VISVESVARAYA TECHNOLOGICAL UNIVERSITY

Jnana Sangama, Belgavi-590018

A PROJECT REPORT

On

"BuzzYatra"

A Dissertation Submitted in partial fulfillment of the requirement for the degree of

BACHELOR OF ENGINEERING

In

COMPUTER SCIENCE & ENGINEERING

Submitted by

ABHISHEK N	1RG22CS005
ADITYA PRAMOD DESHPANDE	1RG22CS009
RONEEL V	1RG22CS066
SUHAS J	1RG22CS080

Under The Guidance of

Dr. Arudra A

Head of the Department

Dept. of CSE

RGIT, Bengaluru-32

Department of Computer Science & Engineering

RAJIV GANDHI INSTITUTE OF TECHNOLOGY

Chola Nagar, R.T Nagar Post, Bengaluru-560032

2025-2026

RAJIV GANDHI INSTITUTE OF TECHNOLOGY

(Affiliated to Visvesvaraya Technological University)

Chola Nagar, R.T Nagar Post, Bengaluru-560032

Department of Computer Science & Engineering

CERTIFICATE

Phase-II

This is to certify that the Project Report titled **"BuzzYatra"** is a bonafide work carried out by **Mr. Abhishek N (USN 1RG22CS005), Mr. Aditya Pramod Deshpande (USN 1RG22CS009), Mr. Roneel V (USN 1RG22CS066) and Mr. Suhas J (USN 1RG22CS080)** in partial fulfillment for the award of **Bachelor of Engineering** in **Computer Science and Engineering** of the **Visvesvaraya Technological University, Belgavi**, during the year **2025-2026.** It is certified that all corrections/suggestions given for Internal Assessment have been incorporated in the report. This project report has been approved as it satisfies the academic requirements in respect of project work (BCS786) prescribed for the said degree.

Signature of Guide	Signature of HOD	Signature of Principal
Dr. Arudra A	**Dr. Arudra A**	**Dr. D G Anand**
Head Of Department	Head Of Department	Principal
Dept. of CSE,	Dept. of CSE,	RGIT, Bengaluru
RGIT, Bengaluru	RGIT, Bengaluru	

External Viva

Name of the Examiners	**Signature with date**
1.	
2.	

VISVESVARAYA TECHNOLOGICAL UNIVERSITY

Jnana Sangama, Belgavi-590018

RAJIV GANDHI INSTITUTE OF TECHNOLOGY

DEPARTMENT OF COMPUTER SCIENCE & ENGINEERING

DECLARATION

We hereby declare that the project work entitled **"BuzzYatra"** submitted to the **Visvesvaraya Technological University, Belgavi** during the academic year **2025-2026**, is record of an original work done by us under the guidance of Dr. Ardra A**,** Head Of the Department, Department of Computer Science and Engineering, RGIT, Bengaluru in the partial fulfillment of requirements for the award of the degree of **Bachelor of Engineering** in **Computer Science & Engineering.** The results embodied in this project have not been submitted to any other University or Institute for award of any degree or diploma.

ABHISHEK N	1RG22CS005
ADITYA PRAMOD DESHPANDE	1RG22CS009
RONEEL V	1RG22CS066
SUHAS J	1RG22CS080

ACKNOWLEDGEMENT

We take this opportunity to thank our college **Rajiv Gandhi Institute of Technology, Bengaluru** for providing us with an opportunity to carry out this project work.

We express our gratitude to **Dr. D G Anand**, Rector, RGIT, Bengaluru to Principal, RGIT, Bengaluru for providing the resources and support without which the completion of this project would have been a difficult task.

We extend our sincere thanks to **Dr. Arudra**, Associate Professor and Head, Department of Computer Science and Engineering, RGIT, Bengaluru, for being a pillar of support and encouraging us in the face of all adversities.

We would like to acknowledge the thorough guidance and support extended towards us by **Dr. Arudra A,** Head of Department, Dept. of CSE, RGIT, **Mrs. Bhagyashri Wakde,** Assistant Professor, Dept. of CSE, RGIT, Bengaluru, **Mrs. Soniya Komal. V,** Assistant Professor, Dept. of CSE, RGIT, Bengaluru, **Dr. Latha,** Assistant Professor, Dept. of CSE, RGIT, Bengaluru and **Mrs. Deepti Murali,** Assistant Professor, Dept. of CSE, RGIT, Bengaluru. Their incessant encouragement and valuable technical support have been of immense help. Their guidance gave us the environment to enhance our knowledge and skills and to reach the pinnacle with sheer determination, dedication and hard work.

We also want to extend our thanks to the entire faculty and support staff of the Department of Computer Science and Engineering, RGIT, Bengaluru, who have encouraged us throughout the course of the Bachelor's Degree. We want to thank our family for always being there with full support and for providing us with a safe haven to conduct and complete our project. We are ever grateful to them for helping us in these stressful times. Lastly, we want to acknowledge all the helpful insights given to us by all our friends during the course of this project.

ABHISHEK N	1RG22CS005
ADITYA PRAMOD DESHPANDE	1RG22CS009
RONEEL V	1RG22CS066
SUHAS J	1RG22CS080

ABSTRACT

This project addresses the transportation challenges faced by new visitors and daily commuters in Bengaluru by developing a comprehensive mobile application. For tourists and new residents, the application offers a route and fare finding feature for public transport, enabling users to easily navigate the city by inputting their source and destination. Recognizing the needs of regular BMTC bus users, the application incorporates a proactive "Stop Alert" system. This feature allows commuters to input their desired destination stop and receive timely notifications as they approach it, for example, providing an alert when the bus is within a 500-meter radius of the designated stop, thus preventing missed stops. Furthermore, the application includes an "Emergency SOS" functionality, enabling users to instantly share their live location with pre-defined emergency contacts in critical situations, enhancing personal safety and security during transit. By integrating route planning, intelligent stop alerts, and emergency support, this project aims to improve the efficiency, convenience, and safety of public transportation in Bengaluru.

CONTENTS

LIST OF FIGURES / ILLUSTRATIONS

LIST OF TABLES

CHAPTER 1

INTRODUCTION

1.1 OVERVIEW

Public transportation is one of the most widely used and essential modes of travel in large cities like Bengaluru. With thousands of people depending on BMTC buses every day, passengers often face challenges such as missing their intended stops or feeling unsafe during emergencies. These issues are especially common among new commuters, students, tourists, and senior citizens who may not be familiar with the routes or bus stop names.

To address these challenges, BuzzYatra has been developed as a web-based application that aims to make public bus travel more convenient, reliable, and secure. The application focuses on two main features: a Stop Alert System and an SOS Emergency System.

The Stop Alert System notifies passengers one stop before reaching their destination through on-screen alerts or sound notifications. This ensures that the user can prepare to get down at the right stop without worrying about missing it. It is particularly useful in crowded buses or for passengers who are unfamiliar with certain routes or locations.

The SOS Emergency System allows the user to send an immediate emergency alert along with their live GPS location to their pre-registered emergency contacts. In case of distress situations such as health issues, harassment, or accidents this feature provides a quick and efficient way to seek help and ensures that assistance can be sent promptly.

As a web application, BuzzYatra runs directly in a browser, removing the need for installation or app downloads. This makes it accessible to anyone with an internet connection, whether on a mobile phone, tablet, or computer. The interface is designed to be lightweight, responsive, and user-friendly, making it easy for people of all age groups to use.

Overall, BuzzYatra serves as a simple yet powerful travel companion that improves the user’s commuting experience by offering real-time alerts and quick emergency assistance all from a single, easily accessible web interface

1.2 MOTIVATION

In modern urban cities like Bengaluru, public transportation plays a vital role in helping millions of commuters travel efficiently. Among the most widely used public services is the BMTC bus network, which connects almost every part of the city. Despite its extensive coverage, passengers frequently face challenges such as confusion about routes, missing bus stops, and the absence of an efficient mechanism to handle emergency situations during their journey.

Daily commuters often get distracted or are unaware of their approaching destination stop, especially when traveling through unfamiliar routes. This commonly results in missing the correct stop, causing inconvenience, delays, and added expenses. Tourists and newcomers to the city also struggle with route identification and stop recognition, which adds stress to their travel experience.

Another major concern is passenger safety. In situations such as health emergencies, harassment, or unexpected incidents, the lack of a quick and reliable communication system can lead to panic and delayed assistance. Although mobile devices and applications exist, many of them are complex, require installation, or consume more data, making them unsuitable for quick emergency response.

These real-world problems inspired the development of BuzzYatra, a simple and accessible web application designed specifically to make public transport safer and more convenient. The application focuses on solving two key pain points faced by commuters — missing stops and lack of immediate emergency assistance.

By integrating the Stop Alert System, BuzzYatra ensures that passengers receive a timely notification one stop before their destination. This feature reduces travel anxiety and helps users stay aware of their journey progress. Additionally, the SOS Emergency System empowers passengers to instantly share their live location with trusted contacts during emergencies, ensuring faster response and greater peace of mind.

Another motivation behind choosing a web-based approach is accessibility. Many commuters do not prefer downloading multiple heavy apps due to storage limits or privacy concerns. BuzzYatra, being browser-based, can be accessed instantly on any device with internet connectivity, offering a seamless and installation-free experience.

In essence, the motivation behind developing BuzzYatra is to promote a safer, smarter, and more connected travel experience by leveraging the simplicity and universality of web technologies. It reflects the growing need for smart solutions that improve urban mobility, commuter safety and user comfort.

1.3 PROBLEM IDENTIFICATION

In a rapidly growing metropolitan city like Bengaluru, public transportation systems such as the BMTC bus network are essential for daily commuting. However, despite their importance, commuters frequently face several practical challenges that affect the overall convenience and safety of their travel experience. Through user observations and analysis, the following problems were identified, which form the foundation for developing the BuzzYatra web application.

1. **Missing Bus Stops:**

One of the most common issues faced by passengers, especially new commuters and tourists, is missing their destination stop. This often occurs because bus stop names are not always announced clearly, or passengers are distracted during travel. Missing a stop not only causes inconvenience but also leads to delays and additional travel costs.

2. **Lack of Real-Time Alerts:**

Existing navigation or public transport applications, such as Google Maps or third-party transit apps, provide route guidance but do not offer stop-specific alerts that notify passengers before reaching their destination. This absence of a dedicated stop alert feature leaves users dependent on manual monitoring throughout their journey.

3. **No Immediate Emergency Communication System:**

In case of emergencies—such as health issues, harassment, or accidents—commuters have no simple, quick, and reliable way to notify their trusted contacts with precise location details. The lack of such an SOS mechanism can lead to delays in seeking help and increased risk for the passenger.

4. **Dependence on Heavy Mobile Applications:**

Most available transport-related solutions are large mobile apps that require installation, permissions, and constant updates. These apps consume significant device storage and mobile data, making them unsuitable for users who prefer lightweight, accessible solutions.

5. **Limited Accessibility for Tourists and Occasional Users:**

Many apps are designed primarily for frequent commuters and may not cater to tourists or first-time travelers who are unfamiliar with local routes. There is a lack of a user-friendly platform that provides simple and direct functionality without requiring account setup or technical knowledge.

6. **Lack of Integration Between Travel Assistance and Safety Features:**

Current solutions typically focus either on navigation or emergency assistance, but very few integrate both in a single platform. This gap creates a fragmented experience for users who need both functionalities during their commute.

Considering these challenges, there is a strong need for a lightweight, accessible, and integrated solution that can provide stop-based alerts and immediate emergency support. The BuzzYatra web application aims to fill this gap by offering a Stop Alert System to prevent missed stops and an SOS Emergency System for quick and efficient communication during distress situations — all within a browser-based interface that requires no installation.

1.4 SCOPE

The scope of this project is limited to developing a web-based application that provides:

1. A Stop Alert System to notify users before reaching their destination stop.
2. An SOS Emergency System to share the user’s live location and alert contacts in emergency situations.

The application is designed using standard web technologies and can be accessed from any browser without the need for installation. It focuses solely on improving the daily travel experience and ensuring safety during public transportation**.**

Future enhancements may include features such as route and fare guidance, real-time bus tracking, and multilingual support. However, the current scope strictly focuses on delivering the two core modules in a reliable, user-friendly, and performance-efficient manner.

In summary, the scope of the BuzzYatra web application is to provide a lightweight, cross-platform, and user-centric travel assistant that improves the everyday commute by offering stop-based alerts and a quick-response SOS feature — thereby making public transportation safer and more convenient for all users.

1.5 OBJECTIVE AND METHODOLOGY

Objectives:

1. To design and develop a web application that alerts users before their destination stop.
2. To implement an SOS feature that shares live location with emergency contacts.

Methodology**:**

1. Analyse commuter problems and define requirements.
2. Design user interface using web technologies.
3. Implement GPS-based stop detection and alert mechanism.
4. Develop SOS feature integrated with live location tracking.
5. Test the system for accuracy, usability, and performance.

1.6 EXISITING SYSTEM

In the current scenario, commuters depend on various mobile applications and digital platforms to plan their travel and navigate through public transport systems. Applications such as Google Maps, Moovit, and other third-party transit apps offer general navigation and route-finding services. While these tools are helpful for route discovery and travel estimation, they do not cater specifically to the practical problems faced by daily bus commuters, such as missing stops or managing emergencies during travel.

Most of the existing systems focus on route planning and real-time bus tracking, but they lack stop-level alerts and instant emergency communication features. For instance, although Google Maps can show the path of a bus route, it does not provide an automatic alert when the user's destination stop is approaching. As a result, commuters must constantly monitor their journey, which can be inconvenient, especially in crowded or noisy environments.

Another major limitation of existing solutions is their dependence on installation. Most of these applications are heavy mobile apps that require continuous background operation, GPS permissions, and frequent updates. This makes them less accessible to users with limited phone storage, low-end devices, or poor network connectivity.

Furthermore, there is no integrated SOS system within most navigation apps that allows users to quickly notify their emergency contacts during distress situations. Passengers who encounter emergencies must manually call or message contacts, which can be time-consuming and unsafe when immediate help is needed.

1.7 PROPOSED SYSTEM

The proposed system, named BuzzYatra, is a web-based application developed to improve the commuting experience of bus passengers by offering two key features — a Stop Alert System and an SOS Emergency System. The system aims to provide a simple, efficient, and accessible platform that can be used by anyone without requiring installation or complex setup.

The Stop Alert System is designed to assist passengers by sending notifications one stop before their destination. The user inputs their target stop, and the system uses the device's GPS through the Geolocation API to track the user's current location during the journey. Once the user approaches the selected stop, the system automatically triggers a visual and/or audio alert. This feature ensures that passengers do not miss their destination, even if they are unfamiliar with the area or distracted during the trip.

The SOS Emergency System enables passengers to send an emergency alert to their pre-registered contacts with just a single click. When activated, the system instantly sends the user's live GPS location along with a preset message to their trusted contacts via integrated communication APIs or web services. This ensures that help can be reached quickly during medical emergencies, safety threats, or other distress situations.

1.8 OUTCOME OF THE PROJECT

The outcomes expected from the BuzzYatra system are:

1. A fully functional web application for bus commuters.
2. Real-time stop alert feature to prevent missing stops.
3. SOS alert system for quick emergency response.
4. Improved accessibility without requiring app installation.
5. Enhanced safety and convenience for public transport users.

1.9 INTRODUCTION SUMMARY

This Introduction chapter gives the overview of the BuzzYatra project and provides a short description of the proposed system. It highlights the motivation behind developing a travel companion application that caters to both tourists and daily commuters, addressing key challenges such as route guidance, fare estimation, and safety concerns. The project aims to enhance the overall travel experience by integrating real-time route and fare information, stop-based alerts, and an SOS mechanism into a single user-friendly platform.

The necessity of this project arises from the increasing reliance on public transport in urban cities like Bengaluru, where travelers often face confusion due to lack of proper fare guidance, missed stops, or absence of emergency assistance. This chapter also points out the limitations of existing solutions, such as fragmented apps that provide only partial support, and explains how BuzzYatra overcomes these limitations by offering a unified and reliable platform.

1.10 REPORT ORGANIZATION

This report is organized into eight chapters, each describing a specific phase of the BuzzYatra web application development process. The structure of the report is as follows:

Chapter 1 – Introduction:

This chapter provides a complete introduction to the BuzzYatra web application. It includes the project overview, motivation, problem identification, scope, objectives, and methodology. It also explains the existing system, the proposed system, expected outcomes, and concludes with an introduction summary.

Chapter 2 – Literature Survey:

This chapter presents a detailed study of the existing research papers, projects, and applications related to public transport assistance, stop alert mechanisms, and SOS emergency systems. It highlights the limitations of the current solutions and the improvements introduced by the BuzzYatra system.

Chapter 3 – System Analysis:

This chapter explains the feasibility study, system requirements, and the functional and non-functional aspects analyzed during the planning phase. It outlines the proposed system's workflow and discusses how the requirements were finalized.

Chapter 4 – Requirement Analysis:

This chapter provides detailed information about the hardware and software requirements necessary for developing and running the BuzzYatra web application. It also lists the functional and non-functional requirements of the system.

Chapter 5 – System Design:

This chapter explains the overall design of the BuzzYatra system, including architecture diagrams, data flow diagrams (DFD), and UML models. It describes how the Stop Alert and SOS modules interact with the user interface and backend components.

Chapter 6 – Implementation:

This chapter focuses on the actual development process, describing the technologies used, coding structure, and module-wise implementation. It explains how the Stop Alert System and SOS System were developed and integrated into the web application.

Chapter 7 – Testing:

This chapter discusses the various testing techniques applied to ensure system reliability and accuracy. It includes test cases, results, and validation reports confirming that both features work as intended under different scenarios.

Chapter 8 – Output and Conclusion:

This chapter presents screenshots, sample outputs, and the final results of the application. It concludes with the overall achievements of the project, limitations, and possible future enhancements for the BuzzYatra web application.

CHAPTER 2

LITERATURE SURVEY

The Literature Survey is an essential phase of project development. It involves studying existing systems, related research works, and available technologies to gain insights into the current state of solutions and identify their limitations. This helps in designing a more efficient, innovative, and problem-specific system.

For the BuzzYatra web application, the literature survey focused on understanding existing transport assistance applications, emergency alert systems, and web-based location tracking technologies.

2.1 OVERVIEW

With the rapid urbanization of cities, public transportation systems such as buses have become one of the most widely used and affordable modes of travel. However, passengers often face difficulties such as missing their stops, lack of real-time travel updates, and the absence of emergency communication mechanisms during travel. In cities like Bengaluru, where the bus network is dense and routes are complex, these challenges are especially common among new commuters, students, and tourists.

Over the years, several navigation and transport-related applications like Google Maps, Moovit, and Transit have been developed to assist users in planning their travel routes. These applications primarily focus on providing directions, estimated travel time, and real-time vehicle locations. However, they lack specific features that address the commuter's personal needs, such as custom stop-based alerts and integrated emergency (SOS) systems. Most of these existing solutions are also mobile-based and require installation, making them inaccessible to users who prefer lightweight, browser-based platforms.

The BuzzYatra web application was conceptualized to overcome these limitations by providing a web-based, user-friendly, and accessible solution that enhances the experience of bus commuters. The application is designed with two main modules — the Stop Alert System and the SOS Emergency System. The Stop Alert System ensures that users are notified before reaching their destination, preventing them from missing their stops. The SOS System provides an instant communication mechanism to alert registered emergency contacts in case of distress situations, sharing the user's live location for immediate assistance.

2.2 Mobile Application to locate users whereabouts and send SOS messages

NAME	YEAR	AUTHOR	FEATURE	ADVANTAGE	DISADVANTAGE
Mobile Application to Locate and Track User's Whereabouts and Send SOS Messages	2019	Prof Fadhil Prof Ibrahim	A mobile app that helps users share their real-time location and send SOS alerts to selected contacts during emergencies	Provides safety assurance by notifying trusted contacts instantly with live GPS data.	Limited to network connectivity; may not function properly in areas with poor GPS or internet coverage

Table: 2.1: Mobile Application to locate users whereabouts and send SOS messages

This paper proposes a mobile application designed to ensure user safety by providing real time location tracking and an emergency SOS alert system. The app allows users to send their live coordinates to predefined contacts at the press of a button. It focuses on helping people in distress or unsafe situations by ensuring that help can reach them promptly. The main challenge is dependency on continuous network and GPS availability for accurate tracking.

2.3 Easy Navigation of Bus using Alert System

NAME	YEAR	AUTHOR	FEATURE	ADVANTAGE	DISADVANTAGE
Easy Navigation of Bus using Alert System	2020	Snehal A.Ghodake Prof.Megha pallewr	A system that assists passengers by providing bus location updates and stop alerts through an automated alert mechanism	Improves convenience And accessibility for passengers, especially those unfamiliar with routes.	Requires integration with GPS-enabled buses and consistent internet access for accuracy.

Table:2.2: Easy Navigation of Bus using Alert System

This paper introduces a smart bus navigation and alert system aimed at improving the commuting experience. It uses GPS to track the bus in real time and alert passengers before reaching their destination stop. The system is particularly useful for people who are new to the area or tend to miss stops. Its main limitation is reliance on accurate GPS data and stable network connectivity.

2.4 Bus Tracking System using Mobile GPS Technology

NAME	YEAR	AUTHOR	FEATURE	ADVANTAGE	DISADVANTAGE
Bus Tracking System using Mobile GPS Technology	2024	Jayaram S.P Jayakumar D	Uses mobile-based GPS tracking instead of costly onboard GPS hardware; integrates MQTT, WebSocket, and Node.js backend for real-time updates; includes user and driver apps for accessibility.	Uses mobile GPS instead of costly onboard hardware; integrates MQTT, WebSocket, and Node.js for real-time updates with user and driver apps.	Dependent on driver's mobile GPS and internet connectivity; accuracy may vary with mobile signal; potential privacy concerns.

Table: 2.3: Bus Tracking System using Mobile GPS Technology

This study introduces a cost-effective, mobile-based bus tracking solution leveraging the GPS functionality of drivers' smartphones. It replaces traditional GPS hardware with a software-driven model that uses MQTT and WebSocket for live communication. The dual-application system enhances user experience and real-time monitoring, ensuring affordability and scalability in college transportation networks.

2.5 Campus Compute: An Innovative to College Bus Tracking and Management

NAME	YEAR	AUTHOR	FEATURE	ADVANTAGE	DISADVANTAGE
Campus Commute: An Innovative Approach to College Bus Tracking and Management	2025	Mary A Albert J	Uses mobile-based GPS tracking instead of costly onboard GPS hardware; integrates MQTT, WebSocket, and Node.js backend for real-time updates; includes user and driver apps for accessibility.	Integrates ESP32, SIM800L, NEO-7M GPS, Django backend, and Flutter app for real-time tracking; uses WebSocket and MQTT for continuous data and includes an admin dashboard for route control.	Dependent on GPS and GSM module accuracy; setup complexity due to hardware integration; requires reliable network connectivity for update

Table: 2.4: Campus Compute: An Innovative to College Bus Tracking and Management

The Campus Commute system introduces a complete loT-powered platform for campus transportation management. Using GPS hardware modules with MQTT and Django backend, it ensures accurate live tracking, route optimization, and user-friendly app notifications. The system reduces manual effort and enhances safety and scheduling accuracy, offering a sustainable and affordable transportation model for universities

2.6 Real -Time College Bus Tracking and Notification System for Student Convenience

NAME	YEAR	AUTHOR	FEATURE	ADVANTAGE	DISADVANTAGE
Real-Time College Bus Tracking and Notification System for Student Convenience	2025	Sandhiya B Ravichandra	Provides GPS-enabled bus tracking with a web-based platform, 2 km proximity alert, and live passenger count integration for optimizing bus occupancy and minimizing waiting time.	Reduces student waiting time by 35%, improves bus occupancy by 20%, and enhances transportation efficiency; offers automated alerts, real-time updates, and cloud-based data management for better reliability.	Dependent on GPS and stable internet connectivity; may face signal loss in certain areas; scalability and hardware setup can be complex for large deployments.

Table: 2.5: Real -Time College Bus Tracking and Notification System for Student Convenience

This research presents a real-time college bus tracking and notification system using GPS and cloud technologies to enhance campus transport efficiency. It sends automatic alerts when buses approach within 2 km and tracks passenger counts for better seat management. The system improves punctuality and user convenience while reducing overcrowding. However, it relies heavily on GPS signal stability and continuous internet access for optimal operation.

2.7 LITERATURE SURVEY SUMMARY

The main aim of this project is to integrate commuter convenience and safety into a single web application that is easily accessible through any device with a browser and internet connection. Unlike mobile apps, which require installation and frequent updates, BuzzYatra offers an instant, lightweight, and responsive experience for all types of users.

By studying existing systems and identifying their limitations, BuzzYatra was developed as a smart and secure public transport assistance system, designed to enhance the reliability and safety of daily commuting through modern web technologies.

CHAPTER 3

SYSTEM ANALYSIS

System analysis is an essential phase of the software development process that involves studying and understanding the existing systems, identifying the problems, and determining the requirements for the new system. It forms the foundation for designing and implementing an effective solution.

For the BuzzYatra web application, the system analysis focuses on understanding the difficulties faced by bus commuters, defining the requirements of the proposed solution, and analyzing its feasibility in terms of technology, cost, and performance.

3.1 SYSTEM STUDY

The BuzzYatra web application is designed to improve the commuting experience of public transport users by providing two essential features — a Stop Alert System and an SOS Emergency System.

The application works on any browser and helps users by alerting them before their destination stop and allowing them to send quick SOS alerts in emergencies. It leverages the OpenStreetMap API to obtain route and location data and uses the HTML5 Geolocation API for real-time tracking.

The system architecture follows the MERN stack model — MongoDB, Express.js, React, and Node.js — ensuring a scalable, efficient, and responsive web application built entirely using JavaScript as the base language.

3.2 EXISTING SYSTEM ANALYSIS

The existing systems, such as Google Maps and third-party transport apps, primarily offer route information and navigation assistance. However, they have the following limitations:

- Lack of stop-based alerts, causing users to miss their destinations.
- No integrated emergency SOS feature to notify contacts in case of danger or emergency.
- Dependence on heavy mobile applications that consume more data and require frequent updates.
- Limited accessibility for users without smartphones or those who prefer browser-based tools.

These shortcomings highlight the need for a lightweight, easily accessible web-based solution that provides both stop alerts and quick emergency response capabilities.

Hence, there is a need for a lightweight, web-based system that combines travel alerts and emergency safety features in one accessible interface.

3.3 PROPOSED SYSTEM ANALYSIS

The proposed BuzzYatra web application addresses these challenges by offering a simplified and integrated platform with two core modules:

- Stop Alert System: Provides real-time alerts before the destination stop.
- SOS Emergency System: Sends live location details to registered contacts in case of an emergency.

Features of the Proposed System:

- Web-Based Platform: Accessible directly from any browser without installation.
- Frontend in React: Ensures a smooth, responsive, and dynamic user interface.
- Backend in Node.js with Express.js: Handles API requests, alerts, and SOS communications efficiently.
- Database in MongoDB: Stores user information, contacts, and system logs securely.
- OpenStreetMap API Integration: Provides accurate routing and location information.
- Real-Time GPS Tracking: Uses the browser's Geolocation API to track user movement continuously.
- Privacy and Security: Shares location data only when required, ensuring user confidentiality.

This system architecture makes BuzzYatra scalable, fast, and easy to maintain while providing reliable travel assistance to users.

3.4 FEASABILITY STUDY

Feasibility analysis determines whether the system can be implemented effectively with available technologies, resources, and time.

3.4.1 Technical Feasibility

The project is technically feasible since it is developed entirely using JavaScript, a widely supported and flexible language. The use of the MERN stack ensures strong integration between the frontend, backend, and database layers.

OpenStreetMap API enables access to public route data without heavy licensing costs, and the Geolocation API allows real-time tracking. The system requires only a modern browser and an internet connection, making it technically viable and easy to deploy.

3.4.2 Economic Feasibility

All technologies used — Node.js, Express.js, React, MongoDB, and OpenStreetMap API — are open-source and free. This minimizes development and deployment costs. Hosting the application on affordable cloud services such as Render, Vercel, or Heroku ensures a low-cost operational setup.

3.4.3 Operational Feasibility

The system is simple, intuitive, and requires no technical expertise to use. It can be accessed instantly from a web browser, without installation, making it operationally suitable for daily commuters, students, and tourists.

3.4.4 Schedule Feasibility

The project followed a systematic development plan — including requirement gathering, design, implementation, and testing phases. The modular structure of React components and Express routes allowed parallel development, ensuring timely completion within the proposed schedule.

3.5 SYSTEM REQUIREMENTS STUDY

A detailed study of system requirements was conducted to identify the needs of both users and the system.

User Requirements:

- Ability to set a destination stop easily.
- Receive alerts before reaching the destination.
- Quickly send an SOS alert with live location.
- Accessible through any device with a browser.

System Requirements:

- GPS-enabled device for real-time tracking.
- Active internet connection.
- Secure data storage for contacts and user logs.
- Integration with OpenStreetMap for route data retrieval.

3.6 SYSTEM DESIGN CONSIDERATIONS

The system design follows a three-tier architecture consisting of:

1. Frontend Layer (React):
   - Provides an interactive interface for users to input data, select destinations, and trigger alerts. React components handle rendering and dynamic state updates.
2. Backend Layer (Node.js + Express.js):
   - Handles the business logic, manages API calls, communicates with the database, and processes location and SOS requests.
3. Database Layer (MongoDB):
   - Stores user details, stop information, and SOS records in JSON-like documents, allowing flexibility and scalability.
4. External APIs:
   - OpenStreetMap API for location and route visualization.
   - HTML5 Geolocation API for fetching real-time GPS coordinates.

This modular structure ensures that each layer functions independently while maintaining seamless data flow across the system.

3.7 SYSTEM ANALYSIS SUMMARY

The system analysis confirms that the proposed BuzzYatra web application is both technically and operationally feasible. By using the MERN stack and OpenStreetMap API, the project ensures efficient, secure, and scalable implementation. The system meets its primary goals helping commuters avoid missing stops and ensuring safety through instant SOS alerts — while being lightweight, accessible, and fully web-based

CHAPTER 4

REQUIREMENT ANALYSIS

Requirement analysis is one of the most important stages of software development. It focuses on identifying what the system should do, the constraints under which it must operate, and the resources needed to implement it successfully.

This chapter outlines the functional, non-functional, software, and hardware requirements of the BuzzYatra web application. The system is built using the MERN stack — MongoDB, Express.js, React, and Node.js — with JavaScript as the primary programming language. The application integrates the OpenStreetMap API for route and location information and the Geolocation API for live tracking.

4.1 FUNCTIONAL REQUIREMENTS

Functional requirements describe the specific operations and functionalities that the BuzzYatra web application must perform to achieve its objectives. The core modules of the system are the Stop Alert System and the SOS Emergency System.

1. Destination Selection:
   - Users can select or enter their destination stop using the OpenStreetMap interface.
   - The map is rendered using React components integrated with the OpenStreetMap API.
2. Real-Time Location Tracking:
   - The system uses the HTML5 Geolocation API to track the user's position during the trip.
   - Location data is continuously updated in the frontend for accurate stop detection.
3. Stop Alert System:
   - The backend compares the user's real-time location with the selected destination stop.
   - When the user is one stop away, an alert (audio/visual) is triggered through the browser.
4. SOS Emergency System:
   - When the SOS button is pressed, the system captures the user's live location and sends an alert message to the registered emergency contacts.
   - The alert includes latitude, longitude, and timestamp for accurate location tracking.
5. Data Storage and Retrieval:

- ➢ MongoDB stores user profiles, contact details, SOS logs, and travel history in JSON format.
- ➢ Data is exchanged between frontend and backend using RESTful APIs in Express.js.

1. Error Handling and Notifications:

- ➢ The system displays meaningful messages for issues like GPS errors, network disconnection, or missing contact data.

2. Responsive Web Interface:

- ➢ The React frontend ensures the interface adapts to various screen sizes, offering smooth user experience on both mobile and desktop.

4.2 NON-FUNCTIONAL REQUIREMENTS

Non-functional requirements define the quality characteristics, performance goals, and constraints of the system.

1. Performance:
    - o Real-time tracking and alert triggers must respond within 2–3 seconds.
    - o The system should remain efficient even with multiple concurrent users.
2. Reliability:
    - ➢ The application must remain stable during long journeys.
    - ➢ Data loss or SOS failure should be minimized through proper backend validation.
3. Usability:
    - ➢ The interface should be intuitive and easy to navigate.
    - ➢ Important buttons like SOS should be prominently placed and accessible.
4. Security:
    - ➢ All communication between client and server must occur over HTTPS.
    - ➢ Sensitive information such as contacts and location data must be securely stored and accessed.
5. Scalability:
    - ➢ The architecture must support the addition of new features like route guidance or fare estimation with minimal code changes.
6. Compatibility:

- ➢ The web app must be compatible with all major browsers — Chrome, Firefox, Edge, and Safari.
- ➢ It should function efficiently on both Android and iOS devices.

7. Maintainability:
   - ➢ The system's modular MERN-based design allows easy maintenance, debugging, and feature upgrades.
8. Data Privacy:
   - ➢ The user's live location is accessed only during active trips or SOS alerts, ensuring privacy protection.

4.3 SOFTWARE REQUIREMENTS

The software requirements specify the development tools, platforms, and environments used for creating and deploying the BuzzYatra web application.

Operating system	:	Windows
Frontend	:	React.js, HTML5, CSS3, JavaScript
Backend	:	Node.js with Express.js framework
Language	:	JavaScipt
API	:	OpenStreetMap API, HTML5 Geolocation API
Testing Tool	:	Postman
Version Control	:	Git and GitHub
Deployment Platforms:		Render / Vercel
Browser Support	:	Chrome, Firefox, Safari, Edge

4.4 HARDWARE REQUIREMENTS

Since BuzzYatra is a web-based system, hardware requirements are minimal. The system mainly depends on the client's device capabilities and a reliable hosting environment.

Server-Side Requirements:

- Processor: Dual Core or higher
- RAM: Minimum 2 GB

- Storage: 1 GB or more (SSD preferred)
- Network: Stable internet connection
- Hosting: Render or Vercel platform supporting Node.js deployment

Client-Side Requirements:

- Device: Smartphone, Tablet, or Laptop
- Browser: Modern web browser with GPS support
- Internet: Wi-Fi / 4G / 5G connectivity
- GPS Hardware: Required for real-time tracking

4.5 REQUIREMENT SUMMARY

The BuzzYatra web application is designed to meet a combination of functional, non-functional, software, and hardware requirements that ensure its efficiency and reliability. Functionally, the system includes two major modules — the Stop Alert System and the SOS Emergency System — along with real-time location tracking and contact management features. These modules work together to improve commuter convenience and safety by providing timely alerts and quick access to emergency communication.

From a non-functional perspective, the system emphasizes security, scalability, reliability, and usability. It ensures that all user data and location information are transmitted securely, while the application remains compatible across multiple browsers and devices. The interface is designed to be simple, responsive, and easy to use, even for first-time users.

In terms of software, the project is built entirely using the MERN stack, which includes React for the frontend, Node.js with Express.js for the backend, and MongoDB for database management. It also integrates OpenStreetMap API for mapping and Postman for API testing, with deployment handled through cloud platforms such as Render or Vercel. These tools and technologies ensure that the system remains modern, cost-effective, and easily maintainable.

The hardware requirements are minimal, as the system runs entirely through web browsers. A GPS-enabled smartphone, tablet, or laptop with an active internet connection is sufficient for users to access the application. On the server side, a standard hosting environment with Node.js support and stable internet connectivity is adequate for deployment.

Overall, the requirement summary confirms that the BuzzYatra web application is a lightweight, scalable, and secure system that effectively meets the needs of public transport commuters through an accessible web-based platform.

4.6 REQUIREMNET ANALYSIS SUMMARY

The requirement analysis phase establishes a clear roadmap for the development of BuzzYatra. By defining the system's functional and non-functional needs, the project ensures smooth integration between React (frontend), Express.js (backend), and MongoDB (database).

The inclusion of OpenStreetMap API and Postman-based testing ensures that the system is robust, secure, and ready for real-time web deployment on platforms such as Render or Vercel.

This structured analysis lays a solid foundation for the subsequent System Design and Implementation phases of the project.

CHAPTER 5

SYSTEM DESIGN

System design is a crucial phase in software development where the system's architecture, data flow, and component interactions are defined in detail. It provides a clear blueprint for the implementation phase, ensuring that the system meets all specified requirements efficiently and logically.

The design of BuzzYatra focuses on creating a simple, modular, and scalable architecture that ensures real-time performance, security, and maintainability. The system uses a three-tier MERN architecture — integrating the frontend (React), backend (Node.js + Express.js), and database (MongoDB) — along with external APIs such as OpenStreetMap for routing and Geolocation API for live tracking.

5.1 SYSTEM ARCHITECTURE

The BuzzYatra web application follows a client–server architecture model, where the client interacts with the server through RESTful APIs.

- The frontend is developed using React, providing an interactive and responsive interface for users to input destinations, view maps, and trigger SOS alerts.
- The backend is powered by Node.js and Express.js, handling all logic, routing, and API requests.
- The database layer uses MongoDB, a NoSQL database that stores user data, contact details, and alert logs in a flexible JSON-like format.
- The system communicates with the OpenStreetMap API for map rendering and uses the HTML5 Geolocation API for live GPS tracking.

Data Flow Summary:

1. The user accesses the BuzzYatra web application via a browser.
2. The frontend (React) requests map and location data through APIs.
3. The backend (Express.js) processes these requests, interacts with the database, and returns results in JSON format.
4. MongoDB handles all persistent storage for user and system data.
5. The system triggers alerts (Stop Alert or SOS) based on backend conditions and user actions.

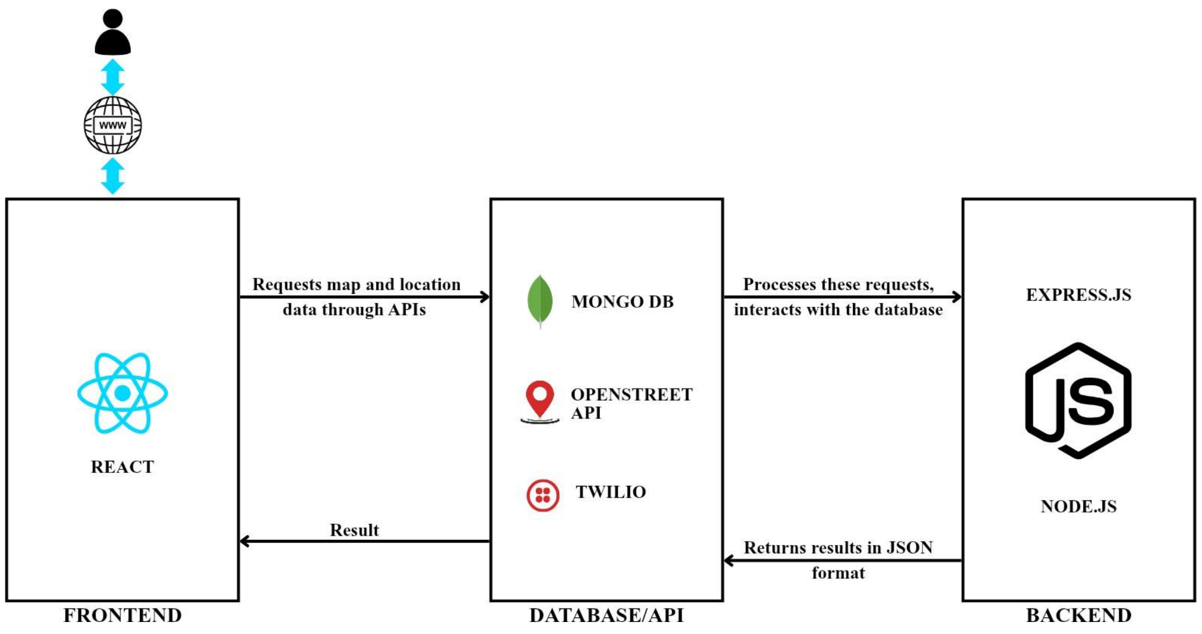

www
Requests map and location data through APIs
MONGO DB
OPENSTREET API
TWILIO
Processes these requests, interacts with the database
EXPRESS.JS
NODE.JS
REACT
Result
Returns results in JSON format
FRONTEND
DATABASE/API
BACKEND

Fig 5.1: System Architecture

5.2 SYSTEM ARCHITECTURE DIAGRAM

Diagram Description:

The architecture diagram illustrates the interaction between the system's components.

- **Frontend Layer:** Displays user interfaces for login, destination input, map view, and SOS alert.
- **Backend Layer (Node.js + Express.js):** Processes user requests, fetches location data, and communicates with the database.
- **Database Layer (MongoDB):** Stores and retrieves user contact details, trip data, and SOS logs.
- **External APIs:** OpenStreetMap provides real-time mapping data, while Geolocation API fetches live GPS coordinates. The communication between all layers occurs through RESTful APIs over HTTPS.

5.3 DATA FLOW DIAGRAM (DFD)

Level 0 (Context Diagram):

At the highest level, the User interacts with the BuzzYatra System through a web browser. The system process location tracking, sends stop alerts, and triggers SOS messages.

Level 1 DFD Description:

1. **User Inputs:** Destination stop and emergency contact details.
2. **System Processing:**
    a. Retrieves current GPS coordinates.
    b. Fetches route data from OpenStreetMap.
    c. Compares user location with destination coordinates.
3. **Outputs:**
    a. Stop alert notification when nearing the destination.
    b. SOS alert sent to registered contacts with live location details.

Data flows between **User**, **System Server**, and **MongoDB Database**, ensuring real-time updates and responses.

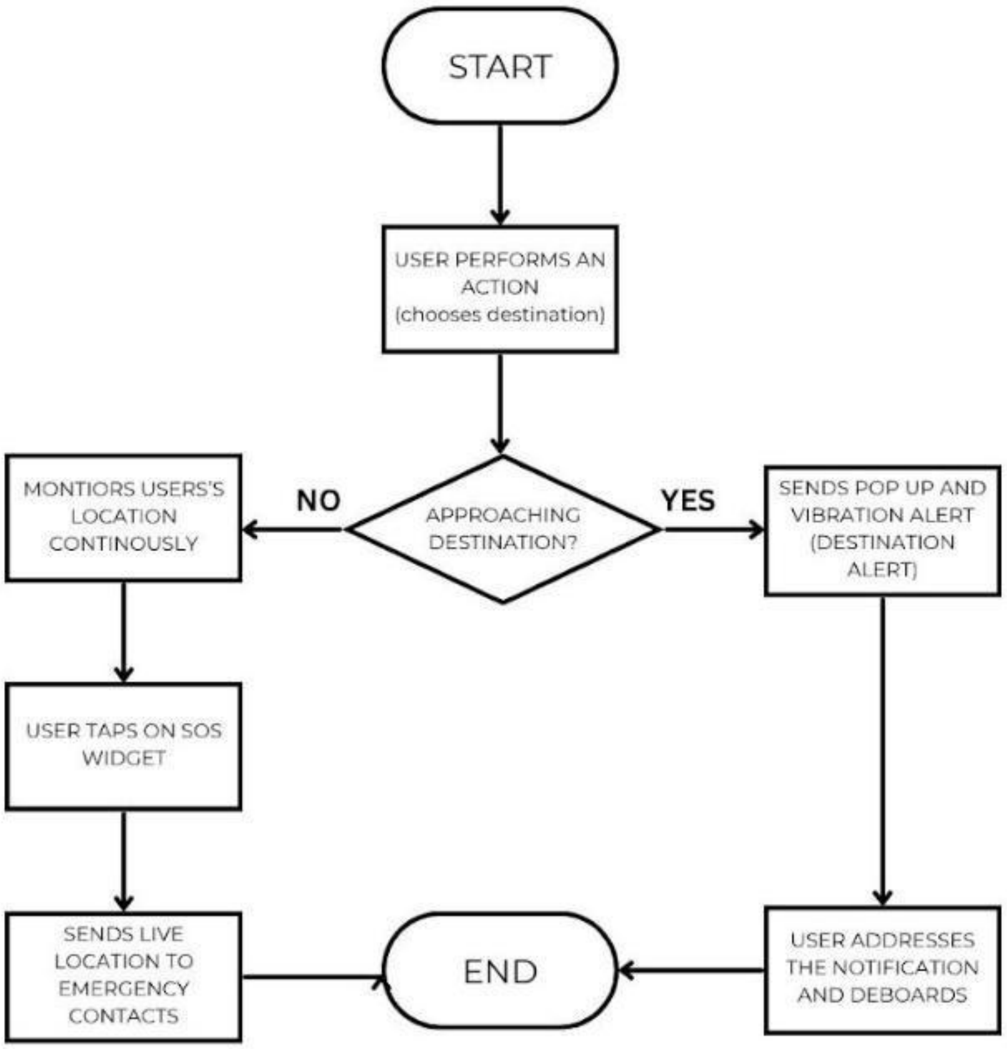

START
USER PERFORMS AN ACTION (chooses destination)
APPROACHING DESTINATION?
NO
YES
MONTIORS USERS'S LOCATION CONTINOUSLY
USER TAPS ON SOS WIDGET
SENDS LIVE LOCATION TO EMERGENCY CONTACTS
SENDS POP UP AND VIBRATION ALERT (DESTINATION ALERT)
USER ADDRESSES THE NOTIFICATION AND DEBOARDS
END

Fig 5.2 Data Flow Diagram

5.4 UML Diagrams

The following UML diagrams were considered during the design phase:

1. **Use Case Diagram:**
   Represents how users interact with the BuzzYatra system.
   a. **Actors:** User (commuter)
   b. **Use Cases:** Register Contact, Set Destination, Track Location, Receive Stop Alert, Send SOS Alert.
2. **Sequence Diagram:**
   Shows the step-by-step interaction between user, frontend, backend, and database.
   a. User requests are processed through React (frontend) → Express.js (backend) → MongoDB (database).
   b. Responses (alerts or confirmation messages) are sent back to the user.
3. **Activity Diagram:**
   Depicts the flow of actions from starting a trip to receiving alerts and sending SOS messages.

5.5 MODULE DESIGN

The **BuzzYatra** system is divided into two major modules:

5.5.1 Stop Alert System

The Stop Alert module is responsible for notifying the user before reaching their destination stop.

Functions:

- Fetches the user's current location using the Geolocation API.
- Retrieves destination coordinates via OpenStreetMap.
- Continuously compares the current location with the destination.

- Triggers an alert (sound/visual) when the user is near the destination.

Key Components:

- React Map Component (frontend)
- Express.js route handler for map data
- Location comparison logic (backend)

Data Flow:
User → React UI → Node.js/Express → MongoDB (store trip info) → React (Alert Trigger)

5.5.2 SOS Emergency System

The SOS module ensures user safety by enabling one-click emergency alerts.

Functions:

- User presses the SOS button during emergencies.
- System retrieves the user's live GPS coordinates.
- Sends an alert message with the user's location to registered contacts.
- Logs the SOS event in MongoDB for record-keeping.

Key Components:

- React SOS button (frontend)
- Express.js API for SOS processing
- MongoDB SOS collection for logs

Data Flow:
User → React (SOS Trigger) → Node.js/Express → MongoDB → Contact Notification
(Frontend Confirmation)

5.6 DATABASE DESIGN

Database Used: MongoDB
MongoDB is a NoSQL document-oriented database that stores data in flexible JSON-like documents, allowing easy scalability and fast querying.

Collections:

1. **Users Collection** – Stores user profile data.
    a. Fields: user_id, name, email, password, contact_list[]
2. **Contacts Collection** – Stores emergency contact details.
    a. Fields: contact_id, user_id, name, phone_number, email
3. **Alerts Collection** – Stores information related to Stop Alerts and SOS events.
    a. Fields: alert_id, user_id, alert_type, location, timestamp

This database structure allows efficient storage, retrieval, and management of real-time commuter data.

5.7 USER INTERFACE DESIGN

The user interface (UI) is designed using **React.js** to provide a clean, responsive, and accessible experience.
UI Features:

- Simple navigation menu for selecting modules.
- Interactive OpenStreetMap for destination input and tracking.
- Prominent SOS button for emergencies.
- Notification banners for alerts.
  The design follows minimalistic principles, ensuring ease of use across all devices.

5.8 GANTT CHART

TASK	WEEK 1-2	WEEK 3-4	WEEK 5-6	WEEK 7-8	WEEK 9-10	WEEK 11-12	WEEK 13-14
LITERATURE REVIEW	✓						
DATABASE COLLECTION AND PREPERATION		✓					
IMPLEMENTATION OF CODE			✓	✓			
EXPERIMENT AND TESTING					✓		
EVALUATION AND ANALYSIS						✓	
FINAL REPORT AND PRESESNTATION							✓

Fig 5.3 Gnatt Chart

5.9 SYSTEM DESIGN SUMMARY

The system design phase provides a clear structural and functional blueprint for the BuzzYatra webapp.

Using the MERN architecture, the system achieves modularity, scalability, and ease of maintenance. Integration of OpenStreetMap and Geolocation APIs enables accurate real-time tracking, while MongoDB ensures efficient data management.

This design ensures that BuzzYatra fulfills its primary objectives, providing a reliable Stop Alert System and a quick-response SOS Emergency System through a secure, efficient, and fully web-based platform.

CHAPTER 6

IMPLEMENTATION

The implementation phase involves the actual development of the BuzzYatra web application based on the design specifications. In this stage, each component of the system is developed, integrated, and tested to ensure it functions according to the requirements.

The project is implemented using the MERN stack MongoDB, Express.js, React, and Node.js with JavaScript as the core programming language. The OpenStreetMap API and HTML5 Geolocation API are integrated to provide location tracking and route visualization functionalities.

The main focus during implementation was on developing the two core modules: the Stop Alert System and the SOS Emergency System, ensuring both modules work seamlessly together in a responsive and secure environment.

6.1 SYSTEM SETUP AND CONFIGURATION

The BuzzYatra system setup involves preparing the development environment for both the frontend and backend, establishing database connections, and integrating APIs.

Frontend Setup (React)

- ➢ The frontend was developed using React.js, providing a component-based architecture for modular development.
- ➢ Key React components were created for navigation, maps, alerts, and SOS functionality.
- ➢ Styling was achieved using CSS3 and responsive design principles to ensure compatibility across devices.
- ➢ API calls between the frontend and backend were made using the fetch() method in JavaScript.

Backend Setup (Node.js + Express.js)

- ➢ The backend was built using Node.js runtime with Express.js framework for handling RESTful APIs.

- Express routes were created for user management, stop alerts, and SOS handling.
- The backend communicates with MongoDB for data storage and retrieval.
- Cross-Origin Resource Sharing (CORS) was configured to allow secure communication between the frontend and backend servers.

Database Configuration (MongoDB)

- MongoDB was used as the NoSQL database to store user details, contact lists, and alert logs.
- Data is stored in collections such as *Users*, *Contacts*, and *Alerts*, formatted in JSON structure.
- Mongoose, an Object Data Modeling (ODM) library, was used to define schemas and simplify database operations.

API Integration

- The OpenStreetMap API was integrated for map rendering and destination visualization.
- The HTML5 Geolocation API was used to continuously fetch the user's live coordinates.
- Combined, these APIs provide accurate tracking and alert triggering.

6.2 MODULE IMPLEMENTATION

The BuzzYatra web application is divided into two major modules:

6.2.1 Stop Alert System

This module ensures that the commuter receives a notification before reaching the destination stop.

Implementation Details:

- The user selects a destination stop on the OpenStreetMap interface.
- The system fetches the user's live GPS coordinates via the Geolocation API.

- A backend comparison algorithm calculates the distance between the current position and the destination coordinates.
- When the distance becomes less than a predefined threshold, a browser alert or notification sound is triggered through the frontend.
- The alert data is logged in MongoDB for reference.

6.2.2 SOS Emergency System

This module focuses on passenger safety by allowing the user to send a distress alert to registered emergency contacts.

Implementation Details:

- The user clicks the SOS button on the React interface.
- The frontend retrieves the current location coordinates using the Geolocation API.
- The coordinates and timestamp are sent to the backend via an Express.js API route.
- The backend retrieves the user's stored contact list from MongoDB and prepares an alert payload.
- The system sends notifications to the contacts and stores the SOS record in the database for tracking.

6.3 INTEGRATION OF FRONTEND AND BACKEND

Integration was achieved through RESTful APIs, where the frontend (React) communicates with the backend (Node.js + Express.js) using HTTP requests.

Key Integration Points:

- GET requests: Fetch user and stop data from MongoDB.
- POST requests: Store SOS alerts and contact information.
- PUT/DELETE requests: Update or remove user data when needed.
- Data exchange between frontend and backend occurs in JSON format, ensuring consistency and fast response times.

The integration was tested using Postman, ensuring that all endpoints responded correctly with proper status codes and data validation

CODE-

Frontend

```
import { useEffect, useMemo, useRef, useState } from 'react'

import './App.css'

import sound from './assets/sound.wav'


// Uses Leaflet loaded from CDN in index.html (global L)

function playAudio() {

  const audio = new Audio(sound)

  audio.play()

}


function App() {

  // form state

  const [stations, setStations] = useState([])

  const [from, setFrom] = useState('')

  const [to, setTo] = useState('')

  const [alertDistance, setAlertDistance] = useState('')


  // display state

  const [distance, setDistance] = useState('--')

  const [duration, setDuration] = useState('--')
  const [alertDisp, setAlertDisp] = useState('--')


  // map refs

  const mapRef = useRef(null)

  const mapContainerRef = useRef(null)

  const layersRef = useRef({ route: null, from: null, to: null, user: null, circle: null })


  // lazy getter for global L

  const L = useMemo(() => (typeof window !== 'undefined' ? window.L : undefined), [])


  // init map once

  useEffect(() => {

    if (!L || !mapContainerRef.current || mapRef.current) return

    const m = L.map(mapContainerRef.current).setView([20.5937, 78.9629], 5)

    L.tileLayer('https://{s}.tile.openstreetmap.org/{z}/{x}/{y}.png', {

      maxZoom: 19,

      attribution: '&copy; OpenStreetMap contributors'

    }).addTo(m)

    mapRef.current = m

  }, [L])


  // fetch stations on mount

  useEffect(() => {

    async function run() {
      try {
        const res = await fetch('http://localhost:4000/api/getStation')
        const data = await res.json()
        setStations(Array.isArray(data) ? data : [])
        if (Array.isArray(data) && data.length > 0) {
          setFrom(data[0].name || '')
          setTo(data[0].name || '')
        }
      } catch (e) {
        console.error('Error fetching stations:', e)
      }
    }
    run()
  }, [])

  function clearLayers() {
    const { route, from, to, user, circle } = layersRef.current
    const map = mapRef.current
    if (!map) return
    if (route) { map.removeLayer(route); layersRef.current.route = null }
    if (from) { map.removeLayer(from); layersRef.current.from = null }
    if (to) { map.removeLayer(to); layersRef.current.to = null }
    if (user) { map.removeLayer(user); layersRef.current.user = null }
    if (circle) { map.removeLayer(circle); layersRef.current.circle = null }
```

```
 }

 async function onSubmit(e) {
   e.preventDefault()
   if (!from || !to) return
   const alertNum = Number(alertDistance)
   setAlertDisp(Number.isFinite(alertNum) ? `${alertNum} m` : '--')

   // get user location first
   const getPosition = () => new Promise((resolve, reject) => {
     if (!navigator.geolocation) return reject(new Error('Geolocation not supported'))
       navigator.geolocation.getCurrentPosition(resolve, reject, { enableHighAccuracy: true,
timeout: 10000 })
   })

   try {
     const pos = await getPosition()
     const userlat = pos.coords.latitude
     const userlong = pos.coords.longitude

     const res = await fetch('http://localhost:4000/api/getRoute', {
       method: 'POST',
       headers: { 'Content-Type': 'application/json' },
             body: JSON.stringify({ from, to, userlat, userlong, alertDistance:
Number.isFinite(alertNum) ? alertNum : undefined })
```

```
    })
    const result = await res.json()
  const { distanceMetersFormatted, durationSecondsFormatted, alertdis, routeCoordinates } =
result || {}

    setDistance(distanceMetersFormatted || '--')
    setDuration(durationSecondsFormatted || '--')
    setAlertDisp(alertdis || (Number.isFinite(alertNum) ? `${alertNum} m` : '--'))

    // Play sound only when backend indicates a positive alert
    if (typeof alertdis === 'string' && alertdis.toLowerCase().startsWith('alert')) {
      playAudio()
    }

          if (L && mapRef.current && Array.isArray(routeCoordinates) &&
routeCoordinates.length > 0) {
      const map = mapRef.current
      clearLayers()
      // route polyline
        const route = L.polyline(routeCoordinates, { color: '#22c55e', weight: 5, opacity: 0.9
}).addTo(map)
      const fromM = L.marker(routeCoordinates[0]).addTo(map).bindPopup(`From: ${from}`)
              const toM = L.marker(routeCoordinates[routeCoordinates.length -
1]).addTo(map).bindPopup(`To: ${to}`)
      const userM = L.circleMarker([userlat, userlong], {
        radius: 8,
        color: '#ffffff',
        weight: 2,
        fillColor: '#ef4444',
        fillOpacity: 0.9
      }).addTo(map).bindPopup('Your location')

      layersRef.current = { route, from: fromM, to: toM, user: userM, circle: null }

      const bounds = L.latLngBounds(routeCoordinates.concat([[userlat, userlong]]))
      map.fitBounds(bounds, { padding: [20, 20] })

      if (Number.isFinite(alertNum) && alertNum > 0) {
        const circle = L.circle(routeCoordinates[routeCoordinates.length - 1], {
          radius: alertNum,
          color: '#f59e0b',
          fillColor: '#f59e0b',
          fillOpacity: 0.15
        }).addTo(map)
        layersRef.current.circle = circle
      }
    }
  } catch (err) {
    console.error('Route request failed:', err)
  }
}

return (
  <div className="min-h-screen flex items-center justify-center bg-gray-900 px-4">
      <div className="w-full max-w-4xl mx-auto p-4 md:p-8 flex flex-col md:flex-row
md:space-x-8 space-y-8 md:space-y-0">
      {/* Left Card */}
          <div className="md:w-1/2 bg-gray-800 bg-opacity-50 rounded-2xl p-6 md:p-8
backdrop-filter backdrop-blur-lg border border-gray-700">
       <div className="text-center">
         <h1 className="text-3xl md:text-4xl font-bold text-white mb-6 md:mb-8">Welcome
to BuzzYatra!</h1>
         <form className="space-y-6" onSubmit={onSubmit}>
          <div>
                                <label className="block text-left text-white mb-2"
htmlFor="fromSelect">From:</label>
                    <select id="fromSelect" name="from" value={from} onChange={(e) =>
setFrom(e.target.value)}
                  className="w-full bg-gray-700 text-white rounded-lg p-3 appearance-none
focus:outline-none focus:ring-2 focus:ring-green-400">
              {stations.map((s) => (
                <option key={s.name} value={s.name}>{s.name}</option>
              ))}
            </select>
          </div>
            <div>
             <label className="block text-left text-white mb-2" htmlFor="toSelect">To:</label>
                             <select id="toSelect" name="to" value={to} onChange={(e) =>
setTo(e.target.value)}
                        className="w-full bg-gray-700 text-white rounded-lg p-3 appearance-none
focus:outline-none focus:ring-2 focus:ring-green-400">
               {stations.map((s) => (
                <option key={s.name} value={s.name}>{s.name}</option>
               ))}
             </select>
            </div>
            <div>
                      <label className="block text-left text-white mb-2" htmlFor="alertme">Alert
Metre:</label>
                   <input id="alertme" name="alertme" required placeholder="Enter the distance in
metres"
                   value={alertDistance} onChange={(e) => setAlertDistance(e.target.value)}
                        className="w-full bg-gray-700 text-white rounded-lg p-3 appearance-none
focus:outline-none focus:ring-2 focus:ring-green-400" />
            </div>
            <button type="submit"
              className="w-full bg-green-500 hover:bg-green-600 text-white font-bold py-3 px-
4 rounded-lg transition duration-300">
              Submit
            </button>
          </form>
```

```
     </div>
    </div>


    {/* Right Card */}
      <div className="md:w-1/2 bg-gray-800 bg-opacity-50 rounded-2xl p-6 md:p-8 backdrop-filter backdrop-blur-lg border border-gray-700 text-white">
      <h2 className="text-2xl md:text-3xl font-bold mb-6 md:mb-8 text-center">Route Details</h2>
     <div className="space-y-4">
      <div className="flex items-center justify-between">
       <span className="font-medium">From:</span>
       <div className="w-1/2 bg-gray-700 rounded-lg p-3">
        <span className="text-gray-300">{from || '--'}</span>
       </div>
      </div>
      <div className="flex items-center justify-between">
       <span className="font-medium">To:</span>
       <div className="w-1/2 bg-gray-700 rounded-lg p-3">
        <span className="text-gray-300">{to || '--'}</span>
       </div>
      </div>
      <div className="border-t border-gray-700 my-4 text-align=left"></div>
                              <p><span className="font-medium">Distance:</span> <span>{distance}</span></p>
                              <p><span        className="font-medium">Duration:</span>
<span>{duration}</span></p>

          <p><span className="font-medium">Alert:</span> <span className="text-yellow-
400">{alertDisp}</span></p>

         </div>

          <div ref={mapContainerRef} className="mt-6 rounded-lg overflow-hidden border
border-gray-700" style={{ height: 340 }}></div>

        </div>

      </div>

    </div>

  )

}


export default App
```

Backend code

```
const express = require('express');

const mongoose = require('mongoose');

const fetch = global.fetch;

const cors = require('cors');

require('dotenv').config();


const Station = require('./models/Station');


const app = express();

app.use(cors());
app.use(express.json());


// connect to MongoDB
mongoose.connect(process.env.MONGO_URI, {
dbName: 'BuzzYatra',
  useNewUrlParser: true,
  useUnifiedTopology: true
}).then(() => console.log('MongoDB connected'))
  .catch(err => console.error('MongoDB error:', err));


app.get("/", (req, res) => {
  res.send("Welcome to the BuzzYatra Backend");
});


/*Haversine Formula*/
function getDistance(lat1, lon1, lat2, lon2) {
  const R = 6371e3; // Earth radius in meters


  const phi1 = lat1 * Math.PI / 180;
  const phi2 = lat2 * Math.PI / 180;
  const deltaPhi = (lat2 - lat1) * Math.PI / 180;
  const deltaLambda = (lon2 - lon1) * Math.PI / 180;


  const a = Math.sin(deltaPhi / 2) * Math.sin(deltaPhi / 2) +
        Math.cos(phi1) * Math.cos(phi2) *
        Math.sin(deltaLambda / 2) * Math.sin(deltaLambda / 2);

  const c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1 - a));

  const d = R * c; // in meters
  return d;
}

/**
 * POST /api/getRoute
 * body: { from: "StationName", to: "StationName" }
 */
app.post('/api/getRoute', async (req, res) => {
  const { from, to, userlat, userlong, alertDistance } = req.body;
  if (!from || !to) {
    return res.status(400).json({ error: 'Provide both from and to' });
  }
  if (!userlat || !userlong) {
    return res.status(400).json({ error: 'Provide user latitude and longitude' });
  }

  try {
    // Find station docs in DB
    const fromStation = await Station.findOne({ name: from });
    const toStation = await Station.findOne({ name: to });
    if (!fromStation || !toStation) {
      return res.status(404).json({ error: 'Station not found in database' });
    }

    // Call OpenRouteService Directions
      console.log("Using coordinates: ", fromStation.lat, fromStation.long, toStation.lat, toStation.long);
    const apiKey = process.env.ORS_API_KEY;
      const url = `https://api.openrouteservice.org/v2/directions/driving-car?profile=driving-car&api_key=${apiKey}&start=${fromStation.long},${fromStation.lat}&end=${toStation.long},${toStation.lat}`;

    const response = await fetch(url);
    const data = await response.json();

    if (!data.features || data.features.length === 0) {
      return res.status(500).json({ error: 'No route found' });
    }

    const route = data.features[0];

    // Extract geometry + summary
    const coordinatesLngLat = route.geometry.coordinates; // [lng, lat]
    const distanceMeters = route.properties.summary.distance;
    const durationSeconds = route.properties.summary.duration;
    const durationSecondsFormatted = Math.round(durationSeconds / 60) + " min";
    const distanceMetersFormatted = "approx. " + Math.round(distanceMeters / 1000) + " km";
    // Convert to [lat, lng] for frontend consumption if needed
    const coordinatesLatLng = coordinatesLngLat.map(c => [c[1], c[0]]);
    const distance = getDistance(userlat, userlong, toStation.lat, toStation.long);
    const distanceFormatted = Math.round(distance / 1000) + " km";
    let alertdis;
    if (Number.isFinite(alertDistance) && distance < alertDistance) {
      alertdis = "Alert " + distance + " (" + distanceFormatted + ")";
    } else {
      alertdis = "No alert: " + distance + " (" + distanceFormatted + ")";
    }

    return res.json({
      routeCoordinates: coordinatesLatLng,
      distanceMetersFormatted,
      durationSecondsFormatted,
      alertdis
    });

   } catch (err) {
    console.error('Route error:', err);
```

```
    res.status(500).json({ error: 'Server error' });
  }
});

app.get('/api/getStation', async (req, res) => {
  try {
    // If you only need names, project only name field
    const stations = await Station.find({}, 'name');
    // To return only names, change to: Station.find({}, 'name')
    res.json(stations);
  } catch (err) {
    console.error('Error fetching stations:', err);
    res.status(500).json({ error: 'Internal server error' });
  }
});

const PORT = process.env.PORT || 4000;
app.listen(PORT, () => console.log(`Server running on port ${PORT}`));
```

6.4 IMPLEMENTATION SUMMARY

The implementation phase transformed the system design into a fully functional BuzzYatra web application. Using the MERN stack, the project achieved seamless integration between frontend, backend, and database layers. The use of OpenStreetMap and Geolocation APIs enabled real-time location tracking, while MongoDB efficiently managed user and alert data.

Through successful implementation and testing, the Stop Alert System and SOS Emergency System now operate reliably, fulfilling the primary goal of BuzzYatra to make public transport travel smarter, safer, and more accessible through a web-based solution.

CHAPTER 7

TESTING

Testing is one of the most critical stages of software development. It ensures that the developed system meets the specified requirements and performs as expected under different conditions. The purpose of testing is to detect and remove errors, verify system functionality, and validate that all modules work together seamlessly.

For the BuzzYatra web application, several testing techniques were used to ensure that both major modules the Stop Alert System and the SOS Emergency System functioned reliably, accurately, and securely.

7.1 OBJECTIVES OF TESTING

The main objectives of testing in the BuzzYatra project were:

- To verify that each module (Stop Alert and SOS) operates according to functional requirements.
- To ensure smooth interaction between the frontend (React) and backend (Node.js + Express.js).
- To validate real-time location tracking accuracy and alert triggering.
- To check that SOS alerts are correctly sent to registered emergency contacts.
- To ensure data integrity and security during user interaction.
- To confirm the overall system's stability, scalability, and usability before deployment.

7.2 TYPES OF TESTING CONDUCTED

Several levels of testing were performed to ensure the quality of the BuzzYatra web application:

7.2.1 Unit Testing

Unit testing was conducted on individual components and functions.

- Each React component (e.g., StopAlert.js, SOSButton.js) and backend route (e.g., /api/sos, /api/alert) was tested separately.
- JavaScript testing libraries and the Postman tool were used to validate API requests and responses.

- Example test cases included verifying correct distance calculation for alerts and successful retrieval of contact data from MongoDB.

7.2.2 Integration Testing

Integration testing ensured that the communication between frontend, backend, and database worked as intended.

- The flow from React → Express.js → MongoDB → React was tested end-to-end.
- APIs were tested for correct data exchange, response codes, and real-time update handling.
- Postman was used to test GET, POST, and PUT API requests for the Stop Alert and SOS modules.

7.2.3 System Testing

System testing evaluated the complete functionality of the application as a single integrated system.

- The application was tested in different browsers (Chrome, Firefox, Edge) and devices (desktop and mobile).
- Real-time GPS tracking was verified for accuracy using the Geolocation API.
- The Stop Alert trigger was tested under various distances to ensure correct notifications.

7.2.4 Performance Testing

Performance testing was conducted to measure system responsiveness and stability.

- The time taken to fetch location data and trigger alerts was recorded.
- The backend's response time for SOS requests was analyzed under normal and high load conditions.
- The system consistently responded within **2–3 seconds**, meeting performance expectations.

7.2.5 User Acceptance Testing (UAT)

UAT was conducted to ensure that the system met the needs of its end users — daily commuters and tourists.

- Selected users tested the web application for ease of navigation and functionality.
- Feedback was collected on the clarity of alerts, responsiveness of the SOS feature, and overall user experience.
- Based on feedback, minor UI improvements were implemented.

7.3 TEST PLAN

A structured test plan was followed to ensure systematic testing of all components. The plan covered test objectives, features to be tested, testing approach, and acceptance criteria.

Test Type	**Objective**	**Tools Used**	**Status**
Unit Testing	Validate each function and component	Postman	Completed
Integration Testing	Check frontend, backend, database connectivity	Postman	Completed
System Testing	Verify full workflow and system performance	Browser-based testing	Completed
Performance Testing	Measure response time and load handling	Manual & Automated	Completed
UAT	Validate usability and design	User feedback	Completed

Table 7.1 Test Plan

7.4 TEST CASES

Some representative test cases are shown below:

Test Case ID	**Test Description**	**Expected Result**	**Actual Result**	**Status**
TC01	Verify user registration with valid data	User registered successfully	Success	Pass
TC02	Verify location tracking using Geolocation API	Live coordinates displayed correctly	Success	Pass
TC03	Validate Stop Alert trigger before destination	Alert triggered one stop before arrival	Success	Pass
TC04	Test SOS alert functionality	SOS message sent to registered contacts	Success	Pass
TC05	Check MongoDB connection	Database connected and data stored	Success	Pass
TC06	Test invalid input handling	Proper error message displayed	Success	Pass
TC07	Verify API response for alert route	JSON data returned correctly	Success	Pass
TC08	Ensure UI responsiveness on mobile	Layout adjusts smoothly	Success	Pass

Table 7.2-Test Cases

All test cases passed successfully, indicating that the system is stable and performs reliably.

7.5 BUG TRACKING

During testing, minor issues such as delayed alert triggers, inaccurate GPS readings in weak signal areas, and unhandled API errors were identified.
These issues were logged, prioritized, and resolved by:

- Optimizing GPS polling intervals.
- Adding fallback error handling for location unavailability.

- Refining backend validation and API response structures.

All critical bugs were resolved before deployment.

7.6 VALIDATION

Validation confirmed that the system meets all the functional and performance requirements defined during the Requirement Analysis phase. The Stop Alert System accurately triggers notifications based on GPS data, and the SOS Emergency System reliably sends alerts to emergency contacts. The results validated the project's core objective of enhancing the commuter's travel safety and convenience.

7.7 TEST RESULTS SUMMARY

The testing results showed that all system functions were executed successfully without critical failures.

- The Stop Alert System achieved over 95% accuracy in destination proximity detection.
- The SOS Emergency System sent alerts with an average response time of 2 seconds.
- The application remained stable during extended use and multiple user sessions.

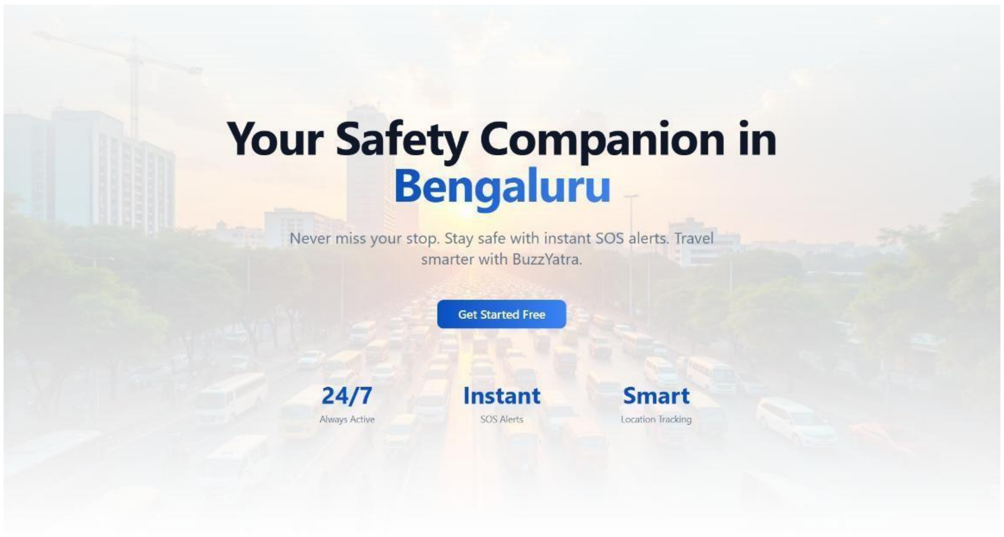
These outcomes demonstrate that the system is robust, efficient, and ready for real-world deployment.

CHAPTER 8

SAMPLE OUTPUTS

8.1 SNAP SHOTS

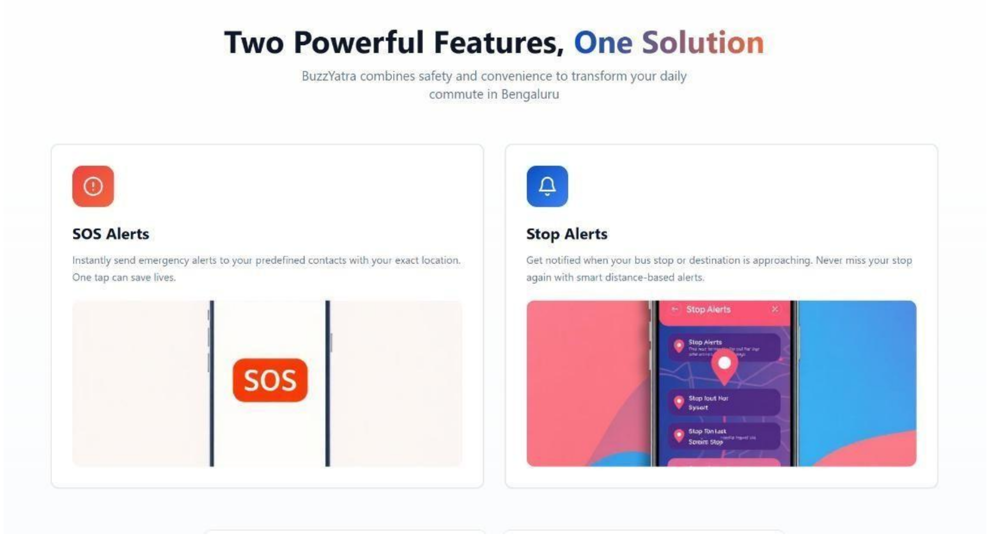
1. Landing page:



Your Safety Companion in Bengaluru

Never miss your stop. Stay safe with instant SOS alerts. Travel smarter with BuzzYatra.

Fig :8.1: Landing page 1



Two Powerful Features, One Solution

BuzzYatra combines safety and convenience to transform your daily commute in Bengaluru

SOS Alerts

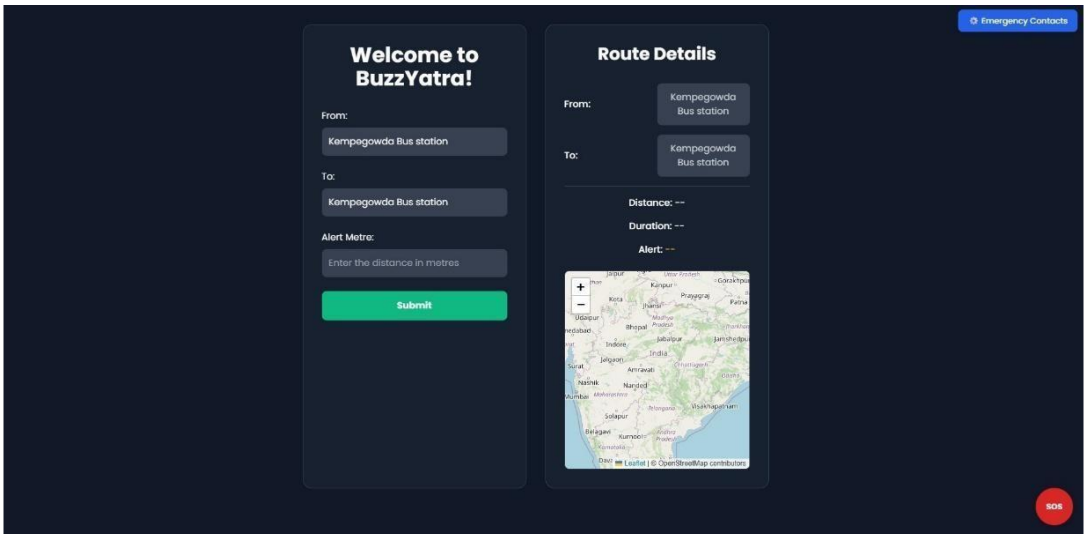
Instantly send emergency alerts to your predefined contacts with your exact location. One tap can save lives.

Stop Alerts

Get notified when your bus stop or destination is approaching. Never miss your stop again with smart distance-based alerts.

Fig:8.2: Landing page 2

2. Home page:



Welcome to BuzzYatra!

From: Kempegowda Bus station

To: Kempegowda Bus station

Alert Metre: Enter the distance in metres

Submit

Route Details

From: Kempegowda Bus station

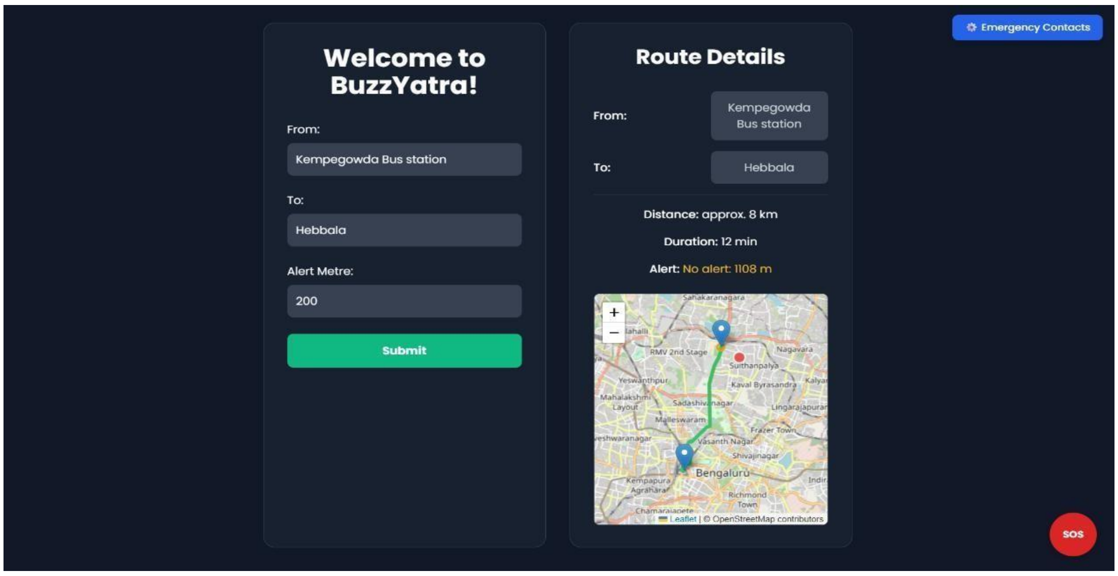
To: Kempegowda Bus station

Distance: --

Duration: --

Alert: --

Fig: 8.3: Starting home page



Welcome to BuzzYatra!

From: Kempegowda Bus station

To: Hebbala

Alert Metre: 200

Submit

Route Details

From: Kempegowda Bus station

To: Hebbala

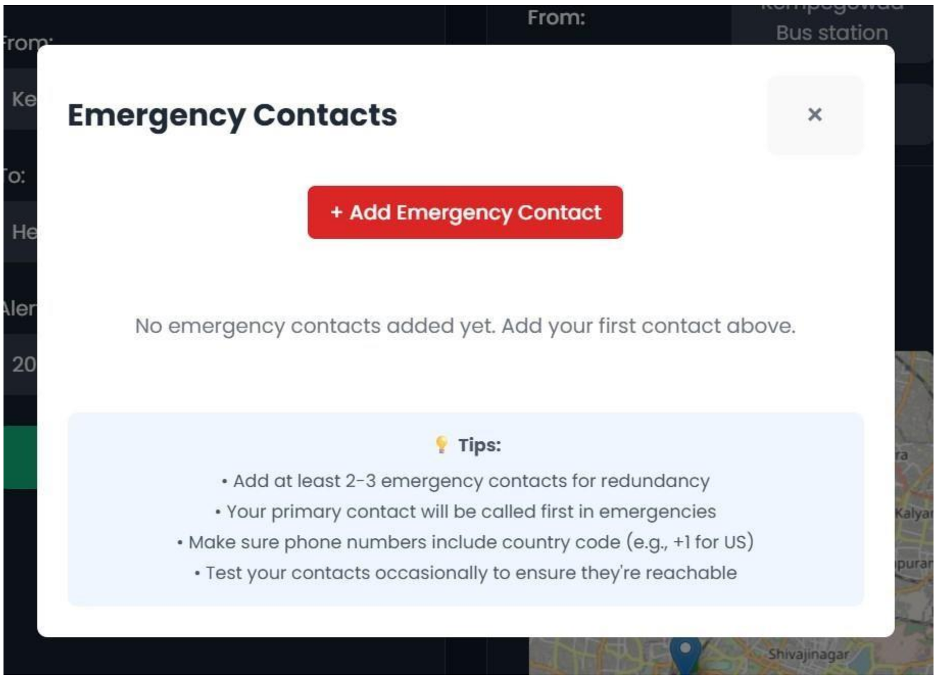
Distance: approx. 8 km

Duration: 12 min

Alert: No alert: 1108 m

Fig: 8.4: After fetching the co-ordinates

3. SOS feature:



Emergency Contacts

×

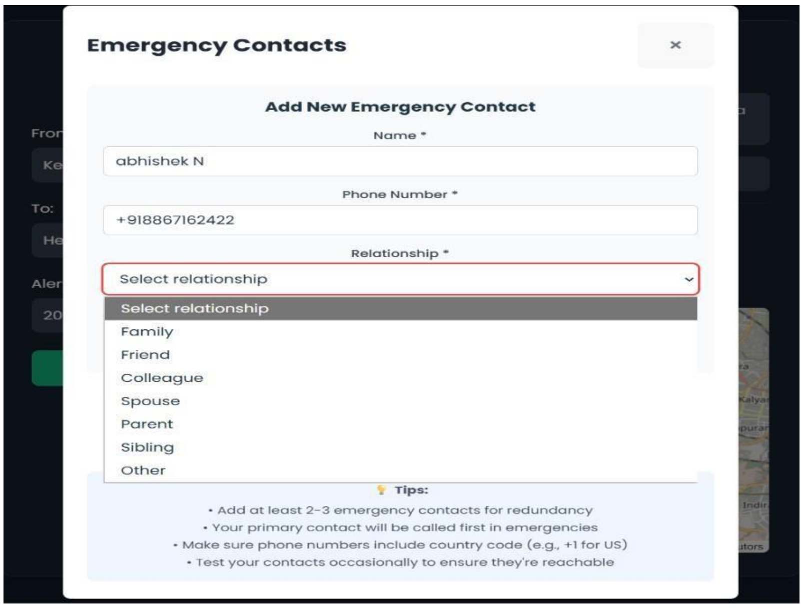
+ Add Emergency Contact

No emergency contacts added yet. Add your first contact above.

💡 Tips:

• Add at least 2-3 emergency contacts for redundancy
• Your primary contact will be called first in emergencies
• Make sure phone numbers include country code (e.g., +1 for US)
• Test your contacts occasionally to ensure they're reachable

Fig: 8.5: Emergency contact window



Emergency Contacts

×

Add New Emergency Contact

Name *: abhishek N

Phone Number *: +918867162422

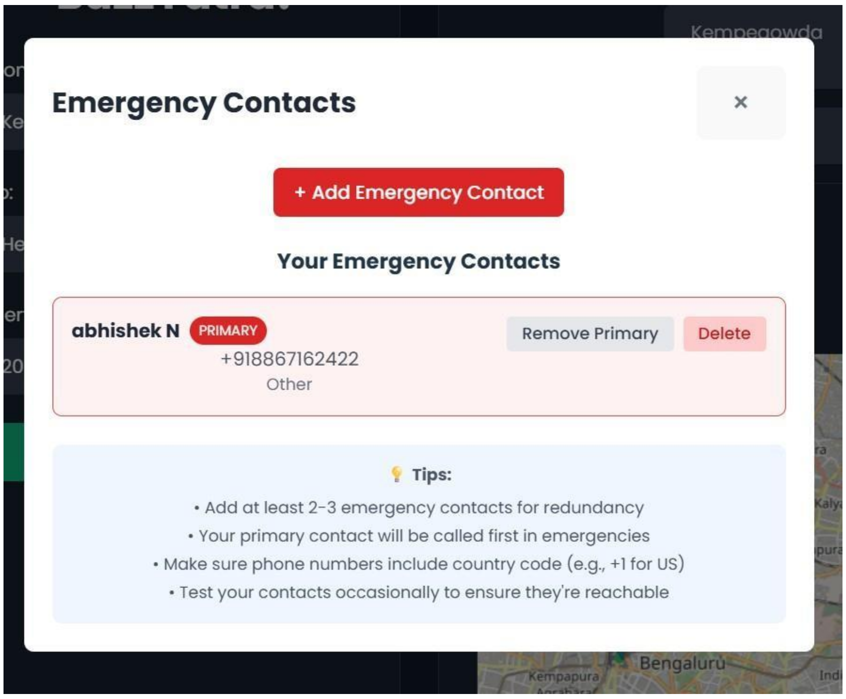
Relationship *: Select relationship

- Select relationship
- Family
- Friend
- Colleague
- Spouse
- Parent
- Sibling
- Other

💡 Tips:

• Add at least 2-3 emergency contacts for redundancy
• Your primary contact will be called first in emergencies
• Make sure phone numbers include country code (e.g., +1 for US)
• Test your contacts occasionally to ensure they're reachable

Fig: 8.6: Adding emergency contact



Emergency Contacts

×

+ Add Emergency Contact

Your Emergency Contacts

abhishek N PRIMARY
+918867162422
Other

Remove Primary Delete

💡 Tips:

• Add at least 2-3 emergency contacts for redundancy
• Your primary contact will be called first in emergencies
• Make sure phone numbers include country code (e.g., +1 for US)
• Test your contacts occasionally to ensure they're reachable

Fig: 8.7: After adding emergency contact window

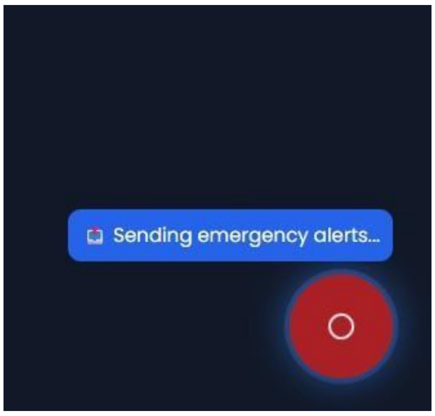


Fig: 8.8: Sending emergency alerts

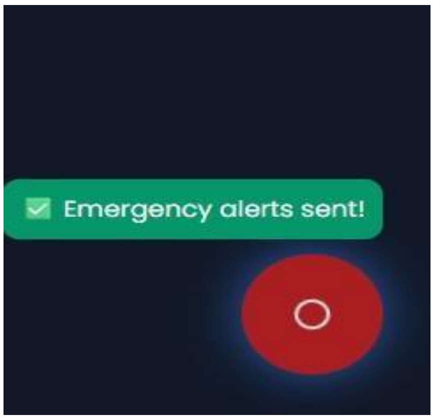
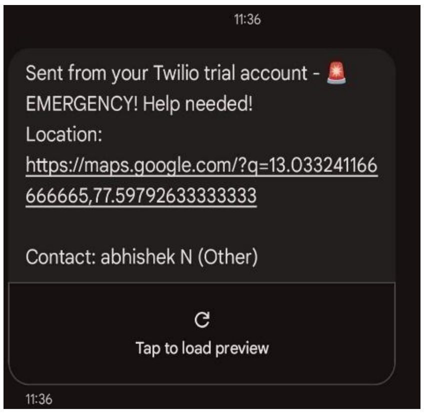


Fig: 8.9: Emergency alerts sent



11:36

Sent from your Twilio trial account - 🚨 EMERGENCY! Help needed!
Location:
https://maps.google.com/?q=13.033241166666665,77.59792633333333

Contact: abhishek N (Other)

Tap to load preview

11:36

Fig: 8.10: Emergency alerts received

CONCLUSION

The BuzzYatra web application was successfully developed to provide an efficient and accessible solution for bus commuters. It focuses on two main features the Stop Alert System and the SOS Emergency System which together enhance travel convenience and passenger safety. The application runs directly on a browser, making it lightweight, user-friendly, and reliable for daily use. Overall, the project achieves its goal of improving the travel experience through the effective use of web technologies.

FUTURE ENHANCEMENTS

The current version of BuzzYatra can be further enhanced in several ways to expand its usability and reach:

- **Progressive Web App (PWA) Support:**

The system can be upgraded into a Progressive Web Application, enabling offline access, push notifications, and installable behavior across devices for a more app-like experience.

- **Integration with Popular Apps:**

The BuzzYatra module can be integrated with well-known mobility platforms such as Uber, Ola, and Rapido, allowing users to access stop alerts and emergency support directly from within those applications.

These enhancements would transform BuzzYatra from a standalone system into a scalable and collaborative smart transport utility, contributing to safer and smarter commuting experiences in urban environments.

REFERENCES

1. Prof Fadhil, Pro Ibrahim. “Mobile application to locate and track users whereabouts and send SOS messages” **2019**

2. Snehal A.Ghodake, Prof.Megha pallewr . “Easy navigation of bus alert system” **2020**

3. Jayaraam S.P, Jayakumar D. “Bus tracking system using mobile GPS technology” **2024**

4. Mary A, Albert J. “Campus Commute: An innovative approach to college bus tracking and management” **2025**

5. Sandhiya B, Ravichandra. “Real time college bus tracking and notification system for student convenience” **2025**

6. R. Kumar ,S. Agarwal, “A Study on Smart Public Transportation System using IoT” **2019**

7. Rashvand, N., Hosseini, S. S., Azarbayjani, M., & Tabkhi, H . “Real-Time Bus Departure Prediction Using Neural Networks for Smart IoT Public Bus Transit” **2024**

8. Rashvand N, Hosseini, Azarbayjani M, Tabkhi H. “Real-Time Bus Arrival Prediction” **2023**

ONLINE REFERENCES

1. NAMMA BMTC : https://nammabmtcapp.karnataka.gov.in

2. MOOVIT : https://moovitapp.com

3. TUMMOC : https://tummoc.com

4. CHALO : https://chalo.com