access: δυνατότητα για εργασία με συστήματα διαχείρισης σχεσιακών βάσεων δεδομένων στην πλατφόρμα Java χρησιμοποιώντας Java Database Connectivity (JDBC) και εργαλεία object-relational mapping καθώς και δυνατότητα για εργασία με NoSQL βάσεις δεδομένων.
- Inversion of control container: παραμετροποίηση των components της εφαρμογής και διαχείριση του κύκλου ζωής των Java objects μέσω του dependency injection.
- Messaging: χρήση Java Message Service (JMS) και βελτίωση της αποστολής μηνυμάτων μέσω standard JMS APIs.
- Model-view-controller: ένα HTTP-and-servlet-based framework που παρέχει τη βάση για web applications και RESTful (representational state transfer Web services)
- Testing: υποστήριξη κλάσεων για εκτέλεση unit tests και integration tests.

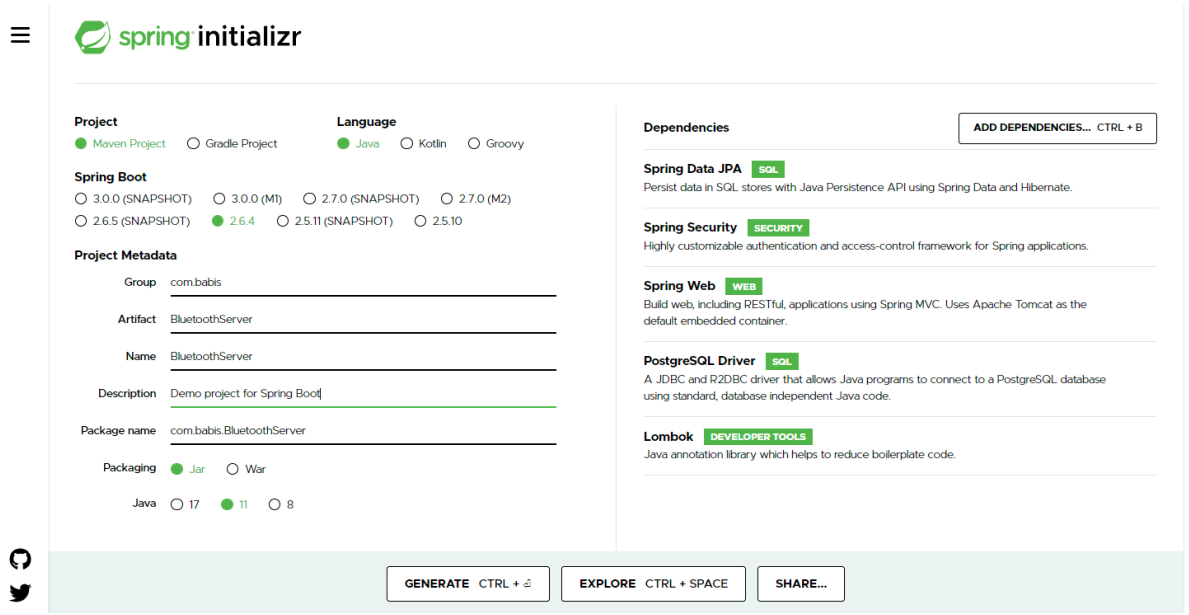
Spring Boot

Το Java Spring Boot (Spring Boot) είναι μια επέκταση του Spring Framework που κάνει την ανάπτυξη εφαρμογών και microservices ταχύτερη και ευκολότερη. Ουσιαστικά αποτελεί τη λύση του Convention over configuration που αναφέρθηκε παραπάνω, για τη δημιουργία αυτόνομων εφαρμογών, έτοιμων για την παραγωγή, τις οποίες μπορούμε απλώς να εκτελέσουμε. Τα κυριότερα χαρακτηριστικά του είναι:

- Επιτρέπει τη δημιουργία αυτόνομων εφαρμογών (Create stand-alone Spring applications)
- Έχει ενσωματωμένο web server (Embed Tomcat, Jetty or Undertow directly (no need to deploy WAR files))
- Αυτόματη παραμετροποιήση (autoconfiguration)
- Παροχή λειτουργιών έτοιμων για την παραγωγή (Provide production-ready features)
- Δεν υπάρχει απαίτηση για παραμετροποιήση XML (No requirement for XML configuration)

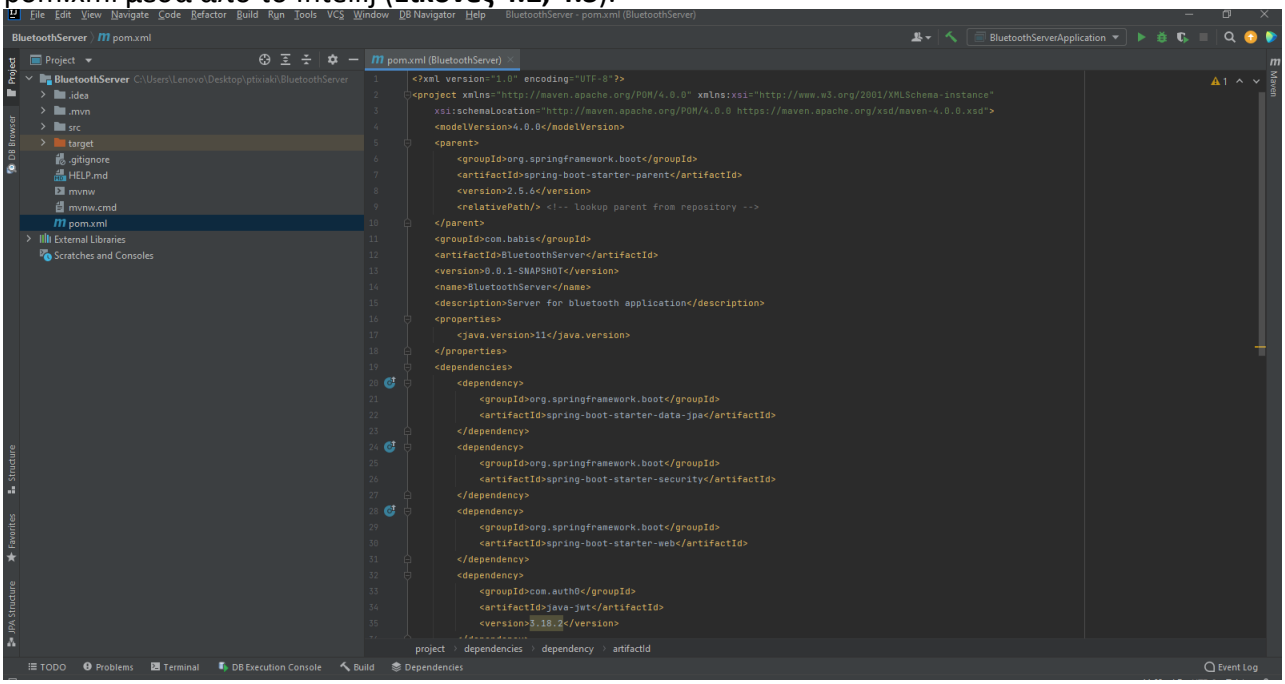
Το Spring Boot, λοιπόν, εξαλείφει την ανάγκη των πολύπλοκων ρυθμίσεων που απαιτούνται για την ανάπτυξη μιας Spring εφαρμογής, δίνοντας τη δυνατότητα στον προγραμματιστή να ασχοληθεί με την παραμετροποίηση μόνο συγκεκριμένων κομματιών ανάλογα με τις ανάγκες του project που θέλει να αναπτύξει.

Για την αρχική κατασκευή του project της παρούσας εργασίας χρησιμοποιήθηκε το Spring Initializr με την προσθήκη των ακόλουθων dependencies: Spring Data JPA, Spring Security, Spring Web, PostgreSQL Driver, Lombok (**Εικόνα 4.1**).

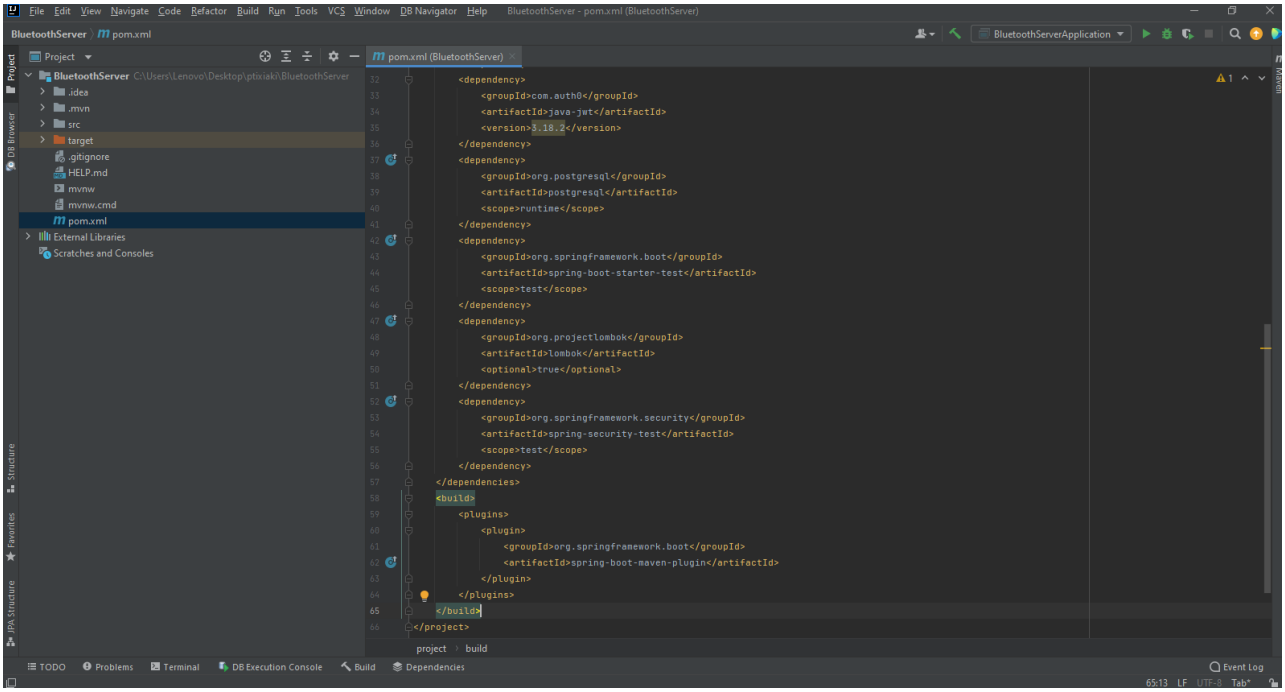


Εικόνα 4.1. Spring Initializr

Μετά τη δημιουργία του project μπορούμε να δούμε τις dependencies αυτές και από το pom.xml μέσα από το Intellij (**Εικόνες 4.2, 4.3**).

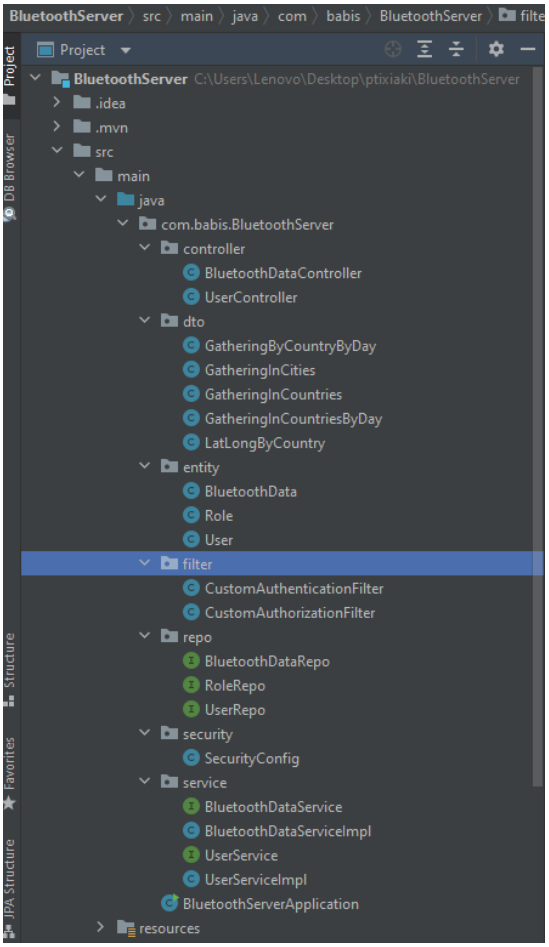


Εικόνα 4.2. pom.xml(1).

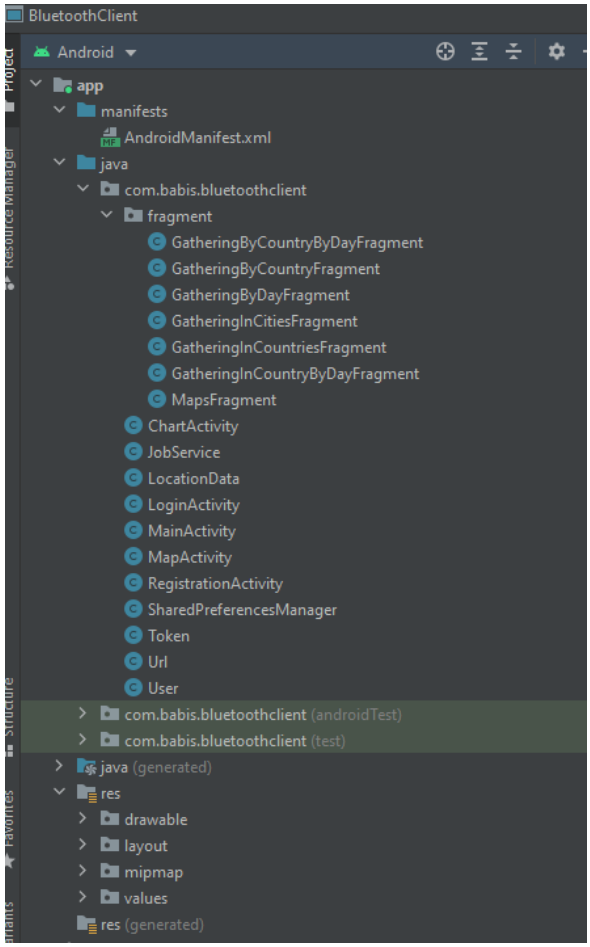


Εικόνα 4.3. pom.xml(2).

Παρακάτω βλέπουμε τις κλάσεις και τα packages που δημιουργήθηκαν για τις ανάγκες του συγκεκριμένου project, τόσο από τη μεριά του IntelliJ IDEA που αφορά την ανάπτυξη του σέρβερ (**Εικόνα 4.4**), όσο και από τη μεριά του Android Studio που αφορά την ανάπτυξη της εφαρμογής (**Εικόνα 4.5**).



Εικόνα 4.4. Δομή project IntelliJIdea



Εικόνα 4.5. Δομή project AndroidStudio

Όπως βλέπουμε στην εικόνα 4.4 τα packages και οι αντίστοιχες, μέσα σε αυτά, κλάσεις έχουν δημιουργηθεί με τρόπο τέτοιο ώστε να είναι ευκρινής η χρήση και των τριών επιπέδων μιας Spring εφαρμογής. Αυτά τα επίπεδα είναι: το Controller Layer που αντιστοιχεί στο package με όνομα controller και τις κλάσεις BluetoothDataController και UserController, το Service Layer που αντιστοιχεί στο package με όνομα service, τα interfaces BluetoothDataService, UserService και τις κλάσεις BluetoothDataServiceImpl και UserServiceImpl και τέλος το Data Access Layer που αντιστοιχεί στο package με όνομα repo και τα interfaces BluetoothDataRepo, RoleRepo και UserRepo.

Σχεδιασμός Βάσης PostgreSQL

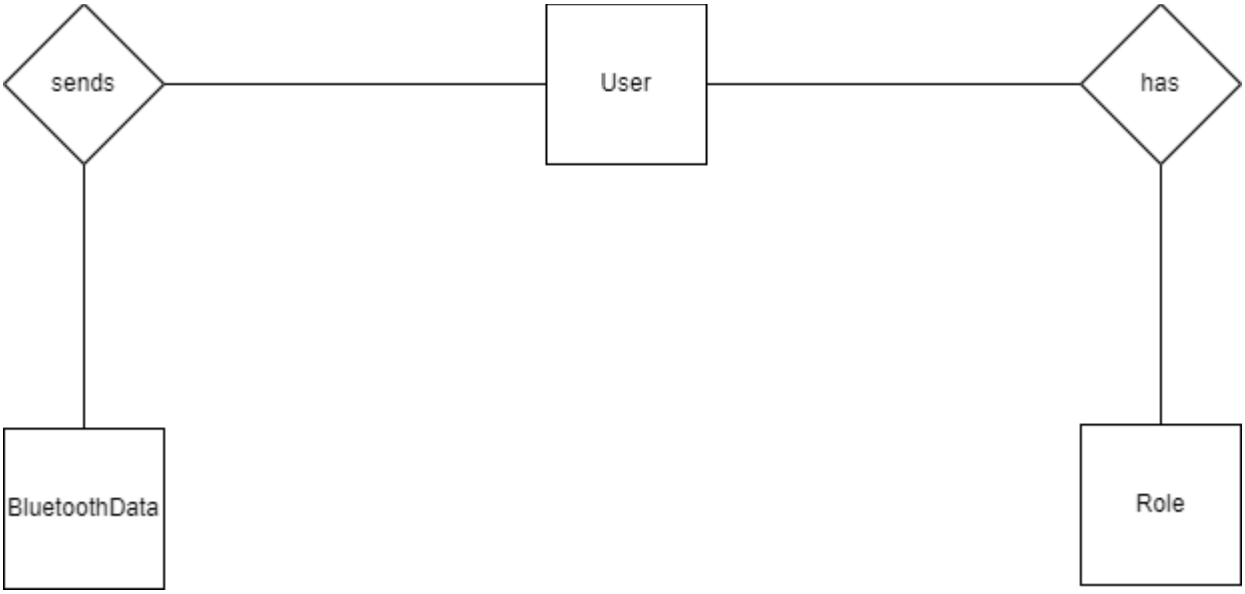
Ανάλυση απαιτήσεων

Η δημιουργία της βάσης δεδομένων για την παρούσα εργασία στηρίχθηκε στις ακόλουθες

προδιαγραφές:

- Να υπάρχει ένας πίνακας “User”, στον οποίο θα αποθηκεύονται οι χρήστες που κάνουν εγγραφή στην εφαρμογή. Ο πίνακας αυτός θα περιέχει το username και το password του κάθε χρήστη.
- Να υπάρχει ένας πίνακας “Role”, ο οποίος θα περιέχει το όνομα του ρόλου των χρηστών. Για τις ανάγκες της εργασίας αυτής έχει χρησιμοποιηθεί μόνο ένας ρόλος, ο “ROLE_USER”, που αφορά απλούς χρήστες με ίδια δικαιώματα πρόσβασης για όλες τις λειτουργίες της εφαρμογής.
- Να υπάρχει ένας πίνακας με όνομα “BluetoothData”, στον οποίο θα αποθηκεύονται τα δεδομένα που θα στέλνονται από την εφαρμογή στο σέρβερ. Τα δεδομένα αυτά θα περιλαμβάνουν τα εξής: city, country, latitude, longitude, num_of_phones (αριθμός κινητών συσκευών που βρέθηκαν), timestamp.

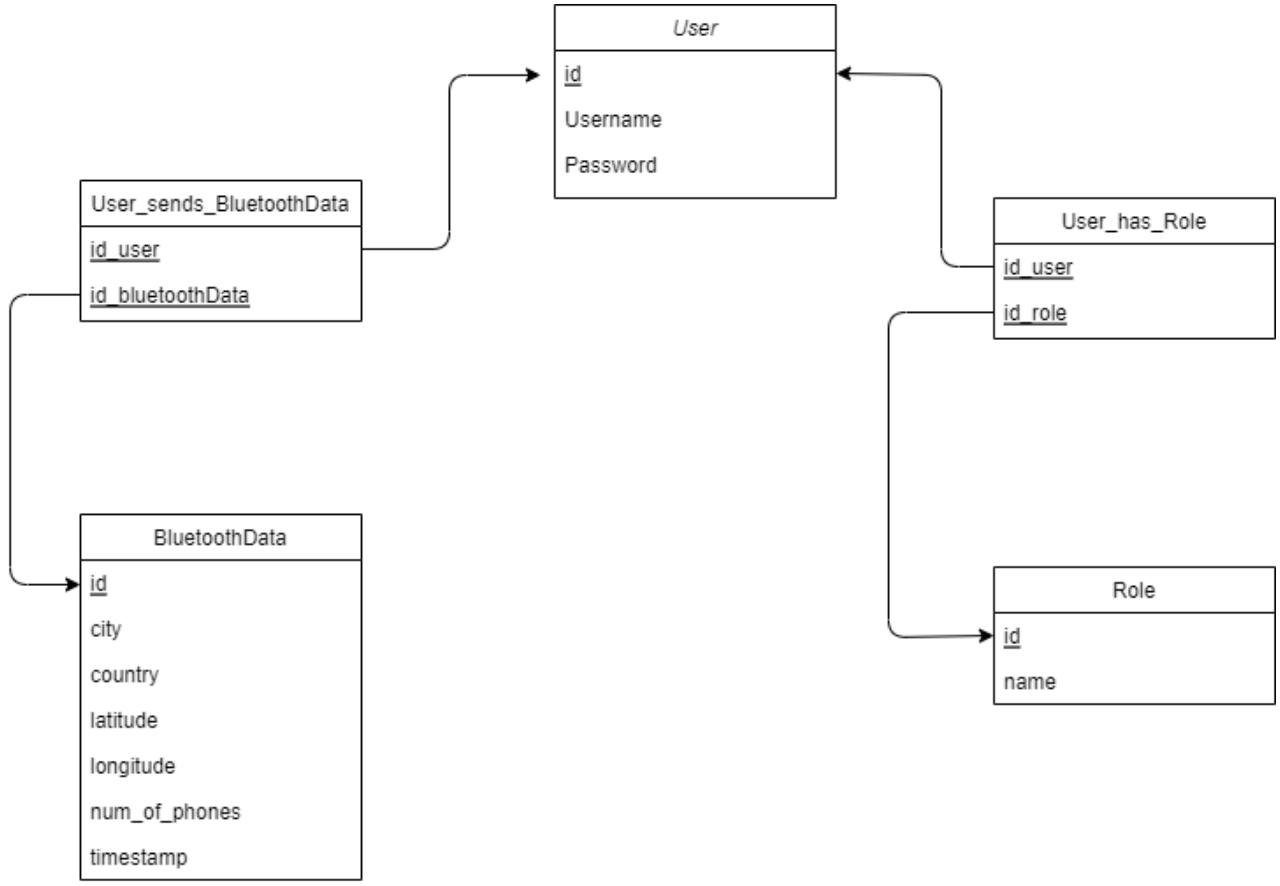
Διάγραμμα Οντοτήτων-Συσχετίσεων



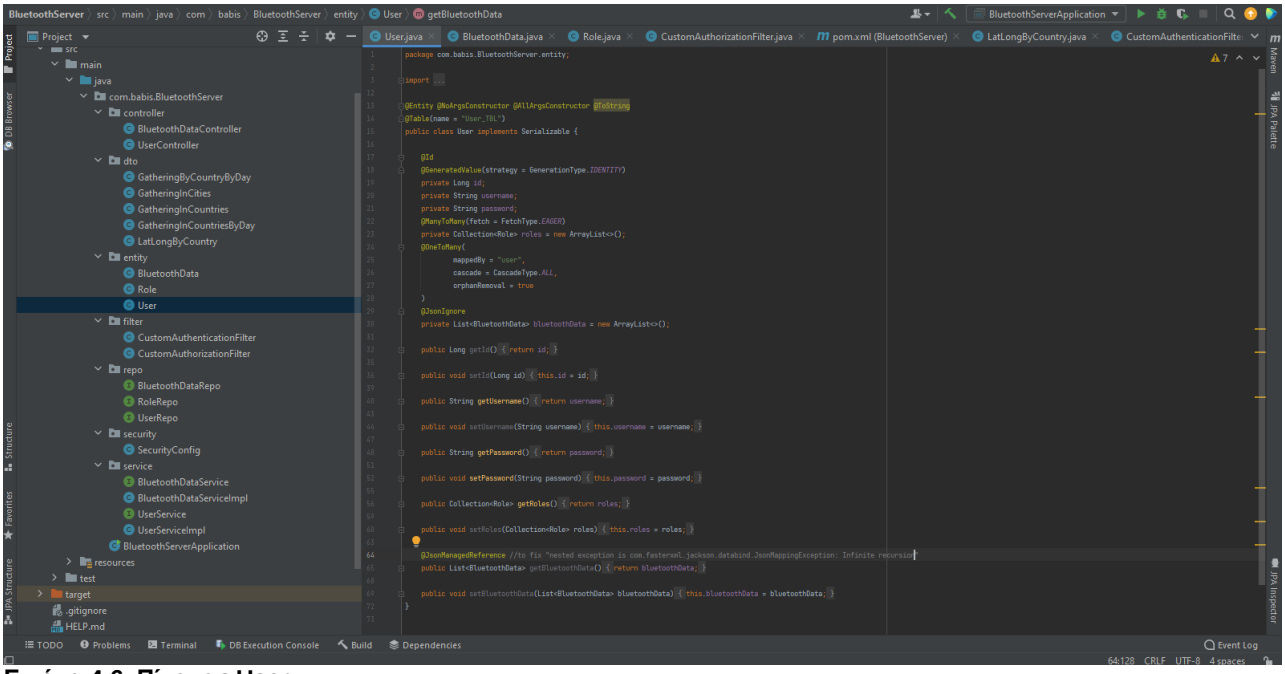
Από το παραπάνω διάγραμμα προκύπτουν 3 οντότητες και 2 συσχετίσεις. Η μία συσχέτιση είναι M:N (πολλά προς πολλά) και η άλλη 1:N (ένα προς πολλά).
Πιο αναλυτικά:

- User_has_Role: η συσχέτιση μεταξύ του πίνακα User και του πίνακα Role είναι πολλά προς πολλά, καθώς ένας χρήστης μπορεί να έχει έναν ή περισσότερους ρόλους και ένας ρόλος μπορεί να ανατεθεί σε έναν ή περισσότερους χρήστες.
- User_sends_BluetoothData: η συσχέτιση μεταξύ των πινάκων User και BluetoothData θα είναι ένα προς πολλά, καθώς ένας χρήστης μπορεί να στείλει δεδομένα μία ή περισσότερες φορές ενώ μια εγγραφή του πίνακα BluetoothData μπορεί να ανήκει μόνο σε ένα χρήστη.

Σχεσιακό σχήμα της ΒΔ



Στο package entity με τις κλάσεις BluetoothData, Role, User γίνεται η υλοποίηση και δημιουργία των πινάκων της βάσης. Στον πίνακα User θα αποθηκεύονται οι χρήστες της εφαρμογής, στον πίνακα Role θα αποθηκεύεται ο ρόλος τους και στον πίνακα BluetoothData θα αποθηκεύονται τα δεδομένα που θα στέλνονται από κάθε χρήστη προς το σέρβερ. Παρακάτω φαίνεται ενδεικτικά η υλοποίηση του πίνακα User (**Εικόνα 4.6**).



Εικόνα 4.6. Πίνακας User

Κατά την εγγραφή κάθε χρήστη στην εφαρμογή, ζητείται από εκείνον να εισάγει ένα όνομα χρήστη και έναν κωδικό της επιλογής του, τα οποία αποθηκεύονται στον πίνακα User. Στην παρακάτω εικόνα (**Εικόνα 4.7**) βλέπουμε ένα παράδειγμα αποθήκευσης των δεδομένων αυτών στη βάση μετά την επιτυχή ολοκλήρωση της διαδικασίας αυτής. Σαν στοιχεία εγγραφής για το παράδειγμα έχουν χρησιμοποιηθεί τα ακόλουθα:
Όνομα χρήστη (username) : “babis”
Κωδικός (password) : “111”

id	password	username
1	$2a$10$6mKXW74vAmiSRf9w0xp7m.jZwJNNWpqAsgj.AVbxcTHj6a1pyGqa.	babis

Εικόνα 4.7. Παράδειγμα εγγραφής χρήστη στη βάση PostgreSQL

Παρατηρούμε λοιπόν τρεις στήλες. Η πρώτη είναι η στήλη με όνομα “id” η οποία αποτελεί και το πρωτεύον κλειδί (primary key) του πίνακα και η οποία δημιουργείται αυτόματα. Η δεύτερη είναι η στήλη με όνομα “password” στην οποία αποθηκεύεται ο κωδικός που εισήγαγε ο χρήστης κρυπτογραφημένος και η τρίτη στήλη με όνομα “username” είναι η στήλη στην οποία αποθηκεύεται το όνομα χρήστη.
Στη συνέχεια βλέπουμε μια εγγραφή του χρήστη “babis” στον πίνακα BluetoothData (**Εικόνα 4.8**).

id	city	country	latitude	longitude	num_of_phones	timestamp	user_id
1	Galatsi	Greece	38.015570000000004	23.752683333333334	5	2022-03-08 12:33:29.801	1

Εικόνα 4.8. Παράδειγμα εγγραφής του χρήστη “babis”

Η πρώτη στήλη “id” είναι το πρωτεύον κλειδί (primary key) του πίνακα και η τελευταία στήλη “user_id” είναι το ξένο κλειδί (foreign key) που υποδεικνύει ότι τα δεδομένα αυτά ανήκουν στο χρήστη με “id” ίσο με 1 ο οποίος στο παράδειγμά μας ειναι ο χρήστης “babis”. Οι υπόλοιπες στήλες αφορούν τα δεδομένα που στέλνονται από την android εφαρμογή στο Spring. Όσον αφορά συγκεκριμένα, τη στήλη “timestamp”, αυτή θα συμπληρώνεται αυτόματα από το ίδιο το Spring. Κάθε φορά δηλαδή που ένας χρήστης θα στέλνει δεδομένα και αυτά θα αποθηκεύονται στον πίνακα BluetoothData, στη στήλη “timestamp” θα συμπληρώνεται αυτόματα η ακριβής ημερομηνία και ώρα αποθήκευσης της συγκεκριμένης εγγραφής. Αυτό επιτυγχάνεται με τη χρήση των annotations “@Temporal” και “@PrePersist” (**Εικόνα 4.9, Εικόνα 4.10**).

```
@Entity
@Table(name = "BluetoothData")
@NoArgsConstructor
@AllArgsConstructor
@ToString
public class BluetoothData implements Serializable {

    @Id
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private Long id;
    @Temporal(TemporalType.TIMESTAMP)
    @Column(nullable = false)
    private Date timestamp;
    private Integer numOfPhones;
    private double longitude;
    private double latitude;
    private String city;
    private String country;
    @ManyToOne(fetch = FetchType.LAZY)
    @JoinColumn(name = "user_id", nullable = false)
    @OnDelete(action = OnDeleteAction.CASCADE)
    @JsonIgnore
    private User user;
```

Εικόνα 4.9. Υλοποίηση @Temporal

```
@PrePersist
private void onCreate(){
    timestamp = new Date();
}

public Date getTimestamp() { return timestamp; }

public void setTimestamp(Date timestamp) { this.timestamp = timestamp; }
```

Εικόνα 4.10. Υλοποίηση @PrePersist

Να σημειωθεί ακόμα ότι με τη χρήση της dependency “Spring Security" κατά τη σύνδεση του χρήστη στην εφαρμογή γίνεται αυθεντικοποίηση του (authentication). Εφόσον η αυθεντικοποίηση είναι επιτυχής δίνονται ένα access token και ένα refresh token τα οποία αποθηκεύονται στα shared preferences της android εφαρμογής. Στα shared preferences αποθηκεύεται επίσης και το username του χρήστη. Ο κώδικας υλοποίησης των sharedPreferences φαίνεται παρακάτω και βρίσκεται μέσα στην κλάση με όνομα “SharedPreferencesManager”(**Εικόνα 4.11**).

```
//store the username in shared preferences
public void saveUsername(String username){
    SharedPreferences sharedPreferencesUser = ctx.getSharedPreferences(SHARED_PREF_NAME_USER, Context.MODE_PRIVATE);
    SharedPreferences.Editor editorUser = sharedPreferencesUser.edit();
    editorUser.putString( s: "username", username);
    editorUser.apply();
}

//return username
public String getUsername(){
    SharedPreferences sharedPreferencesUser = ctx.getSharedPreferences(SHARED_PREF_NAME_USER, Context.MODE_PRIVATE);
    return sharedPreferencesUser.getString( s: "username",  s1: null);
}

//this method will store the access_token and refresh_token in shared preferences
public void saveTokens(Token token){
    SharedPreferences sharedPreferences = ctx.getSharedPreferences(SHARED_PREF_NAME_TOKEN, Context.MODE_PRIVATE);
    SharedPreferences.Editor editor = sharedPreferences.edit();
    editor.putString(KEY_ACCESS_TOKEN, token.getAccess_token());
    editor.putString(KEY_REFRESH_TOKEN, token.getRefresh_token());
    editor.apply();
}

//this method will return the access_token and refresh_token
public Token getTokens(){
    SharedPreferences sharedPreferences =ctx.getSharedPreferences(SHARED_PREF_NAME_TOKEN, Context.MODE_PRIVATE);
    return new Token(
            sharedPreferences.getString(KEY_ACCESS_TOKEN,  s1: null),
            sharedPreferences.getString(KEY_REFRESH_TOKEN,  s1: null)
    );
}
```

Εικόνα 4.11. Shared preferences

Όταν λοιπόν ο χρήστης ζητήσει να συνδεθεί στην εφαρμογή μέσω του σέρβερ, πατώντας το κουμπί “Login” θα εκτελεστεί ο παρακάτω κώδικας (**Εικόνες 4.12(α,β)**). Θα αποθηκευτούν στις String μεταβλητές με όνομα username και password το περιεχόμενο των αντίστοιχων editText: editTextUsername και editextPassword, αφού πρώτα πραγματοποιηθεί έλεγχος ότι τα δύο αυτά editText δεν είναι κενά. Στη συνέχεια θα εκτελεστεί μια StringRequest μέσω της βιβλιοθήκης Volley με τη μέθοδο Post με την οποία θα σταλούν τα credentials (username, password) του χρήστη στο σέρβερ, ο οποίος με τη σειρά του θα ελέγξει αν είναι σωστά (authentication). Η αποστολή των username και password πραγματοποιείται μέσα στην protected Map<String, String> getParams() throws AuthFailureError{…}. Έπειτα, εφόσον ο έλεγχος είναι επιτυχής, θα σταλεί η αντίστοιχη ανταπόκριση στην εφαρμογή, στην οποία θα περιλαμβάνονται τα access token και refresh token. Θα κληθεί λοιπόν το κομμάτι του κώδικα που βρίσκεται μέσα στην public void onResponse(String response){...}. Στο κομμάτι αυτό, θα πραγματοποιηθεί η αποθήκευση στα shared preferences, των access token και refresh token που έστειλε ο σέρβερ, καθώς και του username του χρήστη, ώστε να εφικτή η άμεση επαναχρησιμοποίησή τους όποτε ξανά χρειαστεί.

```
btnLogin.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        if (!editTextUsername.getText().toString().isEmpty() && !editTextPassword.getText().toString().isEmpty()){
            String username = editTextUsername.getText().toString().trim();
            String password = editTextPassword.getText().toString().trim();

            RequestQueue requestQueue = Volley.newRequestQueue( context: LoginActivity.this);

            StringRequest stringRequest = new StringRequest(
                    Request.Method.POST,
                    url.getLOGIN_URL(),
                    new Response.Listener<String>() {
                        @Override
                        public void onResponse(String response) {

                            try {
                                JSONObject object = new JSONObject(response);

                                Token token = new Token(
                                        object.getString( name: "access_token"),
                                        object.getString( name: "refresh_token")
                                );
                                SharedPreferencesManager.getInstance(getApplicationContext()).saveTokens(token);
                                SharedPreferencesManager.getInstance(getApplicationContext()).saveUsername(username);
                            }
                            catch (JSONException exception) {
                                exception.printStackTrace();
                            }
                            finally {
                                Intent intent = new Intent(getApplicationContext(),MainActivity.class);
                                startActivity(intent);
                            }
                        }
                    }, new Response.ErrorListener() {
```

Εικόνα 4.12(α). Παράδειγμα αποθήκευσης access token, refresh token και username στα shared preferences

```
                }, new Response.ErrorListener() {
            @Override
            public void onErrorResponse(VolleyError error) {
                //403: wrong username or password
                if (error.networkResponse.statusCode == 403){
                    Toast.makeText(getApplicationContext(), text: "wrong username or password",
                            Toast.LENGTH_LONG).show();
                }
                else {
                    Toast.makeText(getApplicationContext(), text: "Login failed. \n Unknown error..",
                            Toast.LENGTH_LONG).show();
                }
            }
        }
        ){
            @Override
            protected Map<String, String> getParams() throws AuthFailureError{
                Map<String, String> params = new HashMap<>();
                params.put( k: "username", username);
                params.put( k: "password", password);
                return params;
            }
        };
        requestQueue.add(stringRequest);
    }
    else {
        Toast.makeText(getApplicationContext(), text: "Username or password can not be empty",
                Toast.LENGTH_LONG).show();
    }
  }
});
}
```

Εικόνα 4.12(β). Παράδειγμα αποθήκευσης access token, refresh token και username στα shared preferences

Έτσι για κάθε περαιτέρω επικοινωνία μεταξύ σέρβερ και εφαρμογής δε θα είναι απραίτητο να εισάγονται εκ νέου το όνομα χρήστη και ο κωδικός. Μαζί με κάθε αίτημα για επικοινωνία η εφαρμογή θα στέλνει το access token που έχει λάβει, το οποίο ο σέρβερ θα ελέγχει αν είναι ενεργό και δεν έχει λήξει και εφόσον ο έλεγχος είναι επιτυχής θα δίνεται η απόκριση στο αίτημα. Αν το access token έχει λήξει ο σέρβερ θα ενημερώνει την εφαρμογή και εκείνη με τη σειρά της θα ζητήσει να λάβει καινούργιο με τη χρήση του refresh token.

Στο σημείο αυτό, είναι χρήσιμο να γίνει μια συνοπτική αναφορά στη βιβλιοθήκη Volley, η χρήση της οποίας αναφέρθηκε παραπάνω.

Βιβλιοθήκη Volley

Η Volley είναι μια βιβλιοθήκη HTTP που κάνει τη δικτύωση (networking) για τις Android εφαρμογές ευκολότερη και το πιο σημαντικό, ταχύτερη. Αναπτύχθηκε από τη Google και παρουσιάστηκε κατά τη διάρκεια του Google I/O 2013. Αναπτύχθηκε λόγω της απουσίας, στο Android SDK, μιας networking class, η οποία θα μπορεί να λειτουργεί χωρίς παρεμβολές στο user experience. Διαχειρίζεται την επεξαργασία αιτημάτων δικτύωσης (network requests) και

εξοικονομεί πολύτιμο χρόνο στους προγραμματιστές από το να γράφουν τον ίδιο κώδικα για ένα αίτημα δικτύωσης ξανά και ξανά. Η Volley δεν είναι κατάλληλη για λειτουργίες μεγάλης λήψης ή ροής (large download or streaming operations), καθώς διατηρεί όλα τα responses στη μνήμη κατά την ανάλυση.

Χαρακτηριστικά της Volley

- Χρήση και λειτουργία μιας ουράς προτεραιότητας για τα requests (Request queuing)
- Αποτελεσματική διαχείριση της request cache
- Επεκτασιμότητα και προσαρμογή της βιβλιοθήκης στις εκάστοτε ανάγκες
- Ακύρωση των requests

Πλεονεκτήματα χρήσης της Volley

- Όλη η διεργασία που αφορά το Networking σε μια Android εφαρμογή μπορεί να γίνει με τη βοήθεια της Volley
- Αυτόματη χρονοδρομολόγηση (scheduling) των network requests
- Πολλαπλές ταυτόχρονες συνδέσεις δικτύου
- Δυνατότητα ιεράρχησης των requests με βάση την προτεραιότητα
- Παροχή εργαλείων εντοπισμού σφαλμάτων

Στη συνέχεια, θα δούμε ένα παράδειγμα χρήσης των access token και refresh token. Το κομμάτι του κώδικα που θα παρουσιαστεί θα αφορά το κουμπί “Send Data” (**Εικόνα 4.13**).

```
btnSendData.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        sendData();
    }
});
```

Εικόνα 4.13. Κουμπί “Send Data”

Με το πάτημα του κουμπιού αυτού καλείται η μέθοδος sendData(). Η μέθοδος αυτή στέλνει τα δεδομένα που έχουν προκύψει από την αναζήτηση του Bluetooth και του GPS, καθώς και το username του χρήστη, ώστε να γνωρίζει ο σέρβερ από ποιο χρήστη προέρχονται και να τα αποθηκεύσει ανάλογα. Κατά την αποθήκευσή τους δηλαδή στον πίνακα BluetoothData, να χρησιμοποιηθεί σαν foreign key της εγγραφής, το id του συγκεκριμένου χρήστη. Το περιεχόμενο της sendData() φαίνεται παρακάτω (**Εικόνα 4.14**).

```
public void sendData() {
    JSONObject jsonObjectData = new JSONObject();
    try {
        JSONObject objectUser = new JSONObject();
        objectUser.put( name: "username", username);

        jsonObjectData.put( name: "numOfPhones", numberOfDevices);
        jsonObjectData.put( name: "longitude", longitude);
        jsonObjectData.put( name: "latitude", latitude);
        jsonObjectData.put( name: "city", cityName);
        jsonObjectData.put( name: "country", countryName);
        jsonObjectData.put( name: "user", objectUser);

    } catch (JSONException e) {
        e.printStackTrace();
        Toast.makeText(getApplicationContext(), e.getMessage(), Toast.LENGTH_LONG).show();
    }
    JsonObjectRequest objectRequest = new JsonObjectRequest(
            Request.Method.POST,
            url.getURL_VOLLEY_SEND_DATA(),
            jsonObjectData,
            new Response.Listener<JSONObject>() {
                @Override
                public void onResponse(JSONObject response) {
                    Toast.makeText(getApplicationContext(), text: "Data sent successfully",
                            Toast.LENGTH_LONG).show();
                }
            }, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {

            //403: access token is expired, request for new with refresh token method
            if (error.networkResponse.statusCode == 403) {
                refreshToken();
            } else {
                Toast.makeText(getApplicationContext(), text: "Data have not been sent. \nUnknown error",
                        Toast.LENGTH_LONG).show();
            }
        }
    }
    ) {
        @Override
        public Map<String, String> getHeaders() throws AuthFailureError {
            Token token = SharedPreferencesManager.getInstance(getApplicationContext()).getTokens();
            String access_token = token.getAccess_token();
            Map<String, String> headers = new HashMap<>();
            headers.put( k: "Authorization", v: "Bearer " + access_token);
            return headers;
        }
    };
    requestQueue.add(objectRequest);
}
```

Εικόνα 4.14. Περιεχόμενο μεθόδου sendData().

Αρχικά κατασκευάζεται ένα JSONObject, με όνομα “”jsonObjectData”, στο οποίο εισάγονται τα απαραίτητα προς αποστολή δεδομένα. Αυτά είναι οι μεταβλητές: numOfDevices, longitude, latitude, cityName, countryName, username. Το username είναι ήδη, μετά από την πραγματοποίηση της σύνδεσης του χρήστη, αποθηκευμένο στα sharedPreferences, οπότε και εύκολο να το χρησιμοποιήσουμε όπως φαίνεται παρακάτω (**Εικόνα 4.15**).

```
//get username from shared preferences
username = SharedPreferencesManager.getInstance(getApplicationContext()).getUsername();
```

Εικόνα 4.15. Λήψη username από Shared Preferences

Η μεταβλητή numOfDevices αφορά τον αριθμό των κινητών συσκευών που βρέθηκαν μετά από το πέρας της αναζήτησης του Bluetooth. Οι μεταβλητές latitude, longitude είναι το αποτέλεσμα της αναζήτησης της τοποθεσίας της συσκευής του χρήστη μέσω του GPS και χρησιμοποιούνται για να εξαχθούν η πόλη και η χώρα με τη βοήθεια του Geocoder όπως φαίνεται στο παράκατω απόσπασμα κώδικα (**Εικόνα 4.16**):

```
try {
    geocoder = new Geocoder(getApplicationContext(), Locale.ENGLISH);
    List<Address> addresses = geocoder.getFromLocation(
            latitude,
            longitude,
            maxResults: 1);

    if (addresses.size() > 0){
        cityName = addresses.get(0).getLocality();
        countryName = addresses.get(0).getCountryName();
        textViewCity.setText(addresses.get(0).getLocality());
        textViewCountry.setText(countryName);
    }
}
catch (Exception e){
    Toast.makeText(getApplicationContext(),e.getMessage(),Toast.LENGTH_LONG).show();
}
```

Εικόνα 4.16. Χρήση Geocoder.

Αφού κατασκευαστεί το jsonObjectData, εκτελείται μια JsonObjectRequest μέσω της Volley με τη μέθοδο Post, με την οποία γίνεται η αποστολή του στο σέρβερ. Παράλληλα, όπως φαίνεται στο κομμάτι του κώδικα μέσα στην public Map<String, String> getHeaders() throws AuthFailureError{…} γίνεται και η αποστολή του access token. Αν όλα πάνε καλά, θα εμφανιστεί στο χρήστη, το μήνυμα της επιτυχούς αποθήκευσης των δεδομένων, μέσω ενός Toast, μέσα από την public void onResponse(JSONObject response){…}. Αν το access token έχει λήξει, τότε θα σταλεί από το σέρβερ ένδειξη σφάλματος με status code 403, το οποίο θα λάβει η εφαρμογή μέσα στην public void onErrorResponse(VolleyError error) {…} της Volley. Αν λάβει λοιπόν σφάλμα με αυτό το status code τότε θα κληθεί η μέθοδος refreshToken() με την οποία θα ζητηθεί ένα νέο access token. Παρακάτω βλέπουμε την υλοποίηση της μεθόδου refreshToken() (**Εικόνα 4.17**):

```
//access token is expired, method to get new access token
public void refreshToken() {
    JsonObjectRequest refreshTokenRequest = new JsonObjectRequest(
            Request.Method.GET,
            url.getREFRESH_TOKEN_REQUEST_URL(),
             jsonRequest null,
            new Response.Listener<JSONObject>() {
                @Override
                public void onResponse(JSONObject response) {
                    try {
                        Token token = new Token(
                                response.getString( name: "access_token"),
                                response.getString( name: "refresh_token")
                        );
                        SharedPreferencesManager.getInstance(getApplicationContext()).saveTokens(token);
                    } catch (JSONException e) {
                        e.printStackTrace();
                        Toast.makeText(getApplicationContext(), e.getMessage(), Toast.LENGTH_LONG).show();
                    } finally {
                        sendData(); //make again the sendData request with the new access_token
                    }
                }
            }, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {
            error.printStackTrace();
            Toast.makeText(getApplicationContext(),
                     text "Error: " + error.getMessage().toString(), Toast.LENGTH_LONG).show();
        }
    }
    ) {
        @Override
        public Map<String, String> getHeaders() throws AuthFailureError {
            Token token = SharedPreferencesManager.getInstance(getApplicationContext()).getTokens();
            String refreshToken = token.getRefresh_token();
            Map<String, String> headers = new HashMap<>();
            headers.put( k: "Authorization",  v: "Bearer " + refreshToken);
            return headers;
        }
    };
    requestQueue.add(refreshTokenRequest);
}
```

Εικόνα 4.17. Μέθοδος refreshToken()

Στη μέθοδο αυτή εκτελείται μια GET request, μέσω της Volley, με την οποία στέλνεται το refresh token στο σέρβερ. Αυτό υλοποιείται στο κομμάτι του κώδικα μέσα στην public Map<String, String> getHeaders() throws AuthFailreError {…}. Ο σέρβερ στη συνέχεια, εφόσον όλα πάνε καλά, θα στείλει εκ νέου ένα καινούριο access token μαζί με το refresh token τα οποία θα λάβει η εφαρμογή και θα τα αποθηκεύσει πάλι μέσα στα sharedPreferences. Αφού γίνει η αποθήκευσή τους, θα κληθεί πάλι η μέθοδος sendData(), από την οποία είχε προέλθει το αίτημα για νέο access token, προκειμένου να συνεχιστεί η διαδικασία της αποστολής των δεδομένων. Η υλοποίηση αυτή λαμβάνει χώρα μέσα στην public void onResponse(JSONObject response) {…}.

Ένα ακόμα ιδιαίτερο τμήμα κώδικα, που αξίζει αναφοράς, έχει να κάνει με την υλοποίηση των διαγραμμάτων, η χρήση των οποίων αφορά την προβολή των διάφορων στατιστικών. Στη συνέχεια λοιπόν θα δούμε τη διαδικασία που ακολουθήθηκε, για τη

δημιουργία του πρώτου διαγράμματος, το οποίο δείχνει σε ποιες χώρες παρατηρούνται οι περισσότερες συγκεντρώσεις ατόμων. Η ίδια λογική, φυσικά, έχει ακολουθηθεί και για τα υπόλοιπα διαγράμματα της εφαρμογής που παρουσιάστηκαν στο κεφάλαιο 3.

Αρχικά, στο IntelliJ, μέσα στο package “dto” δημιουργήθηκε η κλάση με όνομα “GatheringInCountries”, το περιεχόμενο της οποίας φαίνεται παρακάτω (**Εικόνα 4.18**).

```
package com.babis.BluetoothServer.dto;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;
import lombok.ToString;

@Data
@NoArgsConstructor
@AllArgsConstructor
@ToString
public class GatheringInCountries {

    private String country;
    private long count;

}
```

Εικόνα 4.18. Κλάση GatheringInCountries

Η κλάση αυτή θέλουμε να αποτελεί ουσιαστικά τον τύπο των δεδομένων, δηλαδή τα jsonObjects, που θα επιστρέφονται στην εφαρμογή, σαν απόκριση από το σέρβερ. Με άλλα λόγια, λοιπόν, όταν ένας χρήστης θα ζητήσει να του εμφανιστεί το συγκεκριμένο διάγραμμα, η εφαρμογή θα στείλει ένα αίτημα στο σέρβερ, με το οποίο θα ζητά να της σταλούν τα δεδομένα από τη βάση PostgreSQL, με τα οποία θα είναι σε θέση να δημιουργήσει το διάγραμμα αυτό. Όσον αφορά το συγκεκριμένο διάγραμμα, τα δεδομένα αυτά θα αφορούν τις χώρες με τις περισσότερες συγκεντρώσεις. Σαν απόκριση, συνεπώς, ο σέρβερ θα πρέπει στείλει ενά jsonArray ο οποίος θα αποτελείται από jsonObjects, ο αριθμός των οποίων θα ειναι ίσος με τον αριθμό των χωρών. Το περιεχόμενο κάθε jsonObject θα είναι το όνομα της χώρας και ο αριθμός εμφάνισης της χώρας αυτής στη βάση, ένα String “country” δηλαδή και ένας long “count”, όπως ακριβώς τα έχουμε ορίσει στην κλάση “ GatheringInCountries”. Για να είναι σε θέση ο σέρβερ να στείλει την απόκριση αυτή, θα πρέπει να του δώσουμε τη δυνατότητα να μπορεί να κάνει αναζήτησει στη βάση και να εξάγει το αποτέλεσμα. Αυτό το κατορθώνουμε με τη χρήση ενός κατάλληλου query μέσα στο Interface “BluetoothDataRepo” όπως ακριβώς φαίνεται στο παρακάτω τμήμα κώδικα (**Εικόνα 4.19**).

```
@Query("select new com.babis.BluetoothServer.dto.GatheringInCountries(b.country,count(b.country)) " +
        "from BluetoothData b " +
        "group by b.country " +
        "order by count(b.country) desc")
List<GatheringInCountries> findGatheringInCountries();
```

Εικόνα 4.19. Query

Κατόπιν, μέσα στο Interface “BluetoothDataService” δηλώνουμε τη μέθοδο με όνομα “List<GatheringInCountries> findGatheringInCountries()”, η υλοποίηση της οποίας γίνεται ακολούθως μέσα στην κλάση “BluetoothDataServiceImpl” (**Εικόνα 4.20**).

```
@Override
public List<GatheringInCountries> findGatheringInCountries() {
    return bluetoothDataRepo.findGatheringInCountries();
}
```

Εικόνα 4.20. Υλοποίηση findGatheringInCountries ()

Το τελευταίο κομμάτι, από τη μεριά του σέρβερ, είναι η κατασκευή του url, με τη χρήση του οποίου η εφαρμογή θα είναι σε θέση να επικοινωνήσει με το σέρβερ, για να λάβει τα δεδομένα των χωρών και του αριθμού που αυτές εμφανίζονται στη βάση. Αυτό το κομμάτι υλοποιείται στην κλάση “BluetoothDataController”, με τη μέθοδο public ResponseEntity<List<GatheringInCountries<> findGatheringInCountries() {…} και το annotation @GetMapping (**Εικόνα 4.21(α,β)**).

```
1   package com.babis.BluetoothServer.controller;
2
3   import ...
20
21  @RestController
22  @RequestMapping("/api/data")
23  @RequiredArgsConstructor
24  public class BluetoothDataController {
25
26      private final BluetoothDataService bluetoothDataService;
```

Εικόνα 4.21(α). Κλάση BluetoothDataController

```
@GetMapping("/gatheringInCountries")
public ResponseEntity<List<GatheringInCountries>> findGatheringInCountries(){
    return ResponseEntity.ok().body(bluetoothDataService.findGatheringInCountries());
}
```

Εικόνα 4.21(β). Υλοποίηση findGatheringInCountries()

Το μόνο που απομένει, πλέον, λοιπόν είναι να δοθεί η δυνατότητα στην εφαρμογή, να μπορεί να ζητά από το Spring να της στείλει τα δεδομένα που επιθυμεί και φυσικά η δυνατότητα να λαμβάνει τα δεδομένα αυτά και να εξάγει το διάγραμμα. Αυτό επιτυγχάνεται με μια Get Request μέσω της Volley. Μέσα στην κλάση GatheringInCountriesFragment, με τη μέθοδο private void loadDataFromServer() {…} υλοποιείται η δυνατότητα αυτή (**Εικόνα 4.22(α,β)**).

```
private void loadDataFromServer() {
    StringRequest stringRequest = new StringRequest(
            Request.Method.GET,
            url.getGatheringInCountries_Url(),
            new Response.Listener<String>() {
                @Override
                public void onResponse(String response) {
                    Log.d( tag: "string", response);

                    ArrayList<BarEntry> yVals1 = new ArrayList<BarEntry>();
                    ArrayList<String> labels  = new ArrayList<String>();

                    try {
                        JSONArray jsonArray = new JSONArray(response);

                        int j = 0;
                        for (int i=(jsonArray.length())-1; i>=0; i--){

                            JSONObject jsonObject = jsonArray.getJSONObject(i);

                            String country = jsonObject.getString( name: "country").trim();
                            String count = jsonObject.getString( name: "count").trim();

                            yVals1.add(new BarEntry(j++,Float.parseFloat(count)));
                            labels .add(country);

                        }
                        XAxis xl = chart.getXAxis();
                        xl.setPosition(XAxis.XAxisPosition.BOTTOM);
                        xl.setDrawAxisLine(true);
                        xl.setDrawGridLines(false);
                        xl.setValueFormatter(new IndexAxisValueFormatter(labels));
                        xl.setLabelCount(labels.size());
                        xl.setGranularity(1);

                        YAxis yl = chart.getAxisLeft();
                        yl.setPosition(YAxis.YAxisLabelPosition.INSIDE_CHART);
                        yl.setDrawGridLines(false);
                        yl.setEnabled(false);
                        yl.setAxisMinimum(0f);

                        YAxis yr = chart.getAxisRight();
                        yr.setPosition(YAxis.YAxisLabelPosition.INSIDE_CHART);
                        yr.setDrawGridLines(false);
                        yr.setAxisMinimum(0f);
```

Εικόνα 4.22(α). Μέθοδος loadDataFromServer()

```
                        yr.setAxisMinimum(0f);

                        BarDataSet set1;
                        set1 = new BarDataSet(yVals1, label: "Countries");
                        set1.setColors(ColorTemplate.COLORFUL_COLORS);
                        ArrayList<IBarDataSet> dataSets = new ArrayList<IBarDataSet>();
                        dataSets.add(set1);
                        BarData data = new BarData(dataSets);
                        data.setValueTextSize(10f);
                        data.setBarWidth(.9f);
                        chart.setData(data);
                        //chartGatheringInCountries.setScaleEnabled(false);
                        chart.setTouchEnabled(false);
                        chart.getDescription().setText("Gathering in countries");
                        chart.invalidate();

                    }
                    catch (JSONException e){
                        e.printStackTrace();
                    }
                }
            }, new Response.ErrorListener() {
        @Override
        public void onErrorResponse(VolleyError error) {
            //403: access token is expired, request for new with refresh token method
            if (error.networkResponse.statusCode == 403){
                //String loadDataFromServer = "loadDataFromServer";
                refreshToken();
            }
            else {
                Toast.makeText(getActivity().getApplicationContext(),
                        text: "Data have not been sent. \nUnknown error",
                                Toast.LENGTH_LONG).show();
            }
        }
    }
    ){
        @Override
        public Map<String, String> getHeaders() throws AuthFailureError {
            Token token = SharedPreferencesManager.getInstance(getActivity().getApplicationContext()).getTokens();
            String access_token = token.getAccess_token();
            Map<String, String> headers = new HashMap<>();
            headers.put( k: "Authorization", v: "Bearer " + access_token);
            return headers;
        }
    };
    requestQueue.add(stringRequest);
}
```

Εικόνα 4.22(β). Μέθοδος loadDataFromServer()

Η εφαρμογή θα στείλει το αίτημα αυτό προς το url “.../api/data/findGatheringInCountries” και θα λάβει την απόκριση του σέρβερ μέσα στην public void onResponse(String response){…}. Το υπόλοιπο τμήμα κώδικα, αφορά τη χρήση των δεδομένων που μόλις λήφθηκαν για την κατασκευή του διαγράμματος. Το διάγραμμά που χρησιμοποιείται για την εξαγωγή όλων των στατιστικών της εφαρμογής είναι το Horizontal Bar Graph με τη βοήθεια της βιβλιοθήκης MPAndroidChart.

5. Συμπεράσματα και μελλοντικές επεκτάσεις

Η πανδημία του ιού Covid-19 μας ανάγκασε δυστυχώς όλους να αλλάξουμε την καθημερινότητα μας, καθώς κληθήκαμε ξαφνικά να περιορίσουμε τις κοινωνικές μας συναναστροφές και δραστηριότητες. Μέσα στο πλαίσιο της κατάστασης αυτής λοιπόν βασίστηκε η ανάπτυξη της εργασίας που παρουσιάστηκε, με σκοπό να αποτελέσει ένα πρόσθετο εργαλείο για τους χρήστες, βοηθώντας τους να αποφεύγουν μέρη με μεγάλο συνωστισμό, συμβάλλοντας έτσι στον περιορισμό της έξαρσης της πανδημίας και στην όσον το δυνατόν ταχύτερη επιστροφή στην κανονικότητα. Είναι μια εφαρμογή εύκολη και κατανοητή ως προς τη χρήση της, με ένα απλό και φιλικό προς το χρήστη περιβάλλον. Για τον εντοπισμό των κοντινότερων συσκευών χρησιμοποιεί το Bluetooth, το οποίο αποτελεί την ενδεδειγμένη επιλογή χάρη στα τεχνικά χαρακτηριστικά του και κύριως χάρη στην εμβέλειά του που είναι τα 10 μέτρα. Σε συνδυασμό λοιπόν και με το Spring Boot το οποίο είναι ένα εργαλείο με τρομερές δυνατότητες, αποτελεί μια εφαρμογή σύγχρονη και χρήσιμη για τους χρήστες.

Σαφώς υπάρχει και η προοπτική για μελλοντική επέκταση της εφαρμογής και εμπλουτισμό της με περισσότερες λειτουργίες και δυνατότητες. Μια τέτοια λειτουργία θα μπορούσε να είναι για παράδειγμα, η προσθήκη ίσως ενός πεδίου στο οποίο ο χρήστης θα μπορεί να δηλώσει αν είναι θετικός στον ιό. Με αυτόν τον τρόπο θα καθίσταται δυνατό, όσοι χρήστες της εφαρμογής βρεθούν σε κοντινή σε αυτόν απόσταση, να λαμβάνουν ειδοποίηση ότι ένα άτομο θετικό στον ιό βρίσκεται κοντά τους εκείνη τη στιγμή. Μια ακόμα επέκταση, θα μπορούσε να είναι να μπορεί ο χρήστης, ανεξάρτητα αν είναι θετικός ή όχι στον ιό, να ανοίγει τον χάρτη, να βλέπει πάνω σε αυτόν την τοποθεσία που βρίσκεται εκείνη τη στιγμή και να έχει τη δυνατότητα να ενημερώνεται για τα κοντινότερα σε αυτόν νοσοκομεία ή φαρμακεία.

Σε κάθε περίπτωση, όμως, έχοντας ολοκληρώσει τη μελέτη του αντικειμένου και την κατασκευή της παρούσας εφαρμογής, μπορούμε να πούμε ότι αποτελεί μια καλή βάση με πολύ συγκεκριμένα και θετικά χαρακτηριστικά και λειτουργίες, που μπορεί φυσικά να εμπλουτιστεί με περαιτέρω βελτιώσεις.

Βιβλιογραφία

https://spring.io/

https://www.baeldung.com/

https://www.javaguides.net/

https://www.postgresql.org/

https://mode.com/blog/postgres-sql-date-functions/

https://developer.android.com/

https://github.com/PhilJay/MPAndroidChart

https://en.wikipedia.org/