Final Report

Skill Development Project III – ICT 3206

Bachelor of Information and Communication Technology (Honors)

Department of Information and Communication Technology
Faculty of Technology
Rajarata University of Sri Lanka

(*Leave this page blank*)

Details of the Project

Project Title	:	Train Sync
Group Number	:	Group 14
Group Name	:	Team ECODUINO
Submission Date	:	09/05/2025

Details of the Group Members

Name	Registration ID	Index No.	Signature
S.A.S.N Samarakkodi	ITT/2021/090	1711	[illegible]
B.R.C.M Bulugalla	ITT/2021/017	1639	[illegible]
C.I.D Fernando	ITT/2021/032	1654	[illegible]
H.M.D.M Hearth	ITT/2021/039	1661	[illegible]
A.M.G.N.G Kalpadeepa	ITT/2021/054	1675	[illegible]
H.M.S.N Herath	ITT/2021/040	1662	[illegible]

Details of Supervisor(s)

Name	:	Mr. Husni Mohamed
Designation	:	Lecturer(T)
Department/ Unit/ Institute	:	Department of Information and Communication Technology
Contact Details	:	(+94)76 638 3023

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Background of the project

Millions of people use railway transportation every day in Sri Lanka and other countries, making it one of the most popular and economical forms of public transportation. Railway systems still face multiple challenges in terms of passenger convenience, safety, and efficiency, although their importance. Risks are increased and overall service quality is decreased by problems like accidents at railroad crossings, dangerous passenger conduct on platforms, a lack of real-time train tracking, and illegal access to reserved compartments. [1]

Hurried people frequently try to board a train before it has completely stopped at railway platforms, which can result in injuries and accidents. In the same way, improper gate control at railroad crossings frequently results in collisions. Additionally, passengers are unhappy because they are left in confusion regarding delays and timetable modifications due to the absence of precise real-time train location and Estimated Time of Arrival (ETA) information. Additionally, the lack of effective access control in the present boarding systems allows unauthorized travelers to enter first- and second-class compartments, leading to overcrowding and security issues.

The Automated Train Platform Safety and Tracking System (TrainSync) has been presented as a solution to these problems. The technology ensures safer, more effective, and more reliable railway operations by combining automation, secure access control, and real-time monitoring. The system makes use of modern technologies including GPS tracking (NEO-6M), ESP32 microcontrollers, ultrasonic sensors, ESP32-CAM, and QR code-based authentication. The overall goal of this project is to address long-standing safety concerns while enhancing efficiency and user experience by converting conventional railway operations into a smart, automated, and passenger-friendly system.

1.2 Purpose and significance of the project

Since railway transportation is essential for public transportation, problems like delays, accidents, and safety risks frequently happen. In order to address these issues, the Automated Train Platform Safety and Tracking System combines modern technologies such as ESP32 microcontrollers, ultrasonic sensors, and QR code verification.

The technology reduces the chance of vehicle-train collisions by introducing automatic railway crossing gates that close whenever a train is detected. In order to avoid early boarding and lower the danger of accidents, passenger gates at platforms are kept shut until the train has completely arrived and stopped. Moreover, passengers may efficiently plan their travels and stay away of unexpected waiting periods with real-time train tracking and accurate Estimated Time of Arrival (ETA) updates. In order to prevent crowding and unwanted access, QR code authentication makes sure that only travelers with valid tickets for first- and second-class carriages are allowed entry.

Overall, this approach improves efficiency, increases passenger safety, and makes train travel more reliable, smooth, and secure.

1.3 Scope of the project

To increase railway safety, efficiency, and passenger convenience, the Automated Train Platform Safety and Tracking System combines automation, real-time tracking, and access control. To ensure safer boarding, fewer accidents at railroad crossings, and more reliable scheduling of trips, it combines a number of hardware and software components. However, there are some restrictions because of financial, network coverage, and infrastructure limits.

Included features of the system are,

- By maintaining gates locked until the train has completely stopped, the Automated Platform Gate System prevents rushing and lowers the number of boarding accidents.

- Accidents involving cars and trains are avoided by the Automated Railway Crossing Gate System, which automatically lowers gates when a train approaches.

- Using a web interface, travelers can access exact train location and arrival timings with the Real-Time GPS Tracking & ETA Computation feature.

- By only allowing passengers with real QR codes to board, the QR Code-Based Boarding System ensures safe access to first- and second-class compartments.

- Flask and ESP32 microcontrollers are used by the Centralized Data Management system to facilitate real-time data transfer between all subsystems.

Exclusions and Limitations of the system are,

- Due to the system's dependence on Dialog coverage, mobile network dependency may cause tracking updates to be delayed in remote or rural locations.

- Due to Low-Cost Hardware Limits, low-cost GPS and QR scanning modules must be used, which could result in less accuracy and performance than more expensive options.

- Automated gates are only placed at large train stations due to limited platform gate deployment, rural stations most likely lack the infrastructure required for this.

- Due to restricted boarding control, only first and second-class reservation compartments can use QR codes, while all other passenger compartments are still work as usual.

1.4 Aim and objectives of the project

By combining platform gates, railway crossing gate automation, GPS tracking and Estimated Time of Arrival (ETA) calculation, and QR code-based boarding access, an automated train safety and tracking system can be developed with the goal of enhancing passenger convenience, safety, and efficiency.

This project's primary goal is to increase railway transportation systems' reliability, effectiveness, and safety through automation and technology. The project's additional objectives are as follows,

1. To design a platform gate that opens only when the train has arrived at the station and is either moving very slowly or has come to a complete stop.

2. To implement an automated railway crossing gate system that ensures safe and efficient management of trains crossing roads, preventing collisions with vehicles.

3. To create a GPS-based tracking system that allows passengers to view the real-time location of trains via a website.

4. To develop an Estimated Time of Arrival (ETA) system that displays train arrival times to passengers on a web interface.

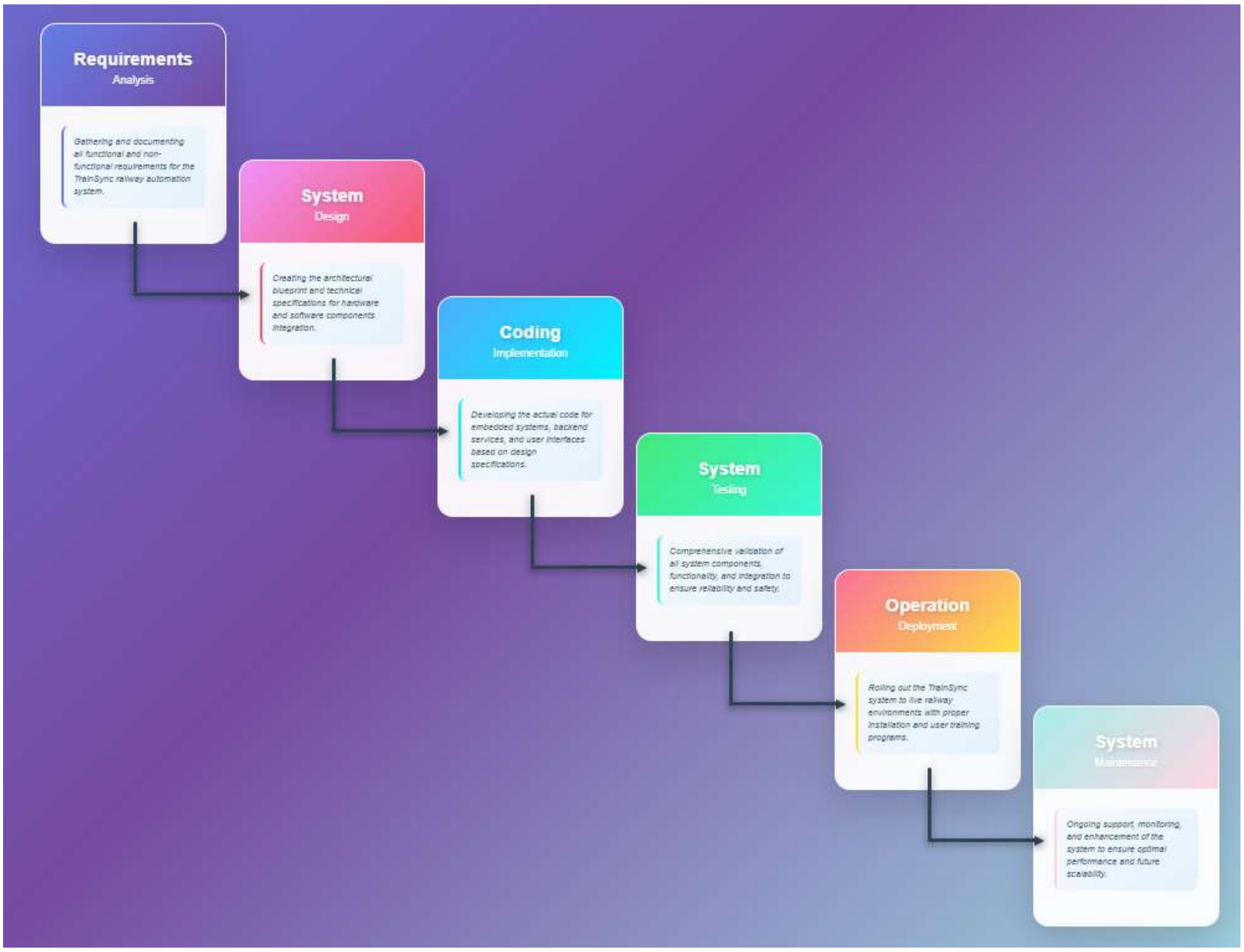
5. To implement a QR code-based door access system for fast and secure boarding of trains.

6. To develop an online platform for train seat reservations that issues e-tickets via email.

7. To utilize Flask and Firebase for backend development and real-time train tracking.

1.5 System design approach

The Waterfall Model, a sequential and linear approach to software development, was selected because it is a disciplined and structured methodology that works well for projects with well-defined requirements. [2]Its methodical approach ensures that every stage is finished before going on to the next, which lowers complexity and helps ensure that the system meets all functional and safety requirements. The steps listed below will be followed when the waterfall model is implemented in our system.



Gathering and documenting all functional and non-functional requirements for the TrainSync railway automation system.

Creating the architectural blueprint and technical specifications for hardware and software components integration.

Developing the actual code for embedded systems, backend services, and user interfaces based on design specifications.

Comprehensive validation of all system components, functionality, and integration to ensure reliability and safety.

Rolling out the TrainSync system to live railway environments with proper installation and user training programs.

Ongoing support, monitoring, and enhancement of the system to ensure optimal performance and future scalability.

Figure 1 Waterfall Model

1. Requirement Analysis

Our TrainSync System offers several key features. Before performing the requirement analysis, we identified and outlined the main functionalities of the system as follows:

Figure 2 Esp 32 controller

- Platform gates equipped with automatic opening mechanisms.
- An automated control system for railway crossing gates.
- Real-time GPS tracking and Estimated Time of Arrival (ETA) system.
- Integrated QR code-based boarding system.

The next step involves specifying the circuit and sensor requirements for the system. The ESP32 Dev Kit serves as the central controller. This cost-effective microcontroller comes with built-in Wi-Fi and Bluetooth, making it popular for smart devices, automation, and Internet of Things projects. In our system, it handles data transmission to the website and Firebase for real-time tracking. [3]

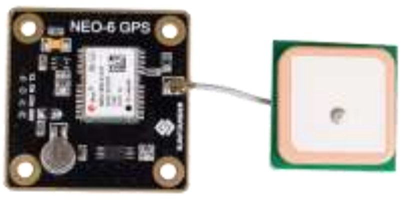


Figure 3 Neo 6M Module

The NEO-6M GPS module is widely used in IoT projects for providing precise location and timing information. In our system, it supports the GPS tracking functionality. Servo motors, which are accurate rotary actuators commonly used in robotics and automation, manage precise angular movements. In our project, they operate both the platform gates and railway crossing gates.

Figure 4 ESP 32 CAM Module

Finally, the ESP32-CAM is a low-cost microcontroller featuring Wi-Fi, Bluetooth, and a camera module, making it suitable for image processing, surveillance, and smart security applications. In TrainSync, it captures QR codes and verify it and make the compartment doors open once verified successfully. [4]

2. System Design

System design involves organizing the structure, components, and interactions of a system. It includes creating detailed plans that define both the overall system architecture and the individual modules. The platform gate mechanism is designed with sensors and control logic. Servo motors are used to lower the gates. When the train is detected by ultrasonic sensors and GPS data confirms that the train has stopped, the servo motors are activated to open the gates.

For the railway crossing gates, the system is designed to close the gates in response to an approaching train. When the ultrasonic sensor detects a train, the gates close and remain shut until a second ultrasonic sensor confirms that the train has passed. To avoid false triggers, such as a bird passing by, the system uses time-based detection alongside the sensors.

The GPS tracking system is built using GPS modules and a 4G router dongle to provide internet connectivity. The ESP32 module transmits real-time train location data to Firebase for tracking purposes.

The QR code-based boarding system, applied to first- and second-class cabin doors, allows automatic door operation. Doors open only when the ESP32-CAM module successfully reads a valid QR code from a passenger's ticket.

3. Implementation and Testing

In the implementation phase of the Waterfall model, the code for each module is developed, and the system is built according to the design specifications. This is followed by testing, which identifies and resolves any bugs or issues to ensure the system functions as intended.

The system uses IoT technology to manage GPS data transmission, autonomously operate platform and railway crossing gates, and implement a QR-based boarding system using an ESP32-CAM module. A web interface and Flask backend are developed to handle GPS tracking and QR code management. Firebase serves as the database, while HTML, CSS, and JavaScript are used for the web interface. QR code scanning is employed to control train door access. The IoT components are programmed using the Arduino IDE, and QR code generation is implemented in Python.

During unit testing, each hardware component including motors, sensors, and communication modules is installed and tested individually. Platform gates are checked to ensure they respond correctly to train detection, and railway crossing gates are verified for prompt activation and deactivation. GPS tracking and ETA predictions are tested for accuracy, and the QR code system is validated to ensure that only authorized codes can unlock the doors.

4. Deployment of the system

We are currently developing a project prototype. For real-world deployment, the system will be implemented at railway stations by installing platform gates, railroad crossing barriers, and integrating sensors and train detection systems. The passenger GPS tracking web application will be activated, and the QR code boarding system will be introduced on selected train routes.

Additionally, railway staff will be trained, and passengers will be informed about the new systems.

The deployment will follow a phased approach, gradually implementing the system over time. This allows each component platform gates, railway crossing gates, GPS tracking, and QR code boarding to be independently tested and optimized, reducing risks and ensuring smooth operation. The phased method is ideal as it minimizes disruption and facilitates gradual adaptation for both passengers and railway personnel.

Deployment phases:

- Phase 1: Install platform gates at selected stations.
- Phase 2: Set up railway barriers at high-traffic locations.
- Phase 3: Implement GPS tracking along designated routes.
- Phase 4: Enable QR code boarding on specific trains.

1.6 Project Work Plan

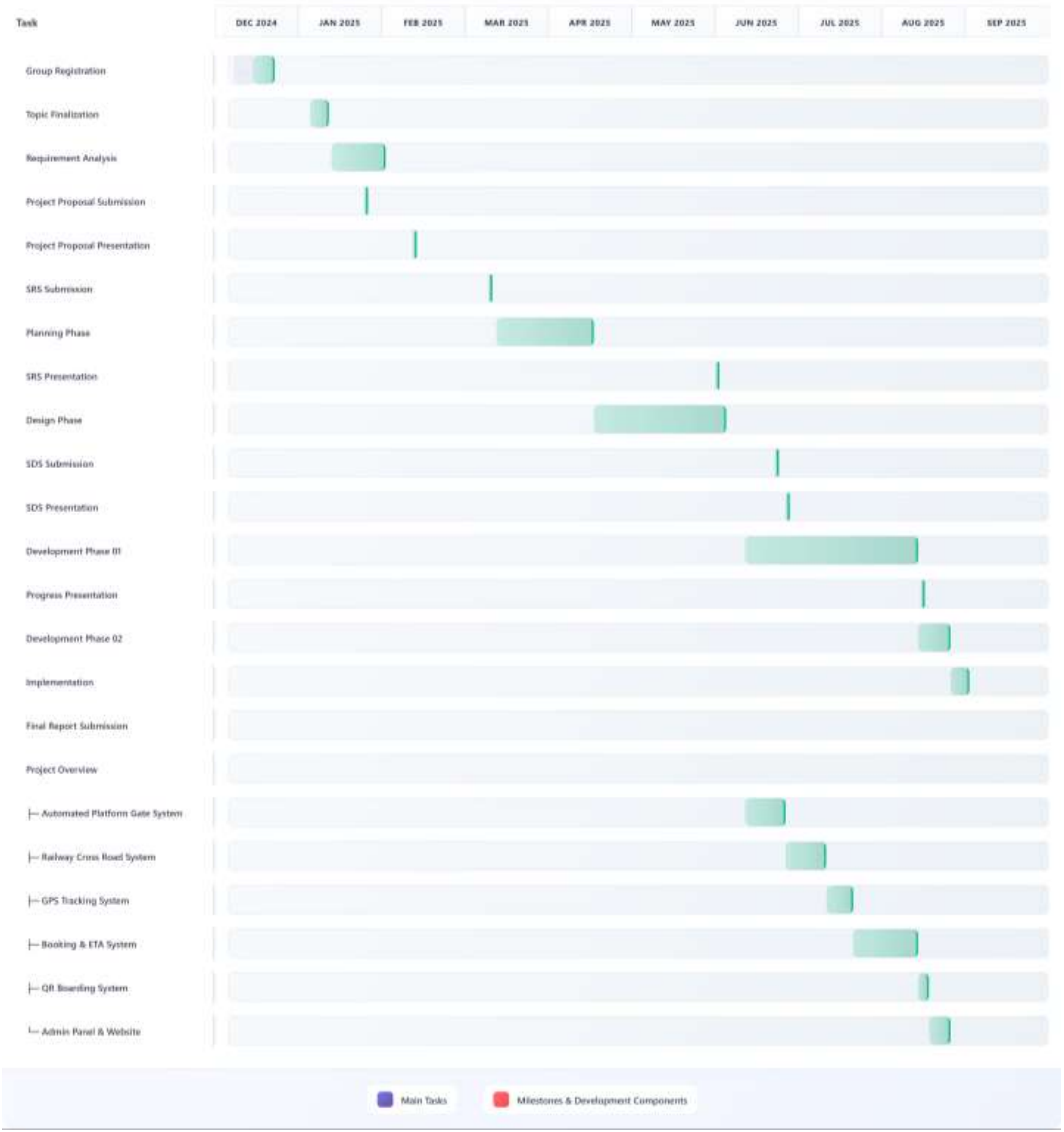


Figure 5 Gnatt Chart

2 System Requirement

2.1 Functional Requirements of the Project

Table 1 Function Name: Control Platform Gate Automatically

Priority Number	01
Function Name	The system must Control Platform Gate Automatically
Description	The system should be able to close the platform gates until the train has stopped and slowed at the platform.
Input	GPS location status and the Ultra Sonic sensor Detection.
Process	Verify train's position and movement, trigger servo motor to open gates if conditions are met
Output	Gates open only when the train has stopped and slowed.
Assumptions/ Constraints	N/A

Table 2 Function Name: Control Railway Crossing Gates Automatically

Priority Number	02
Function Name	The system must Control Railway Crossing Gates Automatically
Description	The system should be able automatically close railway crossing gates when a train is detected approaching.
Input	Ultra Sonic Sensor detection.
Process	Activate servo motors to close railway cross gate when train is detected.
Output	Railway cross gate close before the train reaches the crossing.
Assumptions/ Constraints	Sensors must be accurately calibrated to detect train distance.

Table 3 Function Name: Track Real-time GPS

Priority Number	03
Function Name	The system must Track Real-time GPS
Description	The system should be able to provide real-time location tracking of trains and display ETA for passengers on a web interface.
Input	GPS location data from ESP32 and NEO-6M module.
Process	Use Google API for process GPS coordinates and ETA automatically.
Output	Train's real-time location and ETA displayed on a web application.
Assumptions/ Constraints	Requires stable mobile network connectivity.

Table 4 Function Name: QR Code Based Boarding

Priority Number	04
Function Name	The system must verify QR Code Based Boarding
Description	The system should be allowing only authorized passengers with valid QR codes to access 1st and 2nd class compartments.
Input	QR code scanned by GM-805 module and servo motors open the door.
Process	Verify QR code with Firebase database and when the QR code verified the door will open.
Output	Doors unlock for valid QR codes and invalid QR codes are rejected.
Assumptions/ Constraints	Only implemented for reserved 1st class and 2nd class compartments.

Table 5 Function Name: Communication with Database

Priority Number	05
Function Name	The system must Communicate with Database
Description	The user should be able to view train status, passenger booking data, and QR codes in Firebase.
Input	Train location, QR codes, gate status, and timestamps.
Process	Data is updated and retrieved from Firebase in real time.
Output	Stored data is available for tracking, analytics, and access control.
Assumptions/ Constraints	Mobile network-based data transmission may cause delays.

Table 6 Function Name: Send an Email if booking is made for the user.

Priority Number	06
Function Name	The system must Send an Email if booking is made for the user.
Description	The user should receive a QR Code and the booking confirmation letter through an Email.
Input	Filling the booking details and successfully book the train.
Process	QR code and Confirmation letter made by the system.
Output	QR code and Confirmation letter send to the user.
Assumptions/ Constraints	N/A

Table 7 Function Name: Allow Viewing Train Details

Priority Number	07
Function Name	The system must allow Viewing Train Details
Description	The user should able to view the train roots and train status for booking the seats.
Input	Filling the date that the user intends to travel.
Process	Process the train routs and trains available at the user defining date.
Output	Train Routes and Train that are travel in that route and also the seat availability of that train
Assumptions/ Constraints	N/A

Table 8 Function Name: Allow Admins to Add and Delete Train Routes

Priority Number	08
Function Name	The system must allow Admins to Add and Delete Train Routes
Description	The admins should be able to delete train routes and also add new routes to the website.
Input	The train departure location and the arrival location and then Add or Remove the trains if needed
Process	Remove a train route or Add a train route by the details given by admins
Output	If a train added it should show in the booking page and if removed it should not show in the booking webpage.
Assumptions/ Constraints	N/A

2.2 Non-functional Requirements

Product Requirements

- Under normal operation, the system must handle train tracking, gate control, and QR verification within 7 seconds to ensure accurate train positioning. For gate operations, the ultrasonic sensor requires continuous detection for 5 seconds before sending a signal to close the gates.

- GPS tracking should provide train location updates with a minimum accuracy of 10 meters.

- The system must alert gatekeepers and train masters in case of any issues with the detection systems, allowing them to operate the gates and controls manually using switches.

- The web interface should be easily accessible to passengers at the station via a QR code system. QR codes will be prominently placed throughout the station for convenient scanning.

Organizational Requirements

- In accordance with Railways Ordinance (Chapter 200), Section 3, which mandates passenger safety requirements, the automated platform gate control system should ensure that platform gates remain closed until the train has fully entered and halted. This will reduce the chance of accidents and stop early track access.

- Training on system operation, manual operation in the event of a system error, and passenger assistance with system use must be provided to railway employees.

- Regular maintenance schedules and technical support are necessary to guarantee long-term dependability.

External Requirements

- The system depends on mobile networks for real-time data transmission and tracking. While Dialog’s network provides stable 3G coverage, preliminary tests between Gampola and Hatton showed connectivity issues, requiring the use of a LORA system.

- The system must adhere to Sri Lanka’s Personal Data Protection Act, No. 9 of 2022, to ensure passenger data is collected, stored, and processed securely.

- All hardware components should be weatherproof and robust enough to endure outdoor conditions at railway stations and crossings.

- Passenger awareness programs should be conducted to educate users on how to operate the QR-based boarding and tracking system effectively.

3 System Design

3.1 Architectural Design

3.1.1 System Architecture

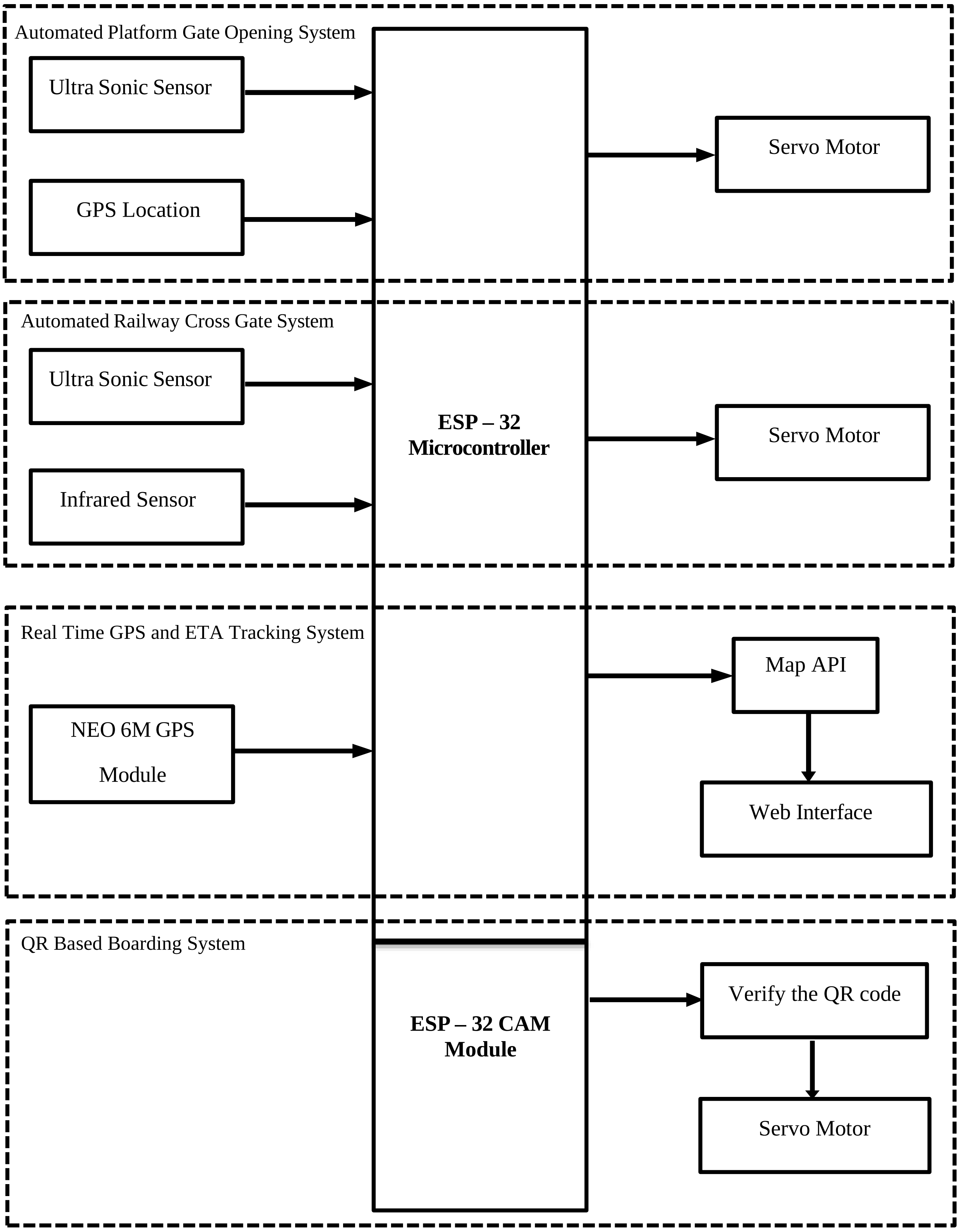


Figure 6 System Architecture of the Hardware Part

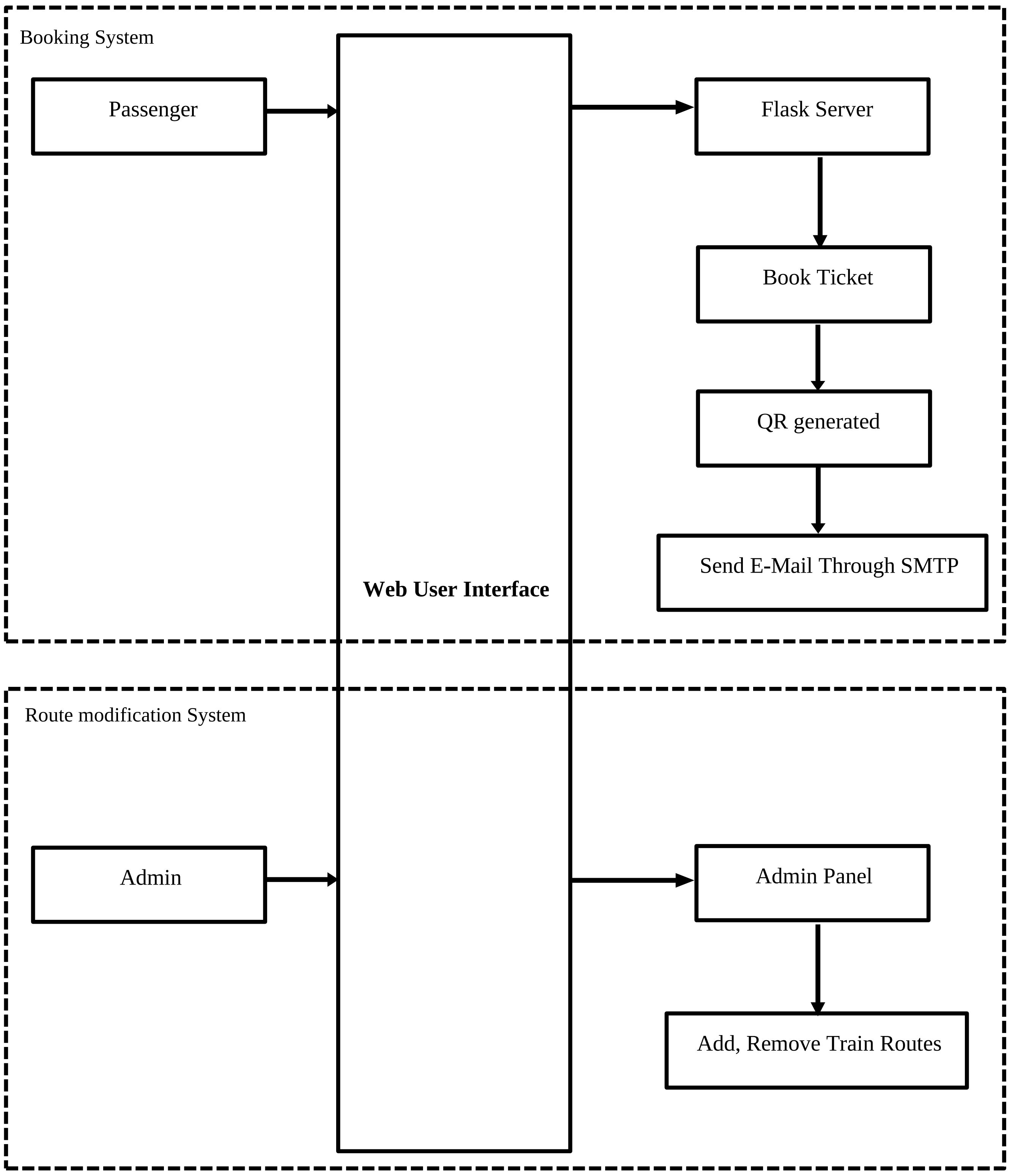


Figure 7 System Architecture of the Website Part

The Automated Platform Gate Opening System uses servo motors to control the opening and closing of platform gates. Gates remain closed until the train has either stopped completely, entered the station, or is moving very slowly. GPS data from the train helps verify its status, while ultrasonic sensors detect the train’s presence. Once conditions are met, the servo motors activate to open the gates safely for boarding.

The Automated Railway Crossroad System also relies on servo motors to manage railway crossing gates. When an approaching train is detected by ultrasonic sensors near the crossing for a continuous five seconds, the gates automatically close. After the train has passed and the sensors confirm it has exited the crossing, the gates reopen.

The Real-Time GPS and ETA Tracking System continuously collects location data from the moving train using the NEO-6M GPS module. This data is transmitted to a central database via a 4G router dongle. Analyzing and displaying the information on a web interface allows passengers to view the train’s precise location and estimated arrival time.

The QR-Based Boarding System uses the ESP32-CAM module to scan passengers’ QR codes. The system verifies the code with valid date and the name of the train, and if valid, sends a signal to the servo motors to unlock doors for first- and second-class reserved compartments ensuring secure boarding.

The Booking System allows passengers to select their travel date and destination through a user interface. Using a GET method, the system retrieves train details from the Flask server, and after payment is completed, an SMTP server sends the passenger’s e-ticket via email.

The Route Modification System enables administrators to manage train schedules through a secure admin panel. By logging in with credentials provided by the Sri Lankan Railway Department, administrators can add or remove trains from routes as needed.

3.1.2 Component design

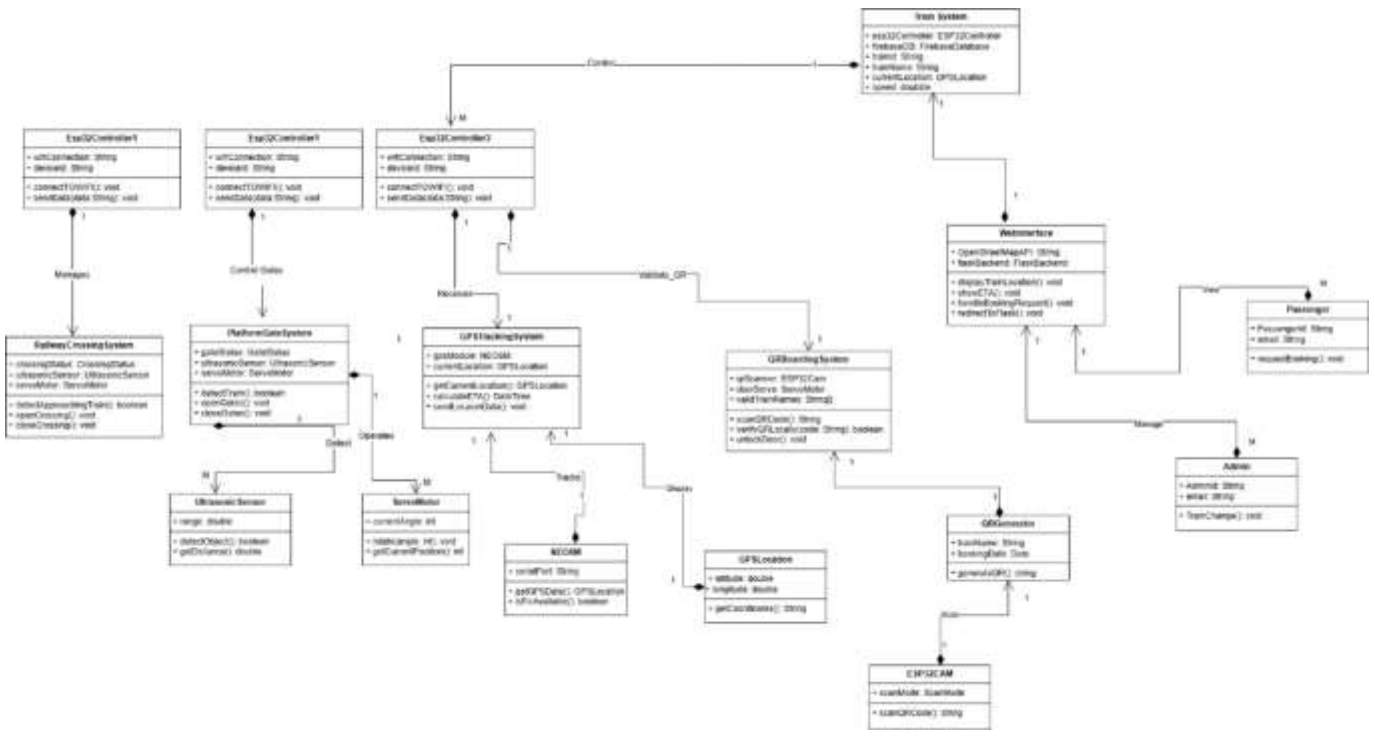


Figure 8 Class Diagram of the System

Drive Link to the Class Diagram

The diagram illustrates a comprehensive smart train automation system that integrates IoT devices, real-time monitoring, automated gate operations, and QR-based passenger boarding. At the core is the TrainSystem, managed by an ESP32Controller and connected to a FirebaseDatabase for storing and retrieving live train data. Each train entity includes details such as ID, current position, and speed, and updates its status through the GPSTrackingSystem powered by a NEO-6M GPS module, which provides location data via the GPSLocation class.

Safety operations are handled by the RailwayCrossingSystem and PlatformGateSystem, where UltrasonicSensors detect approaching trains and ServoMotors automatically control gate movements. On the passenger side, a WebInterface linked to a FlaskBackend enables users to book tickets, generate QR codes through the QRGenerator, and receive them via an EmailService. At train stations, the QRBoardingSystem equipped with an ESP32-CAM scans and validates QR codes, securely managing access by operating door servos.

Together, these components form an intelligent, fully automated train solution that combines embedded hardware, cloud databases, web platforms, and sensor–actuator mechanisms to enhance safety, improve efficiency, and deliver a better passenger experience.

3.1.3 Processes and interaction design

1. Sequence Diagram of Platform Gate System

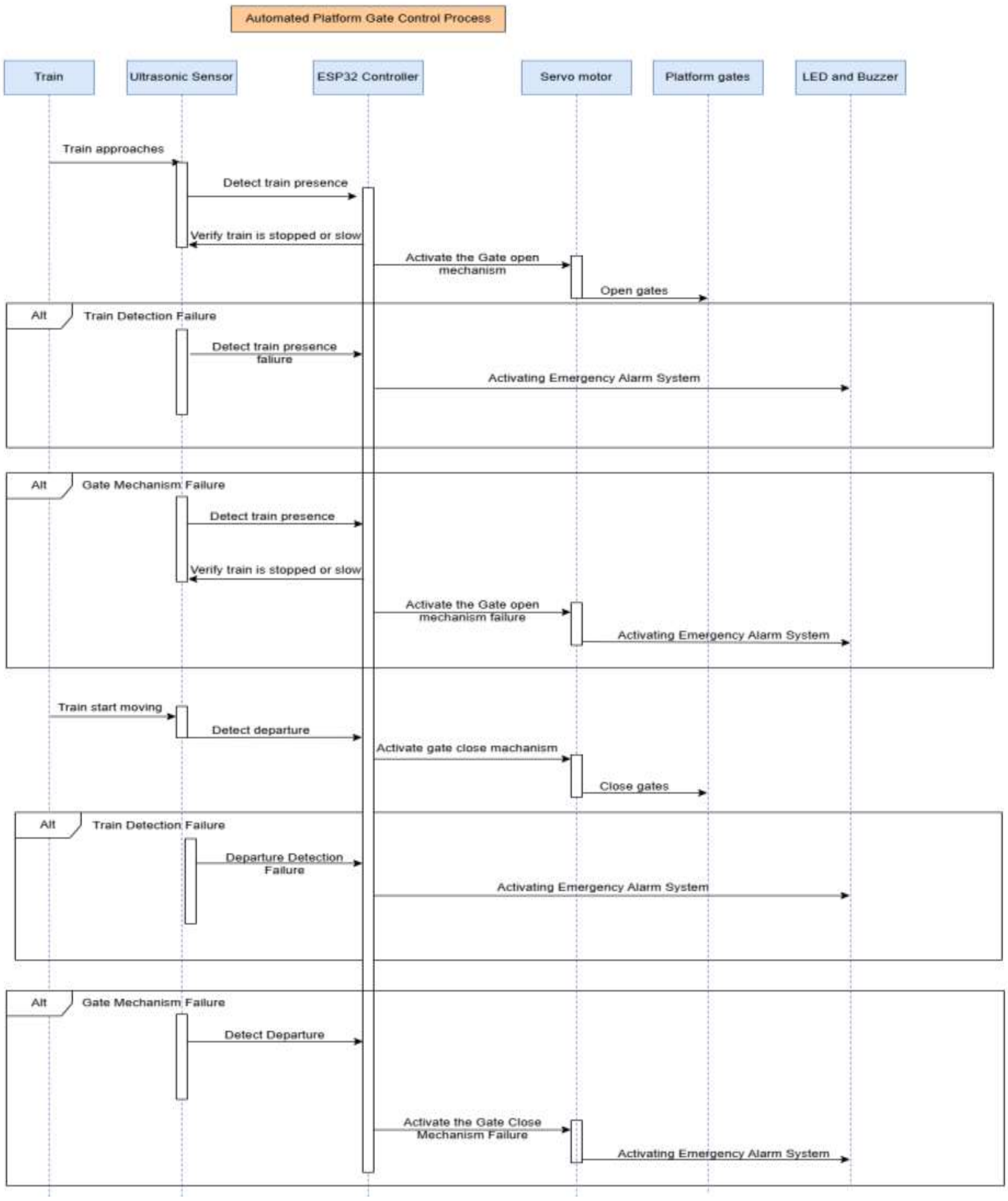

Automated Platform Gate Control Process
Train
Ultrasonic Sensor
ESP32 Controller
Servo motor
Platform gates
LED and Buzzer
Train approaches
Detect train presence
Verify train is stopped or slow
Activate the Gate open mechanism
Open gates
Alt
Train Detection Failure
Detect train presence failure
Activating Emergency Alarm System
Alt
Gate Mechanism Failure
Detect train presence
Verify train is stopped or slow
Activate the Gate open mechanism failure
Activating Emergency Alarm System
Train start moving
Detect departure
Activate gate close machanism
Close gates
Alt
Train Detection Failure
Departure Detection Failure
Activating Emergency Alarm System
Alt
Gate Mechanism Failure
Detect Departure
Activate the Gate Close Mechanism Failure
Activating Emergency Alarm System

Figure 9 Sequence Diagram of Platform Gate System

2. Sequence Diagram for Railway Crossroad System

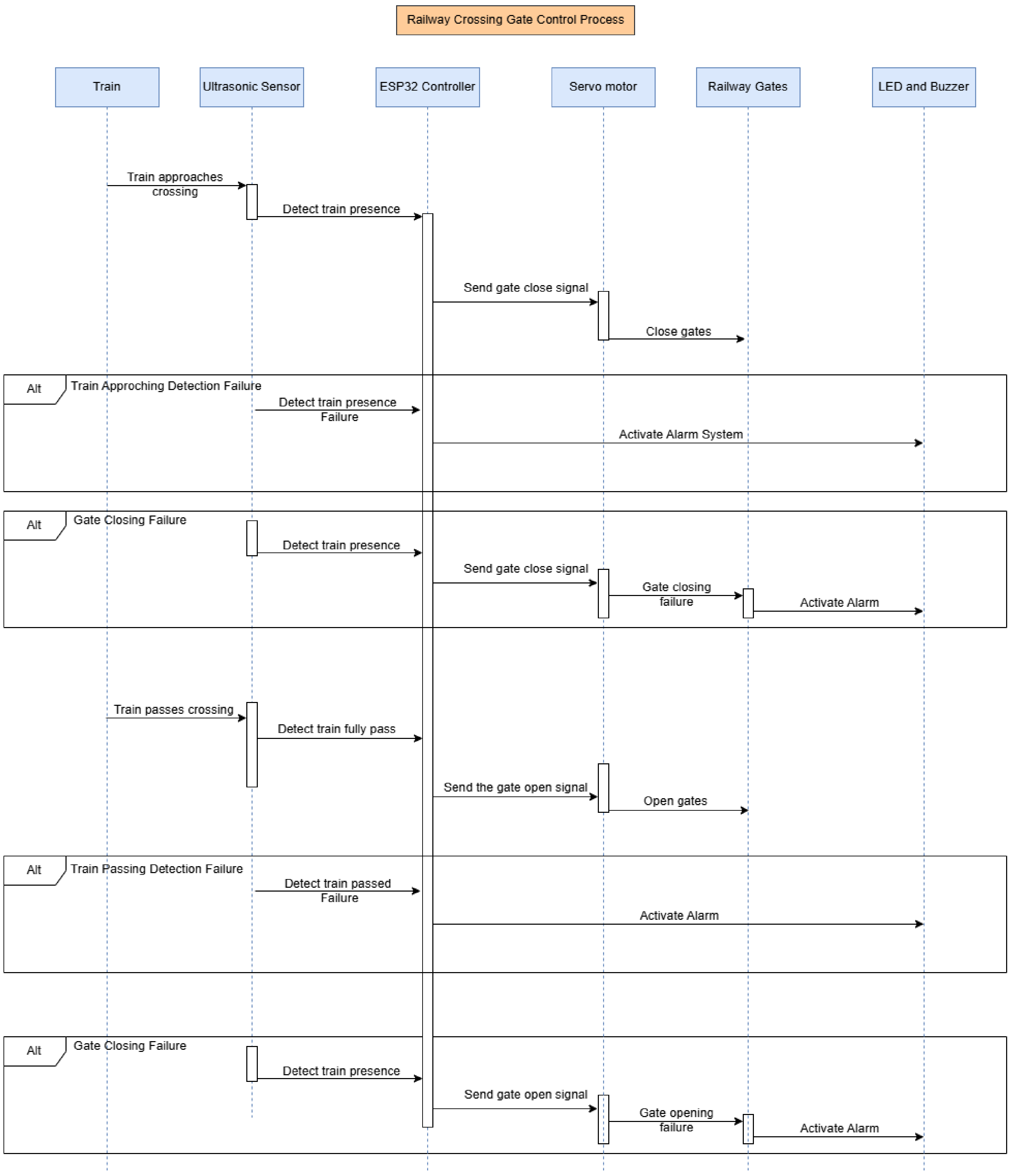


Figure 10 Sequence Diagram for Railway Crossroad System

3. GPS and ETA System

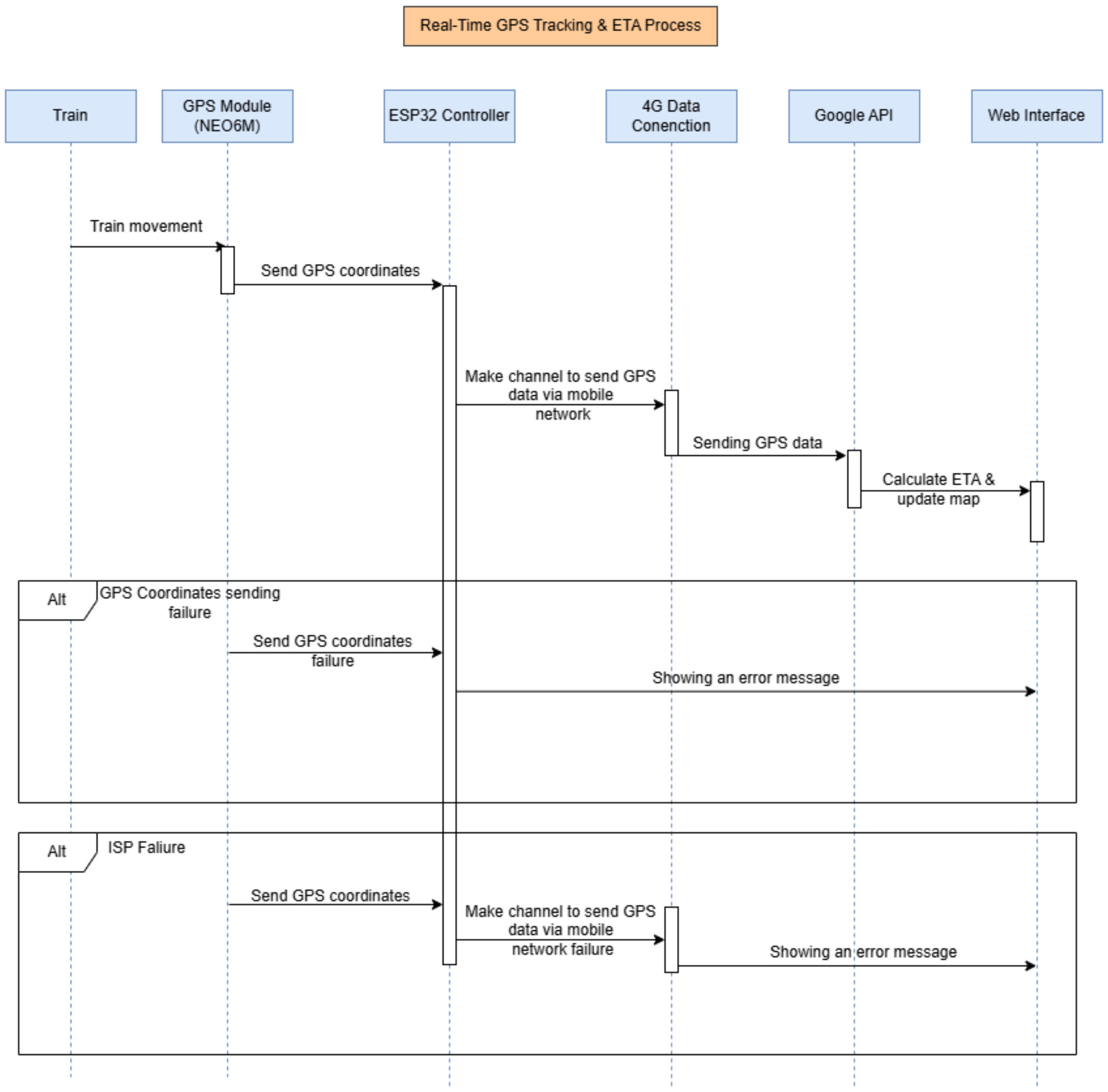

Real-Time GPS Tracking & ETA Process
Train
GPS Module (NEO6M)
ESP32 Controller
4G Data Conenction
Google API
Web Interface
Train movement
Send GPS coordinates
Make channel to send GPS data via mobile network
Sending GPS data
Calculate ETA & update map
Alt
GPS Coordinates sending failure
Send GPS coordinates failure
Showing an error message
Alt
ISP Faliure
Send GPS coordinates
Make channel to send GPS data via mobile network failure
Showing an error message

Figure 11 Sequence Diagram for GPS and ETA System

4. QR Based Boarding System

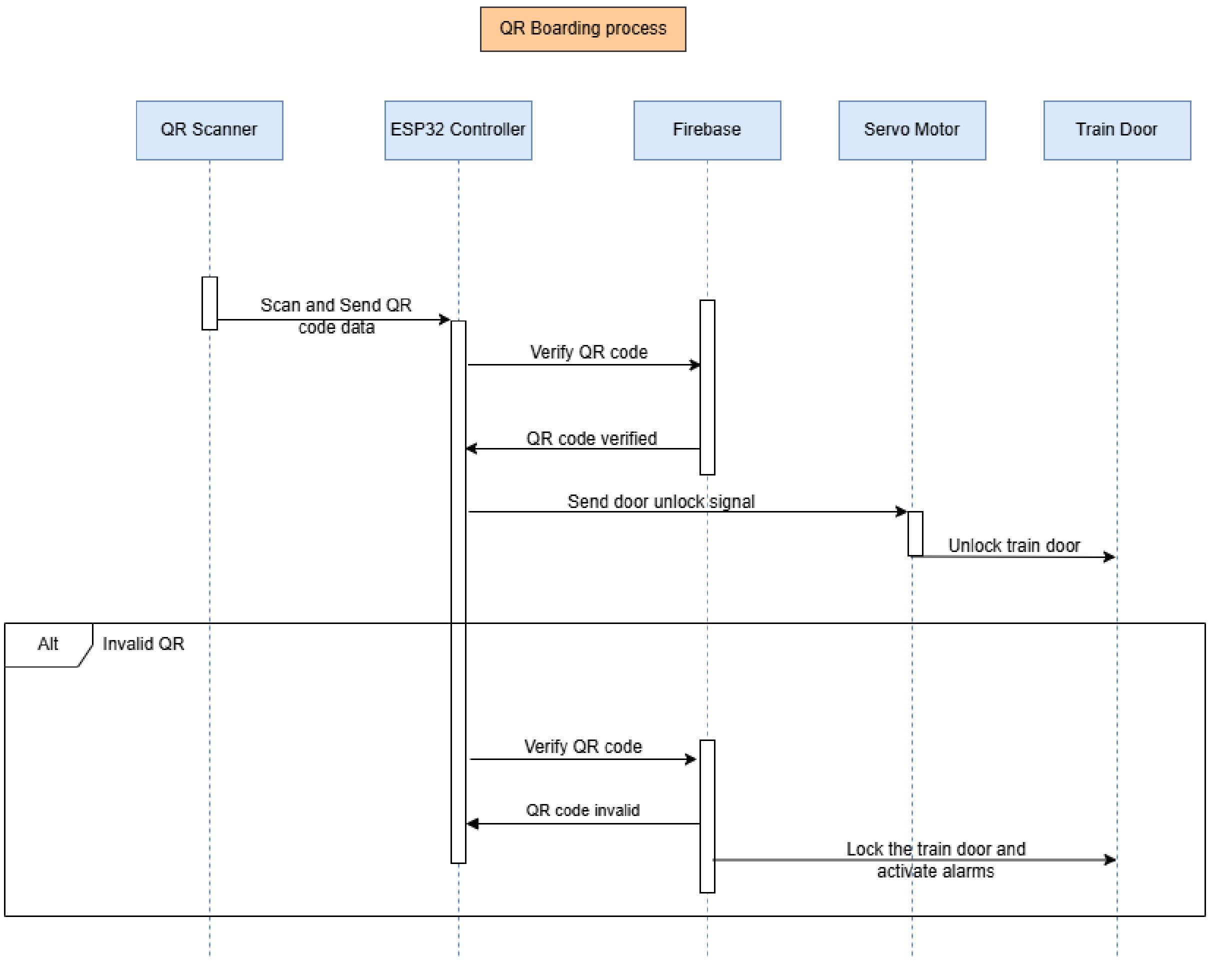


Figure 12 Sequence Diagram for QR Based Boarding System

5. Online Ticket Booking and QR Generation Process

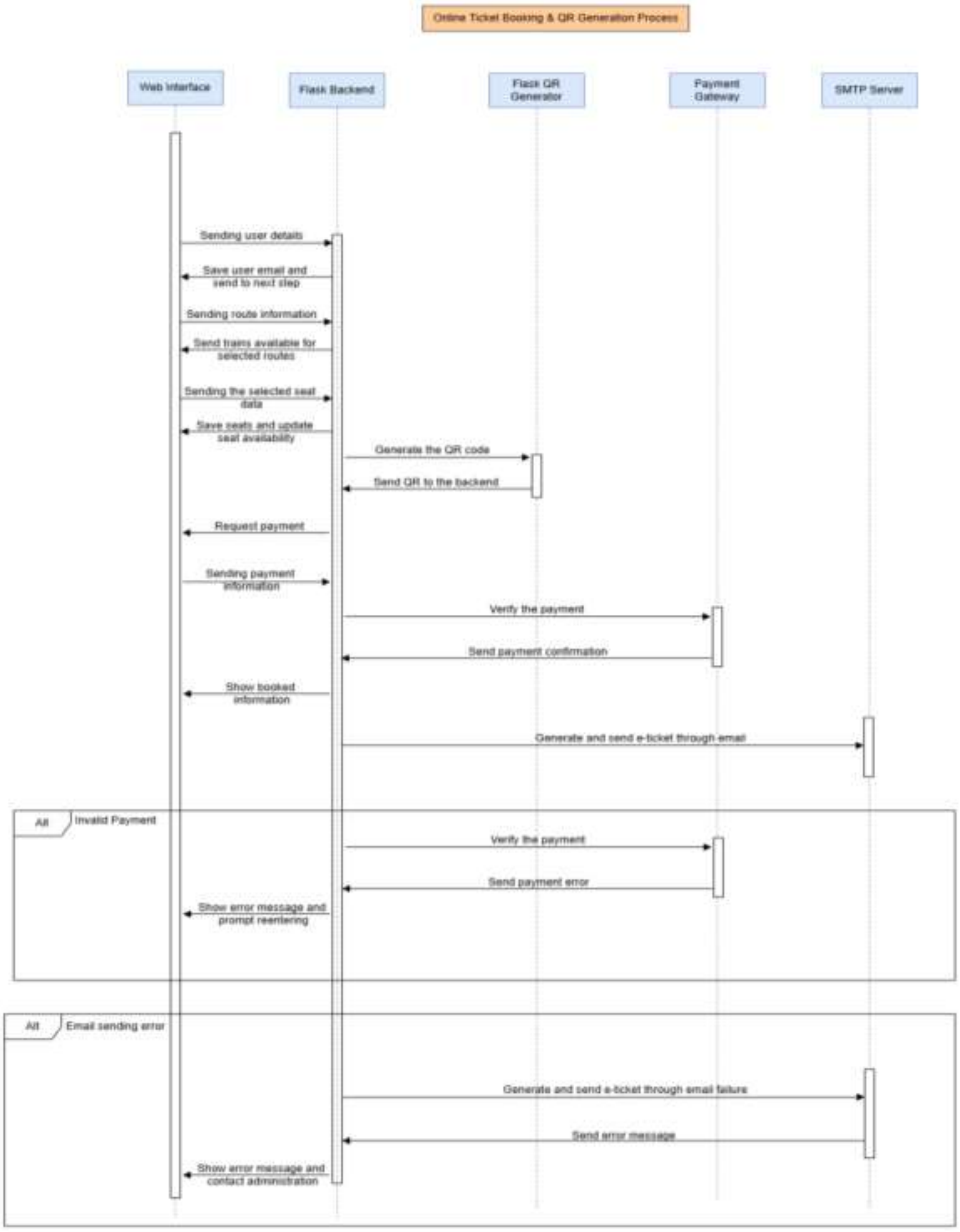

Online Ticket Booking & QR Generation Process
Web Interface
Flask Backend
Flask QR Generator
Payment Gateway
SMTP Server
Sending user details
Save user email and send to next step
Sending route information
Send trains available for selected routes
Sending the selected seat data
Save seats and update seat availability
Generate the QR code
Send QR to the backend
Request payment
Sending payment information
Verify the payment
Send payment confirmation
Show booked information
Generate and send e-ticket through email
Alt
Invalid Payment
Verify the payment
Send payment error
Show error message and prompt reentering
Alt
Email sending error
Generate and send e-ticket through email failure
Send error message
Show error message and contact administration

Figure 13 Sequence Diagram for Online Ticket Booking and QR Generation Process

[Drive Link of the above Diagram]

6. Route Management Process

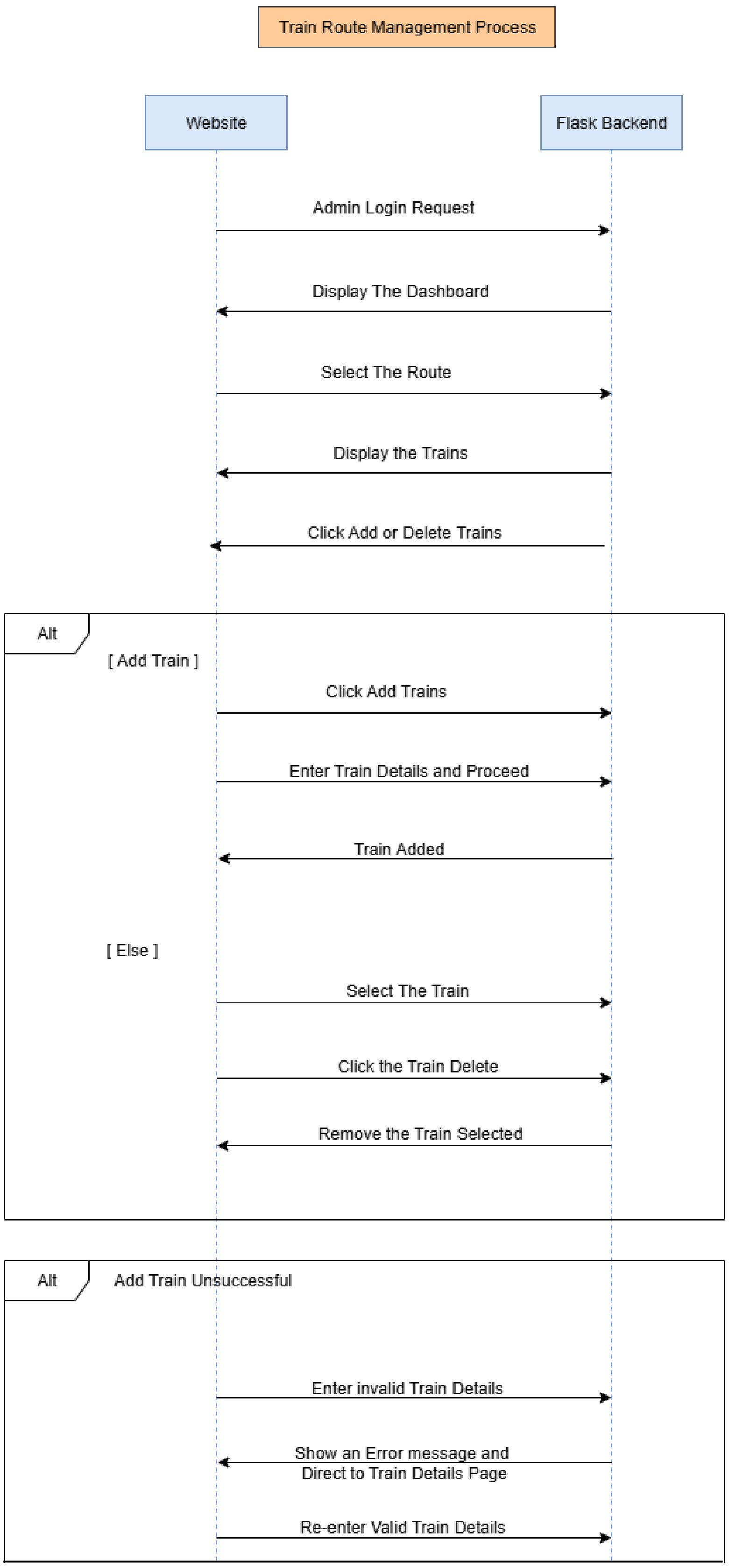

Train Route Management Process
Website
Flask Backend
Admin Login Request
Display The Dashboard
Select The Route
Display the Trains
Click Add or Delete Trains
Alt
[Add Train]
Click Add Trains
Enter Train Details and Proceed
Train Added
[Else]
Select The Train
Click the Train Delete
Remove the Train Selected
Alt
Add Train Unsuccessful
Enter invalid Train Details
Show an Error message and Direct to Train Details Page
Re-enter Valid Train Details

Figure 14 Sequence Diagram for Route Management Process

3.1.4 Tools, libraries, special algorithms and implementation environment

The Automated Train Platform Safety and Tracking System has been implemented using an extensive range of hardware and software technologies to provide real-time monitoring, secure access management, and smooth automation. The cost-effectiveness, scalability, dependability, and simplicity of interaction with both new and existing infrastructure were the main criteria used in the selection of these products. The chosen technologies maintain high levels of accuracy and security while facilitating effective communication between hardware components and the backend system. Additionally, the method places a strong emphasis on modularity, enabling future feature additions or upgrades without interfering with the system's essential operation.

1. Tools
   - Flask

     The main backend web framework for creating RESTful APIs and monitoring important backend functions including database administration, email alerts, reservation management, and QR code creation is Flask. Flask is well-known for its simplicity of use and adaptability. It is lightweight and incredibly extensible, which makes it perfect for rapid prototyping and modular design. Its simple core ensures scalability and maintainability as the system expands by enabling developers to incorporate a variety of extensions for functions like form handling, session management, and authentication. Furthermore, Flask facilitates easy contact between the web application and hardware components by supporting smooth interaction with IoT devices via APIs. This is crucial for the project's automated control and real-time train tracking.

   - Firebase Realtime Database

     Firebase is utilized exclusively for real-time GPS tracking within the system. Acting as a cloud-hosted database, it enables continuous and efficient synchronization of train location data. The ESP32 microcontroller transmits GPS coordinates to Firebase at regular intervals, ensuring that data remains up-to-date. On the frontend, the Firebase JavaScript SDK retrieves this live data, allowing seamless integration with Google Maps for dynamic visualization. This real-time mapping feature not only enhances user experience by providing accurate train locations and estimated arrival times but also supports scalability and low-latency communication, which are critical for safety-sensitive railway operations.

- HTML. CSS. JavaScript

  HTML, CSS, and JavaScript form the core technologies for developing the system's frontend interface. HTML is used to structure the web pages, ensuring semantic and accessible layouts for all user roles, including passengers and administrators. CSS is employed to style the interface, providing a clean, responsive, and mobile-friendly design that adapts to various screen sizes and devices. JavaScript adds interactivity and dynamic functionality, enabling features such as real-time GPS map updates, interactive booking forms, and instant feedback on user actions. Together, these technologies deliver an intuitive and visually appealing user experience while maintaining performance and cross-browser compatibility. Additionally, JavaScript libraries and frameworks can be integrated to enhance UI responsiveness and streamline complex operations like asynchronous data fetching for real-time train tracking and ticket management.

- OpenStreetMap(OSM)

  OpenStreetMap (OSM) is utilized as the mapping service for real-time train location visualization within the system. It provides an open-source and highly customizable alternative to proprietary mapping solutions, ensuring cost-effectiveness and flexibility. The web frontend integrates OSM through JavaScript-based libraries, allowing dynamic rendering of train positions and routes. This integration supports real-time updates by fetching GPS coordinates from Firebase and plotting them on the map interface, enabling passengers and administrators to monitor train movement accurately. Additionally, OpenStreetMap's lightweight nature ensures fast loading times and compatibility with low-bandwidth environments, making it ideal for users in rural or network-limited areas. [5]

2. Libraries

- Libraries that use in flask

  Os – Interact with the operation system.
  Json – Handles JSON data.
  Copy – Allows object copying.
  Flask – Web framework to build APIs/web apps.
  Flask_mail – Sends mail through flask.
  Qrcode – Generates QR codes.
  Io.BytesIO – Memory byte stream
  Dotenv – Loads environment variables from .env.
  Flask_cors – Handles Cross-Origin Resource Sharing.
  Reportlab – creates PDFs Programmically.

- Libraries that use in circuit or Arduino part

  Arduino.h – Core functions for Arduino (ex: digitalwrite, delay).
  WiFi.h – Connect ESP32 to Wi-Fi networks.
  ESP32QRCodeReader.h – Scans and reads QR codes using a camera.
  ESP32Servo.h – Controls servo motors with ESP32.
  ArduinoJson.h – Parses and creates JSON data.
  Esp_camera.h – controls theESP32-CAM module’s camera.

3. Special Algorithms

- QR Code Generation in Flask – Once a booking is confirmed, Flask utilizes the qrcode library to generate a unique QR code based on the reservation details.

- Email Notification to Users – The Flask backend sends emails to passengers through an SMTP server, using the email credentials stored securely in the .env environment file.

- Real-Time Train Location Tracking – The NEO-6M GPS module continuously captures the train’s location data and sends it to the ESP32, which updates Firebase with latitude and longitude coordinates at regular intervals. The web application then retrieves this data via the Firebase SDK for real-time display.

- Train Detection Mechanism – Ultrasonic sensors are employed to detect the presence of a train. To prevent false triggers, the system validates detection by ensuring the object remains in range for at least 5 seconds before confirming it as a train.

4. Development Tools

- Visual Studio Code (VS Code) – Visual Studio Code is the primary Integrated Development Environment (IDE) used for designing and coding the project's web application. It provides a powerful yet lightweight platform with features such as syntax highlighting, debugging, version control integration (Git), and extensive plugin support. The development process involved creating multiple HTML pages, including main.html (the homepage), admin.html, booking.html, and tracking.html, which are interconnected through intuitive navigation buttons. Additionally, VS Code's built-in terminal and live server extensions streamlined the testing process, enabling real-time previews and efficient debugging during development.

- Arduino IDE – The Arduino Integrated Development Environment (IDE) is used for programming the ESP32 and ESP32-CAM boards, which serve as the core controllers for hardware components in the system. The IDE provides an intuitive platform for writing, compiling, and uploading code to the microcontrollers using C/C++ with Arduino libraries. It supports essential features such as serial monitoring for real-time debugging and integration with additional libraries required for tasks like Wi-Fi connectivity, QR code scanning, and servo motor control. This environment ensures reliable communication between the hardware and the backend, enabling smooth automation of platform gates, railway crossings, and QR-based boarding.

- Python IDLE – Python IDLE is used for developing and managing the backend logic of the system. It serves as the primary environment for writing and executing Python scripts that power the Flask-based backend. Key functionalities such as QR code generation, email notifications via SMTP, API handling, and integration with the database are implemented using this tool. Python IDLE offers an interactive shell for quick testing, debugging, and validation of code, making it efficient for implementing and refining backend processes that ensure smooth communication between the web application and IoT components.

3.2 Interface Design

3.2.1 PACT (People, Activities, Contexts, Technologies) analysis of the system

The Automated Train Platform Safety and Tracking System can be created with a human-centered approach due to the structured methodology provided by the PACT (People, Activities, Contexts, Technologies) framework. The tasks that users do, the environments in which they work, the technologies that facilitate these interactions, and the system's interactions with users are all assessed in this research.

1. People

Developing a system that is both accessible and effective requires a thorough understanding of its users. Three main user groups are served by the system are Automated Components (hardware/IoT Devices), Administrators (railway staff), and Passengers.

- Passengers

The main users of TrainSync are passengers, which include long-distance and daily commuters as well as frequent tourists. They interact with the system both physically at stations and on trains, as well as mostly digitally through web platforms and mobile apps.

Efficiency is a top priority for passengers, who look for easy and quick ways to order tickets to save time and depend on accurate real-time train tracking to plan their trips effectively. While accessibility ensures that users with disabilities or little technical expertise can use the system without any trouble, safety is the primary concern, especially for families and other citizens who need to have trust in automated boarding and gate systems.

TrainSync offers to an extensive range of users with different technical skill levels. Mobile apps, digital payments, and real-time notifications are all familiar to knowledgeable users, who frequently make use of advanced functions like automated alarms or train monitoring. Simplified user interfaces, large fonts, voice help, and language support are beneficial for older or less experienced customers who might have trouble scanning QR codes, making reservations online, or navigating apps. Families and children need visual assistance or supervised boarding because young passengers cannot operate automatic systems on their own.

During busy hours, passengers frequently multitask or experience stress, therefore clear error feedback, easy navigation, and visual signals are crucial for smooth operation. Although the system's benefits, travelers still encounter difficulties including misplaced tickets or QR codes that delay boarding, communication difficulties in distant locations that prevent live train tracking or ticket verification, and language limitations that make it difficult to understand system instructions.

- Administrative

Administrative users include station managers, ticketing staff, and operations supervisors who oversee the smooth functioning of the TrainSync system. Their interaction is primarily through digital dashboards, monitoring tools, and backend management systems, although some tasks require physical coordination at stations.

In order to effectively manage train timetables, ticketing, and passenger movement, administrators must be efficient. In order to avoid mistakes in train tracking, seat distribution, and ticket validation, accuracy is essential. In order to keep an eye on operations, resolve conflicts, and ensure compliance to safety rules, they also need system accountability and transparency.

Administrators' technical skills differ. While some may need simplified interfaces and supervised procedures, others may have years of expertise with digital management tools and data analytics. They frequently have to multitask, manage difficult situations, and coordinate with several teams at once. Errors can be decreased and productivity increased with clear, user-friendly interfaces that include customized dashboards and reporting capabilities.

Operations may be interrupted by issues that administrative users encounter, such as system outages, inaccurate ticket data, or inaccessible backend tools. Corrective action may be postponed if there are communication gaps between digital notifications and physical staff. Stress might be increased by large passenger volumes during rush hours or emergencies, therefore effective workflow management and real-time information transfer are crucial.

- Automated Hardware Components

The TrainSync system considers hardware and Internet of Things devices as autonomous devices that communicate with people and one another, even though they are not "people" in the conventional sense. NEO-6M GPS modules, which offer real-time train location and speed data; ESP32 and ESP32-CAM boards, which control gate automation, QR code scanning, and train tracking; and ultrasonic sensors with servo motors, which control platform gates and railroad crossings, are important parts.

For these hardware parts to function properly, they must meet certain requirements. They must be strong and dependable in the face of severe weather, dust, or vibration. Design that is easy to maintain is crucial since it makes replacement, calibration, and inspection simple. With fail-safe procedures in place to stop accidents during malfunctions, fault tolerance is crucial. In order to provide efficient interaction with databases, communication networks, and backend systems, interoperability is also essential.

Real-time feedback from hardware components allows them to communicate with people. For example, when a passenger's QR code is verified, gates may open. They also offer proactive maintenance and safety management by immediately notifying administrators of anomalies, such as sensor failures or blockage detection.

2. Activities

The TrainSync System enables a variety of automated, digital, and physical actions involving hardware, administrators, and passengers. These efforts are intended to facilitate travel, ensure security, and offer up-to-date information.

- Passenger Activities

Both web interfaces and physical scanning stations at platforms and also the automated gates are used by passengers to communicate with the system.

Booking Tickets Online - Booking tickets online involves selecting the travel date, train route, and class of travel, followed by making secure payments through multiple options such as credit/debit cards, e-wallets, or online banking. Once the transaction is complete, passengers receive instant confirmation along with an e-ticket and a QR code embedded in a PDF. The process should be supported by simple and intuitive forms with built-in error prevention, such as auto-validating dates and preventing double bookings, to ensure a smooth and hassle-free experience.

Receiving E-Ticket and QR Code - Receiving e-tickets and QR codes allows passengers to use a single digital or printed document as both their ticket and authentication method for boarding. Users can save the PDF on their device or print it for convenience. It is important to ensure offline access, so passengers in areas with poor network connectivity can still use their tickets without disruption.

Real Time Train Tracking - Real-time train tracking enables passengers to view the live location and estimated time of arrival (ETA) of their train on a responsive map interface. To enhance accessibility, alternative display options such as text-based ETA updates should be provided, ensuring all passengers can access critical information effectively.

Boarding Process - The boarding process involves scanning the QR code at the train gate for automatic access. Passengers receive immediate feedback, with green indicating a successful scan and red signaling an invalid code or error. Staff assistance is available to help with any scanning issues. Gates should be responsive and fail-safe, providing both visual and auditory feedback to ensure smooth and secure boarding for all passengers.

- Administrative Activities

Administrators manage the system and ensure smooth operations. Their activities combine system monitoring, decision-making, and emergency handling.

Managing Train Routes and Schedules - Managing train routes and schedules allows administrators to add and remove train timings and routes through an intuitive admin dashboard.

Emergency Gate Control and Hardware Maintenance - If sensor data is absent or corrupted, perform an emergency override by manually operating platform gates with control switches. When a sensor or actuator is damaged, replacement is also a responsibility.

- Automated Activities

Automated systems reduce manual workload and enhance safety by handling repetitive or critical tasks.

Platform Gate Control - Platform gates remain closed until the train has fully stopped and is detected by sensors at the platform. The servo motors operate only after confirming the train's complete stop. Sensors ensure safety by preventing gates from closing if passengers are detected nearby, reducing the risk of accidents.

Railway CrossGate Control - Ultrasonic sensors detect approaching trains, allowing automated gates to close and open safely in coordination with the train's position and speed. Audible alarms alert nearby vehicles and pedestrians, ensuring safety at railway crossings and also notify the train by signals if there are any vehicle stuck between or under the gates.

Real-Time GPS Data Updates - ESP32 modules collect and transmit GPS data to Firebase every few seconds, enabling the dashboard to provide up-to-date train positions. This allows both passengers and administrators to track train locations in real time for better planning and operational management.

QR Code Validation and Boarding Automation - ESP32-CAM modules scan QR codes at train entry points. Upon successful validation, servo motors automatically unlock the doors to allow boarding. Invalid or duplicate scans trigger alerts, prompting administrator intervention to ensure security and smooth passenger flow.

3. Context

The environment in which TrainSync functions is essential to creating a system that is secure, dependable, and easy to use. The interactions between automated systems, administrators, and passengers are influenced by the environment. These settings can be separated into three categories as technological, social, and physical.

- Physical Context

The physical environment significantly affects system performance, accessibility, and safety. Railway platforms and stations may be indoor, semi-covered, or fully open, exposing hardware to environmental factors such as rain, heat, dust, and humidity, which requires weather-resistant components and robust enclosures. Space constraints can also impact gate placement and passenger flow, necessitating clear signage and physical cues.

Railway crossings are open environments with vehicular traffic, pedestrians, and animals. Automated systems must accurately detect obstacles, track approaching trains, and ensure gates operate safely, with highly reliable sensors and mechanisms to prevent accidents.

Power and network reliability are critical considerations, as outages or fluctuations are common in some areas. Backup power solutions, such as battery packs or solar options, ensure continuous operation, while network interruptions that affect GPS updates and real-time tracking can be mitigated with offline caching or alternative communication methods like LoRa.

Passenger mobility and flow must also be managed, especially during peak hours when platforms can become crowded. Efficient boarding processes, along with clear visual and audio feedback, help guide passenger movement safely and maintain smooth operations.

- Social Context

Human factors, cultural considerations, and behaviors play a crucial role in system adoption and usability. TrainSync serves a diverse user base, ranging from users with technical knowledge to elderly passengers and children, requiring interfaces that accommodate multiple levels of technical literacy through intuitive design.

Behavior in crowded stations can affect system efficiency, as passengers may ignore instructions, rush through gates, or inadvertently block automated systems during peak hours. Clear visual indicators such as lights and signage, auditory alerts, and staff supervision help mitigate these risks.

Cultural and linguistic considerations are also important. Multilingual support, including Sinhala, Tamil, and English, ensures accessibility for all users, while visual icons and color-coded instructions assist passengers who may face reading difficulties or language barriers.

Safety and security awareness varies among passengers, with differing familiarity with digital ticketing or automated gates. Providing instructions, training, and awareness campaigns helps improve compliance, reduce confusion, and ensure a safer, more user-friendly experience.

- Technological Context

The technological environment significantly impacts system reliability, interoperability, and user experience. GPS-based train tracking and real-time updates depend on stable internet or mobile network coverage, while remote or rural areas may require alternative communication methods such as LoRa, offline caching, or delayed data synchronization.

Cross-device compatibility is essential, as passengers access the system through smartphones, tablets, or desktops. The user interface must be responsive and optimized for various screen sizes and resolutions to ensure seamless interaction.

Security measures protect both passengers and administrators, with QR code validation, secure admin logins, and encrypted credentials preventing unauthorized access. Unique, time-bound QR codes further prevent duplication or misuse.

Integration with hardware and backend systems is critical. IoT devices, including ESP32 modules, sensors, and servo motors, must communicate seamlessly with backend servers (Flask and Firebase) for real-time coordination. System updates, maintenance, and alerts should be automated where possible, while manual overrides remain available for critical situations.

4. Technologies

The TrainSync System's technical stack integrates multiple hardware and software components to ensure security, real-time performance, scalability, and overall system reliability. These technologies enable seamless interaction between users, administrators, and physical devices while supporting data synchronization, validation, and dependable communication across the platform.

- Frontend Technologies

The frontend is developed using HTML, CSS, and JavaScript to deliver a clean, responsive, and mobile-friendly interface for passengers and administrators alike. It allows users to book tickets, submit feedback, monitor train schedules, and track trains in real-time on a map. All displays are designed for accessibility and optimized for both smartphones and desktop computers. Leveraging modern UI libraries enhances usability and consistency throughout the interface.

- Backend Technologies

The backend is built with the Flask framework (Python), which manages server-side operations such as routing, QR code generation, email dispatch, data validation, and booking processing. Flask is chosen for its simplicity, modularity, and compatibility with other Python libraries. It communicates with hardware (ESP32) through APIs and delivers dynamic data to the frontend. Additionally, the backend manages admin authentication and secures RESTful API endpoints.

- Realtime Tracking and Mapping

The tracking system utilizes NEO-6M GPS modules on the trains to determine latitude and longitude. These coordinates are sent to Firebase via GSM using a 4G connection. The website integrates OpenStreetMap, enabling passengers to view train locations in real-time and calculate ETAs dynamically. This combination provides a reliable and convenient way for passengers to track train progress and plan timely boarding.

- QR Code and Email Handling

Once a booking is confirmed, the backend generates a unique QR code using the Python qrcode package. This QR code is embedded into a PDF ticket using the ReportLab library and sent to the traveler via Flask-Mail and smtplib. The QR code is then scanned at train entry points and verified through the backend. This system enables quick, seamless, and verifiable boarding via digital authentication.

- IoT and Embedded Technologies

The ESP32 microcontroller acts as the main control unit for embedded operations, including sensor communication, QR code scanning, and servo motor activation. It is programmed with the Arduino framework and connected to peripherals such as ultrasonic sensors (for train detection), the ESP32-CAM, and servo motors (for gate and door control). The ESP32 also processes logic to detect train approach or stop conditions using sensor input or GPS coordinates.

- Sensor and Automation Components

Ultrasonic sensors detect train movement on platforms or at railway crossings. When an object is detected continuously for five seconds, the ESP32 identifies it as a train and activates the corresponding servo motors. Servo motors control the opening and closing of station gates, railway barriers, and train doors during boarding. Automation of these components enhances safety and minimizes the need for manual operation by station staff.

- Communication Technologies

The 4G dongle transmits GPS data to Firebase over mobile networks (Dialog), ensuring connectivity even in the absence of Wi-Fi, as commonly occurs on moving trains. The ESP32 and Flask server communicate via RESTful APIs over HTTP using JSON. This hybrid communication design ensures asynchronous, reliable, and cost-effective data transfer between the hardware and software components.

- Security and Authenticity

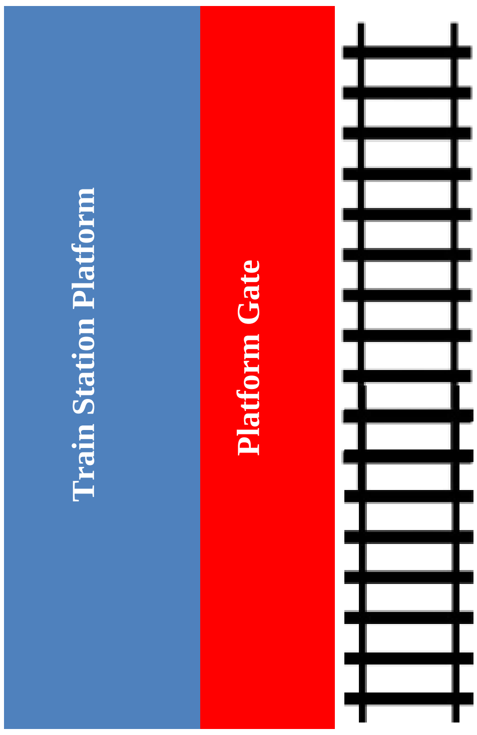
The system implements multiple layers of security. Admin login credentials are securely stored in environment variables (.env files), with authentication managed by the backend through Flask routes. QR codes are unique, time-sensitive, and linked to specific bookings to prevent duplication or misuse. For deployment, HTTPS is recommended to encrypt API traffic between client devices, ESP32 modules, and the server.

3.2.2 Interfaces (software/hardware) of the system

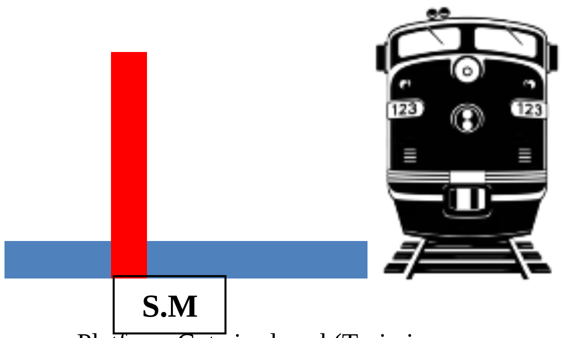
1. Hardware Interface

- Platform Gate System

Plan View of a Train Station Platform

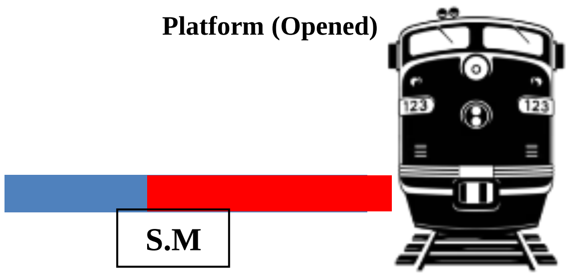


Side View of a Train Station Platform (Closed)

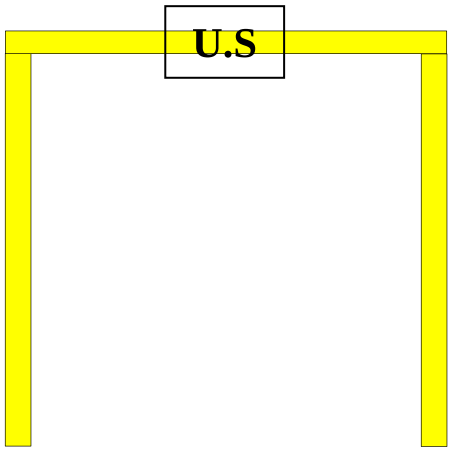


Platform Gate is closed (Train is Approaching)

Side View of a Train Station Platform (Opened)



Platform Gate is Open (Train is stopped in the station)



Placement of the UltraSonic sensor to detect the train

1. S.M – Servo Motor
2. U.S – UltraSonic Sensor

In here Side view of the train station platform the servo motors are placed in the bottom of the moving part of the platform gate so it will push the moving part upwards so the gates will be closed make avoidance of accidents during boarding. For make this servo motor works there is ultrasonic sensor situated far away from the train station that will detect trains.

In here Side view of the train station platform the servo motors are placed in the bottom of the moving part of the platform gate so it will push the moving part upwards so the gates will be closed make avoidance of accidents during boarding. For make this servo motor works there is ultrasonic sensor situated far away from the train station that will detect trains.

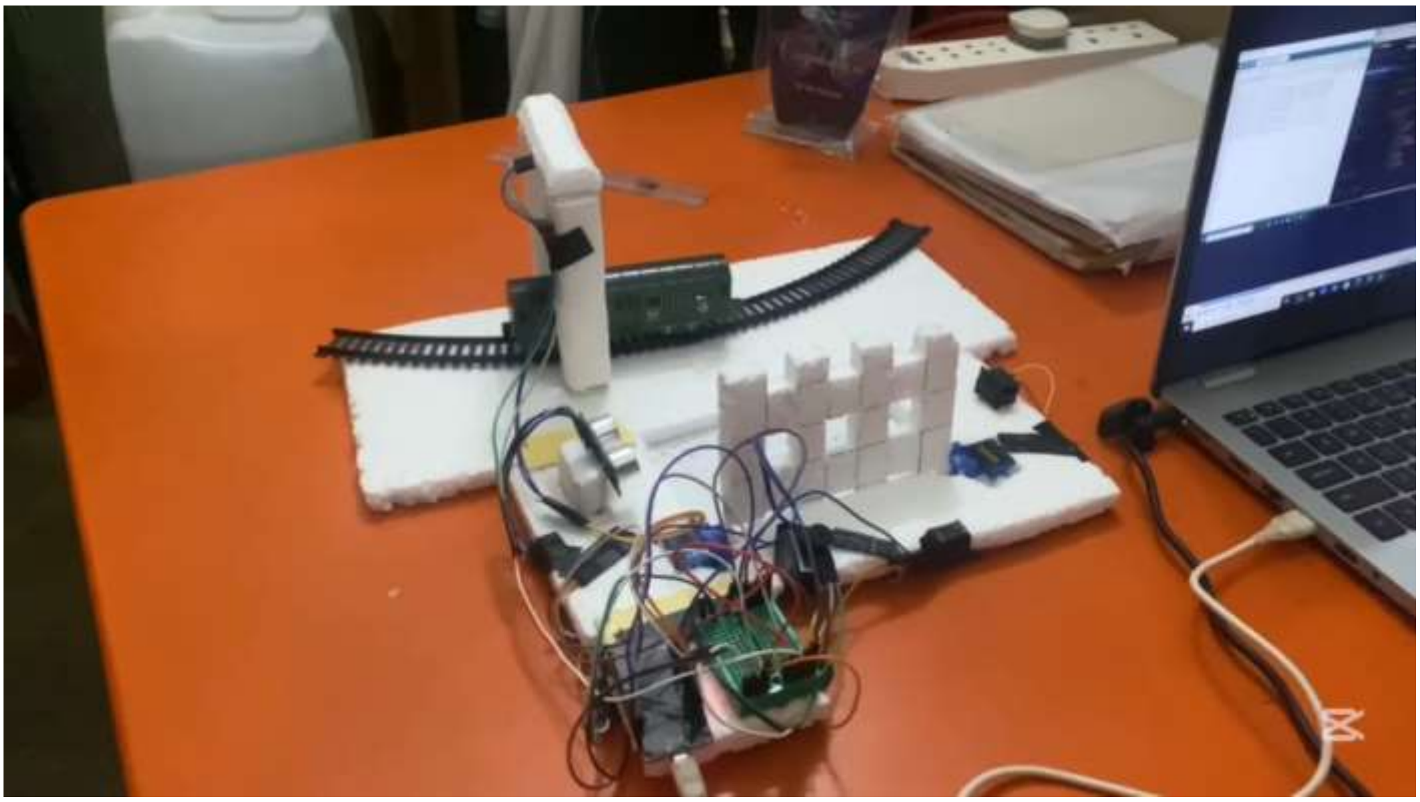
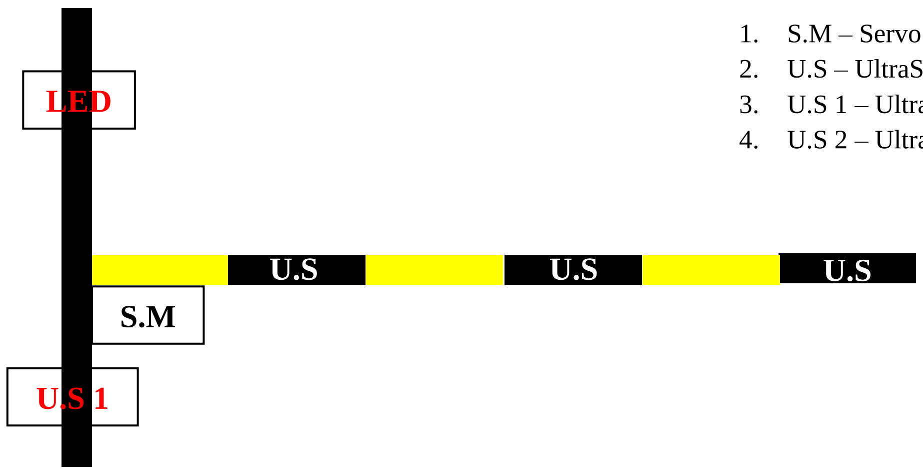


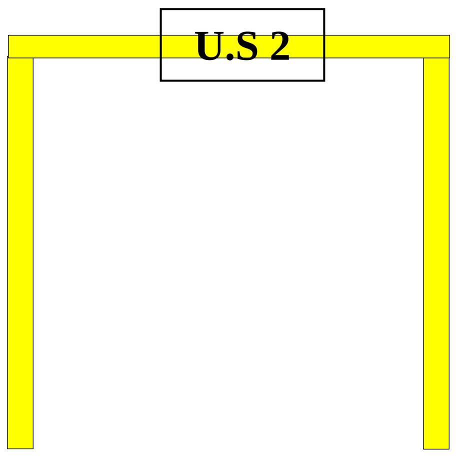
Figure 15 Hardware System of the Platform Gate System

- Railway Platform Gate System

1. S.M – Servo Motor
2. U.S – UltraSonic Sensor
3. U.S 1 – Ultrasonic Sensor for Under gate detection
4. U.S 2 – Ultrasonic Sensor for Train Detection



Front View of a Railway Cross Gate



Placement of the Ultrasonic sensor to detect the train

In here the Ultrasonic sensors in the gate itself will detect if any vehicle is stuck in between the railway cross road gates and also U.S 1 will also detect if a vehicle is under the gates. When one of these ultrasonic sensors triggered the gates will not close servo motors will not work even though the train is detected by U.S 2. If U.S and U.S 1 sensors did not detect anything servo motors will work and close the railway gates automatically.

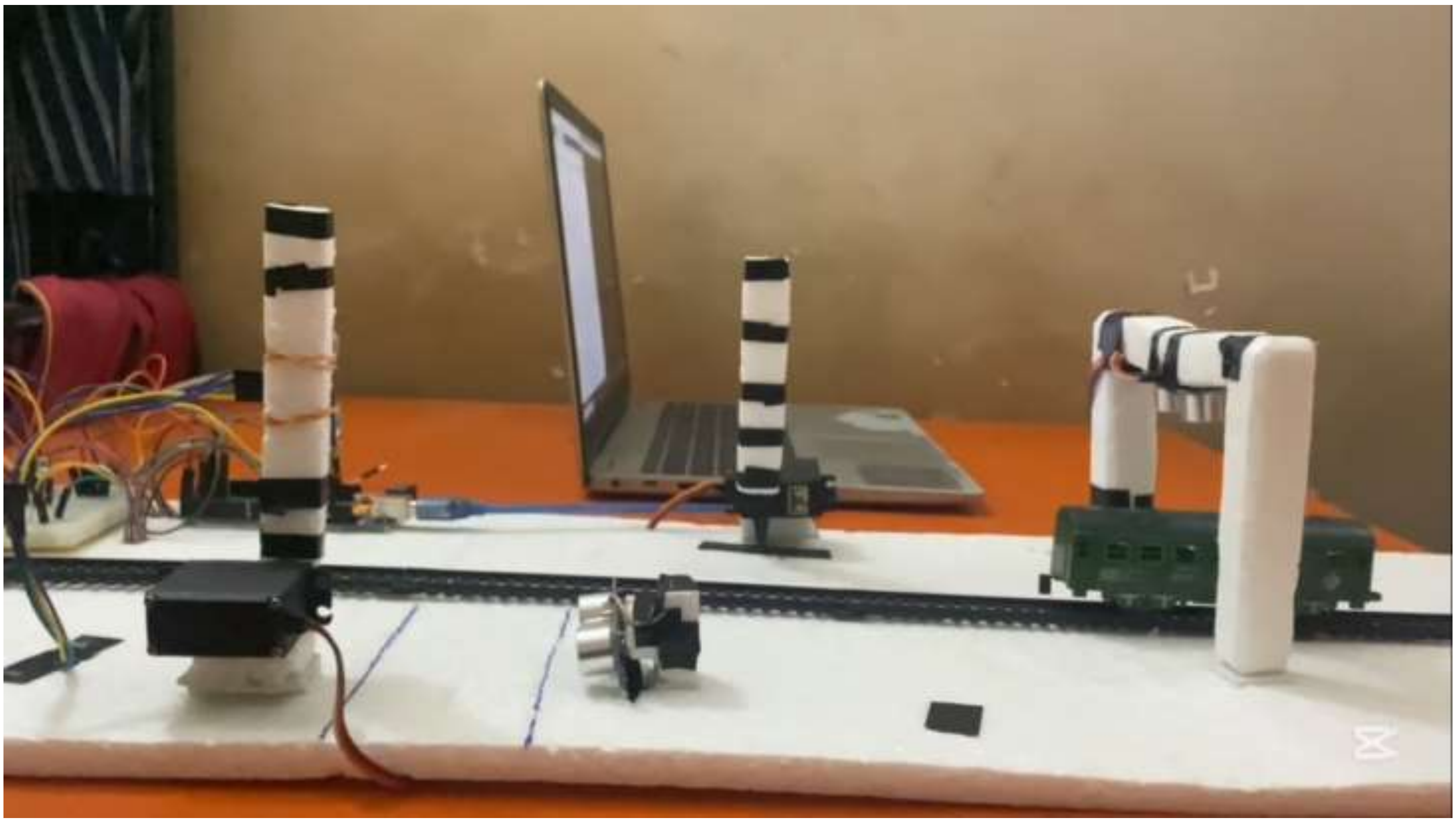


Figure 16 Hardware System of the Railway CrossGate System

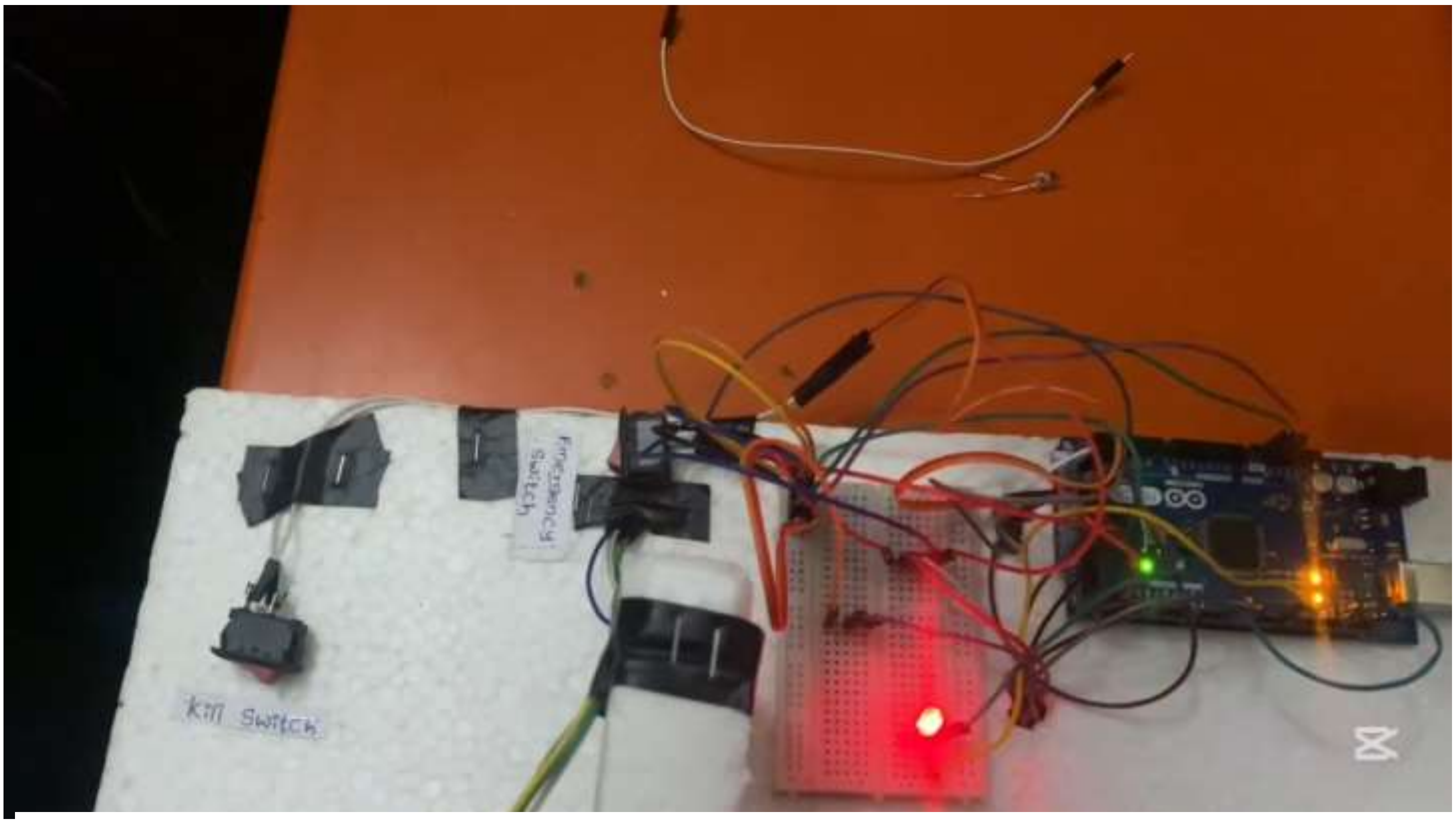
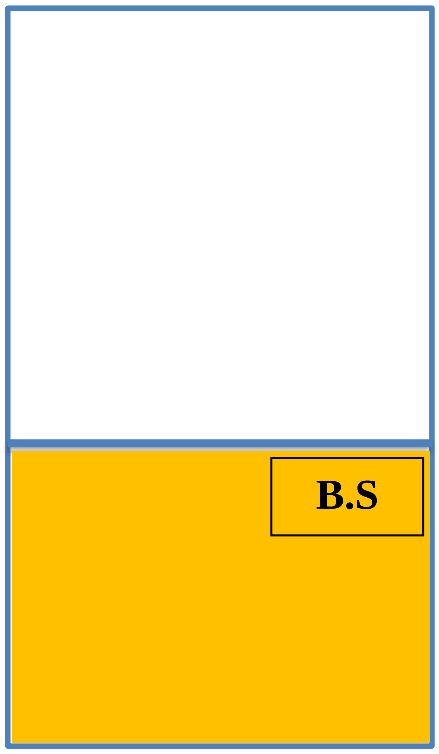
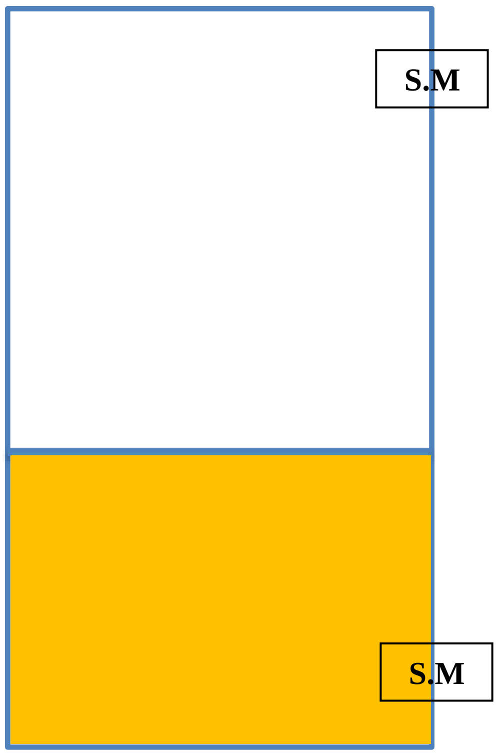


Figure 17 Circuit System of the System

- QR Based Boarding System



Front View of the Boarding Door



Back View of the Boarding Door

1. S.M – Servo Motor
2. B.S – QR Scanning Module

In here the B.S contains ESP 32 CAM that scans QR codes when the QR code is verified the servo motor will start work and it will open the door smoothly.

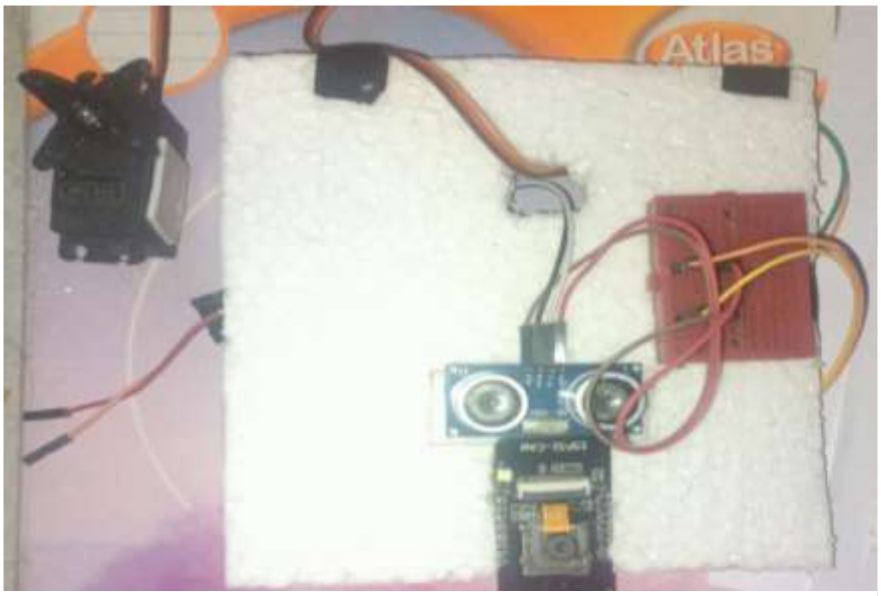
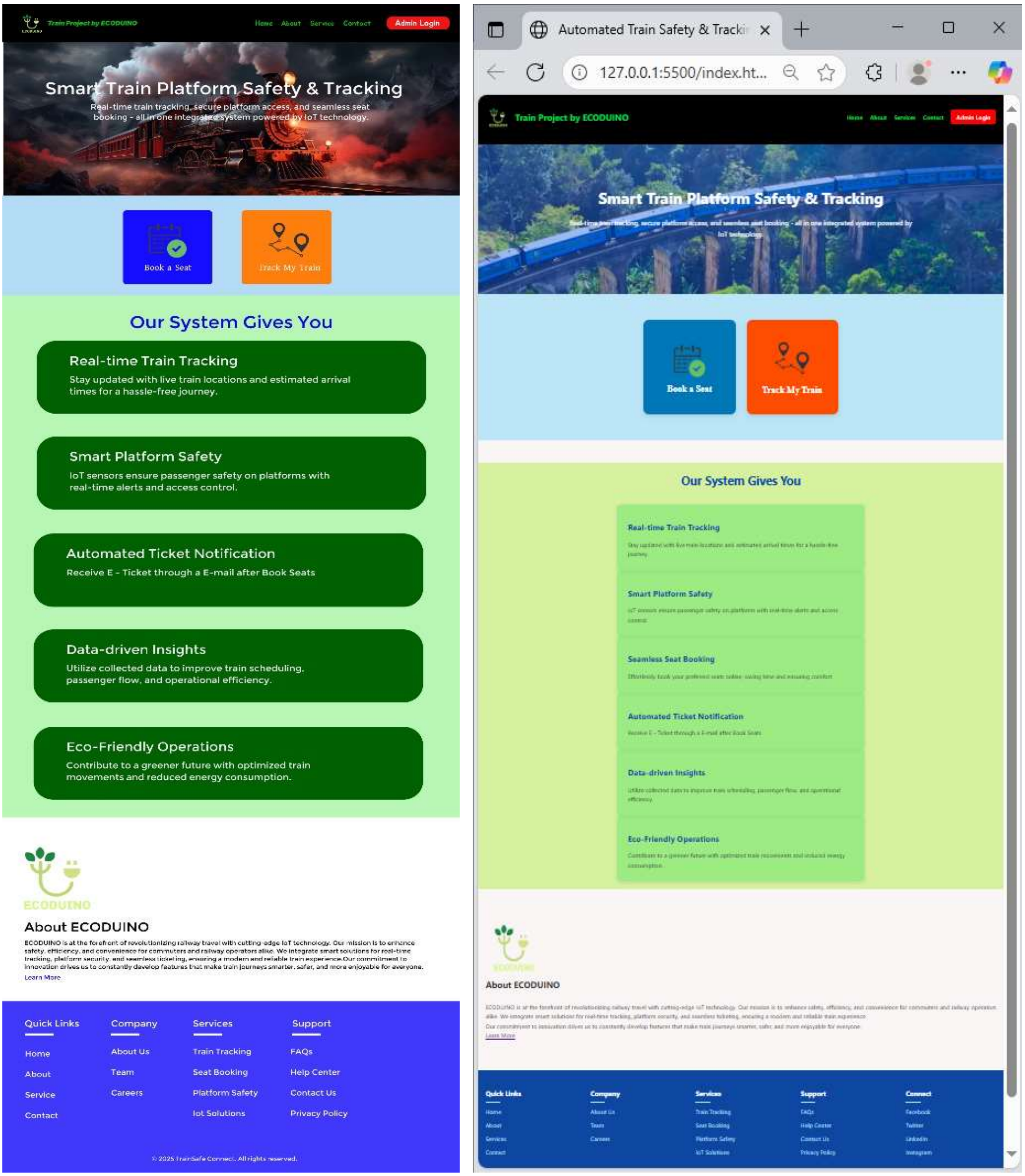


Figure 17 QR Boarding System Scanner

2. Software Interface


Train Project by ECODUINO

Home About Service Contact Admin Login

Smart Train Platform Safety & Tracking

Real-time train tracking, secure platform access, and seamless seat booking - all in one integrated system powered by IoT technology.

Book a Seat Track My Train

Our System Gives You

Real-time Train Tracking

Stay updated with live train locations and estimated arrival times for a hassle-free journey.

Smart Platform Safety

IoT sensors ensure passenger safety on platforms with real-time alerts and access control.

Seamless Seat Booking

Automated Ticket Notification

Receive E - Ticket through a E-mail after Book Seats

Data-driven Insights

Utilize collected data to improve train scheduling, passenger flow, and operational efficiency.

Eco-Friendly Operations

Contribute to a greener future with optimized train movements and reduced energy consumption.

ECODUINO

About ECODUINO

ECODUINO is at the forefront of revolutionizing railway travel with cutting-edge IoT technology. Our mission is to enhance safety, efficiency, and convenience for commuters and railway operators alike. We integrate smart solutions for real-time tracking, platform security, and seamless ticketing, ensuring a modern and reliable train experience.Our commitment to innovation drives us to constantly develop features that make train journeys smarter, safer, and more enjoyable for everyone.

Learn More

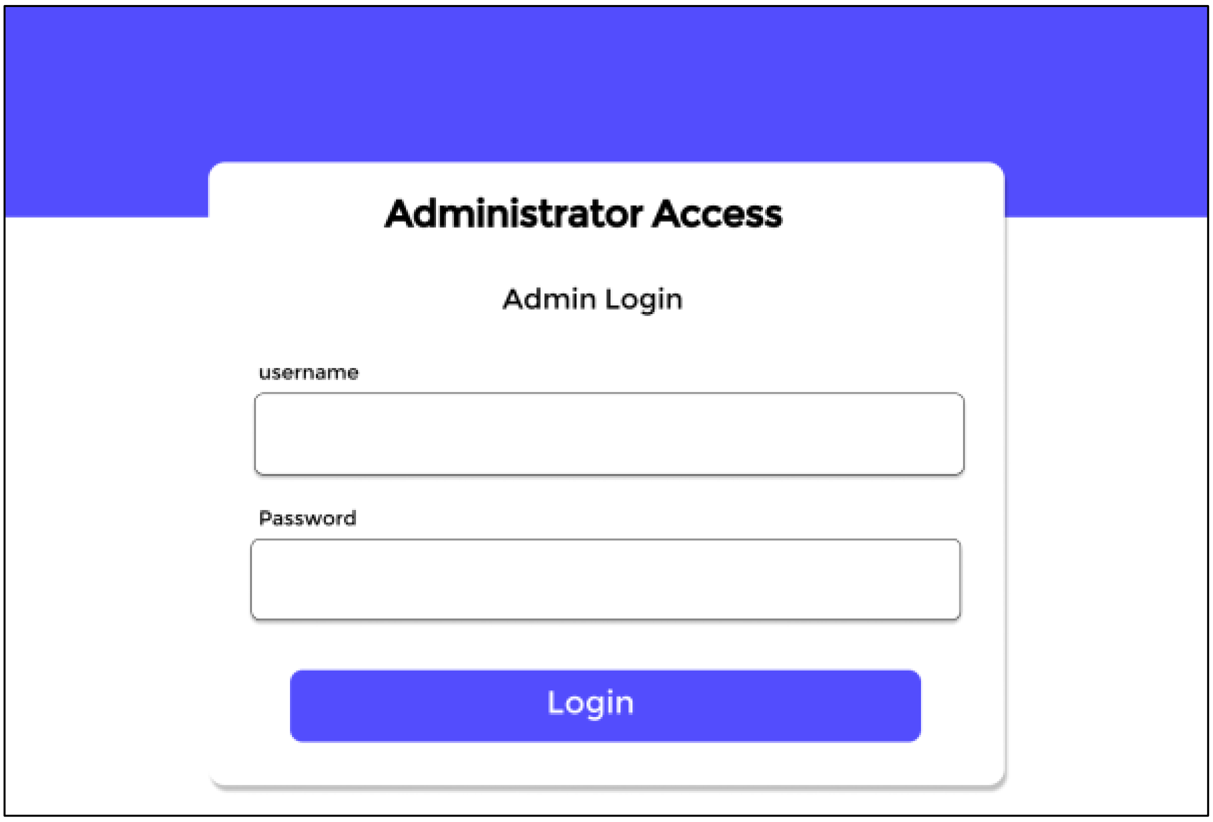
Quick Links	Company	Services	Support	Connect
Home	About Us	Train Tracking	FAQs	Facebook
About	Team	Seat Booking	Help Center	Twitter
Service	Careers	Platform Safety	Contact Us	LinkedIn
Contact		IoT Solutions	Privacy Policy	Instagram

© 2025 TrainSafe Connect. All rights reserved.

Automated Train Safety & Trackin

127.0.0.1:5500/index.ht...

Figure 18 UI of the Home Page



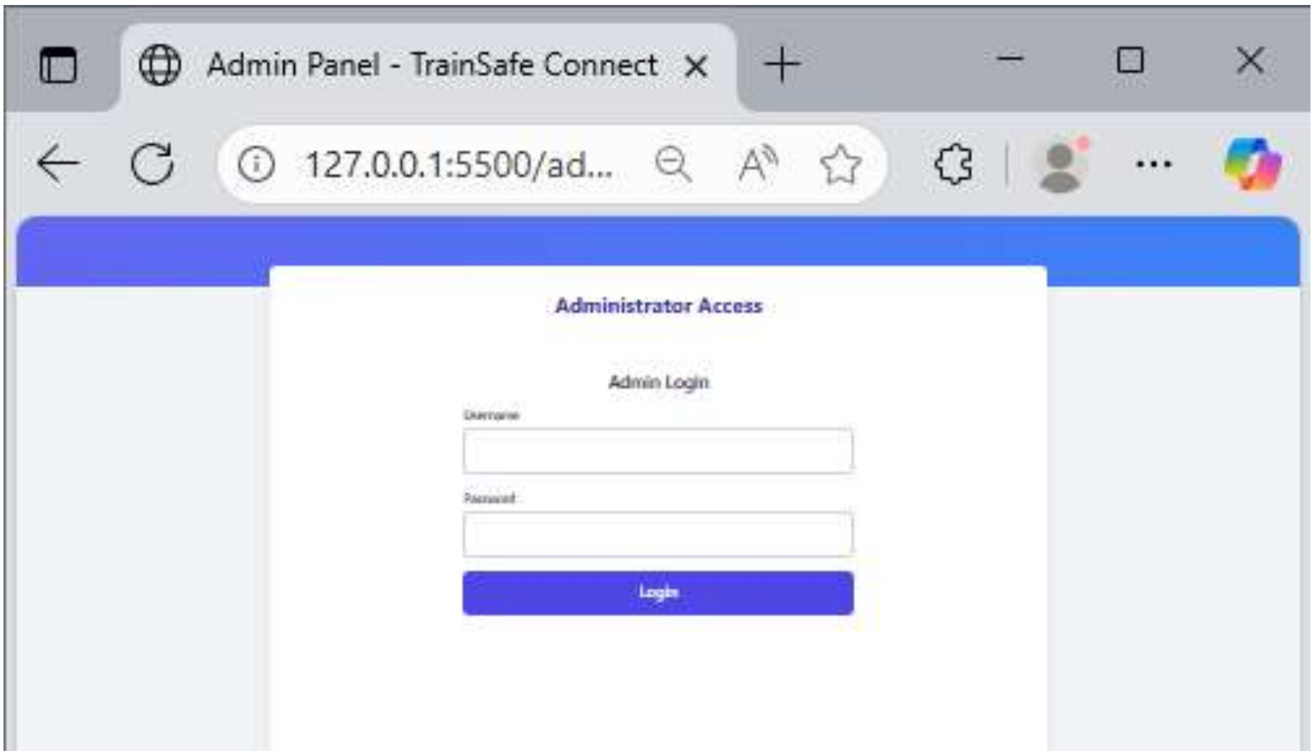
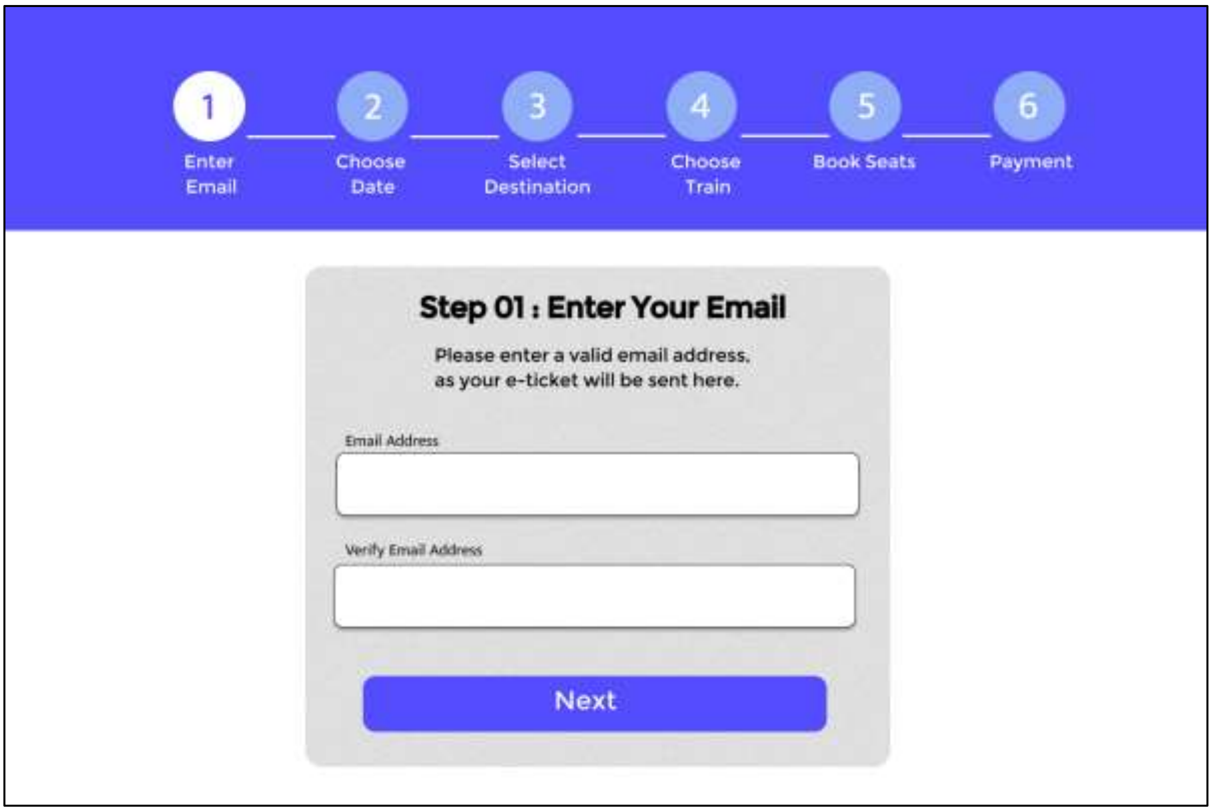
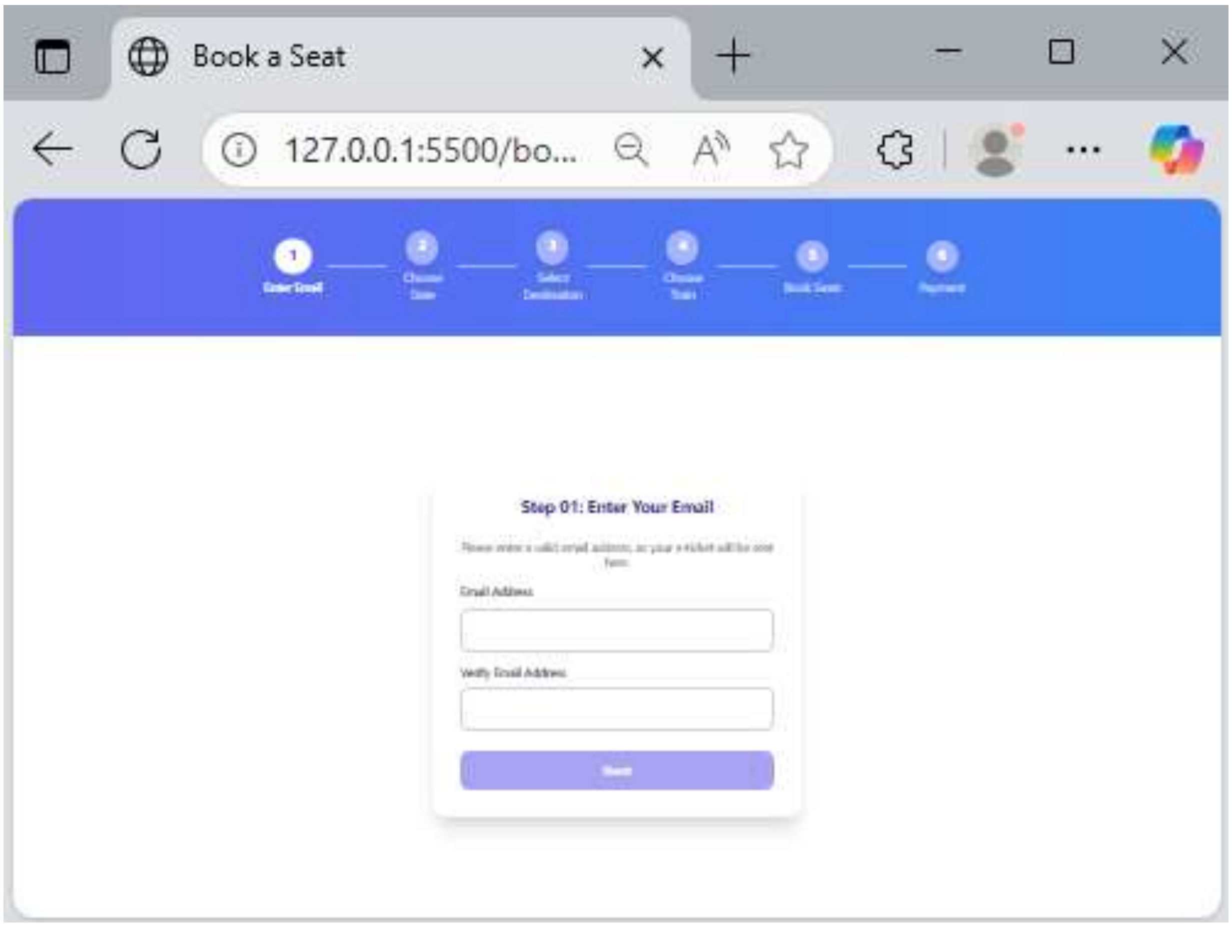


Figure 20 UI of the Admin Page



Step 01 : Enter Your Email

Please enter a valid email address, as your e-ticket will be sent here.



Step 01: Enter Your Email

Please enter a valid email address, as your e-ticket will be sent here.

Figure 21 UI of the Booking Page Step 01

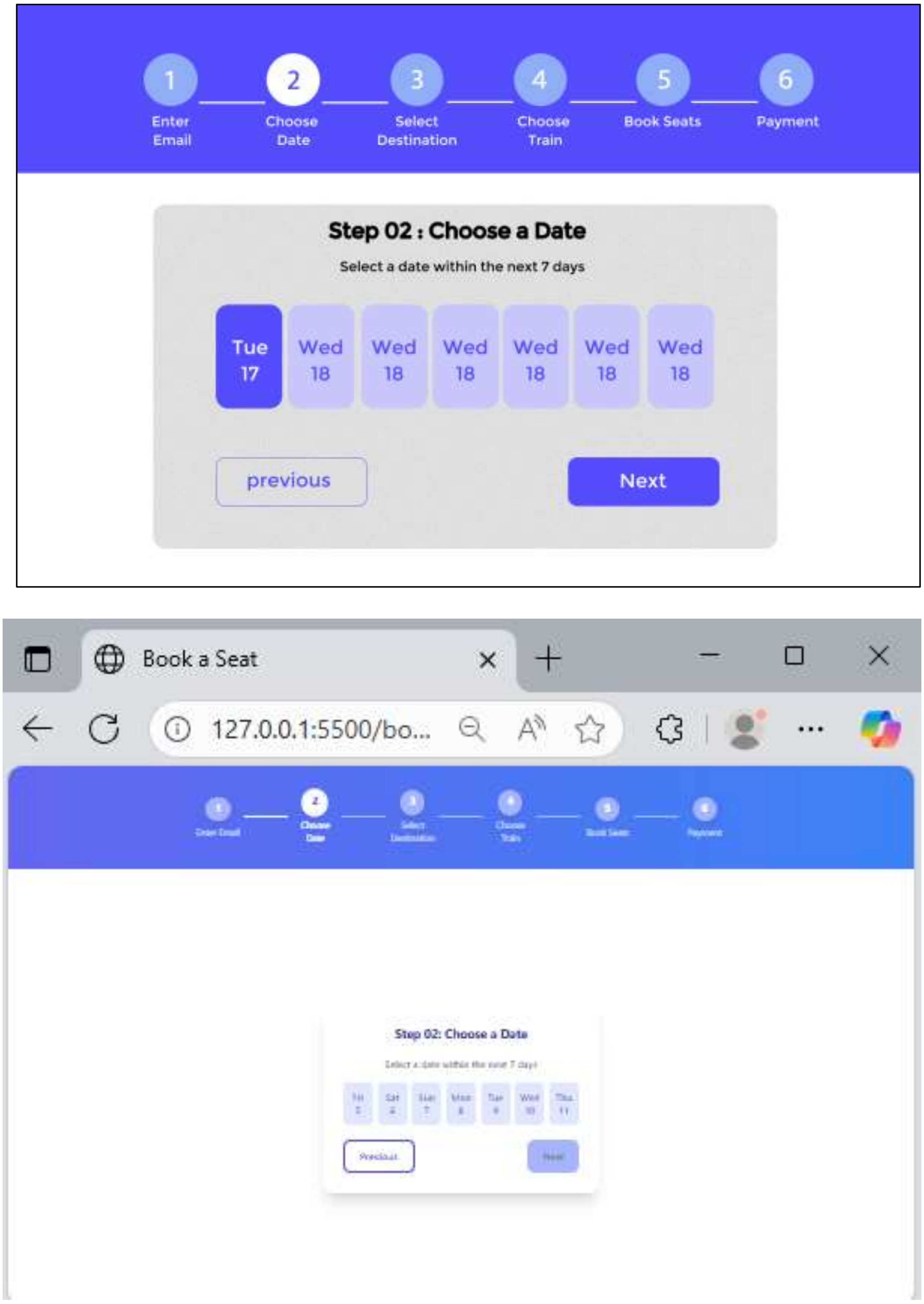
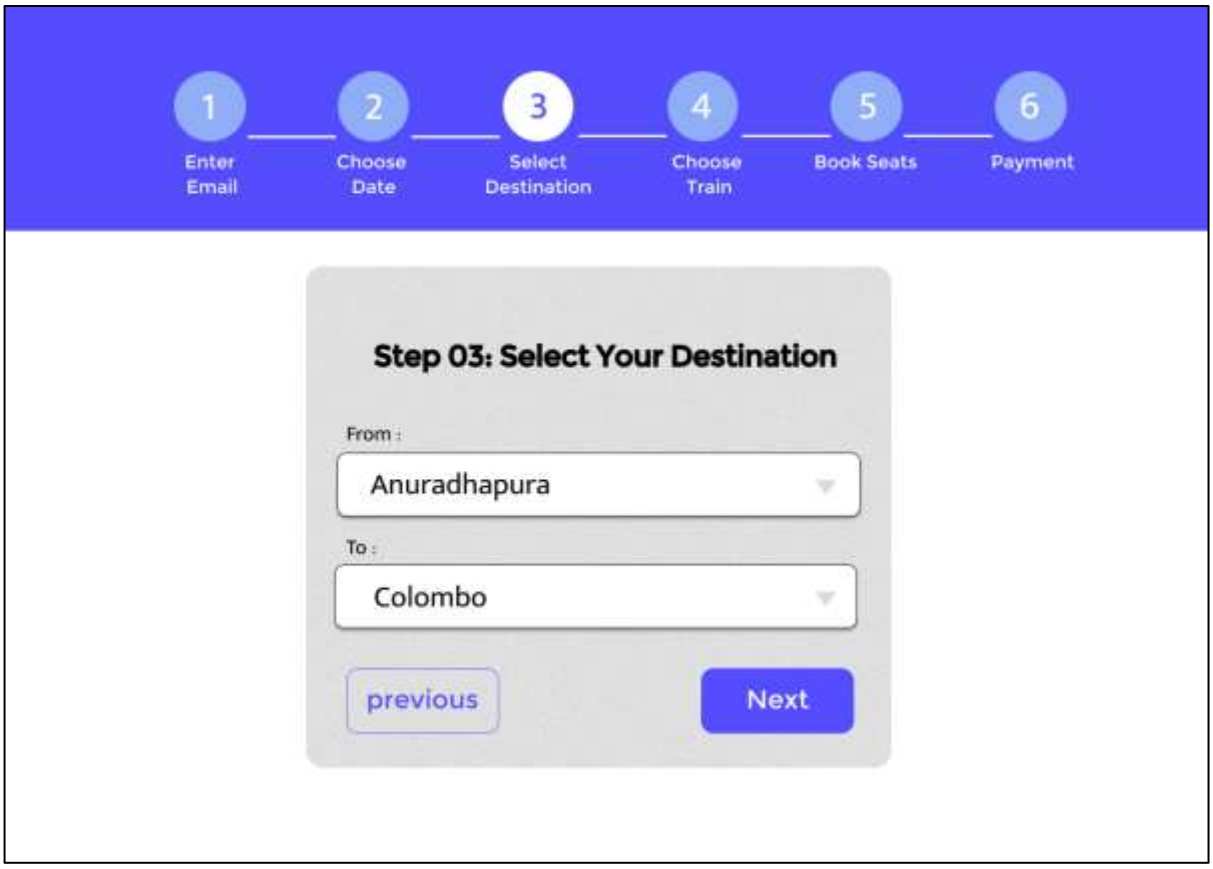


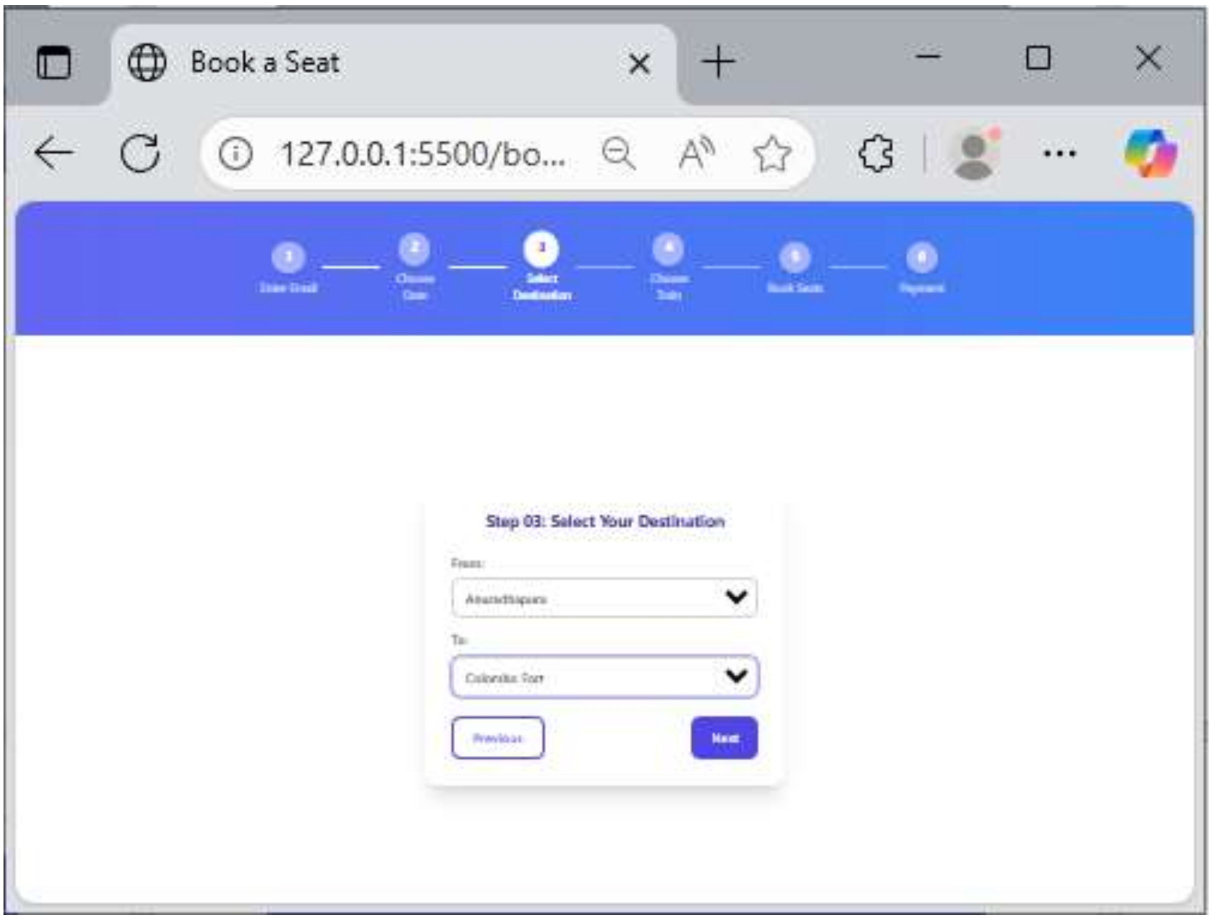
Figure 19 UI of the Booking Page Step 02 Page



Step 03: Select Your Destination

From : Anuradhapura

To : Colombo

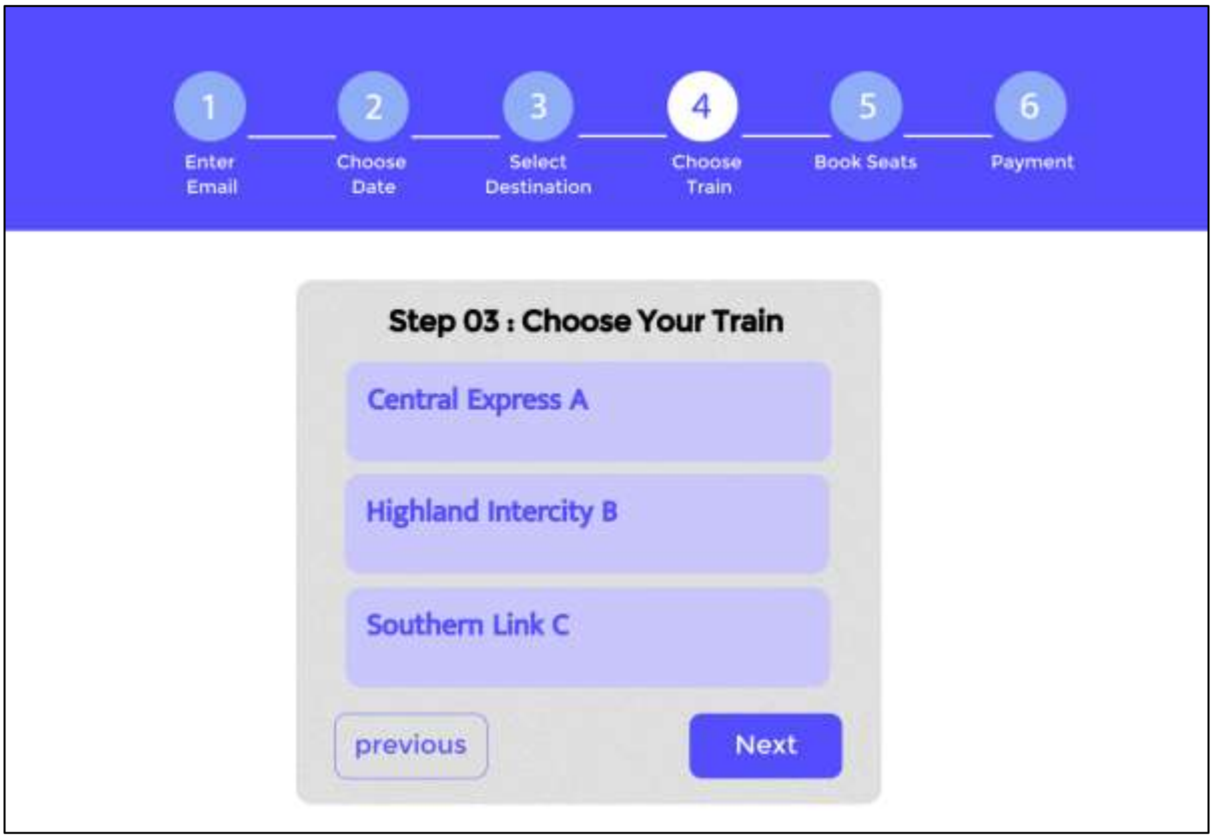


Step 03: Select Your Destination

From: Anuradhapura

To: Colombo Fort

Figure 23 UI of the Booking Step 03 Page



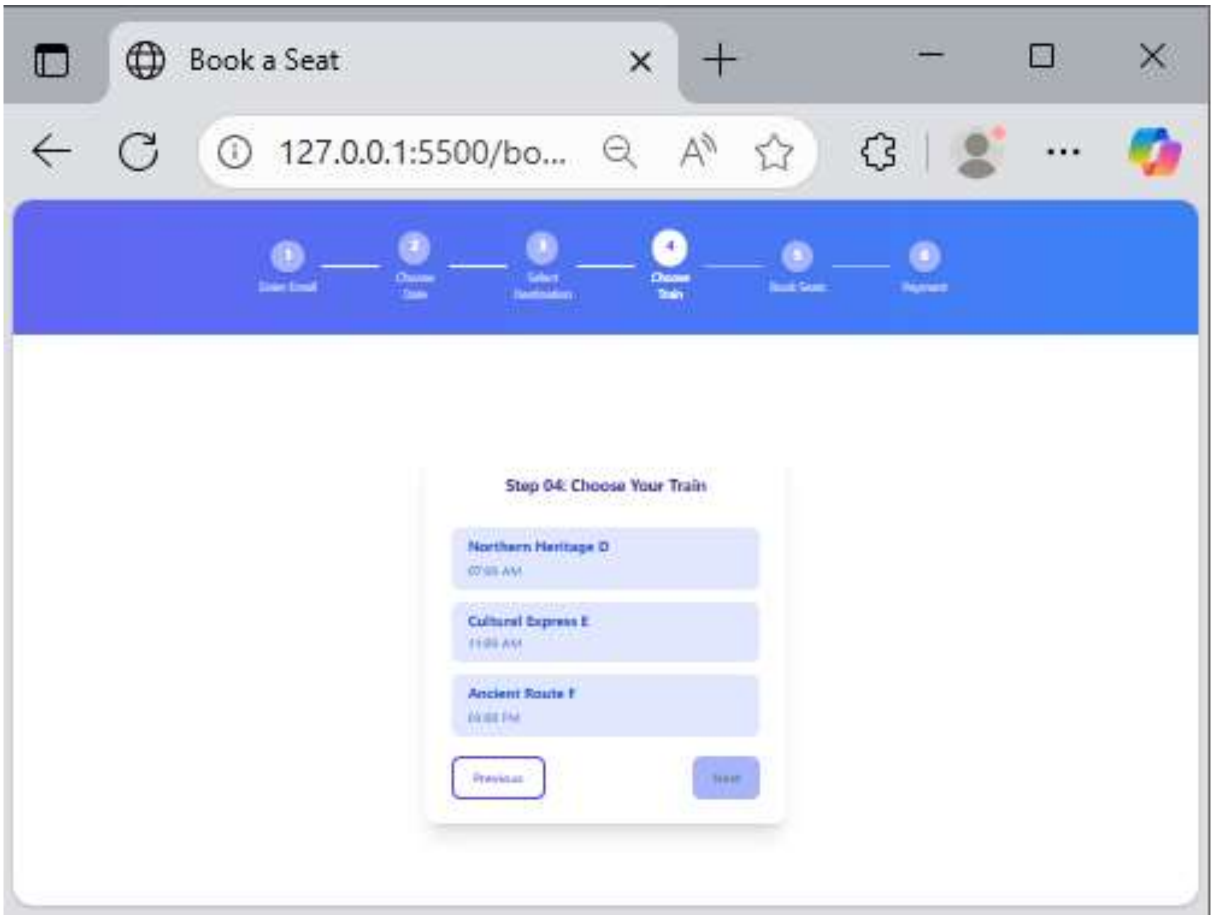
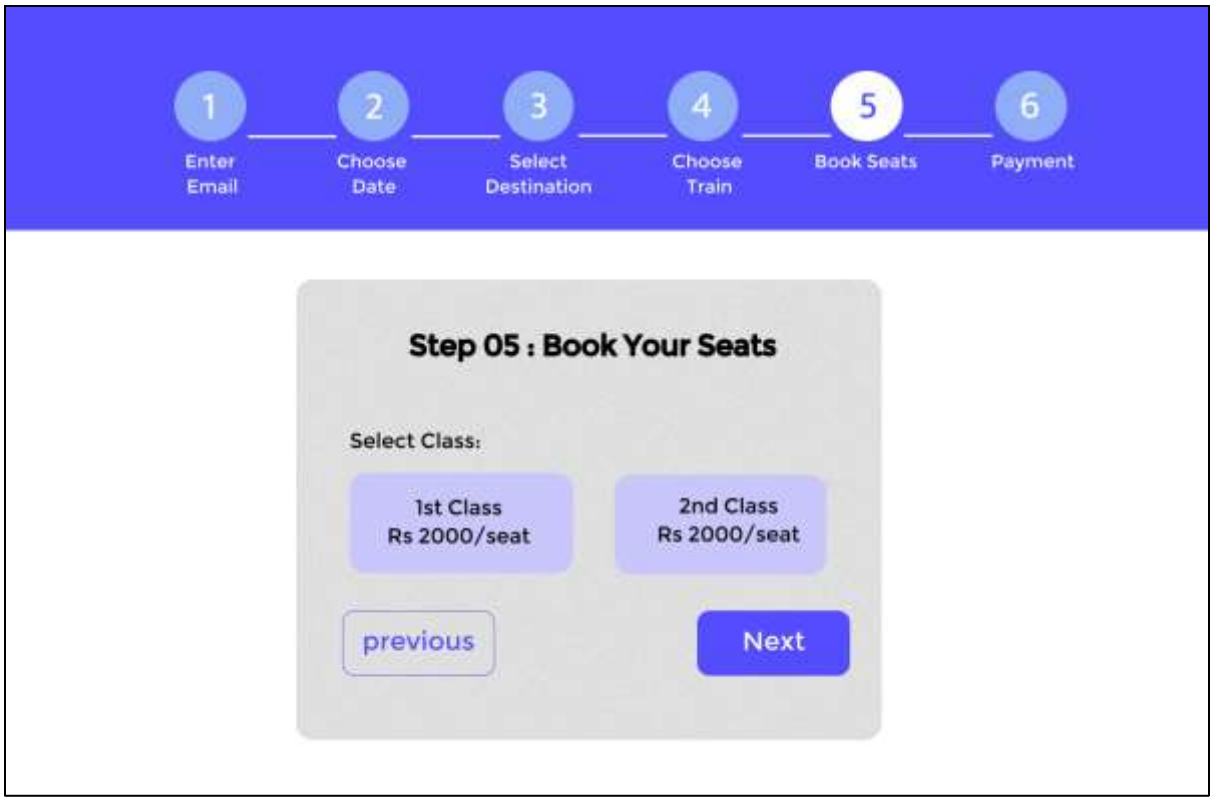


Figure 24 UI of the Booking Step 04 Page



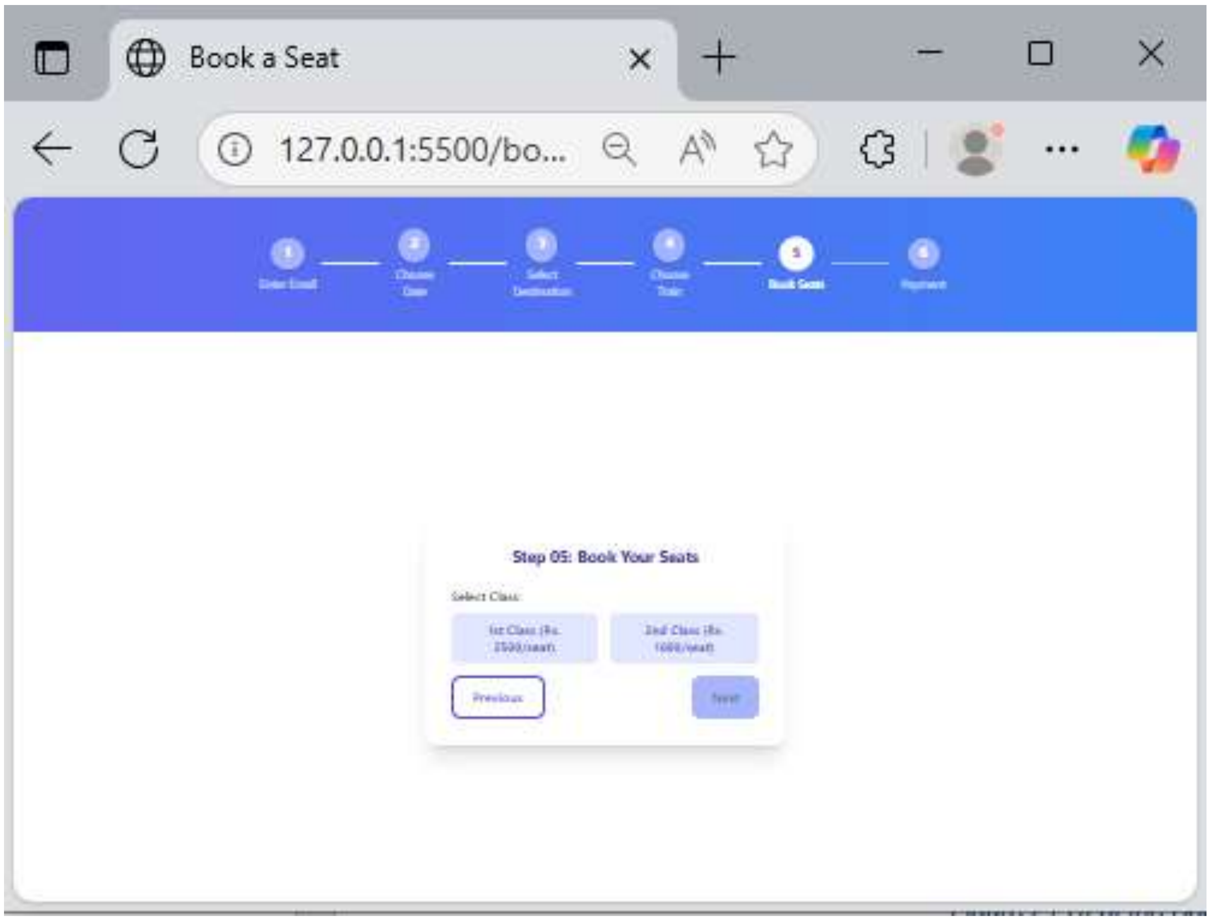


Figure 25 UI of the Booking Step 05 Page

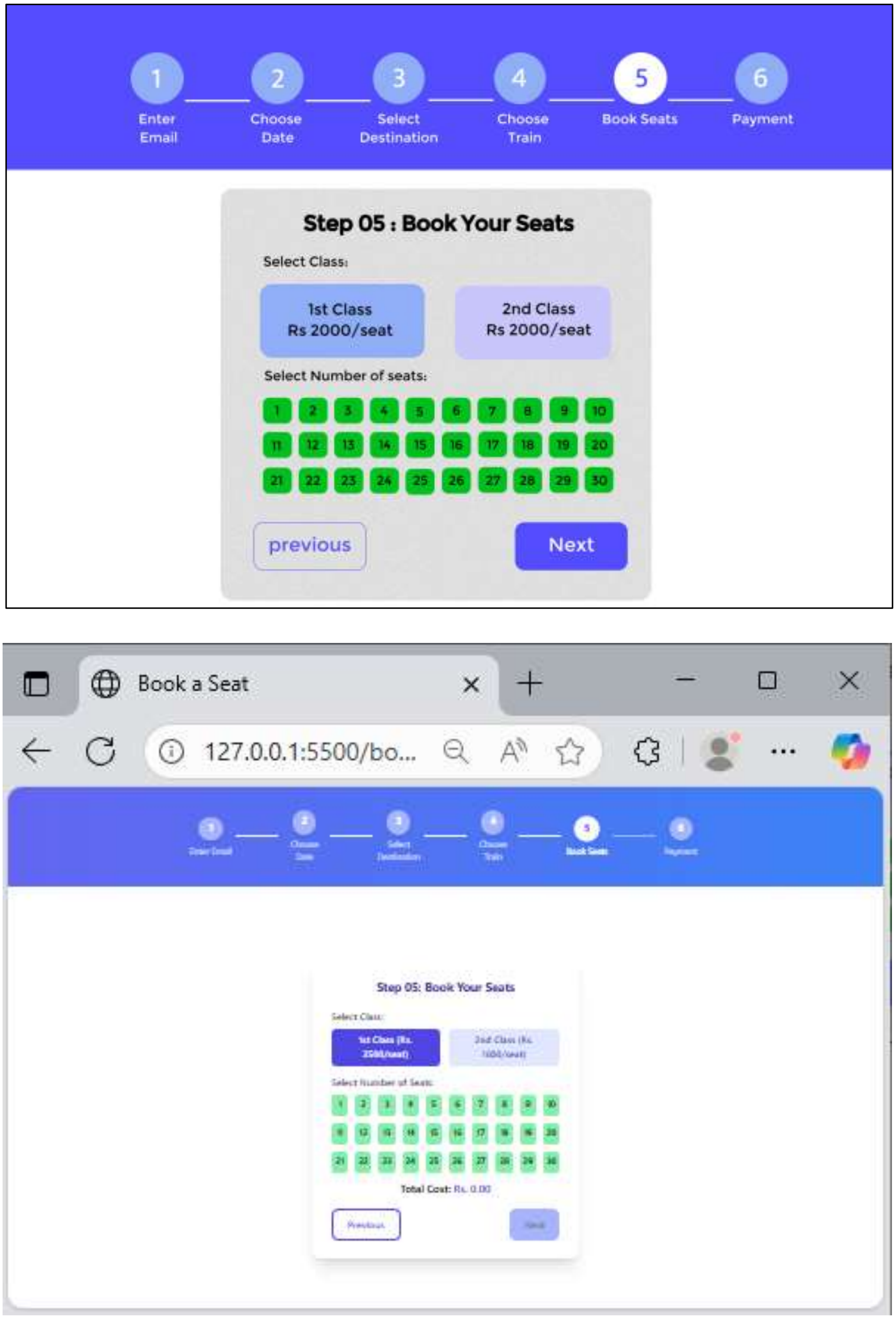
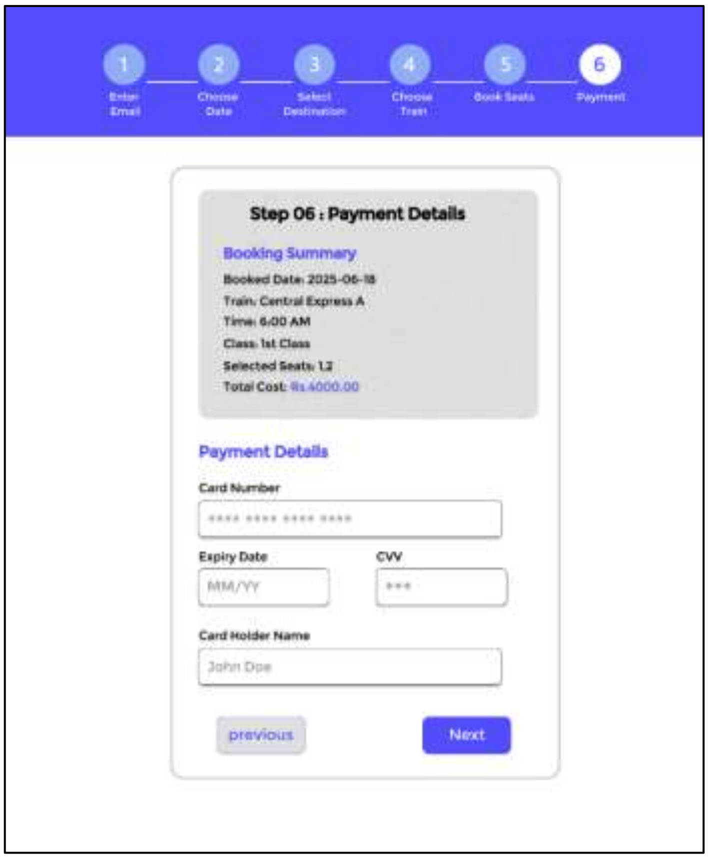


Figure 26 UI of the Booking Step 05 Page



Step 06 : Payment Details

Booking Summary

Booked Date: 2025-06-18

Train: Central Express A

Time: 6:00 AM

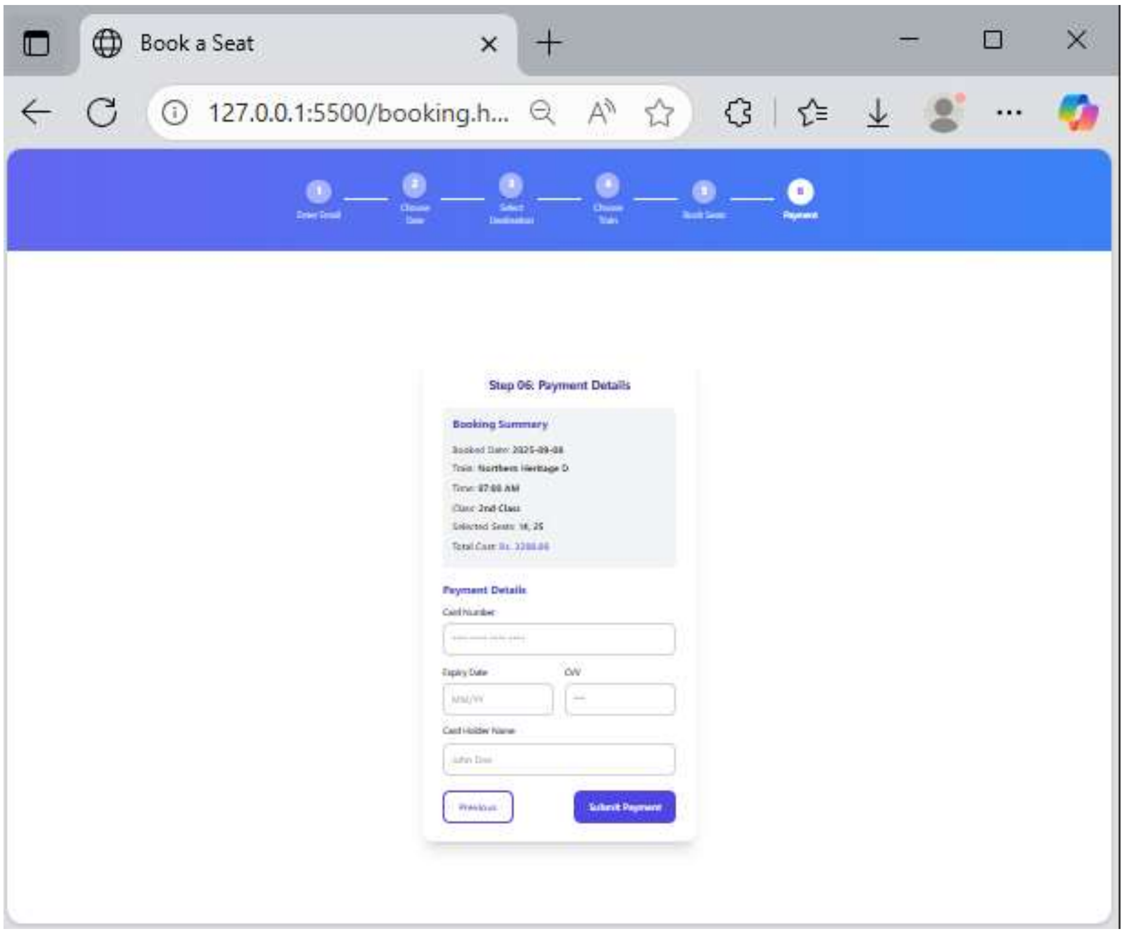
Class: 1st Class

Selected Seats: 1,2

Total Cost: Rs.4000.00

Payment Details

Card Number	**** **** **** ****
Expiry Date	MM/YY
CVV	***
Card Holder Name	John Doe



Step 06: Payment Details

Booking Summary

Booked Date: 2025-09-08

Train: Northern Heritage D

Time: 07:00 AM

Class: 2nd Class

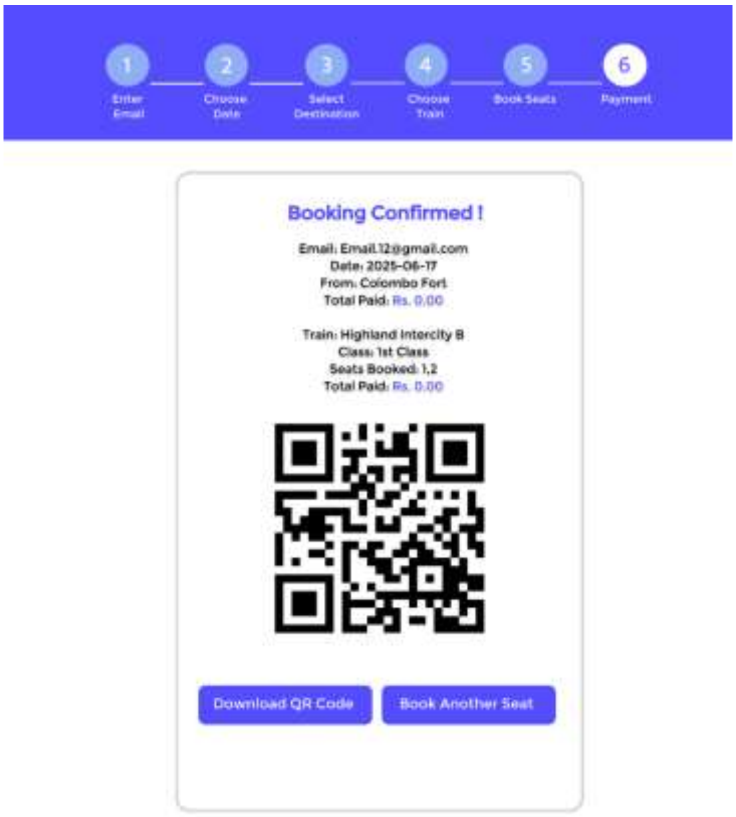
Selected Seats: 16, 25

Total Cost: Rs. 3200.00

Payment Details

Card Number	**** **** **** ****
Expiry Date	MM/YY
CVV	***
Card Holder Name	John Doe

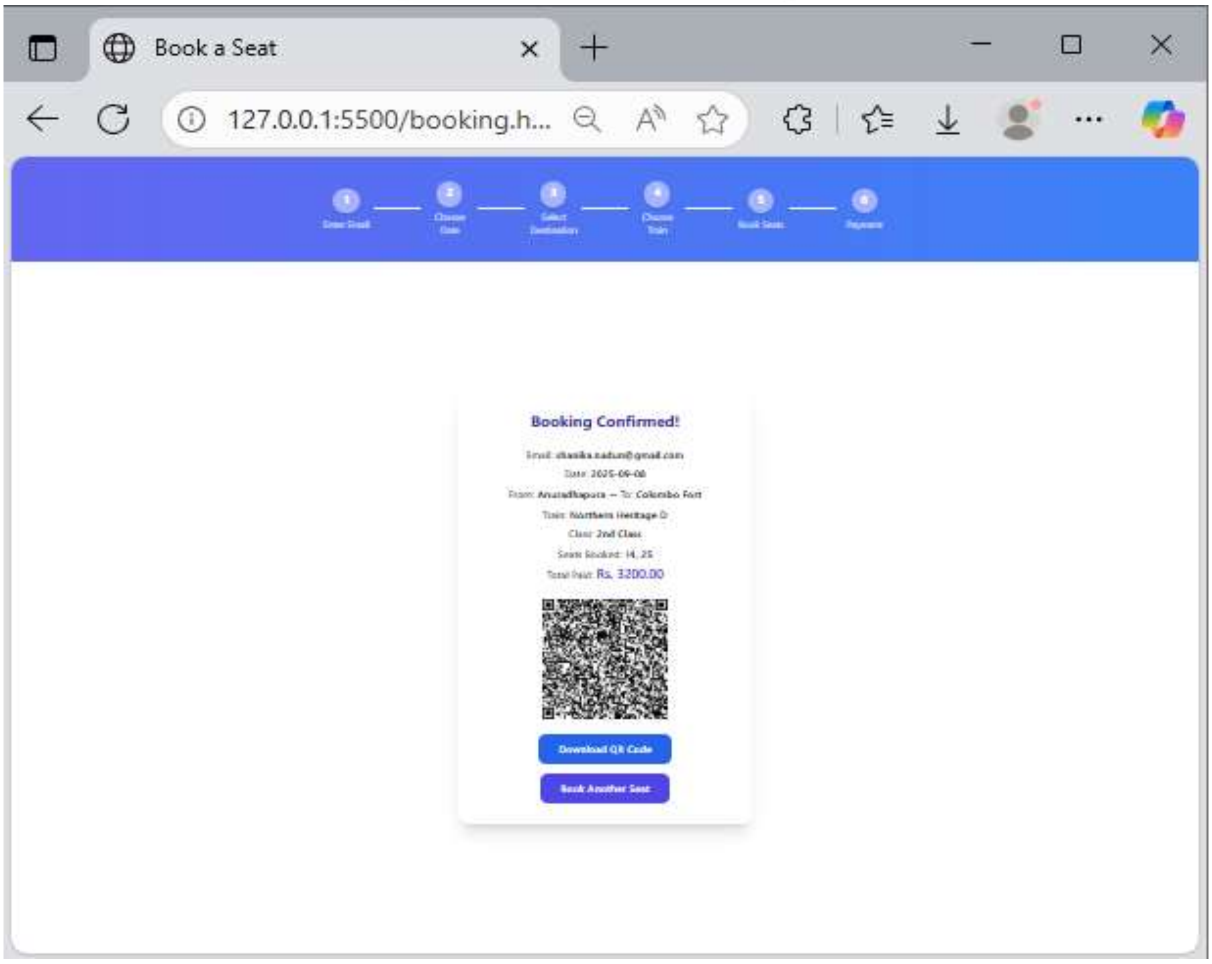
Figure 27 UI of the Booking Confirmation Page



Booking Confirmed !

Email: Email.12@gmail.com
Date: 2025-06-17
From: Colombo Fort
Total Paid: Rs. 0.00

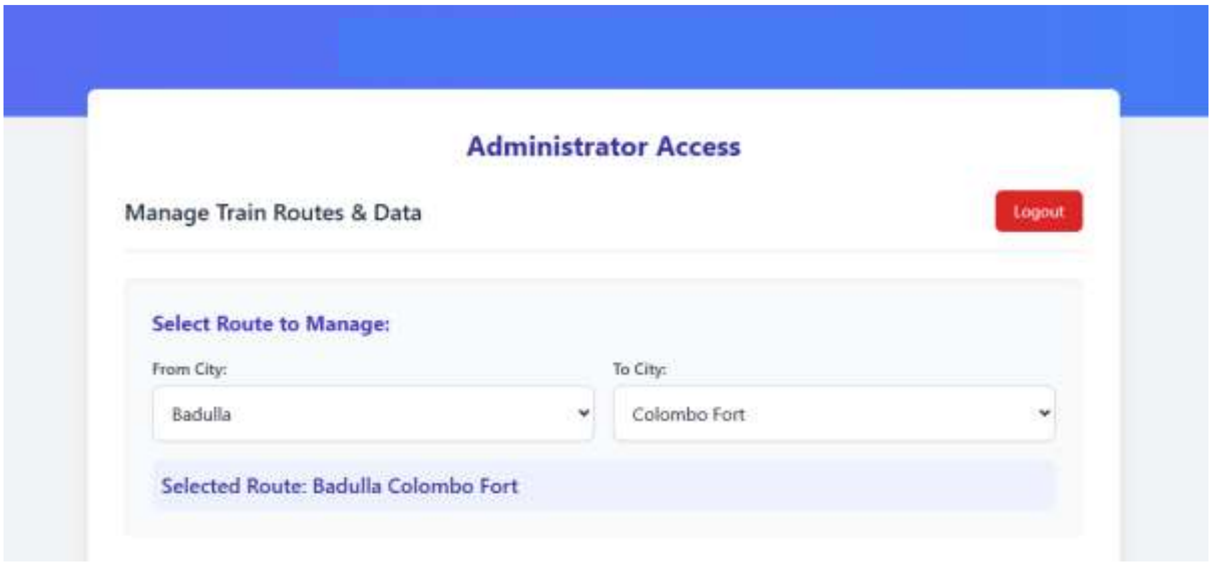
Train: Highland Intercity B
Class: 1st Class
Seats Booked: 1,2
Total Paid: Rs. 0.00



Booking Confirmed!

Email: dhanika.nadun@gmail.com
Date: 2025-09-08
From: Anuradhapura → To: Colombo Fort
Train: Northern Heritage D
Class: 2nd Class
Seats Booked: 14, 25
Total Paid: Rs. 3200.00

Figure 28 UI of the Booking Confirmation Page



Administrator Access

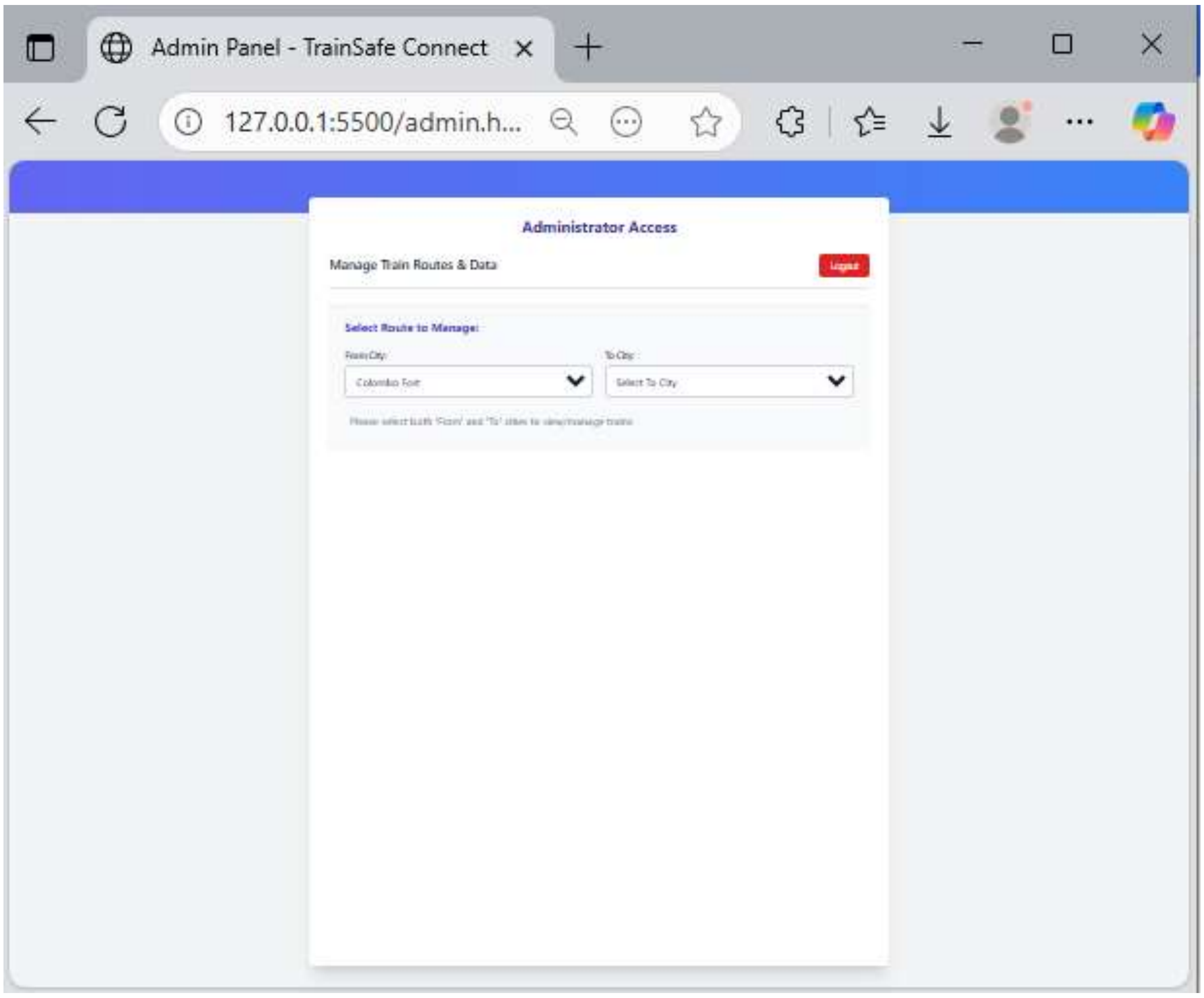
Manage Train Routes & Data

Select Route to Manage:

From City: Badulla

To City: Colombo Fort

Selected Route: Badulla Colombo Fort



Administrator Access

Manage Train Routes & Data

Select Route to Manage:

From City: Colombo Fort

To City: Select To City

Please select both 'From' and 'To' cities to view/manage trains

Figure 29 UI of the Admin Dashboard

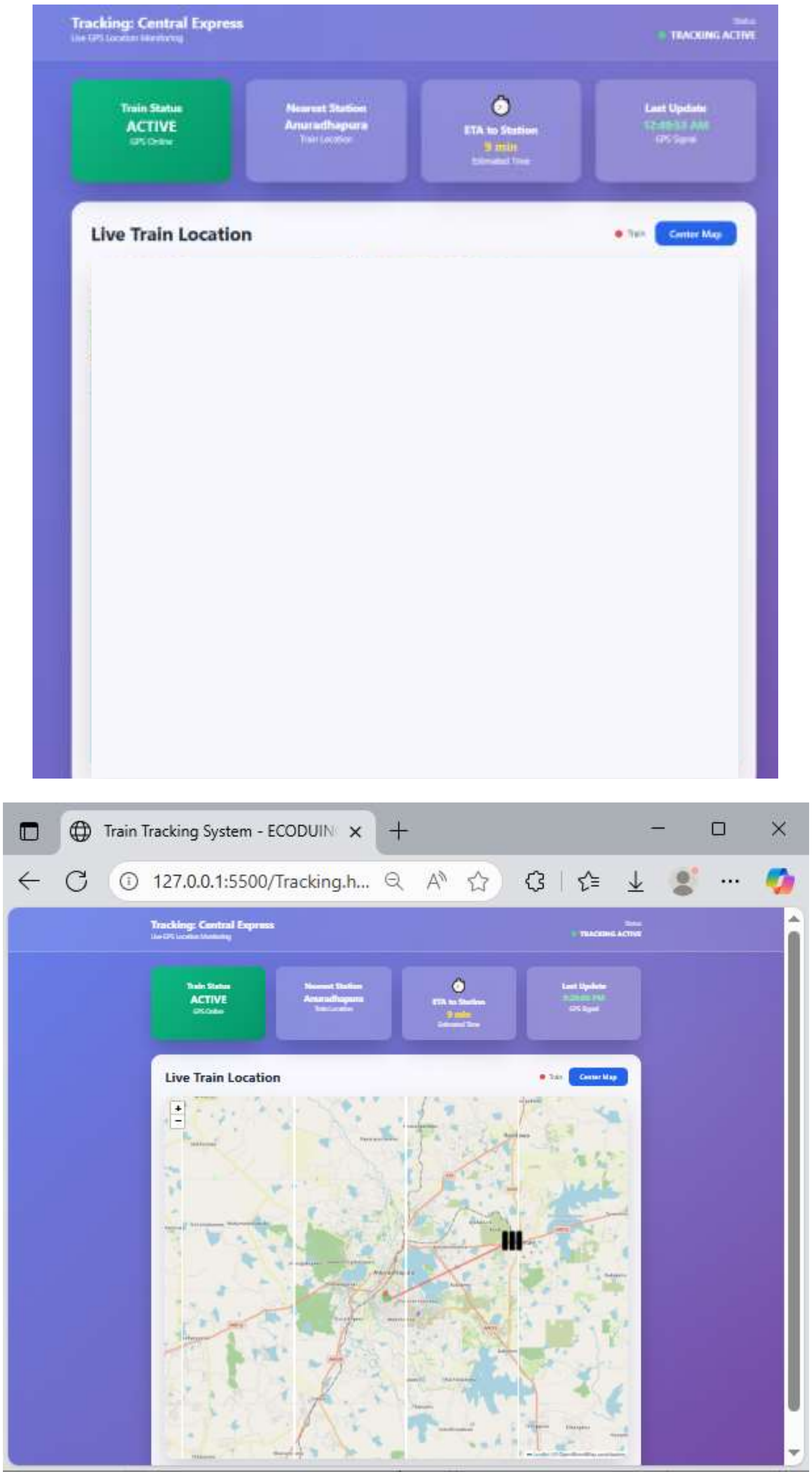


Figure 30 UI of the Tracking Page

3.2.3 Design tools, techniques, templates

The design phase of this system focused on developing easy to use accessible, and responsive interfaces for both passengers and administrative users, as well as designing reliable interaction flows between software and hardware components. The tools and techniques chosen were based on their reliability, flexibility, open-source availability, and suitability for quick prototyping and production deployment. This section describes the design tools, UX/UI approaches, and templates that were used to guide the system's front-end and backend development.

1. UI/UX Design Tools and Techniques

   This step involves the use of tools such as paper sketching and Figma. Low-resolution drafts were first generated to outline the structure of web pages such as the booking page, train timetable viewer, admin panel, and QR scanning screen. Layouts targeted mobile-first design to enable usability on smartphones, tablets, and computers. Throughout the design phase, user personas like "daily commuter," "tourist," and "station admin" were taken into consideration. The UI was created with HTML5 and CSS3, assuring semantic structure and responsive style.

2. Dashboard and Data Display Design

   A customized admin dashboard was created with functionality for viewing current train timetables. Add or remove trains from routes. Validate QR codes manually (if necessary). The dashboard layout was straightforward and informative, with tabular views, buttons, and input forms stylized with minimal CSS. The Google Maps JavaScript API was integrated with the frontend interface to graphically display train whereabouts in real time.

3. QR and PDF Ticket Design

   The reportlab library was used in Python to create styled PDF tickets. Ticket covers passenger and journey details, train name and time, QR code for boarding verification. Fonts, headings, and layout space were all changed to create a professional, readable style ideal for printing or mobile use. QR codes were created with the qrcode Python module. Each code encodes a unique string that corresponds to the booking ID. Codes were stored in PNG format and included in both the frontend for download and the PDF file distributed by email.

3.3 Data Management

3.3.1 Design tools, techniques

The Automated Train Platform Safety and Tracking System’s data management design prioritizes efficient handling of ticket bookings, train schedules, QR code generation, and email confirmations. It relies on a lightweight JSON-based file system and structured REST APIs for backend operations, while real-time GPS tracking is managed through Firebase. All booking-related data is processed server-side using Python and Flask.

1. Data Storage Mechanism

The system employs a file-based storage approach, with all train-related information including IDs, names, schedules, and fares stored in a JSON file called train_data.json. At program startup, the load_train_data() function loads this file into memory, while save_train_data() commits any changes. This method suits small-scale applications by avoiding the overhead of a full database, while maintaining readability and flexibility. During execution, train data is held in a global Python dictionary named train_data. New entries (via POST /api/trains) and updates (via PUT requests) are first modified in this dictionary and then persisted to the JSON file.

2. Data Manipulation and Processing Techniques

All operations on train schedule data are performed through RESTful API endpoints implemented with Flask. For instance, GET /api/trains retrieves all train data, POST /api/trains adds a new train to a route, and DELETE /api/trains/<id> removes a train. These endpoints provide consistent, structured JSON responses for frontend clients, administrative panels, and other services. HTTP status codes (200 OK, 400 Bad Request, 409 Conflict, etc.) along with JSON messages indicate the success or failure of requests.

3. QR Data Handling, Email Formatting, Admin Login, and Data Security

Passenger booking information such as email, selected date, train, class, and cost—is temporarily processed during QR code generation and email delivery via the /api/send-email route. This data is used to create a visually formatted PDF ticket using ReportLab, generate a QR code via the Python qrcode module, and compose an email containing both the ticket and journey details. While this information is not permanently stored, future versions could implement persistent storage using SQLite or PostgreSQL.

Data sent to the /api/send-email endpoint is validated to ensure required fields (e.g., qr_data, recipient_email) are present and properly structured for PDF and email content. SMTP credentials are protected to prevent sensitive information from being hardcoded. Administrative authentication is handled using environment variables (ADMIN_USERNAME, ADMIN_PASSWORD), with the /api/admin/login route verifying these credentials before granting access. Passwords are currently stored in plain text within the .env file for simplicity, but production systems should employ hashed and salted passwords using libraries like bcrypt.

3.3.2 Conceptual database design

The conceptual database design defines the high-level structure of the system's data entities, including their attributes and relationships. It ensures data consistency, integrity, and logical grouping of information for various modules such as train ticketing, QR code verification, and email notifications.Although the current system stores data in JSON files, this conceptual model prepares the system for a future move to a relational database system, such as SQLite or PostgreSQL, via an ORM such as SQLAlchemy in Flask.

The following are the key entities identified in the system, along with their attributes.

1. Passenger

Table 9 Attribute Table for Passenger

Attributes	Description
PassengerID	Primary Key
First Name	Passenger’s first name.
Last Name	Passenger’s last name.
Email	Email address.

2. Train

Table 10 Attribute Table for Train Entity

Attributes	Description
TrainID	Primary Key
TrainName	Train’s name.
TrainNumber	Train’s number.
Route	Train’s route from and to.
Time	Time of departure.
Price	Price of seats.

3. Booking

Table 11 Attribute Table for Booking Entity

Attributes	Description
BookingID	Primary Key
PassengerID	Foreign Key
TrainID	Foreign Key
BookingDate	Date of booking was done.
TravelDate	Date of travel.
Class	Booked class.
TotalAmount	Total Amount paid for seats.
SelectedSeats	Selected seats by the passenger.

4. QRCode

Table 12 Attribute Table for QRCode Entity

Attributes	Description
QRCodeID	Primary Key
BookingID	Foreign Key
QRCodeData	Encoded booking string.

5. Admin

Table 13 Attribute Table for Admin Entity

Attributes	Description
AdminID	Primary Key
Username	Username of the admin.
Password	Password to login.

6. GPS Tracking

Table 14 Attribute Table for GPS Tracking Entity

Attributes	Description
TrackingID	Primary Key
TrainID	Foreign Key
Latitude	Latitude code of the train location.
Longitude	Longitude code of the train location.
Speed	Speed of the train.
Timestamp	Timestamp of the train travel happening.

7. ETA Calculation

Table 15 Attribute Table for ETA Calculation Entity

Attributes	Description
ETAID	Primary Key
TrainID	Foreign Key
CalculatedETA	The time that calculated and showing.
CalculationTime	The time that always calculating.

Relationships are created as below,

- One Passenger – Many Bookings (One to Many Relationship) which means each passenger can book multiple tickets.
- One Train – Many Bookings (One to Many Relationship) which means a train can have many bookings.
- One Booking – One QRCode (One to One Relationship) which means each booking will generate one QR code.
- One Train – Many GPSTracking (One to Many Relationship) this is because the location must constantly update to show accurate location of the train.

- One Train – Many ETACalculation (One to Many Relationship) this is because like in the location tracking it should also continuously update.

Data constraints and rules are following,

- Primary Keys – Each entity has a unique identifier (PassengerID, TrainID, BookingID, etc.).
- Foreign Keys – Maintain referential integrity between Passenger, Train and Booking.
- Seats should be stored in a format that allows parsing (e.g., comma-separated or JSON list).
- Email addresses must follow proper format.
- Qr data must be unique per booking and the admin username and password were predefined and stored in the database and it will never change unless a security issue occurs.

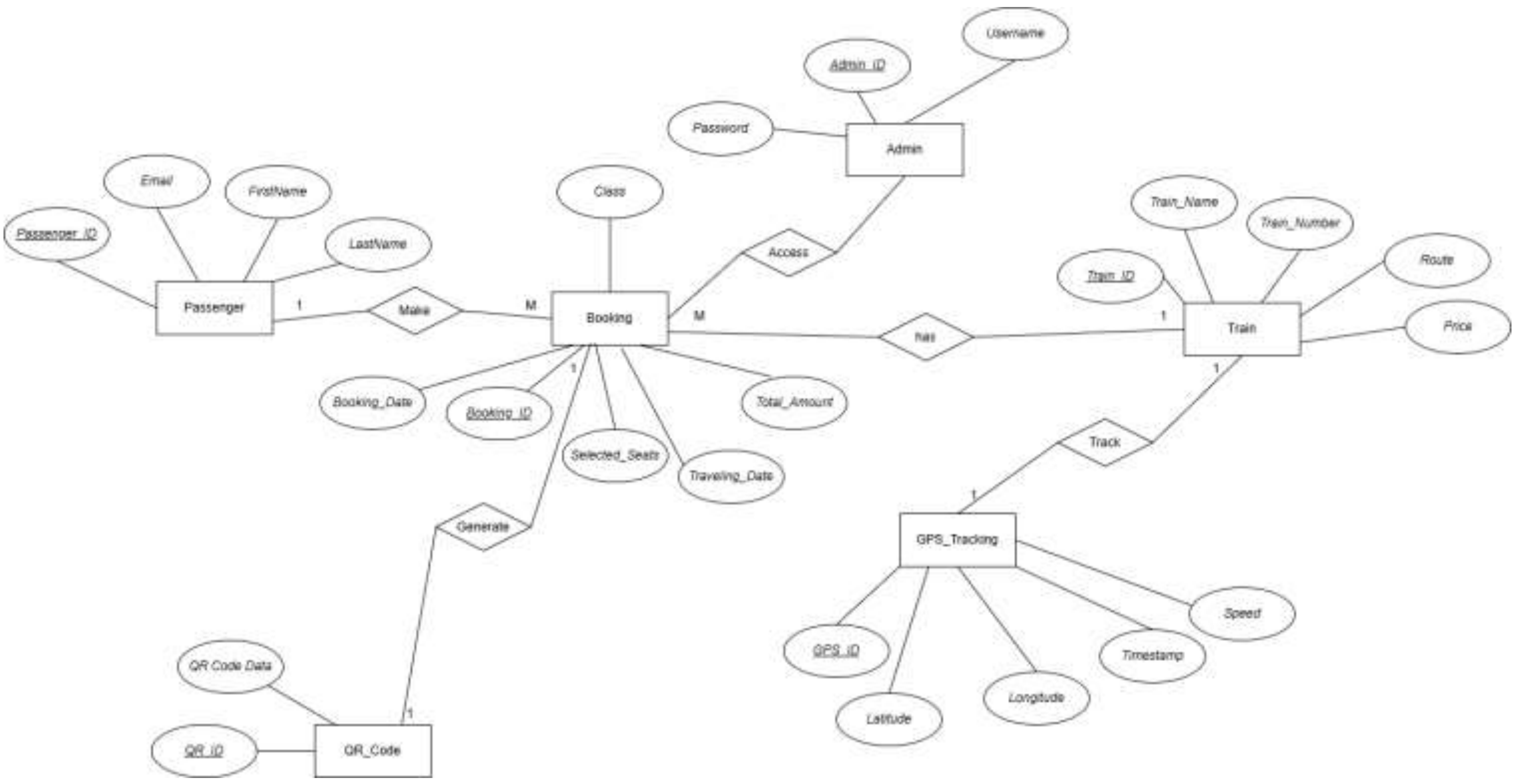


Figure 31 ER diagram of the System

3.3.3 Logical database design and schema refinement

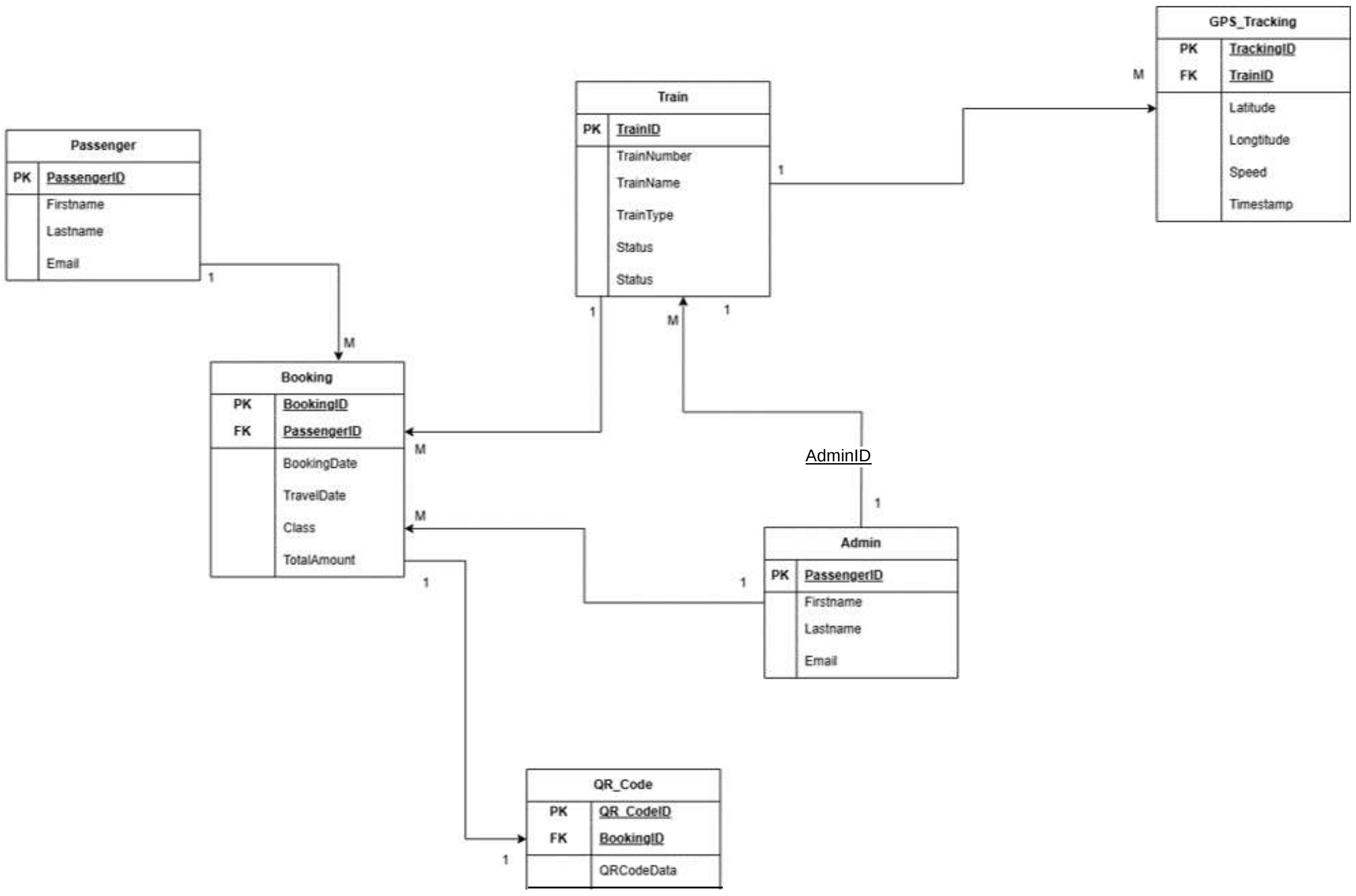
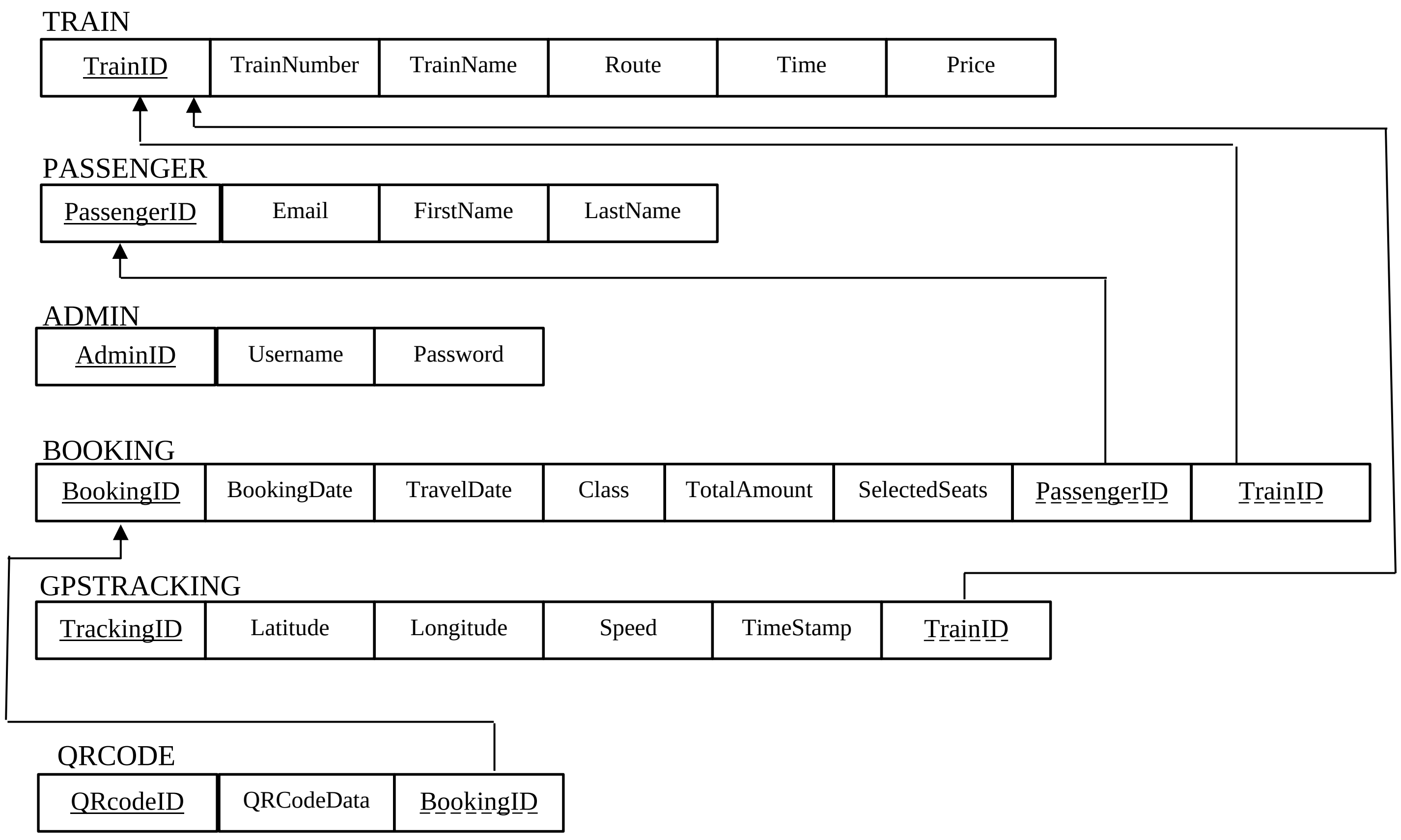


Figure 32 EER Diagram of the System


TRAIN
TrainID
TrainNumber
TrainName
Route
Time
Price
PASSENGER
PassengerID
Email
FirstName
LastName
ADMIN
AdminID
Username
Password
BOOKING
BookingID
BookingDate
TravelDate
Class
TotalAmount
SelectedSeats
PassengerID
TrainID
GPSTRACKING
TrackingID
Latitude
Longitude
Speed
TimeStamp
TrainID
QRCODE
QRcodeID
QRCodeData
BookingID

3.3.4 Physical database design

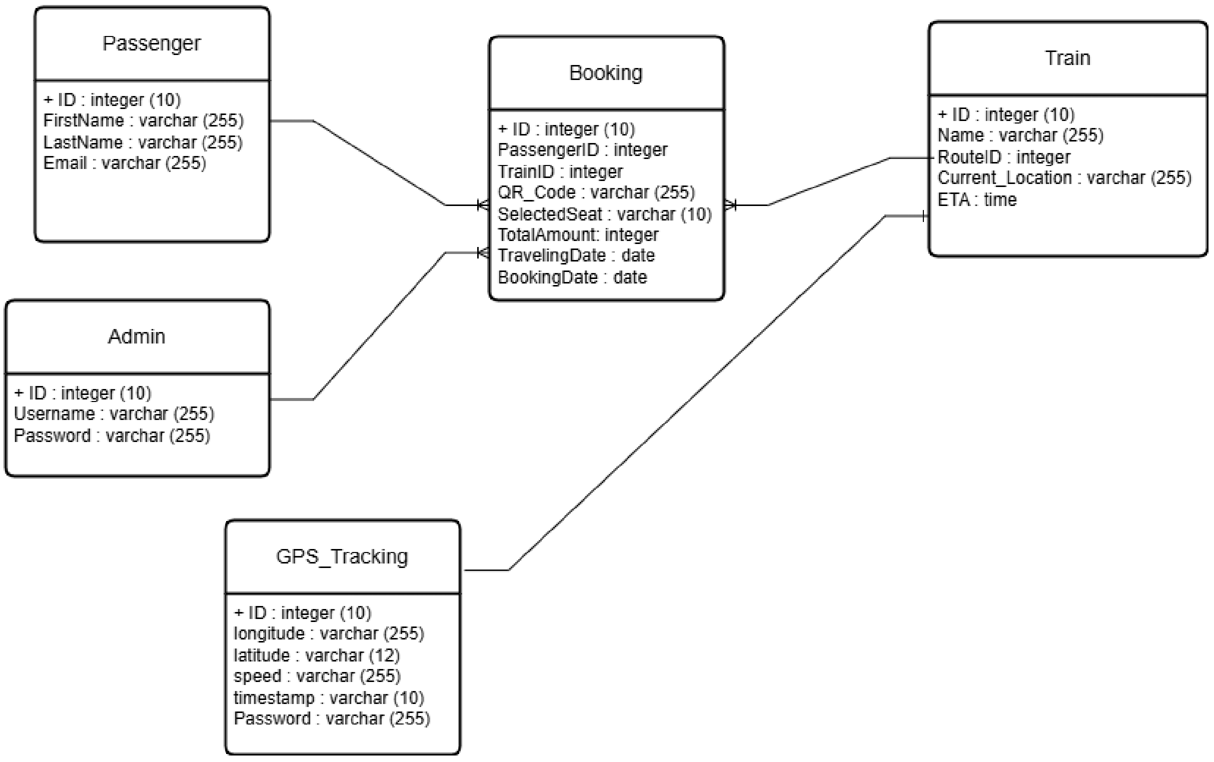


Figure 33 Physical Database Design of the System

3.4 Hardware Design (if available)

1. Railway Cross Road System

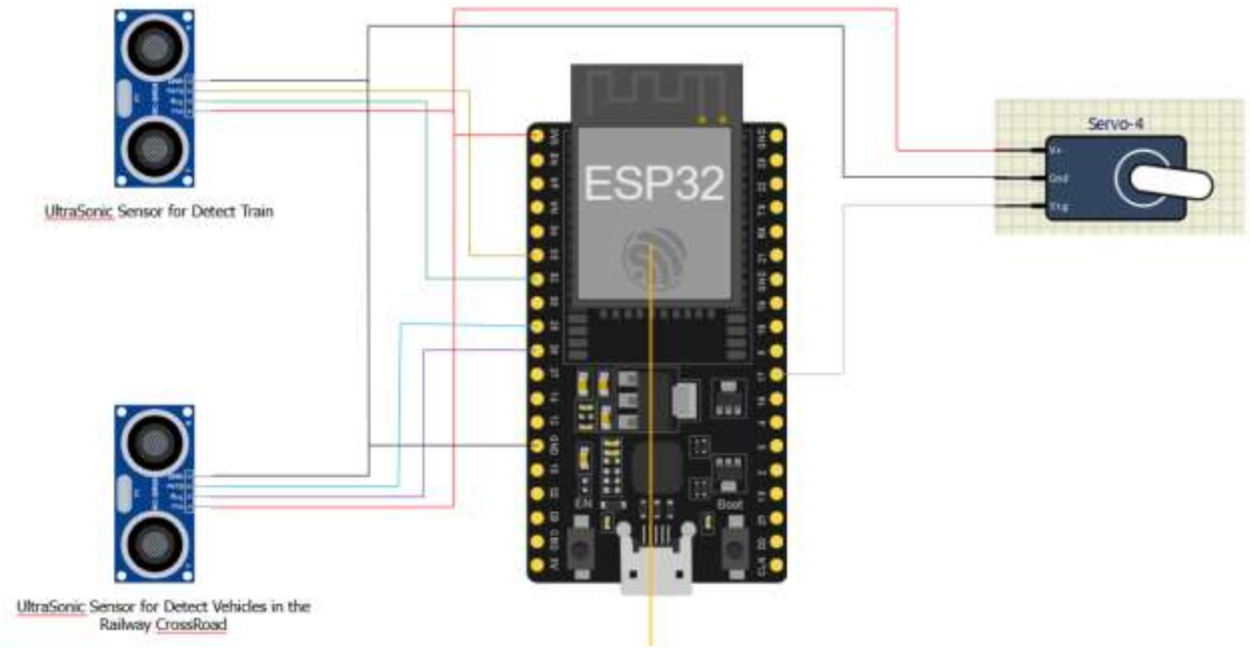

UltraSonic Sensor for Detect Train
ESP32
Servo-4
UltraSonic Sensor for Detect Vehicles in the Railway CrossRoad

Figure 34 Hardware Design of the Railway Cross Road System

This figure shows the hardware implementation of the Railway Crossroad System, which uses an ESP32 microcontroller to automate the functioning of a railway crossing gate. One HC-SR04 ultrasonic sensors are linked to the ESP32: detect an incoming train, while the other detects vehicles on the railway crossing. These sensors always analyze their surroundings and transmit distance information to the ESP32. When the sensor detects a train approaching within a predefined range and no vehicle is detected on the tracks, the ESP32 activates a servo motor that serves as a crossing gate barrier. When the train has passed and the track is clear, the ESP32 releases the gate by rotating the servo back. This system keeps safety by preventing cars from being on the track when a train is nearby, fully automating crossing gate control with no user intervention, and can be expanded to interface with the main Train System via wireless modules for more intelligent coordination.

2. Platform Gate System

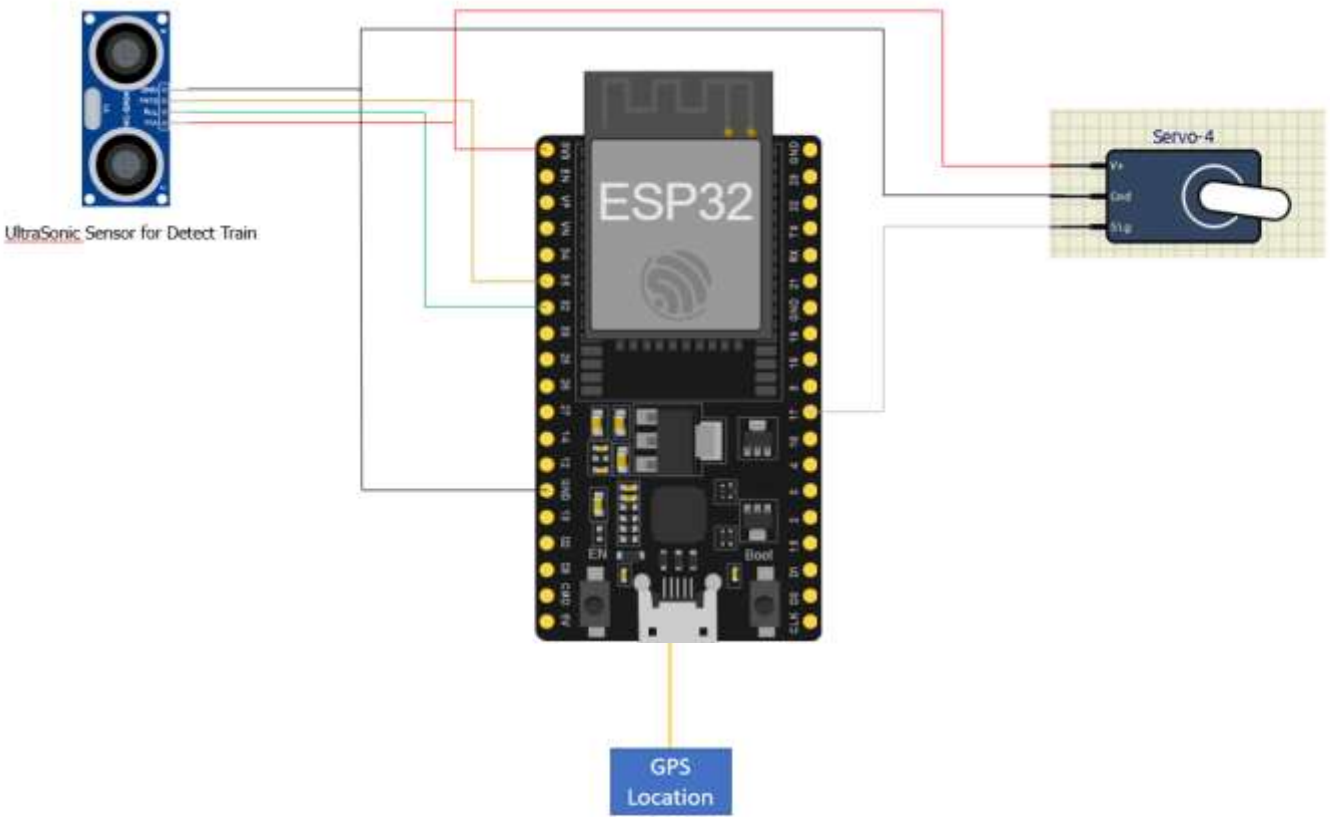


Figure 35 Hardware Design of the Platform Gate System

This figure illustrates the hardware setup for an Automated Platform Gate System, which uses an ESP32 microcontroller, an ultrasonic sensor, and a servo motor to control the opening and closing of gates at a train station platform. In this system, an HC-SR04 ultrasonic sensor is placed along the track to detect the presence of an incoming train. When the sensor detects a train within a specific distance and GPS data comes to and halt, it transmits a signal to the ESP32. The ESP32 interprets this signal and engages the servo motor, which controls the physical gate mechanism opening or closing the platform gate dependent on train proximity. When the train departs and the sensor no longer detects its presence, the ESP32 instructs the servo to reset the gate to its original position. This basic yet effective solution automates passenger control at platforms, ensuring safety by only allowing access while no trains are in action. The technology can also be extended to interface with central train tracking systems, allowing for more synchronized and intelligent platform operations.

3. QR based Boarding System

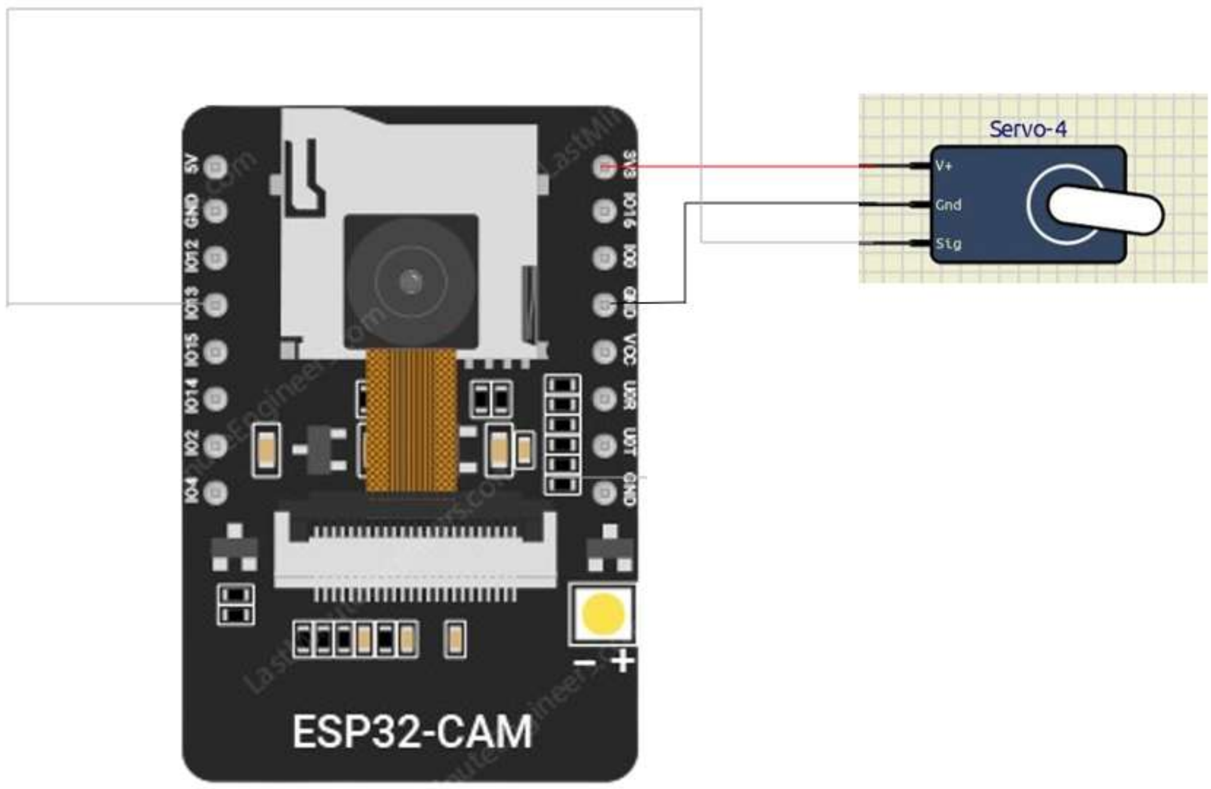


Figure 36 Hardware Design of the QR based Boarding System

This figure shows the hardware setup for a QR-Based Boarding System, which includes an ESP32-CAM module and a servo motor for secure, automated entrance control at train boarding gates. The ESP32-CAM includes an embedded camera that scans passenger QR codes at the gate. After scanning a QR code, the ESP32-CAM compares it to saved or backend booking data (as illustrated in the system architecture using Flask). If the QR code is correct, the ESP32-CAM sends a signal to the attached servo motor, which turns to unlock or open the boarding gate, allowing the passenger to enter. If the QR code is invalid or not recognized, the servo remains inactive, keeping the gate closed. This technology ensures that only authorized people can board, which improves security and streamlines admission without the need for manual verification. It is a critical component of the wider smart railway system, ensuring seamless interface with backend booking and validation services.

4. GPS and ETA System

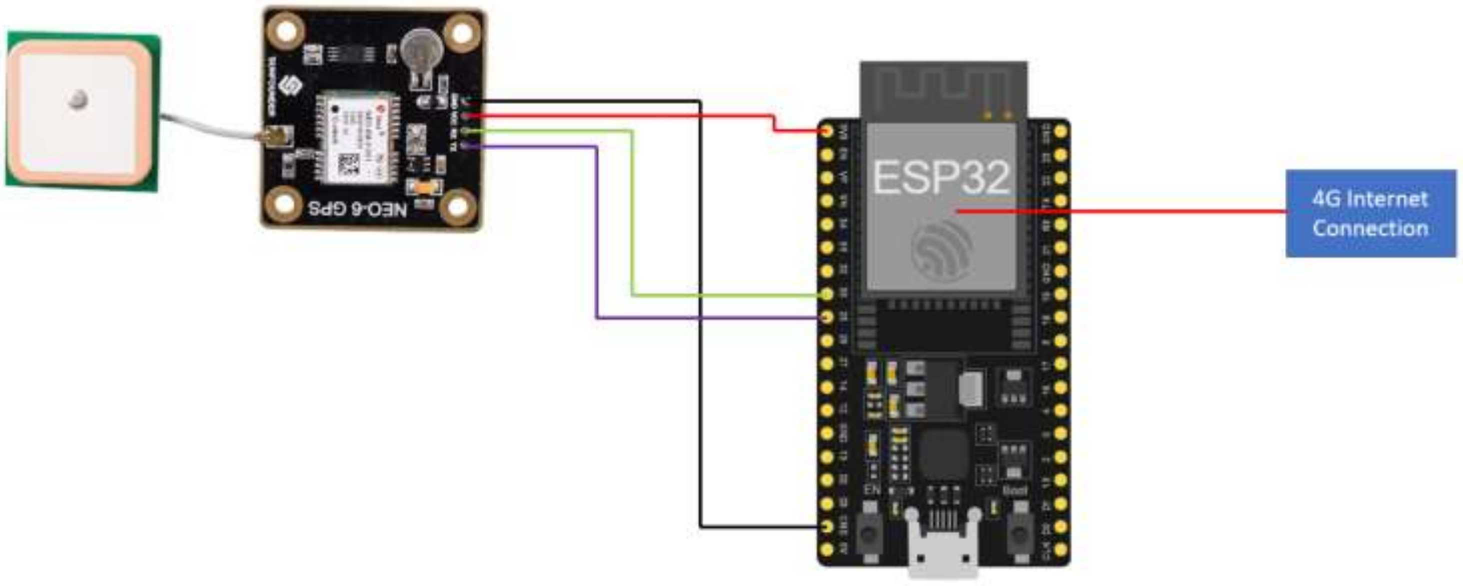


Figure 37 Hardware Design of the GPS and ETA System

This figure shows the hardware setup for a GPS Tracking System in the smart railway infrastructure, which commonly includes an ESP32 microcontroller and a NEO-6M GPS module. In this system, the NEO-6M GPS module connects to the ESP32, which continuously collects real-time GPS coordinates of a moving train. The ESP32 processes these coordinates (latitude and longitude), which can then be used to determine the train's estimated time of arrival (ETA) or track its whereabouts on a live map. The ESP32 may then send this information to a Firebase database, which maintains and updates train location information, allowing the web interface to provide real-time tracking for passengers and control systems.

4 Testing and evaluation

4.1 Testing plan

The goal of the testing phase was to make sure that the TrainSync system's software and hardware operated as planned in practical settings. Acceptance testing, system testing, integration testing, and unit testing were all used. The two main types of testing were software testing for web-based and backend modules and physical testing for embedded hardware and the Internet of Things.

Validating user experience, responsiveness, safety, and dependability were the main goals of the plan. Low-cost testing equipment (ESP32 boards, ultrasonic sensors, servo motors, GPS modules, and ESP32-CAM), internet access for data transfer, and cloud services like Firebase were the primary expenses. Over the course of two months, testing was carried out in several iteration cycles to improve performance and accuracy.

4.2 Testing

4.2.1 Unit testing

1. IoT and Embedded part testing

- Ultrasonic Sensor Unit Testing

Table 16 Positive Test Case for Ultrasonic Sensor

Test Scenario ID	PT-Unit-US-1		Test Case ID	US-1A		
Test Case Description	Object Detection within Range		Test Priority	High		
Pre - Requisite	ESP32 and Ultrasonic sensor connected, powered, Arduino IDE open at 9600 baud		Post - Requisite	Reset system after test		
Execution Steps :						
S.no	**Action**	**Inputs**	**Expected Output**	**Actual Output**	**Test Result**	**Test Comment**
01	Upload ultrasonic test code to ESP32 via Arduino IDE	Code uploaded	Code runs successfully	Code runs successfully	Pass	Program loaded
02	Place a solid object 50 cm in front of sensor	Object at 50 cm	Sensor should detect 50 cm (2 cm tolerance)	Sensor detected 49.8 cm	Pass	Accurate detection
03	Observe output in Serial Monitor at 9600 baud	Serial output	Distance displayed on monitor	Distance displayed as 50 cm	Pass	Positive case validated

Table 17 Negative Test Case for Ultrasonic Sensor

Test Scenario ID	PT-Unit-US-1		Test Case ID	US-1B		
Test Case Description	No object in Range		Test Priority	High		
Pre - Requisite	ESP32 and Ultrasonic sensor connected, powered, Arduino IDE open at 9600 baud		Post - Requisite	Reset system after test		
Execution Steps :						
S.no	**Action**	**Inputs**	**Expected Output**	**Actual Output**	**Test Result**	**Test Comment**
01	Upload ultrasonic test code to ESP32 via Arduino IDE	Code uploaded	Code runs successfully	Code runs successfully	Pass	Program loaded
02	Keep sensor facing empty space (no object within 400 cm)	No object	Output should display “Out of Range” (>400 cm)	Displayed “Out of Range”	Pass	Correct negative behavior
03	Observe output in Serial Monitor at 9600 baud	Serial output	“Out of Range” message displayed	Message shown correctly	Pass	Negative case validated

- Servo Motor Unit Testing

Table 18 Positive Test Case for Servo Motors

Test Scenario ID	PT-Unit-SM-1	Test Case ID	SM-1A
Test Case Description	Servo rotates to valid angle	Test Priority	High
Pre - Requisite	ESP32 and Servo connected, powered, Arduino IDE open at 9600 baud	Post - Requisite	Reset servo to 0° after test
Execution Steps :			

S.no	**Action**	**Inputs**	**Expected Output**	**Actual Output**	**Test Result**	**Test Comment**
01	Upload servo motor test code to ESP32 via Arduino IDE	Code uploaded	Code runs successfully	Code runs successfully	Pass	Program loaded
02	Send command servo.write(90) to servo motor	Command 90°	Servo rotates smoothly to 90°	Servo rotated to 90°	Pass	Movement Correct
03	Observe servo position visually	Servo position	Arm holds position at 90°	Arm held at 90°	Pass	Positive case validated

Table 19 Negative Test Case for Servo Motors

Test Scenario ID	PT-Unit-SM-1	Test Case ID	SM-1B
Test Case Description	Servo rotates beyond limit	Test Priority	High
Pre - Requisite	ESP32 and Servo connected, powered, Arduino IDE open at 9600 baud	Post - Requisite	Reset servo to 0° after test
Execution Steps :			

S.no	**Action**	**Inputs**	**Expected Output**	**Actual Output**	**Test Result**	**Test Comment**
01	Upload servo motor test code to ESP32 via Arduino IDE	Code uploaded	Code runs successfully	Code runs successfully	Pass	Program loaded
02	Send command servo.write(200) to servo motor	Command 200°	Servo should stop at max limit (180°)	Servo stoped at max limit (180°)	Pass	Movement Correct
03	Observe servo response visually	Servo position	Arm does not move beyond 180°	Arm remained at 180°	Pass	Negative case validated

- Neo6M module unit test

Table 20 Testcase for Neo6M Module

Test Scenario ID	PT-Unit-GPS-1	Test Case ID	GPS-1A
Test Case Description	GPS detects valid location in open sky	Test Priority	High
Pre - Requisite	ESP32 and NEO-6M GPS module connected, powered, antenna in open sky, Arduino IDE at 9600 baud	Post - Requisite	Reset GPS after test, clear previous logs
Execution Steps :			

S.no	**Action**	**Inputs**	**Expected Output**	**Actual Output**	**Test Result**	**Test Comment**
01	Upload GPS test code to ESP32 via Arduino IDE	Code uploaded	Code runs successfully	Code runs successfully	Pass	Program loaded
02	Place GPS antenna outdoors (open sky)	Open sky access	GPS acquires satellite fix within 30–60s	GPS fix acquired in ~40s	Pass	Satellite lock successful
03	Observe output on Serial Monitor at 9600 baud	Serial output	Latitude, Longitude, Speed displayed (±10m accuracy)	Correct coordinates shown	Pass	Positive case validated

- ESP – 32 CAM Module

Table 21 ESP32 CAM Module Testcase

Test Scenario ID	PT-Unit-CAM-1	Test Case ID	CAM-1A
Test Case Description	ESP32-CAM opens camera, captures screenshot, and saves it	Test Priority	High
Pre - Requisite	ESP32-CAM module connected and powered, Arduino IDE with camera example sketch uploaded, web server accessible via assigned IP address	Post - Requisite	Reset ESP32-CAM after test and clear stored images if needed
Execution Steps :			

S.no	**Action**	**Inputs**	**Expected Output**	**Actual Output**	**Test Result**	**Test Comment**
01	Upload ESP32-CAM web server sketch to module via Arduino IDE	Code uploaded	Code runs successfully	Code runs successfully	Pass	Program loaded
02	Connect to ESP32-CAM IP via browser (shown in Serial Monitor)	IP address	Camera web interface opens in browser	Web interface opened correctly	Pass	Camera stream accessible
03	Click “Capture” button on web interface	User click	Camera captures and displays image	Camera captured and displayed	Pass	Image captured correctly
04	Save captured image from web interface	Save command	Image stored on local device	Image saved successfully	Pass	Screenshot saved properly

2. Website part unit testing
   - Unit testing of admin page

Table 22 Unit Testcase for admin page

S.no	Action	Inputs	Expected Output	Actual Output	Test Result	Test Comment
01	Open Admin Panel	Click "admin panel" button	Page loads successfully with login form visible	Page loads successfully	Pass	UI loads correctly with Tailwind styling
02	Login with valid credentials	Username: admin Password: 1234	Go to route selection page	Page loads successfully	Pass	Backend API responded successfully
03	Login with invalid credentials	Username: admin Password: wrongpass	Displays error message: "Invalid credentials", remains on login screen	Successfully error message shown and stays in the login screen	Pass	Error handling works as expected
04	Logout	Click Logout button after successful login	Clears session, returns to login page, resets form fields	Successfully returns to the login page	Pass	Local storage cleared and UI reset
05	Select route cities	From City: Colombo Fort To City: Kandy	Displays selected route as "Colombo Fort - Kandy".	Displayed Successfully	Pass	Route name updates dynamically
06	Add new train (valid inputs)	Train ID: T123 Name: Super Express Time: 08:30 1st: 2000 2nd: 1500	New train appears under selected route in the list	Successfully appeared	Pass	Train added and displayed correctly
07	Add new train (empty inputs)	Train ID: (empty) Name: Express Time: 08:30	Error message: "Please fill all fields for the new train and ensure prices are positive"	Error message successfully shown	Pass	Validation working as expected
08	Delete existing train	Click Delete for Train ID: CS002	Confirmation prompt appears, upon confirmation train removed from list	As expected successfully deleted	Pass	Delete function works with confirmation

- Unit Testing the Booking webpage

Table 23 Unit Testcase of the Booking Webpage

S.no	Action	Inputs	Expected Output	Actual Output	Test Result	Test Comment
01	Open booking page	Click “Booking” button	Page loads successfully and shows Step 1 (Enter Email) with disabled Next button	Page loads successfully	Pass	Page UI loads properly
02	Enter mismatched emails	Email: abc@gmail.com Verify Email: xyz@gmail.com	Displays error message: "Emails do not match." and Next button remains disabled	Successfully error message shown	Pass	Email validation working correctly
03	Enter matching emails	Email: abc@gmail.com Verify Email: abc@gmail.com	Date highlighted and Next button enabled	Successfully highlighted and next button works	Pass	Navigation works correctly after validation
04	Select a travel date	Click any date from next 7 days	Clears session, returns to login page, resets form fields	Successfully returns to the login page	Pass	Date selection works properly
05	Leave From/To empty and click Next	From: (empty) To: (empty)	Displays error: "Please select both stations"	Successfully displayed the error message	Pass	Proper validation for empty stations
06	Select same From and To stations	From: Kandy To: Kandy	Displays error: "From and To stations cannot be the same"	Successfully displayed the error message	Pass	Prevents invalid route selection
07	Select valid From & To stations	From: Colombo Fort To: Kandy	Navigates to Step 4 (Choose Train) and shows train list or message if no trains available	Successfully navigated to step 4 page	Pass	Valid route proceeds correctly
08	Select a train	Choose a train from the displayed list	Train highlighted and Next button enabled	Successfully highlighted and next button works	Pass	Train selection works correctly

09	Skip train selection and click Next	Do not select any train	Error message: "Please select a train"	Successfully error message shown	Pass	Prevents moving forward without train selection
10	Choose class and select multiple seats	Class: 1st Class and Choose seats 1, 2, 4	Total cost calculated: Rs. (price × 3) displayed correctly	Successfully calculated the total and displayed	Pass	Seat selection and cost calculation accurate
11	Click Next without selecting any seat	No seats selected	Displays error: "Please select at least one seat"	Successfully error message shown	Pass	Validation for empty seat selection works
12	Enter payment details and confirm	Card No: 4111 1111 1111 1111 Expiry: 12/25 CVV: 123 Name: abc	Moves to Step 7 (Booking Confirmed) and shows booking summary with QR code	Successfully booking confirmed	Pass	Payment and confirmation flow works fine
13	Download QR code	Click Download QR Code	Downloads QR code as an image file named train_ticket_qr_code.png	Successfully downloaded	Pass	QR code generation and download functionality tested
14	Reset booking after confirmation	Click “Book Another Seat”	Resets all fields and navigates to Step 1	Successfully navigate to the first page	Pass	Reset functionality works as intended

- Unit testing for the Tracking webpage

Table 24 Unit Testcase for the Tracking Webpage

S.no	Action	Inputs	Expected Output	Actual Output	Test Result	Test Comment
01	Open tracking webpage	Click “track my train” button	Page loads successfully with the map	Page loads successfully	Pass	UI loads correctly
02	Initialize Map	Call initTracking Map()	Map is displayed with station markers at predefined coordinates	Map loads with markers visible	Pass	Ensures map and stations render correctly
03	Update Train Location (first update)	Call updateTrain OnMap(7.29 06, 80.6337)	Train marker created at given coordinates, nearest station = “Kandy”, ETA calculated	Train marker shown, nearest = “Kandy”, ETA displayed	Pass	Verifies first marker placement works
04	Find Nearest Station	Input: (7.0, 80.5)	Function returns “Anuradhapura” as nearest station	Returns “Anuradhap ura”	Pass	Unit test for findNearestS tation()
05	Calculate ETA with speed	Distance = 40 km, Speed = 80 km/h	ETA = “30 min”	ETA displayed as “30 min”	Pass	Validates ETA calculation logic
06	Center Map (with train marker)	Call centerMap() when marker exists	Map view centers on train’s current location	Map centers correctly	Pass	Tests centerMap() marker behavior

4.2.2 Integrated testing

Table 25 Integrated TestCase of the system

S.no	Action	Inputs	Expected Output	Actual Output	Test Result	Test Comment
01	Train approaching platform and check gate automation	GPS location and Ultrasonic sensor detection	Platform gates remain closed until train fully stops, then gates open	Platform gates stayed closed until train stopped and opened correctly	Pass	Verified GPS, sensor and servo motor integration
02	Train approaching railway crossing	Ultrasonic sensor detects train for 5 seconds	Railway crossing gates automatically close, vehicles stopped	Gates closed automatically, vehicles stopped	Pass	Sensor and servo motor synchronized properly
03	Train pass railway crossing	Ultrasonic sensor detects train for 5 seconds	Railway crossing gates automatically open, vehicles start moving	Railway crossing gates automatically open, vehicles moves	Pass	Sensor and servo motor synchronized properly
04	Passenger books a ticket online	Passenger books a ticket online	QR code generated, processed in flask and email sent to passenger through SMTP server	QR code generated and email received	Pass	Booking, QR, and email modules integrated correctly
05	Passenger scans valid QR code at train door	Valid QR code scanned by ESP32-CAM	Door servo unlocks, passenger allowed boarding	Door unlocked, passenger boarded successfully	Pass	QR scanning, and servo motor integration successful
06	Passenger scans invalid QR code	Fake/expired QR code	Door remains locked, error displayed, security notified	Door remained locked, error displayed, manual check allowed	Pass	Negative case handled properly
07	Train in motion and check GPS tracking	GPS coordinates from NEO-6M, ESP32, Firebase	Website shows train location in real time (updated every 2 to 5 seconds)	Train location updated every 2 to 5 seconds on website	Pass	GPS, ESP32, Firebase, and frontend synced correctly
08	Train ETA calculation	Continuous GPS feed and OpenStreet Maps API	ETA dynamically displayed on passenger web interface	ETA displayed correctly and updated with movement	Pass	ETA calculation and web display accurate

09	Poor GPS signal area	Weak GPS data from NEO-6M	Train location updates delayed, ETA adjusted accordingly	Updates delayed but system adjusted ETA	Pass	GPS fallback handled properly
10	Railway crossing blocked by vehicle	Ultrasonic under-gate or middle the train detection triggered	Gates remain open, warning system activated	Gates stayed open, warning alert activated	Pass	Safety override worked properly
11	Network failure during GPS update	GPS, ESP32 and Firebase data flow interrupted	Location not updated	Train location froze	Pass	Negative case handled
12	Sensor malfunction at crossing and platform gate	Ultrasonic sensor not detecting	Manual override should allow gatekeeper to close gates	System alerted admin, manual override used successfully	Pass	Negative case recovery verified
13	Email delivery integration	Booking completed and system generates QR	QR embedded in PDF and emailed via SMTP	Email received with QR PDF	Pass	Verifies booking, QR and email server integration
14	Firebase data sync	ESP32 sends train location every 2 seconds	Firebase updates instantly, frontend displays	Firebase updated and frontend refreshed	Pass	Confirms IoT, cloud and frontend pipeline
15	Booking and seat availability check	Multiple users attempt to book same seat	When booked it should be read colored and cannot select again.	Booking succeeded, others cannot select the booked seats.	Pass	Validates booking system concurrency
16	QR re-scan prevention	Passenger reuses QR after entry	System rejects reused QR	QR reuse rejected, no duplicate boarding	Pass	Security requirement validated
17	Power failure simulation	ESP32 loses power during train detection	Backup/manual mode enabled	System switched to manual	Pass	Safety-critical failover tested

4.2.3 System testing

Table 26 System Testcase of the System

S.no	Action	Inputs	Expected Output	Actual Output	Test Result	Test Comment
01	Full train arrival and boarding workflow	Train approaches detected by ultrasonic sensor and get GPS updates, platform gates open and QR scanned	Gates closed until stop, GPS updated, ETA shown, QR boarding successful	Gates closed until stop, GPS updated every 2s to 5s, ETA shown, QR scanned and door opened	Pass	End-to-end train arrival and boarding validated
02	Train crosses railway crossing	Ultrasonic detects train, system closes gates	Gates close before train arrives and reopen after train passes	Gates closed automatically and reopened after train cleared	Pass	Confirms crossing safety system
03	Passenger completes booking and boarding	Passenger selects train, pays, receives QR, scans QR	Booking successful, QR emailed, door unlocks	Booking confirmed, QR email received, door unlocked	Pass	Full booking and boarding workflow verified
04	Real-time tracking under weak GPS	Weak GPS signal, intermittent data	Location updates delayed, ETA adjusted accordingly	Location lagged, ETA adjusted	Pass	Fallback handled properly
05	Invalid booking attempt	Wrong date or class input	Error message shown, booking not created	Error displayed, no booking created	Pass	Input validation successful
06	QR code reuse attempt	Passenger scans QR more than once	Second attempt rejected	Second scan rejected	Pass	Boarding security confirmed
07	Vehicle blocking crossing	Ultrasonic detects vehicle under gate or in the middle of the train track	Crossing gates remain open, alert activated	Gates stayed open, alert triggered	Pass	Safety override confirmed
08	Sensor failure simulation	Ultrasonic disabled	System alerts admin, manual override enabled	Alert shown, manual gate control worked	Pass	Fail-safe mode validated

4.2.4 Acceptance testing

Table 27 Acceptance Testcase of the system

S.no	Action	Inputs	Expected Output	Actual Output	Test Result	Test Comment
01	Passenger views train location	Passenger opens tracking webpage	Train location and ETA displayed on screen	Train location and ETA displayed correctly	Pass	Requirement of real-time tracking satisfied
02	Passenger books ticket online	Passenger selects route, enters payment details, confirms booking	Confirmation email sent with QR code	Email received with valid QR code	Pass	Online booking and QR generation validated
03	Passenger scans QR at train door	Passenger presents valid QR code at scanner	Door unlocks, passenger allowed boarding	Door unlocked, boarding completed	Pass	Secure boarding requirement achieved
04	Passenger scans invalid QR	Passenger presents expired or fake QR code	Door remains locked and red LED turns on	Door remained locked, red LED turned on	Pass	Boarding security requirement validated
05	Train approaches railway crossing	Train detected by ultrasonic sensors	Gates close before arrival and reopen after passing	Gates closed automatically and reopened correctly	Pass	Safety requirement at crossings satisfied
06	Train stops at platform	Train arrives at station, fully stopped	Platform gates open only when safe	Platform gates opened only after train stopped	Pass	Accident prevention requirement achieved
07	Admin updates train schedule	Admin adds or remove train schedule	Updated schedule available on passenger booking site	Schedule updated and shown correctly	Pass	Operational requirement satisfied
08	Passenger attempts invalid booking	Passenger enters wrong details or invalid payment	Booking rejected, error message shown	Error displayed, booking not created	Pass	Input validation requirement satisfied

4.3 Test results and conclusion of testing

The testing phase of the TrainSync system demonstrated that all core functionalities performed according to the specified requirements.

Unit Testing confirmed that individual components such as ultrasonic sensors, servo motors, GPS modules, and the ESP32-CAM operated correctly and within expected accuracy levels. Both positive and negative cases were validated successfully.

Integration Testing ensured that hardware components, backend services, and frontend interfaces worked seamlessly together. Functions such as real-time GPS tracking, ETA updates, automated gate operations, booking, and QR-based boarding were integrated without critical failures.

System Testing validated end-to-end workflows. The complete processes of train detection, platform gate operation, railway crossing safety, online booking, QR ticket generation, and passenger boarding were executed successfully. Safety overrides (e.g., blocked crossings, invalid QR codes) were handled as expected.

Acceptance Testing confirmed that the system met both functional and non-functional requirements. Reliability, usability, and safety objectives were achieved, while negative scenarios (sensor failures, weak GPS, reused QR codes, network delays) were managed with appropriate fallbacks or manual overrides.

The overall test results indicate that TrainSync is a reliable and effective solution for enhancing railway passenger safety, operational efficiency, and user convenience. The system achieved the intended objectives, with all critical requirements validated through rigorous testing.

5 Conclusion

5.1 Conclusions of the project

The Automated Train Platform Safety and Tracking System (TrainSync) represents a significant step toward the modernization of railway transportation in Sri Lanka. The project's implementation demonstrated how low-cost IoT hardware and open-source software can be combined to solve long-standing safety issues such as unauthorized boarding, train–vehicle collisions at crossings, and injuries caused by unsafe passenger behavior at platforms.

Through the integration of automated platform gates, intelligent railway crossing barriers, GPS-based tracking with ETA calculation, and QR code–based authentication for reserved compartments, the project successfully addressed critical challenges in the sector. This resulted not only in improved safety standards but also in enhanced convenience for passengers, who can now access real-time train information and board trains more securely.

Moreover, the project validated the use of a phased deployment strategy, where each subsystem such as the gates, GPS tracking, and QR authentication was first developed and tested independently before being integrated into the full solution. This systematic approach minimized risks and ensured that the prototype could realistically scale into real-world implementation with gradual infrastructure upgrades.

Although certain constraints such as network dependency and hardware accuracy were noted, the prototype conclusively proved that a smart railway ecosystem is feasible, practical, and affordable in the Sri Lankan context.

5.2 Lessons learned and skills earned

From a technical perspective, the project allowed us to explore the intersection of hardware, software, and real-time communication. We gained proficiency in programming ESP32 microcontrollers for sensor-based automation, working with the ESP32-CAM for QR code scanning, and integrating GPS modules to transmit live data to Firebase. Additionally, we strengthened our skills in backend development using Flask, handling database operations with Firebase Realtime Database, and building user-friendly interfaces with HTML, CSS, and JavaScript.

During the testing phase of the TrainSync system, our team learned the importance of systematic testing across unit, integration, system, and acceptance levels to ensure the reliability of both hardware and software. By addressing negative scenarios such as weak GPS signals, invalid or reused QR codes, and sensor malfunctions, our team recognized the need for effective fail-safes and manual overrides. Prototyping and iterative refinement helped us calibrate sensors, adjust servo motors, and improve real-time performance. We also gained hands-on experience with essential tools: Arduino IDE and Serial Monitor for IoT debugging, Python IDLE and Flask for backend validation, Firebase Console for monitoring live GPS data, and browser developer tools for testing booking and tracking interfaces. Overall, testing taught our team how to combine hardware and software validation, leverage multiple tools effectively, and work collaboratively to build a robust, safe, and user-friendly system.

Beyond the technical achievements, this project offered important lessons in team collaboration, time management, and problem-solving. Working under limited resources forced us to think creatively and prioritize features that would deliver maximum impact. We also learned the importance of user-centered design, since the system ultimately serves passengers, administrators, and operators who have varying levels of technical literacy.

On a personal development level, each team member gained improved research, documentation, and communication skills, which are essential for professional growth. Collectively, the project has prepared us to face real-world challenges in system design, IoT applications, and technology-driven solutions for public infrastructure.

5.3 Recommendations for further improvements

Although the TrainSync system has shown significant promise, there are several areas that could be refined and expanded to increase its effectiveness and sustainability:

Hardware and Sensor Optimization

- Deploy industrial-grade GPS and ultrasonic modules for improved accuracy and long-term durability.
- Introduce redundant sensors to ensure fail-safe operations during sensor failure or environmental interference.

Enhanced Network Independence

- Implement LoRaWAN or 5G technologies to overcome mobile network dependency, especially in rural areas where Dialog or other providers may have limited coverage.
- Provide offline data caching to ensure continuity of operations even during temporary connectivity issues.

Scalability of Boarding Control

- Extend the QR-based boarding system beyond first- and second-class compartments to include all passenger coaches.
- Introduce facial recognition or biometric verification as an additional layer of security in future upgrades.

User Experience and Accessibility Enhancements

- Develop a dedicated mobile application with real-time notifications, seat reservation options, and digital wallet integration for payments.
- Offer multi-language support (Sinhala, Tamil, English) to improve accessibility for a wider user base.

Integration of Artificial Intelligence and Data Analytics

- Use machine learning models to predict delays, optimize ETA calculations, and manage peak-hour passenger flow.
- Employ predictive maintenance analytics to reduce downtime by monitoring hardware health and performance trends.

Environmental Sustainability

- Introduce solar-powered systems for gates and sensors to reduce reliance on grid electricity.
- Ensure that all components are weather-resistant and eco-friendly, reducing long-term environmental impact.

6 References

[1] S. Lakshitha, "Sri Lanka - 2.4 Railway Assessment," 10 september 2024. [Online]. Available: https://lca.logcluster.org/sri-lanka-24-railway-assessment. [Accessed 04 september 2025].

[2] A. Peet, "Waterfall Model - Software Engineering," geeksforgeeks, 11 july 2025. [Online]. Available: https://www.geeksforgeeks.org/software-engineering/waterfall-model/. [Accessed 02 September 2025].

[3] R. Teja, "Getting Started with ESP32 | Introduction to ESP32," ElectronicsHub USA , 01 April 2024. [Online]. Available: https://www.electronicshub.org/getting-started-with-esp32/. [Accessed 01 September 2025].

[4] S. Hamdi, "A Deep Dive into the ESP32 CAM Module: Unleashing Potential," CodeCraft, 22 November 2023. [Online]. Available: https://full-skills.com/iot/esp32-cam/. [Accessed 02 September 2025].

[5] "Introduction to OpenStreetMap API," PublicAPIs, 10 January 2019. [Online]. Available: https://publicapis.io/open-street-map-api. [Accessed 04 September 2025].

7 Recommendation of supervisor(s) on the final Report

(This section should be filled by the supervisor(s)).

Comments (if any):

I/We certify that, the student engaged continuously with me in developing the proposal and, I am confident that they are adequately competent to defend this viva.

Signature(s) of Supervisor(s):

Date:

8 Viva presentation assessment team

(This section should be filled by the department)

Date of viva presentation: ______________________________

Panel members	**Name**	**Department / Institute**
Chair		
Member		
Member		
Member		
Member		

9 Comments of the assessment team on viva presentation

(This should be filled by the chair of the assessment panel. In case of revision or fail, needed revision or reasons to fail the viva presentation should be mentioned here)

Result of the viva presentation	**Excellent / Good / Pass with revisions / Fail**
Score	
Signature of the panel chair	
Date	