Air Quality Monitor

Vlad Catalin-Andrei AC CTI 4th year

Contents

1. Project Specifications

1.1. Project Description

Air quality monitor, as the name says, is a device that measures the most important parameters from air and sends them to your Android application. Any user with the application installed can connect to the devices spread around the city and check the air quality of that area.

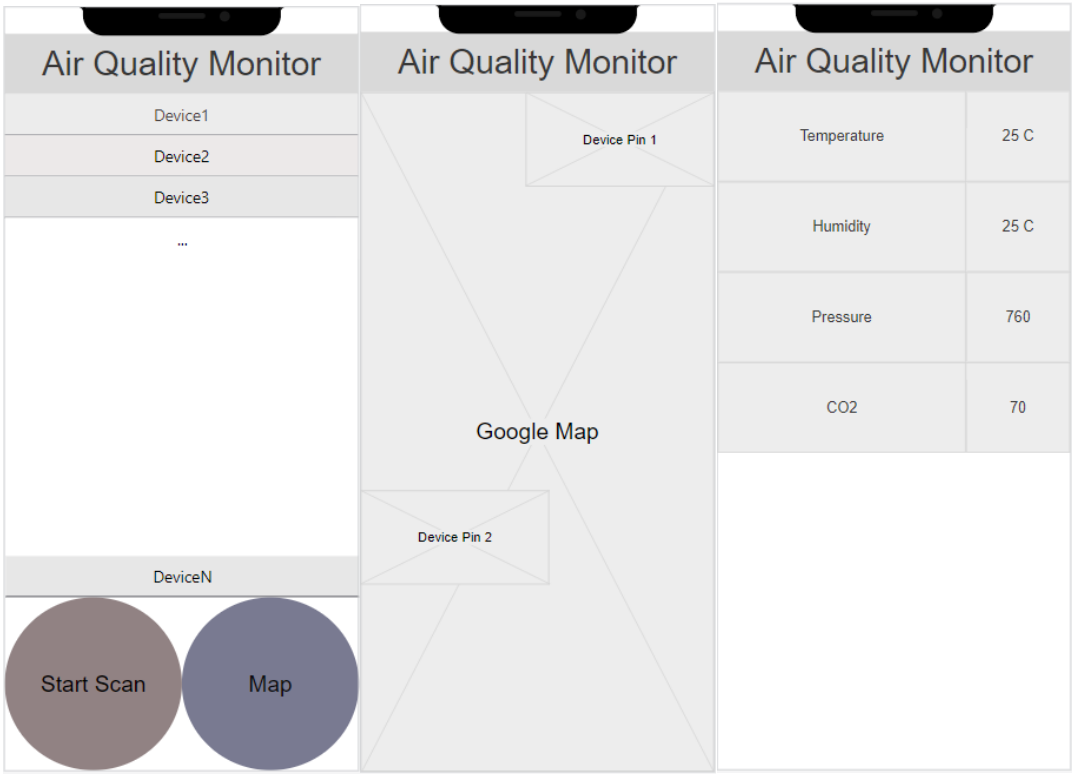
Once the user connects to the device, it sends the collected data to the smartphone and from there to a cloud server where it updates the specific device's data. Once an hour, the user's application is updated with fresh data about all the devices around the town and he can check it by opening Google Maps activity where all the devices location with the latest updates can be seen.

The purpose of this application is to raise awareness of the pollution people are exposed to every day. It is easy to use and you can check the air parameters any time.

1.2. User Stories

	User activities		
	Connect to a device	Visualize distribution of devices in the city	Visualize parameters from a specific area
User stories	As a user, I can connect to a device and retreive data about air quality in that area.	As a user, I can visualize the distribution of devices around the city.	As a user, I can see a detailed view of the parameters of a specific area
		As a user, I can receive once an hour, updated data for every device in the city. The updated data comes from the last user that connected to that device.	

1.3. Wireframes

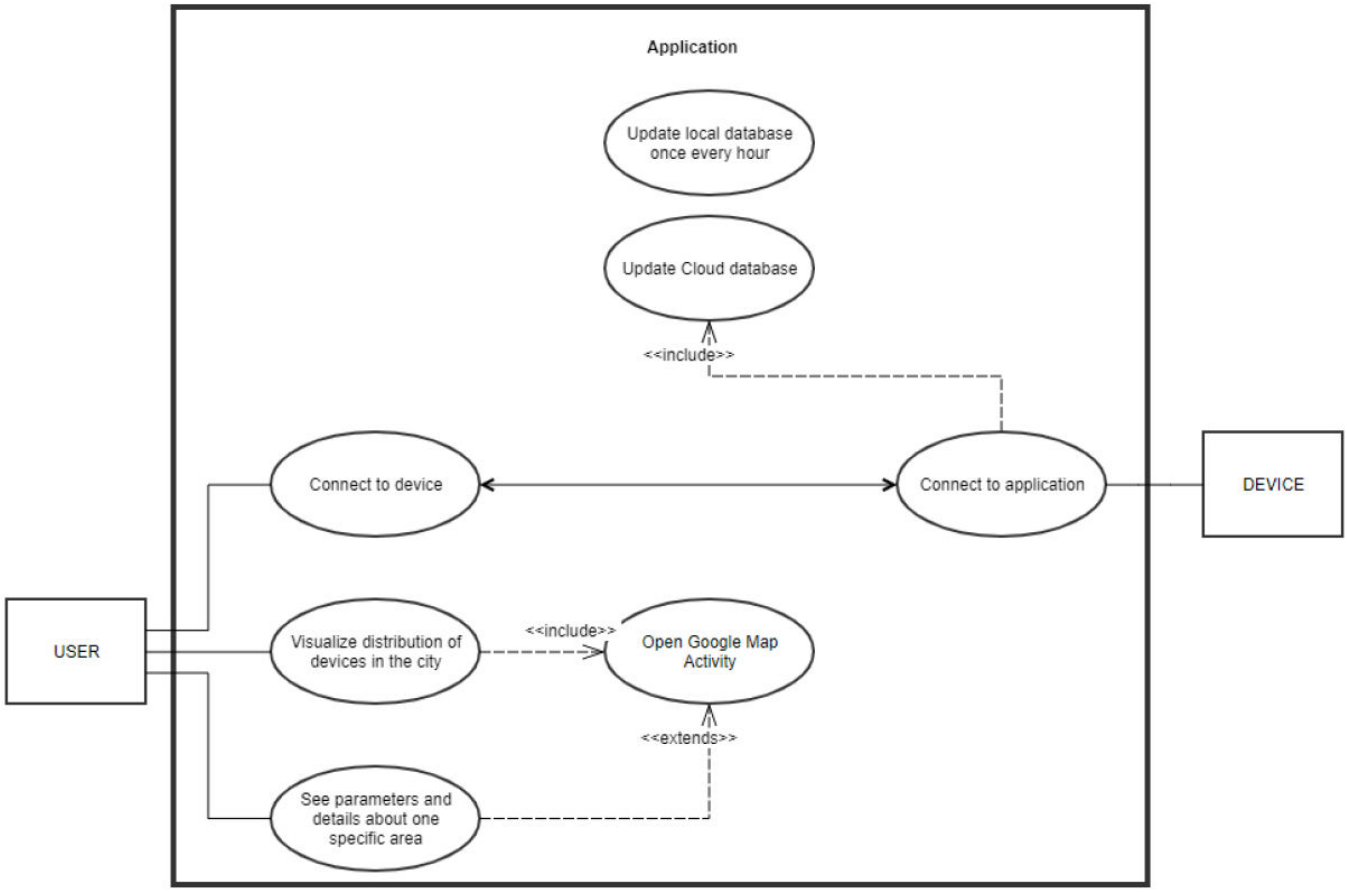


1.4. Tech stack

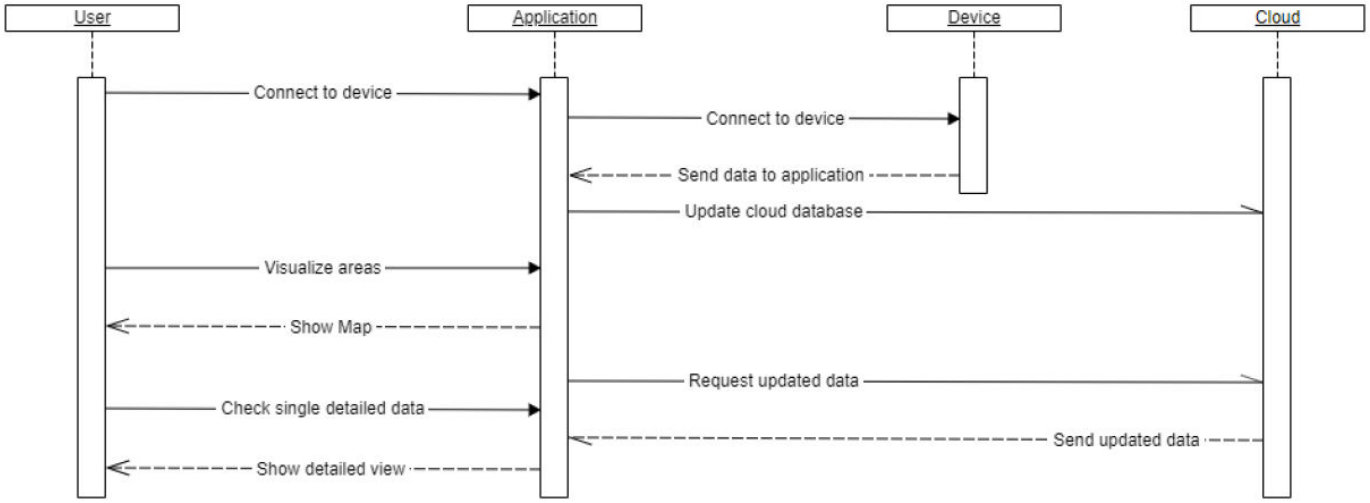
Cloud Infrastructure Services	Firebase
Front-End Technologies	Android Studio
Programming Language & Framework	Java, Android Studio
Database	Realm

2. Project Requirements

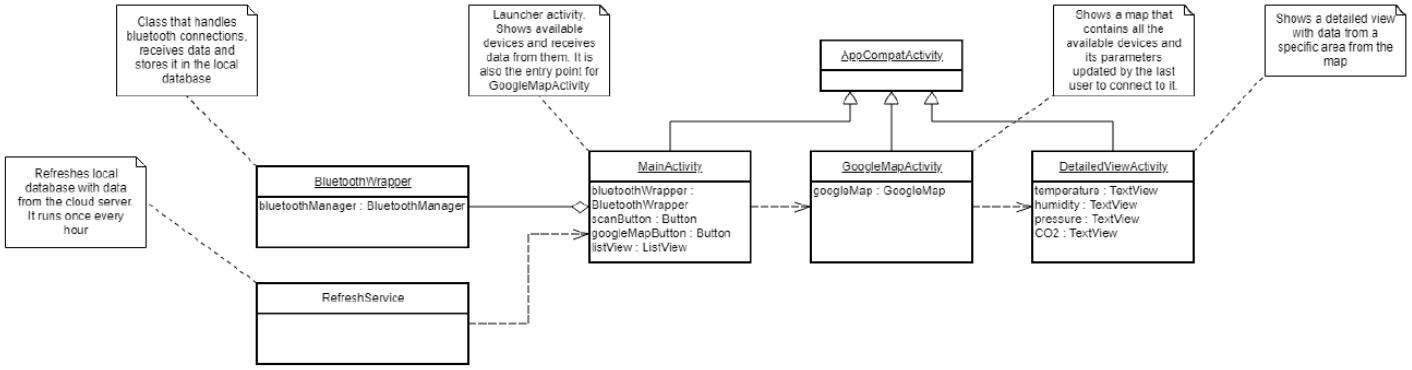
2.1. Use cases



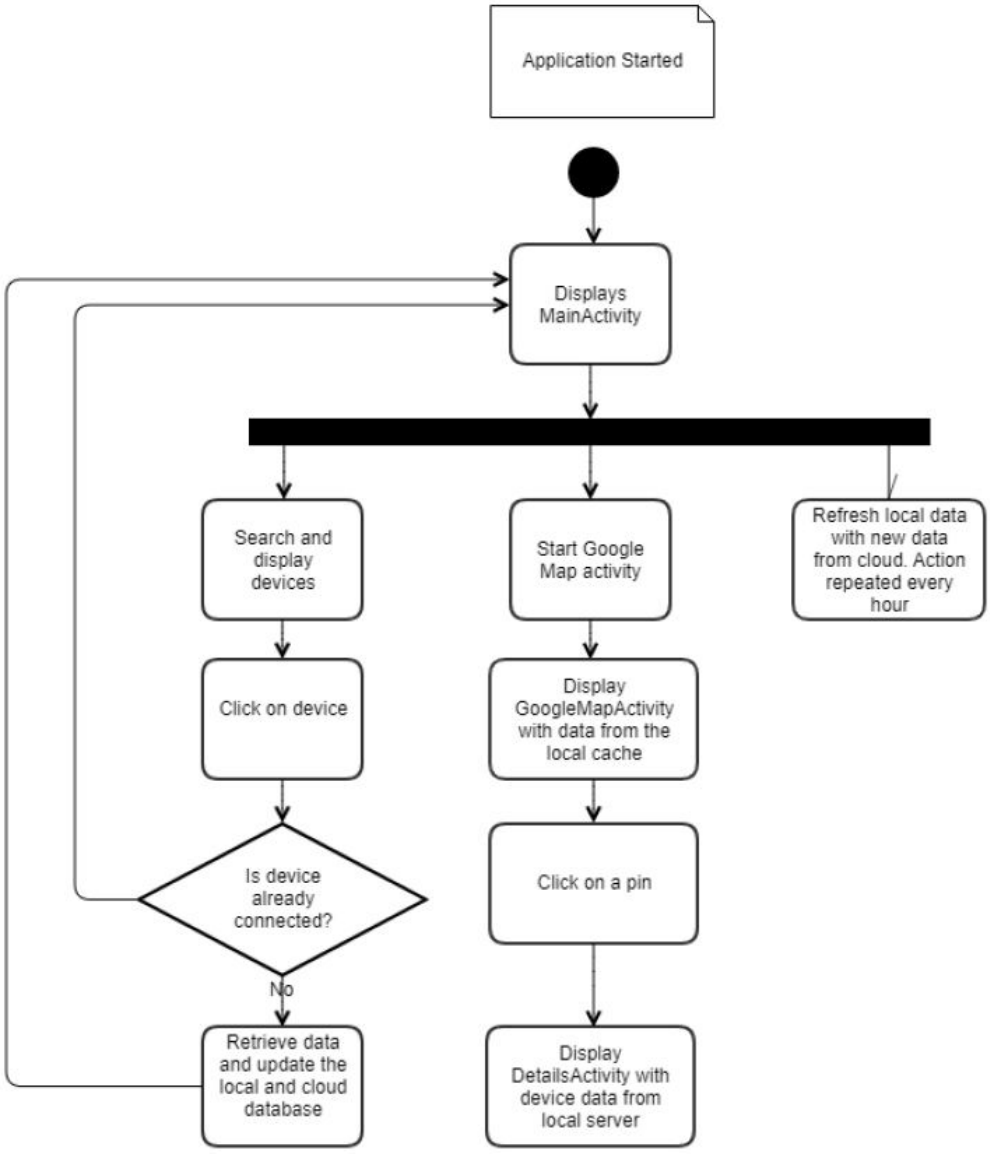
2.2. Sequence diagram



2.3. Class diagram



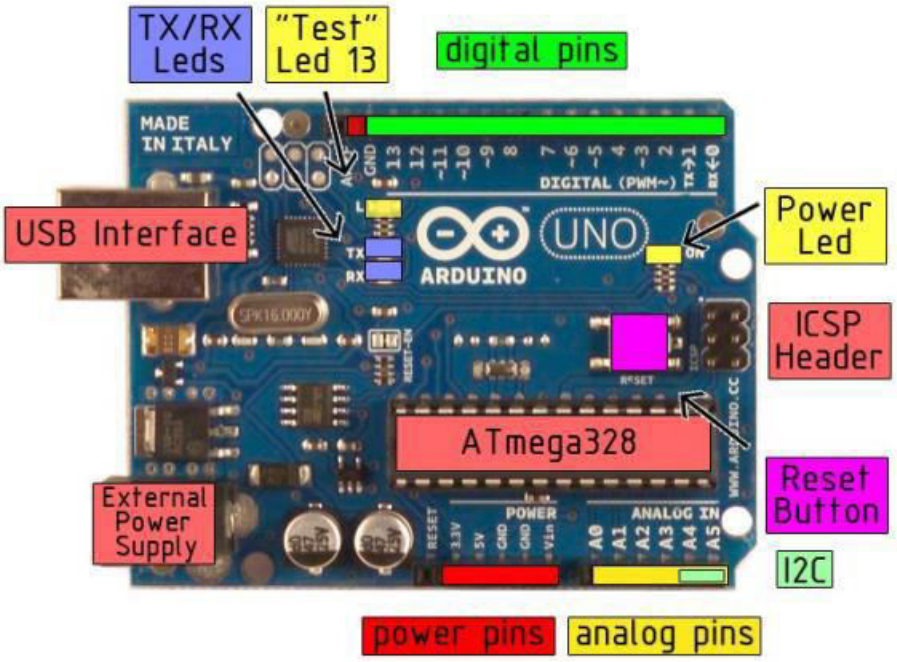
2.4. Activity diagram



3. Hardware Description

3.1. Microcontroller

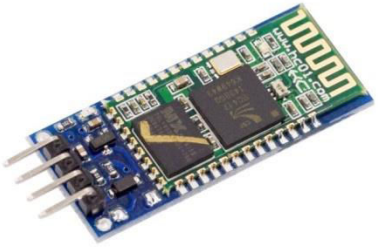
For this project I used an Arduino Uno R3 microcontroller. It is a microcontroller board based on the ATmega328. It has 14 digital input/output pins(of which 6 can be used as PWM outputs), 6 analog input/output pins, a 16MHzclock frequency, a USB connection, a power jack, an ICSP header and a reset button. It has 32kB of Flash memory that gives enough room for most sketches.



The processor of Arduino board uses Harvard architecture where the program code and program data have separate memory. It consists of two memories such as program memory and data memory. Data is stored in data memory and the code is stored in the flash program memory.

3.2. HC-05 Bluetooth module

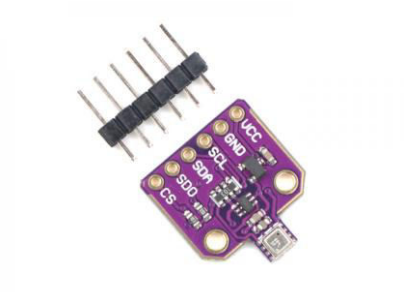
HC-05 module is an easy to use Bluetooth SPP (Serial Port Protocol) module, designed for transparent wireless serial connection setup. Serial port Bluetooth module is fully qualified Bluetooth V2.0+EDR (Enhanced Data Rate) 3Mbps Modulation with complete 2.4GHz radio transceiver and baseband. It uses CSR Bluecore 04-External single chip Bluetooth system with CMOS technology and with AFH(Adaptive Frequency Hopping Feature). It has the footprint as small as 12.7mmx27mm. Hope it will simplify your overall design/development cycle.



Features:

- Data bits:8, Stop bit:1,Parity:No parity, Data control. Supported baud rate: 9600,19200,38400,57600,115200,230400,460800.
- Can be used both as master and as slave
- Typical -80dBm sensitivity
- Low Power 1.8V Operation ,1.8 to 3.6V I/O
- integrated antenna

3.3. BME 680 sensor

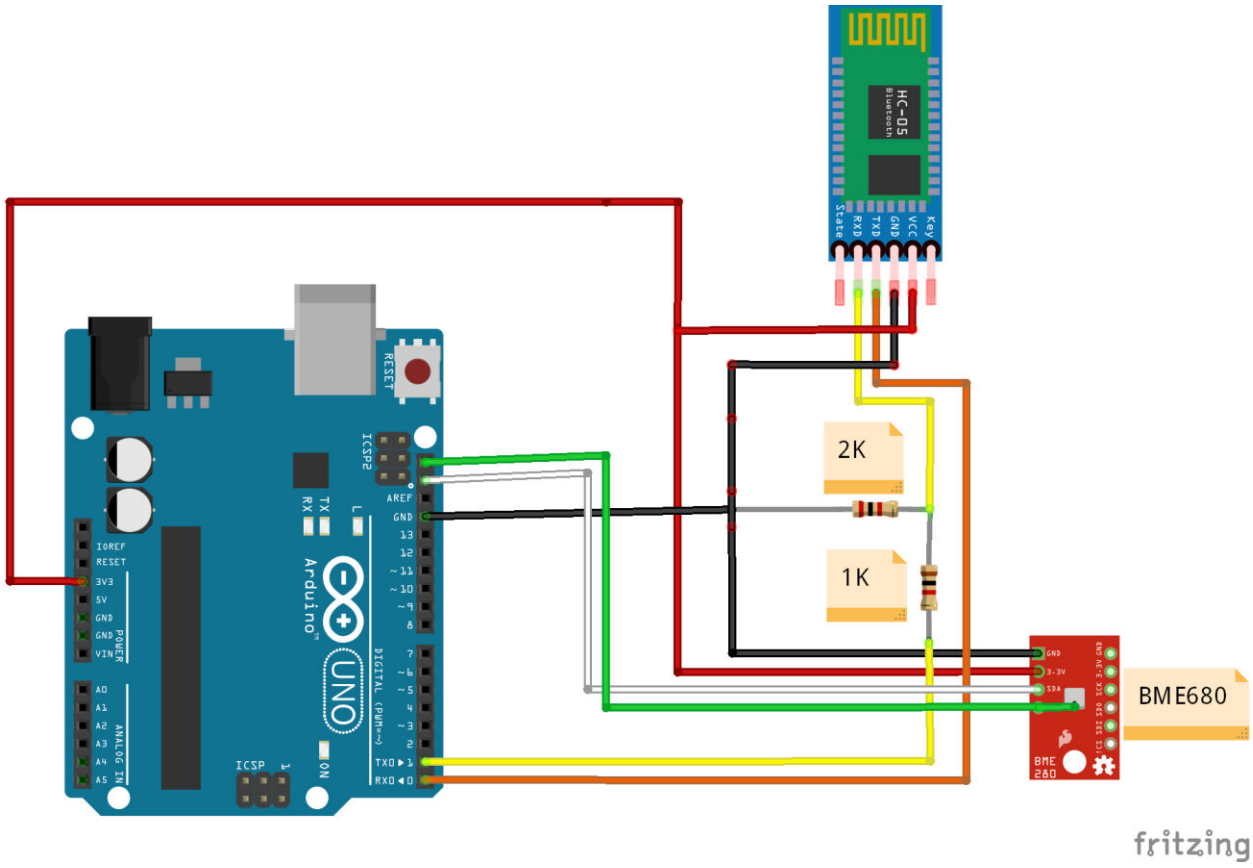


The BME680 is a digital 4-in-1 sensor with gas, humidity, pressure and temperature measurement based on proven sensing principles. The sensor module is housed in an extremely compact metal-lid LGA package with a footprint of only 3.0 × 3.0 mm² with a maximum height of 1.00 mm (0.93 ± 0.07 mm). Its small dimensions and its low power consumption enable the integration in battery-powered or frequency-coupled devices.

Features:

- Low power
- I2C (up to 3.4 MHz) and SPI (3 and 4 wire, up to 10 MHz) interfaces
- Individual humidity, pressure and gas sensors can be independently enabled/disabled
- Gas parameters are compliant to the ISO16000-29 standard
- IAQ can be precisely calculated based on the other parameters

4. Hardware design



5. Arduino program

```
#include <Adafruit_Sensor.h>
#include "Adafruit_BME680.h"

#define SEALEVELPRESSURE_HPA (1013.25)
#define ledPin 13

Adafruit_BME680 bme; // I2C

int ID = 4;
char deviceName[] = "HC-05";
int state = 0;

void setup()
{
   pinMode(ledPin, OUTPUT);
   digitalWrite(ledPin, LOW);
   Serial.begin(9600);

   if (!bme.begin(0x76))
   {
      Serial.println("Could not find a valid BME680 sensor, check wiring!");
      while (1);
   }

   // Set up oversampling and filter initialization
   bme.setTemperatureOversampling(BME680_OS_8X);
   bme.setHumidityOversampling(BME680_OS_2X);
   bme.setPressureOversampling(BME680_OS_4X);
   bme.setIIRFilterSize(BME680_FILTER_SIZE_3);
   bme.setGasHeater(320, 150); // 320*C for 150 ms
}

// Function for reading and sending parameters to Bluetooth module
        void sendParameters()
        {
          double temp = 0.0;
          double hum = 0.0;
          double pres = 0.0;
          double gas = 0.0;

          bme.performReading();

          bme.performReading();
          temp = temp + bme.temperature;
```

```
    hum = hum + bme.humidity;
    pres = pres + bme.pressure;
    gas = gas + bme.gas_resistance;

    bme.performReading();
    temp = temp + bme.temperature;
    hum = hum + bme.humidity;
    pres = pres + bme.pressure;
    gas = gas + bme.gas_resistance;

    bme.performReading();
    temp = temp + bme.temperature;
    hum = hum + bme.humidity;
    pres = pres + bme.pressure;
    gas = gas + bme.gas_resistance;

    bme.performReading();
    temp = temp + bme.temperature;
    hum = hum + bme.humidity;
    pres = pres + bme.pressure;
    gas = gas + bme.gas_resistance;

    if (! bme.performReading())
    {
      Serial.println("Starting transmission");
      Serial.println(ID);
      Serial.println(deviceName);
      Serial.println(0.0f);
      Serial.println(0.0f);
      Serial.println(0.0f);
      Serial.println(0.0f);
      Serial.println(0.0f);
    }
    else{
      Serial.println("Starting transmission");
      Serial.println(ID);
      Serial.println(deviceName);
      Serial.println((temp + bme.temperature)/5.0);
      Serial.println((hum + bme.humidity)/5.0);
      Serial.println((((pres + bme.pressure)/5.0) / 98.7) * 0.75006);
      Serial.println(((gas + bme.gas_resistance)/5.0) / 1000.0);
      Serial.println(bme.readAltitude(SEALEVELPRESSURE_HPA));
    }
}
```

```
void loop()
{
   if(Serial.available() > 0)
   {
      // Checks whether data is comming from the serial port
      state = Serial.read(); // Reads the data from the serial port
   }
   if (state == '0')
   {
      digitalWrite(ledPin, LOW); // Turn LED OFF
      state = 0;
      sendParameters();
   }else if (state == '1')
   {
     digitalWrite(ledPin, HIGH);
     state = 0;
     sendParameters();
   }
}
```

6. Android application

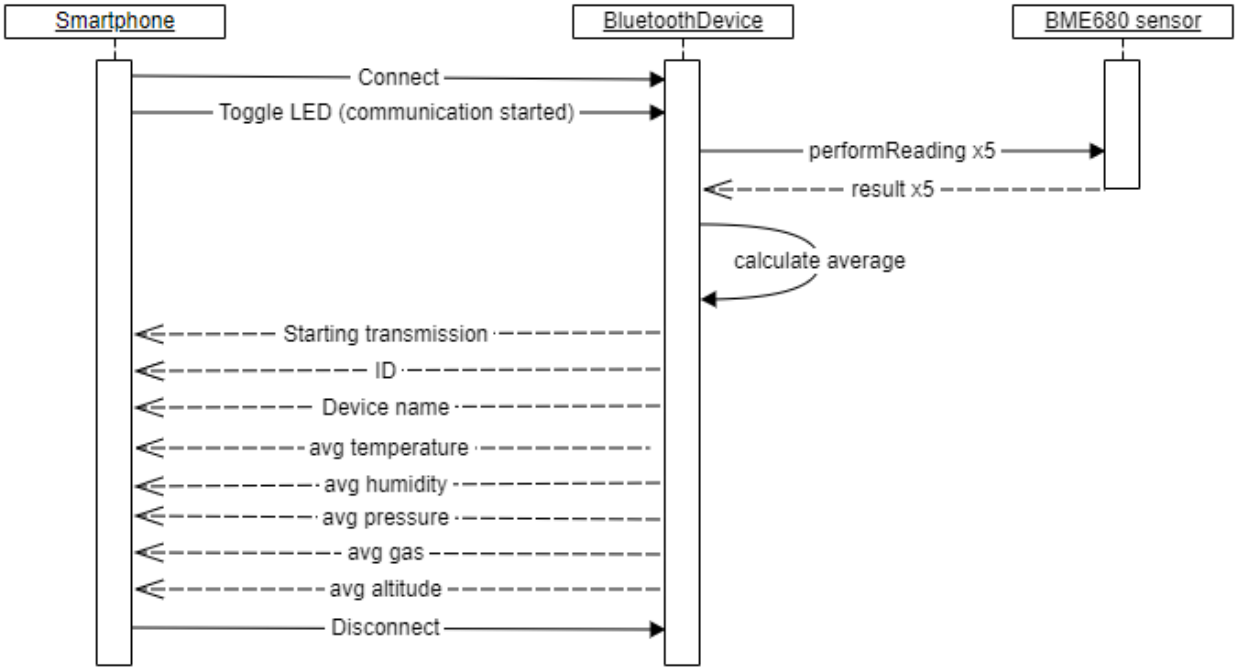
6.1. Classes

The project is structured in packages and classes, as follows:

- com.example.airqualitymonitor
  - FirebaseWrapper – Wrapper class for Firebase
  - GoogleMapActivity – Activity containing a map of all the devices in the area. Marker can be clicked and dialog with details will pop up
  - IAQProcessor – Class containing algorithm for calculating IAQ based on gas and humidity
  - MainActivity – Activity with the main purpose of searching, connecting and receiving data from Bluetooth devices
  - RealmWrapper – Wrapper class for Realm database
- db.model
  - DbEntry – Class representing one Realm database entry
  - DummyDbEntry – Same as DbEntry but for Firebase. The same one cannot be used.
- google.map
  - GPSTracker – Class that finds the current position of the smartphone when it connects to a Bluetooth device.

6.2. Protocol description

The communication between Bluetooth device and smartphone is described below by a sequence diagram:



6.3. Classes description

- FirebaseWrapper holds a reference to the cloud database. It also creates a listener that is called whenever the cloud database is changed. This class is strongly tied to RealmWrapper because the listener also updates the local database.

```
public class FirebaseWrapper
{
    private FirebaseDatabase database_;
    private DatabaseReference myRef_;
    private RealmWrapper realmWrapper_;
    ...
    public FirebaseWrapper(RealmWrapper realm, final Context context)
    {
        ...
        myRef_.addValueEventListener(new ValueEventListener() {
            @Override
            public void onDataChange(DataSnapshot dataSnapshot) {
                // This method is called once with the initial value and again
                // whenever data at this location is updated.
                ...
                // Updates data from local database
                realmWrapper_.createOrUpdateEntry(value.id_, value.deviceName_,
                                value.latitude_, value.longitude_,
                                value.temperature_, value.humidity_,
                                value.pressure_, value.co2_, value.altitude_,
                                value.lastUpdate_);
```

```
                ...
                if(MainActivity.shouldSendNotification)
                {
                    // It also pushes a notification when the cloud database is
                    // changed externally
                    ...
                }

            @Override
            public void onCancelled(DatabaseError error) {
                // Failed to read value
                Log.w("[MainActivity::onCancelled]", "Failed to read value.",
                        error.toException());
            }
        });
    }

    public void push(DummyDbEntry entry)
    {
        // Pushes new data to the cloud database. On update, it also changes the
        // local database
        myRef_.child(entry.id_).setValue(entry);
    }
}
```

- GoogleMapActivity is a Google map that creates markers for every device in the database. The marker can be clicked and a dialog will pop up with a detailed view of the parameters.

```
public class GoogleMapActivity extends FragmentActivity implements
OnMapReadyCallback
{
    private GoogleMap map;
    private Dialog dialog;
    ...
    @Override
    public void onMapReady(GoogleMap googleMap)
    {
        // Fetch all entries from the local database
        RealmResults<DbEntry> entries =
                MainActivity.realmWrapper.getEntireDatabase();

        // Add a marker for each device
        for(DbEntry entry : entries)
        {
            LatLng latLng = new LatLng(Double.parseDouble(entry.latitude_),
                    Double.parseDouble(entry.longitude_));
            googleMap.addMarker(new MarkerOptions().position(latLng)
                    .title(entry.id_ + " - " + entry.deviceName_));
        }

        googleMap.setOnMarkerClickListener(new GoogleMap.OnMarkerClickListener() {
            @Override
            public boolean onMarkerClick(Marker marker) {
                // Fetch all entries from the local database
                RealmResults<DbEntry> entries =
                    MainActivity.realmWrapper.getEntireDatabase();

                for(DbEntry entry : entries)
                {
                    if (marker.getTitle().equals(entry.id_ + " - " +
```

```
                        entry.deviceName_))
                    {
                        // Create dialog with all parameters for the clicked marker
                        String iaqScore = IAQProcessor.calculateIAQ(
                                Double.parseDouble(entry.humidity_),
                                Double.parseDouble(entry.co2_));

                        iaq.setText(iaqScore);
                        ...
                        dialog.show();
                    }
                }
                return false;
            }
        });
    }
}
```

- MainActivity is responsible for connecting and receiving data from Bluetooth devices

```
public class MainActivity extends AppCompatActivity
{
    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        ...
        //Enable GPS
        LocationManager lm = (LocationManager)getApplicationContext()
                .getSystemService(Context.LOCATION_SERVICE);
        if (lm != null && !lm.isProviderEnabled(LocationManager.GPS_PROVIDER))
        {
            Intent intent1 = new Intent(Settings.ACTION_LOCATION_SOURCE_SETTINGS);
            startActivityForResult(intent1, REQUEST_ENABLE_GPS);
        }

        // Eluetooth enable
        if (bluetoothAdapter_ == null)
        {
            Toast.makeText(this, "This device does not support bluetooth",
                    Toast.LENGTH_SHORT).show();
        }
        if (!bluetoothAdapter_.isEnabled()) {
            Intent enableBtIntent = new
                  Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
            startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);
        }
        ...

       // Set item click on ListView
       listView_.setOnItemClickListener(new AdapterView.OnItemClickListener() {
           @Override
           public void onItemClick(AdapterView<?> adapterView, View view,
              int i, long l)
           {
                readFromDeviceViaBluetooth(i);
           }
       });
```

```
            // On startScanButton_ click
            startScanButton_.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View view) {
                    // Stops scanning after a pre-defined scan period
                    handler_.postDelayed(new Runnable() {
                        @Override
                        public void run() {
                            bluetoothAdapter_.cancelDiscovery();
                            Toast.makeText(MainActivity.this, "Stop scan",
                                    Toast.LENGTH_SHORT).show();
                        }
                    }, SCAN_PERIOD);
                    bluetoothAdapter_.startDiscovery();
                    Toast.makeText(MainActivity.this, "Start scan",
                            Toast.LENGTH_SHORT).show();
                }
            });
            // On GPSButton_ click
            gpsButton_.setOnClickListener(new View.OnClickListener() {
                @Override
                public void onClick(View view) {
                    Intent myIntent = new Intent(MainActivity.this,
                            GoogleMapActivity.class);
                    MainActivity.this.startActivity(myIntent);
                }
            });
        }
    private void readFromDeviceViaBluetooth(int index)
    {
        BluetoothDevice device = foundDevices_.get(foundDevicesAddresses_.get(index));
        socket_ = null;

        // Trying to connect to bluetooth device
        try
        {
            socket_ = device.createRfcommSocketToServiceRecord(myUUID_);
            socket_.connect();
        }
        // Trying to connect to bluetooth device - fallback
        catch (IOException e)
        {
            Log.d(TAG, "Couldn't connect the first time. Trying fallback");
            try
            {
                socket_ = (BluetoothSocket)device.getClass()
                            .getMethod("createRfcommSocket",
                                new Class[] {int.class}).invoke(device,1);
                socket_.connect();
            }
            catch(Exception ex)
            {
               Log.d(TAG, "Could not connect not even the second time. Returning...");
                return;
            }
        }
        // Read data form connected Bluetooth device then disconnect
        ...
    }
}
```

- RealmWrapper manages the internal database

```
public class RealmWrapper
{
    ...
    public RealmWrapper(Context context)
    {
        ...
    }
    public void createOrUpdateEntry(final String id, final String deviceName,
                                    final String latitude,final String longitude,
                                    final String temperature,final String humidity,
                                    final String pressure, final String co2,
                                    final String altitude, final String lastUpdate)
    {
        DbEntry entry = realm.where(DbEntry.class)
                .equalTo("id_", id)
                .findFirst();

        realm.beginTransaction();

        if (entry == null)
        {
            // Add a new entry in the local database
        } else {
            // Update the existing information
        }
        realm.commitTransaction();
    }

    public void deleteEntry(final String id)
    {
        DbEntry entry = realm.where(DbEntry.class)
                .equalTo("id_", id)
                .findFirst();

        realm.beginTransaction();

        if (entry != null)
        {
            entry.deleteFromRealm();
        }
        realm.commitTransaction();
    }

    public RealmResults<DbEntry> getEntireDatabase()
    {
        RealmResults<DbEntry> entriesDb = realm.where(DbEntry.class).findAll();
        return entriesDb;
    }

    public void close()
    {
        realm.close();
    }
}
```

Bibliography

www.fritzing.org
www.arduino.cc
https://cdn-shop.adafruit.com/product-files/3660/BME680.pdf
http://www.electronicaestudio.com/docs/istd016A.pdf
https://www.farnell.com/datasheets/1682209.pdf

https://firebase.google.com/docs
https://developer.android.com/guide/topics/connectivity/bluetooth

https://github.com/catalinvlad192/AirQualityMonitor

Optimus Electronic Introduction to Arduino