**Shared Transport Management System**

**By**

Atishi Miral (16BLC1003)

Malolan V. (16BLC1033)

**TABLE OF CONTENTS**

|  |  |  |  |
| --- | --- | --- | --- |
| **S. NO.** | **TITLE** | | **PAGE NO.** |
|  |  | ABSTRACT | 3 |
|  |  |  |  |
| 1 | INTRODUCTION |  | 4-6 |
|  |  |  |  |
|  | 1.1 | OBJECTIVESAND GOALS | 4 |
|  | 1.2 | BENEFITS | 4-5 |
|  | 1.3 | RECEPTION BY THE PUBLIC | 5-6 |
|  |  |  |  |
| 2 | DESIGN |  | 7-12 |
|  | 2.1 | OVERVIEW | 7 |
|  | 2.2 | HARDWARE ANALYSIS | 8-9 |
|  | 2.3 | SOFTWARE ANALYSIS | 10-12 |
|  |  |  |  |
| 3 | REALTIME IMPLEMENTATION | | 19-20 |
|  |  |  |  |
| 4 | CONCLUSION | AND FUTURE WORK | 21 |
|  | 4.1 | RESULT, CONCLUSION AND INTERFERANCE | 21 |
|  | 4.2 | FUTURE WORK | 21 |

5 REFERENCES 22

**ABSTRACT**

Shared transport systems are used by plenty of organizations and institutions for environmental, social, and transportation system benefits. Despite its myriad advantages, shared mobility often poses undesirable situations.

 The vehicle may arrive early, and therefore depart before stipulated time, thus causing passengers to miss the vehicle.

 The vehicle may arrive late or may have a breakdown, leaving the passengers unnotified about the whereabouts of the vehicle, causing them to reach their destination late.

 The driver may be rash with driving, accelerating or decelerating rapidly, thus denying the passengers of a smooth, comfortable journey.

Solutions to the cited problems have been proposed by creating a cloud-based system to

 Track the live location of the vehicle and display on a map interface

 Give real time updates to the passengers (arrival, delay, breakdown)

 Streamline the boarding process using RFID tags which would be present in the institution’s/organization’s ID cards to ensure timely and fair boarding

 Analyse driver’s performance by collecting the driver’s driving statistics.

1. **INTRODUCTION**

***1.1 OBJECTIVES AND GOALS***

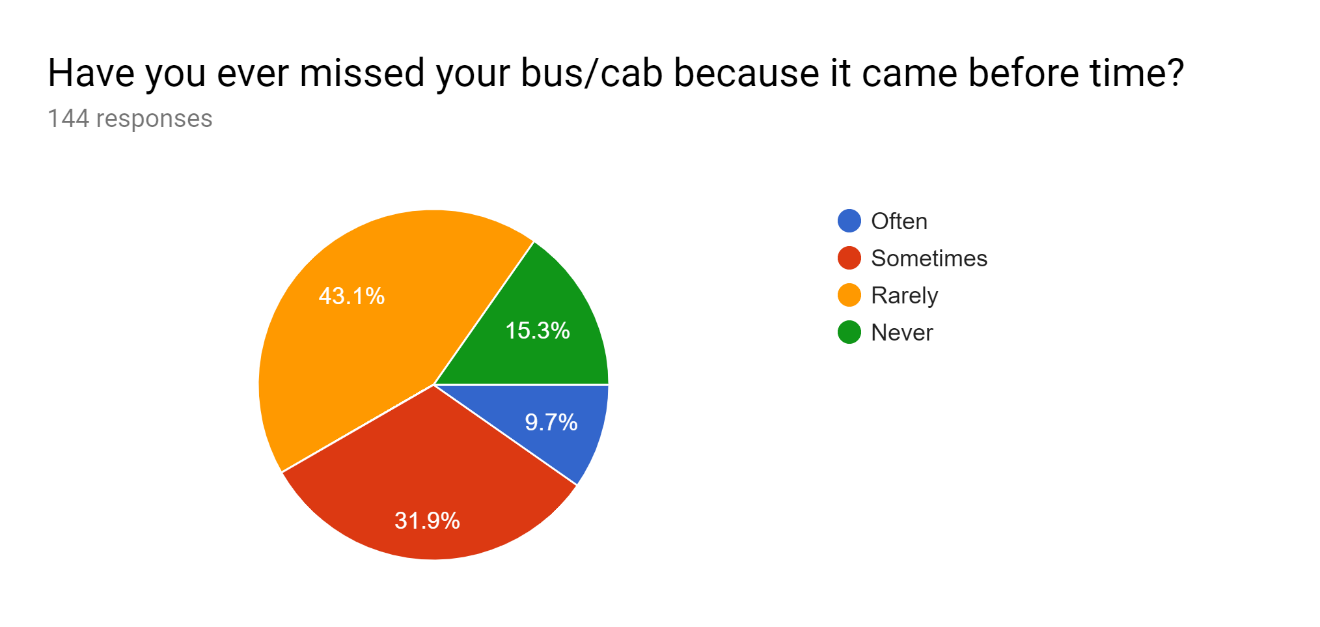
* track the live location of the vehicle and display on a map interface
* give real time updates to the passengers (arrival, delay, breakdown)
* to streamline the boarding process using RFID tags which would be present in the institution’s/organization’s ID cards to ensure timely and fair boarding
* analyse driver’s performance by collecting the driver’s driving statistics

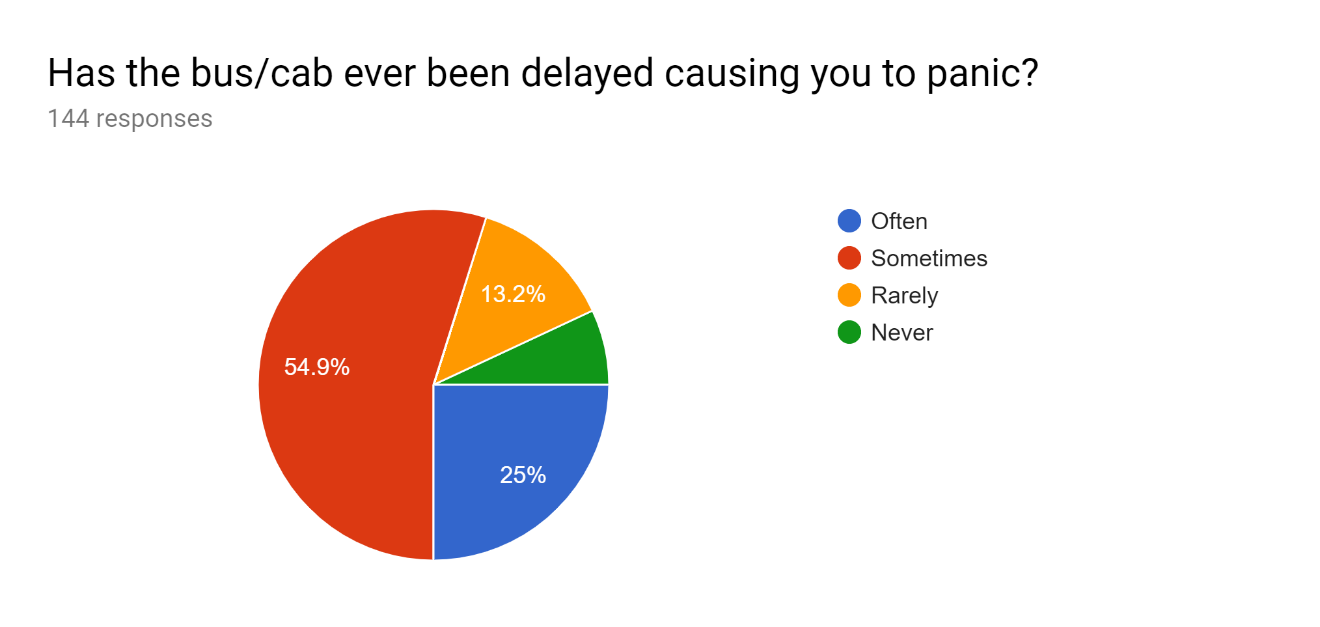
***1.2 BENEFITS***

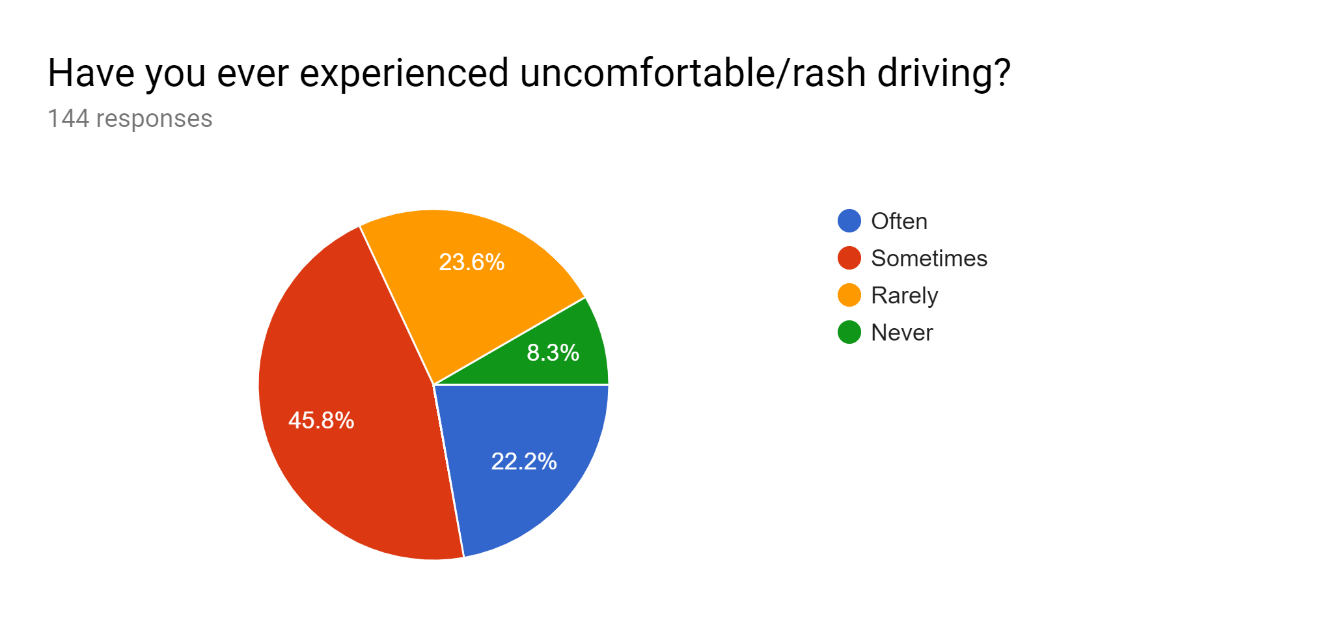
Shared mobility - the shared use of a vehicle - is an innovative transportation strategy that enables users to gain short-term access to transportation modes on an as-needed basis. The term shared mobility includes various forms of carsharing, bike-sharing, ridesharing (carpooling and vanpooling), busses and on-demand ride services. It can also include alternative transit services, such as paratransit, shuttles, and private transit services (called microtransit), which can supplement fixed-route bus and rail services. With diverse options for mobility on the rise, smartphone apps that aggregate these options and optimize routes for travelers are also proliferating. Shared mobility is having a transformative impact on many global cities by enhancing transportation accessibility, while simultaneously reducing driving and personal vehicle ownership. A number of environmental, social, and transportation-related benefits have been reported from the use of shared mobility modes. Several studies have documented reduced vehicle use, ownership, and vehicle miles/kilometers traveled. Cost savings and convenience are frequently cited as popular reasons for shifting to a shared mode. Shared mobility can also provide economic benefits in the form of cost savings, increased economic activity near public transit stations and multimodal hubs, and increased access by creating connections with origin points not previously accessible via traditional public transportation.

This sharing can occur among peers (e.g., community drivers, peer-to-peer carsharing, or bike-sharing) or through businesses (e.g., a carsharing operator). The sharing economy can improve efficiency, provide cost savings, monetize underused resources, and offer social and environmental benefits.

***1.3 RECEPTION BY THE PUBLIC***





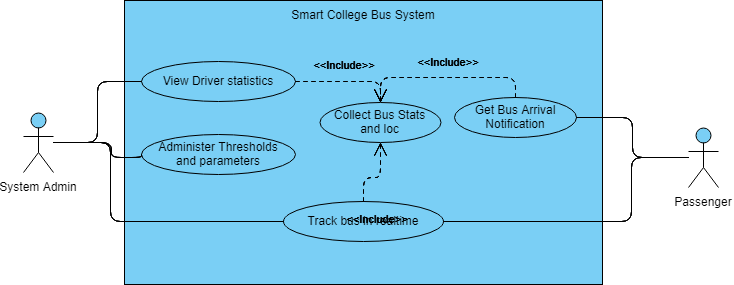


**Figure 1: Results from online survey created via Google Forms**

Before commencing our work, we conducted an online survey using Google forms to gather information on how the product will be received with the aforementioned features. The survey was taken by 144 unique respondents, not including any of the people involved in developing the project.

All poll results clearly show that the majority has “Often” or “Sometimes” faced the ill-effects of the current system. Thus we can conclude that the project will be well received and is of the need in current transport management systems.

**2. SHARED TRANSPORT MANAGEMENT SYSTEM DESIGN**

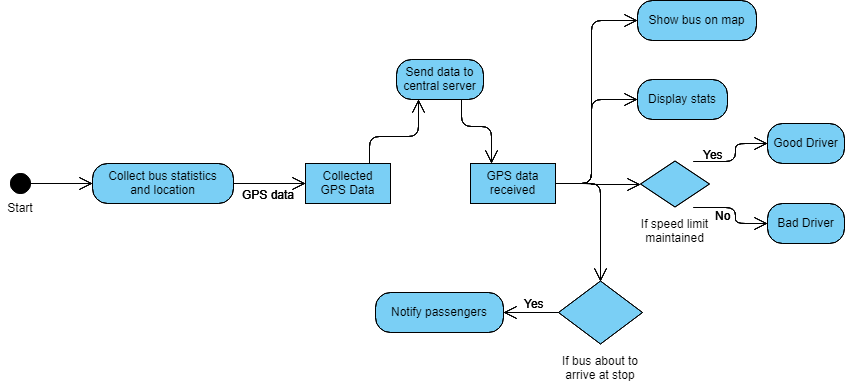
****

**Figure 2. UML Use Case Diagram**

Figure 2 shows the use case diagram made using UML. There are two actors (users) for the system: the administrator and the users of the transport.

The passenger will be able to track the bus’s real time location and will also get notifications regarding the status of the bus. This is done by the system making use of a function to gather the bus’s location and other statistics. The parts pertaining to the location, status and the estimated time of arrival are available to the passenger at the front end.

The administrator can not only view the above mentioned detail, but also gets access to other parameters such as the speed of the vehicle, the accuracy percentage of the data being collected, view in the driver’s profile, which would contain not just the personal and professional information but also the analysis report and performance trend of the driver.

**Figure 3: UML Activity Diagram**

**2.2 HARDWARE ANALYSIS**

***2.2.1 Processor Board***

In order to implement the WoT (Web of Things) based system, a microcontroller/ microprocessor with Wi-Fi is required. The most viable, popular and available options are **Raspberry** Pi and **NodeMCU**. The comparison and final choice are given below:

|  |  |  |
| --- | --- | --- |
| **Feature** | **Raspberry Pi** | **NodeMCU** |
| Image | Image result for raspberry pi | NodeMCU Dev Board v1.0 |
| Cost (in Rs.) | 3605 | 310 |
| Speed | 1.4 GHz | 80 MHz |
| GPIOs | 40 | 16 |
| RAM | 1 Gb | 80 kb |
| Size (in cm) | 8.5 x 5.6 | 4.9 x 2.5 |

From the above minimalistic table, it is clear that the Raspberry Pi is unnecessarily complex for the task at hand. We do not have codes which are too complex or resource demanding. A Raspberry Pi is bulky, consumes too much power, and is costly. A NodeMCU is more than sufficient for the tasks at hand and is available at a lower cost.

**Final Choice:** NodeMCU

The NodeMCU development board consists of an ESP8266 Wi-Fi enabled chip, thus enabling internet access for IoT/WoT cloud-based applications. The board supports various communication protocols like UART, SPI, I2C, etc. which allow interfacing multiple slave devices to the board.

***2.2.2 SENSORS***

***2.2.2.1 GPS Module***

We make use of a GPS sensor module (ublox NEO-6M GY-GPS6MV2) to collect GPS data of the vehicle in real time. This information includes information about the location, such as latitude, longitude, elevation, number of satellites (the more the number of satellites, better the accuracy of the information). GPS receiver module gives output in standard (National Marine Electronics Association) NMEA string format. It provides output serially on Tx pin with default 9600 Baud rate. This NMEA string output from GPS receiver contains different parameters separated by commas like longitude, latitude, altitude, time etc. Each string starts with ‘$’ and ends with carriage return/line feed sequence.



**Figure 4: GPS Module – ublox NEO-6M (GY-GPS6MV2)**

***2.2.2.2 RFID Module***

RFID is an acronym for “radio-frequency identification” and refers to a technology whereby digital data encoded in RFID tags or smart labels (defined below) are captured by a reader via radio waves. RFID is similar to barcoding in that data from a tag or label are captured by a device that stores the data in a database.

An RFID tag will be placed inside every passenger’s institution-issued ID card for attendance, for safety purposes (for educational institutions and companies, rather than for public transport).

****

**Figure 5: RFID Module – RC522**

**2.3 PROGRAMMING LANGUAGES & COMMUNICATION PROTOCOLS**

***2.3.1 Web Server***

An active webserver is required to perform reception of the data from the hardware, execute some algorithms on the data received, and display relevant web pages. The most popular active web server frameworks used by developers are Django and Node.js.

While Django is written and executed in Python, Node.js is a JavaScript runtime built on Chrome's V8 JavaScript engine. V8 compiles and executes JavaScript at lightning speeds mainly due to the fact that V8 compiles JavaScript into native machine code. After a little investigation, it is discovered that the file structure of Django projects is very complex and a lot of scripts and templates have to be written in order to display even simple web pages.

Node.js on the other hand, has a very simple file structure, has simple syntax, and makes web page hosting really simple. As an asynchronous event driven JavaScript runtime, Node is designed to build scalable network applications. Because of the effective use of JavaScript, Node.js also allows developers to create web servers and networking tools. Since both the server-side and client-side code will be written in JavaScript, i.e. it has an effective single codebase, the need for translating has been removed. Additionally, its applications can be run on a variety of servers including Microsoft Windows, Mac OS X, and Unix. Plus, the inbuilt library offers the extended features to applications to act as web servers. Node.js encourages sharing with the presence of the Node Package Manager or NPM. It includes the repository of 50,000 packages, which helps developers to create effective solutions. With the inbuilt NPM, developers can update, share or reuse codes with utmost ease.

Final Choice: Node.js

***2.3.2 Front-end Development***

Vue.js is an open-source JavaScript framework for building user interfaces and single-page applications. Vue.js features an incrementally adoptable architecture that focuses on declarative rendering and component composition. Advanced features required for complex applications such as routing, state management and build tooling are offered via officially maintained supporting libraries and packages. Vue.js extends HTML with HTML attributes called directives. Vue.js directives offer functionality to HTML applications.

Angular.js is another open source framework for building web application front-end and is based on JavaScript. It is maintained by Google developers to address the challenges faced during the development of single page applications. Angular.js is a great option for building long size projects where as Vue,js is ideal for lightweight projects. However, Angular.js is opinionated, meaning that there is a certain way application should be structured. On the other hand, Vue.js is modular and flexible. Angular.js uses two-way binding between scopes, while Vue.js enforces a one-way data flow between components. This makes the flow of data easier to reason about in non-trivial applications. Vue.js has better performance and is much easier to optimize in comparison. In addition, the users need to have an understanding of TypeScript too in order to use Angular.js. Moreover, the learning curve for Vue.js is relatively easier than Angular.js. Vue.js embraces classic web technologies and builds on top of them.

Final Choice: Vue.js

***2.3.3 Mobile App Development***

NativeScript is an Open source framework created by Telerik, a Progress Company for building truly native mobile apps with Angular, Vue.js, TypeScript, or JavaScript. NativeScript enables developers to build native apps for iOS and Android from a single code base, while still enabling the productivity advantages of hybrid approaches. When building the application UI, developers can use libraries which abstract the differences between the native platforms.

If one is developing strictly with the Android SDK or Apple’s Objective-C or Swift, then he will have to learn multiple programming languages which will prove to be exhausting. NativeScript allows the developer to use JavaScript as the core language to build cross platform applications. This is combined with XML and CSS for developing application user interfaces. The use of these web components would need to be combined with a web view. I mean, that is how other mobile frameworks do it. Mobile web views are slow, and worse, they behave differently on different platforms and platform versions. It is difficult to get consistency when using them and results in inconsistent experiences for the users of your applications. NativeScript instead maps the XML layouts and styles at build time to their native counterparts. NativeScript combined with Vue.js, become a fantastic pair for developing immersive mobile experiences. In doing so you can fully reuse skills and code from the web to build beautiful, high performance native mobile apps without web views.

Final Choice: NativeScript

***2.3.4 WiFi (IEEE 802.11)***

IEEE 802.11 is a set of LAN (Local Area Network) protocols that specify the Media Access Control (MAC) and Physical (PHY) layers of the TCP/IP networking architecture for WLAN (Wireless LAN). The Wi-Fi protocol is an implementation of the IEEE 802.11 specifications created by the Wi-Fi Alliance. Wi-Fi is a very popular protocol that is used in most consumer devices such as smartphones and laptops. It is also fast and provides high data bandwidths.

***2.3.5 HTTP***

HTTP (HyperText Transfer Protocol) is the protocol that's used by WWW (World Wide Web) and defines how messages are formatted and transmitted over the internet. This protocol has the following properties:

• Stateless: HTTP messages are executed in a standard manner without any knowledge of previous messages. Hence this protocol is called stateless.

• Client-Server architecture: The HTTP messages are exchanged between two systems where the system which issues the request is known as the client and the system that serves the request is known as the server.

• REST (REpresentational State Transfer): REST is an architectural model for web services. REST promotes interoperability of different web services. RESTful Web services allow the requesting systems to access and manipulate textual representations of Web resources by using a uniform and predefined set of stateless operations.

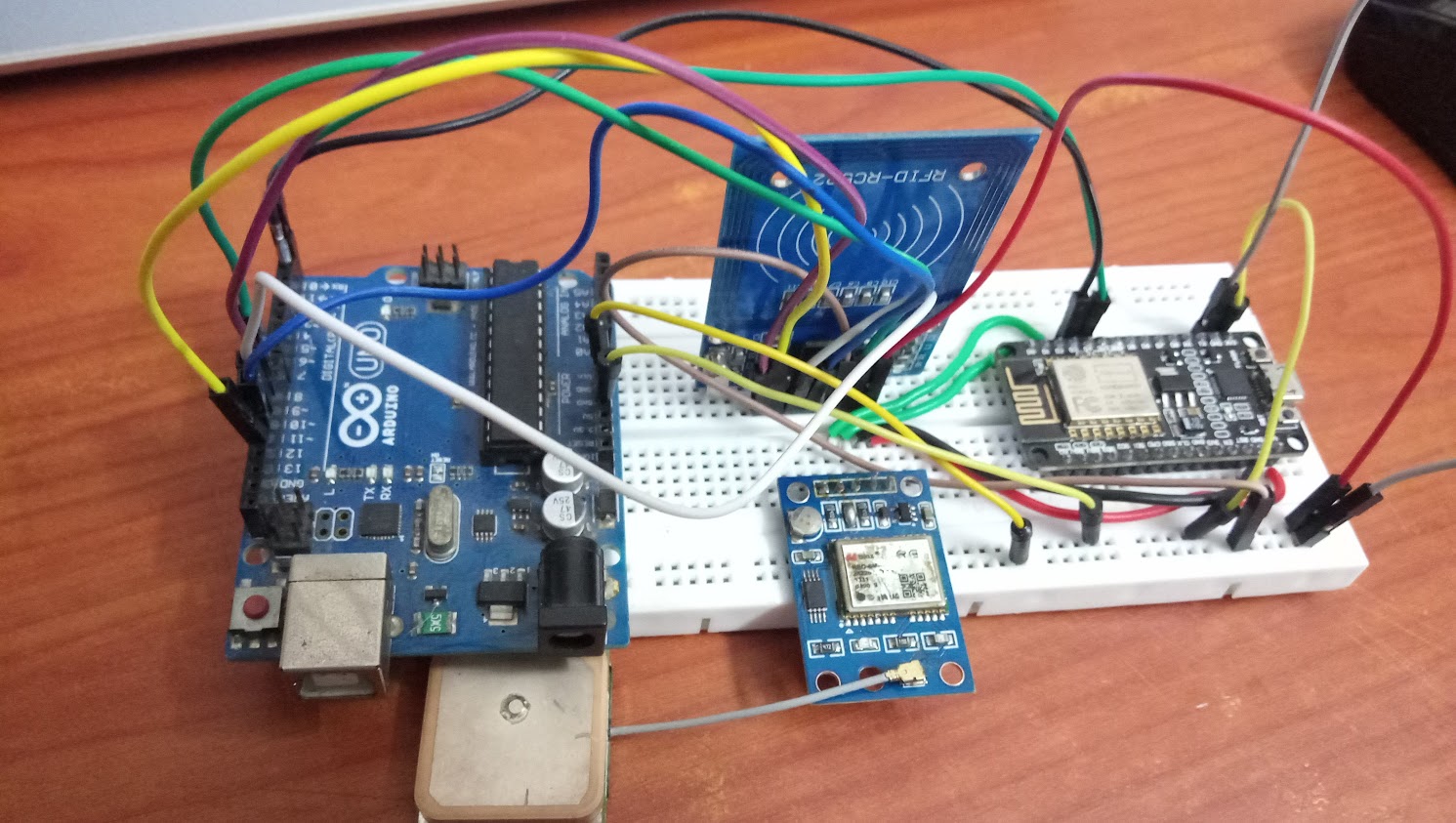
• Self Descriptive: The HTTP messages and responses are self descriptive. The data contained in a HTTP message or response will also contain all relevant metadata. Eg. Status codes.

HTTP was chosen as the networking protocol for this project since it is fast, stateless, highly interoperable due to REST. This provides many advantages such as ease of integration.

1. **SYSTEM IMPLEMENTATION**

**3.1 WORKFLOW**

***3.1.1 On Vehicle***

******

**Figure 6: Hardware setup installed on each vehicle**

The entire hardware set up involves the NodeMCU as a central processor which acts as the master for the whole system.  
The NodeMCU controls the GPS to get information about the current location at regular intervals, processes the raw data and sends it to the backend server hosted as a cloud service via the internet. It does so by making use of WiFi enabled via the ESP8266 on-board.  
The transfer protocol we finally settled upon is HTTP for generating a REST-based application.

The first run was made making use of MQTT

***3.1.2 Server Side***

***3.1.3 Frontend and Mobile App***

**3.2 PROGRAM CODES**

[**https://github.com/Eccentric-Innovators/Shared\_Transport\_Management\_System**](https://github.com/Eccentric-Innovators/Shared_Transport_Management_System)

**Explanation of different code files in the project folder:**

1. **Node:**
   1. **transport\_system:** A system for getting values through MQTT and displaying it in the frontend on the browser**.**
      1. **ex.html:** The frontend HTML page that is displayed to the user which gets the value regularly from the backend and updates it onto the webpage.
      2. **main.js:** The main Node.js file which runs the backend. This is the code that receives MQTT messages and sends them to the frontend.
   2. **post\_http:** A system for getting values through HTTP POST request and displaying in the frontend on the browser.
      1. **node\_modules:** Contains the files for the Node.js modules that are required.
      2. **static:** Contains static files such as HTML and images.
         1. **index.html:** The frontend HTML page that is displayed to the user which gets the value regularly from the backend and updates it onto the webpage.
      3. **index.js:** The main Node.js file which runs the backend. This is the code that receives HTTP POST requests and sends the data to the frontend.
      4. **package.json:** The npm project data file.
   3. **GMaps\_Tracking:** A system for getting values through MQTT and displaying it on Google Maps in the frontend on the browser.
      1. **node\_modules:** Contains the files for the Node.js modules that are required.
      2. **static:** Contains static files such as HTML and images.
         1. **marker\_icon.png:** The icon for the map pin.
         2. **index.html:** The frontend HTML page that is displayed to the user which gets the value regularly from the backend and updates it onto the webpage.
      3. **main.js:** The main Node.js file which runs the backend. This is the code that receives MQTT messages and sends them to the frontend.
   4. **GPS\_Maps:** The final Node.js server project that integrates HTTP requests, GMaps API and the database.
      1. **models:** Contains the schemas for the different databases.
         1. **driver.js:** Contains the schema for drivers.
         2. **passenger.js:** Contains the schema for passengers.
         3. **stats.js:** Contains the schema for the stats collected.
         4. **vehicle.js:** Contains the schema for vehicless.
      2. **node\_modules:** Contains the files for the Node.js modules that are required.
      3. **routes:** Contains the various HTTP routes in order to access the various features of the system.
         1. **driverRouter.js:** The routes on /drivers for manipulating driver data.
         2. **passengerRouter.js:** The routes on /passengers for manipulating passenger data.
         3. **statsRouter.js:** Theroutes on /stats for adding and retrieving stats.
         4. **vehicleRouter.js:** The routes on /vehicle for manipulating vehicle data.
      4. **static:** Contains static files such as HTML and images.
         1. **.well-known:** Contains SSL verification files.
         2. **images:** Contains the required images.
            1. **marker\_icon.png:** The icon for the map pin.
         3. **index.html:** The frontend HTML page that is displayed to the user which gets the value regularly from the backend and updates it onto the webpage.
      5. **views:** HTMLtemplate files for handling errors.
      6. **config.js:** Contains required params for running various services**.**
      7. **index.js:** The main Node.js file which runs the backend, sends the requests to the various routes and manages the databases**.**
2. **Arduino:** Containsthe files pertaining to the Arduino and NodeMCU boards.
   1. **TinyGPS\_OTA\_MQTT\_WiFiMGR:** Code for retrieving the GPS values using TinyGPSPlus library and sending the values through MQTT using PubSubClient library. (NodeMCU)
   2. **TinyGPS\_OTA\_MQTT\_WiFiMGR:** Code for retrieving the GPS values using TinyGPSPlus library and sending the values through HTTP POST request using the inbuilt HTTPClient library. (NodeMCU)
   3. **HTTP\_POST\_test:** Code for checking if the POST messages are being sent from the NodeMCU. (NodeMCU)
   4. **RFID:** Code for getting values from the RFID sensor using Arduino and showing it on the Serial Monitor. (Arduino)
   5. **lib:** Contains required libraries**.**
      1. **pubsubclient:** Library for MQTT on NodeMCU.
      2. **RFID:** Library to get values from the RFID sensor.
      3. **TinyGPSPlus:** Library to get values from the GPS sensor.
      4. **WiFiManager:** Library to help connect NodeMCU to Wifi sources**.**

**4. CONCLUSION AND FUTURE WORK**

**4.1 CONCLUSION**

* Interfacing Google Maps with the backend was successful.
* The system backend is working brilliantly.
* The data collection and analysis are satisfactory

**4.2 FUTURE WORK**

* The front end can be beautified and developed further with a login
* Splitting the system into user and administrator versions distinctly
* Develop a better algorithm once more data is collected