CHAPTER– 4

Implementation

4.1 EXPERIMENTAL SETUP

To successfully implement our GPS-based data logging system, we need to gather several essential components and ensure they are properly configured. The following outlines the key prerequisites and steps involved in our setup.

4.1.1 Prerequisites

1. **GPS Module and Antenna:**
   - GPS Module: We will use a GPS module capable of receiving signals from GPS satellites. The module must connect to at least three satellites in the vicinity to perform trilateration.
   - Antenna: An external antenna is required to enhance signal reception, ensuring the GPS module can accurately determine its position by calculating distances from the satellites.
2. **Arduino UNO R3 Microcontroller:**
   - This microcontroller will serve as the central processing unit of our system. It will handle the communication between the GPS module and other components, process the received data, and execute necessary commands.
3. **SIM800L GSM Module:**
   - Functionality: The SIM800L module will be used to send the obtained latitude and longitude data over GSM networks. It supports basic operations such as sending SMS and connecting to the internet.
   - SIM Card: A 2G SIM card is required for the module to connect to GSM networks and transmit data.
   - Advantages: We opted for a GSM module instead of a Wi-Fi module because setting up Wi-Fi on moving vehicles like buses is challenging and less reliable.

4.1.2 Setup and Configuration

1. **Hardware Connections:**

   - GPS Module: Connect the GPS module to the Arduino UNO R3. The antenna should be positioned to have a clear view of the sky for optimal satellite signal reception.
   - SIM800L Module: Connect the SIM800L module to the Arduino. Ensure proper power supply and SIM card insertion for the module to function correctly.

2. **Programming the Arduino:**
   - Write and upload a program to the Arduino UNO R3. The program should:
   - Initialize the GPS module and start receiving location data.
   - Process the received GPS data to extract latitude and longitude.
   - Send the extracted data to the SIM800L module for transmission.

3. **Data Transmission:**
   - The SIM800L module will send the location data over the GSM network to a predefined server. This server could be hosted locally or remotely, depending on the project's requirements.

4. **Local Server and Data Handling:**
   - XAMPP Server: Set up a local server using XAMPP, which includes Apache, MySQL, PHP, and Perl. This server will handle data storage and retrieval.
   - Node-RED: Use Node-RED, a flow-based development tool for visual programming, to process incoming data. Node-RED will be configured to receive data from the Arduino via USB ports.
   - MySQL Database: The data received by Node-RED will be transferred to a MySQL database. This database will store the GPS data for later access and analysis.
   - phpMyAdmin: Access and manage the MySQL database using phpMyAdmin, a web-based interface that simplifies database operations.

5. **Data Visualization and Analysis:**
   - Once the data is stored in the MySQL database, it can be retrieved and analyzed. Visualization tools can be used to display the data in meaningful ways, such as plotting the GPS coordinates on a map to track the movement of the vehicle.

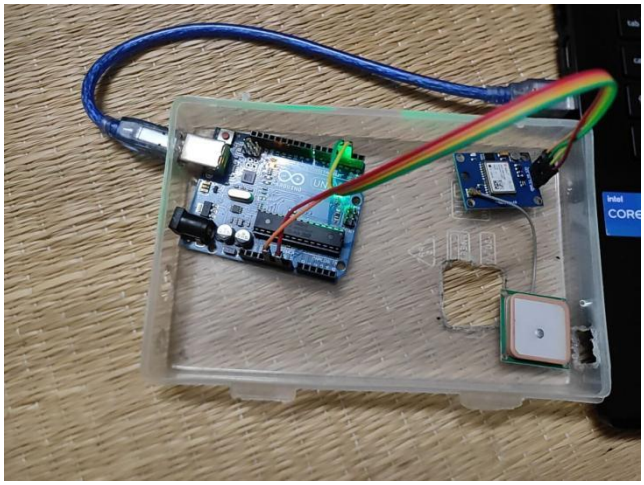


Fig 4.1 Hardware Module

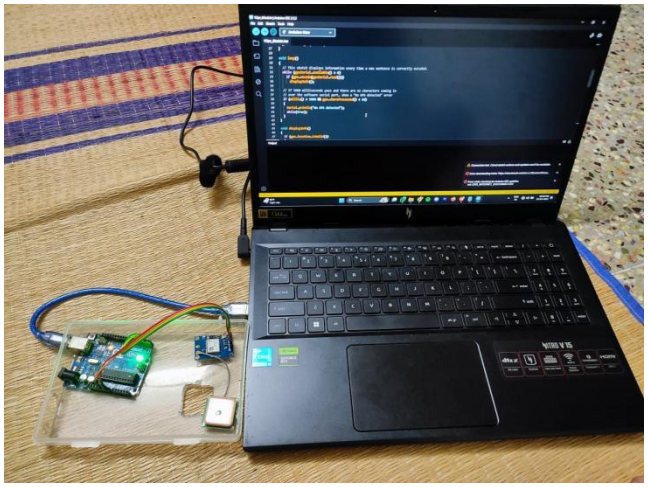


Fig 4.2 System Setup

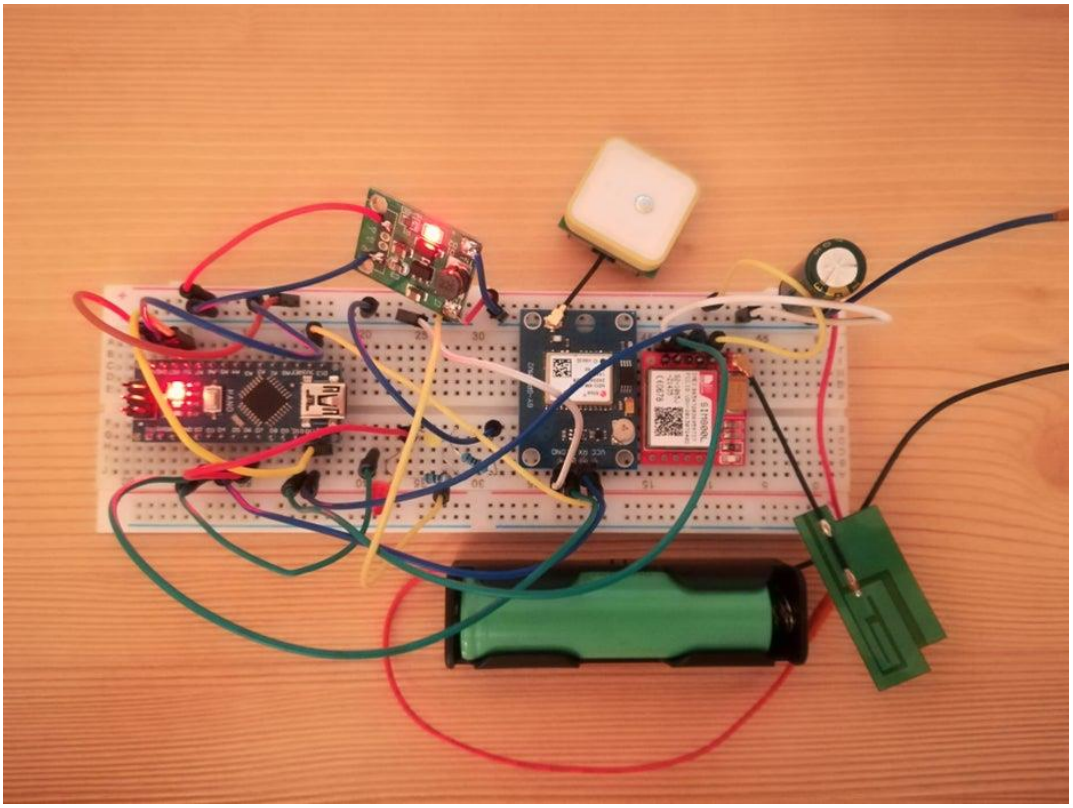


Fig 4.3 Completed Module

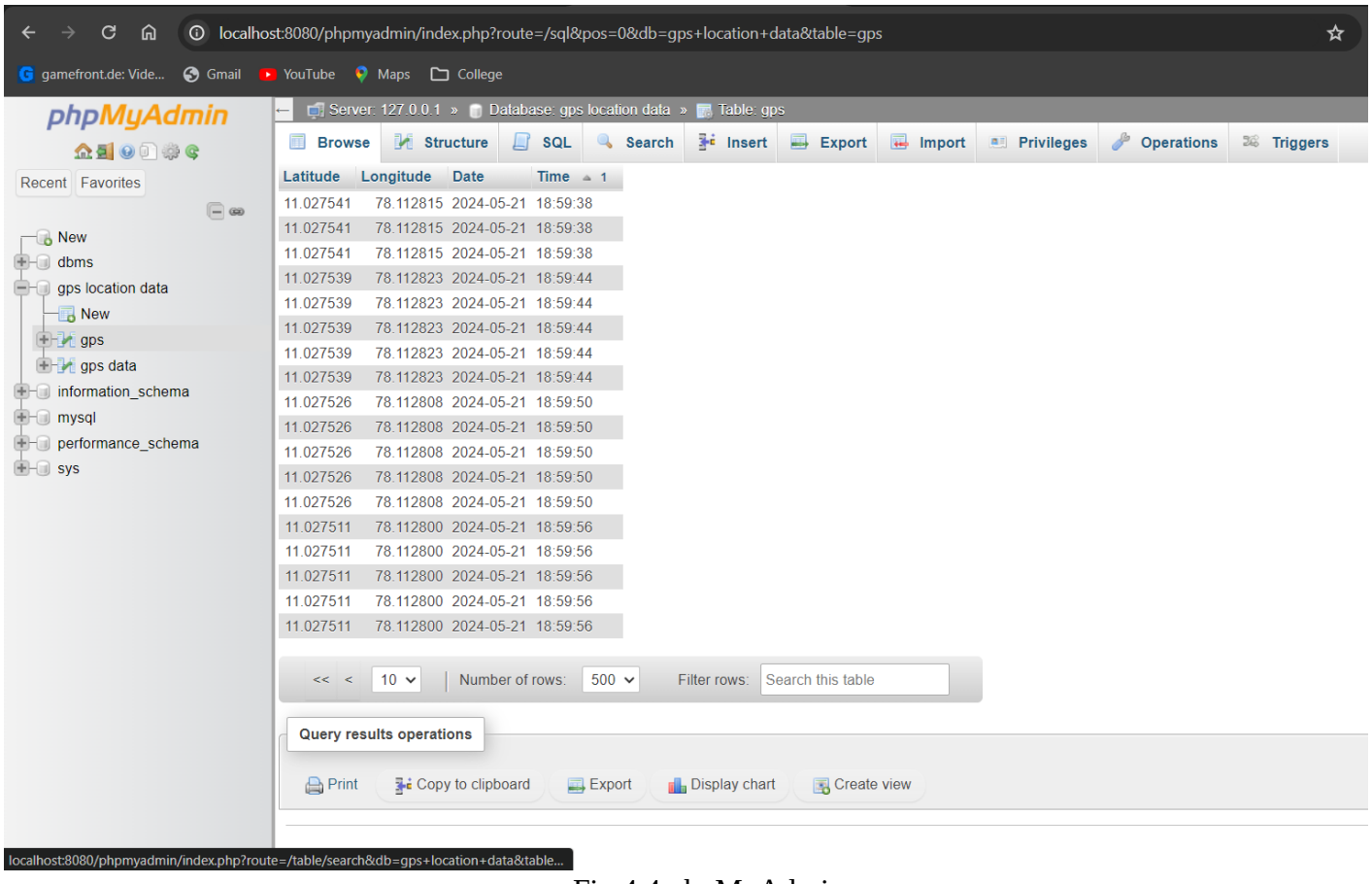


Fig 4.4 phpMyAdmin

4.2 MODULES

This GPS-based data logging system project can be divided into several key modules, each responsible for different aspects of the system's functionality. Below are the main modules of the project:

4.2.1 GPS Data Acquisition Module

Components:

- GPS Module: A device capable of receiving signals from GPS satellites and calculating its position.
- Antenna: Enhances the reception of satellite signals.

Functions:

- Connects to multiple GPS satellites to determine precise location coordinates.
- Outputs latitude, longitude, and other relevant data.
-

4.2.2 Microcontroller Module

Components:

- Arduino UNO R3: The central microcontroller used for processing data and controlling other modules.

Functions:

- Receives data from the GPS module.
- Processes the data to extract relevant information (latitude, longitude, etc.).
- Communicates with other modules (e.g., GSM module) to execute further actions.

4.2.3 GSM Communication Module

Components:

- SIM800L GSM Module: A module for GSM communication that requires a 2G SIM card.

Functions:

- Sends the GPS data (latitude and longitude) to a remote server over GSM networks.
- Handles basic operations like SMS sending and internet connectivity.

4.2.4 Data Transmission and Storage Module

Components:

- XAMPP Server: A local server environment that includes Apache, MySQL, PHP, and Perl.
- Node-RED: A flow-based development tool for visual programming.
- MySQL Database: A database system for storing received GPS data.
- phpMyAdmin: A web-based interface for managing the MySQL database.

Functions:

- Receives GPS data from the Arduino via USB ports.
- Processes and routes data using Node-RED.
- Stores data in a MySQL database on the local server.
- Provides an interface for database management and data retrieval.

4.2.5 Software and Programming Module

Components:

- Arduino IDE: Used for programming the Arduino microcontroller.
- Node-RED Flow Editor: Used for setting up data flows and processing.
- PHP: For server-side scripting to handle data transfer and database interactions.
- SQL: For managing and querying the MySQL database.

Functions:

- Develops and uploads code to the Arduino to handle GPS data acquisition and communication with the GSM module.
- Configures Node-RED to receive and process data from the Arduino.
- Implements PHP scripts for data storage and retrieval.
- Uses SQL queries to manage the database and extract necessary information.

4.2.6 Data Visualization and Analysis Module

Components:

- Data Visualization Tools: Google Maps API will be used to convert the raw data and visualize the data into map data.

Functions:

- Retrieves GPS data from the MySQL database.
- Displays the data in user-friendly formats, such as plotting coordinates on a map.
- Analyzes the data to provide insights, such as tracking the movement of a vehicle over time.

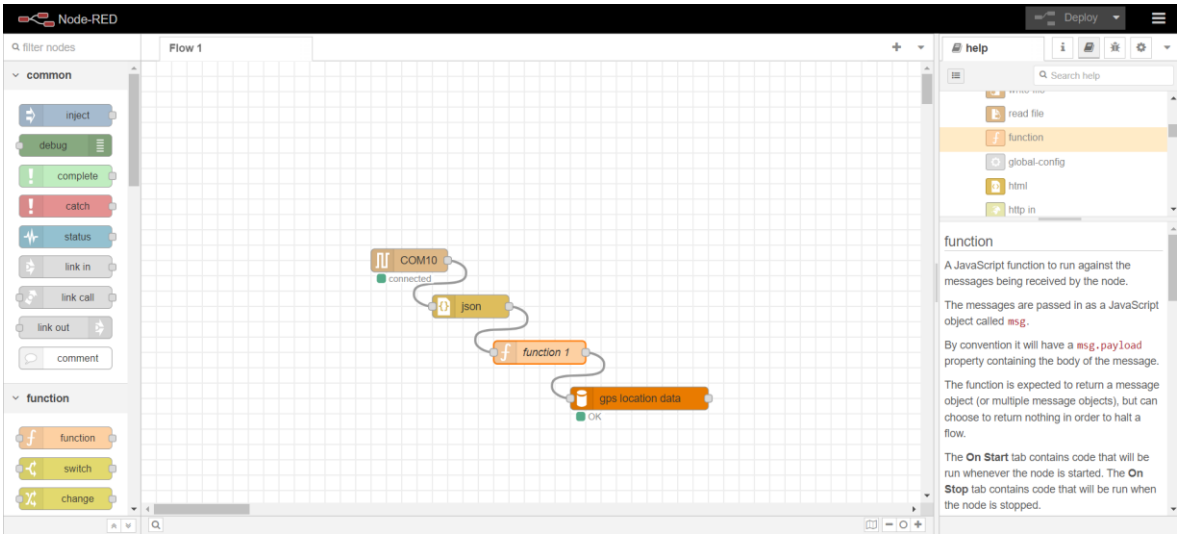


Fig 4.5 Node-Red Server

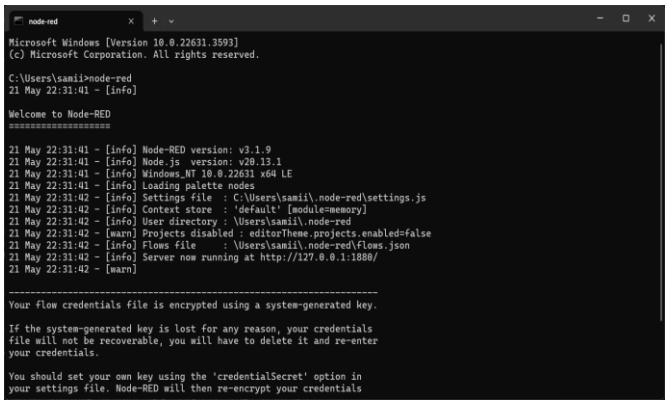


Fig 4.6 Node-Red Running on terminal

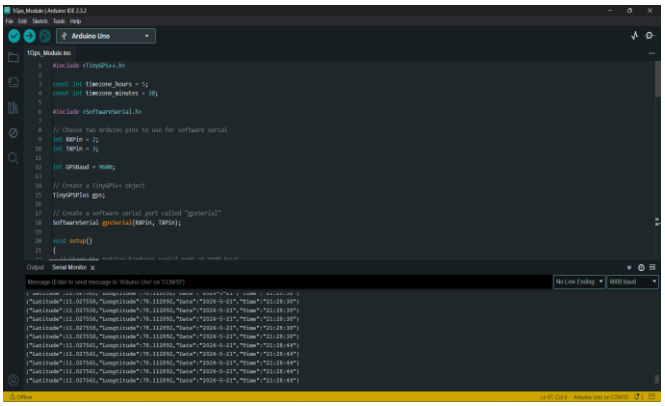


Fig 4.7 Arduino IDE

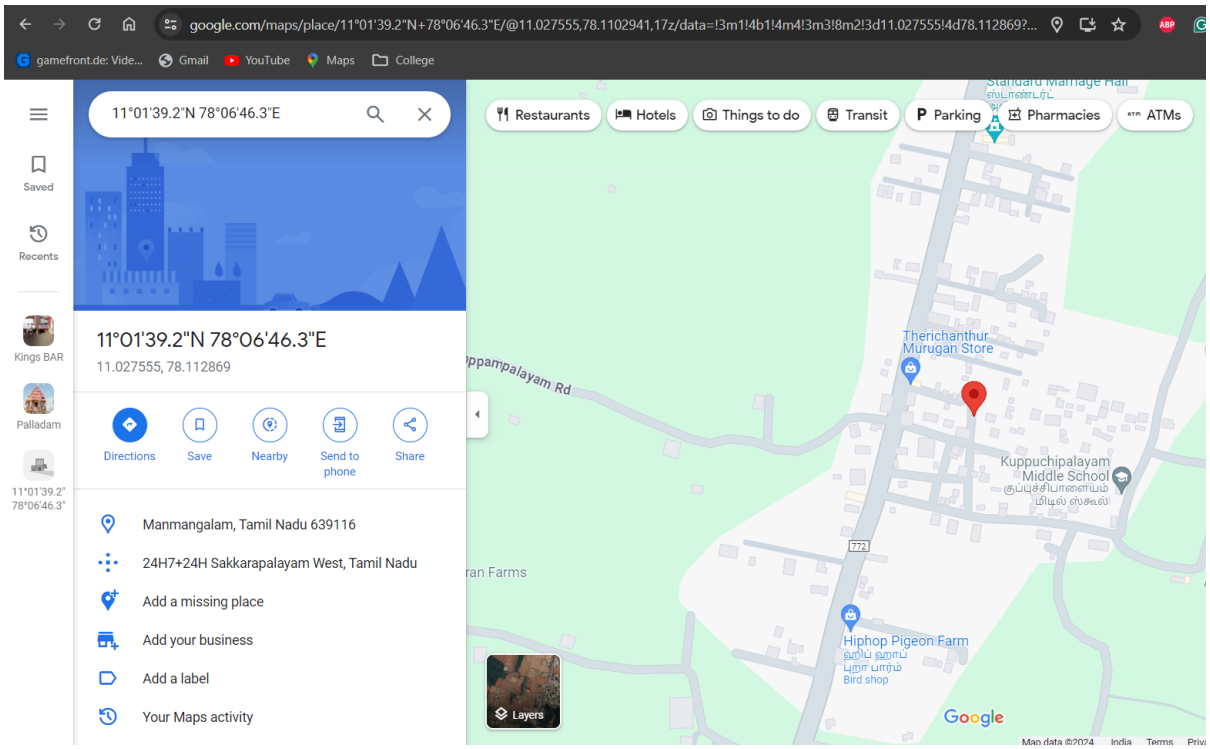


Fig 4.8 Google Maps API

4.3 COST ANALYSIS

TABLE 4.3 COST ANALYSIS

COMPONENT	QUANTITY	COST
ARDUINO UNO R3	1	700
NEO 6M GPS MODULE	1	450
SIM800L - GSM MODULE	1	390
18650 3.7V BATTERY	1	110
DC CONVERTER STEP-UP MODULE 5V	1	100
CONNECTING EQUIPMENT	-	250
	TOTAL	**2,000**

4.4 CODING

Gpsdata.ino

```
#include <NMEAGPS.h>  //parsing the comma seperated values to be more human readable
#include <stdlib.h>   //the lib is needed for the floatToString() helper function
#include <NeoSWSerial.h>  //it is used instead of SoftwareSerial

//if there are more devices, the server must differentiate them by the deviceID
int deviceId = 13;

//SoftwareSerial variables
static const int gpsRX = 3, gpsTX = 4;
static const uint32_t gpsBaud = 9600;
static const int simRX = 10, simTX = 11;
static const uint32_t simBaud = 9600;

//SoftwareSerial instances
NeoSWSerial gpsPort(gpsRX, gpsTX);
NeoSWSerial Sim800l(simRX, simTX);

//GPS instance
NMEAGPS gps;
gps_fix fix;

//for sending data to a remote webserver
String ipAddress = ""; //this is the ip address of our server - e.g. "123.123.123.123"
String APN = ""; //check your Internet provider's website for the APN e.g. "internet"

//helper variables for waitUntilResponse() function
String response = "";
static long maxResponseTime = 5000;
unsigned long lastTime;

//The frequency of http requests (seconds)
int refreshRate = 15; //SECONDS
```

```
  //variables for a well-scheduled loop - in which the sendData() gets called every 15 secs (refresh rate)
  unsigned long last;
  unsigned long current;
  unsigned long elapsed;

  //if there is an error in sendLocation() function after the GPRS Connection is setup - and the number of errors exceeds 3 - the system reboots. (with the help of the reboot pin)
  int maxNumberOfErrors = 3;
  boolean gprsConnectionSetup = false;
  boolean reboot = false;
  int errorsEncountered = 0; //number of errors encountered after gprsConnection is set up - if exceeds the max number of errors the system reboots
  int resetPin = 12;

  //if any error occurs with the gsm module or gps module, the corresponding LED will light up - until they don't get resolved
  int sim800Pin = 2; //error pin
  int gpsPin = 6; //error pin



  //a helper function which converts a float to a string with a given precision
  String floatToString(float x, byte precision = 2) {
    char tmp[50];
    dtostrf(x, 0, precision, tmp);
    return String(tmp);
  }


  void setup(){

    //setup resetPin
    digitalWrite(resetPin, HIGH);
    delay(200);
    pinMode(resetPin, OUTPUT);
```

```
    //init
    Serial.begin(9600);
    gpsPort.begin(gpsBaud);
    Sim800l.begin(simBaud);

    pinMode(sim800Pin, OUTPUT);
    pinMode(gpsPin, OUTPUT);

    Sim800l.write(27); //Clears buffer for safety
    Serial.println("Beginning...");
    delay(15000); //Waiting for Sim800L to get signal
    Serial.println("SIM800L should have booted up");

    Sim800l.listen(); //The GSM module and GPS module can't communicate with the arduino board at once
- so they need to get focus once we need them
    setupGPRSConnection(); //Enable the internet connection to the SIM card
    Serial.println("Connection set up");

    gpsPort.listen();

    last = millis();

  }

  void loop(){

    current = millis();
    elapsed += current - last;
    last = current;
    Serial.println(elapsed);
    while (gps.available(gpsPort)) {
      fix = gps.read();
     }
    if(elapsed >= (refreshRate * 1000)) {
     sendData();
     elapsed -= (refreshRate * 1000);
```

```
    }

    if ((gps.statistics.chars < 10)) {
      //no gps detected (maybe wiring)
      Serial.println("NO GPS DETECTED OR BEFORE FIRST HTTP REQUEST");
    }

    if(reboot){
     digitalWrite(resetPin, LOW);
    }

  }

  void sendData(){
   Serial.println("data to be sent");
    if (fix.valid.location) {
      digitalWrite(gpsPin, LOW);
      String lat = floatToString(fix.latitude(), 5);
      String lon = floatToString(fix.longitude(), 5);
      sendLocation(lat, lon);
    } else {
      sendLocation("-1", "-1");
      digitalWrite(gpsPin, HIGH);
    }
  }

  void setupGPRSConnection(){
   Sim800l.println("AT+SAPBR=3,1,\"Contype\",\"GPRS\""); //Connection type: GPRS
   waitUntilResponse("OK");
   Sim800l.println("AT+SAPBR=3,1,\"APN\",\"" + APN + "\""); //We need to set the APN which our internet provider gives us
   waitUntilResponse("OK");
   Sim800l.println("AT+SAPBR=1,1"); //Enable the GPRS
   waitUntilResponse("OK");
   Sim800l.println("AT+HTTPINIT"); //Enabling HTTP mode
   waitUntilResponse("OK");
```

```
 gprsConnectionSetup = true;
}

//ERROR handler - exits if error arises or a given time exceeds with no answer - or when everything is OK
void waitUntilResponse(String resp){
  lastTime = millis();
  response="";
  String totalResponse = "";
  while(response.indexOf(resp) < 0 && millis() - lastTime < maxResponseTime)
  {
    readResponse();
    totalResponse = totalResponse + response;
    Serial.println(response);
  }

  if(totalResponse.length() <= 0)
  {
    Serial.println("NO RESPONSE");
    digitalWrite(sim800Pin, HIGH);
    if (gprsConnectionSetup == true){
      Serial.println("error");
      errorsEncountered++;
    }
  }
  else if (response.indexOf(resp) < 0)
  {
    if (gprsConnectionSetup == true){
      Serial.println("error");
      errorsEncountered++;
    }
    Serial.println("UNEXPECTED RESPONSE");
    Serial.println(totalResponse);
    digitalWrite(sim800Pin, HIGH);
  }else{
    Serial.println("SUCCESSFUL");
    digitalWrite(sim800Pin, LOW);
```

```
      errorsEncountered = 0;
    }

    //if there are more errors or equal than previously set ==> reboot!
    if (errorsEncountered>= maxNumberOfErrors){
      reboot = true;
    }
  }

  //the function - which is responsible for sending data to the webserver
  void sendLocation(String lat, String lon){
   Sim800l.listen();
   //The line below sets the URL we want to connect to
   Sim800l.println("AT+HTTPPARA=\"URL\", \"http://" + ipAddress +  "/log_info.php?dev_id=13&lat="
+ lat + "&lon=" + lon + "\"");
   waitUntilResponse("OK");
   //GO
   Sim800l.println("AT+HTTPACTION=0");
   waitUntilResponse("200");
   Serial.println("Location sent");
   gpsPort.listen();
  }

  void readResponse(){
    response = "";
    while(response.length() <= 0 || !response.endsWith("\n"))
    {
      tryToRead();
      if(millis() - lastTime > maxResponseTime)
      {
        return;
      }
    }
  }

  void tryToRead(){
```

```
  while(Sim800l.available()){
    char c = Sim800l.read();  //gets one byte from serial buffer
    response += c; //makes the string readString
  }
}
```

Gps.json

```
var value=JSON.parse(JSON.stringify(msg.payload));
value=msg;
var lat = msg.payload.Latitude;
var lon = msg.payload.Longtitude;
var date = msg.payload.Date;
var time = msg.payload.Time;
msg.payload = [lat, lon, date, time];
msg.topic = 'INSERT INTO gps(Latitude, Longitude, Date, Time) VALUES (?,?,?,?);';
return msg;
```

CHAPTER– 5

RESULT AND DISCUSSION

In our project to create a Bus Tracking System using IoT (Internet of Things), we successfully developed a web application and a module that tracks the live location of a bus gives live feed to a database over a live server which is then pulled and a API is used to Visualize the coordinates as maps. This section of the report discusses the results we achieved and the key points of discussion.

Results:

5.1 User Authentication:

User authentication is a critical component of our Bus Tracking System. It enables users to create accounts, log in, and maintain their private information like location history, live location and personal information securely. We employed JWT (JSON Web Tokens) for token-based authentication. This approach enhances security and reduces the need for continuous database queries to check user credentials.

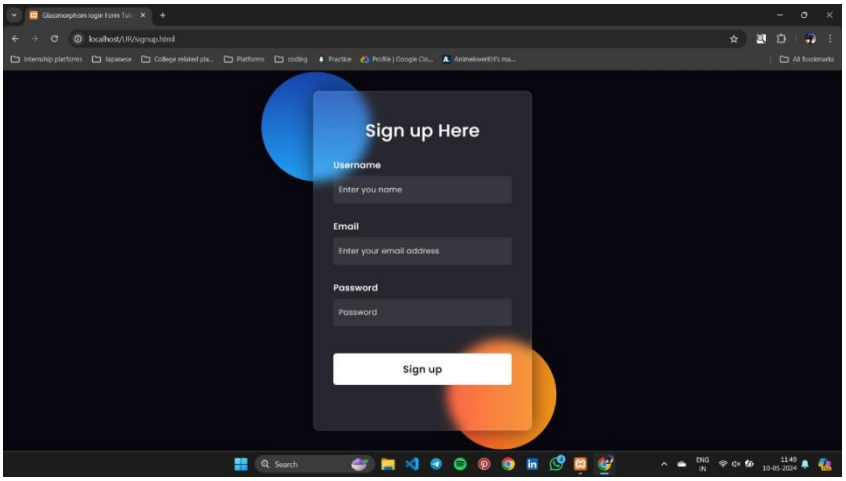


Fig 5.1 Sign-Up Page

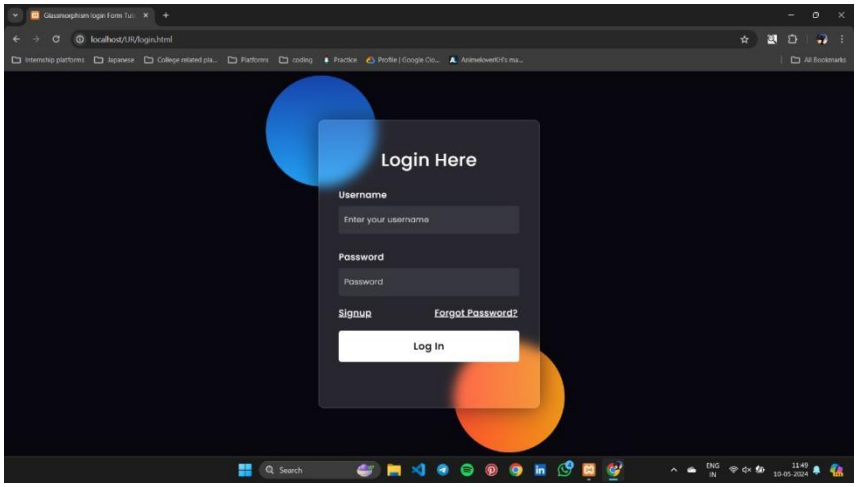


Fig 5.2 Login Page

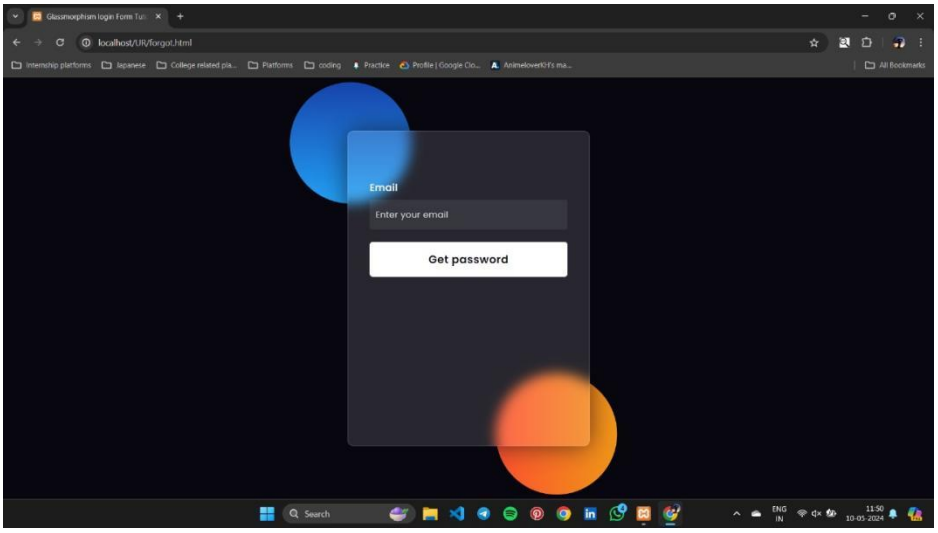


Fig 5.3 Forgot Password Page

5.2 Home Page:

The home page of our website is where the user enters the city or the location or the pincode he is currently in and it displays all the buses that are currently coming to the bus stop and the buses that operate through that station. This uses HTML,CSS and JavaScript for the creation of this webpage.

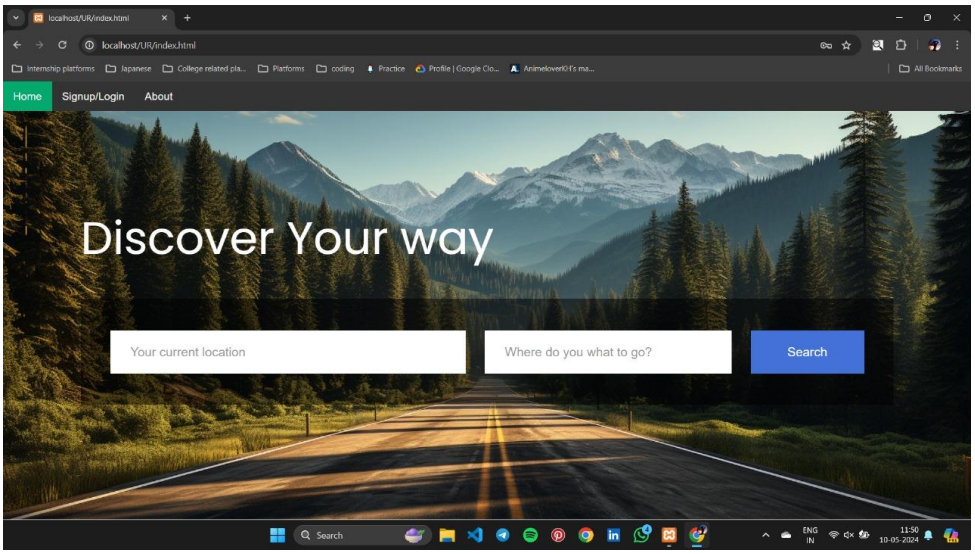
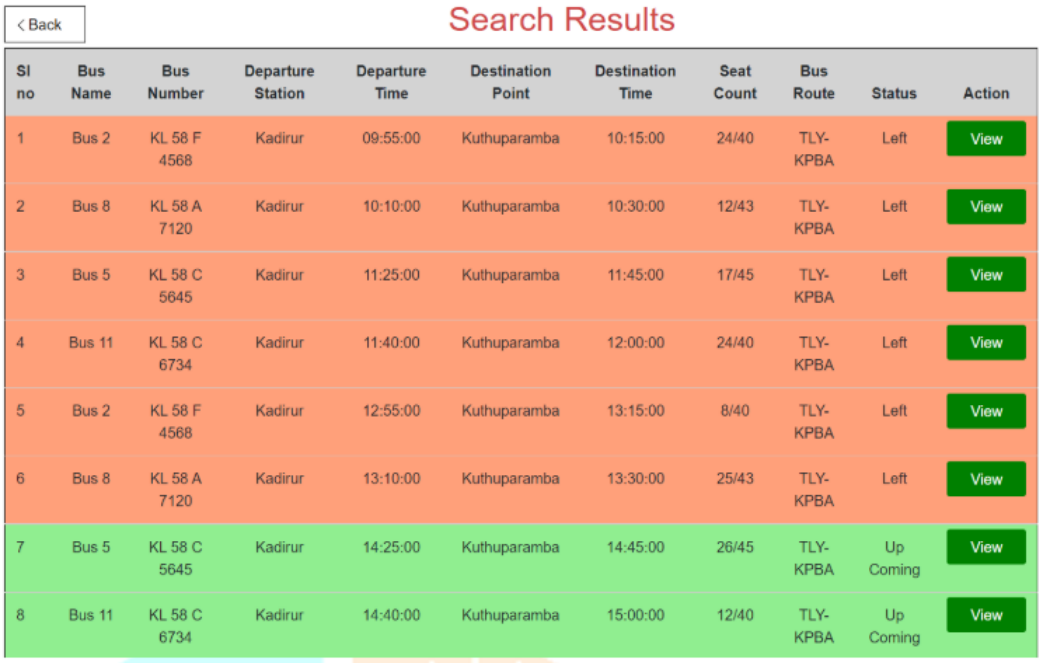


Fig 5.4 Home Page

5.3 Bus Search Results with Seat Vacancy:

The green columns shows that the bus hasn't arrived there yet. The red columns shows that the bus has left that particular station .Individual bus details can be acquired by pressing the 'View' button . Seat count is shown in

(Vacant seats / Total number of seats) manner.



Sl no	Bus Name	Bus Number	Departure Station	Departure Time	Destination Point	Destination Time	Seat Count	Bus Route	Status	Action
1	Bus 2	KL 58 F 4568	Kadirur	09:55:00	Kuthuparamba	10:15:00	24/40	TLY-KPBA	Left	View
2	Bus 8	KL 58 A 7120	Kadirur	10:10:00	Kuthuparamba	10:30:00	12/43	TLY-KPBA	Left	View
3	Bus 5	KL 58 C 5645	Kadirur	11:25:00	Kuthuparamba	11:45:00	17/45	TLY-KPBA	Left	View
4	Bus 11	KL 58 C 6734	Kadirur	11:40:00	Kuthuparamba	12:00:00	24/40	TLY-KPBA	Left	View
5	Bus 2	KL 58 F 4568	Kadirur	12:55:00	Kuthuparamba	13:15:00	8/40	TLY-KPBA	Left	View
6	Bus 8	KL 58 A 7120	Kadirur	13:10:00	Kuthuparamba	13:30:00	25/43	TLY-KPBA	Left	View
7	Bus 5	KL 58 C 5645	Kadirur	14:25:00	Kuthuparamba	14:45:00	26/45	TLY-KPBA	Up Coming	View
8	Bus 11	KL 58 C 6734	Kadirur	14:40:00	Kuthuparamba	15:00:00	12/40	TLY-KPBA	Up Coming	View

Fig 5.6 Search Results

5.4 Bus Timeline And Details / Live Status:

The name in the top of the timeline is the bus stop which the bus has passed most recently .The map in the right side clearly shows the current location of the bus using a red pointer.

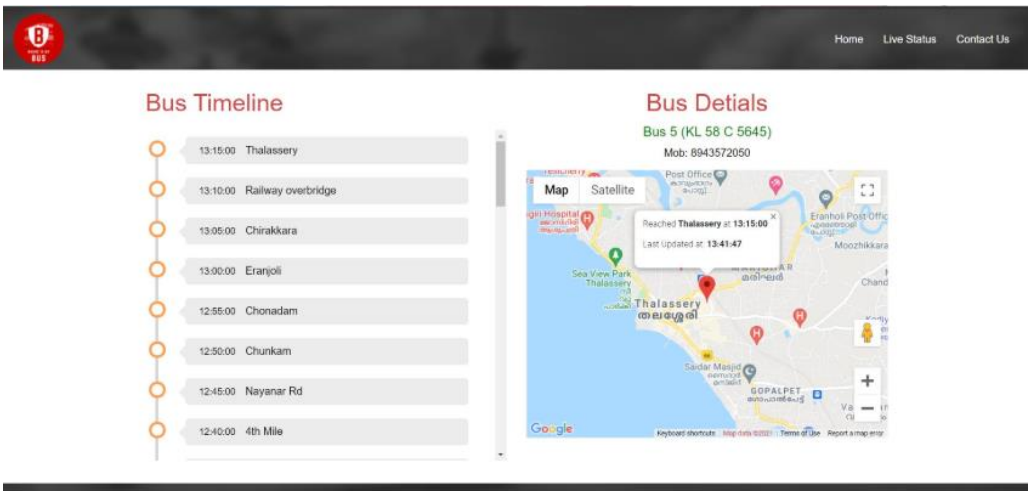


Fig 5.7 Bus Details

5.5 WebHost Server:

The Web Hosting Server module is a crucial part of the GPS-based data logging system. It facilitates the storage, management, and retrieval of data collected by the GPS module. This module is responsible for hosting the local server, managing the database, and providing access to the data through a web interface.We are using a free web hosting server called 000webhost where we host our local php files.

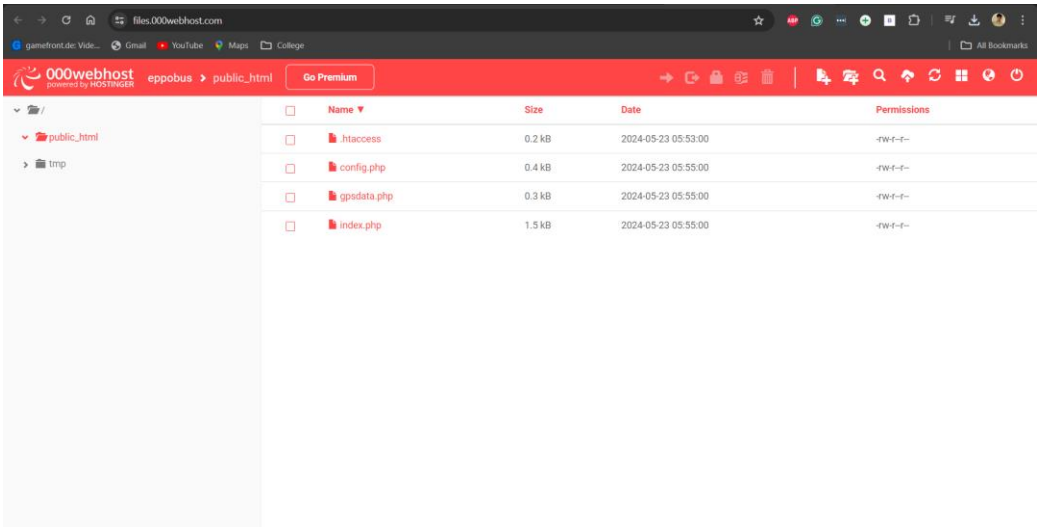


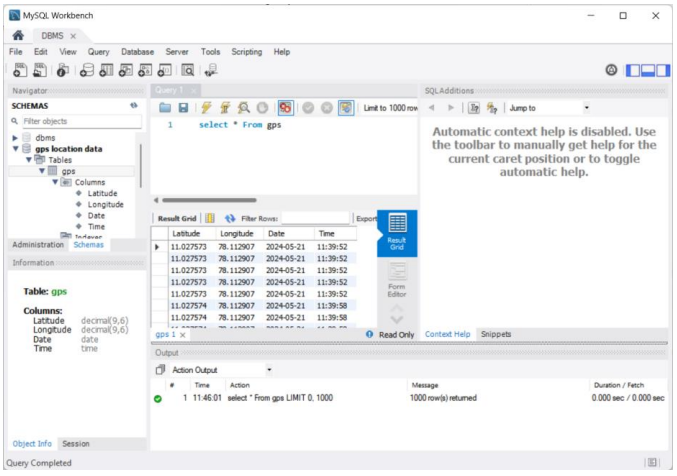
Fig 5.8 000Webhost Server

5.6 Database Connectivity and phpMyAdmin:

The Database and phpMyAdmin module integrates MySQL and phpMyAdmin to provide a robust and user-friendly platform for data storage and management. By using PHP scripts for data insertion and retrieval, this module ensures that GPS data is accurately stored and easily accessible for further analysis and visualization, forming a critical component of the overall GPS-based data logging system.

5.7 MySQL Database:

A relational database management system (RDBMS) used to store GPS data and other relevant information.



5.7.1 phpMyAdmin:

A free and open-source administration tool for MySQL and MariaDB, written in PHP. It provides a graphical interface to manage databases.

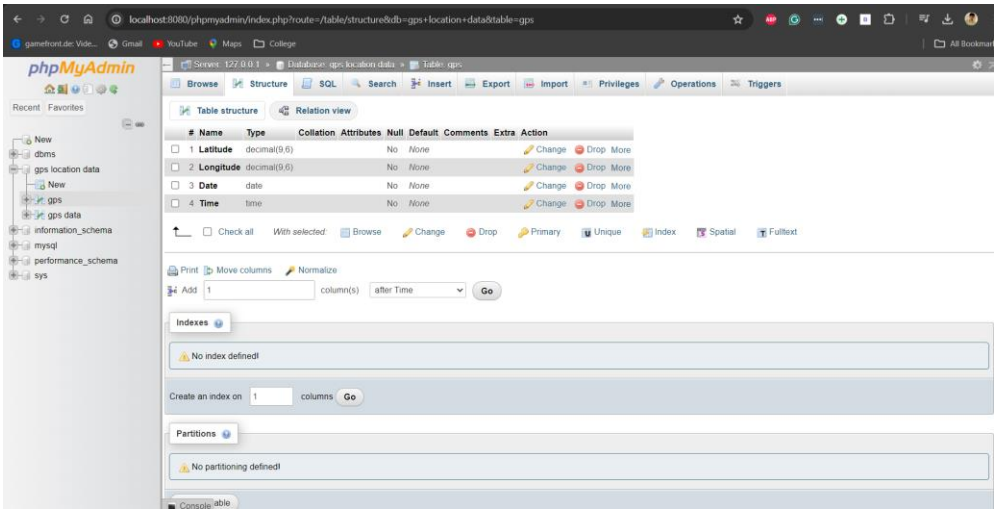


Fig 5.10 phpMyAdmin

5.7.2 XAMPP Server:

The XAMPP Setup and Configuration module involves installing and configuring XAMPP to create a local web server environment that supports Apache, MySQL, PHP, and Perl. This module is essential for hosting the web server, managing the database, and running PHP scripts for the GPS-based data logging system.

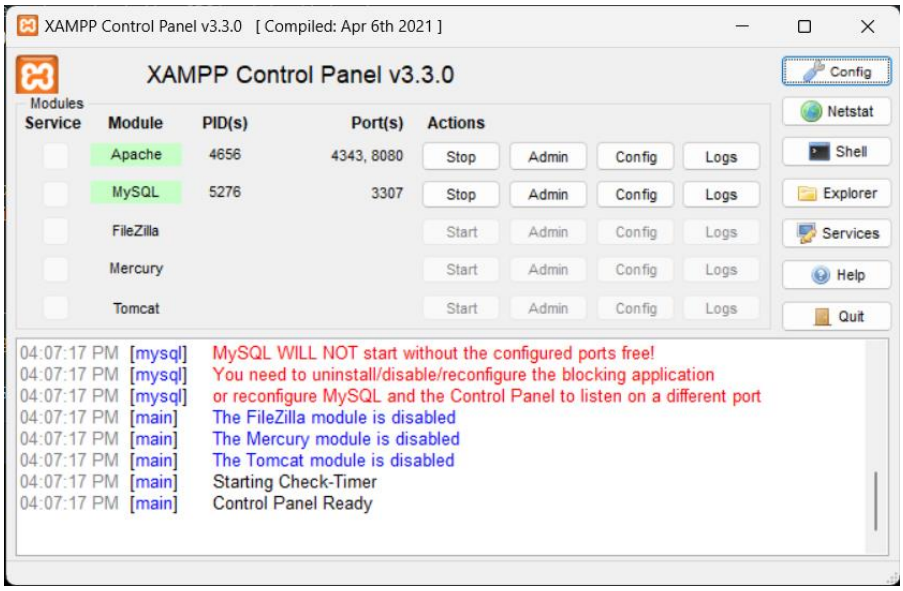


Fig 5.11 XAMPP Server