ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ & ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

ΕΛΛΗΝΙΚΗ ΔΗΜΟΚΡΑΤΙΑ
Εθνικόν και Καποδιστριακόν
Πανεπιστήμιον Αθηνών
ΙΔΡΥΘΕΝ ΤΟ 1837

Μάθημα:

Ανάπτυξη Λογισμικού για Δίκτυα και Τηλεπικοινωνίες

Διδάσκοντες:

Νάνσυ Αλωνιστιώτη, Παντελής Μπαλαούρας

Τίτλος εργασίας:

«Ανάπτυξη Συστήματος Περιβαλλοντικής Διαχείρισης για την Ασφάλεια των Πολιτών σε Καταστάσεις Έκτακτης Ανάγκης»

Ονοματεπώνυμα Φοιτητών:

Βιδάλης Ιωάννης-Ζάννες (1115202000022)

Σγούρας Κωνσταντίνος (1115202000178)

Φεσλιάν Ελένη (1115202000204)

Χριστοφιλοπούλου Βασιλική (1115202000216)

Μάρτιος 2024

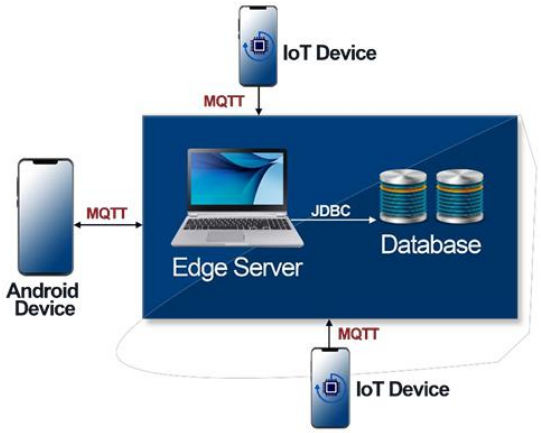
Table of Contents

Chapter 1: Βασικά Στοιχεία

1.1 Περιγραφή εργασίας

Η παρούσα εργασία στοχεύει στην ανάπτυξη ενός προηγμένου συστήματος Περιβαλλοντικής Διαχείρισης, το οποίο επικεντρώνεται στην ασφάλεια των πολιτών κατά τη διάρκεια καταστάσεων έκτακτης ανάγκης. Το σύστημα που αναπτύχθηκε παρέχει ένα ολοκληρωμένο πλαίσιο για τη συλλογή, ανάλυση και διαχείριση δεδομένων που σχετίζονται με τις περιβαλλοντικές συνθήκες και την ασφάλεια.

Για την υλοποίηση της εργασίας δημιουργήθηκε μια εφαρμογή Android και ένας διακομιστή που επικοινωνούν μέσω του πρωτοκόλλου MQTT.



Ειδικότερα, το σύστημα λειτουργεί με τη χρήση Android συσκευών, οι οποίες, συνδεδεμένες μέσω πρωτοκόλλου MQTT, ανταλλάσσουν πληροφορίες με τον Edge Server. Ο τελευταίος είναι υπεύθυνος για τη συγκέντρωση, ανάλυση και ανταλλαγή δεδομένων μεταξύ των Android συσκευών και των στατικών IoT συσκευών, οι οποίες τοποθετούνται στο περιβάλλον και συλλέγουν μετρήσεις μέσω ενσωματωμένων αισθητήρων.

Επιπλέον, ο Edge Server αναλαμβάνει την ανάλυση των δεδομένων που προέρχονται από τις IoT συσκευές για τον εντοπισμό πιθανών κινδύνων. Σε περίπτωση εντοπισμού, προβαίνει σε ενημέρωση των Android συσκευών και καταχωρεί τις εντοπισμένες απειλές σε μια MySQL βάση δεδομένων.

1.2 Εκτέλεση εφαρμογής

Η εφαρμογή εκτελέστηκε στο πρόγραμμα ανάπτυξης εφαρμογών intellij και αρχικά τρέχουμε το αρχείο app για να ξεκινήσει ο server και έπειτα τις δύο εφαρμογές android.

Chapter 2: Εφαρμογή Χρήστη

Η εφαρμογή χρήστη αναφέρεται στην κινητή συσκευή που θέτει σε λειτουργία ο χρήστης (ή οι χρήστες ανάλογα το σενάριο που υλοποιούμε) με σκοπό τον εντοπισμό κάποιου κινδύνου. Επιπρόσθετα, οι συσκευές αυτές είναι υπεύθυνες για την άμεση αποστολή της θέσης τους με χειροκίνητο ή αυτόματο τρόπο.

2.1 MainActivity.java

Σε αυτό το αρχείο κώδικα, ξεκινά η υλοποίηση με την δημιουργία του μενού πλοήγησης της εφαρμογής, στο οποίο ο χρήστης πληκτρολογεί την IP του, το port number του, το συχνότητα μετάδοσης των δεδομένων του (μόνο για τον χειροκίνητο τρόπο). Επιπλέον του δίνεται η δυνατότητα να αλλάξει τον τρόπο με τον οποίο αποστέλλει την τοποθεσία του στον Edge Server, από χειροκίνητο σε αυτόματο και αντίστροφα.

Όπως φαίνεται παρακάτω, δημιουργήσαμε μια μέθοδο η οποία ελέγχει περιοδικά (ανά 10 δευτερόλεπτα) αν υπάρχει σύνδεση στο διαδίκτυο.

```
/* ===== Periodic checks on whether there is a Internet connection =====*/
Handler handler = new Handler();
Runnable checkConnectivity = new Runnable() {
   @Override
   public void run() {
       //Get the system's connectivity manager
ConnectivityManager              cm              =              (ConnectivityManager)
getSystemService(Context.CONNECTIVITY_SERVICE);

//Check if the device is connected to the internet using NetworkCapabilities
NetworkInfo activeNetwork = cm.getActiveNetworkInfo();
boolean isConnected = activeNetwork != null && activeNetwork.isConnectedOrConnecting();

if (isConnected) {
//Device is connected to the internet
   showToast("Device is connected to the internet");
} else {
//Device is not connected to the internet -> Give solution to user via Settings
   AlertDialog.Builder builder = new AlertDialog.Builder(MainActivity.this);
   builder.setTitle("No Internet Connection");
   builder.setMessage("Please activate your internet connection to use this app.");
 builder.setPositiveButton("Settings", new DialogInterface.OnClickListener() {
  @Override
   public void onClick(DialogInterface dialog, int which) {
```

```
  // Open the device's settings where the user can activate their internet connection
          Intent intent = new Intent(Settings.ACTION_WIRELESS_SETTINGS);
           startActivity(intent);}});
     builder.setNegativeButton("Cancel", null);
     builder.show();}

// Schedule the Runnable to run again in 30 seconds
   handler.postDelayed(this, 30000);}};

//Schedule the Runnable to run for the first time immediately
handler.post(checkConnectivity);
```

2.1.1 Χειροκίνητος τρόπος

Για την μετατροπή των αρχείων xml σε csv αρχικά αποθηκεύουμε τα δεδομένα σε πίνακες όπως φαίνεται παρακάτω

```
for(int i=0; i<nodeList.getLength(); i++){
   Element timestepElement = (Element) nodeList.item(i);
String                                    x                                    =
timestepElement.getElementsByTagName("vehicle").item(0).getAttributes().getNamedItem(
"x").getNodeValue();
   xlat[i] = x;
   String                                    y                                    =
timestepElement.getElementsByTagName("vehicle").item(0).getAttributes().getNamedItem(
"y").getNodeValue();
   ylong[i] = y;

//After parsing, call the method to convert to CSV
csvData = convertToCsv(ylong, xlat);  //1st ylong and 2nd xlat -> cause the given data
have wrong layout
}
```

Και αργότερα τα μετατρέπουμε στη ζητούμενη μορφή csv.

```
private String convertToCsv(String[] xArray, String[] yArray) {
   StringBuilder csvContent = new StringBuilder();

   //Add data rows
   for(int j = 0; j < xArray.length; j++) {
       csvContent.append(xArray[j]).append(",").append(yArray[j]).append("\n");
   }
```

```
    //Return the CSV representation
    return csvContent.toString();
}
        //Add data rows
        for(int j = 0; j < xArray.length; j++) {
            csvContent.append(xArray[j]).append(",").append(yArray[j]).append(",").append(tim
eArray[j]).append("\n");
        }

        // Return the CSV representation
        return csvContent.toString();
    }
```

2.1.2 Αυτόματος τρόπος

Προκειμένου να ανακτήσουμε την τοποθεσία με αυτόματο τρόπο, αξιοποιούμε τη μέθοδο μέσω της παρακάτω μεθόδου η οποία επιστρέφει το στίγμα του χρήστη.

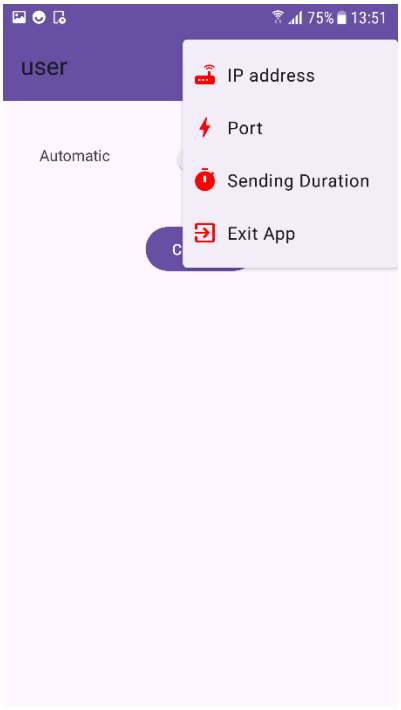
```
private void fetchLocation() {
    if (ActivityCompat.checkSelfPermission(MainActivity.this, ACCESS_FINE_LOCATION) ==
PackageManager.PERMISSION_GRANTED ) {
        Log.d("GPS","GPS has permisiom");
        location_client.getLastLocation().addOnSuccessListener(MainActivity.this,   new
OnSuccessListener<Location>() {
            @Override
            public void onSuccess(Location location) {
                Log.d("GPS","on success");
                if (location != null) {
                    setGlobalLong(location.getLongitude());
                    setGlobalLat(location.getLatitude());
                    Log.d("GPS","found");
                }
            }
        });
    }
}
```

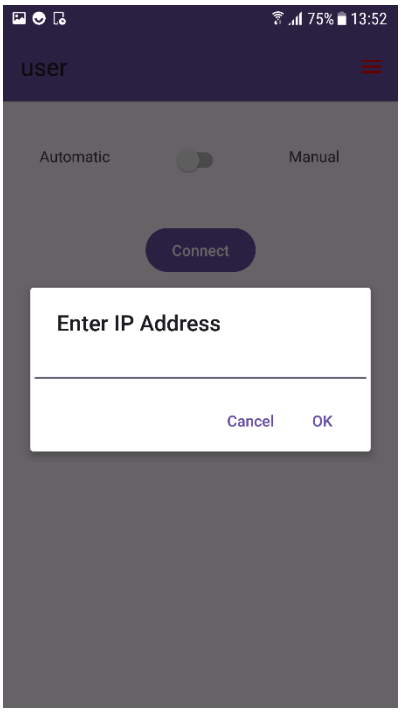
Για την υλοποίηση του τμήματος της εκφώνησης που προβάλλει την οπτικοακουστική ειδοποίηση, δημιουργήθηκε το παρακάτω κομμάτι κώδικα. Η **messageArrived** δηλώνει ότι μόλις δεχτεί η συσκευή Android ένα μήνυμα κινδύνου από τον Edge Server, ανάλογα με την επικινδυνότητα των αισθητήρων παρουσιάζεται το κατάλληλο μήνυμα στον χρήστη και ηχητική ειδοποίηση.

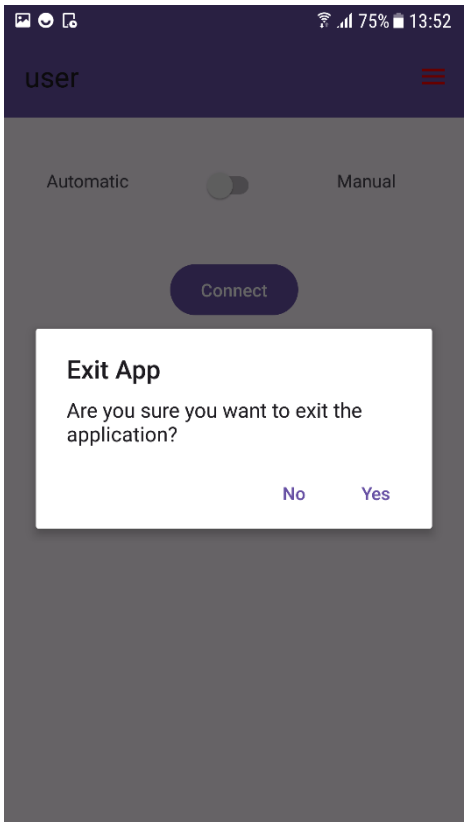
```
public void messageArrived(String topic, MqttMessage message) throws Exception {
   //Convert MQTT message to string
   String receivedMessage = new String(message.getPayload());

   //Check if the message starts with "0,"
   if (!receivedMessage.startsWith("0,")) {
       //Update TextView with the received message
       runOnUiThread(new Runnable() {
           @Override
           public void run() {
               if(receivedMessage.startsWith("H") ||
receivedMessage.startsWith("M")){
                   textView.setText(receivedMessage);
                   dangerCounter=5;
               }
               else {
                   //Reset TextView to not display any message
                   textView.setText("");
               }

               //Only at Moderate/High Danger play audio
               if (receivedMessage.startsWith("H") ||
receivedMessage.startsWith("M")) {
                   //Play sound based on the message
                   playSound(receivedMessage);
               }}});
...
private void playSound(String message) {
   int soundResourceId = R.raw.mid;
   if (message.startsWith("M")) {
       soundResourceId = R.raw.mid;
   }
   else if (message.startsWith("H")) {
       soundResourceId = R.raw.high;
   }
   // Create MediaPlayer instance and play the selected sound
   MediaPlayer mediaPlayer = MediaPlayer.create(this, soundResourceId);
   mediaPlayer.setOnCompletionListener(new MediaPlayer.OnCompletionListener() {
       @Override
       public void onCompletion(MediaPlayer mp) {
           //Release MediaPlayer resources after completion
           mp.release();
```

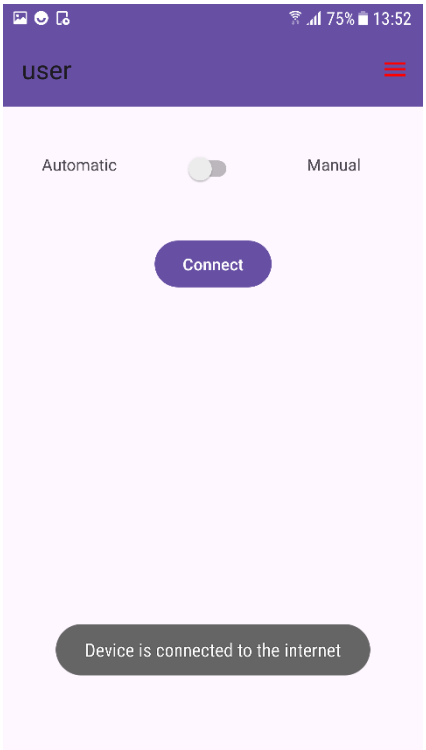
```
        }});
    mediaPlayer.start();}
```

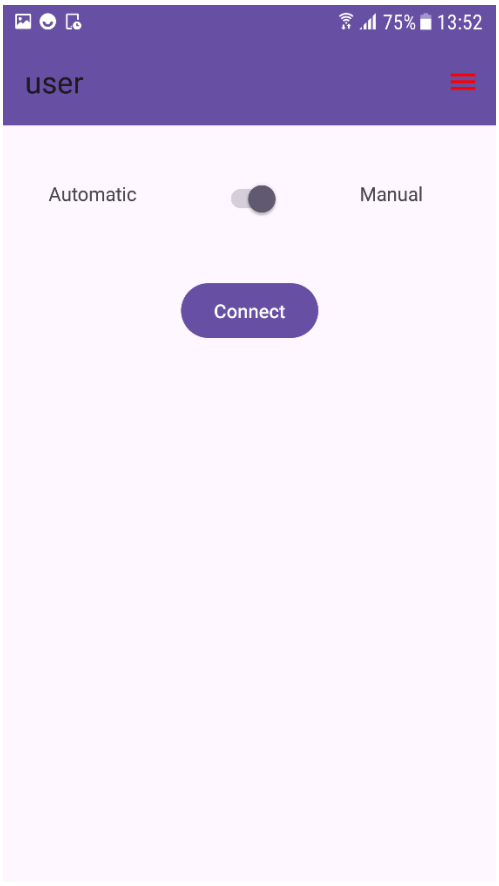
2.2 Στιγμιότυπα

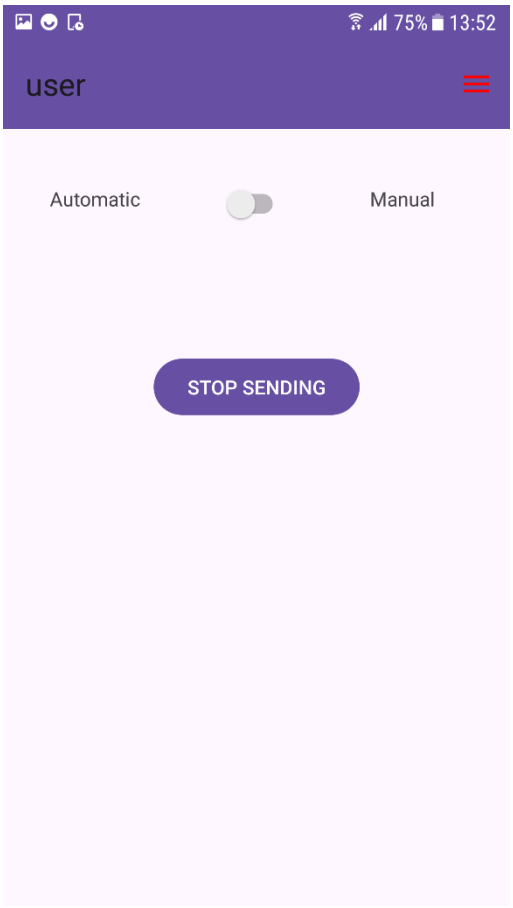


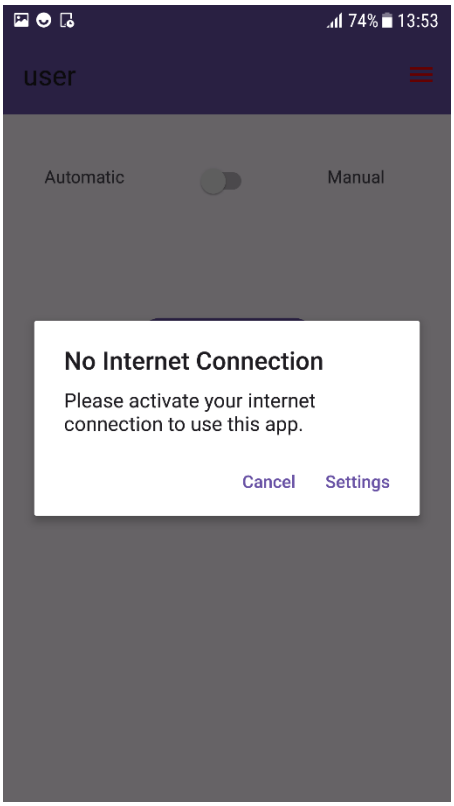


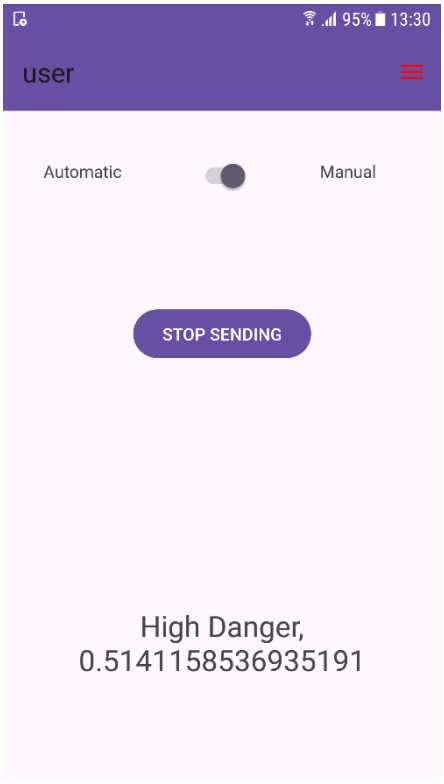





75% 13:52
user
Automatic
Manual
Connect


75% 13:52
user
Automatic
Manual
STOP SENDING


74% 13:53
user
Automatic
Manual
No Internet Connection
Please activate your internet connection to use this app.
Cancel
Settings


95% 13:30
user
Automatic
Manual
STOP SENDING
High Danger,
0.5141158536935191

Chapter 3: Εφαρμογή IoT Αισθητήρων

Για την υλοποίηση αυτού του τμήματος της εργασίας, απαιτούνται μέχρι δύο συσκευές οι οποίες παραμένουν στην ίδια τοποθεσία, με σκοπό την αποστολή δεδομένων που συλλέγονται από ενσωματωμένους αισθητήρες. Επιπρόσθετα, θα πρέπει περιοδικά να αποστέλλουν την τρέχουσα τοποθεσία τους με χειροκίνητο ή αυτόματο τρόπο και το ποσοστό μπαταρίας. Η επικοινωνία γίνεται μόνο με τον Edge Server μέσω πρωτοκόλλου MQTT.

3.1 Λεπτομερής Περιγραφή

Η εφαρμογή χρησιμοποιεί τρεις οθόνες:

- Αρχική Οθόνη
- Οθόνη Ρυθμίσεων
- Οθόνη Προσθήκη Αισθητήρα

3.1.1 Αρχική Οθόνη - MainActivity.java

Στην αρχική οθόνη ο χρήστης μπορεί να κάνει όλες τις βασικές λειτουργίες της εφαρμογής. Αρχικά παρατηρούμε στο πάνω μέρος ένα Tab Layout το οποίο χρησιμοποιείται για την περιήγηση στους αισθητήρες. Κάθε φορά που αλλάζουμε αισθητήρα τα παιδία των αισθητήρων προσαρμόζονται στις τιμές του κάθε αισθητήρα. Τα πεδία περιλαμβάνουν:

- Τον τύπο του αισθητήρα (δεν μπορεί να αλλάξει από τον χρήστη)
- Ένα Slider που δίνει τη δυνατότητα στο χρήστη να αλλάξει την τιμή του αισθητήρα. Οι τιμές μπορούν να επιλεγούν από ενά εύρος [Ελάχιστη τιμή, Μέγιστη τιμή] το οποίο είναι συγκεκριμένο για κάθε αισθητήρα
- Ένα Check Box που δίνει τη δυνατότητα στον χρήστη να ενεργοποιήσει/απενεργοποιήσει τον αισθητήρα.

Κάτω από τα πεδία έχουμε τοποθετήσει ένα κουμπί που ξεκινάει την μετάδοση των τιμών των αισθητήρων στον Edge Server. Όταν πατηθεί γίνεται σύνδεση στον Server (σύμφωνα με τις ρυθμίσεις που έχουμε επιλέξει). Όταν συνδεθεί, κάθε δευτερόλεπτο αποστέλλονται οι τιμές που περιλαμβάνουν:

- Ποιά συσκευή από τις δύο χρησιμοποιούμε.
- Την τοποθεσία της συσκευής (αυτές οι τιμές θα καθορίζονται είτε αυτόματα μέσω του αισθητήρα GPS της συσκευής είτε χειροκίνητα μέσω κάποιων τοποθεσιών που έχουν δοθεί από την εκφώνηση)
- Την τιμή του κάθε αισθητήρα (αν ο αισθητήρας είναι απενεργοποιημένος τότε στέλνουμε την τιμή -100).

- Την μπαταρία της συσκευής.

Τέλος στην πάνω δεξιά γωνία παρατηρούμε το κουμπί που μας μεταφέρει στις ρυθμίσεις.

3.1.2 Οθόνη Ρυθμίσεων - SettingsActivity.java

Στην οθόνη ρυθμίσεων μπορούμε να επιλέξουμε ρυθμίσεις οι οποίες θα χρησιμοποιηθούν για την σύνδεση με τον Edge Server και την αποστολή των δεδομένων. Ο χρήστης μπορεί να επιλέξει:

- Την διεύθυνση IP του Edge Server.
- Την θύρα που χρησιμοποιείται για την επικοινωνία με τον Edge Server.
- Τον τρόπο επιλογής της τοποθεσίας.
- Τον αριθμό της IoT συσκευής.

Τέλος, στην κάτω δεξιά γωνία υπάρχει το κουμπί που μας μεταφέρει στην οθόνη προσθήκης αισθητήρα.

3.1.3 Οθόνη Προσθήκη Αισθητήρα - NewSensor.java

Σε αυτή την οθόνη ο χρήστης μπορεί να δημιουργήσει έναν καινούργιο αισθητήρα. Μπορεί να επιλέξει:

- Τον τύπο του αισθητήρα.
- Την ελάχιστη και την μέγιστη τιμή που μπορεί να πάρει.

Αξίζει να σημειωθεί ότι μπορούν να προστεθούν όσους αισθητήρες θέλουμε. Για τις ανάγκες της εργασίας, στον Edge Server αποστέλλονται οι τιμές μόνο τεσσάρων αισθητήρων από διαφορετικούς τύπους (επιλέγεται αυτός που προστέθηκε τελευταίος).

Τέλος, στο κάτω μέρος της οθόνης υπάρχουν δύο κουμπιά για τη προσθήκη του αισθητήρα ή την ακύρωση της προσθήκης του. Τα κουμπιά μας επιστρέφουν στην οθόνη των ρυθμίσεων.

3.2 Sensor.java

Σε αυτό το αρχείο έχουμε υλοποιήσει τη δομή του αισθητήρα. Έχουν οριστεί όλες οι απαραίτητες μεταβλητές και όλες οι απαραίτητες συναρτήσεις για την αλλαγή τους.

3.3 AndroidManifest.xml

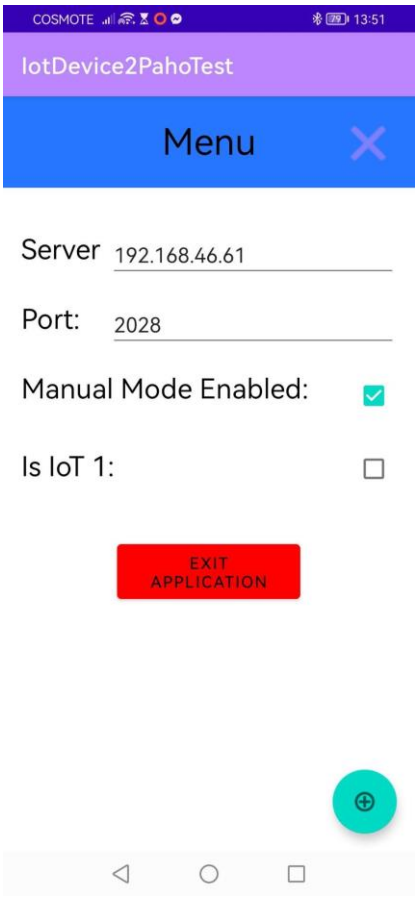
Στο συγκεκριμένο αρχείο υπάρχουν όλες οι άδειες που ζητούνται από τον χρήστη καθώς και η υπηρεσία MQTT.

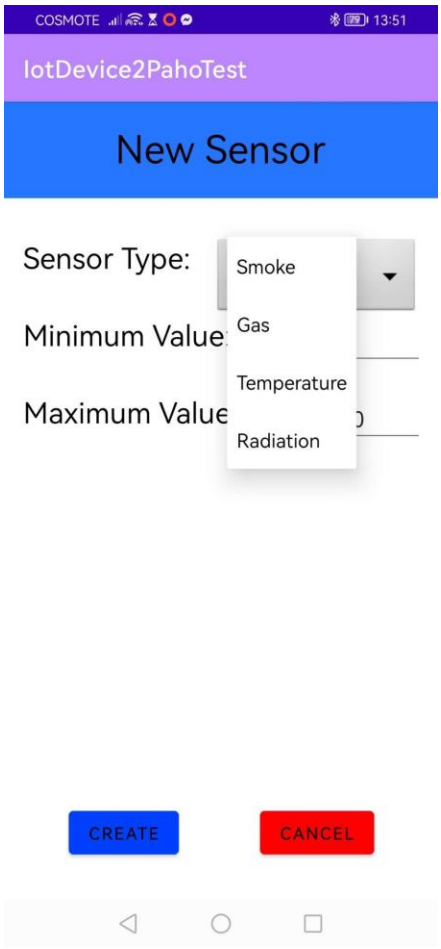
3.4 build.gradle

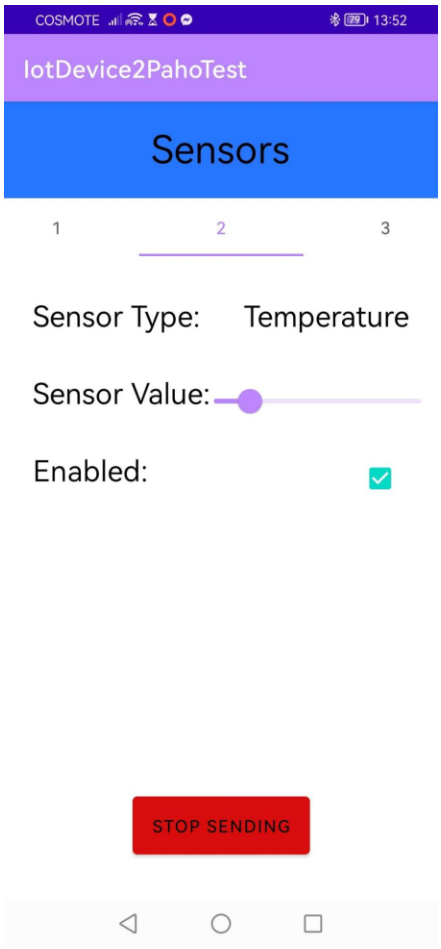
Σε αυτό το αρχείο έχουν τοποθετηθεί όλα τα αρχεία που είναι αναγκαία για την υλοποίηση της εργασίας.

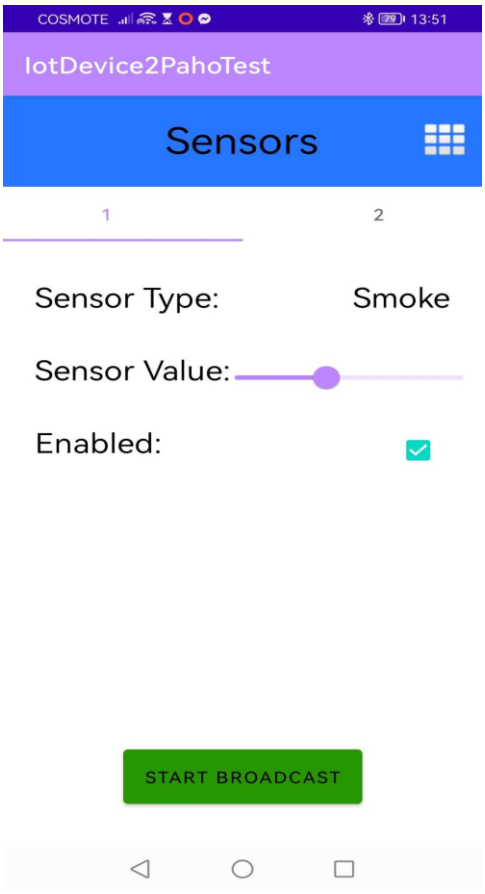
3.5 Στιγμιότυπα

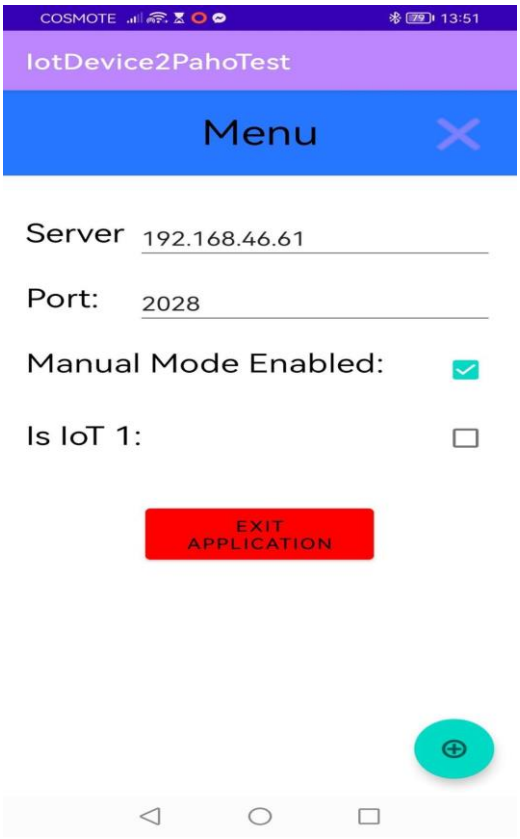
Παρακάτω παραθέτουμε μερικά στιγμιότυπα από την εφαρμογή που δημιουργήσαμε για τις IoT συσκευές και παρουσιάζουμε τη λειτουργικότητά της.








COSMOTE
13:51
IotDevice2PahoTest
Sensors
1
2
Sensor Type:
Smoke
Sensor Value:
Enabled:
START BROADCAST


COSMOTE
13:51
IotDevice2PahoTest
Menu
Server
192.168.46.61
Port:
2028
Manual Mode Enabled:
Is IoT 1:
EXIT APPLICATION

Chapter 4: Edge Server

Η κεντρική λειτουργία του Edge Server στο πλαίσιο αυτής της εφαρμογής επικεντρώνεται στην επεξεργασία δεδομένων αισθητήρων και την ενημέρωση των συσκευών Android σε περίπτωση κινδύνου. Συνεπώς, επειδή πρόκειται για μια πολύπλοκη διαδικασία, ομαδοποιήσαμε και χωρίσαμε τον κώδικα μας έτσι ώστε να είναι ευανάγνωστος και εύκολος στην κατανόηση.

4.1 App

Εδώ, ορίζουμε τα URL που είναι απαραίτητα τόσο για την βάση δεδομένων όσο και για τον online broker, που χρησιμοποιούμε για την επικοινωνία μεταξύ server, IoT συσκευών και Android. Δημιουργούμε ένα νέο server με αυτά τα χαρακτηριστικά και έπειτα τον αρχικοποιούμε.

```
Server server = new Server(mqttBrokerUrl, databaseUrl, databaseUsername, databasePassword);
server.initializeMqttConnections();
```

4.2 Server

4.2.1 Κατασκευαστής – Constructor

Η κλάση αυτή δημιουργεί τα απαραίτητα αντικείμενα για την για την επικοινωνία με τη βάση δεδομένων και την υπηρεσία MQTT.

```
final                          MqttHandler                          mqttController;
final                            Database                             dbController;
Maps maps;
```

Ο κατασκευαστής δέχεται στις παραμέτρους του το URL του broker MQTT, το URL της βάσης δεδομένων, καθώς επίσης το όνομα χρήστη και τον κωδικό πρόσβασης.

Δημιουργεί ένα αντικείμενο της κλάσης Database και ενεργοποιεί την αρχικοποίηση της βάσης δεδομένων. Επιπλέον, δημιουργεί ένα αντικείμενο της κλάσης MqttHandler με τα δοθέντα στοιχεία και ενεργοποιεί τη σύνδεση MQTT και στην συνέχεια αρχικοποιεί τους χάρτες.

```
public Server(String mqttBrokerUrl, String databaseUrl, String username, String password)
throws IOException, SAXException, ParserConfigurationException, InterruptedException {
   this.dbController = new Database(databaseUrl, username, password);
   System.out.println(databaseUrl);

   dbController.Initialization();
   System.out.println("Initialization");
   System.out.println(mqttBrokerUrl);
   this.mqttController = new MqttHandler(mqttBrokerUrl, dbController);
   System.out.println("mqttController");

   this.maps = new Maps();}
```

4.2.2 Μέθοδος initializeMqttConnections():

Καλεί τη μέθοδο Connect() της κλάσης MqttHandler για τη σύνδεση στον broker MQTT.

```
public void initializeMqttConnections()
{
        mqttController.Connect();
}
```

4.3 Database

Η κλάση Database αναπαριστά τον ελεγκτή για τη βάση δεδομένων στην εφαρμογή. Ακολουθεί μια λεπτομερής περιγραφή του κώδικα:

4.3.1 Δηλώσεις μεταβλητών

Αρχικά, δηλώνονται όλες οι μεταβλητές που μας είναι απαραίτητες, καθώς και ο πίνακας που χρειάζεται για την αποθήκευση των δεδομένων, ο οποίος έχει την ακόλουθη μορφή:

```
private final String CreateTable =
        "CREATE TABLE `Project` "
        + " (`timestep` VARCHAR(20) NOT NULL DEFAULT '00-00-0000 00:00:00',"
        + "  `device_id` double NOT NULL DEFAULT 0,"
        + "  `latitude` double DEFAULT NULL,"
        + "  `longitude` double DEFAULT NULL,"
        + "  `Smoke` double DEFAULT NULL,"
        + "  `Gas` double DEFAULT NULL,"
        + "  `Temperature` double DEFAULT NULL,"
        + "  `UV` double DEFAULT NULL,"
        + "  `Danger` integer DEFAULT NULL,"   //0 - not in danger, 1- in danger
        + "  PRIMARY KEY (device_id, timestep)"
        + ")";
```

4.3.2 Κατασκευαστής – Constructor

Ο κατασκευαστής δημιουργεί μια σύνδεση με τη βάση δεδομένων και εκτυπώνει μηνύματα επιτυχούς ή ανεπιτυχούς σύνδεσης.

```
public Database(String URL, String username, String password) {
```

Επιπλέον, υλοποιείται ενα shutdown hook, έτσι ώστε να εξασφαλιστεί ότι πριν την οριστική λήξη της εκτέλεσης του προγράμματος θα αποδεσμευτεί η σύνδεση με την βάση δεδομένων.

```
Runtime.getRuntime().addShutdownHook(new Thread(new Runnable() {
            public void run() {
                try {
                    connection.close();
                } catch (SQLException e) {
```

```
                e.printStackTrace();
            }
            System.out.println("In shutdown hook");
        }
    }, "Shutdown-thread"));
```

4.3.3 Συνάρτηση Αρχικοποίησης - Initialization()

Στην μέθοδο αυτή, ελέγχετε πρώτα ένα υπάρχει σύνδεση με την βάση και εφόσον υπάρχει ελέγχει αν ο πίνακας "Project" υπάρχει ή όχι.

Αν δεν υπάρχει, δημιουργεί τον πίνακα "Project".

Αν υπάρχει, διαγράφει τον πίνακα "Project" και έπειτα τον ανακτά για επαναδημιουργία.

4.3.4 Εισαγωγή - InsertDB

Δημιουργήθηκε και η συνάρτηση που εισάγει στη βάση τα δεδομένα που της παρέχονται ως ορίσματα. Χρησιμοποιείται η μέθοδος χρησιμοποιεί PreparedStatement για αποφυγή επιθέσεων SQL Injection.

```
public void InsertDB(String timestep, double device_id, double latitude, double longitude,
double Smoke, double Gas, double Temperature, double UV, double Danger) {
```

4.4 Mqtt

Η επικοινωνία του Edge Server με τις συσκευές IoT και την Android επιτυγχάνεται μέσω του πρωτοκόλλου MQTT. Τα δεδομένα λαμβάνονται από το αντίστοιχο topic στο οποίο έχει γίνει κάποιο publish από τις IoT. Αρχικά, στον κώδικα που υλοποιήσαμε δηλώνουμε κατάλληλα τις μεταβλητές που θα χρειαστούν για τους αισθητήρες, συνεχίζοντας με τον ορισμό των μεταβλητών για την σύνδεση σε MQTT broker και για το publish μηνυμάτων σε αντίστοιχο topic. Επιτρέπεται έτσι η επικοινωνία και ανταλλαγή μηνυμάτων σε ένα δίκτυο MQTT, η οποία είναι ζωτικής σημασίας για τη δημοσίευση και την εγγραφή σε θέματα. Τέλος, ορίζουμε στους αισθητήρες την κατώτατη τιμή που μπορούν να πάρουν, χωρίς να δημιουργηθεί κίνδυνος.

```
public class MqttHandler implements MqttCallback {
   public boolean android_flag=false, iot1_flag=false, iot2_flag=false;
   private double device_id;
   private double Danger;
   double android_lon, android_lat, iot1_lon, iot1_lat, iot2_lon, iot2_lat;
   double android_id, iot1_id, iot2_id;
   Distance.Point android_marker, iot1_marker, iot2_marker;
   private MqttClient client;
   private final String serverUrl;
   Database dbController;
```

```
    SimpleDateFormat dateFormat = new SimpleDateFormat("dd-MM-yyyy HH:mm:ss");
    private static Map<String, Double> dangerThresholds = new HashMap<>();
    static {
        //Thresholds for every IOT
        dangerThresholds.put("smoke", 0.14);
        dangerThresholds.put("gas",  9.15);
        dangerThresholds.put("temperature", 50.0);
        dangerThresholds.put("uv", 6.0);
    }}
```

Προχωρώντας στον κώδικα, δηλώνουμε μεθόδους-exceptions, ώστε να βεβαιωθούμε για την σωστή σύνδεση στο broker. Με τη χρήση της μεθόδου **calculateDangerLevel** ελέγχουμε αν με βάση τις προϋποθέσεις της εκφώνησης υπάρχει ή όχι κίνδυνος, και αν ναι τι επιπέδου είναι.

```
private String calculateDangerLevel(double smoke, double gas, double temperature,
double uv) {
    if (smoke > dangerThresholds.get("smoke") && gas > dangerThresholds.get("gas")) {
        this.Danger = 1;
        return "High Danger";
    } else if (temperature > dangerThresholds.get("temperature") && uv >
dangerThresholds.get("uv")) {
        this.Danger = 1;
        return "Moderate Danger";
    } else if (gas > dangerThresholds.get("gas")) {
        this.Danger = 1;
        return "High Danger";
    } else if (smoke > dangerThresholds.get("smoke") && gas >
dangerThresholds.get("gas")
            && temperature > dangerThresholds.get("temperature") && uv >
dangerThresholds.get("uv")) {
        this.Danger = 1;
        return "High Danger";
    } else {
        return "Low Danger";
    }}
```

Η μέθοδος **messageArrived** είναι αυτή η οποία χειρίζεται τα μηνύματα που γίνοται publish από τις υπόλοιπες συσκευές και αποτελεί το “κλειδί” της επικοινωνίας. Σαν σχεδιαστική επιλογή, για να μπορούμε να διακρίνουμε ευκολότερα τι είδους συσκευή έχει στείλει το κάθε μήνυμα

χρησιμοποιούμε το πεδίο typeVal, που βρίσκεται στην αρχή κάθε μηνύματος και αν είναι 0 πρόκειται για Android συσκευή, αν 1 υπάρχει μία ενεργή συσκευή IoT και αν υπάρχει και δεύτερη το πεδίο είναι ίσο με 2. Σε κάθε περίπτωση, αποκωδικοποιεί το μήνυμα διακρίνοντάς το σε μικρότερα πεδία-μεταβλητές για επεξεργασία. Ενδεικτικά παραθέτουμε κάποια κομμάτια του κώδικα που υλοποιούν τη λειτουργία που περιγράφηκε.

```
if (typeVal == 0){
    //if android device
    android_flag=true;
    String and_lat = fields[1];
    String and_lon = fields[2];
    String device_id = fields[3];
    android_lat = Double.parseDouble(and_lat);
    android_lon = Double.parseDouble(and_lon);
    android_id = Double.parseDouble(device_id);
    android_marker = new Distance.Point(android_lat, android_lon);

    //Add marker for android
    Maps.addAndroidpoint(android_id, android_lat, android_lon);
}
else{
    // iot devices
    String Smoke = fields[4];
    String Gas = fields[5];
    String Temperature = fields[6];
    String UV = fields[7];
    String Battery = fields[8];
    double SmokeVal = Double.parseDouble(Smoke);
    double GasVal = Double.parseDouble(Gas);
    double TemperatureVal = Double.parseDouble(Temperature);
    double UVVal = Double.parseDouble(UV);
    double Batteryval = Double.parseDouble(Battery);
    …
```

Όλα τα δεδομένα αποθηκεύονται στη βάση δεδομένων και σε περίπτωση κινδύνου με την ύπαρξη 2 IoT συσκευών, όπως απαιτείται από την εκφώνηση υπολογίζουμε το κέντρο της απόστασης τους με βάση των κώδικα που δώθηκε. Στο αρχείο έχουν υλοποιηθεί και άλλες, δευτερεύουσας χρήσης συναρτήσεις, όπως connect(), disconnect(), connectionLost(Throwable cause) κοκ.

4.5 Distance

Το πακέτο αυτό χρησιμοποιείται προς διευκόλυνση μας για τον υπολογισμό των αποστάσεων που απαιτούνται για την απεικόνιση των συσκευών, με βάση τον σύνδεσμο που δόθηκε.[1]

Δημιουργείται μια δομή point η οποία αποθηκεύει μόνο το latitude και longitude για κάθε σημείο.

```
public Point(double lat,double lon )
{
    this.lat=lat;
    this.lon=lon;
}
```

Επιπλέον, υλοποιείται η μέθοδος public static double calculate(Point android,Point Iot1,Point Iot2), η οποία ακολουθεί την ίδια λογική με αυτή που δίνεται στον σύνδεσμο. Αρχικά, ελέγχει εάν υπάρχει και δεύτερο IoT και εφόσον δεν υπάρχει υπολογίζει την απόσταση μεταξύ του ενός IoT και της Android συσκευής, διαφορετικά υπολογίζει το κεντρικό σημείο μεταξύ των IoT και στη συνέχεια υπολογίζει την απόσταση από το Android στο κεντρικό σημείο.

Η συνάρτηση private static double distance(double lat1, double lon1, double lat2, double lon2, String unit), υπολογίζει την απόσταση μεταξύ δύο γεωγραφικών σημείων χρησιμοποιώντας τον τύπο του Haversine. Τα γεωγραφικά σημεία δίνονται με γεωγραφικό πλάτος (lat1, lat2) και γεωγραφικό μήκος (lon1, lon2). Η απόσταση επιστρέφεται σε ναυτικά μίλια (unit.equals("N")) ή χιλιόμετρα (unit.equals("K")).

4.6 Maps

Για την υλοποίηση του χάρτη, δεν χρησιμοποιήσαμε Google Maps Api, αλλά την βιβλιοθήκη JxMapViewer. Χρήσιμο κώδικα για την υλοποίηση αυτή βρήκαμε στο github.[2]

Στο συγκεκριμένο αρχείο ορίζουμε τους περιορισμούς για τις IoT συσκευές, και επακολούθως αρχίζουμε την υλοποίηση της γραφικής διεπαφής για την επισκόπηση των συσκευών που είναι συνδεδεμένες στο δίκτυο.

Εισαγωγικά, έχουμε τη συνάρτηση **calculateDangerLevel** η οποία ελέγχει σύμφωνα με τους περιορισμούς της εκφώνησης ποιοί αισθητήρες βρίσκονται σε κατάσταση κινδύνου και επιστρέφει έναν ακέραιο, ώστε να είναι εύκολη η διάκριση των περιπτώσεων.

```
private static int calculateDangerLevel(double smoke, double gas, double temperature,
double uv) {
   if (smoke > dangerThresholds.get("smoke") && gas > dangerThresholds.get("gas")) {
       return 3;
```

[1] Calculate Di stance by Latitude and Longitude using Java | GeoDataSource

[2] GitHub - msteiger/jxmapviewer2: JXMapViewer2

```
    } else if (temperature > dangerThresholds.get("temperature") && uv > dangerThresholds.get("uv")) {
        return 2;
    } else if (gas > dangerThresholds.get("gas")) {
        return 3;
    } else if (smoke > dangerThresholds.get("smoke") && gas > dangerThresholds.get("gas")
            && temperature > dangerThresholds.get("temperature") && uv > dangerThresholds.get("uv")) {
        return 3;
    } else {
        return 1;
    }
}
```

Η επόμενη σημαντική μέθοδος του αρχείου είναι η **initializeMapViewer,** η οποία είναι υπεύθυνη για την εκτέλεση της γραφικής διεπαφής του χάρτη. Ο κώδικας ρυθμίζει μια τοπική κρυφή μνήμη για την αποθήκευση των πλακιδίων του χάρτη για να καθιστά την χρήση εκτός σύνδεσης εφικτή. Ορίζεται η αρχική θέση του προγράμματος προβολής του χάρτη ως το πανεπιστημιακό campus.

Με την συνάρτηση **addAndroidpoint** τοποθετούμε στο χάρτη που δημιουργήθηκε παραπάνω τον κατάλληλο δείκτη με ροζ χρώμα που δίνει το στίγμα για τη συσκευή Android.

Για να είναι ορατό το επίπεδο κινδύνου κάθε IoT συσκευής ελέγχουμε τον ακέραιο που επέστρεψε η συνάρτηση που περιγράψαμε παραπάνω, και χρωματίζουμε καταλλήλως το στίγμα της IoT συσκευής (συνάρτηση **addIoTpoint**). Προχωρώντας στον κώδικα, ελέγχουμε αν και οι δύο συσκευές βρίσκονται σε κίνδυνο και εκτυπώνουμε τα κατάλληλα μηνύματα. Σε κάθε περίπτωση, ο χάρτης αλλάζει χρωματολογικά ανάλογα με την κατάσταση στην οποία βρισκόμαστε.

```
if (riskLevel == 1) {
    existingWaypoint.setColor(Color.blue);
    existingWaypoint.setCircleRadius(15);
    existingWaypoint.setColorCircle(circleCol);
    deleteRectangle();
} else if (riskLevel == 2) {
    existingWaypoint.setColor(Color.yellow);
    existingWaypoint.setCircleRadius(15);
    existingWaypoint.setColorCircle(circleCol);
}
else {
    existingWaypoint.setColor(Color.red);
```

```
    existingWaypoint.setCircleRadius(15);
    existingWaypoint.setColorCircle(circleCol);
}
...
if (riskLevel == 1)
        {
            MyWaypoint   newWaypoint   =   new   MyWaypoint(String.valueOf(device_id),
Color.blue, gp,15, circleCol);
            newWaypoint.setdanger(riskLevel);
            iotWaypoints.add(newWaypoint);
            updateWaypointPainter();
            mapViewer.repaint(); // Repaint the map to reflect the changes
            newWaypoint.setTooltipText(tooltip);
...
```

Για να είναι ορατή η συνδεσιμότητα κάθε συσκευής IoT, χρησιμοποιήθηκε ένας κατάλληλα χρωματισμένος κύκλος ο οποίος αν είναι πράσινος δηλώνει ότι η συσκευή είναι ενεργή, και αν είναι κόκκινη το αντίθετο.

```
private  static  Color  determineCircleColor(  double  smoke,  double  gas,  double
temperature, double UV) {
    // Determine the color based on device activity
    if (smoke == -100 && gas == -100 && temperature == -100 && UV == -100) {
        // Device is inactive (use red color)
        return Color.red;
    } else {
        // Device is inactive (use red color)
        return Color.green;
    }
}
```

Με τη μέθοδο **createRectangle**, σε περίπτωση που οι δύο συσκευές IoT έχουν σημάνει κίνδυνο, δημιουργείται ένα ορθογώνιο μεταξύ των συσκευών ώστε να δείξουν την περιοχή κινδύνου. Η συνάρτηση **deleteRectangle** διαγράφει αυτό το ορθογώνιο μόλις διασφαλιστεί η ασφάλεια.

```
public static void  createRectangle(double lat,double lon,double lat2, double lon2)
{
    track.clear();
    GeoPosition position2 = new GeoPosition(lat2, lon2); //b
    GeoPosition position4 = new GeoPosition(lat, lon);//d
    GeoPosition position1 = new GeoPosition(lat2, lon); //a
    GeoPosition position3 = new GeoPosition(lat, lon2); //c
```

```
    track.addAll(Arrays.asList(position1,position2,position3,position4,position1));
    RoutePainter routePainter = new RoutePainter(track);
    List<org.jxmapviewer.painter.Painter<JXMapViewer>>       painters       =       new
ArrayList<Painter<JXMapViewer>>();
    painters.add(routePainter);
    painters.add(waypointPainter);
    CompoundPainter<JXMapViewer> painter = new CompoundPainter<JXMapViewer>(painters);
    mapViewer.setOverlayPainter(painter);
}
```

4.6.1 MyWaypoint

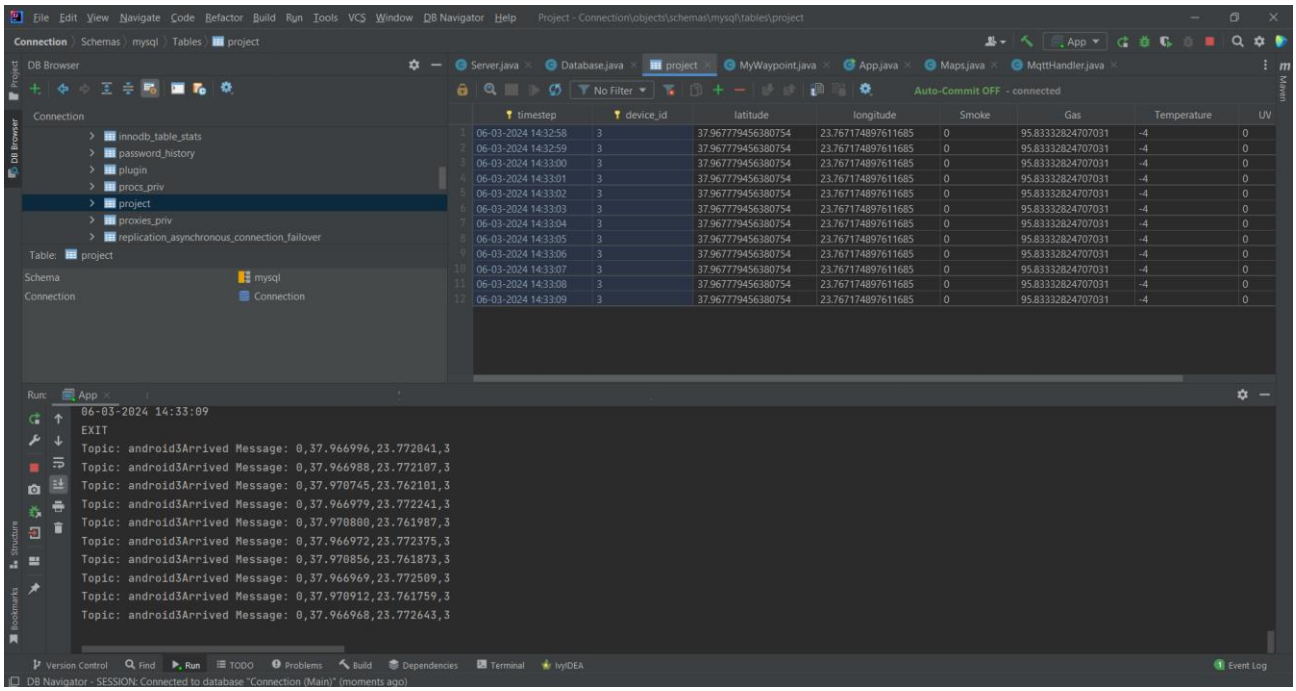
Στο αρχείο **MyWaypoint** ορίζουμε το στίγμα της κάθε συσκευής, χρησιμοποιώντας τις μεταβλητές label, color για το χρώμα της, circleradius για το activity της κάθε IoT, circlecol και danger.

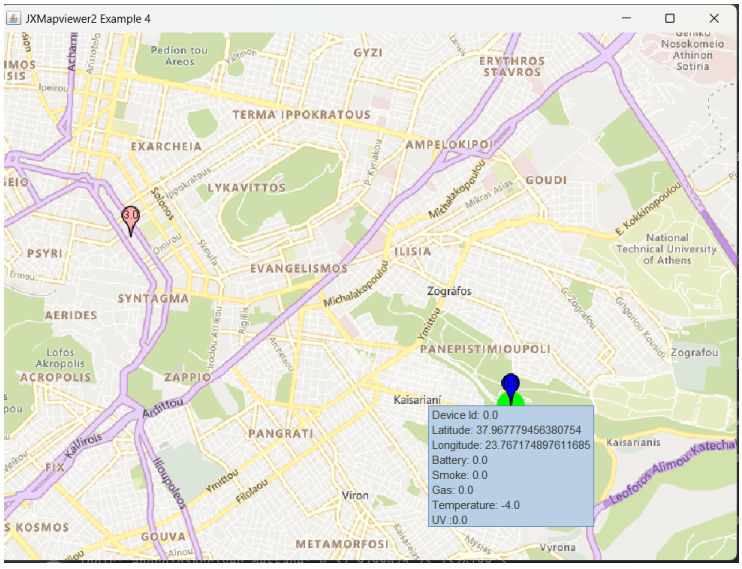
```
public MyWaypoint(String label, Color color, GeoPosition coord,double circleRadius,
Color circleCol)
{
    super(coord);
    this.label = label;
    this.color = color;
    this.circleRadius = circleRadius;
    this.circleCol = circleCol;
    this.danger = 0;
}
```

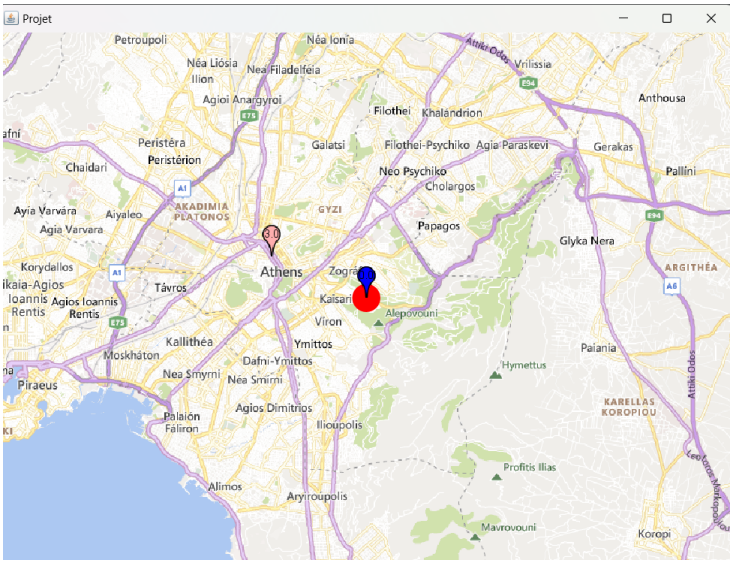
Στις συναρτήσεις που ακολουθούν, καθίσταται εφικτή η εξωτερική επικοινωνία της κλάσης με τα εξωτερικά αντικείμενα, διευκολύνοντας τη δυναμική προσαρμογή και ανάκτηση των χαρακτηριστικών.

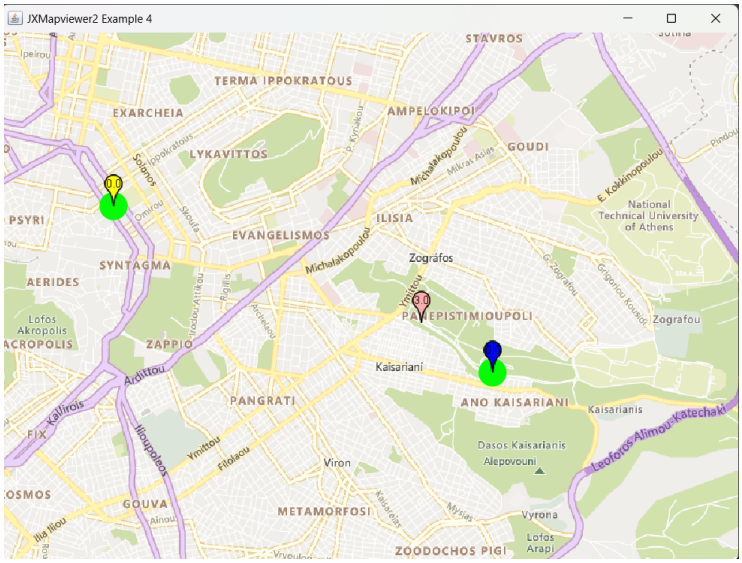
```
public Color getColorCircle() {
    return circleCol;
}
public void setColorCircle(Color color) {
    this.circleCol = color;
}
public double getCircleRadius() {
    return circleRadius;
} ...
```

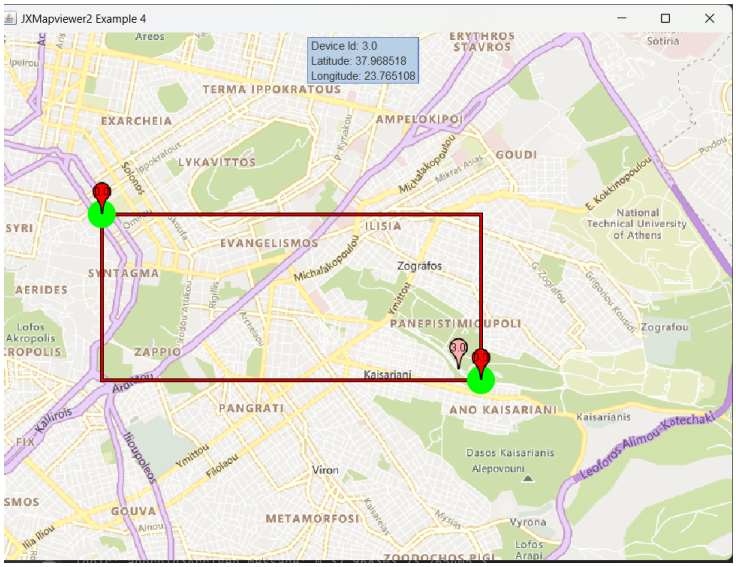
4.6.2 Στιγμιότυπα








JXMapviewer2 Example 4
STAVROS
TERMA IPPOKRATOUS
EXARCHEIA
AMPELOKIPOI
LYKAVITTOS
GOUDI
PSYRI
EVANGELISMOS
ILISIA
SYNTAGMA
AERIDES
Zográfos
PANEPISTIMIOUPOLI
ZAPPIO
Kaisariani
PANGRATI
ANO KAISARIANI
Kaisarianis
Dasos Kaisarianis
Alepovouni
Viron
METAMORFOSI
GOUVA
Vyrona
Lofos Arapi
ZOODOCHOS PIGI
National Technical University of Athens
Zografou
Leoforos Alimou-Katechaki


JXMapviewer2 Example 4
Device Id: 3.0
Latitude: 37.968518
Longitude: 23.765108
TERMA IPPOKRATOUS
EXARCHEIA
AMPELOKIPOI
LYKAVITTOS
GOUDI
EVANGELISMOS
ILISIA
SYNTAGMA
Zográfos
PANEPISTIMIOUPOLI
ZAPPIO
Kaisariani
PANGRATI
ANO KAISARIANI
Dasos Kaisarianis
Alepovouni
Viron
METAMORFOSI
GOUVA
Vyrona
National Technical University of Athens
Zografou
Leoforos Alimou-Katechaki