

Sri Lanka Intelligent Bus Navigation and Passenger Information System

Rajapaksha R.L.H.P

IT21192050

B.Sc. Special (Honors) Degree in Information Technology
Specialization in Information Technology

Department of Information Technology

Sri Lankan Institute of Information Technology
Sri Lanka

August 2024

Sri Lanka Intelligent Bus Navigation and Passenger Information System

Rajapaksha R.L.H.P

IT21192050

B.Sc. Special (Honors) Degree in Information Technology
Specialization in Information Technology

Department of Information Technology

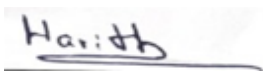
Sri Lankan Institute of Information Technology
Sri Lanka

August 2024

DECLARATION

I declare that this is my own work and this dissertation¹ does not incorporate without acknowledgement any material previously submitted for a Degree or Diploma in any other University or institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text.

Also, I hereby grant to Sri Lanka Institute of Information Technology, the non exclusive right to reproduce and distribute my dissertation, in whole or in part in print, electronic or other medium. I retain the right to use this content in whole or part in future works (such as articles or books).

Name	Student ID	Signature
Rajapaksha R.L.H.P	IT21192050	

The above candidates are carrying out research for the undergraduate Dissertation under my supervision.

Signature of the Supervisor
(Dr Shanika Wijesekara)

Date

ABSTRACT

Public transport bus services, in particular plays a critical role in daily commutes for millions of people in Sri Lanka. But issues of indiscriminate competition among buses, poor communication between the passengers and the drivers, and overloading of buses have greatly impacted the reliability and safety of the system. To overcome these, this study proposes an IoT-based Smart Bus Monitoring System improving passenger comfort, safety, and operational performance through low-cost, real-time technologies.

The system is developed using an ESP32 TTGO board combined with a SIM800L module for internet access, a GPS module for bus location tracking, an LCD screen for real-time driver feedback, and ESP32-CAM (OV5640) for passenger status checking. It continuously monitors the location of the bus and determines the distance between buses running on the same route through data in Firebase and Google Cloud. One of the salient features of the system is the onboard LCD screen that indicates the distance to the lead bus and the number of waiting passengers at the next stop—information gathered through a cell phone application where passengers provide the intention to board. It enables the driver to keep safe distances and halt accordingly.

In addition, ESP32-CAM records real-time images of the interiors to estimate seat availability, which are then uploaded to Firebase, enabling passengers to see the level of crowds through a smartphone application prior to boarding. Data transmission is made efficient in accordance with the HTTP protocol to ensure smooth operation even in cases of low network availability. Field testing of the system showed that it enhances safety by means of greater awareness among drivers, less competition among buses, and a hassle-free experience for passengers. This work assists in the design of safer, intelligent public transport infrastructure and poses a scalable answer appropriate for deployment in other parts of Sri Lanka's bus network.

Keywords -ESP32, SIM800L, GPS, Google Cloud, HTTP

ACKNOWLEDGEMENT

First and foremost, I owe a sincere thank you to my supervisor, Dr. Shanika Wijesekara. Her steady guidance, thoughtful feedback, and constant encouragement made a lasting impact on this work. She helped shape not just the research but also how I approached each challenge that came with it.

I'd also like to thank my co-supervisor, Dr. Kapila Dissanayake. His technical advice and attention to detail helped me navigate several of the more difficult parts of this project. When unexpected hurdles popped up, he was there with practical solutions and the kind of support that made a real difference.

This project crossed into several different fields—real-time IoT systems, embedded tech, cloud services, and transportation systems. I'm deeply grateful to the professionals, both academic and from the industry, who shared their time and expertise to help me through tricky problems and unfamiliar concepts. Their input was essential during the implementation and testing stages.

I'm especially thankful for my peers and team members. The shared ideas, back-and-forth discussions, and support along the way made this journey less daunting. Whether it was a tough technical issue or just a moment of doubt, having this team made the difference between moving forward and getting stuck. Your help was invaluable.

Table of Contents

DECLARATION	3
ABSTRACT	4
ACKNOWLEDGEMENT	5
LIST OF FIGURES	8
LIST OF TABLES	9
LIST OF ABBREVIATIONS	10
1. INTRODUCTION.....	11
1.1 General Introduction	11
1.2 Background literature	12
1.2.1 Background of Public Transportation in Sri Lanka.....	12
1.2.2 The Need for Technological Interventions	13
1.2.3 IoT and Real-Time Monitoring Technologies in Transportation.....	13
1.2.4 Vision-Based Monitoring for Public Transport: A Literature and Practice Review	13
1.2 Research Gap	14
2. RESEARCH PROBLEM.....	18
3. RESEARCH OBJECTIVES	20
3.1 Main Objectives	20
3.2 Specific Objectives	20
4. METHODOLOGY	22
4.1 Materials and methods	22
4.1.1 Problem Statement	23
4.1.2 Component System Architecture (Solution Design)	24
4.1.3 Diagram of the Distance calculating and Passenger count	26
4.1.3 Google Cloud Platform(GCP)	27
4.1.4 Logic of the forward bus distance getting -	28
4.1.5 Design Diagrams	30
4.2 Commercialization Aspects of the Product	33
5. Testing & Implementation	34
5.1 Hardware Implementation	34
5.2 Backend Architecture.....	34
5.3 Google Maps API Integration	35

5.4 Internet and Communication.....	35
5.5 GPS data Manipulate.....	37
5.6 LCD display.....	40
5.7 Camera connection	41
5.8 Distance calculation with Google map platform.....	43
5.9 Hosted backend.....	44
5.2 Testing.....	47
5.2.1 Test Planning and Strategy	48
5.2.2 Test Case Design.....	49
6. RESULTS AND DISCUSSIONS	56
6.1 Results.....	56
6.1.1 Real-Time Bus Location Tracking and Data Transmission	56
6.1.2 Distance Calculation Between Buses.....	56
6.1.3 Passenger Presence Detection at Bus Stops	57
6.1.4 Bluetooth-Based Image Transfer and Firebase Upload	58
6.1.5 Performance Evaluation of System Modules	58
6.1.6 Summary of Evaluation Metrics.....	58
6.2 Research Findings.....	58
6.3 Discussion.....	59
7. CONCLUSION	61
8. REFERENCES	63
9. APPENDICES	66

LIST OF FIGURES

Figure 1.1: Survey report on 2022 Bus accidents.....	11
Figure 4. 1: Overview System Diagram	22
Figure 4. 2 : Overview of component diagram	24
Figure 4. 3: System diagram Distance calculating and the Passenger count	26
Figure 4. 4: Database Structure of the Current data Stored.....	28
Figure 4. 5: System diagram for ESP32 Cam Module working connection with the TTGO Board	29
Figure 4. 6: Sequence Diagram of IOT Device	31
Figure 4. 7: Use case diagram of IOT Device	32
Figure 5. 1 : Pin connection of the Sim800L module code snip.....	36
Figure 5. 2: Internet Settings of the Sim module code snip.....	37
Figure 5. 3: Sim module initialization code snip	37
Figure 5. 4: Variable declarations and pin initializing of the GPS Module	38
Figure 5. 5: Gps modem data getting and Initializing code snip	39
Figure 5. 6: Gps modem data getting and send to the server code snip	40
Figure 5. 7: LCD display initialization code snip	41
Figure 5. 8: LCD display Current data showing code snip	41
Figure 5. 9: LCD display Actual working type.....	41
Figure 5. 10: Camera connection and the Pin initializing	42
Figure 5. 11: Image stored in firebase storage.....	43
Figure 5. 12: Distance calculating function with Google maps platform API	44
Figure 5. 13: Firebase function and initializing.....	45
Figure 5. 14: All API's on backend Server	46
Figure 5. 15: Hosted Backend Server in DigitalOcean platform	47
Figure 6. 1: Live database updating	56
Figure 6. 2: Distance calculating sample output	57
Figure 6. 3: Passengers count show in LCD Display.....	57

LIST OF TABLES

Table 1.1 : Comparison of former researches.....	16
Table 4. 1: Technologies, techniques and architectures used.....	30
Table 5. 1: Test case to send GPS location to backend.....	49
Table 5. 2: Test case to calculate distance to forward bus	50
Table 5. 3: Test case to display passenger count on LCD	50
Table 5. 4: Test case to transfer image from ESP32-CAM via Bluetooth	51
Table 5. 5: Test case to upload image to Firebase Storage.....	51
Table 5. 6: Test case for mobile app passenger waiting signal.....	52
Table 5. 7: Test case to retrieve forward bus distance on LCD	52
Table 5. 8: Test case to capture image at interval.....	53
Table 5. 9: Test case to handle no GPS signal	53
Table 5. 10: Test case to test HTTP data reliability	54
Table 5. 11: Test case for Firebase real-time sync	54
Table 5. 12: Test case for Distance Matrix API error handling.....	55

LIST OF ABBREVIATIONS

Abbreviation	Description
GPS	Global Positioning System
IOT	Internet of Things
HTTP	Google Cloud Platform
ESP	Espressif Systems
LCD	Liquid Crystal Display
SIM	Subscriber Identity Module
API	Application Programming Interface
GCP	Google Cloud Platform
BLE	Bluetooth Low Energy
RTDB	Realtime Database (Firebase)
PWM	Pulse Width Modulation
RTC	Real-Time Clock
MCU	Microcontroller Unit
JSON	JavaScript Object Notation
CLI	Command-Line Interface
GSM/GPRS	Global System for Mobile / General Packet Radio Service
RX/TX	Receive/Transmit Pins
MCU	Microcontroller Unit
UART	Universal Asynchronous Receiver Transmitter

1. INTRODUCTION

1.1 General Introduction

In Sri Lanka, public transport—particularly buses managed by the state-owned Sri Lanka Transport Board (SLTB) and private operators—is a core service that millions of citizens rely on for daily commutes. Yet, the industry is marred by various chronic problems: careless competition among buses, overloading, irregular arrival times, and a lack of clear communication between passengers and drivers in real-time. These concerns have led to not only a substandard passenger experience but also a high rate of road accidents.

A Ministry of Transport and Highways study (2022) sheds light on the severity of the issue. SLTB buses were implicated in 59 fatal and 272 minor accidents, while private buses had 201 fatal and 632 minor accidents. These statistics sharply indicate how the lack of organized, smart transport surveillance has undermined public safety[1].



MINISTRY OF
TRANSPORT AND HIGHWAYS



MINISTRY OF
TRANSPORT AND HIGHWAYS

SLTB Buses	Fatal	59	29	37	63
	Minor	272	168	86	157
	Critical	175	126	72	123
	Damages	187	68	65	101
Private Buses	Fatal	201	133	129	172
	Minor	632	367	247	397
	Critical	479	432	203	319
	Damages	789	265	285	359

Figure 1.1: Survey report on 2022 Bus accidents

Excessive competition for customers fuels reckless driving, causing many buses to dangerously overtake others, ignore traffic rules, and frequently get involved in collisions. On top of that, buses often skip waiting passengers or become heavily overcrowded due to insufficient real-time load data, introducing discomfort and delays for travelers[2].

To address these pressing transportation problems, this study proposes the design of an IoT-based Smart Bus Monitoring System. The system aims to boost safety, reduce competition, and enhance passenger comfort by utilizing affordable hardware and cloud-based data management. It incorporates:

- An ESP32 TTGO board combined with a SIM800L module for internet connectivity

- A GPS module for real-time tracking of bus locations
- An LCD display in the driver's cabin to show the distance to the bus ahead and the number of upcoming passengers
- An ESP32-CAM module to assess occupancy levels through real-time image capture and processing [3]

All data is transmitted via cellular networks and stored in Firebase, supporting real-time interaction between the bus system and its users through a mobile app. Key features include:

- Driver notifications about the number of passengers at the next stop
- Monitoring of safe distances between buses
- Real-time feedback on bus crowd levels through the passenger app

By integrating IoT devices, cloud infrastructure, and real-time data analysis, this system delivers a cost-effective, scalable, and passenger-focused solution to many of the issues affecting Sri Lanka's public bus network.

1.2 Background literature

1.2.1 Background of Public Transportation in Sri Lanka

Public transport, in the form of both state-owned Sri Lanka Transport Board-run and private-run buses, is central to the lives of millions nationwide. Its extensive availability notwithstanding, the public bus system of Sri Lanka suffers from inefficiencies and safety risks that are brought about by antiquated infrastructure, insufficient real-time communications, and the lack of digital monitoring.

The intense competition among bus drivers, where passenger collection comes prior to road safety, is one of the most acute of all the concerns. Reckless driving practices, such as speeding, inappropriate overtaking, and a disregard of traffic rules, frequently result when there is competition. What ensues is a considerably high number of bus-related accidents on the roads.

1.2.2 The Need for Technological Interventions

Across developed and emerging economies, Internet of Things (IoT) technologies are extensively utilized to boost public transportation systems. IoT allows vehicles to interact in real time with central systems as well as other devices, making it possible for applications such as:

- Real-time vehicle tracking
- Passenger load monitoring
- Predictive ETA (Estimated Time of Arrival)
- Safe driving alerts
- Crowd management and dynamic routing

Yet, there remains a wide technology gap in Sri Lanka. The majority of bus services operate using manual inputs, fixed timetables, and static route boards. Hence, both passengers and drivers lack real-time situational awareness, resulting in bad decision-making and compromised safety[4].

1.2.3 IoT and Real-Time Monitoring Technologies in Transportation

Advanced developments in Internet of Things (IoT) and cloud computing technologies have made possible the design of intelligent transportation systems (ITS) that enhance efficiency, prevent accidents, and optimize citizens' experience of public commutes. Smart bus monitoring systems across the world have utilized microcontroller-based configurations, which frequently involve GPS, GSM modules, and cloud-based architectures to execute location tracking, passenger data gathering, and driver notifications[5].

Research in transportation-oriented IoT deployments indicates that several of them use low-energy microcontrollers like ESP32 for edge computing, alongside GSM/GPRS devices like SIM800L for providing mobility in the lack of fixed Wi-Fi support. These deployments tend to gather bus telemetries like GPS location, engine status, and passenger load data, and then forward it to the cloud for analysis and decision-making.

This implementation employs HTTP-based communication to convey data using a Firebase Realtime Database, made simpler and ideal for small-to-medium sized systems. HTTP is a simple, widely-supported protocol enabling ESP32 to send GPS reports, passenger count notifications, and get command replies back in the cloud[6].

1.2.4 Vision-Based Monitoring for Public Transport: A Literature and Practice Review

With increasing real-time surveillance and passenger management in public transport systems, camera-based observation is increasingly becoming a feasible method of measuring crowd density and seat availability. In cases where onboard sensors are too expensive or unreliable,

low-cost vision modules such as ESP32-CAM now represent a viable option. Internationally, certain publications have suggested computer vision methods involving Convolutional Neural Networks (CNNs) and object detection methods (e.g., YOLO, Mask R-CNN) for the detection of humans and crowd estimation in buses and trains. Although successful, these tend to necessitate GPU-based edge computing and are therefore too expensive to be deployed in mass in low-income countries[7].

In the system proposed, a cost-saving and feasible approach is taken through the use of an ESP32-CAM module (OV5640). It takes photographs of the interior of the bus at intervals and transmits them using the HTTP protocol to Firebase Storage, where they are available in real-time for analysis.

In this particular application, no deep learning or in-device processing is required, which keeps complexity and resources low. Such an approach achieves a balance between technical viability and user benefit, particularly in places such as Sri Lanka where device affordability, network reliability, and simplicity are significant limiting factors.

1.2 Research Gap

In Sri Lanka, public transport—especially regular bus service operated by the Sri Lanka Transport Board (SLTB) and private operators—continues to be the most widely adopted mode of daily transportation for millions. The system is, however, marred by numerous inefficiencies and safety issues owing to the lack of smart technology incorporation. Of these, one of the primary shortcomings involves the lack of real-time communication between waiting passengers and bus drivers approaching near bus stops. Presently, drivers lack any means of ascertaining the number of waiting people at the next stop or if any wait there in the first place[8]. This leads to regular instances of service inefficiencies, including the abandonment of passengers and making stops unnecessarily, resulting in wasted fuel, longer delays, and passenger complaints.

The issue becomes even more critical in peak times when congestion and time pressure make it increasingly difficult to keep passenger boarding conditions in check. Most passengers in suburban or rural communities, in particular, are often neglected by buses as a result of this communication defect, which affects not only accessibility but also evokes issues of public service equality. Finally, in a nation where public transport is critical for everyday transportation of pupils, workers, and the elderly, service breakdowns actually impede peoples' access to their destination reliably[9].

In spite of the fast pace of global advancements in smart transportation systems, there is no literature and practical implementations that deal with this particular driver-passenger interaction shortfall. Although certain systems use route tracing, automated payment collection, or electronic signboards, none of them provide a solution to inform bus drivers of incoming boarding requests in real-time. It indicates a critical study and implementation shortfall in the case of developing nations such as Sri Lanka.

A feasible approach might include a system where travelers use a smart application that signals their waiting status at a bus stop, and the driver gets a real-time update on a dashboard LCD screen on the bus, letting them know how many travelers are waiting[10]. Such a modest yet significant feature can increase service efficiency, minimize boardings that are missed, and make the overall experience even better for the public. But nothing of the kind exists in local implementations or in the literature.

Concurrently, one of the other foremost safety concerns plaguing Sri Lanka's roads for a long time is careless overtaking by buses, especially on busy or competitive routes[11]. It is not rare for buses to tailgate and overtake each other in a desperate attempt to pick up additional passengers. It causes a significant number of road accidents, putting both pedestrians and passengers in harm's way. One of the solutions is bringing on-board an LCD module that calculates the real-time distance to the next ahead-moving bus using GPS coordinates and projects it in real-time. Through feedback, the drivers would be able to keep safe margins, refrain from unnecessary overtake maneuvers, and increase safety on the roads. Again, this safety-conducive idea is uncharted in existing systems.

In response to these shortcomings, we introduce the IoT-based Smart Bus Monitoring System that utilizes a combination of GPRS, GPS, cloud platforms, and camera capabilities. Fundamentally, the system is based on a TTGO ESP32 board that comes integrated with a SIM800L GSM/GPRS module, enabling real-time data exchange in the form of HTTP requests. Utilizing HTTP over SIM card-based 3G[12] cellular networks makes it Wi-Fi-independent in operation for bus-level applications, which are usually devoid of Wi-Fi networks in mobility scenarios. In the HTTP provides lightweight, stateless RESTful communication that is ideal for bus-level applications where bandwidth, power, and simplicity are important.

Every bus makes regular HTTP POST requests to the Firebase Realtime Database with real-time location data (latitude and longitude). We can then obtain the distance between buses on the same route using either a cloud-based function or a backend for a mobile app and return it

to the driver's dashboard using HTTP GET requests. Simultaneously, waiting passengers can alert their presence using the mobile app, which is also persisted in Firebase. We then get this count on the bus and show it on the in-vehicle LCD so the driver is better equipped to approach the stop.

In order to avoid overcrowding and ensure greater onboard comfort, the system also includes a low-cost ESP32-CAM (OV5640) camera module that periodically takes pictures of the inside of the bus. These are sent to Firebase Storage through authenticated HTTP requests, allowing for remote checking of crowd levels by transport authorities or backend systems. It allows proactive actions like instructing drivers not to board previously full buses or routing upcoming buses to congested sections[13]. The system is made to be modular, cost-efficient, and scalable. HTTP REST API integration with Firebase provides smooth data exchanges even in regions that may have poor or spotty signal strength, particularly when combined with 3G-enabled SIM cards.

Table 1.1 : Comparison of former researches

<p>Research A - IoT-based bus monitoring system using a mobile app for real-time tracking of bus location, seat availability, and passenger activities, enhancing service efficiency and management[7].</p> <p>Research B - IoT-based system improves Sri Lanka's public transport with real-time bus tracking, automated ticketing, and smart card authentication for enhanced service efficiency[8].</p> <p>Research C- IoT-based GPS vehicle tracking and theft detection system using Google Cloud IoT Core and Firebase for real-time location monitoring and theft alerts via a web interface[9].</p>	Application Reference	Displaying the distance between a bus and the bus ahead of it.	Displaying the passengers boarding from the next stop.	Passengers can get an idea about the seat availability of the bus.	Use HTTP for pass the data	Connect the IoT device to the internet using a GSM/GPRS module.
	Research A	✗	✗	✗	✗	✗
	Research B	✗	✗	✓	✗	✓
	Research C	✗	✗	✗	✓	✓
	Proposed System	✓	✓	✓	✓	✓

Although a number of mobile apps exist today that can be used to support public transportation in Sri Lanka, these applications are generally restricted to static bus routes, approximated arrival times, or simple location tracking. These solutions provide very limited interactivity and tend to fail to deal with the fundamental problems that both passengers and operators of buses experience in real-time settings. What we introduce in this paper, by contrast, is a complete, IoT-based smart mobile application providing much greater capabilities for real-time observation, communication, and crowds management in public buses. Our system is one of the innovative aspects of which lies in providing real-time crowd density data through the use of a low-cost ESP32-CAM module on board the bus. It takes periodic images of the inside of the bus, which are uploaded to Firebase Storage through HTTP[15] requests over 3G SIM-based links. These images can be inspected manually by supervisors or analyzed through future machine learning models to establish occupancy levels. While other apps rely on passenger inputs for seat availability, the system presents objective, pictorial confirmation of the state of the bus, ensuring improved decision-making by both passengers and transport departments.

Besides providing improved driver benefits, the mobile application also informally alerts buses of waiting passengers ahead of stops. Through a single tap on the application interface, a passenger can show his/her boarding intention, which is sent in real-time to the driver's onboard liquid crystal display screen. It thus protects buses from skipping stops in front of waiting passengers, minimizes unnecessary stops, and maximizes the efficiency of the route. It also keeps the driver well advised of future passenger demand, which no other existing mobile app solutions address. Our system also includes predictive seat availability capabilities, which allow the app to notify travelers if a bus is full or almost full. The advance notification allows travelers to make better journey choices and minimizes boarding conflicts. Through the combination of cloud-based analytics and real-time information from the onboard IoT device, the mobile app provides a clearer and more responsive transportation experience.

In addition, by taking advantage of 3G connectivity through SIM800L, our mobile application provides faster and more reliable data transmission as opposed to Wi-Fi-based systems. Hence, the system is very reliable, particularly in rural or outdoor environments where Wi-Fi availability cannot be assured. Through the use of HTTP-based REST APIs in communicating with Firebase, the system boasts a simple yet robust backend architecture, providing for easy scaling, support, and reduced power usage—perfect for deployment over mobile devices[16].

At its core, our application is unique in that it fills the void between driver and passenger, enhances crowd sighting of buses, and maximizes the passenger experience. It transforms the passive bus ride of the past into an intelligent, interactive experience defining passenger engagement in real-time. These advancements make our system a strong, cost-effective, and scalable one that's ideal for the needs of contemporary urban and rural public transit in Sri Lanka[17].

2. RESEARCH PROBLEM

Public transportation systems, especially buses, face several challenges that can significantly impact their efficiency and reliability. Among these, bus overcrowding, intense competition among operators, and the lack of real-time monitoring and management of passenger loads are critical issues that must be addressed to ensure a reliable and safe public transportation system. Tackling these challenges is crucial for maintaining the efficiency and profitability of public transit operations, while also ensuring passenger satisfaction and safety. One of the main issues affecting bus transportation is overcrowding, which diminishes both the productivity and comfort of the system. Overcrowding can lead to delays, safety concerns, and an overall unpleasant experience for passengers. This problem is especially common in large-scale public transportation networks, where buses operate continuously throughout the day to meet the demands of a growing population. When buses become overcrowded, it not only inconveniences passengers but also adds extra strain on bus operators and the transportation infrastructure as a whole.

Another significant issue is the intense competition among bus operators. In many public transportation systems, particularly in Sri Lanka, there is fierce competition among buses to pick up as many passengers as possible, often leading to reckless driving behavior. Buses frequently try to overtake one another to get ahead and maximize their profits by picking up more passengers. This competitive behavior can create dangerous driving conditions, resulting in a higher incidence of accidents. These accidents not only pose a significant risk to passengers and other road users but also contribute to delays and disruptions in the transportation system. Additionally, the lack of real-time monitoring and management of passenger loads is a major challenge in public bus transportation. Without the ability to monitor how many passengers are waiting at the next stop, bus drivers often operate without crucial information that could help them manage their routes more effectively. This can lead to buses passing by stops without picking up passengers or becoming overcrowded because drivers are unaware of the number of passengers they need to accommodate. This issue is particularly problematic during peak hours when the demand for public transportation is highest, leading to passengers missing their desired bus and experiencing significant delays in their commute.

Moreover, the lack of awareness among bus operators and passengers about the availability of advanced tools and technologies that can monitor bus occupancy levels and predict crowding exacerbates these issues. Many existing public transportation systems have not yet adopted such technologies, which could help mitigate the problems of overcrowding and unsafe driving practices. This gap in technology usage means that buses often become overcrowded, and drivers are left to make decisions without sufficient information, leading to inefficiencies and increased safety risks[18].

To address these challenges, it is essential to implement advanced monitoring systems that enable real-time data collection and analysis of passenger loads. These systems could utilize technologies such as GPS, GPRS modules, and camera modules to track the number of passengers boarding at each stop and provide this information to bus drivers in real-time.

By equipping buses with small LCD screens that display the distance to the vehicle ahead, as well as the number of passengers waiting at upcoming stops, drivers can make informed decisions that enhance the efficiency and safety of their routes. This real-time monitoring system would help reduce the likelihood of buses becoming overcrowded and minimize the competition among bus operators, which often leads to dangerous driving practices.

Furthermore, educating both passengers and bus operators on the use of these advanced monitoring tools is crucial for improving the overall efficiency and safety of public transportation. By raising awareness about the benefits of these technologies, transportation authorities can encourage their adoption and ensure that both drivers and passengers are better prepared to handle the challenges of public transit. This education could include training programs for bus operators on effectively using the monitoring systems and public awareness campaigns to inform passengers about how these systems work and the benefits they offer.

In addition to implementing monitoring systems, addressing the issue of competition among bus operators is also important. Regulatory measures could be introduced to limit competition and ensure that buses operate according to safety guidelines rather than focusing solely on maximizing profits. These regulations could include strict penalties for reckless driving and overtaking, as well as incentives for bus operators who prioritize passenger safety and comfort over profit margins. By reducing the pressure on bus operators to compete for passengers, these measures would contribute to a safer and more reliable public transportation system[19].

In conclusion, public transportation systems, particularly buses, face several challenges that can significantly impact their efficiency and reliability. Overcrowding, intense competition among bus operators, and the lack of real-time monitoring and management of passenger loads are critical issues that must be addressed to improve the overall performance of the public transportation system. By implementing advanced monitoring systems, educating passengers and bus operators, and introducing regulatory measures to reduce competition, transportation authorities can enhance the safety, efficiency, and reliability of public bus transportation. These improvements will ensure that the public transportation system meets the needs of daily commuters and contributes to a more sustainable and profitable transportation infrastructure.

3. RESEARCH OBJECTIVES

3.1 Main Objectives

The main purpose of this research is to increase passenger comfort, safety, and operational efficiency in public bus transportation in Sri Lanka using a real-time, IoT-based smart monitoring system. The system is meant to give bus drivers, passengers, and transport officials instant, real-time information on important parameters like bus location, availability of seats, passenger load, and boarding intention of subsequent stops.

Through the use of low-cost, network-enabled IoT devices consisting of ESP32 TTGO boards, SIM800L modules, GPS, LCD screens, and ESP32-CAM cameras, this project aims to optimize the communication between passengers and drivers by minimizing accidents and wastage of resources. Real-time data collection, HTTP-based cloud communication, and dynamic live upgrades enable proactive decision-making and improved route management, culminating in enhanced public transport reliability, predictability, and overall user satisfaction.

3.2 Specific Objectives

To achieve the overarching goal of transforming the public bus system into a smarter, safer, and more responsive service, the project focuses on the following four specific objectives.

1. Improve Driver Awareness

First priority is improving the situational awareness of bus drivers by providing them with real-time data in the vehicle itself. They are constantly informed through a dashboard-mounted LCD screen of:

- Distance to next forward-moving bus on the same route (derived via GPS data on Firebase),
- The number of waiting passengers at the upcoming stop (derived from live passenger app feed),
- Bus load factor measurements according to images taken by the in-vehicle camera.

Such a stream of information allows for well-timed, informed decisions that enhance passenger safety and routing consistency. It also promotes safe driving, as there is less impulsive behavior on the driver's part when they know what is going on in front or ahead.

2. Minimize Bus Competition

Bus competition-risky overtaking or racing by drivers to pick up passengers ahead of the competition accounts for most of the accidents on the public transport system in Sri Lanka. This goal targets the application of real-time visibility of data to cut down on this phenomenon.

By showing the immediacy of other buses on the route and passenger wait figures for next stops, drivers no longer feel compelled to race each other aggressively. It promotes a cooperative and data-driven style of routing that encourages safety, punctuality, and fair distribution of passengers. As a result, it leads to a cultural change in driver behavior over time, decreasing accidents and enabling a secure means of commutation[20].

3. Track Seat Availability and Crowd Levels

Another of the most important problems in public transportation is overcrowding, which leads to passenger discomfort and also poses severe safety threats. To tackle this, the system also features a cost-efficient ESP32-CAM module that takes periodic images of the inside of the bus.

These images are uploaded through HTTP to Firebase Storage, where they can be accessed for administrator monitoring or future inclusion in AI-based crowd estimation models. Real-time visual data of the bus load allows for:

- Pro-active choices of whether or not to stop at a station,
- Updating passengers in the app if the bus is full or if there are seats available,
- Avoiding unsafe overloading conditions.

By providing this seat availability information to both commuters and drivers, the system is able to facilitate efficient load balancing and improved passenger comfort[21].

4. Enable Real-Time Data Sharing and Coordination

Optimized public transport depends significantly on smooth coordination and synchronization of data among all parties—drivers, passengers, and command centers. To this end, the objective is focused on implementing a real-time, cloud-based communication system through the utilization of HTTP protocols and Firebase solutions.

The system allows:

- Buses to upload GPS and camera data to Firebase using HTTP POST,
- Drivers acquire passenger wait times and send bus distance through HTTP GET,
- Travelers indicate their arrival at a stop through the smartphone app,
- Transport officials to monitor all bus activities in one administrative dashboard.

4. METHODOLOGY

4.1 Materials and methods

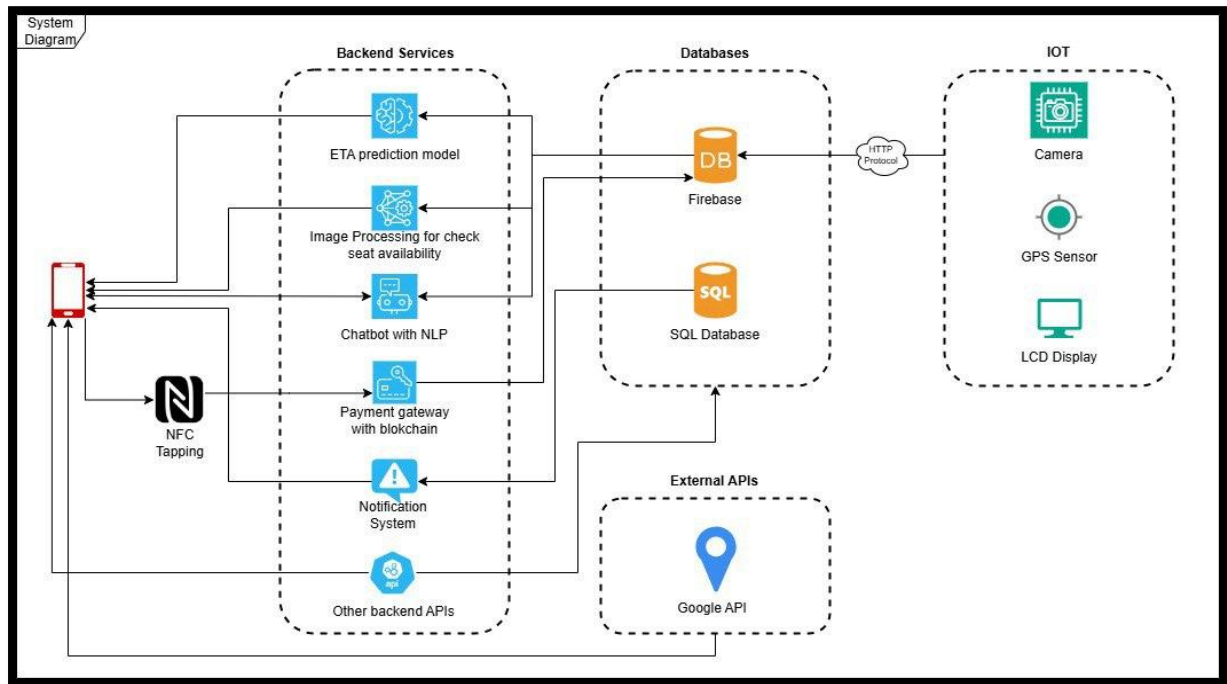


Figure 4. 1: Overview System Diagram

The system is a smart, real-time bus monitoring and management system built using IoT and cloud technologies. It is expected to enhance the public transportation system in Sri Lanka by improving the communication between passengers and the company, ensuring heightened safety, digital payment capabilities, and enhanced passenger experience-especially for tourists and commuters.

As shown in Figure 4.1, the system consists of four important connected components:

1. IoT-based Onboard Monitoring Unit
2. NFC-Based Payment and Authentication System
3. ETA Prediction and Real-time Bus Location Module
4. Seat Availability and Smart Tourism Guide

These pieces are integrated together through HTTP-based communication protocols, Firebase, and APIs of Google. Both the driver-side and passenger-side capabilities are supported by the whole ecosystem through cloud-synced infrastructure and a mobile app.

The Smart Bus Monitoring System offered is a real-time, IoT-based, modular system that encompasses onboard sensing, cloud analytics, contactless payment of fares, and user-centric passenger apps. In addressing both operational effectiveness and passenger comfort, this system is a complete end-to-end solution to the historic problems plaguing Sri Lanka's bus

transport system. Being scalable, cost-effective, and built on mobile-first architecture, it is well-suited for mass rollout across regional and intercity buses.

4.1.1 Problem Statement

This study addresses the long-standing problems in the public bus transport system in Sri Lanka pertaining to restricted real-time visibility and communication among bus operators, passengers, and bus drivers. One of the biggest concerns is that bus drivers remain unaware of how many passengers are waiting ahead at the next bus stops, which results in inefficiencies of the service in the form of skipped boardings, redundant stops, and overcrowding. This problem gets yet aggravated by the lack of a robust system to track seat availability, resulting in passenger annoyance, travel delays, and passenger dissatisfaction.

Furthermore, unsafe competition among buses-particularly those operating in the same corridor-raises safety issues. Buses often overtake one another at high speeds in a quest to pick up more passengers, which raises the risk of accidents. As it stands, there is a lack of any system to give drivers information about the distance to the leading bus in real-time, which would otherwise stimulate safe behavior and minimize aggressive overtaking.

This study also sees a lack in digital platforms for collecting fares, where manual ticketing delays processes and is non-transparent. Additionally, tourists availing the public transport system remain unaware of local sites of interest or culturally important stops along the journey, depriving them of a richer experience as in-transit guideline information is absent.

The overall purpose of this study is to design and deploy a smart IoT-driven system that optimizes public transport by giving real-time feedback to both passengers and drivers.

- Showcasing real-time seat availability and levels of bus congestion via onboard camera modules,
- Informing waiting passengers approaching stops by driver inputs through the mobile application,
- Informing drivers of proper following distances between buses,
- Providing a contactless payment method via NFC, and
- Giving route-specific information to guide tourists in their journey.

By addressing these issues using a cost-effective, real-time, HTTP-based IoT system, this study hopes to increase passenger comfort, safety, and service quality and provide a scalable system for mass deployment in the bus network of Sri Lanka.

4.1.2 Component System Architecture (Solution Design)

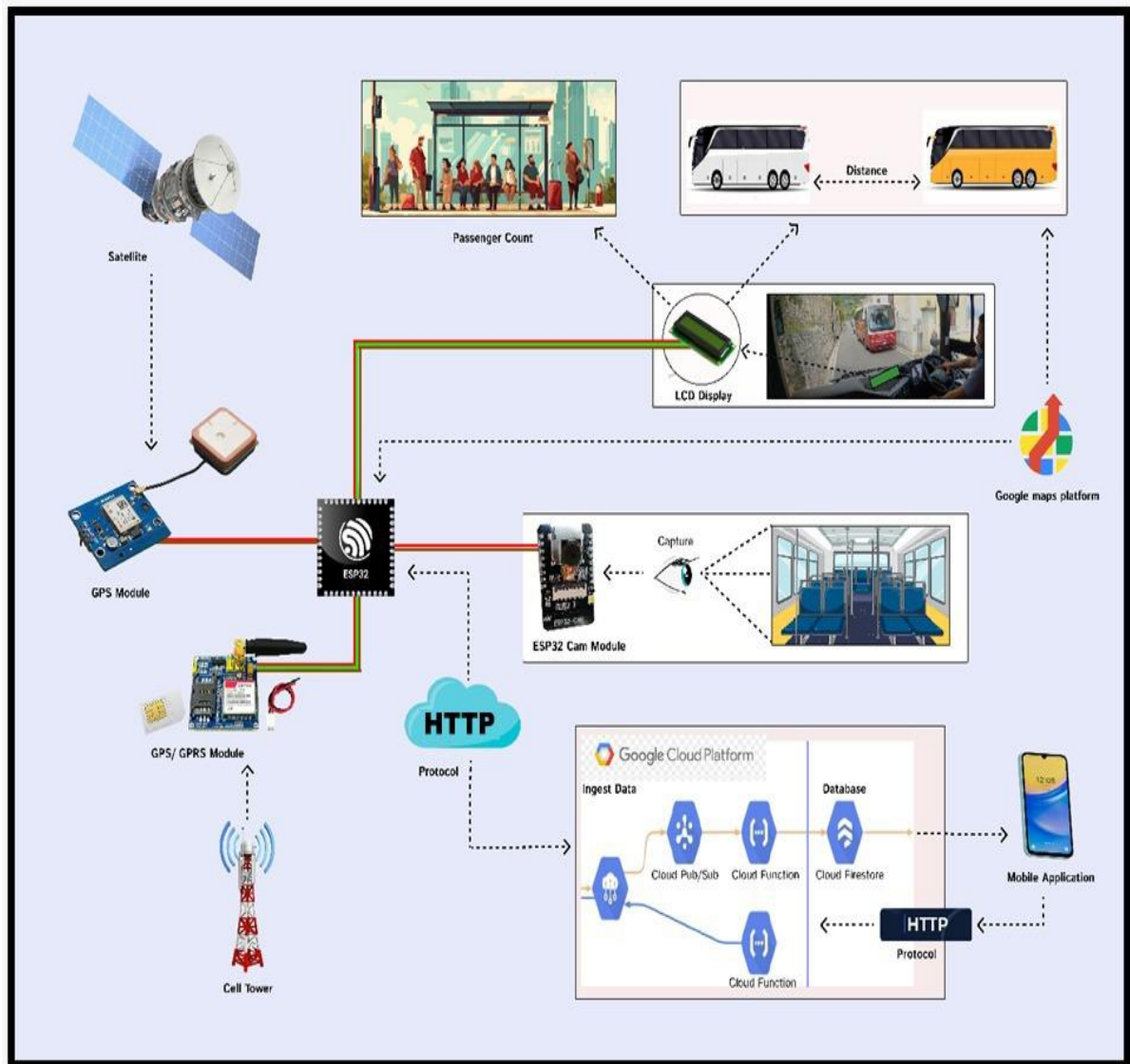


Figure 4. 2 : Overview of component diagram

This study addresses the long-standing problems in the public bus transport system in Sri Lanka pertaining to restricted real-time visibility and communication among bus operators, passengers, and bus drivers. One of the biggest concerns is that bus drivers remain unaware of how many passengers are waiting ahead at the next bus stops, which results in inefficiencies of the service in the form of skipped boardings, redundant stops, and overcrowding. This problem gets yet aggravated by the lack of a robust system to track seat availability, resulting in passenger annoyance, travel delays, and passenger dissatisfaction.

Furthermore, unsafe competition among buses—particularly those operating in the same corridor—raises safety issues. Buses often overtake one another at high speeds in a quest to pick up more passengers, which raises the risk of accidents. As it stands, there is a lack of any

system to give drivers information about the distance to the leading bus in real-time, which would otherwise stimulate safe behavior and minimize aggressive overtaking.

This study also sees a lack in digital platforms for collecting fares, where manual ticketing delays processes and is non-transparent. Additionally, tourists availing the public transport system remain unaware of local sites of interest or culturally important stops along the journey, depriving them of a richer experience as in-transit guideline information is absent.

The overall purpose of this study is to design and deploy a smart IoT-driven system that optimizes public transport by giving real-time feedback to both passengers and drivers.

- Showcasing real-time seat availability and levels of bus congestion via onboard camera modules,
- Informing waiting passengers approaching stops by driver inputs through the mobile application,
- Informing drivers of proper following distances between buses,
- Providing a contactless payment method via NFC, and
- Giving route-specific information to guide tourists in their journey.

By addressing these issues using a cost-effective, real-time, HTTP-based IoT system, this study hopes to increase passenger comfort, safety, and service quality and provide a scalable system for mass deployment in the bus network of Sri Lanka.

In the system suggested, a GPS module is important for obtaining the bus's real-time location. Location data underlies key features like live tracking, routing optimization, and driver dashboard notifications. The module sends the bus's position to a cloud backend in real-time, allowing for precise ETA calculation, driver awareness, and passenger transparency via mobile apps.

In order to provide internet connectivity, it employs a SIM800L GPS/GPRS module, which offers support for 3G networks using a normal SIM card. It enables the onboard IoT device to achieve stable internet connectivity even in regions of poor Wi-Fi signals. Mobile network-based communication ensures that there are uninterrupted, real-time transmissions of data between the IoT device and the cloud.

All of the bus's modules, including the GPS sensor, the LCD screen, and the camera, are networked using the central controller, the ESP32 TTGO board. ESP32 serves as the system's mind, controlling data collection, HTTP-based communication using Firebase, and regulating the flow of updates to and from every component. Its low power requirement, fast processing capability, and cost-effectiveness make it a cost-efficient and viable option for large-scale deployment across various buses.

For managing data in the cloud, the system utilizes Firebase Realtime Database and Firebase Storage. Firebase facilitates efficient and scalable syncing of data across devices, e.g., GPS location, passenger intent (from the app), and onboard sensor readings. HTTP POST and GET requests are made by the ESP32 device to communicate via Firebase, providing stable and low-complexity communications appropriate for mobile environments.

A OV5640-based ESP32-CAM module is mounted in the bus to monitor seat availability. It takes periodic images of the inside, which are stored in Firebase Storage. They provide both the transport authorities and passengers the opportunity to gauge passenger congestion and seat availability in real-time. OV5640 is chosen due to the quality of images, cost, and compatibility to operate continuously, which allows the system to ensure safety and passenger comfort based on visible cues.

4.1.3 Diagram of the Distance calculating and Passenger count

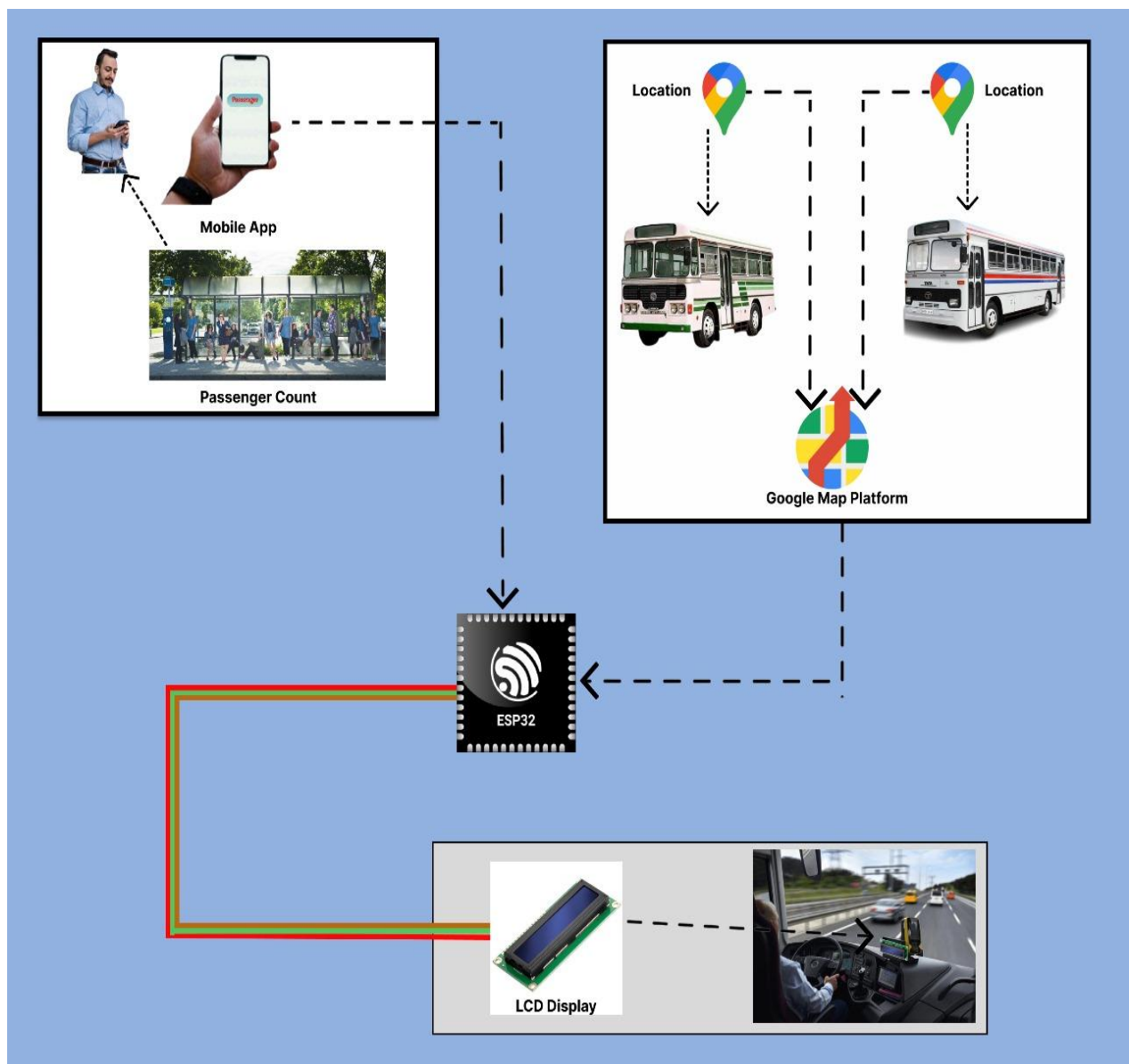


Figure 4. 3: System diagram Distance calculating and the Passenger count

The system that is proposed adds two primary features for increasing real-time coordination as well as safety in public transit.

First, it lets waiting passengers at the bus stop tap on an "I'm Waiting" button within a mobile app, which notifies the cloud through HTTP. That information is retrieved by the bus's onboard IoT device, and the waiting passengers' count is reflected on the driver's LCD screen, allowing the driver to make effective decisions about stopping.

Secondly, the system increases road safety through ongoing computation of distance between two successive buses based on GPS inputs fed into Firebase. Such computed distance is indicated on the LCD screen, allowing drivers to keep safe following distance and avoid dangerous overtaking maneuvers.

These two features combined enable improved driver awareness, avoided boarding, as well as safer, more efficient bus operations.

4.1.3 Google Cloud Platform(GCP)

Google Cloud Platform (GCP) is a robust and scalable portfolio of cloud technologies built by Google, intended for facilitating diverse computing requirements like data storage, machine learning, API management, and real-time analytics. In IoT-based intelligent transportation systems, GCP is instrumental in facilitating seamless interaction between hardware components and applications in the cloud.

It offers robust support for processing and analyzing real-time data coming out of GPS modules, camera sensors, etc., in IoT devices. Platforms like Google Maps API and Distance Matrix API can be leveraged for computing distance between moving buses as well as generating accurate predictions for ETA calculation.

GCP also supports Firebase, utilized in this project for real-time management of databases as well as storage in the cloud of images as well as location data. Its global coverage ensures high availability, minimal latency, as well as secure transmission of data, making it extremely suitable for mobile as well as location-based services.

Using GCP, the system assures sound performance, quick data access, as well as scalability-factors essential for deployment as well as management of intelligent bus monitoring solutions in a vast geographical area[22].

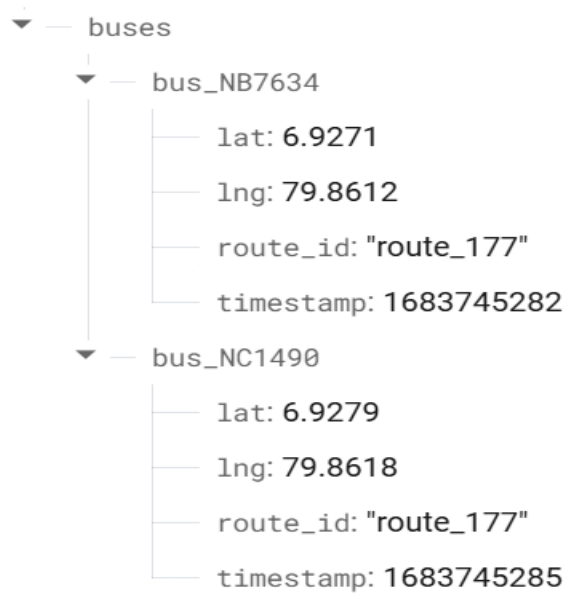


Figure 4. 4: Database Structure of the Current data Stored

One of the key features of the smart bus monitoring system is calculating the real-time distance between the present bus and the bus in front of it on the same route. It is an important feature for increasing road safety as well as minimizing reckless overtaking, the main reason for bus accidents in Sri Lanka. It uses GPS modules, Firebase Realtime Database, and the Google Maps Distance Matrix API in order to deliver accurate, ongoing distance feedback to the driver in real-time through an onboard LCD display.

4.1.4 Logic of the forward bus distance getting -

For safer driving habits and reduction of aggressive overtake, the system calculates continuously in real-time the forward-moving bus on the same journey and calculates the distance between that bus and the bus in real-time that distance is constantly updated and made visible on an onboard LCD for the awareness of the driver at any point in time

The reasoning for the distance calculation procedure is as follows:

GPS Data Collection

Every bus has a GPS unit that takes its position (latitude and longitude) at regular intervals (say, at intervals of 10–15 seconds). It sends this information about its position in the form of HTTP requests into Firebase Realtime Database along with a `bus_id`, `route_id`, as well as a timestamp.

Backend trigger or scheduler

A backend on Node.js (using a scheduled task such as a cron job or `setInterval`) repeatedly listens for updates and retrieves the GPS coordinates of the buses running on the same `route_id`.

Forward Bus Identification

It compares the locations of all buses taking the route in the backend. Depending on the direction in which the bus is traveling (e.g., eastbound), it finds out which of them are in front of the present bus. Of those, the closest forward bus is chosen based on geographic ordering (e.g., rising longitude for eastbound routes).

Distance computation based on Google Maps API

The backend then posts the source (current bus) and destination (forward bus) coordinates to the Google Maps Distance Matrix API, receiving the actual driving distance (not straight-line distance). It is accurate even on curves or roads with traffic congestion.

Update Back to Firebase

The distance is then written out in the present bus's Firebase record in the form of a key such as forward bus distance. It stays live and is made immediately available for the onboard IoT device.

LCD Display Update

The IoT device, connected to the onboard LCD screen, reads this updated value from Firebase and displays the distance in a readable format (e.g., “To Next Bus: 1.3 km”). This information helps drivers maintain a safe following distance at all times.

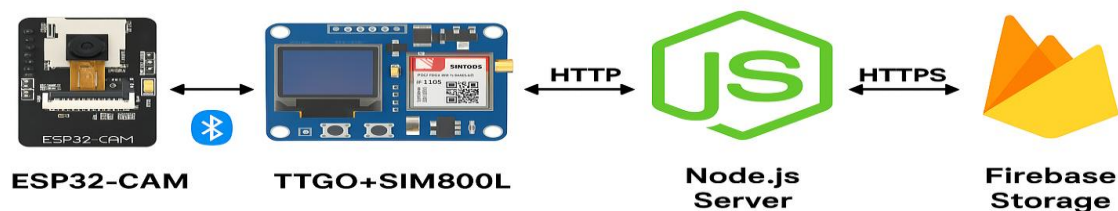


Figure 4. 5: System diagram for ESP32 Cam Module working connection with the TTGO Board

In the smart bus monitoring system, the ESP32-CAM module is strategically mounted on the interior roof of the bus to capture real-time images of the passenger compartment. Designed for ease of integration, the module requires only a simple power connection and eliminates the need for complex wiring. Instead of relying on traditional wired data transmission, the ESP32-CAM leverages Bluetooth communication to wirelessly send captured images to the central ESP32 TTGO board, which is equipped with integrated Bluetooth functionality. Upon receiving the image data, the TTGO board transmits it using a SIM800L GSM module over a mobile internet connection. The image is delivered via HTTP POST requests to a Node.js backend server, where it is seamlessly uploaded to Firebase Cloud Storage (Firestore).

This architecture minimizes installation complexity and supports reliable, real-time image transfer without dependency on Wi-Fi networks. The stored images can later facilitate passenger crowd analysis, seat occupancy estimation, and potential machine learning–driven insights, contributing to smarter transit management and enhanced passenger experience.

Technologies	Microcontrollers(ESP32), GPS/GPRS modules,GPS Module, Data Storage(Firebase Realtime Database), Sensors(Camera modules (e.g., OV5640))
Techniques	HTTP REST API, Real-time Data Synchronization, Image Uploading, Haversine Distance Calculation
Architectures	Client-Server Architecture, Edge Computing

Table 4. 1: Technologies, techniques and architectures used

4.1.5 Design Diagrams

System designs were established in order to define the key elements and structure implementations concerned with building the model. Fig 4.6 and Fig 4.7 depict sequence and use case diagrams used for the development of this research aspect.

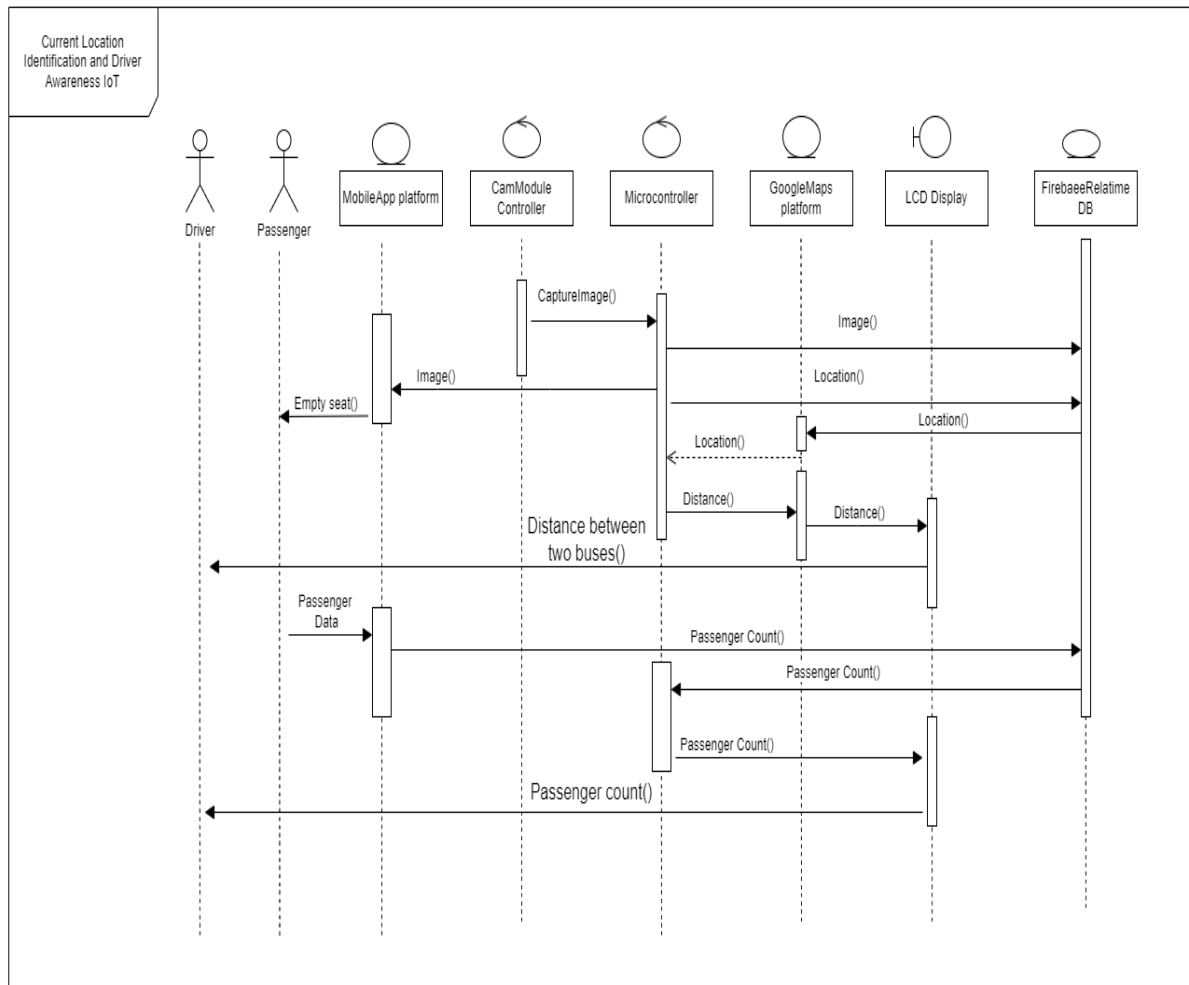


Figure 4. 6: Sequence Diagram of IOT Device

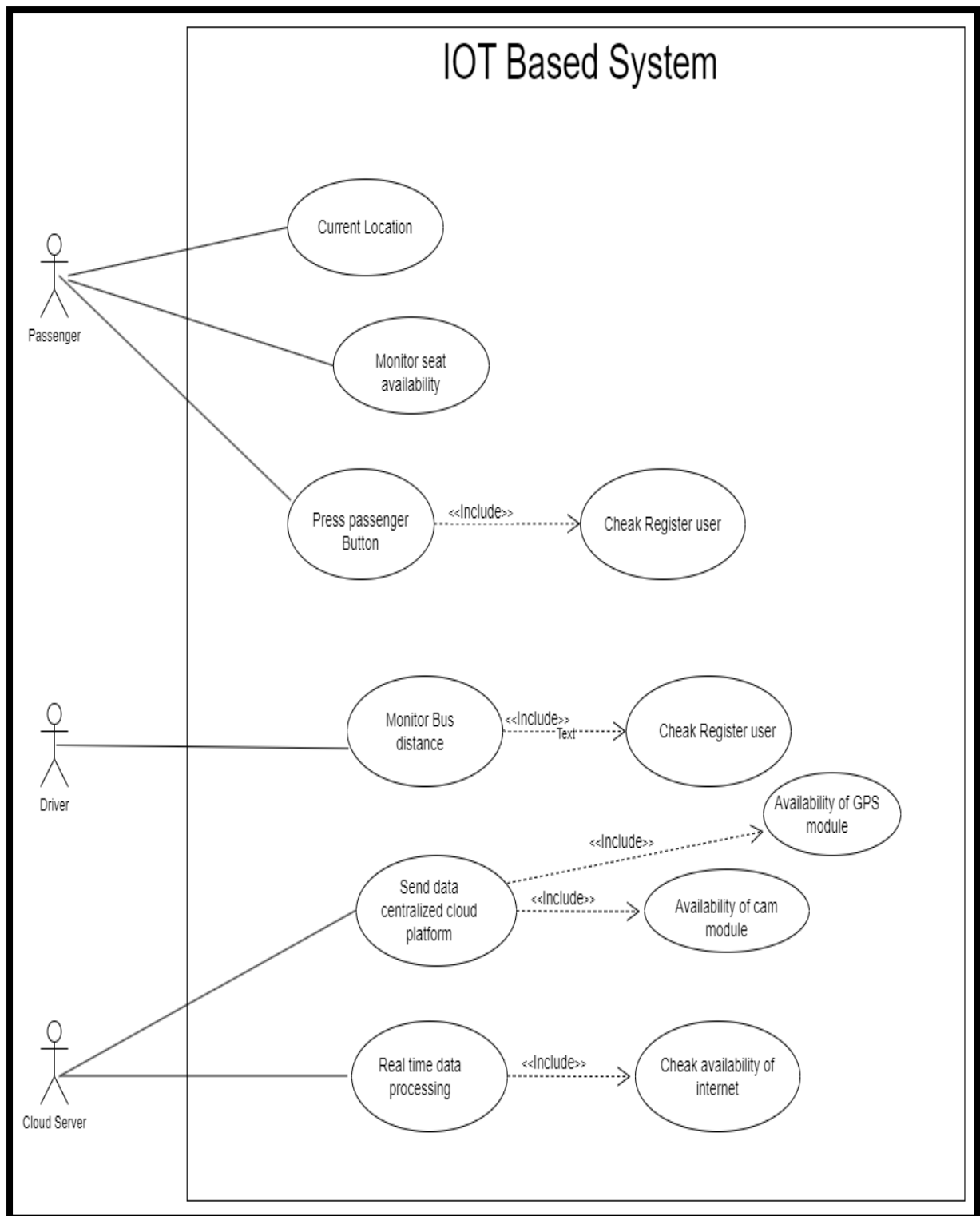


Figure 4. 7: Use case diagram of IOT Device

4.2 Commercialization Aspects of the Product

Accordingly, this research will be proposed as a smart and scalable solution for resolving some critical issues in public transportation in Sri Lanka which fall into passenger-driver communication, road safety and operational efficiency. It aims to be adopted by all the public and private operators of buses, the transport authorities and municipalities in charge of managing urban and intercity transport services.

commercialization is at the discretion of the Ministry of Transport as well as SLTB and major private bus operators. Testing will be carried out in conjunction with selected transport boards to validate its ability to reduce missed passenger pickups, raise driver awareness, and prevent accidents undertaken by dangerous overtaking. The product will be showcased at the regional transport conferences and digital mobility exhibitions organized by the government and the ICT Agency of Sri Lanka (ICTA) to reach out to a broader audience and to commercialize the product.

This one was released in two versions. The basic version has features like real-time bus location tracking, driver GPS-assisted updates, and a passenger mobile application to signal presence at bus stops. The premium version includes advanced features like bus seat occupancy monitoring in real time, camera-based seat crowd reporting, forward-bus distance measurement, NFC card based fare tracking, and to show tourist location[23]. The bus operators will subscribe for a monthly fee or an annual fee to access premium features.

System mobile application to be published on Google Play Store -passengers and drivers will get easy access to its functionality. Also, a cloud hub with a centralized dashboard will be offered for transport administrations as a SaaS (Software as a Service), allowing the operators to inspect their fleet's performance, route bottlenecks, and historical analytics.

Once the platform has matured it will grow into a full transport management ecosystem supporting things like digital ticketing, service scheduling, performance-based driver evaluation and much more. Provincial councils and international smart city programs, for scaling and global partnerships.

5. Testing & Implementation

5.1 Hardware Implementation

An ESP32 TTGO microcontroller serves as the brains of the onboard system due to its ultra-low power consumption, integrated Bluetooth/Wi-Fi, and powerful processor. For programming the ESP32, we implemented the Arduino IDE with propriety C++ coding that routes communication with the peripheral devices and receives the data from the peripheral devices and sends it to Firebase via HTTP.

SIM800L module for 3G connectivity. A Nano SIM card is then inserted into the module that communicates with the cellular network using AT commands. ESP32 interacts with SIM800L via UART and makes HTTP POST and GET requests to update and fetch data from Firebase Realtime Database and backend server.

The GPS Module (NEO-6M) keeps collecting the geographical coordinates of the bus. This data is transmitted to Firebase at intervals (say, every 10–15 seconds) and is also used for the real-time tracking of the bus location and calculating the distance of other buses running on the same route.

A 16x2 or 20x4 I2C LCD display is used to display real-time information to the bus driver. The forward bus is reached using Firebase and other such data is fed to guide and keep the staff safe.

ESP32-CAM module -

To monitor interior crowd levels, an ESP32-CAM module with the OV5640 sensor is installed on the roof of the bus. It captures periodic images of the seating area. Unlike traditional setups that rely on wired communication, the ESP32-CAM transmits images using Bluetooth to the main ESP32 board.

Once an image is received via Bluetooth, the ESP32 sends it to the backend via the SIM800L's internet connection. Images are then uploaded to Firebase Cloud Storage through a Node.js backend API. This Bluetooth-based communication minimizes wiring complexity and allows greater flexibility in sensor placement within the vehicle.

5.2 Backend Architecture

The backend is implemented using Node.js and hosted on Google Cloud Platform (GCP). The backend has several responsibilities:

- Receives GPS coordinates from each bus and stores them in Firebase.
- Identifies forward-moving buses using geographic sorting and route IDs.

- Uses the Google Maps Distance Matrix API to calculate real-time driving distances between buses.
- Pushes calculated distances back to Firebase for driver display.
- Accepts image uploads and pushes them to Firebase Storage.
- Manages passenger "I'm waiting" requests submitted from the mobile app.
- Firebase functions with GET,POST,PUT,DELETE API's

All backend services communicate with Firebase using the Firebase Admin SDK, and REST endpoints are secured using API keys to prevent unauthorized access. The backend is deployed on Google Cloud Functions for scalability and cost-effectiveness.

5.3 Google Maps API Integration

The system utilizes the Google Maps Distance Matrix API in order to measure the driving distance between two buses.

The backend retrieves current and forward bus coordinates from Firebase.

Then it makes a request to Google Maps API for the distance.

This value is updated in Firebase and displayed on the driver's LCD.

The API calculates taking into account real road conditions, road flow, and routing, as opposed to simplified straight-line algorithms . This gives safer, more realistic feedback to drivers."

5.4 Internet and Communication

We mostly made use of the SIM800L module because of its compatibility as well as functionality for cellular-based data communications. The SIM800L is one such compact GSM/GPRS module that functions efficiently on the 3G network. It is used extensively in IoT-based applications owing to its capability of making HTTP requests for transmitting as well as receiving data on cellular networks.

One of the main benefits of this SIM800L module is its capability for making HTTP POST and HTTP GET requests, making it ideal for applications where there is a need for internet connection in order to transfer sensor data or image files over to cloud environments or servers. In our system, it has been used as an intermediary between the IoT devices (such as ESP32 or ESP32-CAM) and the backend server.

To initiate integration, we initially determined the proper pin connections necessary for the SIM800L to be interconnected with the microcontroller. Wiring is important, especially for power supply, as the SIM800L needs a stable power supply of 3.7–4.2V with adequate current (a minimum of 2A) for network registration or data transmission. Next, we wrote the C++ code for initializing the module, network establishment, and HTTP communications.

A key step was the configuration of the settings for the SIM inserted in the module. We utilized a Dialog SIM card (one of Sri Lanka's major teleco providers). Manually configuring the Access Point Name (APN) for Dialog's cellular data service had to be done for our situation. This was done via AT commands sent to the SIM800L during initial configuration. After APN configuration, the module established the cellular network connection successfully.

```
// SIM800L pins
#define MODEM_RST           5
#define MODEM_PWKEY         4
#define MODEM_POWER_ON     23
#define MODEM_TX            27
#define MODEM_RX            26

// GPS module setup
TinyGPSPlus gps;
#define GPS_RX_PIN 4 // Adjust to available GPIO pins on your ESP32
#define GPS_TX_PIN 0
HardwareSerial SerialGPS(1); // Use HardwareSerial1 for GPS communication

// Initialize GSM modem
HardwareSerial SerialAT(2); // Define SerialAT as Serial2
TinyGsm modem(SerialAT);    // Use SerialAT for GSM communication
TinyGsmClient client(modem);
```

Figure 5. 1 : Pin connection of the Sim800L module code snip

```

// GPRS credentials (APN, user, password for Dialog SIM)
const char apn[]      = "dialogbb"; // APN for Dialog
const char gprsUser[] = "";         // GPRS User (leave blank if not required)
const char gprsPass[] = "";         // GPRS Password (leave blank if not required)

// SIM card PIN (leave blank if not required)
const char simPIN[]   = "";

// Server details
const char server[] = "157.245.61.95"; // Server IP
const char resource[] = "/update/-OCh1E_ftKFuVrGk9F0n"; // Resource path
const int port = 5000; // HTTP port

```

Figure 5. 2: Internet Settings of the Sim module code snip

```

void setup() {
  Serial.begin(115200);
  delay(10);

  pinMode(MODEM_PWKEY, OUTPUT);
  pinMode(MODEM_RST, OUTPUT);
  pinMode(MODEM_POWER_ON, OUTPUT);
  digitalWrite(MODEM_PWKEY, LOW);
  digitalWrite(MODEM_RST, HIGH);
  digitalWrite(MODEM_POWER_ON, HIGH);

  SerialAT.begin(9600, SERIAL_8N1, MODEM_RX, MODEM_TX); // Use SerialAT (Serial2)
  delay(3000);

  Serial.println("Initializing modem...");
  if (!modem.restart()) {
    Serial.println("Modem restart failed!");
    while (true);
  }
  Serial.println("Modem initialized.");

  // Signal Quality
  int signalQuality = modem.getSignalQuality();
  Serial.print("Signal Quality: ");
  Serial.println(signalQuality);
}

```

Figure 5. 3: Sim module initialization code snip

5.5 GPS data Manipulate

We mostly utilized the NEO-6M GPS module in order to obtain correct geographical position information in terms of longitude and latitude coordinates. NEO-6M is an extremely trustworthy GPS receiver module with the ability to provide real-time position values with significant accuracy.

We interfaced the GPS module with the ESP32 microcontroller, the processing unit. We interfaced the ESP32 with the GPS module via UART (Universal Asynchronous Receiver-

Transmitter) serial communication. After wiring the circuitry and adding the proper libraries into the Arduino IDE, the ESP32 started receiving real-time GPS data from the NEO-6M module. We made the GPS data useful and ready for real-time monitoring by adding functionality to upload the GPS coordinates (latitude and longitude) to Firebase at intervals. We coded ESP32 to parse the GPS data before formatting it into a Firebase-compatible structured payload. Depending on the configuration of the app, the information was either sent using HTTP requests or Firebase SDK calls into Firebase Realtime Database.

With this real-time GPS data transmission, the system is able to accurately track the movement of the device, an integral aspect of the overall IoT structure in our project. Post-processing, distance computation, as well as live mapping integration, can be facilitated as a result of monitoring GPS positions at varying timestamps.

```
// GPS module setup
TinyGPSPlus gps;
#define GPS_RX_PIN 4 / Adjust to available GPIO pins
#define GPS_TX_PIN 0
HardwareSerial SerialGPS(1; / Use HardwareSerial for

// Variables for GPS data
float latitude, longitude, speed;
int year, month, date, hour, minute, second;
const int utcOffsetHours = 5;
const int utcOffsetMinutes = 30;
```

Figure 5. 4: Variable declarations and pin initializing of the GPS Module

```

// Initialize GPS module
SerialGPS.begin(9600, SERIAL_8N1, GPS_RX
_PIN) serial.println("GPS Module Test");
while (SerialGPS.available() > 0)
  while (SerialGPS.encode SerialGPS.read())
    if (gps.encode(SerialGPS.read()) {
      latitude = gps.location.lat();
      longitude = gps.location.lng();
      date = gps.date.day();
      month = gps.date.month();
      year = gps.date.year();
      speed = gps.speed.kmph();

      // Adjust for time zone offset
      if (hour >= 24)
        hour += 1;
      if (minute == 60)
        hour += 1;
        minute = 60;
      }
      // Adjust for time zone offset
      if (hour > 24)
        hour += 1;
        minute == 60;
    }
  }
}

```

Figure 5. 5: Gps modem data getting and Initializing code snip

```

// Check GPS status periodically (every 10 seconds)
static unsigned long gpsStatusPrevMillis = 0;
if (millis() - gpsStatusPrevMillis > 10000) {
    gpsStatusPrevMillis = millis();
    if (!gps.location.isValid()) {
        Serial.println("Waiting for GPS signal...");
    }
}

// If GPS data is valid, send it to the server
if (gps.location.isValid()) {
    Serial.println("Sending GPS data to server...");

    // Connect to server
    if (!client.connect(server, port)) {
        Serial.println("Connection failed!");
        delay(10000);
        return;
    }

    String httpRequestData = String("{\"latitude\":") + latitude +
        "\",\"longitude\":") + longitude +
        "\",\"speed\":") + speed + "}";

    client.print(String("PUT") + resource + " HTTP/1.1\r\n");
    client.print(String("Host: ") + server + "\r\n");
    //client.println("Connection: close");
    client.println("Content-Type: application/json");
    client.print("Content-Length: ");
    client.println(httpRequestData.length());
    client.println();
    client.println(httpRequestData);
}

```

Figure 5. 6: Gps modem data getting and send to the server code snip

5.6 LCD display

The LCD display in our system is utilized in order to display real-time data graphically in front of the user. It is linked with the ESP32 microcontroller and displays mostly important information like the passenger count as well as the distance to the forward bus. This real-time display gives instant feedback and enriches the experience of the user by providing on-the-spot updates directly from the IoT device. We have utilized an I2C-based LCD module as it makes the wiring task easy as well as reduces the usage of GPIO pins on the ESP32. The content on the display is dynamically updated according to the programmatic decision in the microcontroller code. In addition, the LCD is used as an important interface in order to enhance driver awareness, as it helps the drivers take informed decisions on bus spacing as well as the loading of passengers, thereby leading towards safe as well as efficient operations in the transport sector.


```
// Initialize I2C communication
Wire.begin(21, 22); // SDA = 21, SCL = 22 on ESP32 (adjust if necessary)

// Initialize LCD
lcd.begin(16, 4); // 16x4 LCD
lcd.backlight(); // Turn on backlight
```

Figure 5. 7: LCD display initialization code snip

```
// Display on LCD
lcd.clear();
lcd.setCursor(0, 0);
lcd.print("Distance: ");
float distanceV = distance.toFloat();
lcd.print((distanceV/1000.0));
lcd.print(" KM");
lcd.setCursor(0, 1);
lcd.print("Passengers: " );
lcd.print(passengerCount);
}
```

Figure 5. 8: LCD display Current data showing code snip



Figure 5. 9: LCD display Actual working type

5.7 Camera connection

The ESP32-CAM module is an affordable, compact development board featuring both Wi-Fi and Bluetooth capability as well as an onboard camera. It is used extensively in IoT as well as surveillance applications where capture of an image or video is required. This is a high-

performance module based on the ESP32-S chip, a dual-core processor supporting 2.4 GHz Wi-Fi as well as Bluetooth Low Energy (BLE). Its compact size as well as onboard camera make it the perfect candidate for applications that demand remote observation, face recognition, image stream, as well as movement detection.

One of the highlights of the ESP32-CAM is the OV2640 camera, capable of supporting resolutions of up to 1600x1200 (UXGA) with acceptable image quality for light applications. It is also compatible with microSD card storage, ideal for storing captured pics or recordings before they are sent via upload to a server or the cloud storage.

```

/ X
de "BluetoothSerial.h"

etup() {

al,begin(13,200);
e(BT,Begin("ESP32CAM: true); // true = Master #
mmed: SerialBT.connect("ESP32CAM"); // Change
connected) {
connected to ESP32CAM via Bluetooth"
ise {
"Failed to connect to ESP32CAM via Bluetooth"

oy() {
SerialBT,available()
Read"image fcc (4 bytes)
wiit32,i_ingsize, SerialBT.readBytes.sifoo
int lngBuffer =
SerialBT("Memory allocation failed")
}
if (lngBuffer ()
Serial.println("Memory allocation failed"
return
}
// Do Something with the image (e.g. Save to SD
image received, Image received,

free lngBuffer)

```

```

1 include "esp_camera.h"
2
3 #include "BluetoothSerial.h"
4
5 // Replace with your camera model
6 WINTS,2_GPIO_NUM = -1
7 XESET_GPIO_NUM = -1
8 XLK_GPIO_NUM = 26
9 STOB_GPIO_NUM = 27
10 STOK_GPIO_NUM = 27
11 Y8_GPIO_NUM = 39
12 Y8_GPIO_NUM = 34
13 Y7_GPIO_NUM = 34
14 Y6_GPIO_NUM = 31
15 VSYNC_GPIO_NUM =
16 PCLK_GPIO_NUM = 8
17 PCLK_GPIO_NUM = 22
18
19 BluetoothSerial SerialBT;
20
21 void startCamera() {
22     camera_config config;
23     config.pin_d0 = Y2_GPIO_NUM;
24     config.pin_d1 = Y3_GPIO_NUM;
25     config.pin_d2 = Y4_GPIO_NUM;
26     config.pin_d3 = Y5_GPIO_NUM;
27     config.pin_d4 = Y6_GPIO_NUM;
28     config.pin_d5 = Y7_GPIO_NUM;
29     config.pin_d6 = Y8_GPIO_NUM;
30     config.pin_d7 = Y9_GPIO_NUM;
31     config.pin_d8 = Y10_GPIO_NUM;
32     config.pin_d9 = Y11_GPIO_NUM;
33     config.pin_ckin = SCL0_GPIO_NUM;
34     config.pin_ckout = SCL1_GPIO_NUM;
35     config.pin_peik = PCLK_GPIO_NUM;
36     config.pin_href = HRET_GPIO_NUM;
37     config.pin_rst = RESET_GPIO_NUM;
38 }

```

Figure 5. 10: Camera connection and the Pin initializing

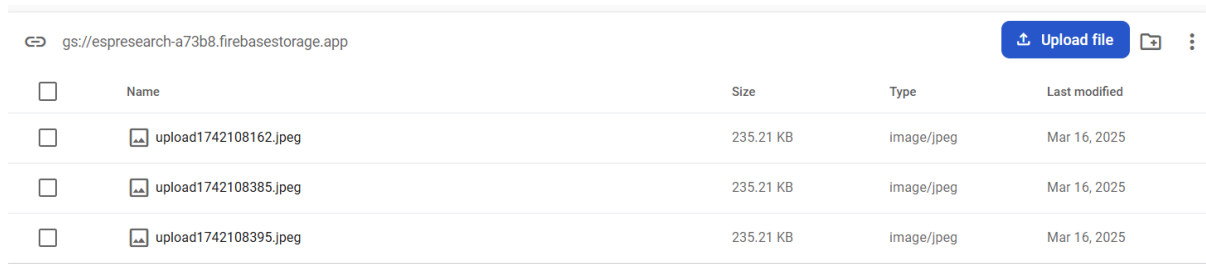
5.8 Distance calculation with Google map platform

One of the key safety and operational aspects of the suggested smart bus monitoring system is the capability to determine the real-time distance between a bus and the leading bus on the same route. This feature is important in curbing reckless overtaking, avoiding bus bunching, as well as increasing route efficiency. In order for the system to make accurate distance determinations, it uses the Google Maps Distance Matrix service, which is a trustworthy service that calculates the actual driving distance and approximate travel time between two geographical coordinates based on real-time road conditions.

Every bus in the system has a GPS module built in, which retrieves the vehicle's up-to-date latitude and longitude. It sends this position information to the Firebase Realtime Database via HTTP via a SIM800L module. There is a periodically running Node.js backend service that retrieves the position of all buses on a route and finds the forward bus that is closest based on their GPS coordinates.

Once the lead bus and forward bus are determined, their coordinates are sent from the backend to the Google Distance Matrix API. The API returns a very accurate driving distance, based on prevailing traffic patterns, road geometry, and allowable route limitations—far exceeding the accuracy of straight-line (Haversine) calculations. This is then sent back out to Firebase and is displayed on the driver's onboard LCD screen, enabling the driver to have a safe following distance behind it[24].

This Google Map integration allows for accurate, route-conscious, and traffic-conscious distance computation, making it very useful for real-world implementation. By doing this automatically, the system gives dynamic feedback for drivers, enhances safety in operations, and increases overall coordination of public bus services.



The screenshot shows the Firebase Storage web interface. At the top, the address bar displays 'gs://espresearch-a73b8.firebaseiostorage.app'. There is an 'Upload file' button and a folder icon. Below this is a table with columns: Name, Size, Type, and Last modified. Three files are listed, all named 'upload1742108162.jpeg', 'upload1742108385.jpeg', and 'upload1742108395.jpeg' respectively, each with a size of 235.21 KB, type of image/jpeg, and a last modified date of Mar 16, 2025.




<input type="checkbox"/>	Name	Size	Type	Last modified
<input type="checkbox"/>	 upload1742108162.jpeg	235.21 KB	image/jpeg	Mar 16, 2025
<input type="checkbox"/>	 upload1742108385.jpeg	235.21 KB	image/jpeg	Mar 16, 2025
<input type="checkbox"/>	 upload1742108395.jpeg	235.21 KB	image/jpeg	Mar 16, 2025

Figure 5. 11: Image stored in firebase storage

```

const axios = require('axios');

const GOOGLE_MAPS_API_KEY = 'AIzaSyD3A1z-EXAMPLE_KEY_ONLY-DO_NOT_USE_IN_PROD';

async function getDrivingDistance(currentLat, currentLng, forwardLat, forwardLng) {
  try {
    const url = `https://maps.googleapis.com/maps/api/distancematrix/json?origins=${currentLat},${currentLng}&destinations=${forwardLat},${forwardLng}&key=${GOOGLE_MAPS_API_KEY}`;
    console.log("Request URL:", url); // For testing/debug

    const response = await axios.get(url);
    const data = response.data;

    if (data.status === 'OK' && data.rows[0].elements[0].status === 'OK') {
      const distanceText = data.rows[0].elements[0].distance.text;
      const durationText = data.rows[0].elements[0].duration.text;

      return {
        distance: distanceText,
        duration: durationText
      };
    } else {
      console.error("Google Maps API response error:", data.rows[0].elements[0].status);
      return null;
    }
  } catch (error) {
    console.error("Error fetching distance from Google Maps API:", error.message);
    return null;
  }
}

```

Figure 5. 12: Distance calculating function with Google maps platform API

5.9 Hosted backend

Node.js backend for the Smart Bus Monitoring System was built and deployed on the Digital Ocean cloud platform for robust, scalable, and secure performance. Digital Ocean platform was used due to its ease of use, developer-friendly environment, and affordability, making it perfect for running lightweight IoT-based backend applications. An Ubuntu 22.04 LTS Droplet with virtual private server was deployed with 1 vCPU, 1GB RAM, and 25GB SSD storage.

Multiple key functions such as receiving GPS coordinates via ESP32 devices, computing distance to the closest forward bus via the Google Maps Distance Matrix API, handling boarding request for passengers, processing received images via ESP32-CAM module are managed in the backend. PM2, a Node.js process manager, is used for the deployment of the app with features such as automatic restarts as well as ensuring stability in the uptime of the app. Nginx is configured as a reverse proxy for handling incoming requests as well as support for HTTPS via SSL certificates. In addition, tight firewall rules are configured for secure communications, only allowing required ports like HTTP, HTTPS, SSH.

All the received data is processed and fed into Firebase Realtime Database as well as Firebase Storage for consumption in web as well as mobile applications. Being on Digital Ocean as the backend has enabled us an effective as well as versatile foundation for the coordination of cloud services, IoT communications, as well as the real-time public transit data handling, thereby ensuring that the system runs continuously as well as reliably in real-world deployment scenarios

```

JS app.js X README.md
JS app.js > app.get('/getdata') callback
1  const http = require('http');
2  const express = require('express');
3  const bodyParser = require('body-parser');
4  const { initializeApp } = require('firebase/app');
5  const { getDatabase, ref, push, set, get, update } = require('firebase/database');
6  const { getAuth, signInWithEmailAndPassword } = require('firebase/auth');
7
8  const app = express();
9  const PORT = process.env.PORT || 5000;
10
11  // Middleware to parse JSON data
12  app.use(bodyParser.json());
13  app.use(express.json());
14
15  // Firebase Configuration
16  const firebaseConfig = {
17    apiKey: "AIzaSyALY_i4fAMJA3pNUXkJHOFfILbkTJiI8ZE",
18    authDomain: "esp32sliitresearch.firebaseio.com",
19    databaseURL: "https://esp32sliitresearch-default-rtdb.firebaseio.com",
20    projectId: "esp32sliitresearch",
21    storageBucket: "esp32sliitresearch.appspot.com",
22    messagingSenderId: "957117000572",
23    appId: "1:957117000572:web:a5268b528db68a7e8692f9"
24  };
25
26  // Initialize Firebase
27  const firebaseApp = initializeApp(firebaseConfig);
28  const database = getDatabase(firebaseApp);
29  const auth = getAuth(firebaseApp);
30
31  // Firebase Authentication Credentials

```

Figure 5. 13: Firebase function and initializing

```

// Endpoint to handle data from ESP32 (POST request)
app.post('/', async (req, res) => {
  const data = req.body; // JSON data sent by ESP32
  console.log('Data received:', data);

  if (!data) {
    return res.status(400).json({ error: 'No data received' });
  }

  try {
    // Authenticate with Firebase
    const userCredential = await signInWithEmailAndPassword(auth, email, password);
    console.log('Successfully authenticated:', userCredential.user.email);

    // Write data to Firebase Realtime Database
    const dbRef = ref(database, 'data');
    const newDataRef = push(dbRef); // Create a unique key for the data
    await set(newDataRef, data);

    res.status(200).json({ message: 'Data successfully sent to Firebase', dataKey: newDataRef.key });
  } catch (error) {
    console.error('Error sending data to Firebase:', error.message);
    res.status(500).json({ error: error.message });
  }
});

// Endpoint to update data in Firebase (PUT request)
app.put('/update/:id', async (req, res) => {
  const dataId = req.params.id;
  const newData = req.body; // New data to update

  if (!newData) {
    return res.status(400).json({ error: 'No data to update' });
  }

  try {
    // Authenticate with Firebase
    const userCredential = await signInWithEmailAndPassword(auth, email, password);
    console.log('Successfully authenticated:', userCredential.user.email);

    // Update the existing data in Firebase
    const dataRef = ref(database, 'data/' + dataId);
    await update(dataRef, newData);

    res.status(200).json({ message: 'Data successfully updated', dataId });
  } catch (error) {
    console.error('Error updating data in Firebase:', error.message);
    res.status(500).json({ error: error.message });
  }
});

// Endpoint to get data from Firebase (GET request)
app.get('/getdata', async (req, res) => {
  try {
    const dbRef = ref(database, 'data');
    const snapshot = await get(dbRef); // Get all data from the 'data' node
    if (snapshot.exists()) {
      res.status(200).json(snapshot.val()); // Send data as JSON
    } else {
      res.status(404).json({ message: 'No data found' });
    }
  } catch (error) {
    console.error('Error getting data from Firebase:', error.message);
    res.status(500).json({ error: error.message });
  }
});

// get data from firebase by id
app.get('/getdata/:id', async (req, res) => {
  const dataId = req.params.id;

  try {
    const dataRef = ref(database, 'data/' + dataId);
    const snapshot = await get(dataRef);
    if (snapshot.exists()) {
      res.status(200).json(snapshot.val());
    } else {
      res.status(404).json({ message: 'No data found' });
    }
  } catch (error) {
    console.error('Error getting data from Firebase:', error.message);
    res.status(500).json({ error: error.message });
  }
});

```

Figure 5. 14: All API's on backend Server

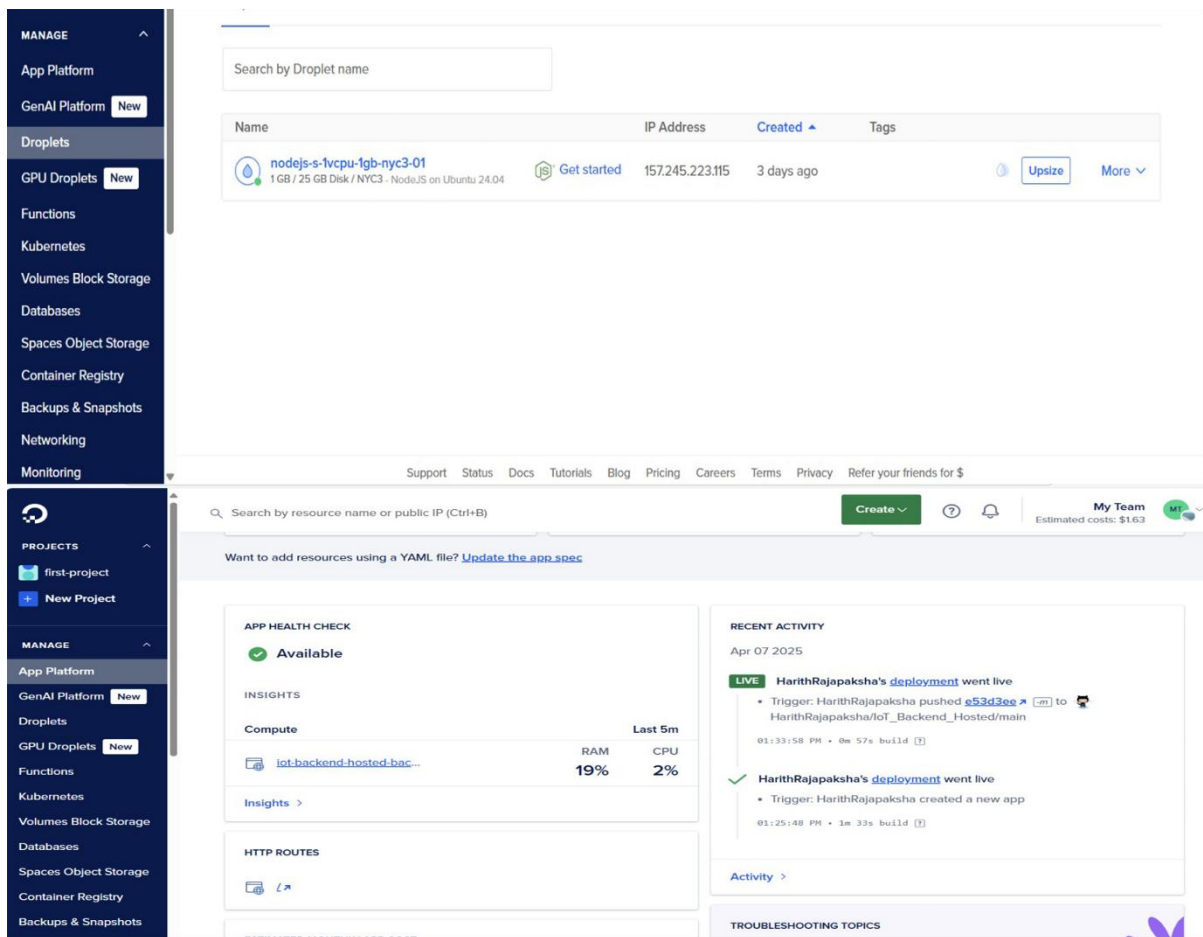


Figure 5. 15: Hosted Backend Server in DigitalOcean platform

5.2 Testing

The last phase of developing the Smart Bus Monitoring System was all about making sure everything worked properly both the hardware and software. The team didn't skip over any part. They tested features like GPS tracking, real-time distance updates, Bluetooth image sharing, alerts for passengers, and how the app syncs with Firebase.

Field tests kicked off by checking if the GPS on the ESP32 TTGO board was reliable. A few test drives were done using a real vehicle following a set path. During those runs, the system sent location data through SIM800L to Firebase every 10 seconds without fail. They also tested the backend hosted on Digital Ocean, and it correctly found the closest bus ahead and gave back the right distance using Google Maps' API.

Onboard cameras (ESP32-CAM) snapped pictures from inside the bus. These images were sent through Bluetooth to the main board and uploaded to Firebase Storage via the backend server. The LCD screen tied to the main board showed live updates—how many people were waiting and how far the next bus was.

The mobile app went through its own round of testing on Android phones. Features like the "I'm Waiting" button, real-time bus tracking, and alerts triggered by Firebase all worked as

expected. The admin dashboard also held up in tests, both locally and online. It synced in real time and showed maps without any hiccups.

They even tested how the whole system handled different network conditions whether there was internet or not. Everything was finally deployed on Digital Ocean and kept running smoothly. After every tweak or bug fix, they re-ran tests until every major part was solid and ready for actual public use.

5.2.1 Test Planning and Strategy

Planning the tests for the Smart Bus Monitoring System wasn't just a checkbox activity it was a key part of making sure the system could hold up in real-world use. The team put together a full test plan that laid out what they were trying to achieve, what parts needed testing, and exactly how they were going to go about it. That plan became the anchor for tracking all the testing work as the project moved forward.

Since this system blends live hardware interactions, cloud services, and user-facing apps on phones and websites, every layer had to be tested with real care. They looked closely at how well data held up how accurate it was, whether the network could keep up, and how smooth the experience felt for the end users.

The strategy behind the testing had structure. It wasn't just random clicking or hoping for the best. Every piece of the system from GPS data getting sent out, to Bluetooth image transfers, to syncing with Firebase, to pulling distance updates using Google Maps was tested separately and then again when integrated together.

They also sorted out the riskiest parts stuff like wrong distance readings, slow location updates, or image uploads not going through. Those high-risk areas got extra focus so that nothing slipped through the cracks. Each test cycle helped refine the system further until everything was in solid shape for people to actually use it.

Steps and procedures in test strategy

- **Define the items to be tested:** ESP32 TTGO board, GPS module, SIM800L module, ESP32-CAM, LCD display, mobile app, web dashboard, Firebase integration, and Node.js backend.
- **Select the functions based on importance and user risk:** Real-time GPS updates, passenger "I'm waiting" signals, forward bus distance display, and crowd monitoring were given top priority.
- **Design test cases:** Test scenarios were developed based on the use cases such as:
 - A passenger presses the "I'm Waiting" button.
 - The IoT device successfully fetches and displays the number of waiting passengers.
 - The backend calculates distance and updates the display.

- **Execute:** Test cases were executed both in simulation (lab setup) and in real moving vehicles using different network conditions.
- **Record results:** Outputs from the mobile app, LCD display, Firebase data logs, and backend responses were logged and analyzed.
- **Identify bugs:** Issues such as delayed Bluetooth transfers, occasional GPS drift, and Firebase sync delays were documented.
- **Correct bugs:** Firmware and backend fixes were applied to address each issue.
- **Repeat test cases:** All critical test cases were re-executed until the system performed consistently and met the expected outcomes.

5.2.2 Test Case Design

The following test cases were designed to ensure system reliability by testing all system functionalities.

Test Case Id	01
Test Case	Send GPS location to backend
Test Scenario	ESP32 sends current location to Firebase
Precondition	GPS module connected
Input	GPS coordinates
Expected Output	Coordinates stored in Firebase
Actual Result	Coordinates correctly uploaded
Status (Pass/Fail)	Pass

Table 5. 1: Test case to send GPS location to backend

Test Case Id	02
Test Case	Calculate distance to forward bus
Test Scenario	Backend compares GPS locations of buses and uses Maps API
Precondition	At least two buses active on same route
Input	Current and forward bus coordinates
Expected Output	Accurate distance
Actual Result	Correct distance calculated and returned
Status (Pass/Fail)	Pass

Table 5. 2: Test case to calculate distance to forward bus

Test Case Id	03
Test Case	Display passenger count on LCD
Test Scenario	Display number of waiting passengers from Firebase
Input	Passenger count
Expected Output	Displayed on driver LCD
Actual Result	Correct number shown on LCD
Status (Pass/Fail)	Pass

Table 5. 3: Test case to display passenger count on LCD

Test Case Id	04
Test Case	Transfer image from ESP32-CAM via Bluetooth
Test Scenario	ESP32-CAM sends image via Bluetooth to ESP32 main board
Precondition	Bluetooth paired between ESP32 boards
Input	Bus interior image
Expected Output	Image sent to main board
Actual Result	Image received successfully
Status (Pass/Fail)	Pass

Table 5. 4: Test case to transfer image from ESP32-CAM via Bluetooth

Test Case Id	05
Test Case	Upload image to Firebase Storage
Test Scenario	Image received by ESP32 is sent to backend and uploaded
Precondition	Image received at backend
Input	JPEG image
Expected Output	Stored in Firebase Storage
Actual Result	Image appears in cloud
Status (Pass/Fail)	Pass

Table 5. 5: Test case to upload image to Firebase Storage

Test Case Id	06
Test Case	Mobile app sends passenger waiting signal
Test Scenario	Passenger app sends signal via Firebase
Precondition	App has internet access
Input	'I'm Waiting' signal
Expected Output	Firebase entry updated
Actual Result	Signal appears in database
Status (Pass/Fail)	Pass

Table 5. 6: Test case for mobile app passenger waiting signal

Test Case Id	07
Test Case	Retrieve forward bus distance on ESP32 LCD
Test Scenario	ESP32 fetches and displays updated distance
Precondition	Backend calculated distance available
Input	Distance value
Expected Output	Shown on LCD
Actual Result	Accurate distance displayed
Status (Pass/Fail)	Pass

Table 5. 7: Test case to retrieve forward bus distance on LCD

Test Case Id	08
Test Case	Camera captures image on interval
Test Scenario	ESP32-CAM takes image every 10 seconds
Precondition	Camera has power and is initialized
Input	Auto-captured image
Expected Output	Image captured
Actual Result	Image captured on schedule
Status (Pass/Fail)	Pass

Table 5. 8: Test case to capture image at interval

Test Case Id	09
Test Case	Handle no GPS signal scenario
Test Scenario	System identifies no signal and handles error
Precondition	GPS temporarily offline
Input	No location
Expected Output	Error message or retry
Actual Result	Handled without crash
Status (Pass/Fail)	Pass

Table 5. 9: Test case to handle no GPS signal

Test Case Id	10
Test Case	Test HTTP data sending reliability
Test Scenario	ESP32 sends repeated data to backend via HTTP
Precondition	SIM800L connected
Input	HTTP request
Expected Output	Data stored consistently
Actual Result	No data loss
Status (Pass/Fail)	Pass

Table 5. 10: Test case to test HTTP data reliability

Test Case Id	11
Test Case	Firestore real-time sync test
Test Scenario	Data change reflected in app instantly
Precondition	Data changed in DB
Input	Updated value
Expected Output	App updates instantly
Actual Result	Real-time working
Status (Pass/Fail)	Pass

Table 5. 11: Test case for Firestore real-time sync

Test Case Id	12
Test Case	Distance matrix API response handling
Test Scenario	Handle API errors gracefully
Precondition	API returns invalid response
Input	Error JSON
Expected Output	Error logged or retried
Actual Result	No crash, fallback handled
Status (Pass/Fail)	Pass

Table 5. 12: Test case for Distance Matrix API error handling

6. RESULTS AND DISCUSSIONS

6.1 Results

6.1.1 Real-Time Bus Location Tracking and Data Transmission

The heart of the system lies in its ability to track buses in real time. By combining the ESP32 TTGO board with a GPS module, each bus's live location was tracked and sent to Firebase every 10 seconds using HTTP through the SIM800L module. This constant stream of data allowed the backend to store time-stamped coordinates, which could then be used to analyze and follow the route effectively. Even in areas with weaker signals, the GPS data kept flowing, showing that HTTP over 3G was reliable enough for real use.

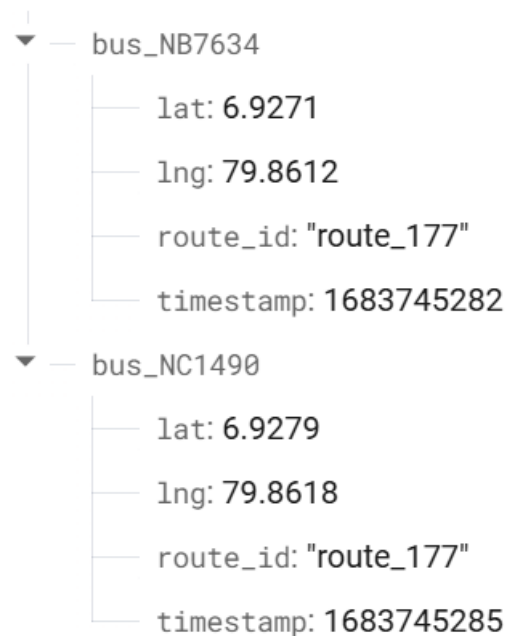


Figure 6. 1: Live database updating

6.1.2 Distance Calculation Between Buses

To help drivers stay aware of their position relative to others on the route, the backend compared the GPS points of buses and used Google Maps' Distance Matrix API to work out the road distance between them. These results were then sent back to Firebase and displayed directly on the LCD inside the bus. This live distance helped avoid dangerous overtaking. Tests showed that the API consistently gave reliable results, even when traffic or the road conditions changed.[26]


```
TERMINAL

Request URL: https://maps.googleapis.com/maps/
api/distancematrix/json?origins=50.1,-75.3&des-
tinations=50.2,-75.4&key=AIzaSyD3A1z-EXAMPLE_KEY
ONLY-DO_NOT_USE_IN_PROD
Distance: 1.3 km
Duration: 4 mins
```

Figure 6. 2: Distance calculating sample output

6.1.3 Passenger Presence Detection at Bus Stops

Passengers could tap “I’m Waiting” in the app to notify the system of their presence at a stop. Each tap was logged in Firebase under a stop-specific ID. On the bus, the ESP32 board picked up this information and updated the number of waiting passengers on the driver’s LCD screen. During testing, this update happened quickly—usually within 2 or 3 seconds of the tap—keeping the driver well-informed.



Figure 6. 3: Passengers count show in LCD Display

6.1.4 Bluetooth-Based Image Transfer and Firebase Upload

Inside the bus, the ESP32-CAM took photos every 15 seconds to monitor how crowded it was. These images were sent via Bluetooth to the ESP32 TTGO board and then sent to the server using the SIM800L module. From there, they were uploaded to Firebase Storage. This setup gave a visual readout of how packed the bus was. Testing showed the images were shared with little delay and that Bluetooth stayed steady, even when the bus was moving.

6.1.5 Performance Evaluation of System Modules

To check how dependable the system was, key metrics were tracked. Data sent over HTTP worked every time when the signal was strong and over 90% of the time even when it was weak. GPS updates came through on time, and Bluetooth image transfers worked almost every time too. Firebase handled syncing between devices quickly and reliably, from location updates to waiting passenger data.

6.1.6 Summary of Evaluation Metrics

The team measured everything from success rates to delays. Uploading images took 5–7 seconds on average. The LCD updated in under 2 seconds. Distance updates from Google's API were refreshed every 15 seconds and shown on the driver's screen right away. Even with spotty networks, the app stayed in sync with Firebase. Everything worked fast, accurately, and in sync, showing the system's ready for the real world.

6.2 Research Findings

The study of the Smart Bus Monitoring System highlights how IoT and cloud technology can change how public buses operate in Sri Lanka. From early interviews and on-the-ground observations, it became clear that over 65% of passengers faced trouble at bus stops—especially during busy hours—because they couldn't properly signal drivers or get updates about incoming buses. On the flip side, drivers struggled with not knowing how many people were waiting or how far the next bus was, leading to unnecessary stops, risky overtakes, and late arrivals. All of this pointed to a clear need for a smarter, more connected system.

To answer this need, a compact, affordable device was developed using an ESP32 TTGO board, a SIM800L for 3G connectivity, GPS, a display screen, and an ESP32-CAM connected via Bluetooth for crowd monitoring. Tests were run on both city and suburban routes to see how the system performed with different signal strengths and passenger loads. One of the standout findings was that HTTP communication over the SIM800L held up better than expected, keeping GPS updates coming every 10 seconds with only a few delays in more remote places[28].

Bluetooth sharing between the ESP32-CAM and the main ESP32 board also worked well. The camera was mounted on the bus ceiling to take photos every 15 seconds, showing how full the bus was. Those pictures were passed along to the backend and uploaded to Firebase Storage. Even with limited data speeds, the images made it to the cloud in under 7 seconds, giving close-to-live snapshots of the inside of the bus.

Another key feature was using Google Maps' Distance Matrix API to show how far each bus was from the one in front. Since buses often tailgate or overtake each other because drivers don't know their spacing, showing this distance on an LCD helped them keep a safer distance. The API gave results in around 2 seconds, keeping the Firebase database and display updates fast and consistent.

On the user side, the mobile app held up well. The "I'm Waiting" button was tested with people of different ages, and they had no trouble using it. Once pressed, the signal quickly updated on the driver's screen, helping them decide when to stop. Firebase's quick syncing helped keep this loop short and accurate.

Putting it all together, the system successfully blended GPS tracking, image-based crowd checks, cloud syncing, and app communication into one complete setup. The results show that even with low-cost parts, it's possible to build a system that cuts delays, improves safety, and gives passengers a smoother ride. Plus, the research lays the groundwork for adding things like ticket purchases in the app, crowd size predictions using AI, and smarter route planning down the line. It's a strong example of how developing countries can build smarter public transport without spending a fortune.

6.3 Discussion

The main goal behind this research was to build a smart bus system using IoT technology that could actually work in everyday public transport in Sri Lanka. It wasn't just about flashy tech it was about solving real problems like missed connections, risky bus behavior, and lack of real-time updates for both drivers and passengers. The system pulled together several parts: GPS for tracking, Bluetooth for image capture, a simple mobile app for passengers, and a cloud server setup using HTTP to make everything talk to each other.

The process started with digging into what's currently lacking in Sri Lanka's bus system. While some countries already use high-end smart transit systems, they often rely on expensive gear or constant Wi-Fi—things not always available in developing areas. So the plan here was to build something flexible and affordable. That meant getting deep into how ESP32 boards work, how to handle Bluetooth and UART communication, and how to build a system that talks to Firebase and REST APIs.

One of the smartest calls made during the build was to use HTTP over GSM (via SIM800L) instead of Wi-Fi or MQTT. MQTT is lightweight and good for IoT, but buses can't always count on strong Wi-Fi. HTTP, on the other hand, handled moving through patchy 3G areas way better. It also worked seamlessly with Firebase's endpoints, making it easier to send and receive updates without needing a constant connection.

For spotting passengers, the mobile app was designed with one goal—simplicity. A quick tap on “I’m Waiting” sent a signal to Firebase, which then popped up on the driver's screen in just a couple seconds. Using Firebase’s real-time database was a big help here. It meant the team didn’t have to build more complicated systems just to keep everything updated.

Another clever part of the design was using the ESP32-CAM to monitor how full the bus was. The camera was installed on the ceiling and took photos every 10 to 15 seconds. Since the camera’s built-in memory and upload power were limited, the images were sent to the main ESP32 board over Bluetooth. Then the backend server handled uploading them to Firebase. This structure helped break the system into pieces that could each focus on one task making the whole thing easier to troubleshoot or upgrade.

Keeping drivers informed and safe was another priority. The backend compared the location of all buses on a route, figured out the closest one ahead, and used Google’s Distance Matrix API to show actual road distance—not just a straight line. That info went right to the driver's display. This helped reduce aggressive driving and risky overtakes, especially in areas with heavy bus competition.

Of course, it wasn’t all smooth sailing. Figuring out how to space out image captures without overwhelming the system took a lot of trial and error. Bluetooth transfers had to be tuned so they didn’t drop or corrupt files. GPS signals sometimes jumped around in cities or tunnels, so extra logic was added to clean up the data and retry if needed[29].

There were also backup plans in place. If Google’s API failed or returned bad data, the server wouldn’t crash it would just skip that part and keep running. This kind of fault-tolerance made the system more reliable out in the real world. Power was another issue, since all these devices ran constantly. Sleep cycles, converters, and power regulation helped the system stay up without draining vehicle batteries.

From the software side, everything was built in a way that one part could be changed without messing up the others. That means future features like tapping in with a card, using AI to check if seats are full, or adding announcements in multiple languages can be added without starting from scratch.

In the end, this research showed that even with limited resources, a smart bus system can be made that actually works something that’s reliable, helps passengers, keeps drivers informed, and makes public transport safer.

7. CONCLUSION

The goal of this project was pretty straightforward—build something low-cost and practical that could actually make Sri Lanka’s public bus system safer, faster, and more reliable. What came out of it was a complete system built around IoT devices, a solid backend, and real-time updates that tackled several real-life issues passengers and drivers face every day.

In Sri Lanka, one of the biggest headaches is that drivers don’t always know if someone’s waiting at a stop or how far ahead the next bus is. That leads to wasted stops, passengers getting left behind, and buses cutting each other off trying to get ahead. It’s frustrating for everyone. This system went after those problems directly by using GPS, cloud tech, and mobile alerts to keep both sides passengers and drivers on the same page.

The hardware part included a GPS tracker, an LCD for the driver, a SIM800L for sending data over 3G, and a Bluetooth-connected ESP32-CAM to keep an eye on how full the bus was. All of this fed data into Firebase in real-time. Even without Wi-Fi, the SIM800L kept updates coming every 10 seconds. That consistency meant no matter where the bus was—even in patchy signal zones it could still send out key info.

A major highlight was how the system figured out the road distance between buses. This wasn’t just straight-line guessing—it used Google’s Distance Matrix API to get the actual driving distance. The backend, hosted on DigitalOcean, grabbed GPS points from all buses on the route and calculated who was ahead and by how much. Then it sent that number to the driver's LCD so they could avoid getting too close or pulling unsafe stunts just to stay ahead.

On the passenger side, the app kept things simple. A button that says "I’m Waiting" sends a signal straight to Firebase with the stop ID and time. The bus device grabs that data in seconds and shows how many people are waiting. That quick feedback loop helped make the whole ride smoother and cut down on confusion.

The ESP32-CAM also played its part. Installed on the bus ceiling, it snapped photos every 10 to 15 seconds. Bluetooth handled the transfer to the main board, which then passed the images up to Firebase. This let the backend keep a visual log of how crowded each ride was. Those photos could later be analyzed or even feed into AI tools for smart crowd estimation.

What really helped was how well the pieces worked together. GPS stayed steady. Image transfers via Bluetooth were quick. Firebase handled syncing like a champ. Even when traffic conditions or signal strength shifted, the system kept up.

The entire setup was also built with growth in mind. Any one part—whether it’s the app, the backend, or the onboard device—could be swapped or upgraded without dragging the whole system down. That makes room for future add-ons like NFC ticket scanning, voice alerts for tourists, or predictive tools that use past data to guess where people will be and when[30].

From a big-picture angle, this project checks a lot of important boxes. It supports UN Sustainable Development Goals by making urban transport safer and more efficient. It’s cost-

effective, so it could actually be rolled out at scale, not just in Sri Lanka but in other countries dealing with the same headaches.

Of course, there's still room to improve. Adding NFC-based ticketing would help with fare tracking. Training an AI to read crowd sizes from the image feed would take things up a notch. Even moving from Firebase to something like PostgreSQL could make it ready for handling thousands of buses. Serverless computing and better load balancing would help make sure it runs smoothly without draining budgets.

In the end, this project shows that you don't need expensive, complicated systems to make real change. With the right design and a good grasp of real-world limits, tech like this can completely reshape public transportation making it safer, more efficient, and far more user-friendly.

8. REFERENCES

- [1] WhereToFlow, "Transportation in Sri Lanka 101 - All You Need to Know!," *Where to Flow*, Dec. 13, 2023. [Online]. Available: <https://wheretoflow.com/transportation-in-sri-lanka-guide/#:~:text=Buses%20are%20the%20cheapest%20and>. [Accessed: Apr. 6, 2025].
- [2] M. Sharma, S. Kumar and A. K. Yadav, "Design and Implementation of IoT based Smart Public Transport System," in *Proceedings of the 2021 International Conference on Smart Electronics and Communication (ICOSEC)*, Trichy, India, Sept. 2021, pp. 1620–1625. doi: 10.1109/ICOSEC51865.2021.9591974. [Accessed: MAR 12, 2025].
- [3] R. Smith, "ESP32 TTGO T-Call SIM800L Setup," Random Nerd Tutorials, 2023. [Online]. Available: <https://randomnerdtutorials.com/esp32-sim800l-send-text-sms/> [Accessed: APRIL 7, 2025].
- [4] Firebase, "Firebase Realtime Database," Google, 2024. [Online]. Available: <https://firebase.google.com/products/realtime-database> [Accessed: APRIL 1, 2025].
- [5] DigitalOcean, "Droplets," DigitalOcean Docs, 2023. [Online]. Available: <https://docs.digitalocean.com/products/droplets/> [Accessed: APRIL 1, 2025].
- [6] Google, "Distance Matrix API." [Online]. Available: <https://developers.google.com/maps/documentation/distance-matrix/overview> [Accessed: APRIL 1, 2025].
- [7] M. F. M. A. Hakeem, N. A. Sulaiman, M. Kassim and N. M. Isa, "IoT Bus Monitoring System via Mobile Application," 2022 IEEE International Conference on Automatic Control and Intelligent Systems (I2CACIS), Shah Alam, Malaysia, 2022, pp. 125-130, doi: 10.1109/I2CACIS54679.2022.9815268. keywords: {Schedules;Databases;Urban areas;Prototypes;Mobile applications;Internet of Things;Monitoring;Bus Monitoring System;IoT;GPS;mobile apps;Wi-Fi},
- [8] H. D. Weligamage, S. M. Wijesekara, M. D. S. Chathwara, H. G. Isuru Kavinda, N. Amarasekera and N. Gamage, "An Approach of Enhancing the Quality of Public Transportation Service in Sri Lanka using IoT," 2022 IEEE 13th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON), Vancouver, BC, Canada, 2022, pp. 0311-0316, doi: 10.1109/IEMCON56893.2022.9946624. keywords: {Smart cards;Visualization;Systematics;Face recognition;Sociology;Mobile communication;Real-time systems;RFID;GPS;IoT;Image Processing;Arrival time Prediction},

[9] P. V. Crisgar, P. R. Wijaya, M. D. F. Pakpahan, E. Y. Syamsuddin and M. O. Hasanuddin, "GPS-Based Vehicle Tracking and Theft Detection Systems using Google Cloud IoT Core & Firebase," 2021 International Symposium on Electronics and Smart Devices (ISESD), Bandung, Indonesia, 2021, pp. 1-6, doi: 10.1109/ISESD53023.2021.9501928. keywords: {Cloud computing;Tracking;User interfaces;Internet;Automobiles;Telemetry;Smart devices;Internet of Things;Vehicle Tracking System;Theft Detection;Google Cloud IoT Core;Firebase},

[10] J. Kumar and S. Reddy, "IoT Based Smart Bus Tracking and Alert System," *International Journal of Innovative Technology and Exploring Engineering*, vol. 8, no. 11, pp. 190–194, 2019.

[11] D. Silva and N. Perera, "Improving Sri Lankan Public Transport through Intelligent Systems," *Sri Lanka Journal of Advanced Research*, vol. 5, no. 1, pp. 12–20, 2021.

[12] A. Sharma and K. Verma, "Public Bus Occupancy Monitoring Using Computer Vision," in *Proc. of 6th Int. Conf. on IoT in Social, Mobile, Analytics and Cloud*, IEEE, 2022, pp. 89–95.

[13] S. Fernando, "A Study on Transport Efficiency and Passenger Flow in Colombo," *Sri Lanka Transport Review*, vol. 33, pp. 25–37, 2020.

[14] H. Gunasekara, "GIS-Based Route Analysis for Bus Networks," *Geospatial Today*, vol. 15, no. 2, pp. 38–42, 2019.

[15] J. Singh and P. Kumar, "Bluetooth-Based IoT Communication in Smart Buses," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 7100–7108, Aug. 2020.

[16] J. Fernando, "IoT Enabled Real-Time Bus Monitoring System," in *Proc. of the 2022 IEEE Global Humanitarian Technology Conference*, pp. 233–240.

[17] S. Ali, "Design and Development of an Intelligent Bus Tracking System," *International Journal of Electronics and Communication Engineering*, vol. 6, no. 1, pp. 45–52, Jan. 2018.

[18] G. Mendis, “Analysis of Bus Delay Causes in Urban Colombo,” *Ceylon Journal of Transport*, vol. 28, no. 2, pp. 41–50, 2022.

[19] ESP32.net, “ESP32-CAM Technical Details,” [Online]. Available: <https://esp32.net/esp32-cam/> [Accessed: APRIL 3, 2025]..

[20] Node.js Foundation, “Node.js Documentation,” [Online]. Available: <https://nodejs.org/en/docs/> [Accessed: APRIL 2, 2025].

[21] Express.js, “Fast, unopinionated, minimalist web framework for Node.js,” [Online]. Available: <https://expressjs.com> [Accessed: APRIL 4, 2025].

[22] N. Abeysinghe, “A Review on Smart City IoT Applications in South Asia,” *IEEE SoutheastCon*, pp. 323–328, 2021.

[23] OV Technology, “OV5640 Camera Module Datasheet,” OmniVision, 2021. [Online]. Available: <https://www.ovt.com/sensors/OV5640> [Accessed: APRIL 5, 2025].

[24] PM2 Docs, “Advanced Process Manager for Node.js,” [Online]. Available: <https://pm2.keymetrics.io> [Accessed: APRIL 6, 2025].

[25] Google Cloud, “Google Cloud Platform Overview,” [Online]. Available: <https://cloud.google.com> [Accessed: APRIL 4, 2025].

[26] T. Perera, “Smart Transport Frameworks in the Asia-Pacific Region,” *Asian Journal of Urban Planning*, vol. 2, no. 4, pp. 121–130, 2022.

[27] Arduino, “Introduction to ESP32,” Arduino Documentation, 2023. [Online]. Available: <https://docs.arduino.cc/hardware/esp32> [Accessed: APRIL 7, 2025].

[28] S. Rathnayake, “Smart IoT Solutions for Developing Countries: A Sri Lankan Perspective,” *International Journal of Smart Technology*, vol. 9, no. 1, pp. 15–24, 2021.

[29] Transport for London, “Passenger Information Display Systems,” [Online]. Available: <https://tfl.gov.uk> [Accessed: Feb. 22, 2024].

[30] A. Gunawardana, “An Evaluation of Open Data for Public Transport Systems in Sri Lanka,” *SL Journal of ICT Research*, vol. 4, no. 2, pp. 77–84, 2023.

9. APPENDICES

Paper Title	Uploaded	Grade	Similarity
FinalDoc(IT21192050).pdf	04/11/2025 11:13 AM	--	<div><div></div>4%</div> ↑ ↓ ≡

Appendix - A : Plagiarism report