Εθνικό και Καποδιστριακό Πανεπιστήμιο Αθηνών
Τμήμα Πληροφορικής και Τηλεπικοινωνιών

Ανάπτυξη Λογισμικού για Δίκτυα και Τηλεπικοινωνίες
Project

Ονοματεπώνυμο: Γιώργος Κολομβάκης
Κυριακή Καββαθά
Σταύρος Ατάτα
Νικόλαος Τρικοίλης
Ανδριάνα Βέρα

Project: Γενική Εισαγωγή

Το project αναπτύχθηκε με στόχο τη δημιουργία ενός Περιβαλλοντικού Διαχειριστικού Συστήματος για την ασφάλεια των πολιτών σε καταστάσεις έκτακτης ανάγκης. Το σύστημα εκμεταλλεύεται Android και IoT συσκευές για τη συλλογή και ανταλλαγή δεδομένων μέσω MQTT, ενώ ο Edge Server αναλύει τα δεδομένα για τον εντοπισμό κινδύνων και ενημερώνει τους χρήστες. Η αποθήκευση ανιχνεύσεων κινδύνου γίνεται σε μια MySQL βάση. Το σύστημα υποστηρίζει δυναμική αποστολή και λήψη γεωγραφικών δεδομένων, ενώ παρέχει γραφική διεπαφή για την επισκόπηση της κατάστασης των συσκευών και των ανιχνευθέντων κινδύνων. Το project ενθαρρύνει την ευελιξία και την παραμετροποίηση μέσω ρυθμίσεων στις Android και IoT εφαρμογές.

Edge Server

Γενικά

΄Ενας edge server είναι ένας διακομιστής που βρίσκεται στο περιφερειακό μέρος ενός δικτύου, πλησιέστερα στα τελικά σημεία χρήστη ή στις συσκευές πηγής δεδομένων. Ο βασικός σκοπός του είναι να παρέχει περιεχόμενο και υπηρεσίες πιο γρήγορα, μειώνοντας την καθυστέρηση, και να μειώσει το φορτίο στο κεντρικό δίκτυο επεξεργάζοντας δεδομένα πιο κοντά στην πηγή τους.

Ειδικότερα

Για να υλοποιηθεί ο συγκεκριμένος Edge Server του Project πρώτα πρώτα υλοποιήθηκε η Βάση Δεδομένων:

1 Βάση Δεδομένων

Η βάση δεδομένων περιέχει έναν πίνακα με την ονομασία **Events**. Αυτός ο πίνακας καταγράφει δεδομένα όπως:

- **IoT device id**: Το μοναδικό αναγνωριστικό της συσκευής IoT.
- **longitude**: Το γεωγραφική μήκος της τοποθεσίας του συμβάντος.
- **latitude**: Το γεωγραφική πλάτος της τοποθεσίας του συμβάντος.
- **smoke sensor**: Τιμές από τον αισθητήρα καπνού.
- **gas sensor**: Τιμές από τον αισθητήρα αερίων.
- **temp sensor**: Τιμές από τον αισθητήρα θερμοκρασίας.
- **uv sensor**: Τιμές από τον αισθητήρα UV.
- **danger degree**: Ο βαθμός κινδύνου του συμβάντος.

1.1 Κλάση **ConnectionToDB**

Η κλάση ConnectionToDB χρησιμοποιείται για τη σύνδεση του edge server με τη βάση δεδομένων. Παρέχει λειτουργίες για τη σύνδεση και την αποσύνδεση από τη βάση, καθώς και για τον έλεγχο της κατάστασης της σύνδεσης.

1.2 Κλάση **Create_and_Insert_Methods**

Αυτή η κλάση περιέχει μεθόδους για τη δημιουργία και εισαγωγή νέων εκδηλώσεων στη βάση δεδομένων.Υποδηλώνει την χρήση των αντικειμένων της κλάσης ConnectionToDB για τη διαχείριση της σύνδεσης με την βάση δεδομένων κατά την εισαγωγή δεδομένων.

---

2 **Calculator**

Παρουσιάζουμε μια σειρά από κλάσεις Java που αναπτύχθηκαν για τη διαχείριση και ανάλυση δεδομένων που προέρχονται από συσκευές IoT και Android, καθώς και για την αξιολόγηση και επεξεργασία κινδύνων που ενδέχεται να ανακύψουν σε πραγματικό περιβάλλον.

2.1 **CreateJson**

Η κλάση CreateJson είναι υπεύθυνη για τη δημιουργία JSON αρχείων από τα δεδομένα που λαμβάνει. Χρησιμοποιεί τις πληροφορίες όπως το ID, και τις συντεταγμένες X και Y, για να δημιουργήσει μια δομή JSON που περιγράφει τη θέση και τον τύπο μιας συσκευής,όπως και την κατάσταση (στατυς) , τον βαθμό επικινδυνότητας μιας IoT συσκευής΄

2.2 **Distance_calc**

Η κλάση Distance calc παρέχει μεθόδους για τον υπολογισμό της γεωγραφικής απόστασης μεταξύ δύο σημείων, καθώς και για τον υπολογισμό του κέντρου μεταξύ δύο τοποθεσιών. Αυτές οι λειτουργίες είναι κρίσιμες για την κατανόηση και της απόστασης των συμβάντων.

2.3 DangerDegreeCalc

Η κλάση DangerDegreeCalc αξιολογεί τους βαθμούς κινδύνου με βάση δεδομένα από αισθητήρες όπως καπνού, αερίου, θερμοκρασίας και UV. Αυτή η κλάση παρέχει μια σημαντική λειτουργία ασφαλείας, αξιολογώντας τα επίπεδα κινδύνου και ενημερώνοντας τους χρήστες ανάλογα.

2.4 Calculator

Η κλάση Calculator αποτελεί τον πυρήνα που αφορά τον υπολογισμό των τιμών που λαμβάνουμε απο τις IoT συσκευών, καθώς συνδυάζει τις λειτουργίες των άλλων κλάσεων για την ανάλυση και την επεξεργασία δεδομένων. Υπολογίζει τον κίνδυνο από συμβάντα, χρησιμοποιώντας δεδομένα από Android και IoT συσκευών και αποστέλλει τις απαραίτητες πληροφορίες στους σχετικούς χρήστες.

---

3 MQTT

Το πακέτο MQTT περιλαμβάνει κλάσεις που εγκαθιδρούν την επικοινωνία μέσω του πρωτοκόλλου MQTT.

3.1 Κλάση Configuration

Η κλάση Configuration καθορίζει τις βασικές ρυθμίσεις για την επικοινωνία με τον MQTT broker και αρχικοποιεί τις κλάσεις Publisher και Subscriber.

3.2 Κλάση Client

Η κλάση Client αρχικοποιεί τους client που θα χρησιμοποιηθούν για την publish και subcribe , διαχειρίζεται τη σύνδεση, την αποσύνδεση και τις επαληθεύσεις σύνδεσης.

3.3 Κλάση Publisher

Η κλάση Publisher διαχειρίζεται τη δημοσίευση μηνυμάτων σε topics και χρησιμοποιεί μια ουρά για την αποθήκευση των δεδομένων προς αποστολή.

3.4 Κλάση Subscriber

Η κλάση Subscriber είναι μέρος του πακέτου MQTT και υλοποιεί την λειτουργικότητα του MQTT συνδρομητή. Αυτή η κλάση είναι υπεύθυνη για τη λήψη δεδομένων από διάφορες πηγές, όπως Android και IoT συσκευές, και την επεξεργασία τους.

- Σύνδεση με **MQTT Broker:** Αρχικοποιείται με συγκεκριμένο broker και client ID, και ρυθμίζει τον εαυτό της ως callback για τα εισερχόμενα μηνύματα.
- Διαχείριση Δεδομένων: Χρησιμοποιεί LinkedBlockingQueues (καταχώρηση σε ουρά) για την αποθήκευση και επεξεργασία δεδομένων από Android και IoT συσκευές.
- Επεξεργασία Μηνυμάτων: Η μέθοδος messageArrived αναλύει τα εισερχόμενα μηνύματα και τα ταξινομεί ανάλογα με την πηγή τους.Καλείται αυτόματα έπειτα απο την κλήση της Function Subscribe ωπου αναμένει μήνυμα στο ήδη ορισμένο topic
- Επικοινωνία με ΄Αλλες Κλάσεις: Συνεργάζεται με την κλάση Calculator για την αξιολόγηση κινδύνου και την κλάση Publisher για την αποστολή δεδομένων προς Android συσκευές.Επίσης με τις κλάσεις ConnectionToDB για την αποθήκευση δεδομένων στην βάση, CreateJson για την επικοινωνία μεταξύ Java και Javascript μέσω αρχείου Jason.

Κατά την λήψη ενός μηνύματος, η κλάση καθορίζει αν τα δεδομένα προέρχονται από Android ή IoT συσκευή .

΄Οταν ένα μήνυμα φτάνει από το MQTT broker, η μέθοδος messageArrived καλείται αυτόματα. Η μέθοδος αυτή:

- Διαχωρίζει τα μηνύματα ανάλογα με την πηγή τους (Android ή IoT συσκευές).
- Αποθηκεύει τα δεδομένα στις αντίστοιχες ουρές (LinkedBlockingQueue) για περαιτέρω επεξεργασία.
- Αξιοποιεί τη λογική και τις λειτουργίες που παρέχονται από την κλάση Calculator για να αξιολογήσει την κατάσταση κινδύνου.

Συγκεκριμένα, εάν λάβει δεδομένα από και τις δύο IoT συσκευές και Android συσκευή, υπολογίζει τον κίνδυνο βάσει αυτών των δεδομένων. Ανάλογα με την κατάσταση, δημιουργεί αντικείμενο dataToSend, το οποίο περιέχει τις πληροφορίες που πρέπει να αποσταλούν πίσω στην Android συσκευή.

Τέλος , η κλάση Subscriber καλείται ένα Thread για να επιτρέπεται η αναμονή παραλαβής μηνυμάτων ώπου αν περαστεί ο ορισμένος χρόνος τερματίζεται.

Ενώ η κλάση Publish καλεί τις συναρτήσεις startThread η οποία ξεκινά τον αντίστοιχο νήμα, ενώ η stopThread την τερματίζει ασφαλώς.

3.5 Κλάση **dataToSend**

Η κλάση dataToSend διαχειρίζεται τα δεδομένα που προορίζονται για αποστολή από τον edge server προς τις συσκευές. Περιλαμβάνει πληροφορίες όπως τα μηνύματα ειδοποίησης και η απόσταση.

3.6 Κλάση **received_android_data**

Η κλάση received_android data περιέχει δεδομένα που έχουν ληφθεί από Android συσκευές. Περιλαμβάνει το ID της συσκευής, τη θέση της και το επίπεδο της μπαταρίας της.

3.7 Κλάση **received_IoT_data**

Η κλάση received_IoT data διαχειρίζεται τα δεδομένα που έχουν ληφθεί από IoT συσκευές. Περιλαμβάνει πληροφορίες όπως το ID της συσκευής, τη θέση, το επίπεδο της μπαταρίας και διάφορες τιμές από αισθητήρες.

---

4 Maven

Παρουσιάζουμε τις βασικές πληροφορίες και τις εξαρτήσεις που ορίζονται στο Maven ‘pom.xml‘ του προγράμματος Java.

Το project χρησιμοποιεί την Apache Software License, Version 2.0.

Ορίζονται εξαρτήσεις για διάφορες βιβλιοθήκες, όπως:

- Eclipse Paho για MQTT
- Gson για JSON parsing
- Google API Client για πρόσβαση σε Google services
- MySQL JDBC Connector για εγκαθίδρυση της επικοινωνίας μεταξύ του Edge Server με την Database

Android Device Documentation

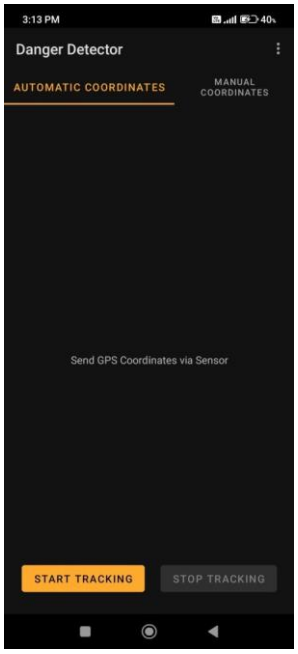
Η δημιουργία του Android προγράμματος πραγματοποιήθηκε με την βοήθεια του Android Studio σε γλώσσα Java και κάποιες έξτρα ρυθμίσεις που μπήκαν στα αντίστοιχα σημεία που ορίζει το Android Studio ώστε να έχουμε ομαλή λειτουργία.

Αρχικά πέρα από την δημιουργία του project όπου μας εκκινεί το Android Studio με κάποια generated αρχεία θα πρέπει να ορίσουμε κάποια πράγματα στο build σύστημα του το όποιο παρέχεται μέσω του εργαλείου Gradle που έρχεται πακέτο με το Android Studio.

Συγκεκριμένα αρχικά θα ορίσουμε τα απαραίτητα dependencies στο ***build.gradle.kts*** στο module app που αφορά το application μας και είναι τα εξής:

- ***AppCompat*** που αφορά τα βασικά components του Android όπως Activities, Fragments κ.λ.π
- ***Material Library*** που αφορά τα UI components του Android όπως Action Bar, χρώματα κ.λ.π
- ***ViewPager 2*** που αφορά την απεικόνιση των fragments μας σε μορφή swipe.
- ***Eclipse Paho*** που αφορά την βιβλιοθήκη για την υλοποίηση του πρωτοκόλου MQTT.

Το τελικό UI του app είναι το εξής:



Κατουσίαν έχουμε ενα Activity που φιλοξενεί 2 fragments που τα ονομάζουμε Automatic Coordinates και Manual Coordinates όπου στο πρώτο παίρνουμε τις συντεαγμένες μέσω gps και τις στέλνουμε μέσω του αντίστοιχου κουμπιού ενώ στη δεύτερη οθόνη πάμε κάνοντας swipe και στέλνουμε τις συντεταγμένες διαβάζοντας από τα αρχείο xml.

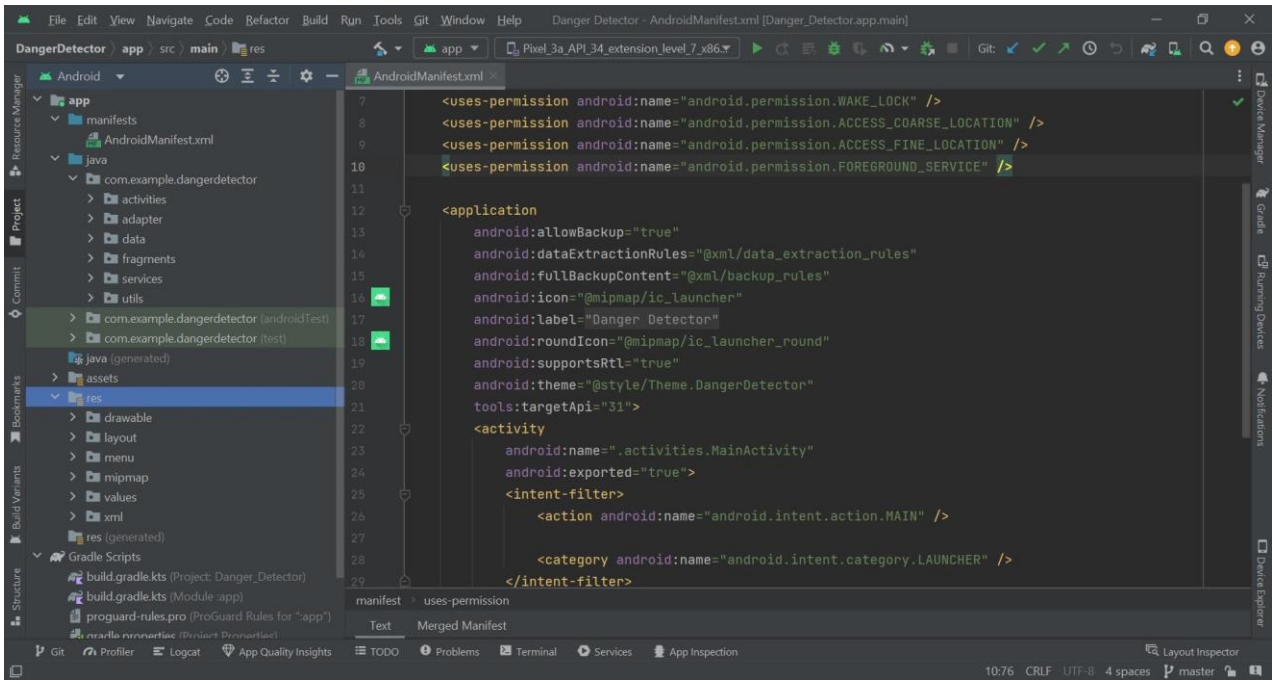
Ειδική αναφορά αξίζει να γίνει στο Paho καθώς για να λειτουργήσει σωστά θα πρέπει να βάλουμε την εξής εντολές στο ***gradle.properties***:
android.useAndroidX=true
android.enableJetifier=true

Αυτό το κάνουμε κάθως το Paho library είναι παλιό και δεν υποστηρίζει το Android Jetpack library που έχει μεταφερθεί το σύνολο των libraries του. Με αυτό τον τρόπο λέμε στο Android οτι παλιό έχει μέσα του το Paho να το αντικαθιστά με τα καινούργια packages που επιβάλει το Jetpack.

Τα κύρια packages / files που μας απασχολούν στην δημιουργία του app μας είναι το AndroidManifest.xml όπου αποτελεί εναν γενικό περιγραφέα του app μας και τι περιέχει όπως activities, εκδόσεις , services όπως το paho, permissions που θα χρειαστούν όπως Internet κλπ.

Συνεχίζοντας έχουμε το package java/com/example/package detector όπου εκεί φιλοξενείται ο κώδικας του app μας και οργανώνεται στα δικά μας packages αλλά και τα res όπου μέσα του είναι τα layouts δηλαδή τα UI που περιγράφει πως θα φαίνεται το γραφικό περιβάλλον της εφαρμογής.

Τέλος στα assets φιλοξενείται τα αρχεία xml με τις συντεταγμένες που μας έχουν δοθεί.



Αρχικά να αναφέρουμε το AndroidManifest.xml όπου εκεί μέσα θα προσθέσουμε τα permissions για Internet και την κατάσταση του, την τοποθεσία που αφορά GPS και Foreground Service που αφορά notification του service όταν ενεργοποιείται το service μας για το GPS.
Ορίζουμε το MainActivity ότι ενεργοποείται πρώτο ως launch activity και τέλος προσθέτουμε τα services του paho και του location καθώς πρέπει να αναφέρουμε οποιοδήποτε service έχουμε στο app.

Ο κώδικας του app έχει οργνωθεί στα παρακάτω packages

- *activites*
- *adapter*
- *data*
- *fragments*
- *services*
- *utils*

Στα activites βρίσκεται το MainActivity, στον adapter η class που είναι υπεύθυνη για την διαχείρηση των fragments.
Στο data βρίσκεται η μοντεολοποίηση που αφορά το parsing των xml αρχείων που έχουν δοθεί.
Στα fragments έχουμε τα δύο fragments που αφορούν τις δυο οθόνες που συμβαίνει η αλληλεπίδραση με τον χρήστη.
Στα services υλοπιείται η λειτουργία που αφορά το MQTT αλλά και η λειτουργία που αφορά το GPS της συσκευής.
Στο utils έχουμε προσθέσει τις λειτουργίες που αφορούν τον έλεγχο internet, το parsing των XML αρχείων αλλά και μια class με κάποια σημαντικά strings όπως το TOPIC για το MQTT.

IoT Documentation

Σε επίπεδο κώδικα:

Οι IoT συσκευές υλοποιήθηκαν με τη χρήση του Android Studio σε γλώσσα Java. Το build του προγράμματος προσδιορίζεται στο εργαλείο Gradle, συγκεκριμένα στο αρχείο **build.gradle.kts** προσθέτουμε τα dependencies:

- **App Combat**, για τη χρήση των Activities, Fragments etc.
- **Material**, για τα Action Bar και τα χρώματα.
- **View Pager 2**, για την κίνηση μεταξύ των Fragments
- **Eclipse Paho**, για την επικοινωνία με τον Edge Server μέσω του πρωτοκόλλου MQTT.

Στο αρχείο **AndroidManifest.xml** προσθέτουμε τα permissions για την τοποθεσία και την σύνδεση του internet:

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission
android:name="android.permission.ACCESS_NETWORK_STATE" />
<uses-permission
android:name="android.permission.WAKE_LOCK"/>
<uses-permission
android:name="android.permission.ACCESS_COARSE_LOCATION" />
<uses-permission
android:name="android.permission.ACCESS_FINE_LOCATION" />
<uses-permission
android:name="android.permission.FOREGROUND_SERVICE" />
```

(Η χρήση του **FOREGROUND_SERVICE** αφορά την ενημέρωση του χρήστη σχετικά με την ενεργοποίηση του GPS). Επίσης το AndroidManifest.xml περιέχει τα activities που χρησιμοποιούνται στην εφαρμογή:

```
<activity
android:name=".activities.IpPortActivity"></activity>
<activity
android:name=".activities.CoordinatesActivity"></activity>
<activity
android:name=".activities.CreateSensorActivity"></activity>
<activity
    android:name=".activities.MainActivity"
    android:exported="true">
    <intent-filter>
        <action android:name="android.intent.action.MAIN" />

        <category
android:name="android.intent.category.LAUNCHER" />
    </intent-filter>
</activity>
```

Γενικά ο κώδικας είναι δομημένος μέσα στο **package iot_ui** και διαχωρίζεται σε directories:

- **Activities**, περιέχει το MainActivity όπου στην κλάση **onCreate** αρχικοποιείται το UI της εφαρμογής (συμπληρωματικά για τις ανάγκες της εργασίας στο σημείο αυτό λαμβάνουμε και το ποσοστό μπαταρίας της IoT συσκευής καθώς και το μοναδικό της **ID**).

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    activityMainBinding =
ActivityMainBinding.inflate(getLayoutInflater());
    setContentView(activityMainBinding.getRoot());

    // Initialize fragments lists
    fragments = new ArrayList<>();
    cordFragments = new ArrayList<>();

    setNetworkCallBack();
    setupAdapter();

    registerReceiver(this.BatteryLevelReciever, new
IntentFilter(Intent.ACTION_BATTERY_CHANGED));

    Constants.DeviceId =
Secure.getString(this.getContentResolver(),Secure.ANDROID_ID);
    Log.d("Android","Android ID : "+Constants.DeviceId);
}
```

Με την setupAdapter ορίζεται η μορφή της αρχικής οθόνης και τα tabs των αισθητήρων.

```
private void setupAdapter() {
    fragments.add(new GasSensorFragment());
    fragments.add(new SmokeSensorFragment());
    fragments.add(new TempSensorFragment());
    fragments.add(new UVSensorFragment());

    adapter = new
CoordinatesOptionAdapter(getSupportFragmentManager(),
getLifecycle(), fragments);
    activityMainBinding.viewPager.setAdapter(adapter);

    new TabLayoutMediator(activityMainBinding.tabLayout,
activityMainBinding.viewPager, true, ((tab, position) -> {
        switch (position) {
            case 0:
                tab.setText("Gas Sensor");
                break;
            case 1:
                tab.setText("Smoke Sensor");
                break;
            case 2:
                tab.setText("Temp Sensor");
                break;
            case 3:
```

```
                tab.setText("UV Sensor");
                break;
        }
    })).attach();
}
```

Ενώ στη συνέχεια δημιουργείται το menu, τα fragments για τα νέα activities(showLocationFragment, showCreateSensorFragment) και τα pop up dialogs για την επιλογή του topic στο οποίο θα συνδεθεί η συσκευή και για την έξοδο από την εφαρμογή.

Καθώς και τα **CoordinatesActivity**, **CreateSensorActivity** και **IpPortActivity** στα οποία δημιουργούνται νέα activities για την ομαλή εναλλαγή μεταξύ των οθονών. Η λογική είναι αντίστοιχη και στα δύο activities και αφορά τη δημιουργία μιας λίστας από fragments κάθε φορά:

```
// Initialize fragments lists
cordFragments = new ArrayList<>();
```

- **Adapter**, περιέχονται οι κλάσεις διασύνδεσης (binding) των xml αρχείων και των fragments.
- **Fragments**, για κάθε tab που βλέπουμε υπάρχει ένα fragment που υλοποιεί τις εκάστοτε ενέργειες.

Πιο συγκεκριμένα στο **GasSensorFragment**(αντίστοιχα λειτουργούν και τα **SmokeSensorFragment**, **TempSensorFragment**, **UVSensorFragment**):

Στην **onCreate** αρχικοποιείται ο αισθητηρας αερίων και ενεργοποιείται η σύνδεση με τον **broker** (μέσω του **MQTT**) ώστε να επικοινωνήσει ο αισθητήρας με τον **EdgeServer**.

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    // Initialize GasSensor object
    gasSensor = new GasSensor(0.0, false); //
Initial value and sensor status
    customMqttService = new
CustomMqttService(requireContext());
    customMqttService.setupMqttClient(); // Create
connection to the mqtt broker
}
```

Στην **onViewCreated** μέσω του **binding** και της **onValueChange** παίρνουμε την τιμή από τον slider του αισθητήρα.

```
public void onViewCreated(@NonNull View view, @Nullable Bundle
savedInstanceState) {
    // Your onViewCreated code
    binding.gasSlider.addOnChangeListener(new
Slider.OnChangeListener() {
        @Override
        public void onValueChange(@NonNull Slider slider,
```

```
float value, boolean fromUser) {
            // Update gas value in GasSensor
            gasSensor.updateGasValue(value);

binding.mainSliderValGas.setText(String.format("%.2f",
value));
            sensor_val = value;
            // Print message
            Log.d("GasSensorFragment", "Slider value changed:
" + value);
        }
    });

    // Setup button click listeners
    setupClickListeners();
}
```

Με την **onDestroy** διακόπτεται η σύνδεση με το **MQTT**

```
public void onDestroy() {
    super.onDestroy();
    customMqttService.disconnect();
}
```

Η **setupClickListeners** συνδέει την τιμή του slider με τα κουμπιά για ενεργοποίηση(**Enable**) και απενεργοποίηση(**Disable**) του MQTT και τον διαμοιρασμό της τιμής του με τον EdgeServer.

```
private void setupClickListeners() {

    binding.Enable.setOnClickListener(v -> {
        disableStartButton();
        enableStopButton();
        gasSensor.setSensorStatus(true); // use of mqtt when
button pressed
        Log.d("Enable Button", "Enable button pressed, mqtt
status: " + gasSensor.isSensorOn() + " sensor value: " +
gasSensor.getGasValue());

        // Send every one second the data with the use of mqtt
        countDownTimer = new CountDownTimer(Long.MAX_VALUE,
1000) {
            @Override
            public void onTick(long millisUntilFinished) {

customMqttService.publishData(String.format("%.2f",
sensor_val), 1);     // Upload the data based on the sensor
            }

            @Override
            public void onFinish() {
                // Not in use
            }
        };

        // Start the timer
```

```
        countDownTimer.start();
    });

    binding.Disable.setOnClickListener(v -> {
        disableStopButton();
        enableStartButton();
        gasSensor.setSensorStatus(false);
        Log.d("Disable Button", "Disable button pressed, mqtt
status: " + gasSensor.isSensorOn());

        countDownTimer.cancel();     // Stop the timer / break
the loop
        customMqttService.stopPublishing();

    });
}
```

Αναφορικά με το **ManualCoordinatesFragment** η υλοποίηση είναι παρόμοια με παραπάνω. Η διαφορά έγκειται στο **setupClickListeners** όπου επιλέγουμε μεταξύ **4 προεπιλεγμένων θέσεων** αισθητήρων.

```
private void setupClickListeners() {

    binding.radioGroup.setOnCheckedChangeListener((group,
checkedId) -> {
        if (checkedId == R.id.radioButtonPOS_1) {
            coordinates = Constants.getPosition(1);
            Log.d("Coordinates", ": " + coordinates );
        } else if (checkedId == R.id.radioButtonPOS_2) {
            coordinates = Constants.getPosition(2);
            Log.d("Coordinates", ": " + coordinates );
        } else if (checkedId == R.id.radioButtonPOS_3) {
            coordinates = Constants.getPosition(3);
            Log.d("Coordinates", ": " + coordinates );
        } else if (checkedId == R.id.radioButtonPOS_4) {
            coordinates = Constants.getPosition(4);
            Log.d("Coordinates", ": " + coordinates );
        }
    });
```

Το **AutomaticCoordinatesFragment** χρησιμοποιεί τον αισθητήρα GPS της συσκευής οπότε και η υλοποίηση του είναι λίγο διαφορετική και θα αναλυθεί παρακάτω.

Με την χρήση της **onAttach** ζητείται από τον χρήστη η συγκατάθεση του για την πρόσβαση στην τοποθεσία του και ενημερώνεται με σχετικό μήνυμα στην οθόνη της εφαρμογής.

```
public void onAttach(@NonNull Context context) {
    super.onAttach(context);
    // Android needs permission from the user in order to use
location from gps
    locationActivityResultLauncher =
```

```
registerForActivityResult(
            new
ActivityResultContracts.RequestMultiplePermissions(), result -
> {
                boolean areAllLocationPermissionsGranted =
true;
                for (boolean permissions : result.values()) {
                    areAllLocationPermissionsGranted =
areAllLocationPermissionsGranted && permissions;
                }

                if (areAllLocationPermissionsGranted) {
                    startLocationService();
                } else {
                    showAlertDialog();
                }
            });
}
```

Τέλος, στο **IpPortFragment** δίνεται η δημιουργία του **URI** επικοινωνίας με τον **EdgeServer** με δύο τρόπους:

-Με τη χρήση του **localhost**, στα πλαίσια της εργασίας.

```
binding.recomended.setOnClickListener(new
View.OnClickListener() {
    @Override
    public void onClick(View view) {
        ip = "broker.emqx.io";
        port = "1883";
        String URI = "tcp://" + ip + ":" + port;
        Constants.HOST_IP = ip;
        Constants.PORT = port;
        CustomMqttService.SERVER_URI = URI;
        Toast.makeText(requireContext(), "URI Broker: " +
"tcp://broker.emqx.io:1883", Toast.LENGTH_SHORT).show();
    }
});
```

-Με τη χρήση των Ip και Port που πληκτρολογεί κάθε φορά ο χρήστης.

```
binding.submit.setOnClickListener(new View.OnClickListener() {
    @Override
    public void onClick(View view) {
        ip = binding.ip.getEditText().getText().toString();
        port =
binding.port.getEditText().getText().toString();

        String URI = "tcp://" + ip + ":" + port;
        Constants.HOST_IP = ip;
        Constants.PORT = port;
        CustomMqttService.SERVER_URI = URI;
        Toast.makeText(requireContext(), "New URI Broker: " +
URI, Toast.LENGTH_SHORT).show();
```

```
    }
});
```

- **Sensors**, κάθε αισθητήρας αρχικοποιείται και με χρήση διαφόρων μεθόδων ανανεώνουμε τις τιμές του(**updateGasValue**, **updateSmokeValue** etc.), ελέγχουμε αν είναι ενεργός(**setSensorStatus**) κ.α.
- **Services**, στα πλαίσια της εργασίας χρειάστηκαν δύο services(**locationService** και **mqttService**).

Το **locationService** αφορά τη δημιουργία ενός **listener** ο οποίος θα παρακολουθεί και θα αντιλαμβάνεται μέσω του GPS την αλλαγή της τοποθεσίας της συσκευής.

Το **mqttService** αναλαμβάνει τη δημιουργία ενός **client** μέσω της **setupMqttClient** ώστε να δημιουργείται η σύνδεση της συσκευής με το MQTT.

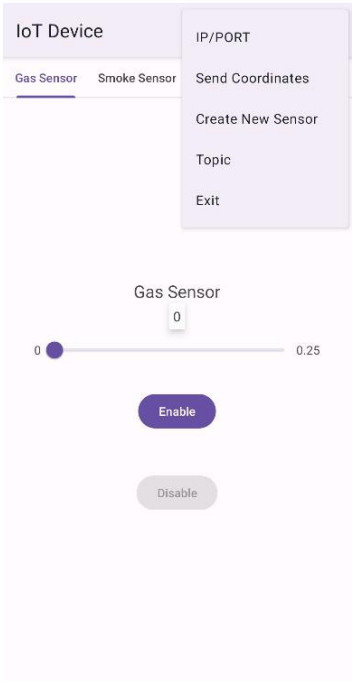
```
public void setupMqttClient() {
    mqttAndroidClient = new MqttAndroidClient(context,
SERVER_URI, "android_app_client");

    try {
        IMqttToken token = mqttAndroidClient.connect();

        token.setActionCallback(new IMqttActionListener() {
            @Override
            public void onSuccess(IMqttToken asyncActionToken)
{
                Log.d(TAG, "MQTT IS CONNECTED!");
                Toast.makeText(context, "MQTT Connection
established", Toast.LENGTH_SHORT).show();
            }

            @Override
            public void onFailure(IMqttToken asyncActionToken,
Throwable exception) {
                Log.d(TAG, "MQTT IS NOT CONNECTED!");
                Toast.makeText(context, "MQTT Connection is
NOT established!", Toast.LENGTH_SHORT).show();
            }
        });
    } catch (MqttException e) {
        e.printStackTrace();
    }
}
```

Με την **subscribe** συνδέεται στο topic της εκάστοτε συσκευής και είναι έτοιμο να κοινοποιήσει δεδομένα στον EdgeServer. Τέλος με τις **publishCoordinates** και **publishData** δημιουργεί τα απαραίτητα μηνύματα προς αποστολή.

- **Utils**, περιέχει το **NetworkMonitor** όπου μέσω του system service παίρνουμε το connectivity στο internet και την παρακολούθηση της κατάστασης σύνδεσης αλλά και την αποδέσμευσή της. Επίσης περιέχει την **Constants** ως βοηθητική κλάση που έχει σταθερές τιμές για τις προεπιλεγμένες θέσεις αισθητήρων καθώς και κάποια βοηθητικά strings για την λειτουργία του προγράμματος.

Ενώ μέσα στο directory **res/layout** είναι υλοποιημένα τα xml για το interface της εφαρμογής.

Σε επίπεδο interface:

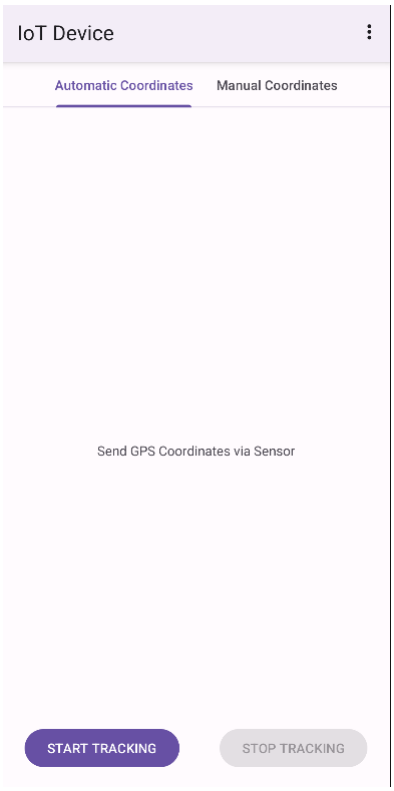


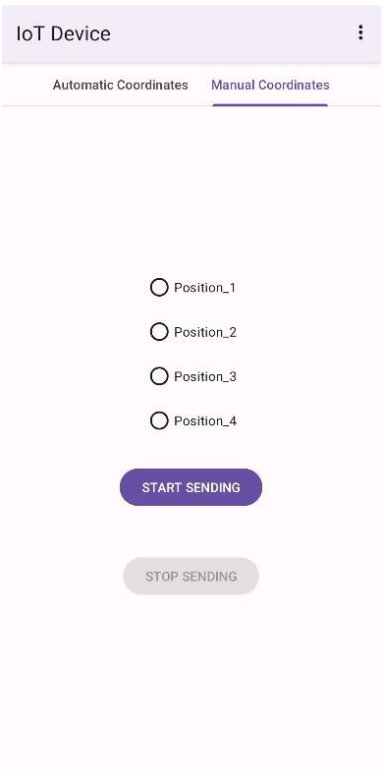
Η **αρχική οθόνη** της εφαρμογής αποτελείται από 5 tabs με τους προ εγκατεστημένους αισθητήρες και ένα menu με τις εξής επιλογές:

Κάθε αισθητήρας αποτελείται από έναν slider με το προκαθορισμένο εύρος τιμών της μέτρησης. Καθώς και δύο buttons για την ενεργοποίηση και απενεργοποίηση του διαμοιρασμού των δεδομένων του αισθητήρα.

- IP/PORT: Επιλογή για τον ορισμό Ip και Port επικοινωνίας με τον Edge Server. Κουμπί **Recommended** χρήση του προεπιλεγμένου localhost, ενώ το κουμπί **Submit URL** χρησιμοποιεί την Ip και Port που επιλέγει ο χρήστης.

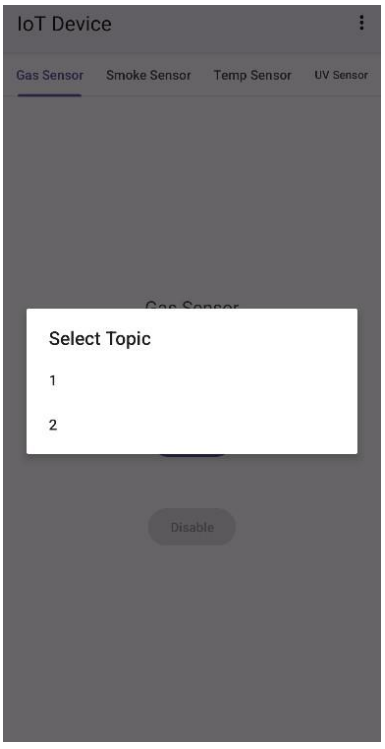
- Send Coordinates: Δυνατότητα επιλογής χειροκίνητου(4 προεπιλεγμένες θέσεις) ή αυτόματου τρόπου αποστολής της θέσης του αισθητήρα.



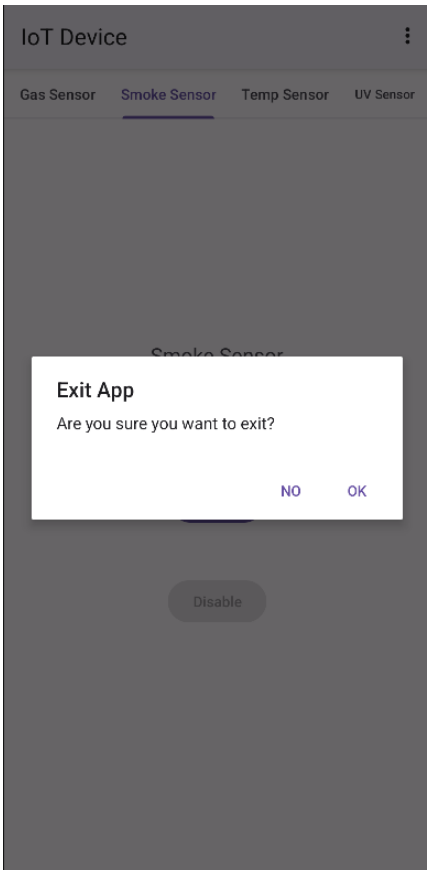


- Create New Sensor: Επιλογή δημιουργίας νέου τύπου αισθητήρα. Αποτελείται από ένα ListView με τους διαθέσιμους τύπους αισθητήρα που μπορούμε να δημιουργήσουμε. Επίσης δίνεται στο χρήστη να πληκτρολογήσει τα όρια για το εύρος τιμών του αισθητήρα. Η διαδικασία ολοκληρώνεται με το κουμπί Submit που σηματοδοτεί τη δημιουργία του νέου αισθητήρα.

- Topic: Εμφανίζεται ένα pop up μήνυμα με δυνατότητα επιλογής από το χρήστη μεταξύ δύο topic για να διαλέξει με ποια θύρα του EdgeServer θα επικοινωνήσει.



- Exit: Έξοδος από την εφαρμογή, με εμφάνιση προειδοποιητικού μηνύματος.


IoT Device
Gas Sensor
Smoke Sensor
Temp Sensor
UV Sensor
Exit App
Are you sure you want to exit?
NO
OK
Disable