DriveSync

Car Rental App Documentation

Group Members:

1. Ozodbek Adkhamov (210111)
2. Diyorbek Majidov (210125)
3. Jakhongir Odilov (2110057)
4. Asror Kodirov(210083)

Outline

1. Introduction
2. Project overview
3. Functional and Non-functional Requirements
4. Activity diagram
5. System and hardware requirements
6. Application Portability
7. Blackbox testing
8. Conclusion

I. Introduction

The DriveSync documentation presents a thorough overview of the DriveSync project, a Flutter application aimed at optimizing the car rental experience for users. DriveSync offers a user-friendly interface and robust functionality designed to simplify the process of renting vehicles for a diverse range of users.

Developed with a focus on efficiency and usability, DriveSync enables users to browse, book, and manage car rentals seamlessly using their mobile devices. Leveraging the versatility of the Flutter framework, DriveSync ensures smooth performance and compatibility across various platforms, including Android and iOS.

This documentation serves as a comprehensive resource for understanding the features, functionalities, and technical aspects of the DriveSync app. From installation guidelines to detailed explanations of core features, developers, users, and stakeholders can rely on this documentation to navigate DriveSync effectively.

Explore DriveSync and discover how it transforms the car rental experience through its intuitive interface and innovative solutions. Let DriveSync be your trusted companion in simplifying and enhancing the car rental process.

Main features:

- Login
- Register
- Reset password
- Log out
- Edit profile details
- Delete Account
- See list of cars
- Add car
- See car detail

II. Project overview

DriveSync is an easy-to-use car rental app. You can find, book, and manage your car rentals right from your phone. It's simple to use, and you can choose from a variety of cars. Whether you need a car for a trip or daily use, DriveSync makes renting a car easy and convenient for everyone.

1. Code Structure

/lib
- /models
  - -car_model.dart
  - -car_model.g.dart
  - -user_model.dart
  - -vehicle.dart
- /Screens
  - /login
    - /widgets
      - -login_footer_widget.dart
      - -login_form_widget.dart
      - -login_header_widget.dart
    - - authentication_functions.dart
    - - forget_password_mail.dart
    - - login_screen.dart
    - - mail_verification.dart
    - - otp_screen.dart
    - - sign_up.dart
    - - splash_screen.dart
    - -user_functions.dart
  - - MyLIstingsScreens.dart
  - -add_car.dart
  - -car_details.dart
  - -cars_home_page.dart
  - -cars_list_page.dart
  - -nav-page.dart
  - -nav_page.dart
  - -notifications_page.dart
  - -profile_screen.dart
  - -update_car_screen.dart
  - -update_profile_screen.dart

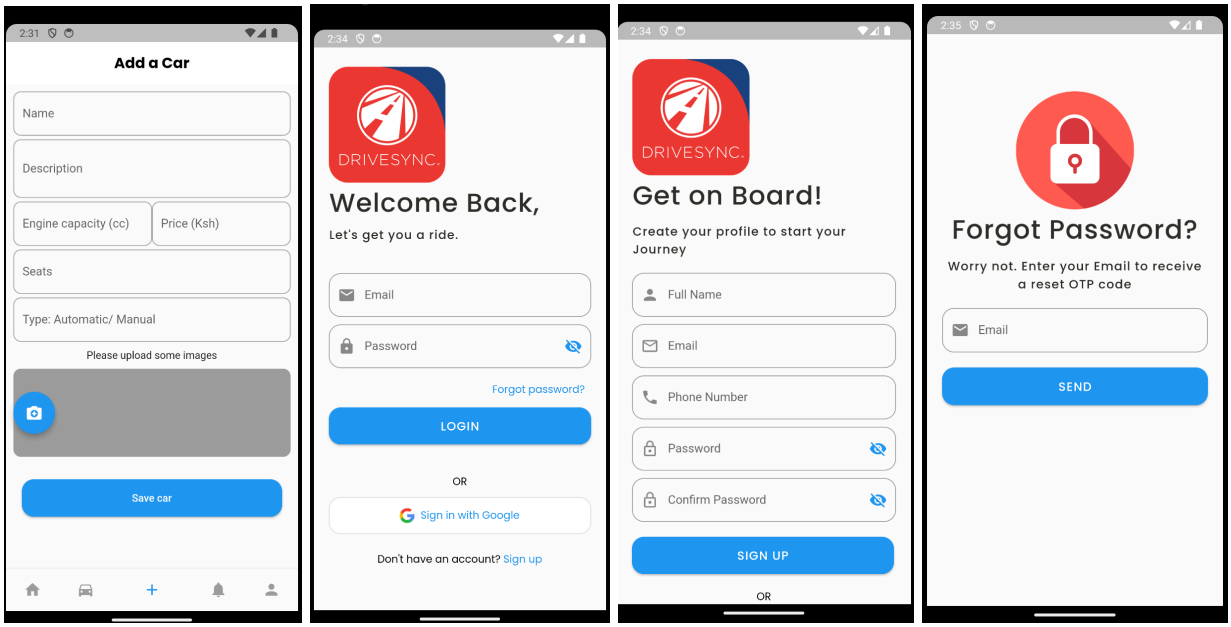
- /utils
  - -colors.dart
  - -tabs.dart
  - -utils.dart

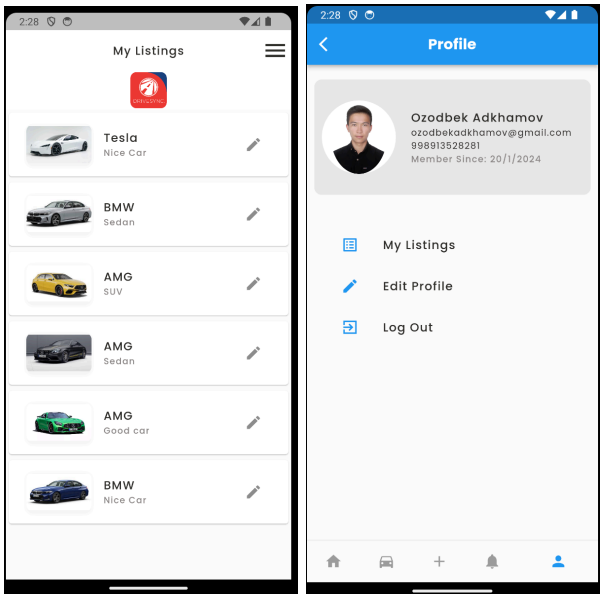
- /widgets/form

- - firebase_options.dart
- - main.dart
- - remote_config.dart

2. Screens:

Screens:





1. Login Screens:
   - login_screen.dart: This screen serves as the entry point for user authentication. It provides options for users to log in, sign up, or reset their passwords.
   - otp_screen.dart: This screen handles OTP verification during the login process.
   - forget_password_mail.dart: Users can initiate the password reset process through this screen.
   - mail_verification.dart: Handles email verification after user registration.
   - sign_up.dart: Allows users to create a new account.
   - splash_screen.dart: A splash screen displayed while the app is loading or initializing.

2. User Listings Screens:
   - MyListingsScreens.dart: Displays a list of car listings owned by the current user.
   - add_car.dart: Allows users to add new car listings to the platform.
   - car_details.dart: Displays detailed information about a specific car listing.
   - update_car_screen.dart: Enables users to update and edit existing car listings.

3. Car Listings Screens:
   - cars_home_page.dart: The homepage of the app, showcasing a curated selection of car listings.
   - cars_list_page.dart: Displays a comprehensive list of available car listings.
   - car_details.dart: Offers detailed information about a specific car listing.

4. Navigation Screens:
   - nav_page.dart: The main navigation page of the app, providing access to different sections and features.
   - notifications_page.dart: Displays notifications and alerts for the user.
   - profile_screen.dart: Shows the user's profile information and settings.
   - update_profile_screen.dart: Allows users to update their profile information.

5. Widgets:
   - /widgets: Contains reusable widgets used across various screens for consistent UI elements.
     - login_footer_widget.dart: Footer widget displayed on login screens.
     - login_form_widget.dart: Form widget for user login.
     - login_header_widget.dart: Header widget for login screens.

3. Database handling

Firebase Integration

DriveSync, the car rental service app, seamlessly integrates with Firebase services, enhancing functionality and user experience:

- Authentication: Firebase Authentication ensures secure user authentication, verifying identities and managing access control.

- Remote Config: Firebase Remote Config enables dynamic configuration management, allowing real-time updates to app behavior without requiring new app releases.

- Firestore Database: Firestore serves as the primary NoSQL database, providing real-time data synchronization, offline support, and robust security features for DriveSync.

Firestore Database Integration

DriveSync's database handling relies on Firestore to manage crucial data aspects:

- Data Models: Structured data models like `Car` and `UserModel` define attributes and behaviors associated with car listings and user profiles, facilitating seamless mapping between Firestore documents and Dart objects.

- CRUD Operations: Firestore enables efficient CRUD operations for managing car listings, user profiles, rental reservations, and other essential data entities within collections and documents.

- Structured Data Organization: Firestore organizes data into collections and documents, facilitating hierarchical data organization and efficient querying to support DriveSync's operational needs.

- Real-time Data Synchronization: Firestore's real-time synchronization capabilities ensure seamless data propagation and synchronization across DriveSync users' devices, ensuring consistent and up-to-date information regarding car availability, rental bookings, and user interactions.

III. Requirements

Functional Requirements

Functional requirements outline the core functionalities that DriveSync must support to fulfill its purpose effectively:

1. User Registration:
   - DriveSync facilitates user registration, allowing individuals to create accounts either as customers seeking rental services or as car owners wishing to list their vehicles for rent.
   - User registration involves providing essential details such as name, email address, and password.

2. Vehicle Listings:
   - Car owners are empowered to list their vehicles on DriveSync, providing comprehensive information such as make, model, year, mileage, transmission type, fuel type, and rental pricing.
   - Additionally, car owners may include high-quality images of their vehicles to attract potential renters.

3. Vehicle Search and Booking:
   - DriveSync offers customers a robust search functionality, enabling them to browse available vehicles based on location, date, vehicle type, and other preferences.
   - Users can view detailed information about each vehicle, including availability calendars, rental rates, and vehicle specifications.
   - Upon selecting a vehicle, customers can proceed to book it for specific dates and times directly through the app.

4. Rental Management:
   - Car owners have access to a dedicated dashboard within DriveSync to manage their vehicle listings effectively.
   - Owners can update vehicle availability, adjust rental rates, respond to booking inquiries, and maintain vehicle information such as maintenance records and insurance details.
   - DriveSync ensures that the rental management process is streamlined and intuitive for car owners, allowing them to maximize their rental potential.

Non-Functional Requirements

Non-functional requirements encompass aspects of DriveSync's performance, scalability, user interface, and reliability:

1. Performance:
   - DriveSync is designed to deliver optimal performance, ensuring fast loading times, responsive interactions, and efficient data retrieval.
   - Performance testing is conducted regularly to identify and address potential bottlenecks and optimize system responsiveness.

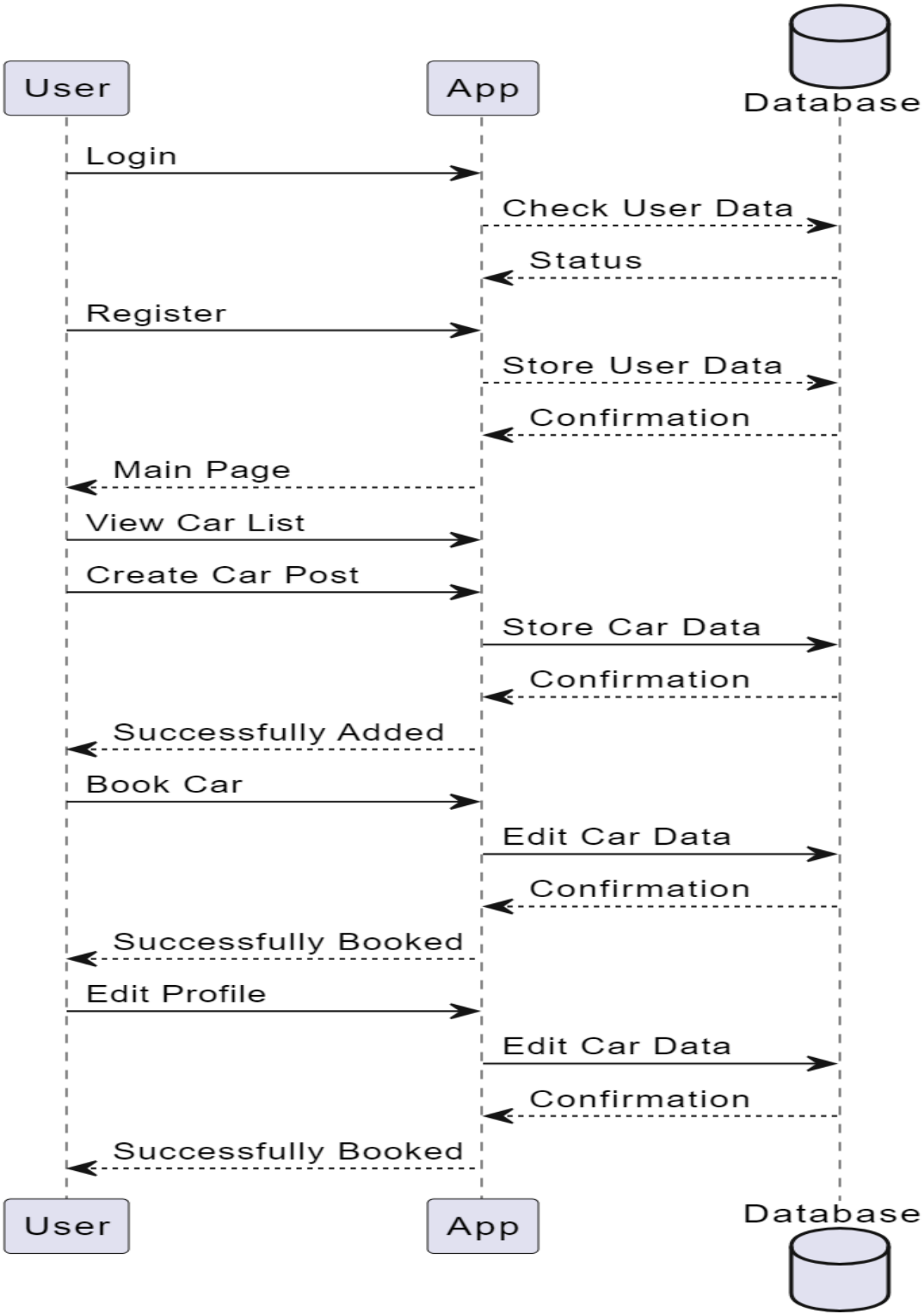
2. Scalability:
   - DriveSync is architected to scale gracefully with increasing user traffic, expanding vehicle listings, and growing transaction volumes.
   - Scalability measures include horizontal scaling of server resources, database optimizations, and efficient caching mechanisms.

3. User-Friendly Interface:
   - DriveSync features an intuitive and visually appealing user interface (UI) designed to enhance user experience and engagement.
   - The UI is characterized by clear navigation pathways, intuitive controls, and consistent design patterns across different app screens.

4. Reliability:
   - DriveSync prioritizes system reliability, aiming to minimize service disruptions, downtime, and data loss.
   - Measures such as data redundancy, automated backup systems, and failover mechanisms are implemented to ensure uninterrupted service availability.

By adhering to these functional and non-functional requirements, DriveSync aims to deliver a robust, user-centric car rental platform that meets the needs and expectations of both customers and car owners alike.

IV. Activity diagram


User
App
Database
Login
Check User Data
Status
Register
Store User Data
Confirmation
Main Page
View Car List
Create Car Post
Store Car Data
Confirmation
Successfully Added
Book Car
Edit Car Data
Confirmation
Successfully Booked
Edit Profile
Edit Car Data
Confirmation
Successfully Booked
User
App
Database

This picture provides an overview of the user interaction flow within a car rental application depicted through a Unified Modeling Language (UML) diagram. The diagram illustrates the sequence of actions performed by a user interacting with the application, including login, registration, viewing car listings, creating car posts, booking cars, and editing profiles.

User Interaction Flow:

Login:

- The user initiates the interaction by logging into the application.
- The application verifies the user's credentials by checking the user data stored in the database.
- Upon successful validation, the application proceeds to the main page.

Registration:

- If the user is not registered, they can choose to register.
- The application collects user data and stores it in the database.
- After storing the user data, the application confirms the registration.

View Car List:

- Once logged in, the user can view the list of available cars.
- This information is retrieved from the database and displayed to the user.

Create Car Post:

- Users can create posts to advertise their cars for rental.
- The application collects car data from the user and stores it in the database.
- After storing the car data, the application confirms the successful addition of the car post.

Book Car:

- Users can select a car from the list and proceed to book it.
- The application facilitates the booking process by updating the car's data in the database to reflect its booking status.
- Confirmation of successful booking is sent to the user.

Edit Profile:

- Users have the option to edit their profile information.
- The application allows users to make changes to their profile, which are then updated in the database.
- Upon successful update, the application confirms the changes to the user.

Conclusion:

The UML diagram illustrates the sequential flow of interactions between the user, the application, and the database within a car rental system. It provides a clear understanding of how users can navigate through various functionalities such as logging in, registering, viewing car listings, creating car posts, booking cars, and editing profiles. This diagram serves as a useful tool for developers and stakeholders to comprehend the user journey and design or improve the car rental application accordingly.

V. Development lifecycle

The mobile application development lifecycle for DriveSync encompasses 12 weeks and involves collaborative efforts from team members, including frontend developers, backend developers, full-stack developers, and QA engineers. Despite the specialization of team members, each individual contributes to various aspects of the project, ensuring comprehensive development and quality assurance throughout the lifecycle.

Week 2: Planning and Requirements Gathering

- Define project scope, objectives, and deliverables.
- Establish timelines and allocate resources.
- Conduct requirements gathering sessions.
- Document functional and non-functional requirements.
- Define user stories and prioritize features for implementation.

Week 3: Design and Prototyping

- Create wireframes and design user interfaces for Welcome, SignUp, and LogIn pages.
- Develop prototypes for user testing and feedback.
- Review design mockups and provide input on technical feasibility.
- Ensure alignment of frontend and backend components with design specifications.

Week 4-12: Development and Implementation

- Implement frontend components for Welcome, SignUp, and LogIn pages.
- Develop backend APIs for car posting, booking system, and user authentication.
- Integrate frontend and backend components to create a cohesive user experience.
- Implement push notifications and payment processing functionalities.
- Set up database schemas and ensure proper data handling.

Week 13: Testing and Quality Assurance

- Conduct comprehensive blackbox testing on all implemented functionalities.
- Identify and log defects or inconsistencies for resolution.
- Work closely with the development team to prioritize and address identified issues promptly.
- Ensure application stability and adherence to quality standards.
- Define acceptance criteria for each feature and conduct user acceptance testing.

Week 14-15: Refinement and Optimization

- Focus on optimizing application performance and user experience based on testing feedback.
- Implement enhancements and refinements to UI components and backend functionality.
- Iterate on design elements and user flows based on user feedback and testing outcomes.

- Address any scalability or performance bottlenecks identified during testing phases.
- Incorporate user feedback to refine features and enhance overall usability.

By following this timeline, DriveSync aims to efficiently develop, test, and deploy a high-quality car rental application that meets the needs and expectations of its users.

VI. System and Hardware Requirements

DriveSync is designed to operate efficiently across a variety of devices, ensuring a seamless user experience while providing robust performance and compatibility. To ensure DriveSync functions optimally, users should consider the following system and hardware specifications:

System Requirements

- Operating System:
  - DriveSync is compatible with Android devices running Android 5.0 (Lollipop) or later versions.
  - For iOS devices, DriveSync supports iOS 9.0 or later versions.

Hardware Requirements

- For Android Devices:

- Processor: DriveSync performs best on devices equipped with ARM or x86 processors clocked at a minimum

speed of 1.5 GHz. This ensures smooth navigation and quick response times throughout the application.
- RAM (Random Access Memory): It is recommended to have at least 2 GB of RAM to support DriveSync's multitasking capabilities and ensure efficient data processing.
- Storage: DriveSync requires a minimum of 50 MB of available storage space for installation. Additional storage space is necessary for caching data and storing downloaded resources, so users should ensure ample storage capacity on their devices.
- Display: A high-resolution display enhances the visual experience of DriveSync's user interface. A recommended screen resolution of 720x1280 pixels or higher ensures crisp graphics and clear text rendering.

- For iOS Devices:

- Processor: DriveSync performs optimally on devices equipped with Apple A9 chip or later. These processors offer enhanced performance and efficiency, contributing to a smooth user experience.

- RAM (Random Access Memory): It is recommended to have at least 2 GB of RAM to support DriveSync's memory-intensive operations and facilitate seamless multitasking.
- Storage: DriveSync requires a minimum of 50 MB of available storage space for installation. Users should also allocate additional storage space for caching data and storing downloaded resources to ensure uninterrupted functionality.
- Display: A high-resolution display with a minimum screen resolution of 750x1334 pixels is recommended for an immersive user experience. This ensures crisp visuals and clear readability of DriveSync's interface elements.

Additional Considerations

- Internet Connection:
  - DriveSync relies on a stable internet connection for accessing its features and services. Users should have access to Wi-Fi or cellular data networks to browse vehicle listings, make bookings, and receive real-time updates.
  - DriveSync may include features such as real-time availability updates, map integration for location-based searches, and secure payment processing, all of which require a consistent internet connection for optimal functionality.

By adhering to these system and hardware requirements, users can optimize their DriveSync experience, enjoying a fluid and efficient car rental platform tailored to their needs and preferences.

VII. Application Portability

DriveSync prioritizes application portability to ensure accessibility across a diverse range of devices and platforms. By leveraging the capabilities of the Flutter framework, DriveSync delivers a seamless user experience while maintaining consistency and performance across various operating systems and device types.

Flutter Support Library

Flutter's support library offers a comprehensive set of widgets and tools designed to facilitate cross-platform development. DriveSync leverages these components to create a unified user interface and user experience across Android and iOS devices. Key components of the Flutter support library include:

- Widgets: Flutter provides an extensive collection of widgets, including material design and Cupertino-style widgets, to create platform-specific UI elements.
- Layouts: DriveSync utilizes layout widgets such as Row, Column, Stack, and Flex to create responsive and adaptable UI layouts that adjust dynamically to different screen sizes and orientations.

- MediaQuery: The MediaQuery class enables DriveSync to retrieve device-specific information such as screen size, pixel density, and orientation, allowing for adaptive UI design and layout adjustments.

Responsive Screen Layouts

DriveSync employs responsive design principles to ensure optimal presentation and usability across a variety of screen sizes and resolutions. By utilizing flexible layouts and adaptive UI components, DriveSync adapts seamlessly to devices ranging from smartphones to tablets, offering a consistent and intuitive user experience.

Key strategies for achieving responsive screen layouts in DriveSync include:

- Flexibility: DriveSync's UI components are designed to flexibly adapt to different screen sizes and aspect ratios, ensuring content remains accessible and readable across a wide range of devices.
- Media Queries: DriveSync utilizes media queries to detect device characteristics and adjust UI elements accordingly. By defining breakpoints based on screen width and height, DriveSync optimizes layout and content presentation for each device type.
- Dynamic Sizing: DriveSync employs dynamic sizing techniques to scale UI elements proportionally based on device specifications. This ensures that text, images, and interactive elements remain legible and visually appealing across varying screen densities and resolutions.

Cross-Platform Compatibility

DriveSync's cross-platform compatibility extends beyond Android and iOS devices to encompass web and desktop platforms. By harnessing the power of Flutter's multi-platform support, DriveSync can reach a broader audience and provide consistent functionality across different environments.

Key features of DriveSync's cross-platform compatibility include:

- Web Support: DriveSync leverages Flutter's web support to deliver a native-like experience on web browsers, allowing users to access DriveSync's features seamlessly from desktop and laptop computers.
- Desktop Support: DriveSync extends its reach to desktop platforms, including Windows, macOS, and Linux, through Flutter's desktop embedding capabilities. Users can enjoy the same intuitive interface and functionality on desktop computers as they do on mobile devices.

By embracing application portability through Flutter, DriveSync ensures that users can access its features and services from a wide range of devices and platforms, enhancing convenience, accessibility, and user engagement.

VIII. Blackbox Testing

Blackbox testing is a crucial aspect of DriveSync's development process, aimed at ensuring the reliability, functionality, and usability of the application from an end-user perspective. By conducting comprehensive blackbox tests, DriveSync identifies potential issues, validates core functionalities, and enhances overall user satisfaction.

Blackbox testing encompasses various test scenarios designed to evaluate DriveSync's features, workflows, and interactions. Test cases are executed from an external perspective, simulating user actions and inputs to assess the application's behavior under different conditions.

Test Scenarios:

1. User Registration:
   - Verify that users can successfully create accounts and register with DriveSync.
   - Validate the registration process, including input validation, error handling, and account creation.

2. User Login:
   - Ensure that registered users can log in securely to access DriveSync's features and functionalities.
   - Validate login credentials, authentication processes, and session management.

3. Vehicle Browsing and Booking:
   - Test the vehicle browsing functionality to ensure users can search for available vehicles based on location, date, and preferences.
   - Validate the booking process, including vehicle selection, reservation confirmation, and payment processing.

4. Rental Management:
   - Test car owners' ability to manage their vehicle listings, including adding new listings, updating availability, and responding to booking inquiries.
   - Validate administrative functionalities such as editing listing details, adjusting rental rates, and viewing booking history.

Expected Outcomes:

1. User Registration:
   - Users should be able to register successfully without encountering errors or issues.
   - Account creation should be seamless, with users receiving confirmation of their registration.

2. User Login:
   - Registered users should be able to log in securely using valid credentials.
   - Login attempts should be authenticated, granting users access to DriveSync's features upon successful authentication.

3. Vehicle Browsing and Booking:
   - Users should be able to browse available vehicles and view detailed information about each listing.
   - Booking transactions should be processed smoothly, with users receiving timely confirmation of their bookings.

4. Rental Management:
   - Car owners should be able to manage their vehicle listings effectively, with changes reflected accurately in DriveSync's database.
   - Administrative functionalities should be intuitive and user-friendly, enabling car owners to update listings, respond to inquiries, and track rental activity.

Actual Results:

1. User Registration:
   - Registration process: No errors encountered; users successfully created accounts.
   - Account creation: Confirmation emails sent promptly, verifying successful registration.

2. User Login:
   - Authentication process: Users logged in securely using valid credentials.
   - Session management: Login sessions persisted accurately, granting users access to DriveSync's features.

3. Vehicle Browsing and Booking:
   - Browsing functionality: Users navigated through vehicle listings smoothly, accessing detailed information about available cars.
   - Booking process: Transactions processed without errors, with users receiving timely confirmation of their bookings.

4. Rental Management:
   - Listing management: Car owners managed their vehicle listings effectively, with updates reflected accurately in DriveSync's database.
   - Administrative functionalities: Car owners accessed and utilized administrative features seamlessly, facilitating efficient rental management.

By conducting thorough blackbox testing and analyzing results systematically, DriveSync strives to deliver a reliable, functional, and user-friendly car rental application that meets the needs and expectations of its users.

IX. Conclusion

DriveSync represents a significant advancement in car rental applications, offering users a seamless platform for renting vehicles with ease. Throughout this documentation, we explored DriveSync's functionalities, system requirements, development lifecycle, and testing procedures.

With robust features like user registration, vehicle listings, and rental management tools, DriveSync ensures a streamlined rental process for both customers and car owners. The 12-week development lifecycle involves collaborative efforts from frontend developers, backend developers, full-stack developers, and QA engineers, aiming for high-quality results.

Through rigorous blackbox testing and continuous refinement, DriveSync strives to deliver reliability, usability, and innovation in the car rental industry. As DriveSync evolves, we remain committed to enhancing your car rental experience.