Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών

Τμήμα Πληροφορικής και Τηλεπικοινωνιών

Ανάπτυξη Λογισμικού για Δίκτυα και Τηλεπικοινωνίες

ΕΡΓΑΣΙΑ ΧΕΙΜΕΡΙΝΟΥ ΕΞΑΜΗΝΟΥ

Ιανουάριος 2022

Βασίλειος Πάσιος - 1115201700121

Δημήτριος Κοσμάς-Λέκκας - 1115201200224

Ιωάννης Σακκάς - 1115201500140

Κωνσταντίνος Μονογιούδης - 1115201400110

Κωνσταντίνος Πατούνας - 1115201400155

Πλάτων Μηλίτσης - 1115201700087

Εισαγωγή

Η εργασία αυτή έχει πραγματοποιηθεί από εμάς σε 2 ομάδες.

Η μία (Δημήτριος Κοσμάς-Λέκκας, Βασίλειος Πάσιος, Πλάτων Μηλίτσης) ανέλαβε την υλοποίηση του edge server και χρησιμοποίησε java, τον eclipse paho mqtt client και javascript (node.js & svelte).

Η άλλη(Κωνσταντίνος Μονογιούδης, Κωνσταντίνος Πατούνας, Ιωάννης Σακκάς), ανέλαβε το κομμάτι του android και χρησιμοποίησε java, android studio, και τον eclipse paho mqtt client.

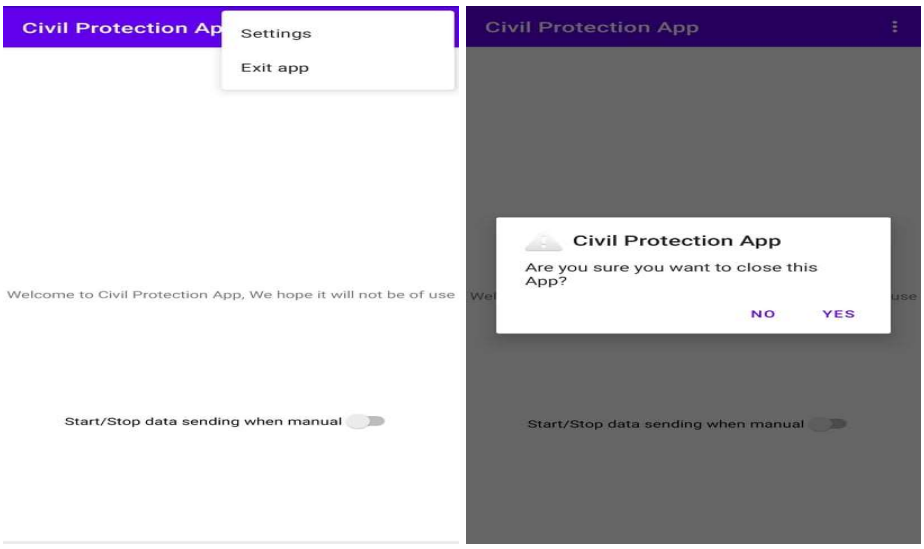
Παρουσίαση android εφαρμογών

Android Device

Θα ξεκινήσουμε με την παρουσίαση του Client με όνομα εφαρμογής Civil Protection App.

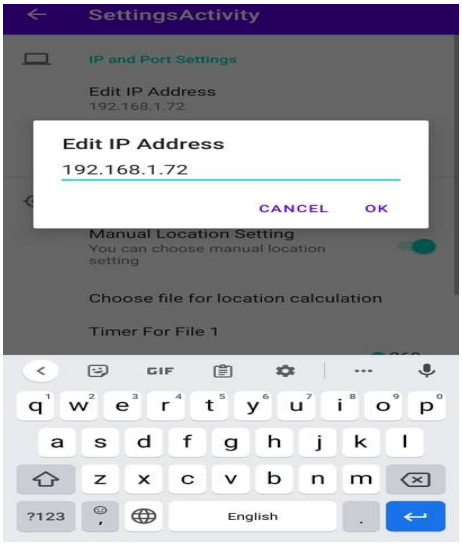
- Η εφαρμογή μας έχει ένα μενού όπως ζητείται από την εκφώνηση. Συγκεκριμένα, υπάρχουν δύο επιλογές, μία για τις ρυθμίσεις και μία για την έξοδο από την εφαρμογή. Το μενού υλοποιείται κάνοντας inflate το αντίστοιχο xml στην onCreateOptionsMenu().

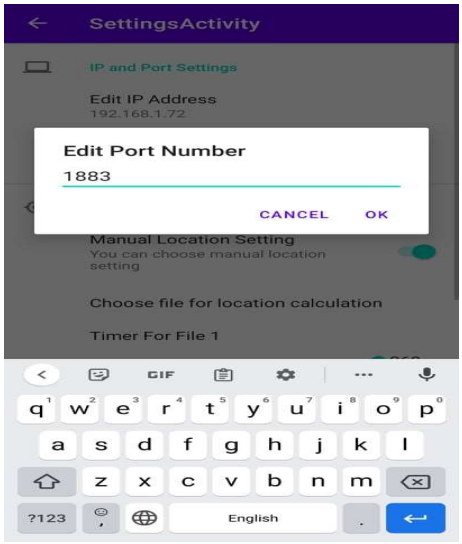
- Η επιλογή ενός εκ των δύο (settings και exit) υλοποιείται μέσω της onOptionsItemSelected(). Στην περίπτωση της εξόδου, δημιουργούμε έναν alertdialog που ρωτά τον χρήστη αν θέλει να τερματίσει την εφαρμογή. Σε περίπτωση θετικής απάντησης η εφαρμογή τερματίζει.



- Στην περίπτωση της επιλογής των ρυθμίσεων, ξεκινάμε μια καινούρια Settings activity. Επιλέγουμε να χρησιμοποιήσουμε το fragment PreferenceFragmentCompat ως εξής:
  Κάθε φορά που γίνεται η επιλογή settings, ξεκινάμε την Settings Activity η οποία γίνεται replace με το settingsfragment, που κάνει setPreferencesFromResource() από το αντίστοιχο xml. Το PreferenceFragmentCompat, τα στοιχεία του οποιου θα αναλύσουμε παρακάτω, καθώς και οι τιμές αυτών αποθηκεύονται σε ένα αρχείο sharedPreferences, επομένως κάθε φορά που ξεκινάμε τη Settings Activity, έχουμε μια ενημερωμένη έκδοση του και μπορούμε να εφαρμόσουμε τις αλλαγές που κάνει ο χρήστης.
- Για την υλοποίηση των ρυθμίσεων έχουμε φτιάξει 2 edit text preferences, μία για την IP και μια για την Port του broker. Οι default τιμές τους είναι “tcp://broker.hivemq.com:” και “1883”. Έχουμε χρησιμοποιήσει τον public broker στη διεύθυνση αυτή για εύκολο και αυτόνομο από τον server debugging και αποφασίσαμε να τον αφήσουμε default σε περίπτωση που θέλετε να κάνετε κάτι αντίστοιχο.

  Έχουμε φτιάξει επίσης ένα switch preference που «κλειδώνει» και «ξεκλειδώνει» την χειροκίνητη επιλογή τοποθεσίας. Ξεκλειδώνοντας τις χειροκίνητες επιλογές έχουμε μια list preference με τα δύο αρχεία και τον αντίστοιχο slider-μετρητή για το χρονικό διάστημα που επιθυμεί ο χρήστης όπως ζητάτε από την εφαρμογή. Επιλέξαμε να μην αφήσουμε στην τύχη την επιλογή του αρχείου, αφενός γιατί δεν έχει πολύ νόημα και αφετέρου για να μπορεί να γίνει εύκολα από τον χρήστη έλεγχος σωστών τιμών και παρακολούθηση της αλλαγής της θέσης του. Η εφαρμογή εξασφαλίζει ότι ο χρήστης μπορεί να αλλάζει με επιτυχία από το ένα αρχείο σε άλλο και από αυτόματο σε χειροκίνητο τρόπο.





- Ολοκληρώνοντας το μενού ρυθμίσεων θα αναλύσουμε το επόμενο στοιχείο που είναι ο αυτόματος και χειροκίνητος τρόπος επιλογής τοποθεσίας. Για τον αυτόματο τρόπο, έχουμε υλοποιήσει ένα service με όνομα location service. Χρησιμοποιούμε 4 κλάσεις από την δημόσια βιβλιοθήκη της google, com.google.android.gms.location.

  Με την εγκατάσταση της εφαρμογής ζητάται από τον χρήστη η άδεια χρήσης της τοποθεσίας του. Εάν αυτή μας δοθεί, τότε ξεκινάμε το service, από τη MainActivity, με τη χρήση της startLocationService(), δείχνοντας και το ανάλογο μήνυμα στο χρήστη, για επαλήθευση της αυτόματης επιλογής. Ξεκινώντας το service, χρησιμοποιούμε την κλάση LocationResult, για να πάρουμε την τελευταία τοποθεσία του τερματικού. Όσο η επιλογή μας για την τοποθεσία του τερματικού είναι αυτόματη, το service τρέχει και ελέγχει συνεχώς για αλλαγές στην τοποθεσία, τις οποίες καταγράφει και αποθηκεύει στο αρχείο των shared preferences. Όταν θέλουμε χειροκίνητο τρόπο επιλογής τοποθεσίας, η service σταματάει (καλούμε την stopLocationService()). Την ίδια διαδικασία ακολουθούμε και όταν τερματίζουμε την εφαρμογή.

- Η χειροκίνητη επιλογή υλοποιείται ως εξής:
  Στα assets της εφαρμογής, έχουμε τα δύο αρχεία xml που πρέπει να μετατραπούν σε csv. Με την έναρξη της, η MainActivity καλεί την Parse(), για τα δύο αυτά αρχεία.

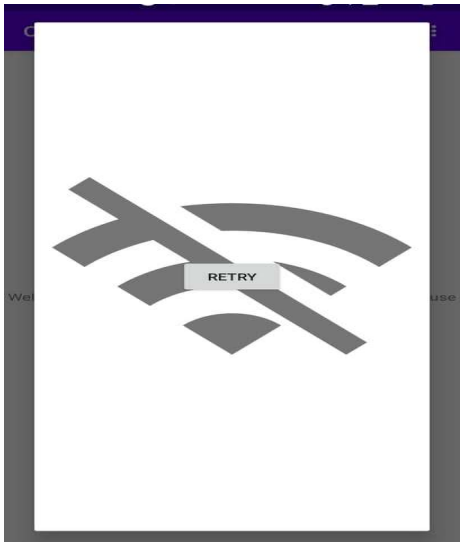
  Χρησιμοποιούμε τον XML Pull Parser, ένα συνιστώμενο -από τους developers του android studio- interface που ορίζει μία xml ανάλυση. Για κάθε αρχείο δημιουργούμε μια λίστα τύπου timestep, η οποία αποτελεί μια κλάση με στοιχεία τη χρονική στιγμή, το γεωγραφικό μήκος και πλάτος.

  Μετά το τέλος του αρχείου, δημιουργούμε ένα csv αρχείο για το αντίστοιχο xml, και χρησιμοποιούμε τη λίστα των timesteps για να γράψουμε στο αρχείο τα στοιχεία που θέλουμε(γεωγραφικές συντεταγμένες και δευτερόλεπτα). Τα στοιχεία χωρίζονται με κόμμα και μετά από κάθε timestep αλλάζουμε γραμμή. Χρησιμοποιούμε outputStream. Τα αρχεία csv αποθηκεύονται στο φάκελο Downloads της συσκευής μας. Εδώ πρέπει να σημειώσουμε ότι κατά την εγκατάσταση και εκκίνηση της εφαρμογής, ζητείται από τον χρήστη η αντίστοιχη άδεια για δημιουργία αρχείων στη συσκευή. Επιλέξαμε να αποθηκεύονται δυναμικά και να μην τα συμπεριλάβουμε στα assets για να μπορεί να γίνει έλεγχος ορθότητας των αρχείων, όμως αυτή η επιλογή έχει ένα μειονέκτημα. Κάθε φορά που χρησιμοποιούμε την εφαρμογή, τα αρχεία θα πρέπει να διαγράφονται από τη συσκευή μας, για να λειτουργήσει σωστά την επόμενη φορά.

Στη συνέχεια, χρησιμοποιούμε την readfile() για τα δύο αρχεία. Η readfile μπαίνει στον φάκελο των Downloads και ψάχνει το αντίστοιχο csv. Εάν αυτό υπάρχει, χρησιμοποιούμε την κλάση CsvReader, της δημόσιας βιβλιοθήκης openCSV για να διαβάσουμε το αρχείο. Κάθε γραμμή είναι και ένα timestep, όπως είδαμε πριν και έτσι την επεξεργαζόμαστε. Χρησιμοποιώντας το readnext() του CSVReader, φτιάχνουμε ένα object τύπου timestep, το οποίο αποθηκεύεται σε μια λίστα τύπου timestep, rows_1 ή rows_2, ανάλογα με το csv.

Στο εξής, θα παίρνουμε τις συντεταγμένες απευθείας από τη λίστα, χωρίς να χρειάζεται να διαβάσουμε το αρχείο. Με αυτόν τον τρόπο θεωρούμε ότι μειώνουμε την υπολογιστική αναγκαιότητα της εφαρμογής, αφού δεν χρειάζεται να διαβάζει συνεχώς από έναν εξωτερικό φάκελο κάποιο αρχείο.

- Για τον έλεγχο της σύνδεσης στο διαδίκτυο, έχουμε υλοποιήσει έναν broadcast receiver τον NetworkCg. Ο receiver γίνεται register στην onStart() της MainActivity και σε περίπτωση αποτυχίας σύνδεσης με το διαδίκτυο, εμφανίζει ένα alertdialog με κουμπί επιλογής retry. Εάν αποκατασταθεί η σύνδεση πρέπει να πατήσουμε το retry για να συνεχίσουμε την εκτέλεση της εφαρμογής.



- Προχωράμε αναλύοντας την MainActivity και τα function της που δεν έχουμε αναλύσει ήδη. Στην onCreate() έχουμε υλοποιήσει έναν listener, ο οποίος ελέγχει για αλλαγές προερχόμενες από το settings. Οι αλλαγές διαβάζονται από το αρχείο των sharedPreferences και αποθηκεύονται σε global μεταβλητές, πχ IP, Port για να χρησιμοποιηθούν από την εφαρμογή αντίστοιχα.

- Η κύρια λειτουργία της εφαρμογής είναι προφανώς η επικοινωνία με τον broker. Η σύνδεση με αυτόν γίνεται μέσω της ConnectToBroker(), η οποία

παίρνει τα global string IP, Port και τα ενώνει, για να συνδεθεί στη σωστή διεύθυνση. Οι αλλαγές σε αυτά τα string, που κάνει ο χρήστης, μας οδηγούν στην DisconnectFromBroker() και καλούμε ξανά την Connect που διαβάζει εκ νέου τη διεύθυνση του broker και συνδέεται. Σε περίπτωση επιτυχούς σύνδεσης δημιουργούμε έναν καινούριο mqtt client και ενεργοποιούμε ξανά τη δυνατότητα για να κάνουμε publish και subscribe.

- Η αποστολή διανυσμάτων από τον Client υλοποιείται μέσω ενός handler που τρέχει κάθε δευτερόλεπτο. Κάθε δευτερόλεπτο, ελέγχουμε εάν είμαστε συνδεδεμένοι με τον Broker. Στη συνέχεια, ελέγχουμε τον τρόπο επιλογής τοποθεσίας. Η επιλογή αυτή αποθηκεύεται από το settings σε ένα Boolean sharedPreference και από εκεί σε μία global Boolean μεταβλητή. Σε περίπτωση αυτόματης επιλογής οδηγούμαστε κατευθείαν σε publish. Σε περίπτωση χειροκίνητης επιλογής, έχουμε υλοποιήσει έναν switch που ελέγχει την διακοπή της αποστολής των διανυσμάτων. Εάν η επιλογή είναι true, τότε περνάμε σε publish.

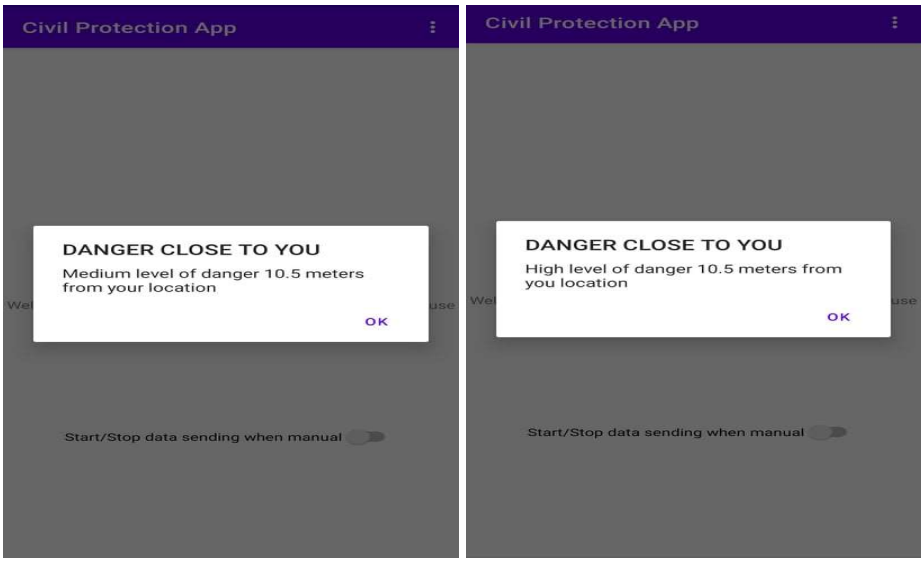
  Εδώ υπάρχει μια λεπτομέρεια που πρέπει να αναφέρουμε. Η εκφώνηση ζητά σε δύο σημεία δύο διαφορετικά πράγματα, την διακοπή αποστολής διανυσμάτων και τον τερματισμό της αποστολής διανυσμάτων, αναφερόμενη στην ίδια λειτουργία. Θεωρήσαμε ότι έπρεπε να υλοποιήσουμε έναν τρόπο να γίνεται start-pause και όχι start-stop, αφενός γιατί υπάρχει η επιλογή αλλαγής μεταξύ των αρχείων αλλά και επειδή θεωρήσαμε χρησιμότερο να έχουμε έναν τρόπο να συνεχίσουμε την εκτέλεση της εφαρμογής από το διάνυσμα που είχαμε σταματήσει.

- Στον handler δημιουργούμε δύο μεταβλητές στο αρχείο shared preferences, τύπου int. Εκεί αποθηκεύουμε τον μετρητή του δευτερολέπτου του διανύσματος που θα στείλουμε. Σε περίπτωση που ο τρέχων μετρητής ξεπεράσει την επιθυμητή τιμή χρόνου εκτέλεσης αρχείου, που ορίζει ο χρήστης από τα settings, δεν κάνουμε publish.

- Η συνάρτηση publish() είναι υπεύθυνη για τα mqtt μηνύματα που στέλνονται στον broker. Η συνάρτηση καλείται μέσω του handler κάθε δευτερόλεπτο. Μέσα στη συνάρτηση δημιουργούμε ένα καινούριο JSONObject. Εκεί,αρχικά αποθηκεύουμε το αναγνωριστικό της συσκευής. Επιλέξαμε το androidID, τον 15ψήφιο αναγνωριστικό αριθμό της συσκευής, τον οποίο και παίρνουμε στη main. Στη συνέχεια, σε περίπτωση αυτόματης επιλογής τοποθεσίας, ανακτούμε τις shared preferences longitude, latitude(όπως αναφέραμε παραπάνω) και τις αποθηκεύουμε στο JSONObject.
  Σε περίπτωση που έχουμε χειροκίνητη επιλογή, ελέγχουμε από ποιο αρχείο επιθυμεί ο χρήστης να γίνει αποστολή διανυσμάτων, παίρνουμε το τρέχον δευτερόλεπτο από την αντίστοιχη global int μεταβλητή(όπως αναφέραμε

παραπάνω) και βρίσκουμε το αντίστοιχο timestep που έχουμε αποθηκεύσει στη θέση αυτή. Από εκεί ανακτούμε τις συντεταγμένες και τις αποθηκεύουμε στο JSONObject.Και στις δύο περιπτώσεις, δημιουργούμε ένα καινούριο mqtt message και σαν payload περνάμε το JSONObject μας ως string.

- Subscribe, σε ένα topic γίνεται μέσα στην ConnectToBroker(), με τους κατάλληλους ελέγχους. Σε περίπτωση που λάβουμε μήνυμα κινδύνου από τον broker, καλούμε την dangerMessage(). Εκεί γίνεται έλεγχος για το επίπεδο του κινδύνου και αντίστοιχα εκτυπώνεται στον χρήστη το κατάλληλο οπτικουαστικό μήνυμα.



IoT Device

Τα περισσότερα στοιχεία των εφαρμογών μας είναι κοινά. Θα επικεντρωθούμε λοιπόν σε αυτά τα οποία δεν είναι.

- Ξεκινώντας από το μενού, επιλέξαμε να έχουμε ένα μενού δύο επιλογών και όχι τριών, όπως ζητείται από την εκφώνηση. Η τρίτη επιλογή, η πρόσθεση νέου αισθητήρα, γίνεται με τρόπο που θα εξηγήσουμε παρακάτω. Η μόνη διαφορά μεταξύ Client και IoT συσκευής, όσων αφορά τα μενού και τις επιλογές αυτών, είναι στο κομμάτι των ρυθμίσεων της χειροκίνητης επιλογής. Στην εφαρμογή Sensors, με το «ξεκλείδωμα» της χειροκίνητης επιλογής, έχουμε να επιλέξουμε μία εκ των τεσσάρων δοθέντων τοποθεσιών. Έχουμε ορίσει ως default τη μία εξ αυτών, για να διασφαλίσουμε ότι εάν δεν γίνει επιλογή θέσης από απροσεξία η εφαρμογή θα συνεχίσει να στέλνει σωστά δεδομένα.
- Σε περίπτωση χειροκίνητης επιλογής, οι shared Preferences longitude, latitude, αλλάζουν, ανάλογα με την επιλογή του χρήστη, μέσω της getManualPos().

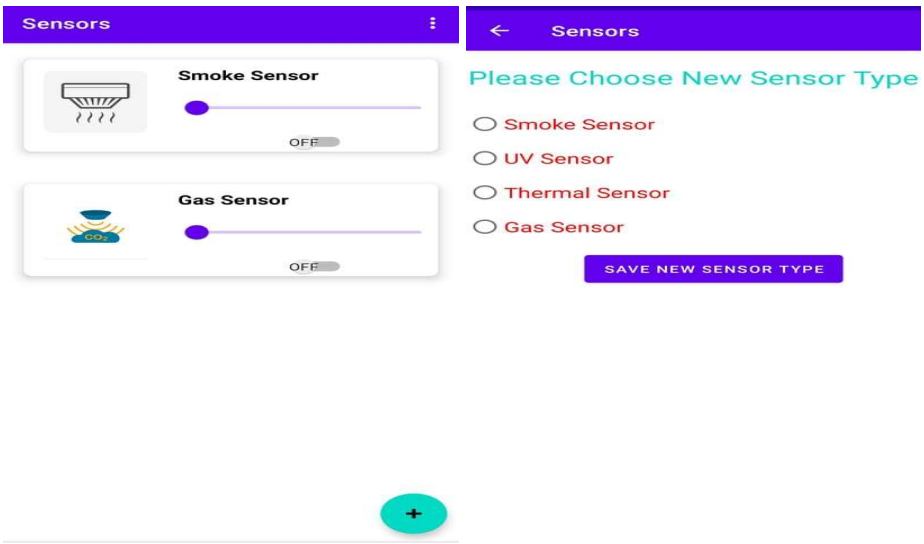
- Η δημιουργία νέου αισθητήρα είναι το σημείο της εργασίας που θεωρούμε ότι υπάρχει η μεγαλύτερη ευελιξία ως προς την υλοποίηση της. Για τη δημιουργία του αισθητήρα η σειρά των βημάτων έχει ως εξής:
- Από τη main πατάμε στο floating button(+). Αυτό μας κατευθύνει σε ένα νέο Activity, το PickSensorType. Έχουμε υλοποιήσει ένα πολύ απλό interface στο οποίο ο χρήστης επιλέγει μία από τις 4 επιλογές αισθητήρα.

  Την PickSensorType , την έχουμε ξεκινήσει από τη main με την startForResult. Όταν αποθηκεύσουμε την επιλογή μας για τον νέο αισθητήρα, η PickSensorType γυρίζει ένα αποτέλεσμα τύπου string, στην main.

  Ανάλογα με την επιλογή του χρήστη, καλείται η addSensor() με το αντίστοιχο όρισμα. Η addSensor με τη σειρά της, τοποθετεί σε μία δυναμική λίστα τύπου Sensors τον καινούριο αισθητήρα. Κάθε αισθητήρας έχει 4 στοιχεία, τον τύπο του σε string, το logo του, τη τιμή του και μια Boolean μεταβλητή που έχει χαρακτήρα on/off. Για τη γραφική απεικόνιση των αισθητήρων, χρησιμοποιούμε ένα cardview. Αυτό περιέχει τον τύπο του αισθητήρα, ένα slider με το εύρος των τιμών του, το logo του και ένα switch ενεργοποίησης/απενεργοποίησης

  Η λίστα μας «ζει» μέσα σε ένα RecyclerView. Κάθε φορά που προστίθεται ένας καινούριος αισθητήρας σε αυτήν, ο adapter του RecyclerView κάνει bind μεταξύ του αντικειμένου τύπου Sensor με έναν ViewHolder. Κατά τη διάρκεια αυτή ελέγχει τον τύπο του, και ανάλογα αλλάζει το εύρος του slider (πχ -5 έως 80 βαθμοί κελσίου). Κάθε φορά που ο χρήστης θέλει να αλλάξει την τιμή του αισθητήρα, ή να τον απενεργοποιήσει, ο ViewHolder που έχει γίνει bind με το αντίστοιχο Sensor Object, αντιλαμβάνεται τις αλλαγές και αλλάζει τις τιμές του.

  Με την υλοποίηση αυτή έχουμε σημαντικά πλεονεκτήματα. Από τη μία έχουμε μειώσει κατά πολύ την υπολογιστική αναγκαιότητα, αφού οι αισθητήρες μας δεν είναι τίποτα άλλο από αντικείμενα σε μία λίστα. Από την άλλη, μπορούμε να έχουμε ένα τεράστιο αριθμό αισθητήρων, ταυτόχρονα.



- Περνώντας στο κομμάτι του publish, πρέπει να αναφέρουμε ότι χρησιμοποιούμε ακριβώς την ίδια διαδικασία όπως και στον client. Εάν συνδεθούμε, κάθε δευτερόλεπτο μέσω ενός handler, καλούμε τη συνάρτηση pub() που κάνει publish. Ως ξεχωριστό topic της κάθε IoT συσκευής, χρησιμοποιούμε και πάλι το androidID, ώς εξής “IoT/androidID”. Ως payload του mqtt message,χρησιμοποιούμε εμφωλευμένα JSON. Στο εξωτερικό αντικείμενο, αποθηκεύουμε τις γεωγραφικές συντεταγμένες, το androidID της συσκευής και το ποσοστό της μπαταρίας. Το τελευταίο το ανακτούμε χρησιμοποιώντας την κλάση Battery Manager του android λειτουργικού. Στο εμφωλευμένο object, δημιουργούμε ένα JSON array, με καταχωρήσεις JSON object, ένα για κάθε ενεργό αισθητήρα. Για κάθε στοιχείο της λίστας από αισθητήρες, ελέγχουμε εάν είναι ενεργός, και σε θετική περίπτωση ανακτούμε τις τιμές του και τις αποθηκεύουμε στο JSON object. Συγκεκριμένα ανακτούμε τον τύπου του, την τιμή του και τη θέση του στη λίστα(για να διευκολύνουμε τον server σε περιπτώσεις πολλαπλών αισθητήρων του ίδιου είδους). Τέλος, το εξωτερικό αντικείμενο, ορίζεται ως payload του μηνύματος και γίνεται publish.

Παρουσίαση Edge Server

Το κομμάτι του Edge Server, σε αντίθεση με αυτό των android εφαρμογών, είναι reactive. Η γενική λειτουργία είναι η αναμονή για λήψη μηνυμάτων μέσω του MQTT broker και ειδοποίηση σε περιπτώσεις κινδύνου. Όλες οι πληροφορίες που λαμβάνονται από την επικοινωνία μέσω του MQTT broker αποθηκεύονται σε βάση δεδομένων (mySQL/mariaDB). Πιο αναλυτικά:

Βάση Δεδομένων

Η βάση δεδομένων έχει 2 tables. Στο ένα αποθηκεύονται τα devices με τις σχετικές τους πληροφορίες όπως longitude, latitude, deviceID κ.λπ. Στο άλλο αποθηκεύονται τα events με, αντίστοιχα, όλες τις μετρήσεις που έρχονται από τις συσκευές IoT μαζί με απαραίτητες πληροφορίες όπως το timestamp, το deviceID και sensorID κ.λπ. Τέλος, σημειώνεται ότι για να πραγματοποιηθεί η επικοινωνία με τη βάση έχει χρησιμοποιηθεί ο JDBC driver.

Σύνδεση με MQTT broker

Κάνοντας κάποιες βασικές ρυθμίσεις συμπεριλαμβανομενου του ορισμού port (1883) και της callback function (βλ. παρακάτω), εγκαθιδρύεται η σύνδεση με την MQTT υπηρεσία. Μεταξύ άλλων κατά την εγκαθίδρυση της σύνδεσης με τον MQTT broker ορίζεται ο μέγιστος αριθμός μηνυμάτων που θα αποστέλλεται χωρίς να δοθεί επιβεβαίωση της παραλαβής του μηνύματος (setMaxInflight) αλλα και κατά πόσο θα διατηρείται το MQTT session μετά από πιθανές επανεκκινήσεις και επανασυνδέσεις του σέρβερ (setCleanSession).

MQTT Callback

Η callback function είναι μια συνάρτηση μέσω της οποίας ο Edge Server θα διαχειρίζεται events που έχουν να κάνουν με τη σύνδεση του στον MQTT broker.

Συγκεκριμένα, έχουν γίνει Override οι ορισμοί των συναρτήσεων για την περίπτωση:

a) Διακοπής σύνδεσης
b) Παραλαβής μηνύματος
c) Ολοκλήρωσης αποστολής μηνύματος

Βασικές λειτουργικότητες

Κατά τη ροή των εισερχομενων μηνυμάτων ο Edge Server πρέπει να διεκπεραιώσει κάποιες βασικές λειτουργίες. Αυτές πραγματοποιούνται μέσα από τις παρακάτω συναρτήσεις:

- **fnParseJsonFile** - Πρόκειται για τη συνάρτηση που καλείται σε μεγάλο βαθμό να ερμηνεύσει τα μηνύματα που έρχονται μέσω του MQTT broker. Τα μηνύματα αυτά έχουμε προβλέψει - σε συνεργασία με τους δημιουργούς της εφαρμογής για τις IoT συσκευές - ότι θα είναι σε μορφή JSON. Έτσι, χρειαζόμαστε μια συνάρτηση για να πάρουμε τις τιμές από το JSON object. Στην πραγματικότητα η συνάρτηση περιέχει και την αποθήκευση πληροφοριών για events και devices στη βάση δεδομένων.
- **fnComputeDangerLevel** - Η συνάρτηση αυτή λαμβάνει ως arguments τις τιμές 4 μετρήσεων διακριτού τύπου και υπολογίζει ποιο είναι το επίπεδο κινδύνου με βάση αυτές. Επιστρέφει 0 στην κανονικότητα, 1 για μέτριο

κίνδυνο και 2 για υψηλό κίνδυνο. Σημειώνουμε ότι, ενώ οι IoT συσκευές μπορεί να στέλνουν πλήθος μετρήσεων (π.χ. 2 ή και περισσότερες μετρήσεις από 2+ διαφορετικούς αισθητήρες ακτινοβολίας), έχουμε θεωρήσει ότι στα πλαίσια της υλοποίησης γίνεται δεκτή μόνο η μέγιστη από τις τιμές που έχουν σταλεί.
- **fnAlertAboutDangerLevel** - Χρησιμοποιείται όταν παρατηρείται επίπεδο κινδύνου με βάση τις μετρήσεις που λαμβάνονται από κάποια συσκευή IoT. Για την ενημέρωση γίνεται publish στο συμφωνηθέν topic ώστε να ληφθεί κανονικά από την android συσκευή.

Asset Functions

Κατά τη διαδικασία της παραλαβής μηνυμάτων ο Edge Server πρέπει να κάνει διάφορες διεργασίες, οι οποιες για καλύτερη οργάνωση του κώδικα έχουν χωριστεί σε συναρτήσεις. Περιληπτικά:

- **fnGetDatabaseConnection** - εγκαθιδρύει επικοινωνία με τη mySQL βάση δεδομένων και επιστρέφει μια μεταβλητή τύπου Connection. Χρησιμοποιείται από άλλες συναρτήσεις όταν χρειάζεται να γίνει κάποιο insert στη βάση δεδομένων.
- **fnGetTimestamp** - επιστρέφει την ημερομηνία τη στιγμή που καλείται. Τη χρειαζόμαστε κάθε φορα που γίνεται νέα εγγραφή κάποιου event που απαιτεί timestamp.
- **fnCalculateDistanceFromAndroid -** Στη συνάρτηση δίνονται οι συντεταγμένες της/των IoT συσκευής/συσκευών και υπολογίζεται η απόσταση από το σημείο στο οποίο βρίσκεται η android συσκευή. Χρησιμοποιείται όταν πρέπει να δοθεί σήμα κινδύνου, όπου η android συσκευή ενημερώνεται για την απόσταση από το IoT που σημείωσε το επίπεδο κινδύνου.
- **fnGetAndroidLocation -** Αυτή η συνάρτηση παίρνει τη θέση της android συσκευής από τη βάση και την επιστρέφει. Χρησιμοποιείται για να μπορούμε να δούμε τη θέση του android σχετικά με τις IoT συσκευές, όπως και την απόσταση μεταξύ τους.
- **executeVoidSqlQuery** - Μια απλή συνάρτηση που κάνει ένα query στη βάση και δε περιμένει κάποιο αποτέλεσμα. Χρησιμοποιείται για απλούστευση και κανονικοποίηση της διαδικασίας των INSERT.
- **fnGetLastEntry** - Συνάρτηση για εσωτερική χρήση και debugging.

Οπτικοποίηση

Για την οπτικοποίηση των αποτελεσμάτων έχουμε δημιουργήσει - στη λογική αρχιτεκτονικής τριών επιπεδων - ένα frontend με το Svelte framework, το οποίο χρησιμοποιεί το google maps API για την εμφάνιση χάρτη. Αυτό επικοινωνεί με τη βάση δεδομένων (mySQL/mariaDB) μέσω rest API υλοποιημένο σε Node.

Backend

Το API είναι αρκετά απλό στη λειτουργικότητα του. Το σημαντικό κομμάτι είναι η σύνδεση με τη βάση. Έχουν δημιουργηθεί 2 routes, τα /events και /devices ώστε να μπορούν να επιστρέφονται τα στοιχεια που έχουν ληφθεί από τα publish του android και έχουν αποθηκευτεί στους αντίστοιχους πίνακες.

Frontend

Το frontend αποτελείται εξ ολοκληρου από έναν χάρτη και τα περιεχόμενα του, τα οποια παρέχονται από το google maps api.

Πιο συγκεκριμένα, μέσα στο χάρτη εμφανίζονται τα παρακάτω:

a) markers για τις IoT συσκευές. Κάθε IoT συσκευή θα έχει διαφορετικό icon ανάλογα με το επίπεδο κινδύνου όπως και περικλείεται από ένα κύκλο διαφορετικού χρώματος ανάλογα με το αν η συσκευή είναι ενεργή η όχι. Σε περίπτωση κινδύνου και στις 2 IoT συσκευές, εμφανίζεται ένα area κινδύνου που ορίζεται από τα 2 αυτά σημεία.
b) marker για την android συσκευή με το αντίστοιχο εικονίδιο για να διαχωρίζεται από τις συσκευές IoT.

Εκτέλεση

Το σύστημα που περιγράψαμε αποτελείται από αρκετά διακριτά κομμάτια. Παρακάτω αναφέρουμε προαπαιτούμενα αλλα και συγκεκριμένες οδηγίες για την εκτέλεση τους.

Edge Server

Προαπαιτούμενο για την εύρυθμη λειτουργία του Edge Server είναι να βρίσκονται active τα services του MQTT client αλλα και της mySQL.

Για τη λειτουργία του Edge Server έχει δημιουργηθεί Makefile, οποτε στο root (/) των αρχείων απλώς κάνουμε <make>.

Node Backend

Το source code του Node API είναι το /rest_api κομμάτι στα αρχεία του project. προαπαιτουμενη είναι η εγκατασταση της nodejs (v11+) και του αντιστοιχου package manager npm.

Αρχικά θα πρέπει να κάνουμε <cd rest_api> και <npm install>. Αυτή η διαδικασια εγκαθιστα ολα τα απαραιτητα πακετα μεσω του node package manager. Στη συνεχεια κάνουμε <node server.js> για να “τρέξουμε” το API.

Αυτή τη στιγμή είναι ρυθμισμένο να διατίθεται στο port 4000.

Svelte Frontend

Το source code του Svelte Frontend είναι το /svelte_front κομμάτι στα αρχεία του project. προαπαιτουμενη einai η εγκατασταση ths nodejs (v11+) και του αντιστοιχου package manager npm.

Αρχικά θα πρέπει να κάνουμε <cd rest_api> και <npm install>. Αυτή η διαδικασία εγκαθιστά όλα τα απαραίτητα πακέτα μέσω του node package manager. Στη συνεχεια κάνουμε <npm run dev> για να “τρέξουμε” το localhost.

Αυτή τη στιγμή είναι ρυθμισμένο να διατίθεται στο port 8080 και άρα το αποτέλεσμα μπορεί να το δει κανείς σε browser, στο http://localhost:8080/