# Public Transport Optimization

## Phase 3: Development Part 1

In this phase that we start building the project by writing necessary code for the ESP32 module and program for application.

### *Sensors Used:*

- Gyroscope
- Accelerometer
- GPS

### *Working Flow*

- The code starts by defining the variables that will be used in the program.
- Next, there are two functions: one to read GPS data and another to analyze it.
- The function reads GPS data from the device and stores it in gps >data .
- It then analyzes the data using an machine learning algorithm.
- In order for this code to work properly, you need to include three libraries: stdio , stdlib , and time .
- The code attempts to read the GPS data and print it on the console.
- #include // Include the stdio library #include // Include the stdlib library #include // Include time library #include // Include the GPS library
- The code starts with a function to read the GPS sensor data.
- The first line of code is void readGPSData(struct GpsData *gps) { The next line of code is } This means that this function has finished and no more functions will be called in it.
- This function starts by declaring a structure named gps, which contains information about the GPS sensor.
- Then, it declares another structure named accel, which contains information about the accelerometer sensor.
- Finally, it declares another structure named gyroscope, which contains information about the gyroscope sensor.
- It then implements each of these sensors' reading methods and ends by returning an integer value for each one (the number of bytes written).
- The last two lines are just used as placeholders so that when you call other functions they can write their results into these structures without having to declare them again every time they're needed.
- The code implements the reading of GPS sensor data, accelerometer data, and gyroscope data.
- The code above is a function that reads the GPS sensor data.
- The GPS sensor will provide latitude and longitude values to the gps structure.
- The gps structure will then be updated with these values.
- The code above is a function that reads the accelerometer data.
- The AccelerometerData structure will be updated with acceleration values in this function as well.
- The code starts by using the Google Maps Directions API to request traffic information.
- The response is then processed to find high and low traffic areas.
- Finally, the results are displayed on a map using Google Maps JavaScript API.
- The code starts by creating an object called directions with two properties: startLocation and destinationLocation .
- It also creates an object called result that will hold all of the data from the response.
- Next, it calls getRoute() which returns a route object that contains information about how long it will take for you to travel between two locations in your car or public transportation (e.g., bus).

- Then, it uses setMapType() method to change the map type from satellite view to road view so that you can see where roads are located on top of each other or near each other as well as what types of roads they are (e.g., highway, city street).
- Finally, it sets up some variables before calling analyzeTrafficData() function which takes in a route object and analyzes its data based on time based intervals like every 10 minutes or every hour so that we can find out when there's more traffic than usual at certain times during day/night hours respectively.
- The code is used to find high and low traffic areas using Google Maps.
- The code uses the Google Maps Directions API to request traffic information.
- The response is processed to find high and low traffic areas.
- The code starts by initializing the three sensors.
- The GPSData structure is initialized with a pointer to an instance of the struct, and then it is assigned values for latitude, longitude, altitude, speed, and course.
- The AccelerometerData structure is initialized with a pointer to an instance of the struct and then it is assigned values for x axis acceleration in meters per second squared ($m/s^2$), y axis acceleration in $m/s^2$, z axis acceleration in $m/s^2$.
- The GyroscopeData structure is initialized with a pointer to an instance of the struct and then it is assigned values for pitch angle (degrees) and roll angle (degrees).
- Then these structures are passed into functions that will analyze them: gps_getLatLong(), accel_getXAcceleration(), gyro_getPitchAngle() & gyro_getRollAngle().
- These functions return data from their respective sensors as well as other information such as accuracy or battery life remaining on each sensor.
- Finally this code calls main() which starts up your application's main loop where you can do whatever you want!
- The code attempts to initialize the sensors (specific to your hardware and libraries) The code does not provide any additional information on how to do this.

### *Esp32 microcontroller*

```
#include <TinyGPS++.h>
#include <WiFi.h>
#include <HTTPClient.h>
#include <Wire.h>
#include <gps.h> // Include the GPS library
#include <accelerometer.h> // Include the accelerometer library
#include <gyroscope.h> // Include the gyroscope library

#define TX_PIN 2   // TX pin of NEO-6 GPS module
#define RX_PIN 15  // RX pin of NEO-6 GPS module
#define PIR_SENSOR_PIN 12  // Pin for the PIR sensor

const char* ssid = "wifi_ssid";
const char* password = "wifi_password";
const char* serverUrl = "http://server_under_construction";

TinyGPSPlus gps;
int passengerCount = 0;

void setup() {
  Serial.begin(115200);
  Serial1.begin(9600, SERIAL_8N1, RX_PIN, TX_PIN); // Initialize the GPS module
  pinMode(PIR_SENSOR_PIN, INPUT);
```

```
  // Connect to Wi-Fi
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED) {
    delay(1000);
    Serial.println("Connecting to WiFi...");
  }
  Serial.println("Connected to WiFi");
}

void loop() {
  // Check PIR sensor for passenger detection
  if (digitalRead(PIR_SENSOR_PIN) == HIGH) {
    // Passenger boarded
    passengerCount++;
    Serial.println("Passenger boarded. Count: " + String(passengerCount));
    delay(1000); // Debounce
  }

  while (Serial1.available() > 0) {
    if (gps.encode(Serial1.read())) {
      if (gps.location.isUpdated()) {
        float latitude = gps.location.lat();
        float longitude = gps.location.lng();

        // Send GPS data and passenger count to a server
        sendDataToServer(latitude, longitude, passengerCount);
      }
    }
  }
  delay(10000);  // Update GPS data every 10 seconds
}

void sendDataToServer(float latitude, float longitude, int passengers) {
  HTTPClient http;

  String data = "lat=" + String(latitude, 6) + "&lng=" + String(longitude, 6) + "&passengers=" +
String(passengers);
  http.begin(serverUrl);
  http.addHeader("Content-Type", "application/x-www-form-urlencoded");

  int httpResponseCode = http.POST(data);

  if (httpResponseCode > 0) {
    String response = http.getString();
    Serial.println("HTTP Response Code: " + String(httpResponseCode));
    Serial.println(response);
  } else {
    Serial.println("HTTP Error");
  }

  http.end();
}
```

***Java application program***

```
import android.os.AsyncTask;
import android.support.v7.app.AppCompatActivity;
import android.os.Bundle;
import android.widget.TextView;
```

```
import org.json.JSONException;
import org.json.JSONObject;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.URL;
public class MainActivity extends AppCompatActivity {
private TextView gpsTextView;
private TextView passengerCountTextView;
@Override
protected void onCreate(Bundle savedInstanceState) {
super.onCreate(savedInstanceState);
setContentView(R.layout.activity_main);
gpsTextView = findViewById(R.id.gpsTextView);
passengerCountTextView =
findViewById(R.id.passengerCountTextView);
// Perform an HTTP request to your server to get GPS data and
passenger count
new RetrieveGPSDataTask().execute();
}
private class RetrieveGPSDataTask extends AsyncTask<Void, Void,
JSONObject> {
@Override
protected JSONObject doInBackground(Void... voids) {
try {
URL url = new URL("http://server_under_construction");
HttpURLConnection connection = (HttpURLConnection)
url.openConnection();
connection.setRequestMethod("GET");
InputStream inputStream = connection.getInputStream();
BufferedReader reader = new BufferedReader(new
InputStreamReader(inputStream));
StringBuilder result = new StringBuilder();
String line;
while ((line = reader.readLine()) != null) {
result.append(line);
}
return new JSONObject(result.toString());
} catch (IOException | JSONException e) {
e.printStackTrace();
return null;
}
}
@Override
protected void onPostExecute(JSONObject result) {
if (result != null) {
try {
double latitude = result.getDouble("lat");
double longitude = result.getDouble("lng");
int passengers = result.getInt("passengers");
// Display the GPS coordinates and passenger count in
the app
gpsTextView.setText("Latitude: " + latitude +
"\nLongitude: " + longitude);
passengerCountTextView.setText("Passengers: " +
passengers);
} catch (JSONException e) {
e.printStackTrace();
```

```
}
} else {
gpsTextView.setText("Error retrieving GPS data.");
passengerCountTextView.setText("Passenger count not
available.");
}
}
}
}
```