WePool

The prefect balance between Privacy & convenience

Created by: Aviel Smolanski & Elior Uzan

June 2025

Institution: Afeka College of Engineering

Table of contents

1. Abstract 3
2. Introduction ..... 4 - 5
3. Background and Related Work ..... 6 – 7
4. System Design, Methods & Approach ..... 8 – 11
   - 4.1 Functional Requirements ..... 8 - 9
   - 4.2 Non – Functional Requirement 9
   - 4.3 System Architecture ..... 10 – 13
     - 4.3.1 System's main algorithm 11- 12
5. Implementation ..... 14 – 16
6. Experiments & Results ..... 17 – 18
7. Discussion ..... 19 – 20
8. Conclusion & Future Work ..... 21 – 22
9. User Guide & App Walkthrough ..... 23 – 29
   - 9.1 App Screenshots ..... 24 - 29
   - 9.2 Link to app preview video 29
10. References 30

Abstract

In modern urban environments, traffic congestion, environmental concerns, and the rising cost of transportation have intensified the need for efficient carpooling solutions. This project presents WePool, a smart, organization-based carpooling system designed to streamline daily commutes for employees within the same company. The core motivation behind WePool is to reduce redundant vehicle usage by intelligently matching drivers and passengers according to shared destinations, time preferences, and organizational affiliation.

The system addresses several key challenges in the carpooling domain: ensuring security and trust by limiting matches to verified company users; optimizing routes using the Google Maps API; and managing dynamic ride updates through real-time Firebase integration. The platform supports distinct user roles (driver, passenger, HR manager), and allows for flexible ride planning in both workbound and homebound directions.

Our implementation includes live navigation with automatic waypoint tracking, personalized notifications via Firebase Cloud Messaging, and advanced filtering and sorting capabilities for active and historical rides. Testing on real use cases demonstrated reliable performance in ride matching, low latency in updates, and high user engagement through intuitive UI/UX.

The project concludes that integrating cloud technologies with real-time location services can offer scalable, secure, and user-friendly carpooling solutions tailored for enterprise environments. WePool demonstrates how software can meaningfully reduce traffic volume and commuting stress while fostering collaboration within the workplace.

Introduction

Motivation: Why is the problem important?

In today's urbanized and congested environments, the daily commute has become a major source of frustration for millions of employees. Excessive traffic, rising fuel costs, air pollution, and limited parking availability all contribute to an inefficient and stressful transportation system. While carpooling has long been recognized as a practical solution to these issues, its adoption remains limited due to the lack of flexible, reliable, and secure platforms that suit the specific needs of organized groups — such as company employees. Traditional ridesharing apps are primarily designed for public or open-use cases, lacking the trust, coordination, and simplicity required for intra-organizational carpooling. This gap presents a unique opportunity to build a customized system that not only improves employee mobility, but also reduces environmental impact and operational costs for organizations.

Problem Statement: What is the problem we are addressing?

Despite the known benefits of carpooling, many companies struggle to implement it effectively. Employees often find it difficult to coordinate ride schedules, match with appropriate colleagues, and trust unknown drivers or passengers. Existing solutions do not offer a closed, company-specific environment, nor do they support flexible planning for both morning and evening commutes. Furthermore, traditional platforms rarely address critical features such as multi-stop navigation, real-time updates, or approval workflows — all of which are essential in a professional context. As a result, employees either rely on inefficient personal transport or avoid carpooling altogether.

Project Goals and Objectives

The main goal of this project is to develop WePool — a secure and intelligent carpooling system tailored for employees of the same company. Its specific objectives include:

- Creating a robust user authentication system with role-based access (Driver, Passenger, HR Manager).
- Implementing reliable ride planning for both workbound and homebound directions.
- Matching passengers and drivers based on route, schedule, and company affiliation.

- Supporting ride approval flows, live updates, and dynamic route recalculations.
- Providing live navigation with automatic tracking of pickups and drop-offs.
- Enabling company HR managers to monitor employee participation and manage users efficiently.
- Sending targeted real-time notifications to enhance coordination and engagement.

Brief Overview of Our Approach and Main Contributions

Our approach combines cloud technologies, real-time database, geolocation services, and a user-friendly mobile interface to deliver an end-to-end carpooling solution. We built the system using Firebase for authentication, Firestore for data storage, and the Google Maps API for advanced route calculations and navigation. Drivers can create and manage rides, while passengers can search and request to join based on timing, direction, and proximity. All route planning supports detour analysis and multi-waypoint navigation. Once a ride starts, WePool launches Google Maps with all planned stops and tracks progress automatically. In addition, the system sends push notifications for approvals, arrivals, and schedule updates.

Our main contributions include:

- A private, secure carpooling network for company employees only.
- A flexible dual-direction ride model (to work and back home).
- A fully functional mobile interface with integrated navigation.
- An HR management dashboard to handle employee oversight.
- Real-time syncing and intelligent ride coordination between multiple users.

Together, these components create a system that is both practical and scalable, designed to address real-world needs in corporate commuting.

Background and Related Work

Related Work, Tools, Algorithms, and Datasets

The growing interest in smart mobility and carpooling systems has led to the development of various ridesharing platforms such as UberPool, BlaBlaCar, and Waze Carpool. These platforms aim to connect drivers and passengers based on route and availability. However, they were built for the general public and often lack the specificity and control required for intra-organizational use.

Our project, WePool, focuses on solving these challenges within a closed company environment, ensuring that all users belong to the same verified organization. To achieve this, we relied on several modern tools and services:

- Firebase: for authentication (including role management), Firestore for real-time database syncing, and Cloud Functions for backend logic and notifications [2].
- Google Maps API: for route calculation, geocoding, and navigation [1].
- Jetpack Compose & Android Studio: for building a modern, responsive mobile UI.
- Firebase Cloud Messaging (FCM): for real-time notifications when rides are updated, started, or when users are approved/rejected [2].
- Ramer–Douglas–Peucker (RDP) algorithm: to simplify polylines (routes), which significantly reduces the number of calls made to the Google Maps Directions API [3][4].

Comparison of Existing Approaches

Commercial ride-sharing platforms primarily match users from the public domain and rely heavily on on-demand service models. While services like Waze Carpool offered work-related ride matching, they do not restrict users to a specific company, nor do they offer features like role-based user flows (driver/passenger/HR), approval of ride requests, or automatic waypoint navigation.

In contrast, WePool introduces:

- Closed user authentication based on company (ensuring safety and familiarity).
- Ride creation for fixed or flexible routes in both directions: to work and back home.

- Approval-based ride requests, giving drivers full control over who joins the ride.
- Automatic Google Maps navigation, triggered from the app, including multi-stop routing.
- Simplified polylines using the RDP algorithm to reduce backend API load and latency [3].

Gaps Addressed by WePool

Existing solutions fall short in several key areas that are critical in workplace carpooling:

- Lack of company exclusivity: Most apps do not verify users based on employment or organizational affiliation.
- No approval mechanism: Passengers can often join a ride without driver consent.
- Poor integration with daily routines: Existing platforms do not support two-directional planning (to work and home) or schedule-based planning.
- Cost inefficiency: High-frequency route recalculations via the Google Maps API can become expensive. WePool reduces the frequency of these calls using the Ramer–Douglas–Peucker algorithm [3][4] and offline algorithm developed by us, which simplifies route paths and avoids using this API in infeasible cases altogether.

References (IEEE Format)

[1] Google Developers, "Directions API Overview," *Google Maps Platform*. [Online]. Available: https://developers.google.com/maps/documentation/directions/ .

[2] Firebase, "Firebase Documentation." [Online]. Available: https://firebase.google.com/docs .

[3] U. Ramer, "An iterative procedure for the polygonal approximation of plane curves," *Comput. Graph. Image Process.*, vol. 1, no. 3, pp. 244–256, 1972.

[4] D. H. Douglas and T. K. Peucker, "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature," *Cartographica: The International Journal for Geographic Information and Geovisualization*, vol. 10, no. 2, pp. 112–122, 1973.

System Design, Methods & Approach

Functional Requirements

The WePool system is designed to manage intra-organizational carpooling using a role-based model. Key functions include:

- User Authentication & Role Management
    - Users log in using organization-specific credentials. Roles include: HR Manager, Driver, and Passenger.
    - Banned users are denied access (through de-activation).
- Ride Creation (Driver)
    - Drivers can create rides by entering destination, direction (to work / home), start or arrival time, seat availability, and a maximum allowed detour.
- Ride Booking (Passenger)
    - Passengers can search and book available rides according to time, location, and direction.
    - The system validates ride matches based on deviation threshold and time alignment.
- Ride Matching Logic
    - For workbound trips: maximum 30-minute difference between driver's and passenger's arrival time (we only show rides that arrive earlier than the passenger's desired arrival time) , if they meet the driver's max deviation threshold.
    - For homebound trips: maximum 15-minute difference from driver's departure time (we only show rides that depart later than the passenger's desired departure time) , if they meet the driver's max deviation threshold.
- Notifications
    - Passengers receive alerts when a driver approves or starts a ride.
    - Drivers are notified of new ride requests.
    - Notifications include updates from Google Maps API.

- History & Policy Enforcement
    - Users can view their ride history.
    - Canceling rides or passengers too close to the ride departure time is not allowed , and the figures are decided by the organization itself (we set the default for workbound rides at 60 minutes and for homebound rides at 10 minutes). In the case of a user violating these rules (especially not showing up to a planned ride) an inquiry by the company's HR manager might ensue following which his account could possibly be deactivated.

Non-Functional Requirements

- Performance:
  The system must support up to 10,000 users with a ride matching response time under 2 seconds.
- Availability:
  Must maintain 99.9% uptime, excluding scheduled maintenance.
- Scalability:
  The backend must support onboarding new companies and scale internally for growing user bases.
- Usability:
  The Android app should be clean, responsive, and intuitive, using modern mobile UX principles.
- Integration:
  The app integrates with Google Maps APIs for route validation and optimization.
- Security:
  All communication is encrypted (HTTPS), and access is managed using role-based access control (RBAC).
- Portability:
  The system supports Android 8.0 and above, aligning with the minimum version that fully supports Jetpack Compose and Kotlin features.

System Architecture

WePool follows a client-cloud model using Firebase as its backend infrastructure. Each system function is modular and communicates through real-time cloud services.

Mobile App (Client Side):

Acts as the primary interface for all users (drivers, passengers, HR managers, and admins).

- Built with Kotlin using Jetpack Compose in Android Studio.
- Sends requests and receives updates via Firebase services.

Firebase Cloud Backend:

Firebase replaces the need for a traditional server by offering cloud-based services:

- Firebase Authentication handles secure login and role-based access.
- Firestore (a NoSQL cloud database) stores all app data, including users, rides, companies, and ride requests.
- Firebase Cloud Functions process background logic like ride updates, detour checks, and notification triggers.
- Firebase Cloud Messaging (FCM) sends real-time push notifications about ride approvals, changes, or live updates.

Ride Matching Service:

- Uses the Google Maps API to calculate optimal routes and evaluate detours.
- Implements a matching algorithm that ensures passengers fit within driver-defined limits (e.g., a 10-minute max detour).
- Optimizations like the Ramer–Douglas–Peucker algorithm reduce API usage by simplifying route paths.

Ride Matching Algorithm (System's main Algorithm):

The route matching algorithm operates in four sequential stages, combining fast offline heuristics with precise online validation to determine whether a pickup point can be added to an existing ride without exceeding the allowed detour time. The process is designed to be both scalable and accurate, optimizing for performance, quota efficiency, and user experience.

Only if the candidate pickup point successfully passes all three offline stages - route simplification, spatial proximity filtering, and offline detour estimation - the algorithm proceeds to the final step, which involves a real-time Google Maps API call to validate the detour using live routing data.

Step 1: Route Simplification

Purpose: Optimize performance by reducing the number of coordinates in the route while preserving its geometric shape.

- The encoded route polyline is decoded and simplified using a global tolerance parameter (default: 50 meters).
- The simplification is performed using the **Douglas-Peucker algorithm**, which recursively removes intermediate points that deviate minimally from the original shape.
- This yields a significantly shorter and cleaner path for further analysis.

Justification: Reducing route complexity at this stage ensures scalability and responsiveness in mobile environments, especially when handling numerous matching operations.

Step 2: Spatial Proximity Filtering

Purpose: Filter the simplified route to include only points that are spatially close to the pickup location.

- Each coordinate from the simplified route is checked to see whether its one-way great-circle (airline) distance to the pickup location is less than the configured threshold (default: 2000 meters).
- Only those within range are retained as initial candidate points for potential detour evaluation.

Justification: This spatial filter narrows down the dataset early, enabling the algorithm to focus only on nearby points, thus avoiding unnecessary processing in later stages.

Step 3: Offline Detour Time Estimation

Purpose: Evaluate whether each candidate point is temporally feasible as a pickup detour, using offline heuristics.

- For each nearby point, the round-trip distance to the pickup location is computed and divided by the average speed (default: 600 m/min) to estimate detour duration.
- If the combined detour time (current + estimated) remains within the allowed threshold, the point is marked as a valid candidate for full evaluation.

Justification: This step efficiently filters out infeasible points using a lightweight estimation formula, ensuring only promising candidates proceed to the costly online stage.

Step 4: Online Route Evaluation (Google Maps API)

Purpose: Accurately assess the real-world impact of the detour using road network data and traffic conditions.

- For each remaining candidate, a new route is calculated using the Google Directions API, incorporating the original start, destination, existing pickup stops, and the new candidate stop.
- The total route duration is recomputed, and the added detour time is calculated.
- If the total detour time remains within the allowed threshold, the pickup is approved, and precise pickup/drop-off times are updated.

Justification: This final validation stage guarantees route realism, integrity, and compliance with user time constraints, leveraging up-to-date traffic-aware data.

Database (Firestore):

A flexible, scalable real-time NoSQL database used to:

- Store user data, rides, ride requests, and company settings.
- Sync instantly with the mobile app for real-time updates.

HR Manager & Admin Interfaces:

Accessible via the app, these dashboards allow:

- HR Managers: to view and manage employees, ride usage, and policies.
- Admins: to manage companies and access cross-organization analytics.

Architecture Flow Summary:

- Users interact with the mobile app.
- The app communicates directly with Firebase to:
  - Authenticate users and manage roles.
  - Store and retrieve data from Firestore.
  - Validate locations and routes using the Google Maps API.
  - Send and receive notifications via FCM.
- All components operate without a traditional server, using Firebase's cloud services for full backend functionality.

Technology stack

Component	Technology Used
Mobile Development	Kotlin, Jetpack Compose
IDE	Android Studio
Backend	Firebase Cloud Functions (Kotlin/Java)
Database	Firebase Firestore (NoSQL)
Authentication	Firebase Authentication
API's	Google Maps API, Firebase Cloud Messaging (FCM)
Navigation & Routing	Google Maps (via Directions API)
Location Services	Google Play Services
Analytics	Firestore usage stats, custom logs
Hosting	Firebase Backend

Implementation

Practical Details of How the System Was Built

The system was developed as a native Android application using Kotlin and Jetpack Compose, targeting Android 8.0 and above. Development was done in Android Studio, with Firebase serving as the cloud-based backend.

The system architecture follows a modular client-cloud model, where the mobile app communicates directly with Firebase services for authentication, database operations, notification handling, and business logic.

Real-time ride matching and routing logic are powered by the Google Maps Directions API, accessed from within the app and Firebase Cloud Functions. The application adheres to MVC architecture for maintainability and modularity.

Key Modules, Data Structures, and Components

🧩 Modules

- Authentication Module
  - Handles secure login using Firebase Authentication.
  - Enforces role-based access for Admin, HR Manager, Driver, and Passenger.
- Ride Management Module
  - Allows drivers to create, view, and manage rides.
  - Passengers can search for and request to join rides.
  - Matching is handled based on time alignment and detour feasibility.
- Notification Module
  - Sends push notifications using Firebase Cloud Messaging (FCM) to notify users of ride requests, approvals, updates, and navigation status.
- Ride Matching & Routing Module
  - Integrates with Google Maps Directions API to calculate and evaluate detours.

    - Uses the Ramer–Douglas–Peucker algorithm to reduce route complexity and minimize API costs.
- Admin & HR Interfaces
    - Admins manage company registration, users, and system-wide settings.
    - HR managers can control company-level settings, view user history, and enforce policy.

🗂️ Key Data Structures (Firestore Collections)

- Users:
    - Fields: uid, active, banned, companyCode, Email, favoriteLocations, fcmToken, lastLoginTimeStamp, name, phoneNumber and roles.
- Rides:
    - Fields: active, arrivalTime, availableSeats, companyCode, currentDetourMinutes, date, departureTime, destination, direction, driverId, encocdedPolyline, maxDetourMinutes, notes, occupiedSeats, originalRoute, passengers, pickupStops, rideId and startLocation.
- RideRequests:
    - Fields: detourEvaluationResult, notes, passengerId, pickupLocation, requestId, rideId and status.
- Companies:
    - Fields: active, companyCode, companyId, companyName, createdAt, employees, hrManagerUid, location, logoUrl and updatedAt.
- Notifications:
    - Triggered via backend events (ride approved, started, canceled, etc.)

Interfaces

- User Interface (Jetpack Compose)
    - Fully responsive UI for Android, with clear flows for different roles.

    - Includes dynamic forms, filters, and Google Maps components.
- Driver Dashboard
    - Offers ride creation and viewing requests from other passengers and active rides.
- Passenger Dashboard
    - Offers ride join and viewing all ride requests and active rides.
- HR Manager Dashboard
    - Custom interface for tracking employee rides and handling moderation actions (e.g., deactivation of specific user or company).

Challenges Encountered and How They Were Solved

- Efficient Ride Matching
    - Problem: Google Maps API limits and cost constraints made frequent route recalculations expensive.
    - Solution: Integrated the Ramer–Douglas–Peucker algorithm to simplify route polylines, reducing unnecessary API calls.
- Dynamic Navigation Between Multiple Stops
    - Problem: Managing navigation with multiple pickup/drop-off points and keeping passengers updated was complex.
    - Solution: Opened Google Maps with pre-defined waypoints for drivers and used background services to monitor trip progress and notify passengers in real time.
- Role-Specific Navigation and Access Control
    - Problem: Avoiding user confusion between driver/passenger/HR/Admin screens.
    - Solution: Created a role-based navigation system that routes users to the correct flow immediately after login.
- Notification Reliability
    - Problem: Ensuring push notifications arrived reliably across devices.
    - Solution: Used Firebase Cloud Messaging with proper device token registration per session, and managed lifecycle events to avoid missed updates.

Experiments & Results

Experimental Setup

Software Environment

The system was developed and tested as a native Android application using:

- Android Studio with Kotlin and Jetpack Compose
- Firebase services including Firestore, Authentication, Functions, and Cloud Messaging
- Google Maps Directions API for real-time route calculation and navigation
- Excel for test case management (STD), bug tracking, and test documentation

Hardware Environment

- Testing was done on physical Android devices running Android 11 through 14
- Devices with GPS, internet access, and Google Maps installed

Evaluation Metrics

The following success criteria were defined:

- 100% of sanity tests must pass
- At least 85% of test cases must be executed
- Minimum of 81% of executed tests must pass
- No Critical or High severity bugs allowed at release
- Up to 5% Medium, and 10% Low severity bugs allowed

Results

Quantitative Results

- Total test cases defined: **20**
- Test cases executed: **20** (100%)
- Passed tests: **19** (95% of executed)
- Blocked/unexecuted tests: 0

- Bugs found:
    - **Critical**: 0
    - **High**: 0
    - **Medium**: 1
    - **Low**: 0

Analysis & **Discussion**

Strengths

- High test pass rate with over 90% of executed cases passing
- Zero critical/high bugs, confirming stability and robustness
- Responsive UI and real-time syncing verified across different roles
- Smooth integration with Google Maps API and Firebase Cloud Messaging
- Efficient ride matching performance under time and detour constraints

Weaknesses / Limitations

- one medium severity bug remains.
- No testing on iOS (not supported in current version)

Benchmark Comparison

Compared to similar solutions:

- WePool provides company-restricted ride matching
- Approval workflows, real-time updates, and admin control are not available in most public apps
- Performance-wise, ride matching completes within 2 seconds, in line with industry standards

Discussion

Insights Gained from the Results

The testing process confirmed that WePool delivers a robust and responsive carpooling experience tailored to intra-organizational needs. The high pass rate of executed test cases (over 90%) and absence of critical or high-severity bugs suggest strong stability and reliability in real-world use.

We also found that the system's role-based structure (Driver, Passenger, HR Manager, Admin) worked well in guiding users through the intended flows with minimal confusion. Real-time features, such as live ride tracking and notifications, functioned smoothly and significantly improved user engagement and coordination.

From a development perspective, the use of Firebase services and Google Maps API proved to be efficient and scalable for our purposes, simplifying backend management while still offering powerful capabilities like secure authentication, and real-time updates.

Limitations of Our Approach

Despite the successful implementation, a few limitations remain:

- Platform support: The system currently supports only Android. Users on iOS or web platforms are not supported at this stage.
- Device dependency: Some features, such as GPS tracking and live navigation, depend on device permissions, battery optimizations, and internet stability, which vary across Android models.
- Hardcoded constraints: Certain policies (e.g., detour limits, ride timing rules) are currently hardcoded and not editable by the company without developer intervention.
- Manual test coverage: Testing was conducted manually, which limits coverage depth. Automated testing could improve long-term maintainability.

Potential Improvements and Alternative Solutions

To enhance the system further, we propose the following improvements:

1. Cross-platform support: Develop an iOS version or a responsive web interface to support a broader range of users.
2. Admin-configurable settings: Allow HR/Admin users to define ride matching policies, detour thresholds, and penalties dynamically through the dashboard.
3. In-app navigation view: Integrate a basic in-app navigation view instead of relying entirely on Google Maps redirection.
4. Offline resilience: Add support for basic offline functionality (e.g., storing requests locally until reconnected).
5. Automated testing: Incorporate unit tests and UI test automation (e.g., using Espresso or Robolectric) to reduce regression effort and catch bugs earlier.
6. Analytics dashboard: Add visual analytics tools for HR managers and admins to better understand user engagement, ride trends, and environmental impact.

Conclusion & Future Work

This project successfully developed and deployed WePool, a secure and intelligent carpooling system designed specifically for employees within the same organization. Key accomplishments include:

- A fully functional native Android app built with Kotlin and Jetpack Compose
- Seamless integration with Firebase (Authentication, Firestore, Cloud Messaging, Functions)
- Support for role-based flows: Driver, Passenger, HR Manager, Admin
- Real-time ride matching, route optimization using Google Maps Directions API, and live navigation
- An intuitive UI with notification support, favorite location management, and ride filtering
- Over 90% of executed test cases passed, with zero critical bugs, demonstrating the system's reliability and quality

Project Impact

WePool addresses real-world problems such as commuting inefficiency, high costs, and environmental concerns by enabling company employees to share rides easily and securely. Its closed-community approach, supported by HR moderation and approval workflows, fills a gap left by public carpooling platforms that often lack organizational control and trust.

From a technical perspective, the use of cloud services and modular architecture allowed for rapid development and strong real-time capabilities without managing traditional backend servers. The project also strengthened our knowledge in areas like mobile architecture, user experience design, API integration, and quality assurance processes.

Future Research and Development

While the current version of WePool is stable and functional, several enhancements could be pursued in future iterations:

- Cross-platform development: Create an iOS or web version to broaden accessibility
- Admin-controlled policy engine: Let HR/Admins define detour limits and ride timing rules dynamically
- Analytics dashboard: Visual insights for HR/Admins on ride usage, user behavior, and sustainability metrics
- Automated testing suite: Introduce UI and unit testing frameworks to improve regression testing and reduce QA overhead
- Offline support and fallback: Implement limited offline functionality and recovery in case of network disruptions
- Machine learning for matching: Use ride history and patterns to recommend rides or detect optimal matches over time

User guide & App walkthrough

🔧 Prerequisites

Android Studio Hedgehog or later

Firebase project with:

- Firestore, Auth, Messaging, Functions, and Storage enabled

Google Cloud Project with:

- Maps SDK for Android
- Places API
- Directions API
- Static Maps API (if map images are used)

⚙️ Setup

git clone https://github.com/aviel3899/WePool.git

Open in Android Studio and sync Gradle.

Configure Firebase:

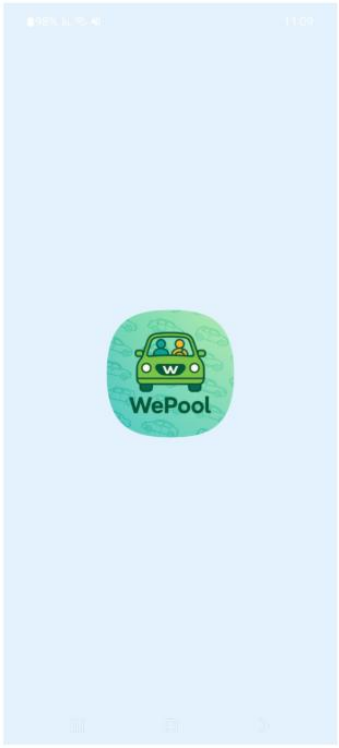
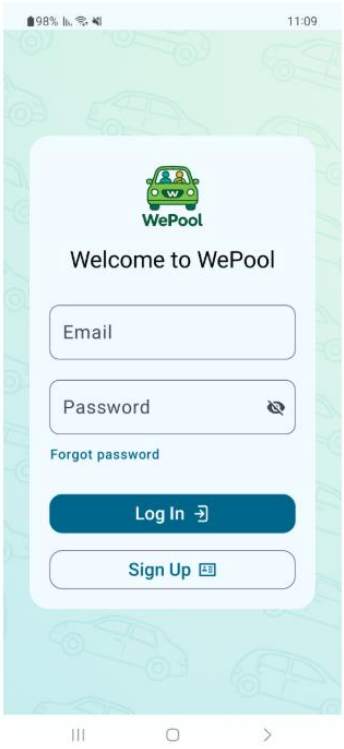
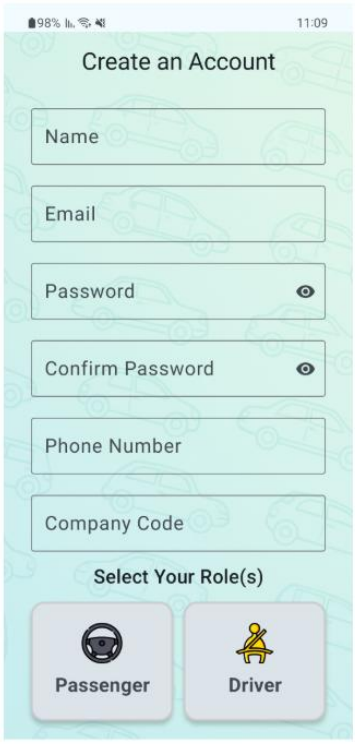
- Place your google-services.json file in the /app directory.
- Ensure your Firebase project includes the necessary APIs and services.

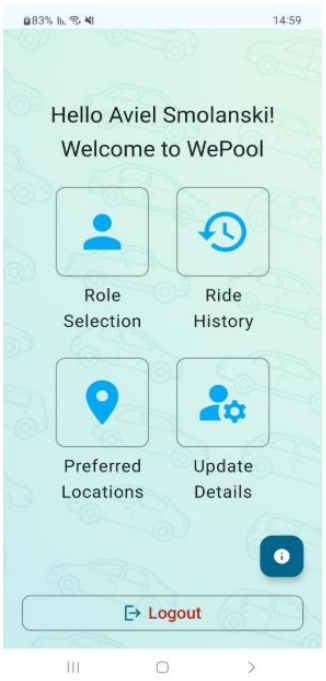
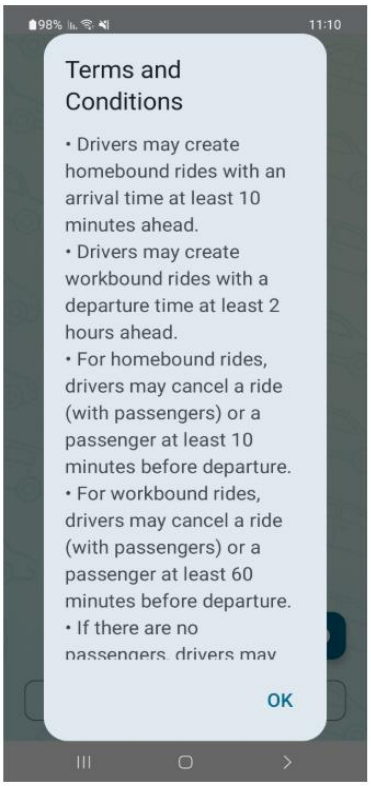
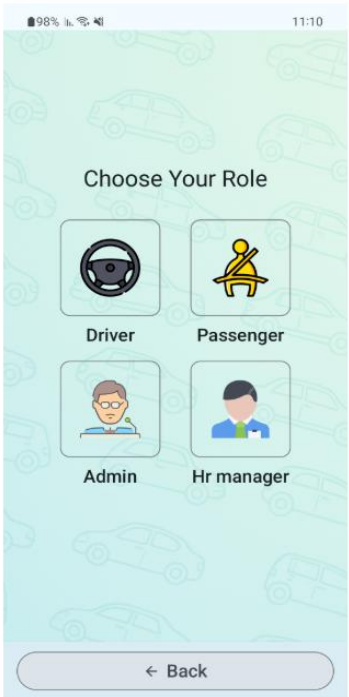
Run the app on an emulator or physical Android device (with Google Play Services).

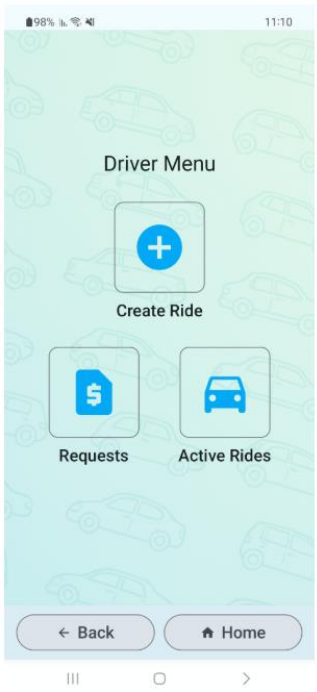
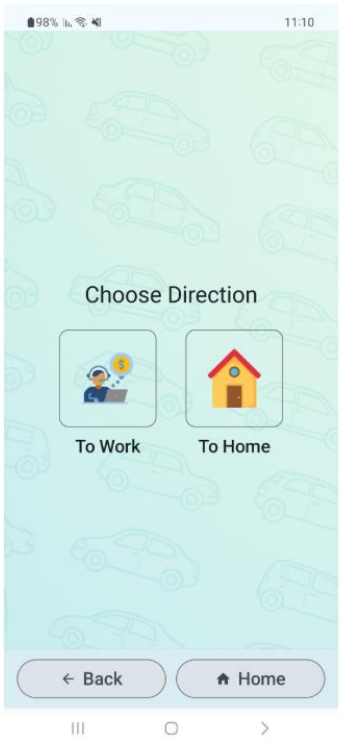
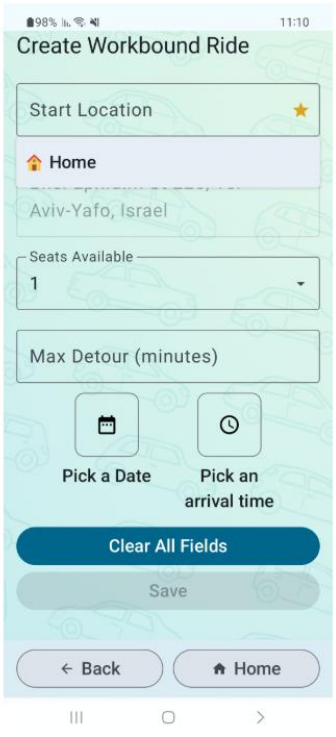
Important note !

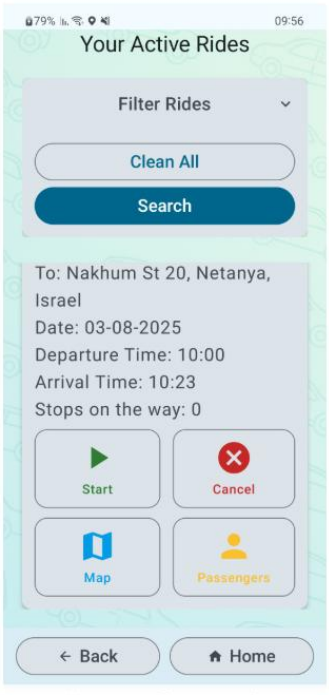
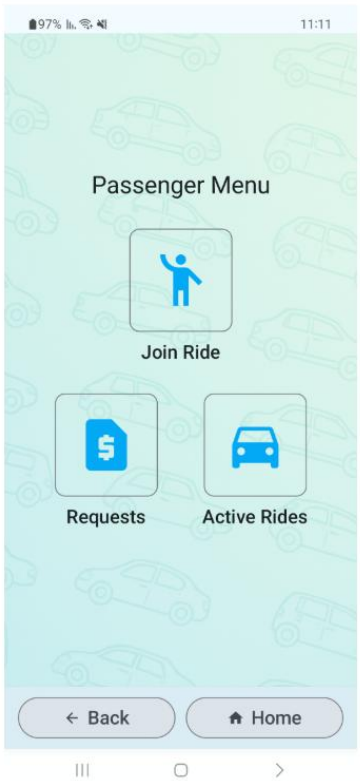
To make the app work smoothly, you must first set an admin account. For your own convenience, in the source code go to infrastructure → config and within the AdminConfig object file change list of authorizedAdminEmails to your own email. Use this email address when signing up. Now you are the admin!

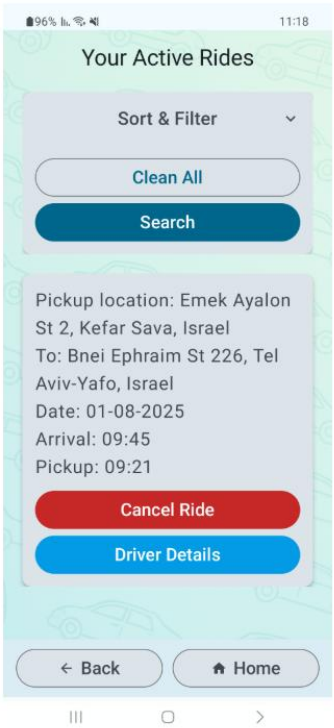
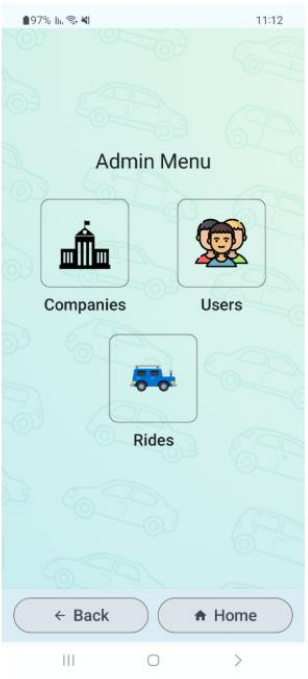
Screenshots

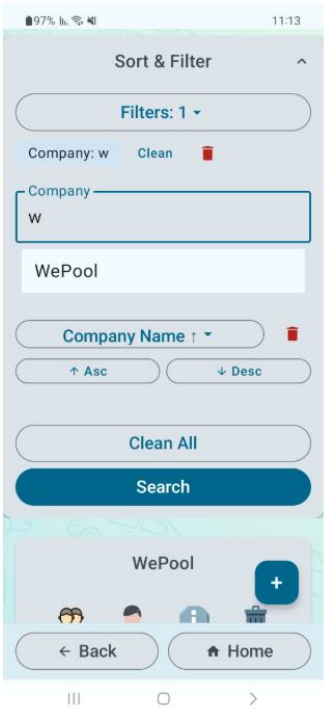
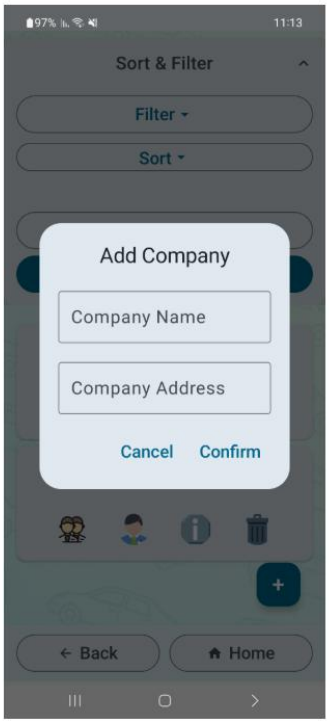
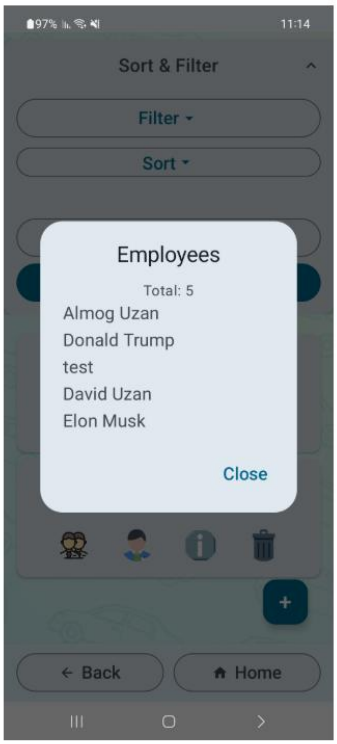
Logo	Login	Signup
		

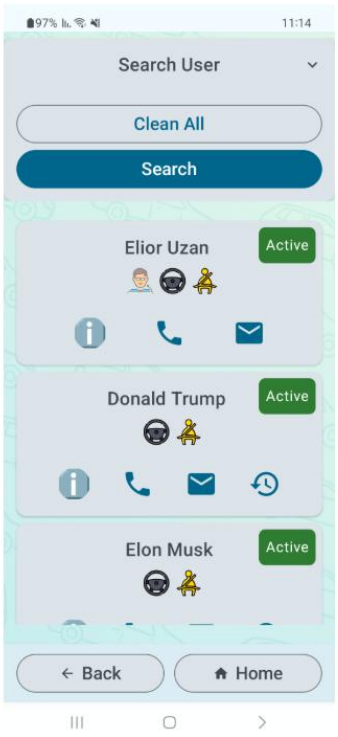
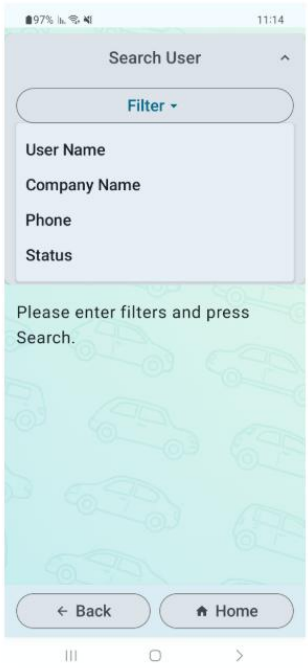
Home Page	Terms & Condotions	Role Selection
		

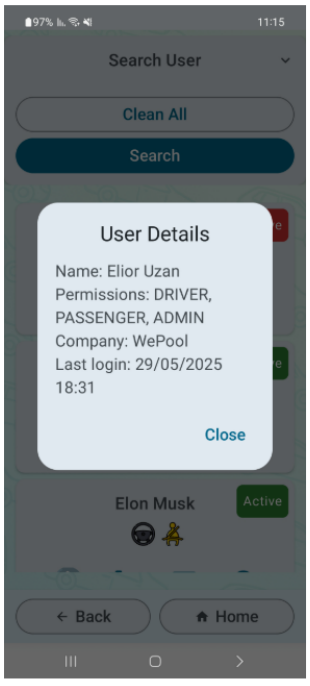
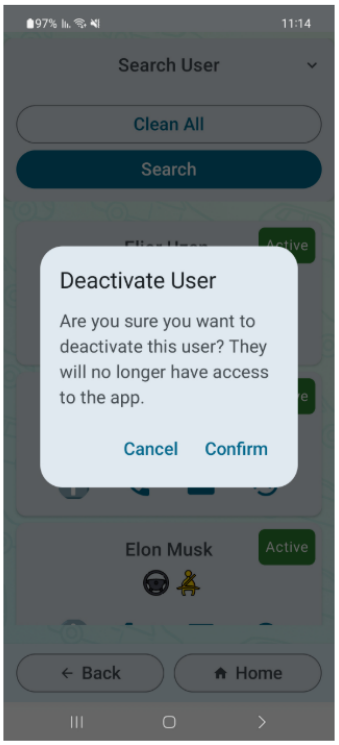
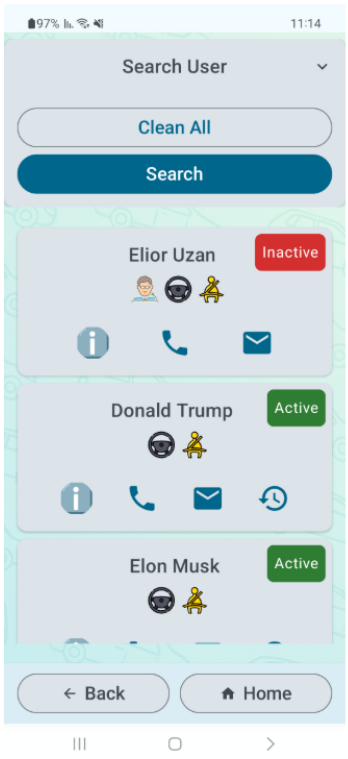
Driver Menu	Choose Ride Direction	Ride Creation
		

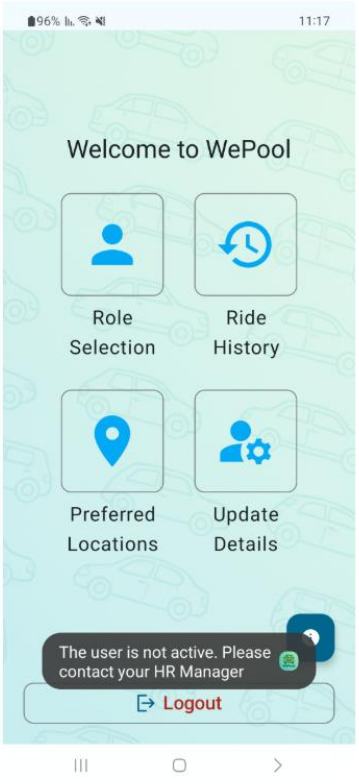
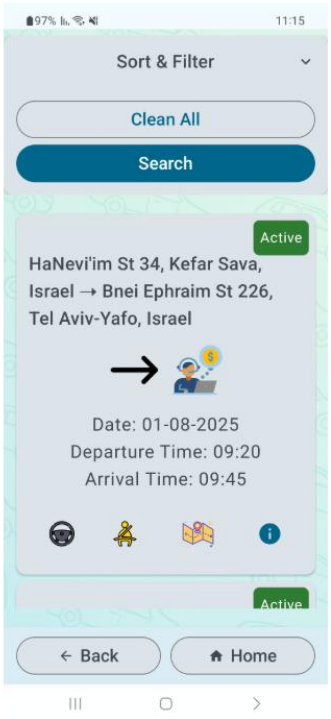
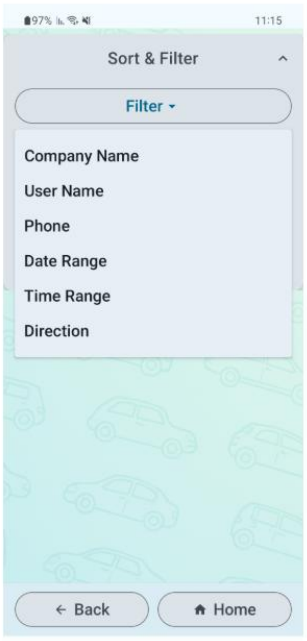
Driver Active Rides	Passenger Menu	Join a Ride
		

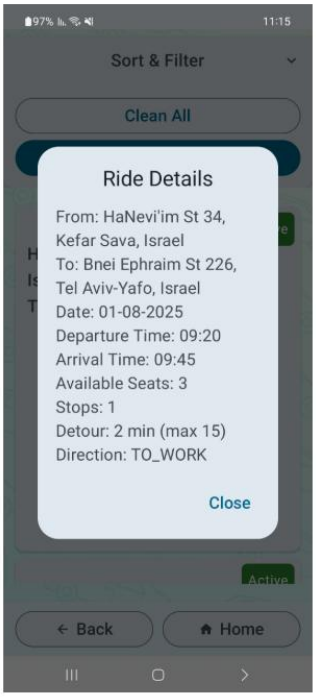
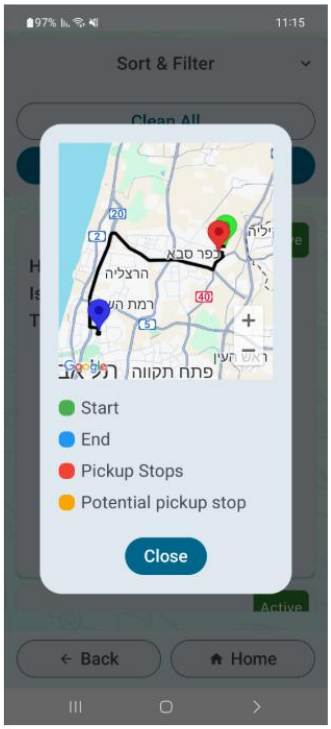
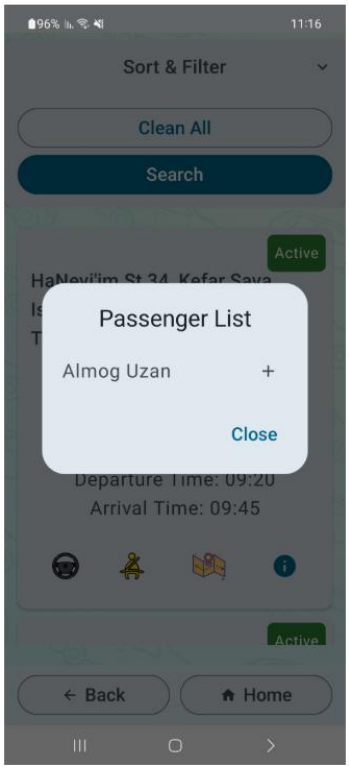
Passenger Active Rides	Admin Menu	Admin Compamies Manage
		

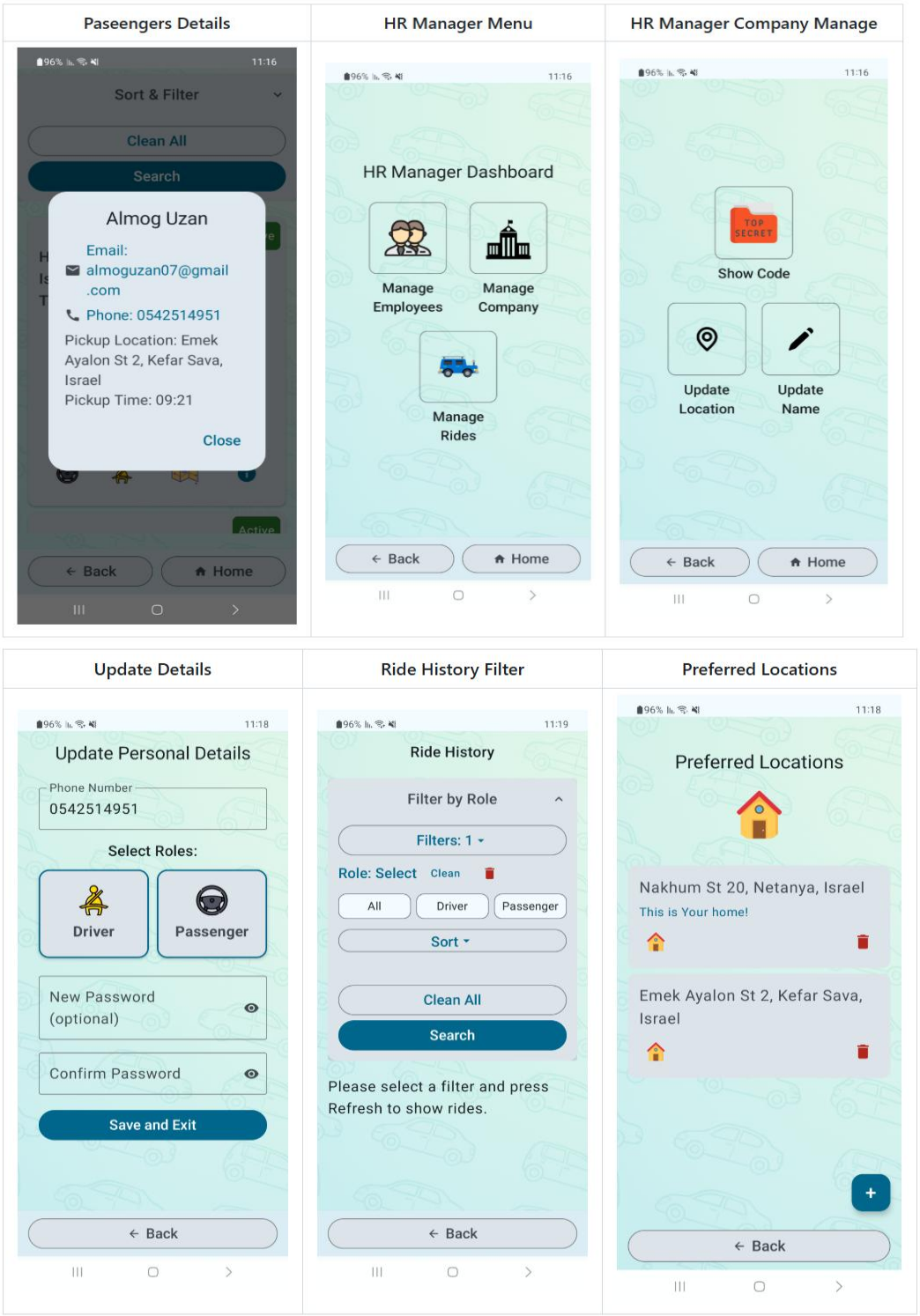
Company Filter	Add Company	Company Employees
		

Company's HR Manager	Admin Users Manager	User Filter
		

User Details	Deactivate User	User Deactivated
		

User Deactivated Home Page	Admin Rides Manage	Ride Filter
		

Ride Details	Ride Dynamic Map	Ride Passengers List
		



For a more engaging walk – through this app, you can click the link below:

https://www.youtube.com/watch?v=InsN7Mb17d0

References

[1] Google, "Directions API | Google Maps Platform Documentation," [Online]. Available: https://developers.google.com/maps/documentation/directions/

[2] Firebase, "Firebase Cloud Messaging," [Online]. Available: https://firebase.google.com/docs/cloud-messaging.

[3] D. H. Douglas and T. K. Peucker, "Algorithms for the reduction of the number of points required to represent a digitized line or its caricature," *The Canadian Cartographer*, vol. 10, no. 2, pp. 112–122, 1973.

[4] Wikipedia, "Ramer–Douglas–Peucker algorithm," [Online]. Available: https://en.wikipedia.org/wiki/Ramer–Douglas–Peucker_algorithm.

[5] OpenAI, "ChatGPT," [Online]. Available: https://chat.openai.com. Assisted in drafting, clarifying, and structuring project content.