IoT Project Document

Team Members

Kholoud Khaled Attia Mahmoud
Mennat Allah Mamdouh Mohamed Ahmed Asfour
Mohamed Arafat Abdel Moneim Arafat
Mohamed Atta Faheem
Mohamed Emad El-deen Ahmed Khalifa
Mostafa Mohammed Abdel-moneam Abdo
Omar Mohamed Abdel-latif Alfarghaly
Sarah Ahmed Mohamed Abd Ellatif Elnady
Yasmin Abdel Latif Mohammed Hussein Kloub

Group number: 1

Supervised by: Dr. Ahmed Shawky Moussa

Submission date: May 2018

Table of Contents

1. Introduction

Many activities nowadays need time, actions to trigger them or other activities' results to start. The 'Internet of Things' (IoT) track focuses on the automation of sequential or parallel tasks depending on the internet connections available between all parts of devices.

The 'Smart Home' is an IoT application in which the house can manage, start, stop, enable and disable your devices/appliances depending on some triggers such as the owner's location, the climate, the smoke, sound and light conditions, etc. A great number of IoT applications including smart homes can be autonomous and thus require minimal user intervention.

The system described here is a simple smart home that collects various sensory data through different sensors and microprocessor-based boards installed and spread over several house rooms. Through GPS tracking, our system is also able to calculate how far the owner of the house is. Based on that, it takes the appropriate actions to prepare for the owner's arrival.

In this document, we illustrate our smart home system and its architecture. In addition, we give a detailed description of the various components used. We also show a demo of how it works and we conclude with possible future extensions of the system.

2. Background

The ‘Internet of Things’ is a network of smart physical devices that communicate together and exchange data. These devices or “things” range from appliances (refrigerators, washing machines, air conditioners, …) to wearable items (shoes, wristbands, …). They are “smart” devices because they contain sensors that collect data from the physical world and accordingly take the appropriate actions. They are also wirelessly connected to each other and to the internet in order to exchange data.

There are several various IoT applications available and their main goal is to make our lives easier by automating as much of the repetitive, attention-requiring tasks as possible. The IoT applications contribute to a lot of different fields including health, agriculture, manufacturing, home automation, etc.

One of the widely used applications of IoT is the ‘Smart Home’. A smart home is a house with an automated system that controls lighting, climate, appliances, security, etc. to make people’s lives easier. Some smart homes can be managed using phones or tablets or even voice commands from the house owners.

3. Proposed System

3.1. Overview

Our ‘Smart Home’ is composed of different devices; each device performs a certain task as we will discuss in section 3.3. The smart home system controls the operation of the individual devices and monitors their status. It can also track the homeowner’s location (using a mobile application we made) and start the devices when the owner is close to home thus relieving the homeowner from the burden of turning on each device manually when he/she arrives.

3.2. Objective

The goal of our project is to use the Intel Galileo development boards to make small and simple IoT systems and prove that such systems can be connected together through the internet to make a larger IoT application (which is our smart home).

3.3. Devices & Their Function

3.3.1. Smart Light Monitor

Some street lamps as well as solar walkway lights use photoresistors to detect the absence of sunlight and turn on the lights. We will use a photoresistor, that is, a light sensor, specifically; an electronic component that provides a variable resistor that changes the resistor value based on the incident light intensity. The smart light monitor uses a Galileo board with a light sensor to measure the light intensity in a room and takes the appropriate action (e.g. turn on/off a LED) according to the collected sensory data.

3.3.2. Smart Sound Detector

The smart sound detector uses a Galileo board with a sound sensor to detect any sound (clap,...) and perform a certain task (e.g. turn on the light).

3.3.3. Smart Motion Detector

The smart motion detector uses a Galileo board with a motion sensor to detect the movement of humans, animals or any other objects and perform a certain task (e.g. turn on/off the light).

3.3.4. Smart Temperature Indicator

The smart temperature indicator uses a Galileo board with a temperature sensor to measure the temperature of a room.

3.3.5. Fire Alarm

The fire alarm uses a Galileo board connected to a smoke sensor to monitor the smoke conditions and alerts the user when there is a fire (when it senses flame) by turning on the buzzer. The user can stop the buzzer sound by pressing a push button and that way the system makes sure that the user is aware of the existence of the fire.

4. Required Components

The following table shows the components we used to make our project. These items are used to build the individual devices in our system as we discussed before.

Table 4.1. All the components required in our project

Item	***Amount***
Intel Galileo development board	9
Breadboard	9
Jumper wires	-
LDR (Light-Dependent Resistor)	4
Temperature sensor (LM35)	1
Sound sensor	1
Motion sensor	1
Smoke detector	1
LED	8
Buzzer	1
Push button	1

5. Design

5.1. Architecture

The architecture we used in this project is the client/server architecture (illustrated in figure 5.1.1). The server is responsible for controlling the start/stopping of the client applications and monitoring them. It also contains the location database which stores the location information of the owner when it is received and contains the service that allows map applications to get the location data available in the database. On the other hand, we have multiple clients; each client application runs on one of the devices mentioned before and communicates with the server.

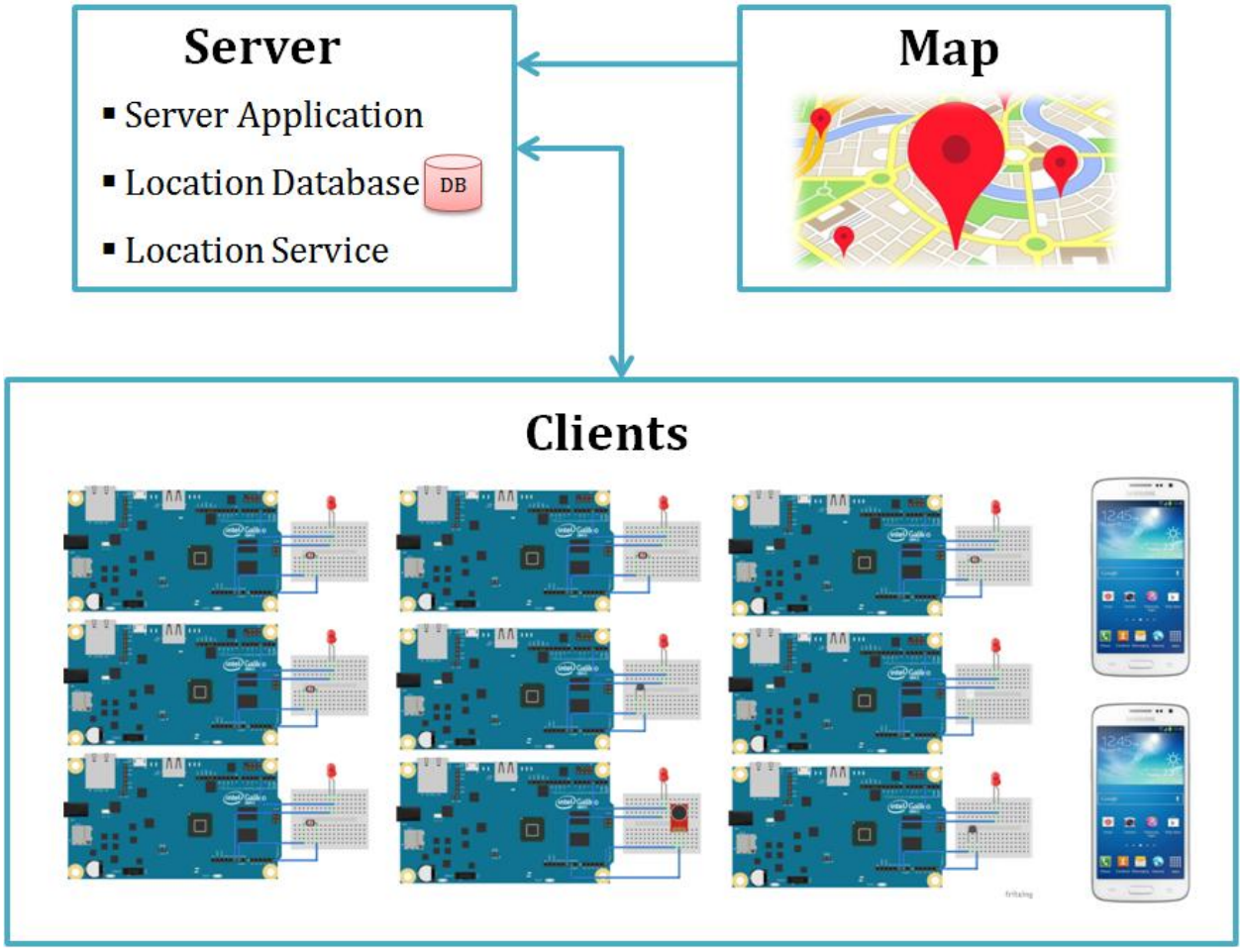

Figure 5.1.1. The architecture of our smart home

5.2. *Process Flowchart*

In this subsection, we present the flowchart of the server's process (figure 5.2.1). As you can see, the clients (including phone clients) connect to the server and when the server receives location data from a phone client, it calculates the distance between the house and the owner's location. According to this distance, the server either sends a 'start'/'close' command to the other clients so that their applications can start/stop working. The location data can also be accessed from and shown on a map application that gets that data from the service running on the server.

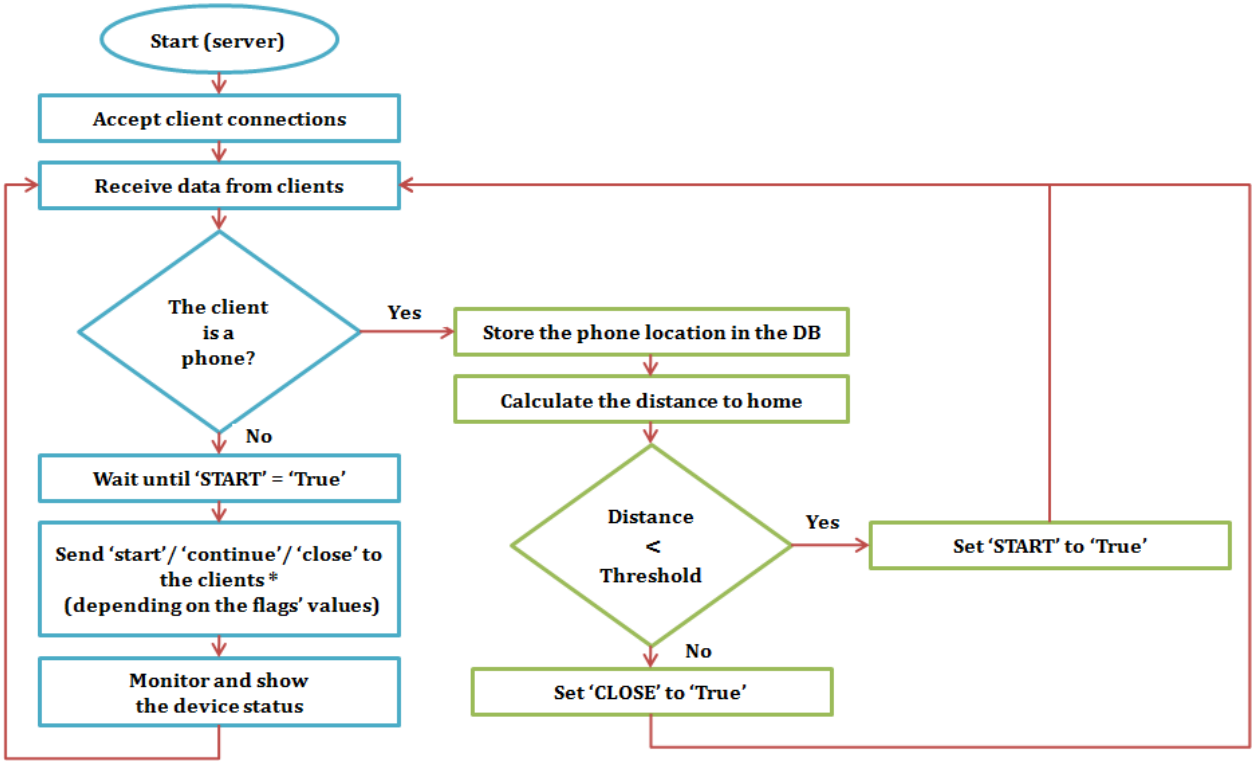

Figure 5.2.1. The server's process flowchart

6. *Implementation & Testing*

In this subsection, we present some details of the code used in the three main parts of our project which are: the server, the android app and the clients.

- **Server:**

The server consists of 4 main functionalities. The first is a database. We used a simple sqlite library in our implementation:

```
def createDatabase(databaseName):
     connection = sqlite3.connect(databaseName)
     cursor = connection.cursor()
     sql_command = """
     CREATE TABLE Location (
     ID INTEGER PRIMARY KEY,
     Board_ID VARCHAR(10) NOT NULL,
     Latitude DOUBLE NOT NULL,
     Longitude DOUBLE NOT NULL,
     Date DATE NOT NULL,
     Time TIME NOT NULL
     );
     """
     try:
          cursor.execute(sql_command)
     except:
          pass
     connection.commit()
connection.close()
```

In this snippet we create a simple table to store our mobile location information. This table consists of longitude, latitude, date, time and board ID. We also created methods to insert and fetch data from the database like:

```
def saveData(databaseName,date,boardID,longitude,latitude):

def saveJsonData(databaseName,locationData):

def insert_update(databaseName,data):

def deleteAllData(databaseName):

def getAllData(databaseName): # return data as list of
dictionary

def getMostRecentData(databaseName):

def get(databaseName,board_id):
```

Our next part in the server is a simple api that can be called by our javascript code (which is responsible for drawing) to get the last location to be drawn on the map.

```
from flask import Flask, render_template
import sqliteDB, json

DB_PATH = "Board_Location.db"
app = Flask(__name__)
@app.route('/get_loc')
def get_loc():
    return json.dumps(sqliteDB.getMostRecentData(DB_PATH))
app.run(debug=True)
```

The next part is the javascript used to draw the map. We used a library called **gmplot** draw the map and show the movements fetched from the database. It uses google maps API. The map made using the 'MapMaker' application can run on the server or use the api described above to get the location data from anywhere. This is a small snippet that shows how we used the Gmplot library to draw our basic map. However, the "MapMaker' adds more parts to the gmplot map to enable it to continuously show the user's location while he/she is moving.

```
def
initialize_map(center_lat,center_long,zoom,color,num_markers)
:
    gmap =
gmplot.GoogleMapPlotter(center_lat,center_long,zoom)
    gmap.marker(center_lat,center_long,color = color)
    gmap.draw(MAP_NAME)

    map_file = open(MAP_NAME,'r')
    map_data = map_file.read()
    map_file.close()
```

The final part in our server is the controller that controls the logic of the whole project. It checks the incoming locations from mobiles and if they are close enough to home it will order the sensors to run and if no mobile is close enough it will order the sensors back to sleep and also serves as the central hub of all communications as it opens a port and keeps listening to it.

```
def calculate_distance(loc1,loc2):
   R = 6373.0  # approximate radius of Earth in km
   lat1  = radians(loc1[0])
   long1  = radians(loc1[1])
   lat2  = radians(loc2[0])
   long2  = radians(loc2[1])
   dlong = long2 - long1
```

```
    dlat = lat2 - lat1
    a = sin(dlat/2)**2 + cos(lat1) * cos(lat2) * sin(dlong/2)**2
    c = 2 * atan2(sqrt(a),sqrt(1 - a))
    distance = R * c * 1000
    return distance

class ClientThread(threading.Thread):
    def __init__(self,clientIP,connection):
        threading.Thread.__init__(self)
        self.ip = clientIP
        self.connection = connection

    def run(self):
        global START,CLOSE
        #print('Thread opened with IP',self.ip)
        i = 0
        while i < ITERATIONS:
            data = self.connection.recv(BUFFER_SIZE)
            data = data.split("}") # more than 1 json object
            data = data[len(data)-2] + "}"
            try:
                data = json.loads(data)
                print ('Message from ',self.ip,' ----- ',data)
                if (data['source'] == 'temp' or data['source'] == 'light'
                    or data['source'] == 'sound'
                    or data['source'] == 'motion'
                    or data['source'] == 'smoke'):
                    if not START:
                        while not START:
                            time.sleep(1)
                        self.connection.send('start')
                        #print(data)
                    else:
                        time.sleep(2)
                        if CLOSE or i == ITERATIONS-1:
                            self.connection.send('close')
```

```
                    break
                else:
                    self.connection.send('continue')
            else: # source = phone
                my_id = int(data['source'].split('_')[1])
                if my_id == 0: # home location
                    print(data)
                    global HOME_LOCATION
                    HOME_LOCATION = (data['lat'],data['long'])
                    MapMaker.initialize_map(HOME_LOCATION[0],
                                            HOME_LOCATION[1],20,
                                            color_dict[0],PHONE_BOARDS)
                    webbrowser.open("map.html")
                    break
                else:
                    sqliteDB.saveJsonData(DB_NAME,data)
                    # Calculate distance between home location and me
                    distance = calculate_distance((data['lat'],
                                                   data['long']),
                                                  HOME_LOCATION)
                    print('Distance from ',data['source'],
                          'to home = ',distance)
                    self.connection.send(json.dumps(
                                           {'distance':distance}))
                    if distance <= 10.0 and not START: # Turn on devices
                        START = True
                        print("Start devices -----")

                    if distance >= 10.0 and START and not CLOSE:
                    # Turn off devides
                        CLOSE = True
                        print("Close devices -----")
                    time.sleep(2)
        except:
            print("JSON ERROR!")
            self.connection.send('error')
```

```
            i += 1
        print (self.ip,' ----- connection closed')
        self.connection.close()
###############################################################################
sqliteDB.createDatabase(DB_NAME)

my_server = socket.socket()
my_server_ip = raw_input("IP: ") #"192.168.1.102"
my_server_port = int(raw_input("Port: ")) #5002

my_server.bind((my_server_ip,my_server_port))
my_server.listen(9)  # wait for client connection

i = 0
while i < TOTAL_BOARDS:
   client,address = my_server.accept() # accept connection request
   print('Thread # ',i,' ----- connected to client:', address)
   thread  = ClientThread(address,client)
   thread.start()
   i += 1

my_server.close()
```

- **Clients:**

All client applications connect to the server and communicate with it. At first they send a json object containing the ‘source’ which is the device the client runs on (light, sound, …). Then, they wait until they receive the ‘start’ command from the server. As soon as they get it, they start running their sensor applications, get the readings from sensors, send their status to the server and receive a command (either ‘continue’ or ‘close’) from the server. When they receive ‘close’, the clients stop their applications. The following code shows what happens when the server sends ‘start’ to the motion client for instance.

```
if data == 'start':
     while 1:
          jsonData["motion"] = int(lib.return_string())
          print jsonData
          s.send(json.dumps(jsonData))
          data = s.recv(buffer_size)
          print data
          if  data == 'close':
               break
```

- **Android App:**

The android app consists of two major parts which are: TCP client to connect to the server, and a location provider to send to the server. For the TCP we used a very simple client to connect to the IP and port of the server.

```
public class TcpClient {
    public void sendMessage(final String message);
    public void stopClient();
    public void run();
    public interface OnMessageReceived();
}
```

The second major part is getting the location. We used *Fused Location Provider API* to get the location. Fused location provider is a location API in Google Play services that intelligently combines different signals to provide the location information, and manages the underlying location technologies, such as GPS and Wi-Fi. For example, you can request the most accurate data available, or the best accuracy possible with no additional power consumption.

```
public class MainActivity extends AppCompatActivity {

    private FusedLocationProviderClient mFusedLocationClient;
    private LocationRequest mLocationRequest;
    private LocationCallback mLocationCallback;


    private void createLocationRequest() {
        mLocationRequest = new LocationRequest();
        mLocationRequest.setInterval(10000);
        mLocationRequest.setFastestInterval(2000);
      mLocationRequest.setPriority(LocationRequest.PRIORITY_HIGH_ACCURACY);

        LocationSettingsRequest.Builder builder = new
LocationSettingsRequest.Builder()
                .addLocationRequest(mLocationRequest);
    }

        private void stopLocationUpdates() {
        mFusedLocationClient.removeLocationUpdates(mLocationCallback);
    }

    private void startLocationUpdates() {

        if (ActivityCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_FINE_LOCATION) != PackageManager.PERMISSION_GRANTED &&
ActivityCompat.checkSelfPermission(this,
Manifest.permission.ACCESS_COARSE_LOCATION) != PackageManager.PERMISSION_GRANTED)
{
            ActivityCompat.requestPermissions(this,
                    new String[]{Manifest.permission.ACCESS_FINE_LOCATION},1);
```

```
        }


        mFusedLocationClient.requestLocationUpdates(mLocationRequest,
                mLocationCallback,
                null /* Looper */);
    }

    private void initLocationCallback()
    {
        mLocationCallback = new LocationCallback() {
            @Override
            public void onLocationResult(LocationResult locationResult) {
                if (locationResult == null) {
                    return;
                }
                List<Location> locations=locationResult.getLocations();
                if(locations.size()>0)
                {
                    longitude = locations.get(0).getLongitude();
                    latitude = locations.get(0).getLatitude();
                    locationTextView.setText("(" + String.valueOf(longitude) + ","
+ String.valueOf(latitude) + ")");
                    if (sendAutoCheckBox.isChecked()) {
                        sendLocation();
                    }
                }

            };
        };

    }

    private void sendLocation()
    {
        JSONObject obj = new JSONObject();
        try {
            obj.put("long", longitude);
            obj.put("lat", latitude);
```

```
            obj.put("source", boardNameEditText.getText().toString());
        } catch (JSONException e) {
            e.printStackTrace();
            return;
        }
        if (mTcpClientSend != null) {
            mTcpClientSend.sendMessage(obj.toString());
        }
    }
    //rest of class

}
```

7. Results

The following figures are screenshots of the results of the map application. The application got the user's location data from the database using the api running on the server and showed them on the map. The red marker represents the 'home' while other markers represent the owners. The lines drawn represent the paths that the user moved along.

Figure 7.1. Map screenshot #1

Figure 7.2. Map Screenshot #2

8. Conclusion

The IoT technology has lately seen a variety of applications in daily life. In this document, we described in detail a simple Smart Home application. We discussed the project components, design, implementation and testing situations. Our Smart Home project could be easily expanded to more home appliances such as ovens and TVs and it may be used with multiple home owners, with each owner having different needs and requirements. Furthermore, although the project made use of simple components such as LEDs and breadboards, the described model can be easily adapted to real-life situations where LEDs are replaced by light bulbs and breadboards by professional components casing.

9. References

[1]https://navinbhaskar.wordpress.com/2015/09/03/python-on-intel-galileoedison-part6-light-sensor/

[2]https://learn.sparkfun.com/tutorials/galileo-experiment-guide