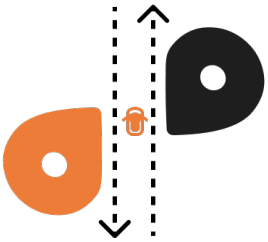
AN-NAJAH NATIONAL UNIVERSITY

FACULTY OF ENGINEERING AND INFORMATION TECHNOLOGY

COMPUTER ENGINEERING DEPARTMENT

Software Graduation Project:
Wassilni



Submitted by

Adam Abdalhafez

Amro Sous

Supervisor

Dr.Amjad AbuHassan

Presented in partial fulfilment of the requirements for Bachelor degree in Computer Engineering.

Jan 26, 2025

Abstract

The Wassilni project aims to enhance public transportation by providing a more efficient and user-friendly booking system. With increasing urban congestion and demand for reliable transport, Wassilni simplifies the process of booking public buses, reducing waiting times and offering real-time tracking. The system integrates modern technologies, including GPS, AI, and mobile applications, to improve the commuting experience.

Key features of the project include a React Native mobile application for riders to register, select destinations, and book buses based on proximity and route availability. The driver's route and pick-up points are dynamically generated, allowing real-time tracking of the vehicle. The system also incorporates checkpoints, which serve as predefined stops along the route. If a checkpoint is closed, the system suggests alternative routes to ensure seamless travel. Additionally, an AI-powered service updates checkpoint statuses based on driver communication, improving route planning and reducing delays.

To further enhance the user experience, the system includes a waiting queue feature that notifies riders when no buses are currently available, helping them manage expectations. A driver rating system allows passengers to provide feedback, promoting service quality and accountability. The admin web dashboard enables administrators to manage drivers, routes, and monitor performance.

The back-end is deployed on AWS ECS with MongoDB for efficient data handling, particularly for geospatial indexing to locate nearby vehicles and checkpoints. The system leverages the Google Maps API for routing and estimated arrival times. Unlike private ride-hailing services such as Uber or Careem, Wassilni focuses on optimizing public transportation, making it an innovative solution for improving daily commutes.

Dedication

This work is dedicated to our family, whose unwavering support and encouragement have been a constant source of inspiration throughout this journey. To our parents, for their sacrifices and belief in our dreams, and to our friends, who have always been there to cheer us on. Finally, to our mentors and professors, whose guidance and knowledge have been invaluable.

Acknowledgment

We would like to express our sincere gratitude to our supervisor, Dr. Amjad Abu Hassan, for his guidance and support throughout this project. We also appreciate An-Najah National University for providing us with the resources and environment to develop our work.

Additionally, we extend our thanks to everyone who contributed, whether through ideas, feedback, or solutions—big or small. Your support has been invaluable in helping us complete this project.

Disclaimer

The report has been written by students from An-Najah National University's Faculty of Engineering's Computer Engineering Department. There may be grammatical and content errors in the content, as it has not been edited or corrected beyond editorial changes made after the assessment process. The opinions expressed are entirely those of the students, as are any conclusions and suggestions. An-Najah National University disclaims all liability and responsibility for any consequences that may arise from using this report for purposes other than those for which it was originally commissioned.

Contents

List of Figures

List of Tables

1 Introduction

Transportation is a vital element of modern life, facilitating the movement of people, goods, and services across regions and countries. It enables access to essential resources, supports economic activities, and fosters social connections. Efficient transportation systems are crucial for improving productivity, reducing travel time, and enhancing the overall quality of life. Whether by road, rail, air, or sea, transportation serves as the backbone of economies and plays a central role in the functioning of societies.

In this context, the optimization of public transportation systems has become increasingly important. By improving the efficiency and reliability of services, public transportation can contribute significantly to sustainable urban development and enhance the daily commute for millions of people. This application seeks to address key challenges in public transportation, providing solutions that benefit both drivers and passengers alike.

1.1 Problem

Public transportation systems face numerous challenges that affect both drivers and passengers. Drivers encounter difficulties in managing queues and determining the proper turn order, leading to disputes when one driver arrives earlier than another. They also experience delays due to no-show passengers, and they face challenges with handling high-denomination currency and providing exact change. Drivers also lack real-time updates on the status of barriers on the roads and are often unaware of alternative routes to avoid closed barriers. Additionally, there are issues with passenger identification, especially in group bookings when multiple passengers are picked up from the same location.

Passengers face their own set of challenges, including long waiting times when no cars are available, and the inconvenience of having to travel to garages even when cars could pass nearby. Payment issues, similar to those faced by drivers, also arise due to high-denomination currency and the inability to provide change. Passengers also lack real-time updates on road conditions, including barriers and alternative routes to their destinations. Furthermore, passengers are often unaware of the current location of the car and whether it has left the garage, leading to uncertainty and frustration. Lastly, passengers struggle to select the most appropriate garage to reach their destination efficiently.

1.2 Objectives

The objectives of this application are designed to address the challenges faced by both drivers and passengers. For drivers, the system will develop a transparent queue management system to ensure fair turn allocation. It will also provide real-time updates on passenger no-shows to minimize delays and integrate a digital payment system to resolve issues related to high-denomination currency. Additionally, drivers will receive real-time updates about the status of road barriers and suggestions for alternative routes when barriers affect their planned paths. QR code scanning will also be implemented to facilitate easy passenger identification, especially during group bookings.

For passengers, the system will reduce waiting times by notifying them of car availability in real time. It will allow passengers to book a ride from their current location instead of requiring them to travel to garages unnecessarily. The system will also resolve payment issues by offering secure digital payment options. Real-time updates on road and barrier conditions will be provided to passengers, and alternative routes will be displayed for better travel planning. The system will enable passengers to track the real-time location of the car and its journey progress, offering more certainty. Lastly, passengers will be guided in selecting the most suitable garage for their destination, optimizing their travel experience.

1.3 Scope

This application is designed to cater to three primary user groups: drivers, passengers, and administrators.

For drivers, the application will allow them to view their assigned garage, enter the queue, monitor seat reservations, photograph passenger QR codes for quick identification, and track their routes. Drivers will also have the ability to communicate with fellow drivers on the same route, as well as receive real-time updates on barriers and alternative routes to ensure smooth travel.

For passengers, the application will enable them to book seats in cars from their nearest garage or any point along the route. They will have access to real-time trip updates, allowing them to track the car's location and receive notifications about delays. Passengers will be able to communicate with their driver and fellow passengers, facilitating a smoother travel experience. Additionally, passengers will have the option to apply for driver positions through job postings within the app.

Administrators will be provided with robust tools for managing and monitoring the entire system. They will be able to access real-time data, manage driver assignments, track passenger bookings, and make informed decisions to optimize the public transportation network.

1.4 Importance

This application is critical for addressing inefficiencies in public transportation systems by leveraging advanced technology to create a smarter, more connected transportation network. By enhancing the user experience for both passengers and drivers, the system improves operational efficiency and ensures that users have access to the most up-to-date information in real time. This includes providing drivers with alternative routes when barriers are detected and reducing delays caused by no-show passengers.

The application also facilitates secure and transparent transactions by integrating a digital payment system, resolving issues related to high-denomination currency. For administrators, the system offers robust management and monitoring tools that empower decision-making and help optimize resources.

Overall, by solving these challenges, the application contributes to a more efficient and user-friendly public transportation network. It improves mobility, reduces delays, saves time for users, and enhances the overall experience for both drivers and passengers.

1.5 Report Organization

This report is organized into several sections to provide a comprehensive overview of the Wassilni project. The structure is as follows:

- **Introduction:** Provides an overview of the project, including its objectives, scope, and importance.
- **Constraints and Earlier Coursework:** Discusses the challenges faced during the project and the lessons learned from earlier coursework.
- **Methodology:** Describes the tools, methods, and programming languages used in the project, along with the implementation of system features.
- **System Features Implementation:** Details the development of the mobile application, web application, and backend system.
- **Conclusion:** Summarizes the project's achievements and contributions.
- **Future Work:** Outlines potential improvements and extensions for the project.
- **References:** Lists the sources cited in the report.

2 Constraints and Earlier Coursework

2.1 Design

One of the most significant obstacles we encountered during this project was the **lack of experience in design talents**. As we progressed, we realized that our team had limited expertise in structuring, visualizing, and refining the design elements necessary for an optimal user experience. Without sufficient prior experience, we faced difficulties in making intuitive design decisions, selecting appropriate color schemes, structuring layouts effectively, and ensuring that the user interface aligned with modern standards. Additionally, the absence of specialized design knowledge meant that we had to spend considerable time researching best practices. As a result, the design phase required extra effort, collaboration, and learning, making it one of the most time-consuming and demanding aspects of the project.

2.2 Difficulty in Information Gathering

Understanding how the public transportation system operates posed a significant challenge. Gathering accurate information about bus routes, schedules, and garage protocols required extensive research and communication with various stakeholders. Identifying how buses are queued, how drivers receive trip assignments, and the factors influencing route planning involved consulting with transportation officials, garage managers, and drivers. Additionally, the lack of centralized documentation made it difficult to standardize processes, requiring us to rely on observations and direct inquiries to ensure our system aligns with real-world operations.

2.3 Routes Map

The biggest challenge we faced in our project was managing transportation routes due to the complex and restrictive nature of our region. Being in an occupied country, many roads are either inaccessible or forbidden, making it difficult to design efficient routes. Unlike other regions where planning relies on traffic conditions and distance, we had to consider political and security constraints, which impacted our mobility. This forced us to find alternative paths that were both legally accessible and logistically feasible. Many mapping services didn't reflect these restrictions, so we couldn't rely on standard digital maps to generate optimal routes. Instead, we manually verified routes, cross-checked multiple sources, consulted local drivers, and tested paths for accuracy. Customizing the maps was time-consuming, as each modification required careful validation to ensure the roads were correct and suitable for the community's transportation needs. This experience highlighted the challenges of managing routes in restricted environments and emphasized the importance of flexible, context-aware mapping systems.

3 Methodology

3.1 Tools, Methods and Programming Languages

3.1.1 Tools

Figure 4.1: Android Studio

Android Studio is the official IDE for Android app development, offering a comprehensive environment for building and testing Android applications.

Figure 4.2: AWS

Amazon Web Services (AWS) is a cloud platform offering scalable computing power, storage, and development tools for building and managing applications.

Figure 4.3: Docker

Docker is a platform for developing, shipping, and running applications in containers, ensuring consistency across environments.

Figure 4.4: Expo

Expo is a framework for building React Native applications, providing tools and services to streamline development and testing processes.

Figure 4.5: Gemini

Gemini is a collaborative design and prototyping tool that facilitates teamwork and enhances design workflows.

Figure 4.6: Git

Git is a version control system for tracking changes in source code, enabling collaborative development and efficient code management.

Figure 4.7: Leaflet

Leaflet is an open-source JavaScript library for interactive maps, supporting mobile-friendly design and customizable mapping features.

Figure 4.8: MongoDB

MongoDB is a NoSQL database management system, ideal for storing and retrieving unstructured data with high performance and scalability.

Figure 4.9: Postman

Postman is a collaboration platform for API development, allowing users to test, document, and monitor APIs efficiently.

Figure 4.10: Stream Chat

Stream Chat is a scalable API for integrating chat functionality into applications with features like real-time messaging and user presence.

Figure 4.11: Stripe

Stripe is a payment processing platform for managing online transactions, offering secure and customizable payment solutions.

Figure 4.12: VS Code

Visual Studio Code (VS Code) is a lightweight, extensible code editor supporting multiple programming languages and a rich plugin ecosystem.

3.1.2 Framework

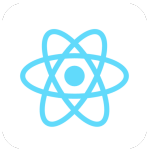


Figure 4.13: ReactJS

ReactJS is a JavaScript library for building user interfaces, particularly single-page applications where efficiency and responsiveness are key.

Figure 4.14: React Native

React Native is a framework for building native mobile applications using JavaScript and React, allowing for cross-platform development with shared code.

Figure 4.15: Express.js

Express.js is a minimal and flexible Node.js web application framework, used to build APIs and web applications efficiently.

Figure 4.16: Python

Python is a versatile, high-level programming language widely used for web development, automation, data science, and artificial intelligence.

3.2 System Features Implementation

3.2.1 Mobile application

3.2.1.1 Welcome & Onboarding

Welcome Screen

Figure 4.17: welcome Page

Guidance slides

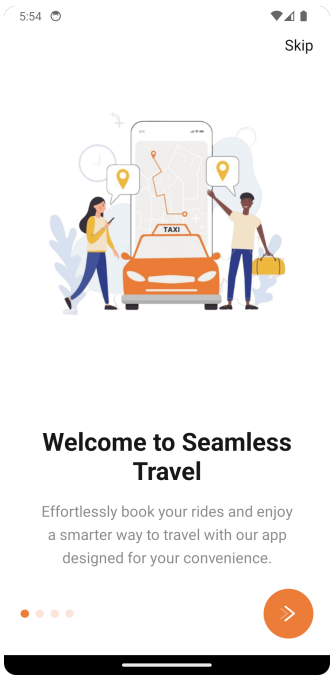


Figure 4.18: Welcome to Seamless Travel

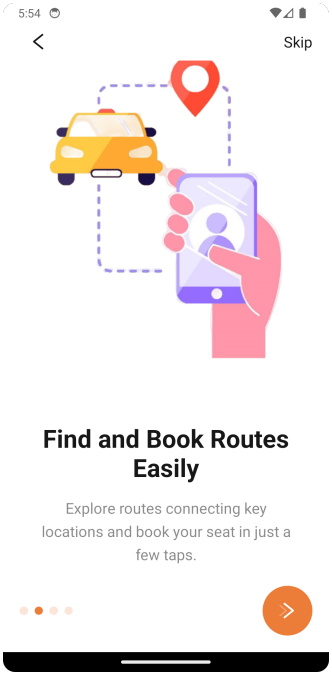


Figure 4.19: Find and Book Routes Easily

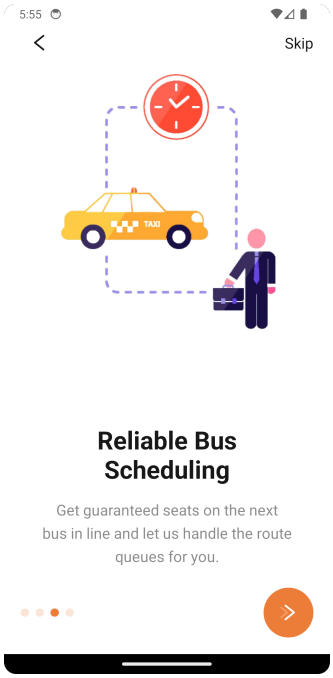


Figure 4.20: Reliable Bus Scheduling

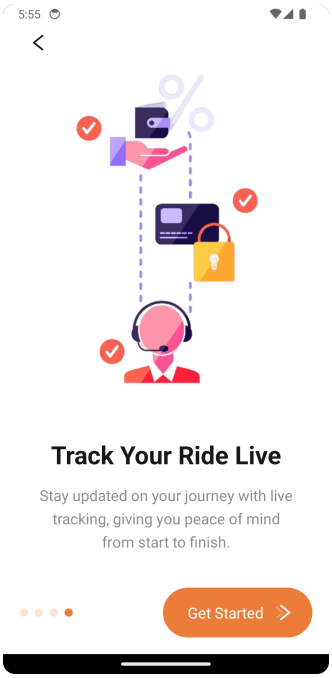


Figure 4.21: Track Your Ride Live

Choose Role

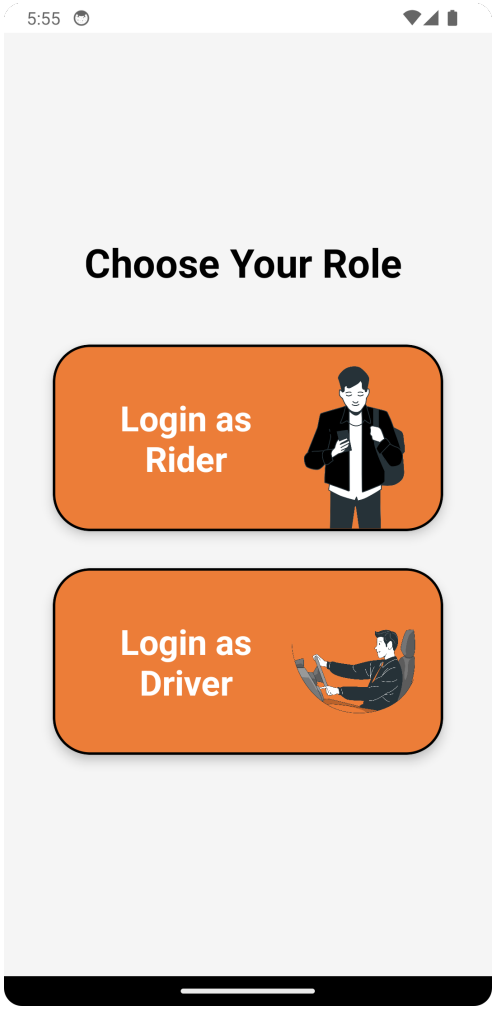


Figure 4.22: Choose Role

3.2.1.2 Rider Side

Overview of the Booking Process

The booking process workflow illustrates the sequential steps involved in a transportation system, ensuring a seamless experience for users. The process begins with the rider accessing the reservation page, where they can specify their destination on a map. The system then displays available routes and garages for the selected route, allowing the rider to choose a starting point. Once selected, the rider can reserve a car; if no car is available, the system provides alternatives such as queuing for a future car or booking in advance. Upon successful booking, the rider proceeds to payment, securely handled via Stripe. Additional features include chat functionality during the trip and the ability for drivers to submit job applications. Riders also have the option to cancel reservations, with refund policies varying depending on whether the trip has started. This comprehensive workflow highlights the integration of reservation, payment, queue management, and communication features, providing a smooth and efficient booking experience for both riders and drivers.

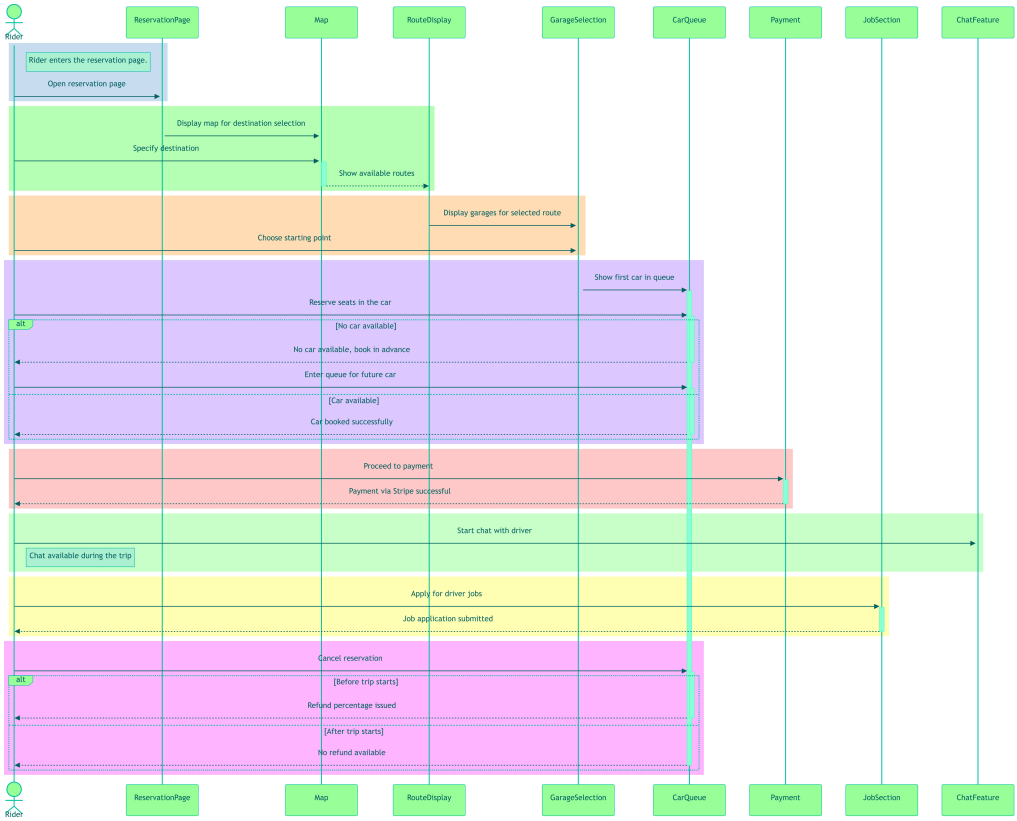


Figure 4.23: Booking Rider Process

Registration System

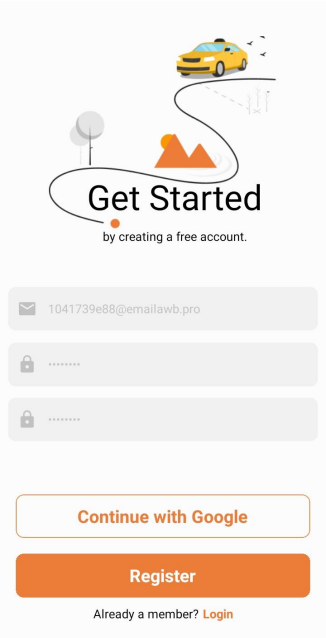


Figure 4.24: User Registration Page

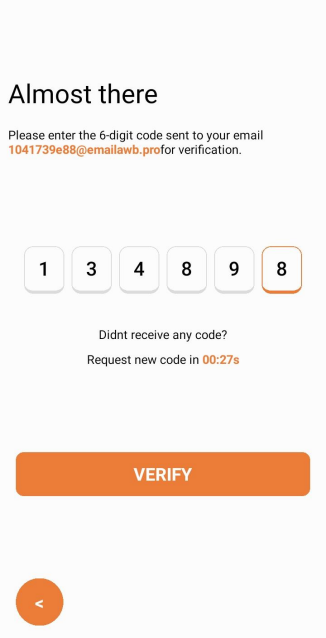


Figure 4.25: Account Verification

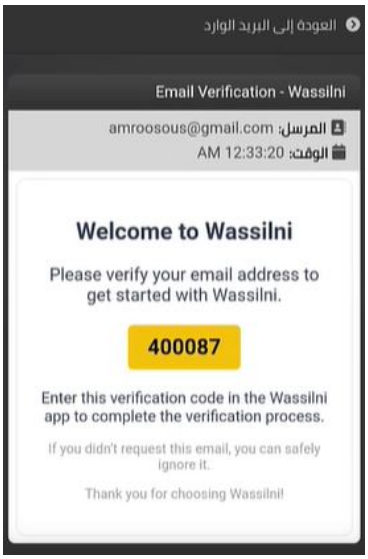


Figure 4.26: Email Verification Code

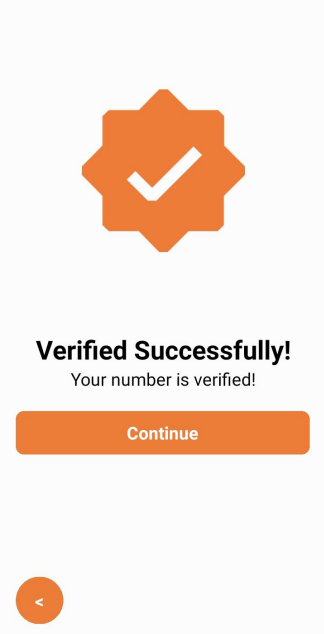


Figure 4.27: Successful Verification

The basic information of the user can be entered on the registration screen Figure 4.24 to start creating a new account in the Wassilni application. After pressing the Register button, the application saves the information in the Wassilni database, generates a unique authentication ID that is used to identify the user, and stores the necessary data in the database.

In the next step, you can verify the account by receiving a verification code sent to your email 4.26. The user enters the code on the verification page4.25 to complete the

verification process. Upon successful verification, a confirmation message is displayed, as shown in 4.27,and then you can Login with your account.

Login

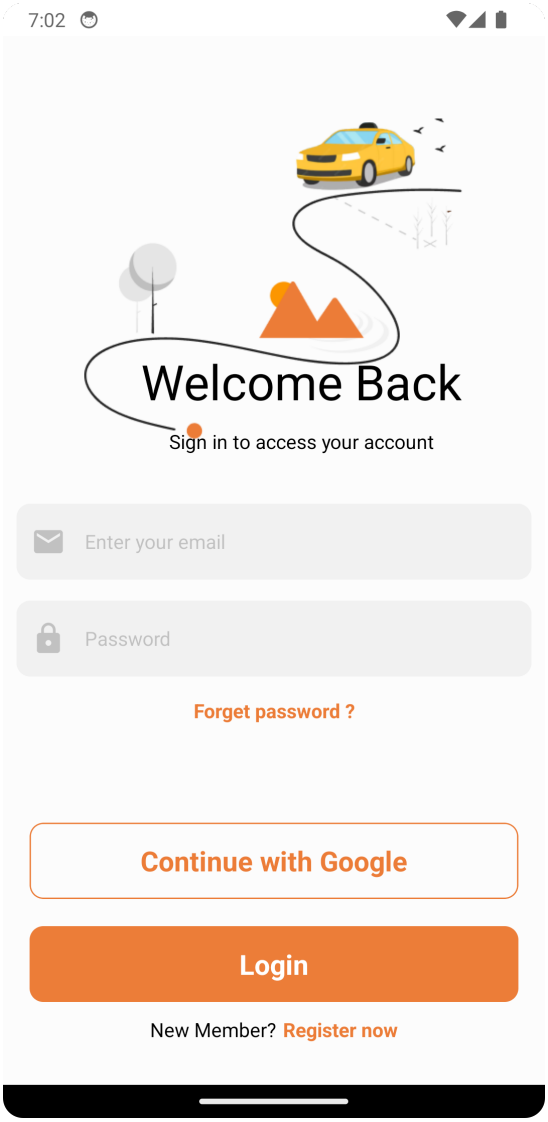


Figure 4.28: Login Page

The Login page Figure 4.56 allows users to enter their credentials to access their accounts in the Wassilni application.

Home Screen

The home page features a beautiful design with various functional elements that enhance the user experience.

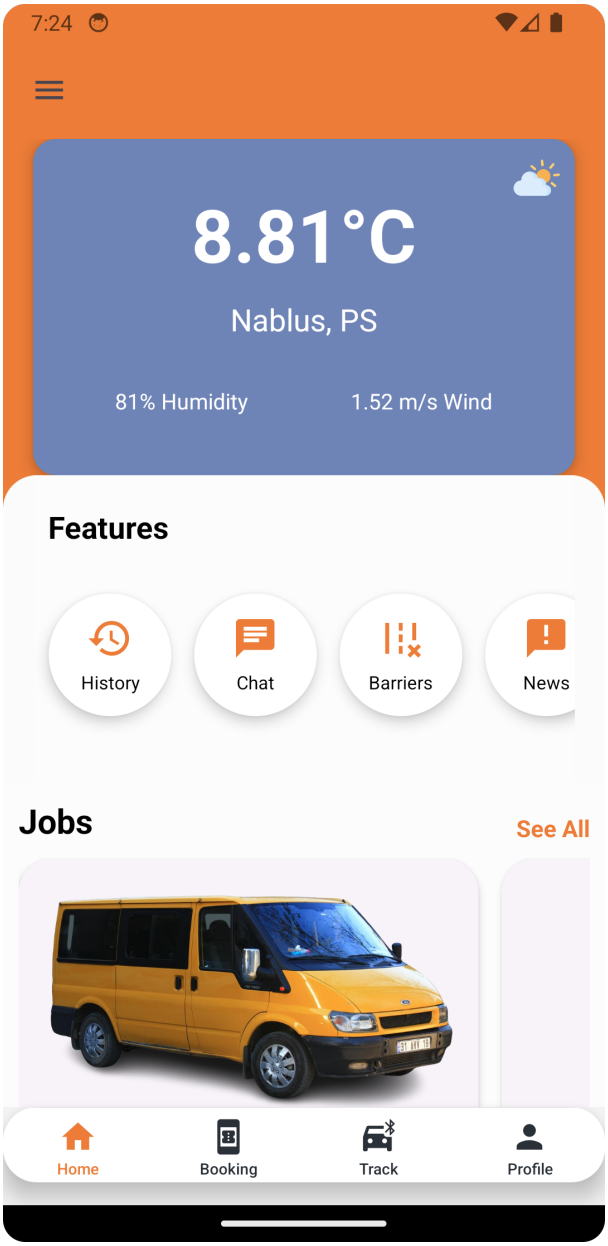


Figure 4.29: Home Page Design

This layout includes weather conditions for your area and a set of features that direct you to specific pages or actions.

Booking History

The booking history section allows you to view your past bookings. It provides an option to cancel any booked flights that have not yet been received or have not ended, ensuring a seamless user experience for managing travel plans.

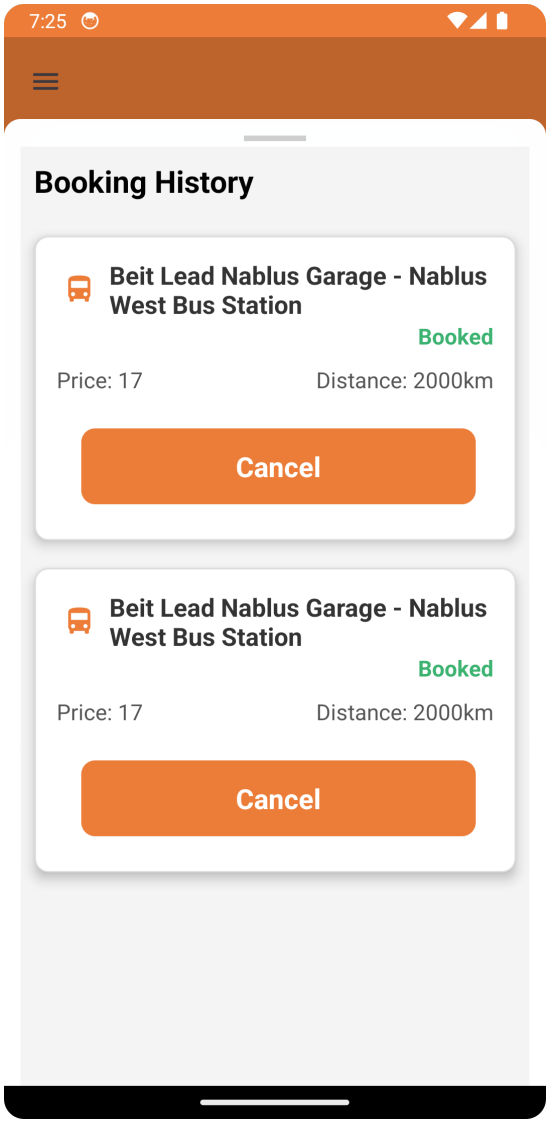


Figure 4.30: Booking History

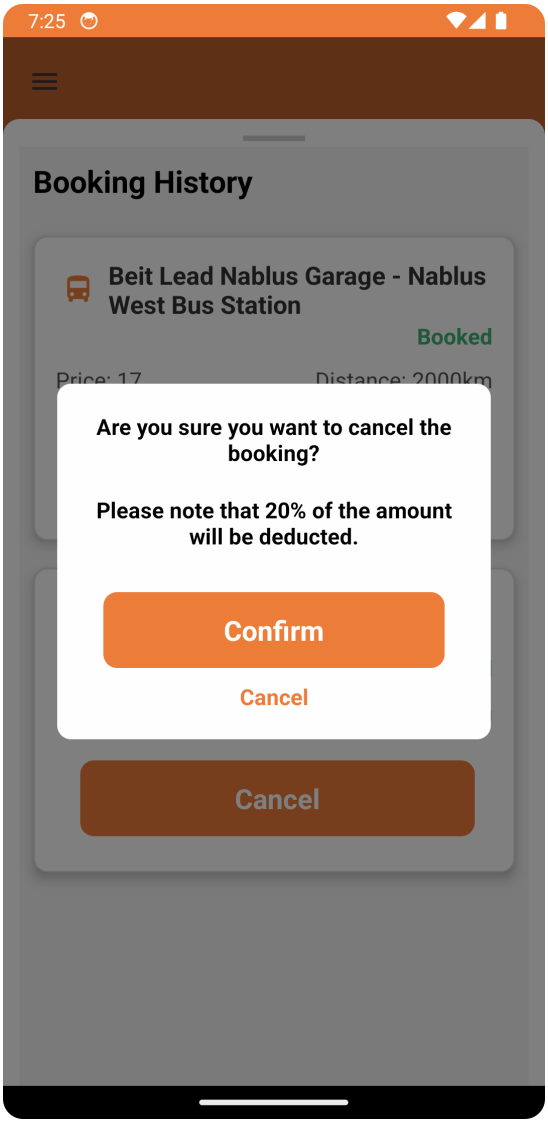


Figure 4.31: Booking Cancel

You can navigate through this feature to manage and cancel bookings as necessary.

General News

This section displays the latest updates and news relevant to your journey or area. You can stay up-to-date with current events and important information affecting your travel plans.

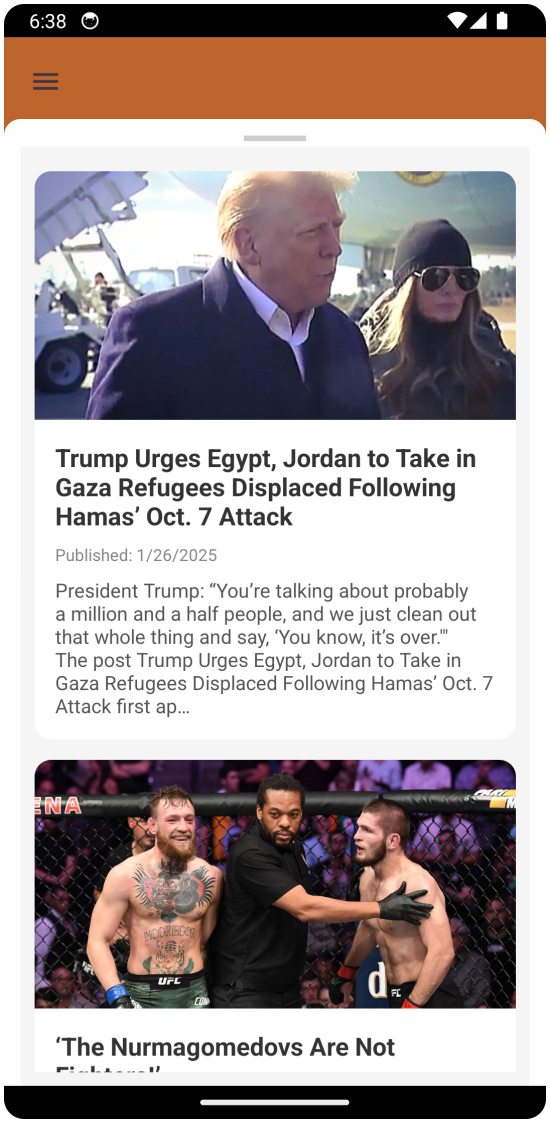


Figure 4.32: General News

Stay informed with the latest happenings that may impact your route or travel choices.

Chat Feature

The conversation feature lets you engage in chats to get updates or information about your route, barriers, or general inquiries. This helps you interact with the system in a dynamic and convenient manner.

Drawer Menu

The drawer menu offers additional functionalities. You can find:

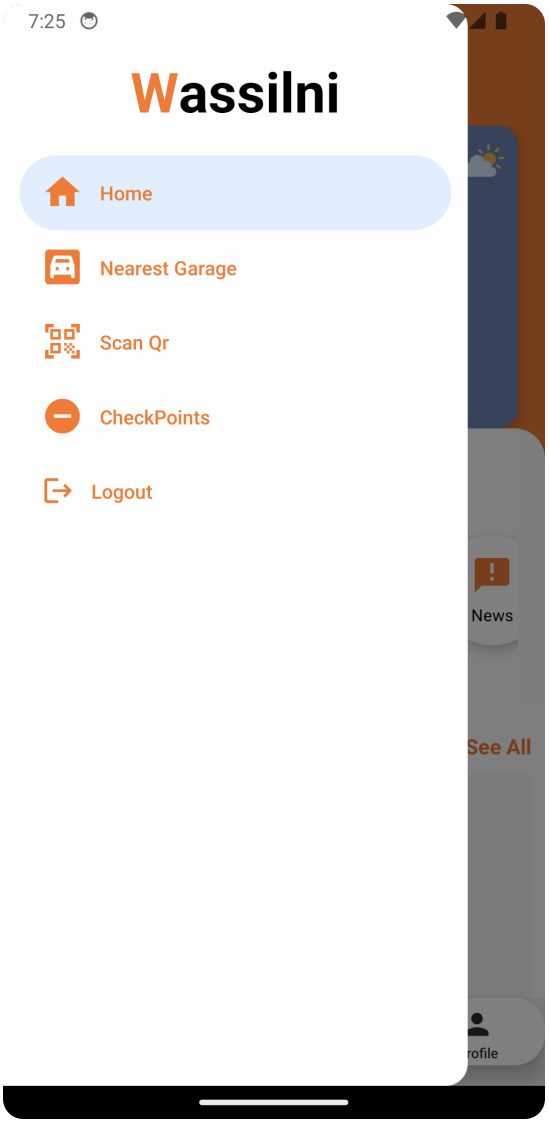


Figure 4.33: Drawer Menu

- A map showing garages near your current location as shown in 4.34.
- A list of garages near your location as shown in 4.35.
- The status of all barriers as shown in 4.33.
- An option to log out of your account.

Nearest Garage

The Nearest Garage section helps you locate the closest garages to your current location. This feature displays a map view for easy navigation and a list view to show available garages with additional details like distance and contact information.

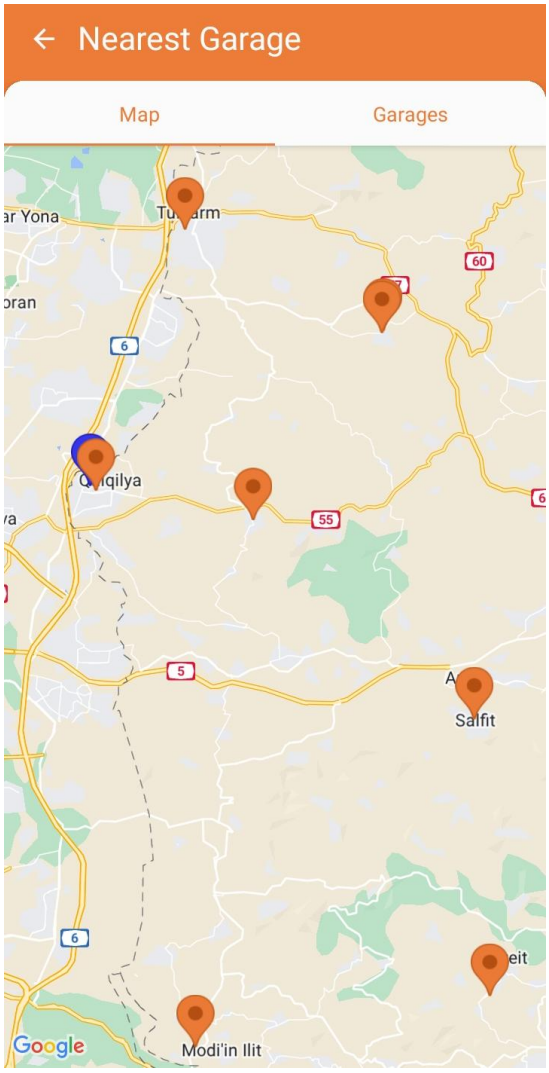


Figure 4.34: Garages Near Location Map

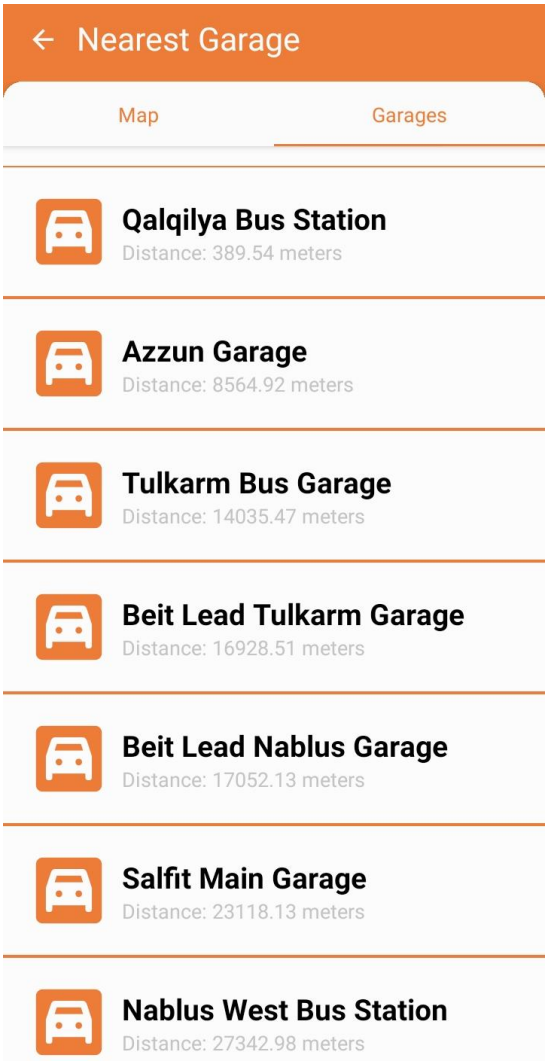


Figure 4.35: Garages Near Location List

The map view provides a clear visual representation of nearby garages, making it easy for users to select the most convenient option based on proximity. The list view offers a more detailed list of garages, showing useful information such as the name, address, and contact number, which enhances the user's ability to make informed decisions when selecting a garage.

Status of Checkpoints

The status of Checkpoints allows you to check real-time information about the nearby obstacles or blockages that may affect your travel route.

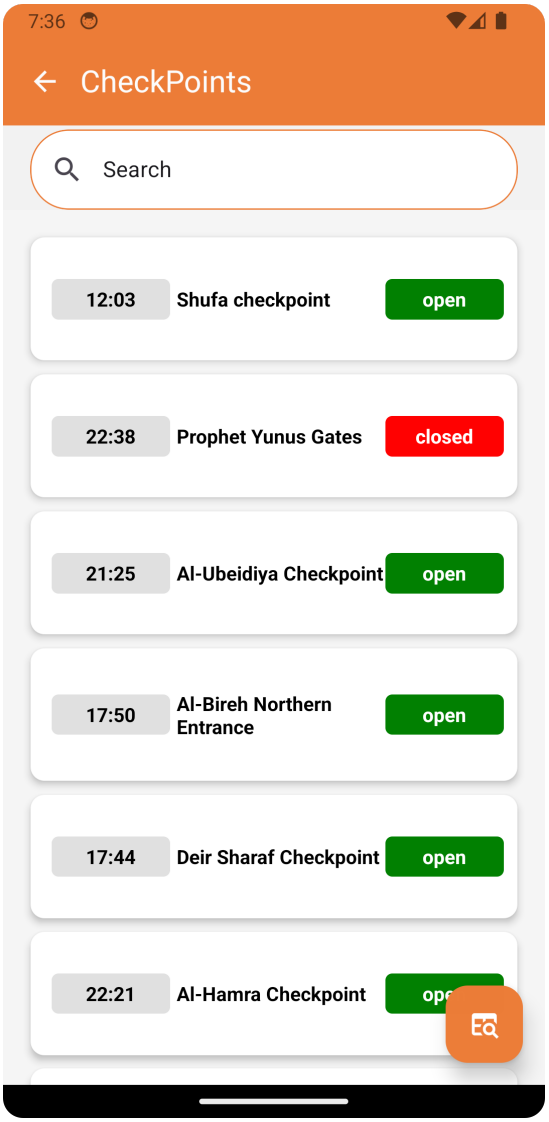


Figure 4.36: Status of Nearby Barriers

This feature ensures you stay informed about the conditions affecting your travel.

Job Application for Drivers

You can apply for a vacant job to become a driver in the application or an official driver on one of the lines. Once you choose the job that suits you, you can click to view the job requirements.

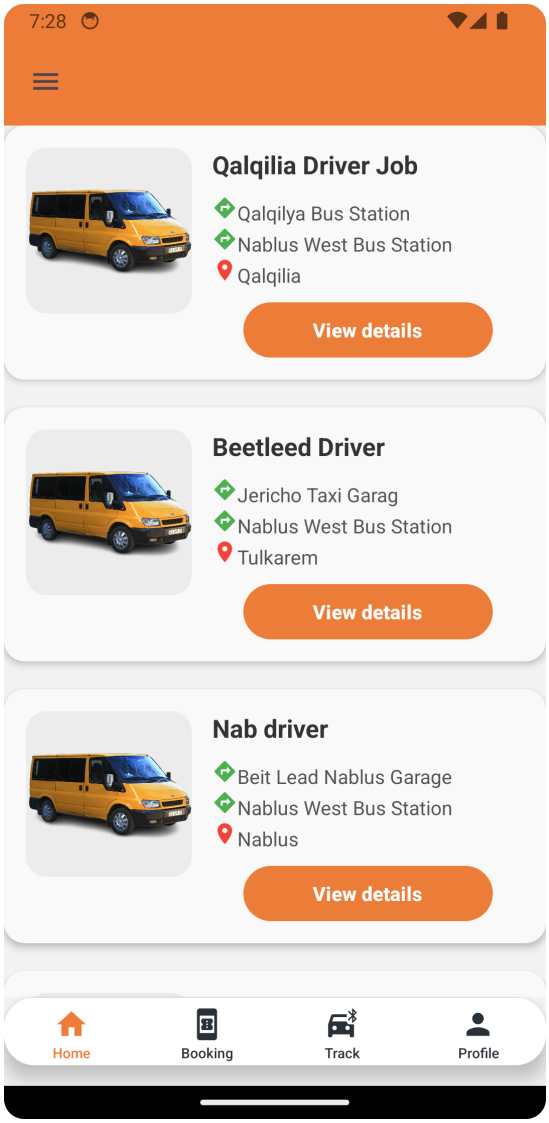


Figure 4.37: Vacant Jobs for Drivers

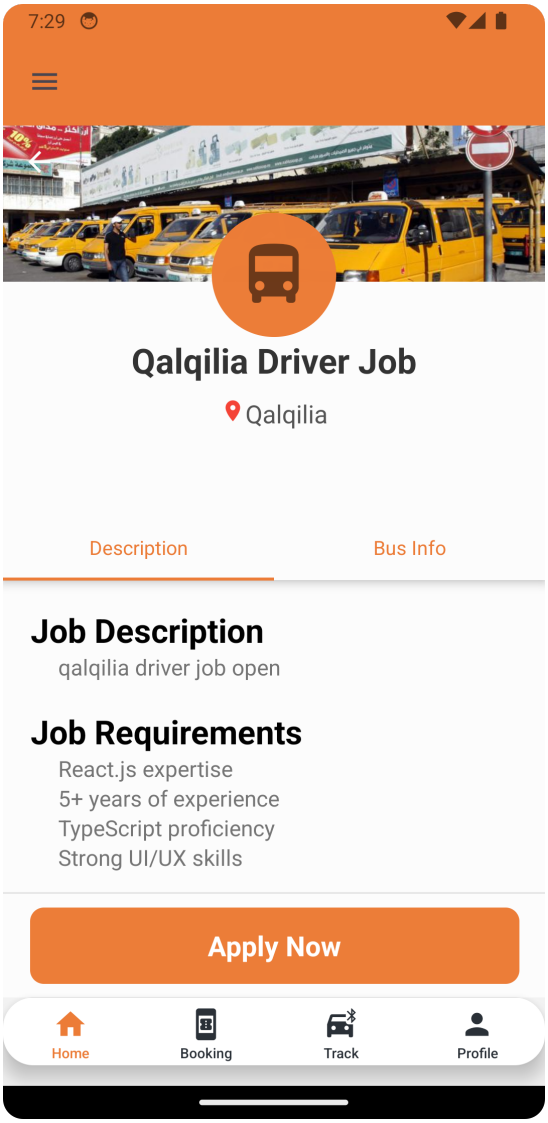


Figure 4.38: Job Requirements

After reading the requirements, you will need to upload your CV and a valid driver's license.

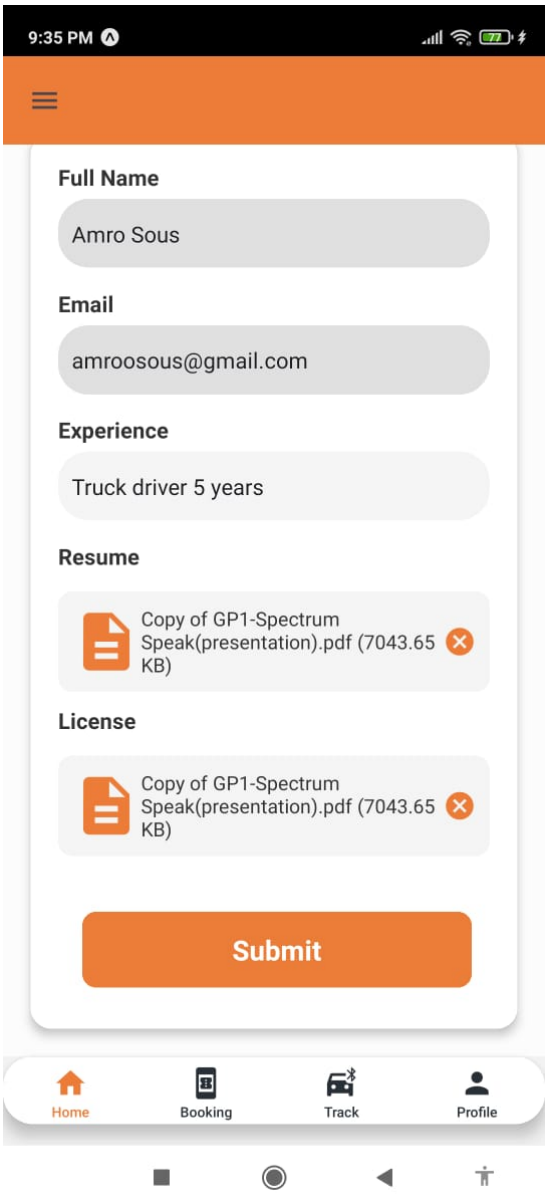


Figure 4.39: Submit Job Application Form

Figures 4.37, 4.38, and 4.39 illustrate the job application process for drivers: 4.37 shows the vacant jobs for drivers, 4.38 presents the job requirements, and 4.39 demonstrates the submission of the job application form.

Booking Process

At the beginning of the booking process, the application requires permission to access your current location to provide the best results, as shown in Figure 4.74.

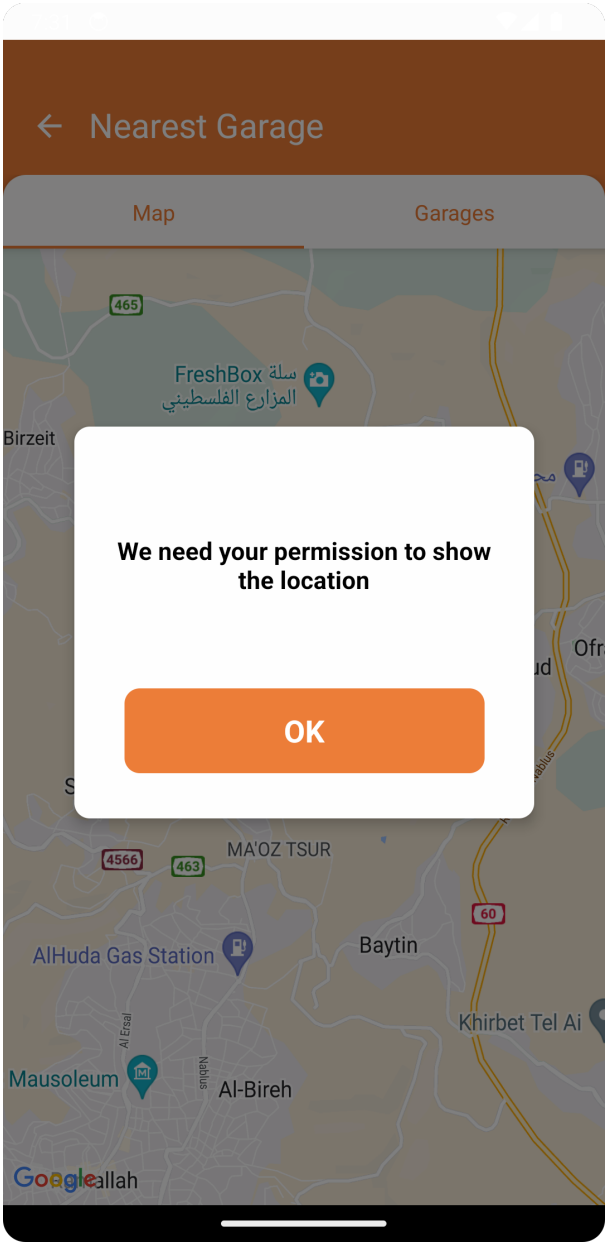


Figure 4.40: Granting Location Permission

Next, the user must enter the destination, which can be a city or a well-known location on the maps, as shown in Figure 4.41. A marker appears on the selected location, as depicted in Figure 4.42.

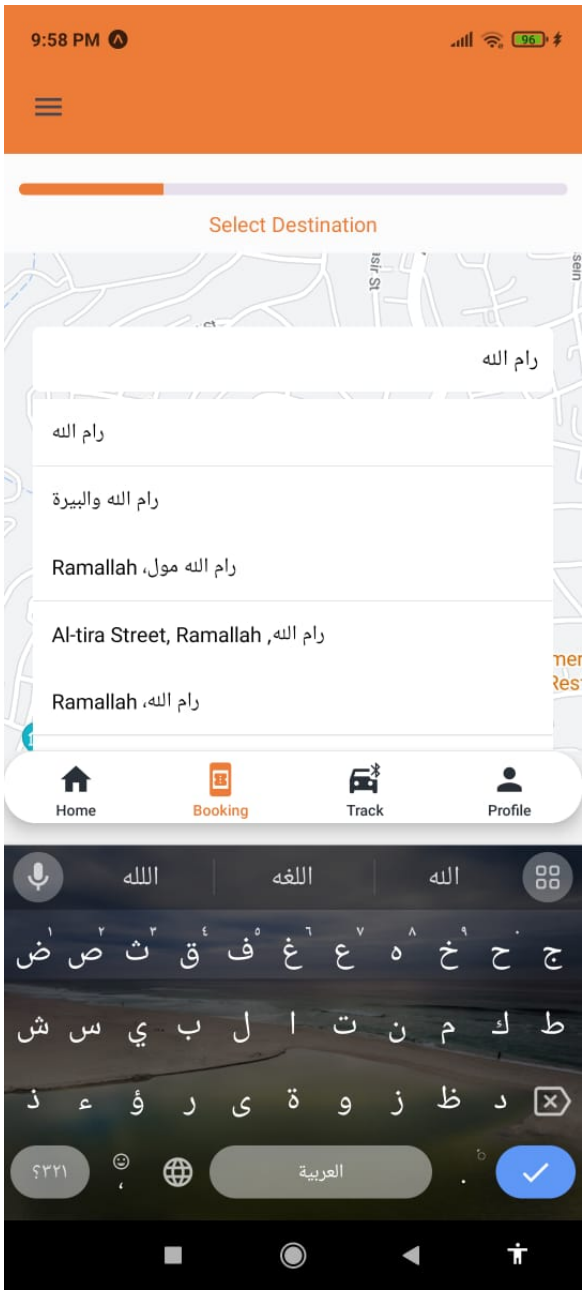


Figure 4.41: Entering Destination

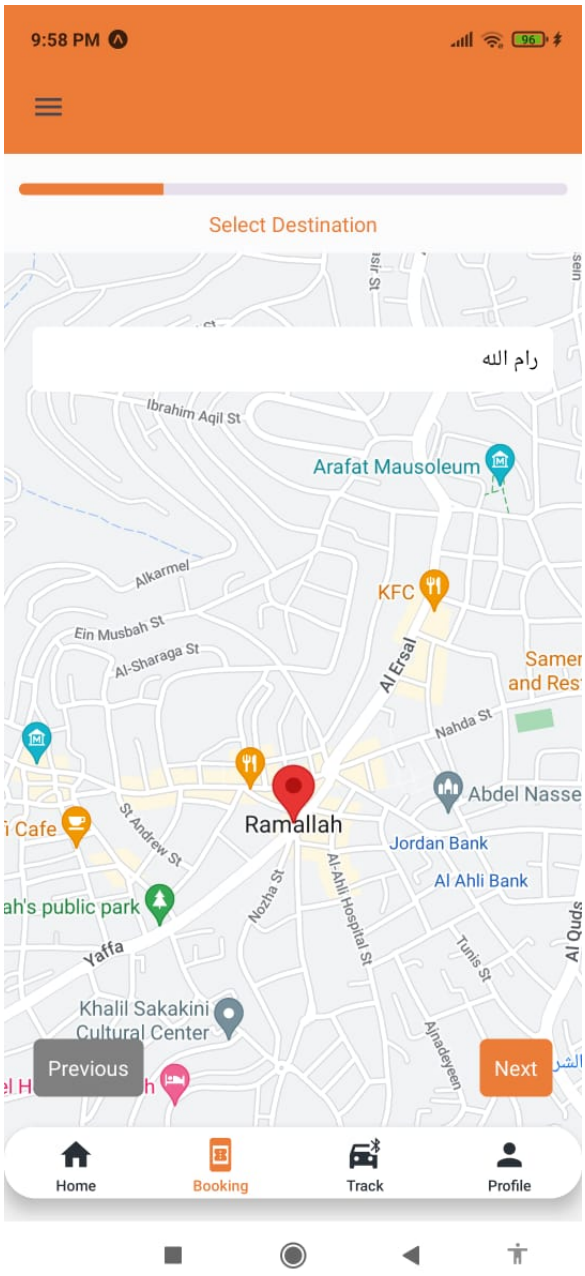


Figure 4.42: Selected Destination Marker

After proceeding to the next step, the application suggests garages to use for reaching the destination, as shown in Figure 4.43.

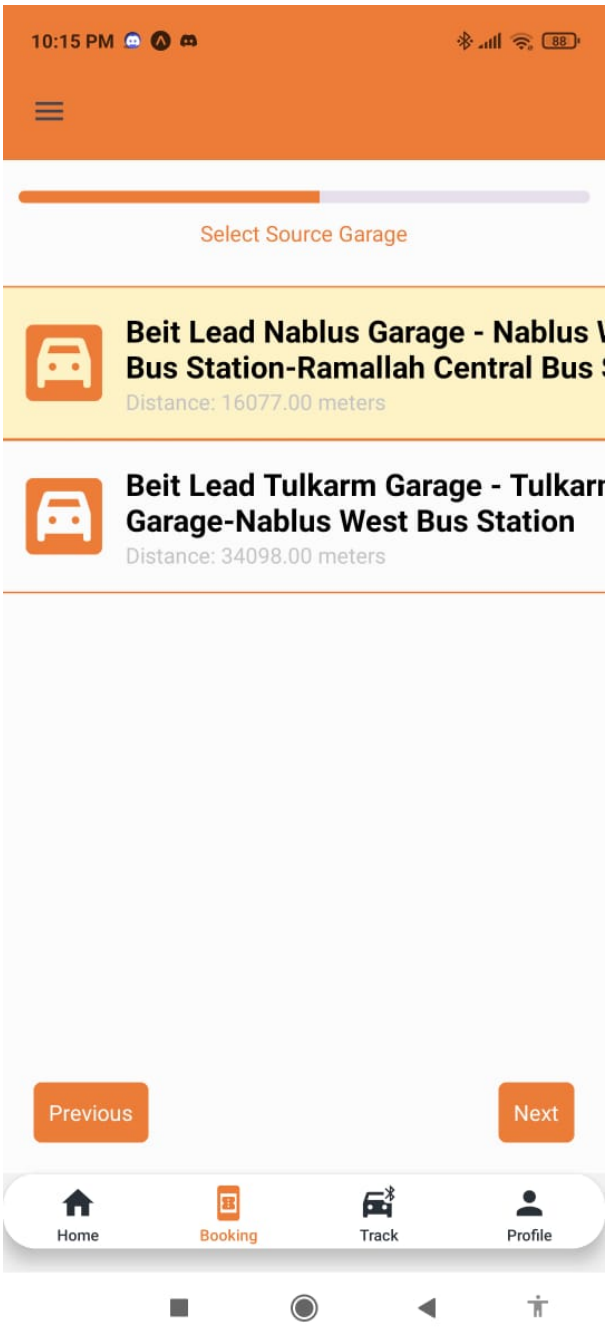


Figure 4.43: Garage Suggestions

The next step displays the full route with available stoppoints for booking, as illustrated in Figure 4.44. Upon selecting a stoppoint (Figure 4.45), the process moves to the fourth step.

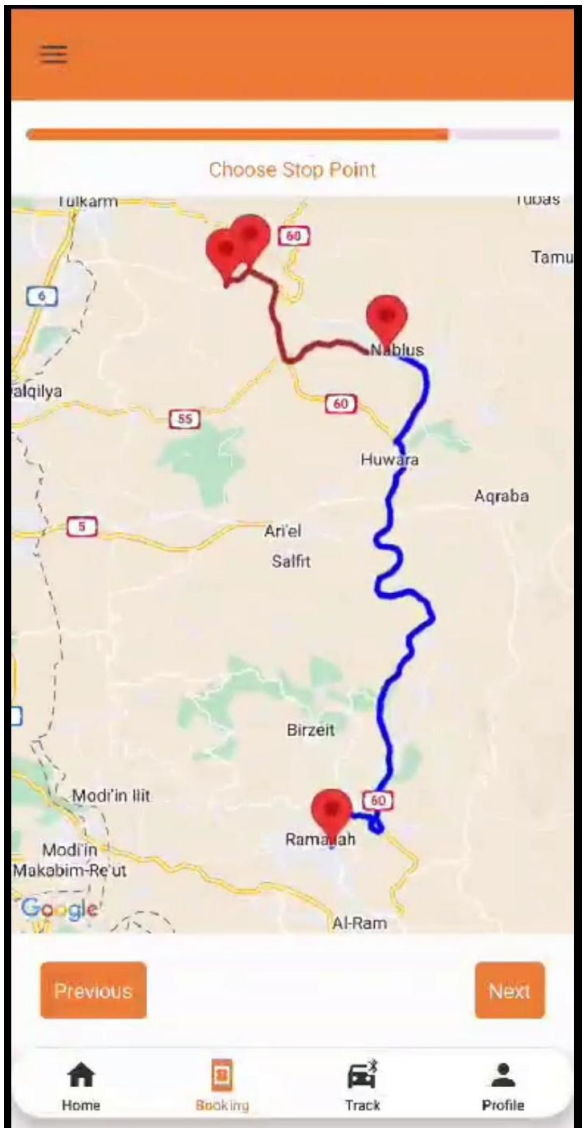


Figure 4.44: Route with Available Stoppoints

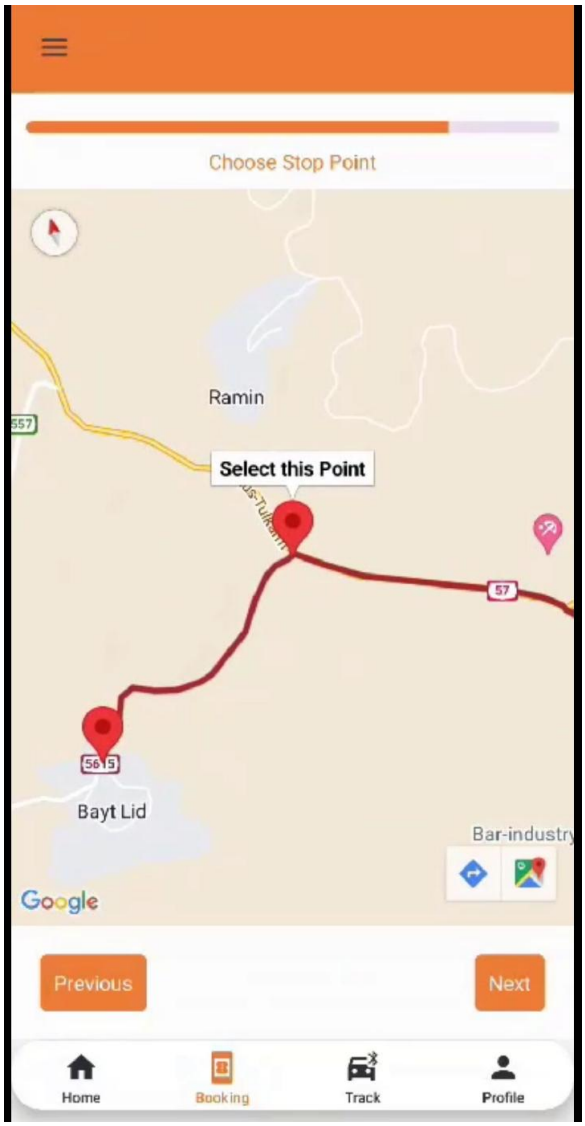


Figure 4.45: Selected Stoppoint

This step has two possible cases:

- **Case 1: A car is available in the garage**
  The available car is displayed, and users can book seats by clicking on them, as shown in Figure 4.46. The user can select any number of seats and proceed to payment (Figure 4.47). Once confirmed, the car waits for departure, and the rider can book additional trips.

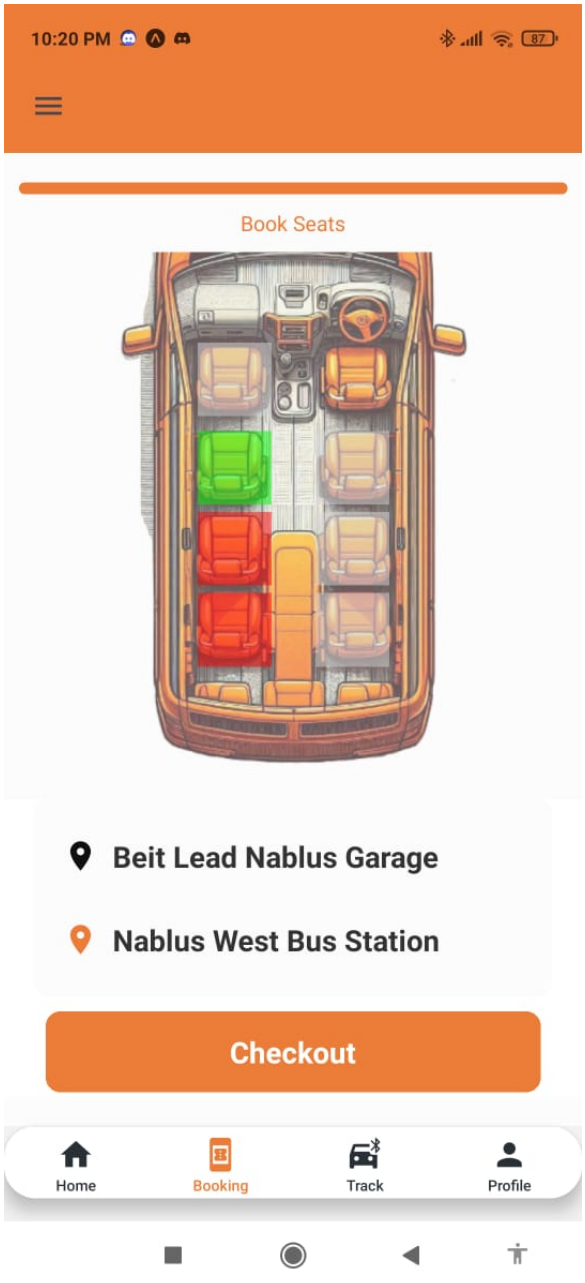


Figure 4.46: Selecting Seats for Booking

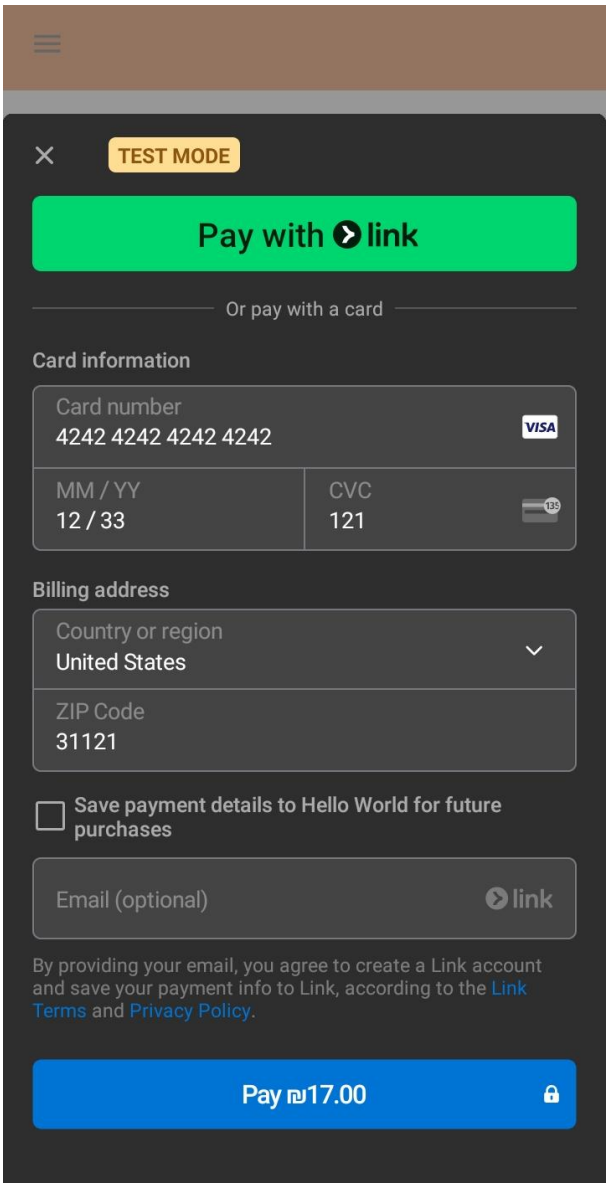


Figure 4.47: Payment Confirmation

- **Case 2: No car is available in the garage**
  If no car is available, a reservation card appears, allowing users to queue for buses, as shown in Figure 4.48. Users can book up to seven passengers (Figure 4.49), and they receive a waiting number until their turn arrives. Once a seat is assigned, they are automatically registered and notified.

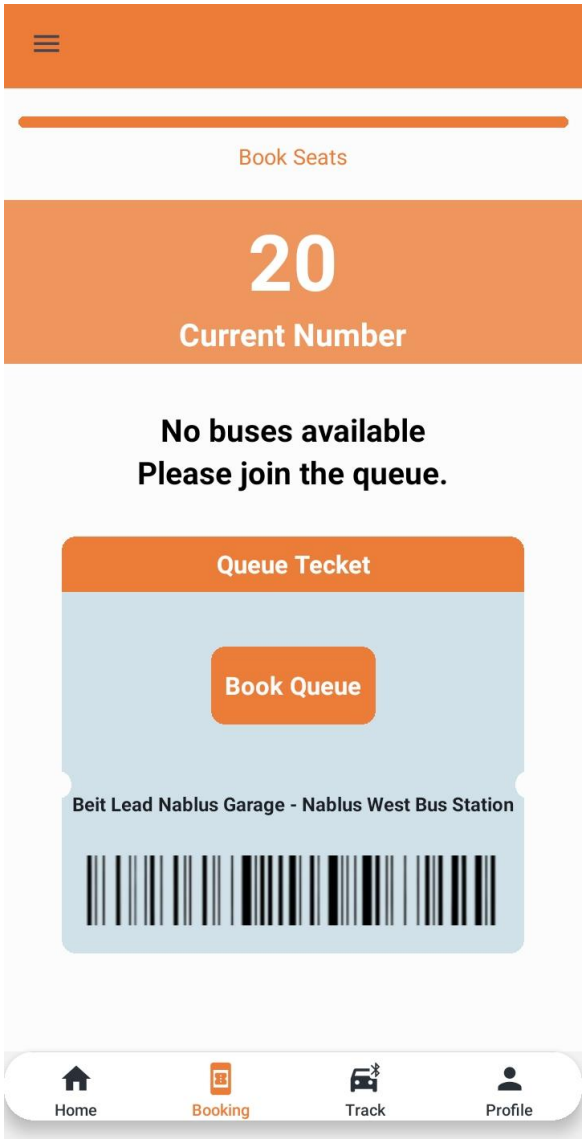


Figure 4.48: Queueing for a Ride

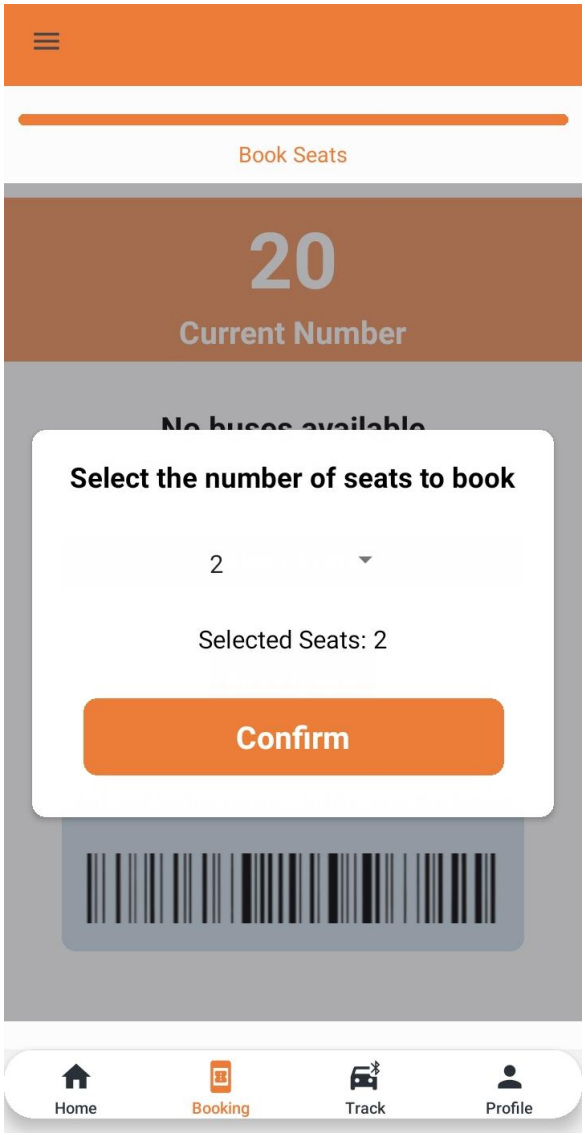


Figure 4.49: Selecting Number of Passengers in Queue

Track Bookings

When you go to the track, you can view the trips that you have booked as shown in the figure below.

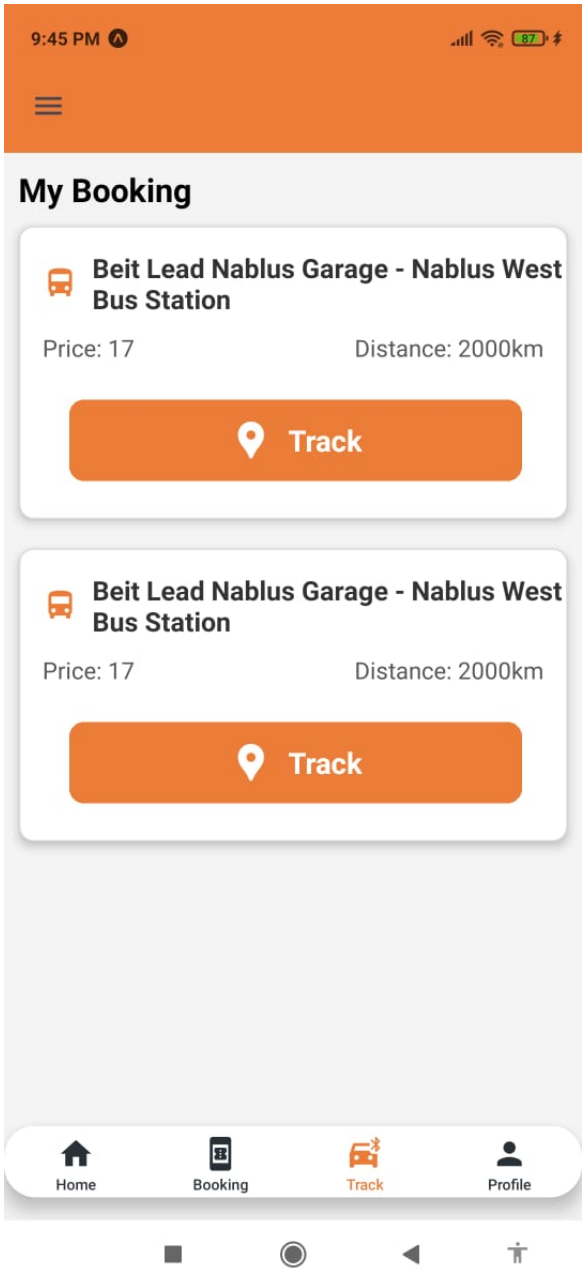


Figure 4.50: Track Bookings

Tracking the Trip

When you click on the trip you want to track, there are messages for communication between the rider and the car, as shown in the figure below.

Figure 4.51: Messages between Rider and Car

Trip Ticket

The ticket for the selected trip is displayed, confirming that you are following this trip, as shown in the figure below.

Figure 4.52: Trip Ticket

Track Map

You can see the entire trip on the map, as shown in the figure below.

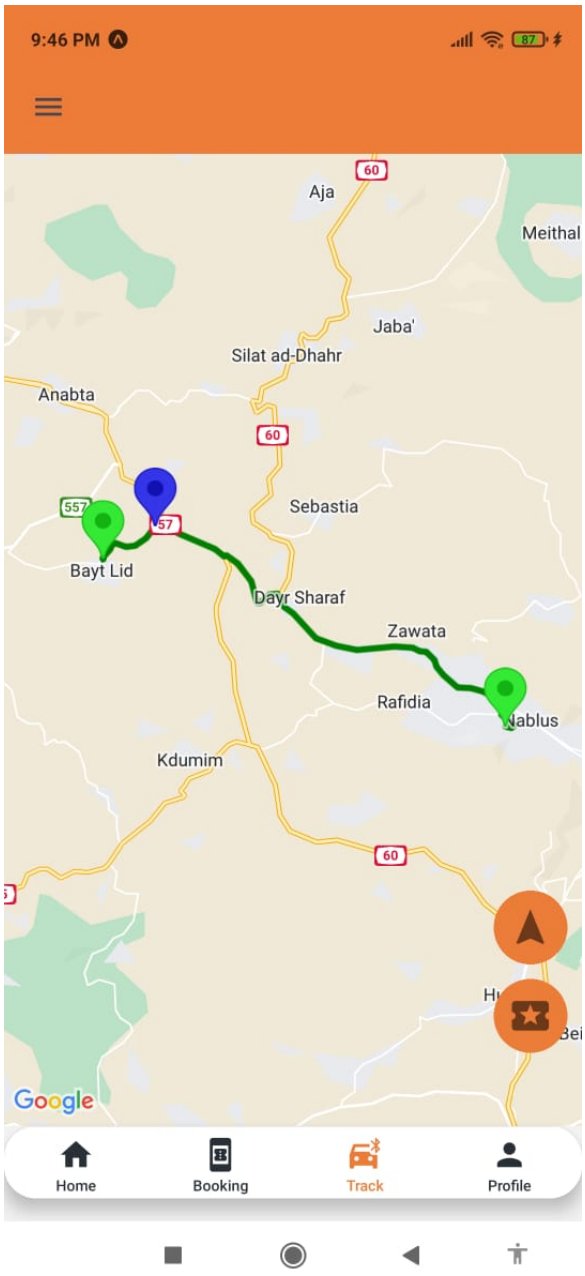


Figure 4.53: Track Map

Driver and Trip Rating

At the end of the trip, a list appears to evaluate the driver and the trip, as shown in the figure below.

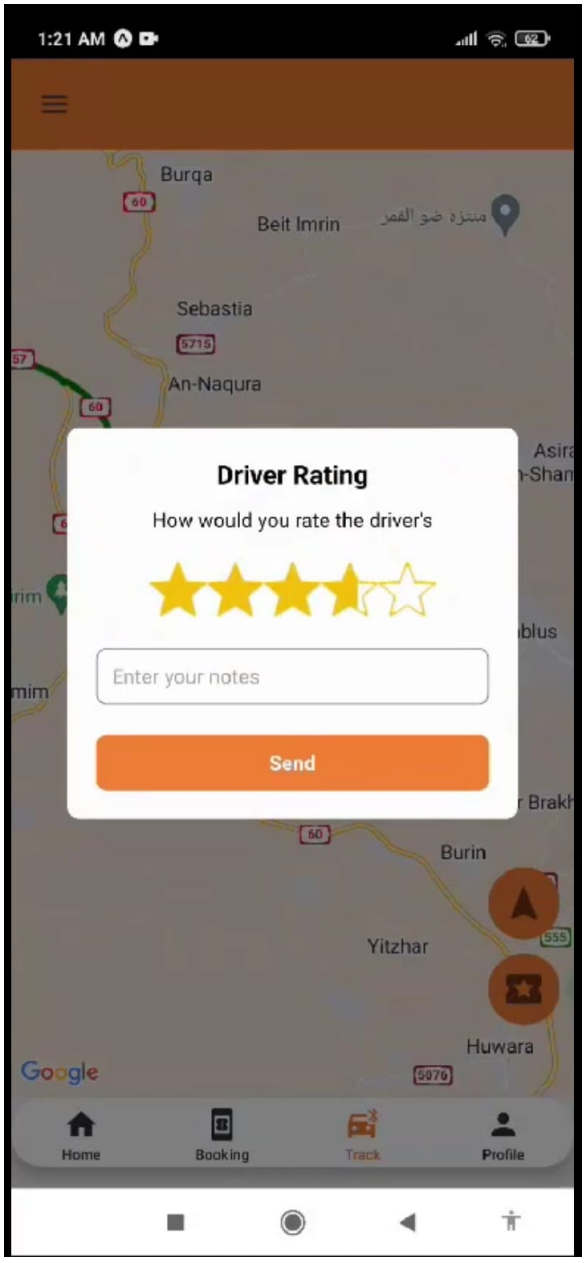


Figure 4.54: Driver and Trip Rating

3.2.1.3 Driver Side

Overview of Process

The trip booking workflow in this image illustrates the step-by-step process from a driver's perspective, ensuring an efficient and organized transportation system. The process begins with the driver starting work at an assigned garage, where the system verifies the assignment and retrieves relevant details. Next, the driver enters the queue and receives their position, allowing for efficient scheduling. Once in the system, the driver can view reserved seats, and the system displays available and booked seats, enabling the driver to manage passenger reservations. After successfully booking passengers, the system confirms the reservations, and the driver proceeds to start the trip. During the journey, a trip tracking screen is activated for real-time monitoring. Additionally, the driver can view road conditions and receive important navigation updates to ensure a smooth trip. Passenger management is also facilitated, as the driver can access the passenger list and reservation details. Finally, with all necessary information in place, the driver is fully prepared to start the trip with passengers. This workflow seamlessly integrates driver assignment, queue management, passenger booking, trip tracking, and navigation support, creating a streamlined and effective transportation system.

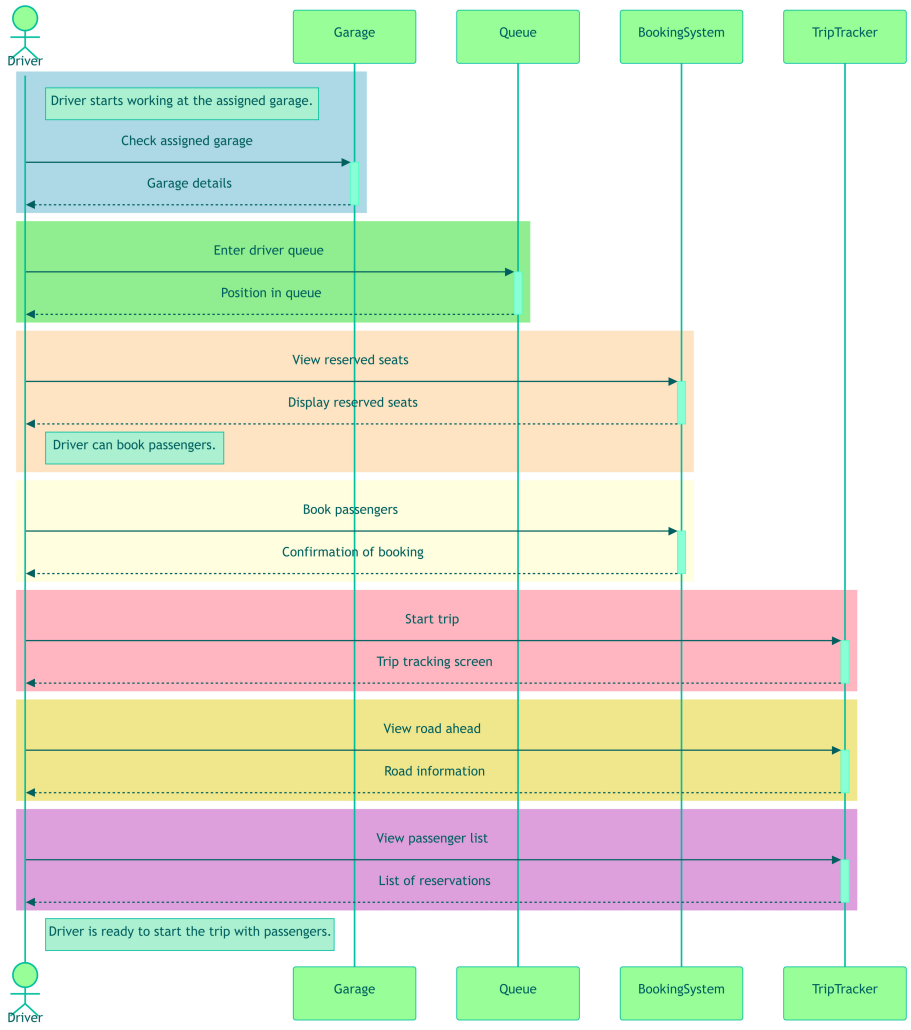


Figure 4.55: Booking Rider Process

Driver's Login Page

The login page allows the driver to access the application and use its features.

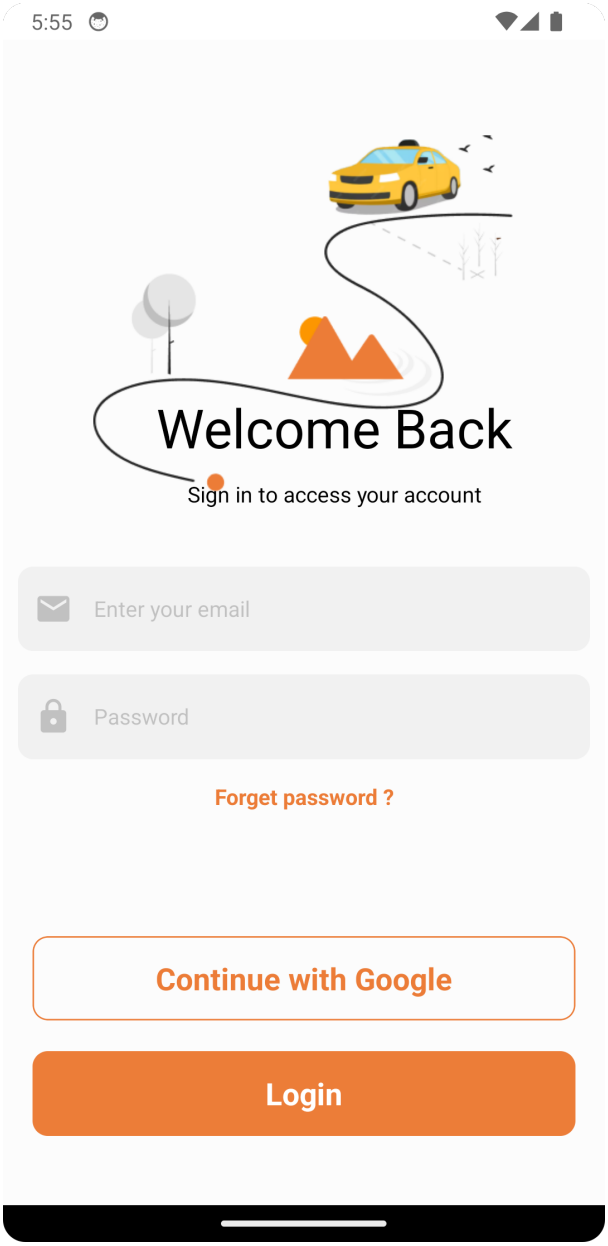


Figure 4.56: Driver's Login Page

Home Page

After logging in, the home page displays weather conditions, a chat for communication among drivers, Palestine news, personal bus information, and a queue list for vehicles in the garage.

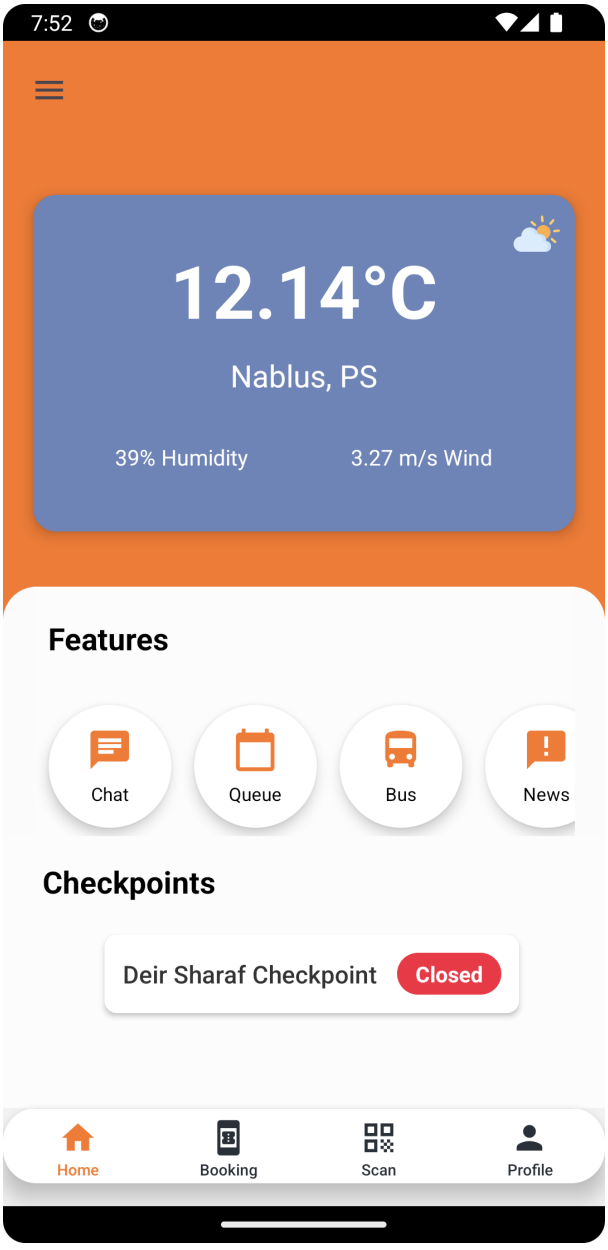


Figure 4.57: Home Page

Route Chat

A chat system enables drivers from the same line to discuss road conditions and common checkpoints.

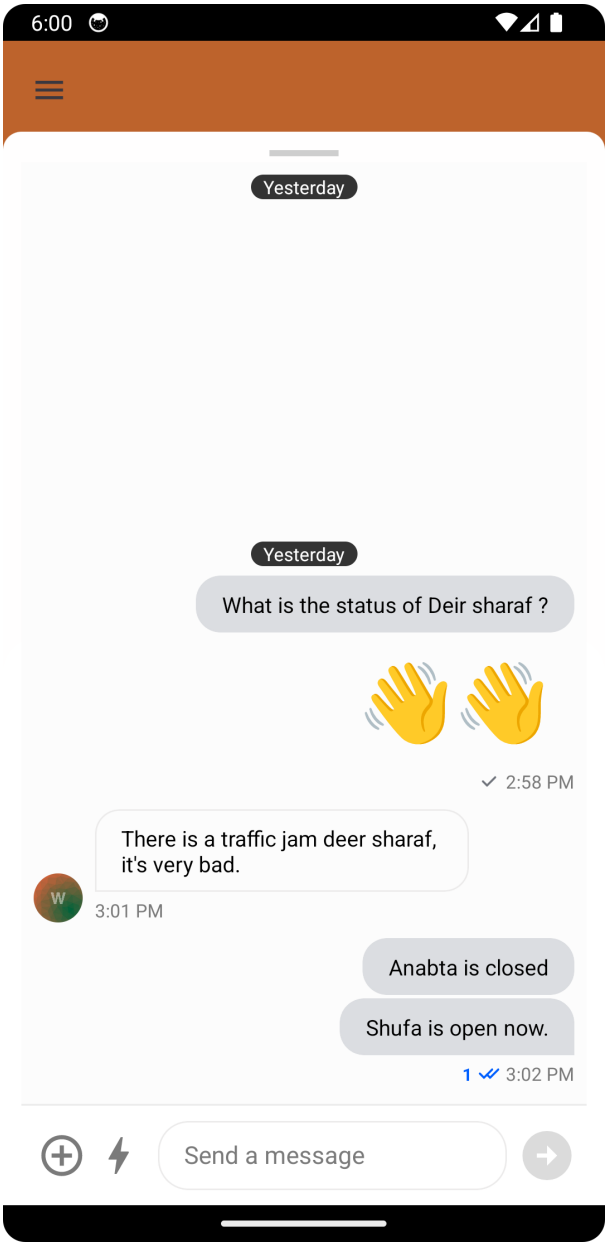


Figure 4.58: Route Chat

News Section

The home page also features Palestine news updates.

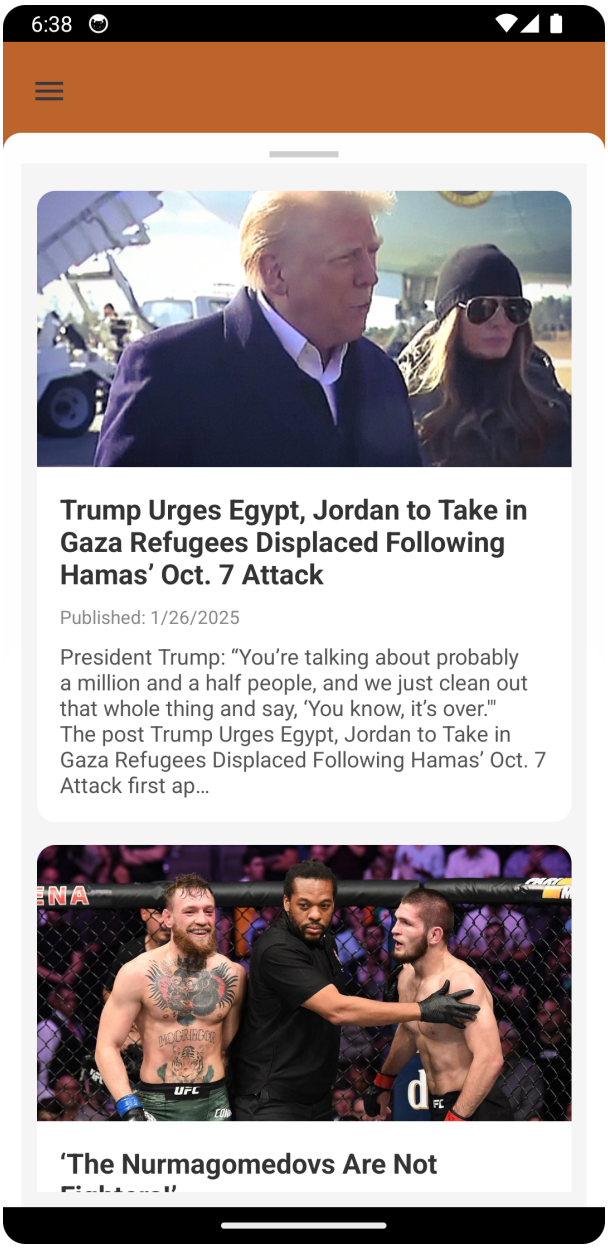


Figure 4.59: News Section

Bus Profile

Drivers can view and manage their personal bus information.

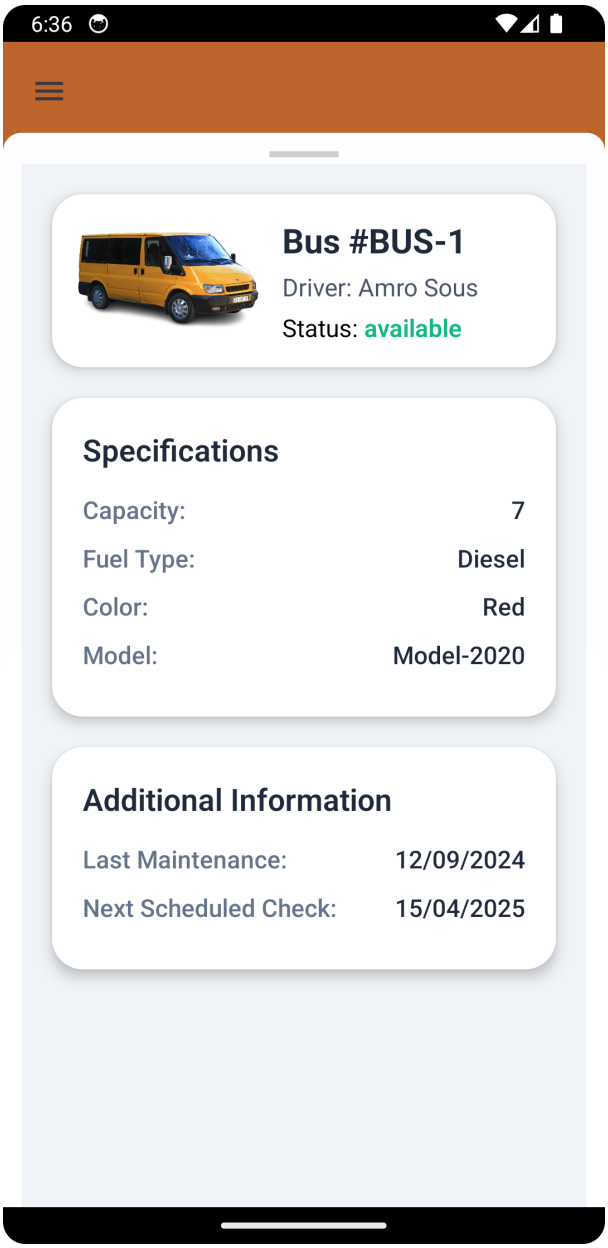


Figure 4.60: Bus Profile

Queue List and Bus Info

The waiting list of cars in the garage is arranged based on arrival time. The driver can start or end work from this section. Clicking on a driver's name in the queue displays their detailed information.

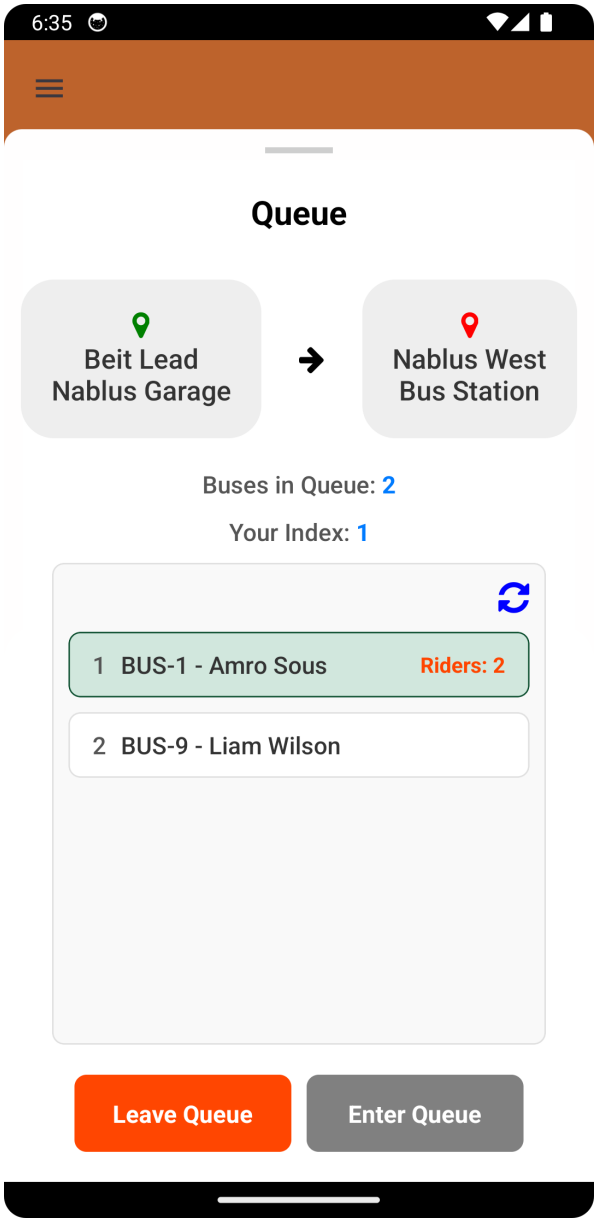


Figure 4.61: Queue List

Figure 4.62: Queue Bus Info

Booking Process

The driver sees the number of passengers who have booked seats in their vehicle.

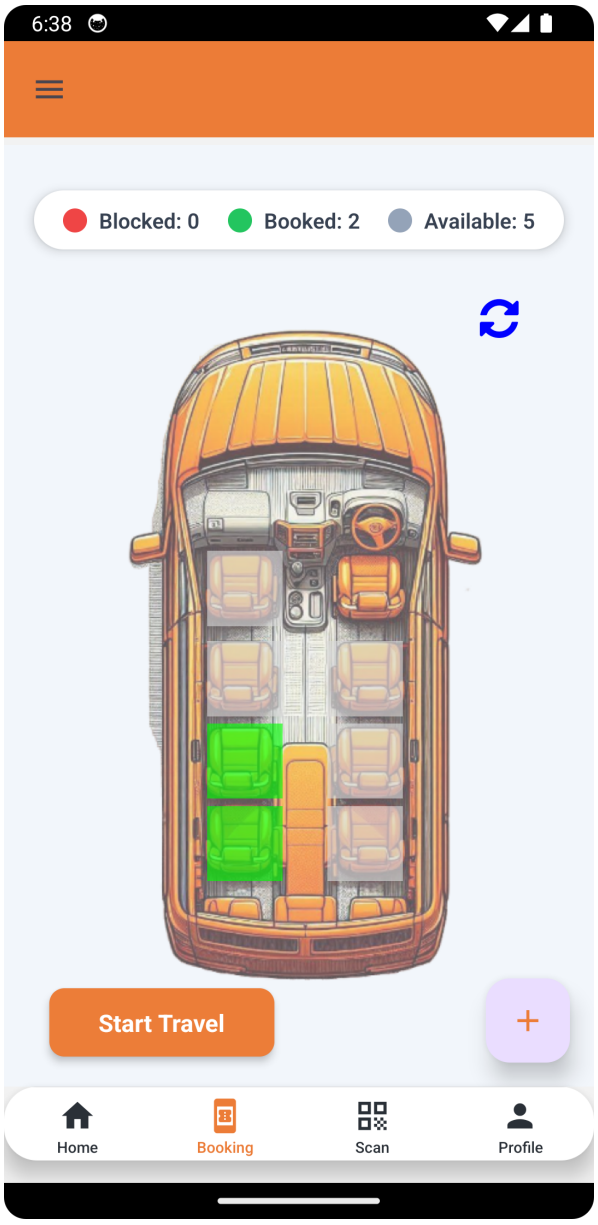


Figure 4.63: Booking Process

Manual Booking

The driver can manually book seats for people without the application.

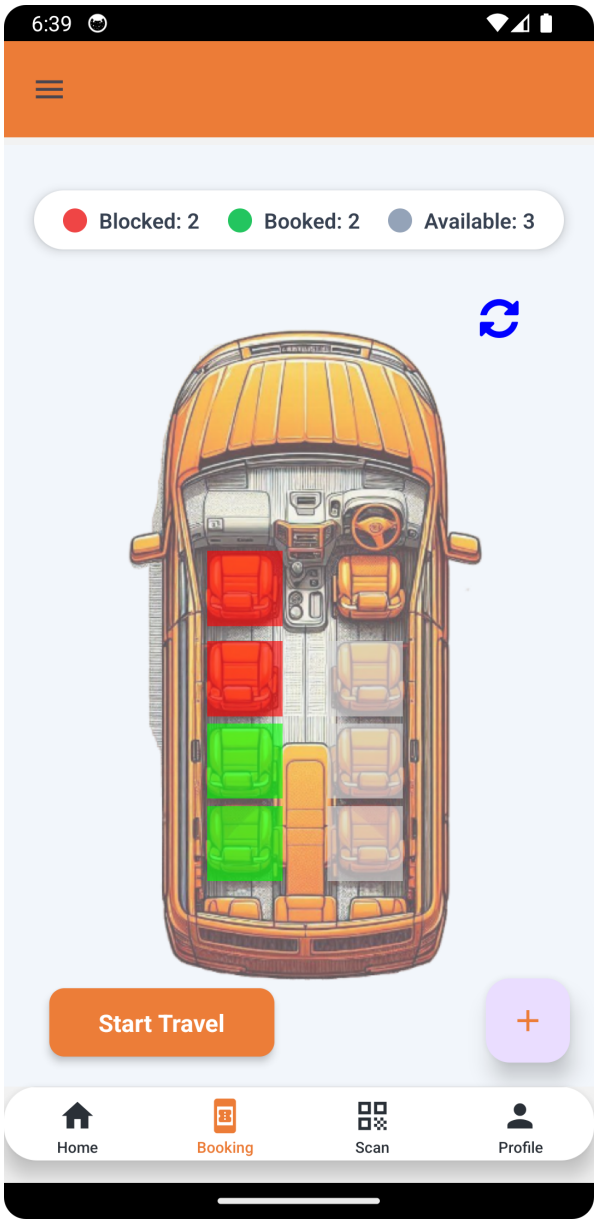


Figure 4.64: Manual Booking

Booking Notifications

The driver receives a notification when a passenger books a seat.

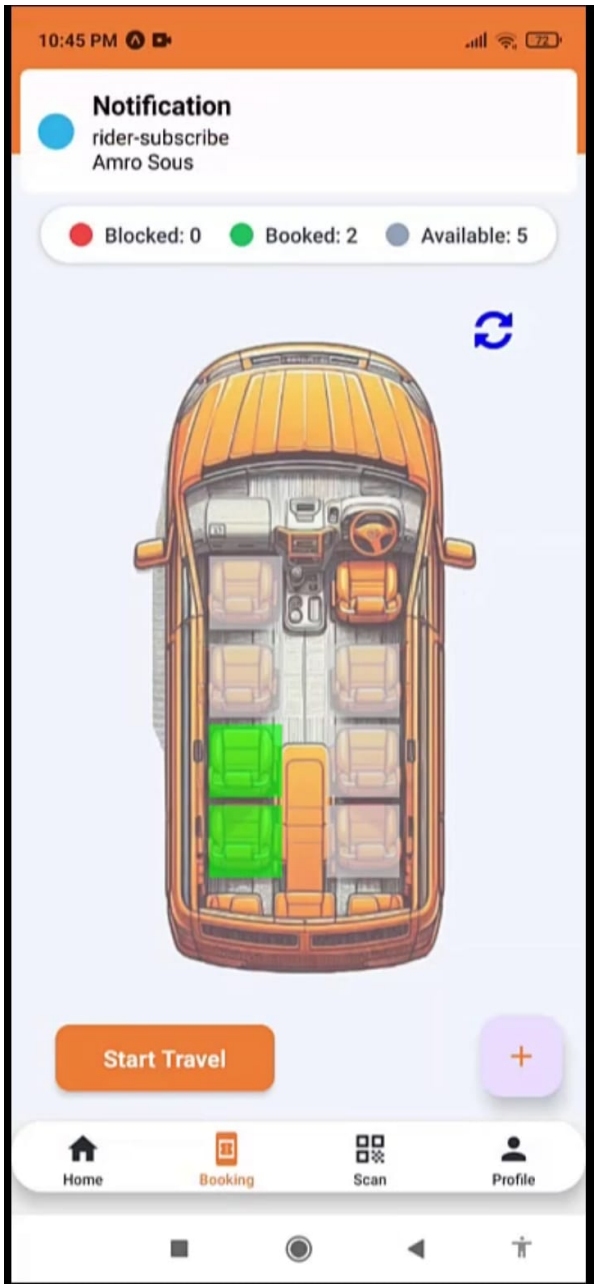


Figure 4.65: Booking Notification

Trip Map and Checkpoints

The driver can view the trip map, including checkpoint statuses and passenger stop points.

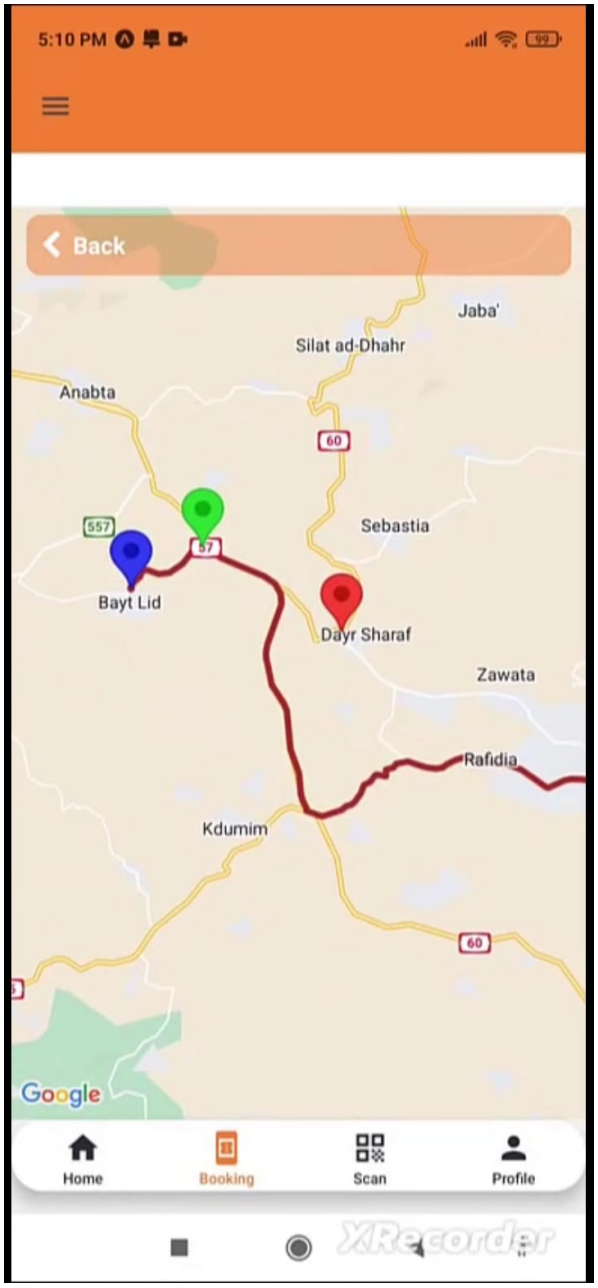


Figure 4.66: Trip Map View

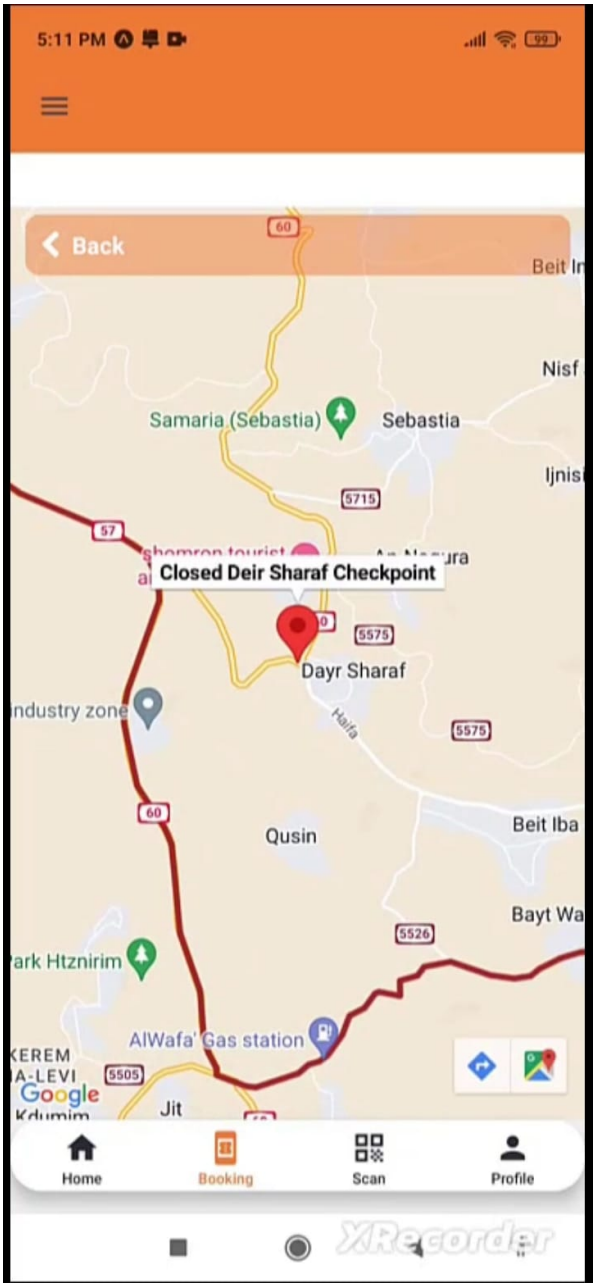


Figure 4.67: Checkpoint Status

Booking Notification

Passengers' details and their stop points can be viewed. Additionally, passengers are notified when the vehicle reaches full capacity.

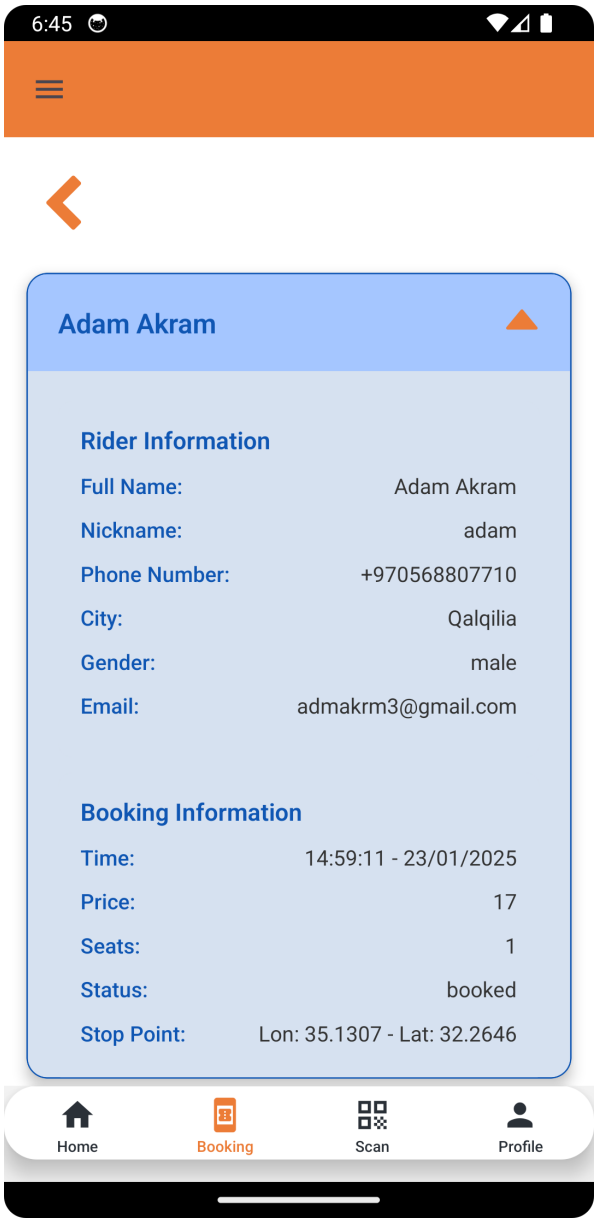


Figure 4.68: Booking Information

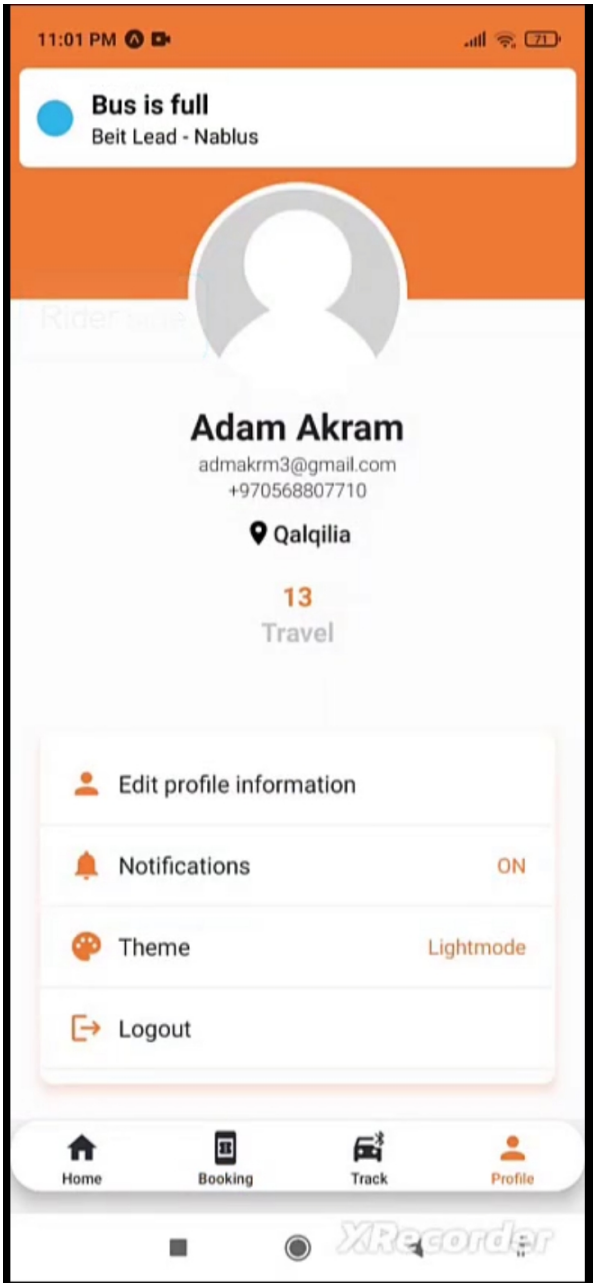


Figure 4.69: Bus Full Notification

Trip Start Notification

A notification is sent to passengers five minutes before departure.

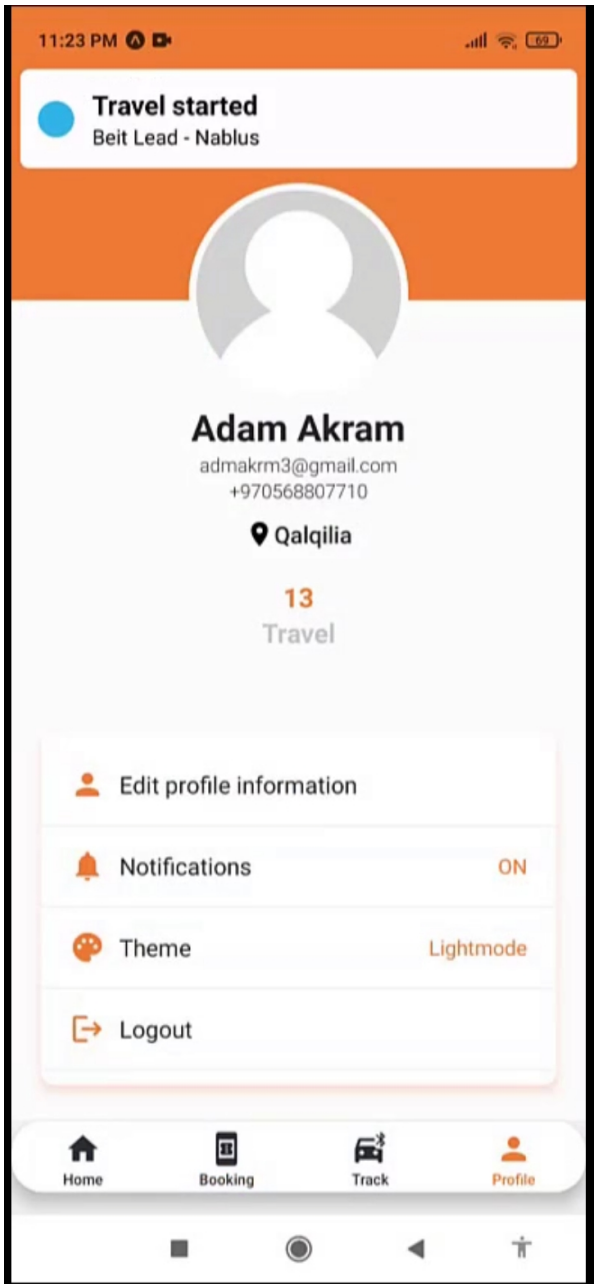


Figure 4.70: Trip Start Notification

Trip Chat

The driver can communicate with passengers during the trip.

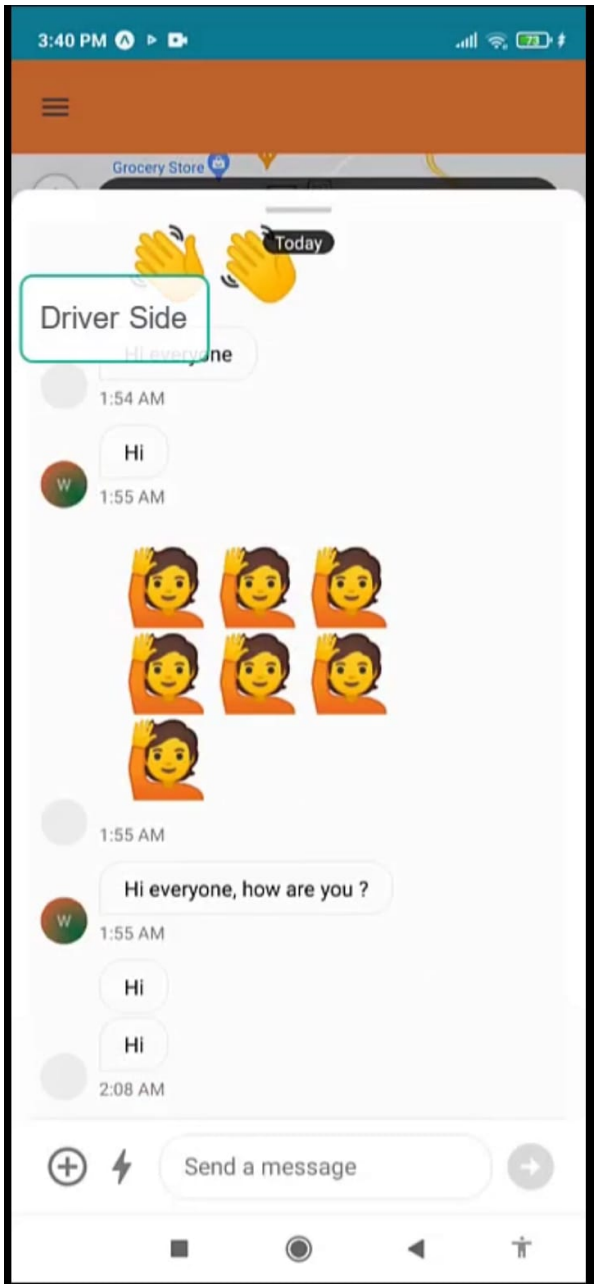


Figure 4.71: Driver-Passenger Chat

Route Tracking

The road can be displayed from different perspectives with detailed information about the route.

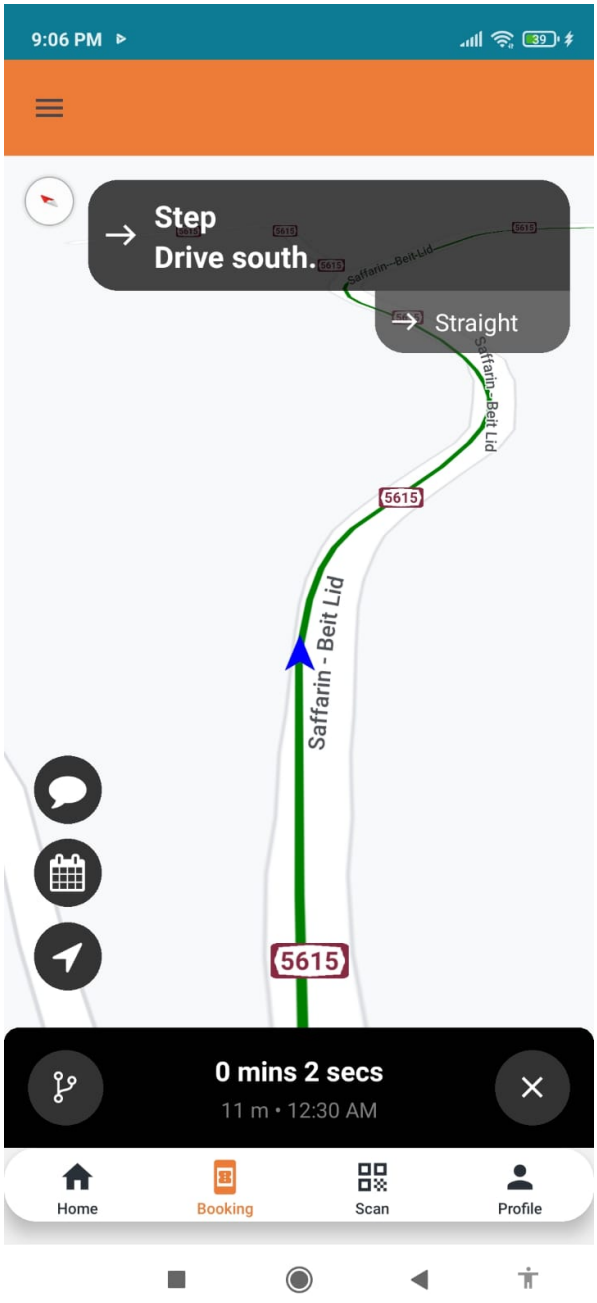


Figure 4.72: Route Tracking

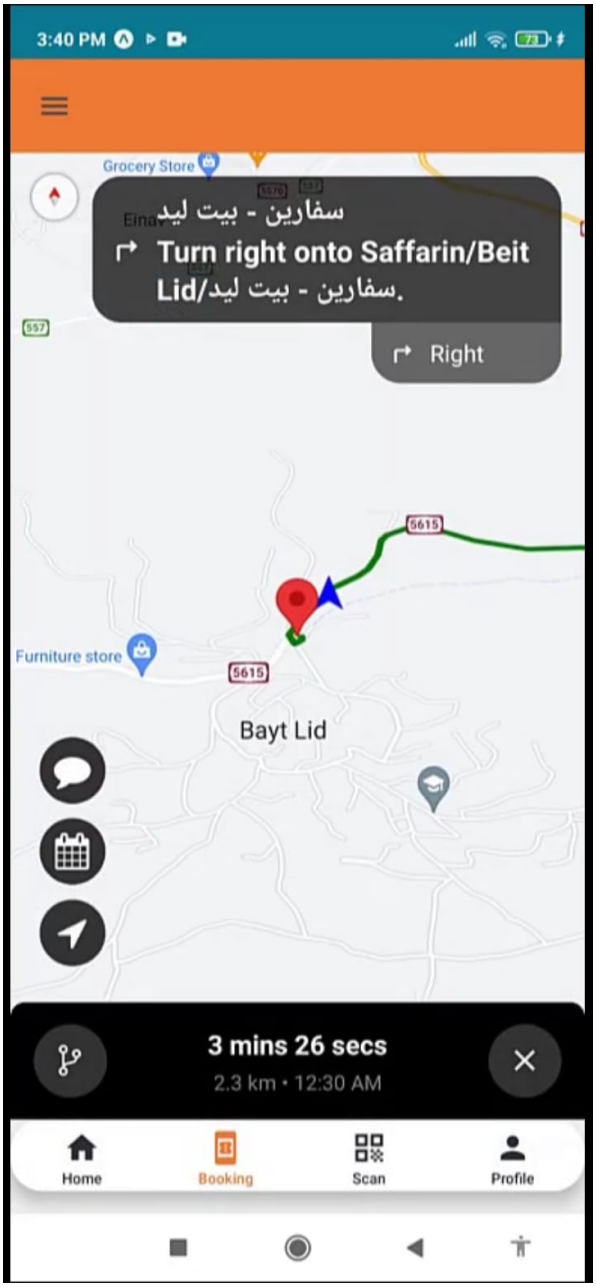


Figure 4.73: Top View Tracking

QR Code Scanning

Drivers can scan a rider's ticket via QR code scanning.

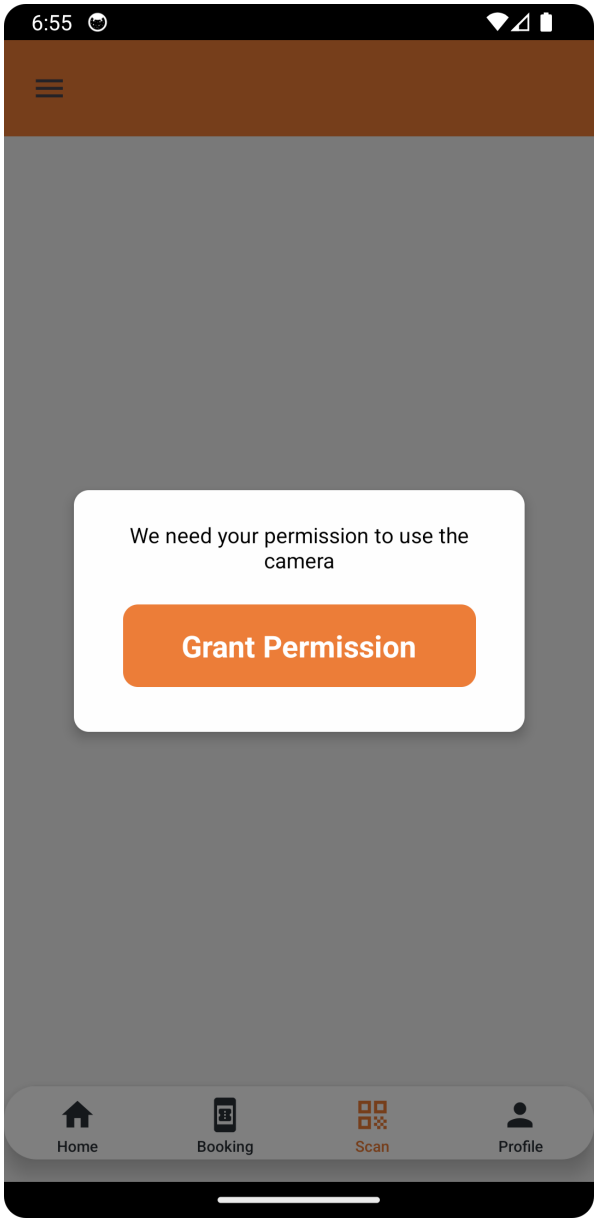


Figure 4.74: Grant Camera Permission

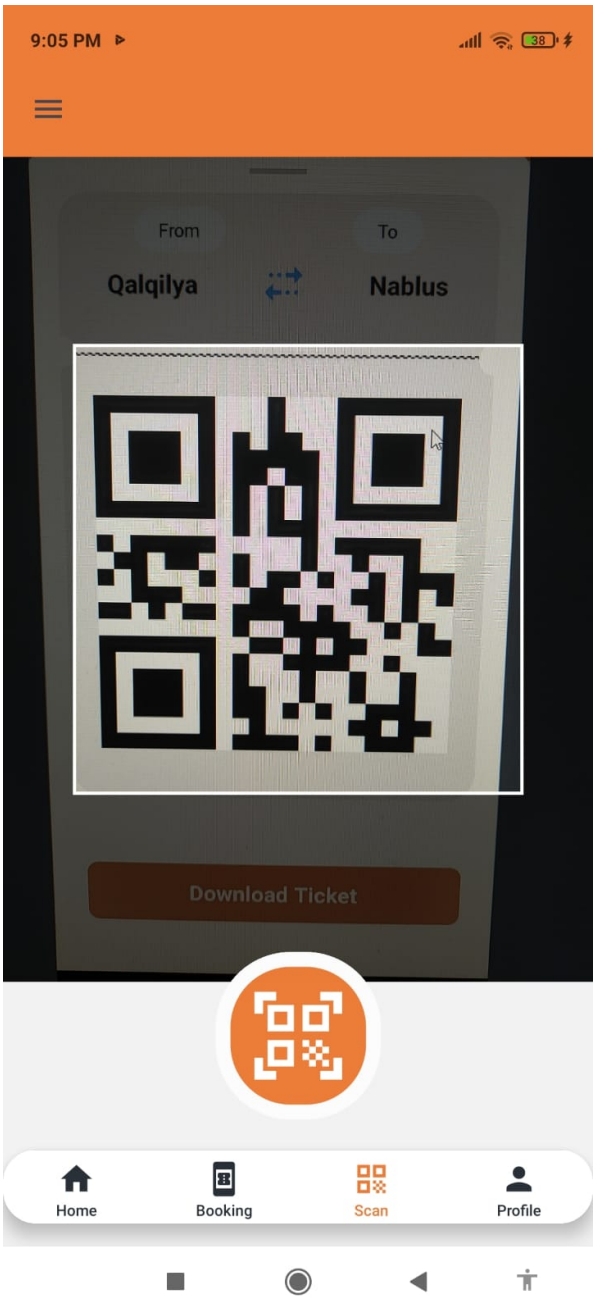


Figure 4.75: QR Code Scanning

Driver Profile

Drivers can view and edit their profile information.

Driver Profile and Edit Profile

The driver can view their profile and edit their details.

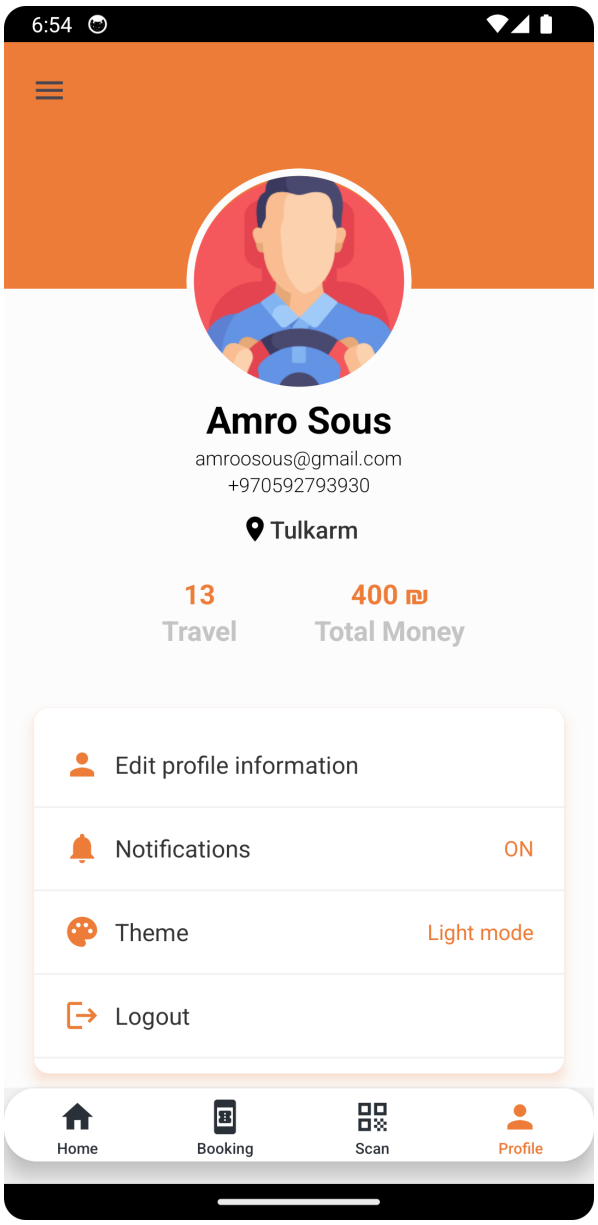


Figure 4.76: Driver Profile

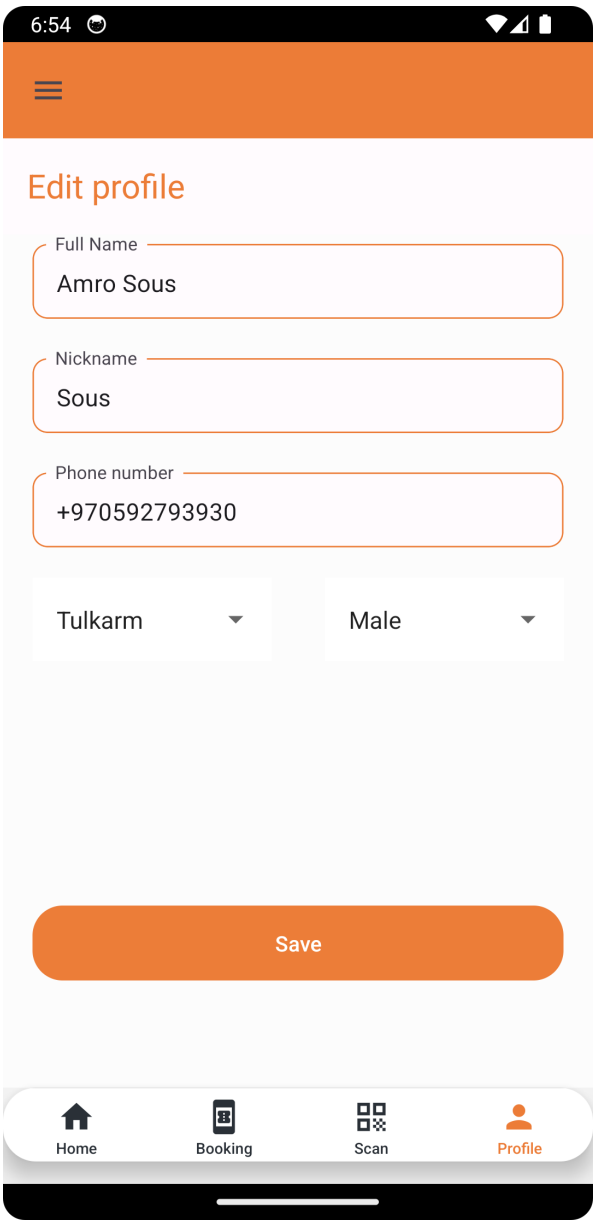


Figure 4.77: Edit Profile

3.2.2 Web Application

3.2.2.1 Admin

Login Page

The login page allows the admin to enter their credentials to log into the system.

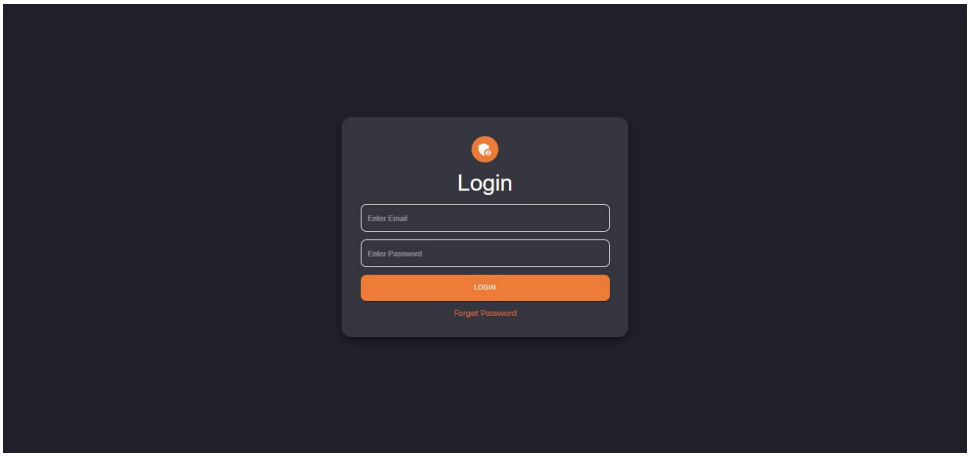


Figure 4.78: Login Page

Main Dashboard

After logging in, the admin is directed to the main page of the dashboard, which contains key statistics and operational details about the application. It includes:

- Number of cars, routes, and garages.
- List of the latest barrier statuses.
- General statistics about the application and cars.
- A tracking table showing the status of cars on the road.

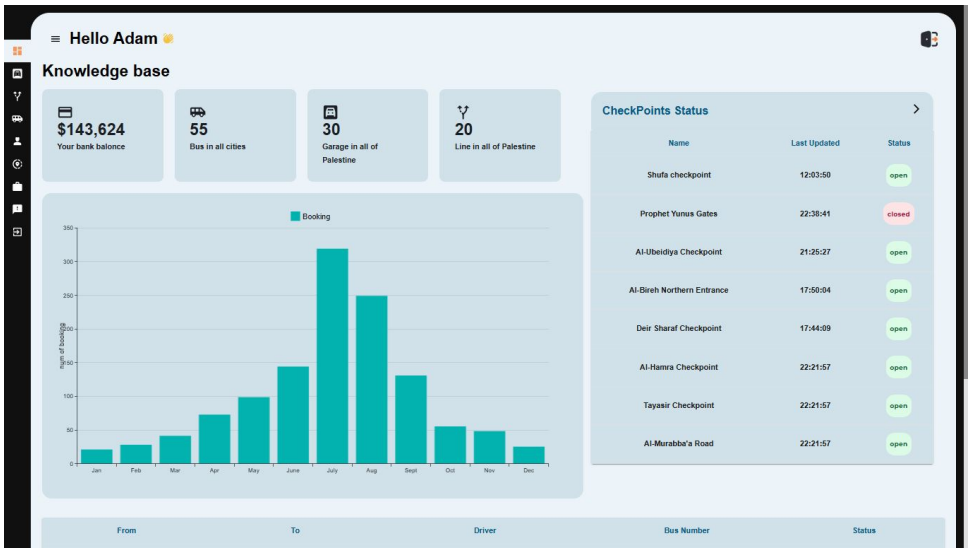


Figure 4.79: Main Dashboard

Checkpoints Section

The checkpoints section displays a complete list of barriers. The admin can edit their location and name or delete them.

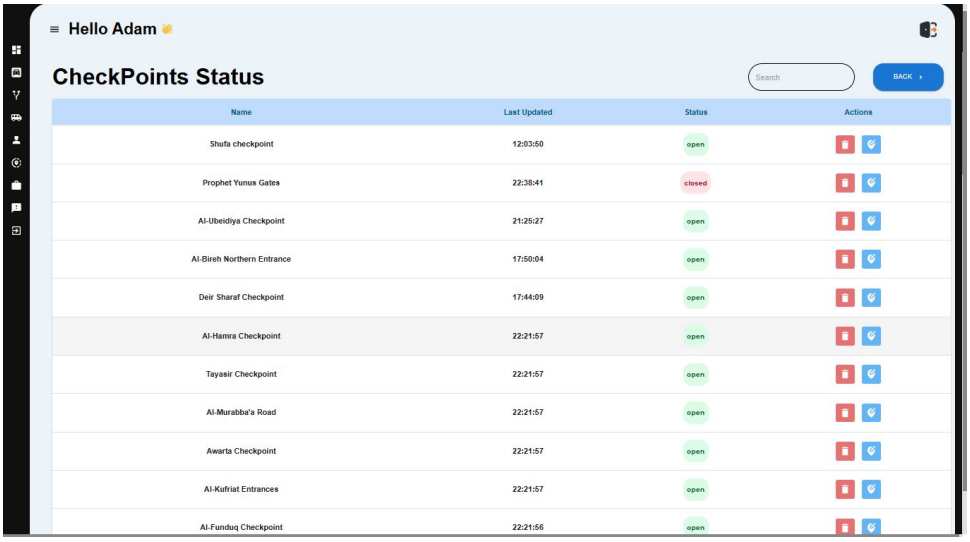


Figure 4.80: Checkpoints List

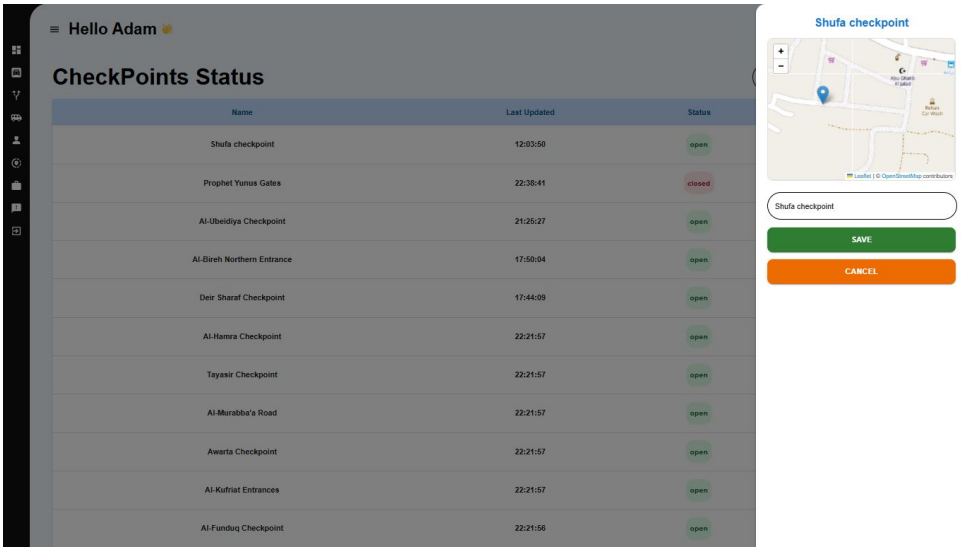


Figure 4.81: Edit Checkpoint

Garages Section

The garages section displays a list of all garages, and the admin can add new garages by clicking on any point on the map. The admin can also search for garages by name, edit, or delete them.

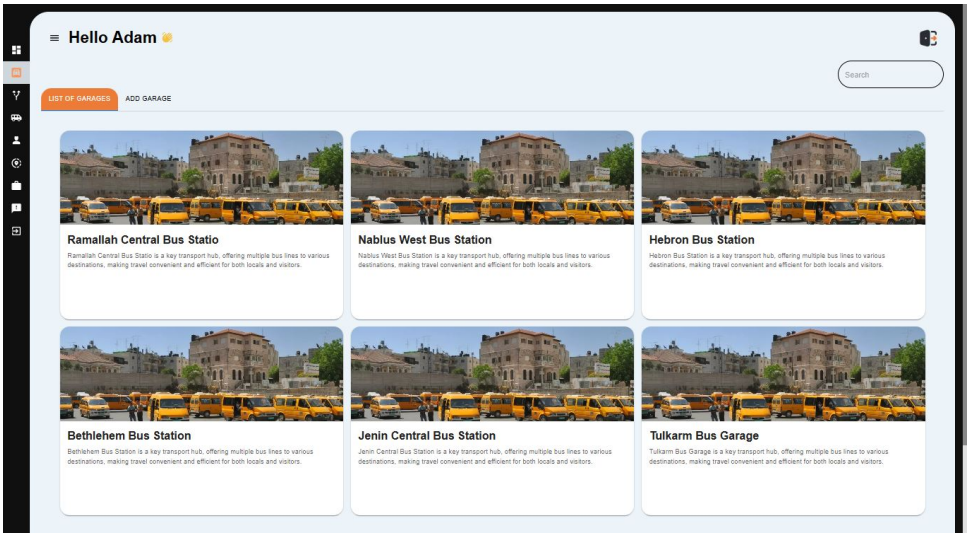


Figure 4.82: Garages List

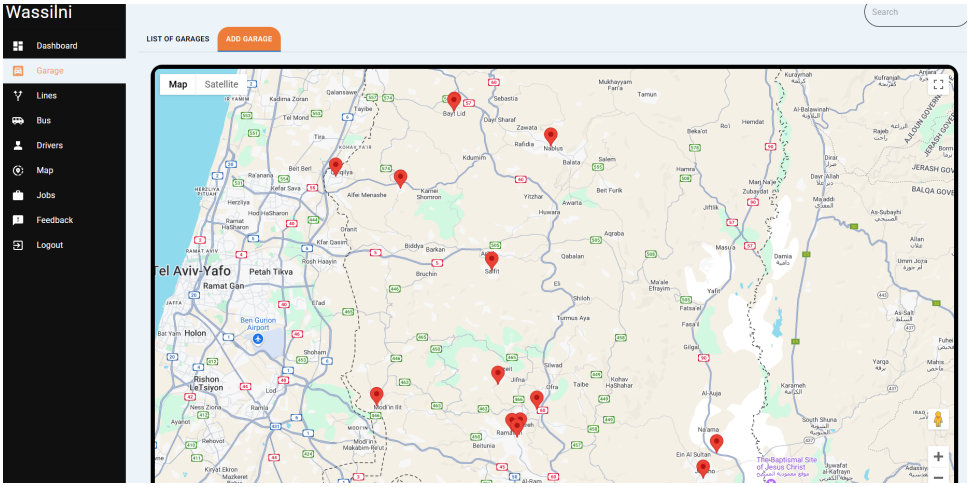


Figure 4.83: Add Garage on Map

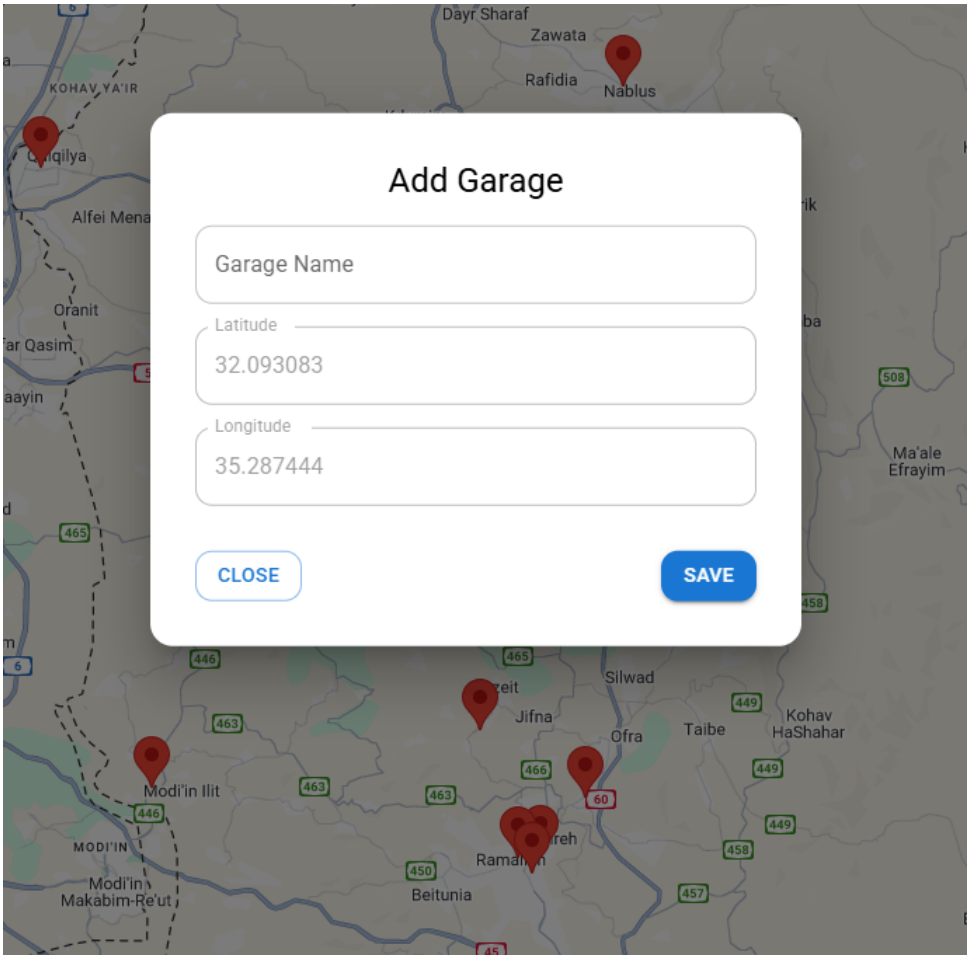


Figure 4.84: Add New Garage

Routes Section

The routes section contains a map and settings for adding routes and stop points. The admin can control the path of each route and view all checkpoints on the map.

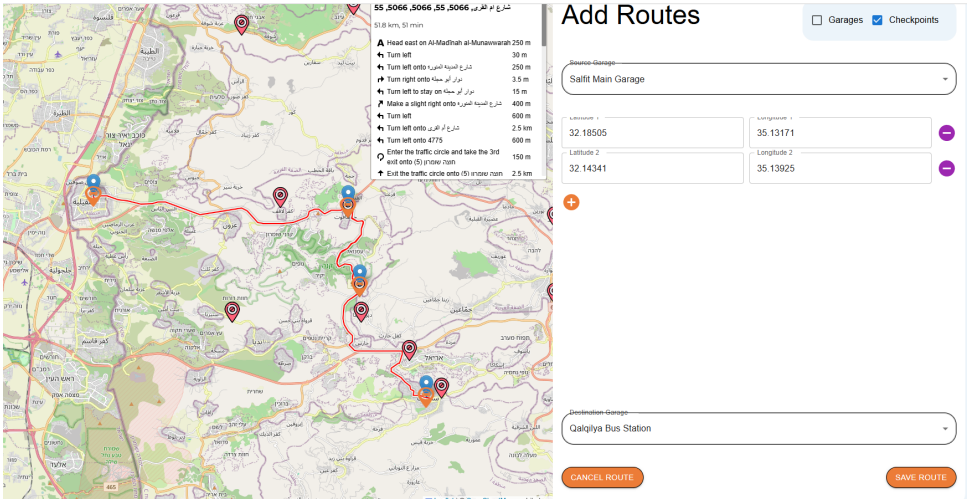


Figure 4.85: Add Route and Stop Points

Buses Section

The buses section displays a list of buses. The admin can add, edit, or delete buses from the system.

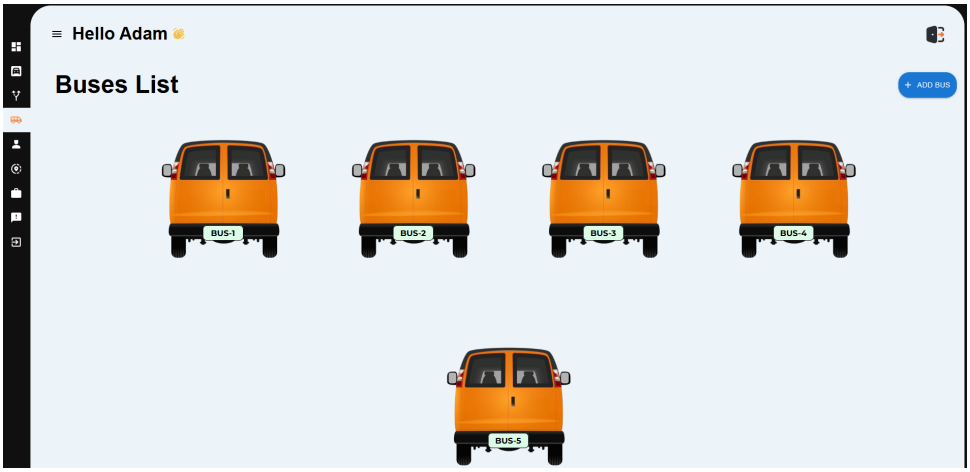


Figure 4.86: Buses List

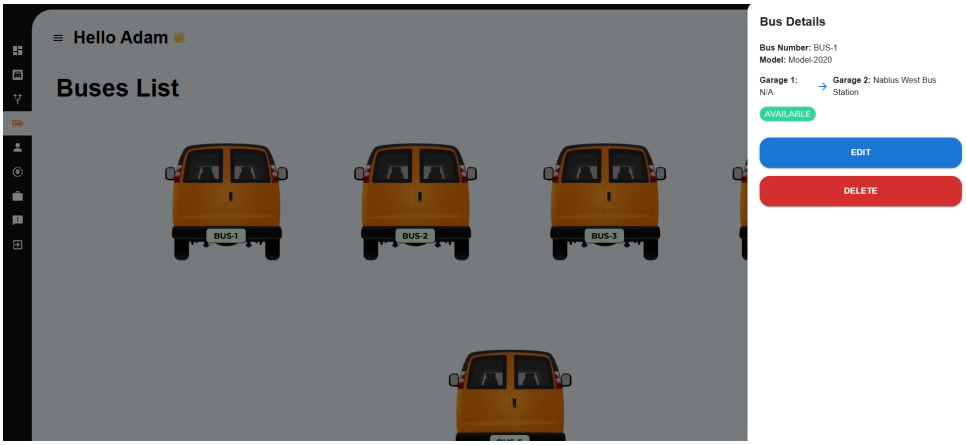


Figure 4.87: Edit Bus Information

Drivers Section

The drivers section displays a list of drivers in the form of driving licenses. The admin can add, edit, or delete drivers, and search for a specific driver's name.

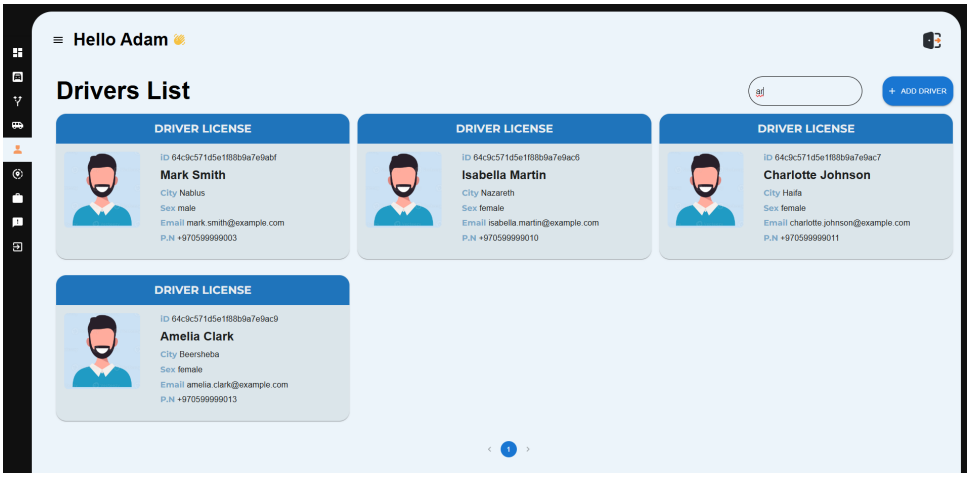


Figure 4.88: Drivers List

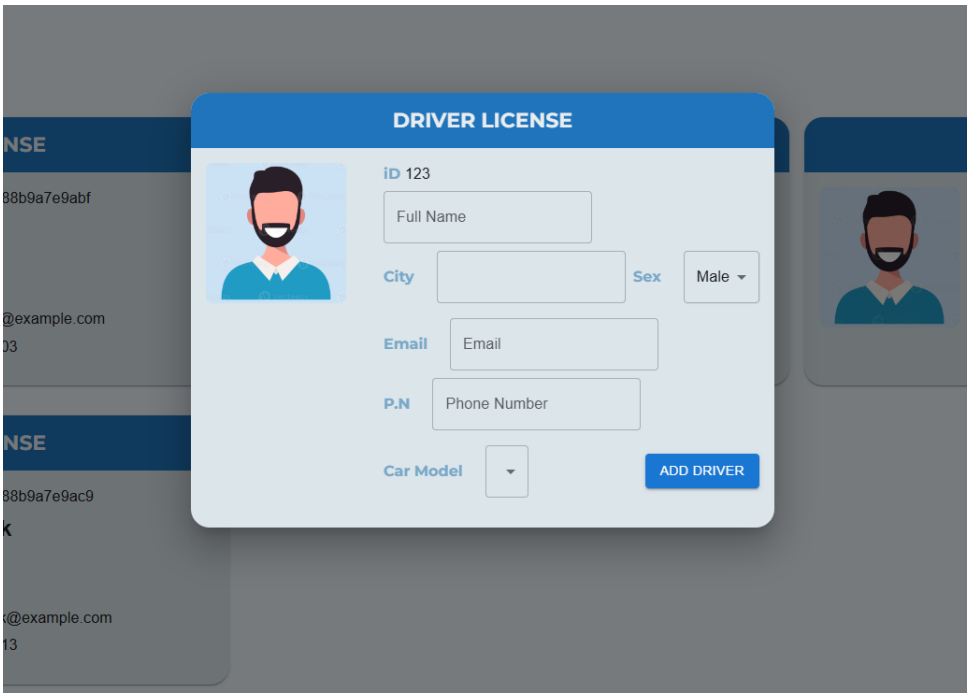


Figure 4.89: Add New Driver

Live Track Section

The live track section displays a map with all the cars operating on their routes. The admin can track the current status of each car.

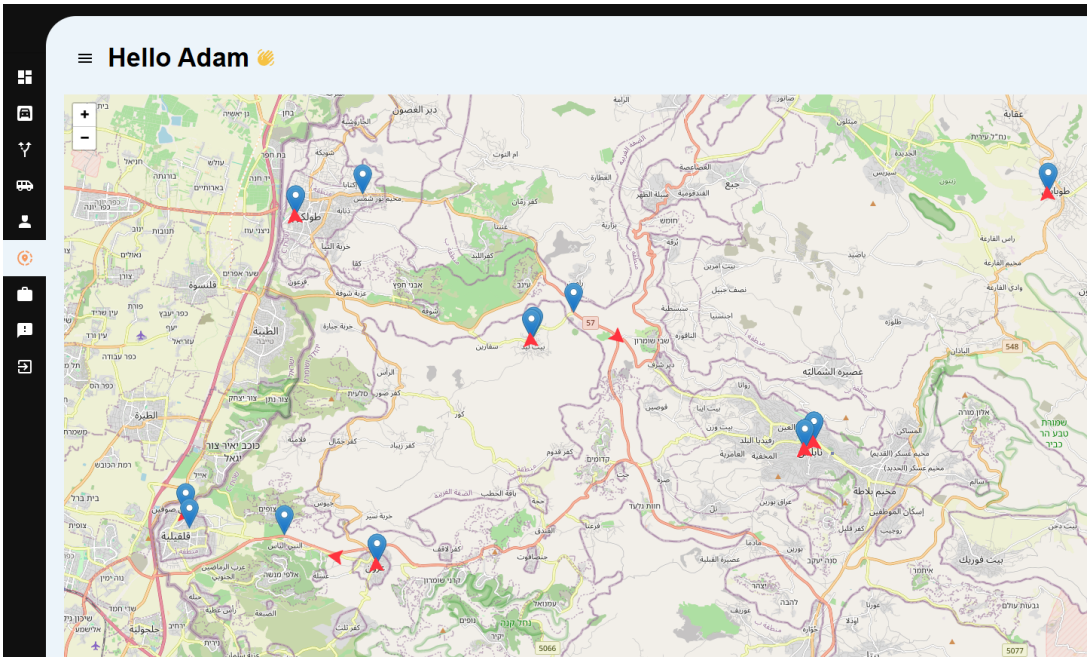


Figure 4.90: Live Car Tracking Map

Work Section

In the work section, the admin can publish specific job listings that will be visible to riders. The admin can also view the applicants' resumes and determine whether they should be accepted, rejected, or invited for an interview. Applicants will receive email notifications with the results.

Figure 4.91: Published Jobs List

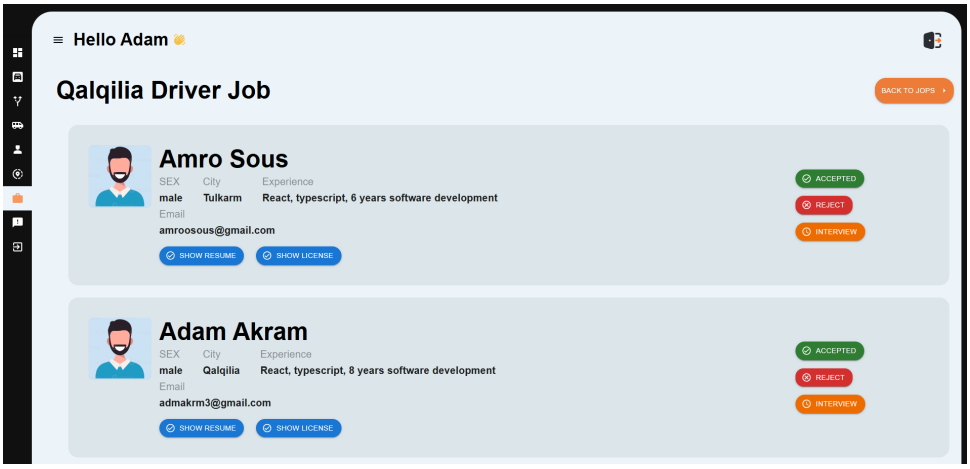


Figure 4.92: Candidates for Job

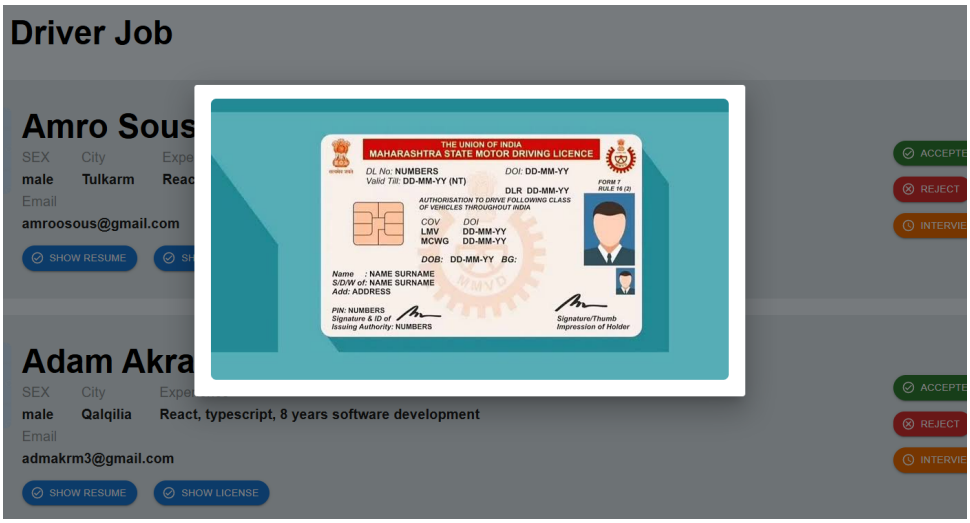


Figure 4.93: Candidate License

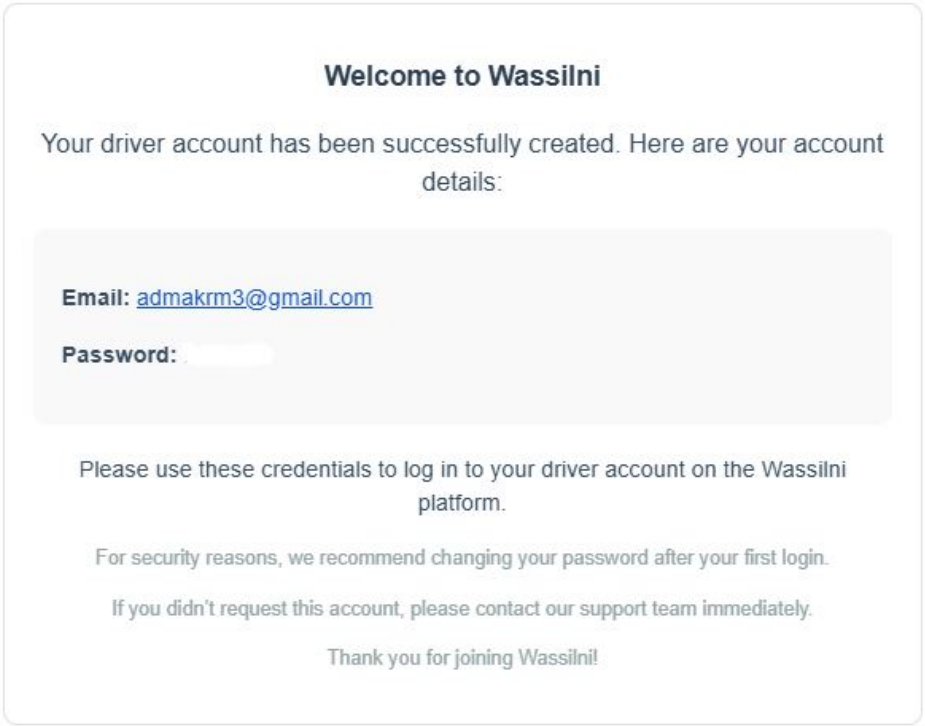


Figure 4.94: Job Acceptance Notification

Feedback Section

The feedback section displays statistics and ratings for each rider, including the feedback provided for their experience.

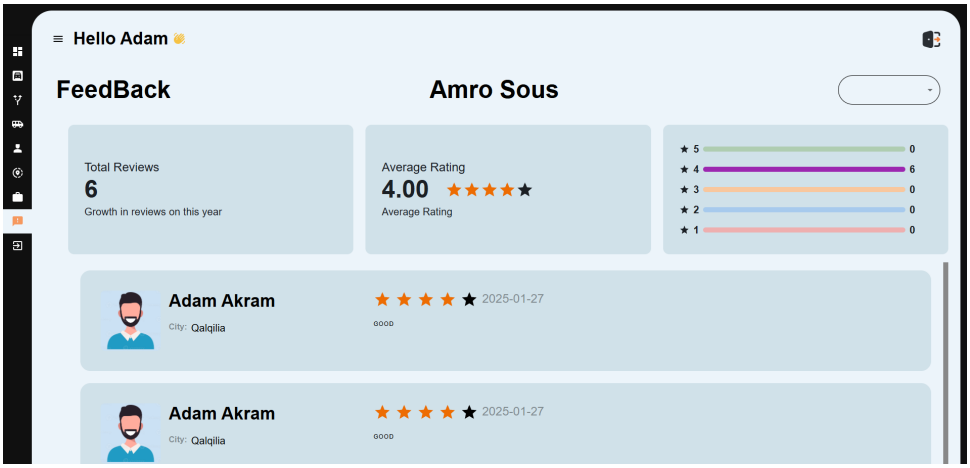


Figure 4.95: Rider Ratings and Feedback

3.2.3 Backend

The back-end system of our public transportation booking application has been designed with scalability, performance, and modularity in mind. Our primary goal was to ensure efficient communication between services, seamless handling of real-time updates, and the ability to process large volumes of data quickly. The architecture employs a microservices approach, where each component is responsible for a specific functionality, ensuring clear separation of concerns and ease of maintenance. All back-end services are containerized using Docker and orchestrated with Docker Compose, enabling consistent deployment across different environments. The use of industry-standard APIs, caching mechanisms, and advanced AI services ensures a robust and responsive user experience. Figure 4.96 shows an overview of the system.

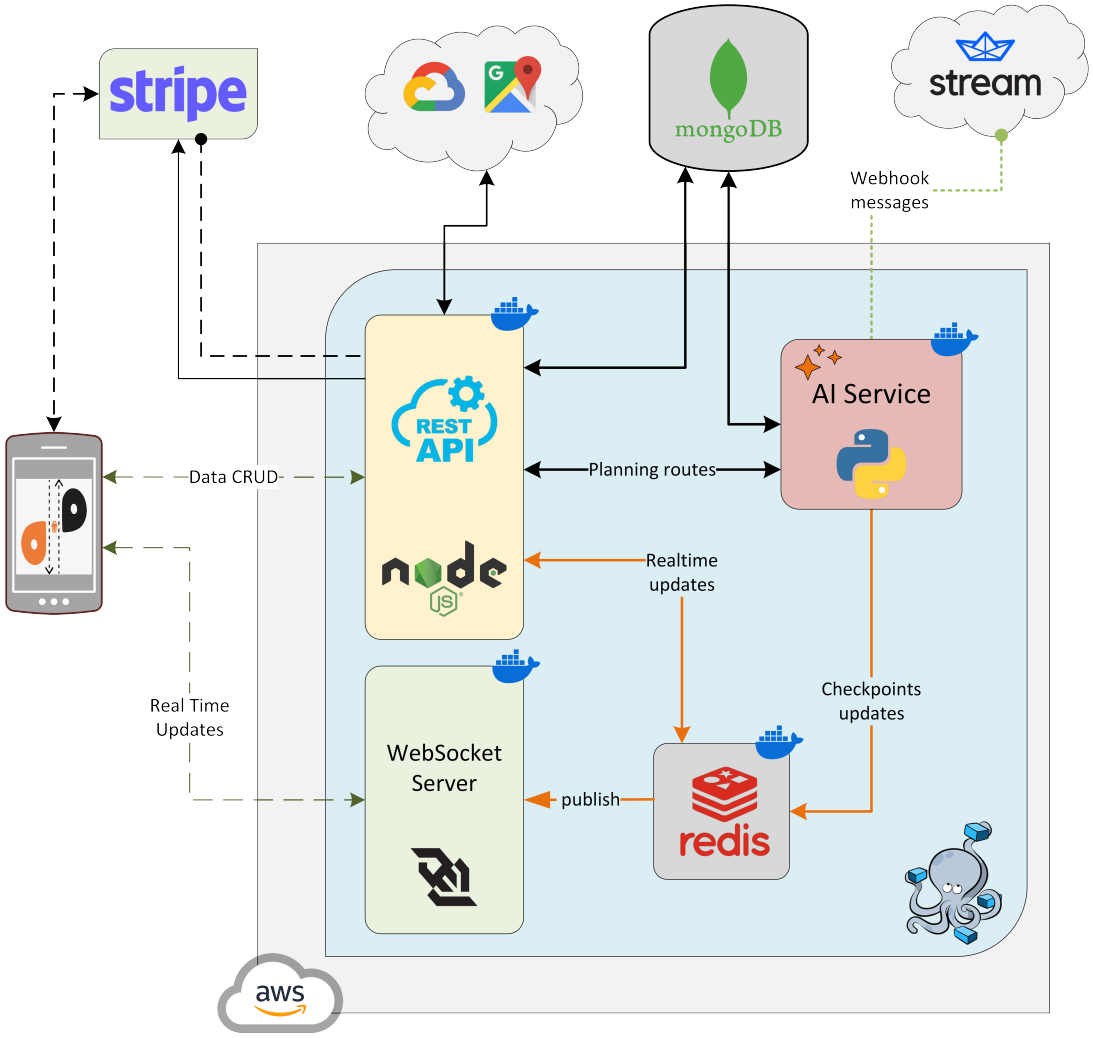


Figure 4.96: System Architecture Diagram

3.2.3.1 RESTful API

The RESTful API serves as the main interface between client applications and the back-end system. Built using the lightweight and flexible **`Express.js`** framework, this service is designed to handle client requests, process them, and interact with other back-end services such as the database, Redis, and external APIs.

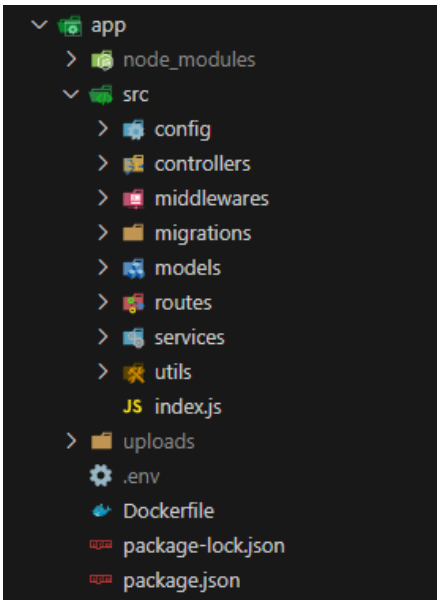


Figure 4.97: REST API file structure

The service ensures secure communication and robust functionality through the integration of multiple packages and techniques. For authentication and authorization, we utilize JSON Web Tokens (JWT) generated using the **`jsonwebtoken`** library. Passwords are securely hashed using **`bcryptjs`** before being stored in the MongoDB database, managed efficiently with **`mongoose`**. Input validation is rigorously handled with **`express-validator`**, reducing the risk of malformed or malicious requests.

Security is further enhanced by employing **`helmet`**, which adds various HTTP headers to protect the app, and **`cors`** to enable cross-origin requests while maintaining control over allowed origins. **`express-rate-limit`** safeguards against denial-of-service attacks by limiting the number of requests a client can make within a given time frame. Error handling and logging are managed through **`express-async-errors`** and **`winston`**, ensuring smooth debugging and consistent tracking of application performance.

For handling file uploads, the service leverages **`multer`**, while Google Drive APIs (accessed through **`googleapis`**) facilitate secure storage of uploaded images and files. Sensitive configuration data such as API keys and database credentials are managed using **`dotenv`**, ensuring they remain secure and environment-specific.

Route planning and geospatial computations are enriched by **`@turf/turf`**, which allows the generation of polygons around routes for checkpoint determination. Development processes are streamlined with **`mongoose-data-seed`**, enabling the easy seeding of data for testing purposes.

The RESTful API service operates as the backbone of the backend system, communicating with other services such as the AI service via HTTP and the WebSocket server for real-time updates. The integration of Stripe APIs facilitates secure and reliable payment processing, while **`stream-chat`** ensures seamless communication between users and the AI service through webhooks.

Together, these technologies and techniques create a resilient, scalable, and efficient service that meets the complex requirements of the application.

3.2.3.2 WebSocket Server Service

The WebSocket server is a dedicated service designed to handle real-time updates and notifications, and is responsible for maintaining long-term connections with client devices, enabling seamless real-time communication. This service handles updates such as bus location tracking, booking notifications, and travel status updates, ensuring that riders and drivers are always informed about critical changes in real time.

The separation of the WebSocket server from the RESTful API allows the backend to handle numerous concurrent connections efficiently. This modular approach ensures real-time updates remain fast and reliable, even during peak usage. Redis' low-latency operations further enhance the scalability of this service.

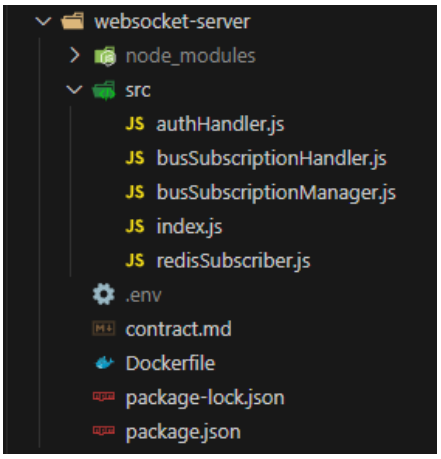


Figure 4.98: WebSocket Server file structure

Integration with Redis

To efficiently manage real-time updates, the WebSocket server utilizes **`Redis`** as an intermediary for data exchange. The server subscribes to specific channels in Redis and processes updates published by the RESTful API or other backend components. By leveraging Redis' pub/sub mechanism, the WebSocket server ensures low-latency communication and reliable delivery of notifications.

Authentication and Authorization

Authentication is handled through JSON Web Tokens (JWT), ensuring that only verified clients can establish a connection with the WebSocket server. Upon connection, the client sends a valid JWT, which the server verifies to confirm the user's identity.

Authorization is enforced using Redis data structures:

- **Riders:** Added to a Redis set `` `bus:${busId}:subscribers` ``, authorizing them to receive notifications related to a specific bus.
- **Drivers:** Associated with a bus using a Redis key-value pair `` `bus:${busId}:driver` ``, ensuring that only the designated driver receives relevant updates.

Notifications and Data Exchange

The WebSocket server exchanges various types of notifications and data with riders and drivers, tailored to their roles in the system. The details of these notifications, including their channels, payloads, and descriptions, are summarized in Table 1.

Notification Type	**Channel**	**Description**
Location Updates	`location-updates`	Sends real-time location updates for buses.
Bus Full	`bus-full`	Notifies when a bus reaches its maximum capacity.
Rider Booking	`rider-subscribe`	Notifies the driver when someone books their bus.
Start Travel	`start-travel`	Notifies riders when the bus starts its journey.
Checkpoints Status	`checkpoints`	Updates the status of checkpoints in real time.

Table 1: Types of Notifications Exchanged via the WebSocket Server

3.2.3.3 AI Service and RAG Application

The AI service, built using Python, plays a key role in the system by automating two essential tasks:

- Efficiently interpreting natural language inputs, such as driver chats, to update checkpoint statuses in real time.
- Optimize route planning for riders by calculating the best routes using graph-based algorithms.

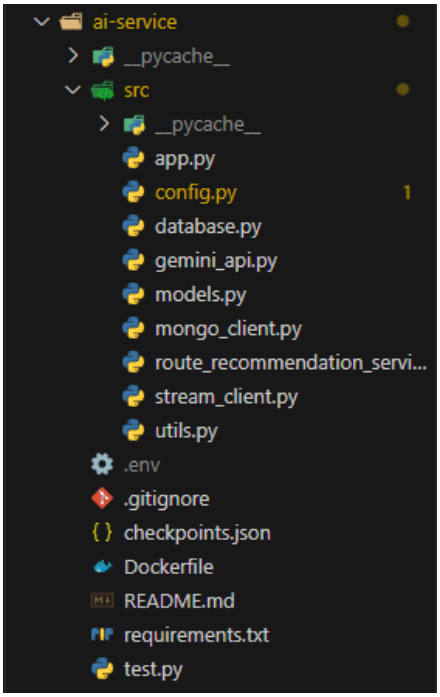


Figure 4.99: AI service file structure

Updating Checkpoint Statuses

The main responsibility of the AI service is to update the status of checkpoints based on conversations between drivers.

The initial approach to updating checkpoint statuses involved fine-tuning a model using Google Cloud's Vertex AI. Approximately 5,000 documents were generated with the help of ChatGPT to create variations in spellings and sentences in Arabic related to checkpoint statuses. While the model achieved reasonable performance, several challenges emerged:

- Non-standardized outputs, making it difficult to map checkpoint names or IDs to the database.
- Inability to handle complex or indirect sentences effectively.

USER	حاجز دير شرف مسكر.
AI	{ "name": "حاجز دير شرف", "status": "closed"}
USER	دير شرف مأزم تعال من صرة
AI	{ "name": "دير شرف", "status": "traffic"}

Figure 4.100: Fine Tuning Results

To address these limitations, the system transitioned to a Retrieval-Augmented Generation (RAG) approach illustrated in Figure 4.101, which combines:

- Knowledge Retrieval: Using **`ChromaDB`** to fetch relevant checkpoint information based on vector similarity.
- Contextual Understanding: Employing an LLM to process the retrieved information along with the user message to generate structured, JSON-formatted outputs.

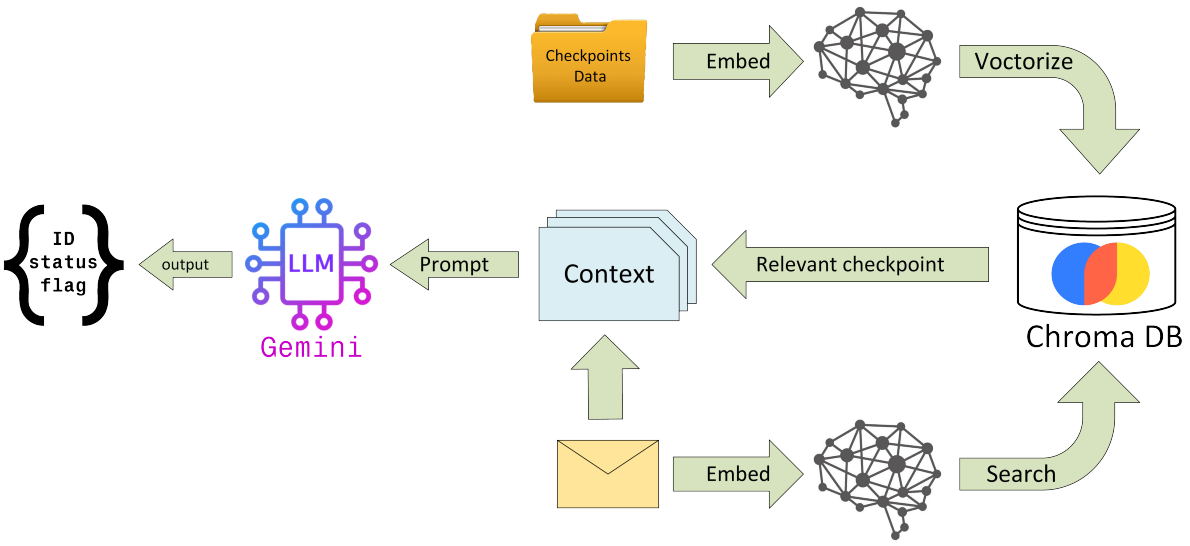


Figure 4.101: RAG application components

⋆ **Data Storage:** The system uses **`ChromaDB`** to store vectorized embeddings of checkpoint data, with the help of **`'models/text-embedding-004'`** embedding model from Gemini API. Each checkpoint record includes attributes such as:

- `id`: Unique identifier of the checkpoint.
- `name`: Name of the checkpoint.
- `spellings` (`[ar, en]`): Variations of the checkpoint name in Arabic and English.
- `description` (`[ar, en]`): Descriptions of the checkpoint in both languages.
- `location`: Geographic coordinates of the checkpoint.

⋆ **Query Processing:** The system uses **`'models/gemini-1.5-pro-latest'`**, a Large Language Model (LLM), to interpret messages with context about checkpoints. The query provided to the LLM includes the checkpoint information and a user message. The LLM is tasked with returning a JSON response containing:

- Whether the checkpoint status can be determined (`found: True/False`).
- The checkpoint ID and name.
- The checkpoint status (`"closed", "open", "traffic"`).

⋆ **Integration:** Updates from the LLM are pushed to Redis for real-time notification and stored in MongoDB for persistence.

Route Planning for Riders

The AI service also supports route planning by generating optimized travel plans for riders. This process involves:

- Graph Representation: A directed graph of the transportation network is built using **`networkx`**.
- Pathfinding Algorithm: The system employs the **A*** algorithm to find the optimal route, using a specified `search_metric` as the cost function.
- Output: The service returns an array of recommended plans, providing riders with efficient travel options.

Challenges in the RAG System The RAG approach mitigated many issues from the fine-tuning model, including standardizing outputs and improving accuracy for complex messages. However, challenges in processing Arabic language inputs persist, leading to a focus on English for the current implementation.

3.2.3.4 Deployment

The deployment of the backend API is automated using **Pulumi** with **JavaScript** to provision infrastructure on **AWS**. The deployment process ensures a streamlined and reliable approach to managing services, logging, and database connectivity.

Deployment Process

The backend API is deployed to **AWS ECS (Elastic Container Service)**. The deployment workflow is triggered automatically using **GitHub Actions** whenever new code is pushed to the `deploy` branch. The automation includes:

- Executing the Pulumi project code to deploy the API.
- Managing infrastructure resources such as ECS services and networking configurations and environment variables.
- Returning the AWS service URL after successful deployment.

Figure 4.102 shows a screenshot of the GitHub Actions workflow successfully triggering the deployment process.

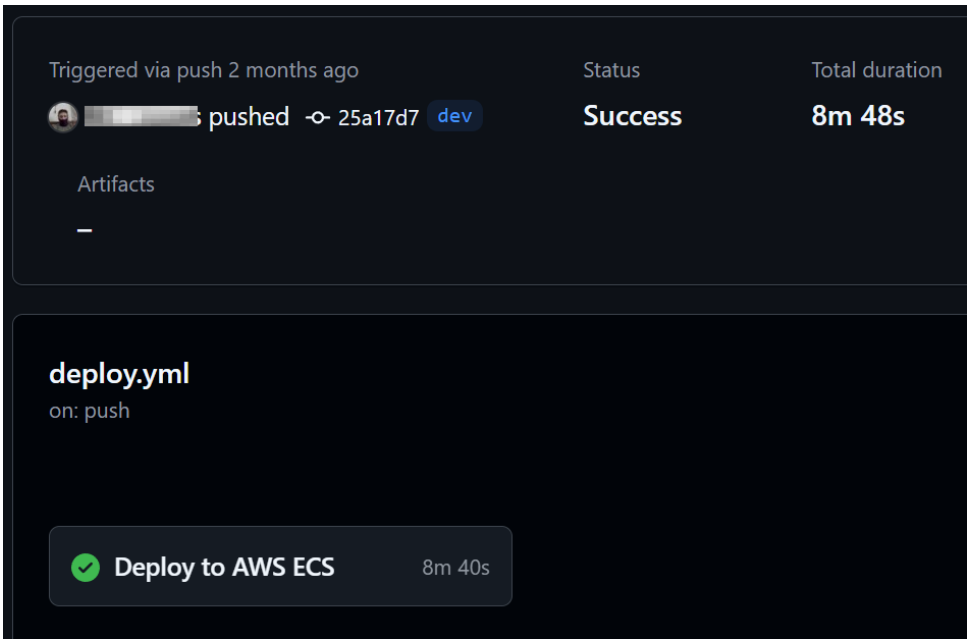


Figure 4.102: GitHub Actions successfully triggering deployment.

Infrastructure and Services

The infrastructure provisioned by Pulumi includes:

- ECS Service: Hosts and manages the backend API containers.
- AWS CloudWatch: Used for logging and monitoring API activity.
- VPC Connectivity: Establishes a secure connection with MongoDB.

The Pulumi dashboard provides an overview of the deployed services and the AWS URL of the API. Figure 4.103 presents a screenshot of the Pulumi dashboard displaying the created services.

```
     Type                                  Name                          Status
 +   pulumi:pulumi:Stack                   wassilni-container-dev        created (487s)
 +   ├─ aws:ecs:Cluster                    cluster                       created (12s)
 +   ├─ aws:cloudwatch:LogGroup            app-log-group                 created (4s)
 +   ├─ awsx:ecs:FargateService            service                       created (0.00s)
 +   │  ├─ aws:ec2:SecurityGroup           service-sg                    created (8s)
 +   │  ├─ awsx:ecs:FargateTaskDefinition  service                       created (0.00s)
 +   │  │  ├─ aws:iam:RolePolicyAttachment service-execution-9a42f520     created (0.00s)
 +   │  │  ├─ aws:ecs:TaskDefinition       service                       created (0.00s)
 +   │  │  ├─ aws:cloudwatch:LogGroup      service                       created (4s)
 +   │  │  ├─ aws:iam:Role                 service-task                  created (4s)
 +   │  │  └─ aws:iam:Role                 service-execution             created (4s)
 +   │  └─ aws:ecs:Service                 service                       created (59s)
 +   ├─ awsx:ecr:Repository                repo                          created (8s)
 +   │  ├─ aws:ecr:Repository              repo                          created (4s)
 +   │  └─ aws:ecr:LifecyclePolicy         repo                          created (0.00s)
 +   ├─ awsx:lb:ApplicationLoadBalancer    loadbalancer                  created (4s)
 +   │  ├─ aws:ec2:SecurityGroup           loadbalancer                  created (4s)
 +   │  ├─ aws:lb:Listener                 loadbalancer-0                created (0.00s)
 +   │  ├─ aws:lb:LoadBalancer             loadbalancer                  created (392s)
 +   │  └─ aws:lb:TargetGroup              loadbalancer                  created (4s)
 +   ├─ awsx:ecr:Image                     image                         created (0.00s)
 +   │  └─ docker:index:Image              12fda807-container            created (88s)
 +   ├─ aws:iam:Role                       taskRole                      created (4s)
 +   └─ aws:iam:RolePolicyAttachment       logPolicyAttachment           created (4s)

Outputs:
    url: "http://loadbalancer-be8d69b-1892533454.us-west-2.elb.amazonaws.com"

Resources:
    + 24 created
```

Figure 4.103: Pulumi dashboard showing deployed services and API URL.

3.2.4 Database

The project utilizes **MongoDB** as the primary database, managed through **Mongoose** as the ORM (Object-Relational Mapping) layer. MongoDB's flexibility and scalability make it well-suited for handling the dynamic and geospatial data requirements of our system.

3.2.4.1 Why MongoDB?

Given the nature of our project, MongoDB was chosen over relational databases due to the following advantages:

- **Flexible Schema**: Our system deals with dynamic data structures, such as ride requests, user interactions, and checkpoint updates, which benefit from MongoDB's schema-less design.
- **Geospatial Indexing**: MongoDB provides built-in support for geospatial queries, allowing efficient retrieval of the nearest entities, such as the closest buses or checkpoints, which is essential for real-time navigation.
- **Scalability**: As a NoSQL database, MongoDB scales horizontally, making it ideal for handling large datasets and high-traffic scenarios in a distributed system.
- **Efficient Real-time Updates**: The project's WebSocket server requires a fast and efficient way to store and retrieve real-time updates, which MongoDB's document-based model handles effectively.

3.2.4.2 Database Structure

The database consists of multiple collections representing different entities, such as `riders`, `drivers`, `buses`, and `checkpoints`. Figure 4.104 provides an overview of the database schema.

3.2.4.3 ORM and Data Seeding

To interact with MongoDB, we use **Mongoose**, which provides a structured way to define schemas and manage relationships within collections. Additionally, for the development process, we use **mongoose-data-seed** to pre-populate the database with test data, ensuring smooth testing and simulation of real-world scenarios.

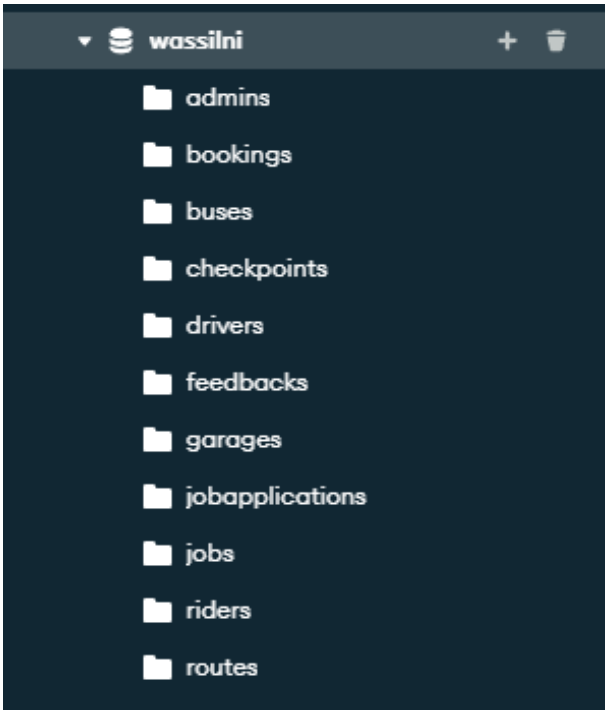


Figure 4.104: Database collections.

4 Conclusion

4.1 Summary

The Wassilni project represents a significant step forward in addressing the challenges of public transportation systems. By leveraging modern technologies such as GPS, AI, and mobile applications, the project has developed a comprehensive solution that enhances the commuting experience for both passengers and drivers. Key features of the system include real-time tracking, dynamic route planning, secure digital payments, and efficient queue management. These features not only improve operational efficiency but also provide users with a more convenient and reliable transportation option.

The project successfully implemented a mobile application for riders and drivers, a web-based admin dashboard, and a robust backend system. The integration of geospatial technologies, AI-based checkpoint updates, and real-time notifications ensures that users have access to the most up-to-date information. Additionally, the use of cloud-based infrastructure and containerization technologies, such as Docker and AWS ECS, ensures scalability and reliability.

4.2 Future Work

While the Wassilni project has achieved its primary objectives, there are several areas for future improvement and expansion. These include:

4.3 Enhancements to AI and Machine Learning

- **Improved Checkpoint Updates:** Further refine the AI-based checkpoint update system to handle more complex and nuanced inputs, particularly in Arabic.
- **Predictive Analytics:** Incorporate predictive analytics to forecast traffic conditions and optimize routes in real time.

5 References

References

[1] Pixegami, “Rag + langchain python project,” 2024, accessed: Jan 25, 2025. [Online]. Available: https://www.youtube.com/watch?v=tcqEUSNCn8I

[2] G. Cloud, “Google cloud console,” 2025, accessed: Jan 25, 2025. [Online]. Available: https://console.cloud.google.com/