

Vehicle Monitoring and Tracking System

(V.M.T.S)



SYNDICATE MEMBERS

Abdul Rehman

Ameer Hamza Hassan

Talha Mazhar

Tehreem Mateen Zia

SUPERVISOR

Lt Col. Dr. Hasnat Khurshid

Submitted to the faculty of Department of Electrical Engineering,
Military College of Signals, National University of Sciences and Technology,
in partial fulfillment for the requirements of B.E Degree in Electrical Engineering.

(June, 2021)

CERTIFICATION OF CORRECTION & APPROVAL

Certified that work contained in this thesis titled “**Vehicle Monitoring and Tracking System**”, carried out by “NC Abdul Rehman, NC Ameer Hamza Hassan, NC Talha Mazhar and NC Tehreem Mateen Zia” under the supervision of “Lt Col Dr. Hasnat Khurshid” partial fulfillment of Degree of Bachelors of Electrical Engineering, in Military College of Signals, National University of Sciences and Technology, Islamabad during the academic year 2020-2021 is correct and approved. The material that has been used from other sources has been properly acknowledged/ referred.

Approved by Supervisor

Lt Col. Dr. Hasnat Khurshid

Department of EE, MCS

Dated: _____

DECLARATION

No portion of work presented in this thesis has been submitted in support of another award or qualification in either this institute or anywhere else.

Plagiarism Certificate (Turnitin Report)

This thesis has been checked for Plagiarism. Turnitin report endorsed by Supervisor is attached.

Signature of Student

Abdul Rehman

Reg# 00000218159

Signature of Student

Ameer Hamza Hassan

Reg# 00000218240

Signature of Student

Talha Mazhar

Reg# 00000209656

Signature of Student

Tehreem Mateen Zia

Reg# 00000211714

Signature of Supervisor

Acknowledgements

We are thankful to our Creator Allah Subhana - Watala to have guided us throughout this work at every step and for every new thought which You setup in our mind to improve it. Indeed, we could have done nothing without Your priceless help and guidance. Whosoever help us throughout the course of our thesis, whether our parents or any other individual was Your will, so indeed none be worthy of praise but You.

We are profusely thankful to my beloved parents who raised us when we were not capable of walking and continued to support us throughout in every department of our life.

We would also like to express special thanks to our supervisor Dr. Hasnat Khurshid for his unconditional support and guidance. His assistance and valuable suggestions enable us to meet our objectives in this endeavor. We are also very grateful for Digital Image Processing course which he has taught us. We can surely say that we haven't learnt any other engineering subject comprehensively than the one which he has taught.

Finally, we would also like to express our gratitude to NC Ameer Hamza Hassan who offered his car for experiment and many others who have rendered valuable assistance to our study.

*Dedicated to our exceptional parents and adored siblings whose
tremendous support and cooperation led us to this wonderful
accomplishment.*

Abstract

Since the start of twentieth century, the role of automobile has become vital. Almost everyone who drives a car faces trouble regarding its performance, maintenance and most importantly, its tracking. One would like to maintain and solve the main issues of his/her car without having the need to go to a mechanic. The proposed system provides a GSM based vehicle tracking and monitoring system. This system renders service about vehicle's internal performance and tracking via short message service (SMS). In VMTS composition, we have utilized the car's "On-Board Diagnostics (OBD-II)" port to extract input data of various features of the car and translate this raw data by means of an IC- chip named ELM-327 into human readable form. The GPS module provides the tracking of car, whereas the GSM module sends the car's location and important monitoring information to user's mobile device via SMS. A message will pop-up on mobile device to notify its user about the car's location and other relevant monitoring data to the mobile device.

Keywords: OBD-II, ELM-327, GPS, GSM, AT Commands

Table of Content

	Pag
Certificate of Correction and Approval.....	ii
Declaration.....	iii
Plagiarism Certificate (Turnitin Report).....	iv
Acknowledgements.....	v
Dedication.....	vi
Abstract.....	vii
Table of Contents.....	viii
List of Figures.....	ix
List of Tables.....	x
Chapter1: Introduction.....	1
1,1 Overview of VMTS... ..	2
1.2 Problem Statement.	2
1.3 Research Methodology.....	2
1.4 Project Description.....	3
1.5 Scope and Objective.....	3
1.6 Deliverables... ..	4
1.7 Organization of Document.....	4
Chapter 2: Literature Review.....	5
2.1 Background... ..	6
2.2 Research and Literature Review.....	6
Chapter 3: Design Requirement.....	8

3.1 Project Hardware...	9
3.1.1 Car's OBD II Port Pin Configuration and Protocols...	9
3.1.2 ELM-327...	11
3.1.3 Bluetooth Module...	12
3.1.3.1 Why Bluetooth Module?...	12
3.1.3.2 HC-05...	12
3.1.4 GSM SIM900A...	13
3.1.5 Arduino Module...	13
3.1.6 GPS Tracker	14
3.2 Project Software	15
3.2.1 Arduino IDE.....	15
3.2.2 Android Emulator.....	15
3.2.3 Visual Studio	16
3.2.4 Flutter (Front End).....	16
3.2.5 Firebase Server (Back End).....	17
Chapter 4: Working.....	18
4.1 Project Hardware Working.....	19
4.1.1 How ECU communicates with car's OBD Port.....	19
4.1.1.1 OBD-II Parameter IDs.....	19
4.1.2 ELM Data Extraction.....	20
4.1.3 Arduino UNO.....	21
4.1.4 Bluetooth Module Configuration with Arduino.....	21
4.1.4.1 Handling PIDs.....	23
4.1.4.2 Code for Engine Coolant Temperature.....	23

4.1.4.3	Code for Battery Voltage.....	26
4.1.4.4	Code for Engine Load.....	26
4.1.4.5	Code for Air Intake Temperature.....	26
4.1.4.6	Code for Throttle Position.....	26
4.1.4.7	Code for Vehicle Speed.....	27
4.1.4.8	Code for Engine Revolution per minute (RPM).....	27
4.1.5	GPS Tracker	28
4.1.6	Send Data over GSM.....	28
4.2	Project Software Working.....	30
4.2.1	Send Data to Mobile Device through GSM.....	30
4.2.2	Front End Coding for Monitoring and tracking.....	32
4.2.3	Firebase Authentication	45
Chapter 5: Results and Analysis.....		46
Chapter 6: Conclusion and Next Steps		52
6.1	: Conclusion.....	53
6.2	: Future Work.....	53
 APPENDIX A		
Abbreviation		
 APPENDIX B		
Synopsis		
 APPENDIX C		
Demonstration Outline		
 APPENDIX D		
AT Commands		
 REFERENCES		

List of Figures:

Figure 1: OBD-II Port Pin Configuration 10

Figure 2: ELM-327 Pin Configuration..... 11

Figure 3: Bluetooth Module HC – 05..... 12

Figure 4: GSM Module SIM900A..... 13

Figure 5: Arduino UNO 13

Figure 6: GPS Tracker 14

Figure 7: Arduino IDE 15

Figure 8: Android Emulator 15

Figure 9: Visual Studio 16

Figure 10: Flutter 16

Figure 11: Firebase 17

Figure 12: ELM Data Extraction Flow Chart..... 21

Figure 13: Data Send over GSM 28

Figure 14: Block Diagram 29

Figure 15: Hardware Interface Unit..... 29

Figure 16: Application Sign in Page 40

Figure 17: Application Registration Page 40

Figure 18: Application Home Page..... 44

Figure 19: Firebase Authentication Page..... 45

Figure 20: Engine Coolant Temperature Results..... 47

Figure 21: Engine Load Results 48

Figure 22: Throttle Position Results..... 48

Figure 23: Engine RPM Results..... 49

Figure 24: Battery Voltage Results	49
Figure 25: Air Intake Temperature Results.....	50
Figure 26: Message sending command.....	50
Figure 27: Message Viewed on Mobile.....	51
Figure 28: Tracking Device.....	51
Figure 29: Location of Vehicle	51

List of Tables:

Table 1: SAE J1979 Protocols	10
Table 2: OBD-II Parameter IDs	19

CHAPTER 1: INTRODUCTION

Chapter 1: Introduction

1.1 Overview of VMTS

VMTS is an abbreviation for “Vehicle Monitoring and Tracking System”. It provides real time information about car’s performance and tracking via short message service (SMS). This system is capable of handling an amount of data received through the On-Board Diagnostics (OBD-II) port for diagnostic purposes.

1.2 Problem Statement

Car Diagnostic Sensors are being used worldwide. They are used to monitor speed, temperature, vehicle coolant system, faulty radiator etc. but do not provide tracking and monitoring at the same time. They are expensive. Mostly people do not even know about potential slot OBD port in their cars and end up paying a fee to the mechanic to check the vulnerabilities in his/her car.

1.3 Research Methodology

In developing hardware system and testing web application we have attained knowledge about certain terms. For this effort, a variety of databases and sources of information were examined. Below are the methods that were used to gather information.

Databases:

1. Research Papers from Google Scholar.
2. Google search engine.
3. University Library to find appropriate books and papers.

Search Terms Used:

1. OBD Protocols
2. OBD Applications
3. ELM-327 Pin Description

4. GSM module Connectivity
5. Flutter SDK
6. Dart Programming Language (VS Code)
7. Firebase Server

From these databases and searches most relevant data has been gathered. All non-relevant literature and information is disregarded.

1.4 Project Description

The VMTS project consists of two parts i-e hardware build system and software application. The user can purchase the hardware kit at a very low cost and download the mobile application from google play store. Hardware kit contains ELM-327 that will connect directly with OBD-II port which in turn is integrated to the Arduino Module, GPS tracker and GSM module.

Software implementation will utilize Visual Studio Code as the editor for flutter SDK with Dart as the programming language with firebase as the backend server. Flutter is used to make the front-end of the application and Firebase for the server side. The user will need to insert a SIM card for the GSM module that will communicate with mobile device through SMS.

1.5 Scope and Objectives

The project's scope is to design the low-cost system which give offline tracking and online monitoring.

This technology can be used:

- If some valuable products need to be transported then continuous monitoring and tracking is required.
- By rent -a-car service for optimum working of the car.
- Parental control can be set if car's speed goes beyond a certain speed or any mishap occur.

- By a layman with little knowledge of car to perform scans & identify them by his own.

The major goal of this initiative is to

- To correctly implement hardware that will communicate car's data to user mobile device.
- Correct development of application that would be able to retrieve vehicle monitoring parameter.
- Allow user to be able to obtain provided information about their car they want.

1.6 Deliverables

- ✓ Hardware Device
- ✓ Web Application

1.6 Organization of Document

This document is divided in to following main parts:

1. The first part includes the introduction, problem statement, scope and objective of the project.
2. The second part explains the summary of the literature review and background.
3. The third part is related to the hardware and software description of the project
4. The fourth part includes detailed hardware design, software implementation and working of the project.
5. The fifth part shows the experimental analysis and observation.
6. The sixth part shows the conclusion and future modification.
7. The seventh part gives the timeline of project implementation, Appendices which is abbreviations used in the thesis, AT commands summary and synopsis of this project.

CHAPTER 2: LITERATURE REVIEW

Chapter 2: Literature Review

2.1 Background

Almost in all cars, Engine Control Unit commonly known as ECU is designed to control vehicle engine by getting values from various sensors present in a car. If there is any fault in these sensors, the check engine light also known as Malfunction Indication Lamp turns on, indicating the fault. The faults can be removed by connecting the OBD scanner and once the fault code is cleared the check engine light itself turns off.

OBD is onboard diagnostics – a process to scan the vehicle with an onboard computer. OBD scanner is a device that enable us to read and find error codes of car. There are two versions of OBD i-e OBD-1 and OBD-2.

Before OBD-I, each car manufacturer had its own set of standards, and mechanics had to purchase expensive scan gear from those manufacturers. In 1987, OBD-I was introduced, marking the beginning of standardization of OBD. It had sensors that retrieved diagnostic information and alerted the driver of engine problems. However, it had many problems and fell short in terms of accuracy and supported only car manufacture before **1995**.

As a result, in **1996** OBD-II came into existence after certain variations. It can be connected hands freely via Bluetooth or Wi-fi. It is Universal and one scanner can support different manufacturers. It provides data with high accuracy.

2.2 Literature Review

There are so many proposed research papers available on the OBD system, several applications have been built for use on mobile phones using Android platform.

We have focused particularly on OBD scanner and application developed for monitoring car's parameter like speed and live location of car. The following research papers were consulted to get guidance related vehicle monitoring and tracking are summarized below:

1. Android-Based Universal Vehicle Diagnostic and Tracking System

This paper has been proposed by Ashraf tahat in year 2012. In this paper, they used a cheap hardware unit and an Android application to monitor an automobile using an OBD system of vehicle and a GPS tracker to pinpoint its whereabouts. They used Bluetooth module to acquire desire data from ECU of the vehicle.

2. Vehicle Monitoring Controlling and Tracking System by Using Android Application.

This paper has been proposed by Rajeevan and Payagala in the year 2016. In this paper they have discussed monitoring and controlling of vehicle functions like vehicle doors, parking lights, side mirrors and location by using sensor actuator module, communication module and android application.

CHAPTER 3: DESIGN REQUIREMENT

Chapter 3: Design Requirement

3.1: Project Hardware

The project utilizes the following hardware components:

3.1.1 Car's OBD-II Port Pin Configuration and Protocols:

The OBD-II port is a 16-pin connector either beneath the steering wheel or beneath the dashboard. With the use of a particular scan tool, a driver or any person can read error codes.

OBD-II port has standardized pin configuration listed below.

Pin 1: For Makers

Pin 2: For SAE J1850 PWM and VPW

Pin 3: For Makers

Pin 4: Earth

Pin 5: Earth

Pin 6: For ISO 15765-4 CAN

Pin 7: The K-Line of ISO 9141-2 and ISO 14230-4

Pin 10: For SAE J1850 PWM

Pin 14: For ISO 15765-4 CAN

Pin 15: The K-Line of ISO (141-2 and ISO 14230-4

Pin 16: Power from car battery

The following figure shows the OBD2 port pin layout.

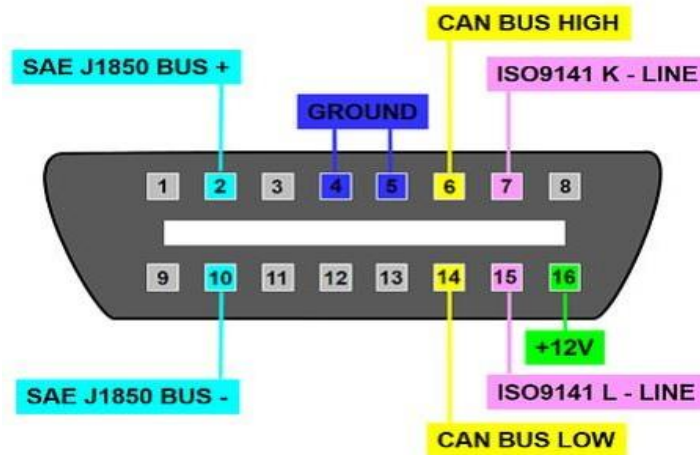


Figure 1: OBD-II Port Pin Configuration

We created this system to read specific parameters from a car with an OBD-II port that supports the SAE J1979 (11 bits ID) signaling protocol and display them on a mobile app.

SAE J1979 is a standard that specifies how to ask an ECU for various performance parameters. SAE J1850 Pulse Width Modulation PWM, SAE J1850 Variable Pulse Width VPW, ISO 9141-2, ISO 14230-4 Keyword Protocol 2000, and ISO 15765 CAN 250/500kbaud 11/29bit ID are the five OBD-II signaling methods defined by this standard protocol.

Table 1: SAE J1979 Protocols

Protocols	Data Transfer Rate	Vehicle Type
SAE J1850 PWM	41.6 kB/sec	Ford
SAE J1850 VPM	10.4 kB/sec	GM Vehicle
ISO 9141-2	10.4 kbit/sec	Asian and European vehicles (2000 – 2004)
ISO 14230-4 kWP	10.4 kbit/sec	2003+ vehicle
ISO 15765	CAN500/250 kbit/sec	Since 2008

3.1.2 ELM-327

The ELM-327 OBD-II interface is a car diagnostic tool that sends car's internal data from an OBD-II compliant vehicle to laptops, PC's, Android phones, tablets, iPhones, and iPads. It converts data from the car's On-Board Diagnostic port into a human-readable format. This technology allows you to read real-time data from the ECU (vehicle computer) and display it in a clear and concise manner. It has following features:

- Stand-by mode with power control
- RS232 connection
- Looks for Protocols on its own.
- Set up with AT commands
- OBD-II port for power

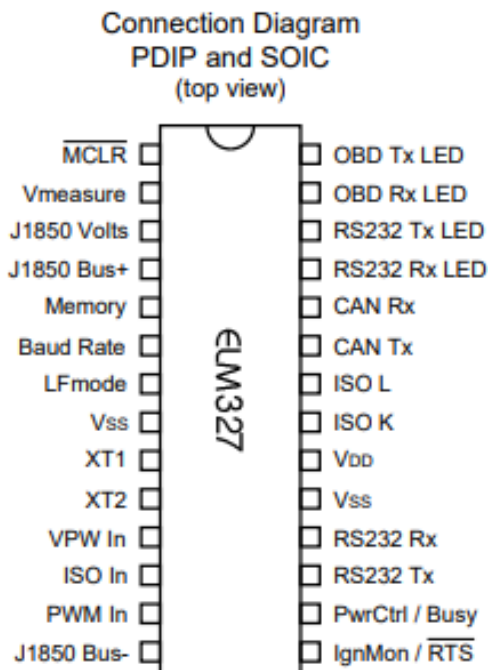


Figure 2: ELM-327 Pin Configuration

3.1.3 Bluetooth Module

Bluetooth is a short-range wireless technology that can be used to communicate between two microcontrollers, such as the Arduino, or with any other Bluetooth device, such as a phone or laptop. It communicates with the microcontroller via a serial port (USART). It operates on the 2.45 GHz frequency band. There is a point-to-point or multi-point connection with a maximum range of 10 meters. It has a 1MBps data transfer rate.

3.1.3.1 Why Bluetooth Module?

Now the question is, why do we need a Bluetooth module? Well, this is because some cars can have their OBD port on the dashboard which is not very secure location to place tracking device. Someone can easily remove the device. For that what we have done is, we have split the device into two parts; one for tracking and the other for monitoring in such a way that both communicate through Bluetooth. Monitoring device is plugged in OBD port however, the Tracking device can be placed in any hidden place such as doors, under the seats. By doing so, whenever someone removes the Monitoring device plugged in OBD port, the tracker still works.

3.1.3.2 HC-05

The HC-05 Bluetooth module is a tiny wireless communication module. It uses the serial port protocol to communicate with the microcontroller. The data baud rate is 9600 bits per second, while the command mode baud rate is 38400 bits per second. You may switch between master and slave mode with it. AT Commands are supported, and Tx (transmitter) and Rx (receiver) pinouts are used to control them.

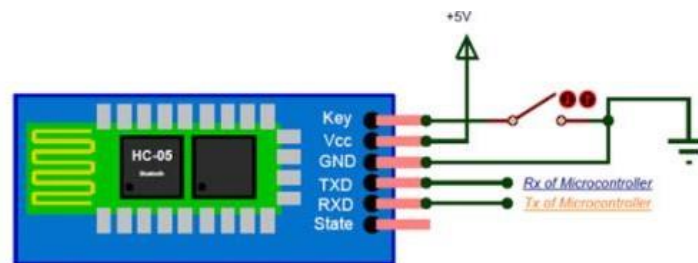


Figure 3: GSM Module SIM900A

3.1.4 GSM SIM900A

It is a dual-band GSM/GPRS engine that operates on EGSM 900MHz and DCS 1800 MHz frequencies and can automatically search both bands. The AT commands can be used to change the frequency. It requires a single supply voltage of 3.4 V to 4.5 V. In SLEEP mode, the current consumption is as low as 1.5 mA, thanks to a power-saving feature.

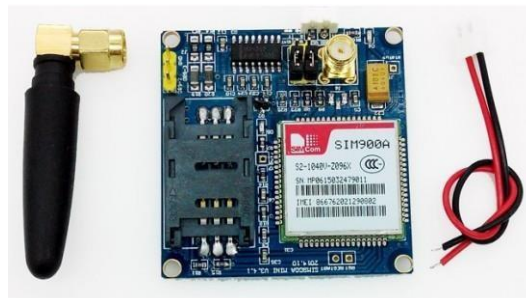


Figure 4: GSM Module

3.1.5 Arduino Module

Arduino is a cross-platform (Windows, macOS, Linux) application based on simple hardware and software. ATmega328P microcontroller is used in Arduino UNO. In comparison to Arduino Mega Board, it is simple to use. There are six analogue pin inputs, fourteen digital pins, a USB connector, a power jack, and an ICSP header on this board.



Figure 5: Arduino UNO

3.1.6 GPS Tracker

A GPS tracking unit uses the Global Positioning System to determine the movement or geographic location of vehicle, assets, or person. When the GPS signal is received, the GPS module calculates the coordinates. It has much memory for data loggers to save coordinates. Data pusher includes a GSM or GPRS modem for sending data to a computer through SMS or GPRS in form of IP Packets.



Figure 6: GPS Tracker

3.2 Project Software

3.2.1 Arduino IDE:

Arduino Integrated Development Environment sometimes known as the Arduino Software, allows users to build and upload programs. It typically supports the C and C++ programming languages. It contains a code editor, message area, text console, toolbar with buttons for common functions, and menu system. The serial monitor can also be used to display data loops that are continuously monitored.



Figure 7: Arduino IDE

3.2.2 Android Emulator:

An Android emulator is a computer-based virtual mobile device that mimics the functionality of an Android handset running the Android operating system. It enables the development and testing of programs without the use of many physical devices of varying configurations. The emulator mimics nearly all of the features of a real Android handset. Incoming phone calls and text messages may be stimulated, device position can be specified, different network speeds can be stimulated, Google Play can be accessed, and much more.



Figure 8: Android Studio

3.2.3 Visual Studio

Visual Studio is a Microsoft integrated development environment for creating computer programs, websites, web applications, web services, and mobile applications. It supports different programming languages and enables the code editor and debugger to work with practically any programming language as long as a language-specific service is available.

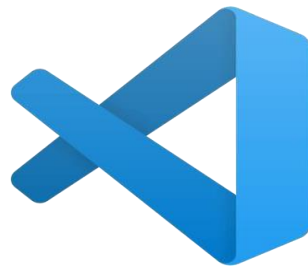


Figure 9: Visual Studio

3.2.4 Flutter (Front End)

Flutter is an opensource SDK created by Google. It is designed to support mobile application that run on both Android and iOS, as well as interactive applications that runs on web pages or on the Desktop. It uses a variety of widgets to carry out a fully functioning application. Flutter uses Dart Programming Language.



Figure 10: Flutter

Fire Base:

Backend-as-a-Service (BaaS) app development platform Firebase provides hosted backend services such as a real-time database, cloud storage, authentication, crash reporting, machine learning, remote setup, and static file hosting. Flutter is supported by Firebase.



Figure 11: Firebase

CHAPTER 4: WORKING

Chapter 4: Working

4.1 Project Hardware working

4.1.1 How ECU communicates with car's OBD Port

We can communicate with cars ECU using standard OBD-II PIDs. This can be done by configuring ELM with Arduino. For example, 0105 SAE J1979 Standard protocol to get values of Engine Coolant Temperature.

4.1.1.1 OBD-II Parameter IDs

On board Diagnostic II parameter IDs are codes that used as a diagnostic tool and request for car's input data from the ECU of car. **SAE standard J1979** is a protocol that defines many OBD-II PIDs.

The most recent OBD-II standard, SAE J1979., lists ten diagnostic service. J1979 used to refer services as 'modes' before 2002. They are as follows

Table 2: OBD-II Parameter IDs:

Modes PIDs (hex)	Description
01	Used to identify what powertrain data, available to the scan tool.
02	Shows freeze frame data.
03	The “confirmed” emission-related diagnostic fault codes are displayed. The defects are identified by accurate numeric, 4-digit numbers displayed on the screen.
04	Delete stored diagnostic problem codes and values
05	The oxygen sensor monitor screen is displayed, as well as the test data acquired concerning the oxygen sensor.
06	This command is used to request on board monitoring test services for both continuously and intermittently monitored systems..
07	Requests emission-related diagnostic problem codes that have been observed during the current or previous driving cycles.

08	It could allow an off board test equipment to regulate the working of a system, test, or component on-board.
09	Get information about the vehicle
0A	It displays a list of stored emission-related “permanent” diagnostic issue codes.

4.1.2 ELM Data Extraction

The ELM-327 IC may communicate with Bluetooth module. The results will show on PC like >88 degrees C.

The character '>' indicates that the device is idle and waiting for the character. Simply type >AT 'space' Z followed by the return key:

```
>AT Z
```

The IC will be reset. If you obtain an unusual reading, it's possible that the baud rate is incorrect. By monitoring the message content, ELM-327 may simply find where the received characters should be directed. Commands for ELM-327's internal usage must start with characters 'AT,' while commands for vehicle can only contain ASCII codes for hexadecimal digits (0 to 9 and A to F).

All communications to the ELM-327 should be terminated with a carriage return characters hex '0D' before being acted upon, whether it be a 'AT' type internal command or a hex string for OBD bus. ELM-327 will signal a question mark (?) if the message is partial or unclear.

It's important to remember that ELM-327 is a protocol interpreter that makes no changes to validate the OBD messages that you transmit. It just makes sure that hexadecimal digits are received, converted to bytes, and then transmitted out to the OBD port. It has no way of knowing whether a message transmitted to the car was sent with error.

ELM-327 is not case-sensitive and it also ignores space characters (tab, etc.). When a single carriage return character is received, it can repeat any command (AT or OBD). If you've given a command (e.g 01 0C to get RPM), you don't have to resend the complete command to repeat the request to the vehicle — only send a carriage return character, ELM will then resend that command on your behalf.

Because the memory can only store one command, there is no providing in the existing

ELM-327 for additional storage.

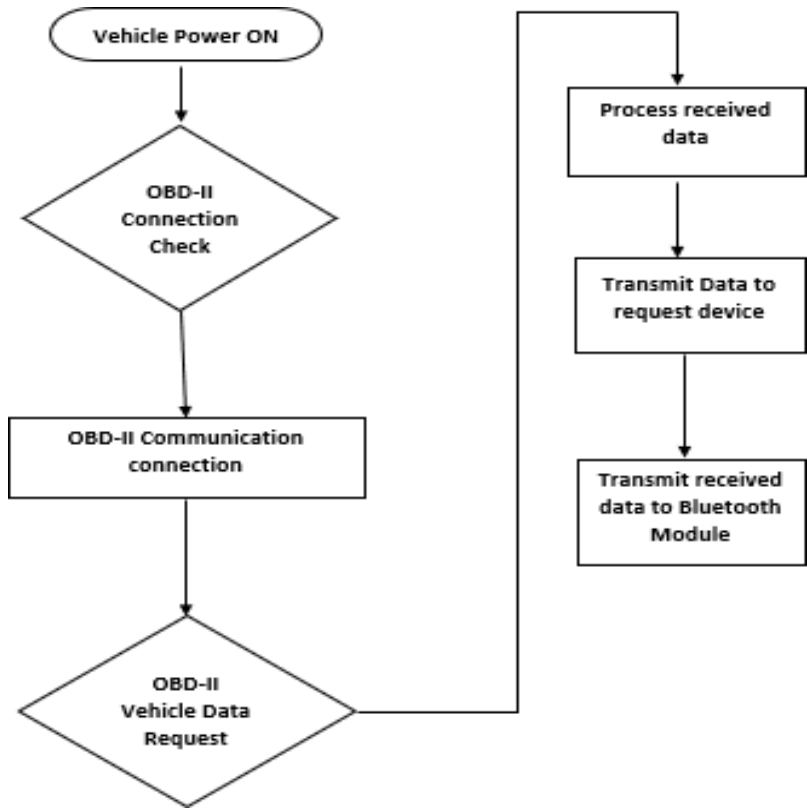


Figure 12: ELM Data Extraction Flow Chart

4.1.3 Arduino UNO

Arduino is used as a main microcontroller in this project. Its configuration with Bluetooth module, GPS module and GSM makes it a main part of the project.

4.1.4 Bluetooth Module Configuration with Arduino

First Bluetooth module is configured by Arduino in Master mode in order to receive the data extracted through ELM. For Configuration we uploaded following code on the Arduino UNO board.

```
#include <SoftwareSerial.h>

SoftwareSerial Bluetooth(2,3);

char c=' ';

void setup() {

  Serial.begin(9600);

  Serial.println("ready");

  Bluetooth.begin(38400);

}

void loop() {

if(Bluetooth.available()){

  c=Bluetooth.read();

  Serial.write(c);

}

if(Serial.available()){

  c=Serial.read();

  Bluetooth.write(c);

}

}
```

For configuration, following commands were used.

- AT+Reset
- AT+ORGL (Set to original)
- AT+ROLE=1 (Set to Master)

- AT+CMODE=0 (Set a specific address to connect to.)
- AT+BIND=AABB,CC,112233
- AT+PAIR=AABB,CC,112233,20 (,20 means 20 second timeout)
Hence the data gets visible on serial monitor of Arduino.

4.1.4.1 Handling PIDs

After setting up the Bluetooth module, next step is to access the required data. Since OBD port works with standard Parameter IDs (PIDs), the data extraction is carried out by using those IDs as an input to the car. For example, if you want vehicle's speed, you would need to send an input message to the car demanding values for Vehicle's speed. This work is done using an Arduino code,

4.1.4.2 Code for Engine Coolant Temperature

```
int inData = 0;

char inChar = 0;

String BuildINString = "";

String DisplayString="";

String WorkingString="";

long DisplayValue;

long A;

void setup() {

  Serial.begin(9600);

  // begin Serial Monitor with 9600 buad

}
```

```
void(* resetFunc) (void) = 0;

void loop() {

    // put your main code here, to run repeatedly:

    BuildINString = "";
```

```
    Serial.println("0105");

    // Send Coolant PID request 0105

    delay(2000);

    // Receive complete string from the serial buffer

    Read();

    Serial.print(BuildINString);

    BuildINString.trim();

    BuildINString = BuildINString.substring(0, 15);

    WorkingString = BuildINString.substring(11,13);

    A = strtol(WorkingString.c_str(),NULL,16);

    //convert hex to decimal

    DisplayValue = A;

    DisplayString = String(DisplayValue -40) + " degree C    ";

    // Subtract 40 from decimal to get the right temperature

    Serial.print(DisplayString);
```

```
delay(1000);  
  
Serial.println("AT Z");  
  
delay(500);
```

```
resetFunc();  
  
delay(2000);  
  
}  
  
void Read(){  
  
    BuildINString="";  
  
    while(Serial.available() > 0)  
  
    {  
  
        inData=0;  
  
        inChar=0;  
  
        inData = Serial.read();  
  
        inChar=char(inData);  
  
        if(inChar != "?" && inChar != ">" && inChar != " "){  
  
            BuildINString = BuildINString + inChar;  
  
        }}  
  
}
```

The code gives you value of **Engine Coolant Temperature** in Celsius scale. Same code can be changed for different parameters.

4.1.4.3 Code for Battery Voltage

For voltage values, same code is used with some changes that include.

- Using “AT RV” instead of ‘0105”.
- For battery voltage, no further calculations are required so Voltage values can be seen just by using

```
Serial.print ( BuildINString );
```

4.1.4.4 Code for Engine Load

Similarly, we can get Engine load in percentage by,

- Using “0104” instead of ‘0105”.
- For engine load, further calculations are required so

```
DisplayValue = DisplayValue * (100/255);
```

- Hence obtained data is converted in Percentage. Data can be displayed by,

```
DisplayString = String(DisplayValue) + " % ";  
Serial.print (DisplayString);
```

4.1.4.5 Code for Intake Air Temperature

For intake air temperature, we just need to change PID in Arduino code above,

- Using “010F” instead of ‘0105”.
- Result is shown in Celsius scale.

4.1.4.6 Code for Throttle Position

Similarly, we can get Engine load in percentage by,

- Using “0111” instead of ‘0105”.
- For throttle position, further calculations are required so

```
DisplayValue = DisplayValue * (100/255);
```

- Hence obtained data is converted in Percentage. Data can be shown by,

```
DisplayString = String(DisplayValue) + " % ";  
Serial.print (DisplayString);
```

4.1.4.7 Code for Vehicle Speed

Vehicle Speed is varying data and its values change every second. But fortunately, further calculations are not required since the speed values are directly received.

- Using “010D” instead of ‘0105’.
- Data can be displayed as,

```
DisplayString = String(DisplayValue) + " km/h ";  
Serial.print (DisplayString);
```

4.1.4.8 Code for Engine Revolution per minute (RPM)

For engine rpm, the code mentioned above is altered such that to use PID 010C. Hence

- Using “010C” instead of ‘0105’.
- The string received by passing this PID is a bit complex.

41 0C 0E 3E

Here 0E represents A and 3E represents another variable B.

```
long B;  
BuildINString = BuildINString.substring(0, 17);  
B = BuildINString.substring(15,17);
```



```
//convert to hex
```

```
B = strtol(B.c_str(),NULL,16);
```

- So for further calculation since we have got A and B so,

```
DisplayValue = ((255 * A)+B) / 4;
```

- RPM value can be displayed as,

```
DisplayString = String(DisplayValue) + " rpm ";
```

```
Serial.print (DisplayString);
```

4.1.5 GPS Tracker

The Transmission pin of the GPS module is linked to the Receiver pin of the Arduino, and the other way round. The appropriate sketch is uploaded to the Arduino, which receives location data in the form of Longitude and Latitude.

4.1.6 Send Data over GSM

Code given below shows how message is sent to the mobile user. However, this is just a basic code, the conditions are not added here like what happens when user clicks Refresh button etc.

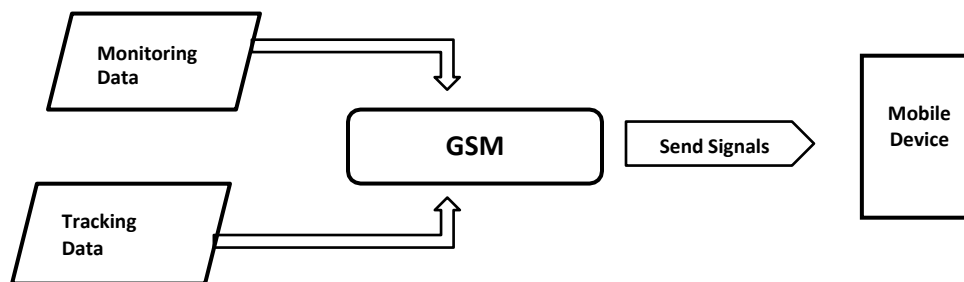


Figure 13: Send Data Through GSM

SMS service provides offline tracking hence making the project more effective. SMS is received on mobile device and shown on a mobile application.

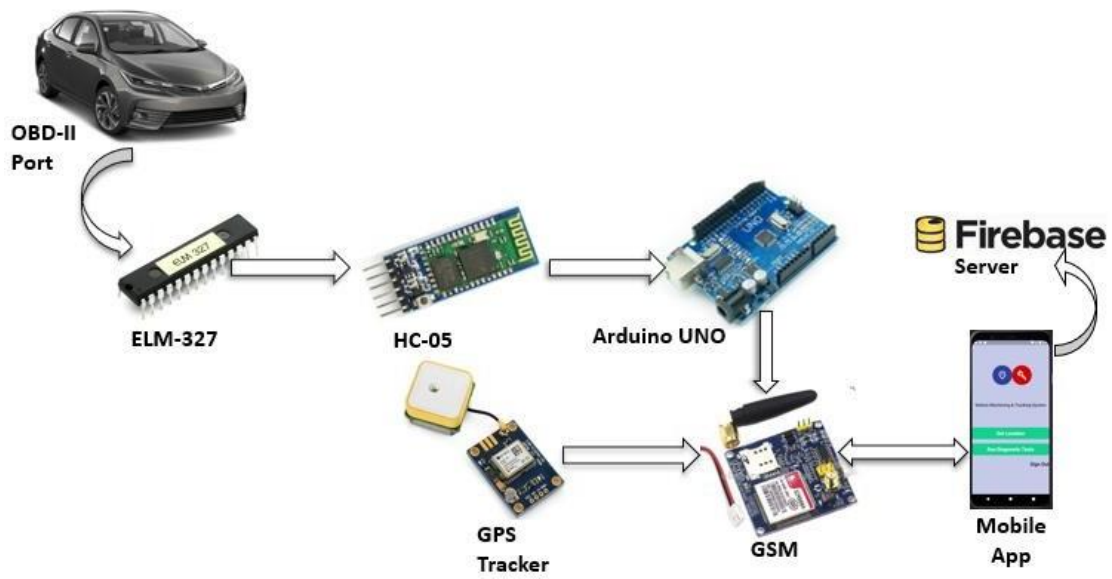


Figure 14: Block Diagram

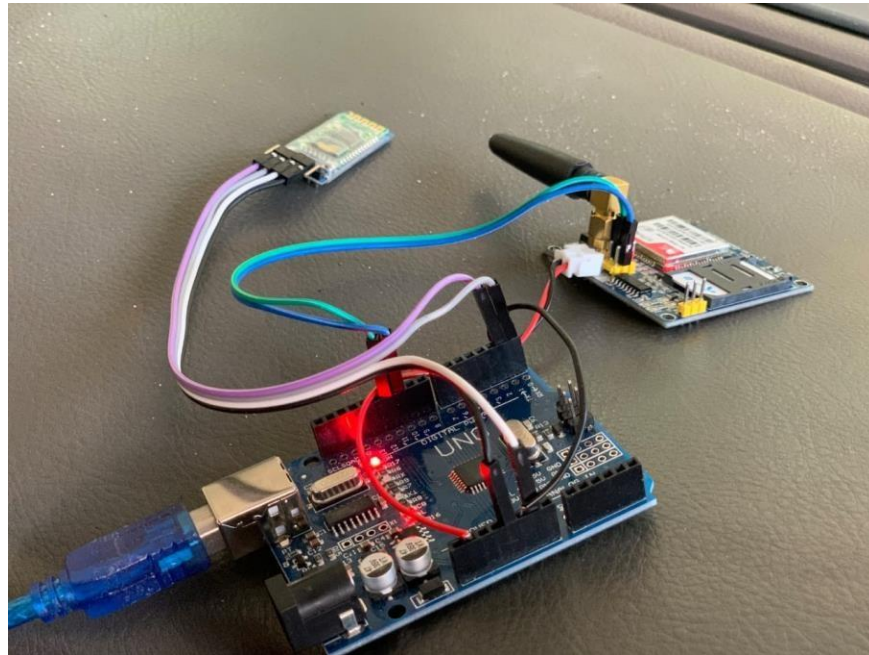


Figure 15: Hardware Interface Unit

4.2 Project Software Working

4.2.1 Send Data to Mobile Device through GSM

Following code is to send data to user's mobile phone providing an offline tracking and monitoring experience.

```
#include <SoftwareSerial.h>
SoftwareSerial SIM900A(10,11);
void setup()
{
  SIM900A.begin(9600); // baud rate of GSM Module
  Serial.begin(9600); // baud rate of Serial Monitor (Arduino)
  Serial.println ("SIM900A Ready");
  delay(100);
  Serial.println ("Type s to send message or r to receive message");
}
```

```
void loop()
{

  SendMessage();
  delay(1*10^7);
  //big delay so that user is not disturbed.
}

void SendMessage()
{
  Serial.println ("Sending Message");
```

```

SIM900A.println("AT+CMGF=1"); //Sets the GSM Module in Text Mode
delay(1000);

//Mobile phone number to send message
SIM900A.println("AT+CMGS=\"+921234567890\"");
delay(1000);

// Message content
SIM900A.println("L:" + longitude + "\n L: " + latitude + "\n temp: " + Coolant_Temperature +
"\n Load: " + Engine_load + "\n BV: "+Battery_voltage);
delay(100);

Serial.println ("Finish");

SIM900A.println((char)26);// ASCII code of CTRL+Z
delay(1000);
}

void RecieveMessage()
{
// AT Command to receive a live SMS
SIM900A.println("AT+CNMI=2,2,0,0,0");
delay(1000);
}

```

This is a general code, however the conditions are not added here such as what happens if user presses the “Refresh” button on mobile app etc. This code is just to show how the GSM module would send data.

4.2.2 Front End Coding for Monitoring and tracking

The following code is written in the Visual Studio Code editor for the Flutter SDK.

```
first_app > lib > mainsignin.dart > main
1  import 'package:firebase_core/firebase_core.dart';
2  import 'package:first_app/src/app.dart';
3  import 'package:first_app/src/screens/login.dart';
4  import 'package:first_app/src/screens/home.dart';
5  import 'package:flutter/material.dart';
6
7  Run | Debug | Profile
8  void main() async {
9    debugShowCheckedModeBanner:
10   false;
11   WidgetsFlutterBinding.ensureInitialized();
12   await Firebase.initializeApp();
13   runApp(LoginScreen());
14 }
```

```
1  import 'package:first_app/copy.dart';
2  import 'package:flutter/material.dart';
3  import 'package:flutter/services.dart';
4  import 'package:first_app/src/constants.dart';
5
6  class LoginScreen extends StatefulWidget {
7    @override
8    _LoginScreenState createState() => _LoginScreenState();
9  }
10
11 class _LoginScreenState extends State<LoginScreen> {
12   bool _rememberMe = false;
13
14   Widget _buildEmailTF() {
15     return Column(
16       crossAxisAlignment: CrossAxisAlignment.start,
17       children: <Widget>[
18         Text(
19           'Email',
20           style: kLabelStyle,
21         ), // Text
22       ],
23     );
24   }
25 }
```

```

22   SizedBox(height: 10.0),
23   Container(
24     alignment: Alignment.centerLeft,
25     decoration: BoxDecorationStyle,
26     height: 60.0,
27     child: TextField(
28       keyboardType: TextInputType.emailAddress,
29       style: TextStyle(
30         color: Colors.white,
31         fontFamily: 'OpenSans',
32       ), // TextStyle
33       decoration: InputDecoration(
34         border: InputBorder.none,
35         contentPadding: EdgeInsets.only(top: 14.0),
36         prefixIcon: Icon(
37           Icons.email,
38           color: Colors.white,
39         ), // Icon
40         hintText: 'Enter your Email',
41         hintStyle: kHintTextStyle,
42       ), // InputDecoration

```



```

43     ), // TextField
44     ), // Container
45   ], // <Widget>[]
46 ); // Column
47 }
48
49 Widget _buildPasswordTF() {
50   return Column(
51     crossAxisAlignment: CrossAxisAlignment.start,
52     children: <Widget>[
53       Text(
54         'Password',
55         style: kLabelStyle,
56       ), // Text
57       SizedBox(height: 10.0),
58       Container(
59         alignment: Alignment.centerLeft,
60         decoration: kBoxDecorationStyle,
61         height: 60.0,
62         child: TextField(
63           obscureText: true,

```

```

64   style: TextStyle(
65     color: Colors.white,
66     fontFamily: 'OpenSans',
67   ), // TextStyle
68   decoration: InputDecoration(
69     border: InputBorder.none,
70     contentPadding: EdgeInsets.only(top: 14.0),
71     prefixIcon: Icon(
72       Icons.lock,
73       color: Colors.white,
74     ), // Icon
75     hintText: 'Enter your Password',
76     hintStyle: kHintTextStyle,
77   ), // InputDecoration
78   ), // TextField
79   ), // Container
80 ], // <Widget>[]
81 ); // Column
82 }
83
84 Widget _buildForgotPasswordBtn() {

```

```

85     return Container(
86       alignment: Alignment.centerRight,
87       child: FlatButton(
88         onPressed: () => print('Forgot Password Button Pressed'),
89         padding: EdgeInsets.only(right: 0.0),
90         child: Text(
91           'Forgot Password?',
92           style: kLabelStyle,
93         ), // Text
94       ), // FlatButton
95     ); // Container
96   }
97
98   Widget _buildRememberMeCheckbox() {
99     return Container(
100       height: 20.0,
101       child: Row(
102         children: <Widget>[
103           Theme(
104             data: ThemeData(unselectedWidgetColor: Colors.white),
105             child: Checkbox(

```

```

106       value: _rememberMe,
107       checkColor: Colors.green,
108       activeColor: Colors.white,
109       onChanged: (value) {
110         setState(() {
111           _rememberMe = value;
112         });
113       },
114     ), // Checkbox
115   ), // Theme
116   Text(
117     'Remember me',
118     style: kLabelStyle,
119   ), // Text
120 ], // <Widget>[]
121 ), // Row
122 ); // Container
123 }
124
125 Widget _buildLoginBtn() {
126   return Container(

```



```

127 padding: EdgeInsets.symmetric(vertical: 25.0),
128 width: double.infinity,
129 child: RaisedButton(
130   elevation: 5.0,
131   onPressed: () {
132     Navigator.of(context)
133       .pushReplacement(MaterialPageRoute(builder: (context) => OBD()));
134   },
135   padding: EdgeInsets.all(15.0),
136   shape: RoundedRectangleBorder(
137     borderRadius: BorderRadius.circular(30.0),
138   ), // RoundedRectangleBorder
139   color: Colors.white,
140   child: Text(
141     'LOGIN',
142     style: TextStyle(
143       color: Color(0xFF527DAA),
144       letterSpacing: 1.5,
145       fontSize: 18.0,
146       fontWeight: FontWeight.bold,
147       fontFamily: 'OpenSans'

```

```

148     ), // TextStyle
149   ), // Text
150 ), // RaisedButton
151 ); // Container
152 }
153
154 Widget _buildSignInWithText() {
155   return Column(
156     children: <Widget>[
157       Text(
158         '- OR -',
159         style: TextStyle(
160           color: Colors.white,
161           fontWeight: FontWeight.w400,
162         ), // TextStyle
163       ), // Text
164       SizedBox(height: 20.0),
165       Text(
166         'Sign in with',
167         style: kLabelStyle,
168       ), // Text

```

```

169     ], // <Widget>[]
170   ); // Column
171 }
172
173 Widget _buildSocialBtn(Function onTap, AssetImage logo) {
174   return GestureDetector(
175     onTap: onTap,
176     child: Container(
177       height: 60.0,
178       width: 60.0,
179       decoration: BoxDecoration(
180         shape: BoxShape.circle,
181         color: Colors.white,
182         boxShadow: [
183           BoxShadow(
184             color: Colors.black26,
185             offset: Offset(0, 2),
186             blurRadius: 6.0,
187           ), // BoxShadow
188         ],
189         image: DecorationImage(
190           image: logo,
191         ), // DecorationImage
192       ), // BoxDecoration
193     ), // Container
194   ); // GestureDetector
195 }

```

```

196
197 Widget _buildSocialBtnRow() {
198   return Padding(
199     padding: EdgeInsets.symmetric(vertical: 30.0),
200     child: Row(
201       mainAxisAlignment: MainAxisAlignment.spaceEvenly,
202       children: <Widget>[
203         _buildSocialBtn(
204           () => print('Login with Facebook'),
205           AssetImage(
206             'assets/logos/facebook.jpg',
207           ), // AssetImage
208         ),
209         _buildSocialBtn(
210           () => print('Login with Google'),
211           AssetImage(
212             'assets/logos/google.jpg',
213           ), // AssetImage
214         ),
215       ], // <Widget>[]
216     ), // Row
217   ); // Padding
218 }
219

```

```

220 Widget _buildSignupBtn() {
221   return GestureDetector(
222     onTap: () {
223       Navigator.of(context)
224         .pushReplacement(MaterialPageRoute(builder: (context) => OBD()));
225     },
226     child: RichText(
227       text: TextSpan(
228         children: [
229           TextSpan(
230             text: 'Don\'t have an Account? ',
231             style: TextStyle(
232               color: Colors.white,
233               fontSize: 18.0,
234               fontWeight: FontWeight.w400,
235             ), // TextStyle
236           ), // TextSpan
237           TextSpan(
238             text: 'Sign Up',
239             style: TextStyle(
240               color: Colors.white,
241               fontSize: 18.0,
242               fontWeight: FontWeight.bold,
243             ), // TextStyle
244           ), // TextSpan
245         ],

```

```

246       ), // TextSpan
247     ), // RichText
248   ); // GestureDetector
249 }
250
251 @override
252 Widget build(BuildContext context) {
253   return Scaffold(
254     body: AnnotatedRegion<SystemUiOverlayStyle>(
255       value: SystemUiOverlayStyle.light,
256       child: GestureDetector(
257         onTap: () => FocusScope.of(context).unfocus(),
258         child: Stack(
259           children: <Widget>[
260             Container(
261               height: double.infinity,
262               width: double.infinity,
263               decoration: BoxDecoration(
264                 gradient: LinearGradient(
265                   begin: Alignment.topCenter,
266                   end: Alignment.bottomCenter,
267                   colors: [
268                     Color(0xFF73AEF5),
269                     Color(0xFF61A4F1),
270                     Color(0xFF478DE0),
271                     Color(0xFF398AE5),

```

```

272     ],
273     stops: [0.1, 0.4, 0.7, 0.9],
274   ), // LinearGradient
275   ), // BoxDecoration
276   ), // Container
277   Container(
278     height: double.infinity,
279     child: SingleChildScrollView(
280       physics: AlwaysScrollableScrollPhysics(),
281       padding: EdgeInsets.symmetric(
282         horizontal: 40.0,
283         vertical: 120.0,
284       ), // EdgeInsets.symmetric
285       child: Column(
286         mainAxisAlignment: MainAxisAlignment.center,
287         children: <Widget>[
288           Text(
289             'Sign In',
290             style: TextStyle(
291               color: Colors.white,
292               fontFamily: 'OpenSans',
293               fontSize: 30.0,
294               fontWeight: FontWeight.bold,
295             ), // TextStyle
296           ), // Text
297           SizedBox(height: 30.0),

```

```

298     _buildEmailTF(),
299     SizedBox(
300       height: 30.0,
301     ), // SizedBox
302     _buildPasswordTF(),
303     _buildForgotPasswordBtn(),
304     _buildRememberMeCheckbox(),
305     _buildLoginBtn(),
306     _buildSignInWithText(),
307     _buildSocialBtnRow(),
308     _buildSignupBtn(),
309   ], // <Widget>[]
310   ), // Column
311   ), // SingleChildScrollView
312   ) // Container
313 ], // <Widget>[]
314 ), // Stack
315 ), // GestureDetector
316 ), // AnnotatedRegion
317 ); // Scaffold
318 }
319 }
320

```

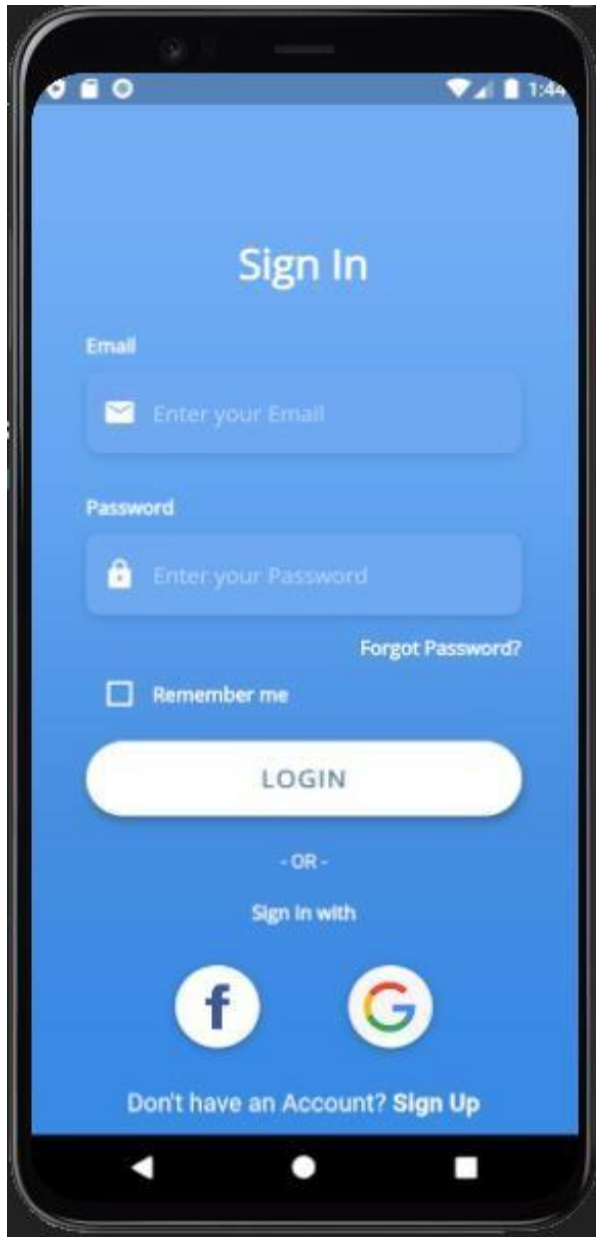


Figure 16: Application Sign in Page

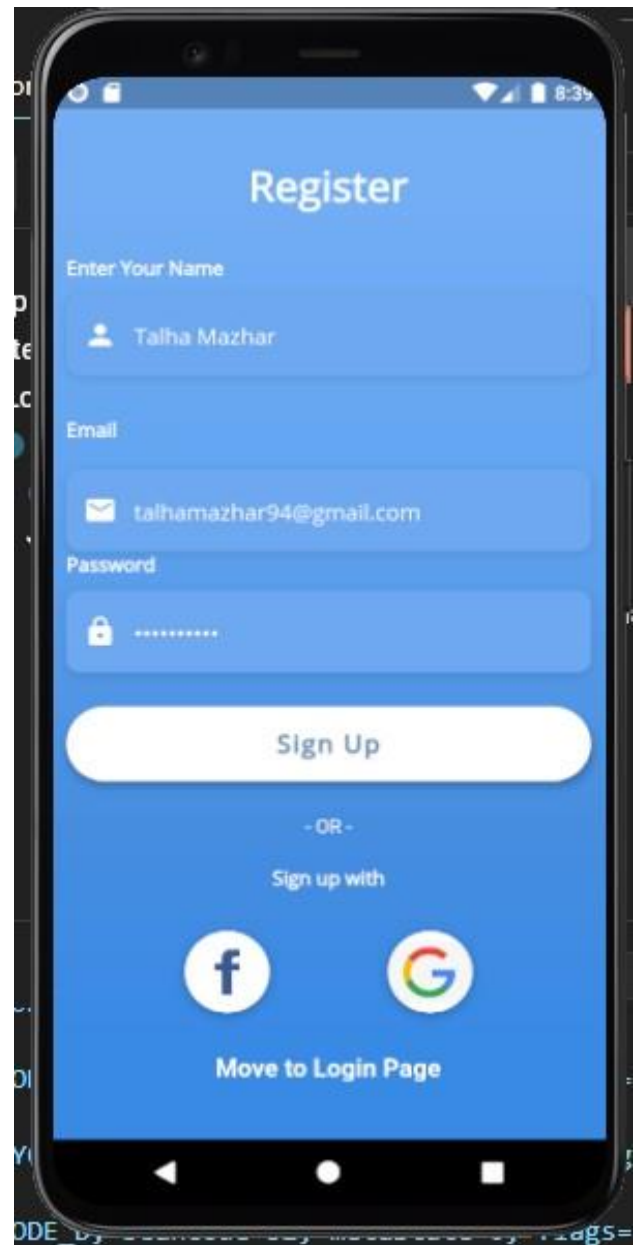


Figure 17: Application Registration Page

The coding for the Home Page and its result is shown in the following figures:

```
1  import 'package:first_app/src/screens/login.dart';
2  import 'package:flutter/cupertino.dart';
3  import 'package:flutter/material.dart';
4  import 'package:firebase_auth/firebase_auth.dart';
5
6  class OBD extends StatelessWidget {
7    @override
8    Widget build(BuildContext context) {
9      return MaterialApp(
10         title: 'Vehicle Monitoring Tracking System',
11         debugShowCheckedModeBanner: false,
12         home: MyHomePage(),
13       ); // MaterialApp
14     }
15   }
16
17   class MyHomePage extends StatelessWidget {
18     final auth = FirebaseAuth.instance;
19     @override
20     Widget build(BuildContext context) {
21       return new Scaffold(
22         backgroundColor: Color(0xFFC5CAE9),
23         body: Center(
24           child: Column(mainAxisAlignment: MainAxisAlignment.center, children: <
25             Widget>[
26               new Stack(alignment: Alignment.center, children: <Widget>[
27                 new Container(
28                   margin:
29                     new EdgeInsets.only(bottom: 90.0, right: 5.5, left: 120.0),
30                   height: 100.0,
31                   width: 100.0,
32                   decoration: new BoxDecoration(
33                     borderRadius: new BorderRadius.circular(90.0),
34                     color: Colors.redAccent.shade700), // BoxDecoration
35                   child: new Icon(
36                     Icons.build_outlined,
37                     size: 50.0,
38                     color: Colors.white,
39                   ), // Icon
40                 ), // Container
41                 new Container(
42                   margin:
43                     new EdgeInsets.only(right: 100.0, bottom: 90.0, left: 15.5),
44                   height: 100.0,
45                   width: 100.0,
46                   decoration: new BoxDecoration(
47                     borderRadius: new BorderRadius.circular(250.0),
48                     color: Colors.indigo.shade700), // BoxDecoration
49                   child: new Icon(
50                     Icons.place_outlined,
51                     size: 50.0,
52                     color: Colors.white,
53                   ), // Icon
```

```

54     ) // Container
55   ]), // <Widget>[] // Stack
56   new Container(
57     padding: EdgeInsets.only(bottom: 100.0),
58     child: Row(
59       mainAxisAlignment: MainAxisAlignment.center,
60       children: [
61         Text(
62           'Vehicle Monitoring & Tracking System',
63           textScaleFactor: 1.5,
64           style: TextStyle(fontWeight: FontWeight.bold),
65         ) // Text
66       ],
67     ), // Row
68   ), // Container
69   new Row(
70     children: <Widget>[
71       Expanded(
72         child: Padding(
73           padding: const EdgeInsets.only(bottom: 20.0),
74           child: InkWell(
75             onTap: () {
76               print("Container clicked");
77             },
78             child: new Container(
79               height: 60.0,

```

```

80         alignment: Alignment.center,
81         decoration: BoxDecoration(
82           color: Color(0xFF18D191),
83           borderRadius: new BorderRadius.circular(10.0)), // BoxDecoration
84         child: Text('Get Location',
85         style: new TextStyle(
86           fontWeight: FontWeight.bold,
87           fontSize: 25.0,
88           color: Colors.white)), // TextStyle // Text
89       ), // Container
90     ), // InkWell
91   ), // Padding
92 ) // Expanded
93 ], // <Widget>[]
94 ), // Row
95 new Row(
96 children: <Widget>[
97   Expanded(
98     child: Padding(
99       padding: const EdgeInsets.only(top: 1.0),
100      child: InkWell(
101        onTap: () {
102          print("Container clicked");
103        },
104        child: new Container(
105          height: 60.0,
106          alignment: Alignment.center,

```

```

107   decoration: BoxDecoration(
108     color: Color(0xFF18D191),
109     borderRadius: new BorderRadius.circular(10.0)), // BoxDecoration
110   child: Text('Run Diagnostic Tests',
111     style: new TextStyle(
112       fontWeight: FontWeight.bold,
113       fontSize: 25.0,
114       color: Colors.white)), // TextStyle // Text
115   ), // Container
116   ), // InkWell
117   ), // Padding
118   ) // Expanded
119 ], // <Widget>[]
120 ), // Row
121 new Row(
122   children: <Widget>[
123     Expanded(
124       child: Padding(
125         padding: const EdgeInsets.only(bottom: 10.0),
126         child: InkWell(
127           onTap: () {
128             auth.signOut();
129             Navigator.of(context).pushReplacement(MaterialPageRoute(
130               builder: (context) => LoginScreen())); // MaterialPageRoute
131           },
132           child: new Container(

```

```

133     height: 60.0,
134     alignment: Alignment.bottomRight,
135     decoration: BoxDecoration(
136       color: Color(0xFFC5CAE9),
137       borderRadius: new BorderRadius.circular(10.0)), // BoxDecoration
138     child: Text('Sign Out',
139       style: new TextStyle(
140         fontWeight: FontWeight.bold,
141         fontSize: 20.0,
142         color: Colors.white), // TextStyle
143     ), // Text
144   ), // Container
145   ), // InkWell
146   ) // Padding
147   ), // Expanded
148   ], // <Widget>[]
149   ) // Row
150   ]), // <Widget>[] // Column
151   ), // Center
152   ); // Scaffold
153 }
154 }
155

```

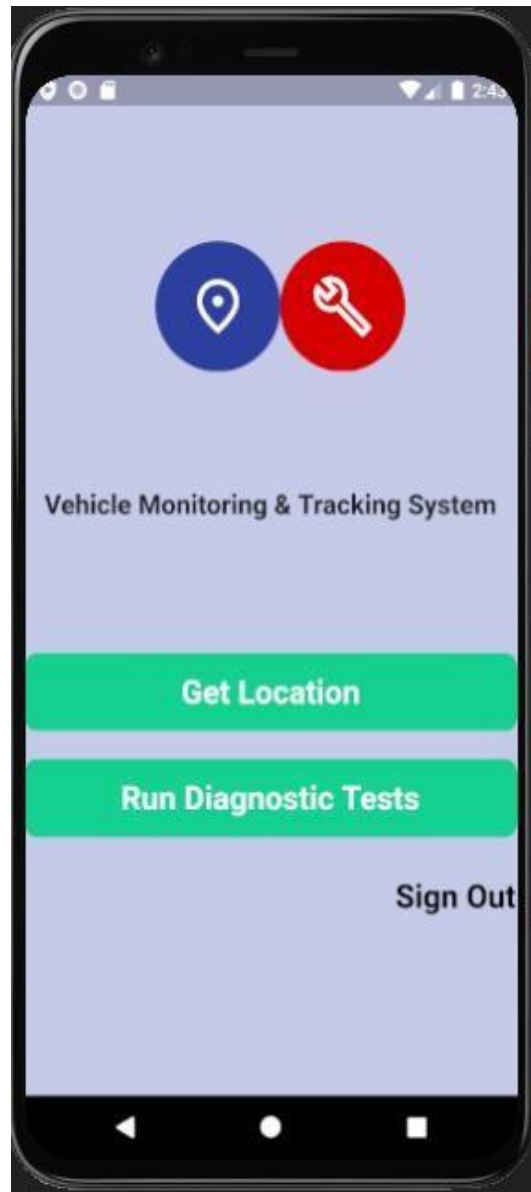



Figure 18: Application Home Page

4.2.3 Firebase Authentication:

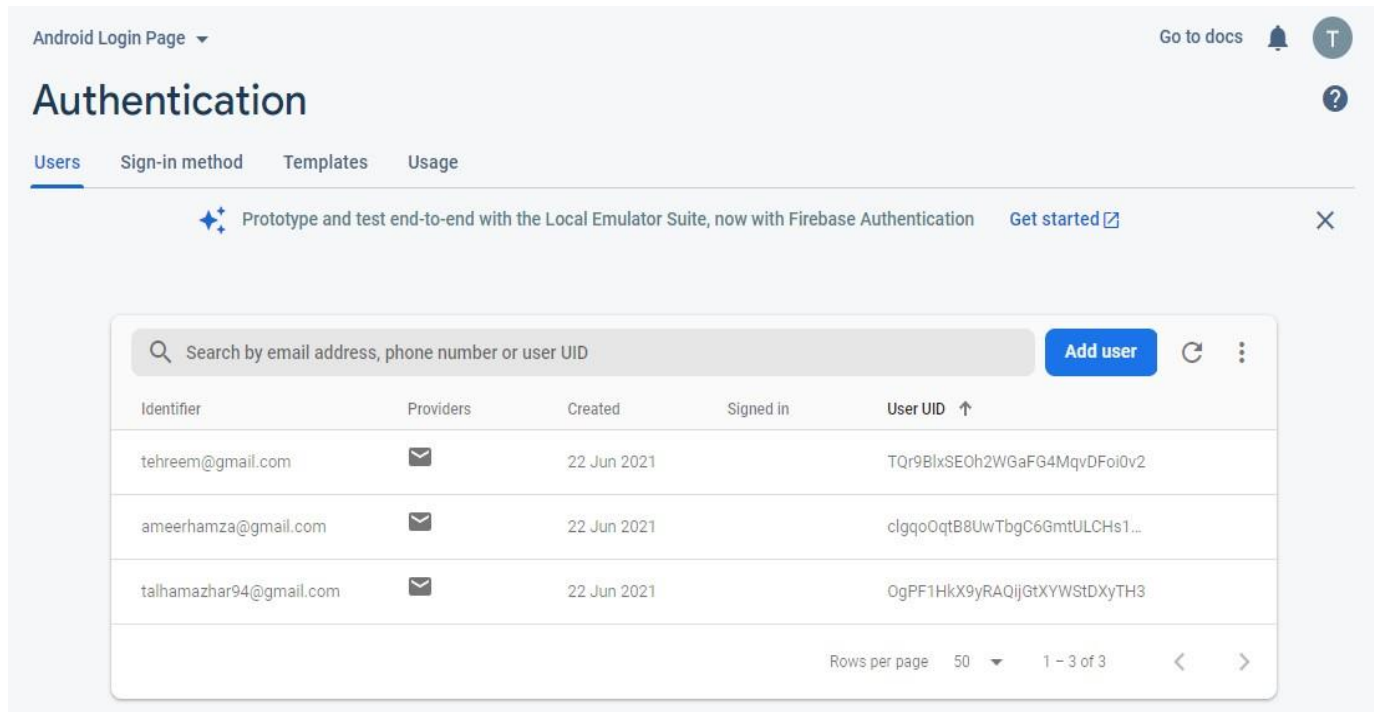


Figure 19: Firebase Authentication Page

We have used firebase for server. Firebase retrieve real time database. The task is how to get a data and how to perform a simple queries on data. The data at server will show results that are discussed in next chapter.

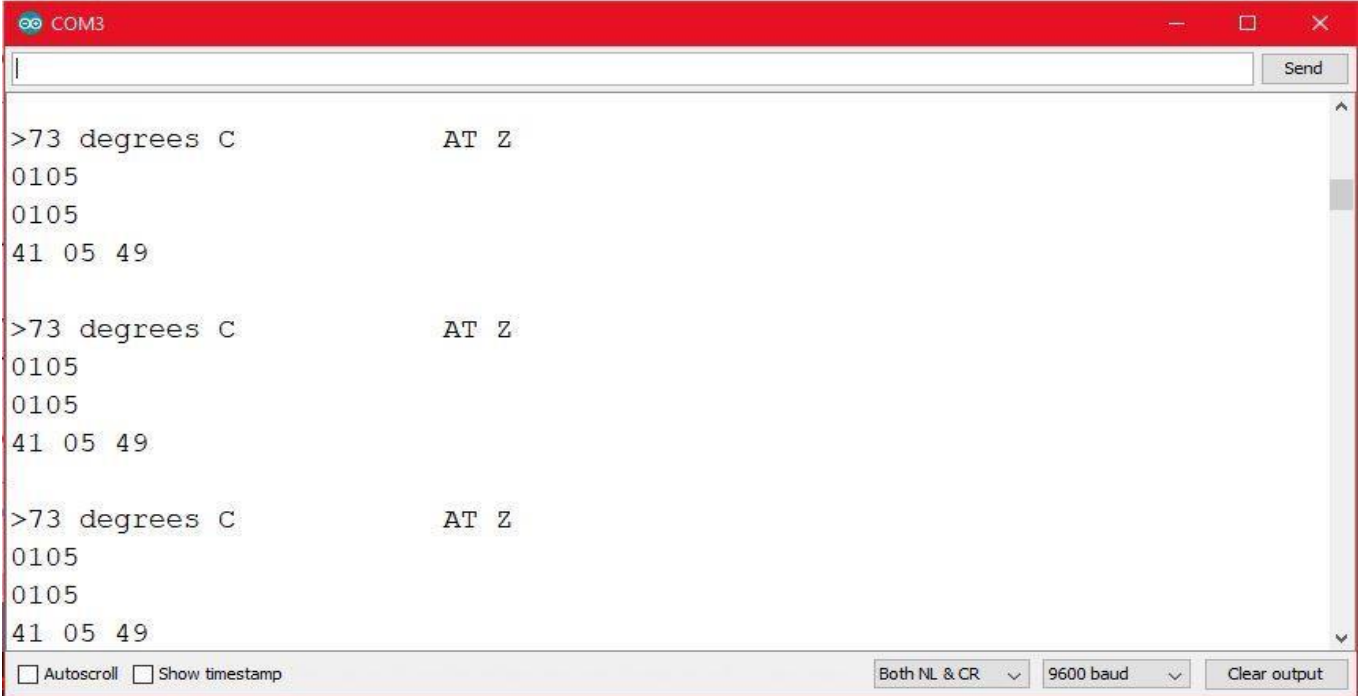
CHAPTER 5: RESULTS

Chapter 5: Results

Results include real time tracking and monitoring of various car monitoring like speed, engine rpm, engine coolant temperature, engine load, battery voltage and air filters.

Engine Coolant Temperature:

Results are in degree Celsius,



The screenshot shows a serial terminal window titled 'COM3'. The window contains three identical blocks of data, each representing a temperature reading. Each block consists of four lines: a command prompt '>', the text '>73 degrees C', the text 'AT Z', and the number '0105'. Below these three blocks, the text '41 05 49' is displayed. At the bottom of the window, there are several controls: a checkbox for 'Autoscroll', a checkbox for 'Show timestamp', a dropdown menu set to 'Both NL & CR', a dropdown menu set to '9600 baud', and a 'Clear output' button. A 'Send' button is located at the top right of the terminal area.

```
>73 degrees C      AT Z
0105
0105
41 05 49

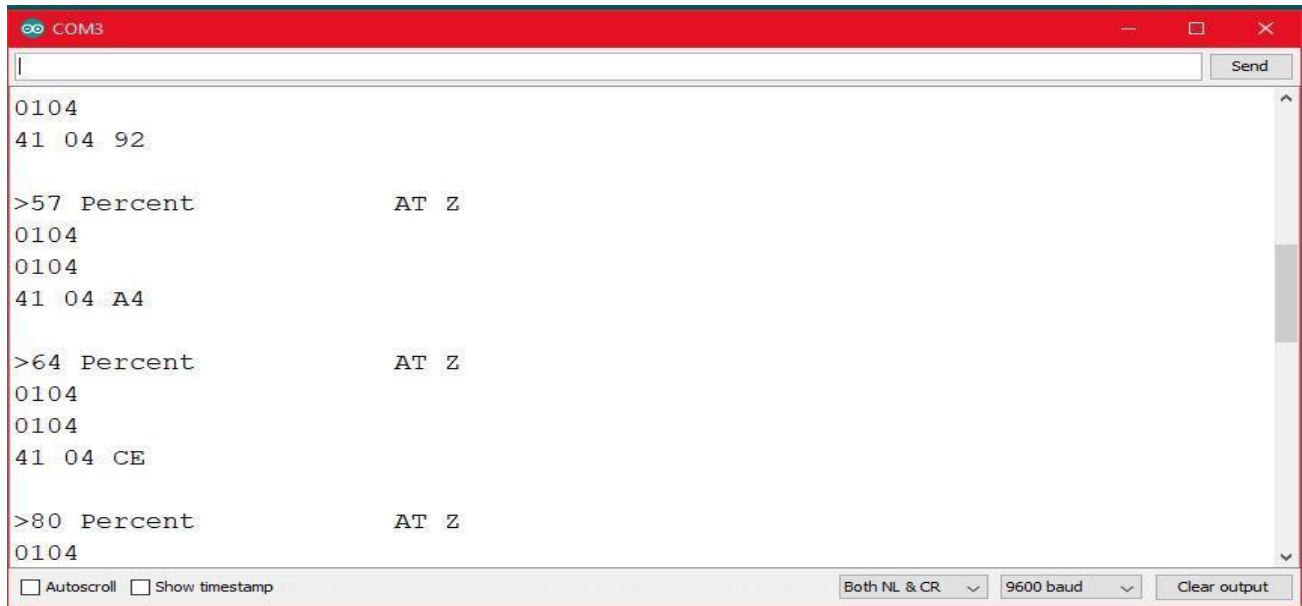
>73 degrees C      AT Z
0105
0105
41 05 49

>73 degrees C      AT Z
0105
0105
41 05 49
```

Figure 20: Engine Coolant Temperature Results

Engine Load:

Result shown in Percentage load,



The screenshot shows a serial terminal window titled 'COM3'. The output consists of three sets of data, each starting with a command and followed by a response. The first set shows a command '>57 Percent' and a response 'AT Z' followed by '0104' and '41 04 92'. The second set shows a command '>64 Percent' and a response 'AT Z' followed by '0104' and '41 04 A4'. The third set shows a command '>80 Percent' and a response 'AT Z' followed by '0104'. The terminal interface includes a 'Send' button, checkboxes for 'Autoscroll' and 'Show timestamp', and dropdown menus for 'Both NL & CR' and '9600 baud', along with a 'Clear output' button.

```
COM3
|
|
0104
41 04 92

>57 Percent          AT Z
0104
0104
41 04 A4

>64 Percent          AT Z
0104
0104
41 04 CE

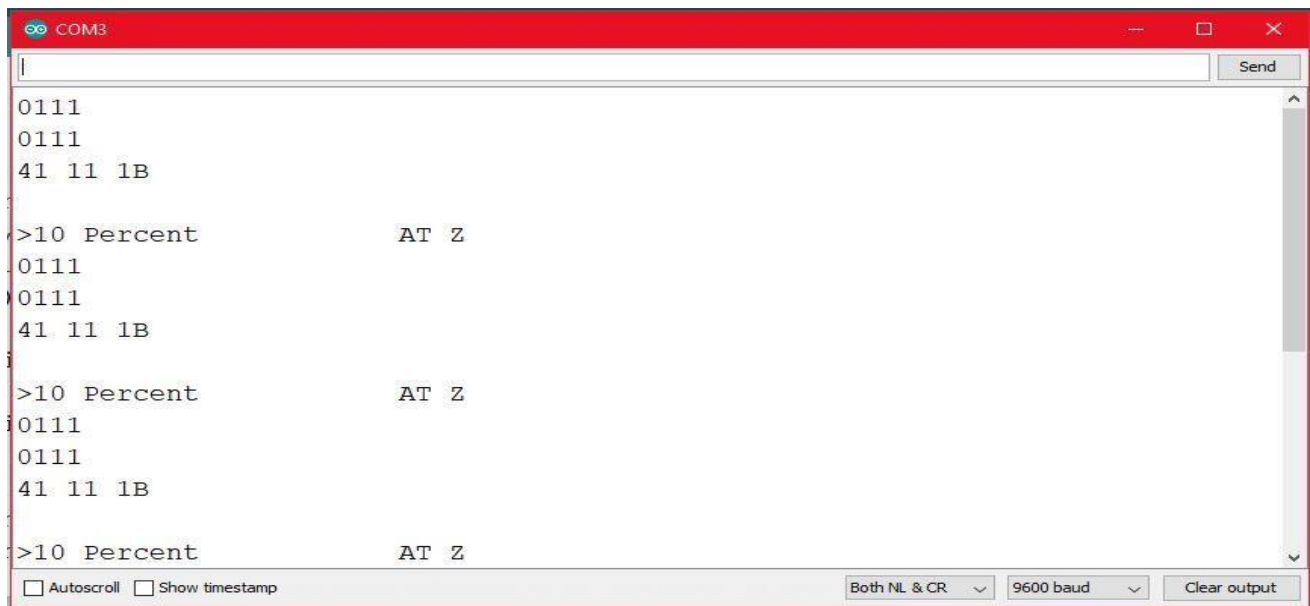
>80 Percent          AT Z
0104

 Autoscroll  Show timestamp
Both NL & CR 9600 baud Clear output
```

Figure 21: Engine Load Results

Throttle Position:

Result shown in Percentage load,



The screenshot shows a serial terminal window titled 'COM3'. The output consists of three sets of data, each starting with a command and followed by a response. The first set shows a command '>10 Percent' and a response 'AT Z' followed by '0111' and '41 11 1B'. The second set shows a command '>10 Percent' and a response 'AT Z' followed by '0111' and '41 11 1B'. The third set shows a command '>10 Percent' and a response 'AT Z' followed by '0111' and '41 11 1B'. The terminal interface includes a 'Send' button, checkboxes for 'Autoscroll' and 'Show timestamp', and dropdown menus for 'Both NL & CR' and '9600 baud', along with a 'Clear output' button.

```
COM3
|
|
0111
0111
41 11 1B

>10 Percent          AT Z
0111
0111
41 11 1B

>10 Percent          AT Z
0111
0111
41 11 1B

>10 Percent          AT Z
0111
0111
41 11 1B

 Autoscroll  Show timestamp
Both NL & CR 9600 baud Clear output
```

Figure 22: Throttle position Results

Engine RPM :

Revolutions Per Minute values (x1000),

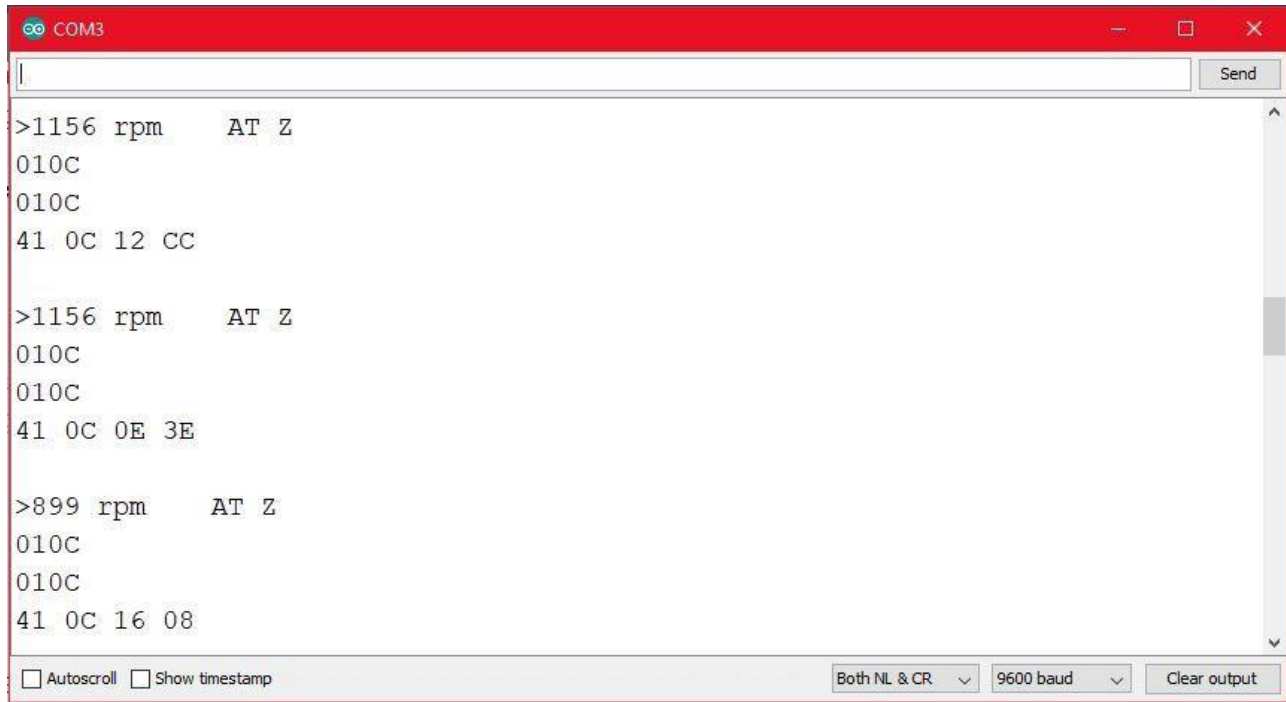


Figure 23: Engine RPM Results

Battery Voltage:

Battery Voltage in Volts,



Figure 24: Battery Voltage Results

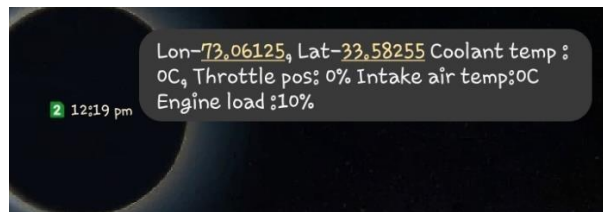


Figure 27: Message viewed on Mobile Screen



Figure 28 Tracking Device

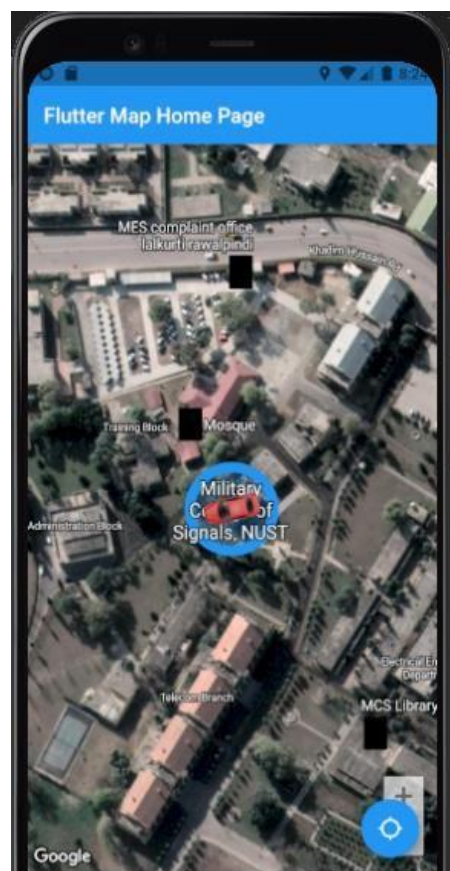


Figure 28: Location of vehicle

CHAPTER 6: CONCLUSION & NEXT STEPS

Chapter 6: Conclusion and Next Steps

This section will focus on how VMTS will provide its users effort-less monitoring and tracking of vehicle and what future improvements need to be done to make this project more feasible.

6.1 Conclusion:

The GSM based Vehicle Tracking and Monitoring system promises its user about real time tracking and monitoring of vehicle's parameter such as speed, coolant temperature, fuel consumption and other provided information via short message service. It can be used by nearly all automobiles that use standard Protocol. In Pakistan, as population is increasing, traffic vehicles are also increasing and unfortunately road accidents, theft of cars etc. also increasing. For an average man it is difficult to afford such an expensive tracking and monitoring sensors. VMTS provides the solution of tracking and monitoring at the same time as it is combination of cheap hardware and web application so any one can afford it.

6.2 Future Work:

Due to a shortage of time, many various tests, experiments, and modifications have been postponed. Future development will focus on expanding security services, parking sensors, and resolving issues that arise when a cellular network is unavailable.

APPENDICES

APPENDIX-A

SYNOPSIS

V.M.T.S (Vehicle Monitoring and Tracking System)

Extended Title: GSM Based Vehicle Monitoring and Tracking System through OBD Port
Brief Description of The Project / Thesis with Salient Specifications: This project aims to track and monitor any vehicle in real time at mobile application. The latitudes/longitudes, speed, coolant temperature and other provided information will be sent to its user via SMS.
Scope of Work: Wireless Access, Embedded System, Monitoring and Tracking.
Academic Objectives: <ul style="list-style-type: none">• To develop understanding of how monitoring and tracking is done.• Being able to get familiar with different languages (Flutter)• Develop critical thinking & problem solving skills that will also help us in Professional life.
Application / End Goal Objectives: Making an application that can <ul style="list-style-type: none">○ Monitor vehicle○ Record speed○ Monitor Engine Temperature○ Engine RPM○ Battery Voltage○ Live tracking○ Web Application
Previous Work Done on The Subject: Idea has been published in research papers and some parts have been implemented throughout world using different applications.
Material Resources Required: <ol style="list-style-type: none">1. Tracking device2. Device to provide engine temperature, car speed e.tc.
No of Students Required: 4 (Four) Group Members: <ol style="list-style-type: none">1. Abdul Rehman (Group Coordinator)2. Ameer Hamza Hassan3. Talha Mazhar4. Tehreem Mateen Zia
Special Skills Required: Programming in <ul style="list-style-type: none">• Flutter• Firebase

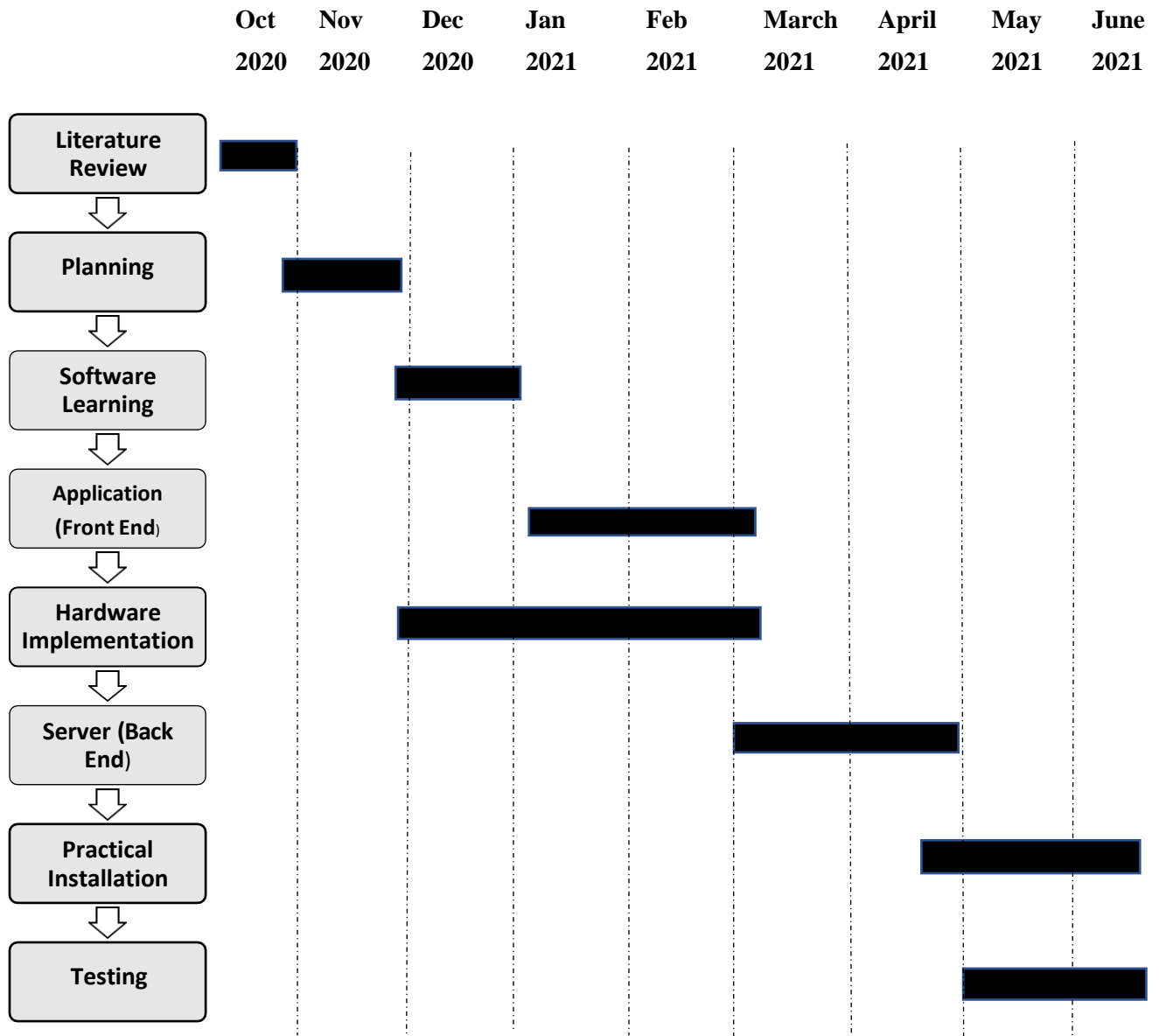
APPENDIX-B

Abbreviations List:

OBD	On Board Diagnostic
ECU	Engine Control Unit
GSM	Global System for Mobile Communication
GPS	Global Positioning System
AT	ATtention
SMS	Short Message Service
IDE	Integrated Circuit Board
PIDs	Parameter IDs
SAE	Society of Automotive Engineers
ISO	International Organization for Standardization

APPENDIX-C

Demonstration Outline



Appendix-D

AT Commands

Several parameters within the ELM327 can be adjusted in order to modify its behaviour. These do not normally have to be changed before attempting to talk to the vehicle, but occasionally the user may wish to customize these settings – for example by turning the character echo off, adjusting a timeout value, or changing the header bytes. In order to do this, internal 'AT' commands must be used.

Those familiar with PC modems will immediately recognize AT commands as a standard way in which modems are internally configured. The ELM327 uses essentially the same method, always watching the data sent by the PC, looking for messages that begin with the character 'A' followed by the character 'T'. If found, the next characters will be interpreted as an internal configuration or 'AT' command, and will be executed upon receipt of a terminating carriage return character. If the command is just a setting change, the ELM327 will reply with the characters 'OK', to say that

it was successfully completed.

Some of the following commands allow passing numbers as arguments in order to set the internal values. These will always be hexadecimal numbers which must generally be provided in pairs. The hexadecimal conversion chart in the OBD Commands section (page 31) may be helpful if you wish to interpret the values. Also, you should be aware that for the on/off types of commands, the second character is the number 1 or the number 0, the universal terms for on and off.

The remainder of this page, and the two pages following provide a summary of all of the commands that the current version of the ELM327 recognizes. A more complete description of each command begins on page 12. Note that the settings which are shown with an asterisk (*) are the default values.

AT Command Summary

General Commands

<CR>	repeat the last command
BRD hh	try Baud Rate Divisor hh
BRT hh	set Baud Rate Timeout
D	set all to Defaults
E0, E1	Echo off, or on*
FE	Forget Events
I	print the version ID
L0, L1	Linefeeds off, or on
LP	go to Low Power mode
M0, M1	Memory off, or on
RD	Read the stored Data byte
SD hh	Save Data byte hh
WS	Warm Start (quick software reset)
Z	reset all
@1	display the device description
@2	display the device identifier
@3 cccccccccc	store the @2 identifier

Programmable Parameter Commands

PP xx OFF	disable Prog Parameter xx
PP FF OFF	all Prog Parameters disabled
PP xx ON	enable Prog Parameter xx
PP FF ON	all Prog Parameters enabled
PP xx SV yy	for PP xx, Set the Value to yy
PPS	print a PP Summary

Voltage Reading Commands

CV dddd	Calibrate the Voltage to dd.dd volts
CV 0000	restore CV value to factory setting
RV	Read the input Voltage

Other

IGN	read the IgnMon input level
------------	-----------------------------

References:

1. Tahat, A., Said, A., Jaouni, F., & Qadamani, W. (2012, June). Android-based universal vehicle diagnostic and tracking system. In *2012 IEEE 16th International Symposium on Consumer Electronics* (pp. 137-143). IEEE.
2. Rajeevan, A., Payagala, N. K., & Lanka, S. (2017). Vehicle monitoring controlling and tracking system by using android application. *International Journal of Technical Research and Applications*, 4(1), 114-119.
3. ELM327, O. B. D. (2012). to RS232 Interpreter. *ELM Electronics*.
4. Mahoney, S. M. (2008). Creating a Wireless OBD-II Scanner.
5. Information retrieved from <https://happilyembedded.wordpress.com/2015/07/23/thirst-for-knowledge-sae-j1979-protocol/>
6. Information Retrieved from <https://ukdiss.com/examples/android-system-application-ehicles-on-board-diagnostics.php>

Plagiarism-1

ORIGINALITY REPORT

10 %	7 %	4 %	7 %
SIMILARITY INDEX	INTERNET SOURCES	PUBLICATIONS	STUDENT PAPERS

PRIMARY SOURCES

1	Submitted to Arts, Sciences & Technology University In Lebanon Student Paper	2 %
2	www.makeuseof.com Internet Source	2 %
3	docplayer.net Internet Source	1 %
4	Submitted to Auston Institute of Management and Technology Student Paper	1 %
5	Submitted to The University of the South Pacific Student Paper	1 %
6	hdl.handle.net Internet Source	1 %
7	Submitted to University of Mauritius Student Paper	<1 %
8	electrosome.com Internet Source	<1 %

9	en.wikipedia.org Internet Source	<1 %
10	Submitted to University of Hertfordshire Student Paper	<1 %
11	www.scribd.com Internet Source	<1 %
12	Mark Sellnau, Matthew Foster, Wayne Moore, James Sinnamon, Kevin Hoyer, William Klemm. "Second Generation GDCI Multi-Cylinder Engine for High Fuel Efficiency and US Tier 3 Emissions", SAE International Journal of Engines, 2016 Publication	<1 %
13	es.scribd.com Internet Source	<1 %
14	Ashraf Tahat, Ahmad Said, Fouad Jaouni, Waleed Qadamani. "Android-based universal vehicle diagnostic and tracking system", 2012 IEEE 16th International Symposium on Consumer Electronics, 2012 Publication	<1 %
15	Bhagyeshwari Chauhan, Avni Jain, Tanmay Chaturvedi, Sandeep Saini. "User Interactive and Assistive Fleet Management and Eco-Driving System", 2015 IEEE Region 10 Symposium, 2015 Publication	<1 %

16

jp.ic-on-line.cn
Internet Source

<1%

Exclude quotes Off

Exclude matches Off

Exclude bibliography Off