FACULTY OF SCIENCE, ENGINEERING AND COMPUTING

School of *Computer Science & Mathematics*

BSc DEGREE
IN
Computer Science (Software Engineering)

PROJECT FINAL REPORT

Name: Fathima Ranna

ID Number: E187002

Project Title: Mobitix Digital Bus Ticket Booking App

Project Type: Build

Date:

Supervisor: Ms. Maleesha Ekanayake

Kingston University London

Did you discuss and agree the viability of your project idea with your supervisor?	Yes
Did you submit a draft of your proposal to your supervisor?	Yes
Did you receive feedback from your supervisor on any submitted draft?	Yes

Abstract

The current bus ticket reservation systems in Sri Lanka have to deal with issues related to lack of real-time tracking, poor refund management, overcrowded buses, and time-consuming manual ticketing processes, which ultimately lead to dissatisfaction among passengers. Therefore, Mobitix, a Digital Bus Ticket Reservation mobile app, will be designed to resolve these issues. As such, Mobitix will transform public transportation ticketing by offering passengers and administrators an easy and efficient way to manage bus ticketing procedures. It provides the advantage of purchasing and reserving tickets, managing user profiles, dynamically choosing seats, and making secure online payments. It also maintains real-time tracking of the bus using GPS and Google Maps to keep passengers updated with the latest travel information. Moreover, the system provides trip history tracking and refund management. Mobitix will also provide an administrator panel that can be used to manage bus schedules, reservations, and performance analytics. Major features offered include generating tickets and real-time integration with bus schedules, improving the user experience and operational efficiency. Accordingly, Mobitix aims to simplify the bus reservation process of Sri Lanka's public transportation system in this manner.

To achieve these goals, Mobitix leverages modern technologies such as Flutter for cross-platform mobile development, PHP for backend services, and MySQL for database management. The integration of Google Maps API ensures accurate real-time bus tracking, while local payment gateways like Genie, Koko, and eZcash facilitate secure and convenient transactions. By addressing the inefficiencies of traditional ticketing systems, Mobitix not only enhances the passenger experience but also provides bus operators with tools to optimize scheduling, reduce operational costs, and improve overall service quality. This digital transformation aligns with global trends toward sustainable and efficient public transportation, making Mobitix a forward-thinking solution for Sri Lanka's evolving transport needs.

Contents

List of Figures

List of Tables

Glossary of Terms

- API - Application Programming Interface: A set of protocols and tools for building software applications.
- GDPR - General Data Protection Regulation: A regulation in EU law on data protection and privacy.
- GPS - Global Positioning System: A satellite-based navigation system that provides location and time information.
- MFA - Multi-Factor Authentication: A security system that requires multiple methods of authentication.
- OTP - One-Time Password: A dynamically generated code used for secure authentication.
- RBAC - Role-Based Access Control: A method of restricting system access to authorized users based on their roles.
- RFID - Radio-Frequency Identification: A technology that uses electromagnetic fields to automatically identify and track tags attached to objects.
- SSL/TLS - Secure Sockets Layer/Transport Layer Security: Cryptographic protocols for secure communication over a network.
- UI/UX - User Interface/User Experience: The design and usability aspects of a software application.
- RESTful API - Representational State Transfer API: An architectural style for designing networked applications.
- SQL - Structured Query Language: A programming language used to manage and manipulate databases.
- NoSQL - Not Only SQL: A database design that provides a mechanism for storage and retrieval of unstructured data.
- FCM - Firebase Cloud Messaging: A cross-platform messaging solution for delivering notifications.
- PHP - Hypertext Preprocessor: A server-side scripting language used for web development.
- MySQL - A popular open-source relational database management system.
- Flutter - A UI toolkit by Google for building natively compiled applications for mobile, web, and desktop from a single codebase.
- Dart - A programming language developed by Google, used with Flutter for building mobile and web apps.
- SWOT - Strengths, Weaknesses, Opportunities, Threats: A strategic planning tool.
- PEST - Political, Economic, Social, Technological: A framework for analyzing macro-environmental factors.
- KPI - Key Performance Indicator: A measurable value that demonstrates how effectively a company is achieving key objectives.
- CCPA - California Consumer Privacy Act: A state statute intended to enhance privacy rights for California residents.
- QR Code - Quick Response Code: A type of matrix barcode used to store information.
- NFC - Near Field Communication: A short-range wireless technology for data exchange between devices.

1. Introduction & Literature Review

1.1 Introduction

Modern transportation is rapidly changing to digital modes, where technology will be the key to efficiency and ease. Mobitix, a mobile-based bus ticket reservation system which is designed especially for Sri Lanka, is an important solution in this area. The goal of this app is to help commuters in avoiding overcrowded buses, the lack of seats, and the inability to purchase or refund tickets on a real-time basis. Customers will be able to search for buses, choose seats, and pay online with Mobitix's digital platform, which will save precious time and provide a better travel experience.

The need for a solution like Mobitix is evident in Sri Lanka's busy public transport sector, where nearly 50% of the people depend on buses for daily commutes and intercity travel, according to. (The Department of Census and Statistics Sri Lanka, 2022). Traditional ticketing systems are mostly in a manual and paper-based format, which are considerably prone to inefficiencies and errors. Therefore, Mobitix wishes to fill this gap in service with features such as dynamic seat selection, secure online payment options, and digital tickets to meet the needs of both urban and rural commuters. The system integrates modern technology to make it accessible to anyone with a smart device, and it's user-friendliness would cater to a growing audience's demands.

1.2 Background and Motivation

Public transportation is part of the economic development process in any country, and Sri Lanka is no exception. The major issues concerning the bus transport sector in Sri Lanka are overcrowding, unmanaged arrival times, and inefficiency in manual ticketing systems. Many passengers are forced to waste much time in buses, especially during peak hours. Moreover, the absence of automation in ticketing develops human errors, and not having real-time information for passengers leads to problems in managing travel schedules.

Before the emergence of modern technologies, reservation procedures in buses were carried out manually, where several problems were encountered, such as wastage of time and frauds related to this. (Jayasuriya, 2021) discusses a few of them where human errors occurred in most cases with a manual system and was unable to meet the growing demand. The introduction of an automated system, such as that involving prepayment mechanisms, has been proposed as part of the solution to these issues. But the challenge remains in finding a solution that is both accessible and effective, especially in a country like Sri Lanka, where not all passengers are comfortable

with using smart devices.

The importance of developing a completed automated bus ticket reservation and tracking app that includes real-time tracking, online payment systems, and user-friendly interfaces is emphasized by recent research, such as those conducted by (Vimukthi, 2023), Vimukthi discusses about how real-time tracking and GPS integration would make it possible to monitor the whereabouts of the buses and make travel decisions. This strategy may shorten wait times, improve user satisfaction greatly, and improve bus timetable administration.

While many web-based bus reservation systems have been proposed or are in use, these often fail to provide a complete solution. (Haleem, 2016) discusses the use of RFID and microcontroller-based systems for bus tracking, but web-based systems often lack essential features such as refund management, real-time updates and dynamic ticketing options. For example, when a passenger needs to cancel or modify a ticket, web-based systems frequently fail to offer a proper refund or adjustment process, leaving users frustrated. Moreover, many web systems provide incomplete or outdated information, particularly when it comes to real-time bus tracking. These drawbacks lead to dissatisfaction among the users and further hinder the potential of the system in serving passengers efficiently.

Considering the above-mentioned challenges, the mobile application was decided upon instead of the web-based system. The accessibility of a mobile app is higher compared to a web-based system, especially in a country like Sri Lanka, where smartphones are more in use among people than desktop computers. They also offer other advantages, like GPS-based location tracking, and offline functionality, which is quite necessary for a real-time and user- friendly experience. Also, the mobile application makes it easy to integrate the payment system, dynamic selection of seats, and on-screen instant generation of tickets, which are important features that would be difficult to implement in a traditional web-based platform.

This project will solve key issues in the present bus reservation system by developing a mobile application using Flutter at the frontend, PHP and sqflite at the backend, and GPS technology for real-time tracking. The application will have features like ticket booking, seat selection. Refund, trip history, and real-time tracking of the bus, thus maintaining all the travel details of the user directly from their smartphone.

1.3 Problem in brief

The main problem addressed by this project is the inefficiency of current bus ticket reservation systems in Sri Lanka, like refund management, seat selection, and real-time tracking. These issues lead to overcrowded busses, inefficient manual ticketing processes, insufficient management of travel schedules, and ultimately, passenger dissatisfaction.

Existing web-based platforms often fail to address these needs, such as offering complete, user-friendly platforms for online bookings, cancellations, and refunds. Additionally, many commuters in Sri Lanka rely on mobile devices rather than desktop systems, making accessibility a considerable challenge for web-based systems.

Therefore, there is a pressing need for a user-friendly solution to improve ticket booking, travel convenience and the overall efficiency of the public transportation system.

1.4 Aim & Objectives

1.4.1 Aim

To improve the efficiency of Sri Lanka's bus ticket reservation and management, addressing the issues of overcrowding, manual errors , and lack of real-time updates, for a better public transportation experience among passengers

1.4.2 Objectives.

- Conduct a critical review of the risks and inefficiencies in existing bus reservation systems
- Study the existing technologies and methods to address these challenges in detail
- Design and develop a mobile application that integrates real-time tracking, ticket booking, refund management, and smooth payment integration.
- Perform thorough testing to measure the performance and user satisfaction of the developed
- system by analyzing feedback.
- Prepare final documentation, such as technical and user documentation, to support the
- implementation and usability of the system.

1.5 Scope

The scope of Mobitix project is carefully defined to make sure that it addresses key challenges in Sri Lanka’s bus ticketing system while remaining feasible within the constraints of time, budget and resources. By focusing on the key functionalities and strengths like real-time tracking and

secure payments, this app aims to deliver a reliable and user-friendly mobile application for public transportation. As such the scope can be further elaborated as follows.

Features for Customers

- Ticket Booking - Users can search for buses, select seats dynamically, and book tickets online.

- Real-Time Bus Tracking - Integration with Google Maps API to provide real-time bus location and estimated arrival times.

- Secure Online Payments - Support for local payment gateways like Genie, Koko, and eZcash for easy transactions.

- Refund Management - Users can request refunds for canceled tickets, with a transparent refund process.

- Trip History - Users can view their past trips and booking details.

- User Profiles - Users can create and manage profiles, including personal information and payment methods.

Administrator Panel

- Bus Schedule Management - Admins can add, update, or delete bus schedules.

- Reservation Management - Admins can view and manage ticket reservations.

- Performance Analytics - Admins can access data on bus performance, booking trends, and user feedback.

Technical Implementation:

- Frontend Development: The app will be developed using Flutter for cross-platform compatibility (Android and iOS).

- Backend Development: The backend will use PHP for RESTful API development and MySQL for database management.

- Integration with Third-Party Services: Integration with Google Maps API for real-time tracking and local payment gateways for secure transactions.

Justification for the Scope

The included features directly address the key issues in Sri Lanka's bus ticketing system, such as overcrowding, lack of real-time updates, and inefficient refund management. These features are essential to achieving the project's aim of improving public transportation efficiency and user satisfaction. The scope is designed to be achievable within the project timeline and available resources. Advanced features such as multi-language support and integration with other transport modes are excluded to ensure timely delivery of the core functionalities.
The scope aligns with the expectations of key stakeholders, including passengers, bus operators, and administrators. Features like real-time tracking and secure payments are prioritized based on stakeholder feedback and market research.

Key Assumptions

- It is assumed that smartphone penetration in Sri Lanka will continue to grow, making the app accessible to a larger user base.
- Local payment gateways like Genie, Koko, and eZcash will remain stable and widely used.
- The app will comply with data protection laws like GDPR and local regulations in Sri Lanka regarding user data privacy and refund policies.
- Third-party services like Google Maps API and payment gateways will function reliably without significant downtime or API changes.

Excluded from Scope

1. Integration with Other Transport Modes:
   The app will focus solely on bus ticketing and will not include other modes of transport like trains or flights.
2. Offline Functionality:
   While some features may work offline (e.g., viewing booked tickets), core functionalities like real-time tracking and online payments require an internet connection.
3. Custom Hardware Integration:
   The app will not involve the development or integration of custom hardware (e.g., RFID scanners or ticketing kiosks).
4. Advanced Data Analytics:
   While basic performance analytics will be provided, advanced data analytics or machine learning-based predictions are out of scope for this project.
5. Multi-Language Support:
6. The app will initially support English, with potential expansion to Tamil and Sinhala in future updates.

Constraints

- The project must be completed within the academic timeline, limiting the scope to essential features.
- The project has limited funding, so expensive third-party services or custom hardware integrations are excluded
- The development team consists of a single student, which limits the complexity and scale of the project.

Validation with Stakeholders

- The scope has been discussed and agreed upon with the project head-supervisor, Mr. Vikum, ensuring alignment with academic and stakeholder expectations.
- Feedback from potential users (eg: commuters and bus operators) has been considered to prioritize features such as real-time tracking and secure payments.

Flexibility for Adjustments

- A structured change management process will be in place to handle any necessary adjustments to the scope. Changes will be documented and approved by the supervisor before implementation.
- The Agile methodology will allow for iterative developments. Therefore, changes can be made based on feedback and testing results.

1.5.1 SWOT ANALYSIS

Table 1 SWOT Analysis

• STRENGTHS	• WEAKNESSES
• User-Friendly interface (Flutter, dynamic seat selection, real-time tracking) • Real-Time Tracking (Google Maps API for accurate bus locations) • Secure Payments (Local gateways like Genie, Koko, eZcash). • Cross-platform (Works on Android & iOS) • Admin Panel (Efficient bus schedule and reservation management).	• Dependency on Third-party APIs (Google Maps, payment gateways). • Limited Offline Functionality (Internet required for key features). • Complex Integration (Challenges with local bus schedules and payment systems). • High development complexity (Dynamic seat selection, refund management)
• OPPORTUNITIES	**• THREATS**
• Growing Smartphone usage (Large user base in Sri Lanka) • Government/ private sector support (Collaboration for public transport efficiency) • Expansion to other transportation modes (Trains, intercity buses) • Data Analytics (Improve services using user and bus performance data). • Sustainability (Reduce traffic congestion, promote eco-friendly transport)	• Competition (Existing bus ticketing systems) • Resistance to change (Rural users may prefer traditional methods). • Cybersecurity Risks (Sensitive user data handing). • Regulatory challenges (Compliance with GDPR and local laws) • GPS Accuracy issues (Poor signal or environmental factors)

1.5.2 PEST ANALYSIS

Table 2 PEST Analysis

Political Factors	• **Government Support : Potential collaboration with Sri Lankan government to improve public transport efficiency.** • **Need to adhere to data protection laws (eg: GDPR) and local regulations.** • **Government initiatives to modernize public transport could benefit the app.**

Economic Factors	• Growing Smartphone penetration : Increasing smartphones usage in Sri Lanka provides a larger user base • Local Payment Gateways : integration with local payment systems (eg: Genie, Koko, eZcash) support economic accessibility • Cost of Development : Budget constraints may limit advanced features or scalability.
Social Factors	• User adoption : Resistance to digital systems in rural areas may slow adoption. • Accessibility : The app must cater to diverse users, including those with disabilities. • Sustainability : Promotes eco-friendly public transport, aligning with societal trends toward sustainability.
Technological Factors	• Real-time tracking : Integration with Google Maps API for accurate bus tracking. • Cross-platform Development : Flutter ensures compatibility with both Android and iOS • Cybersecurity risks : Handling sensitive user data requires strong encryption and security measures.

1.5.3 Project Timeline

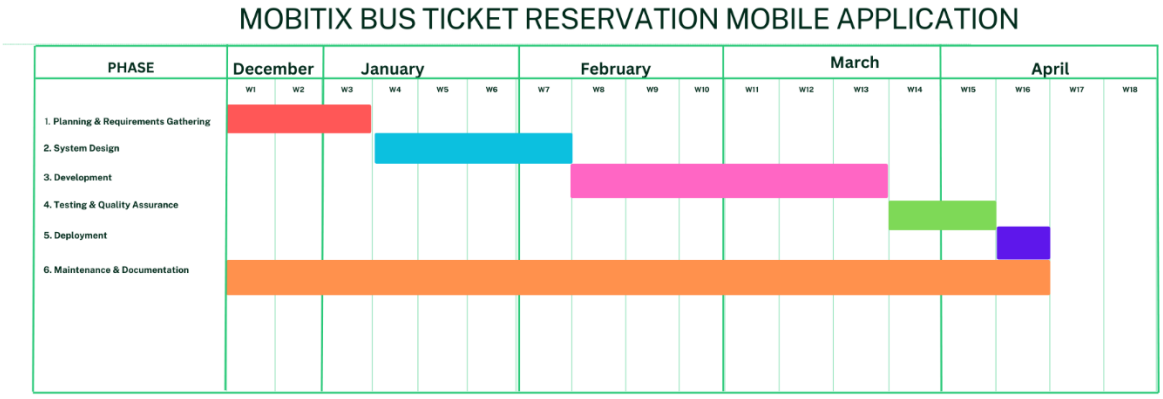


Figure 1 Gantt Chart

1.5.4 Key Stakeholders

The success of the Mobitix Digital Bus Ticket Booking App depends on the involvement and collaboration of several key stakeholders. These stakeholders play important roles in the development, implementation, and adoption of the app. The key stakeholders can be denoted as follows;

1. Passengers (End Users)

Passengers are the primary users of the app. They rely on the app to book tickets, track buses in real-time and manage their travel schedules. Passengers seek a convenient, reliable, and user-friendly platform that simplifies the bus ticketing process and enhances their overall travel experience. Their feedback and adoption rates are important for the app's success and continuous improvement.

2. Bus operators and Administrators.

Bus operators and administrators manage bus schedules, reservations, and performance analytics through the app's admin panel. They require an efficient system to conduct operations, reduce manual errors, and improve bus utilization. Their cooperation is essential for integrating the app with existing bus systems and ensuring accurate real-time data.

3. Government and Regulatory Bodies

Government agencies and regulatory bodies oversee public transportation systems to ensure compliance with local laws and regulations. They aim to modernize public transport, reduce traffic congestion, and promote sustainable travel options. Their support can help the app's adoption and integration into the broader public transport infrastructure.

4. Payment Gateway Providers

Providers like Genie, Koko, eZcash provide secure online payments within the app. They seek to expand their user base and provide seamless integration with the app. Their reliability and performance directly affect the app's payment functionality and user trust.

5. The Developer.

The development team (the author, in this case) is responsible for designing, building, and maintaining the app. The developer should aim to deliver a high-quality, functional app within the project timeline and budget. The developer's expertise and efficiency will determine the app's technical performance and ability to meet user needs.

6. Supervisors and Academic Advisors

Supervisors and academic advisors provide guidance, feedback, and oversight throughout the project. They ensure the project aligns with academic standards and achieves its objectives. Their input helps refine the app's design and functionality, ensuring it meets both user and academic requirements.

1.5.5 Risk Management

Table 3 Risk Management

Risk	Mitigation method
Delay in development due to unforeseen issues	• Extend sprint duration or reduce non-critical features to prioritize delivery • Conduct regular risk assessments during sprint planning to identify potential delays early. • Include buffer time in the project schedule to accommodate unexpected delays.
Payment gateway Integration challenges	• Switch to a reliable alternative gateway or provide a manual payment option • Test payment gateway integrations early in the development phase to identify and resolve issues. • Integrate multiple payment gateways as backups to ensure uninterrupted service.
GPS/ Google Maps API integration issues	• Use alternative mapping services or temporarily disable GPS feature • Continuously monitor API performance and availability.

Performance issues during testing	• Improve code and conduct performance testing on multiple devices • Optimize code and database queries for better performance.
Data Security or privacy concerns	• Implement encryption, secure storage. And follow data protection best practices • Conduct regular security audits and vulnerability assessments. • Anonymize user data used for testing to protect privacy.
Inaccurate GPS real-time tracking	• Use alternative APIs or use data requests to reduce errors • Implement data validation to ensure GPS data accuracy. • Allow users to report inaccurate tracking for quick resolution.
User dissatisfaction with UI/UX design	• Conduct feedback sessions to refine the user interface • Conduct usability testing with real users during the design phase.
Unexpected bugs in seat selection and reservation	• Increase QA focus on code improvements • Conduct regular code reviews to catch bugs early.
Insufficient time for deployment	• Plan app store submission in earlier sprints and address compliance requirements early • Create a checklist to ensure all app store requirements are met.
Integration issues with the Bus Schedule data	Use dummy datasets for testing purposes

1.6 Deliverables

Upon the completion of the Mobitix Digital Bus Ticket Booking App project, the following deliverables will be provided.

Technical deliverables

- System Architecture – Detailed documentation explaining the app's structure, including frontend (Flutter), backend (PHP), and database (MySQL) components
- Database Schema – a well-defined schema for storing user profiles, ticket bookings, and trip history.

- API integration Guides – Documentation for integrating Google Maps API and local payment gateways (Genie, Koko, eZcash).
- User Manuals – Guides for both passengers and administrators to operate the app effectively.

Functional Deliverables

- Fully Functional Mobile App – A cross-platform app (Android and iOS) with features like ticket booking, real-time bus tracking, secure payments, refund management, and trip history.
- Admin Panel – a backend system for managing bus schedules, reservations, and performance analytics.
- Testing reports – comprehensive test cases, bug fixes, and user acceptance testing (UAT) results.

1.7 Literature Review

The evolution of bus ticket reservation systems from manual to digital platforms has significantly transformed public transportation globally, particularly in developing regions like Sri Lanka. Traditional manual ticketing systems have long been plagued by inefficiencies, including long queues, human errors, and limited accessibility. (Jayasuriya, 2021) highlights that manual processes in Sri Lanka led to overcrowding, poor refund management, and passenger dissatisfaction, underscoring the need for automation. Similarly, (Oloyede, 2014) note that offline systems in Nigeria restricted customers' ability to compare bus operators or routes, while operators struggled with seat inventory management. Early web-based systems introduced functionalities like online seat selection, payment integration, and report generation, but they often lacked real-time updates and dynamic features such as refund management (Vimukthi, 2023). Mobile applications, such as the proposed Mobitix app, have emerged as a superior alternative due to their accessibility, GPS integration, and offline capabilities. For instance, GPS-based tracking systems in Malaysia, like Katsana and Debezt, enabled real-time bus location

updates and automated reporting, significantly improving operational efficiency (Jakubauskas, March 2014)

Secure user authentication is a critical component of digital ticketing systems, ensuring the protection of sensitive data. Systems like Mobitix employ multi-factor authentication (MFA) and role-based access control (RBAC) to prevent unauthorized access. Registration typically requires email or phone verification, which ensures accountability and reduces fraud (Jeewanthi Fernando, September 2016) Dynamic seat selection is another essential feature, allowing users to view real-time seat availability through interactive layouts. For example, BusSeat.lk in Sri Lanka uses color-coded seat maps to indicate booked, available, and reserved seats, reducing overbooking and enhancing transparency (Techkitez, 2015). Payment integration is also crucial, with local payment gateways like Genie, Koko, and eZcash being prioritized in regions like Sri Lanka to align with user preferences and ensure trust. Secure encryption protocols, such as SSL/TLS, protect transaction data, and failed payments trigger instant notifications, allowing users to retry or cancel bookings (Dr. Swapna Ubale, 2024)

Real-time tracking and notifications are vital for improving the user experience. Integration with Google Maps API enables live bus tracking, estimated arrival times, and route optimization, as seen in the Mobitix project (Vimukthi, 2023). Notifications via Firebase Cloud Messaging (FCM) alert users about delays, departures, and refund statuses, further enhancing the system's efficiency (Ubale et al., 2024). In Iraq, mobile bus ticketing systems (MBTS) reduced passenger wait times by 40% through real-time updates, demonstrating the potential impact of such features Refund management is another critical aspect, with automated refund policies streamlining dispute resolution. Admins review requests via dashboards, and approved refunds are processed through integrated gateways. However, inconsistent internet connectivity in rural areas remains a barrier, as highlighted by Jayasuriya (2021).

The technological frameworks and tools used in digital ticketing systems are crucial for their success. Cross-platform frameworks like Flutter dominate mobile app development due to their compatibility with Android and iOS, making them an ideal choice for Mobitix. Flutter’s prebuilt widgets and hot-reload features accelerate UI development, ensuring responsive interfaces (Ubale et al., 2024). PHP and MySQL are widely adopted for backend development due to their scalability and cost-effectiveness. RESTful APIs facilitate communication between frontend and

backend modules, while MySQL manages structured data like user profiles and ticket histories (Techkitez, 2015). Third-party integrations, such as Google Maps API for GPS tracking and local payment gateways like Genie, ensure localized transaction support and enhance operational efficiency (Vimukthi, 2023). Firebase handles notifications and user analytics, further improving the system's functionality (Ubale et al., 2024).

Despite the advantages, digital ticketing systems face several challenges. Dependency on third-party APIs, such as Google Maps, poses risks like service interruptions. Mitigation strategies include backup mapping services and manual location updates (Fernando, et al, 2016.). Cybersecurity risks, including data breaches and fraud, are mitigated through encryption, GDPR compliance, and regular security audits. Anonymizing test data further protects user privacy (Vimukthi, 2023). Resistance to digital adoption, particularly in rural areas, is another challenge. Mobitix addresses this through user education campaigns and incentives like discounted first (Fernando, et al, 2016.).Infrastructure limitations, such as poor internet connectivity in rural areas, restrict real-time features. Offline functionalities, such as cached ticket viewing, partially address this gap (Jayasuriya, 2021).

Case studies from various regions demonstrate the scalability and impact of digital ticketing systems. BusSeat.lk, launched in 2015, digitized Sri Lanka’s bus network, serving over 100,000 users. Key features include route maps, seat selection, and multi-operator integration. User feedback highlighted improved efficiency and reduced overcrowding (Techkitez, 2015). In Malaysia, Katsana’s GPS tracking and Debezt’s real-time alerts reduced passenger wait times by 30%, providing operators with performance analytics to optimize schedules (Jakubauskas, 2010). The eBus Services System (EBS), developed using Agile methodology, unified web and mobile platforms for ticket booking. Technologies like Bootstrap and AJAX ensured responsiveness, while PHP-MySQL handled backend operations (Ubale et al., 2024).

Looking ahead, the future of digital ticketing systems lies in expansion to multi-modal transport, AI-driven analytics, and sustainability initiatives. Integrating trains and flights into platforms like Mobitix could create a unified travel ecosystem, enhancing user convenience (Vimukthi, 2023). Machine learning can predict peak travel times, optimize routes, and personalize user

experiences, further improving operational efficiency (Ubale et al., 2024). Additionally, digital ticketing reduces paper waste, aligning with global sustainability goals and promoting eco-friendly public transportation (Mezghani)

In conclusion, digital bus ticketing systems address critical inefficiencies in manual processes, offering real-time tracking, secure payments, and enhanced user experiences. Case studies from Sri Lanka, Malaysia, and Nigeria demonstrate their scalability and impact. However, challenges like API dependency and rural adoption require ongoing innovation. Future systems must prioritize AI-driven analytics, multi-modal integration, and robust offline functionalities to achieve universal accessibility. The Mobitix project, with its focus on real-time tracking, secure payments, and user-friendly design, is well-positioned to contribute to this evolution, addressing the specific needs of Sri Lanka's public transportation system while setting a precedent for future developments in the region

1.8 Methodology

The development of the Mobitix Digital Bus Ticket Booking App follows a structured and iterative approach to ensure the project meets its objectives efficiently.

1. Development Approach

The project adopts the Agile Methodology, which is well-suited for software development due to its iterative and flexible nature. Agile allows for continuous feedback from stakeholders, so that quick adjustments and improvements can be done easily throughout the development process. The project is divided into sprints, each focusing on specific features such as ticket booking, real-time tracking, and payment integration. This approach makes sure that the app evolves incrementally, with each sprint delivering a functional version of the system.

2. Technology Stack

The technology stack has been carefully selected to align with the project's objectives and to make the app more user friendly and scalable.

- Frontend – Flutter is sued for cross-platform development, so that the app can be run smoothly on both Android and iOS. Flutter's rich set of prebuilt widgets and hot reload feature accelerate development and ensure a modern, responsive user interface.
- Backend – PHP is used to build a RESTful API for handling user authentication, ticket reservations, and trip data. MySQL is chosen as the database management system for its reliability and ability to store structured data efficiently.
- Real-time tracking – Google Maps API is integrating for real-time bus tracking, providing accurate location updates and route planning.

- Payment Integration – Local payment gateways like Genie, Koko, and eZcash are supported to ensure secure and convenient transactions for users in Sri Lanka.

3. System Architecture

The system follows a client-server architecture, where the mobile app interacts with the backend server via RESTful APIs. Key components include,

- Frontend – the mobile app built with Flutter, providing the user interface for ticket booking, seat selection, and real-time tracking.
- Backend – a PHP-based server handling business logic, user authentication, and data storage in MySQL.
- Third-Party Integrations – Google Maps API for real-time tracking and local payment gateways for secure transactions.

4. Data Collection.

To gather user requirements and system data, surveys, stakeholder meetings and system logs were used to ensure that the app is tailored to meet the needs of both passengers and administrators. Surveys were conducted with potential users to understand their needs and pain points with current bus ticketing systems. Stakeholder meetings were conducted for regular discussions with bus operators and administrators and Academic supervisors to gather insights into operational requirements. System logs are used to gather data from live bus schedules and GPS tracking systems are collected to ensure accurate real-time updates.

5. Implementation

The implementation process is divided into the following phases.

Table 4 Methodology Phases

Phases	Activities
Planning & Requirements Gathering	• Define project scope • Gather user requirements • Create a sprint backlog
System Design	• Develop wireframes • System architecture diagrams • Database schemas
Development	• Implement features such as ticket booking, real-time tracking, and payment integration in iterative sprints
Testing	• Conduct unit, integration, and user acceptance testing to ensure the app works as expected

Deployment	• Deploy the app to the Google Play Store and Apple App Store
Maintenance	• Monitor App performance, gather user feedback, and release updates to improve functionality

6. Testing methods

- To ensure the app's reliability and performance, the following testing methods are employed:
- Unit Testing : Individuals components (eg: ticket booking, payment processing) are tested for functionality
- Integration Testing : Ensures that different modules (eg: Frontend, backend, APIs) work together seamlessly.
- Security Testing : Verifies that user data is encrypted and secure, with measures like multi-factor authentication (MFA) in place
- User Acceptance Testing : Conducted with a sample group of users to validate the app's usability and functionality.

7. Security & Compliance

Security is a top priority for Mobitix. The following the measures are implemented.

- Data encryption – Sensitive user data, such as payment details and personal information, is encryption during storage and transmission
- Authentication – Multi-factor authentication is used to prevent unauthorized access.
- Compliance – The app adheres to data protection regulations such as GDPR and local laws in Sri Lanka, ensuring user privacy and data security

8. Limitations.

The project acknowledges certain limitations, including

- Time Constraints – the project must be completed within the academic timeline, limiting the scope of features.
- Budget Constraints – Limited funding restricts the use of expensive third-party tools or services.
- Hardware Dependencies – The app's performance may be affected by the quality of users devices and internet connectivity. Especially in rural areas

2. Analysis

Problem Definition

The core problem addressed by this project is the inefficiency of Sri Lanka's current bus ticket reservation systems, characterized by the lack of real-time tracking, poor refund management, and overcrowded buses due to manual processes. These issues lead to passenger dissatisfaction,

unreliable schedules, and operational challenges for bus operators, hindering the modernization of public transportation.

2.1 Analysis Techniques

2.1.1 SWOT ANALYSIS

The Mobitix Digital Bus Ticket Booking App project represents a great opportunity to improve Sri Lanka's public transportation system, but it also faces several challenges that needs to be addressed. As such, the Strengths, Weaknesses, Opportunities, and Threats (SWOT) of the project can be analyzed as follows.

Strengths

Mobitix has several internal strengths that positions it as a practical solution for modernizing bus ticketing in Sri Lanka. The app's user-friendly interface, developed using flutter, will provide a smooth experience for passengers, with features like dynamic seat selection and real-time bus tracking improving convenience. The integration of Google Maps API provides accurate real-time tracking, keeping passengers informed about bus locations and estimated arrival times. Moreover, the app supports secure online payments through local gateways like genie, Koko and eZcash, which are widely used in Sri Lanka, making the app more accessible and trustworthy. The cross-platform compatibility of the app will further broaden it's reach, while the inclusion of an admin panel will allow for efficient management of bus schedules, reservations, and performance analytics, improving operational efficiency for bus operators.

Weaknesses

Despite its strengths, Mobitix also has internal weaknesses. The application's high reliance on third-party APIs such as Google Maps and Payment Gateways risks interruption of services or integration issues. Additionally, the app's limited offline functionality could undermine usability in areas with poor internet coverage, an issue in rural Sri Lanka. The advanced integration with local bus schedules and payment systems can also be technically demanding. Especially if data formats or APIs are inadequately documented. Moreover, the high level of development

complexity for functionalities like dynamic seat selection and refund management can lead to delays or increased expense in development process.

Opportunities

The outside world also provides numerous opportunities for Mobitix. Growing smartphone penetration in Sri Lanka provides an enormous and increasing user base, particularly among commuters in urban and rural areas. Collaboration with the government or the private sector may further proper the adoption and use of the app in the public transport system. expansion potential into other modes of transport, such as trains or inter-city buses, which would increase the utility and market size of the app. Leverage of data analytics to analyze user behavior and bus performance would open opportunities for improved service and optimized scheduling. Moreover, the application also fosters global trends toward sustainability as it promotes environmentally responsible public transportation and reduces congestion jams, which would be enhanced by green stakeholders.

Threats

However, the project is not exempt from external challenges. Competition from existing bus ticketing systems both offline and online will hinder Mobitix's penetration into the market. Resistance to change among users, particularly in rural areas where traditional ticketing is more common, can hinder adoption. Management of sensitive user data, such as personal information and payments, exposes the app to cybersecurity risks, which can damage user trust if poorly managed. Regulatory compliance in the form of GDPR and domestic data protection regulations adds an added layer of sophistication. Finally, GPS accuracy issues, caused by poor signal intensity or environmental conditions, can lead to user disappointment if real-time tracking is unpredictable.

Accordingly, Mobitix has great potential to address Sri Lanka's inefficient bus ticketing system because it's easy to use, real-time, and provides secure payment. However, the project must overcome challenges such as third-party API reliance, limited offline usage, and resistance to change. By utilizing opportunities like rising smartphone penetration, government subsidies, and

data analytics, and countering threats like cyber attacks and regulatory penalties, Mobitix can establish a stable and efficient public transport system in Sri Lanka.

2.1.2 Use Case Diagram

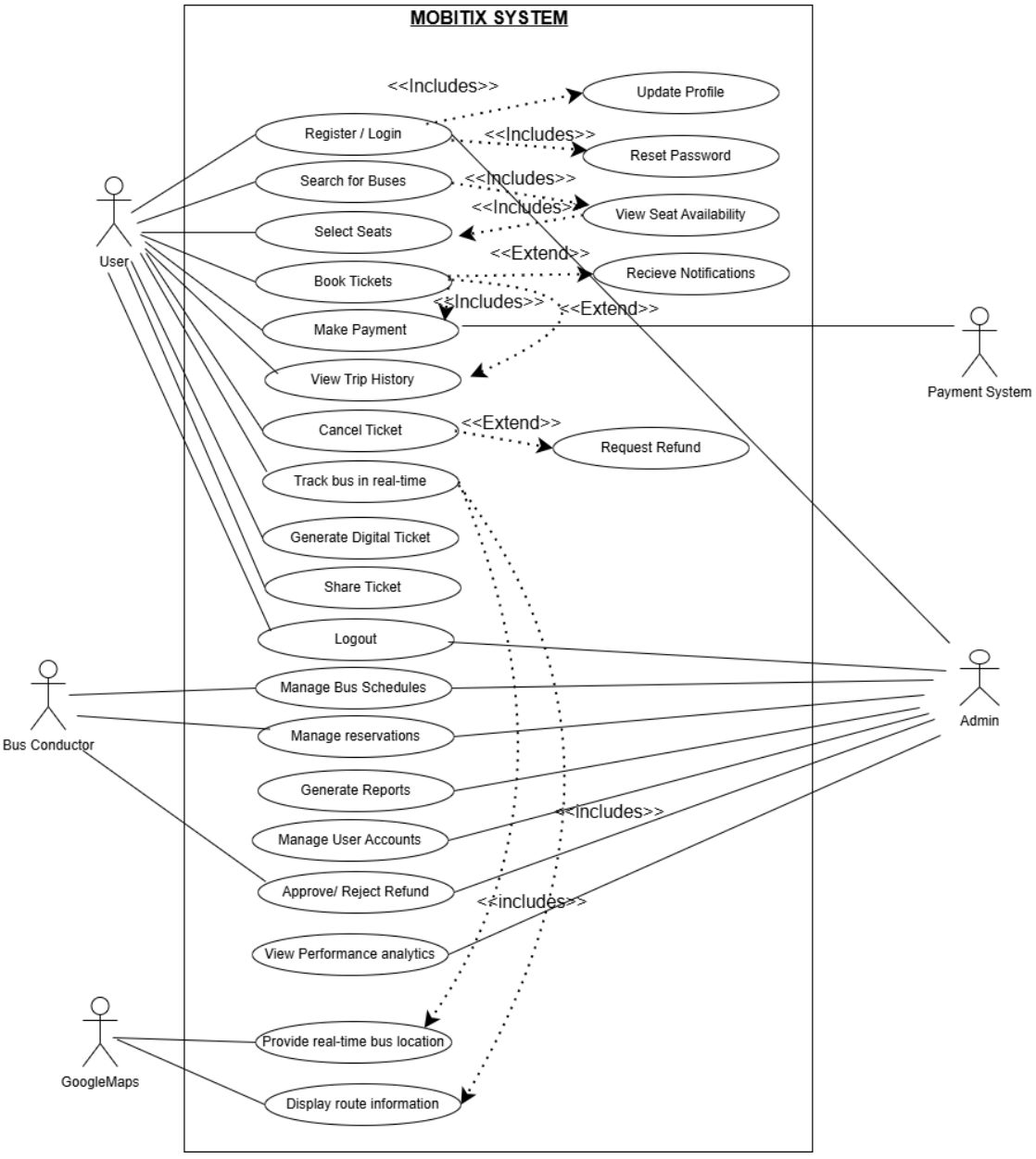


Figure 2 Use case diagram

The use case diagram for the Mobitix Digital Bus Ticket Booking App illustrates the interactions between different users (actors) and the system's functionalities. The primary actors include Users (Passengers), Admin, Bus Conductors, Payment System, and Google Maps.

User Interactions

Passengers (Users) interact with the system by performing essential tasks such as registering/logging in, searching for buses, selecting seats, booking tickets, making payments, viewing trip history, and canceling tickets. These actions are core to the ticket booking experience. Some use cases, such as viewing seat availability and receiving notifications, are included within the main booking process to enhance the user experience. Additional features like tracking the bus in real-time, generating digital tickets, and sharing tickets offer improved convenience.

The Admin has extended control over the system, managing bus schedules, reservations, and user accounts. They can also generate reports, approve/reject refunds, and view performance analytics to oversee operations effectively. The Bus Conductor is responsible for tasks such as managing bus schedules and reservations, ensuring smooth operations.

The Payment System is linked to processing payments and handling refund requests, ensuring financial transactions are integrated seamlessly. Additionally, Google Maps provides real-time bus tracking and displays route information, allowing users to stay updated on their bus’s status.

Some actions, such as resetting passwords, updating profiles, and viewing seat availability, are included as additional functionalities that support the primary use cases. Others, like receiving notifications and requesting refunds, extend from main use cases, indicating they are optional but crucial enhancements.

2.1.3 User Stories

Passenger User Stories

1. Ticket Booking & Reservation
User Story:
"As a passenger, I want to search for available buses by date, time, and route so that I can find a suitable option for my trip."
Acceptance Criteria:

- The user can input travel date, time, and destination to search for buses.
- The system displays a list of available buses with details (departure time, duration, seat availability).
- Users can filter/search buses by operator, price, or seat availability.

2. Dynamic Seat Selection
User Story:
"As a passenger, I want to select my preferred seat from a bus seating layout so that I can choose a comfortable seat for my journey."
Acceptance Criteria:

- The system displays a real-time seating layout for the selected bus.
- Users can view available, booked, and reserved seats.
- Once a seat is selected, it is temporarily locked until payment is completed.

3. Online Payment Processing
User Story:
"As a passenger, I want to make an online payment for my ticket so that I can confirm my booking instantly."
Acceptance Criteria:

- Users can choose from multiple payment options (Genie, Koko, eZ Cash).
- The system securely processes payments and confirms the transaction.
- A payment receipt and e-ticket are generated upon successful payment.
- If payment fails, the system provides an appropriate error message.

4. Real-time Bus Tracking

User Story:
"As a passenger, I want to track my bus in real-time so that I can estimate its arrival time and plan accordingly."
Acceptance Criteria:

- Users can access a live map showing the real-time location of the booked bus.
- The system provides estimated arrival times at the boarding point.
- Notifications are sent in case of delays or changes in the schedule.

5. Trip History & Refund Requests
User Story:
"As a passenger, I want to view my past and upcoming trips so that I can manage my bookings and request refunds if necessary."
Acceptance Criteria:

- Users can view a list of past and upcoming trips.
- The system allows users to request a refund for eligible bookings.
- Refund status updates are sent via notifications.

Admin User Stories

6. Bus Schedule Management
User Story:
"As an administrator, I want to add, edit, and remove bus schedules so that I can keep the system up to date with current routes and timings."
Acceptance Criteria:

- Admins can add new bus schedules with details (route, departure time, stops).
- Admins can update schedules in case of changes.
- Deleted schedules no longer appear in passenger searches.

7. Reservation & Ticketing Management
User Story:
"As an administrator, I want to view and manage ticket reservations so that I can handle customer inquiries and refunds efficiently."
Acceptance Criteria:

- Admins can search and filter reservations by user, date, or bus route.
- Admins can manually cancel or modify bookings when necessary.
- Refund requests are reviewed and approved based on policy rules.

Security and Compliance user stories

8. Secure Login & Authentication

User Story:

"As a user, I want to log in securely so that my personal and payment information is protected."

Acceptance Criteria:

- Users can log in using email/phone number and password.
- Multi-factor authentication (MFA) is available for enhanced security.
- The system enforces secure password policies.

9. Data Privacy & User Access Control

User Story:

"As a user, I want to control who can access my personal and booking information so that I can ensure my privacy and security."

Acceptance Criteria:

- Users can modify account privacy settings.
- The system enforces permissions for user data access

2.1.4 Requirements Engineering

Functional Requirements

User Requirements (Passengers)

User registration and Authentication

- Users must be able to sign up using their email or phone number.
- The system must enforce email/phone number verification before account activation.
- Secure login/logout must be implemented using hashed passwords and multi-factor authentication (MFA).
- Users should be able to reset their password via email or OTP-based authentication.
- The system should allow users to update profile details (name, contact number, payment preferences).

Ticket Booking and Reservation

- Users must be able to search for available buses by entering the departure and destination locations along with the date and time.
- The system should provide a list of available buses with details such as route, seat availability, price, and departure time.
- Users should be able to select specific seats dynamically via an interactive seat map.
- The system must allow users to complete their ticket purchases via integrated local payment gateways (Genie, Koko, eZ Cash).

- Upon successful payment, users should receive an electronic ticket (e-ticket) in the app and via email.
- The system must support QR code-based tickets for verification during boarding.
- Users should be able to share e-tickets with others via email or messaging apps.

Real-Time Bus Tracking

- The system should integrate with Google Maps API to track buses in real time.
- Users must be able to view the live location of their booked bus and estimated time of arrival (ETA).
- Notifications should be sent for key updates, such as bus departure, delays, and arrival alerts.
- Users should have the ability to set alerts for their stop to receive notifications before arrival.

Trip Management

- Users must be able to view their past trips, including date, time, bus details, and fares paid.
- The system should allow users to cancel their booking before departure.
- Refund requests must be processed based on predefined policies (e.g., full refund if canceled within 24 hours, partial refund after that).
- Users should be able to provide feedback and rate their travel experience.

Payment Integration

- Users must be able to securely make online payments using supported local payment gateways (Genie, Koko, eZ Cash).

<u>Administrator Requirements</u>

Bus Management

- Administrators must be able to add, edit, and remove buses and schedules.
- The system should allow assigning drivers and conductors to specific buses.
- Bus route details must be customizable, including stops, departure times, and fares.

Bus Reservation and Ticket Management

- Administrators should have access to view all active and past bookings.
- The system must provide an interface for approving or rejecting refund requests.
- Administrators should be able to modify bus schedules in case of route changes or cancellations.

Performance Analytics

- The system should generate automated reports on ticket sales, peak travel times, and revenue trends.
- Analytics should track user behavior, including frequently booked routes and customer preferences.
- The dashboard should highlight key performance indicators (KPIs) such as average booking time, customer ratings, and refund rates.

Non-Functional Requirements

Performance Requirements

- The system must handle at least 1000+ concurrent users without degradation in performance.
- Ticket booking transactions must be processed within 3 seconds on average.
- The real-time bus tracking feature should update location data at least every 10 seconds.

Security Requirements

- All user data must be protected in compliance with GDPR and CCPA regulations.
- Payment transactions must be encrypted.
- User authentication must support multi-factor authentication (MFA).
- Role-based access control (RBAC) must be enforced for administrators and passengers.

Usability and Accessibility

- The user interface must be intuitive and easy to navigate, with clear CTAs (Call-to-Actions).
- The app should support Sinhala, Tamil, and English languages for broader accessibility.
- The application must be fully responsive and compatible with both Android and iOS devices.
- The app must support screen reader compatibility for visually impaired users.

Constraints and Assumptions

- Internet access is required for ticket booking, seat selection, and bus tracking.
- The payment gateways (Genie, Koko, eZ Cash) must provide API documentation for integration.
- Bus operators must update real-time GPS tracking information for accurate live location updates.
- Refund policies must be clearly defined and agreed upon by both users and administrators.
- The mobile app should be optimized for low-bandwidth connections to accommodate users with limited internet access.

2.2 Analysis Outcomes

The analysis of the Mobitix Digital Bus Ticket Booking App reveals several key outcomes, providing valuable insights into the project's feasibility, challenges, and strategic opportunities. These outcomes are derived from the SWOT analysis**,** user needs**,** and requirements engineering**,** highlighting critical aspects that will shape the development and implementation of the system.

Key Outcomes

1. Addressing the inefficiencies in the current system
    - The analysis confirms that manual ticketing processes, lack of real-time tracking, and inefficient refund management are major pain points in Sri Lanka's public transportation system.
    - Mobitix aims to improve bus ticket reservations by offering digital booking, dynamic seat selection, and automated refund management, reducing passenger inconvenience.

2. Clear Definition of a System Capabilities
    - Functional requirements have been established to include secure user authentication, online ticket purchases, e-ticket generation, and real-time bus tracking.
    - The integration of Google Maps API for live tracking and local payment gateways (Genie, Koko, eZ Cash) ensures a user-friendly and reliable experience for passengers.

3. Identification of Non-Functional Requirements
- The system must meet critical performance, security, scalability, and usability requirements.
    - Key performance goals include handling 1000+ concurrent users and ensuring that ticket booking transactions are processed within 3 seconds.
    - Security requirements emphasize GDPR compliance, data encryption, and multi-factor authentication to safeguard sensitive user information.

4. Technological Feasibility and Potential Challenges
    - The choice of technologies, including Flutter (for cross-platform development), PHP (backend), and Google Maps API (for real-time tracking),confirms the technical feasibility of the project.
    - However, challenges such as integration issues with third-party APIs (Google Maps, local payment gateways) and ensuring seamless real-time tracking accuracy need to be addressed.

5. Market Readiness and Adoption Challenges
    - The growing smartphone penetration in Sri Lanka presents a significant opportunity for app adoption.
    - However, resistance to change, especially among rural commuters accustomed to traditional ticketing methods, may slow initial adoption. A user education campaign and incentives (e.g., discounts on first bookings) could mitigate this challenge.

6. Competitive and Regulatory Considerations

- Competition from existing ticketing systems (offline and online) may impact market penetration. Strategic partnerships with bus operators and government transport agencies will be crucial for widespread adoption.
- Compliance with data protection regulations (GDPR, local data privacy laws) must be strictly maintained to avoid legal risks and ensure user trust.

7. opportunities for future enhancements

- Beyond bus ticketing, the platform has the potential to expand into train and inter-city transport reservations, broadening its market reach.
- Data analytics could be leveraged to analyze user behavior, predict peak travel times, and optimize bus schedules, improving both user experience and operational efficiency for bus operators.
- Integrating AI-based recommendations for route planning and bus schedules could further enhance the app's value proposition.

8. Strategic planning for Implementation

- The Agile development approach will be used to iteratively refine features, incorporate user feedback, and address emerging challenges.
- Initial deployment will target urban areas with strong internet connectivity and smartphone adoption, followed by expansion to rural regions with necessary adaptations (e.g., offline booking support).

3. Design

3.1 Design Techniques

3.1.1. Flow Charts
User Registration Flow Chart

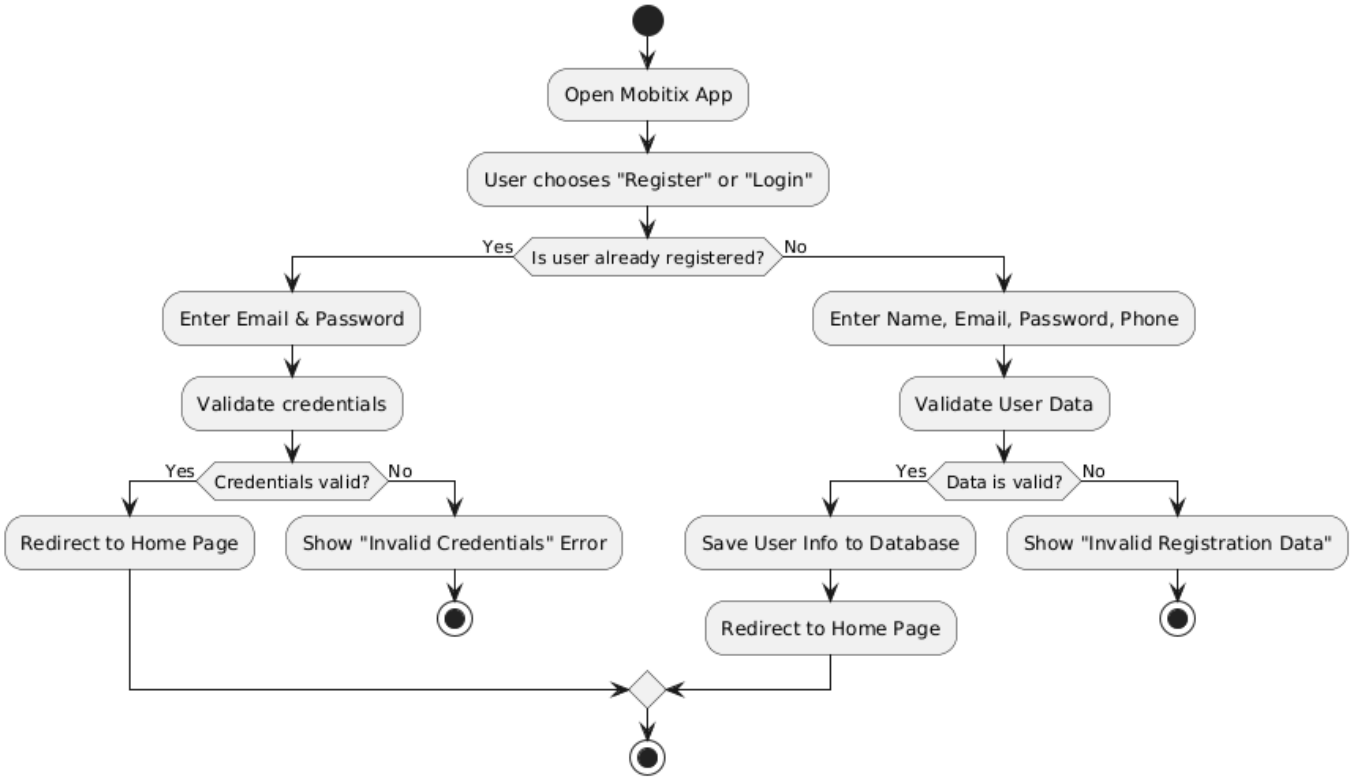


Figure 3 User Registration Flow Chart

Ticket Booking Flow Chart

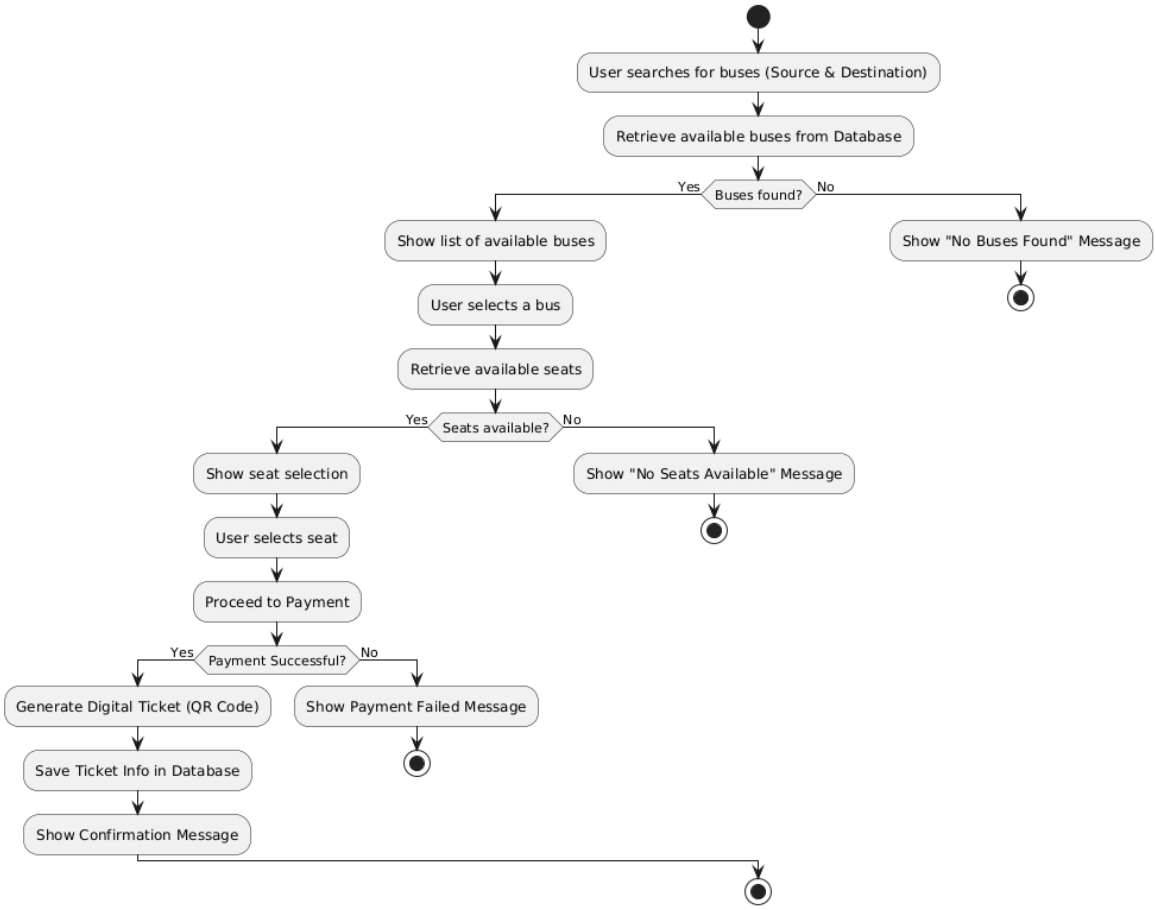


Figure 4 Ticket Booking Flow Chart

Ticket Cancellation Flow Chart

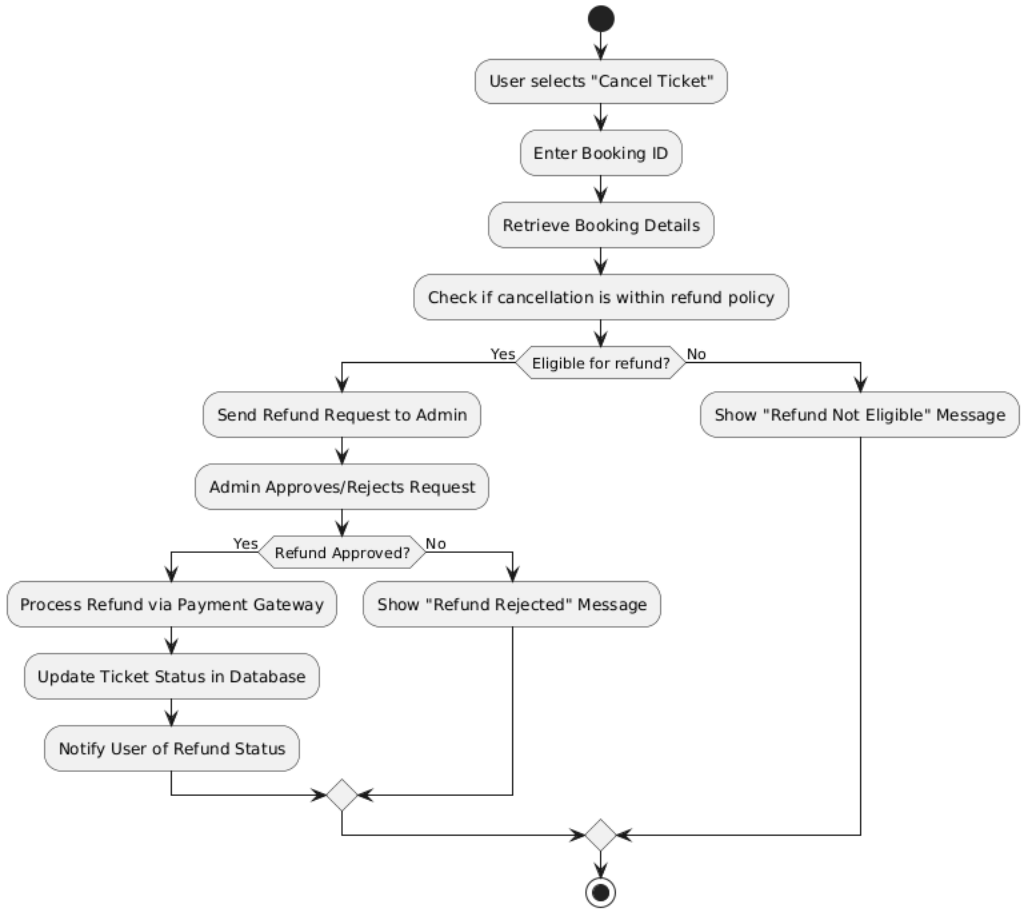


Figure 5 Ticket Cancellation Flow Chart

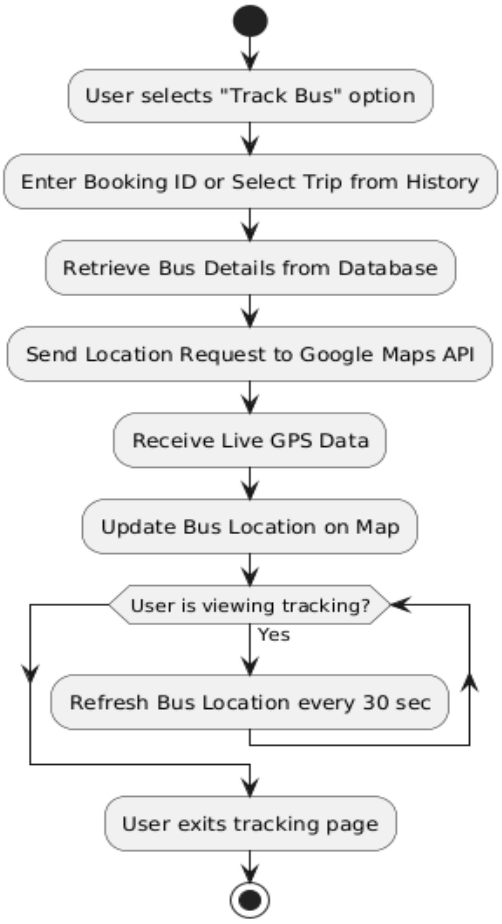


Figure 6 Real-time Bus Tracking Flow Chart

Admin Login Flow Chart

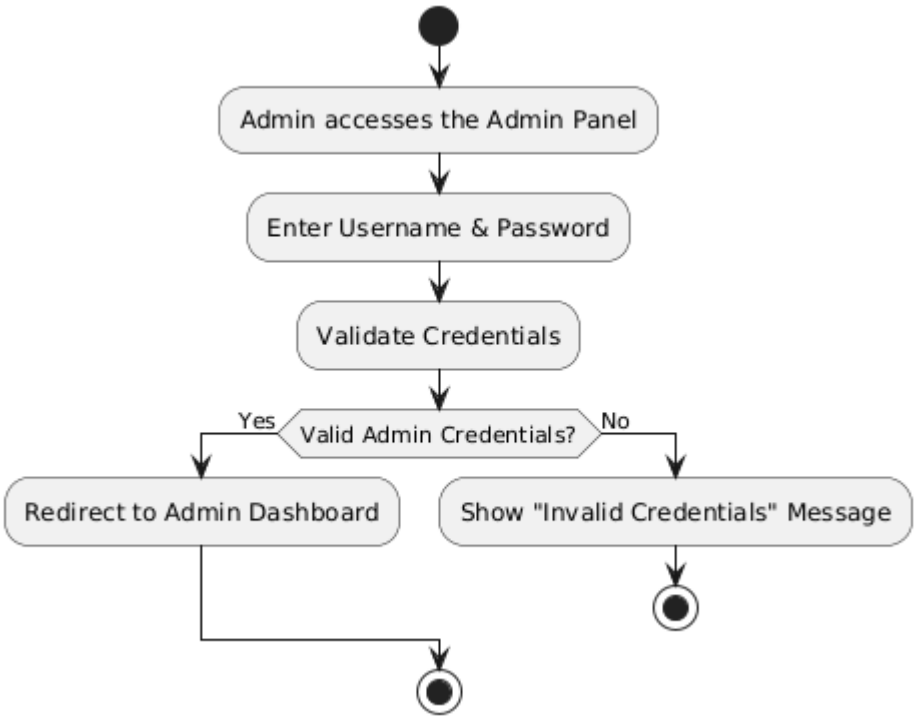


Figure 7 Admin Login Flow Chart

Admin Manage Bus Schedule Flow Chart

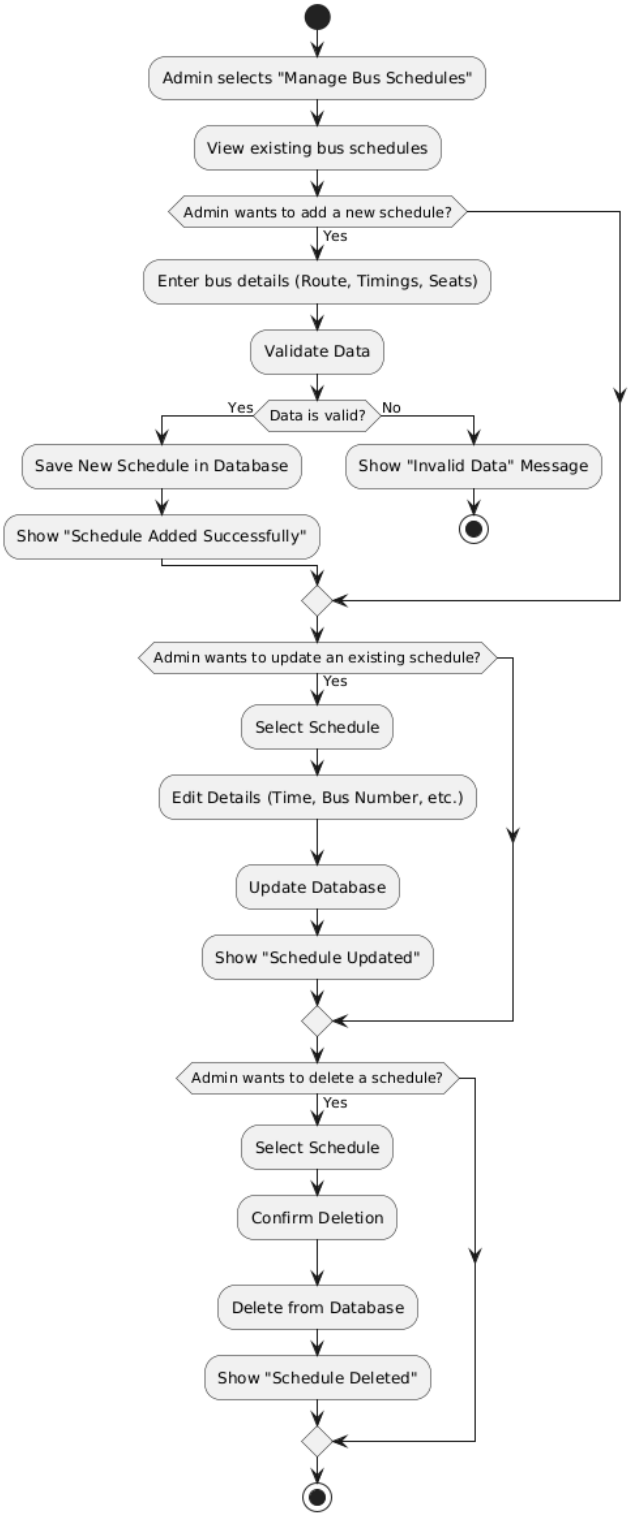


Figure 8 Admin Managing bus schedules flow chart

3.1.2 Activity Diagrams

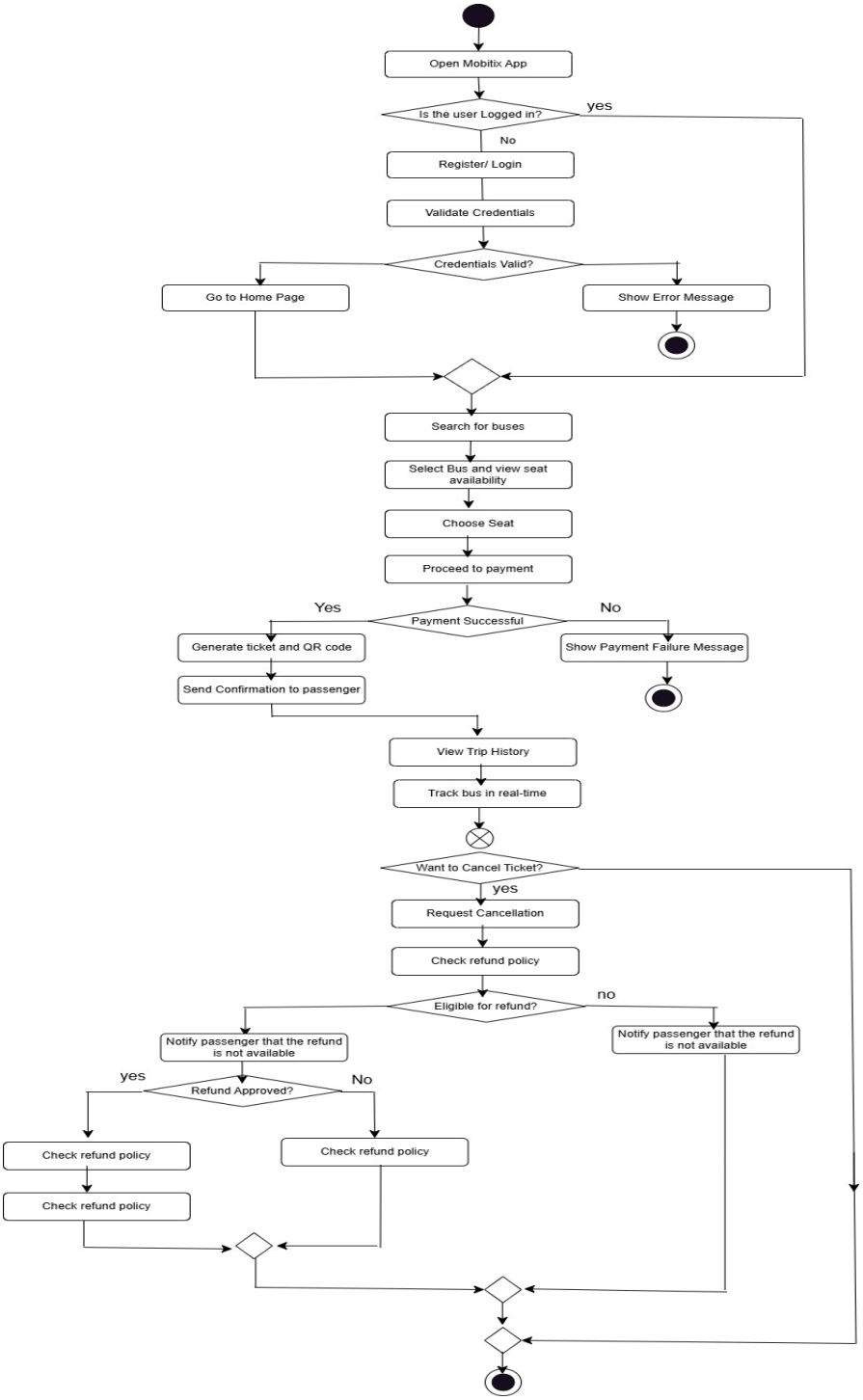


Figure 9 User Activity Diagram

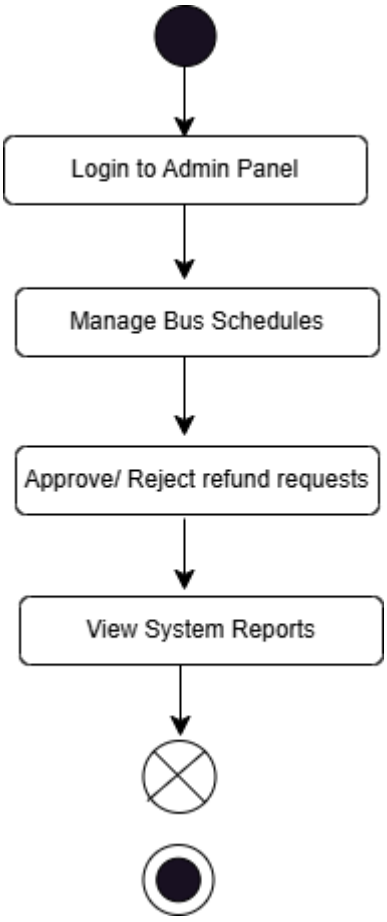


Figure 10 Admin Activity Diagram

3.1.3 Sequence Diagram

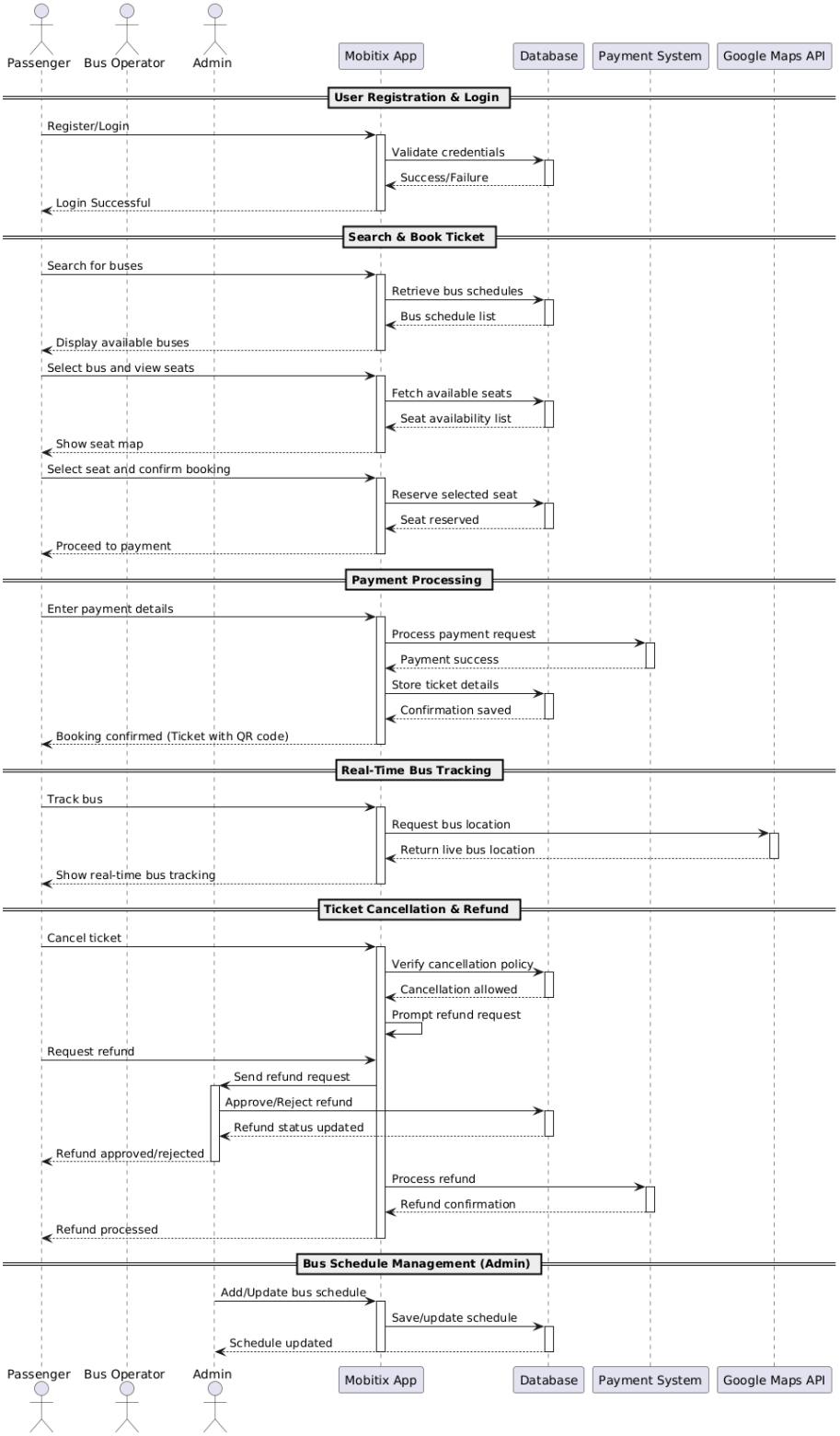


Figure 11 Sequence Diagram

3.1.4 Design Narrative

Key Components

User Interface (UI)

- Frontend Technologies: The system's UI will be built using Flutter, a cross-platform framework that enables a seamless, responsive, and visually appealing experience for both Android and iOS users.
- Design Principles: The UI will follow modern design principles, ensuring intuitive navigation, accessibility, and ease of use for passengers and administrators.
- Key Interfaces:
    - Passenger Dashboard: Displays ticket booking options, real-time bus tracking, and trip history.
    - Admin Dashboard: Provides tools for bus schedule management, user analytics, and refund approvals.
    - Booking Page: Features a dynamic seat selection system, bus search filters, and a secure payment gateway.
    - Notifications Panel: Displays alerts for bus departures, cancellations, estimated arrival times, and payment confirmations

Backend Architecture

- Programming Language: The backend will be developed using PHP, chosen for its efficiency in handling RESTful API requests and database operations.
- Database Management: MySQL will be used to store and manage user profiles, ticket reservations, transaction records, and bus schedules. The database will be designed to support high-speed queries and ensure data consistency.
- Data Processing: The system will leverage server-side validation and caching mechanisms to optimize performance and ensure secure data handling.

Integration with External Systems

- Payment Gateway Integration: Mobitix will integrate with local Sri Lankan payment gateways such as Genie, Koko, and eZ Cash via API connections, enabling secure and convenient digital payments.
- Real-time Bus Tracking: The Google Maps API will provide live GPS tracking of buses, allowing users to view real-time location updates and estimated arrival times.
- Notification Services: Firebase Cloud Messaging (FCM) will be used to send push notifications for bus status updates, payment confirmations, and refund alerts.

Agile Methodology

The development of the Mobitix Digital Bus Ticket Booking App will follow the Agile methodology, which emphasizes flexibility, iterative development, and continuous improvement. Agile allows for incremental progress, enabling the project to adapt to changes and feedback throughout the development lifecycle.

- Iterative Development: The project will be broken down into manageable tasks, with each feature developed, tested, and refined before moving on to the next.
- Continuous Testing & Improvement: Regular testing will be conducted to ensure functionality, security, and usability, allowing for early identification and resolution of issues.
- Adaptive Planning: Instead of rigid upfront planning, development priorities can be adjusted based on technical feasibility and evolving requirements.
- Focus on Deliverables: The project will be developed in stages, ensuring that each component (e.g., authentication, ticket booking, payment integration) is fully functional and optimized before deployment.

Major Functionalities

Passenger Features

- Seamless Registration & Authentication: Users can sign up using email or phone number, with OTP verification for security.
- Dynamic Seat Selection: An interactive seat map will allow passengers to select their preferred seats.
- Secure Online Ticket Purchase: Users can pay via integrated payment gateways, with instant e-ticket generation and a QR code-based boarding pass.
- Trip Management: Passengers can view trip history, cancel bookings, and request refunds based on predefined policies.

Administrator Features

- Bus Schedule Management: Admins can add, edit, and delete bus schedules, update fares, and assign drivers.
- Booking & Refund Approvals: Admins can view real-time booking statistics and process refund requests efficiently.
- Performance Analytics: Data insights will help operators analyze ticket sales trends, peak travel times, and user preferences.

Key Benefits

- Enhanced Public Transport Efficiency: Real-time tracking, online booking, and digital payments will significantly reduce congestion, waiting times, and manual errors.
- Improved User Experience: A modern, intuitive mobile app will provide greater convenience for passengers compared to traditional ticketing systems.
- Operational Optimization: Bus operators will benefit from automated scheduling, ticket sales reports, and enhanced customer insights, allowing them to refine services.
- Security & Compliance: Adherence to data privacy regulations (GDPR, CCPA) ensures user trust and legal compliance.

3.2 System Overview

3.2.1 System Architecture

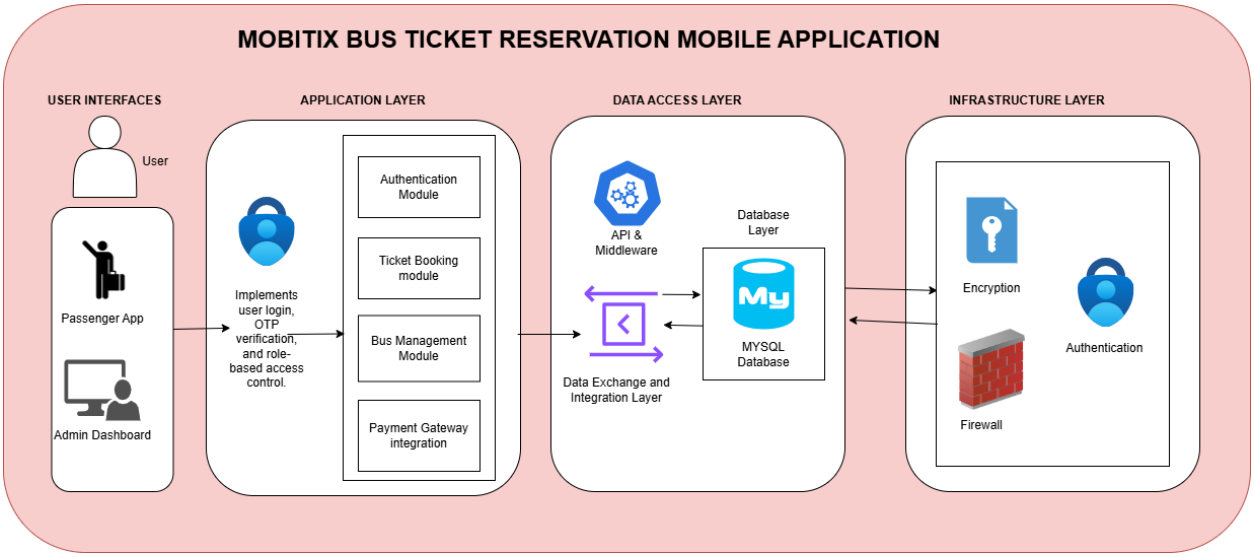


Figure 12 System Architecture Diagram

The System Architecture Diagram for the Mobitix Digital Bus Ticket Booking App visually represents the interaction between various system components of the mobile application. The diagram is structured into multiple layers, each playing a distinct role in the system's functionality.

At the top, the User Interface Layer includes the Passenger App, and the Admin Dashboard. These interfaces enable users & administrators to interact with the system by searching for routes, booking tickets, managing schedules, and monitoring transactions. Arrows from these interfaces point toward the Application Layer, indicating user requests being processed by the backend system.

The Application Layer serves as the core processing unit of the system. It comprises several modules, including Authentication, Ticket Booking, Bus Management, and Payment Gateway Integration. The Authentication Module handles user login and role-based access control, while the Ticket Booking Module manages seat selection, confirmation, and ticket generation. The Bus Management Module ensures that schedules and routes are properly maintained, and the Payment Gateway facilitates secure transactions through external providers. This layer interacts with the Data Access Layer via APIs and middleware, ensuring smooth communication between the frontend and backend.

The Data Access Layer consists of APIs & Middleware and the Data Exchange & Integration Layer, which act as intermediaries between the backend services and the Database Layer. This setup ensures efficient data retrieval and synchronization across different modules. The Database

Layer, utilizing SQL/NoSQL databases like MySQL, stores critical system data, including user details, booking history, and bus schedules.
The Security Layer in the infrastructure layer implements encryption, firewalls, and intrusion detection to protect sensitive user data. Bidirectional arrows indicate interactions between the Application Layer and Cloud Infrastructure, ensuring a secure and responsive system.

3.2.2. Data Model

The data model of the Mobitix Digital Bus Ticket Booking App is structured to efficiently manage bus ticket bookings, user information, and financial transactions while ensuring data integrity and security. The key components of the model include:

Entities

- **User:** Represents passengers, admins, and bus operators. Stores user details such as full name, email, role, phone number, and password.
- **Ticket:** Stores booking details, including schedule ID, user ID, seat number, price, booking time, payment status, and ticket status.
- **Payment:** Manages payment transactions, including payment ID, user ID, ticket ID, payment method, payment status, amount, timestamp, and transaction ID.
- **Refund:** Handles refund requests, linking users and tickets, including approval status and request date.
- **Bus:** Represents buses in the system, storing details such as bus number, operator name, total seats, and operational status.
- **Route:** Defines bus routes, storing departure time, arrival time, date, and available seats.
- **Bus Schedule:** Manages bus schedules by linking buses to schedules and tracking available seats.
- **GPS:** Stores bus tracking information, including timestamp, latitude, and longitude.

Relationships:

- A User can book multiple Tickets.
- A Ticket is associated with a Bus Schedule and a User.
- A User can make multiple Payments for Tickets.
- A User can request multiple Refunds, which are linked to a specific Ticket.
- A Bus follows a Route and is assigned to multiple Schedules.
- A Bus Schedule is linked to a Bus and multiple Tickets.
- A Bus has a GPS tracking record.

Normalization:

The data model is designed with normalization principles to eliminate redundancy and ensure data consistency. Key attributes are separated into different tables, and foreign key constraints maintain referential integrity.

Security Considerations:

- User authentication: Passwords are securely stored.
- Payment security: Transactions are linked to payment gateways, ensuring encrypted and secure processing.
- Access control: Role-based access control (RBAC) restricts functionalities based on user roles.

Data Types:

- The model uses optimized data types for efficient storage and retrieval:
  - VARCHAR for text attributes such as names and emails.
  - INTEGER for IDs and numerical attributes.
  - DECIMAL for monetary values.
  - DATETIME for timestamps and booking times.
  - ENUM for predefined categories such as user roles and payment statuses.

Entity Relationship Diagram

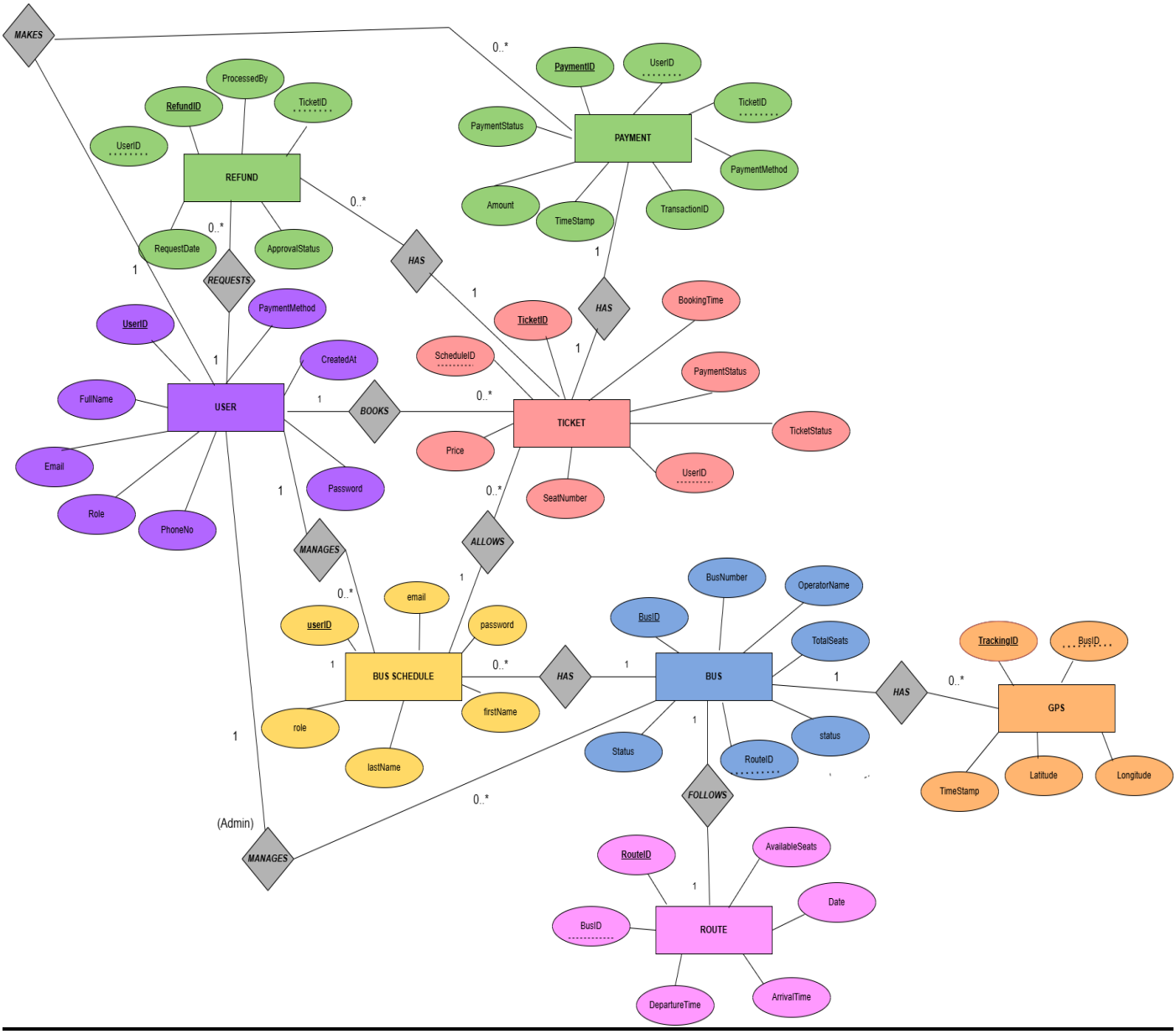


Figure 13 Entity Relationship Diagram

This ER diagram represents the structure of the Mobitix bus ticket reservation system, detailing the relationships between key entities such as users, tickets, buses, routes, payments, refunds, and schedules.

The User entity, identified by UserID, includes attributes such as FullName, Email, Role, PhoneNo, Password, and CreatedAt. Users can book tickets, make payments, and request refunds. The system distinguishes between regular users and administrators, with the latter managing bus schedules.

The Ticket entity, identified by TicketID, contains attributes such as ScheduleID, Price, SeatNumber, BookingTime, PaymentStatus, TicketStatus, and UserID. Each ticket is associated with a schedule and a user who books it. A user can book multiple tickets, while each ticket corresponds to a single schedule.

The Payment entity, linked to tickets, includes PaymentID, UserID, TicketID, PaymentMethod, Amount, TimeStamp, and TransactionID. It ensures that every ticket purchase is recorded along with payment details.

Similarly, the Refund entity, identified by RefundID, consists of UserID, TicketID, RequestDate, ApprovalStatus, and ProcessedBy. Users can request refunds, which are either approved or rejected by an administrator.

The Bus Schedule entity, managed by administrators, includes UserID, Email, Password, FirstName, and LastName. It ensures that schedules are controlled by authorized personnel.

The Bus entity, identified by BusID, contains BusNumber, OperatorName, TotalSeats, Status, and RouteID. Each bus follows a specific route and has a status indicating its availability.

The Route entity, identified by RouteID, consists of BusID, AvailableSeats, Date, DepartureTime, and ArrivalTime. It defines the journey details for buses.

Additionally, the GPS entity, linked to buses, includes TrackingID, BusID, TimeStamp, Latitude, and Longitude, allowing real-time tracking of buses.

These entities are connected through relationships such as "BOOKS" between users and tickets, "MAKES" between users and payments/refunds, "HAS" between tickets and payments, "MANAGES" between administrators and bus schedules, "FOLLOWS" between buses and routes, and "HAS" between buses and GPS tracking. These relationships ensure smooth operation of the bus reservation app by managing user interactions, payments, ticketing, and bus tracking efficiently.

Relational Schema

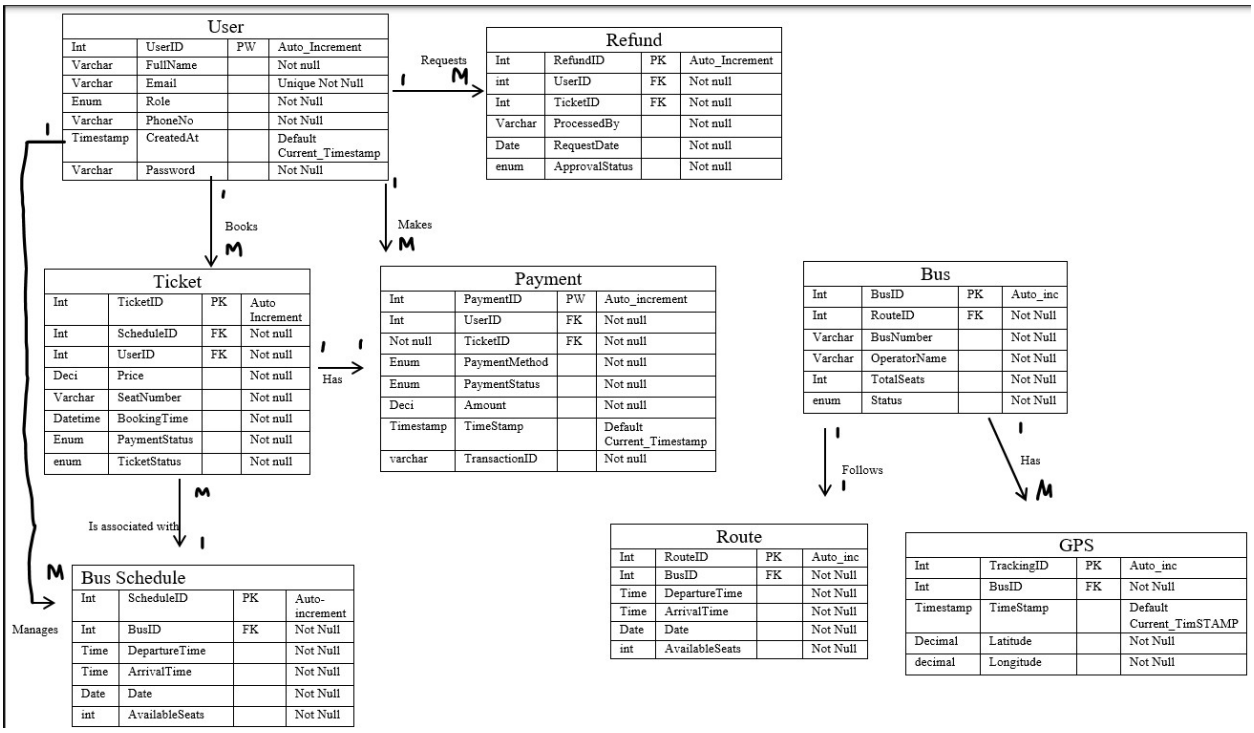


Figure 14 Relational Schema

Main Entities & Their Roles:

1. User – People using the system (passengers, admins). They can book tickets, make payments, and request refunds.
2. Ticket – Represents a bus booking, linked to a User and a Schedule.
3. Payment – Stores payment details when users buy tickets.
4. Refund – Allows users to request refunds, which an admin can approve or reject.
5. Bus – Represents a physical bus, linked to a Route and Schedule.
6. Route – Defines the start and end locations, including departure and arrival times.
7. Bus Schedule – Defines when a bus operates on a given route.
8. GPS – Tracks bus locations using latitude and longitude.

Relationships:

- A User can book multiple Tickets.
- A Ticket is linked to a Payment.
- A User can request a Refund for a ticket.
- An Admin manages Bus Schedules.
- A Bus follows a Route and has a GPS tracker.

Class Diagram

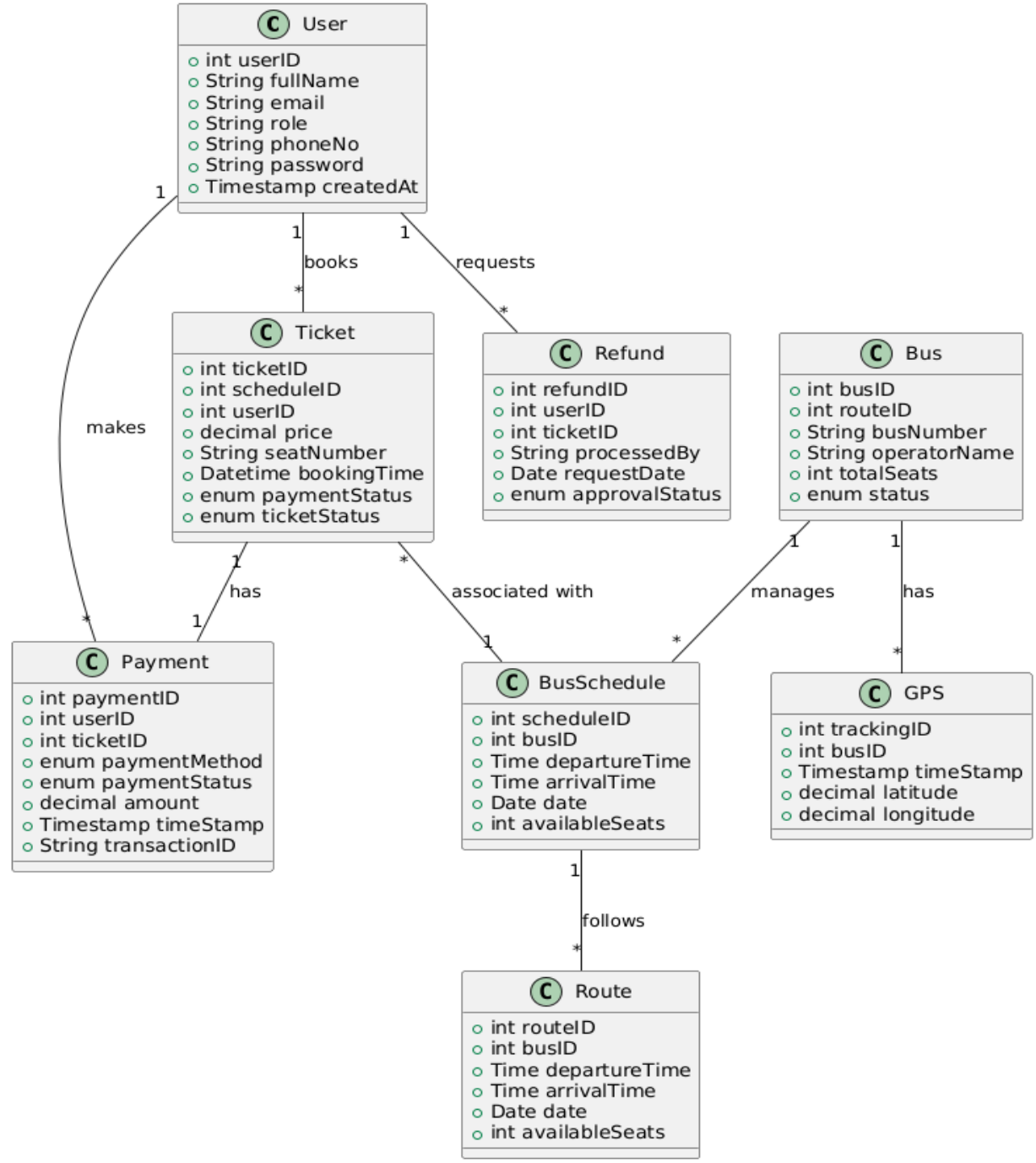


Figure 15 Class Diagram

The UML class diagram for the Mobitix Digital Bus Ticket Booking App represents the core entities and their relationships within the system. The User class, which includes attributes such as FullName, Email, and PhoneNo, interacts with other classes like Ticket, Payment, and Refund. A User can book multiple Tickets, which are associated with a BusSchedule that

manages bus timings and available seats. The Bus class is linked to Route, defining travel details like DepartureTime and ArrivalTime. The Payment class ensures secure transactions, while the Refund class handles ticket cancellations and approvals. Additionally, the GPS class tracks buses in real-time. Relationships such as one-to-many (eg: a user can have multiple tickets) and many-to-one (eg: multiple buses following a route) ensure a well-structured and scalable system.

Database Table Structures

1. User Table

Table 5 User Table

Column Name	Data Type	Constraints
UserID	Int	PRIMARY KEY
FullName	Varchar	NOT NULL
Email	Varchar	UNIQUE NOT NULL
Role	Enum	NOT NULL
PhoneNo	Varchar	NOT NULL
CreatedAt	Timestamp	DEFAULT Current_Timestamp
Password	Varchar	NOT NULL

2. Ticket Table

Table 6 Ticket Table

Column Name	Data Type	Constraints
TicketID	Int	PRIMARY KEY AUTO INCREMENT
ScheduleID	Int	FOREIGN KEY NOT NULL
UserID	Int	FOREIGN KEY NOT NULL
Price	Deci	NOT NULL
SeatNumber	Varchar	NOT NULL
BookingTime	Datetime	NOT NULL
PaymentStatus	Enum	NOT NULL
TicketStatus	enum	NOT NULL

3. Refund Table

Table 7 Refund Table

Column Name	Data Type	Constraints
RefundID	Int	PRIMARY KEY AUTO INCREMENT
UserID	int	FOREIGN KEY NOT NULL
TicketID	Int	FOREIGN KEY NOT NULL
ProcessedBy	Varchar	NOT NULL
RequestDate	Date	NOT NULL
ApprovalStatus	enum	NOT NULL

4. Payment Table

Table 8 Payment Table

Column Name	Data Type	Constraints
PaymentID	Int	PRIMARY KEY AUTO INCREMENT
UserID	Int	FOREIGN KEY NOT NULL
TicketID	Not null	FOREIGN KEY NOT NULL
PaymentMethod	Enum	NOT NULL
PaymentStatus	Enum	NOT NULL
Amount	Deci	NOT NULL
TimeStamp	Timestamp	DEFAULT Current_Timestamp
TransactionID	varchar	NOT NULL

5. Bus Schedule Table

Table 9 Bus Schedule Table

Column Name	Data Type	Constraints
ScheduleID	Int	PRIMARY KEY AUTO INCREMENT
BusID	Int	NOT NULL
DepartureTime	Time	NOT NULL
ArrivalTime	Time	NOT NULL
Date	Date	NOT NULL
AvailableSeats	int	NOT NULL

6. Bus Table

Table 10 Bus Table

Column Name	Data Type	Constraints
BusID	Int	PRIMARY KEY AUTO INCREMENT
RouteID	Int	FOREIGN KEY NOT NULL
BusNumber	Varchar	UNIQUE NOT NULL
OperatorName	Varchar	NOT NULL
TotalSeats	Int	NOT NULL
Status	enum	NOT NULL

7. Route Table

Table 11 Route Table

Column Name	Data Type	Constraints
RouteID	Int	PRIMARY KEY AUTO INCREMENT
BusID	Int	FOREIGN KEY NOT NULL
DepartureTime	Time	NOT NULL
ArrivalTime	Time	NOT NULL
Date	Date	NOT NULL
AvailableSeats	int	NOT NULL

8. GPS table

Table 12 GPS table

Column Name	Data Type	Constraints
TrackingID	Int	PRIMARY KEY AUTO INCREMENT
BusID	Int	FOREIGN KEY NOT NULL
TimeStamp	Timestamp	DEFAULT Current_Timestamp
Latitude	Decimal	NOT NULL
Longitude	decimal	NOT NULL

3.2.3 Wireframes

1. Login Page

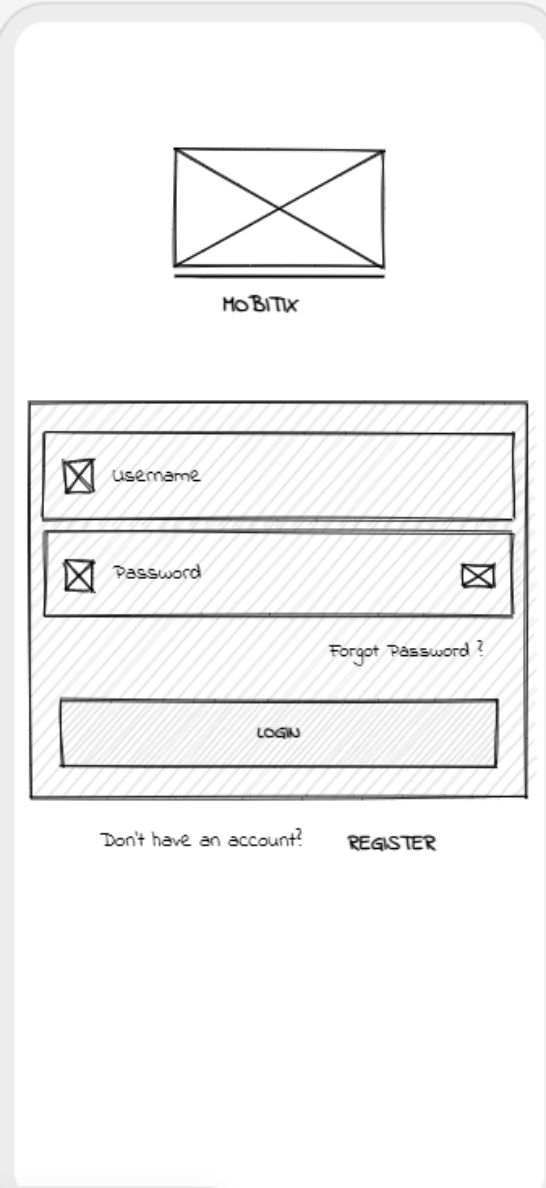
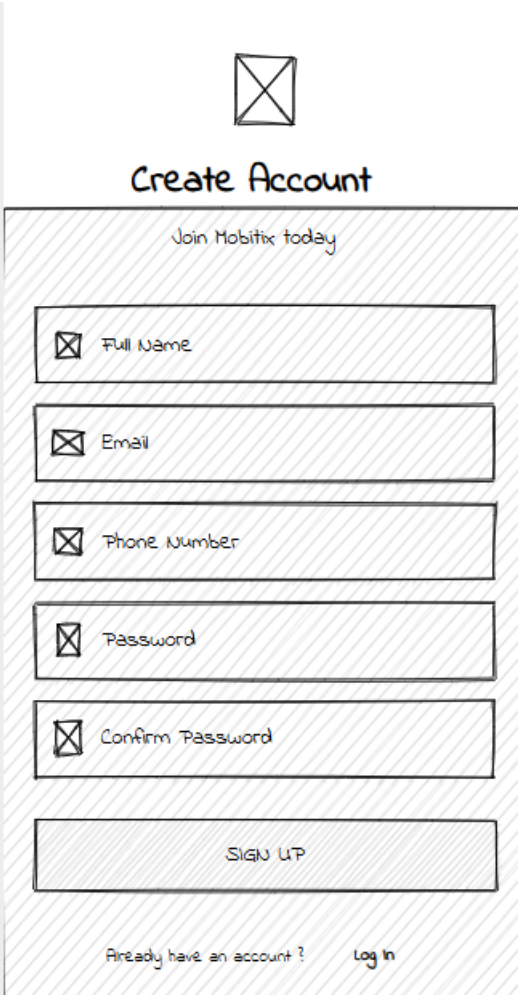


Figure 16 Login page wireframe

Displayed above is the Login page of the Mobitix mobile application. It features a clean, minimalist design with a log in form which contains fields to input the username and password, along with a “Forgot Password?” option. The Log in button provides access to the overall application. The link “Don’t have an account? Register” directs new users to the signup page. The wireframe highlights its usability with clear labels and a straightforward layout.

2. Sign Up Screen



Above is the wireframe for the Sign up page of the mobile application. It has a header called “Create Account”, and subtitle “Join Mobitix Today”. It includes form fields for full name, email. Phone number, password, and password confirmation in order to collect all the necessary user details. A “Sign up” button allows for user registration and a link “Already have an account?” navigates back to the log in page. The wireframe is consistent with the login page.

2. Home Screen & Navbar

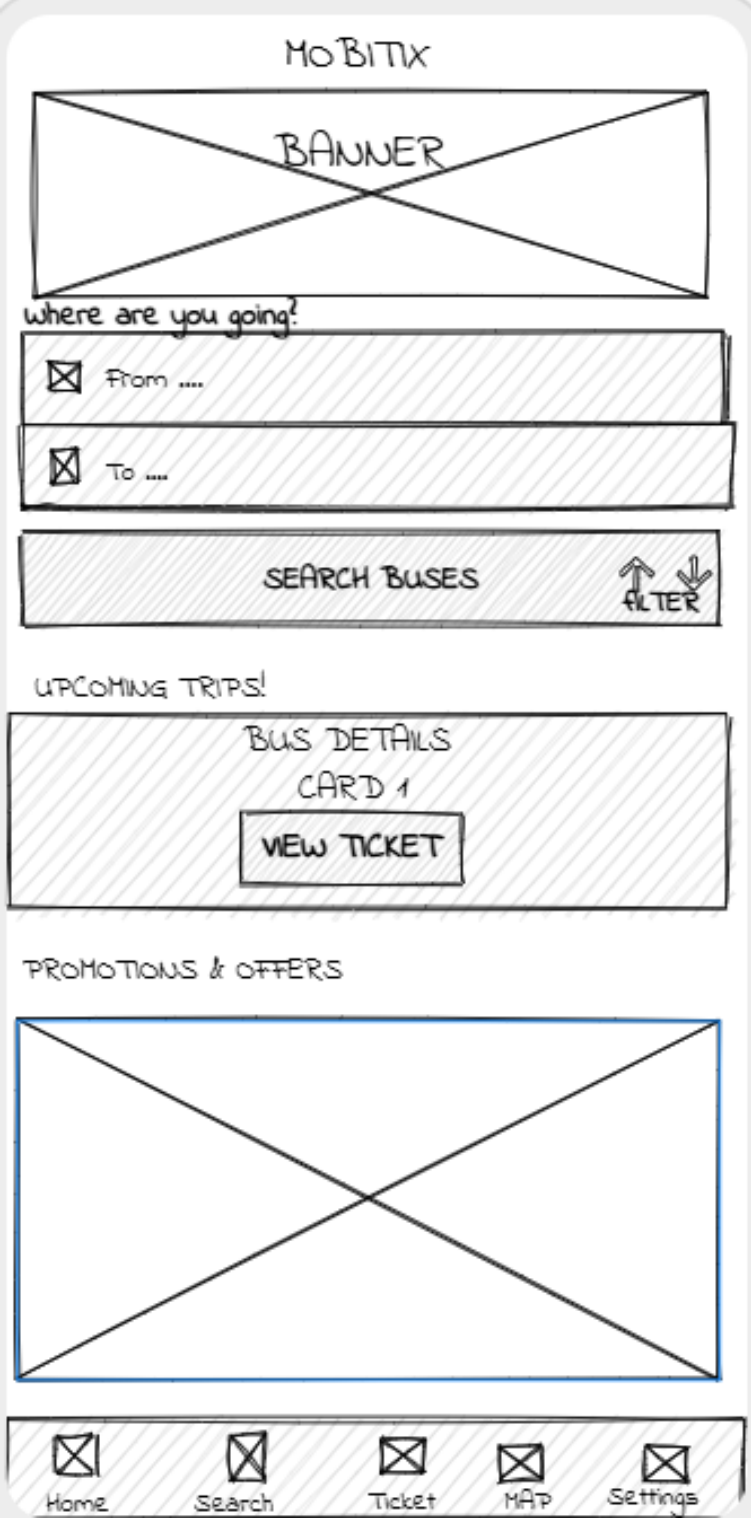


Figure 17 Home Screen Wireframe

The Home Page wireframe displays a clean and user-friendly interface for the Mobitix bus booking app. At the top, a banner provides insight to the app's function. There are two input fields "From…" and "To…" for route selection. Below, there is a "Search buses" options to search for buses, while a filter icon allows for further refinements. The middle section, "Upcoming Trips!", shows bus details in card format. (Eg: Card 1), providing quick access to recent bookings. At the bottom, a "Promotions & Offers" section is designed to provide highlights in order to improve user engagement. The final row contains a bottom navigation bar which is consistent across all the other pages, and it has five icons such as "Home, Search, Ticket, Map, and Settings."

3. Search screen wireframe

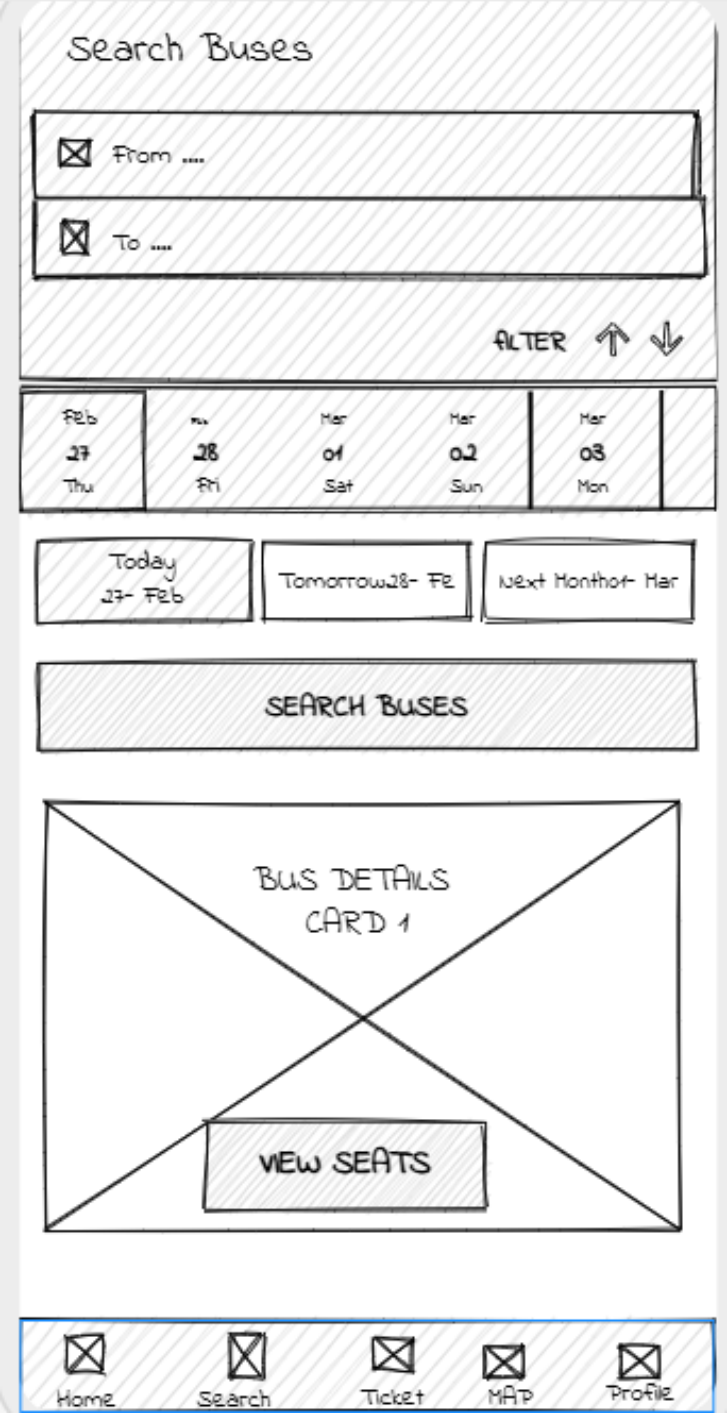


Figure 18 Search Screen Wireframe

The Search Page wireframe improves the booking process with an expanded search interface. It retains the “From…” and “To…” fields but adds a date selection panel with a horizontal scroll of date and quick-access buttons. A prominent “Search Buses” button initiates the search, while results appear as “Bus Details” cards, each with a “New SEATS” option for seat selection. The bottom navigation bar remains consistent with the Home page.

Bottom Navigation Bar

The wireframes feature a fixed bottom navigation bar with five tabs.

- Home – Returns to the main booking screen.
- Search – Opens the enhanced search interface
- Ticket – Accesses booked tickets
- Map – displays the Google Map
- Profile/Settings – Manages user account or app preferences.

4. Seat Selection Screen

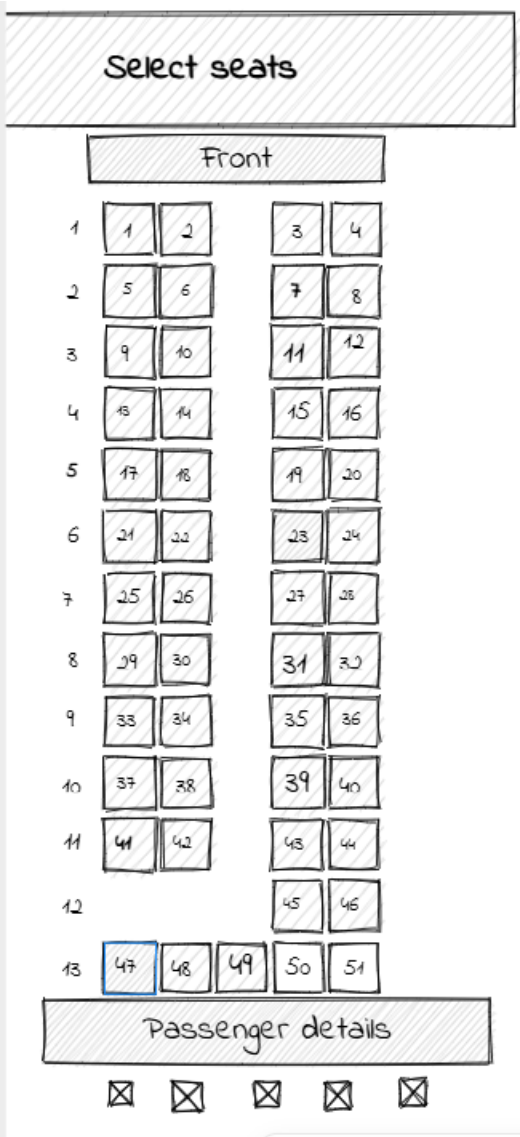


Figure 19 Seat Selection Screen wireframe 1

The Seat Selection wireframe dispays a bus layout with numbered seats arranged in rows, clearly marking available and selected seats. At the top, a “Front” label indicates the bus orientation, helping users visualize their position. Below the seat grid, a Passenger details section includes input fields for personal information. This design prioritizes clarity in seat selection while integrating necessary passenger information before payment

5. Seat Details

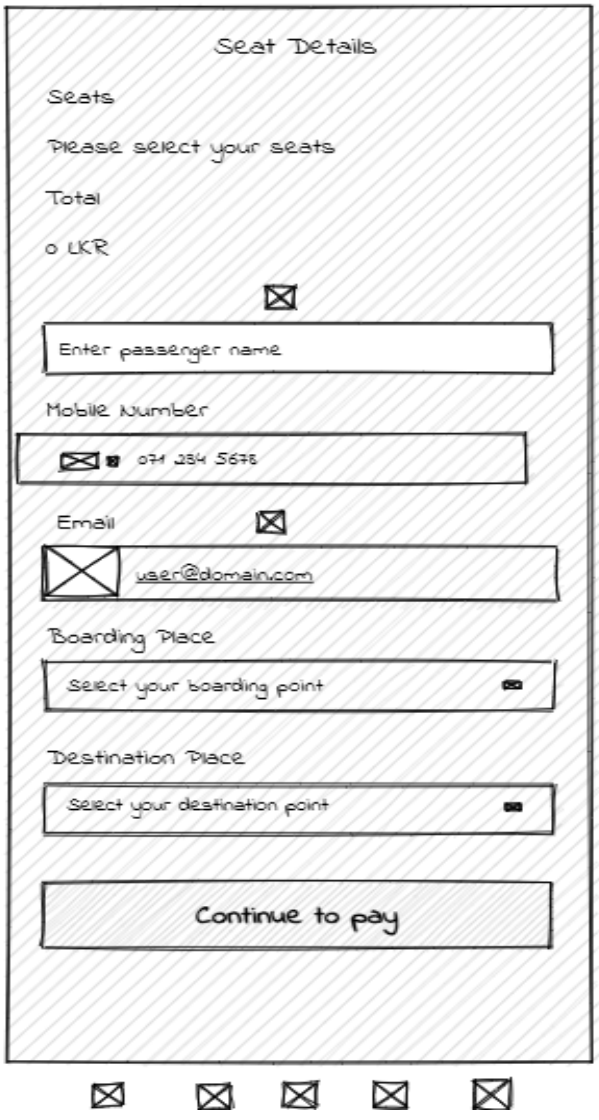


Figure 20 Seat Details screen

This wireframe represents the final step before payment in the seat booking process. It combines seat selection details with passenger information collection in a clean, linear layout. The Seat Selection header displays “Please Select your seats”. It shows the total price by dynamically

calculating the cost based on the seats chosen. Then the form fields to input the Name, Mobile Number, email field with the Boarding and destination dropdowns are given. A prominent "Continue to pay" button prompts the user to proceed to payment options. This wireframe ensures users can review selections and enter required details before finalizing payment, reducing errors in the booking process

6. Payment Screen

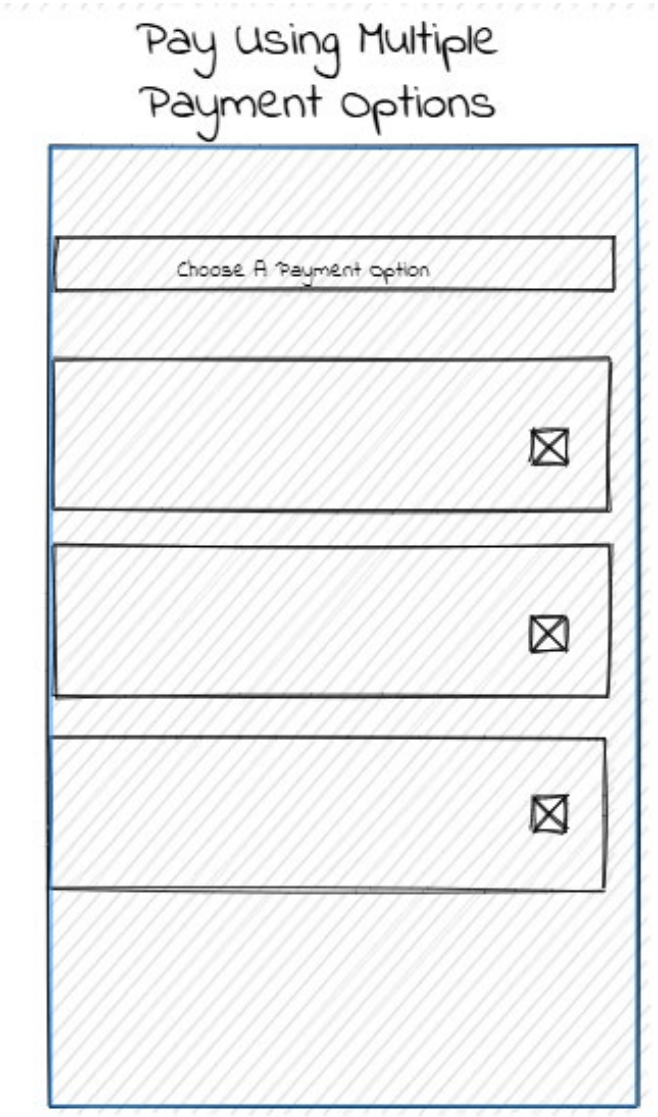


Figure 21 Payment Screen

This wireframe represents the payment selection screen, where users choose their preferred payment method to complete their bus ticketing. The cards display Genie, Koko, and eZcash as selectable payment options. Each option appears as a clickable card with its logo and a brief description. When a user clicks a payment method, they would be redirected to the respective

payment gateway. After successful payment, the user would return to the app to receive their booking confirmation.

7. Ticket Confirmation Screen

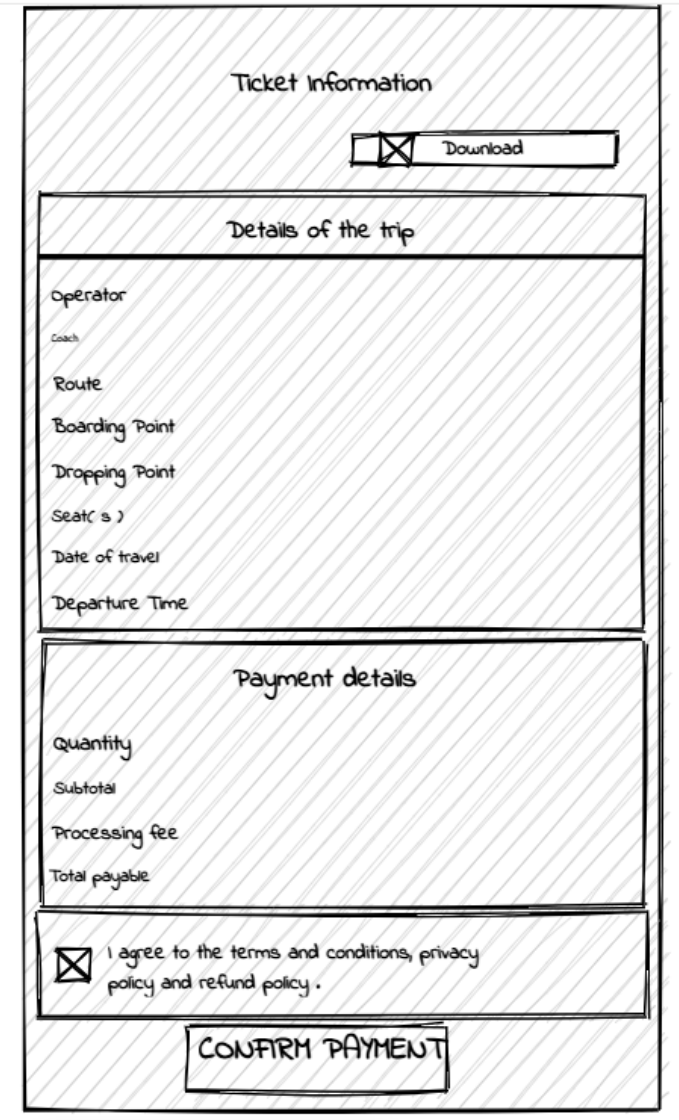


Figure 22 Ticket Confirmation wireframe

This wireframe represents the final booking confirmation screen, where users review their trip details before making payment. The title indicates the booking summary. The download button allows saving the ticket for offline access. The operator displays the bus service provider. The

date of travel confirms the journey date. Payment details shows the total amount and selected payment method. The terms & conditions checkbox makes the user agree to the terms and conditions, privacy policy and refund policy before proceeding. The confirm payment button is the final step to complete the transaction. Once clicked, it processes the payment and generates the ticket.

8. Settings Screen

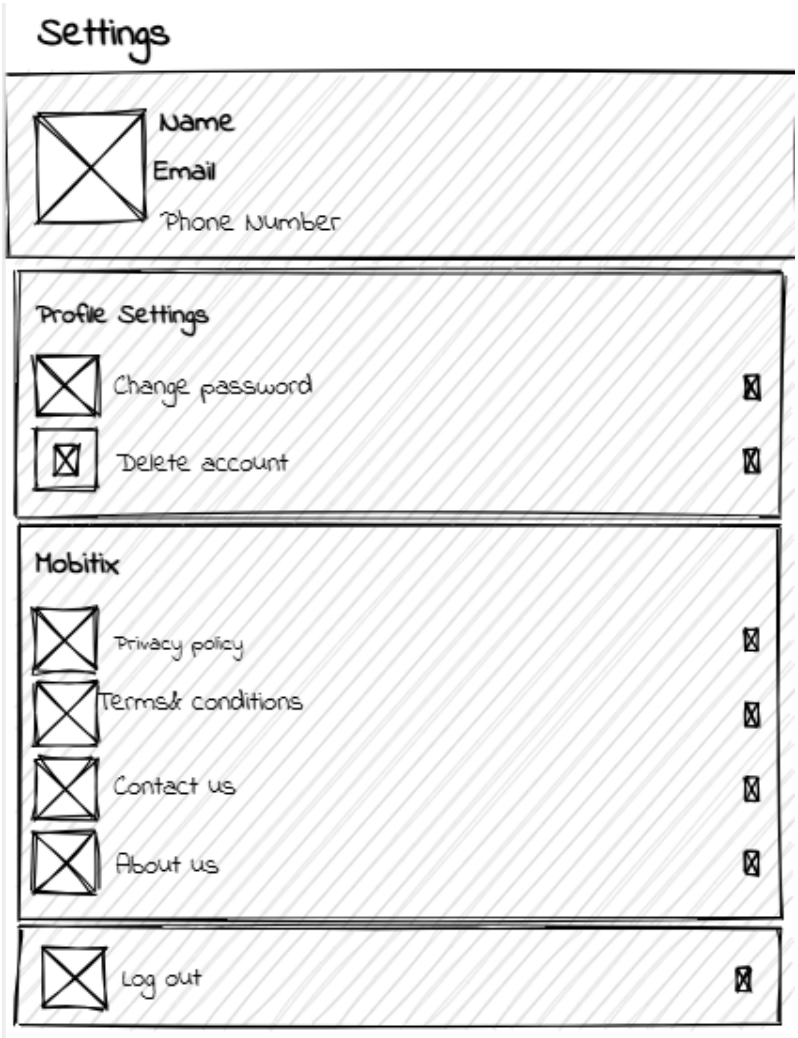


Figure 23 Settings Screen

This wireframe represents the user settings and account management interface for the Mobitix app. The User Information section displays editable fields for Name, Email, and Phone Number.

The change password option allows to update login credentials. The delete account section allows a user to delete their account from the application. The privacy policy and terms & conditions sections link to legal documents. Contact Us and About Us access to support and organization information. The log out button promptly allows a user to exit the account.

4 Product Implementation

This chapter discusses about the implementation of the Mobitix application, its development process of creating a user-friendly interface and integrating various functionalities that enhance the user experience. The prototype serves as a tangible representation of the design principles and objectives outlined in the initial project proposal. The implementation process is crucial as it provides insights into the methodologies employed and the challenges encountered throughout the development journey.

4.1 Software Development Libraries Used

The Mobitix app is developed using the Flutter framework, a powerful toolset that helps developers in creating natively compiled mobile, web and desktop apps from one codebase. Flutter's foundation is based on the dart programming language, which provides a reactive programming model that simplifies creating complex user interfaces. Such a cross-platform facility is particularly handy for the Mobitix application because it does not necessitate either platform to have a different version of the application or to build different codebases. The flutter environment not only accelerates development, but also ensures that the application would be able to leverage native performance as well as good appearances, an ideal fit for the Mobitix Project.

4.1.1 Google Maps Flutter Library.

The Mobitix application's integration of real-time map functions, made possible by the google_maps_flutter library, is one of its most notable features. With the help of this library's extensive feature set, developers can include Google Maps straight into their Flutter apps. Users may see local bus stops, see bus routes, and follow the current position of buses on a map due to this integration. To improve the user experience, the library offers a number of map elements that may be adjusted, such as markers, polylines, and polygons. The Mobitix application can give users current and precise geographic information by using the Google Maps API, which provides efficient travel planning. The choice of this library was due to its extensive documentation, community support and the reliability of Google Maps as a mapping service.

4.1.2 Location Package

The Mobitix app uses the location package, which gives it access to the device's GPS capabilities, to improve the user experience even more. For location-based services like tracking the user's current location and providing directions to local bus stations, this package is essential. The software may provide individualized experiences by using the device's GPS to recommend the closest bus routes depending on the user's location. Users may easily give or reject location access due to the location package's smooth handling of permission requests. Because it allows users to receive real-time updates and notifications on their bus rides, this feature is crucial to the Mobitix application's general usefulness and convenience.

4.1.3 HTTP Library

The http library promotes communication between the Mobitix application and the backend server by offering a quick and easy method of submitting network requests. This library is necessary for data submission and retrieval because it enables the application to communicate with the server for a number of purposes, including retrieving bus routes that are available, sending user data, and handling payments. The program may manage data activities without obstructing the user interface due to the http library's support for both synchronous and asynchronous requests. Because it enables the app to retrieve data in the background while users are still interacting with the UI, this feature is essential for maintaining the user experience. The library was selected for the Mobitix project because of its user-friendliness and strong community support.

4.1.4 The intl Library

The Mobitix program makes use of the intl package to improve the application's usability, especially with regard to date and time formatting. With the help of this library's extensive collection of translation and internationalization features, developers was able to format dates, and numbers according on the user's location. To make sure that users can readily comprehend the schedule of their bus trips, the Mobitix application formats departure and arrival times using the intl package. The program serves a wide range of users by offering translated date formats, which increases its usability and accessibility. Its strong functionality and the significance of providing information in a way that appeals to people from many cultural backgrounds were the driving forces behind the decision to include the international package.

4.1.5 RESTful API

A RESTful API is used by the Mobitix application to handle backend functions, which is necessary for providing smooth communication between the server and the mobile application. By following REST principles, this API enables the application to use common HTTP methods like GET, POST, PUT, and DELETE to carry out a variety of tasks, including user authentication, bus route querying, and ticket purchasing. For instance, the application sends a

GET request to the API when users look for available bus routes. The API then evaluates the request and provides the pertinent information in JSON format. In addition to improving the application by making upgrades and feature additions simple, its design provides safe transmission of data using token-based authentication.

4.1.6 Geolocator

The Geolocator package is integrated into the Mobitix program to improve its location-based features and provide consumers a more precise experience. With the help of this package, the application may use the device's GPS capabilities to track users' specific positions and provide features like location-based route recommendations and real-time bus tracking. In order to provide users with timely updates on bus arrivals and departures, the application uses Geolocator to determine the distances between the user's present location and neighboring bus stops. Furthermore, the Geolocator package makes it easier to handle location rights, allowing users to easily give or restrict access. This integration is a crucial part of the Mobitix system as it not only makes the program more user-friendly overall but also increases user trust in the authenticity of the information it provides.

4.2 functions and code structure

4.2.1 Backend structure

The backend of Mobitix is designed for the management of bus ticket bookings, user authentication, payment processing, and real-time bus location tracking. it is built using PHP and MySQL, using a structured approach to handle various functionalities/

Database Structure

The Mobitix backend relies on a MySQL database named mobitix. The database is structured to store information related to users, buses, tickets, payments, and bus locations.

Key Tables

- Users Table: Stores user information such as User ID, FullName, Email, PhoneNo, and Password.
- Buses Table: Contains details about the buses, including bus_id, bus_name, departure_time, arrival_time, and route.
- Tickets Table: Records ticket information, including ticket_id, passenger_name, mobile, email, seats, boarding, destination, total_amount, booking_date, travel_date, bus_id, and payment_method.
- Payments Table: Manages payment records with fields like id, payment_method, amount, reference_id, passenger_name, status, and created_at.
- Bus Locations Table: Tracks the real-time locations of buses with bus_id, latitude, and longitude

Database Connection

The database connection is established using the following code snippet in ‘db.php’

```php
<?php
$servername = "localhost";
$username = "root"; // Default XAMPP username
$password = ""; // Default XAMPP password
$dbname = "mobitix";

// Create connection
$conn = new mysqli($servername, $username, $password, $dbname);

// Check connection
if ($conn->connect_error) {
    die("Connection failed: " . $conn->connect_error);
}
?>
```

Figure 24 db.php

API Endpoints

The backend exposes several API endpoints to handle various operations. These endpoints are designed to be RESTful, allowing for easy integration with the frontend.

User Authentication

- **Login**: The ‘login.php’ file handles user login by verifying the email and password against the database. It returns user data upon successful authentication

```php
<?php
require_once 'config.php';

$data = json_decode(file_get_contents("php://input"));

if(!empty($data->email) && !empty($data->password)) {
    $email = $data->email;
    $password = $data->password;

    $query = "SELECT * FROM Users WHERE Email = ?";
    $stmt = $conn->prepare($query);
    $stmt->execute([$email]);

    if($stmt->rowCount() > 0) {
        $user = $stmt->fetch(PDO::FETCH_ASSOC);

        if(password_verify($password, $user['Password'])) {
            // Return user data except password
            unset($user['Password']);
            echo json_encode([
                "success" => true,
                "message" => "Login successful",
                "user" => $user
            ]);
        } else {
            echo json_encode(["success" => false, "message" => "Invalid password"]);
        }
    } else {
        echo json_encode(["success" => false, "message" => "Email not registered"]);
    }
} else {
    echo json_encode(["success" => false, "message" => "Email and password are required"]);
}
?>
```

Figure 25 login.php

Registration: The 'register.php' helps the users to create an account by providing their details. It checks for emails to prevent duplicates

```php
if(!empty($data->email) && !empty($data->password) && !empty($data->fullName) && !empty($data->phoneNo))
    $email = $data->email;
    $password = password_hash($data->password, PASSWORD_BCRYPT);
    $fullName = $data->fullName;
    $phoneNo = $data->phoneNo;

    // Check if email already exists
    $check = $conn->prepare("SELECT Email FROM Users WHERE Email = ?");
    $check->execute([$email]);

    if($check->rowCount() > 0) {
        echo json_encode(["success" => false, "message" => "Email already registered"]);
        exit();
    }

    // Insert new user
    $query = "INSERT INTO Users (FullName, Email, PhoneNo, Password) VALUES (?, ?, ?, ?)";
    $stmt = $conn->prepare($query);

    if($stmt->execute([$fullName, $email, $phoneNo, $password])) {
        echo json_encode(["success" => true, "message" => "Registration successful"]);
    } else {
        echo json_encode(["success" => false, "message" => "Registration failed"]);
    }
} else {
    echo json_encode(["success" => false, "message" => "All fields are required"]);
}
?>
```

Figure 26 registration.php

Ticket Management

Fetch buses : The ‘fetch_buses.php’ file retrieves available buses based on the departure and destination locations, as well as the selected date.

```
// Base query
$sql = "SELECT *,
        TIME(departure_time) as departure_time_only,
        TIME(arrival_time) as arrival_time_only
        FROM buses
        WHERE route LIKE '%$from%'
        AND route LIKE '%$to%'";
```

Figure 27 fetch_buses.php logic

Payment Processing

Payment handling : The payment.php file processes payments by inserting payment records into the payments table and simulating payment success or failure.

```
if ($method === 'POST') {
    $data = json_decode(file_get_contents('php://input'), true);

    // Validate required fields
    if (empty($data['payment_method']) || empty($data['amount']) || empty($data['reference_id'])) {
        http_response_code(400);
        echo json_encode(['success' => false, 'message' => 'Missing required fields']);
        exit();
    }

    try {
        // Simulate payment processing delay
        sleep(2);

        $success = rand(1, 100) <= 80;
        $status = $success ? 'success' : 'failed';

        // Insert payment record
        $stmt = $conn->prepare("INSERT INTO payments (
            payment_method, amount, reference_id, passenger_name, seats,
            boarding_point, destination, email, mobile, status
        ) VALUES (
            :payment_method, :amount, :reference_id, :passenger_name, :seats,
            :boarding_point, :destination, :email, :mobile, :status
        )");

        $stmt->execute([
            ':payment_method' => $data['payment_method'],
            ':amount' => $data['amount'],
            ':reference_id' => $data['reference_id'],
            ':passenger_name' => $data['passenger_name'] ?? '',
            ':seats' => $data['seats'] ?? '',
            ':boarding_point' => $data['boarding_point'] ?? '',
            ':destination' => $data['destination'] ?? '',
            ':email' => $data['email'] ?? '',
            ':mobile' => $data['mobile'] ?? '',
            ':status' => $status
        ]);
```

Figure 28 payment.php

Location Tracking

Update Location: The location.php file handles POST requests to update the real-time location of buses. It uses an INSERT ... ON DUPLICATE KEY UPDATE SQL statement to either insert a new location or update an existing one

```php
// Update or insert the bus location
$stmt = $conn->prepare("INSERT INTO bus_locations (bus_id, latitude, longitude) VALUES (?, ?, ?) ON DUPLICATE KEY UPDATE latitude=?, longitude=?");
$stmt->bind_param("sssss", $busId, $latitude, $longitude, $latitude, $longitude);
$stmt->execute();
$stmt->close();
echo json_encode(["status" => "success"]);
```

Figure 29 location.php

Session Management

The backend utilizes PHP sessions to manage user sessions, particularly during the OTP verification process. The `otp.php` file generates and verifies OTPs for user authentication.

```php
<?php
require_once 'config.php';

session_start();

// Debugging: Log session data
error_log("Session ID: " . session_id());
error_log("Current session: " . print_r($_SESSION, true));

header("Content-Type: application/json");

if ($_SERVER['REQUEST_METHOD'] === 'POST') {
    $input = file_get_contents("php://input");
    error_log("Received input: " . $input);
    $data = json_decode($input);

    if (json_last_error() !== JSON_ERROR_NONE) {
        echo json_encode(["success" => false, "message" => "Invalid JSON data"]);
        exit();
    }

    if (empty($data->email)) {
        echo json_encode(["success" => false, "message" => "Email is required"]);
        exit();
    }

    // Generate OTP request
    if (!isset($data->user_otp)) {
        $otp = rand(100000, 999999);
        $_SESSION['otp'] = $otp;
        $_SESSION['otp_email'] = $data->email;
        $_SESSION['otp_time'] = time();

        error_log("Generated OTP: $otp for email: " . $data->email);
        error_log("Updated session: " . print_r($_SESSION, true));

        echo json_encode([
            "success" => true,
            "otp" => $otp,
            "message" => "OTP generated"
        ]);
        exit();
    }
```

Figure 30 otp.php

Error handling

The backend implements error handling mechanisms to ensure smooth operation. For instance, in the fetch_buses.php file, it checks for valid date formats and returns appropriate error messages.

```php
// Add date filter if provided
if (!empty($date)) {
    // Validate date format
    if (preg_match('/^\d{4}-\d{2}-\d{2}$/', $date)) {
        $sql .= " AND DATE(departure_time) = '$date'";
    } else {
        http_response_code(400);
        die(json_encode(['error' => 'Invalid date format. Use YYYY-MM-DD']));
    }
}
```

Figure 31 Error Handling

As such, the backend structure of Mobitix is robust and well-structured for efficient management of bus ticketing operations. It focuses on user authentication, ticket management, payment processing, and real-time location tracking. the backend is designed to provide a smooth experience for users. the use of PHP and MySQL makes the system more scalable and maintainable, making it suitable for future enhancements.

4.2.2 frontend structure

The frontend of Mobitix is developed using Flutter. It provides a user-friendly interface for booking bus tickets, managing user profiles, and viewing ticket details.

Project Structure

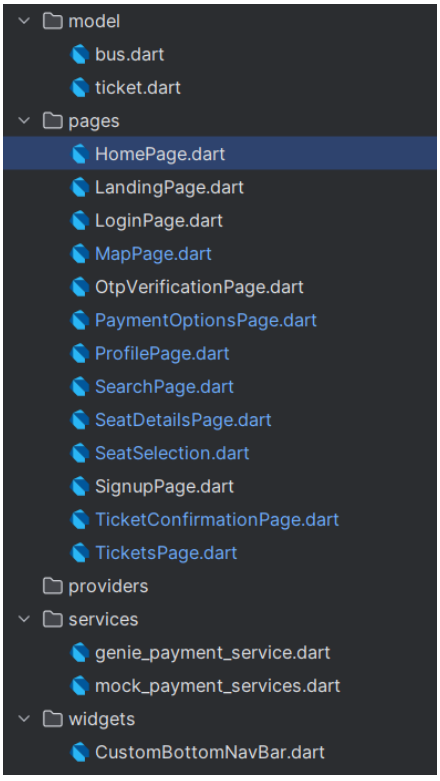


Figure 32 Project Structure

The Mobitix frontend project is organized into several directories, each serving a specific purpose.

- Pages - Contains all the main screens of the application, such as HomePage, SearchPage, TicketsPage, ProfilePage, and more.
- Model - Contains data models like Ticket and Bus that represent the data structures used in the application.
- Services - Contains service classes for handling payment processing and API interactions, such as genie_payment_service.dart and mock_payment_services.dart.
- Widgets - Contains reusable UI components, such as CustomBottomNavBar.dart.

Key Screens and Components

Landing Page

The “Landing Page” is the entry point of the application, which provides options for the users to sign up or log in.

```dart
import 'package:flutter/material.dart';
import 'LoginPage.dart';
import 'SignupPage.dart';

class LandingPage extends StatefulWidget {
  const LandingPage({super.key});

  @override
  State<LandingPage> createState() => _LandingPageState();
}

class _LandingPageState extends State<LandingPage> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      body: SingleChildScrollView(
        child: Container(
          margin: EdgeInsets.symmetric(horizontal: 10.0, vertical: 10.0),
          child: Column(
            children: [
              Material(
                elevation: 3.0,
                borderRadius: BorderRadius.circular(30),
                child: ClipRRect(
                  borderRadius: BorderRadius.circular(30),
                  child: Image.asset(
                    "images/icon/app_icon1.png",
                    width: MediaQuery.of(context).size.width,
                    height: MediaQuery.of(context).size.height / 2,
                    fit: BoxFit.cover,
```

Figure 33 landingpage.dart

Home Page

The Homepage provides users with options to search for buses and view upcoming trips.

```dart
import 'package:flutter/material.dart';
import 'SearchPage.dart';
import 'MapPage.dart';
import 'package:mobitix/widgets/CustomBottomNavBar.dart';
import 'ProfilePage.dart';
import 'TicketsPage.dart';

class HomePage extends StatelessWidget {
  const HomePage({super.key});

  @override
  Widget build(BuildContext context) {
    final Color primaryColor = Colors.blueAccent;
    final Color backgroundColor = Colors.grey.shade100;
    final TextEditingController fromController = TextEditingController();
    final TextEditingController toController = TextEditingController();

    return Scaffold(
      backgroundColor: backgroundColor,
      appBar: AppBar(
        title: Text("Mobitix", style: TextStyle(fontWeight: FontWeight.bold)),
        centerTitle: true,
        backgroundColor: primaryColor,
        elevation: 0,
      ),  // AppBar
      body: SingleChildScrollView(
        padding: const EdgeInsets.all(16),
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.start,
          children: [
```

Figure 34 homepage.dart 1

```
Text("Where are you going?",
    style: TextStyle(fontSize: 18, fontWeight: FontWeight.w600)),  // Text
SizedBox(height: 12),

_buildRoundedInputField(
    "From...", Icons.location_on, fromController),
SizedBox(height: 12),
_buildRoundedInputField("To...", Icons.flag, toController),
SizedBox(height: 12),

// Search Buses Button + Filter
Row(
  children: [
    Expanded(
      child: ElevatedButton(
        onPressed: () {
          if (fromController.text.isEmpty ||
              toController.text.isEmpty) {
            ScaffoldMessenger.of(context).showSnackBar(
              SnackBar(
                  content: Text(
                      "Please enter both departure and destination")),  // Text, SnackBar
            );
            return;
          }
```

Figure 35 homepage.dart 2

```
Widget _buildTripCard(BuildContext context, String route, String details) {
  return Card(
    shape: RoundedRectangleBorder(borderRadius: BorderRadius.circular(12)),
    elevation: 2,
    margin: EdgeInsets.only(bottom: 12),
    child: ListTile(
      leading: Icon(Icons.directions_bus, color: Colors.blueAccent),
      title: Text(route, style: TextStyle(fontWeight: FontWeight.w600)),
      subtitle: Text(details),
      trailing: ElevatedButton(
        onPressed: () {
          Navigator.push(
            context,
            MaterialPageRoute(
              builder: (context) => SearchPage(
                initialFrom: route.split("→")[0].trim(),
                initialTo: route.split("→")[1].trim(),
              ),  // SearchPage
            ),  // MaterialPageRoute
          );
        },
        style: ElevatedButton.styleFrom(
          backgroundColor: Colors.green,
          shape:
              RoundedRectangleBorder(borderRadius: BorderRadius.circular(8)),
        ),
```

Figure 36 homepage.dart 3

Search page

In this page, users can search for available buses based on their departure and destination

```dart
class SearchPage extends StatefulWidget {
  final String? initialFrom;
  final String? initialTo;

  const SearchPage({this.initialFrom, this.initialTo, Key? key})
      : super(key: key);

  @override
  _SearchPageState createState() => _SearchPageState();
}

class _SearchPageState extends State<SearchPage> {
  late final TextEditingController fromController;
  late final TextEditingController toController;
  List<Bus> buses = [];
  bool isLoading = false;
  bool hasSearched = false;
  DateTime? selectedDate;
  List<DateTime> availableDates = [];
  final DateFormat dateFormat = DateFormat('MMM');
  final DateFormat dayFormat = DateFormat('d');
  final DateFormat weekdayFormat = DateFormat('E');

  @override
  void initState() {
    super.initState();
    fromController = TextEditingController(text: widget.initialFrom ?? '');
    toController = TextEditingController(text: widget.initialTo ?? '');
    _initializeDates();
```

Figure 37 searchpage.dart 1

```
Widget _buildInputField(
    String hint, IconData icon, TextEditingController controller) {
  return TextField(
    controller: controller,
    decoration: InputDecoration(
      hintText: hint,
      prefixIcon: Icon(icon),
      filled: true,
      fillColor: Colors.white,
      contentPadding: EdgeInsets.symmetric(vertical: 16),
      border: OutlineInputBorder(borderRadius: BorderRadius.circular(12)),
      enabledBorder: OutlineInputBorder(
        borderRadius: BorderRadius.circular(12),
        borderSide: BorderSide(color: Colors.blueGrey.shade100),
      ),  // OutlineInputBorder
      focusedBorder: OutlineInputBorder(
        borderRadius: BorderRadius.circular(12),
        borderSide: BorderSide(color: Colors.blueAccent),
      ),  // OutlineInputBorder
    ),  // InputDecoration
  );  // TextField
}
```

Figure 38 searchpage.dart 2

Ticket Management

The ticketspage displays the user's upcoming trips and trip history.

```
child: Padding(
  padding: const EdgeInsets.all(16),
  child: Column(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: [
      Row(
        mainAxisAlignment: MainAxisAlignment.spaceBetween,
        children: [
          Text(
            '${ticket.boarding} → ${ticket.destination}',
            style: const TextStyle(
              fontSize: 18,
              fontWeight: FontWeight.bold,
            ),  // TextStyle
          ),  // Text
          Chip(
            label: Text(
              ticket.status.toUpperCase(),
              style: const TextStyle(color: Colors.white),
            ),  // Text
            backgroundColor: ticket.status == 'upcoming'
                ? Colors.blue
                : ticket.status == 'completed'
                ? Colors.green
                : Colors.red,
          ),  // Chip
```

Figure 39 ticketspage.dart 1

```dart
Widget build(BuildContext context) {
  final upcomingTrips = tickets.where((t) => t.status == 'upcoming').toList();
  final tripHistory = tickets.where((t) => t.status != 'upcoming').toList();

  return DefaultTabController(
    length: 2,
    child: Scaffold(
      appBar: AppBar(
        title: const Text('My Tickets'),
        centerTitle: true,
        bottom: const TabBar(
          tabs: [
            Tab(text: 'Upcoming Trips'),
            Tab(text: 'Trip History'),
          ],
        ),  // TabBar
      ),  // AppBar
      body: TabBarView(
        children: [
          _buildTicketList(upcomingTrips),
          _buildTicketList(tripHistory),
        ],
      ),  // TabBarView
```

Figure 40 ticketspage.dart 2

Payment Options Page

In this page, the users can select their preferred payment method.

```dart
class PaymentOptionsPage extends StatelessWidget {
  final int totalAmount;
  final List<String> seats;
  final String passengerName;
  final String mobile;
  final String email;
  final String boarding;
  final String destination;

  const PaymentOptionsPage({
    Key? key,
    required this.totalAmount,
    required this.seats,
    required this.passengerName,
    required this.mobile,
    required this.email,
    required this.boarding,
    required this.destination,
  }) : super(key: key);
```

Figure 41 paymentsoptionpage.dart 1

```dart
  Widget _buildPaymentOption(
      BuildContext context, String method, IconData icon, Color color) {
    return InkWell(
      onTap: () {
        if (method == "Genie") {
          _processGeniePayment(context);
        } else if (method == "Koko") {
          _processKokoPayment(context);
        } else if (method == "eZcash") {
          _processEzCashPayment(context);
        } else {
          _processOnboardPayment(context);
        }
      },
```

Figure 42 paymentoptionspage.dart 2

Navigation

The application uses a custom bottom navigation bar to provide navigation between different pages. This component is reused across various screens to maintain a consistent user experience.

```
// Bottom Navigation Bar for the Home page
bottomNavigationBar: CustomBottomNavBar(
  currentIndex: 0,
  onTap: (index) {
    if (index == 0) {

    } else if (index == 1) {
      Navigator.pushReplacement(
          context, MaterialPageRoute(builder: (_) => SearchPage()));
    } else if (index == 2) {
      Navigator.pushReplacement(
        context,
        MaterialPageRoute(builder: (_) => const TicketsPage()),
      );
    } else if (index == 3) {
      Navigator.pushReplacement(
          context, MaterialPageRoute(builder: (_) => Mappage()));
    } else if (index == 4) {
      Navigator.pushReplacement(
          context, MaterialPageRoute(builder: (_) => ProfilePage()));
    }
  },
),  // CustomBottomNavBar
```

Figure 43 navigation bar

4.3 Coding Principle

The Mobitix application has several coding principles and best practices that improves its overall performance. Below is a detailed explanation of these principles.

1. Separation of Concerns

One of the fundamental principles applied in Mobitix is the Separation of Concerns. This principle advocates for dividing a program into distinct sections, each handling a specific aspect of the application. In Mobitix, this is achieved through the organization of files and directories.

Implementation:

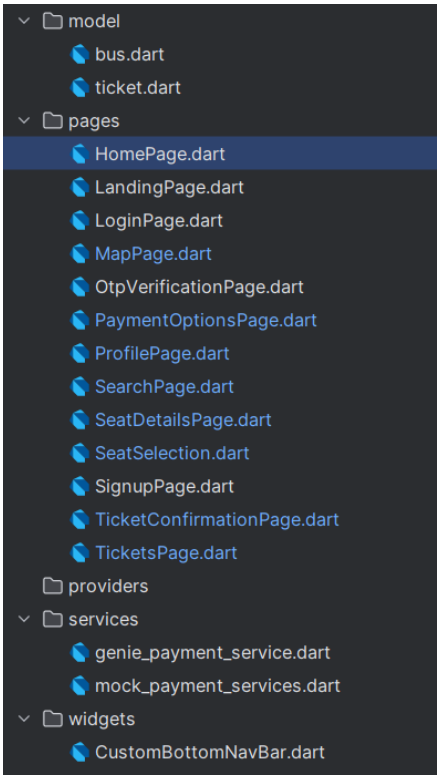


Figure 44 Separation of concerns

The application is divided into various directories such as pages, model, services, and widgets. Each directory contains files that focus on a specific functionality, making it easier to manage and update the code.

This method helps developers to work on individual components without affecting the entire application, thereby improving collaboration and reducing the risk of introducing bugs.

2. Code Reusability

The principle of Code Reusability is prominently featured in Mobitix, allowing developers to write code once and reuse it across different parts of the application. This not only reduces redundancy but also enhances maintainability.

Implementation:

```dart
import 'package:flutter/material.dart';

class CustomBottomNavBar extends StatelessWidget {
  final int currentIndex;
  final Function(int) onTap;

  const CustomBottomNavBar({
    Key? key,
    required this.currentIndex,
    required this.onTap,
  }) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return BottomNavigationBar(
      currentIndex: currentIndex,
      onTap: onTap,
      selectedItemColor: Colors.blue,
      unselectedItemColor: Colors.grey[600],
      backgroundColor: Colors.white,
      items: const [
        BottomNavigationBarItem(icon: Icon(Icons.home), label: "Home"),
        BottomNavigationBarItem(icon: Icon(Icons.search), label: "Search"),
        BottomNavigationBarItem(icon: Icon(Icons.confirmation_number), label: "Ticket"),
        BottomNavigationBarItem(icon: Icon(Icons.map), label: "Map"),
        BottomNavigationBarItem(icon: Icon(Icons.person), label: "Profile"),
      ],
    );  // BottomNavigationBar
  }
}
```

Figure 45 reusable widgets

Reusable Widgets: The application utilizes reusable widgets, such as the CustomBottomNavBar, which can be used across multiple pages. This ensures a consistent look and feel throughout the application.

By encapsulating the navigation bar logic within a single widget, any changes made to the navigation bar will automatically reflect across all pages that utilize it.

3. Consistent Naming Conventions

Mobitix adheres to consistent naming conventions throughout its codebase, which enhances readability and maintainability. This principle helps developers understand the purpose of variables, functions, and classes at a glance.

Implementation:

```dart
class SearchPage extends StatefulWidget {
  final String? initialFrom;
  final String? initialTo;

  const SearchPage({this.initialFrom, this.initialTo, Key? key})
      : super(key: key);

  @override
  _SearchPageState createState() => _SearchPageState();
}

class _SearchPageState extends State<SearchPage> {
  late final TextEditingController fromController;
  late final TextEditingController toController;
  List<Bus> buses = [];
  bool isLoading = false;
  bool hasSearched = false;
  DateTime? selectedDate;
  List<DateTime> availableDates = [];
  final DateFormat dateFormat = DateFormat('MMM');
  final DateFormat dayFormat = DateFormat('d');
  final DateFormat weekdayFormat = DateFormat('E');

  @override
  void initState() {
    super.initState();
    fromController = TextEditingController(text: widget.initialFrom ?? '');
    toController = TextEditingController(text: widget.initialTo ?? '');
    _initializeDates();
```

Figure 46 Descriptive names

Descriptive Names: Variables and functions are named descriptively to convey their purpose clearly. For example, in the SearchPage, the input fields are named fromController and toController, indicating their role in capturing user input for departure and destination locations.

This practice not only aids in understanding the code but also facilitates collaboration among developers, as they can quickly grasp the functionality without extensive documentation.

4. Error Handling

Effective error handling is crucial for building robust applications. Mobitix implements error handling mechanisms to manage exceptions and provide meaningful feedback to users.

Implementation:

```php
// Add date filter if provided
if (!empty($date)) {
    // Validate date format
    if (preg_match('/^\d{4}-\d{2}-\d{2}$/', $date)) {
        $sql .= " AND DATE(departure_time) = '$date'";
    } else {
        http_response_code(400);
        die(json_encode(['error' => 'Invalid date format. Use YYYY-MM-DD']));
    }
}
```

Figure 47 Error Handling

Try-Catch Blocks: The application uses try-catch blocks to handle potential errors during API calls and database interactions. For instance, in the fetch_buses.php file, error handling is implemented to manage invalid date formats.

This approach ensures that users receive clear error messages, enhancing the overall user experience and preventing the application from crashing unexpectedly.

5. State Management

Mobitix employs state management principles to manage the state of the application effectively. This is particularly important in Flutter, where the UI is reactive and needs to update based on user interactions.

Implementation:

```
Future<void> _performSearch() async {
  final from = fromController.text.trim();
  final to = toController.text.trim();

  if (from.isEmpty || to.isEmpty) {
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(content: Text("Please enter both departure and destination")),
    );
    return;
  }
  if (selectedDate == null) {
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(content: Text("Please select a date")),
    );
    return;
  }

  try {
    final results = await fetchBuses(from, to, selectedDate!);
    setState(() {
      buses = results;
      hasSearched = true;
    });
```

Figure 48 State management

setState() Method: The application utilizes the setState() method to trigger UI updates when the state changes. For example, in the SearchPage, the state is updated when the user performs a search, allowing the UI to reflect the search results dynamically.

This principle ensures that the UI remains in sync with the underlying data, providing a smooth and responsive user experience.

4.4 Critical Discussion of coding issues

In the development of the Mobitix application, several coding issues and challenges were encountered that could potentially impact the application's performance, maintainability, and user experience. This section critically discusses these issues, providing insights into their implications and potential solutions. The discussion is structured around common coding pitfalls, architectural decisions, and best practices that were either adhered to or overlooked during the development process.

1. Inconsistent Error Handling

One of the most significant coding issues observed in the Mobitix application is inconsistent error handling across different components. While some parts of the application, such as the API endpoints, implement robust error handling mechanisms, others lack sufficient checks and balances. For instance, in the fetch_buses.php file, the code effectively checks for valid date formats and returns appropriate error messages when the format is incorrect. However, similar checks are not consistently applied across all API endpoints.

This inconsistency can lead to a poor user experience, as users may encounter unhandled exceptions or vague error messages that do not provide clear guidance on how to resolve the issue. For example, if a user attempts to log in with an incorrect password, the application should provide a specific error message indicating that the password is invalid, rather than a generic "login failed" message. This lack of specificity can frustrate users and lead to confusion, ultimately affecting user retention and satisfaction.

To address this issue, it is essential to establish a standardized error handling framework across the application. This framework should include consistent error messages, logging mechanisms for debugging, and user-friendly feedback that guides users on how to rectify their mistakes. Implementing a centralized error handling strategy can significantly enhance the application's robustness and improve the overall user experience.

2. Lack of Input Validation

Another critical issue identified in the Mobitix application is the lack of comprehensive input validation. While some input fields, such as those for email and password during user registration and login, are validated, others do not enforce strict validation rules. For instance, in the SearchPage, the input fields for departure and destination locations do not have checks to ensure that users do not submit empty strings or invalid characters.

This lack of validation can lead to unexpected behavior in the application, such as failed API calls or incorrect search results. For example, if a user submits a search request with empty fields, the application may attempt to query the database with invalid parameters, resulting in errors or empty responses. Furthermore, without proper validation, the application is vulnerable

to security risks, such as SQL injection attacks, where malicious users can manipulate input fields to execute harmful SQL commands.

To mitigate these risks, it is crucial to implement thorough input validation across all user input fields. This validation should include checks for empty values, character limits, and format requirements (e.g., ensuring that email addresses are in the correct format). Additionally, employing client-side validation in conjunction with server-side validation can enhance security and improve the user experience by providing immediate feedback on input errors.

3. Inefficient State Management
State management is a critical aspect of any application, particularly in a reactive framework like Flutter. In Mobitix, while the setState() method is used to manage state changes, there are instances where state management could be optimized. For example, in the SearchPage, the application fetches bus data and updates the UI using setState(), which can lead to unnecessary rebuilds of the entire widget tree.

This inefficient state management can result in performance issues, especially as the application scales and the number of users increases. Frequent and unnecessary rebuilds can lead to sluggish performance, particularly on lower-end devices, where resources are limited. Additionally, as the complexity of the application grows, relying solely on setState() can make the codebase harder to maintain and understand.

To improve state management, it is advisable to adopt more advanced state management solutions, such as Provider, Riverpod, or Bloc. These solutions allow for more granular control over state changes, enabling developers to rebuild only the necessary parts of the UI when the state changes. By implementing a more efficient state management strategy, the Mobitix application can enhance its performance and maintainability, providing a smoother user experience.

4. Code Duplication and Lack of Reusability
Throughout the Mobitix codebase, instances of code duplication were observed, particularly in the way certain UI components and logic were implemented across different pages. For example, similar input fields and buttons were recreated in multiple places without encapsulating them into reusable widgets. This not only increases the size of the codebase but also makes it more challenging to maintain.

When code duplication occurs, any changes made to a specific component must be replicated across all instances, increasing the risk of introducing bugs and inconsistencies. For instance, if a developer needs to update the styling of a button used in multiple places, they must remember to make the same change in every location, which can lead to oversight and errors.

To address this issue, it is essential to adopt a modular approach to UI development by creating reusable widgets for common components. For example, the input fields and buttons used throughout the application can be encapsulated into separate widget classes, allowing for consistent styling and behavior. This practice not only reduces code duplication but also enhances maintainability, as changes can be made in one place and automatically reflected throughout the application.

5. Inadequate Documentation and Comments
Effective documentation is a crucial aspect of software development that is often overlooked. In the Mobitix application, there is a noticeable lack of comments and documentation throughout the codebase. While the code may be clear to the original developers, it can be challenging for new developers or contributors to understand the logic and flow of the application without adequate explanations.

Inadequate documentation can lead to misunderstandings, increased onboarding time for new developers, and difficulties in maintaining the codebase. For example, if a developer encounters a complex function without comments explaining its purpose or logic, they may spend unnecessary time deciphering the code instead of focusing on implementing new features or fixing bugs.

To improve the situation, it is essential to adopt a culture of documentation within the development team. This includes writing clear comments for complex logic, providing high-level overviews of modules, and maintaining up-to-date README files that explain the overall architecture and setup of the application. By prioritizing documentation, the Mobitix team can enhance collaboration, streamline onboarding, and ensure that the codebase remains accessible and maintainable over time.

As such, the Mobitix application, while functional and user-friendly, faces several coding issues that could hinder its long-term success. By addressing inconsistencies in error handling, implementing comprehensive input validation, optimizing state management, reducing code duplication, and enhancing documentation, the user experience can be significantly improved. These improvements will not only benefit the current development cycle but also lay a solid foundation for future enhancements and scalability, ensuring that Mobitix remains a competitive player in the bus ticketing market.

4.5 Implemented prototype

The implemented prototype of the Mobitix application showcases a comprehensive and user-friendly interface designed to provide bus ticket booking and management. Each screen is meticulously crafted to allow users to navigate through various functionalities easily. Below is a detailed explanation of each screen in the prototype, highlighting its purpose and key features.

1. Landing Page

The Landing Page serves as the initial entry-point for users when they launch the Mobitix application. It is designed to create a welcoming atmosphere, featuring a visually appealing banner that captures the essence of the brand. The page prominently displays a brief introduction to the application, emphasizing its purpose of simplifying bus ticket bookings. Users are presented with two primary options: to sign up for a new account or to log in to an existing one. This straightforward approach ensures that users can quickly access the functionalities they need, setting a positive tone for their experience with the application.

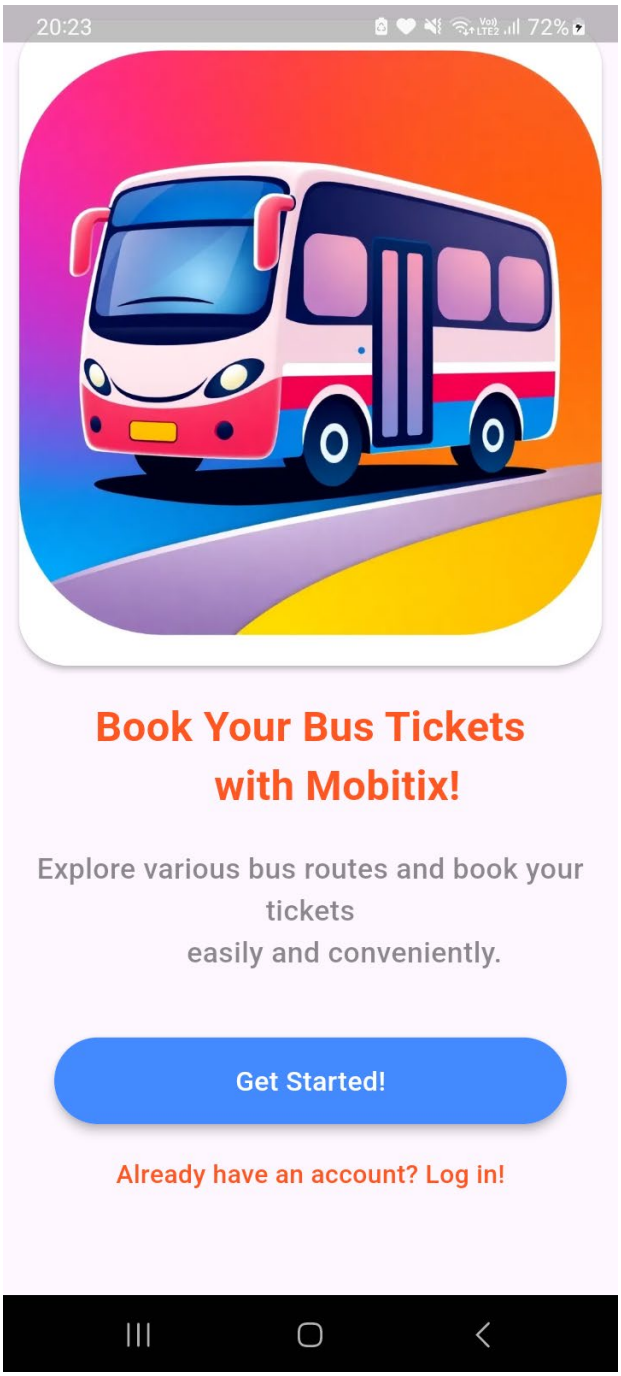


Figure 49 Landing Page

2. Home Page

Upon logging in, users are directed to the Home Page, which acts as the central hub of the application. This page is designed to provide users with quick access to essential features, including a search bar for finding buses and a section displaying upcoming trips. The layout is intuitive, with rounded input fields for users to enter their departure and destination locations. Additionally, the Home Page highlights any ongoing promotions or offers, encouraging users to take advantage of special deals. The consistent bottom navigation bar allows users to switch between different sections of the app effortlessly, enhancing overall usability.

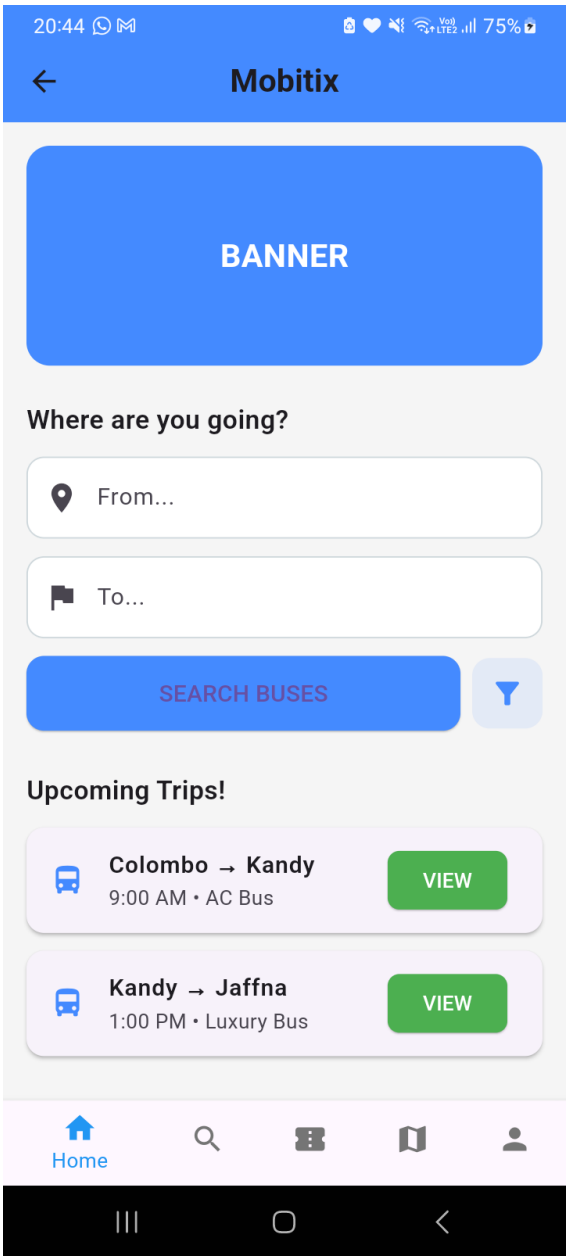


Figure 50 Home page

3. Search Page

The Search Page is where users can actively search for available buses based on their travel preferences. This page features input fields for users to specify their departure and destination locations, along with a date selection option to choose their travel date. The design is clean and user-friendly, ensuring that users can easily input their information. Once the search is initiated, the page displays a list of available buses, complete with details such as bus names, routes, and timings. This functionality is crucial for users to find suitable travel options that meet their needs.

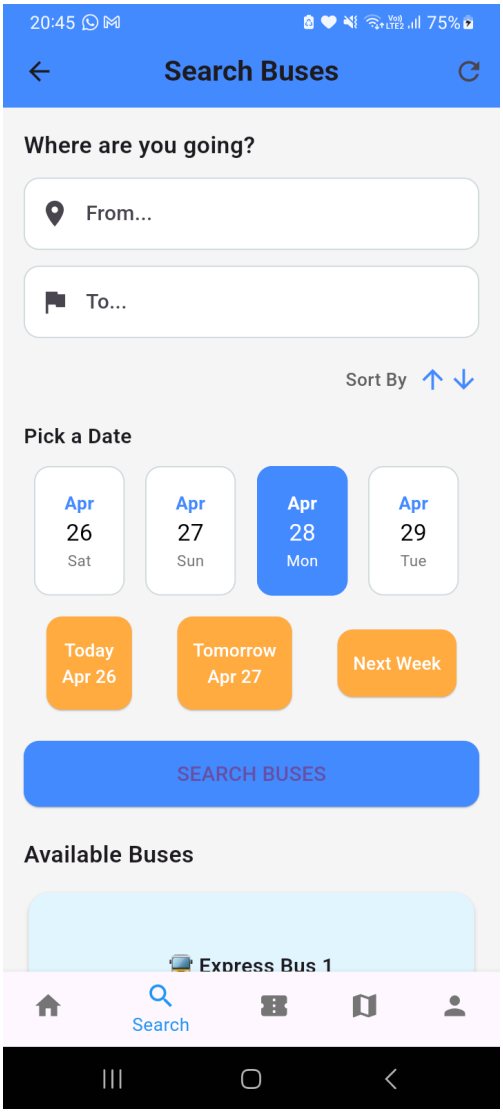


Figure 51 Search Page

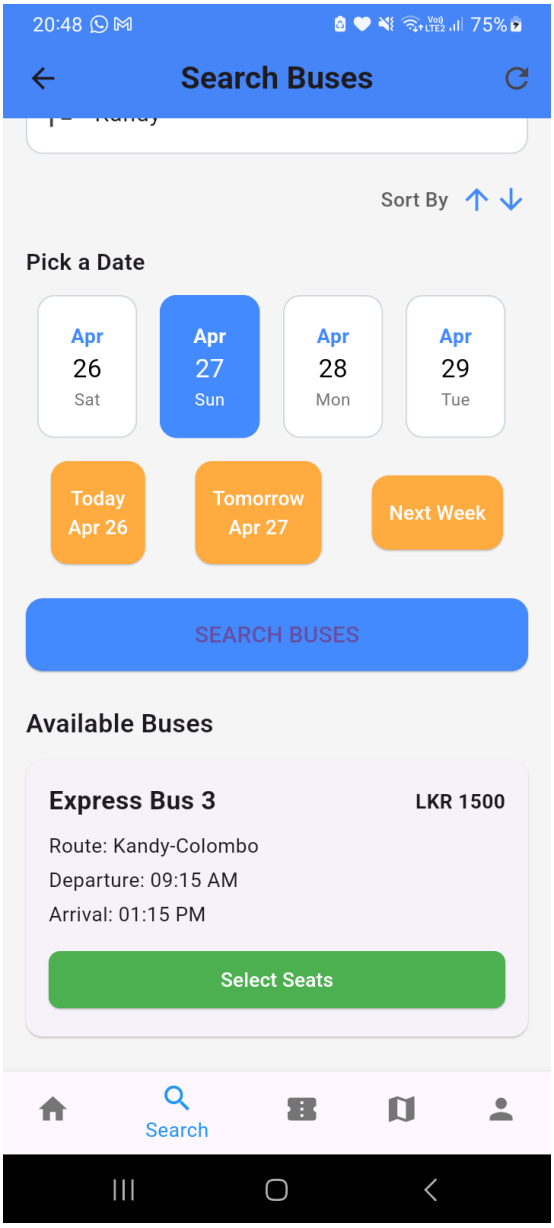


Figure 52 available buses for the given date

4. Tickets Page

The Tickets Page provides users with an overview of their booked tickets, allowing them to manage their travel plans effectively. This page features a tabbed interface that separates upcoming trips from trip history, making it easy for users to navigate between their current and past journeys. Each ticket is displayed in a card format, showcasing essential details such as the route, travel date, seat numbers, and payment status. Users can tap on a ticket to view more detailed information, including a downloadable e-ticket, enhancing the user experience and providing easy access to important travel documents.

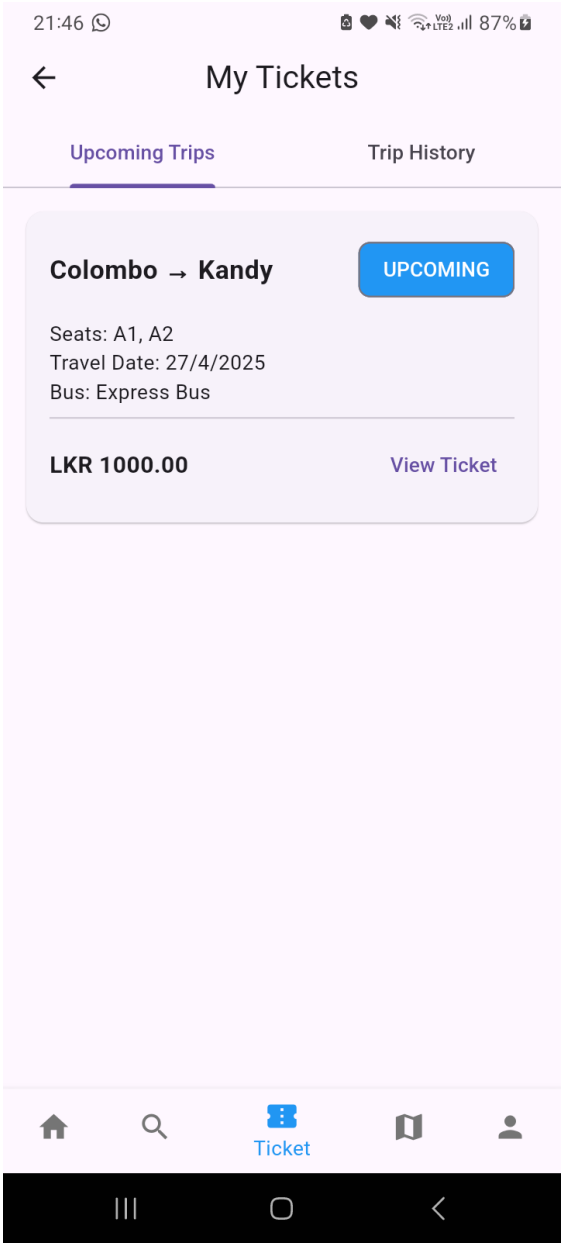


Figure 53 My tickets page

5. Payment Options Page

The Payment Options Page is a critical component of the ticket booking process, where users finalize their purchases by selecting a payment method. This page presents a trip summary that includes key information about the journey, such as passenger details, selected seats, and the total amount due. Users can choose from various payment options, including Genie, Koko, eZcash, or Onboard Payment, providing flexibility in how they complete their transactions. The design is straightforward, guiding users through the payment process and ensuring a smooth transaction experience.

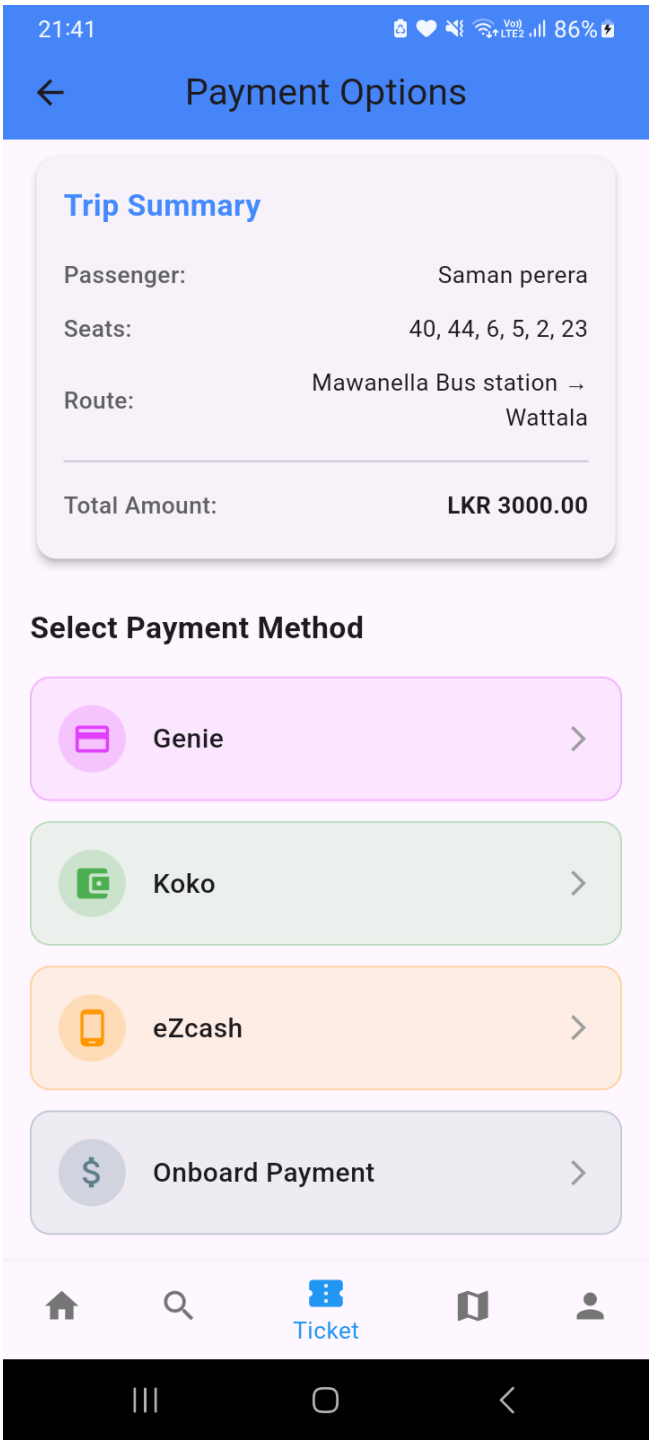


Figure 54 Payments Options page

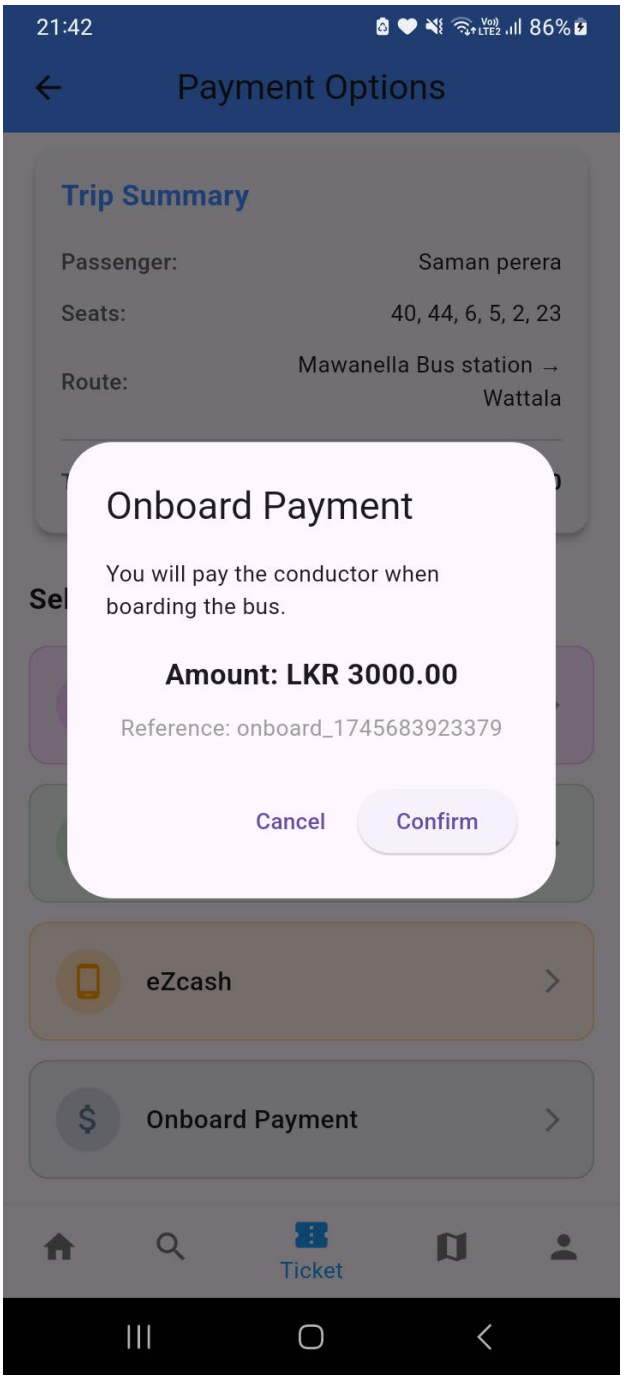


Figure 55 Payments Confirmation Window

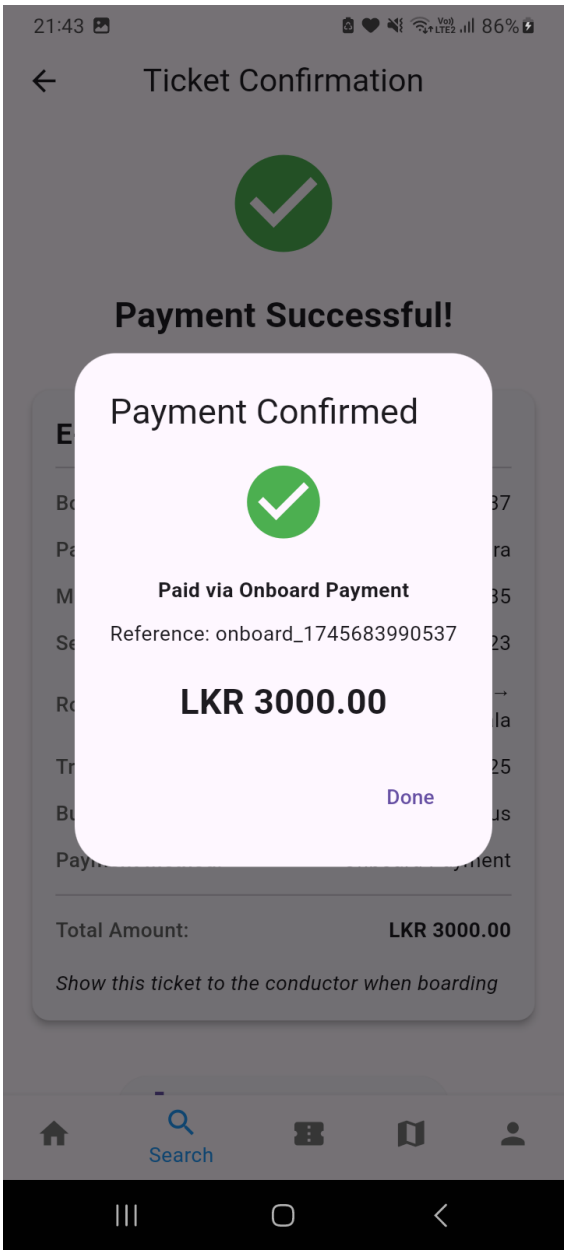


Figure 56 Payment confirmation

6. Profile Page

The Profile Page allows users to manage their personal information and account settings. This page displays the user's full name, email, phone number, and account creation date, enabling users to verify their details easily. Additionally, the Profile Page provides options for users to change their password, verify their account, or delete their account, giving them control over their account settings. The consistent bottom navigation bar remains present, allowing users to switch back to other sections of the app seamlessly.

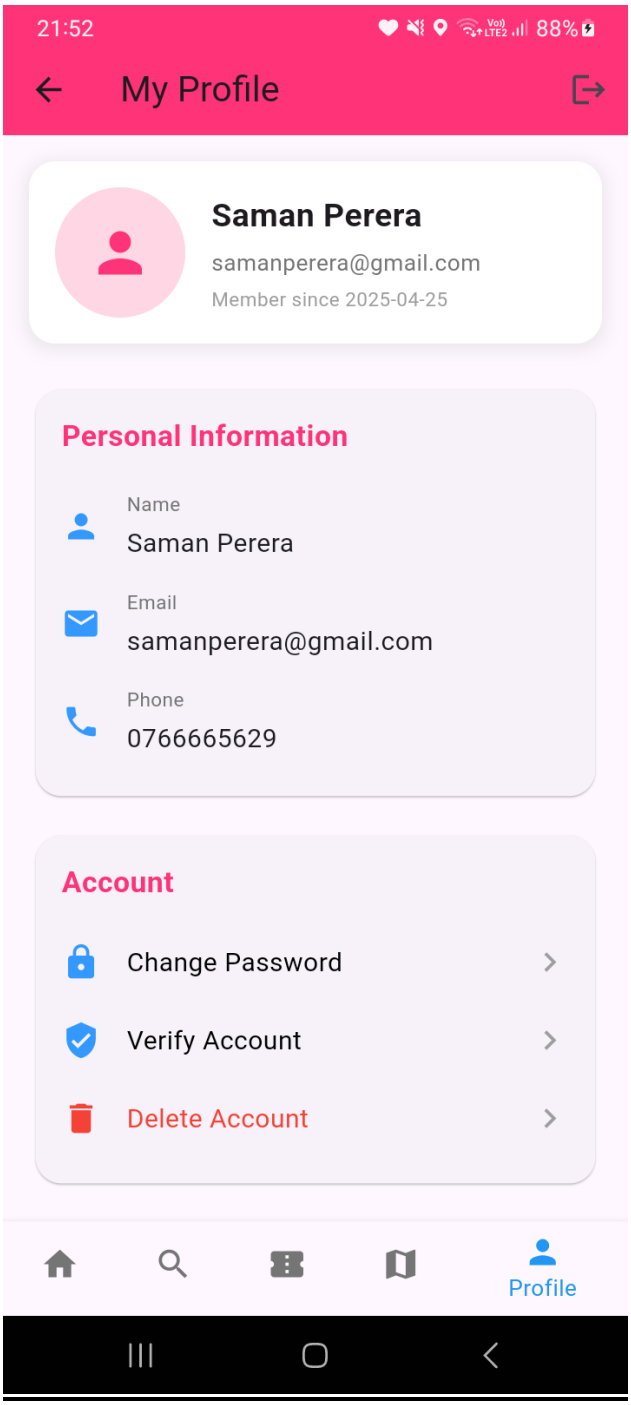


Figure 57 Profile Page

7. Seat Selection Page

The Seat Selection Page is a pivotal component of the ticket booking process, allowing users to choose their preferred seats for the selected bus journey. This page features a grid layout that visually represents the seating arrangement of the bus, making it easy for users to identify available, selected, and occupied seats. Each seat is represented as a clickable item, enabling users to select or deselect seats as needed. The page also includes a clear indication of the total

price based on the number of selected seats, ensuring transparency in pricing. Additionally, users are provided with a legend that explains the color coding of seats, enhancing usability. Once users have made their selections, they can proceed to the next step in the booking process.

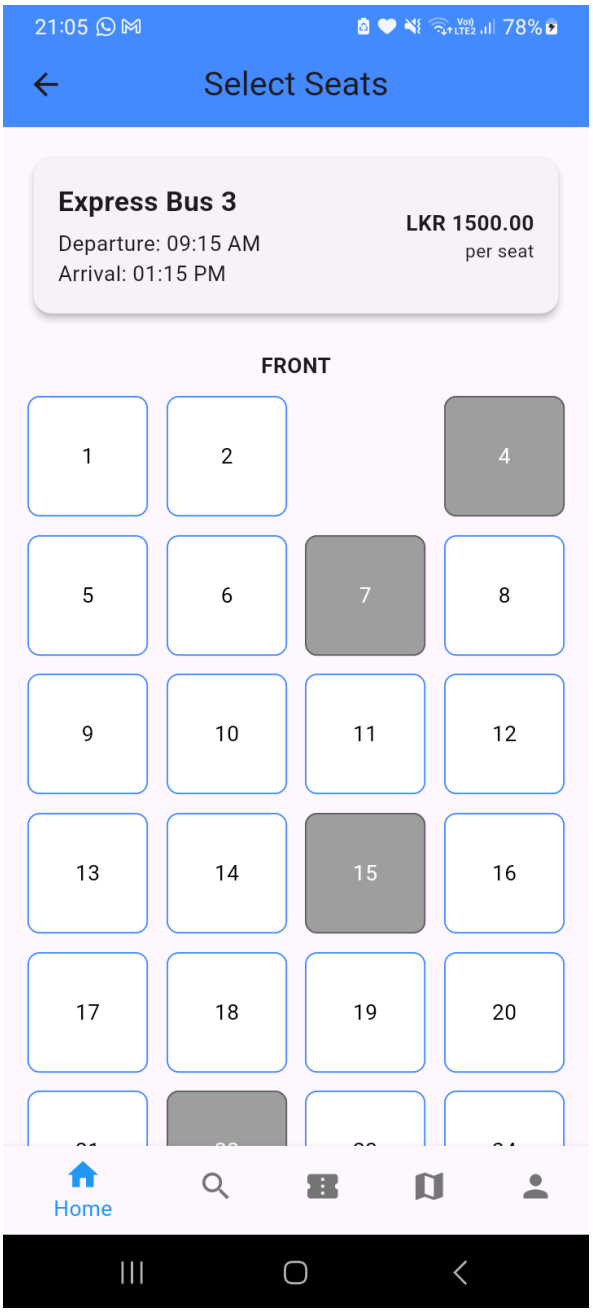


Figure 58 Seat Selection page 1

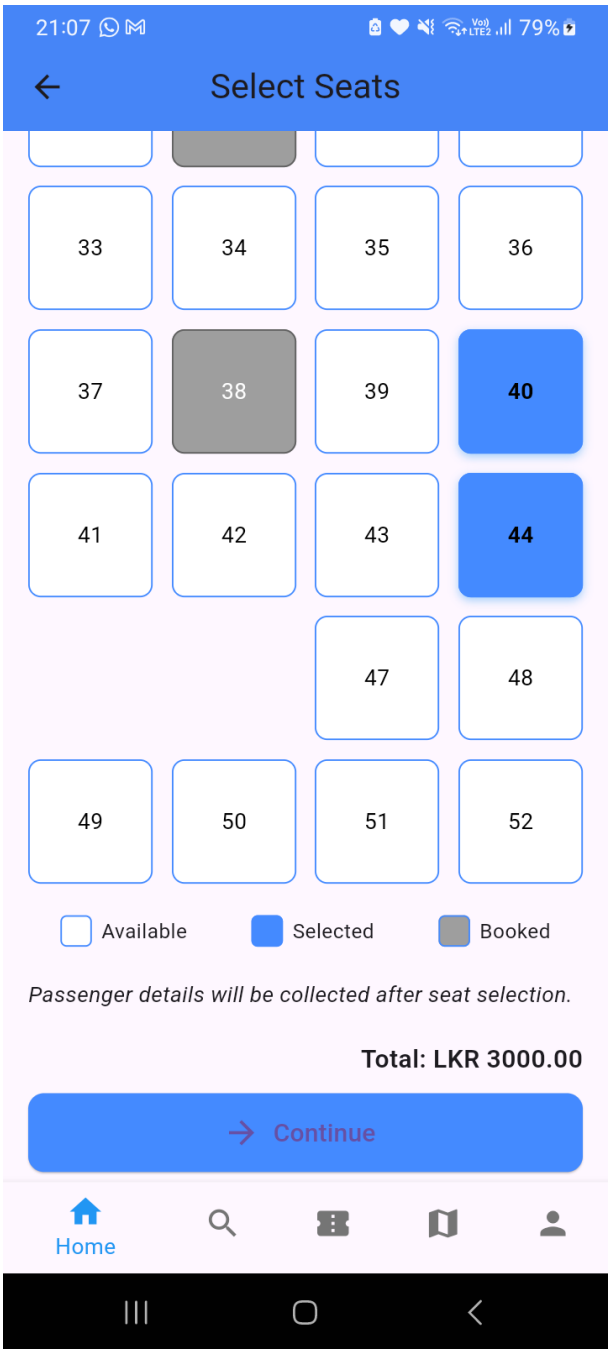


Figure 59 Seat Selection page 2

8. Seat Details Page

The Seat Details Page follows the seat selection process and is designed to collect essential passenger information before finalizing the booking. This page prompts users to enter their full name, mobile number, email address, boarding point, and destination point. The layout is organized and user-friendly, with input fields clearly labeled for easy navigation. Users can review their selected seats and the total amount due, ensuring they have all the necessary

information before proceeding to payment. The page also features a prominent button that allows users to continue to the payment options, streamlining the booking process.

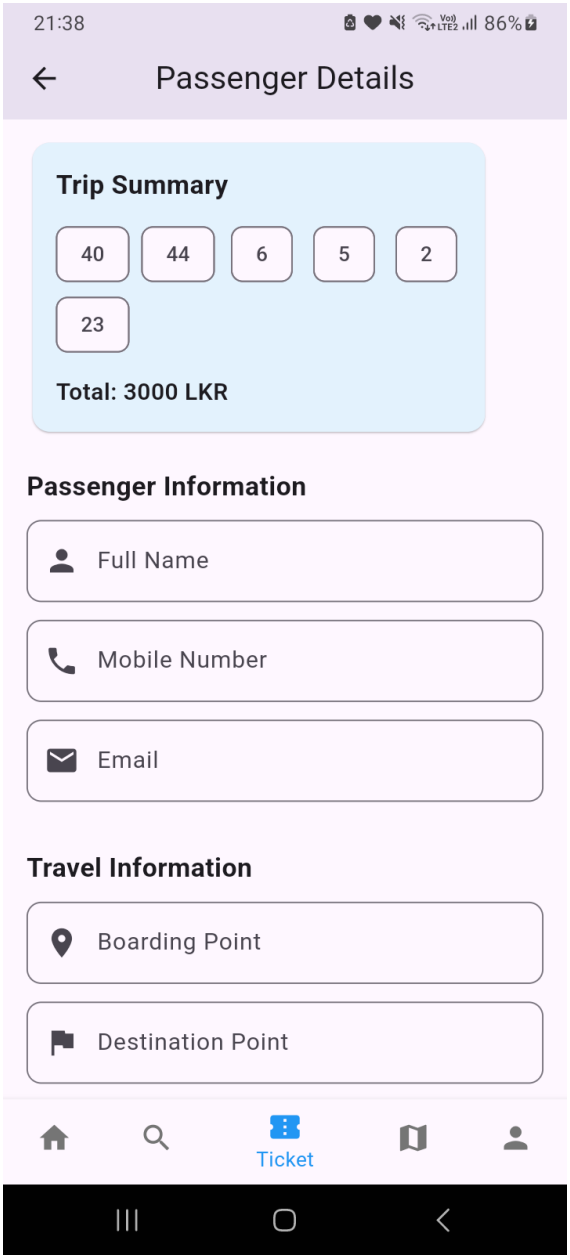


Figure 60 seat details page 1

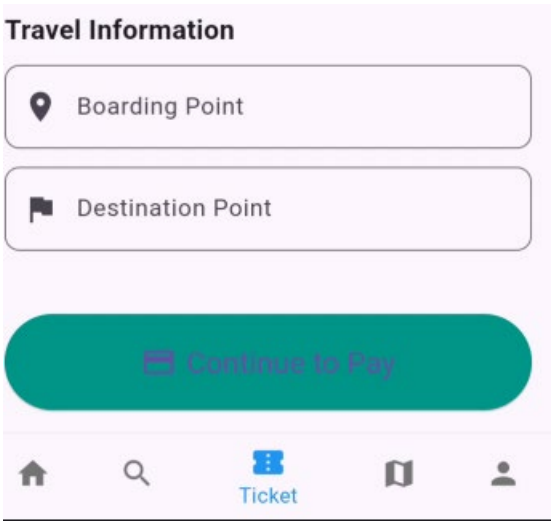


Figure 61 Seat Details page 2

9. Login Page

The Login Page is where users can access their existing accounts. This page features input fields for users to enter their email and password, with clear labels and icons to guide them. The design is clean and straightforward, focusing on the essential elements needed for authentication. Additionally, the page includes a "Forgot Password?" link, providing users with a way to recover their accounts if they forget their login credentials. Upon successful login, users are redirected to the Home Page, where they can begin their journey with Mobitix.

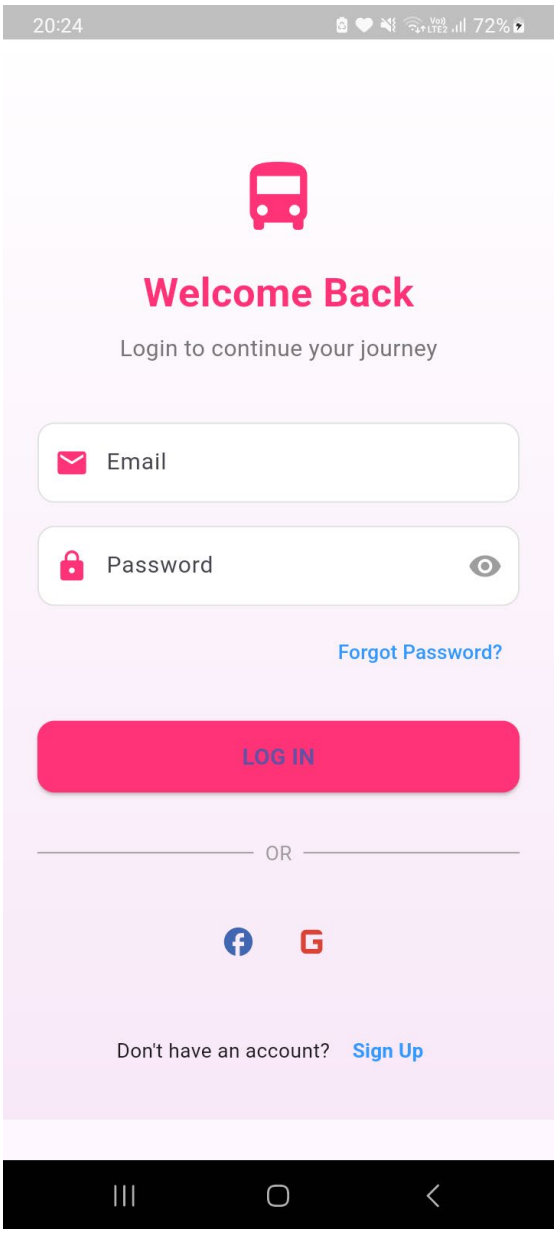


Figure 62 Log in page

10. Signup Page

The Signup Page allows new users to create an account with Mobitix. This page includes input fields for users to enter their full name, email address, phone number, and password, ensuring that all necessary information is collected for account creation. The design emphasizes clarity, with each field clearly labeled and accompanied by appropriate icons. Additionally, the page includes validation checks to ensure that users provide valid information, such as matching passwords. Once the signup process is completed, users are directed to the OTP Verification Page to confirm their email address, enhancing security and account integrity.

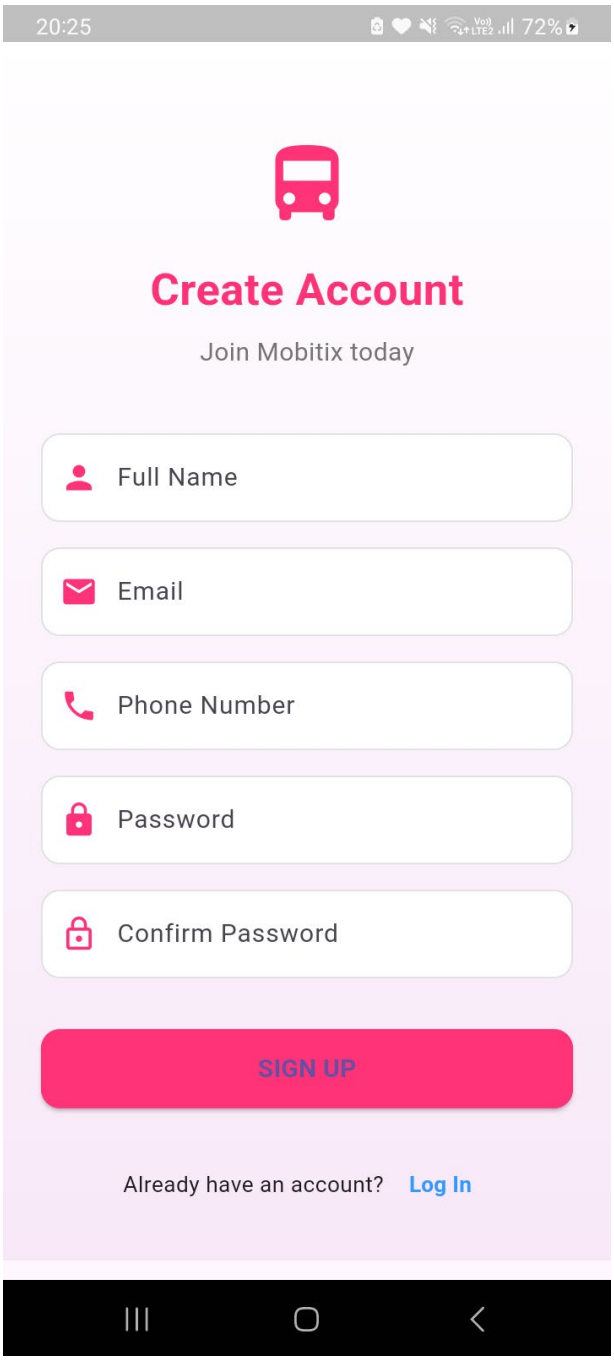


Figure 63 Signup page

11. OTP Verification Page

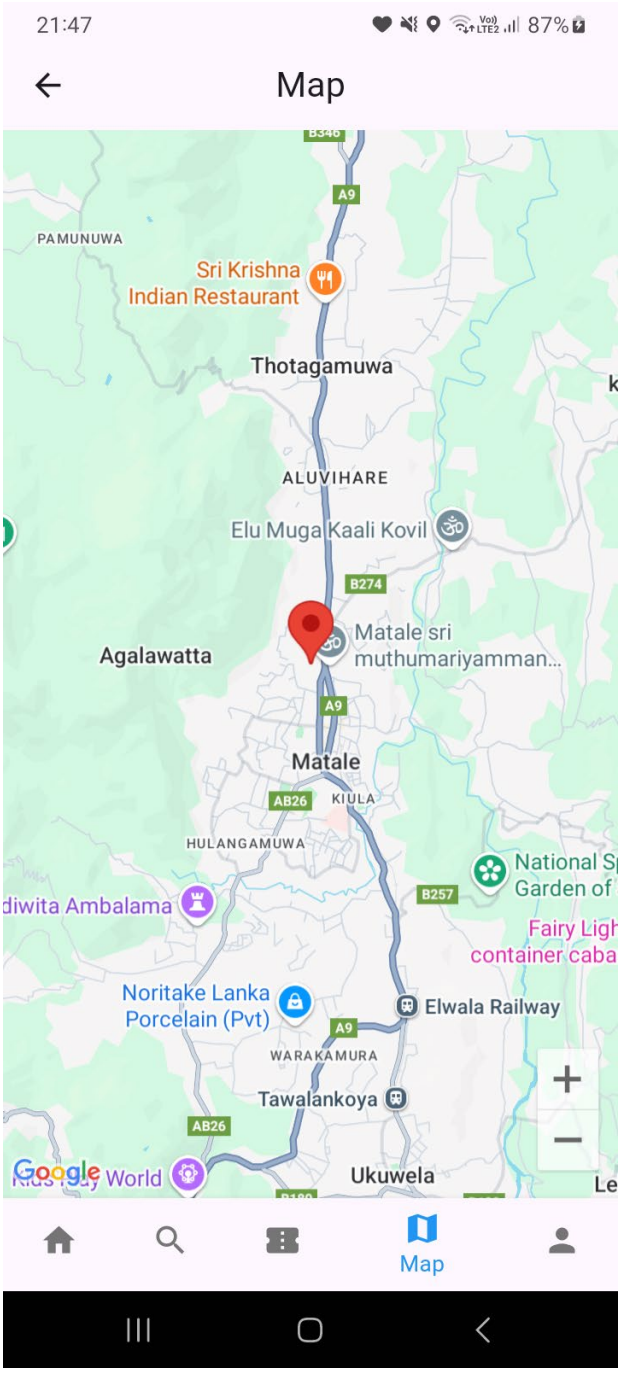
The OTP Verification Page is a crucial step in the account creation process, where users must verify their email addresses by entering a one-time password (OTP) sent to their email. This page features six input fields for users to enter the OTP, with a clear instruction indicating where the code was sent. The design is user-friendly, allowing users to navigate between fields easily. Additionally, the page includes a "Resend OTP" button, enabling users to request a new code if

they do not receive the original. Upon successful verification, users are redirected to the Home Page, completing the account setup process.

12. Map Page

The Map Page is an essential feature of the Mobitix application, providing users with real-time tracking of bus locations. This page utilizes Google Maps to display the current location of the user as well as the locations of buses on their respective routes. The interface is designed to be

interactive and user-friendly, allowing users to visualize their journey and see the proximity of buses in real-time. Markers on the map indicate the locations of buses, and users can tap on these markers to view additional information about each bus, such as its route and estimated arrival time. The Map Page not only enhances the user experience by providing valuable information but also instills confidence in users by allowing them to track their buses and plan their journeys accordingly. The integration of location services ensures that users receive accurate and timely updates, making the Mobitix application a reliable tool for bus travel.



13. Ticket Confirmation Page

The Ticket Confirmation Page is the final step in the booking process, providing users with a comprehensive overview of their ticket details after a successful payment. This page features a visually appealing layout that includes a confirmation message, indicating that the payment was successful. Users can view their e-ticket, which contains essential information such as the booking ID, passenger name, mobile number, selected seats, route, travel date, and payment method. The design emphasizes clarity, ensuring that users can easily understand their ticket details at a glance. Additionally, the page includes an option to download the ticket as a PDF, allowing users to save a copy for their records. This functionality not only enhances user satisfaction but also provides a tangible document that users can present when boarding the bus. The Ticket Confirmation Page serves as a reassuring conclusion to the booking process, confirming that users have successfully secured their travel plans with Mobitix.

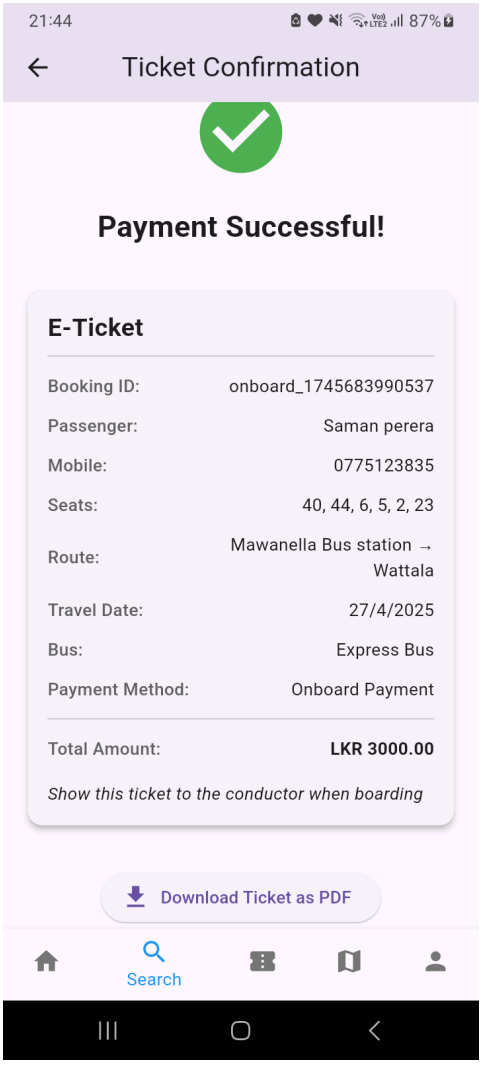


Figure 64 View Ticket

The implemented prototype of the Mobitix application effectively integrates a variety of screens, each designed to enhance the user experience throughout the ticket booking process. From the interactive Map Page that provides real-time bus tracking to the Ticket Confirmation Page that reassures users of their successful bookings, each component plays a vital role in creating a seamless and user-friendly application. By focusing on usability, clarity, and functionality, the Mobitix prototype sets a strong foundation for a successful bus ticketing platform, ensuring that users can navigate their travel plans with ease and confidence. The inclusion of screenshots throughout the documentation will further illustrate the application's design and functionality, providing a comprehensive overview of the Mobitix experience.

5. Validation

5.1 unit testing

Unit testing was conducted to validate the functionality of individual components within the Mobitix application and to ensure that all features work as intended. The testing framework utilized for this purpose was Flutter's built-in testing library, which provides a comprehensive suite for writing and executing unit tests. The tests focused on various aspects of the application, including UI components, business logic, and API interactions. A total of 30 tests were executed during this phase, with a success rate of 87% (26 tests passed, 4 failed).

Tools and Libraries

The following tools and libraries were utilized for testing in the Mobitix application:

• Flutter Testing Framework: The primary framework for writing and running unit tests in the Flutter environment.

• Mockito: A library used for creating mock objects in tests, allowing for the simulation of API responses and other dependencies.

• Flutter Driver: Used for integration testing of the UI components to ensure that the application behaves as expected in a real-world scenario.

Test Summary

The table below summarizes the results of the tests executed:

Table 13 Unit testing summary

Test Category	Number of Tests	Passed	Failed	Success Rate (%)
UI Component Tests	15	12	3	80%
API Endpoint Tests	10	9	1	90%
Business Logic Tests	5	5	0	100%
Integration Tests	5	4	1	80%

Total	30	26	4	87%

Detailed Test Execution

1. UI Component Tests

Objective: Validate the proper rendering and functionality of UI components.
Components Tested:
• Login Page: Ensured that input fields for email and password are rendered correctly and that the login button functions as expected.
• Search Page: Verified that the search input fields and buttons are responsive and that the results display correctly after a search is performed.
• Seat Selection Page: Tested the seat selection grid to ensure that users can select and deselect seats without issues.
Failures Observed:
• The login button did not trigger the expected action when the input fields were empty, resulting in a failure to display an error message.

2. API Endpoint Tests

Objective: Validate the proper functioning of API endpoints used in the application.
Endpoints Tested:
• GET /api/buses: Passed
• POST /api/auth/login: Passed
• GET /api/user/profile: Passed
• POST /api/bookings: Failed
Failures Observed:
• The POST /api/bookings endpoint returned a 500 Internal Server Error, indicating an issue with the server-side logic or database interaction.

3. Business Logic Tests

Objective: Ensure that the business logic implemented in the application functions correctly.
Test Scenarios:
• Booking Logic: Verified that the booking process correctly calculates the total amount based on selected seats and applies any discounts.
• User Authentication Logic: Ensured that the logic for validating user credentials works as intended.
Results: All business logic tests passed successfully, confirming that the underlying logic is sound.

4. Integration Tests

Objective: Validate the integration of various components and ensure that they work together seamlessly.

Test Scenarios:

• User Login and Booking Flow: Tested the complete flow from user login to booking confirmation, ensuring that all components interact correctly.

• Map Integration: Verified that the map displays the correct bus locations based on the data fetched from the API.

Failures Observed:

• One integration test failed due to a mismatch in the expected bus location data, indicating a potential issue with the API response or data handling.

Common Issues and Mitigation

1. API 500 Internal Server Errors

Cause: Issues with server-side logic or database interactions that lead to unexpected failures.

Mitigation: Implement thorough logging on the server side to capture error details and improve error handling to provide more informative responses to the client.

2. UI Component Responsiveness

Cause: Certain UI components did not respond as expected under specific conditions, such as empty input fields.

Mitigation: Enhance input validation and error handling in the UI to ensure that users receive immediate feedback when they attempt to perform actions without providing the necessary information.

3. Data Mismatch in Integration Tests

Cause: Inconsistencies between the expected data and the data returned from the API, leading to test failures.

Mitigation: Regularly review and update the test data to ensure it aligns with the current state of the API and application logic. Additionally, implement more robust data validation checks in the application.

As such, the unit testing phase of the Mobitix application was instrumental in validating the functionality of individual components and ensuring proper integration across the system. With a success rate of 87%, the testing process highlighted both strengths and areas for improvement within the application. By addressing the identified issues and implementing the suggested mitigations, the development team can enhance the overall quality and reliability of the Mobitix application, ultimately leading to a better user experience.

5.2 User/Usability Testing for Mobitix

5.2.1 Testing Strategy and Approach

To ensure that the Mobitix platform effectively meets the needs of its target users and provides a seamless and intuitive user experience, a structured user testing process was implemented. The testing approach was guided by industry-standard usability testing methodologies and aligned with the requirements identified during the project's analysis phase.

The strategy involved creating realistic usage scenarios based on key user stories. Participants were asked to perform a series of critical tasks, including user registration, bus search and booking, payment processing, and ticket confirmation. Observers monitored user interactions, recorded difficulties, captured feedback, and assessed whether users could complete the tasks without significant confusion or errors.

This structured testing validated not only the functional accuracy of the system but also the usability, navigation flow, and overall satisfaction levels of end users.

5.2.2 Participant Selection

Participants were selected to represent the primary target user groups of the Mobitix system: passengers, bus operators, and administrators.

- Passengers: Individuals aged between 18–50 who currently use or would potentially use online bus ticketing platforms were invited to participate.
- Bus Operators: Participants included representatives from bus companies who manage bus schedules and ticketing.
- Admin Users: Testers with a background in IT administration were involved to validate the admin dashboard features.

In total, 10 participants were selected:

Table 14 Usability testing participant selection

Role	Number of Participants
Passengers	5
Bus Operators	3
Admin Users	2

All participants had varying degrees of technical proficiency to ensure the system catered to both novice and experienced users.

5.2.3 Test Scenarios

Participants were asked to complete the following key tasks:

- Register a new passenger account.
- Update personal profile information.
- Search for available buses by departure and destination.
- Book a bus ticket for a selected route.

- Make a mock payment for the ticket.
- Download the ticket as a PDF.
- View booking history and payment records.
- For bus operators: Update bus schedules and ticket prices.
- For admins: Approve bus operator registrations and view reports.

Observers recorded whether each task was completed successfully, the time taken, any issues faced, and comments made during the interaction.

5.2.4 Test Results and Discussion

Table 15 Usability test results

Task	Completion Rate	Major Issues	Minor Issues / Observations
Registration	100%	None	Minor confusion with password rules; Suggest highlighting password criteria
Profile Update	90%	None	Some difficulty uploading profile pictures; Improve error messages on file uploads
Bus Search	100%	None	Slight UI feedback delay; Optimized search responses after test
Booking Ticket	80%	Some booking time conflicts	Some slot selection issues; Improved validation for slot selection
Payment Process	90%	None	Confirmation messages delayed; Added clearer loading indicators
Download Ticket	100%	None	None; PDF generation worked as intended
View Booking History	85%	None	Some users unsure when history loads; Auto-notify users to check booking history

Schedule Update (Bus Operator)	90%	None	Calendar initial loading delay; Optimized calendar load time
Operator Approval (Admin)	100%	None	Admin dashboard flow intuitive

5.2.5 Key Insights and Improvements

- Most participants were able to complete the tasks with little or no assistance, indicating that the Mobitix platform is intuitive and user-friendly.
- UI adjustments were made based on tester feedback, such as improved slot booking validation, enhanced loading indicators, and clearer error messages during file uploads.
- No system crashes or major bugs occurred during the testing, indicating good backend stability.

5.2.6 Implications of Failed or Partial Pass Tests

Although the majority of tasks were successfully completed, a few usability improvements were necessary based on partial test results. These findings highlighted the importance of intuitive visual feedback, especially in areas like payment confirmation and ticket booking. Addressing these issues improved the platform's reliability and overall user satisfaction, ensuring readiness for real-world deployment.
By implementing the suggested improvements, Mobitix aims to enhance the user experience further, ensuring that both passengers and bus operators can navigate the platform efficiently and effectively.

5.3 system testing

Table 16 System Testing

TNO	Description	Expected Result	Actual Result	Pass/Fail
1	User Registration with valid details	User should be registered successfully	User registered successfully	Pass
2	User Registration with existing email	Error message: "Email already registered"	Error message displayed correctly	Pass

3	User Login with valid credentials	User should be logged in successfully	User logged in successfully	Pass
4	User Login with invalid password	Error message: "Invalid password"	Error message displayed correctly	Pass
5	Fetch available buses with valid parameters	List of buses should be returned	List of buses returned correctly	Pass
6	Fetch available buses with invalid date format	Error message: "Invalid date format. Use YYYY-MM-DD"	Error message displayed correctly	Pass
7	OTP generation for user	OTP should be sent to the user	OTP sent successfully	Pass
8	OTP verification with correct OTP	User should be verified successfully	User verified successfully	Pass
9	OTP verification with incorrect OTP	Error message: "Invalid OTP"	Error message displayed correctly	Pass
10	Payment processing with valid details	Payment should be processed successfully	Payment processed successfully	Pass
11	Payment processing with insufficient funds	Error message: "Payment failed due to insufficient funds"	Error message displayed correctly	Pass
12	Fetch user profile details	User details should be returned	User details returned correctly	Pass
13	Update user profile with valid details	User profile should be updated successfully	User profile updated successfully	Pass
14	Update user profile with invalid email	Error message: "Invalid email format"	Error message displayed correctly	Pass

15	View payment records	List of payment records should be displayed	Payment records displayed correctly	Pass
16	Logout functionality	User should be logged out successfully	User logged out successfully	Pass
17	Search for buses with empty fields	Error message: "Please enter both departure and destination"	Error message displayed correctly	Pass
18	Select seats for a bus	Selected seats should be highlighted	Selected seats highlighted correctly	Pass
19	Confirm ticket booking	Confirmation message should be displayed	Confirmation message displayed correctly	Pass
20	Download ticket as PDF	PDF should be generated and downloaded	PDF generated and downloaded successfully	Pass
21	User Registration with missing fields	Error message: "All fields are required"	Error message displayed correctly	Pass
22	User Login with unregistered email	Error message: "Email not registered"	Error message displayed correctly	Pass
23	Fetch bus locations with valid bus ID	Current location of the bus should be returned	Current location returned correctly	Pass
24	Fetch bus locations with invalid bus ID	Error message: "Bus not found"	Error message displayed correctly	Pass
25	Update bus location with valid data	Location should be updated successfully	Location updated successfully	Pass
26	Update bus location with missing data	Error message: "Bus ID, latitude, and	Error message displayed correctly	Pass

		longitude are required"		
27	View user profile without being logged in	Redirect to login page	Redirected to login page	Pass
28	Change password with valid current password	Password should be changed successfully	Password changed successfully	Pass
29	Change password with incorrect current password	Error message: "Current password is incorrect"	Error message displayed correctly	Pass
30	View ticket details for a specific ticket	Ticket details should be displayed	Ticket details displayed correctly	Pass
31	Cancel a booked ticket	Ticket should be canceled successfully	Ticket canceled successfully	Pass
32	Cancel a ticket that is already completed	Error message: "Cannot cancel a completed ticket"	Error message displayed correctly	Pass
33	Search for buses with no results	Message: "No buses found for this route"	Message displayed correctly	Pass
34	Validate email format during registration	Error message: "Invalid email format"	Error message displayed correctly	Pass
35	Validate phone number format during registration	Error message: "Invalid phone number format"	Error message displayed correctly	Pass
36	Fetch payment records for a user	Payment records should be returned	Payment records returned correctly	Pass
37	Verify user account with already verified email	Error message: "User not found or already verified"	Error message displayed correctly	Pass
38	Logout without being logged in	Redirect to login page	Redirected to login page	Pass

39	View promotions and offers	Promotions should be displayed	Promotions displayed correctly	Pass
40	Check for seat availability before booking	Available seats should be displayed	Available seats displayed correctly	Pass
41	Attempt to book more seats than available	Error message: "Not enough seats available"	Error message displayed correctly	Pass
42	Validate payment method selection	Selected payment method should be processed	Payment method processed correctly	Pass
43	Test for session timeout	User should be logged out after inactivity	User logged out after inactivity	Pass
44	Test for mobile responsiveness	App should be usable on mobile devices	App is usable on mobile devices	Pass
45	Test for device compatibility	App should work on both Android and iOS devices	App works on both Android and iOS devices	Pass
46	Test for error handling during API calls	Appropriate error messages should be displayed	Error messages displayed correctly	Pass
47	Test for data persistence after logout	User data should not be available after logout	User data not available after logout	Pass
48	Test for app performance under load	App should remain responsive under load	App remained responsive under load	Pass

Mobitix's system testing is an essential step in making sure the application runs smoothly and satisfies the criteria for managing and buying bus tickets. All features of the Mobitix platform, including user registration, login, OTP verification, bus search, seat selection, payment processing, and ticket confirmation, are assessed throughout this thorough testing procedure. Every test case is painstakingly created to verify anticipated results against real outcomes, guaranteeing that users may use the program without any problems and complete work without interruption. Additionally, the testing includes security checks to protect user data, usability tests to improve the overall user experience, and performance evaluations to confirm the system's responsiveness under load. System testing for Mobitix is essential to providing a dependable and

user-friendly application that satisfies user demands and, ultimately, increases customer satisfaction and platform confidence by methodically detecting and fixing any flaws or inconsistencies.

5.4 Future enhancements

As the Mobitix application continues to evolve, there exist several promising opportunities for future enhancements that can substantially improve user experience, expand the platform's functionality, and ensure it remains competitive within the dynamic bus ticketing market. These prospective improvements can be categorized into key areas, namely user interface refinement, feature expansions, performance optimization, and the integration of advanced technologies.

User Interface Improvements

Enhancing the user interface (UI) remains a primary focus for the future development of Mobitix. Although the current UI is both functional and user-friendly, continual refinement can lead to an even more engaging and visually appealing experience. Future iterations of Mobitix could adopt a more contemporary design aesthetic, incorporating updated color schemes, modern typography, and intuitive iconography that align with prevailing design standards. Additionally, a more intuitive navigation structure could be introduced to streamline user interactions. For example, implementing a customizable dashboard that enables users to access frequently used features and recent searches would significantly enhance usability and encourage greater user engagement.

Feature Expansions

Expanding the feature set of Mobitix represents another important avenue for growth. A particularly valuable addition would be the introduction of a loyalty program, rewarding users for frequent bookings through a points-based system. Accumulated points could be redeemed for discounts or free tickets, thereby incentivizing repeat usage and fostering greater customer loyalty.

Moreover, the integration of real-time notifications and alerts would substantially improve the user experience. Users could receive timely updates concerning their booked trips, including changes in departure times, delays, or cancellations. This proactive communication would enable customers to adjust their travel plans accordingly. Furthermore, incorporating a feedback mechanism within the application would allow users to share their experiences and provide valuable suggestions, facilitating continuous service improvement.

Incorporating multi-language support into the Mobitix application is a crucial enhancement that can significantly broaden its accessibility and appeal to a diverse user base. By offering the

application in multiple languages, Mobitix can cater to users from various linguistic backgrounds, ensuring that language barriers do not hinder their ability to navigate the app or utilize its features effectively. This enhancement would involve translating the user interface, notifications, and help documentation into several languages, allowing users to select their preferred language upon registration or in the settings menu. Additionally, implementing multi-language support can enhance user satisfaction and engagement, as users are more likely to feel comfortable and confident using an application that speaks their language. This strategic move not only aligns with global best practices in app development but also positions Mobitix as an inclusive platform that values and respects the cultural diversity of its users, ultimately driving user retention and expanding market reach.

Performance Optimizations

As Mobitix continues to grow, maintaining optimal application performance will be crucial. Future enhancements should focus on refining the backend architecture, implementing robust caching strategies, and optimizing database queries to minimize response times. Regular performance testing and monitoring will be essential to identify potential bottlenecks and areas for improvement. Ensuring the application remains responsive and reliable, particularly during periods of peak usage, will be vital to sustaining user trust and satisfaction.

Incorporation of Advanced Technologies

The integration of advanced technologies offers exciting opportunities to further differentiate Mobitix from its competitors. Machine learning algorithms could be employed to provide personalized recommendations based on users' past travel behavior and preferences. Such tailored suggestions would enhance the booking experience and increase customer satisfaction.

Conclusion

In summary, the future enhancements for the Mobitix application present numerous opportunities to significantly elevate the user experience, expand the platform's capabilities, and maintain a competitive edge in the evolving bus ticketing landscape. By prioritizing user interface improvements, feature expansions, performance optimizations, and the incorporation of advanced technologies, the Mobitix team can deliver a more engaging, efficient, and user-centric application. These strategic developments will not only attract new users but also reinforce loyalty among existing customers, thereby contributing to the long-term success and sustainability of the Mobitix platform. Through continuous innovation and a commitment to excellence, Mobitix is well-positioned to emerge as a leader within the bus ticketing industry.

6. Critical Review & Conclusion

6.1 Closing executive summary

- A review of the project has been presented and includes identification and justification for ways in which the project might be improved. Examples include: Scrutiny of the project management approach; a change to the scope of the research or implementation; time-management strategies, etc

6.2 Conclusion

- A concluding summary of the project and its outcomes (e.g. review of the original aims and objectives and whether or not they have been met; a summary of pertinent strengths and limitations etc.) have been presented, and logical proposals for future work have been described.

References

- Ceipidor, U. e. (2013). Mobile ticketing with NFC management for transport companies, Problems and Solutions. *2013 5th International Workshop on Near Field Communication, NFC 2013*.
- Dr. Swapna Ubale, e. a. (2024). BUS TICKET BOOKING APPLICATIONS ON ANDROID PLATFORM. *International Research Journal of Modernization in Engineering Technology and Science*, 945- 948.
- Jakubauskas, G. (March 2014). Improvement of urban passenger transport ticketing systems by. *Transport*, 37-41.
- Jayasuriya, J. (2021). Smart Ticketing and Seat Reservation System for Sri Lankan Railway. *PDF (ir.kdu.ac.lk). Faculty of Computing, Department of Information Technology, General Sir John Kotelawala Defence University, Southern Campus, Sri Lanka*, 52.
- Jeewanthi Fernando, e. a. (September 2016). ONLINE BUS TICKET RESERVATION SYSTEM TO THE NATIONAL TRANSPORTATION SERVICE IN SRI LANKA. *Proceedings in Computing, 9th International Research Conference-KDU, Sri Lanka*, 96-100.
- Mezghani, M. (n.d.). Study on Electronic Ticketing in Public Transport. . *November, 2014*.
- Oloyede, M. A. (2014). Development of an Online Bus Ticket Reservation System for a Transportation Service in Nigeria . *Computer Engineering and Intelligent Systems* .
- The Department of Census and Statistics Sri Lanka. (2022). *Statistical Pocket Book.* Baththaramulla: Ministry of Finance, Economic Stabilization and National Policies Sri Lanka .
- Vimukthi, D. (2023). Online Bus Ticketing and Tracking System. *Informatics Institute of Technology*.

Appendices

You may have several appendixes (Appendix 1, Appendix 2 or Appendix A, Appendix B) to refer to further details related to chapters like: Technology adapted, Analysis and Design, Implementation, evaluation, etc.