Title: BLE Based Asset Tracking System

Thesis Report

A DISSERTATION SUBMITTED IN PARTIAL FULFILMENT
FOR THE DEGREE OF

Master of Technology

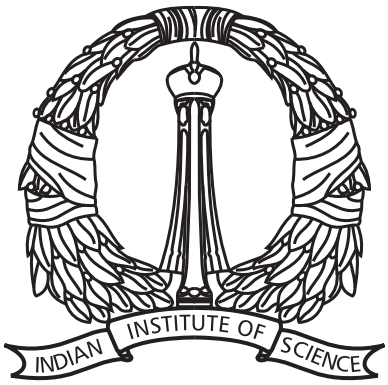
IN THE
FACULTY OF ENGINEERING

BY

SHUBHAM KUMAR SAURAV - 18246
AMIT KUMAR MISTRI - 17937

GUIDED BY

PROF. JOY KURI, PROF HARESH DAGALE



DEPARTMENT OF ELECTRONIC SYSTEMS ENGINEERING
INDIAN INSTITUTE OF SCIENCE, BANGALORE

JUNE 2022

COPYRIGHT © 2022 IISC
ALL RIGHTS RESERVED

Synopsis

There is a high risk of theft and misplacement of the luggage while traveling, and hence there is a need for a system that ensures the security of the assets during travel. The indoor positioning system is helpful in places where other technologies like GPS do not provide much precision. Bluetooth Low Energy can be used for indoor tracking systems as it consumes significantly less energy, is cheap, and is available on every smartphone. The navigation beacon-gateway-cloud system is our choice for indoor tracking and securing solution for our assets. This project aims to develop an asset tracking product using Bluetooth Low Energy technology that keeps track of the asset's presence in a definite safe area and triggers an alarm once the object is out of that zone. We are providing an end-to-end solution for asset security during travel.

Acknowledgments

Our deepest gratitude is to our advisors, Prof.Joy Kuri, and Prof. Haresh Dagale. We have been amazingly fortunate to have advisors who gave us the freedom to explore on our own and, at the same time, the guidance to recover when our steps faltered. Their patience and support helped us to finish this dissertation. We would like to thank both of them for their support. Finally, we would like to thank our family of DESE.

Contents

Chapter 1

Pre-Study Phase

We are focusing on methods to secure our assets indoors and while traveling. We need to develop a product that has a tracker which could be attached to our assets and monitored from a mobile application. The product should keep tracking assets in small intervals of time and update the current status of assets in a cloud database. Finally, in the end, we will be able to make an asset tracking and securing product, both hardware, and software. It will track the asset through our mobile application and tracker attached to our asset and triggers an alarm in case of any attempt of misplacing or theft of the asset. Also, the On-Board-Diagnostic data of the Bus will be stored in the database to provide a monitoring system for the vehicle.

1.1 Background

We need indoor positioning methods to detect the presence of assets in a safe area around 10 meters radius. We are not looking for very accurate positioning of assets. We are satisfied even if accuracy is in the range of five to ten meters. To achieve this, we can look at the connection status of tags and gateways and detect the presence of our assets.

1.2 Functional aspects

- We use the asset tracking system for safety of our assets.
- There are various methods to implement this system like WiFi, BLE, and Ultra wide band.

- We can use the BLE connection-disconnection status to detect the presence of the assets.

1.3 Characteristics and performance

We note the following points regarding performance:

- The product checks the connection status of Tags attached to Assets with the Gateway provided in the Bus to detect its presence.
- Performance requirement includes low power consumption and moderate location accuracy.
- Cloud Database service is used to store the information of the Assets, users, conductors, and the Bus.

1.4 User aspects

The following points should be noted

- Mobile apps should be user-friendly and easy to operate for even elderly people.
- There is no need for interaction of the user with beacons or trackers.
- Only one-time installation of tracker and beacons is required, which should be a quick process of easier steps.
- Application size should be small and quickly installable on mobile phones.

1.5 Environment aspects

The following are important points:

- From an environmental point of view, the product can be considered a consumer electronic.
- Vibration may cause the connectors to become loose.

1.6 Power supply

The following are important points:

- The product contains a battery-operated tracker to be attached to assets and application to be installed in mobile phones.

1.7 Safety

The following points are important

- Tracker should be fireproof and should not heat up or cause a fire in any case.
- It should not have sharp edges, which could harm the user's body.
- It should be human operation friendly.

1.8 Reliability

The following points are important:

- Warranty/guarantee will depend on the component manufacturer.

1.9 Product/market Survey

- The following table Tab. 1.1 compares the two available products:

Features	**MOKOSmart H1 Key-chain Beacon**	**MOKOSmart W1 Wear-able Beacon**
Size	32x32x6 mm	244x38x10 mm
Battery model	CR2032	CR2032
Battery capacity	240 mAh	240 mAh
Battery life	16 months	12 months
Battery Replacable	Yes	Yes
Max Range	60m(+4dBm)	30m(+4dBm)
Waterproof	Yes	Yes
Protocols	iBeacon and Eddys-tone:UID,URL,TLM	iBeacon and Eddys-tone:UID,URL,TLM
Firmware updating	OTA	OTA
Built in sensor	Motion LIS3DH	Motion LIS3DH
Certification	CC&FCC	CC&FCC
Additional tech	RFID,RGB	RFID,RGB

Table 1.1: Product Comparison

1.10 User Survey

The following points are noted:

- The product is intended to be used for securing baggage or any other asset, particularly while traveling.
- Even non-technical users with basic knowledge of smartphones should benefit from the product.
- The opinion of people is that the accuracy of location will benefit tracking.
- One of the hard-to-meet requirements is to make a quick and accurate decision on the current location of the object.

1.11 Wish

We are considering a scenario where the product will be used on intercity buses. Typically this travel can last for a day or two. However, charging or replacing batteries after each journey is not convenient. Therefore, generally, it should last for a year if that is possible.

Chapter 2

Study Phase

In this chapter, we discuss possible approaches to the asset tracking problem and the supporting technologies that can be used. We survey products available in the market. Further, we present the state of the art that researchers have published on a similar topic. Indoor location tracking can be done by using various approaches like Bluetooth low energy and UWB.

2.1 Ultra wide band

2.1.1 Introduction

Ultra-wideband is a technology for the transmission of information over vast bandwidth (500 MHZ or greater). Ultra-wideband (UWB) technology has the advantage of high precision distance measurements. UWB transmissions are different from radio transmission in the way that they transmit information by generating radio signals at specific intervals of time.

2.1.2 Working

According to the IEEE paper mentioned in the reference section [1] , we can use the UWB method for asset tracking. The UWB pulse (center and right image) is only two nanoseconds (ns) wide. This protects the signal from interference with the reflected signals. Also, these have much faster rise and fall edges than the narrowband signals. According to the authors, the UWB allows precise determination of arrival time and distance. Based on the time difference between arrivals of wave-front in different antennas, we can measure phase difference at

different antennas. By measuring phase differences at different antennas, we can measure the angle at which the wave is coming. This method gives more accurate results compared to time difference measurements.

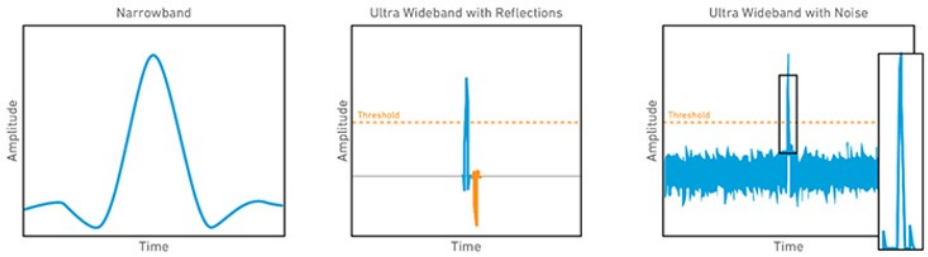


Figure 2.1: Comparing narrow-band, Impulse radio with direct(blue) and reflected signal(orange)

2.1.3 Advantages

According to the IEEE paper mentioned in the reference on [2], The precision-accurate capabilities of UWB are what sets it apart from other location solutions. Location readings must be very precise, and the estimated area of the Asset must be as small as possible. This technology helps to develop an accurate indoor positioning system. It must be reliable even in uncertain conditions to ensure the security of Assets. Also, it must consume low power and should be affordable so that it can be embedded with smartphones and tags. The advantages of UWB are as follows:

- **Ultra Accurate**: It provides centimeter accuracy.
- **Ultra-Relaible**: It has high immunity to multipath and interference.
- **Real Time**: It has very low latency of less than 50ms.
- **Secure**: Since the transmission occurs at specific intervals, it prevents from rebroadcasting of the information and hence provides security.

2.2 Bluetooth Low Energy

2.2.1 Introduction

Bluetooth Low Energy (BLE) is a low-power wireless communication technology that allows us to communicate over short distances. Some of those devices are smartphones, smartwatches, fitness trackers, wireless headphones, and computers that are using BLE to create a seamless experience between our devices.

2.2.2 Working

A BLE beacon acts as a server and keeps on broadcasting signals which can be read by a BLE client at distances ranging from 0 to 300 feet. The technology is optimized to conserve battery life, so beacons can last for around 3 years. The Beacons use common cellphone batteries for power – which are inexpensive and easy to replace.This technology can be used indoors or outdoors.

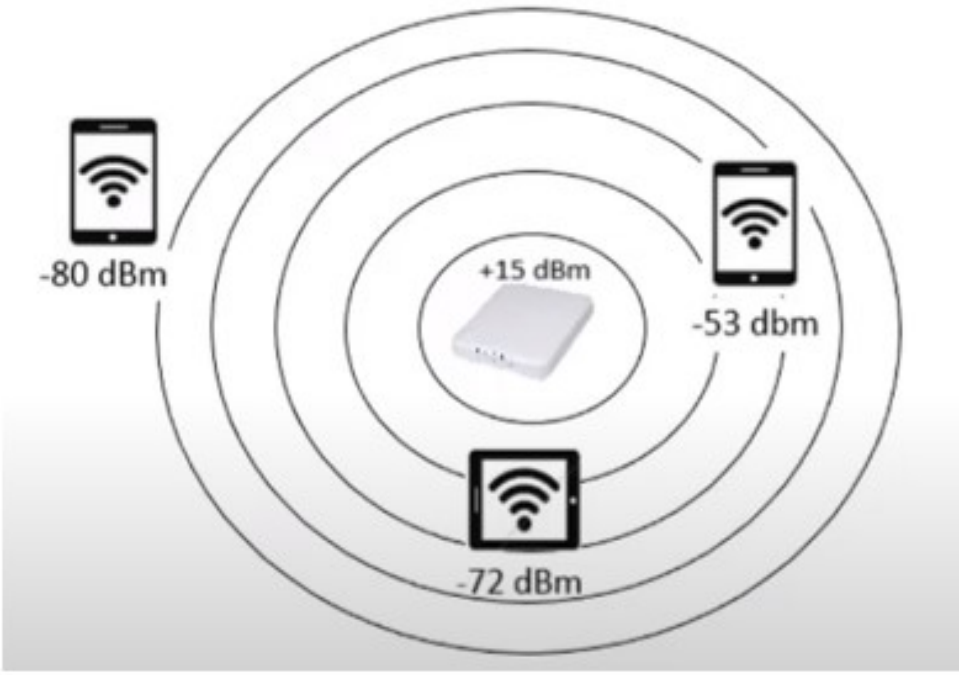


Figure 2.2: Received signal power(decibel-milliwatts) dependency on distance

The beacons transmit a string of data that contains MAC ID, RSSI value, and the transmission power of beacons. RSSI (Received signal strength index) value depends on the power received

by the receiver, which again depends on the distance from the sender. Therefore, RSSI value can be used to calculate the distance from the sender. Using three different beacons and calculating the distance of the Asset from it, we will calculate the exact location of the asset. The distance is calculated from the RSSI value using linear regression. After the calculation of distance, we use hybrid trilateration to calculate the exact coordinates of the Asset, as shown in the figure Fig. 2.3. Our asset is located at a marked star position, which is the intersection of the three loci drawn from beacons.

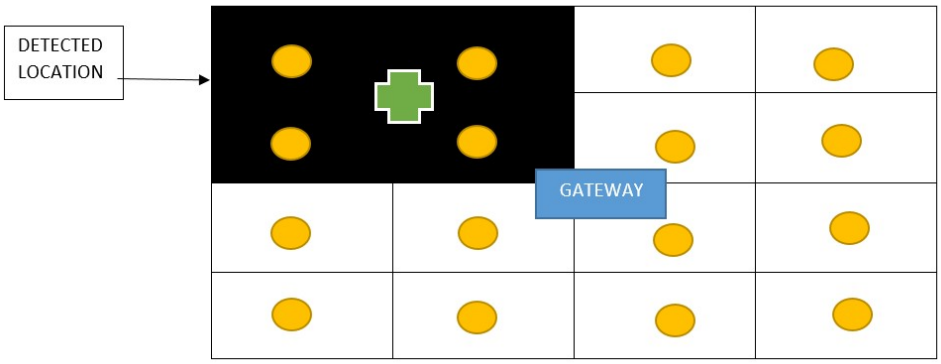


Figure 2.3: Asset tracking using RSSI value

$$Distance = 10^{(\frac{MeasuredPower - RSSI}{10N})} \tag{2.1}$$

2.2.3 Advantages of BLE

- Consumes low power compared to other technologies.
- Lower cost of modules and chipsets, even when compared to other similar technologies.
- Most importantly, its existence in most smartphones in the market.

2.3 Comparision between BLE and UWB

The following table Tab.2.1 shows the comparision between BLE and UWB technology.

Features	**BLE**	**UWB**
Location Accuracy	Less than 5m	Less than 50cm
Range	Optimal 0-25m upto 100m	Optimal 0-50m upto 200m
Latency	Typically 3-5 sec to get the location	less than 1ms to get location
Nominal Tx Power	0-10dbm	-41.3dbm/MHz
Cost	Low	High
Frequency	2.4GHz	3.1-10.GHz
Data Rate	upto 2Mbps	upto 27Mbps
LifeTime	More than 3 years	Less than 3 years

Table 2.1: Technology Comparison

2.4 Methodology

2-d tracking

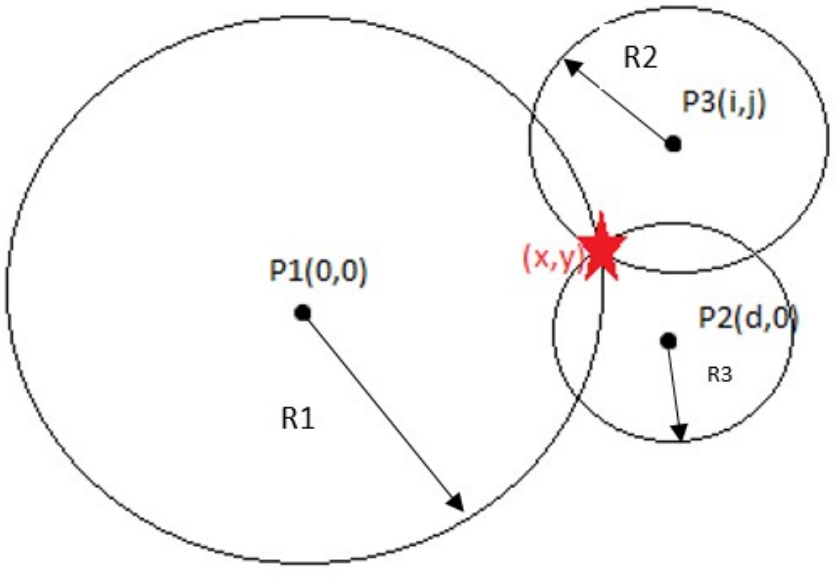


Figure 2.4: Asset tracking using RSSI value

From figure Fig. 2.4 shown we get following equations using locus of circle.

$$r_1^2 = x^2 + y^2 \tag{2.2}$$

$$r_2^2 = (x-d)^2 + y^2 \tag{2.3}$$

$$r_3^2 = (x-i)^2 + (y-j)^2 \tag{2.4}$$

On solving these three equation, we get the co-ordinates of our asset as follows:

$$x = \frac{r_1^2 - r_2^2 + d^2}{2d} \tag{2.5}$$

$$y = \frac{r_1^2 - r_3^2 + i^2 + j^2}{2j - 2i} \tag{2.6}$$

3-d tracking

In 3-d location, if tracker and beacon are not located at same height then one extra z co-ordinate comes in picture. We need to consider spherical locus.

From figure Fig. 2.4 shown we get following equations using locus of circle.

$$r_1^2 = x^2 + y^2 + z^2 \tag{2.7}$$

$$r_2^2 = (x-d)^2 + y^2 + z^2 \tag{2.8}$$

$$r_3^2 = (x-i)^2 + (y-j)^2 + z^2 \tag{2.9}$$

On solving these three equation, we get the co-ordinates of our asset as follows:

$$x = \frac{r_1^2 - r_2^2 + d^2}{2d} \tag{2.10}$$

$$y = \frac{1}{j}(1 - \frac{r_3^2 - r_1^2}{i(i-2x)}) \tag{2.11}$$

2.5 MongoDB

2.5.1 Introduction

MongoDB is a document based non-SQL database. Each entry in MongoDB is a document consisting of key-value pairs. Different documents can have different number of key-value pairs. We have a cluster which contains several databases, inside each database we have different collections. Each collection consists of a particular type of documents. Each database consists of collections corresponding to a particular event.

Document based database systems provides following advantages:

- The documents has entries in format of native datatypes in many programming languages, e.g. string, int etc.
- It's dynamic schema supports fluent polymorphism as different document entries are allowed to have different key-value pairs.

In SQL database we have to design series of tables consisting of rows and columns, and at each intersection we provide a different entry.

2.5.2 MongoDB vs Relational Database

In relational databases, the information is stored in a tabular format where each row-column intersection corresponds to a single entry, whereas, in non-relational databases like MongoDB, the documents are structured in a **key**:**value** pair format where inside a value we get the flexibility to add sub-documents or enter an array of values.
Following images in figure Fig. 2.5 and Fig. 2.6 shows sample documents stored in MongoDB:

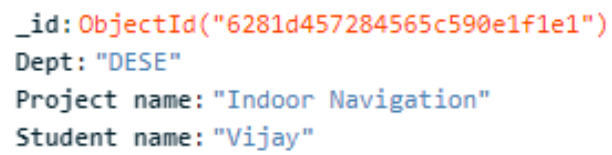


Figure 2.5: Sample document for single value entry in MongoDB

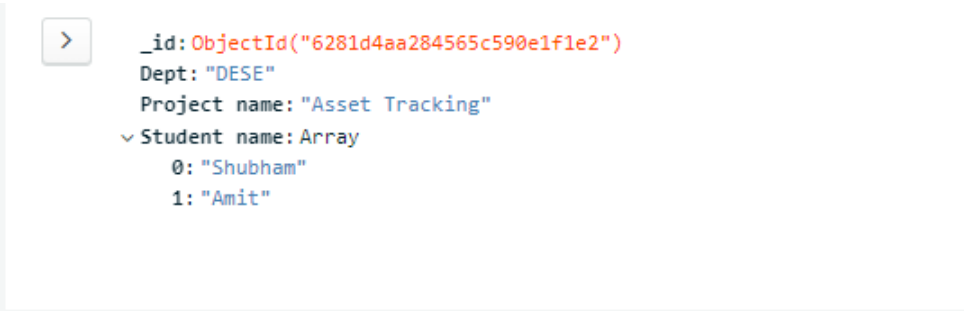


Figure 2.6: Sample document for array value entry in MongoDB

Hence from figure Fig. 2.6 we can see that MongoDB allows us to add an array in a single value entry in a document. This feature is not available in Relational databases like MySQL. To add an entry similar to document in figure Fig. 2.6 in Relational database, there can be anyone of the following tables Fig. 2.7 and Fig. 2.8:

Id	Dept.	Name	Student	Guide
001	DESE	Asset Tracking	Shubham,Amit	Haresh Dagale, Joy Kuri
002	DESE	Indoor Navigation	Vijay	Haresh Dagale

Figure 2.7: Sample Table 1 for array value entry

Now in the first table shown in figure Fig. 2.7 the name of various students are seperated by a comma. If we store our data in this format, if we need to run a query for the name "Shubham," we will have to use the **LIKE** clause in our query, which will need a full table scan. Also, updates and deletion in this kind of table are very complex as we need to match which student we need to replace or both of them. Hence it increases our cost and time to perform query in case of Table 1 kind of entry.

Id	Dept.	Name	Student
001	DESE	Asset Tracking	Amit
001	DESE	Asset Tracking	Shubham
002	DESE	Indoor Navigation	Vijay

Figure 2.8: Sample Table 2 for array value entry

Now if we use the second kind of table as shown in figure Fig. 2.8, we have to make multiple entries with same id for different Student names. Now in case of query, it will fetch all the entries having same project name and we will have to use **JOIN** command to join them.
In a magnetic disk, it takes around 5ms to seek to a memory location after which reading speed is fast around 40-80 MBps. So seek takes around 99 percent of time of query. In case of multiple entries in Table we will need to seek multiple times which increases our query time, whereas in case of MongoDB same entry is made as an array and the information is stored in a sequential format. So, we need to seek a single time and goon reading sequentially in case of MongoDB and therefore it reduces our query time.
Also in case of Relational databases the entries in each row are made one by one corresponding to each column. But in case of MongoDB the entire entry is inserted as an isolated document at once. Therefore in case of MongoDB we do not encounter a partial failure while inserting a new entry whereas it may happen that in case of Relational database some entries in a row may get corrupted.

2.5.3 Advantages of MongoDB

- **Atomicity and Isolation** : Each insertion is an seperate document and there is no chance of partial success in case of MongoDB. Either the complete information is uploaded successfully of it is a complete failure
- **Data Locality** : We have the option to add entries in an array or sub-document format which ensures that all the entries are stored sequentially and which in turn saves our time during a query.
- **Full Cloud based application data platform** : Along with database MongoDB provides us Realm and Chart Services to integrate our apps with our database. Also it is the only globally distributed multi-cloud database, i.e. it allows to enable applications that uses two or more cloud services at the same time.
- **Flexibility in documents** : There is no condition to have similar entries in each documents, we have the flexibilty to add or remove fields in different entries, i.e. different documents may have different keys.
- **High Performance(speed)** : Values corresponding to a key are allowed to have a sub-document or array which increases the query time. Also we get functionalities like **insertMany**, **updateMany** which offers a significant performance boost.
- **Cost Effective** : We can set our cluster to automatically scale when needed and keep our costs minimum. For lower usage cases we can deploy a free cluster where we will be charged only on the actual usage.

2.6 MIT App Inventor

MIT App Inventor is an App-development platform for Android and IOS applications. It is based on coding with blocks which saves our time for typing. It is a powerful platform with lots of extensions available on the internet, which reduces our efforts.

Advantages of MIT App Inventor:

- Everything is done in a select and drop manner. This means we can select a particular chunk of code and drop it in our code. Hence, no typing.

- Easy to test our app. With the help of the AI companion, we can visualize the current changes in the app on our mobile phones.
- Lots of resources are available on internet to learn to build apps on this platform.
- Power of native apps with a simple UI.

2.7 On Board Diagnostic(OBD)

2.7.1 Introduction

Figure 2.9: OBD Connector in Vehicle

On Board Diagnostic (OBD) is a system provided in vehicles for self diagnostic and reporting capability. There are various sensors provided inside the vehicle to monitor the engine parameters like Coolant Temperature, Air Pressure etc. The engine control unit depends on the readings of these sensors to perform various tasks like calculating the accurate air-fuel mixture etc. It is very necessary to ensure that the sensors are in a good condition and the readings are in safe limits to get a good riding experience and for better vehicle maintenance.There is a OBD connector provided below the steering wheel in any vehicle as shown in figure Fig. 2.9. This connector helps us to communicate to the ECU to get values of OBD parameters.

2.7.2 Communication Protocol

The communication with this OBD connector is done by following Controller Area Network(CAN) Bus Protocol. The CAN bus is a broadcast type of bus where all nodes can hear all transmissions. It is based on request response type communication. In our case we make a request for an OBD Parameter information and in response we get the value stored in Engine control unit. Each CAN Bus frame for request or response contains a 11 bit identifier followed by 8 bytes of data as shown in figure Fig. 2.10.

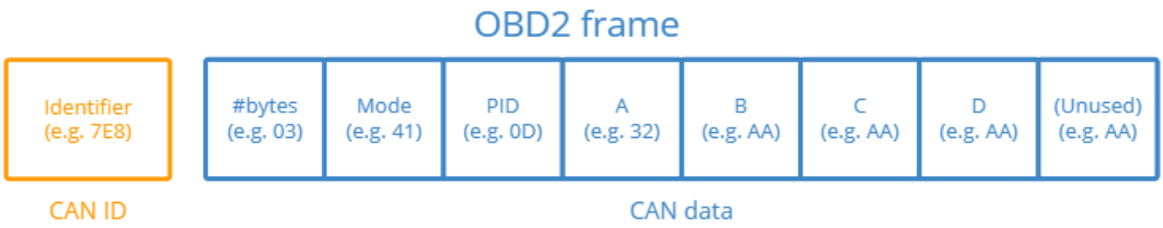


Figure 2.10: CAN frame for OBD

- **Identifier**:For OBD2 messages, the identifier is standard 11-bit and used to distinguish between request messages (ID 7DF) and response messages (ID 7E8 to 7EF).
- **Length**: The first byte in the OBD Data frame corresponds to the number of bytes of remaining data in frame.
- **Mode**: The second byte in OBD Data frame contains the information of the service being used. There are ten OBD diagnostic modes or services in OBD2 Standard.
  - **01**: Show current data
  - **02**: Show freeze frame data
  - **03**: Show stored Diagnostic Trouble Codes(DTC)
  - **04**: Clear Diagnostic Trouble codes and stored values
  - **05**: Test results for Oxygen sensors(Non CAN only)
  - **06**: Test results for system monitoring and oxygen sensors for CAN
  - **07**: Show pending DTCs
  - **08**: Control operations of On-Board System
  - **09**: Request Vehicle Information

  - **10**: Peramanent DTCs

- **PID**: For each mode, a list of standard OBD2 Parameter IDs(PIDs) exists for various parameters. Following are the list of some standard PIDs in hexadecimal notation:

  - **Distance Travelled**: 31
  - **Vehicle Speed**: 0D
  - **Engine Coolant Temperature**: 05
  - **Engine Run Time**: 1F
  - **Throttle Position**: 11
  - **Intake Air Temperature**: 0F
  - **Mass Air Flow**: 10
  - **Manifold Absolute Pressure**: 0B
  - **Fuel Tank Level**: 2F
  - **Engine Speed**: 0C

- **A,B,C,D** : These Data bytes are the used to calculate the value of the corresponding OBD parameter using standard formula. Following are the formulas for calculating the given PIDs:

  - **Distance Travelled**: 256A + B
  - **Vehicle Speed**: A
  - **Engine Coolant Temperature**: A - 40
  - **Engine Run Time**: 256A +B
  - **Throttle Position**: 100A/255
  - **Intake Air Temperature**: A-40
  - **Mass Air Flow**: (256A + B)/100
  - **Manifold Absolute Pressure**: A
  - **Fuel Tank Level**: 100A/255
  - **Engine Speed**: (256A + B)/4

2.7.3 OBD Parameters

The following are the OBD parameters we are interested in our project.

2.7.3.1 Coolant Temperature

Engine Coolant helps in protecting the engine from overheating.The antifreeze present in the coolant helps in keeping the temperature normal and prevents it from freezing when the weather is cold. It also raises the boiling temperature of the coolant for averting overheating when the weather is hot, as well as fights corrosion. Failure of Engine Coolant can lead to cause damage in engine and other components of vehicle. The average engine operating temperature lies between 75 to 105 degrees Celsius. Lower Temperature causes more fuel consumption whereas higher temperature causes engine damage . When the temperature reaches 80 to 90 degrees Celsius, the thermostat lets the coolant flow through and start reducing the heat generated.
At the time of failure Engine Coolant Temperature(ECT) sensor output go more than 4.91 V or less than -40°C.

2.7.3.2 Intake Air Temperature

The intake air temperature (IAT) sensor is used by the Powertrain control module (PCM) to determine the temperature and therefore the density of air entering the engine. Typically, the PCM sends a 5-volt reference to the IAT sensor. The IAT sensor then varies its internal resistance according to air temperature and sends a return signal back to the PCM. The PCM then uses this formation to determine fuel injector control and other outputs. Colder temperatures require more fuel, which the engine control module is programmed to calculate. If the intake temperature sensor makes the engine control unit inject the wrong amount of fuel then it may cause problem in starting the vehicle. The engine control unit relies on the IAT sensor's information, and if a false signal is sent, the fuel efficiency decreases or increases significantly.
The temperature registered by IAT Sensor must be in range ±10°C of ambient temperature. In case of any failure in IAT Sensor the registered value is either negative or more than 300°C. If the IAT sensor shows this faulty results then it must be repaired or replaced in order to ensure better fuel consumption and engine load.

2.7.3.3 Manifold Absolute Pressure

The MAP sensor measures the absolute pressure inside the intake manifold of the engine. At sea level, atmospheric pressure is about 101.325 KPa. When the engine is off, the absolute pressure inside the intake equals atmospheric pressure, so the MAP will indicate about 101.325 KPa . With a running engine, intake manifold vacuum usually runs around 60KPa to 67 KPa. If the reading from sensor goes far beyond this range, it indicates a faulty sensor.

- A high MAP value indicates high engine load and hence high fuel consumption. It also increases the amount of hydrocarbon and carbon monoxide emissions from the vehicle. Hydrocarbons and carbon monoxide are some of the chemical components of smog.
- A low MAP value indicates low engine load and hence low fuel consumption. This results in decrease in power of engine. Also decrease in fuel injection increases the combusion chamber temperature resulting an increase in the amount of nitrogen oxides (a chemical component of smog) production within the engine.
- A bad MAP sensor results in failure of emission test of vehicle.

2.7.3.4 Mass Air Flow

The mass airflow sensor measures the amount of air passing through air filter and entering the engine. For an engine to perform properly, it needs a specific amount of air-fuel mixture. A bad estimation of this mixture leads to decreased fuel economy, power and more. A dirty air filter makes the MAF sensor dirty leading to false readings. If the vacuum hoses are leaking, it will give the same error code as mass airflow sensor failure. When the MAF sensor stops working, the engine control unit is not able to calculate correct value of air-fuel mixture. This causes partial and sometimes permanent failure of engines. It also affects overall mileage . With the engine at idle, the MAF's PID value should read anywhere from 2 to 7 grams/second (g/s) and rise to between 15 to 25 g/s at 2500 rpm.

2.7.3.5 Vehicle Speed

This parameter shows the speed with which the vehicle moves. This will be helpful in checking the driving behaviour of the driver. So, that in case of over speeding necessary actions can be taken by the bus company.

2.7.3.6 Engine Speed

This parameter shows the Engine speed in RPM. This will be helpful in reading it's effect on other paramters like Fuel consumption etc.

2.7.3.7 Fuel Tank Level

This parameter shows the level of fuel present in the vehicle. If there is a sudden rise in fuel it indicates that the fuel has been filled, whereas, a sudden decrease in it's value implies that there is a leakage in fuel tank or the fuel has been taken out by some unfair means.

2.7.3.8 Engine Run Time

This parameter shows the total run time of engine after last start.

2.7.3.9 Distance Travelled

This parameter shows the total Distance Travelled by the bus after last reset. It will be helpful to bus company for deciding the servicing time for the Bus. Also it will be useful to calculate the mileage and keep record of the bus behaviour.

2.8 Conclusion

- From reading the material and papers, we have built a robust understanding of asset tracking methods.
- We learned about various methods of asset tracking.
- We have done calculations for accurate tracking of our Assets using RSSI value.
- We need a product that is able to detect the presence of assets in the range of 10 meters with low energy consumption and at a lower price.
- UWB provides accuracy much higher than needed in this project resulting in a very high cost. Also, we need mobile phones for tracking. But only a few mobile phones come with UWB technology in it.
- BLE provides accuracy in a range that is acceptable for this project at a very lower price of around Rs 600 to Rs 1000. Also, every smartphone has Bluetooth in it.
- Therefore, we chose the Bluetooth Low Energy approach as a solution to our asset tracking problem.

- We have selected MongoDB as our Cloud Database because of it's data locality, flexibilty and high performance qualities.
- We have selected MIT App Inventor as our App development platform as it is powerful and easy to understand platform.
- This system will have very simple hardware with very easy and user-friendly application software and monitoring system with data storage in cloud.

Chapter 3

Design Phase

In the design phase, we will discuss the architecture of our model, various components used, and their application.

3.1 Architecture

The Architecture of our model consists of majorly five components: Tags, Gateway, Master Gateway, Cloud Database, and Android apps. These are merged to form a complete Asset Securing system. The Architecture diagram is shown in figure Fig. 3.1. All the one-way communications are shown using single-headed arrows and two-way communications are shown by double-headed arrows. The mode of communication like UART, CAN, SMS ,etc. are shown in black text and the data being shared are shown in red text. The gateway handles the asset securing tasks and the master gateway helps in the health management of the vehicle and in the OTP verification process.

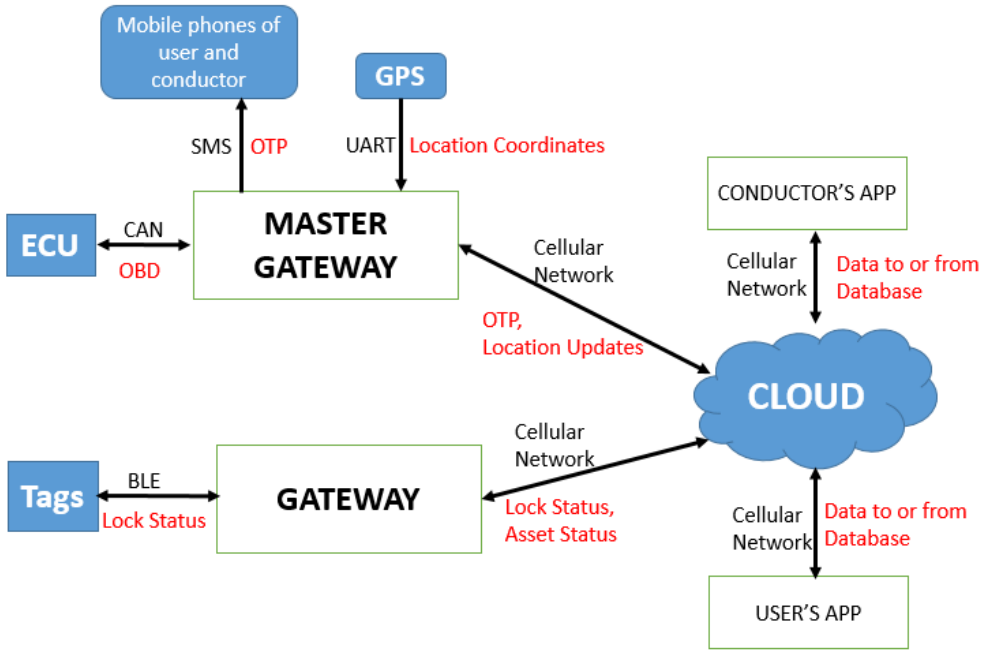


Figure 3.1: Architecture of BLE based Asset Tracking System

3.2 Hardware Design

3.2.1 Tags

The tags are kind of locks to be provided to the passengers when boarding the bus. These tags will be attached to the assets of the passengers. Passengers will have to make an entry in the database using their mobile app, where they need to scan the QR code provided above the Tags. Anyone can install the mobile application, but for successful addition of entry in the database, one must scan a valid QR code above Tag. Each Tag has a different Tag id which will be provided to the customers after successfully scanning Tag QR.

The job of Tag is to communicate with Gateway present in Bus via BLE continuously. Whenever the Asset is taken far away from the Gateway without unlocking TAG, then TAG gets disconnected from the Gateway. In that situation, the Gateway updates the insecurity status in the database and activates the Alarming System.

When deboarding the bus, the passenger will need to remove the entry from the database with the TAG ID attached to their assets to unlock the Tag from their luggage.

We use the ESP32 board to work as BLE Beacon and design the lock system with a Servo motor connected to it. The ESP32 board is programmed to continuously communicate with the

Gateway over BLE, and on receiving the LOCK/UNLOCK command from Gateway, it performs the locking mechanism accordingly. The final structure of TAG will look like a general lock, and the ESP32 and servo motor will be embedded inside the lock's body. The locking will be done digitally using the mobile app.

The passenger gets a Tag id and sets a password at the time of making an entry. The same can be used as login credentials and will be required for the removal of entry and unlocking of TAG. Once the entry is removed from the database, the user will be able to remove the TAG from their Asset. The following figure Fig. 3.2

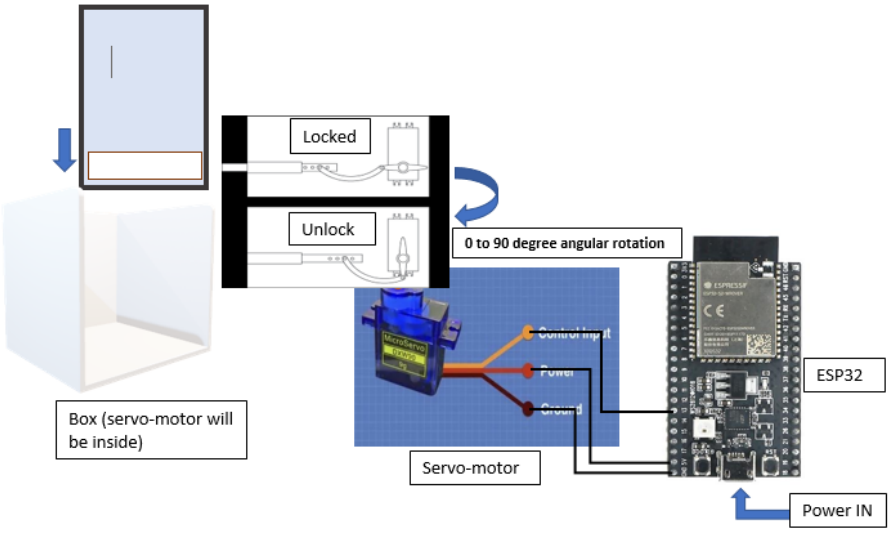


Figure 3.2: Block diagram of TAG

3.2.2 Gateway

BLE Gateway will be present in the Bus, whose task is to update the Asset Status of all the TAGS in the Cloud database. The Gateway communicates with the TAGS over BLE. Whenever it finds any TAG disconnected, which means the TAG is taken away from the safe zone of Gateway, then it updates the Asset Status as **Unsafe** otherwise, it keeps it as **Safe**.

The Gateway will be fixed in the Bus and provided internet connection to communicate with the database. Also, whenever an entry is removed from the database, the Gateway's job is to send UNLOCK command to the corresponding TAG. The circuit diagram of the Gateway is shown in figure Fig. 3.3

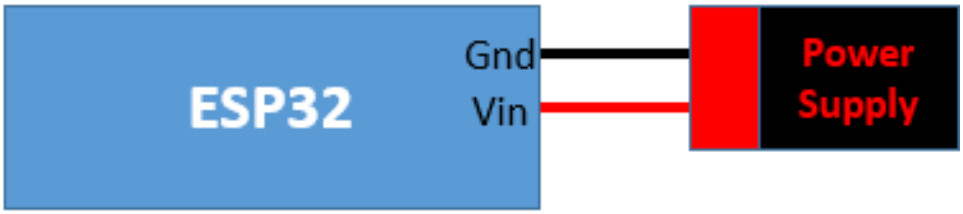


Figure 3.3: Circuit Diagram of BLE Gateway

3.2.2.1 Master Gateway

There is a Master Gateway which is provided with GSM module, GPS Module and OBD connector. It performs the following tasks in loop:

- **OTP Task**: At the time of new registration and password reset, an One Time Password (OTP) is required for verification. These OTPs along with the corresponding mobile number are stored in the database during verification process. The job of master gateway is to find all the pending OTPs one by one and send them to the dedicated mobile number using the GSM module.
- **OBD Task**: The master gateway keeps requesting for all the OBD parameters to the Engine Control Unit (ECU) and after collecting the data, it makes an entry in the database with all the parameters and their corresponding values. These values help us to analyse the performance and health condition of bus.
- **Location Update**: The Global Positioning System (GPS) Module provided to master gateway helps in getting the live updates of location coordinates. These location coordinates are used to create a Google Map URL pointing to that specific location. These location coordinates and the Map URL are stored in the database in specific intervals. This entries helps in re-tracking of the route taken by the bus.

The circuit diagram of the Master Gateway is shown in figure Fig. 3.4

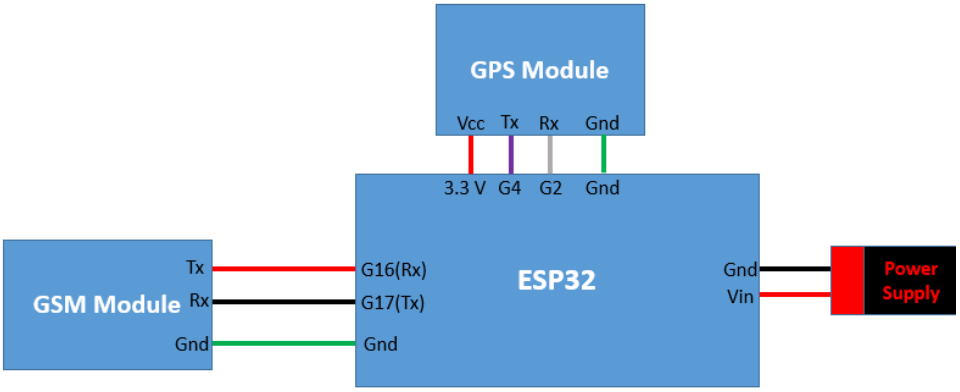


Figure 3.4: Circuit Diagram of Master Gateway

3.3 Database Design

3.3.1 Cloud Database

Cloud Database is used to store Employee and Passenger information. The status of Assets is also updated here. All the status checks are performed in Gateway, and accordingly, the data in the database is updated.

At the time of boarding the Bus, the user makes the entry of TAG in the database. Some additional information like Name, Source, Destination, password, Contact, etc., are stored along with TAG ID. For connecting the app to the database, the TAG ID and password are used as login credentials. Only when the entered credentials match with the credentials stored in the Database login is allowed. After login, both the apps continuously monitor the Status of the TAG. The setup is such that whenever the TAGS are connected to the Gateway, the document corresponding to that TAG in the database is marked with **Safe** in the Asset Status field.

We are using MongoDB Atlas as our Cloud Database. All the informations related to user and employee are stored in the database **ASSET INFO**. The employee informations are stored in the collection named **Employee Data** and the user informations are stored in the collection named **Luggage data** , the informations regarding the locks are stored in the collection named **Lock Status**, and the informations regarding OTP are stored in the collection named **OTP Data**. All the informations regarding the Bus and travel are stored in the database named **BUS INFO**. The OBD data is stored in the collection named **Bus Data** and the location informations are stored in the collection named **Bus Location Data**. For each new entry, a separate document is inserted. Following diagram in figure Fig. 3.5 shows the data arrangement in MongoDB.

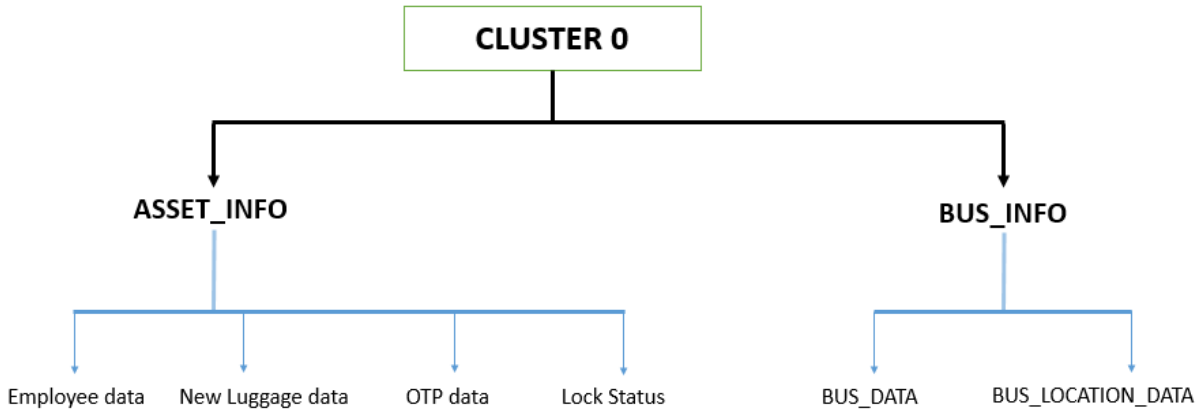


Figure 3.5: Arrangement of data in MongoDB

3.3.2 Collection for Employee Data

The following figure Fig. 3.6 shows a sample document stored in **Employee data**. Each document in the collection contains five fields.

```
_id: ObjectId("621dc641283bb1c2358e0e68")
Employee_id: "Emp_58"
Name: "amit"
Contact: "7988639512"
Address: "mrigasira"
Password: "eWAYckJiF3lpPeVaJKF41r/uJmFYC4bURoRF56AMBTJ688RI7i4JDk3QcBQMqYshbgTHJX..."
ts: 2022-03-01T07:07:45.984+00:00
```

Figure 3.6: Sample document in Employee data

- **Emp id** : Each employee has a unique employee id which is provided to them at the time of registration
- **Name** : It contains the name of employee
- **Contact** : It contains the contact number of the employee, which is verified by OTP at the time of registration.
- **Address** : It contains the address of the employee
- **Password** : It contains the encrypted form of the password set by the Employee at the time of registration. This password can be decrypted only by a private key provided in the app. Encryption and decryption is done by RSA (Rivest-Shamir-Adleman encryption) Algorithm, which is an asymmetric encryption technique.

3.3.3 Collection for Luggage Data

Following figure Fig. 3.7 shows a sample document stored in **Luggage Data**. Each document in the collection contains seven fields

Figure 3.7: Sample document in Luggage data

- **Tag id** : Each Tag has a unique Tag id which is added to database by scanning QR code provided in Tag at the time of boarding the bus
- **Name** : It contains the name of Passenger
- **Contact** : It contains the contact number of passenger which is verified by OTP at the time of registration
- **Source** : It contains the Source station of passenger
- **Destination** : It contains the Destination station of passenger
- **Password** : It contains the encrypted form of the password set by passenger at the time of registration. This password can be decrypted only by a private key provided in the app. Encryption and decryption is done by RSA (Rivest-Shamir-Adleman encryption) Algorithm which is an asymmetric encryption technique
- **Asset info**: It contains security status of Asset in which a particular tag is attached. This is marked as **Safe** when the Tag is connected to the Gateway and as **Unsafe** if it gets disconnected. Whenever the Asset becomes unsafe, alarming system gets activated in the apps of conductor and passenger.

3.3.4 Bus Location Data

This collection contains the location coordinates of the Bus stored along with a timestamp during a journey. This data will be helpful in tracking the actual path covered by the Bus and

generating the route in Google Map. The following figure Fig. 3.8 shows the sample document in this database. Each document consists of three fields:

- **latitude**: This field contains the latitude information of the bus.
- **longitude**:This field contains the longitude information of the bus.
- **ts**: This field contains the timestamp value at which the above values were stored.

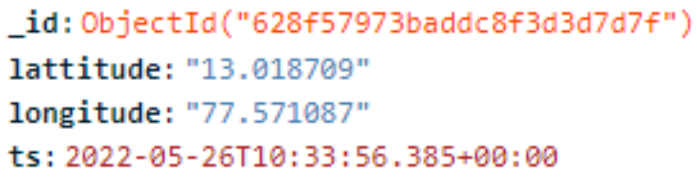


Figure 3.8: Sample Document in Bus Location Data

3.3.5 Collection for Bus Data

Following figure Fig. 3.9 shows a sample document stored in **Bus data**. Each document in the collection contains eleven fields

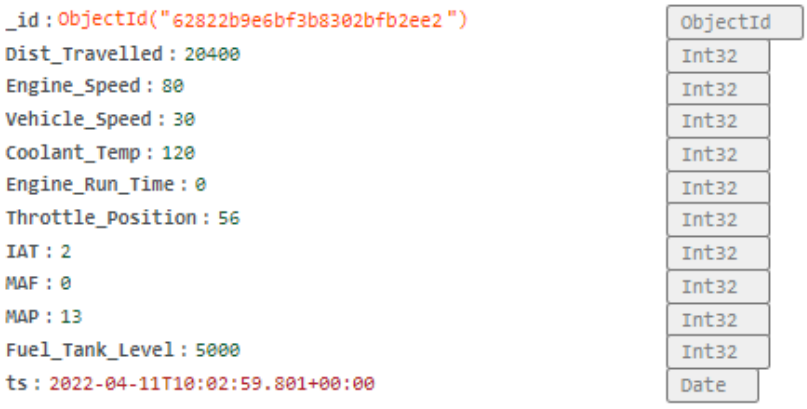


Figure 3.9: Sample document in Bus data

- **Dist Travelled**: This parameter shows the distance travelled by bus from last reset in kilometers

- **Engine Speed**: This parameter shows the RPM speed of engine
- **Vehicle Speed**: This parameter shows the vehicle speed in Kmph
- **Coolant Temp**: This parameter shows the current Tempaerature of collant in deg Celcius
- **Engine Run Time**: This parameter shows the engine run time in seconds
- **Throttle Position**: This parameter shows the current position of throttle in percentage
- **IAT**: This parameter shows the intake air temperature in deg Celcius
- **MAP**: This parameter shows the Manifold Air Pressure in KPa
- **MAF**: This parameter shows the Mass air flow in grams/sec entering the engine
- **Fuel Tank Level**: This parameter shows the fuel tank level in percentage

3.3.6 Collection for OTP Data

Following figure Fig. 3.10 shows a sample document stored in **OTP data**. Each document in the collection contains 2 fields:

- **Mob** : This field contains the mobile number of user. One of the document has value as **OTP Generator** in this field. The document corresponding to that value contains the number of pending OTPs.
- **OTP** : This field contains the OTP to be sent to the above mobile number.

3.3.7 Collection for Lock Status

This collection contains the lock information. The following figure Fig. 3.11 shows a sample document in this collection. Each document consists of two fields:

- **Tag id**: This field contains the Tag id of the Tag.
- **Lock Status**: This field contains the status of lock corresponding to above Tag. The locks in use (i.e. in the locked condition), are marked as one, and the others are marked as zero.

```
_id: ObjectId("626f7aa040798b38ab596315")
Mob: "OTP Generator"
OTP: "1"
```

```
_id: ObjectId("6290bcdda2635746132787b6")
Mob: "6261613772"
OTP: "4588"
```


Figure 3.10: Sample document in OTP data


```
_id: ObjectId("624a8108c2e4edac55b0ef9c")
Tag_id: "bid_qr_001"
Lock_Status: "1"
```

```
_id: ObjectId("624a814cc2e4edac55b0ef9d")
Tag_id: "bid_qr_002"
Lock_Status: "0"
```


Figure 3.11: Sample Document in Lock Status

3.4 Realm functions

To perform actions like add, update, delete entry in database, we need to write functions in MongoDB. These functions are attached to an http Endpoint in MongoDB Realm. To access these functions we just need to make an http request(get/post/delete) to the endpoint URL with payload containing the information to be sent.

3.4.1 Realm functions for Luggage Data

The following functions are written in MongoDB Realm to perform specific actions to the documents stored in collection **Luggage Data**:

- **get**: This function is used to get any document information from collection **Luggage data** in the database **ASSET INFO**. It contains an argument that contains the **TAG ID** of Tag and returns the document containing that id.
- **post**: This function is used to post any document in the collection **Luggage data** in the database **ASSET INFO**. This function contains an argument that contains the serial form of data to be entered in the document. This data is converted to JSON format and uploaded as a separate entry in the collection. It also updates the **Lock Status** corresponding to the current Tag id as '1' in the collection **Lock Status**.
- **update safe**: This function is used to update the **Asset Status** of a certain document in the collection **Luggage data** with the value **Safe**. The Gateway makes a request to endpoint connected to this function whenever the Asset is in Safe Zone, i.e., Tag is connected to the Gateway. This function has an argument that contains the TAG ID of the Tag. The document related to that TAG ID is updated.
- **update unsafe**: This function is used to update the **Asset Status** of a certain document in the collection **Luggage data** with the value **Unsafe**. The Gateway makes a request to endpoint connected to this function whenever the Asset is in Safe Zone, i.e., Tag is connected to the Gateway. This function has an argument that contains the TAG ID of the Tag. The document related to that TAG ID is updated.
- **delete**: This function is used to remove the entry of document from the collection **Luggage data**. This function has an argument that contains the TAG ID of the Tag. The document related to that TAG ID is deleted. It also updates the **Lock Status** corresponding to the current Tag id as '0' in the collection **Lock Status**.

3.4.2 Realm functions for Employee Data

The following functions are written in MongoDB Realm to perform specific actions to the documents stored in collection **Employee Data**:

- **get Employee**: This function is used to get the information of an employee, with the employee id given in the argument of this function, from the collection **Employee data**.
- **post employee**: This function is used to post the information of an employee, with employee details given in the argument of this function, from the collection **Employee data**.
- **get unsafe**: This function is used to get the details of all the entries in collection **Luggage Data** of database "ASSET INFO," which are marked as **Unsafe** in the "Asset status" key.
- **update employee contact**: This function is used to update the contact number of an employee in the document related to their Employee id. The argument of this function contains a document with the contact number and employee id of the employee. This function is used in the conductor app after verifying the password of the employee. The contact number will only be updated after verifying the password of the employee.
- **update employee password**: This function is used to update the password of an employee in the document related to the Employee id of them. The argument of this function contains a document with the password and employee id of the employee. This function is used in the conductor app after verifying the OTP sent to the contact number of the employee. Password will only be updated after verifying OTP sent to the contact number of the employee. NOTE: Either the password or the contact number of the employee can be modified at a time. And modifying one will need the information of another one as verification.
- **delete Employee**: This function will be used by the employee at the time of resigning from the bus company to remove the entry from the collection **Employee data**.
- **get lock data**: This function will be used by Tags to check whether the lock should be enabled or disabled in Tag

3.4.3 Realm functions for Bus Data

The following functions are written in MongoDB Realm to perform specific actions to the documents stored in collection **Bus Data**:

- **post bus**: This function will be used by the Gateway to upload the documents consisting the OBD parameters in ”Bus Data”

3.4.4 Realm functions for OTP Data

The following functions are written in MongoDB Realm to perform specific actions to the documents stored in collection **OTP Data**:

- **get OTP**: This function will be used by Gateway to get any pending OTP from database
- **post OTP**: This function will be used by mobile apps to generate a random OTP for entered mobile number and store in database
- **delete OTP**: This function will be used by Gateway to delete the document corresponding to the sent OTP from database

3.4.5 Realm functions for Lock Status

The value of lock status are updated by post and delete functions for the collection **Luggage Data**. The following function is written in MongoDB Realm to get the lock information stored in collection **Lock Status**:

- **get lock data**: This function takes Tag id as arguments and returns the lock status stored in document corresponding to that id.

3.4.6 Realm functions for Bus Location Data

The following functions are written in MongoDB Realm to perform specific actions to the documents stored in collection **Bus Location Data**:

- **update last loc**: This function will be used by the gateway having GPS module to update last location co-ordinates to the database. This updated last location value will be used by various other functions to get the latest location of bus.
- **get last loc**: This function will be used by android apps to get the last location of the Asset where it was connected. Also at the time of disconnection of tag, along with updating the Asset Status as unsafe, the last connected location will also be updated in database

document. This will be helpful for Gateway to send SMS containing Google Map URL of last connected location of Asset to the user's phone number.

- **update ts**: This function is used to add timestamp value to documents of database. Timestamp makes our Charts to be plotted with respect to time on X axis.

3.5 Charts for Data stored in MongoDB

The data stored in MongoDB can be plotted in form of a chart using **MongoDB Charts** feature. These charts are helpful for analyzing the data stored in a better way. The critical parameters which may affect the health of Bus, the security of Assets etc can be monitored easily. Following charts have been plotted in MongoDB which keeps updating every hour automatically:

- **Coolant Temperature** This chart shows the variation of Coolant Temperature of Bus with time.
- **IAT** This chart shows the variation of Intake Air Temperature of Bus engine with time.
- **MAP** This chart shows the variation of Absolute Manifold Pressure of Bus with time.
- **MAF** This chart shows the variation of Mass Air Flow entering the Bus engine with time.
- **Vehicle Speed** This chart shows the variation of Vehicle Speed with time.
- **Engine Run Time**This chart shows the engine run time since last start with time.
- **Fuel Tank Level** This chart shows the variation of Fuel level of Bus with time.
- **Lock Status** This chart shows the total number of locked Tags and unlocked or unused Tags available in Bus.
- **Asset Status** This chart shows the total number of safe and unsafe luggage in Bus.

3.6 Software Design

3.6.1 Android apps

3.6.1.1 User App

User app is an Android app made for the passengers entering the bus. All the passengers need to install this app for attaching the TAG to their Assets. This app allows users to connect to the **Luggage Data** collection of the **ASSET INFO** database in MongoDB. After successful installation user gets three options on the initial screen for **addition**, **connection**, and **removal** of an entry of Tag in the database. The addition option is used at the time of entering the bus. Once successful entry is done, users can use the TAG ID and Password as their login credentials to navigate to Connect Screen. Once successfully connected, the user will be able to continuously monitor the safety of their Assets. At the time of deboarding the bus, the user will need to remove the entry from the database in order to unlock the TAG. For removal of entry, the user will have to enter their login credentials in the remove screen of the app.

User app has the following screens as shown in figure Fig. 3.12:

- **Initial Screen**: This is the first screen that will be visible to the user after opening the app. The user gets three options on this screen ”Add Entry,” ”Connect,” and ”Remove Entry.” At the time of boarding, the user must select the ”Add Entry” option, which will navigate the user to the ”Add Entry” Screen.
- **Add Entry**: This screen allows the user to add an entry in the ”Luggage data” collection. Users need to fill in some information and also scan the QR code provided above the TAG to complete this process. The user is also asked to set a password which will be helpful at the time of login.
- **Connect screen**: This screen is opened once the user selects the ”Connect” option in the initial screen of the app. This screen asks the user to enter their login credentials. After successful login, another screen, ”Screen Mongo,” is opened.
- **Screen Mongo**: This screen is for continuous monitoring of the security of the Asset of the user. The user is able to see the Asset status, and also, whenever the asset status becomes Unsafe, it triggers an alarm in the user’s mobile phone.
- **Remove Entry**: This screen is used to remove the Tag entry from database and unlock the tag at the time of arrival of destination.

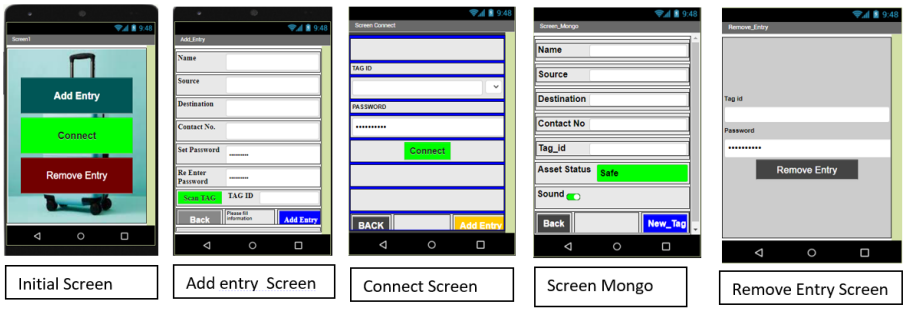


Figure 3.12: Screens of User App

3.6.1.2 Conductor App

This app is for the conductors of the bus, which will allow them to connect to the ”Employee data” and ”Luggage data” collection of database ”ASSET INFO.” This app allows conductors to create their employee accounts, connect and get data of passengers and their assets, and remove their employee accounts.

The conductor app has the following screens as shown in figure Fig. 3.13:

- **Initial Screen**: This screen is the first screen that appears after opening the conductor app. The conductors get three options on this screen, ”Create New Account,” ”Login,” ”Remove Entry.” At the time of joining the bus company, conductors need to select the ”Create New Account” option, which navigates them to the sign-up screen.
- **Sign up Screen**: This screen helps conductors to create an employee account. Conductors are asked to enter some information and set a new password. After filling the conductors need to scan the Authorization card provided in the office of the bus company. After the correct card is scanned, then only the new employee account is created. The employee gets their unique Employee id at the time of registration. They will need this ID and Password at the time of login.
- **Login Screen**: This screen allows registered conductors to log in and connect to the database. After successful login, ”Screen Connected” is opened.
- **Screen Connected**: This is the final screen that is used to monitor the Asset status of all the assets present in the bus. This screen displays the list of all the unsafe luggage. Also, the Conductor gets options like query, removal of passenger entry in this screen.

- **Forgot Password**: This screen allows the Conductor to change their Password after verifying OTP sent to their registered mobile number.
- **Remove Entry**: This screen is used to remove the employee entry from the database. This should be done at the time of resignation of any employee.

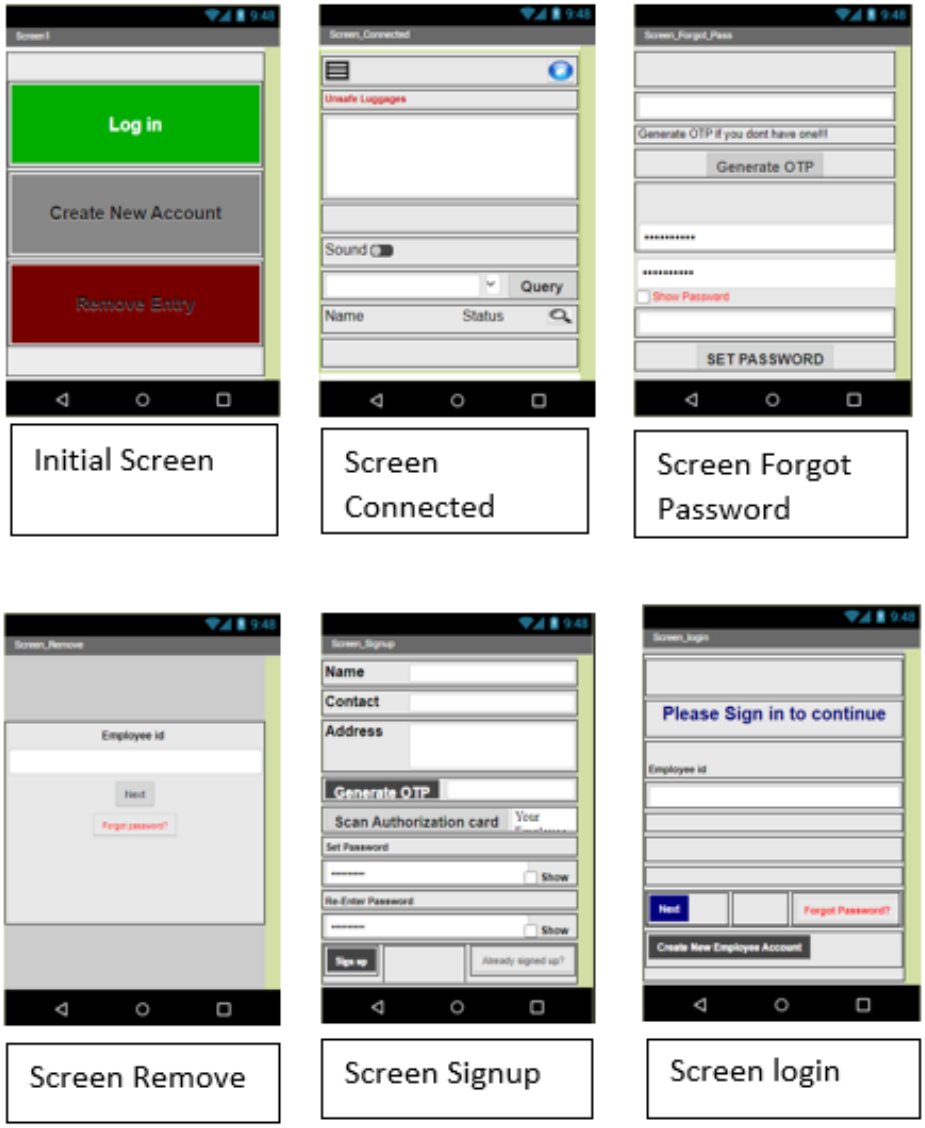


Figure 3.13: Screens of Conductor App

Chapter 4

Engineering Phase

In this chapter, we will discuss the flowchart of various tasks being performed and the experimental results which we got after using the components described in the design phase.

4.1 Circuit Diagram

4.1.1 Tag

The following figure Fig.4.2 shows the circuit diagram of the tag. It consists of ESP32 board, Lock, and power supply.Figure Fig. 4.1 shows the picture of actual tag.

4.1.2 Gateway

The following figure Fig. 4.3 shows the circuit diagram of the Gateway. It consists of ESP32 board, and power supply. Figure Fig. 4.4 shows the picture of actual Gateway.

4.1.3 Master Gateway

The following figure Fig. 4.5 shows the circuit diagram of the Master Gateway. It consists of ESP32 board, GPS module, GSM module, and power supply. Figure Fig. 4.6 shows the picture of Master Gateway.

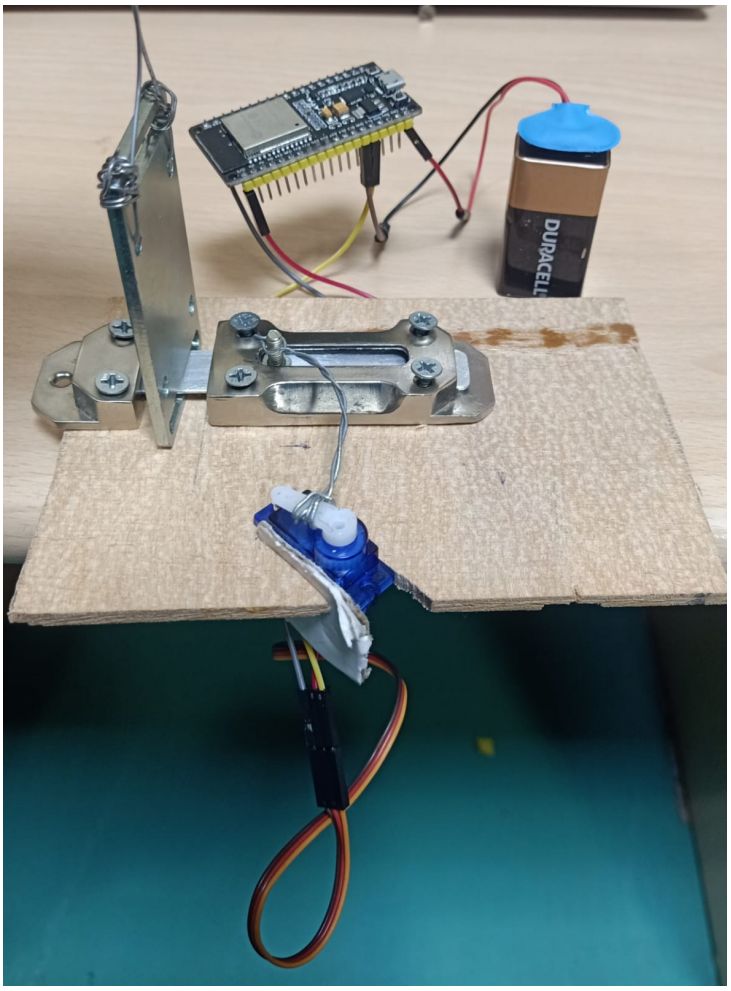


Figure 4.1: Picture of the Tag

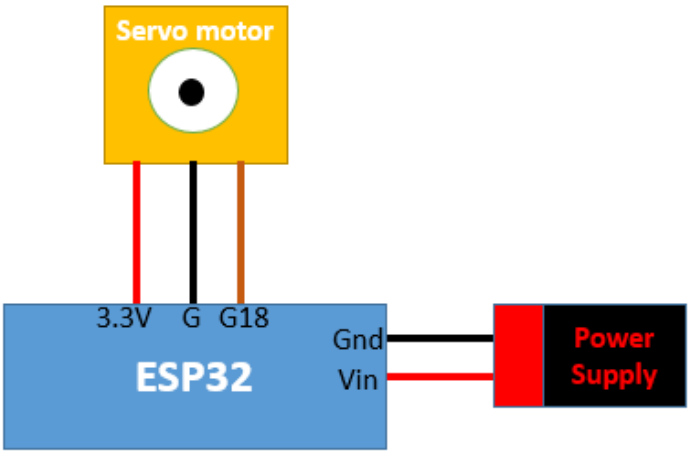


Figure 4.2: Circuit diagram of the Tag

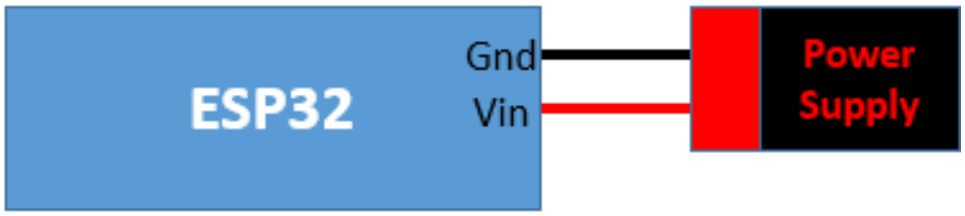


Figure 4.3: Circuit diagram of the Gateway

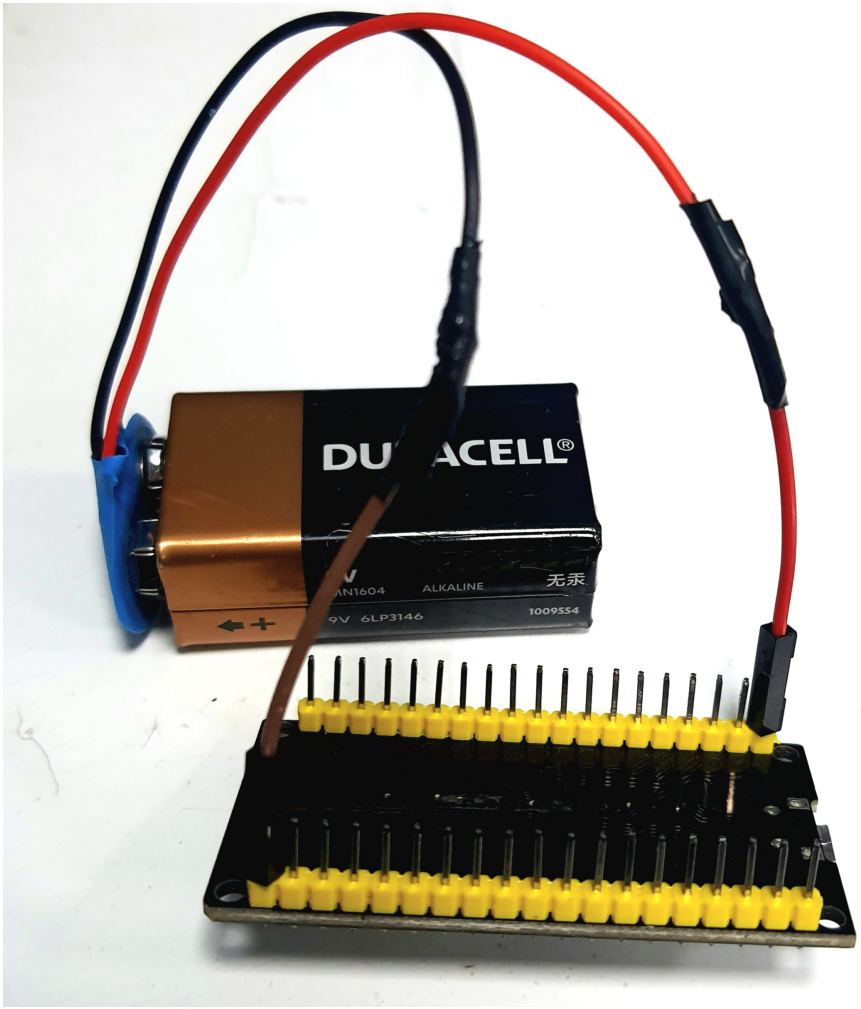


Figure 4.4: Picture of the Gateway

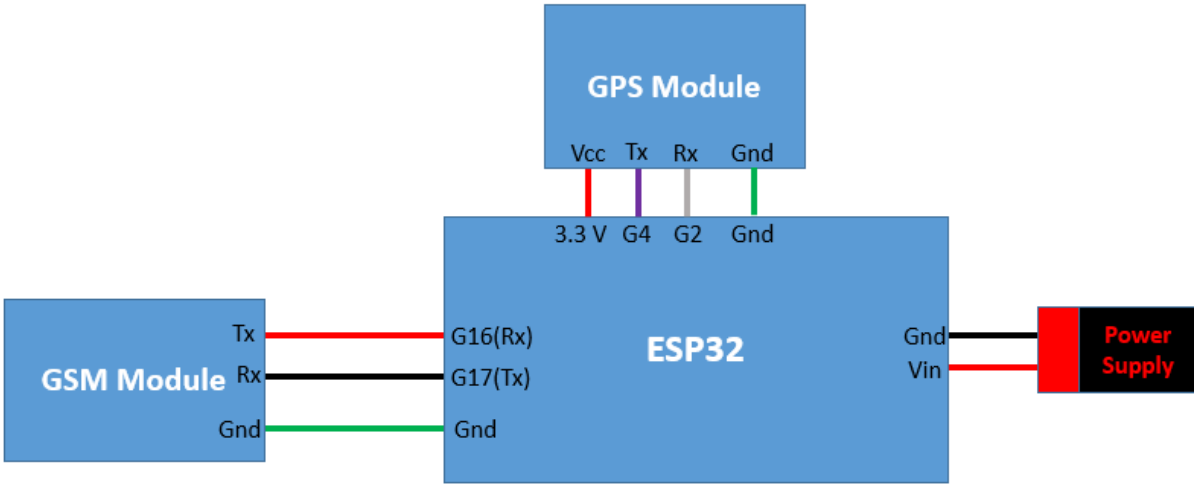


Figure 4.5: Circuit diagram of the Master Gateway

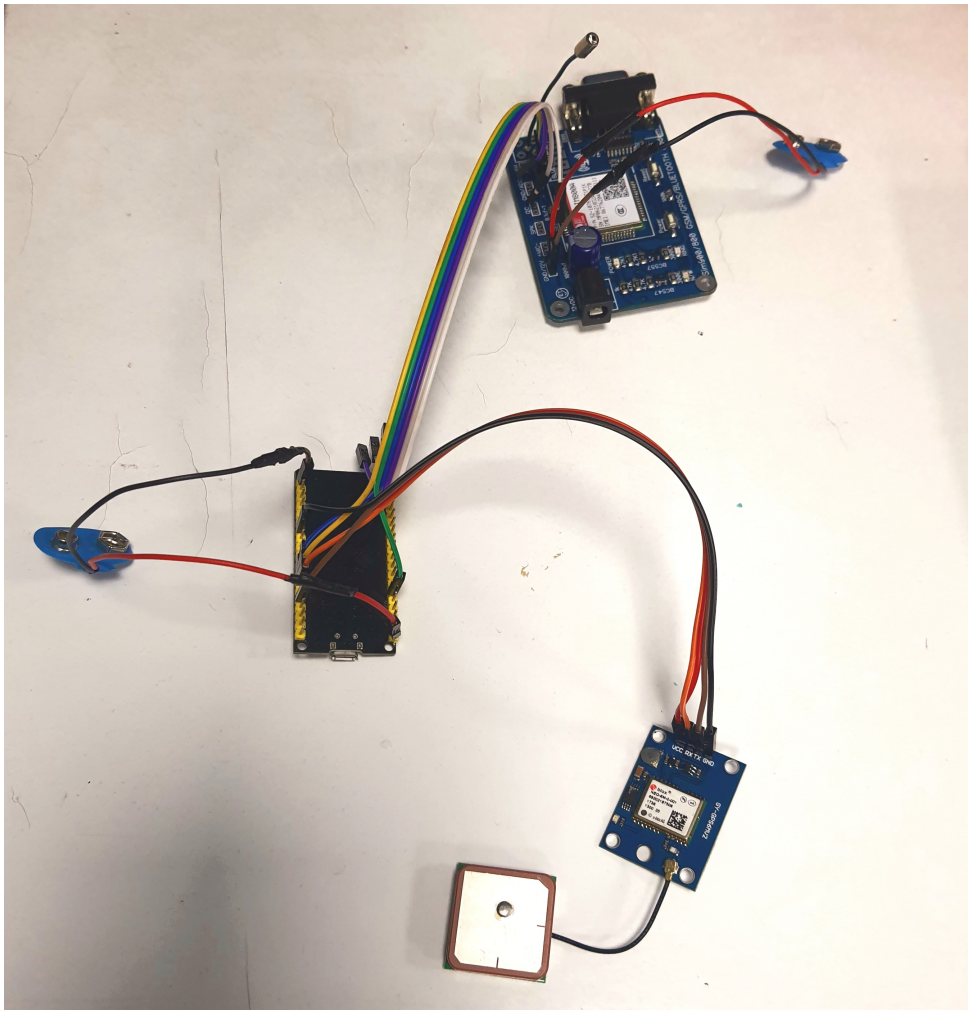


Figure 4.6: Picture of the Master Gateway

4.2 Flowcharts and Block Diagrams

4.2.1 Asset Status update

The asset status field in our database contains the security status of our Luggage. This Status is decided on the BLE connection status of Gateway with the Tags attached to the Luggage. All the connected Tags are stored as safe, and those tags which get disconnected due to sufficient distance from Gateway are marked as unsafe. The following flowchart in figure Fig. 4.7 shows this task procedure:

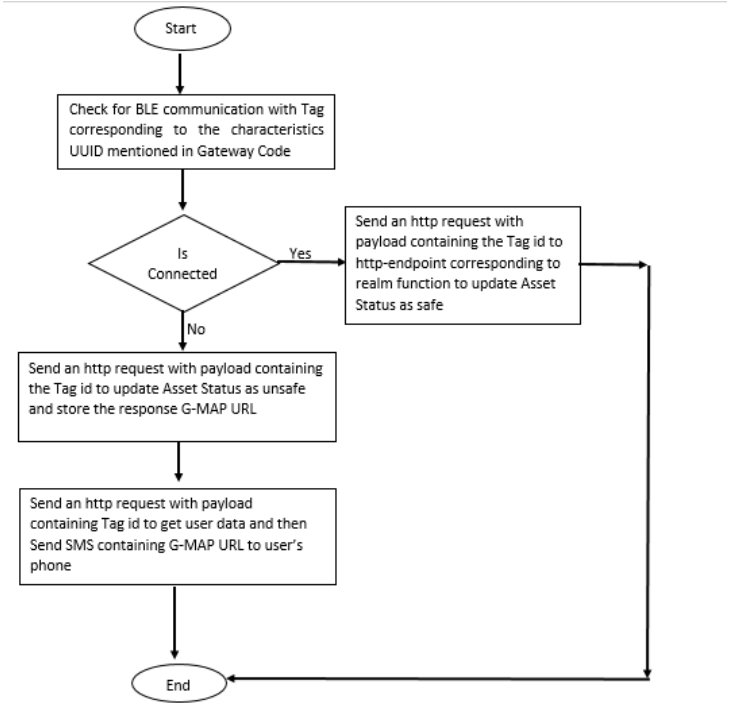


Figure 4.7: Flowchart for Asset Status Update by Gateway

4.2.2 Task performed by Tag

The Tags are attached to the Luggage of the user. It keeps broadcasting BLE packets and receives the characteristic value to perform the required action with lock. The following flowchart in figure Fig. 4.8 shows this task procedure:

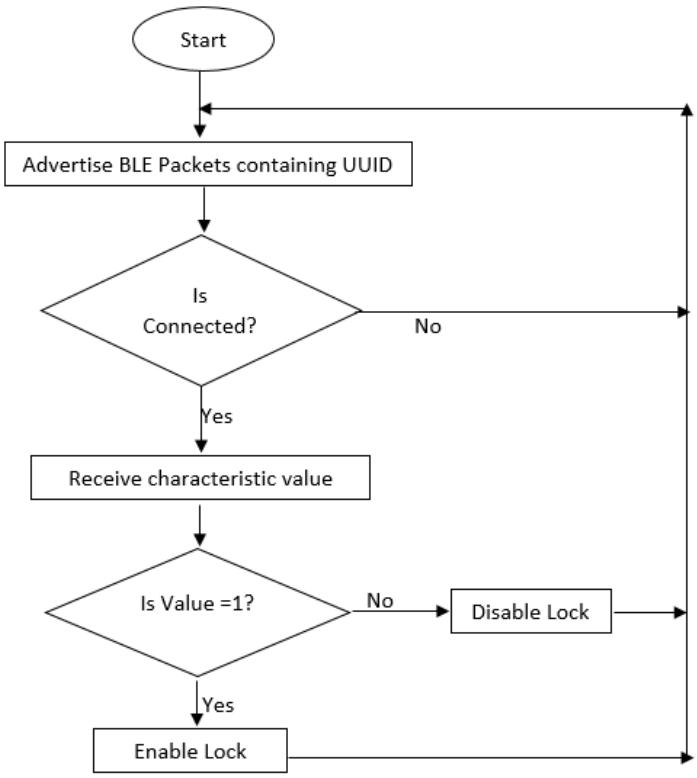


Figure 4.8: Flowchart for Tasks performed by Tag

4.2.3 Task performed by Gateway

The Gateway performs multiple tasks, as shown in the flowchart in figure Fig. 4.9. It keeps performing these tasks in loop.

4.2.4 OTP Generation

At the time of mobile number verification for a new employee and at the time of Password reset, an One Time Password(OTP) Generation is required. The following Block Diagram in figure Fig. 4.10 shows the complete procedure. The information above the arrow in red color shows the mode of communication, and that below the arrow inside curly braces shows the data transferred. A random OTP is generated by MongoDB, which is used for verification.

4.2.5 Last Location update

GPS module is used to track the longitude and latitude coordinates of the Bus location, which is stored in the database by Master Gateway. The following flowchart in figure Fig. 4.11 shows

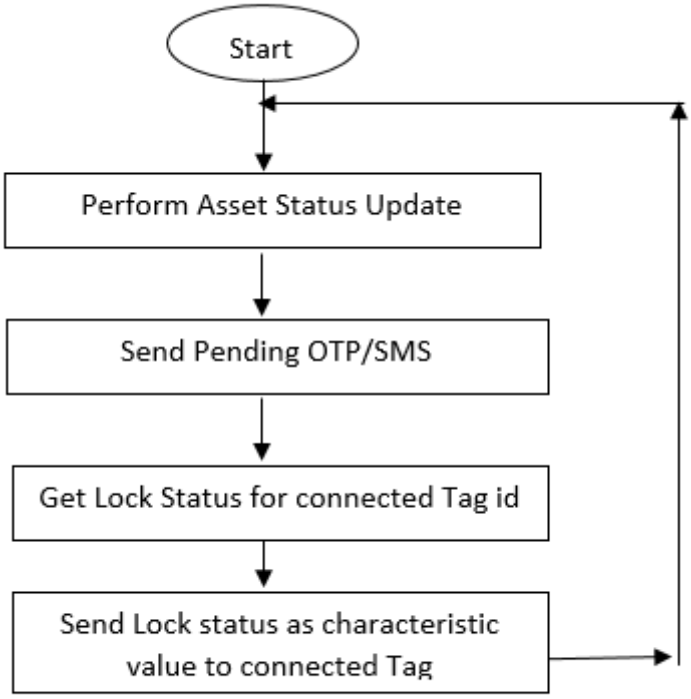


Figure 4.9: Flowchart for tasks performed by Gateway

this process.

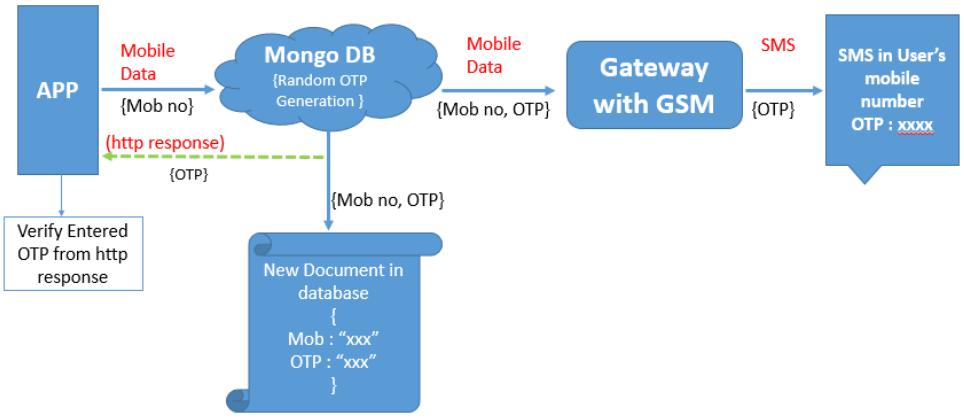


Figure 4.10: Block Diagram for OTP Generation

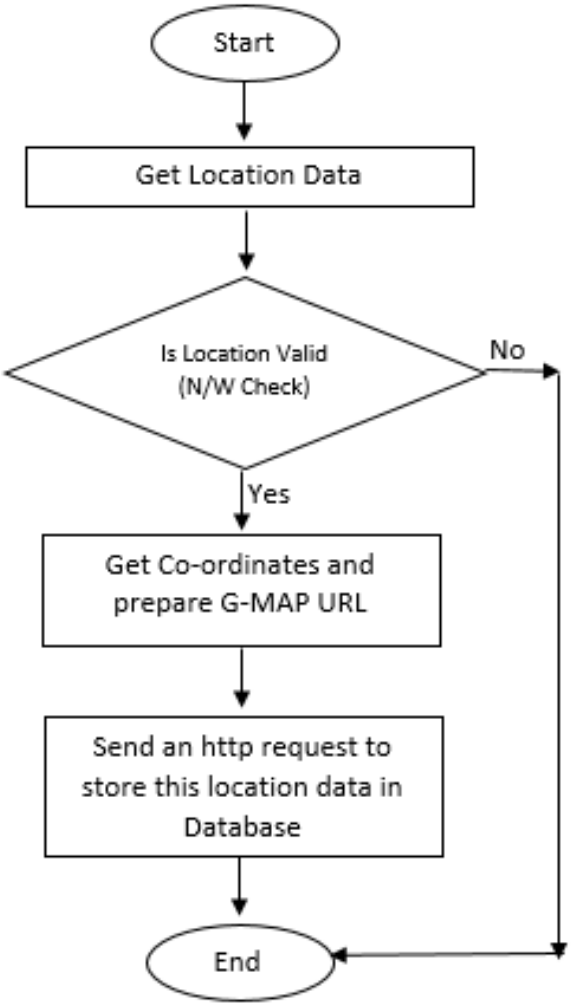


Figure 4.11: Flowchart for Location Update by Master Gateway

4.3 Data Stored in MongoDB Atlas Cloud

The following Tree in figure Fig. 4.12 shows the arrangement of databases and collections inside our cluster. The cluster name is Cluster0, which contains the databases named **ASSET INFO** and **BUS INFO**.

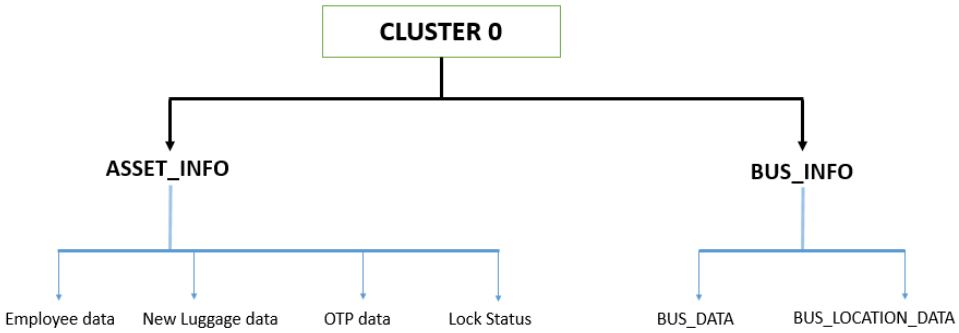


Figure 4.12: Arrangement of data in MongoDB

4.3.1 New Luggage Data

This collection contains the data of all the passengers and their luggage status. The following figure Fig. 4.13 shows a sample document in this collection.

Figure 4.13: Sample Document in New Luggage Data

4.3.2 Lock Status

This collection contains the lock information. The lock is used, i.e., in the locked condition, it is marked as one, and others are marked as zero. The following figure Fig. 4.14 shows a sample document in this database.

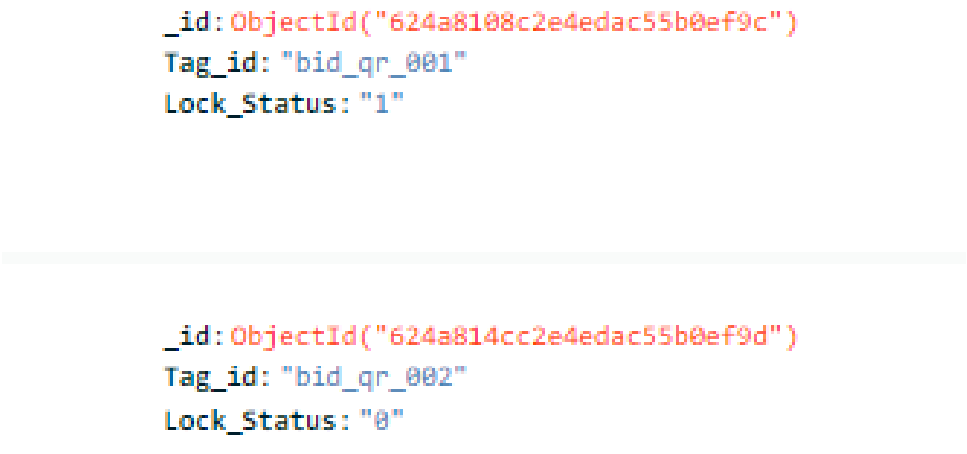


Figure 4.14: Sample Document in Lock Status

4.3.3 OTP Data

This collection contains the pending OTP information. The first document contains the total number of pending OTP, and the rest documents which has the field **valid** contain information regarding the awaiting OTP. The following figure Fig. 4.15 shows a sample document in this database.

Figure 4.15: Sample Document in OTP Data

4.3.4 Employee Data

This collection contains the information of employees working in the bus company. The following the figure Fig. 4.16 shows the sample document in this database.

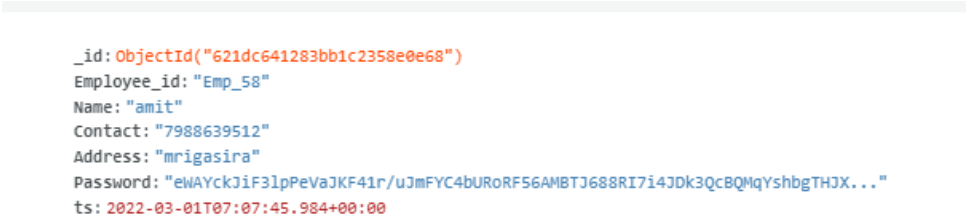


Figure 4.16: Sample Document in Employee data

4.3.5 Bus Data

This collection contains the information of the On-Board Diagnostic parameter and the location coordinates of the Bus on a journey. The following figure Fig. 4.17 shows the sample document in this database.

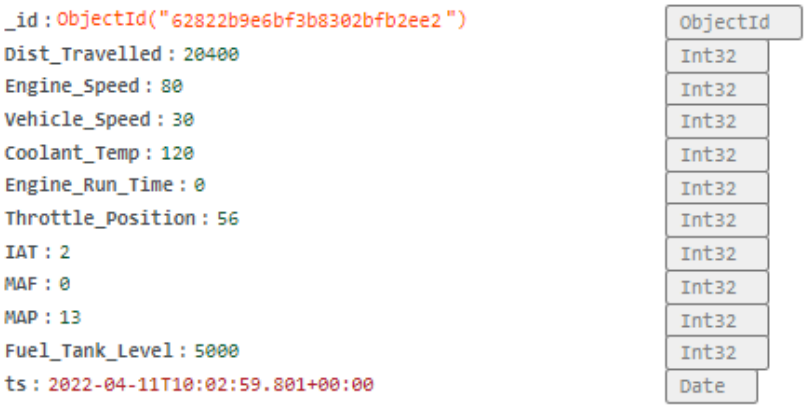


Figure 4.17: Sample Document in Bus Data

4.3.6 Bus Location Data

This collection contains the location coordinates of the Bus stored along with a timestamp during a journey. This data will be helpful in tracking the actual path covered by the Bus and generating the route in Google Map. The following figure Fig. 4.18 shows the sample document in this database.

```
_id: ObjectId("628f57973baddc8f3d3d7d7f")
lattitude: "13.018709"
longitude: "77.571087"
ts: 2022-05-26T10:33:56.385+00:00
```


Figure 4.18: Sample Document in Bus Location Data

4.4 Charts for the data stored in database

All the charts prepared are refreshed automatically every hour. The unit of time at which the samples are plotted and can be changed from seconds to years.

4.4.1 OBD Data Stored

The following charts show the variation of On-Board-Diagnostic parameters of the Bus with respect to time recorded during the journey.

4.4.1.1 Vehicle Speed

This graph shows the variation in vehicle speed(Kmph) with respect to time. This chart will be helpful in deciding whether the Bus Driver over speeds the Bus. The ideal vehicle speed must be less than 75 kmph. The sample chart for this parameter is shown in figure Fig. 4.19

4.4.1.2 Coolant Temperature

This graph shows the variation of Coolant Temperature(deg C) with respect to time. This chart will be helpful to decide whether the Bus is overheating due to any fault and also to check the condition of the Coolant Temperature sensor. For a sensor working in good condition, the coolant temperature must not be less than -40°C. The sample chart for this parameter is shown in figure Fig. 4.20

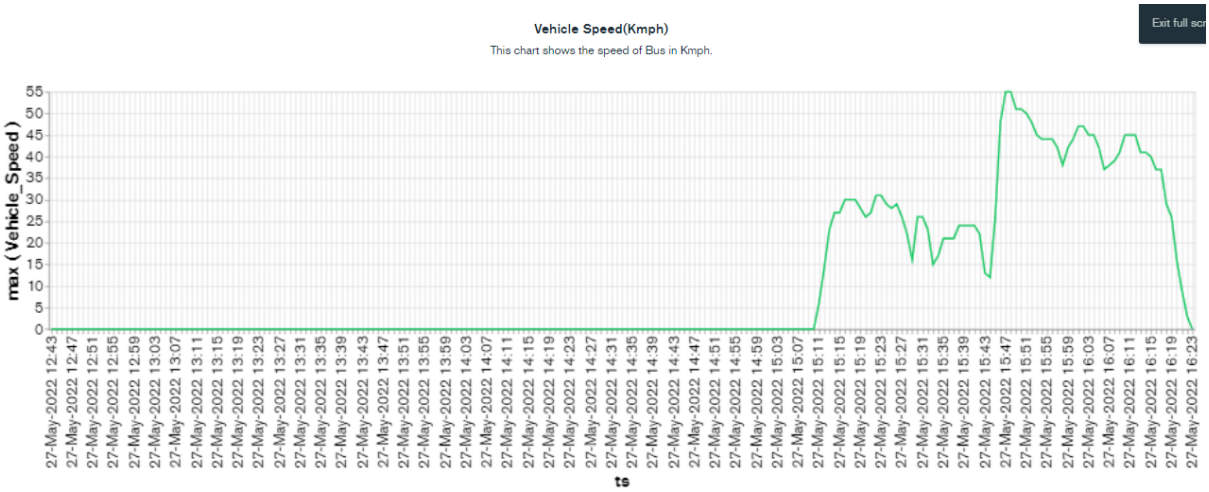


Figure 4.19: Vehicle Speed vs Time Chart

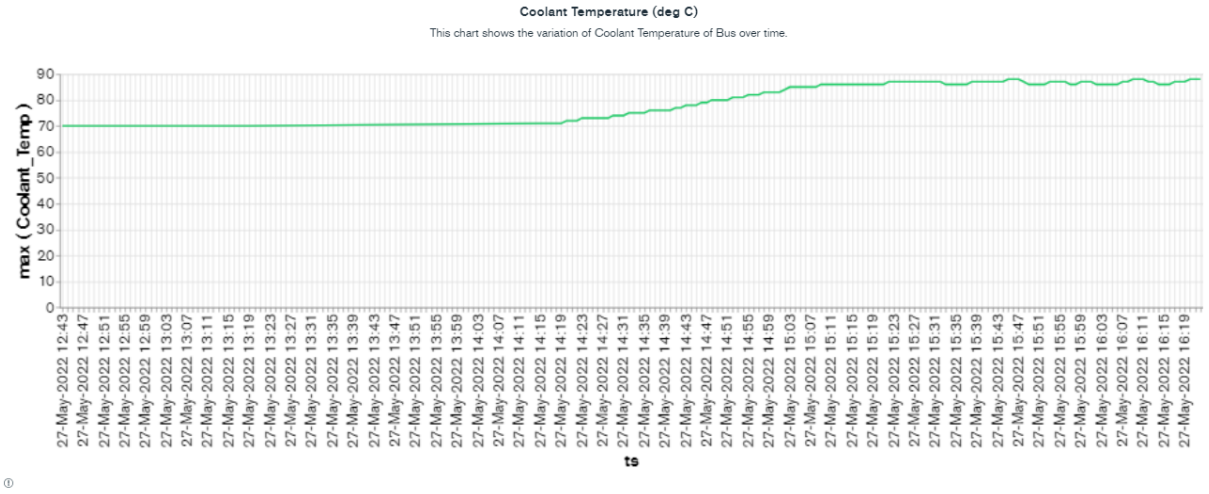


Figure 4.20: Coolant Temperature vs Time Chart

4.4.1.3 IAT

This graph shows the variation of Intake Air Temperature (deg C) with respect to time. This chart will be helpful in deciding whether the Bus engine IAT is within the specified limits and whether the sensor is working fine. The reading of this sensor must be in the range of ±10 °C of ambient temperature. Due to a faulty intake temperature sensor, the engine control module may think that the engine's air is colder or warmer than it actually is. A false signal may cause the Power-train control module (PCM) to miscalculate the air and fuel mixture, resulting in a drop in acceleration. The sample chart for this parameter is shown in figure Fig. 4.21

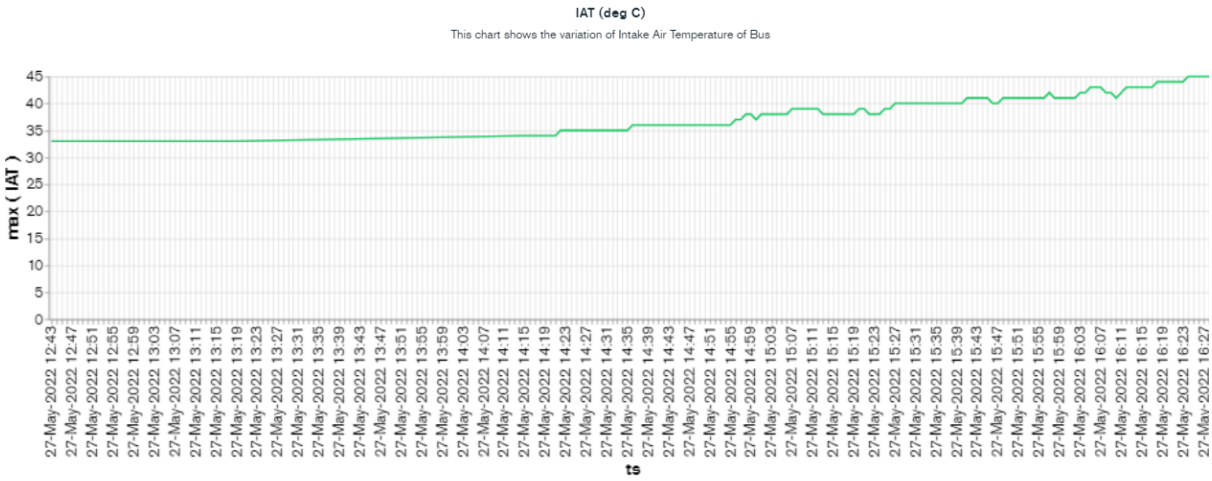


Figure 4.21: IAT vs Time Chart

4.4.1.4 MAP

This graph shows the variation of Manifold Absolute Pressure (KPa) with respect to time. This chart will be helpful to decide whether the Bus engine MAP is giving the correct reading and to detect any fault in the MAP sensor. A MAP sensor that measures high intake manifold pressure indicates high engine load to the Power-train control module (PCM). This results in an increase in fuel being injected into the engine. This, in turn, decreases the overall fuel economy. It also increases the amount of hydrocarbon and carbon monoxide emissions from our vehicles to the surrounding atmosphere. Hydrocarbons and carbon monoxide are some of the chemical components of smog. The sample chart for this parameter is shown in figure Fig. 4.22

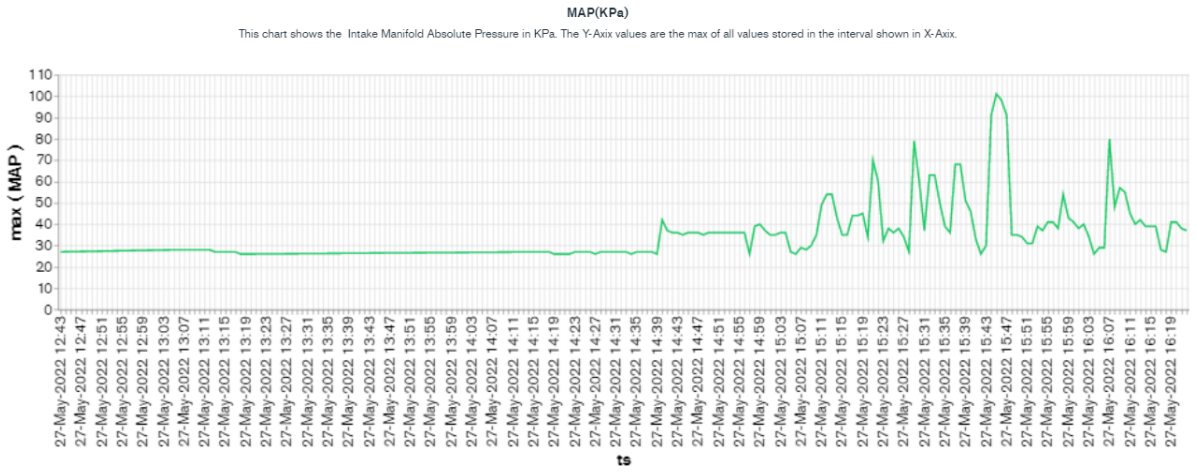


Figure 4.22: MAP vs Time Chart

4.4.1.5 MAF

This graph shows the variation of Mass Air Flow (grams/sec) entering the bus engine with respect to time. This chart will be helpful in deciding whether the Bus engine MAF sensor is working fine. With the engine at idle, the MAF's PID value should read anywhere from 2 to 7 grams/second (g/s) at idle and rise to between 15 to 25 g/s at 2500 RPM. The sample chart for this parameter is shown in figure Fig. 4.23

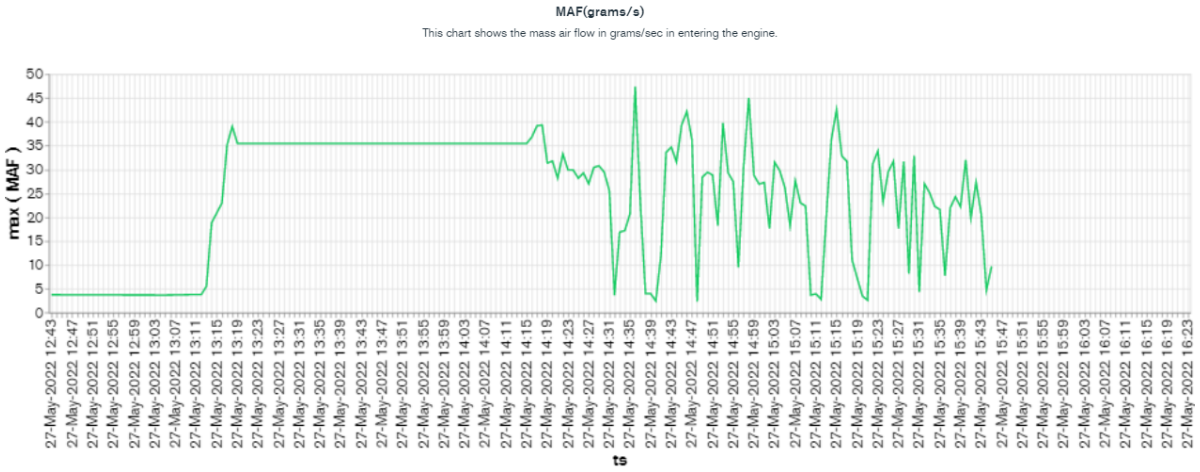


Figure 4.23: MAF vs Time Chart

4.4.1.6 Fuel Tank Level

This graph shows the variation of Fuel Level (Percentage) with respect to time. This chart will be helpful to check the absolute bus fuel expenditure in a journey. The sample chart for this parameter is shown in figure Fig. 4.24

4.4.2 Asset Security Status of Passengers

This graph shows the Asset Status by count in Bus. This chart will be helpful in checking the number of safe and unsafe Luggage. The sample chart for this parameter is shown in figure Fig. 4.25

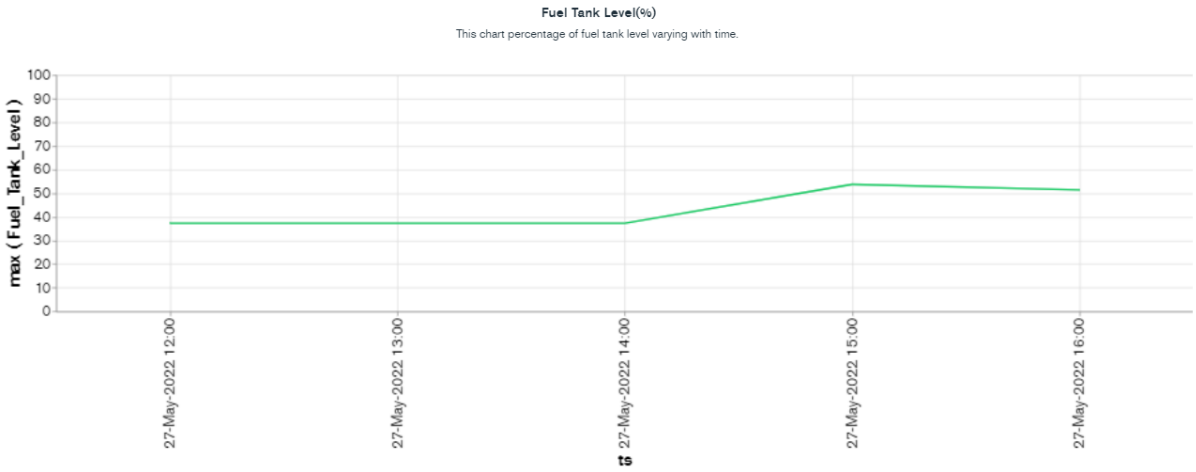


Figure 4.24: FUEL TANK LEVEL vs Time Chart

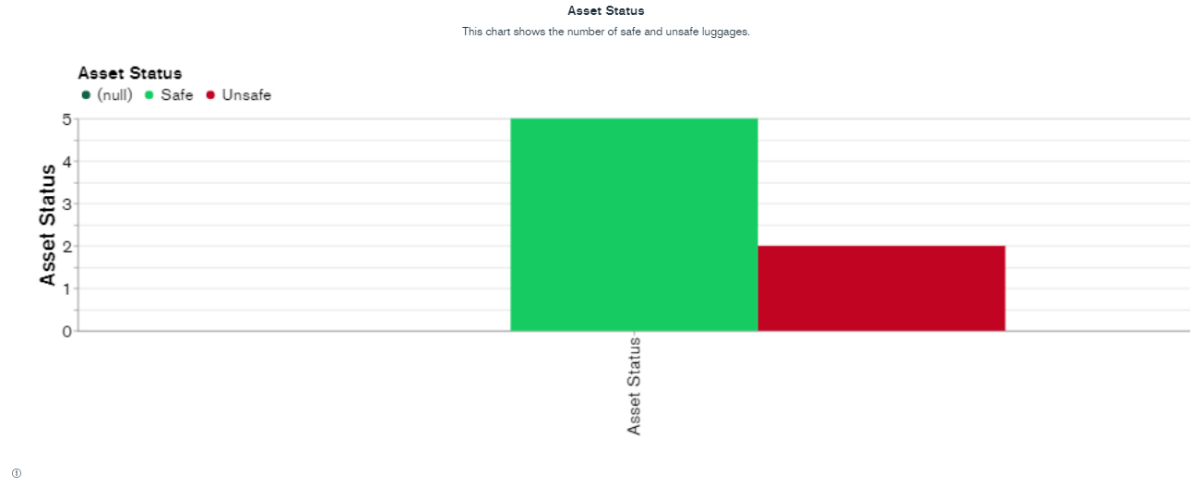


Figure 4.25: Asset Info vs Time Chart

4.4.3 Lock Status of Tags

This graph shows the Lock Status by count in Bus. This chart will be helpful in checking the number of used and unused Tags currently present in Bus. The used tags must be in Lock condition having Status ’1’, and the unlocked or unused will have Status as ’0’ as shown. The sample chart for this parameter is shown in figure Fig. 4.26

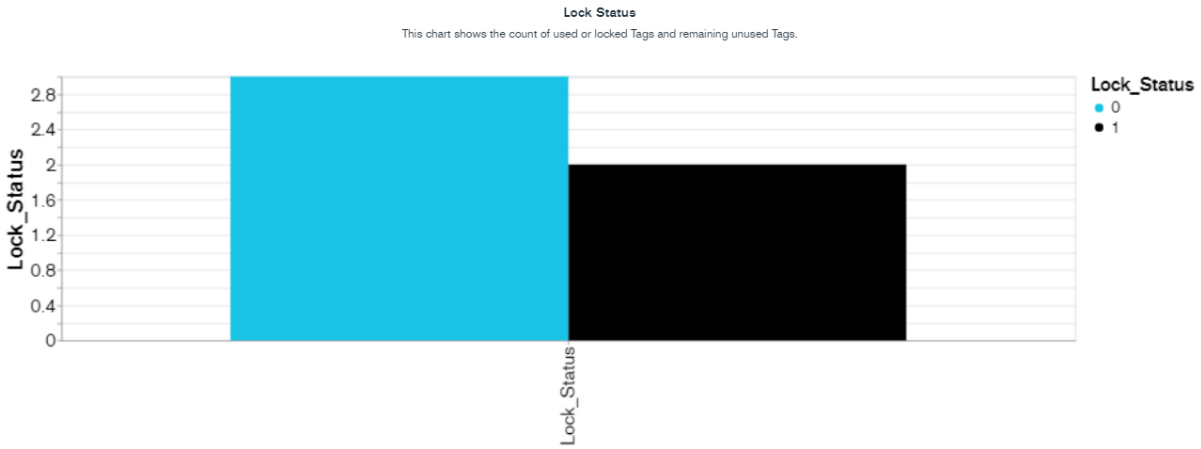


Figure 4.26: Lock Status vs Time Chart

4.5 Google Map Integration

4.5.1 Last connected location of Tag

Whenever the connection of Gateway from Tag is lost, the Asset Status is marked as **Unsafe**, and an SMS is sent with the last connected location information on the user's phone. This location information is actually a Google Map URL that, when clicked, opens G-Map pointing to the location of Tag when it was last connected. The following figures Fig. 4.27 and Fig. 4.28 shows the SMS and Google map location pointed by URL.

4.5.2 Re-Tracking of Route followed by the Bus

The location coordinates stored in the collection **Bus Location Data** can be exported as a csv file which can be imported into the my-maps service provided by Google to track the actual route covered by Bus. The coordinates were stored as a csv file named **new track**. After importing the coordinates csv file, we get the following map as shown in the figure Fig. 4.29. The route tracked is a collection of individual location points of all the data uploaded. On zooming we can spot IISC.The zoomed view of map is shown in figure Fig. 4.30

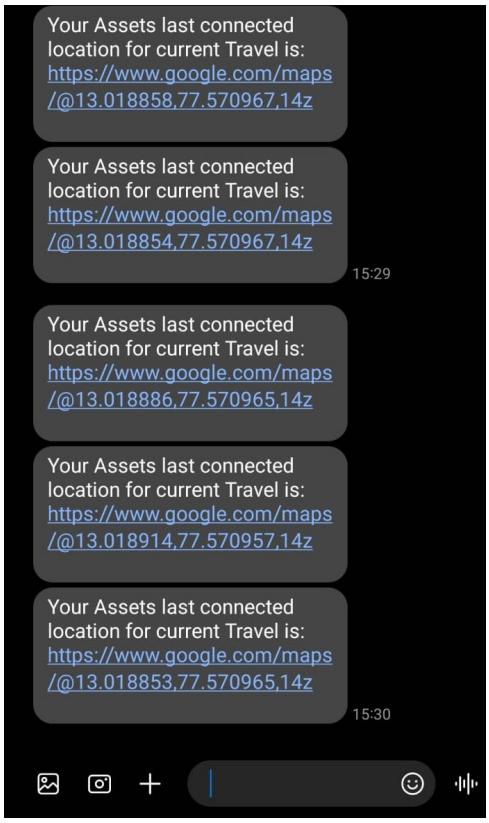


Figure 4.27: SMS received to User

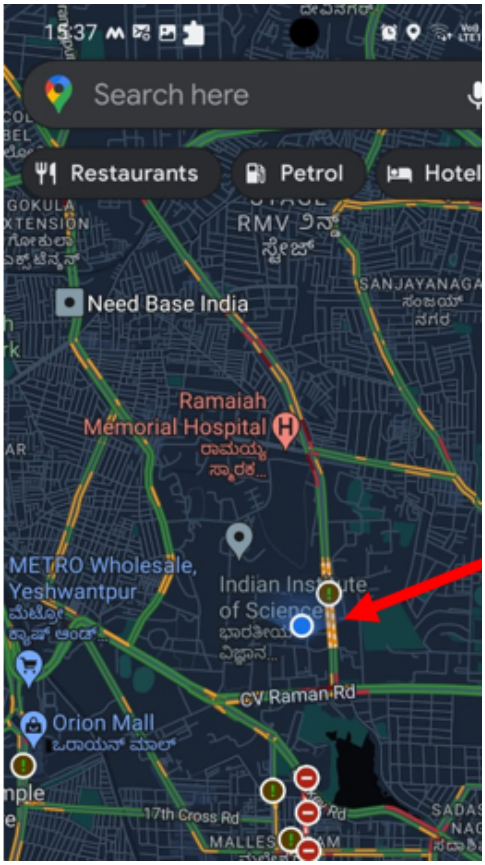


Figure 4.28: Last Connected location of Tag

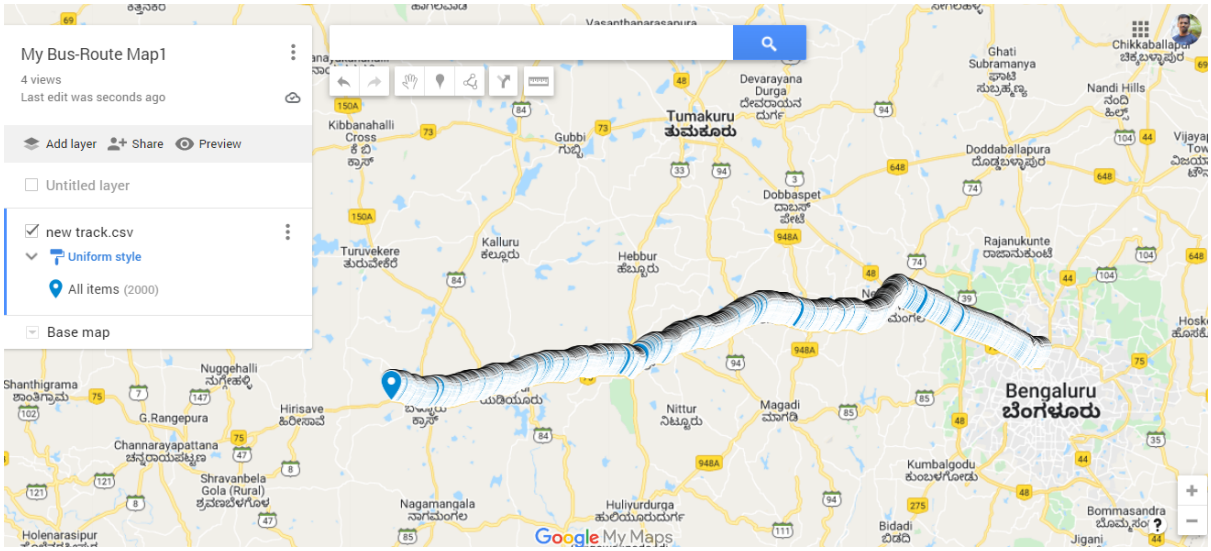


Figure 4.29: Location Co-ordinates plotted to form Google Map route

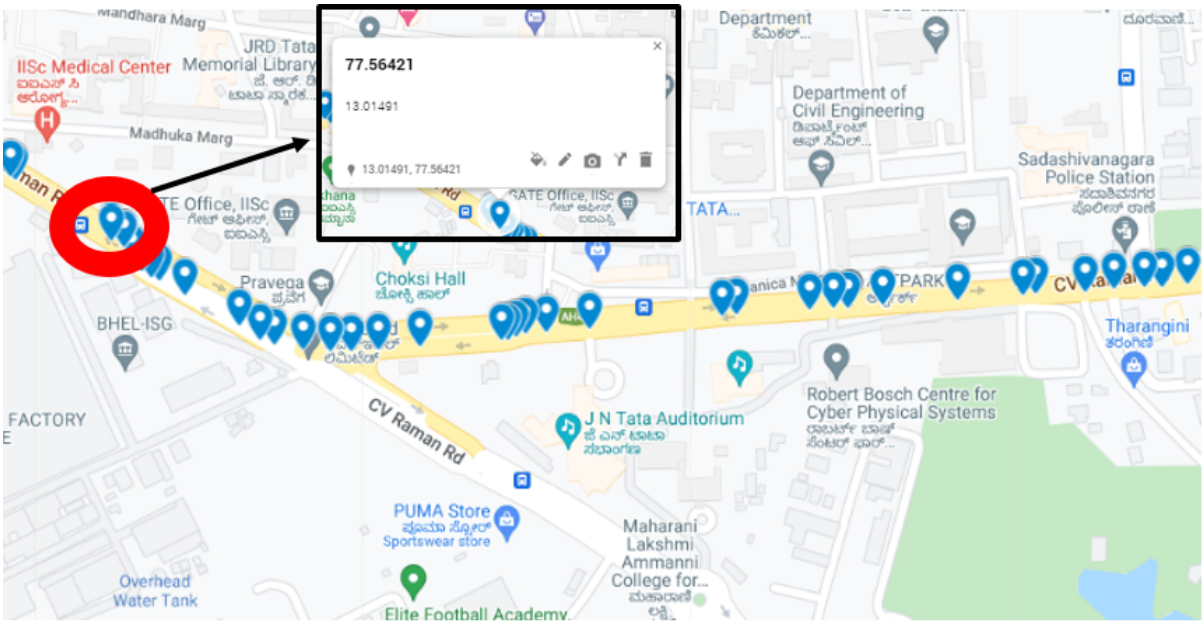


Figure 4.30: Zoomed view of Google Map route

4.6 Discussions

The following challenges were faced while building this project:

- **Handling sequential and parallel processes**: While building the apps for this project, there were some tasks that were required to occur sequentially, while there were tasks that ran parallelly. Initially, all the tasks were running parallelly, which created a problem for the sequential tasks. This problem was handled by using flags.
- **Handling HTTPS requests**: It is necessary to check the response code after making an HTTP request to ensure that the request was made successfully.
- **MongoDB Payload**: The payload in MongoDB can be directly accessed through their console, but in case the requests are made from ESP32, we have to use body.text() function to access the text inside payload.
- **Deploying changes in MongoDB**:After every change, we need to save and then deploy the changes to make them effective. Just by saving the changes and not deploying will revert to the old situation once the browser is closed.

4.7 Conclusion

We have successfully designed indoor location tracking using BLE-enabled Tags. We have provided Hardware for BLE Gateway and Tags along with software for User, Conductor to add an entry, monitor security status, connect to MongoDB database, etc. Also, we are storing the On Board Diagnostic Data of Bus for better surveillance by the Bus company. Also, using the GPS module, we have successfully designed the system to upload the location coordinates of the Bus at certain intervals of time. This helps the Bus company to re-track the route covered by the Bus and also the users to get the last connected location of their Tag. This is a complete end-to-end solution for users traveling in Buses, conductors, and the Bus company.

Chapter 5

Concluding Remarks

5.1 User Instructions

5.1.1 Development Environment - ARDUINO IDE

5.1.1.1 Software Installation

The programming codes for Tag and Gateway are written in Arduino IDE. Following are the instructions to setup Arduino IDE on a computer:

- Get the latest version of the app from the link **click here**
- When the download finishes, proceed with the installation, and please allow the driver installation process when you get a warning from the operating system
- Choose the components to install as shown in figure Fig. 5.1
- Choose the installation directory as shown in figure Fig. 5.2
- The process will extract and install all the required files to execute the Arduino Software (IDE) properly, as shown in figure Fig. 5.3
- After installation is complete open arduino.exe and in the Toolbar at top, Navigate to **Tools**→ **Board**→ **Esp32Arduino** → **Esp32 Dev Module**

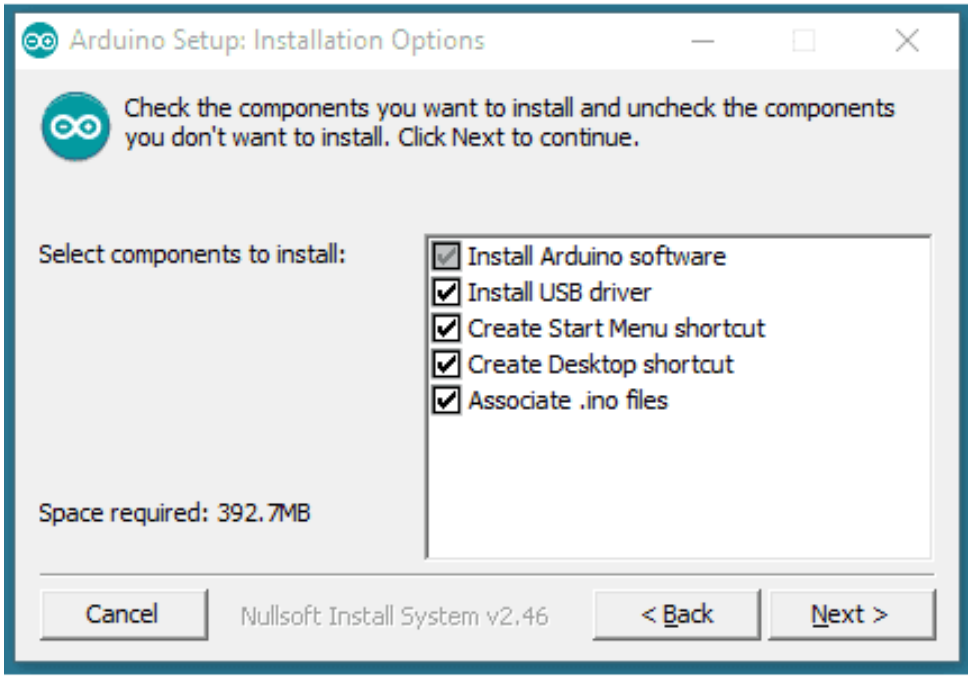


Figure 5.1: Installation options

5.1.1.2 Package Installation

We need the following packages to be installed in our Arduino IDE for our programs. Follow the steps below to install the packages named **GSM**, **Servo**, **ACAN**, **ArduinoJson**, **CAN**, **ELMDuino**, **Dictionary**, **ESP32Servo**, **ESPSoftwareSerial**, **OBD2**.

- Open Arduino IDE software
- In the Toolbar at top, Navigate to **Tools**→ **Manage Libraries** or simply press **ctrl+shift+I** . A box as shown in figure Fig. 5.4 will appear.
- In the search box search for the package name, select version and click install.

5.1.1.3 Creating a New Project

Follow the following instructions to create a new project in Arduino IDE and add the code for Gateway and Tag.

- Open Arduino IDE software
- In the Toolbar at top, Navigate to **File**→ **New** or simply press **ctrl+N**

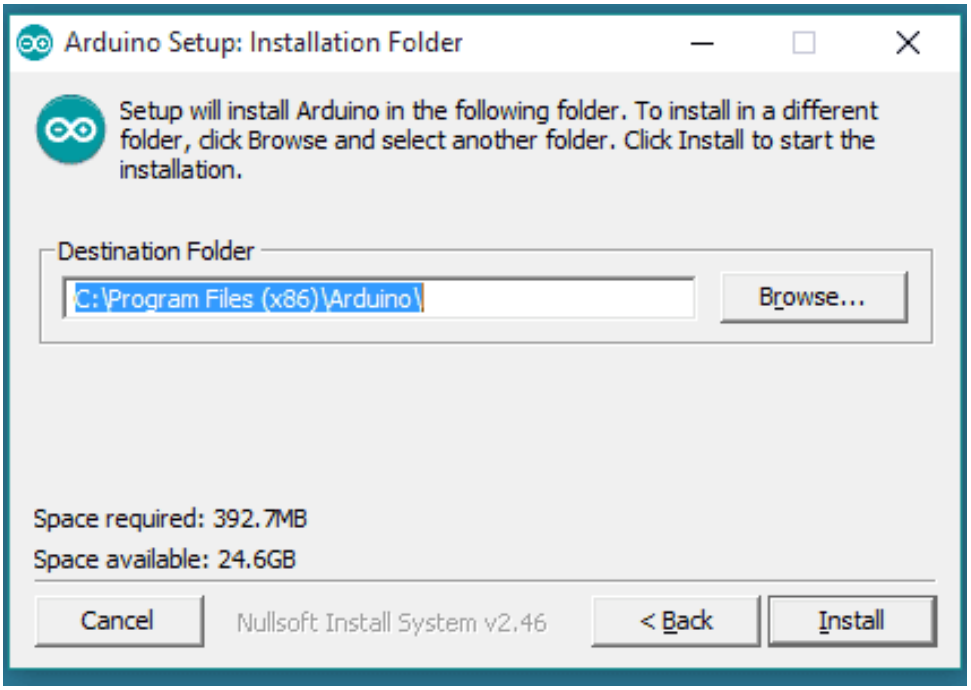


Figure 5.2: Installation folder

- A new sketch window will appear as shown in figure Fig. 5.5
- Remove all the text and copy the code (for Tag or Gateway) from the appendix section.
- To add header files press **ctrl+Shift+N** and add the header file name with extension ".h" and hit enter. Add the code for header file from the appendix section.
- In the Toolbar at top, Navigate to **File**→ **Save** or simply press **ctrl+S** and enter the name of program to save.

5.1.1.4 Uploading Project to the board

Follow the following instructions to upload the program to esp32 development board

- Select the COM port by navigating to **Tools**→ **Port** → **COM ***
- Navigate to **Tools**→ **Board**→ **Esp32Arduino** → **Esp32 Dev Module**
- In the Toolbar at top, Navigate to **Sketch**→ **Upload** or simply press **ctrl+U** to upload the code in board
- To open the serial monitor, navigate to **Tools**→ **Serial Monitor** or press **ctrl+shift+M**

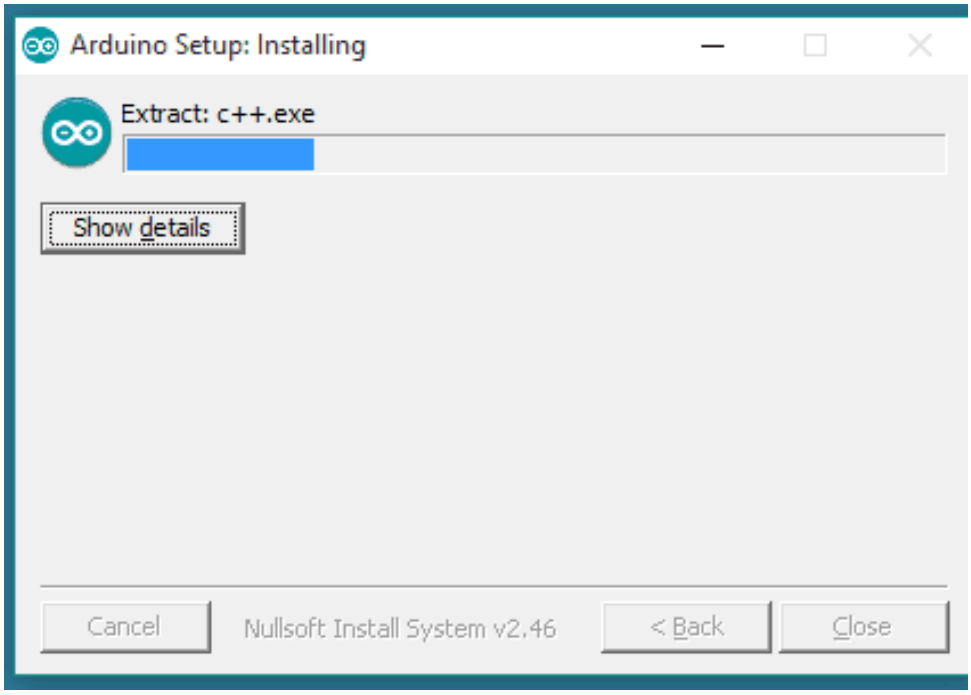


Figure 5.3: Installation setup

5.1.2 MongoDB

5.1.2.1 Initialization

Follow the following steps to initialize the MongoDB:

- Open the MongoDB home page by clicking on the link **MongoDB Home**
- Click on **Try Free** button and create account by filling the information required
- After successfully creating the account, reopen **MongoDB Home**
- Click on **Sign in** and enter login details
- Create a new cluster by clicking **Create** Button
- Choose the Shared option and click on **Create Cluster** to create a free cluster with shared RAM and 512MB data storage

5.1.2.2 Creating a database and collection

- After successful creation of cluster, click on **Browse Collections** button

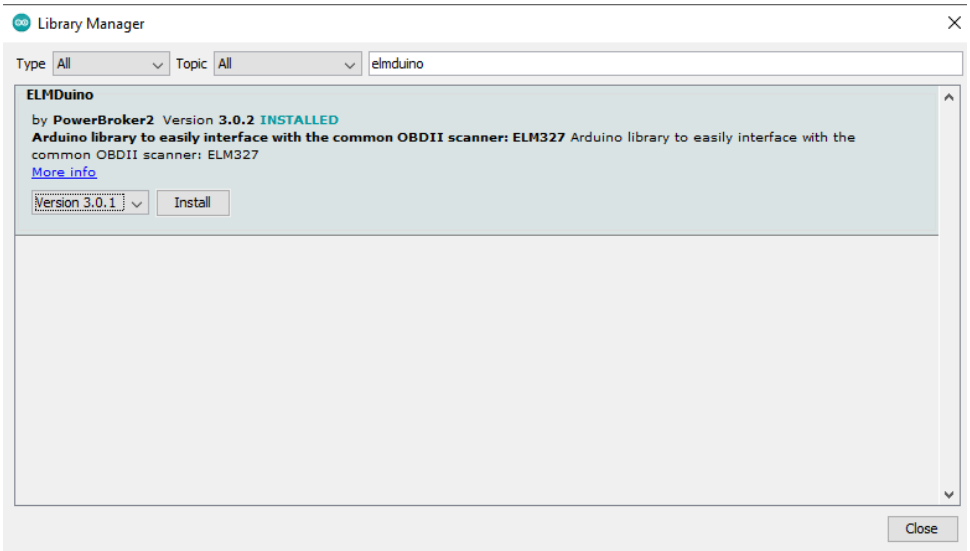


Figure 5.4: Arduino Library Manager

- Click on **Create Database** Button and enter the name of database and the first collection name inside it
- To add more collections inside a database, we need to select the database and click on the **+** sign

5.1.2.3 Providing Access to Cluster

- After creating Cluster, click on the **Network Access** Button on the left panel
- Select **Add IP Address** button and enter our IP Address. We can also select the **Allow access from anywhere** options
- Click on **Database Access** on the left panel and then Database Access → Add a new Database user → Permission(Atlas Admin)→ create

5.1.2.4 Deploying the Realm service

- After creating Cluster, click on the **Realm** Button on the top panel
- Add a Realm Service by clicking on **Create New App**
- Name the app, add the cluster name to be linked

Figure 5.5: New Sketch window in Arduino

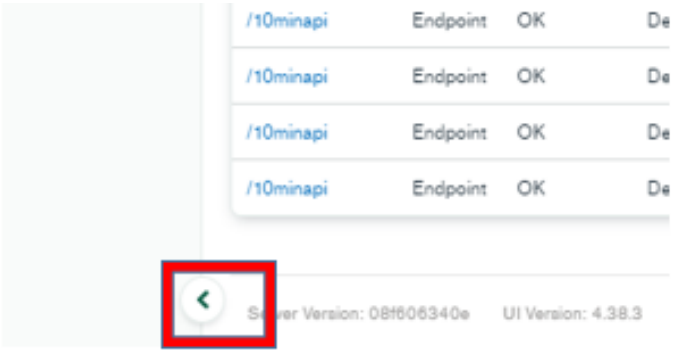


Figure 5.6: Side menu button in MongoDB

Figure 5.7: MongoDB Function Setting

- If a side menu containing options like **DATA Access**, **BUILD**, **MANAGE** doesn't opens, then scroll down till end of page and click on the side menu arrow button as shown in figure Fig. 5.6

5.1.2.5 Creating a Realm Function

- After Creating Realm App successfully, open the app in MongoDB Realm
- In the side menu click on the button **Functions**
- Click on **Create New Function** option
- Change the settings as shown in figure Fig. 5.7
- Write the Function in **Function Editor** and click on **Save draft**
- Click on **Deploy** option at the top
- We can run and Test the function by using the console provided on the button

NOTE: It should be kept in mind that while running from console, to refer to the argument passed to function, we use **payload** in function and while accessing it from our boards and apps we use **payload.body.text()**

5.1.2.6 Creating an HTTPS ENDPOINT and connecting to function

- After Creating Realm App successfully, open the app in MongoDB Realm
- In the side menu click on the button **HTTPS Endpoints**
- Click on **Add an Endpoint** option
- Change the settings as mentioned below
  - **Route**: Add Route name starting with a backward slash
  - **Enabled**: Turn ON
  - **Operation Type**: Copy the **Endpoint URL** and save it for future

  - **HTTP Method**: Choose the method of http request required for this endpoint. In this project all the endpoints are **POST**
  - **Respond With Result**: Turn ON
  - **Return Type**: Select EJSON
  - **Function**: Add the function to be executed when this endpoint is called
  - **Authorization**: Select **No additional authorization**
  - **User Settings**: All turned off

- Click on **Save Draft**
- Click on **Deploy** option at the top

5.1.2.7 Creating charts in MongoDB

- After creating Cluster, click on the **Charts** Button on the top panel
- Create a new dashboard by clicking on **Add Dashboard** button
- Click on **Data Sources** option in left menu panel and add the **collection** required by clicking on **Add data source** option
- Open the dashboard by clicking on it
- Add a Chart Service by clicking on **Add chart** option
- Select the Data source in **Choose Data Source** option
- Edit the options and click on **Save and close**

5.1.3 User Manual for User App

5.1.3.1 Adding an Entry

- Step 1: Open the User app and select Add Entry button
- Step 2: New Screen will open. Fill in the necessary information and set a new password. After that, select the Scan TAG button on-screen, QR Scanner app will open. Scan QR code provided above the Tag. Select Add Entry Button to make a successful entry and complete the registration process. Note:

  - User must have installed any QR code scanner app from play store in order to scan TAG - QR code.
  - User must remember Tag id and Password for the login process.

- Step 3: After successful registration, another screen will open where you can monitor the status of your Asset. In case of any crash, users can use the Tag id and Password as their login credentials to connect again. Asset status shows whether our Asset is Safe or not. When the color of Asset Status is Green, and it is marked as safe, then our Asset is still connected and safe. When this becomes Red and marked as unsafe, the user must inform the Conductor as their Asset is at risk.

5.1.3.2 Connecting to added TAG

- Step 1: Open ble user app and select Connect button
- Step 2: Another screen will open. Fill in TAG ID and Password used in the registration process and click on Connect button.
- Step 3 After Successful connection, a new screen will open where you will be able to monitor Asset Status continuously. This is the same screen you get after successful registration

5.1.3.3 Removing Entry of a TAG

- Step 1: Open ble user app and select the Remove Entry button
- Step 2: A new screen will open. Enter TAG ID and Password and click on Remove Entry Button

Note: Once an entry is removed, the Tag will be unlocked, and Asset will be free. Once removed can't be added again without Scanning the TAG QR code. This functionality must be used at the time of Deboarding the bus.

5.1.4 User Manual for Conductor App

5.1.4.1 Creating New Employee Account

- Step 1: Open ble conductor app and select Create New Account button

- Step 2: Another screen will open as shown in figure
- Step 3: Fill necessary information and Generate OTP. You will receive an OTP in the provided contact number. Enter OTP.
- Step 4: Click on the Scan Authorization card Button, QR code scanner will open. Scan the authorization card. After successful Authorization, the Button color will change to Green.
- Step 5: Set New Password. Use the checkbox to make Password Visible.
- Step 6: Click on Sign-up Button to make an Entry.

5.1.4.2 Logging in

- Step 1: Open conductor App and select on login Button.
- Step 2: A new screen will open up.
- Step 3: Enter Employee id and click on Next Button.
- Step 4: A new Password box and a sign-in button will appear. Enter Password and click on the Sign-in button.
- Step 5: A new screen will open where you can monitor the status of all the Assets.

5.1.4.3 Removing Entry

- Step 1: Open conductor app and click on Remove Entry button
- Step 2: Another screen will open as shown in the figure. Enter Employee id and click the Next button.
- Step 3: Enter Password and Click on Remove ID Button

5.1.4.4 Forgot Password

If employees forget their password, they can use the Forgot Password option to set a new password.

- Step 1: Open ble conductor app and select Login Button

- Step 2: Click on the Forgot Password button at the bottom
- Step 3: A new screen will open up as shown in figure
- Step 4: Enter Employee id (Emp xxx)
- Step 5: Click Generate OTP to receive OTP on the registered mobile number
- Step 6: Enter New Password
- Step 7: Enter OTP received on the registered mobile number
- Step 8: Click on SET PASSWORD Button

5.2 Corrections

This section will contain any last minute correction that would have been undocumented in the engineering and fabrication chapter. Till now we don't have any correction.

5.3 Suggestions for next gen

The following points are suggested to the next generation:

- The information of settings and options provided regarding the **Cloud Database** initialization may not be available with updates. But similar kind of settings must be available in the website and the documentation page.
- It should be kept in mind that the GPS module testing must be performed in the open sky environment. GPS module cannot receive signal in an indoor environment.
- The informations mentioned in the header file secrets contains our specific WiFi and MongoDB credentials. It must be modified accordingly.
- Following the steps mentioned in 5.1, one can easily setup the essentials of this project.

Bibliography

[1] Adriano Moreira, Maria João Nicolau, António Costa, and Filipe Meneses. Indoor tracking from multidimensional sensor data. In *2016 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, pages 1–8, 2016.

[2] Michael Stocker, Bernhard Großwindhager, Carlo Alberto Boano, and Kay Römer. Towards secure and scalable uwb-based positioning systems. In *2020 IEEE 17th International Conference on Mobile Ad Hoc and Sensor Systems (MASS)*, pages 247–255, 2020.

[3] Zhu Jianyong, Luo Haiyong, Chen Zili, and Li Zhaohui. Rssi based bluetooth low energy indoor positioning. In *2014 International Conference on Indoor Positioning and Indoor Navigation (IPIN)*, pages 526–533, 2014.

[4] Mohammadreza Akbari. Esp32 bluetooth low energy (ble) on arduino ide. `https://electropeak.com/learn/esp32-bluetooth-low-energy-ble-on-arduino-ide-tutorial/`. [Online; accessed 20-Dec-2021].

[5] Santos Sara. Esp32 publish data to cloud. `https://randomnerdtutorials.com/esp32-sim800l-publish-data-to-cloud/`. [Online; accessed 8-Mar-2022].

[6] Mohamed Shameer. How to send sms from gsm module to mobile. `https://create.arduino.cc/projecthub/shameermohamed/how-to-send-sms-from-gsm-module-to-mobile-881569`. [Online; accessed 2-Apr-2022].

[7] Sharma Ruchir. How to interface gps module (neo-6m) with arduino. `https://create.arduino.cc/projecthub/ruchir1674/how-to-interface-gps-module-neo-6m-with-arduino-8f90ad`. [Online; accessed 3-May-2022].

[8] Muchika. Vehicle tracking system based on gps and gsm. https://create.arduino.cc/projecthub/muchika/vehicle-tracking-system-based-on-gps-and-gsm-57b814. [Online; accessed 16-May-2022].