Capstone Project Phase B

Carpool

Match, Drive, Share

project number
24-1-D-29

Supervisor
Dr. Zeev Barzily

Authors
Ravid Goldin, ravid.goldin@e.braude.ac.il
Ben Ben Baruch, ben.ben.baruch@e.braude.ac.il

Date: September 22, 2024

Table of Contents

Abstract

Carpool is a mobile application designed to create ride-sharing groups for drivers with common destinations, addressing the challenges of traffic congestion and inefficient commuting. This document outlines the Phase B development of Carpool, detailing its software architecture based on the MVC pattern, cross-platform development using Flutter, backend services with Firebase, and route visualization with OpenStreetMap.
The implementation of key features is discussed, including the points-based driving rotation system, group creation and management, and user authentication. The document addresses development challenges, testing methodologies, and user interface design considerations aimed at fostering a sense of community among users. It also covers the engineering process, functional and non-functional requirements, and plans for future enhancements.
By focusing on building lasting commuting communities and providing a fair, transparent system for sharing driving responsibilities, Carpool aims to offer a sustainable, cost-effective, and community-oriented transportation solution.

Keywords: Flutter • Firebase • OpenStreetMap • Cloud • MVC • Ridesharing • Commute • Drivers • Groups • Points System

Chapter 1
Introduction

In In today's world, many areas suffer from inefficient public transportation systems, leading to many people driving alone to common destinations such as workplaces or educational institutions. This situation results in significant traffic congestion, increased fuel consumption, and higher vehicle wear and tear. Additionally, it contributes to environmental pollution and associated health risks due to increased emissions. [1]

The Carpool project aims to address these challenges by offering an innovative mobile application designed to connect drivers who travel regularly to the same destination. Unlike traditional carpooling apps that often focus on one-time rides or random matches, our approach emphasizes the creation of permanent groups for consistent carpooling. This focus on routine and shared goals fosters a sense of community and trust among participants, making the process more enjoyable and reliable.

1.1 Problem Definition

- Many individuals find themselves driving alone to common destinations like workplaces or educational institutions [3], resulting in increased traffic congestion, higher fuel consumption, and elevated stress levels. This situation not only impacts individual commuters but also contributes significantly to environmental pollution and urban congestion. [2]

- The difficulty in forming consistent and reliable carpooling arrangements. While ad-hoc ride-sharing solutions exist, they often lack the structure and consistency needed for daily commuters. This inconsistency can lead to unreliable transportation options and a lack of community building among regular commuters.

- The challenge of fairly distributing the responsibilities and costs associated with carpooling. Without a structured system, carpooling arrangements can become imbalanced, with certain individuals bearing a disproportionate share of the driving duties or expenses. This imbalance can lead to dissatisfaction and the eventual breakdown of carpooling arrangements.

- The environmental impact of numerous single-occupancy vehicles on the road. The prevalence of solo drivers contributes significantly to carbon emissions and air pollution, especially in urban areas during peak commuting hours. This not only affects air quality but also accelerates climate change concerns.

1.2 Solution

Our solution is a mobile application designed specifically for drivers commuting to workplaces, universities, or colleges. The app focuses on creating permanent carpooling groups based on shared schedules and routes, facilitating a more organized and consistent ride-sharing experience. Users can create or search for carpool groups based on specific criteria such as schedule, route, and available seats in the car, ensuring they connect with others who have compatible commuting patterns. The app implements a fair, points-based system where the member with the lowest points becomes the next driver, receiving a point after completing a drive. This ensures an equitable distribution of driving responsibilities over time. Each group establishes fixed pickup locations, clearly displayed on an in-app map, streamlining the pickup process and eliminating confusion. By focusing on permanent groups with set meeting points and utilizing a points-based rotation system, our Carpool app aims to make carpooling a more reliable, fair, and efficient option for daily commuters, ultimately creating stable carpooling communities with shared commuting goals.

1.3 Application's Stakeholders

The potential stakeholders of the Carpool system are:

- **Daily Commuters**: Individuals who travel regularly to the same destination, such as workplaces or universities, and are looking for a reliable and cost-effective solution for shared transportation.

- **Employers and Organizations**: Companies and institutions interested in promoting sustainable transportation options for their employees, reducing parking needs, and enhancing employee satisfaction. [2]

- **Environmental and Community Groups**: Organizations focused on reducing traffic congestion and pollution, who see carpooling as a tool for promoting eco-friendly practices and community building.

- **Transportation Authorities**: Municipalities and governmental bodies interested in supporting sustainable commuting options to alleviate traffic congestion and improve urban mobility.

Chapter 2
Solution Description

2.1 Software Architecture

Our carpool application's software architecture is designed to support both Android mobile and web platforms, offering users flexibility in accessing the service. This multi-platform approach is built using Flutter [4], a popular open-source framework that allows for cross-platform development with a single codebase. The application integrates with Firebase for backend services, utilizing its authentication system for secure user management and Firestore for efficient, real-time data storage and synchronization. The architecture leverages various open-source libraries, including UI widgets for enhanced user interface components, and Flutter Map integrated with OpenStreetMap for robust route visualization and mapping functionality. This modular and interconnected structure enables a seamless carpooling experience across devices, combining user-friendly features with cloud-based services to deliver a responsive and scalable platform. The design emphasizes smooth interaction between users, the application (whether on mobile or web), and external services, creating a comprehensive solution for carpooling needs.

- **User:** Users interact with the Carpool application by creating or joining carpool groups, selecting preferred meeting points, and viewing route maps. The interaction is bi-directional, meaning users initiate actions and receive feedback from the system in real time.

- **Application:** The application serves as the core of our architecture, handling user requests, managing data, and interacting with external services. It encompasses the business logic, processes user inputs, and determines how data is presented to users, ensuring a seamless experience.

- **Firebase:** Firebase [5], as an integral part of the architecture, provides backend services, including real-time data updates and user authentication. The application communicates with Firebase to handle user data, group information, and push notifications, ensuring that users are always updated about group activities and scheduling.

- **OpenStreetMap:** OpenStreetMap [6] is an open-source maps service used to provide interactive maps, allowing users to view meeting points, routes, and stations along the way. This integration ensures accurate location services and a smooth user experience.

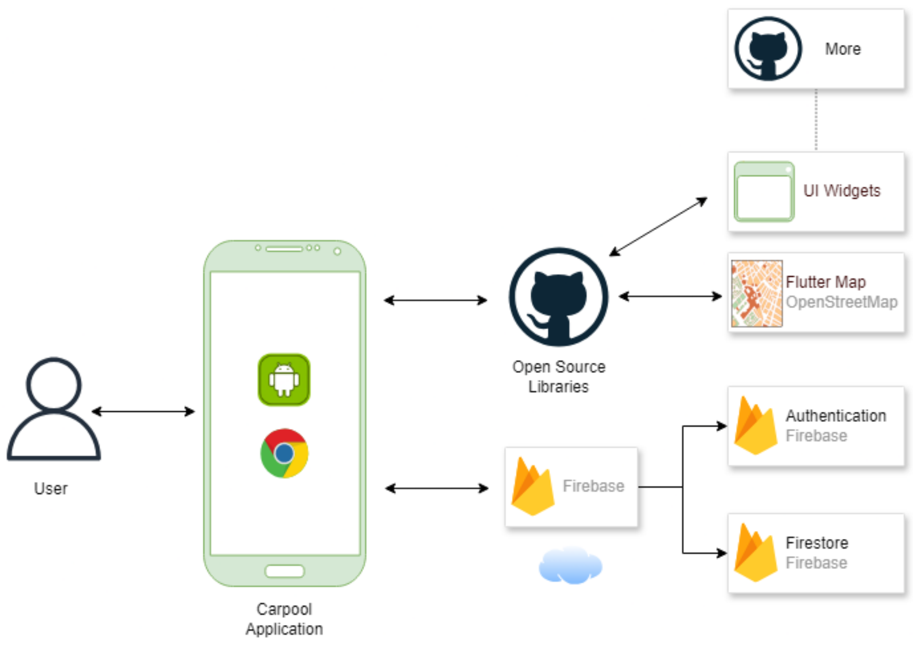


Figure 2.1: *Software architecture of the Carpool application, illustrating the interaction between the user, application, Firebase services, and OpenStreetMap for location-based services.*

2.2 Application Architectural Pattern (MVC)

The Carpool application follows the Model-View-Controller (MVC) architectural pattern, which is suitable for structuring a scalable and maintainable mobile application [7]. This pattern separates the application into three main components:

- **Model**: This layer is responsible for managing the data and business logic of the application. It handles data retrieval from Firebase, processes the information (like user and groups information), and ensures data integrity.

- **View**: The View layer manages the user interface. It displays data from the Model and listens to user actions, like selecting a route or joining a carpool group. The user interface is designed to be intuitive and responsive, with features like map-based route displays and easy-to-navigate menus.

- **Controller**: The Controller acts as an intermediary between the View and the Model. It handles user inputs from the View, updates the Model, and then refreshes the View with the updated data. For example, when a user sets a preferred pickup point, the Controller updates the Model and triggers a View update to reflect the new route on the map.

This separation ensures that changes to one component (like updating the interface) do not directly impact the other layers, making the application more maintainable and easier to extend.

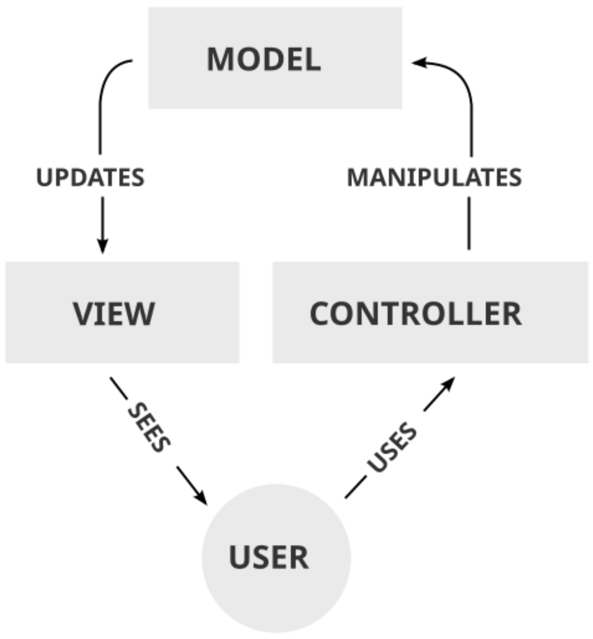


***Figure 2.2**: Illustration of the Model-View-Controller (MVC) pattern in the Carpool application, showing the interaction between the user, Model, View, and Controller components.*

2.3 Package Structure

The Carpool application's package structure follows the MVC (Model-View-Controller) architectural pattern, ensuring clear separation of concerns, maintainability, and scalability. The components are organized into distinct folders representing the responsibilities of each layer within the MVC structure.

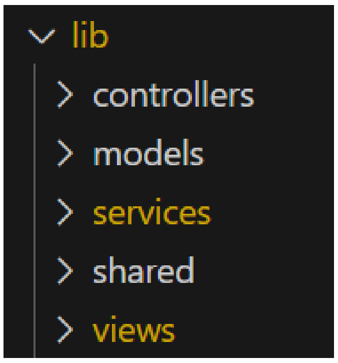
- **Controllers**: Contains the controller classes responsible for handling user inputs, managing interactions between the Model and the View, and updating the View based on data changes. Examples include controllers for managing user profiles, handling notifications, and controlling group pages.

- **Models**: Contains the data models that represent the structure of the application's data. This includes entities like user profiles, rides, and group details. The models are also responsible for data handling and communication with external services.

- **Views**: Contains the UI components such as pages, widgets, and other elements responsible for presenting data to the user and capturing user inputs. Each screen in the application, like the home page, login page, and ride search page, is represented here.

- **Services**: Provides classes that interact with external APIs, databases, and other backend services. These services manage data retrieval, storage, and updates across the application.
- **Shared**: Contains reusable components, utilities, and constants that are shared across the application, such as common functions, themes, and configuration settings.

This structure allows for easy maintenance, testing, and future expansion of the application while adhering to best practices in software development.



```
v lib
  > controllers
  > models
  > services
  > shared
  > views
```

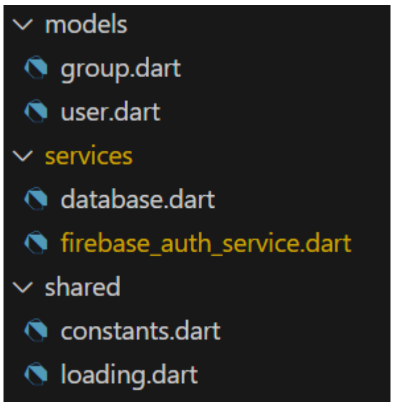
Figure 2.3: *Directory structure of the Carpool application, showing the organization of files into controllers, models, services, shared components, and views under the 'lib' folder.*

2.3.1 Data Package

- **models**: Contains the data classes that represent the core entities in the application. For example, User and Group. These classes define the structure of the data and provide the foundation for all data operations within the application, such as user and group management.

- **services**: Contains classes responsible for handling interactions with external services, such as Firebase. These classes manage tasks like data retrieval, storage, and real-time updates. For example, firebase_auth_service.dart handles user authentication, while database.dart manages data operations related to users and groups.

- **shared**: Contains utility functions, constants, and shared data structures that are used across different parts of the application. For instance, constants.dart stores constant values, and loading.dart includes functions related to loading indicators.

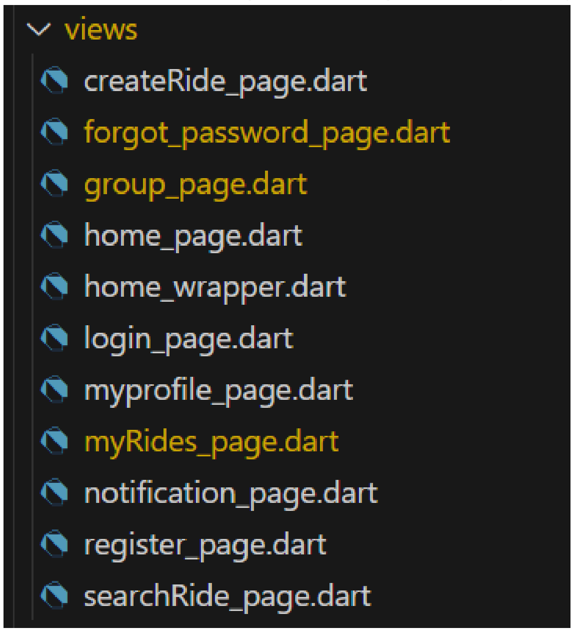


```
models
  group.dart
  user.dart
services
  database.dart
  firebase_auth_service.dart
shared
  constants.dart
  loading.dart
```

Figure 2.4: *Directory structure of the Data Package, showing the organization of models, services, and shared resources.*

2.3.2 UI Package

In the views package, we have organized each screen of the Carpool application into its own file, such as home_page, login_page, and myRides_page, among others. Each file represents a specific screen or component of the user interface, encapsulating the layout and logic for that part of the application. This organization helps keep the codebase modular and maintainable, ensuring that each screen's components are grouped together logically.



```
views
  createRide_page.dart
  forgot_password_page.dart
  group_page.dart
  home_page.dart
  home_wrapper.dart
  login_page.dart
  myprofile_page.dart
  myRides_page.dart
  notification_page.dart
  register_page.dart
  searchRide_page.dart
```

Figure 2.5: *Structure of the UI Package, showing the organization of different screen components under the views folder.*

2.3.3 Shared Package

The shared package in the Carpool application contains utility classes that are used across different parts of the application. These shared resources help maintain consistency and efficiency in the codebase. The key files in this package include:

- **constants.dart**: This file defines constant values that are used throughout the application. Notably, it includes the destination point for the carpool rides, which is set to "Braude Academic College." By centralizing these values, the application ensures that updates to key settings are easy to manage and apply uniformly across the system.

- **loading.dart**: This file provides a reusable loading widget that is displayed during asynchronous operations. The loading indicator helps improve user experience by visually indicating that the application is processing a request, such as fetching data or submitting information.

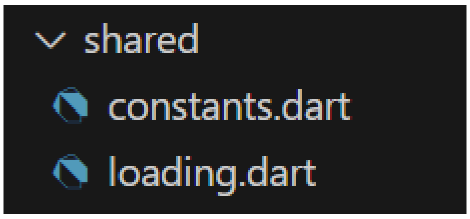


Figure 2.6*: Structure of the Shared Package, showing the organization of utility classes like constants.dart and loading.dart.*

2.3.4 Controller Package

The controllers package in the Carpool application contains classes that manage the interaction between the user interface (UI) and the data models. Controllers are responsible for processing user inputs, updating the model, and ensuring that the view reflects the current state of the data. This separation of concerns allows the application to maintain a clean and organized architecture.
Each controller in this package is tied to a specific screen or functionality within the app:

- **create_ride_page_controller.dart**: Manages the logic for creating new carpool rides, including handling user inputs and validating ride details before submission.
- **group_page_controller.dart**: Handles the interactions on the group page, such as joining a group, leaving a group, and viewing group details.
- **home_page_controller.dart**: Manages the main dashboard, coordinating the display of available rides, notifications, and user actions on the home screen.
- **notification_page_controller.dart**: Oversees the logic related to user notifications, ensuring that alerts are processed and displayed correctly.
- **profile_page_controller.dart**: Manages user profile data, allowing users to update their personal information and preferences.
- **search_ride_page_controller.dart**: Handles the search functionality, processing user queries to find suitable carpool rides.

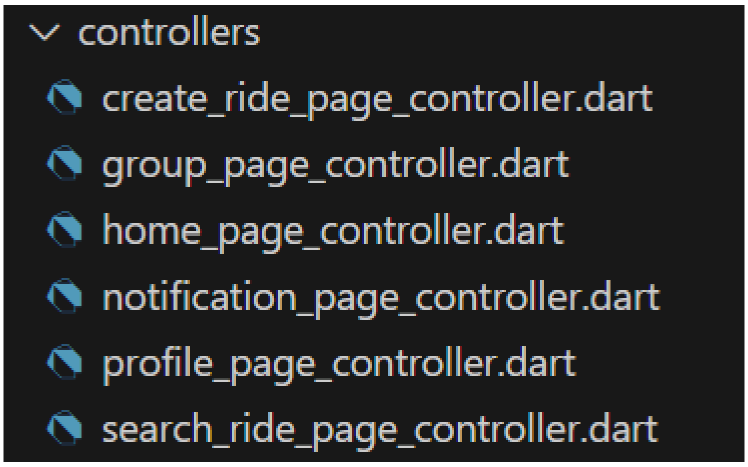

controllers
create_ride_page_controller.dart
group_page_controller.dart
home_page_controller.dart
notification_page_controller.dart
profile_page_controller.dart
search_ride_page_controller.dart

Figure 2.7*: Structure of the Controller Package, showing the organization of controllers responsible for managing UI interactions.*

2.4 Database Structure

The database structure for Carpool was designed with a focus on modularity, scalability, and efficient data access. By organizing data into distinct collections within Firebase, the app can perform targeted queries, retrieving only the necessary information. This approach optimizes read operations, enhancing performance and reducing operational costs.

Each collection in the database corresponds to a key feature or entity within the Carpool ecosystem. This structure allows for easy expansion of features and efficient management of user data, ride information, and group details. The database is organized into the following main collections:

- **Users**: Stores information about each user, including their profile details (name, email, phone number, address), available seats in their car, and the groups they are a part of.
- **Groups**: Stores the details of each carpool group, including meeting points, members, ride schedules, and other related information. Each group document includes a list of members, selected days, and times for the rides.
- **Notifications**: Manages notifications sent to users, including details like the message body, timestamp, read status, and the user ID associated with the notification.

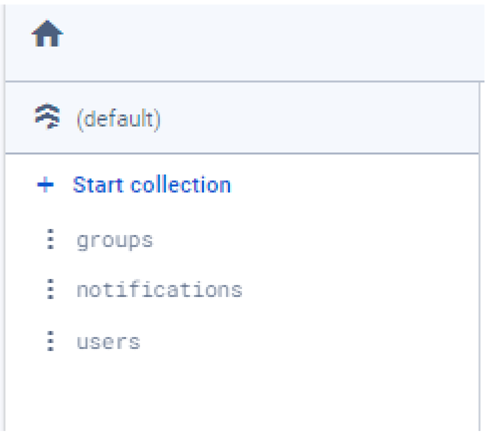
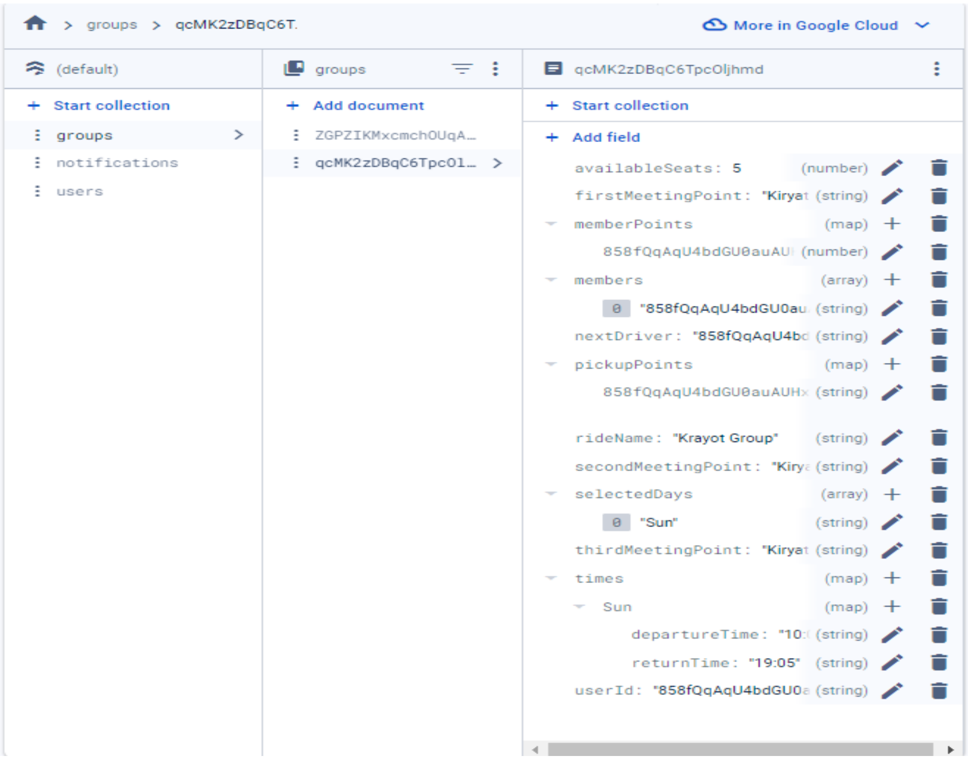


__Figure 2.8__: Structure of the Firebase database collections, showing the organization of user, group, and notification data.

2.4.1 Collection: Groups

The "groups" collection contains documents that represent each carpool group in your application. Each document contains detailed information about the specific group.
Example of a document in the "groups" collection:

- **availableSeats**: The number of available seats in the car for the group (number).
- **firstMeetingPoint**: The first meeting point for the ride (string).
- **memberPoints**: A map that tracks points for each group member. The keys are user IDs, and the values are the points assigned to those members (map).
- **members**: An array containing the user IDs of all members in the group.
- **nextDriver**: The user ID of the next driver who is scheduled to drive (string).
- **pickupPoints**: A map of pickup points where members will be picked up during the ride.
- **rideName**: The name of the group or ride (string).
- **secondMeetingPoint**: The second meeting point along the route (string).
- **selectedDays**: An array listing the days of the week when the ride occurs (e.g., "Sun" for Sunday).
- **thirdMeetingPoint**: The third meeting point along the route (string).
- **times**: A map containing the departure and return times for the selected days.
    - **departureTime**: The time when the ride starts (string).
    - **returnTime**: The time when the ride ends (string).
- **userId**: The user ID of the group creator or administrator (string).



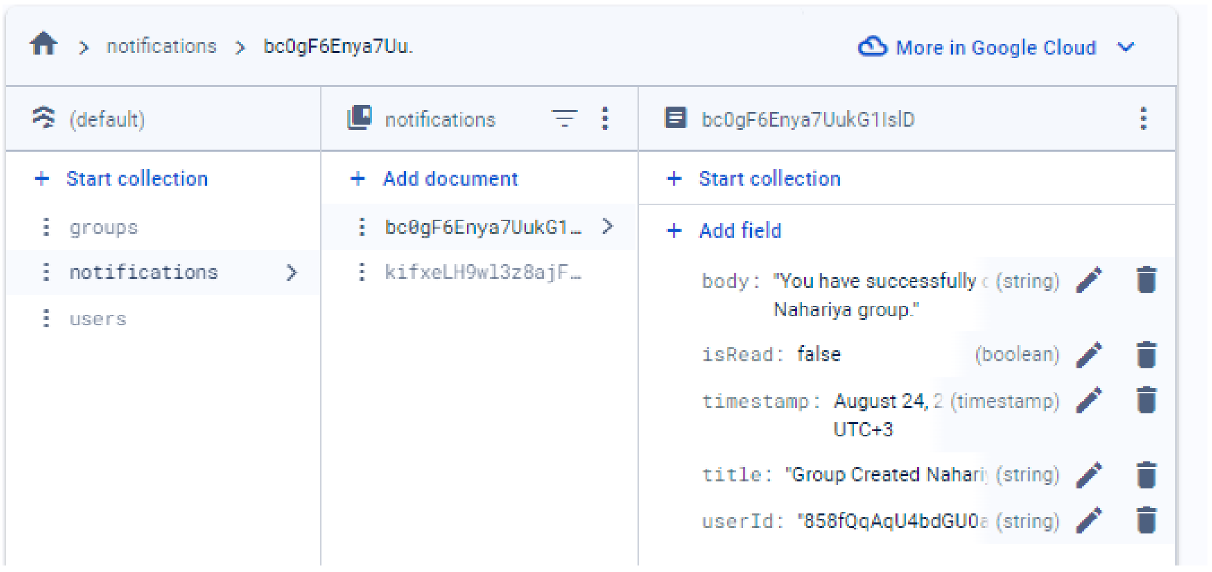
```
availableSeats: 5                    (number)
firstMeetingPoint: "Kiryat           (string)
memberPoints                         (map)
  858fQqAqU4bdGU0auAU                (number)
members                              (array)
  0 "858fQqAqU4bdGU0au               (string)
nextDriver: "858fQqAqU4bd            (string)
pickupPoints                         (map)
  858fQqAqU4bdGU0auAUHx              (string)

rideName: "Krayot Group"             (string)
secondMeetingPoint: "Kirya           (string)
selectedDays                         (array)
  0 "Sun"                            (string)
thirdMeetingPoint: "Kiryat           (string)
times                                (map)
  Sun                                (map)
    departureTime: "10:              (string)
    returnTime: "19:05"              (string)
userId: "858fQqAqU4bdGU0a            (string)
```

__Figure 2.9__: Example of a document in the "groups" collection within the Firebase database, displaying the structure and fields used to store carpool group information.

2.4.2 Collection: Notifications

The "notifications" collection contains all the notifications that are sent to users in the application. Each notification is represented by a unique document with fields that describe the notification's details.
Example of a document in the "notifications" collection:

- **body**: The message content of the notification (string). In this example, it contains the message "You have successfully created the Nahariya group."
- **isRead**: A boolean value indicating whether the notification has been read by the user (true or false).
- **timestamp**: The date and time when the notification was sent (timestamp). In this example, it shows "August 24, 2024 at 9:54:36 AM UTC+3".
- **title**: The title of the notification (string). In this example, it is "Group Created Nahariya group."
- **userId**: The user ID of the recipient of the notification (string). This links the notification to a specific user.



notifications > bc0gF6Enya7Uu.   More in Google Cloud

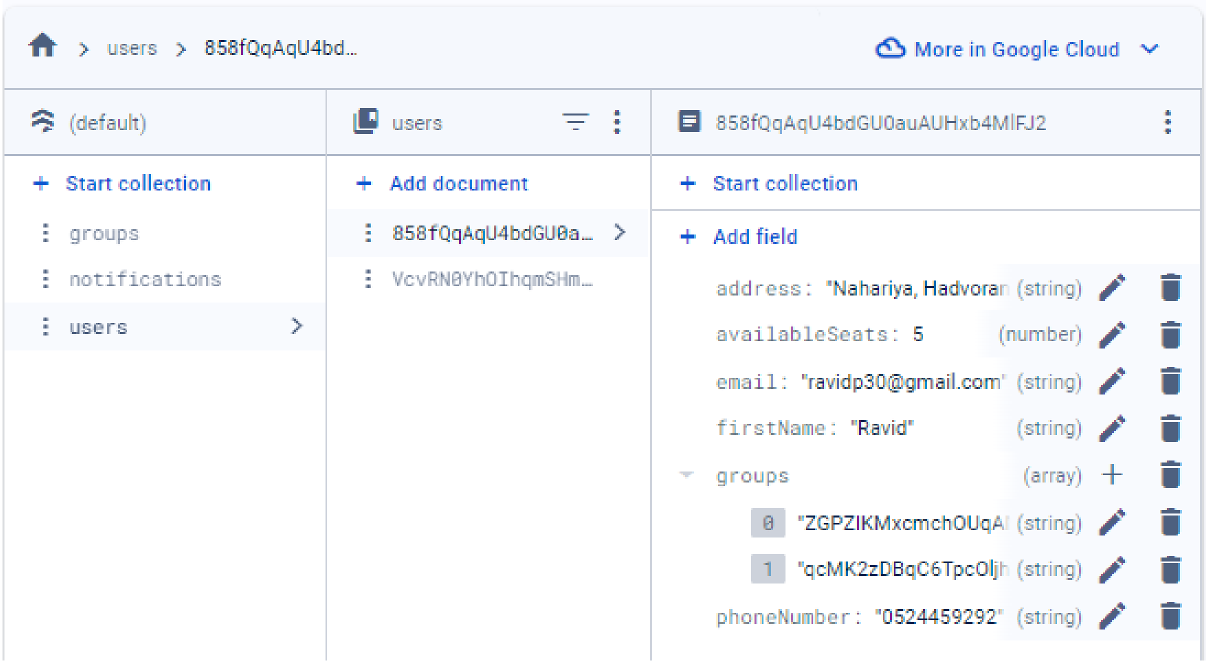
(default)	notifications	bc0gF6Enya7UukG1IslD
+ Start collection	+ Add document	+ Start collection
groups	bc0gF6Enya7UukG1…	+ Add field
notifications	kifxeLH9wl3z8ajF…	body: "You have successfully c… Nahariya group." (string)
users		isRead: false (boolean)
		timestamp: August 24, 2… UTC+3 (timestamp)
		title: "Group Created Nahari… (string)
		userId: "858fQqAqU4bdGU0… (string)

Figure 2.10: Example of a document in the "notifications" collection within the Firebase database, showing the fields used to store notification details sent to users.

2.4.3 Collection: Users

The "users" collection contains documents that represent each user in your application. Each document includes essential information about a user.
Example of a document in the "users" collection:

- **address**: The user's address (string). In this case, it is "Nahariya, Hadvoranit".
- **availableSeats**: The number of available seats the user has in their car (number).
- **email**: The user’s email address (string).
- **firstName**: The user’s first name (string). In this example, it is "Ravid".
- **groups**: An array containing the IDs of the groups the user is a member of.
- **phoneNumber**: The user’s phone number (string).



address: "Nahariya, Hadvoran (string)
availableSeats: 5 (number)
email: "ravidp30@gmail.com" (string)
firstName: "Ravid" (string)
groups (array)
0 "ZGPZIKMxcmchOUqA (string)
1 "qcMK2zDBqC6TpcOljh (string)
phoneNumber: "0524459292" (string)

Figure 2.11: *Example of a document in the "users" collection within the Firebase database, showing the fields used to store essential user information.*

Chapter 3
Analysis and Conclusions

3.1 Challenges and Solutions

Throughout the development of the Carpool application, several challenges were encountered, each requiring a unique solution. These challenges ranged from engineering and technical problems to database structures, architecture implementation, and user interaction. In this section, we will explore these challenges in detail, along with the innovative solutions devised to overcome them.

3.1.1 Architecture Implementation (MVC)

- **Challenge:** Implementing the MVC architectural pattern presented a significant challenge due to the complexity of maintaining clear separation of concerns. Although dividing the application into Model, View, and Controller layers is a well-established pattern, ensuring seamless communication between these layers requires careful planning. The goal was to maintain a modular structure while ensuring the application remained responsive and efficient.
- **Solution:** To address this challenge, we systematically defined the roles of each layer:
  - **Model:** Manages data structures and business logic.
  - **View:** Handles the presentation and visual elements of the application.
  - **Controller:** Acts as an intermediary between the Model and View, handling user inputs and updating the View based on Model changes.
- We focused on clearly separating the responsibilities of each component while ensuring that changes in one layer had minimal impact on the others. This approach allowed us to maintain code modularity and made the application more maintainable and scalable.

3.1.2 Database Structure

- **Challenge**: Designing an efficient Firebase database structure for Carpool that minimizes data operations while supporting complex relationships between users, groups, and rides. The structure needed to allow quick data retrieval and updates, particularly for frequently accessed information, while optimizing performance and reducing costs.
- **Solution**: We implemented a carefully planned database structure leveraging Firebase's capabilities. The design includes separate collections for users, groups, and notifications, with strategic use of sub-collections and denormalization techniques. This structure enables efficient querying and updating of data, supports key features like group management and ride scheduling, and maintains scalability for future

growth. By organizing data based on access patterns and relationships, we minimized I/O operations and improved overall application performance.

3.1.3 Map Integration

- **Challenge**: Integrating a reliable and cost-effective mapping service into our Flutter-based Carpool application presented a significant challenge. Accurate location services are crucial for coordinating meeting points and visualizing ride routes. Our initial plan to use Mapbox was hindered by its limited support for Flutter applications. We then considered Google Maps, but the associated costs for API usage were prohibitive for our project scope.
- **Solution**: To overcome this challenge, we implemented an open-source solution utilizing OpenStreetMap. This approach was facilitated by Flutter-compatible packages such as flutter_map [8], latlong2, and geocoding.
  By leveraging these tools, we were able to integrate interactive maps, handle geographical coordinates, and convert addresses to coordinates efficiently. The implementation of OpenStreetMap with these Flutter packages allows our application to provide accurate and up-to-date location information, ensuring that users can reliably view meeting points, plan routes, and visualize stations along the way.
  This solution not only met our technical requirements but also aligned with our project's budget constraints and open-source philosophy.

3.1.4 UI Experience

- **Challenge**: Creating an intuitive and user-friendly interface for the Carpool app presented a significant challenge. The app needed to efficiently present complex information such as ride schedules, group details, and route maps while remaining easy to navigate for users of varying tech-savviness. Balancing functionality with simplicity was crucial, especially for features like joining or creating carpool groups, scheduling rides, and viewing route information.
- **Solution**: To overcome this challenge, we adopted an iterative design approach focused on user feedback and usability testing. We created multiple prototypes and conducted testing sessions with potential users from diverse backgrounds. Based on their feedback, we made several key improvements to the interface. We simplified the group creation and joining process. A clean, map-centric interface was designed for displaying routes and meeting points using OpenStreetMap. We also created a dashboard-style home screen that provides quick access to active groups page, and notifications. These UI enhancements significantly improved the app's usability, making it easier for users to navigate the complexities of carpooling arrangements. The redesigned interface not only improved user satisfaction but also increased user engagement with key features of the app.

3.2 Results and Conclusions

The primary objective of our Carpool project was to develop a functional and scalable application that addresses the needs of commuters looking for an efficient ride-sharing solution. We are pleased to report that we have successfully implemented the core functionalities of the application, including user authentication, profile management, group creation and joining, ride scheduling, and route visualization using OpenStreetMap.

Our application's architecture is based on the Model-View-Controller (MVC) pattern, which has provided a robust foundation for future enhancements and maintainability. This structure has allowed us to efficiently manage the complex interactions between users, groups, and rides that are central to our carpooling system.

One of the main challenges we encountered was effectively dividing and managing the workload among team members. We overcame this by adopting a collaborative approach, assigning specific components or features to each team member based on their strengths. Regular communication and close collaboration were key, especially when integrating different parts of the application or solving complex issues related to ride matching and scheduling or points system algorithms.

In our decision-making process, we prioritized essential features that form the core of a carpooling application. We focused on creating a solid foundation that meets the fundamental needs of our users, such as easy group formation, efficient ride scheduling, and clear route visualization. Our approach involved starting with a basic UI design and iteratively improving it as we enhanced the app's functionality, allowing us to continuously refine the user experience.

While we have achieved our main goals, there are areas where we see room for improvement. For instance, our current implementation is primarily optimized for mobile devices, and we recognize the need to enhance compatibility across a wider range of devices and screen sizes. Additionally, while our use of Flutter allows for cross-platform development, we haven't yet fully optimized the app for iOS devices.

Looking ahead, we plan to address these limitations and further enhance the application. Our future roadmap includes improving device compatibility, and potentially integrating more advanced features such as real-time traffic updates and real-time location of the driver. These enhancements aim to improve the overall user experience and broaden the appeal of our Carpool application to a wider audience of commuters seeking sustainable and community-oriented transportation solutions.

3.3 Lessons Learned

Reflecting on our journey developing the Carpool application, we're pleased with our overall approach and outcomes. Our choice to use Flutter for cross-platform development proved invaluable. The integration of Firebase as our backend solution streamlined our data management processes, particularly crucial for handling real-time updates in ride schedules and group information.

One of our most significant challenges was implementing an effective mapping solution. After exploring various options, our decision to use OpenStreetMap with Flutter-compatible packages not only solved our immediate needs but also aligned with our project's open-source philosophy. This experience taught us the importance of flexibility in technology choices and the value of community-supported solutions.

In hindsight, we recognize areas for improvement in our development process. We could have benefited from earlier and more frequent user testing, particularly for key features like group creation and ride scheduling. This would have allowed us to refine our user interface and experience based on real-world feedback, potentially leading to a more intuitive final product.

Our database structure, while functional, could have been optimized further. A more in-depth analysis of data relationships, especially between users, groups, and rides, might have resulted in a more efficient schema. This insight underscores the importance of thorough planning in the early stages of development, particularly for applications with complex data interactions.

The implementation of our points-based system for fair ride allocation was a novel feature that required several iterations to perfect. Starting this development earlier in the process could have allowed for more comprehensive testing and refinement.

Despite these areas for improvement, we're proud of our team's ability to adapt to challenges and deliver a functional carpooling solution. The project has given us valuable insights into developing community-oriented applications, balancing user needs with technical constraints, and the importance of scalable design in mobile app development.

Moving forward, we plan to focus more on performance optimization across different devices and invest more time in user experience research. We've learned the importance of creating not just a functional app, but one that truly resonates with our target users' daily commuting needs.

This experience has equipped us with a deeper understanding of the unique challenges in developing shared mobility solutions. We're excited to apply these lessons to future projects, continuing to innovate in the realm of sustainable and community-driven transportation technology.

3.4 Project Metrics

Throughout the development of our Carpool application, we established key metrics to guide our progress and measure success. Our primary goal was to deliver a functional and scalable ride-sharing solution, while also prioritizing team learning and collaboration.

We aimed to deepen our understanding of cross-platform development using Flutter, with a focus on mastering the Model-View-Controller (MVC) architectural pattern. The successful integration of open-source technologies, particularly OpenStreetMap for route visualization, was a significant metric that aligned with our goal of creating a cost-effective, community-oriented solution.

Regular check-ins with our project advisor ensured we stayed on track and received valuable feedback. We balanced technical achievements with user-centric design, focusing on creating an intuitive interface for group management and ride scheduling, as well as implementing a fair, points-based system for ride allocation.

Reflecting on these metrics, we believe we have largely met our goals. The completion of a functional Carpool application capable of facilitating ride-sharing groups and managing complex scheduling demonstrates our technical achievement. The knowledge gained in Flutter development, Firebase integration, and open-source mapping solutions has significantly enhanced our mobile app development skills.

This experience of creating a solution that addresses real-world transportation challenges has been rewarding, reinforcing our commitment to developing technology that promotes community engagement and sustainable practices.

Chapter 4
Testing Process

Throughout the development of our Carpool application, we implemented a comprehensive manual acceptance testing strategy to ensure the app met our specified requirements and was user ready. Our process began with testing basic screen layouts and navigation functionality, allowing us to verify the app's overall structure and user flow before implementing detailed features. This approach provided a solid foundation for the subsequent development stages.

As we progressed, we conducted thorough manual testing after completing each major feature, including user authentication, group management, ride scheduling, and map integration using OpenStreetMap. We paid special attention to testing the points-based system for fair ride allocation across various scenarios. By simulating real-world usage patterns and consistently testing throughout the development cycle, we were able to iteratively refine the app's functionality and user experience. This rigorous testing approach ensured that our final product met our vision for an efficient and user-friendly carpooling solution.

4.1 Navigation Testing

Test Case	**Steps**	**Expected Result**	**Result**
Navigation from the login screen to the register screen	Open the app, click on “Don’t have an account? Sign up”	The app should navigate to the register screen.	The app navigates to the register screen.
Navigation from the login screen to home screen after a successful login	Open the app, enter valid login credentials, and click on the “Sign in” button	The app should navigate to the home screen.	The app navigates to the home screen.
Navigation from the login screen to reset password screen	Open the app, click on “Forget Password?”	The app should navigate to the forget password page.	The app navigates to the forget password page.
Navigation from the home screen to My Rides page using the navigation bottom bar.	On the home screen, click on “My Rides” in the navigation bottom bar.	The app should navigate to my rides page.	The app navigates to my rides page.
Navigation from the home screen to Notifications page using the navigation bottom bar.	On the home screen, click on “Notifications” in the navigation bottom bar.	The app should navigate to the notifications page.	The app navigates to the notifications page.

Navigation from the home screen to My Profile page using the navigation bottom bar.	On the home screen, click on “My Profile” in the navigation bottom bar.	The app should navigate to my profile page.	The app navigates to my profile page.
Navigation from my profile screen to Home page using the navigation bottom bar.	On the my profile screen, click on “Home” in the navigation bottom bar.	The app should navigate to the home page.	The app navigates to the home page.
Navigation from home screen to Search a ride page.	On the home screen, click on the “Find a ride” button.	The app should navigate to the search page.	The app navigates to the search page.
Navigation from home screen to Create a ride page.	On the home screen, click on the “Create a ride” button.	The app should navigate to the create page.	The app navigates to the create page.
Navigation from my rides page to a group page.	On my rides page, click on a ride group.	The app should navigate to the group page.	The app navigates to the group page.

4.1 Functionality Testing

Component	**Test Case**	**Result**
User Registration	Register a new user with valid details	Registration successful, user data stored in Firebase
User Registration	Register a new user with invalid details (e.g. missing email or existing email)	Registration failed, error message displayed
User Login	Login with valid details	Login successful, user navigates to the home screen
User Login	Login with invalid details	Login failed, error message displayed

Profile page	Update user’s name	Profile update successfully, changes reflected in Firebase
Profile page	Update phone number	Profile update successfully, changes reflected in Firebase
Profile page	Update address	Profile update successfully, changes reflected in Firebase
Profile page	Update available seats	Profile update successfully, changes reflected in Firebase
Profile page	Update available seats to a small amount while being in a group with bigger amount of drivers	Profile update failed, error message display that user needs to leave the specific group before updating available seats.
Search page	Click on the search button on the search page to show all groups available.	Search page displays all the groups that were created.
Search page	Search by specific filters like schedule, meeting points or name.	Search page displays all the groups that meet the filters criteria.
Search page	Search for a group name that doesn’t exist.	Search page displays an error message “No rides found”.
Create group page	Create a group with all required information.	Ride group opened successfully, changes reflected in Firebase and a notification was sent.
Create group page	Create a group with missing information like group name.	An error message displayed: “This field is required”.
Create group page	Create a group with a higher amount of available seats than the user has in his profile.	The application blocks the user from adding more seats than his available seats.
Create group page	Create a group with return time earlier than the departure time	An error message displayed.
Notifications page	Click trash icon to delete all the notifications	The notifications page being cleared and “No notifications found” displayed.

Notifications page	Click the “Mark as read” icon.	The notifications page shows all the notifications as “read” status.
Group page	Show all the information relevant to the group	The group page shows all the information of the group such as meeting points, times, members and map.
Group page	Join a group through the menu.	Select a pickup point popup appears, then user choose and a display message appears about joining a group. Changes reflected in Firebase and a notification was sent.
Group page	Join a group with a higher number of available seats than the user has in his profile.	An error message appears: “You can’t join a group because you don’t have enough available seats”.
Group page	Change a pickup point through the dropdown.	A new pickup point being set for the user. Changes reflected in Firebase and a notification was sent.
Group page	Leave a group through the menu	The user left the group successfully. Changes reflected in Firebase and a notification was sent.
Group page	Report user through the menu	Creating a user report. The user will be removed from the group in case there is a majority of votes. Send a notification when the user leaves.
Group page	Clicking on a user’s name in members table	Opens the user details to make a quick call.
Points system	The driver clicks the end drive button	The driver receives a point. Changes reflected in Firebase and a notification was sent.
Points system	The driver clicks end drive much earlier than the return time	The application displays an error message “You can’t click the end drive until 10 mins before the return time”.

Point system	The application randomize a driver when all drivers have the same amount of points	The application is successfully choosing a random driver when all have the same amount of points.
Points system	The application chooses the driver with the lowest amount of points.	The application is successfully choosing a driver with the lowest amount of points.

Chapter 5
User Guide

The Carpool app begins with the authentication screen. Here, users are presented with two options: they can either sign in to an existing account or create a new one to join the carpooling community.

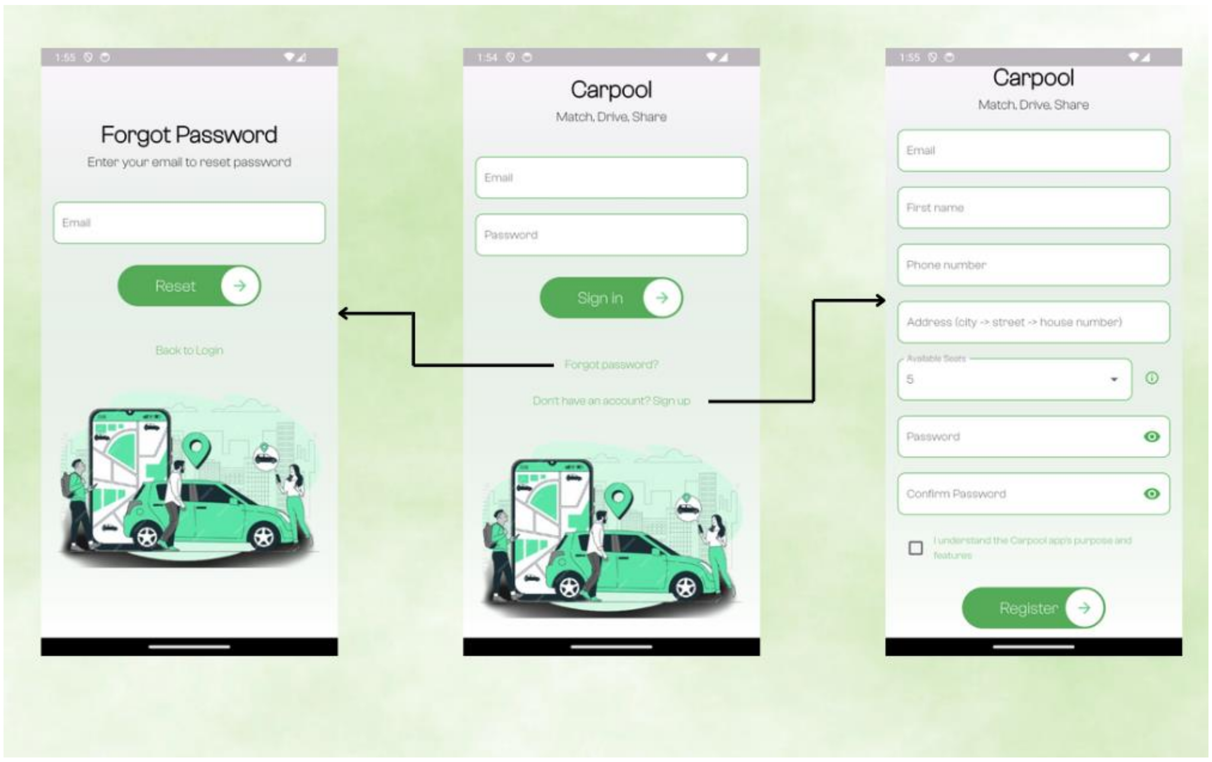


Figure 5.1: *Authentication screen of the carpool app, allowing users to sign in or register.*

After successful authentication, users are directed to the app's home page. This central hub offers access to key features of the Carpool app. From here, users can manage their profile, search for available carpool groups, view their current groups, check notifications, and create new ride groups. To enhance user navigation and provide quick access to core functionalities, we implemented a bottom navigation bar. This feature ensures users can easily switch between main sections of the app, creating a seamless and intuitive carpooling experience.

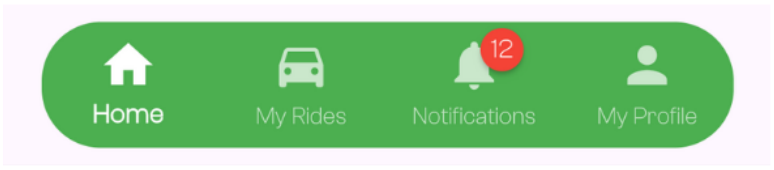


Figure 5.2: *Navigation bar.*

The Carpool app's navigation bar features five key tabs (Home, My Rides, Notifications, My Profile), each offering quick access to essential functions:

1. Home: The app's central hub serves as the main dashboard for users' carpooling activities. Upon opening the app, users are greeted with a personalized overview of their carpooling world. The home screen features:
    - Quick-access buttons for finding a ride or creating a new one, streamlining the user's ability to engage in carpooling activities.
    - A clean, intuitive interface that prioritizes the most common user actions.

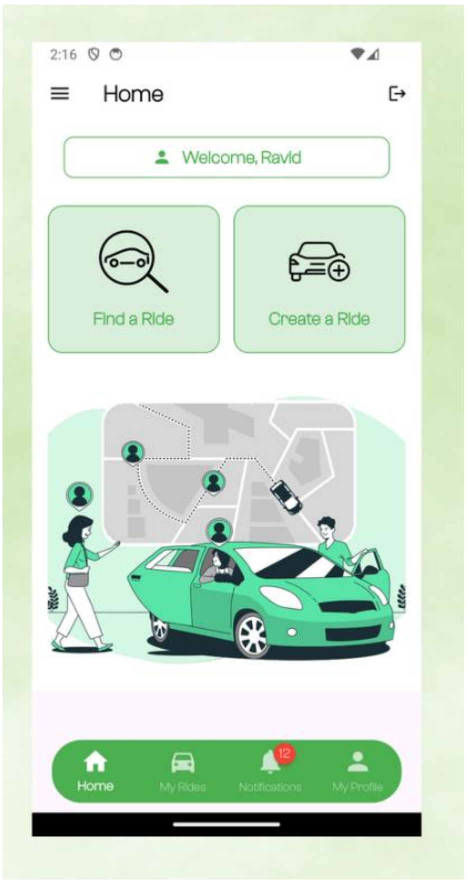


Figure 5.3*: Home screen.*

2. My Rides: This comprehensive ride management center allows users to view and organize all their carpooling commitments. Key features include:
    - A clear view of all scheduled rides the user has joined, both as a driver and as a passenger.
    - Detailed information for each ride, including the route, the next assigned driver, and the complete schedule.
    - Easy-to-read layout that helps users quickly understand their upcoming carpooling responsibilities.

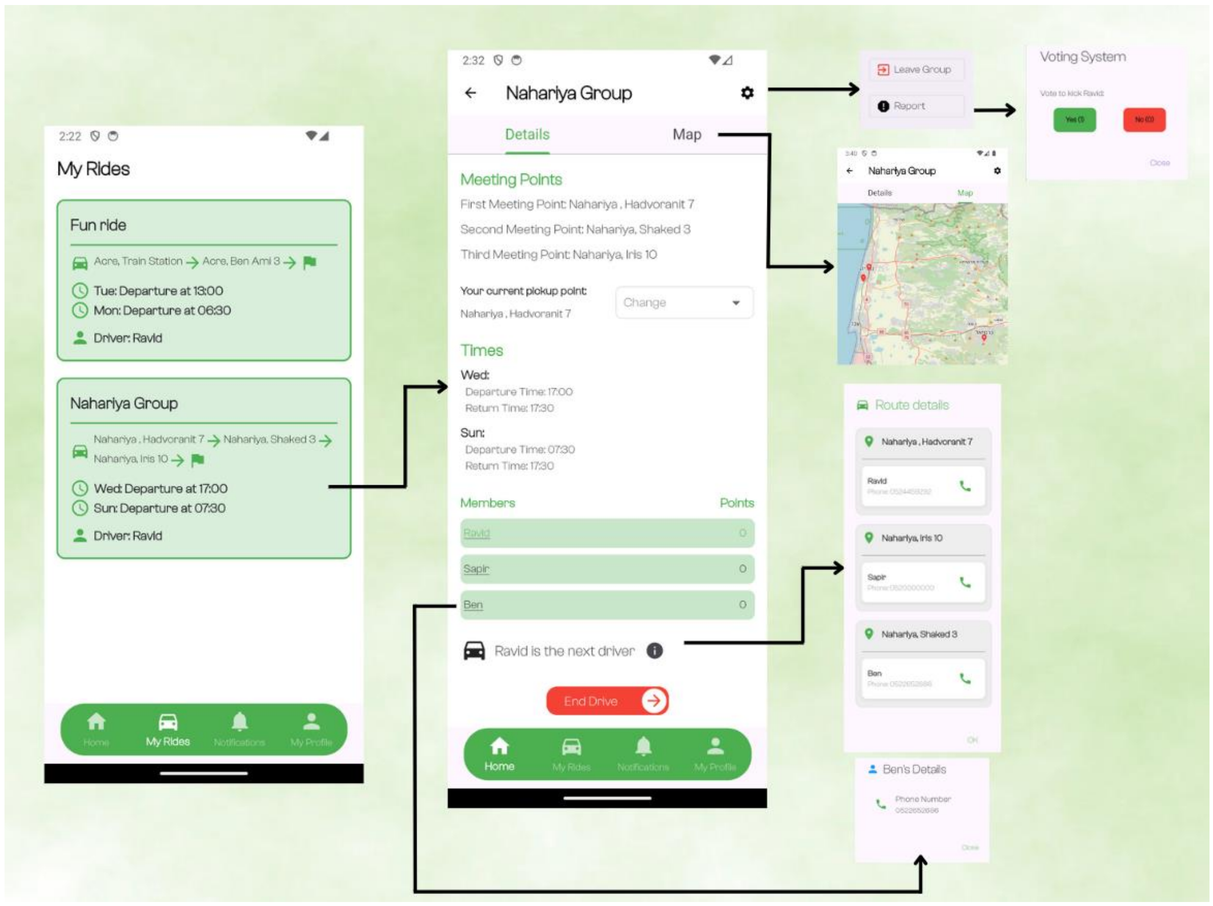


***Figure 5.4**: Detailed view of the "My Rides" section, showcasing the group ride details, route map, and voting system.*

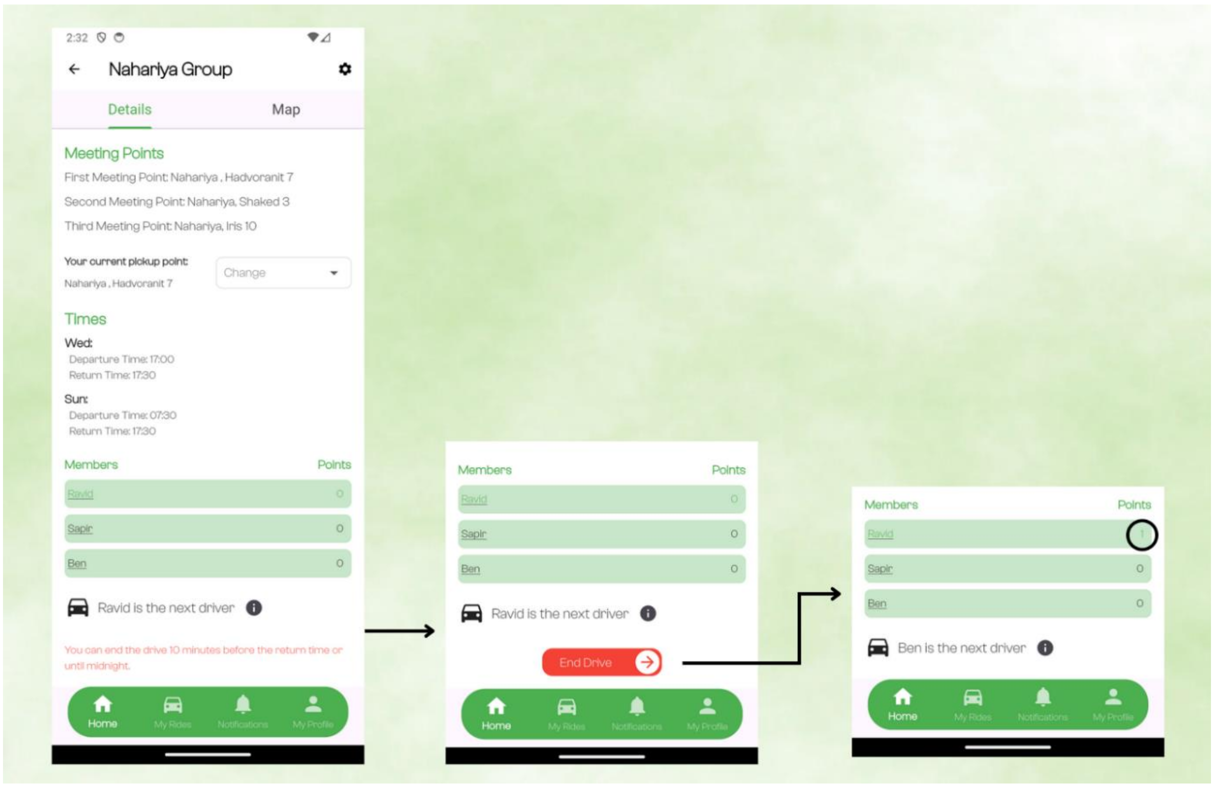
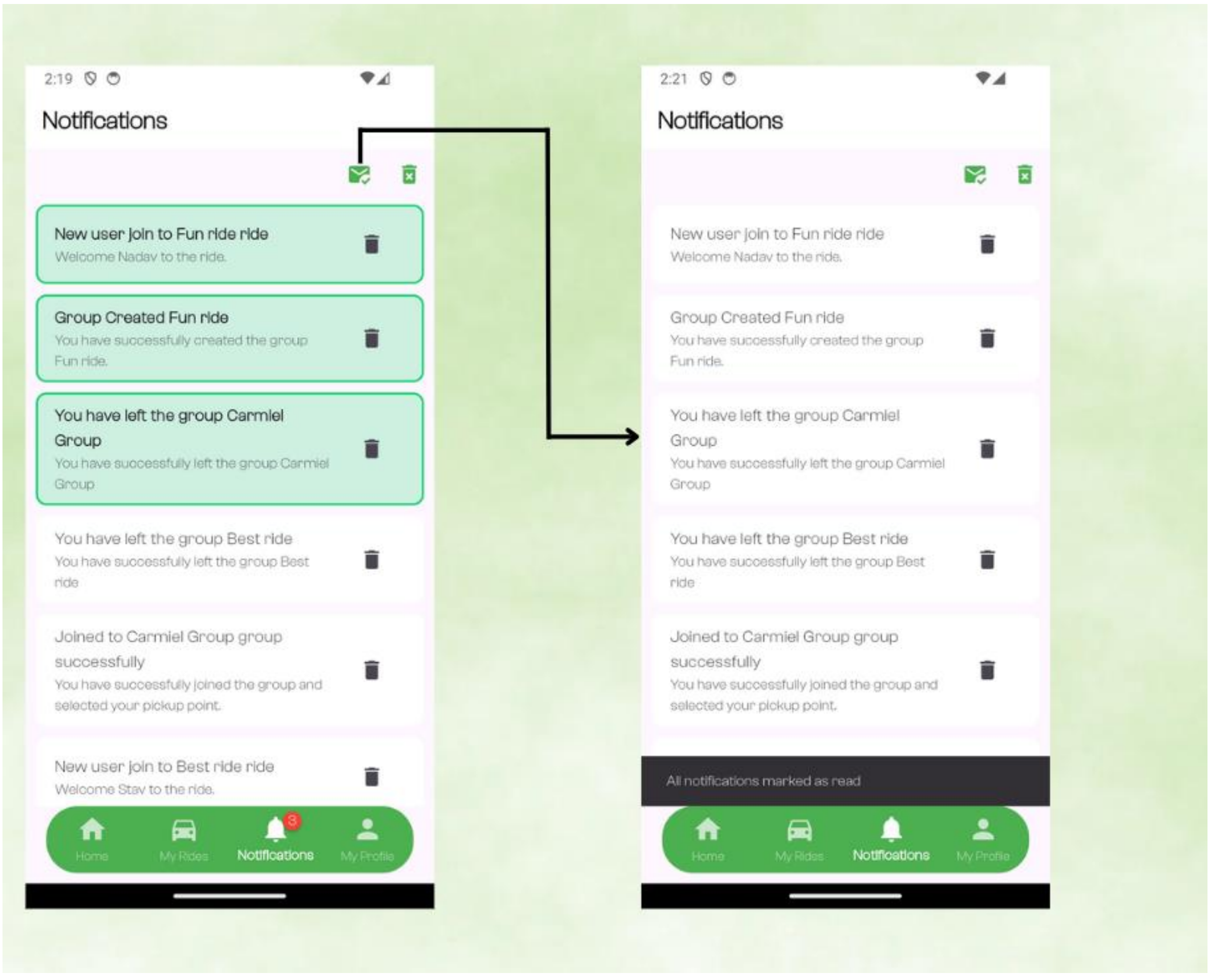


***Figure 5.5**: Navigation within a group ride, displaying member details, driver assignment, and completion of a ride.*

3. Notifications: This centralized communication center keeps users informed about all aspects of their carpooling experience. It provides:

- Real-time alerts for group-related actions such as joining or leaving a group, and notifications for the start and end of each drive.
- Timely updates about points received, reinforcing the app's fair usage system.
- Prompt notifications about any changes in pickup points, ensuring users are always informed about their ride details.


2:19

Notifications

New user join to Fun ride ride
Welcome Nadav to the ride.

Group Created Fun ride
You have successfully created the group Fun ride.

You have left the group Carmiel Group
You have successfully left the group Carmiel Group

You have left the group Best ride
You have successfully left the group Best ride

Joined to Carmiel Group group successfully
You have successfully joined the group and selected your pickup point.

New user join to Best ride ride
Welcome Stav to the ride.

2:21

Notifications

New user join to Fun ride ride
Welcome Nadav to the ride.

Group Created Fun ride
You have successfully created the group Fun ride.

You have left the group Carmiel Group
You have successfully left the group Carmiel Group

You have left the group Best ride
You have successfully left the group Best ride

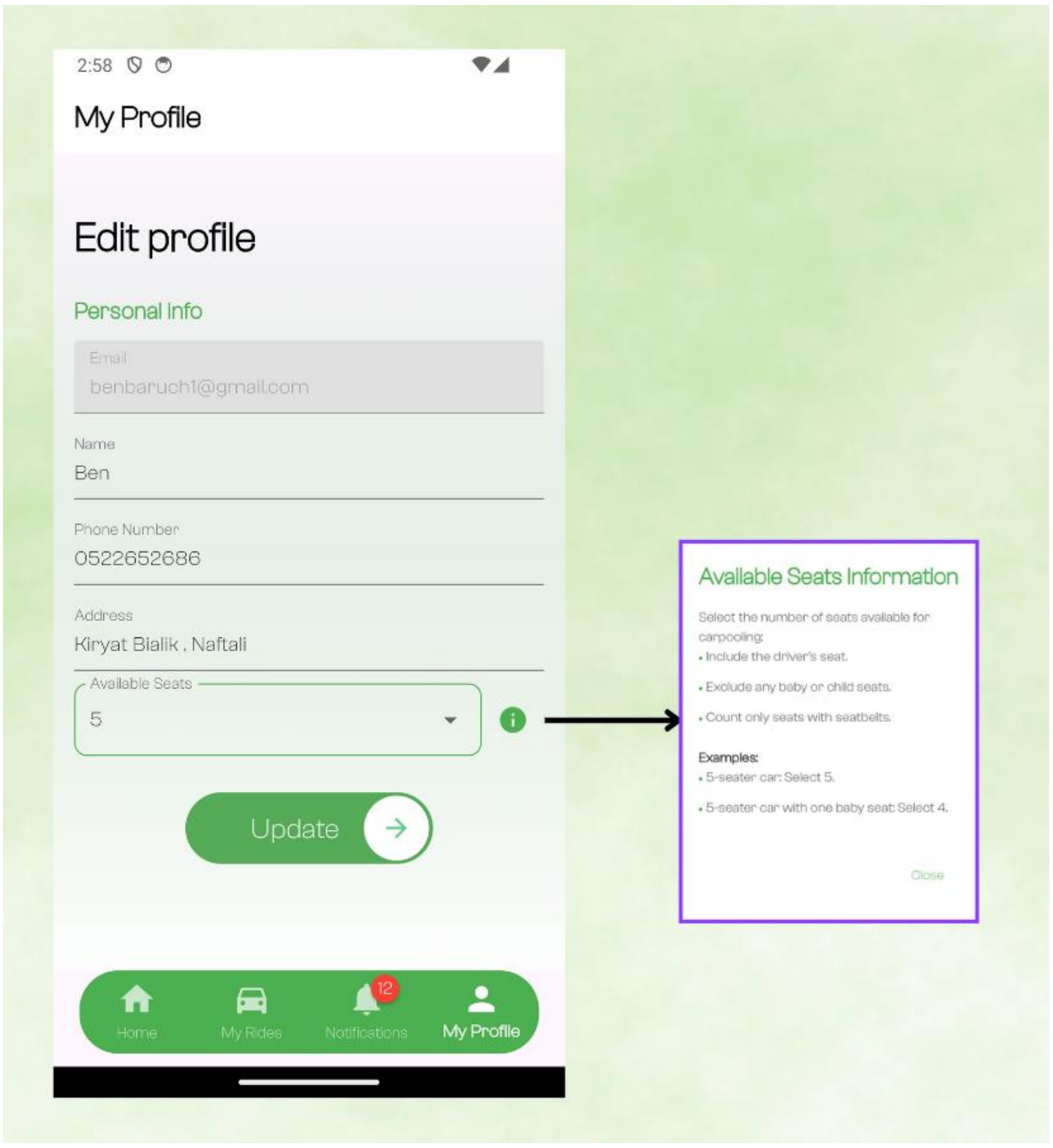
Joined to Carmiel Group group successfully
You have successfully joined the group and selected your pickup point.

All notifications marked as read

***Figure 5.6**: Notifications center, providing real-time alerts and updates on group-related actions, points received, and changes in ride details.*

4. My Profile: This personal control center allows users to manage their account and tailor their carpooling experience. Users can:

- Edit personal information including name, address details, and phone number, ensuring their profile remains up-to-date.
- Manage available seat information, crucial for determining which groups they can join based on their vehicle capacity.


2:58

My Profile

Edit profile

Personal Info

Email
benbaruch1@gmail.com

Name
Ben

Phone Number
0522652686

Address
Kiryat Bialik , Naftali

Available Seats
5

Available Seats Information

Select the number of seats available for carpooling:
- Include the driver's seat.
- Exclude any baby or child seats.
- Count only seats with seatbelts.

Examples:
- 5-seater car: Select 5.
- 5-seater car with one baby seat: Select 4.

Figure 5.7: *My Profile section, enabling users to edit personal information and manage available seat capacity for carpooling.*

In the home screen page, there are two main buttons of core functionalities of the application. Find a ride and create a ride button.

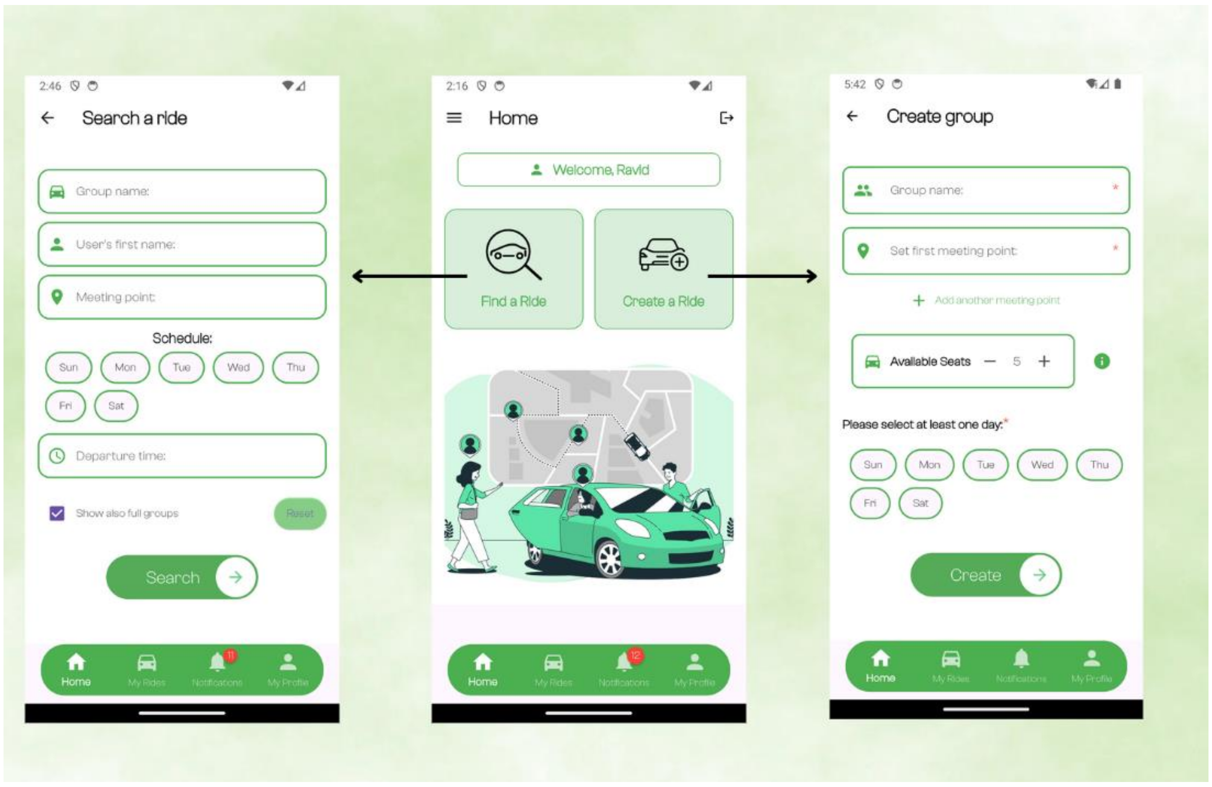
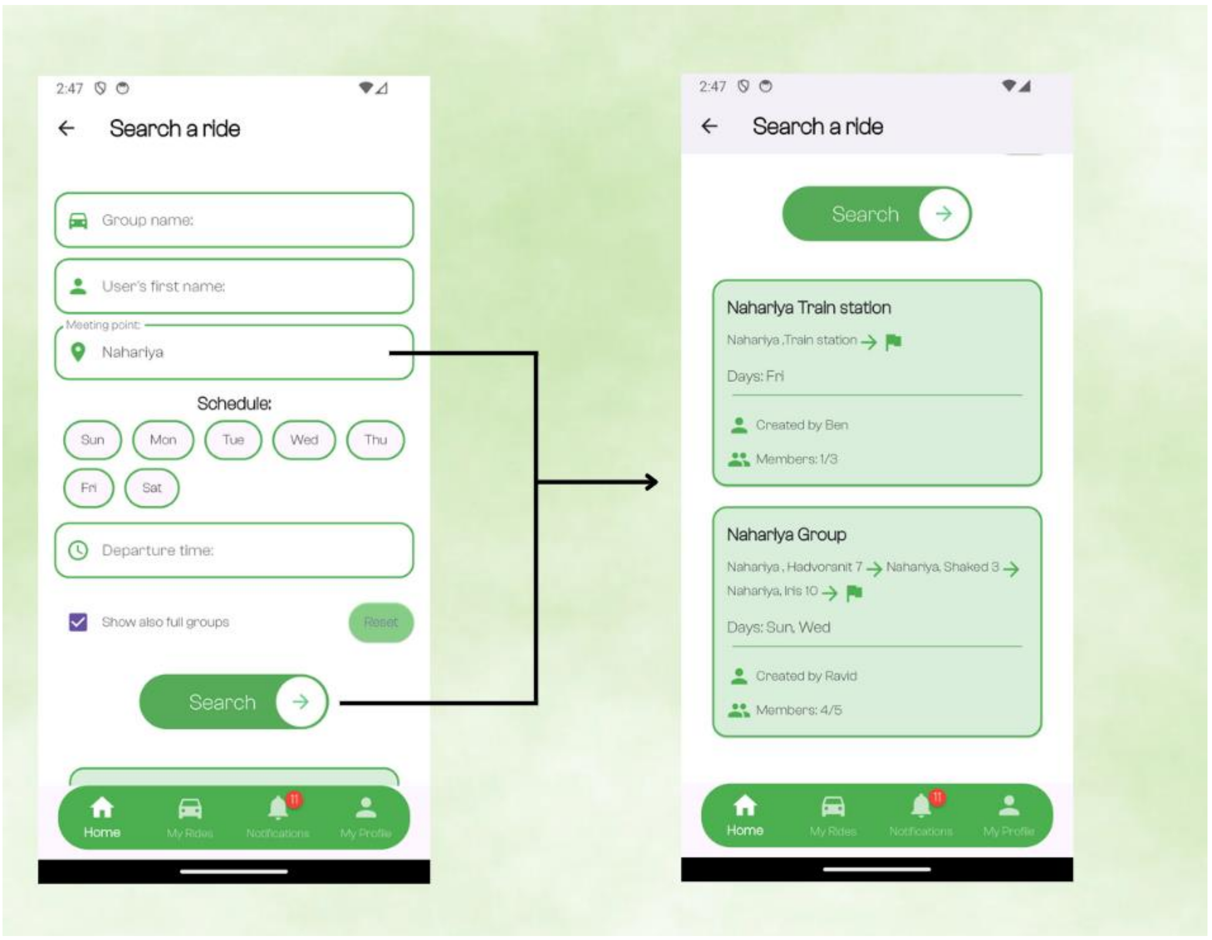


***Figure 5.8**: Core functionalities of the carpool app on the home screen, featuring the "Find a Ride" and "Create a Ride" buttons.*

Find a Ride

The "Find a Ride" feature allows users to find existing carpool groups that match their needs. This powerful search tool offers flexibility and precision in locating suitable rides.



Meeting point: Nahariya

Nahariya Train station
Nahariya ,Train station
Days: Fri
Created by Ben
Members: 1/3

Nahariya Group
Nahariya , Hadvorsnit 7 → Nahariya, Shaked 3 → Nahariya, Iris 10
Days: Sun, Wed
Created by Ravid
Members: 4/5

Figure 5.9: *The "Find a Ride" feature, showcasing the search tool for locating carpool groups based on specific criteria.*

Key features of the Find a Ride page include:

- **Group Name**: Users can search for specific groups by entering the group name.
- **User's First Name**: This option allows searching for groups that include a particular user.
- **Meeting Point**: Users can find groups based on convenient pickup locations.
- **Schedule**: Search for groups that align with your preferred carpooling days and times.
- **Show Full Groups**: A checkbox option to include or exclude groups that are already at full capacity.

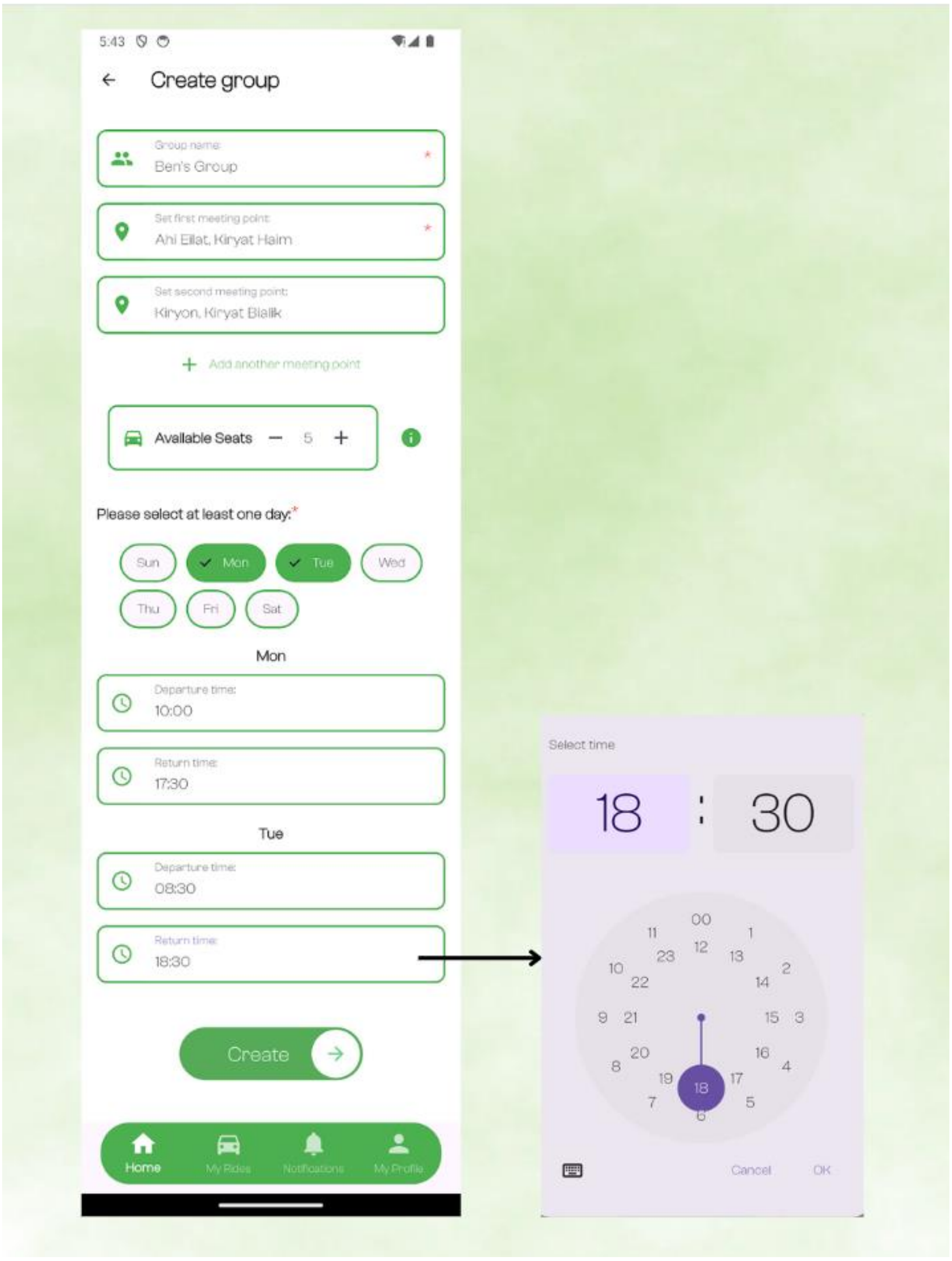
To use the search function:

1. Enter desired search criteria in any of the provided fields.
2. Check the "Show Full Groups" box if you want to see all available groups, including those at capacity.
3. Click the "Search" button to view results.

If the user clicks "Search" without entering any criteria, the app will display all available groups, giving a comprehensive view of carpooling options in your area.

Create a Ride

The "Create a Ride" feature empowers users to start their own carpool groups, tailored to their specific needs and schedules.


5:43

Create group

Group name: Ben's Group *

Set first meeting point: Ahi Eilat, Kiryat Haim *

Set second meeting point: Kiryon, Kiryat Bialik

Add another meeting point

Available Seats — 5 +

Please select at least one day:*

Sun ✓ Mon ✓ Tue Wed Thu Fri Sat

Mon

Departure time: 10:00

Return time: 17:30

Tue

Departure time: 08:30

Return time: 18:30

Create

Home My Rides Notifications My Profile

Select time

18 : 30

__Figure 5.10__: The "Create a Ride" feature, allowing users to start and customize their own carpool groups, including meeting points, available seats, and schedules.

When creating a new ride group, users need to provide the following information:

1. **Group Name**: Choose a unique and descriptive name for your carpool group.
2. **Meeting Points**: Set up to three pickup locations for your group.
    - You must specify at least one meeting point.
    - Each additional meeting point allows for more flexibility in your carpool route.
3. **Available Seats**: Indicate how many passengers your vehicle can accommodate.
    - This number must not exceed the available seats you've listed in your profile.

    - Accurate seat information ensures proper group matching and capacity management.
4. **Schedule**: Define your carpooling routine.
    - Select at least one day of the week for your carpool.
    - For each selected day, specify the time(s) for your rides.
    - You can set multiple time slots per day if needed (e.g., morning and evening rides).

To create a new ride group:

1. Fill in all required fields with accurate information.
2. Review your entries to ensure they reflect your intended carpool arrangement.
3. Click the "Create Group" button to finalize and activate your new carpool group.

Remember, creating a well-defined group helps attract the right carpooling partners and ensures a smooth, organized carpooling experience for all participants.

Chapter 6
Maintenance Guide

6.1 Requirements

Software Requirements

- Flutter SDK 2.8.0 or later
- Dart 2.15.0 or later
- Android Studio 4.2 or later (for Android development)
- Xcode 13 or later (for iOS development)
- VS Code or IntelliJ IDEA (optional, but recommended)

Hardware Requirements

- Computer with at least 8GB RAM and a dual-core processor
- Android or iOS device for testing (optional, but recommended)

6.2 Installation

1. Clone the repository:
   ```
   git clone https://github.com/benbaruch1/carpool
   ```

2. Navigate to the project directory:
   ```
   cd carpool_app
   ```

3. Get the dependencies:
   ```
   flutter pub get
   ```

4. Open the project in your preferred IDE (Android Studio, VS Code, or IntelliJ IDEA).

5. Ensure you have set up an emulator or connected a physical device.

6. Run the app:
   ```
   flutter run
   ```

7. The application should launch on the device or emulator, and you should be able to interact with it.

You're all set!

6.3 Firebase

For updating Firebase configuration or database structure, please contact the project administrator at benbaruch1[at]gmail.com. Include details about the update you wish to make in your email. If needed, a permission will be granted.

6.4 Contribution

We welcome and appreciate any contributions to improve the Carpool app. If you have a suggestion or improvement, please follow these steps:

1. Fork the repository.
2. Create your feature branch (e.g., git checkout -b feature/your-feature-name).
3. Make your changes and commit them (git commit -am 'Add some feature').
4. Push to the branch (git push origin feature/your-feature-name).
5. Create a new Pull Request.

Alternatively, you can open an issue with the tag "enhancement" to suggest improvements without coding them yourself.
All contributions will be reviewed and, if approved, merged to enhance the application.

6.5 Contact

- Email: benbaruch1@gmail.com or ravidp90@gmail.com
- GitHub: https://github.com/benbaruch1/carpool

For any questions, issues, or support needs, please don't hesitate to reach out through the provided contact information.

References

[1] Yoshifumi Konishi and Akari Ono (2024), Is Ride-sharing Good for Environment? - https://econpapers.repec.org/paper/keodpaper/2024-014.htm

[2] Shaheen, Susan, PhD Cohen, Adam, MCRP Bayen, Alexandre, PhD (2018), The Benefits of Carpooling -
https://escholarship.org/uc/item/7jx6z631

[3] Mark Mather (2018), Most U.S. Workers Still Driving Alone -
https://www.prb.org/resources/commuting-most-u-s-workers-still-driving-alone/

[4] Flutter, an open-source framework that allows for cross-platform development with a single codebase -
https://flutter.dev/

[5] Firebase, a comprehensive app development platform that provides tools and infrastructure for backend services, including real-time data storage, authentication, and cloud functions –
https://firebase.google.com/

[6] OpenStreetMap, a collaborative project to create a free, editable map of the world, used for route visualization and mapping functionality in applications -
https://www.openstreetmap.org

[7] Model-View-Controller (MVC), a software architectural pattern for implementing user interfaces, which divides an application into three interconnected components -
https://developer.mozilla.org/en-US/docs/Glossary/MVC

[8] Flutter Map, A versatile mapping package for Flutter -
https://github.com/fleaflet/flutter_map