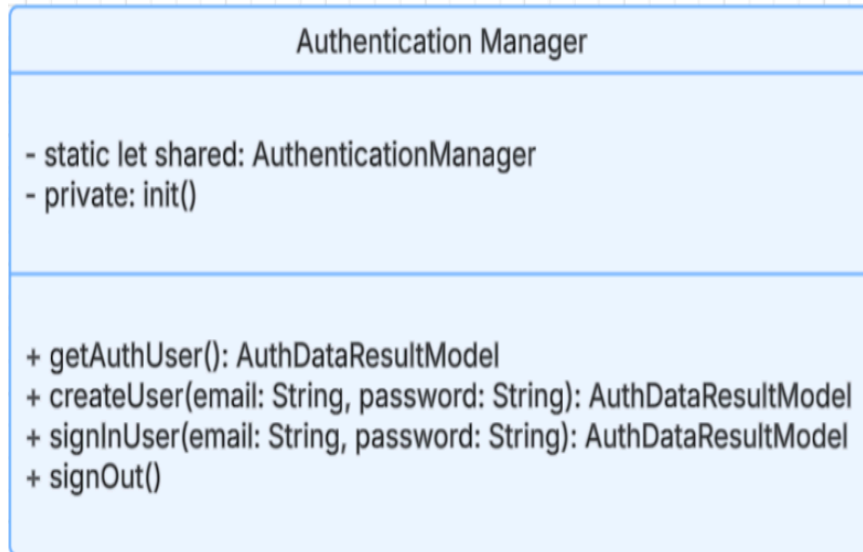


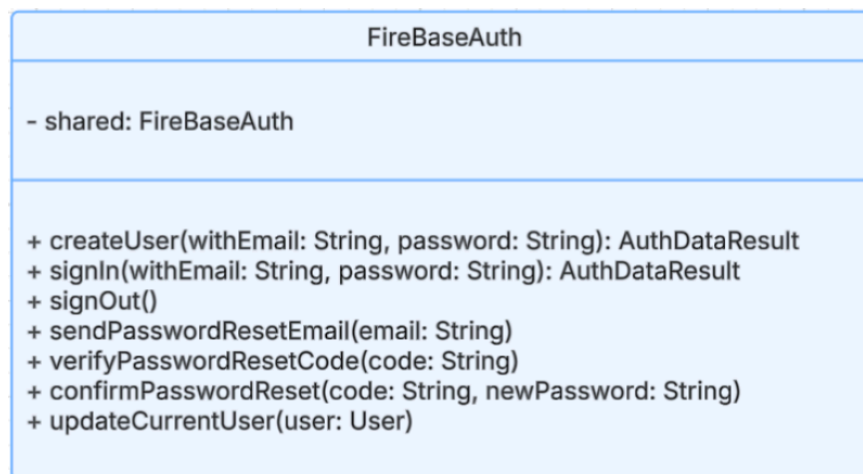
Class Diagrams and Descriptions:

Singleton Pattern Implementation:



Description:

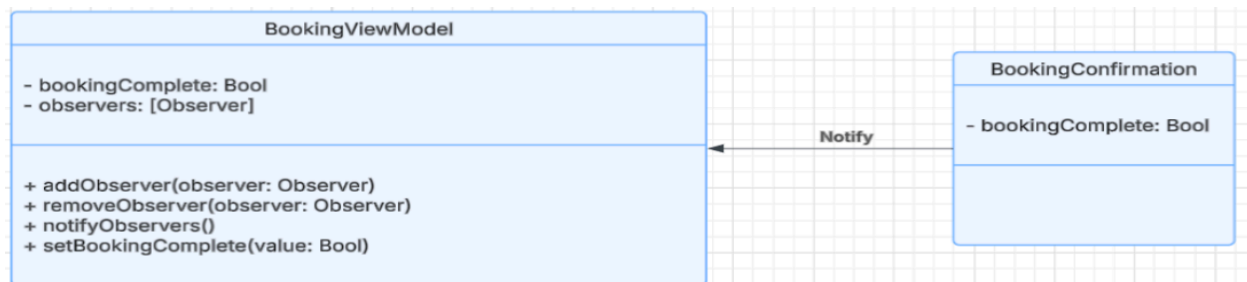
The AuthenticationManager class follows the Singleton design pattern, ensuring that only one instance is created and shared throughout the application. This class provides methods for user authentication, including retrieving the current authenticated user, creating an account, signing in, and signing out. The private init() prevents external instantiation, while the shared instance allows global access.



Description:

The FirebaseAuth class provides an abstraction layer for handling user authentication using Firebase. It includes methods for creating a user (createUser), signing in (signIn), and signing out (signOut). Additionally, it supports password management with functionality for sending password reset emails (sendPasswordResetEmail), verifying reset codes (verifyPasswordResetCode), and confirming password resets (confirmPasswordReset). The class also allows for updating the current user information through the updateUser method. The shared instance ensures a single point of access for these authentication operations.

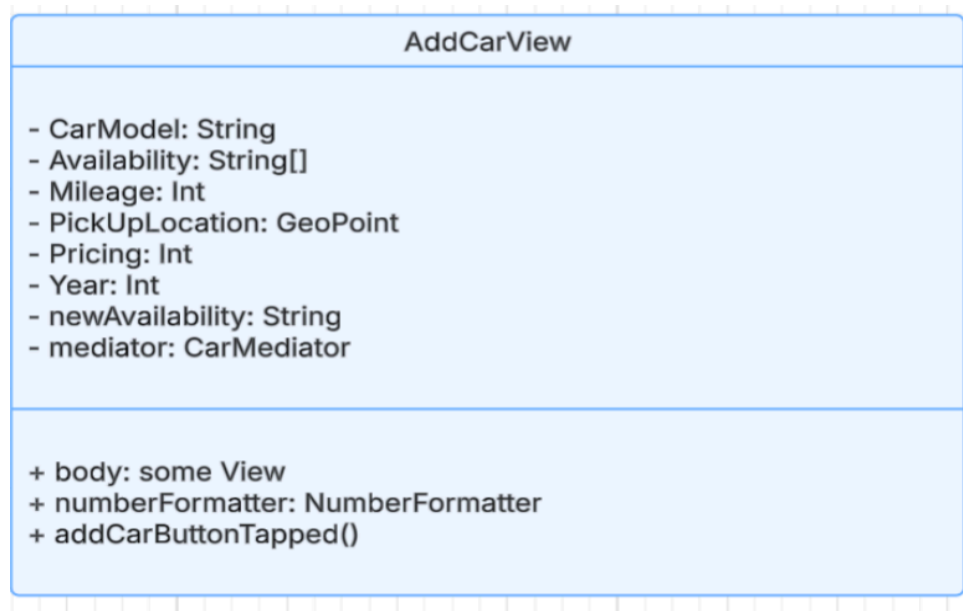
Observer Pattern Implementation:



Description:

The BookingViewModel class implements the Observer design pattern, acting as the Subject that manages a list of observers and notifies them when the bookingComplete state changes. It provides methods like `addObserver()`, `removeObserver()`, and `notifyObservers()`, and uses `setBookingComplete()` to update the state and trigger notifications. The BookingConfirmation class is a ConcreteObserver that listens to changes in bookingComplete via the `update()` method and updates the UI with booking details when the state is set to true. The Observer protocol defines the `update()` method, enabling observers like BookingConfirmation to respond to state changes in BookingViewModel.

Mediator Pattern Implementation:



Description:

AddCarView represents the user interface (UI) component that allows the user to input car details in the application. It is a View in SwiftUI that includes form fields for the car model, availability, mileage, pickup location, pricing, and year. This class provides the UI for users to interact with and fill in the necessary data to add a car.

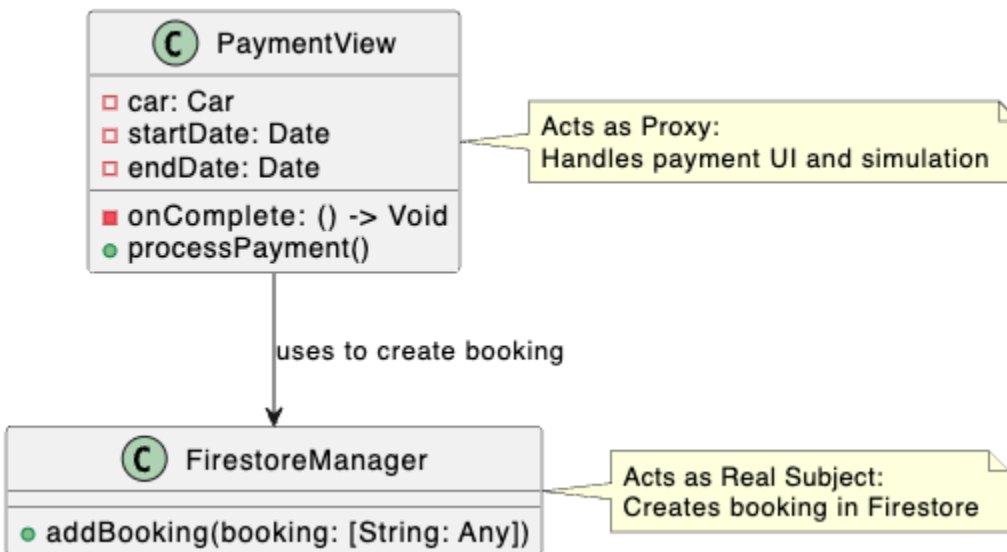
Builder Pattern Implementation:



Description:

In the DriveShare application, the Builder Pattern is employed to streamline the creation of car listings. The AddCarView assumes the role of the director, systematically gathering critical attributes of a car listing—such as model, availability dates, mileage, pickup location, pricing, and year—via user input fields. Upon the user's command to finalize the addition, these attributes are transmitted to the FirestoreManager, which operates as the builder. The FirestoreManager subsequently constructs the car listing object or generates a corresponding document within the Firestore database. This architectural approach distinctly separates the responsibility of data collection from the process of object construction.

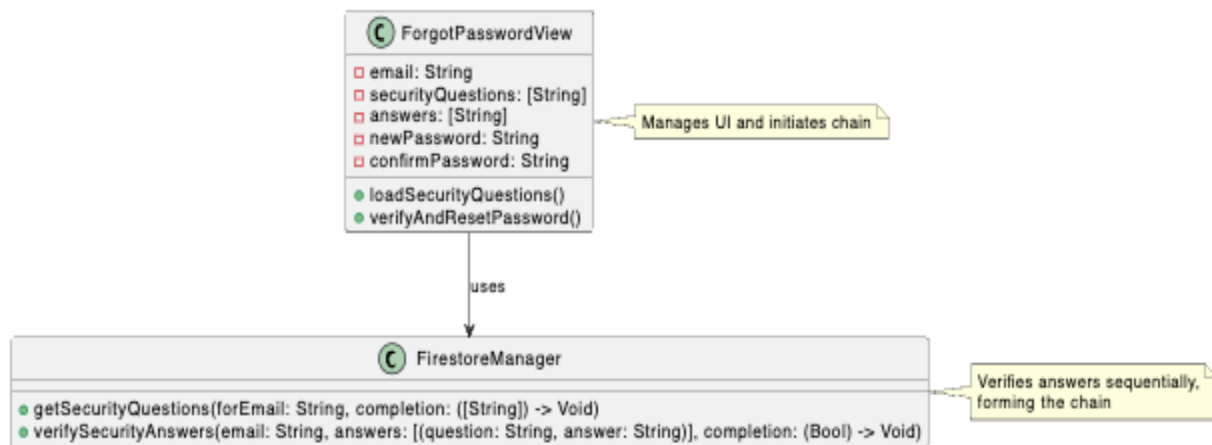
Proxy Pattern Implementation:



Description:

The Proxy Pattern is applied within the payment integration framework of the DriveShare application. The PaymentView serves as a proxy, overseeing the user interface dedicated to payment processing and simulating the transactional workflow. It is tasked with collecting and validating essential payment details—including card number, cardholder name, expiry date, and CVV—prior to authorizing further action. Upon successful validation, the PaymentView delegates the creation of a booking to the FirestoreManager, which executes the requisite database operations. This abstraction establishes a controlled intermediary layer, ensuring that access to the booking system is mediated through the proxy. Consequently, bookings are instantiated only after the payment simulation has been satisfactorily completed, reinforcing security.

Chain of Responsibility Pattern Implementation:



Description:

The Chain of Responsibility Pattern governs the password recovery mechanism within the DriveShare application. The **ForgotPasswordView** orchestrates this process, initiating the sequence with email verification to retrieve a set of predefined security questions from the **FirestoreManager**. The user is subsequently required to provide accurate responses to these questions, with each answer subjected to sequential verification. This establishes a chain-like structure wherein each verification step must be successfully completed to advance to the next, culminating in the authorization of a password reset only upon the correct resolution of all questions.

Database Schemas and Descriptions:

Bookings	
carId	String
carModel	String
endDate	Date
isUserOwner	Boolean
ownerId	String
renterId	String
startDate	Date
status	String
timestamp	Date
totalPrice	Int

Description:

This database schema handles the Bookings aspect of our driveshare application on the backend side of things. In this schema as you can see, each car has an ID which is stored as a string, each car has a model which is also stored as a string, there is an end and start date which is of course represented by a date. There is a check for true or false whether or not the user is a owner of the car, the owner and renter has an ID stored as a string, status of the car is stored as a string, the timestamp has a date and of course the total price of the car is stored as an int.

carList	
Availability	String[]
CarModel	String
Mileage	Int
PickUpLocation	Coordinate
Pricing	Int
Year	Int
userId	String

Description:

This database schema handles the car listings aspect of our driveshare application on the backend side of things. In this schema as you can see, availability of cars is stored as an array of strings. The car model and id of the user are stored as strings. The mileage of the car, the price and year of the car all are stored in the database as integers. The pick up location of the car is stored as a coordinate.

Conversations	
carModel	String
lastMessage	String
lastMessageTimeStap	Date
isUserOwner	String[]
ownerId	String
renterId	Int

Description:

This database schema handles the conversations aspect of our driveshare application on the backend side of things. In this schema as you can see, the model of the car and the id of the owner are both stored as String variables. The last message is stored as a string and the user owner Ids are stored in an array String. The last message time stamp is also stored as a date. The ID of the renter is stored as an integer.

Messages	
content	String
isRead	Boolean
recieverId	String
relatedCarId	String
senderId	String
timestamp	Date

Description:

This database schema handles the messages aspect of our driveshare application on the backend side of things. In this schema as you can see, content is stored as a string. There is also a boolean that checks if the messages are read by users. The receiver of the car has an ID, the related car has an ID, and the sender has an ID all stored as strings. And finally the timestamp is stored as a date.

passwordRecoveryAttempts	
email	String
isSuccessful	Boolean
timestamp	Date

Description:

This database schema handles the password recovery attempts aspect of our driveshare application on the backend side of things. The email of the user is stored as a string. To check if the password recovery is successful, it is stored as a boolean variable. And again, similar to other database schemas in the project, timestamp is stored as date.

securityQuestions	
answer	String
question	String
questionIndex	Int
userId	String

Description:

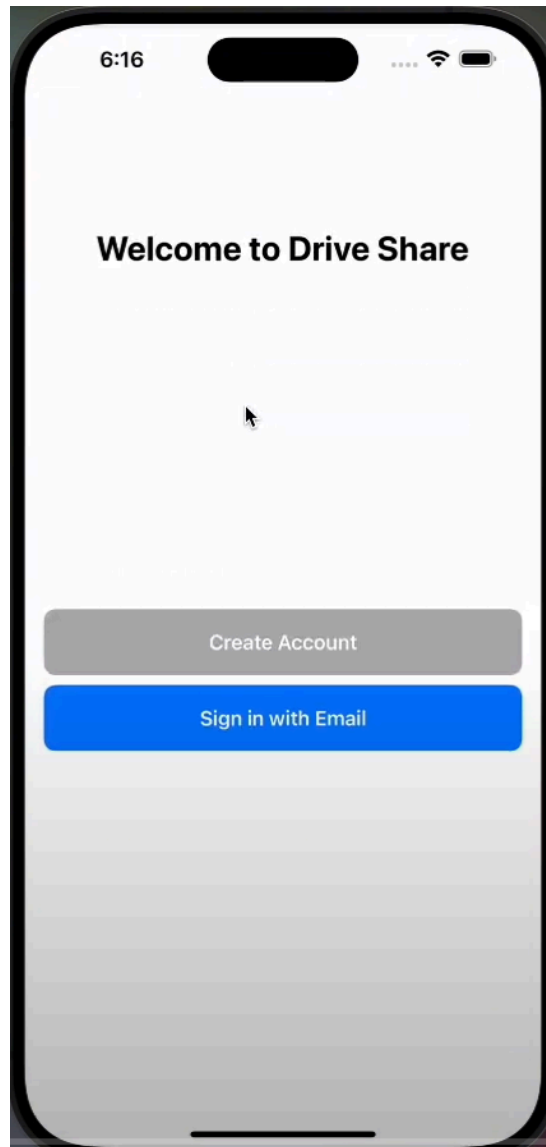
This database schema handles the security questions aspect of our driveshare application on the backend side of things. The answer to the security questions, the questions themselves, and the user ID, are all stored as a string. The question index is of course stored as an integer.

Reviews	
carId	String
comment	String
isOwnerReview	Boolean
rating	Int
recipientId	String
reviewerId	String
reviewerName	String
timestamp	Date
title	String

Description:

This database schema handles the reviews aspect of our driveshare application on the backend side of things. The ID of the car, the comments on the reviews, the IDs of the reviewer and recipients, and the review name and the title are all stored as strings. There is a boolean to check if the owner reviewed. The rating is stored as an integer and the timestamp is stored as a date.

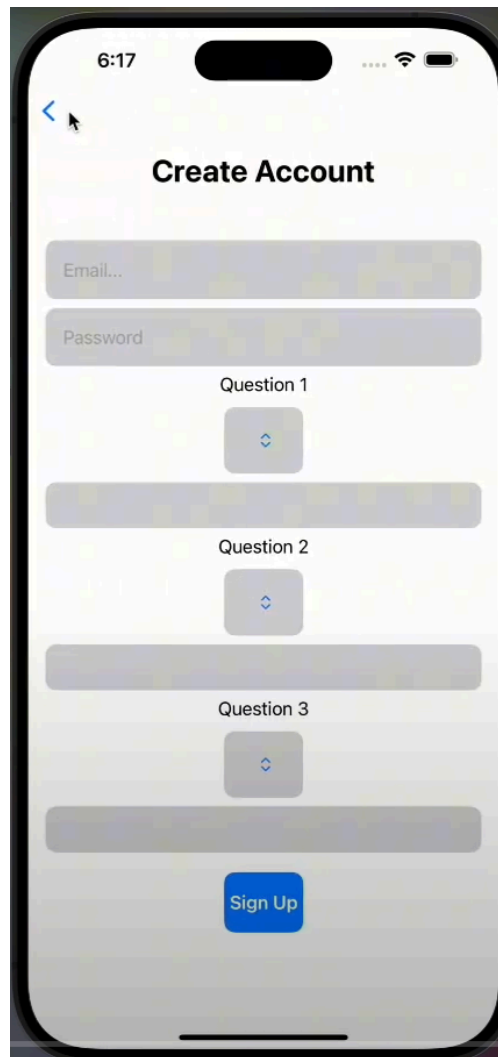
Welcome Screen:



Description:

This is the welcome screen of the application. It allows you to create an account or sign in with email.

Create Account Screen:



A mobile app mockup of a 'Create Account' screen. The screen is white with a black status bar at the top showing the time '6:17' and signal/battery icons. A blue back arrow is in the top left corner. The title 'Create Account' is centered in bold black text. Below the title are three input fields: 'Email...', 'Password', and a third unlabeled field. Each input field is followed by a 'Question' label and a dropdown menu with a blue double-headed arrow icon. The questions are 'Question 1', 'Question 2', and 'Question 3'. At the bottom is a blue 'Sign Up' button. The entire screen is framed by a black border.

6:17

<

Create Account

Email...

Password

Question 1

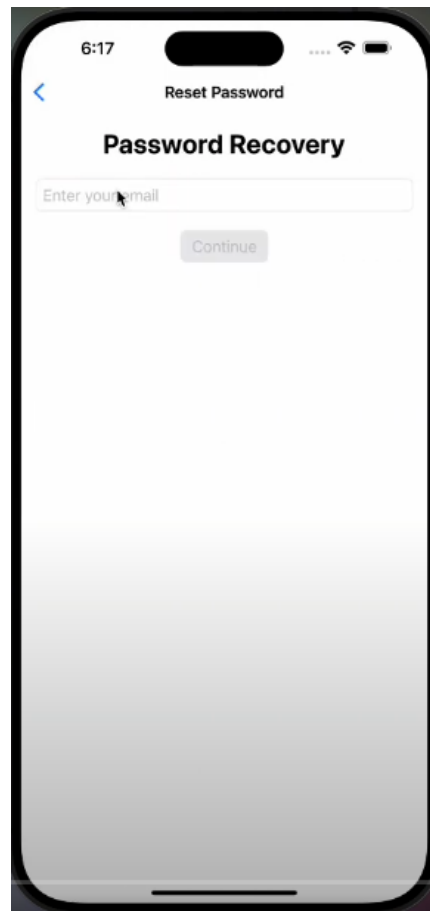
Question 2

Question 3

Sign Up

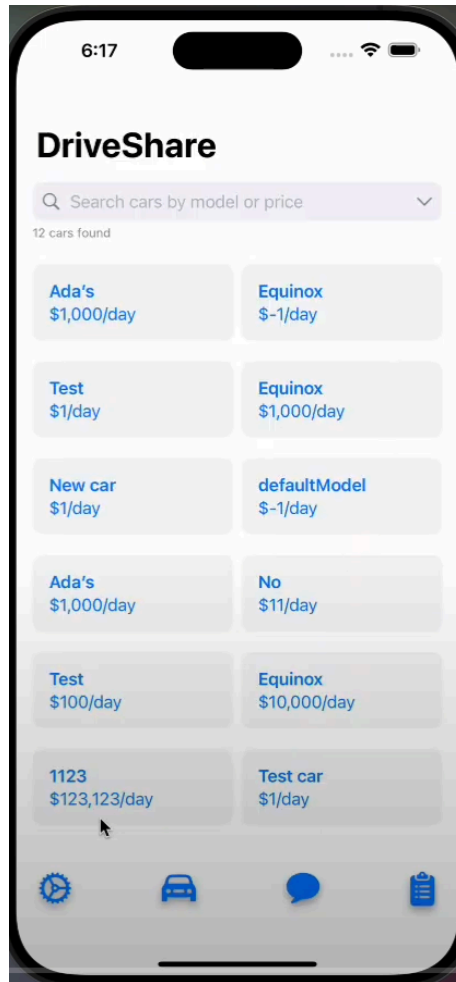
This is the screen that will let you create an account.

Password Recovery:



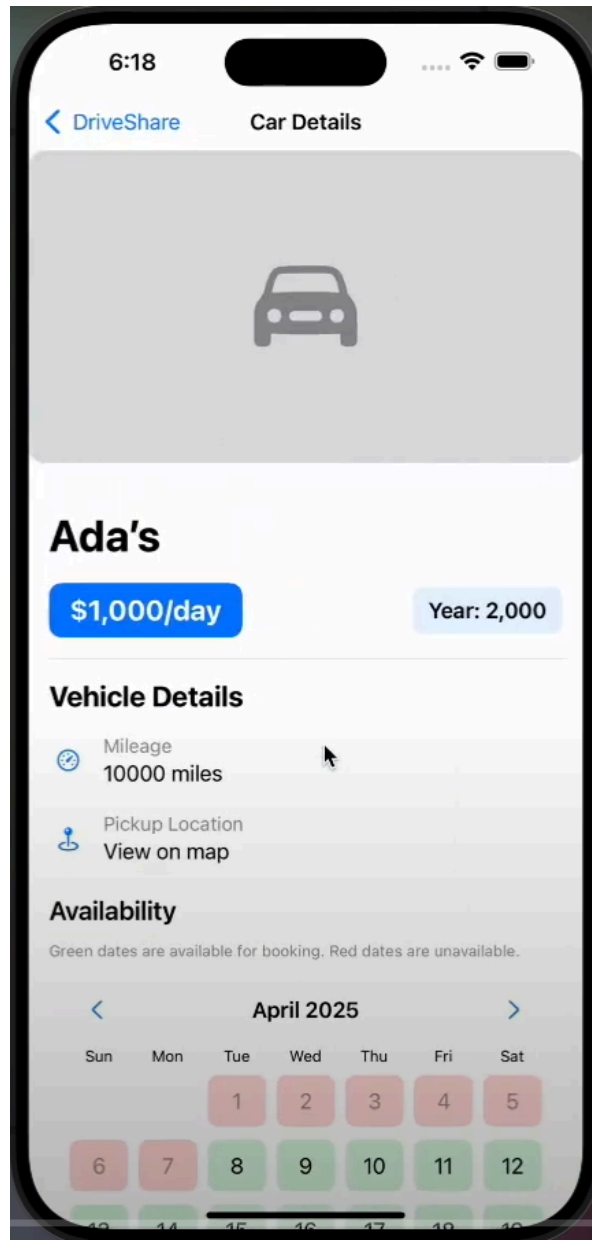
This allows you to recover your password.

DriveShare Screen:



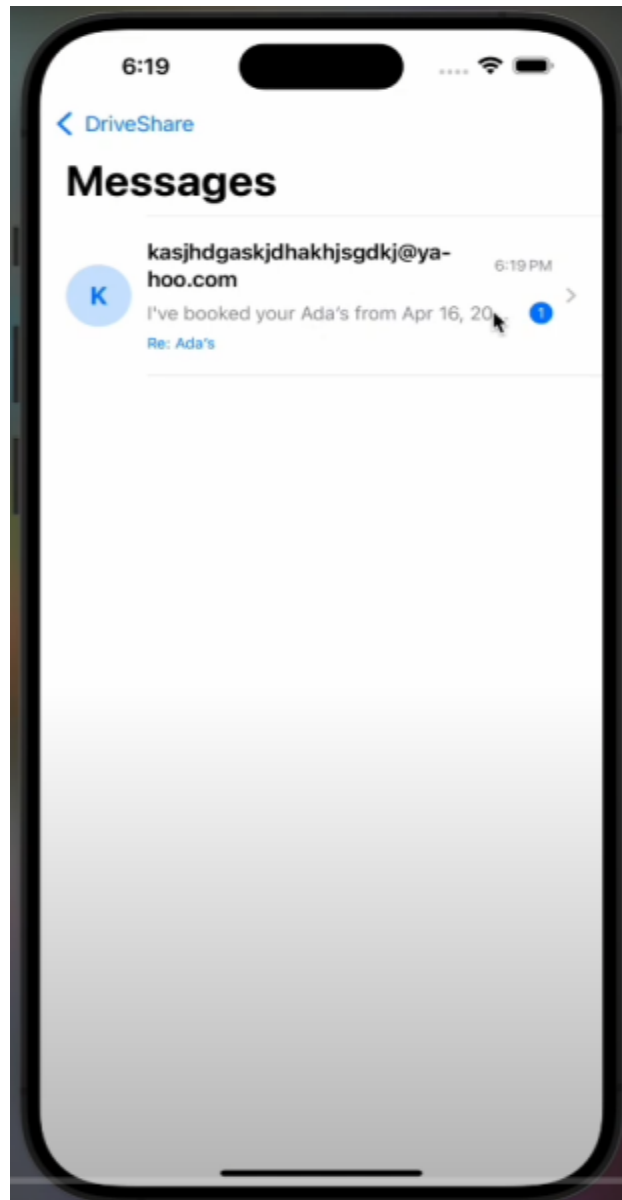
This screen is the main that will let you navage to the car view, messaging appointments, and settings. You can also view car listings here, it also lets you search the car listings.

Car Details:



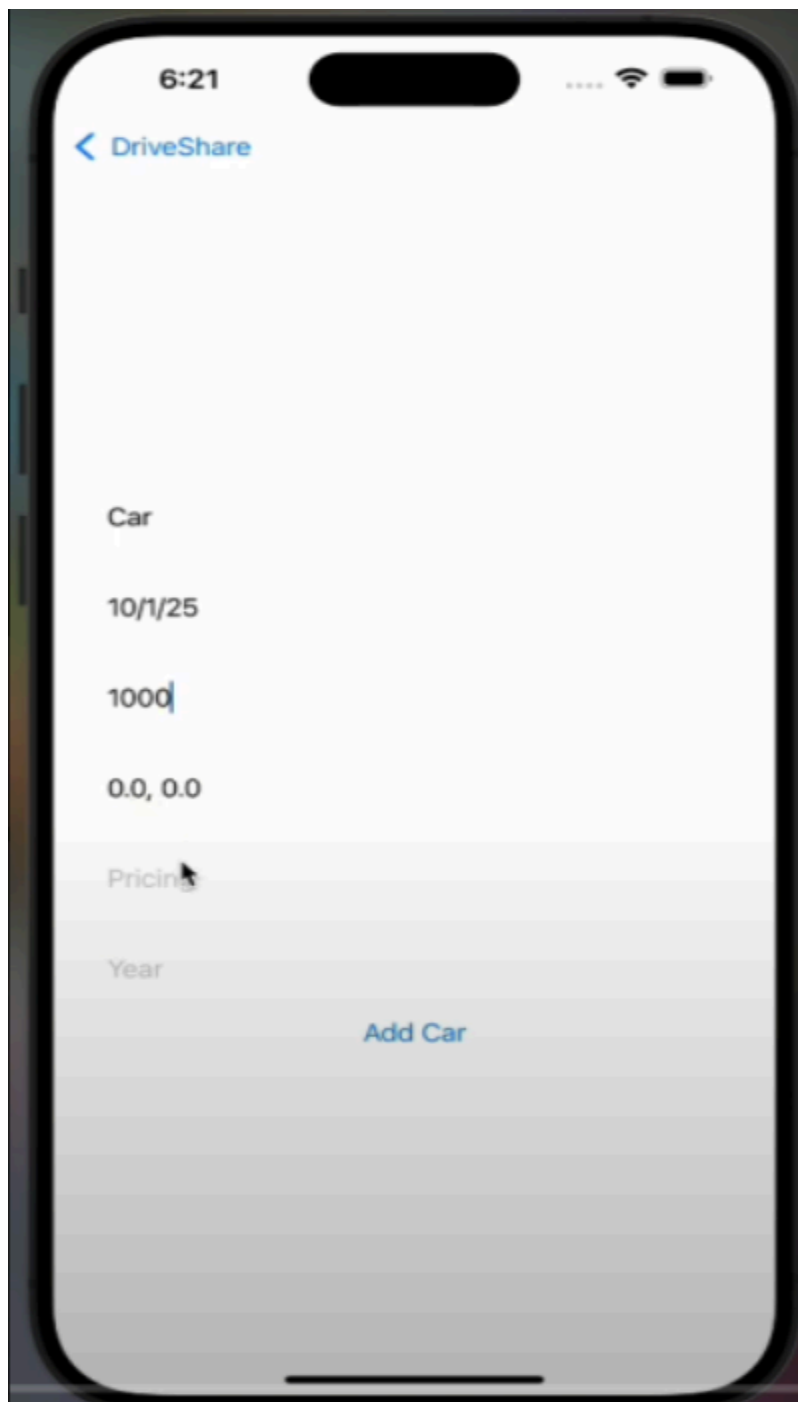
This screen allows you to view details about the car such as how much mileage and where to pick up the car. You can also reserve the car for a different day and the user who owns the car.

Messages Screen:



This screen is where you can message to different users regarding the car application. It can be from user to renter or owner and vice versa.

Adding a Car Screen:



The screenshot shows a mobile application interface for adding a car. At the top, the status bar displays the time 6:21, signal strength, Wi-Fi, and battery icons. Below the status bar is a navigation bar with a blue back arrow and the text 'DriveShare'. The main content area contains several input fields: a text field labeled 'Car' with the value '1000', a date field labeled '10/1/25', a text field labeled '1000', and a text field labeled '0.0, 0.0'. Below these fields are three more input fields labeled 'Pricing', 'Year', and 'Add Car'. The 'Add Car' field is highlighted in blue.

This screen is where you can add a car. You can include attributes like the pricing, year, date, and car model to add the car.