جامعة الإسكندرية
ALEXANDRIA UNIVERSITY
Faculty of Business

Faculty of Business

Management Information Systems Department

Graduation Project

SAFE ZONE

An Application for Parking & car care center services

Supervised By:

Dr: Mohamed Moustafa Al-Abbassy

Submitted By:

Abdelrahman Nabil Ibrahim

Abdelrahman Sherif Mostafa

Abdelrahman Othman Abd El Megid Hasan

Mohamed fouad Mohamed zaki

Mohammed Abd El Aziz Selim Ghareeb

Mohammed Mohammed Abd El Fatah Bekhet

Yousef Sherif Salah Ali

Reem Esam El Deen Mohammed Abd El Rahman

Salma Gamal Mohammed Eid

Youmna Mohammed Saad Zaghlol

Rana Said Mohammed Ali

Nayera Abdelshafi Muhammad Bakr

June 2024

بِسْمِ ٱللَّهِ ٱلرَّحْمَٰنِ ٱلرَّحِيمِ

صدق الله العظيم

(سوره النجم)

Acknowledgement:

Dear Professors,

As we reach the culmination of our graduation project, we find ourselves reflecting on this journey with immense gratitude. Your guidance, wisdom, and unwavering support have been the cornerstones of our success.

Throughout this challenging and rewarding experience, your expertise has not only enhanced our academic knowledge but also inspired us to push the boundaries of our capabilities. Your constructive feedback and encouragement have been invaluable, enabling us to transform our ideas into a cohesive and meaningful project.

We are profoundly thankful for the time and effort you have invested in our growth

Special thanks to Dr. Mohamed Mostafa & Mr. Bahaa

for their endless support, kind and understanding spirit during our project.

Thanks also go to:

DR. Yasser Abd El-Ghaffar

DR. Ghada El Khayat

DR. Abeer Ameer

Table of content

Chapter (1):- Introduction

(1.1) Introduction & Idea

The Safe Zone app is designed to provide users with a secure and convenient solution for parking their vehicles. It allows users to find and reserve parking spaces in designated Safe Zone parking lots. Once parked, users can monitor their vehicles remotely using live camera feeds accessible through the app. The app offers real-time surveillance of the parking area, allowing users to keep an eye on their vehicles from anywhere. They can view live video streams of their parked cars, ensuring peace of mind and add security against theft or damage.

(1.2) Problems & Solutions

The Safe Zone app addresses several problems related to vehicle parking and security:

Lack of Secure Parking: It provides a solution for finding secure parking spaces, which can be a significant concern in crowded urban areas.

Time-Consuming Search for Parking: The app helps users quickly find available parking spaces without the need to drive around searching, saving time and reducing stress.

Remote Monitoring: By offering live camera feeds, the app solves the problem of not being able to monitor parked vehicles, giving users the ability to check on their vehicle's safety remotely.

Theft and Damage: The real-time surveillance feature acts as a deterrent against theft and damage, addressing concerns about vehicle security in parking lots.

Convenience: The app adds convenience by allowing users to reserve parking spaces in advance & enjoying with service packages as they want or booking extra services.

Peace of Mind: With the ability to monitor their vehicle at any time, users gain peace of mind knowing that their vehicle is under surveillance and less likely to be tampered with.

(1.3) Motivation

The motivation behind the Safe Zone app is to enhance personal safety and security for individuals who need to park and monitor their vehicles. It aims to address concerns such as:

Providing a Secure Environment: Ensuring that users have access to safe parking spaces where their vehicles are less likely to be subjected to theft or damage.

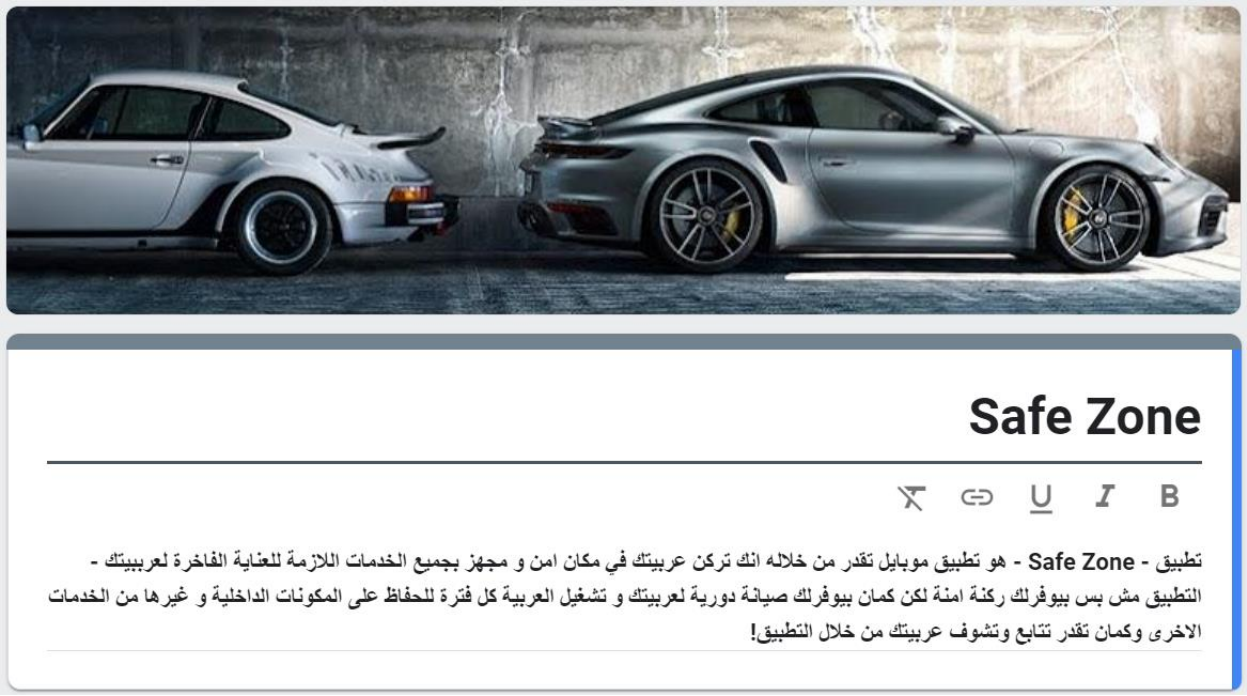
Ease of Access: Offering a convenient way to find and reserve parking spaces, reducing the time and effort typically spent searching for a spot.

Real-Time Surveillance: Allowing users to monitor their vehicles remotely, giving them peace of mind knowing that they can keep an eye on their cars at any time.

Emergency Assistance: Enabling users to quickly alert Safeguarding Services when help or assistance is needed, thereby extending the reach of campus safety and security.

These motivations are centered around the real-time needs of individuals who use parking facilities daily and seek a reliable and efficient way to ensure their vehicles'

safety. The app's features are designed to cater to these needs by providing a user-friendly platform for parking space reservation and vehicle surveillance.



(1.4) Mission

- **To expand & stand still in the market.**
- **To raise our market share & to have our stamp.**
- **Start from Egypt as a country as this service is barely rare.**

(1.5) Vision

We seek for providing High quality services for cars of the A-class specially with safety tools.

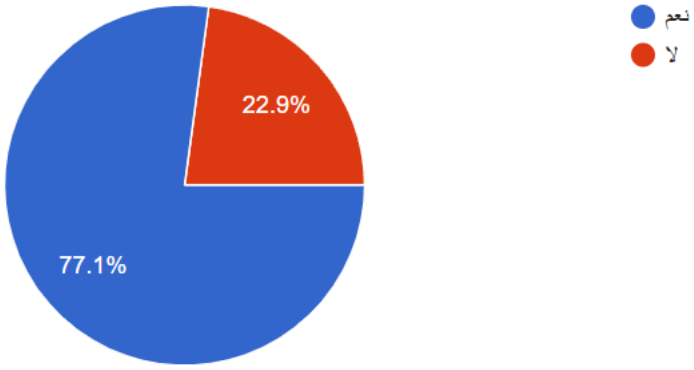
Chapter (2): Survey and Technology Tools

(2.1) Survey

We conducted this survey to extract some data about our potential users and we got 35 responces that differ between users who have cars and users who are seeking to

هل انت مهتم بصيانة سيارتك وركنها لفترة اثناء سفرك؟

35 ردًا



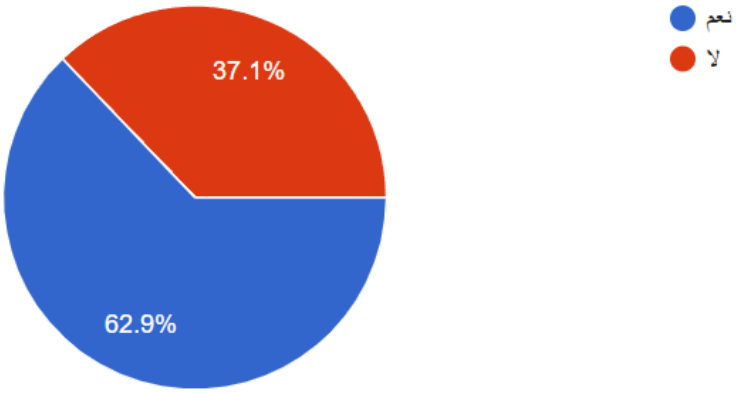
have one.

27 responses came to answer that they have cars that represents 77.1% of total responses.

About more than half response came to that they would prefer keeping their cars in a

هل تمتلك سيارة حاليا؟

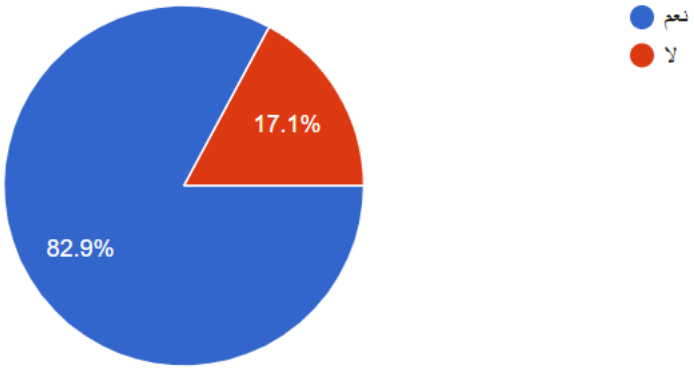
35 ردًا



trustful garage for a long period while traveling.

هل انت مهتم بفكرة متابعة سيارتك والتحقق من سلامتها من خلال تطبيق وانت في مكانك؟

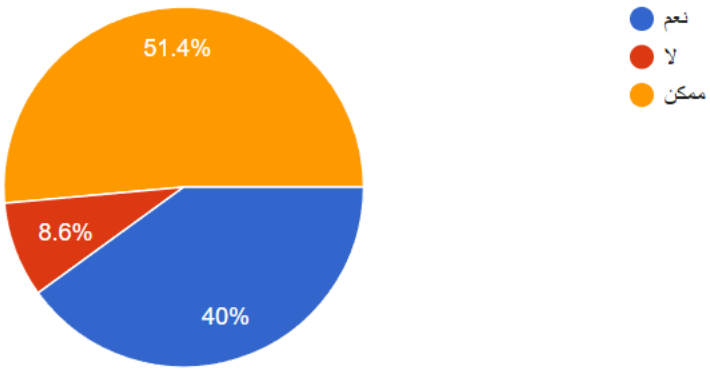
35 ردًا



Most of responses cared about monitoring their car safety from the application.

عمرك فكرت تسيب عربيتك في جراج او مكان امن وموثوق لفترة طويلة لو مسافر؟

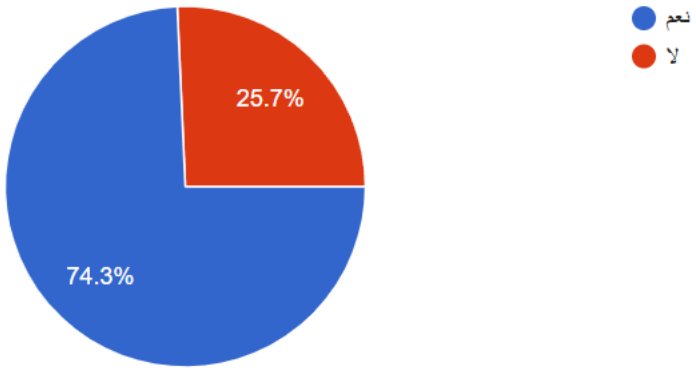
35 ردًا



Most of responses came to care about paying to provide maintenance & cleaning to their cars through the application.

هل انت مهتم انك تدفع لخدمات الصيانة والتنظيف للسيارة من خلال التطبيق؟

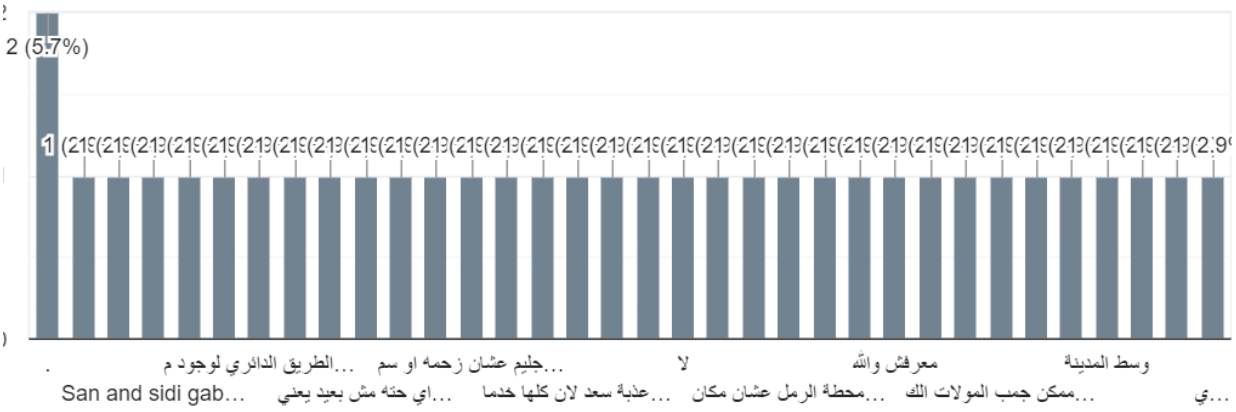
35 ردًا



Responses differed between City center & near to malls and other related areas.

تقدر ترشح مكان مميز لبناء مقر Safe Zone في المكان ده؟ برجاء توضيح السبب.

35 ردًا

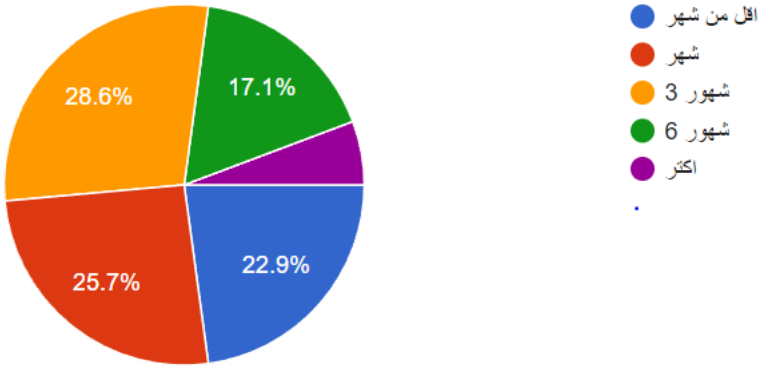


Most of responses tend to monitor their cars through the live camera feed.

تحب تشوف عربيتك وتتابعها ازاي؟

عايز تسيب عربيتك وتستفيد بخدمات الابلكيشن لمدة قد ايه؟

35 ردًا

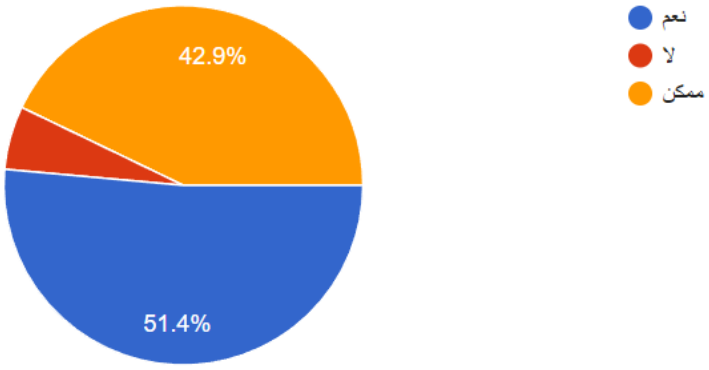


responses preferred to keep their car in safe zone care center between 6 months and 1 month.

Most of responses came to think that safe zone app will be common in Egypt.

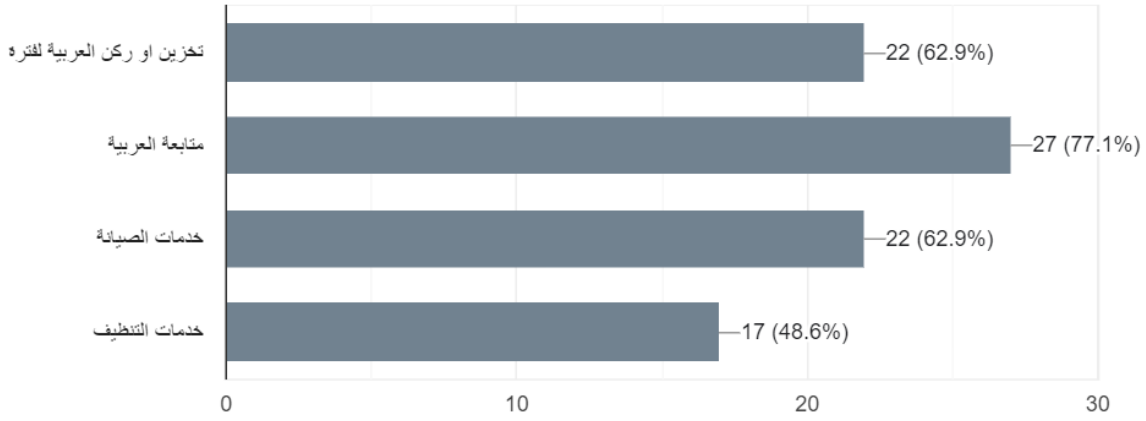
هل تعتقد ان ابلكيشن Safe Zone هيكون شائع في مصر؟

35 ردًا



ايه من الخدمات اللي بيوفرها التطبيق هتكون مهتم بيها اكتر؟

35 ردًا

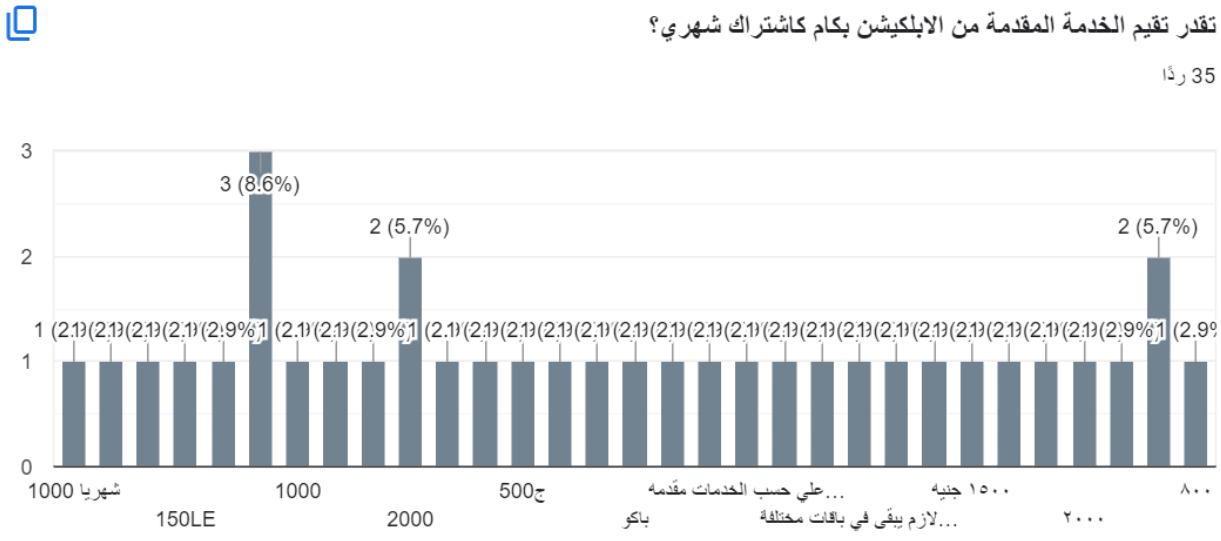


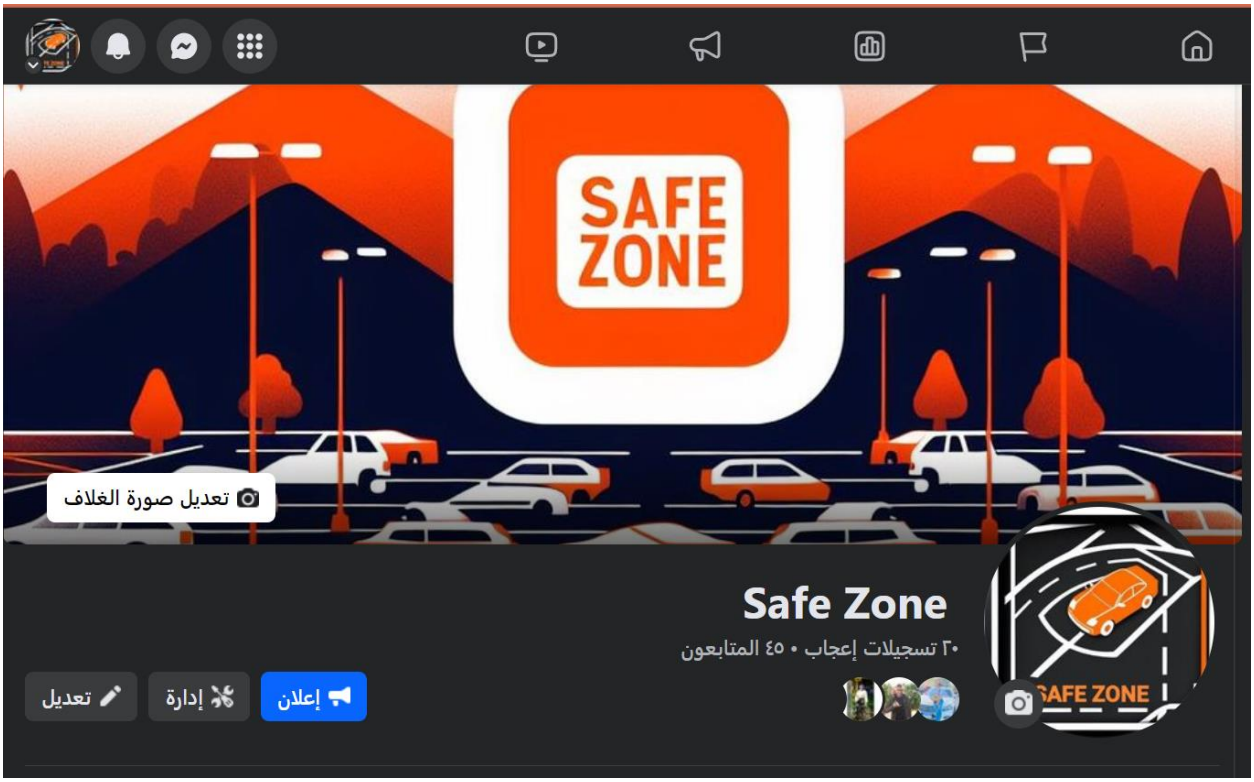
Most of responses cared the most about their car monitoring service.

Responses were between 500 to 2000 given to the services introduced from the application.

(2.2) Social Media platforms

Safe Zone Facebook page



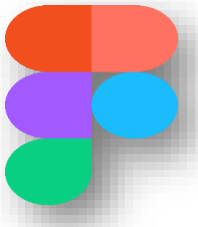


Safe Zone --> link

2.3 Tools and technology used:

Diagramming Software:

Figma



is a cloud-based vector design tool that is widely used in the field of UI/UX design. It allows UX and UI designers to create, collaborate on, and share their designs. Here are some key features of Figma:

- **Real-time Collaboration: Figma is built for collaboration across the entire product development team, allowing you to design experiences together.**
- **Prototyping: Figma's prototyping tools make it easy to build high-fidelity, no-code interactive prototypes right alongside your designs.**
- **Design Consistency: With reusable assets in shared libraries, you can standardize components and variables, saving time and maintaining consistency.**
- **Customer Journey Diagramming: You can easily diagram your customer journey and user flows in Figma using Figma's whiteboarding tool.**

{ URL : https://www.figma.com/ }

Lucidchart:

is a web-based diagramming and visualization application used for creating flowcharts, organizational charts, mind maps, wireframes, UML diagrams, and other types of visual representations.

1. Diagram Creation: Lucid chart allows users to create a wide range of diagrams including:

- **Flowcharts**
- **Mind maps**
- **Network diagrams**
- **Organizational charts**

{URL: https://www.lucidchart.com/pages/ }

Edraw Max:

is a versatile diagramming software developed by EdrawSoft, used for creating a wide range of diagrams and charts. Here are some key features and aspects of Edraw Max:

{URL: https://www.edrawmax.com/ }

draw.io:

is a technology stack for building diagramming applications, and the world's most widely used browser-based end-user diagramming software.used to create a wide variety of diagrams and flowcharts

{URL: https://www.drawio.com/}

Database Tools :

Cloud Firestore :

is a flexible, scalable database for mobile, web, and server development offered by Firebase and Google Cloud. It's a NoSQL database, meaning it stores data in a document format rather than rigid tables like traditional relational databases.

{URL: https://firebase.google.com/docs/firestore}

Cloud Storage:

for Firebase is built on fast and secure Google Cloud infrastructure for app developers who need to store and serve user-generated content, such as photos or videos.

{URL: https://firebase.google.com/docs/storage/}

Front-End Tools:

Flutter:

is an open-source UI software development toolkit created by Google. It is used for building natively compiled applications for mobile (iOS and Android), web, and desktop (Windows, macOS, Linux) from a single codebase.

{URL: https://flutter.dev/}

Dart:

is an open-source, general-purpose programming language developed by Google. It is designed for building web, server, desktop, and mobile applications. Dart is the language behind the Flutter framework, which is used for creating cross-platform mobile and web applications

{URL: https://dart.dev/}

Back-End:

Firebase:

is a comprehensive platform developed by Google for building and managing mobile and web applications. It provides a suite of tools and services to help developers create high-quality apps, improve user engagement, and grow their business.

{URL : https://firebase.google.com/ }

Chapter 3 (Analysis and Design)

Customer segment description “Persona”:

Name: Mohamed Hassan

Software Engineer

Demographics:

- Age: 28
- Gender: Male
- Occupation: Software Engineer
- Marital Status: Single
- Location: Alexandria, Egypt

User Persona 1

Background:

Meet Mohamed, is a young professional living in Alexandria, Egypt and works as a software engineer. He decided to take a vacation and travel for 3 months to Italy, so he want to leave his car parked in a safe place until his return

Goals

- Ensure car safety
- Find a convenient solution for parking his car securely, especially when traveling or going on vacations.
- Access his car remotely to monitor its condition

Challenges

- Busy schedule which makes him has no time for car maintenance
- Concerns about car safety

How Safe Zone App Helps:

- Monitoring car remotely 24/7
- Provides periodic maintenance
- Provide Safe place with guarded with security

Yasmin Abdelhamid

Entrepreneur

Demographics:

- Age: 36
- Gender: Female
- Occupation: Founder and CEO of a Business Startup
- Marital Status: Married With 2 Kids
- Location: Alexandria, Egypt

User Persona 2

Background:

Meet Yasmin, Yasmin is a determined entrepreneur who founded her own business startup in Cairo, Egypt. With a vision to innovate and disrupt traditional industries, As a wife and mother of two young children, Yasmin has a very busy day.

Goals

- Ensure the safety and reliability of her car for commuting to important business meetings
- Find a convenient solution for parking her car securely, especially when attending business conferences or client meetings

Challenges

- Limited Time due to her work and family
- Limited knowledge of the detailed car maintenance needs

How Safe Zone App Helps:

- Wide range of car care services
- Cleaning services & Periodic maintenance
- Monitoring of her car through live cam enabling her to check its status and, ensuring the safety of her vehicle and her family.

User Story:

User story 1

1. As a traveler, I want to browse through the different car care packages offered by Safe Zone, so I can choose the one that best suits my needs and duration of travel.

Accept criteria

1. The app must display a list of all available car care packages

2. The detailed information should include a comprehensive description

3. Users should be able to filter the list of packages based on various criteria

User story 2

2. As a car owner, I want to subscribe to a car care package through the Safe Zone app, so I can leave my car in the secure parking area and have it maintained while I'm away.

Accept criteria

1. The app must provide a clear and straightforward process for users to subscribe to a car care package.

2. User-sensitive information like credit card details should be encrypted and protected according to industry standards.

3. Upon successful subscription, the app should provide a confirmation message with details of the chosen packageand total cost

User story 3

As a Safe Zone user, I want to schedule my car drop-off and pick-up dates and times through the app, so I can seamlessly manage my car's stay

Accept criteria

1. The app must provide a dedicated and easily accessible section for users to schedule their car drop-off and pick-up

2. The interface should be clear and user-friendly, enabling easy selection of dates and times

3. Upon selecting dates and times, the app must provide a summary for user confirmation before finalizing the booking

User story 4

4. As a Safe Zone user, I want to access live camera feeds of the parking area, so I can monitor my car remotely and ensure its safety.

Accept criteria

1. Access to live camera feeds should be restricted to authorized Safe Zone users through secure login with valid credentials.

2. Live camera feeds should provide a clear and comprehensive view of the entire secure parking area

3. The live video stream should be of sufficient quality to allow users to easily identify their car and observe any potential concerns

User story 5

5. As a Safe Zone user, I want to request additional car care services on demand, such as car washing or detailing, through the app, so I can have my car pampered while it's parked.

Accept criteria

1. Each service should be described with details like its features, pricing, and estimated completion time.

2. Users should be able to easily select the desired service from the list and choose a convenient date and time for execution.

3. Users should be able to securely pay for the additional service through the app using a saved payment method or a new payment option.

User story 6

6. As a Safe Zone user, I want to receive notifications about upcoming maintenance needs and service recommendations based on my subscribed package, so I can stay informed about my car's health.

Accept criteria

1 . The system should automatically trigger notifications based on the subscribed package and the user's car information.

2. They should explain the importance of the service and its potential consequences if neglected.

3. Notifications should be personalized based on the user's subscribed package and car information.

Business Model Canvas :

Our business model:

Business to Customer / B2C

Our business model is based on serving the customers, our company provide services and the customers are getting benefit of these services.

Revenue Model: Based on subscription in services packages that the website provides.

Competitive Analysis & SWOT Analysis:

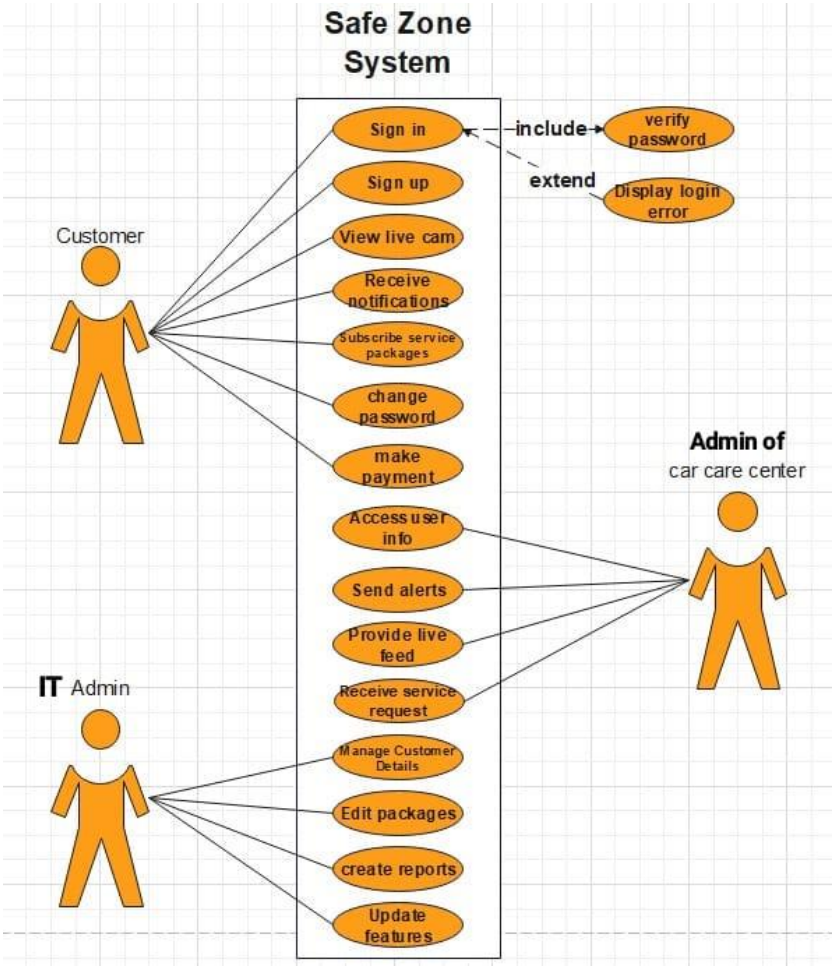
Competitive Analysis			
Competitors / Comparison	Safe Zone	Traditional Parking Facilities (Garage)	Car Maintenance and Repair Shops
Convenience	Offers subscription-based packages with transparent pricing for parking and car care services, potentially providing cost savings compared to paying for parking and maintenance separately	Typically offer parking services only, with limited or no additional maintenance options. Users may need to visit the facility physically to park or retrieve their cars.	Provide specialized services for maintenance and repair but do not usually offer parking facilities. Users must separately find parking when visiting these shops.
Accessibility	Users can access parking and car care services through the mobile app, making it accessible from anywhere with an internet connection.	Located in specific physical locations, requiring users to travel to the facility to park their cars.	Also located in physical locations, often separate from parking facilities, requiring users to travel to the shop for maintenance and repairs.
Security	Offers secure parking facilities with 24/7 surveillance and remote monitoring through live cam features, enhancing the safety of users' vehicles.	Security measures may vary, with some offering basic surveillance or attendants but not always guaranteeing the safety of vehicles from theft.	Focus on providing maintenance and repair services rather than parking security, although some may have basic security measures in place.
Cost	Offers subscription-based packages with transparent pricing for parking and car care services, potentially providing cost savings compared to paying for parking and maintenance separately.	Charge fees for parking, which can vary depending on location and duration of parking.	Costs for maintenance and repair services can vary based on the type of service required and the extent of the repairs needed.
Customer Experience	Provides a seamless and user-friendly experience through the mobile app, allowing users to easily manage their subscriptions, access their cars remotely, and receive alerts and notifications about their vehicle's status.	Offer a more traditional experience with in-person interactions and manual processes for parking and payment.	Focus on providing personalized service and expert advice for maintenance and repair needs, but may not offer the same level of convenience or digital interaction as Safe Zone.
Strengths	Comprehensive solution offering parking, maintenance, and repair services through a convenient mobile app. Enhanced security features and additional services like on-demand maintenance.	Established physical presence in local communities, offering convenience and familiarity to users. Established customer base and brand reputation built over time.	Expertise and personalized service in maintenance and repair tasks. Established customer trust and loyalty due to long-standing presence in the community.
Weaknesses	Reliance on technology may pose adoption challenges among less tech-savvy users. Requires significant initial investment in infrastructure and technology development.	Limited scope of services compared to Safe Zone, focusing solely on parking. Relatively low-tech approach may not offer advanced security features or digital conveniences.	Limited scope of services compared to Safe Zone, focusing solely on maintenance and repair. Relatively low-tech approach may not offer advanced digital conveniences or security features.
Opportunities	Expansion into new markets and partnerships with car manufacturers or rental companies. Diversification of services to include additional automotive-related offerings.	Adoption of digital platforms and technology to enhance customer experience and service offerings. Expansion into new markets or diversification of services to capture additional revenue streams.	Adoption of digital platforms and technology to enhance customer experience and service offerings. Expansion into new markets or diversification of services to include additional automotive-related offerings.
Threats	Competition from existing parking facilities, repair shops, and potential new entrants. Regulatory changes and technological risks impacting service reliability.	Competition from newer, more technologically advanced solutions like Safe Zone. Changing consumer preferences and economic downturns impacting demand for parking services.	Competition from newer, more technologically advanced solutions like Safe Zone. Changing consumer preferences and economic downturns impacting demand for maintenance and repair services.

This chapter discusses the analysis and design phases of the project. It shows the different diagrams built to show how the system work, what does it consist of, and whom are the users.

3.1. System Analysis:

3.1.1. UML Diagrams:

Use case diagram:



Actors:

Customer: Users of the app who want to park and monitor their vehicles.

Car Care Center: Provides maintenance services and live feed of the car care process.

IT Admin: Manages the app's system, service packages, and customer details.

The Use Cases:

 # Customer

1- Signup/Login: To access the app and its features.

2- Reserve Parking: To find and reserve a parking space.

3- View Live Camera: To monitor their vehicle remotely. –

4- Receive Notifications: For updates on vehicle status and parking lot conditions. " Car Care Center"

5- Receive Service Requests: From customers for vehicle maintenance.

6- Access User Info: For servicing purposes.

7- Send Alerts: About vehicle status during servicing.

8- Provide Live Feed: Of the car care process upon request.

 # Admin

1- Manage Customer Details: For administrative purposes.

2- Edit Service Packages: To update service offerings.

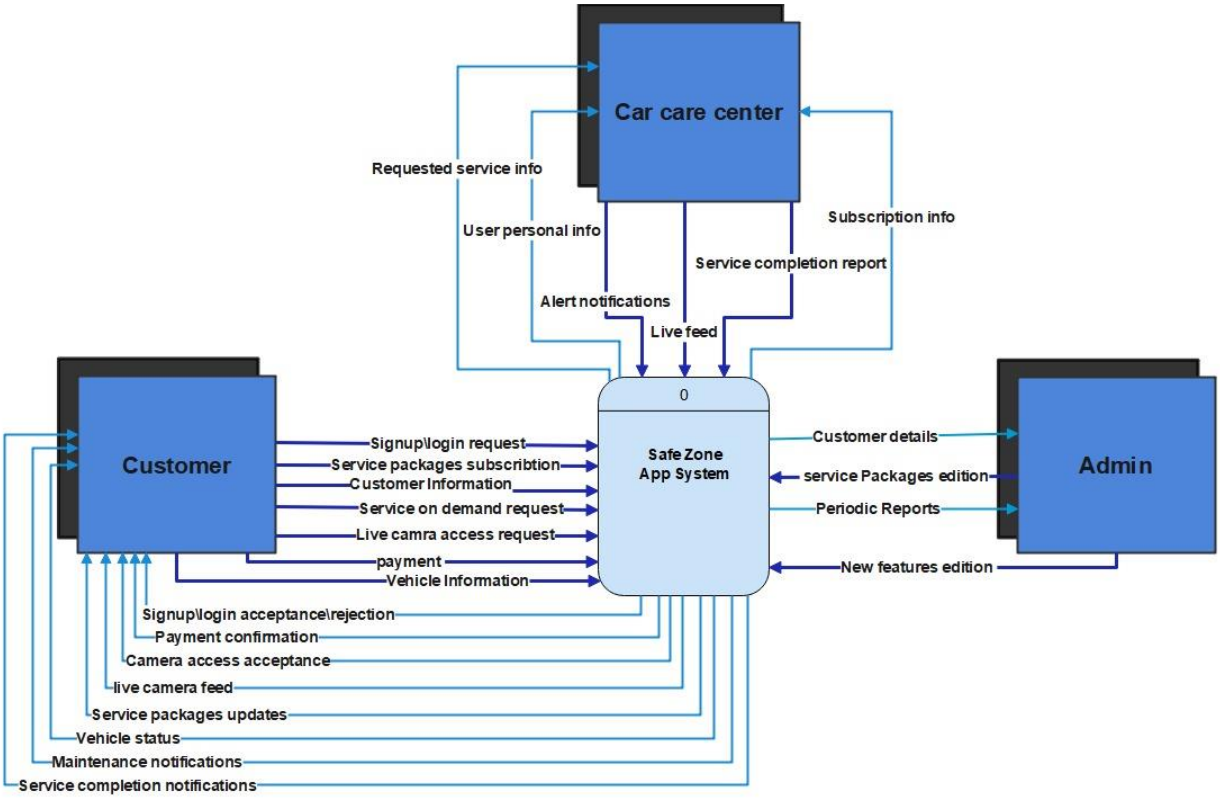
3- Generate Reports: For business analytics.

4- Update System Features: To enhance app functionality.

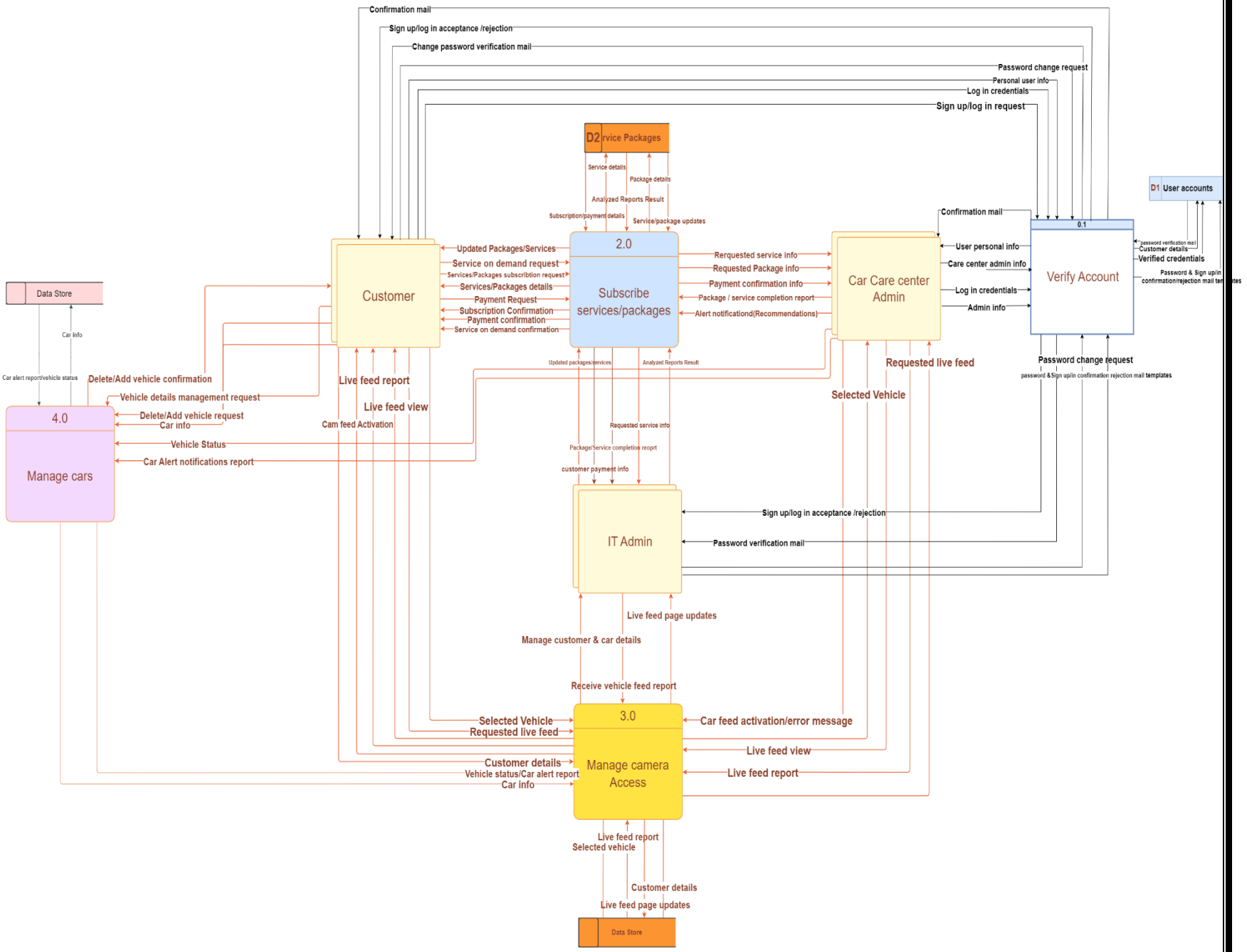
System - *Safe Zone App System**: Facilitates all interactions, processes payments, and provides real-time surveillance capabilities.

3.2.DFD:

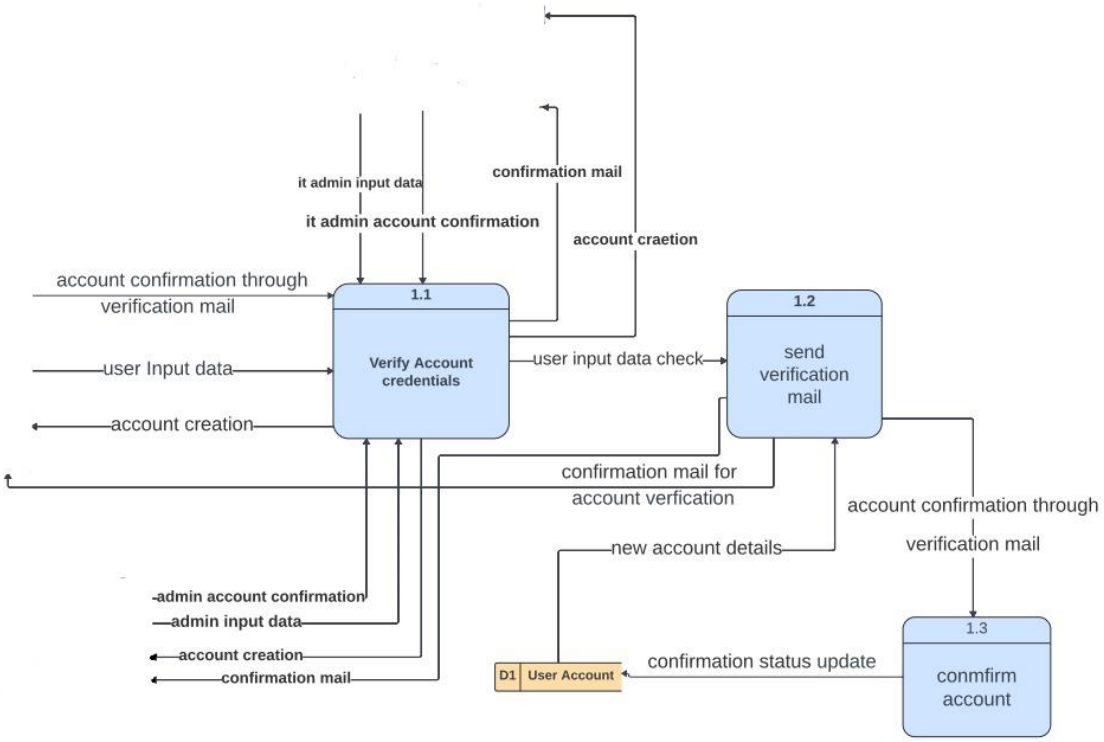
context diagram



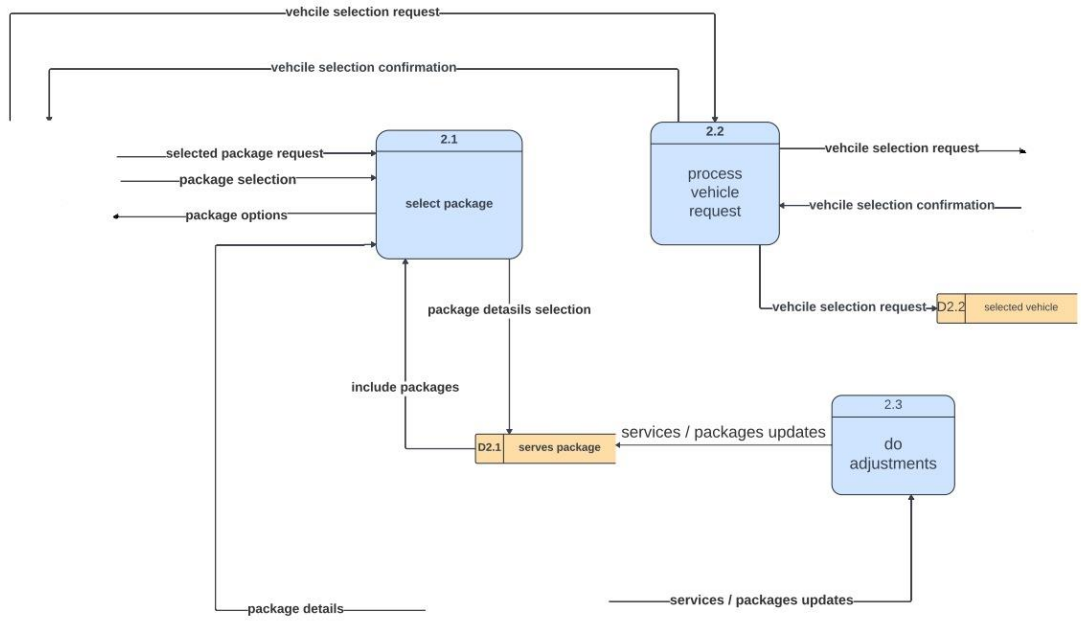
Level 0 DFD:


Confirmation mail
Sign up/log in acceptance /rejection
Change password verification mail
Password change request
Personal user info
Log in credentials
Sign up/log in request
D2 rvice Packages
Service details
Package details
Analyzed Reports Result
Subscription/payment details
Service/package updates
D1 User accounts
Confirmation mail
0.1
Verify Account
password verification mail
Customer details
Verified credentials
Password & Sign up/in confirmation/rejection mail templates
Data Store
Car Info
Car alert report/vehicle status
Delete/Add vehicle confirmation
Vehicle details management request
Delete/Add vehicle request
Car info
Vehicle Status
Car Alert notifications report
4.0
Manage cars
Customer
Updated Packages/Services
Service on demand request
Services/Packages subscribtion request
Services/Packages details
Payment Request
Subscription Confirmation
Payment confirmation
Service on demand confirmation
2.0
Subscribe services/packages
Rerquested service info
Requested Package info
Payment confirmation info
Package / service completion report
Alert notificationd(Recommendations)
Car Care center Admin
User personal info
Care center admin info
Log in credentials
Admin info
Live feed report
Live feed view
Cam feed Activation
Updated packages/services
Analyzed Reports Result
Requested live feed
Selected Vehicle
Password change request
password &Sign up/in confirmation rejection mail templates
Requested service info
Package/Service completion reoprt
customer payment info
IT Admin
Sign up/log in acceptance /rejection
Password verification mail
Live feed page updates
Manage customer & car details
Receive vehicle feed report
3.0
Manage camera Access
Selected Vehicle
Requested live feed
Customer details
Vehicle status/Car alert report
Car info
Car feed activation/error message
Live feed view
Live feed report
Live feed report
Selected vehicle
Customer details
Live feed page updates
Data Store

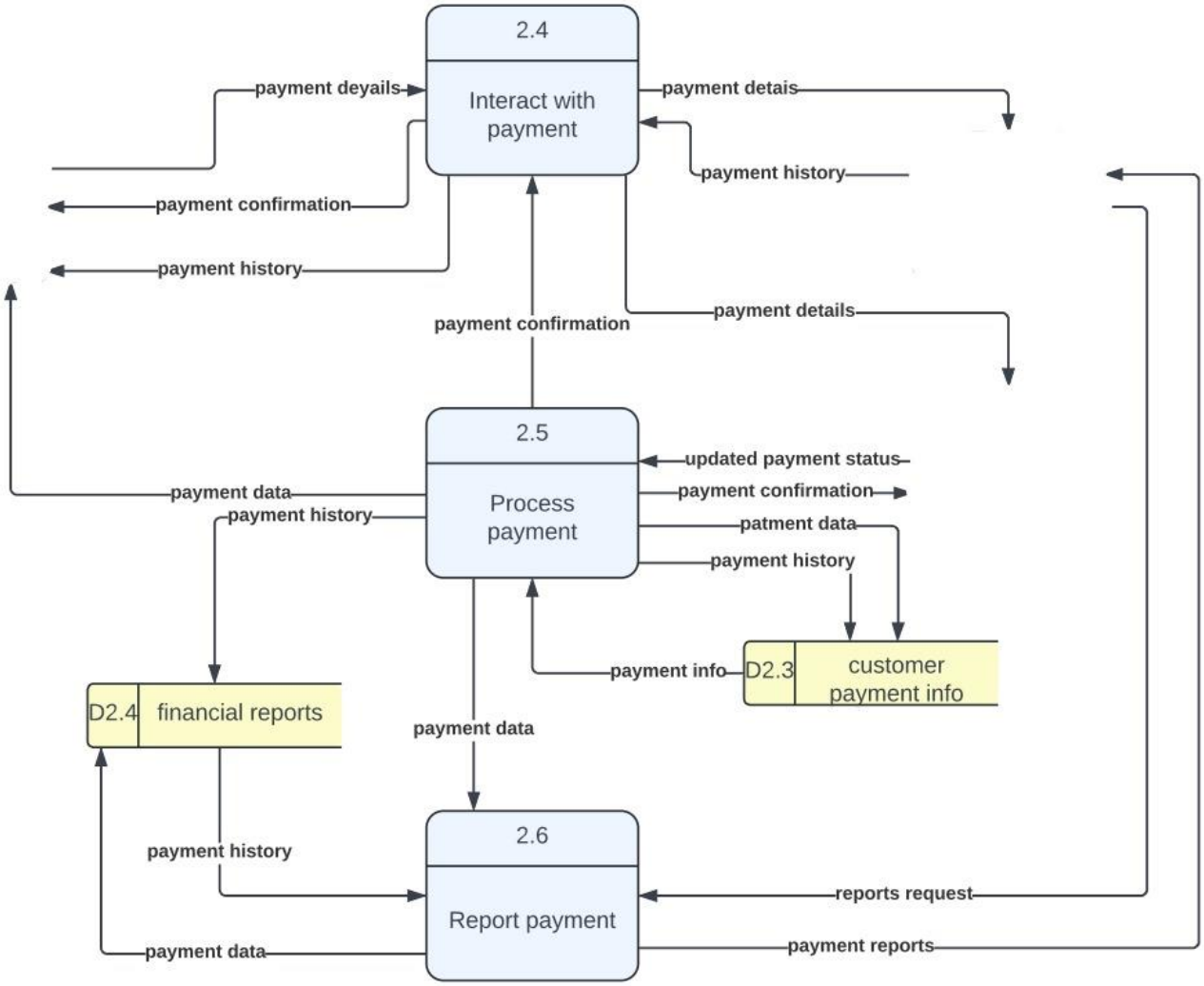
child diagrams of verify account process



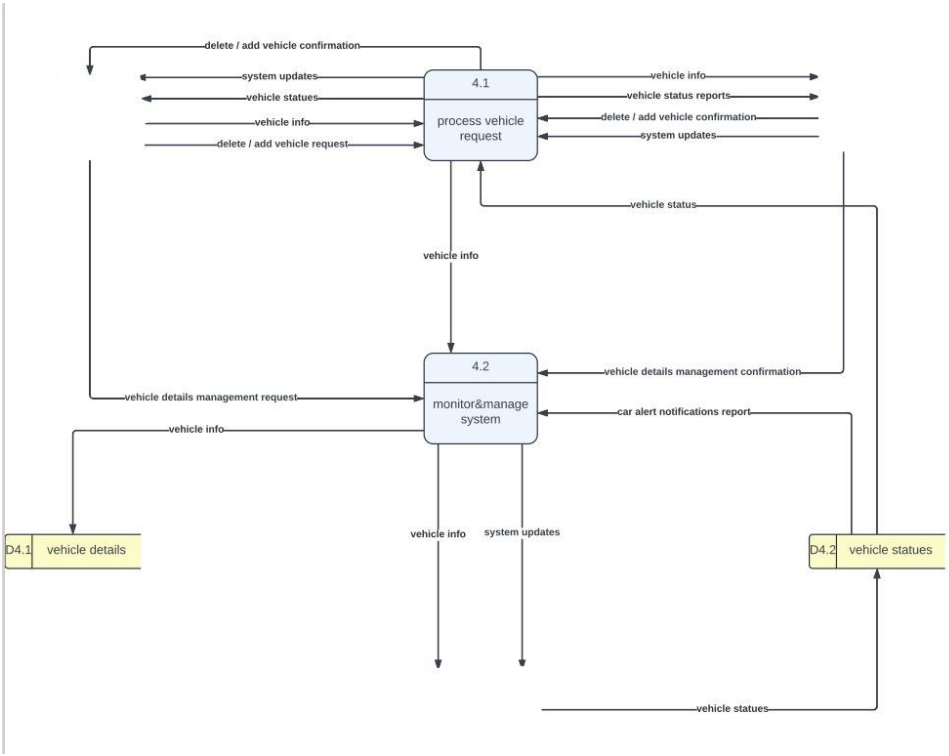
child diagrams of the Service/package process:



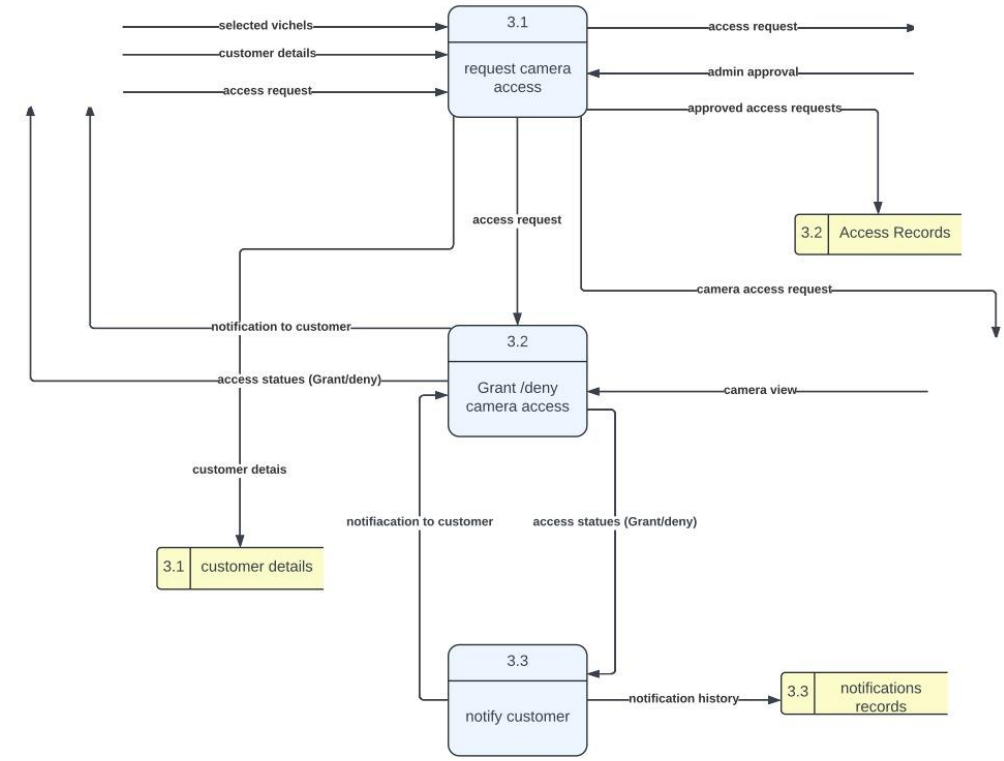
2nd child diagrams of subscribe services/ packages process:



child diagrams for manage cars process:

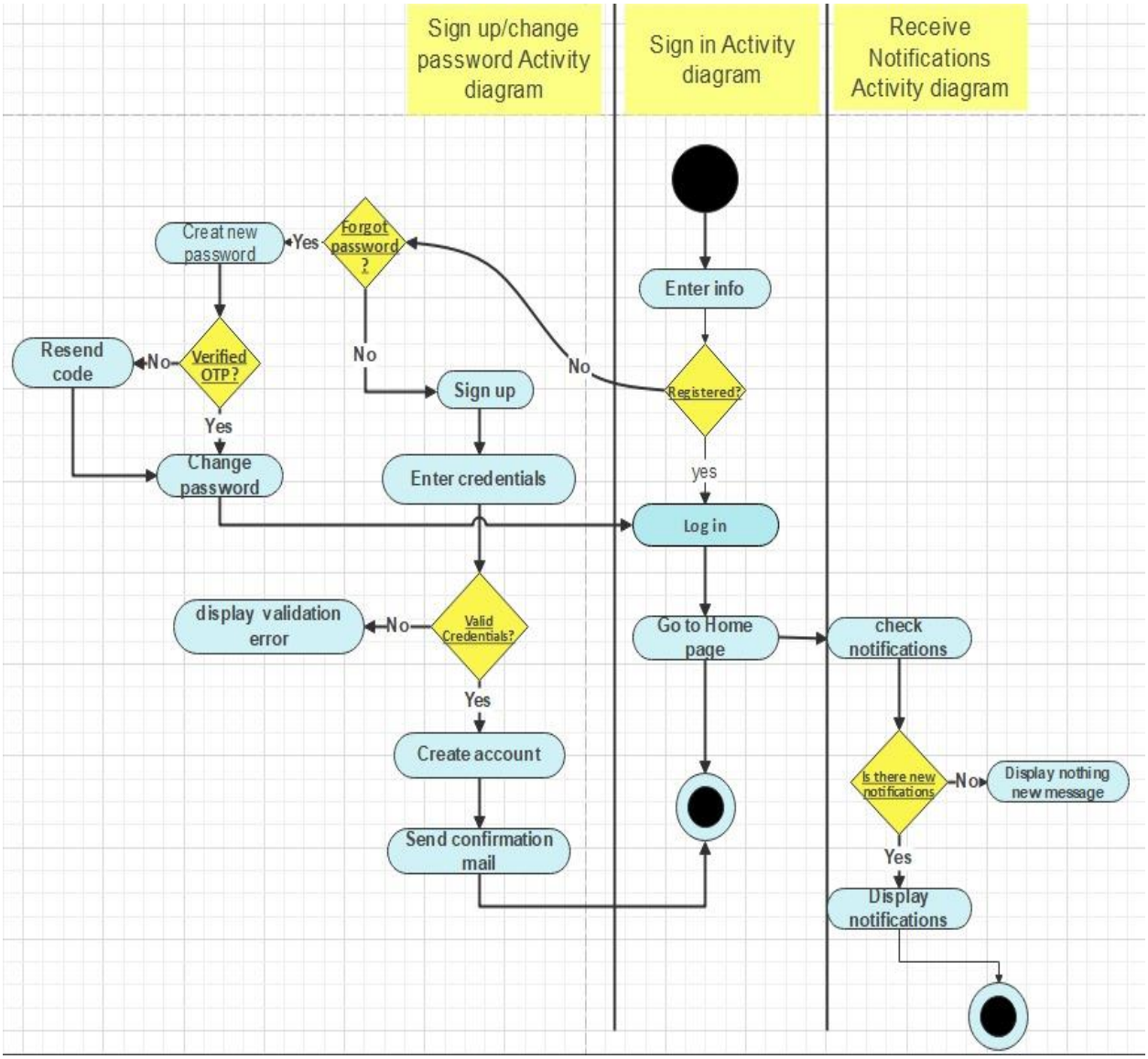


child diagrams for camera access process:



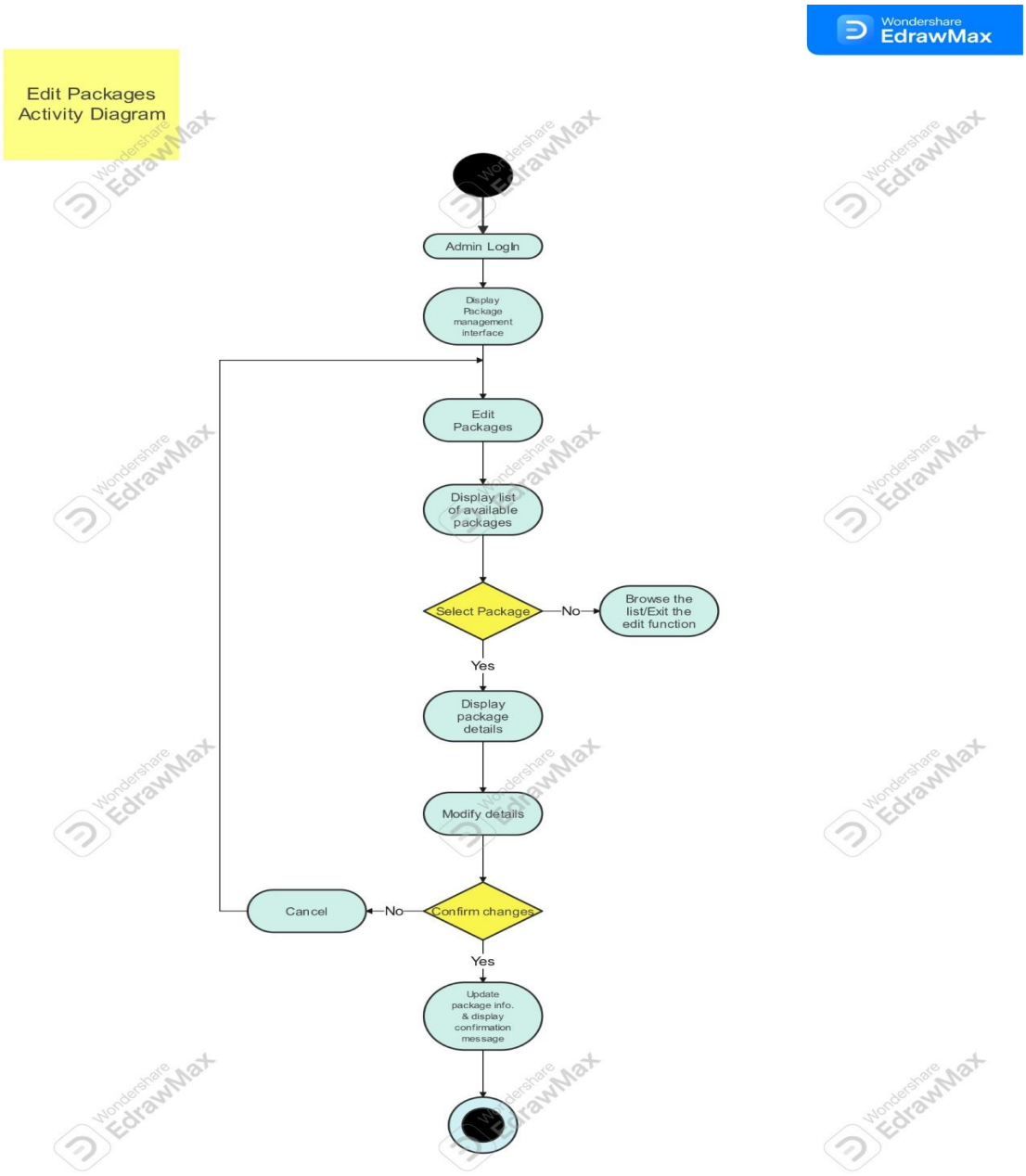
3.3. Activity Diagram:

sign up , sign in , change password & receive notifications activity diagrams:



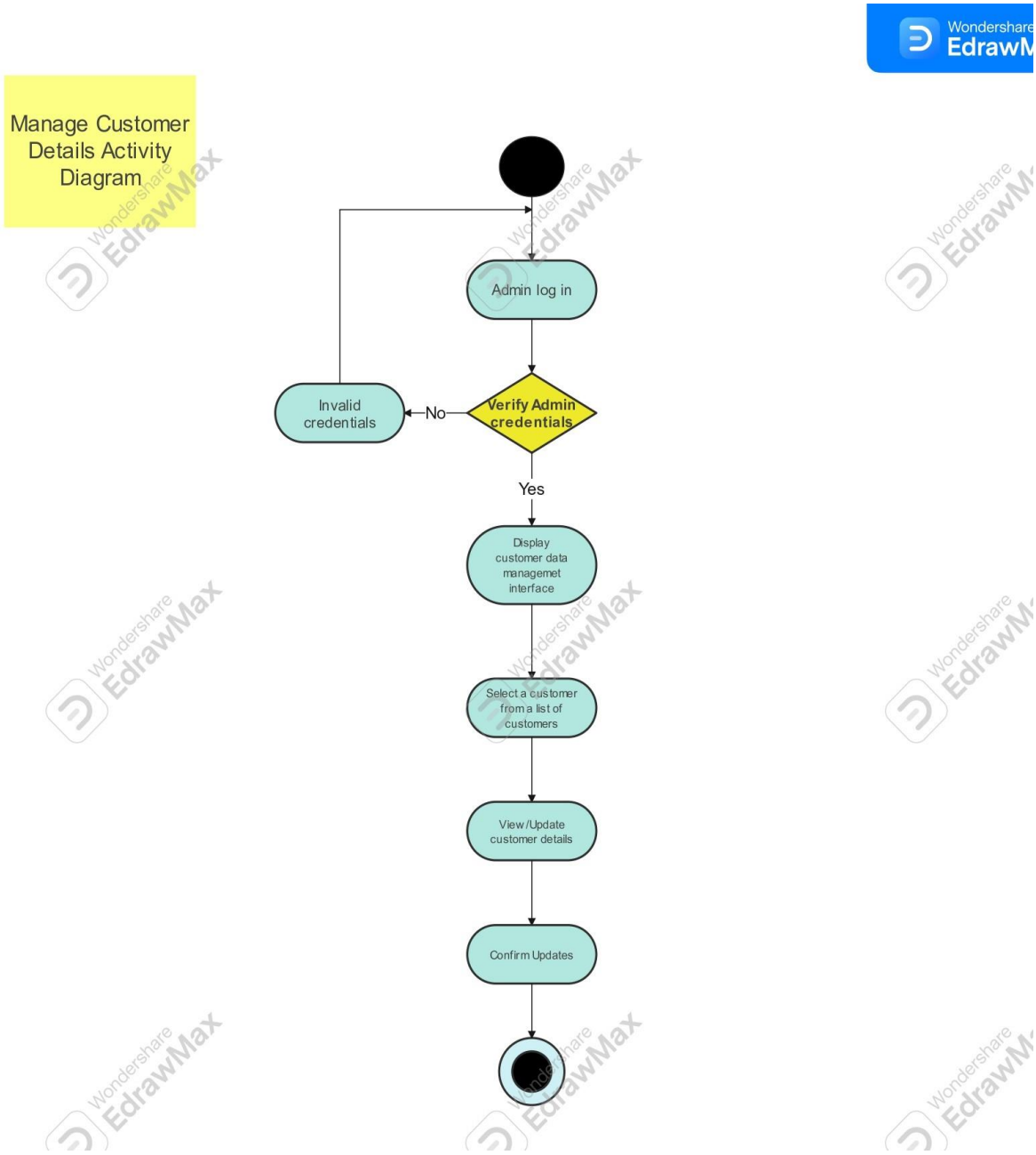
The flowchart depicts the process of adding a new vehicle and managing its details in a mobile app. Users can start by entering their details and capturing images of their car and license plate. Then, they can select their vehicle type, brand, model, and license plate number. Finally, the app saves the vehicle details securely.

Edit Packages Activity Diagram:



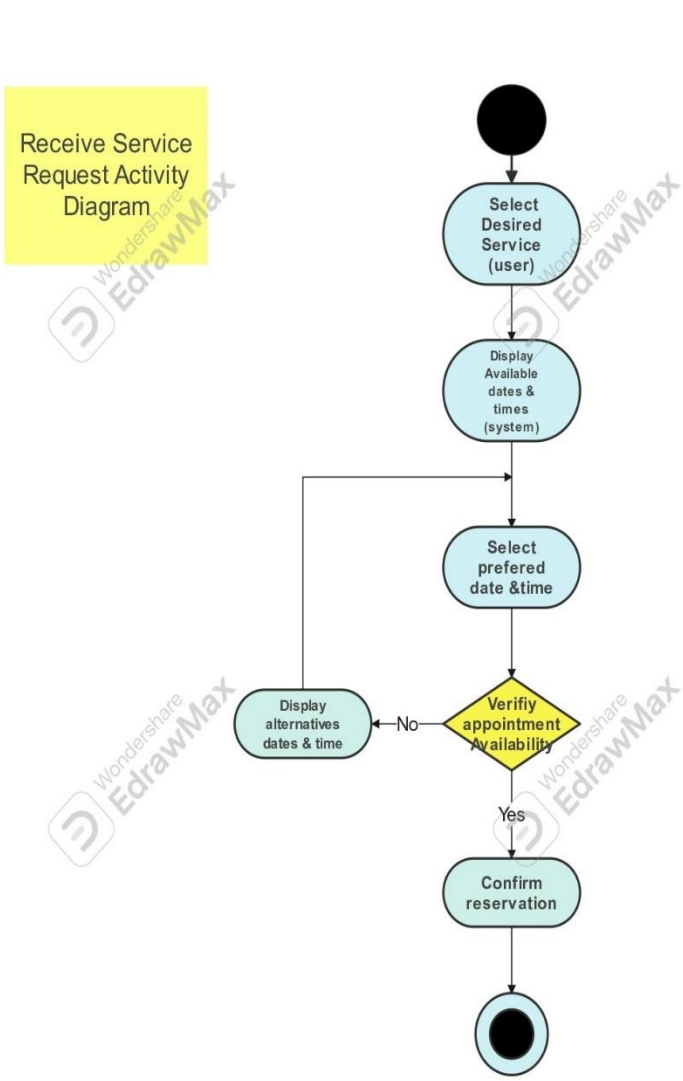
The flowchart outlines steps to subscribe to a service package. It starts with browsing available packages. The user then selects a package and confirms their subscription. If confirmed, they can proceed to payment. The user chooses a payment method and enters their information. Finally, the system processes the payment and updates the account.

Manage Customer Details Activity Diagram:



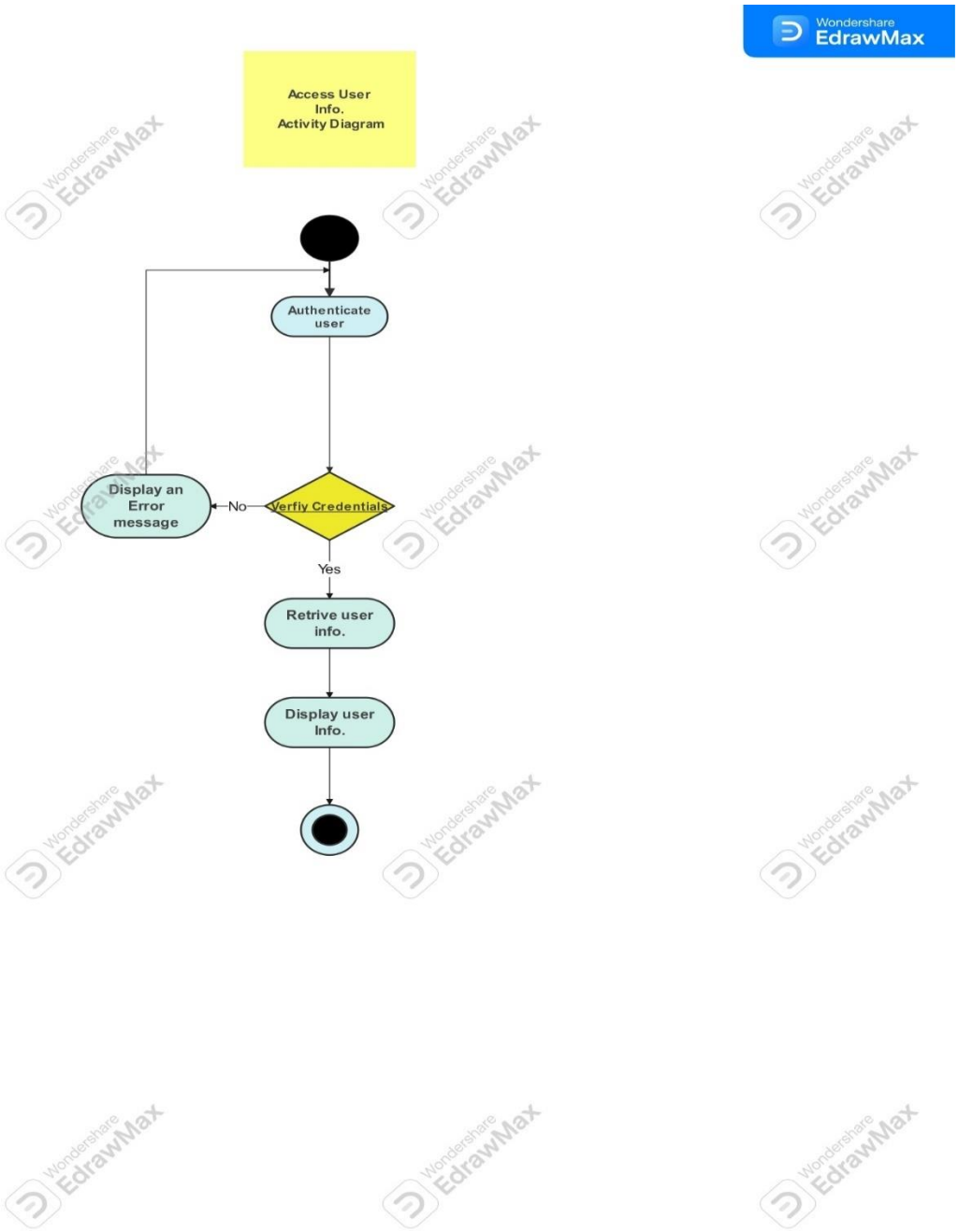
The image is a flowchart showing how to add a vehicle to a car service app. First, the user enters their details. Next, they capture images of their car and license plate. Then, they choose their vehicle information, including type, brand, model, and license plate number. Finally, the app saves the vehicle details securely.

Receive Service Request Activity Diagram:



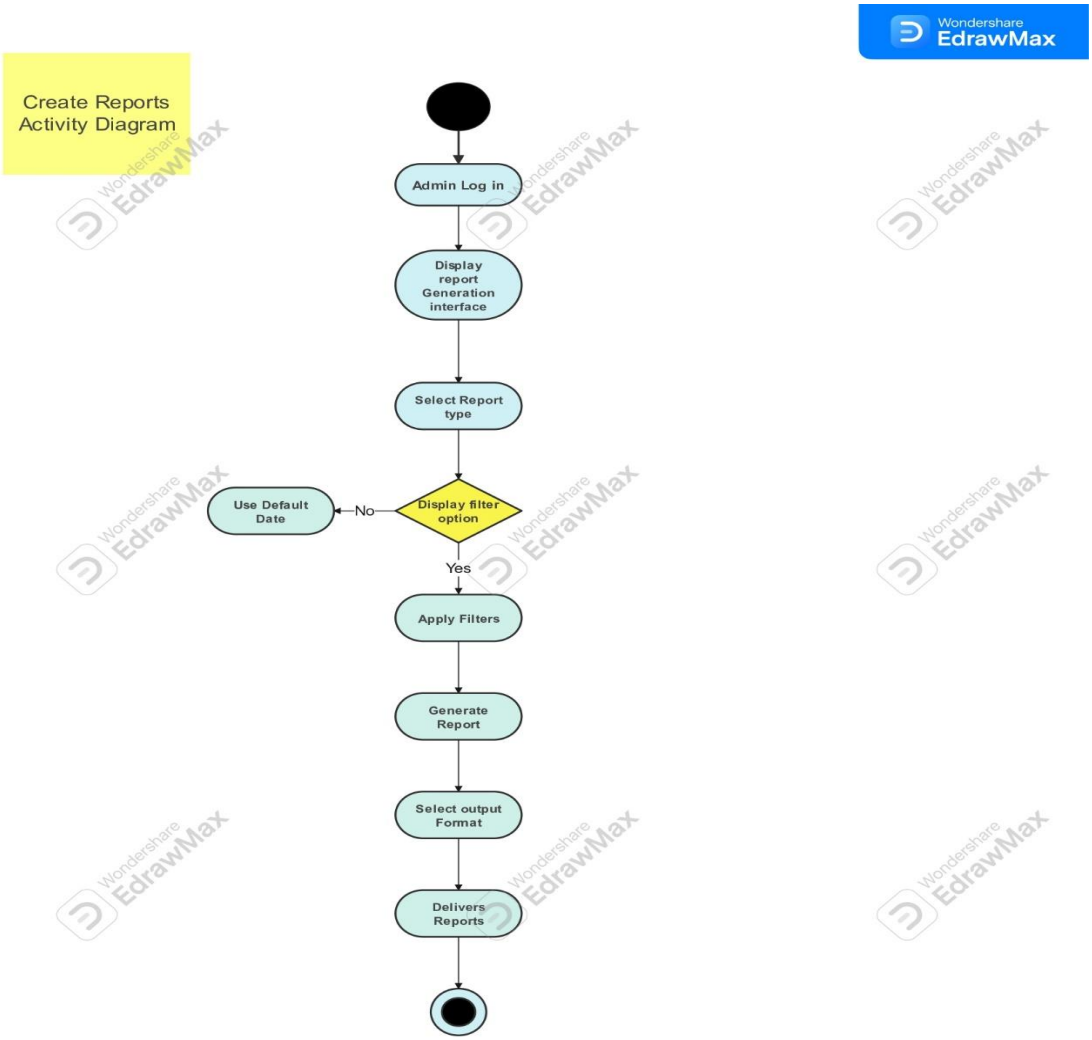
The image is a flowchart of a typical service request process. It begins with a customer submitting a service request. Next, the system displays a list of available services for the customer to choose from. The customer then selects their desired service and a preferred date and time. The system checks availability and displays confirmation or offers alternative dates and times. Finally, the customer confirms the reservation.

Access user Info Diagram:



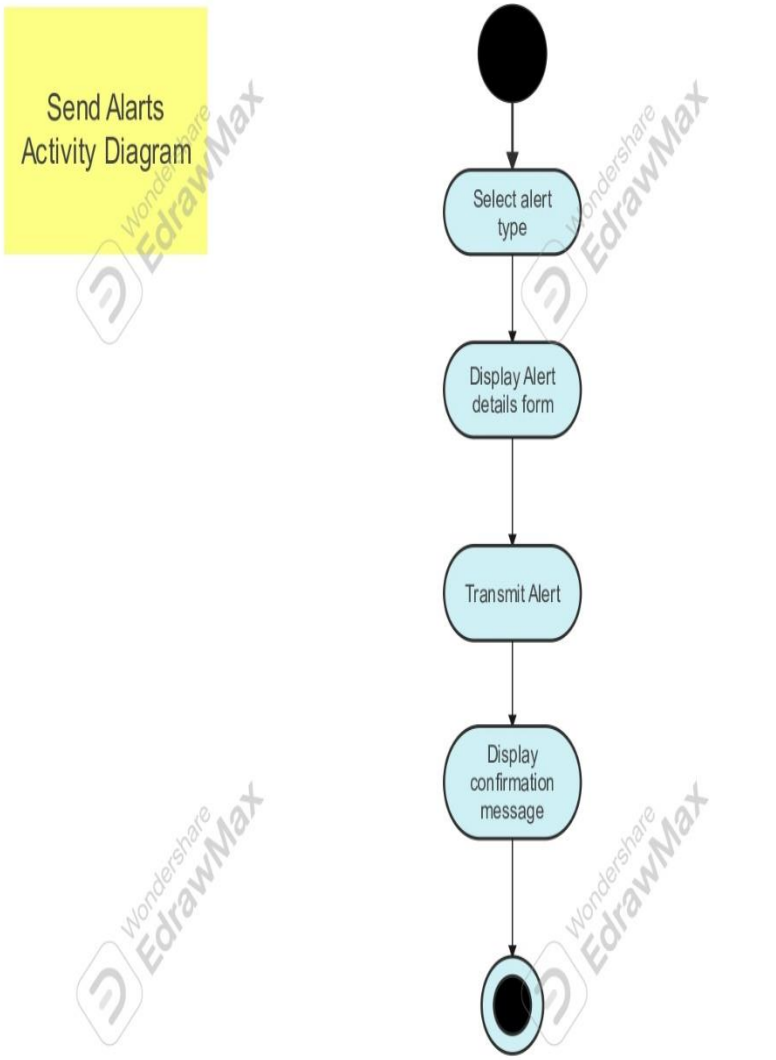
The flowchart outlines the process of adding a new vehicle to a mobile app. It begins with fetching the user's name. Then, the user adds images of their car and license plate. Next, they enter details like vehicle type, brand, model, and license plate number. Finally, the app saves the vehicle information securely.

Create Reports Activity Diagram:



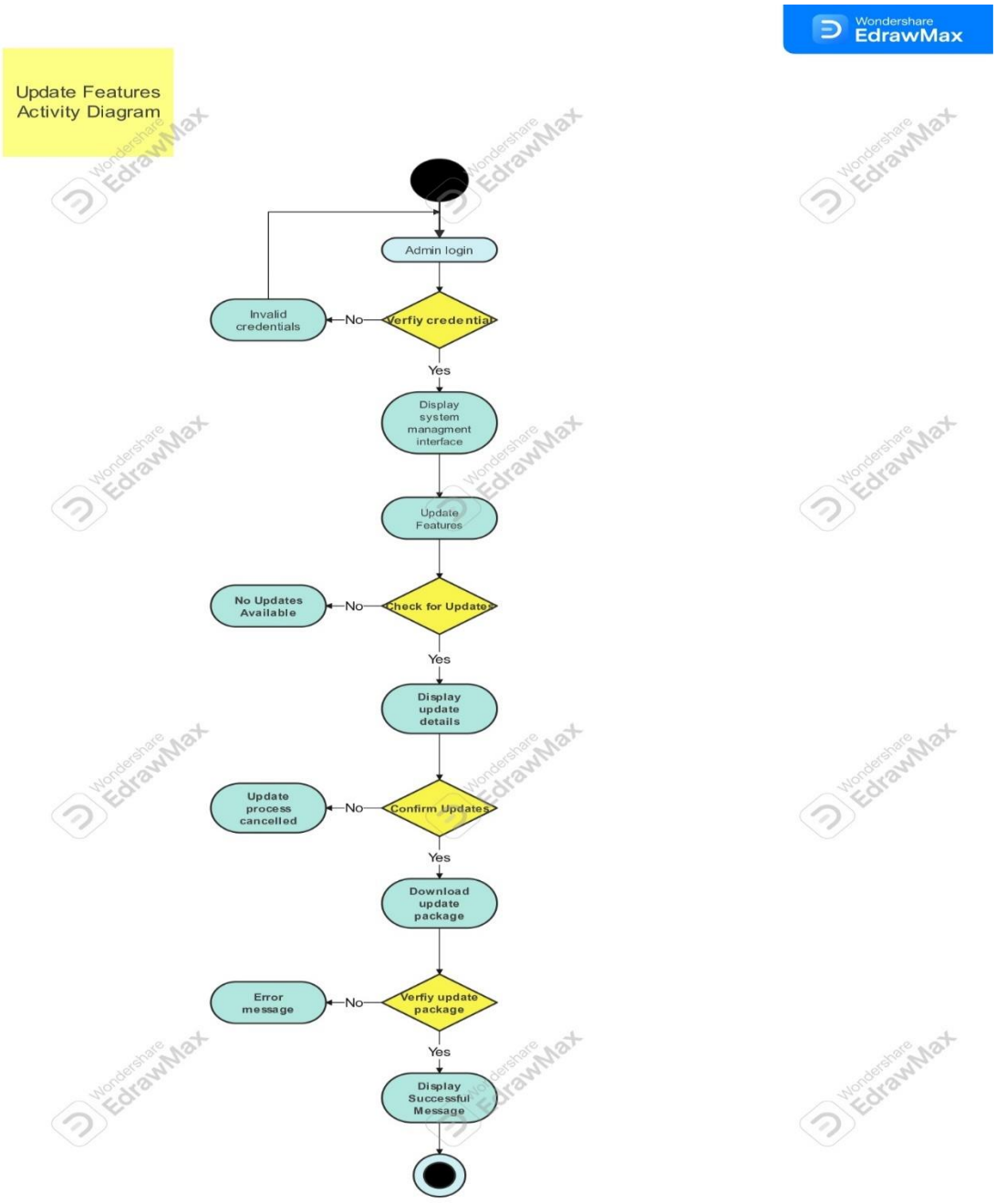
This flowchart outlines adding a vehicle to a car service app. First, users input their details. They then capture images of their car and license plate. Next, they select their vehicle's type, brand, model, and license plate number. Finally, the app securely saves the vehicle data.

Send Alerts Activity Diagram:



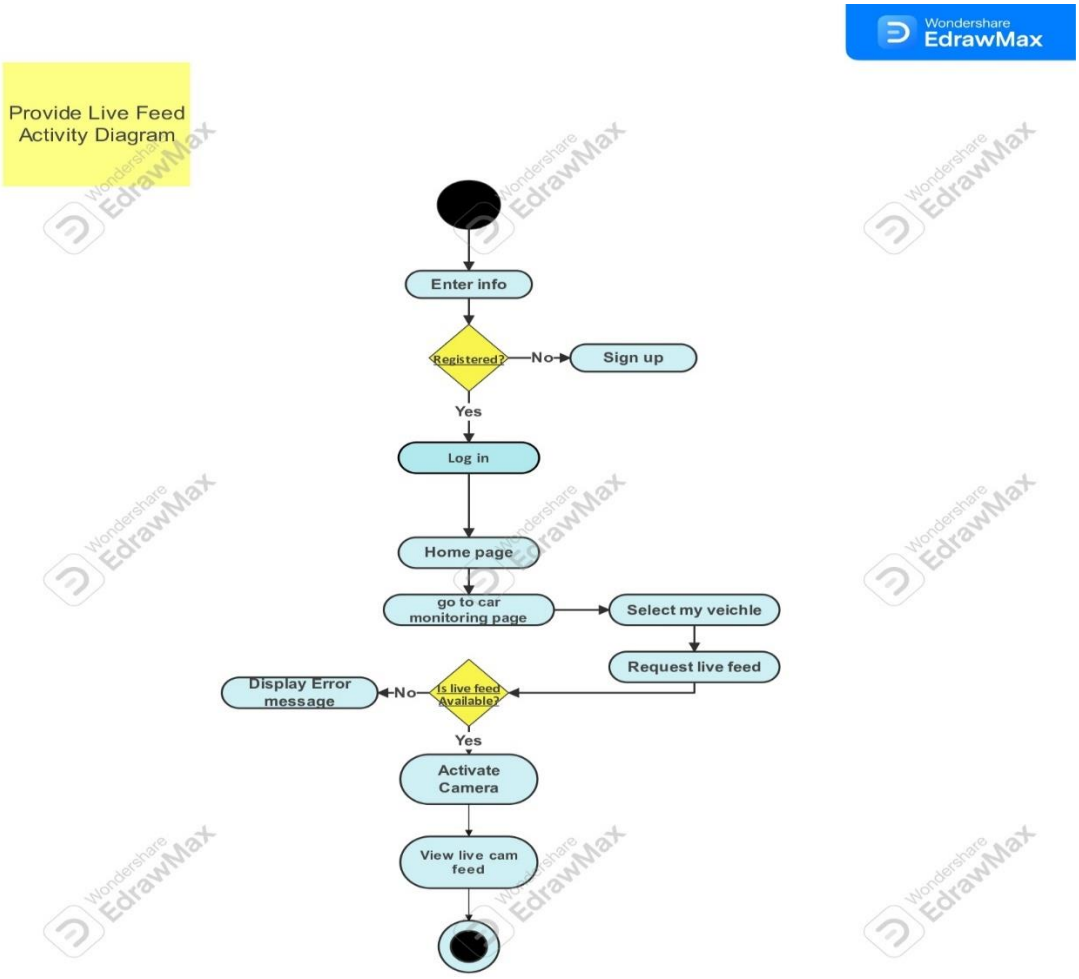
The flowchart outlines adding a vehicle to a car service app. First, the user fetches their user data. Then, they capture images of their car and license plate. Next, they enter details like vehicle type, brand, model, and license plate number. Finally, the app securely saves the vehicle information.

Update Features Activity Diagram:



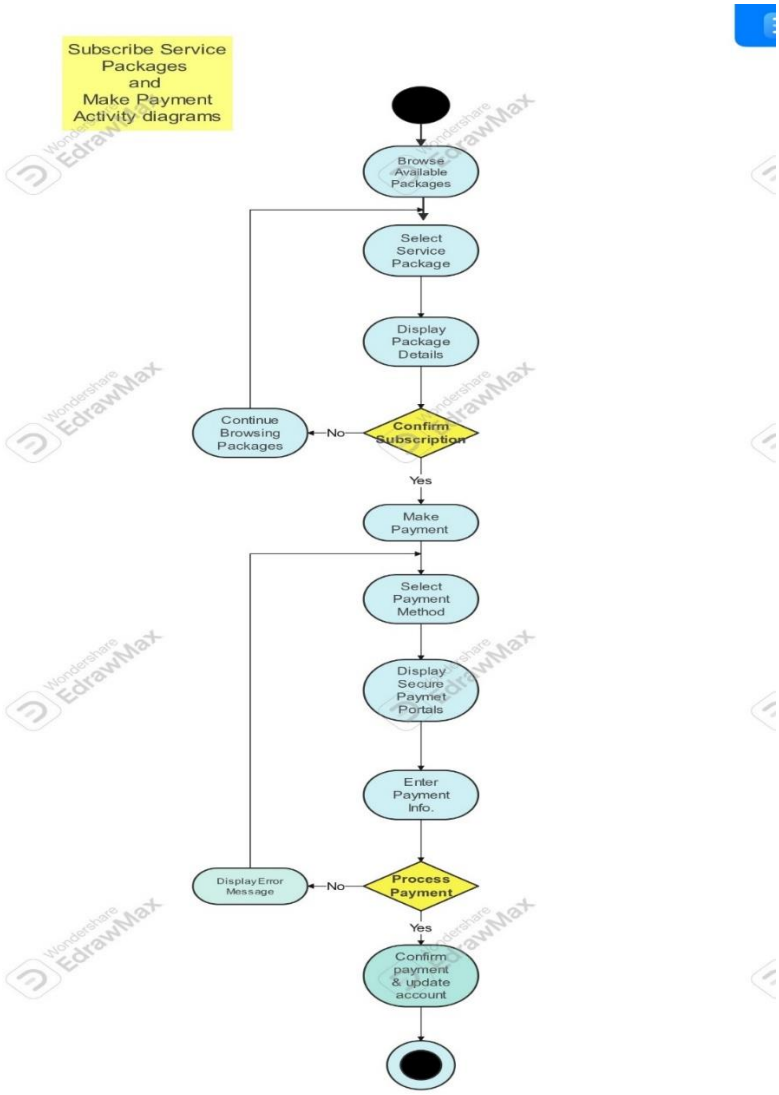
This flowchart outlines adding a vehicle to a car service app. First, the app fetches the user's name. Then, the user captures images of their car and license plate. Next, they select their vehicle information, including type, brand, model, and license plate number. Finally, the app securely saves the vehicle details.

Provide Live Feed Activity Diagram:



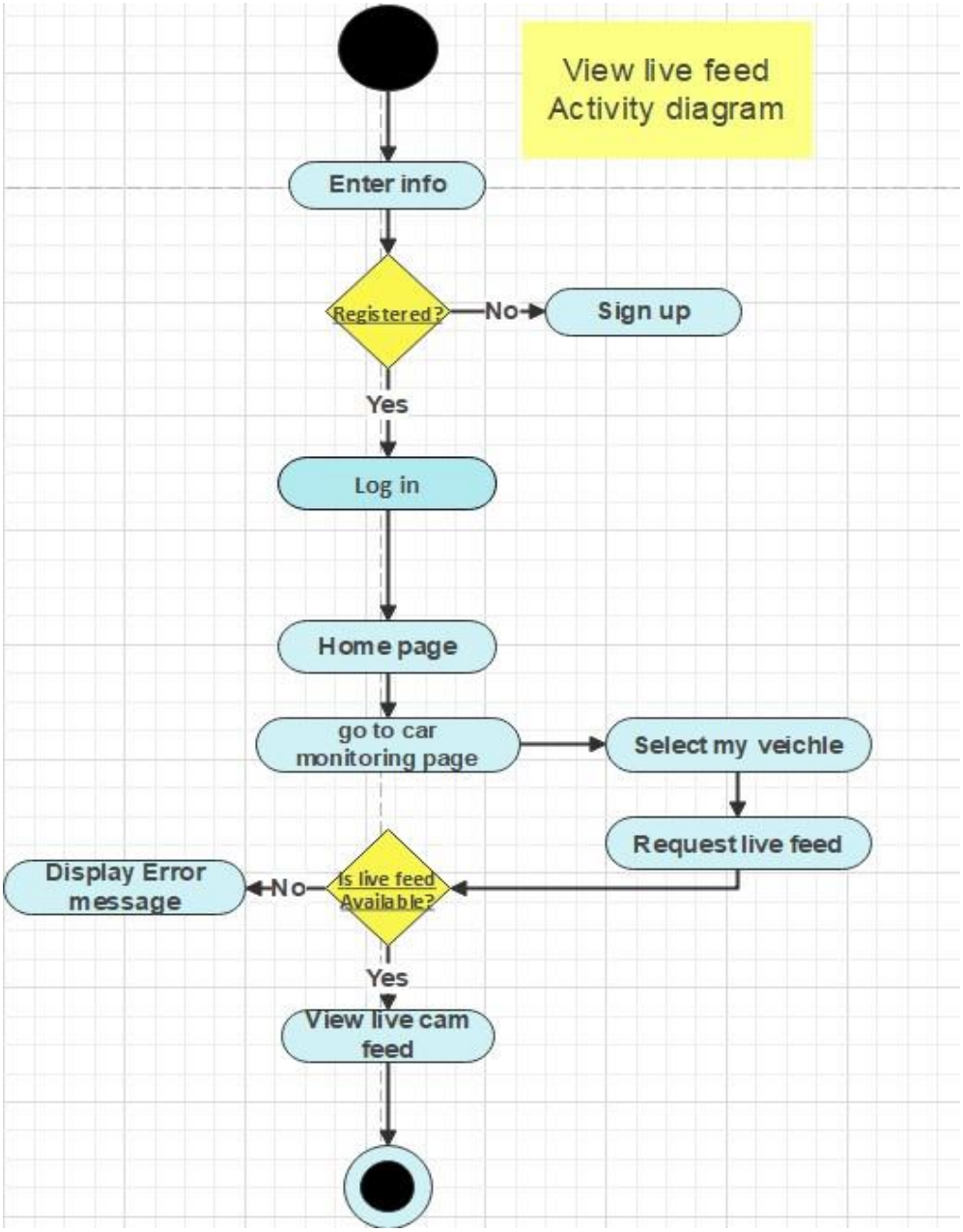
This flowchart outlines adding a vehicle to a car service app. First, users provide their details. They then capture images of their car and license plate. Next, they choose their vehicle's make, model, and license plate number. Finally, the app securely saves the vehicle data.

Subscribe Service Package and Make Payment Diagram:



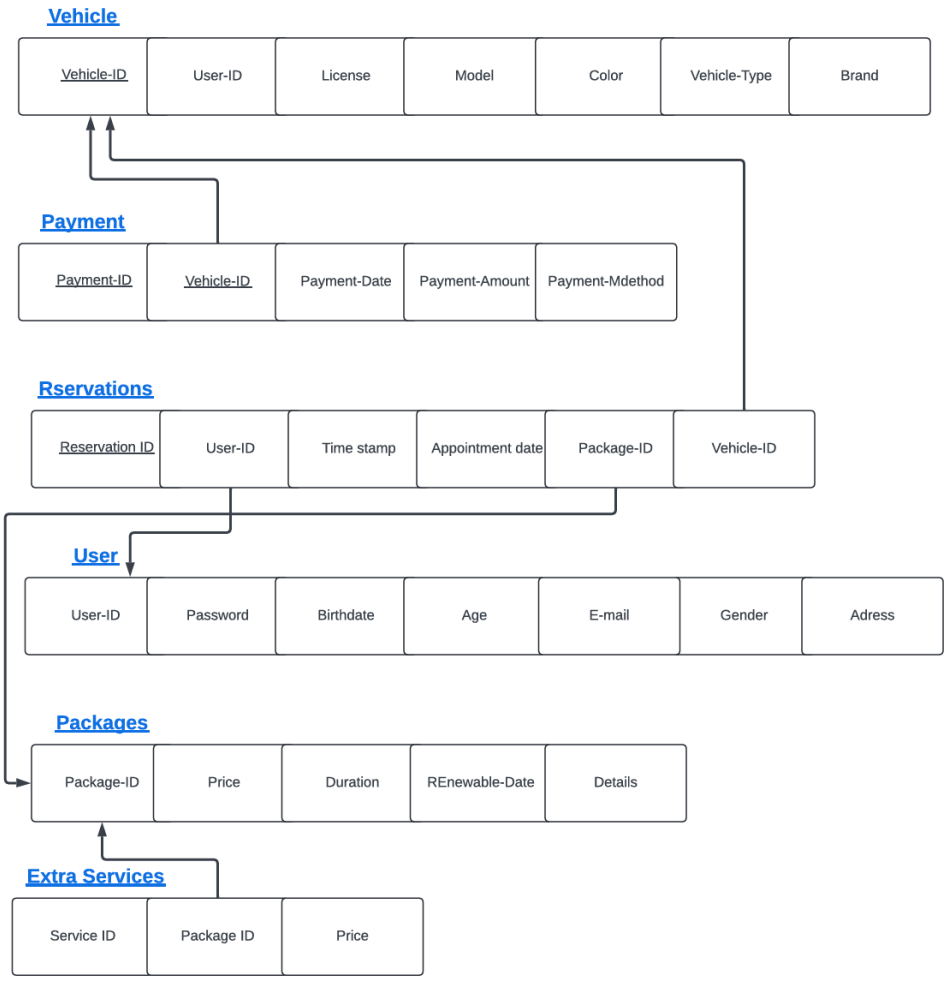
The flowchart shows how to subscribe to a service package and make a payment. It starts with browsing available service packages. Then, the user selects a service package and confirms the subscription. If the user confirms, they can proceed to make a payment. The user selects a payment method and enters their payment information. Finally, the system processes the payment and updates the account.

View live feed Activity Diagram:

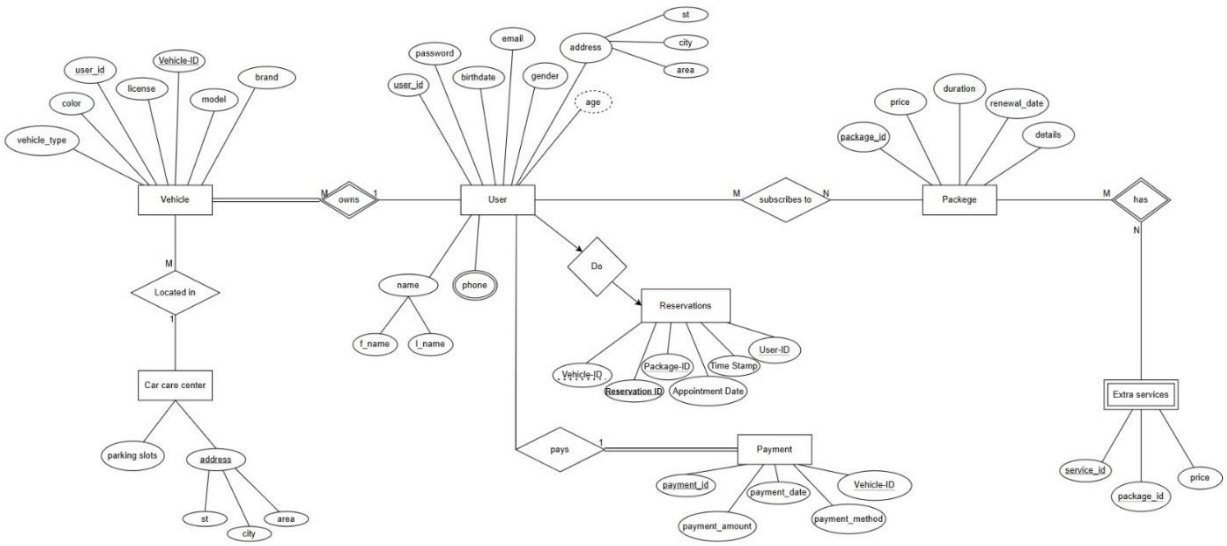


The flowchart depicts steps to view a live feed on a website.
First, enter the website and log in. If successful, proceed to the homepage and select the desired live feed. Finally, request the live feed. An error message appears if the feed is unavailable.

Schema:

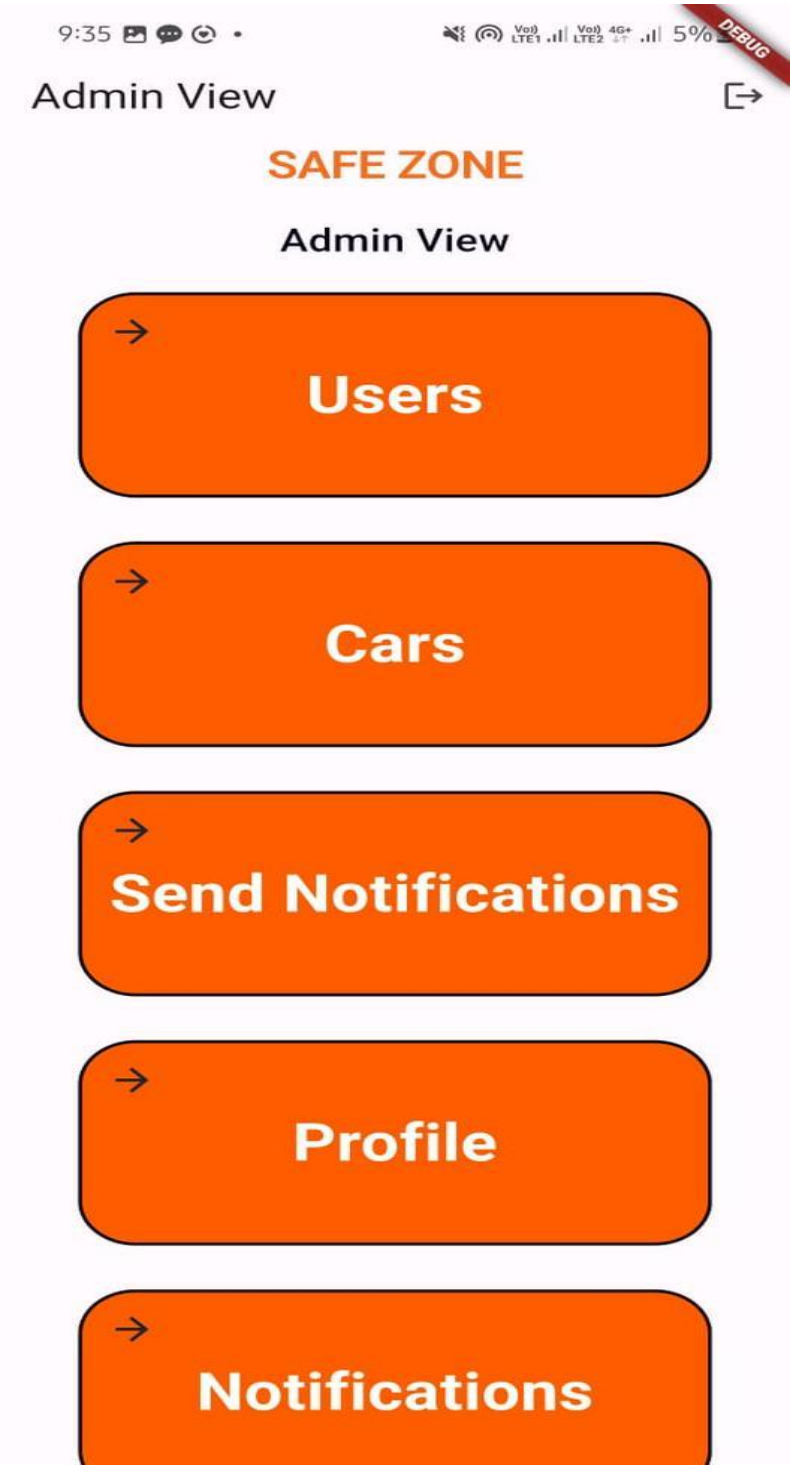

Vehicle
Vehicle-ID
User-ID
License
Model
Color
Vehicle-Type
Brand
Payment
Payment-ID
Vehicle-ID
Payment-Date
Payment-Amount
Payment-Mdethod
Rservations
Reservation ID
User-ID
Time stamp
Appointment date
Package-ID
Vehicle-ID
User
User-ID
Password
Birthdate
Age
E-mail
Gender
Adress
Packages
Package-ID
Price
Duration
REnewable-Date
Details
Extra Services
Service ID
Package ID
Price

Erd model:


Vehicle
user_id
Vehicle-ID
license
brand
color
model
vehicle_type
owns
M
1
User
password
email
address
st
city
area
user_id
birthdate
gender
age
subscribes to
M
N
Packege
price
duration
renewal_date
package_id
details
has
M
N
Extra services
service_id
package_id
price
Located in
M
1
Car care center
parking slots
address
st
city
area
name
f_name
l_name
phone
Do
Reservations
User-ID
Vehicle-ID
Package-ID
Time Stamp
Reservation ID
Appointment Date
pays
1
Payment
payment_id
payment_date
Vehicle-ID
payment_amount
payment_method

Chapter 4 (Implementation)

4.1. Admin View Page



Admin view page

Overall Description:

The Admi View class presents an admin dashboard with options to view user data, car information, send notifications, access the profile, and view received notifications. Each feature is represented by a stylized frame that allows the admin to navigate to specific pages upon tapping.

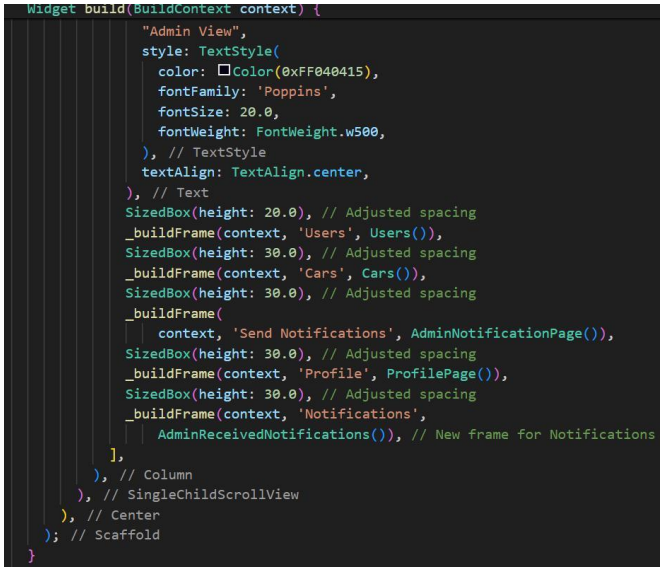
This Flutter app is designed to provide a user-friendly interface for admins to manage different aspects of the system effectively

```dart
import 'package:flutter/material.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:safe_zone/admin_received_notifications.dart';
import 'users.dart';
import 'cars.dart';
import 'profile.dart';
import 'admin_notifications.dart';
```

1-Imports:

Description: Importing necessary packages and files for the admin view functionality.

```
class AdminView extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Admin View'),
        actions: [
          IconButton(
            icon: Icon(Icons.logout),
            onPressed: () {
              FirebaseAuth.instance.signOut();
            },
          ), // IconButton
        ],
      ), // AppBar
      body: Center(
        child: SingleChildScrollView(
          child: Column(
            mainAxisAlignment: MainAxisAlignment.center,
            children: [
              Text(
                'SAFE ZONE',
                style: TextStyle(
                  color: Color(0xFFF86C1D),
                  fontFamily: 'Poppins',
                  fontSize: 24.0,
                  fontWeight: FontWeight.w600,
```

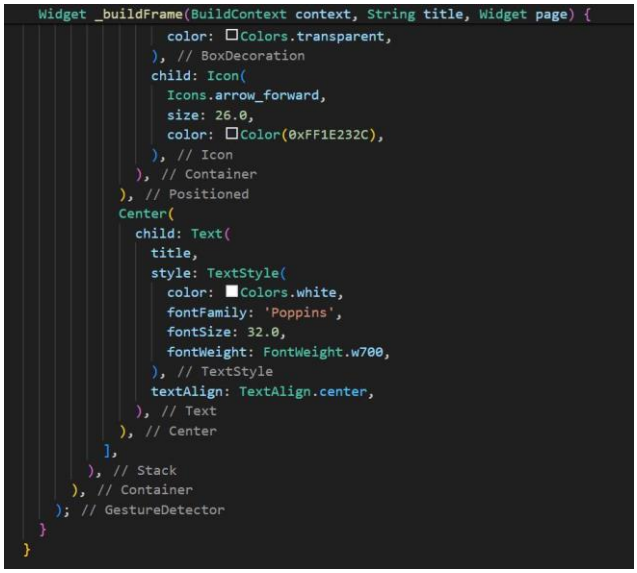


```
Widget build(BuildContext context) {
                "Admin View",
                style: TextStyle(
                  color: Color(0xFF040415),
                  fontFamily: 'Poppins',
                  fontSize: 20.0,
                  fontWeight: FontWeight.w500,
                ), // TextStyle
                textAlign: TextAlign.center,
              ), // Text
              SizedBox(height: 20.0), // Adjusted spacing
              _buildFrame(context, 'Users', Users()),
              SizedBox(height: 30.0), // Adjusted spacing
              _buildFrame(context, 'Cars', Cars()),
              SizedBox(height: 30.0), // Adjusted spacing
              _buildFrame(
                  context, 'Send Notifications', AdminNotificationPage()),
              SizedBox(height: 30.0), // Adjusted spacing
              _buildFrame(context, 'Profile', ProfilePage()),
              SizedBox(height: 30.0), // Adjusted spacing
              _buildFrame(context, 'Notifications',
                  AdminReceivedNotifications()), // New frame for Notifications
            ],
          ), // Column
        ), // SingleChildScrollView
      ), // Center
    ); // Scaffold
  }
```

2-AdminView Class:

Description: Defines a stateless widget representing the admin view interface.

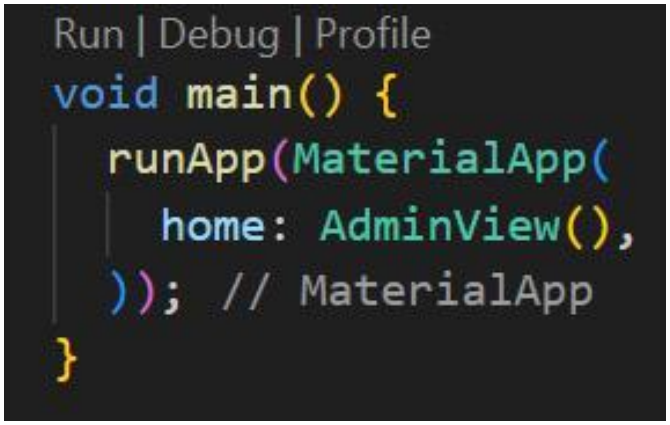
```
Widget _buildFrame(BuildContext context, String title, Widget page) {
  return GestureDetector(
    onTap: () {
      Navigator.push(
        context,
        MaterialPageRoute(builder: (context) => page),
      );
    },
    child: Container(
      width: 330.0,
      height: 140.0,
      decoration: BoxDecoration(
        borderRadius: BorderRadius.circular(30.0),
        color: Color(0xFFFF5C00),
        border: Border.all(
          color: Colors.black,
          width: 2.0,
        ), // Border.all
      ), // BoxDecoration
      child: Stack(
        children: [
          Positioned(
            left: 13.0,
            top: 10.0,
            child: Container(
              width: 20.0,
              height: 30.0,
              decoration: BoxDecoration(
                color: Colors.transparent,
              ), // BoxDecoration
```



```
Widget _buildFrame(BuildContext context, String title, Widget page) {
                color: Colors.transparent,
              ), // BoxDecoration
              child: Icon(
                Icons.arrow_forward,
                size: 26.0,
                color: Color(0xFF1E232C),
              ), // Icon
            ), // Container
          ), // Positioned
          Center(
            child: Text(
              title,
              style: TextStyle(
                color: Colors.white,
                fontFamily: 'Poppins',
                fontSize: 32.0,
                fontWeight: FontWeight.w700,
              ), // TextStyle
              textAlign: TextAlign.center,
            ), // Text
          ), // Center
        ],
      ), // Stack
    ), // Container
  ); // GestureDetector
}
```

3-_buildFrame() Widget Function:

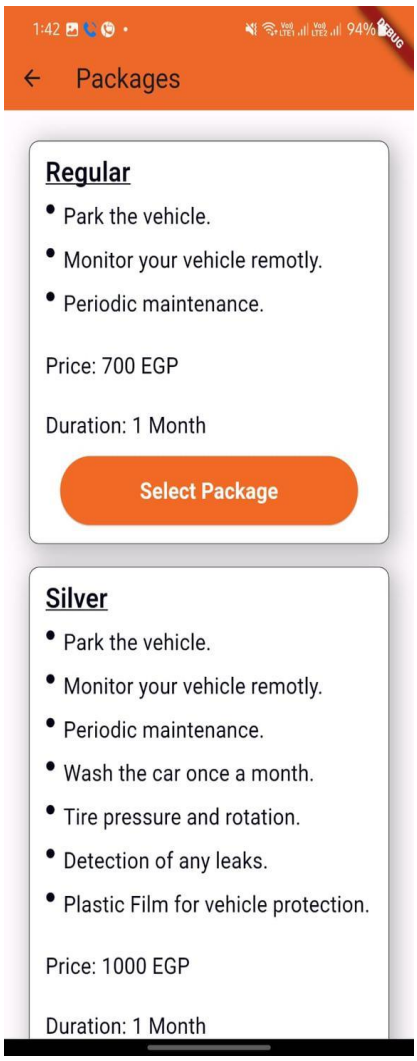
Description: Creates a stylized clickable frame widget for different functionalities.

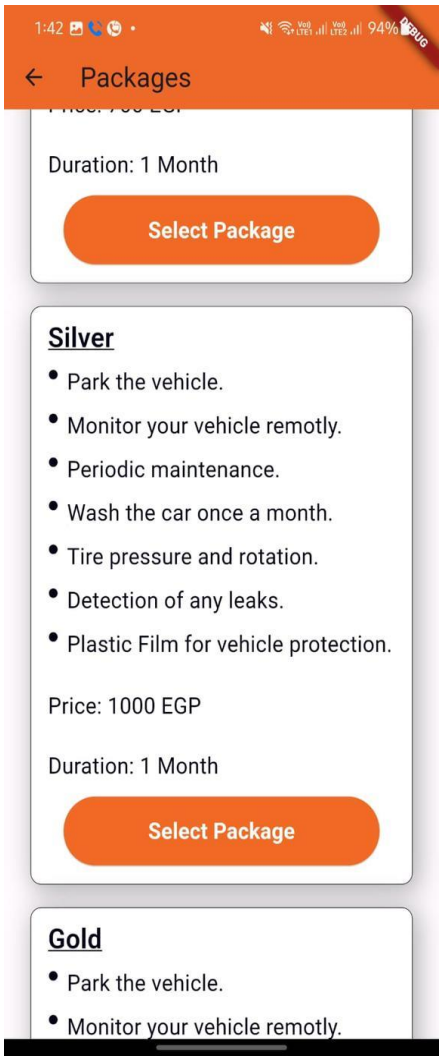


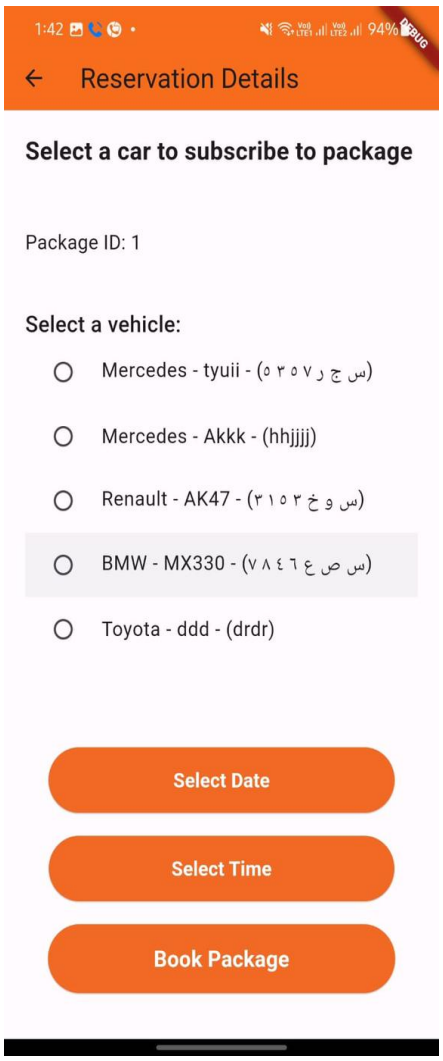
4-Main Function:

Description: Initializes the app with the AdminView widget as the starting screen.

4.2. Package Reservation







This Flutter app lets users manage package reservations for vehicles. Here's how it works:

- Users can browse available packages with details like name, services, price, and duration.
- They can select a package and proceed to make a reservation.
- The app allows them to choose a vehicle from their list (fetched from Firebase).
- Users can then select a date, time, and book the package.
- The app checks for available slots before saving the reservation and notifies the admin upon booking.

Overall, the app offers a user-friendly way to manage package reservations with real-time data updates using Firebase.

```dart
import 'package:flutter/material.dart';
import 'package:firebase_core/firebase_core.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:safe_zone/home.dart';

Run | Debug | Profile
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();
  runApp(Reservation());
}
```

1. Initial Setup and Firebase Initialization:

Description: This snippet sets up the initial Flutter app, initializes Firebase services, and starts the app with the Reservation widget.

```dart
class Reservation extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Packages(),
    ); // MaterialApp
  }
}
```

2. Reservation Class Definition:

Description: Defines a stateless widget Reservation that sets the Packages widget as the home screen of the app.

```
            children: packages.map<Widget>((package) {
              var packageName = package['package_name'];
              var packageServices =
                  List<String>.from(package['package_services']);
              var packageId = package.id;
              var packagePrice = package['price'];
              var packageDuration = package['duration'];
              return _buildPackageBox(context, packageName, packageServices
                  packageId, packagePrice, packageDuration);
            }).toList(),
          ); // Column
        },
      ), // StreamBuilder
    ), // SingleChildScrollView
  ); // Scaffold
}

Widget _buildPackageBox(BuildContext context, String title,
    List<String> features, String packageId, packagePrice, packageDuration)
  return Container(
    width: double.infinity,
    decoration: BoxDecoration(
      borderRadius: BorderRadius.circular(10.0),
      border: Border.all(color: Colors.black, width: 0.5),
      color: Color.fromRGBO(255, 255, 255, 1),
      boxShadow: [
        BoxShadow(
          color: Colors.black.withOpacity(0.3),
          offset: Offset(4.0, 4.0),
          blurRadius: 20.0,
```

```
class Packages extends StatelessWidget {
  final String? selectedPackageId; // Package ID selected by the user

  Packages({this.selectedPackageId});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Packages'),
        backgroundColor: Color(0xFFF86C1D),
      ), // AppBar
      body: SingleChildScrollView(
        padding: EdgeInsets.all(20.0),
        child: StreamBuilder(
          stream: FirebaseFirestore.instance.collection('packages').snapshots(),
          builder: (context, snapshot) {
            if (!snapshot.hasData) {
              return Center(
                child: CircularProgressIndicator(),
              ); // Center
            }
            var packages = snapshot.data!.docs;
            // Filter packages if selectedPackageId is not null
            if (selectedPackageId != null) {
              packages = packages
                  .where((package) => package.id == selectedPackageId)
                  .toList();
            }
            return Column(
              crossAxisAlignment: CrossAxisAlignment.center,
```

3. Packages Class for Displaying Packages:

Description: Manages the display of packages fetched from Firebase and allows users to select a package for reservation.

```
Widget _buildPackageBox(BuildContext context, String title,
    List<String> features, String packageId, packagePrice, packageDuration
  return Container(
    width: double.infinity,
    decoration: BoxDecoration(
      borderRadius: BorderRadius.circular(10.0),
      border: Border.all(color: Colors.black, width: 0.5),
      color: Color.fromRGBO(255, 255, 255, 1),
      boxShadow: [
        BoxShadow(
          color: Colors.black.withOpacity(0.3),
          offset: Offset(4.0, 4.0),
          blurRadius: 20.0,
          spreadRadius: 2.0,
        ), // BoxShadow
      ],
    ), // BoxDecoration
    margin: EdgeInsets.symmetric(horizontal: 5.0, vertical: 10.0),
    child: Padding(
      padding: const EdgeInsets.all(16.0),
      child: Column(
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          Text(
            title,
            style: TextStyle(
              color: Color(0xFF040415),
              fontFamily: 'Roboto Serif',
              fontSize: 22.0,
              fontWeight: FontWeight.w600,
```

```
70    Widget _buildPackageBox(BuildContext context, String title,
                children: [
                  Text(
                    '\nDuration: ',
                    style: TextStyle(
                      color: Color(0xFF040415),
                      fontFamily: 'Roboto Serif',
                      fontSize: 18.0,
                      fontWeight: FontWeight.w400,
                      height: 1.11111,
                    ), // TextStyle
                  ), // Text
                  Text(
                    '\n$packageDuration Month',
                    style: TextStyle(
                      color: Color(0xFF040415),
                      fontFamily: 'Roboto Serif',
                      fontSize: 18.0,
                      fontWeight: FontWeight.w400,
                      height: 1.11111,
                    ), // TextStyle
                  ), // Text
                ],
              ), // Row
              SizedBox(height: 16.0),
              Center(child: _buildSubscriptionBox(context, packageId)),
            ],
          ), // Column
        ), // Padding
      ); // Container
    }
```

4. _buildPackageBox Widget:

Description: Creates a container to display package details like name, services, price, and duration for each package.

```dart
Widget _buildSubscriptionBox(BuildContext context, String packageId) {
  return ElevatedButton(
    onPressed: () {
      Navigator.push(
        context,
        MaterialPageRoute(
            builder: (context) => ReservationScreen(packageId: packageId)),
      );
    },
    style: ElevatedButton.styleFrom(
      foregroundColor: Colors.white,
      backgroundColor: Color(0xFFF86C1D), // foreground color
      padding: EdgeInsets.symmetric(vertical: 16.0, horizontal: 80.0),
      shape: RoundedRectangleBorder(
        borderRadius: BorderRadius.circular(30.0),
      ), // RoundedRectangleBorder
    ),
    child: Text(
      'Select Package',
      style: TextStyle(
        fontSize: 18.0,
        fontWeight: FontWeight.bold,
      ), // TextStyle
    ), // Text
  ); // ElevatedButton
}
}
```

5. _buildSubscriptionBox Widget:

Description: Renders an elevated button allowing users to select a package for reservation and navigate to the ReservationScreen

```dart
class ReservationScreen extends StatelessWidget {
  final String packageId;

  const ReservationScreen({required this.packageId});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Reservation Details'),
        backgroundColor: Color(0xFFF86C1D),
      ), // AppBar
      body: Padding(
        padding: EdgeInsets.all(20.0),
        child: VehicleSelection(packageId: packageId),
      ), // Padding
    ); // Scaffold
  }
}
```

6. ReservationScreen Class:

Description: Defines a screen for reservation details, including vehicle selection and booking of the selected package.

```dart
class VehicleSelection extends StatefulWidget {
  final String packageId;

  const VehicleSelection({required this.packageId});

  @override
  _VehicleSelectionState createState() => _VehicleSelectionState();
}

class _VehicleSelectionState extends State<VehicleSelection> {
  String? selectedVehicleId;
  DateTime? selectedDate;
  TimeOfDay? selectedTime;
  late Future<List<DocumentSnapshot>> vehiclesFuture;

  @override
  void initState() {
    super.initState();
    vehiclesFuture = _fetchVehicles();
  }

  Future<List<DocumentSnapshot>> _fetchVehicles() async {
    final user = FirebaseAuth.instance.currentUser;
    if (user != null) {
      final vehiclesSnapshot = await FirebaseFirestore.instance
          .collection('vehicles')
          .where('userId', isEqualTo: user.uid)
          .get();
      return vehiclesSnapshot.docs;
    }
    return [];
  }
```

7. VehicleSelection Class:

Description: Manages the selection of vehicles for reservation, including date and time selection.

```dart
Future<List<DocumentSnapshot>> _fetchVehicles() async {
  final user = FirebaseAuth.instance.currentUser;
  if (user != null) {
    final vehiclesSnapshot = await FirebaseFirestore.instance
        .collection('vehicles')
        .where('userId', isEqualTo: user.uid)
        .get();
    return vehiclesSnapshot.docs;
  }
  return [];
}
```

8. _fetchVehicles Method:

Description: Asynchronously fetches vehicles from Firestore based on the current user.

```dart
Future<void> _selectDate(BuildContext context) async {
  final DateTime? picked = await showDatePicker(
    context: context,
    initialDate: DateTime.now(),
    firstDate: DateTime.now(),
    lastDate: DateTime(2101),
  );
  if (picked != null && picked != selectedDate) {
    setState(() {
      selectedDate = picked;
    });
  }
}

Future<void> _selectTime(BuildContext context) async {
  final TimeOfDay? picked = await showTimePicker(
    context: context,
    initialTime: TimeOfDay.now(),
  );
  if (picked != null && picked != selectedTime)
    setState(() {
      selectedTime = picked;
    });
}
```

9. _selectDate and _selectTime Methods:

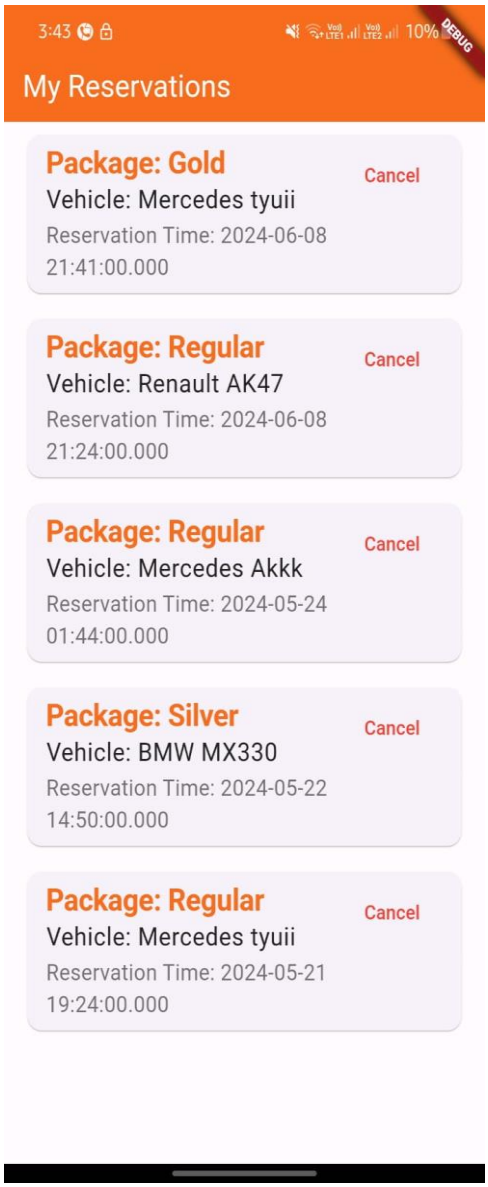
Description: Functions to handle the selection of date and time for the appointment.

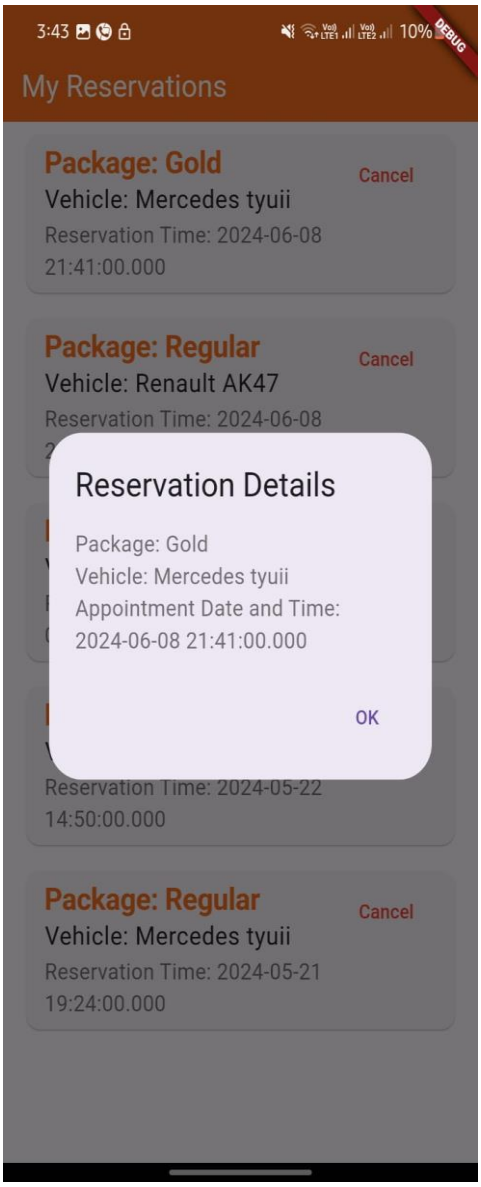
```dart
void _saveReservation() async {
  final user = FirebaseAuth.instance.currentUser;
  if (user != null &&
      selectedVehicleId != null &&
      selectedDate != null &&
      selectedTime != null) {
    final appointmentDateTime = DateTime(
      selectedDate!.year,
      selectedDate!.month,
      selectedDate!.day,
      selectedTime!.hour,
      selectedTime!.minute,
    );

    // Check the total reservations
    final reservationSnapshot =
        await FirebaseFirestore.instance.collection('reservations').get();
    if (reservationSnapshot.size >= 10) {
      ScaffoldMessenger.of(context).showSnackBar(
        SnackBar(content: Text('No parking slots available.')),
      );
      return;
    }

    // Save the reservation
    final reservationRef =
        await FirebaseFirestore.instance.collection('reservations').add({
      'userId': user.uid,
      'vehicleId': selectedVehicleId,
      'packageId': widget.packageId,
      'appointmentDateTime': appointmentDateTime,
```

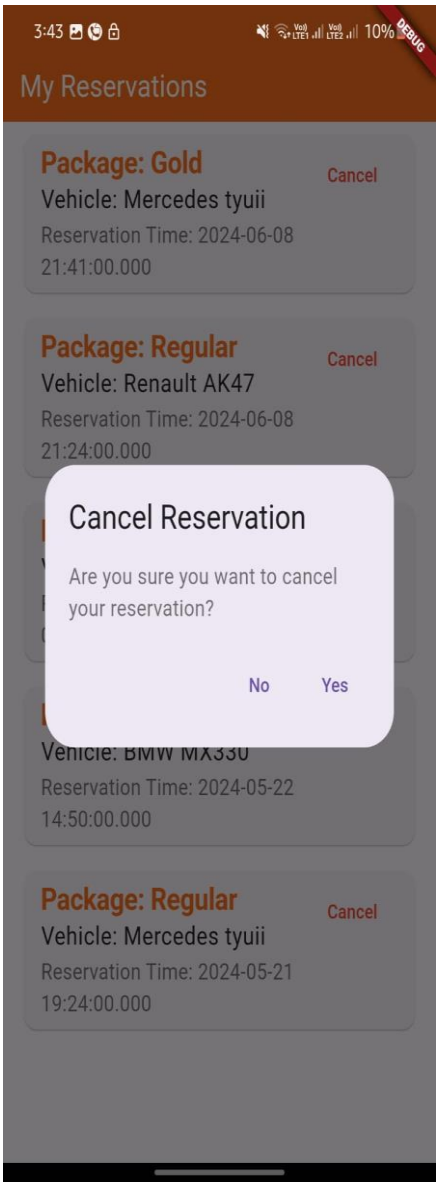
10. _saveReservation Method:

Description: Saves the reservation details in Firestore after selecting a vehicle, date, and time, and notifies the admin about the new reservation.

4.3. Reservations Page :







My Reservations Page:

This summary highlights the key functionalities behind managing reservations in the app:

Fetching Data:

Users' reservations are retrieved based on their ID and ordered chronologically.

Package names and vehicle details (brand & model) are fetched from Firestore for display.

Managing Reservations:

Users can cancel reservations with confirmation prompts and success messages.

New reservations are saved to Firestore with the selected vehicle and package.

User Interface:

The app utilizes FutureBuilders to fetch data asynchronously and update the UI with reservations, package/vehicle details, and available vehicles.

Various Flutter widgets create the user interface for displaying information and allowing interactions.

In essence, these functions work together to efficiently manage user reservations within the app.

```
Run | Debug | Profile
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();
  runApp(MyReservations());
}
```

1-Firebase Initialization in main():

Description: Initializes the Flutter app and Firebase services.

```
Future<List<QueryDocumentSnapshot>> _fetchReservations() async {
  final user = FirebaseAuth.instance.currentUser;
  if (user != null) {
    final querySnapshot = await FirebaseFirestore.instance
        .collection('reservations')
        .where('userId', isEqualTo: user.uid)
        .orderBy('timestamp', descending: true)
        .get();
    return querySnapshot.docs;
  }
  return [];
}
```

2-Fetching Reservations in _fetchReservations():

Description: Queries Firestore to fetch reservations for the current user

```dart
Future<String> _fetchPackageName(String packageId) async {
  final doc = await FirebaseFirestore.instance
      .collection('packages')
      .doc(packageId)
      .get();
  return doc['package_name'];
}
```

3-Fetching Package Name in _fetchPackageName(packageId

Description: Retrieves the package name from Firestore using the package ID.

```dart
Future<Map<String, String>> _fetchVehicleDetails(String vehicleId) async {
  final doc = await FirebaseFirestore.instance
      .collection('vehicles')
      .doc(vehicleId)
      .get();
  return {
    'brand': doc['brand'],
    'model': doc['model'],
  };
}
```

4-Fetching Vehicle Details in :

Description: Retrieves brand and model details of a vehicle from Firestore using the vehicle ID.

```dart
void _cancelReservation(String reservationId) async {
  await FirebaseFirestore.instance
      .collection('reservations')
      .doc(reservationId)
      .delete();
  ScaffoldMessenger.of(context).showSnackBar(
    SnackBar(content: Text('Reservation canceled successfully!')),
  );
  setState(() {});
}
```

5-Canceling a Reservation in _cancelReservation(reservationId):

Description: Deletes a reservation from Firestore based on the reservation ID.

```dart
Future<void> _showCancellationConfirmationDialog(String reservationId) async {
  return showDialog<void>(
    context: context,
    builder: (BuildContext context) {
      return AlertDialog(
        title: Text('Cancel Reservation'),
        content: Text('Are you sure you want to cancel your reservation?'),
        actions: <Widget>[
          TextButton(
            child: Text('No'),
            onPressed: () {
              Navigator.of(context).pop();
            },
          ), // TextButton
          TextButton(
            child: Text('Yes'),
            onPressed: () {
              Navigator.of(context).pop();
              _cancelReservation(reservationId);
            },
          ), // TextButton
        ], // <Widget>[]
      ); // AlertDialog
    },
  );
}
```

6-Showing Cancellation Dialog in _showCancellationConfirmationDialog(reservationId):

Description: Shows a dialog to confirm the cancellation of a reservation.

7-Firebase Operations and UI Setup in MyReservations & ReservationScreen:

Description: Sets up the Firebase instance and defines UI elements for displaying reservations and their details.

8-Animation Setup in _ReservationListState:

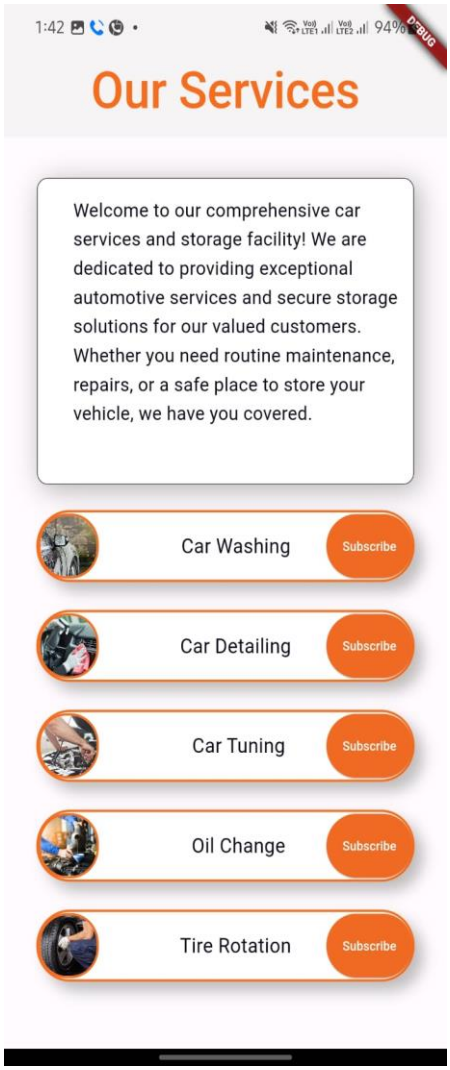
Description: Initializes an animation controller for smooth transition effects in the reservation list

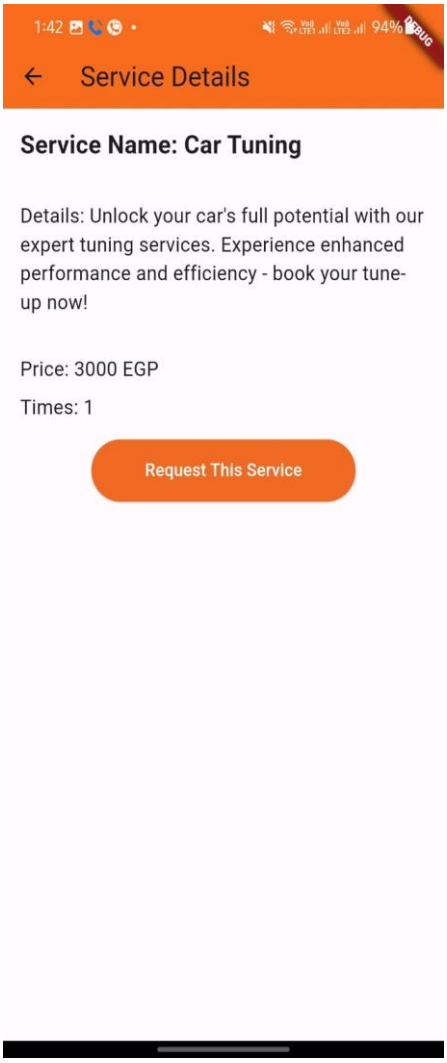
```
void _saveReservation() async {
  final user = FirebaseAuth.instance.currentUser;
  if (user != null && selectedVehicleId != null) {
    await FirebaseFirestore.instance.collection('reservations').add({
      'userId': user.uid,
      'vehicleId': selectedVehicleId,
      'packageId': widget.packageId,
      'timestamp': FieldValue.serverTimestamp(),
      'appointmentDateTime':
          Timestamp.now(), // Add a timestamp for the appointment
    });
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(content: Text('Reservation saved successfully!')),
    );
    Navigator.pop(context);
  } else {
    ScaffoldMessenger.of(context).showSnackBar(
      SnackBar(content: Text('Please select a vehicle.')),
    );
  }
}
```

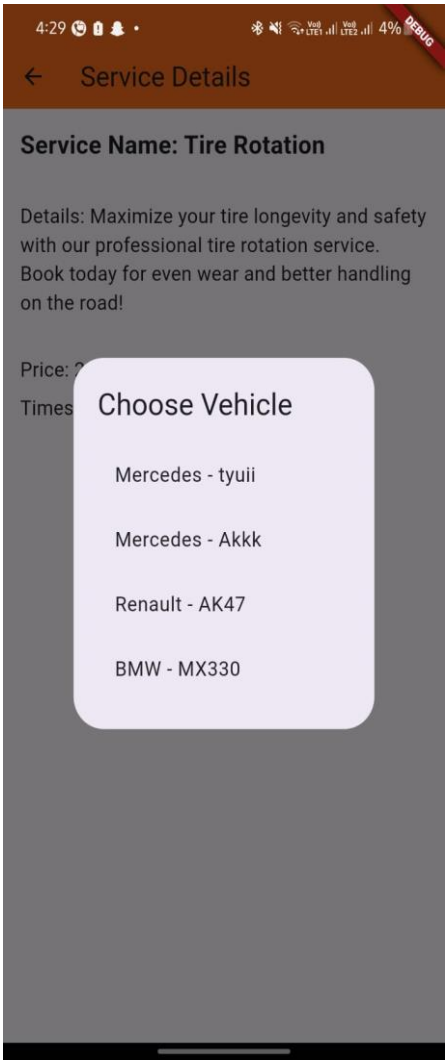
9-Saving a Reservation in _saveReservation() in VehicleSelection:

Description: Saves a new reservation linked to the selected vehicle and package ID in Firestore.

4.4. Car Services







This code creates a mobile app for a car service business. Here's what users can do:

- Browse and view details of various automotive services offered.
- Subscribe to service plans and choose vehicles for those services.
- Request specific services for their vehicles.
- Manage these actions through a user-friendly interface built with Flutter widgets.

The app interacts with Firebase for authentication and Firestore to manage service information, subscriptions, and requests.

```
Run | Debug | Profile
void main() {
  runApp(Services());
}
```

7. Main Function – main():

Description: Initializes the app and runs the top-level widget Services.

```
class Services extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: Scaffold(
        appBar: AppBar(
          title: Text(
            'Our Services',
            style: TextStyle(
              color: Color(0xFFF86C1D),
              fontFamily: 'Roboto Serif',
              fontSize: 40,
              fontStyle: FontStyle.normal,
              fontWeight: FontWeight.w500,
              height: 0.5,
            ), // TextStyle
          ), // Text
          centerTitle: true,
          backgroundColor: Color.fromRGBO(237, 237, 237, 0.5),
          toolbarHeight: 80,
        ), // AppBar
        body: ServicesList(),
      ), // Scaffold
    ); // MaterialApp
  }
}
```

7. Widget – Services:

Description: Sets up the app theme and structure, displaying a list of services.

```
class ServicesList extends StatelessWidget {
  Widget build(BuildContext context) {
                    fontStyle: FontStyle.normal,
                    fontWeight: FontWeight.w400,
                    height: 1.5,
                  ), // TextStyle
                ), // Text
              ), // Padding
            ), // Container
          ), // Padding
          SizedBox(height: 20),
          Padding(
            padding: const EdgeInsets.symmetric(horizontal: 30),
            child: Column(
              crossAxisAlignment: CrossAxisAlignment.stretch,
              children: [
                _buildServiceBox('1.jpg', 'Car Washing', context),
                SizedBox(height: 20),
                _buildServiceBox('2.jpg', 'Car Detailing', context),
                SizedBox(height: 20),
                _buildServiceBox('3.jpg', 'Car Tuning', context),
                SizedBox(height: 20),
                _buildServiceBox('4.jpeg', 'Oil Change', context),
                SizedBox(height: 20),
                _buildServiceBox('5.jpg', 'Tire Rotation', context),
              ],
            ), // Column
          ), // Padding
        ],
      ), // Column
    ); // SingleChildScrollView
  }
```

```
class ServicesList extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return SingleChildScrollView(
      child: Column(
        children: [
          SizedBox(height: 35),
          Padding(
            padding: const EdgeInsets.symmetric(horizontal: 30),
            child: Container(
              width: 350,
              height: 264,
              decoration: BoxDecoration(
                borderRadius: BorderRadius.circular(10),
                border: Border.all(color: Colors.black, width: 0.5),
                color: Colors.white,
                boxShadow: [
                  BoxShadow(
                    color: Colors.black.withOpacity(0.3),
                    blurRadius: 20,
                    offset: Offset(4, 4),
                  ), // BoxShadow
                ],
              ), // BoxDecoration
              child: Padding(
                padding: EdgeInsets.only(left: 33, right: 14, top: 15),
                child: Text(
                  'Welcome to our comprehensive car services and storage facili
                  style: TextStyle(
                    color: Color(0xFF040415),
                    fontFamily: 'Roboto Serif',
                    fontSize: 15
```

3. Widget – ServicesList:

Description: Displays a list of services with introductory text and service boxes.

```
Widget _buildServiceBox(
    String imagePath, String title, BuildContext context) {
  return FutureBuilder<DocumentSnapshot>(
    future: FirebaseFirestore.instance
        .collection('services')
        .where('service_name', isEqualTo: title)
        .limit(1)
        .get()
        .then((value) => value.docs.first),
    builder: (context, snapshot) {
      if (snapshot.connectionState == ConnectionState.waiting) {
        return CircularProgressIndicator();
      }
      if (!snapshot.hasData || !snapshot.data!.exists) {
        return SizedBox(); // Return an empty widget if data doesn't exist
      }

      var serviceId = snapshot.data!.id;
      var serviceName = snapshot.data!['service_name'];

      return Container(
        height: 62,
        width: 309,
        margin: EdgeInsets.symmetric(vertical: 2),
        padding: EdgeInsets.only(
            left: 0), // Add padding to move content to the left // EdgeInsets
        decoration: BoxDecoration(
          borderRadius: BorderRadius.circular(60),
          border: Border.all(color: Color(0xFFF86C1D), width: 2),
          color: Colors.white,
```

4. Widget - _buildServiceBox():

Description: Builds individual service boxes within the services list.

```
Widget _buildServiceBox(
    String imagePath, String title, BuildContext context) {
  return FutureBuilder<DocumentSnapshot>(
    future: FirebaseFirestore.instance
        .collection('services')
        .where('service_name', isEqualTo: title)
        .limit(1)
        .get()
        .then((value) => value.docs.first),
    builder: (context, snapshot) {
      if (snapshot.connectionState == ConnectionState.waiting) {
        return CircularProgressIndicator();
      }
      if (!snapshot.hasData || !snapshot.data!.exists) {
        return SizedBox(); // Return an empty widget if data doesn't exist
      }
```

5. Service Data Retrieval – FutureBuilder:

Description: Fetches service details from Firestore based on the selected service name.

```
class ServiceDetailPage extends StatelessWidget {
  final String serviceId;

  ServiceDetailPage({required this.serviceId});

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text('Service Details'),
        backgroundColor: Color(0xFFF86C1D),
      ), // AppBar
      body: FutureBuilder<DocumentSnapshot>(
        future: FirebaseFirestore.instance
            .collection('services')
            .doc(serviceId)
            .get(),
        builder: (context, snapshot) {
          if (snapshot.connectionState == ConnectionState.waiting) {
            return Center(child: CircularProgressIndicator());
          }
          if (!snapshot.hasData || !snapshot.data!.exists) {
            return Center(child: Text('Service not found'));
          }

          var serviceData = snapshot.data!.data() as Map<String, dynamic>;
          var serviceName = serviceData['service_name'];
          var serviceDetails = serviceData['service_details'];
          var servicePrice = serviceData['service_price'];
          var times = serviceData['times'];
```

6. Widget – ServiceDetailPage:

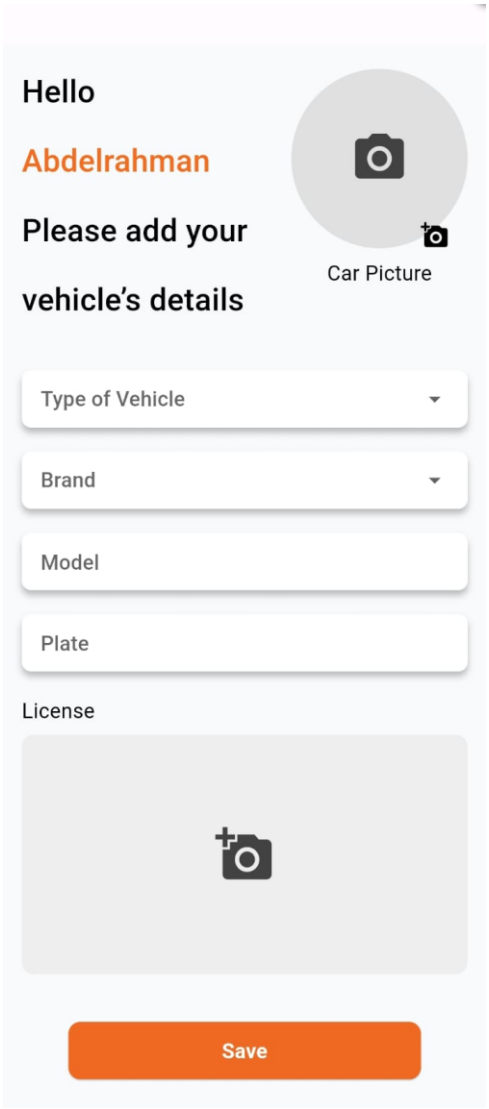
Description: Displays detailed information about a selected service and allows users to request the service.

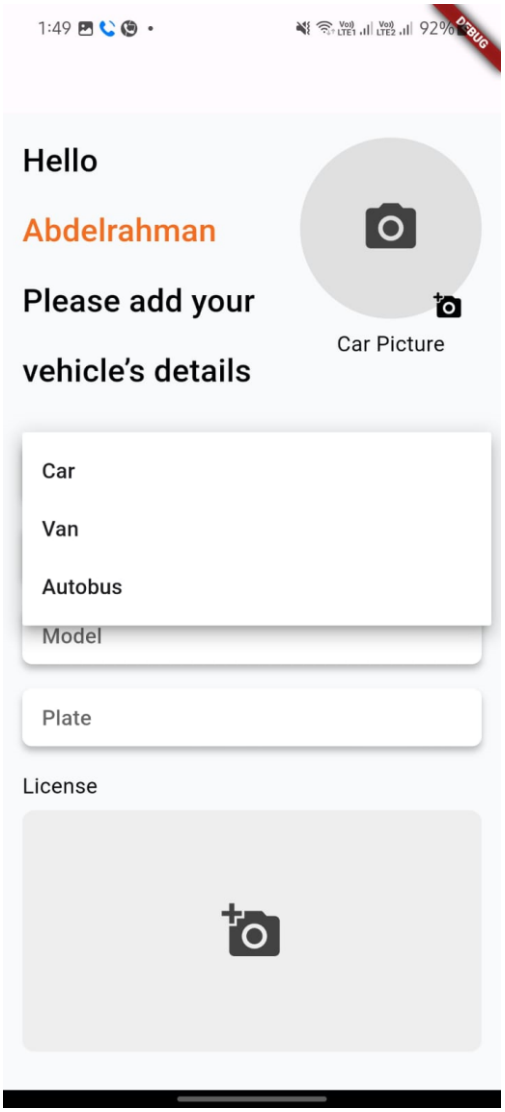
```dart
class ChooseVehicleDialog extends StatelessWidget {
  final List<QueryDocumentSnapshot> vehicles;

  ChooseVehicleDialog({required this.vehicles});

  @override
  Widget build(BuildContext context) {
    return AlertDialog(
      title: Text('Choose Vehicle'),
      content: SingleChildScrollView(
        child: Column(
          children: vehicles.map((vehicle) {
            var vehicleData = vehicle.data() as Map<String, dynamic>;
            var vehicleName = (vehicleData['brand'] ?? 'Unknown Brand') +
                ' - ' +
                (vehicleData['model'] ?? 'Unknown Model');
            return ListTile(
              title: Text(vehicleName),
              onTap: () => Navigator.pop(context, vehicle.id),
            ); // ListTile
          }).toList(),
        ), // Column
      ), // SingleChildScrollView
    ); // AlertDialog
  }
}
```

7. Dialog – ChooseVehicleDialog:

Description: Displays a dialog for users to choose a vehicle for requesting a service.

4.5. Add Vehicle Page:





This code creates a mobile app for users to manage their vehicles. Here's what users can do:

- Add details about their vehicles, including capturing images of the car and license plate.
- Enter information like vehicle type, brand, model, and license plate number.

The app uses Firebase for several functionalities:

- **Authentication:** Ensures only registered users can add vehicles.
- **Firestore:** Stores all the vehicle details entered by the user.

- **Firebase Storage:** Stores the uploaded images of the car and license plate securely.

This allows users to easily manage their vehicle information within the app.

```
Run | Debug | Profile
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();
  runApp(AddCar());
}
```

1-Firebase Initialization in main():

Description: Initializes the Flutter app and Firebase services.

```
Future<void> _fetchUserName() async {
  try {
    User? user = FirebaseAuth.instance.currentUser;
    if (user != null) {
      DocumentSnapshot userDoc = await FirebaseFirestore.instance
          .collection('users')
          .doc(user.uid)
          .get();
      setState(() {
        _firstName = userDoc['firstName'];
      });
    }
  } catch (e) {
    print("Error fetching user name: $e");
  }
}
```

2-Fetching User Name in _fetchUserName():

Description: Retrieves the first name of the current user for personalization.

```dart
Future<void> _getCarImage() async {
  final pickedFile =
      await ImagePicker().pickImage(source: ImageSource.gallery);
  if (pickedFile != null) {
    setState(() {
      _carImage = File(pickedFile.path);
    });
  }
}
```

3-Image Selection for Car in _getCarImage():

Description: Implements the functionality for selecting a car image from the device's gallery

```dart
Future<void> _getLicenseImage() async {
  final pickedFile =
      await ImagePicker().pickImage(source: ImageSource.gallery);
  if (pickedFile != null) {
    setState(() {
      _licenseImage = File(pickedFile.path);
    });
  }
}
```

4-Image Selection for License in _getLicenseImage():

Description: Implements the functionality for selecting a license image from the device's gallery.

```dart
Future<void> _saveCarDetails() async {
  if (_formKey.currentState?.validate() ?? false) {
    setState(() {
      _isUploading = true;
    });

    try {
      String? carImageUrl;
      String? licenseImageUrl;

      if (_carImage != null) {
        carImageUrl = await _uploadFile(_carImage!, 'car_images');
      }

      if (_licenseImage != null) {
        licenseImageUrl = await _uploadFile(_licenseImage!, 'license_images');
      }

      User? user = FirebaseAuth.instance.currentUser;
      if (user != null) {
        await FirebaseFirestore.instance.collection('vehicles').add({
          'userId': user.uid,
          'vehicleType': _selectedVehicleType,
          'brand': _selectedBrand,
          'model': _modelController.text,
          'plate': _plateController.text,
          'carImageUrl': carImageUrl,
          'licenseImageUrl': licenseImageUrl,
        });
      }
```

```dart
      ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(content: Text('Vehicle details saved successfully')),
      );

      // Navigate to the home screen after showing the success message
      Navigator.push(
        context,
        MaterialPageRoute(builder: (context) => HomeScreen()),
      );
    } catch (e) {
      print('Error saving vehicle details: $e');
      ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(content: Text('Failed to save vehicle details')),
      );
    } finally {
      setState(() {
        _isUploading = false;
      });
    }
  }
}
```

5-Saving Vehicle Details in _saveCarDetails():

Description: Validates the input data and saves the vehicle details, including uploaded images, in Firestore.

```dart
Future<String> _uploadFile(File file, String folder) async {
  try {
    String fileName =
        '${DateTime.now().millisecondsSinceEpoch}_${file.path.split('/').last}';
    Reference storageRef =
        FirebaseStorage.instance.ref().child('$folder/$fileName');
    UploadTask uploadTask = storageRef.putFile(file);
    TaskSnapshot snapshot = await uploadTask;
    return await snapshot.ref.getDownloadURL();
  } catch (e) {
    print('Error uploading file: $e');
    return '';
  }
}
```

6-Image Upload to Firebase Storage in _uploadFile():

Description: Handles the uploading of files (car and license images) to Firebase Storage and retrieves their download URLs.

```dart
Widget _buildDropdown(BuildContext context, String hint, String? value,
    List<String> items, ValueChanged<String?> onChanged) {
  return Container(
    height: 48,
    padding: const EdgeInsets.symmetric(horizontal: 16.0),
    decoration: BoxDecoration(
      color: Colors.white,
      borderRadius: BorderRadius.circular(8.0),
      boxShadow: const [
        BoxShadow(
          color: Colors.black26,
          blurRadius: 4,
          offset: Offset(0, 4),
        ), // BoxShadow
      ],
    ), // BoxDecoration
    child: DropdownButton<String>(
      dropdownColor: Colors.white,
      isExpanded: true,
      value: value,
      items: items.map((item) {
        return DropdownMenuItem<String>(
          value: item,
          child: Text(item),
        ); // DropdownMenuItem
      }).toList(),
      onChanged: onChanged,
      hint: Text(hint),
      underline: const SizedBox(),
    ), // DropdownButton
  ); // Container
```

7-Dropdown Builder in _buildDropdown():

Description: Creates a dropdown widget for selecting vehicle type and brand.

```dart
Widget _buildTextField(String hint, TextEditingController controller) {
  return Container(
    height: 48,
    padding: const EdgeInsets.symmetric(horizontal: 16.0),
    decoration: BoxDecoration(
      color: Colors.white,
      borderRadius: BorderRadius.circular(8.0),
      boxShadow: const [
        BoxShadow(
          color: Colors.black26,
          blurRadius: 4,
          offset: Offset(0, 4),
        ), // BoxShadow
      ],
    ), // BoxDecoration
    child: TextFormField(
      controller: controller,
      validator: (value) {
        if (value == null || value.isEmpty) {
          return 'Please enter some text';
        }
        return null;
      },
      decoration: InputDecoration(
        border: InputBorder.none,
        hintText: hint,
      ), // InputDecoration
    ), // TextFormField
  ); // Container
}
```

8-Text Field Builder in _buildTextField():

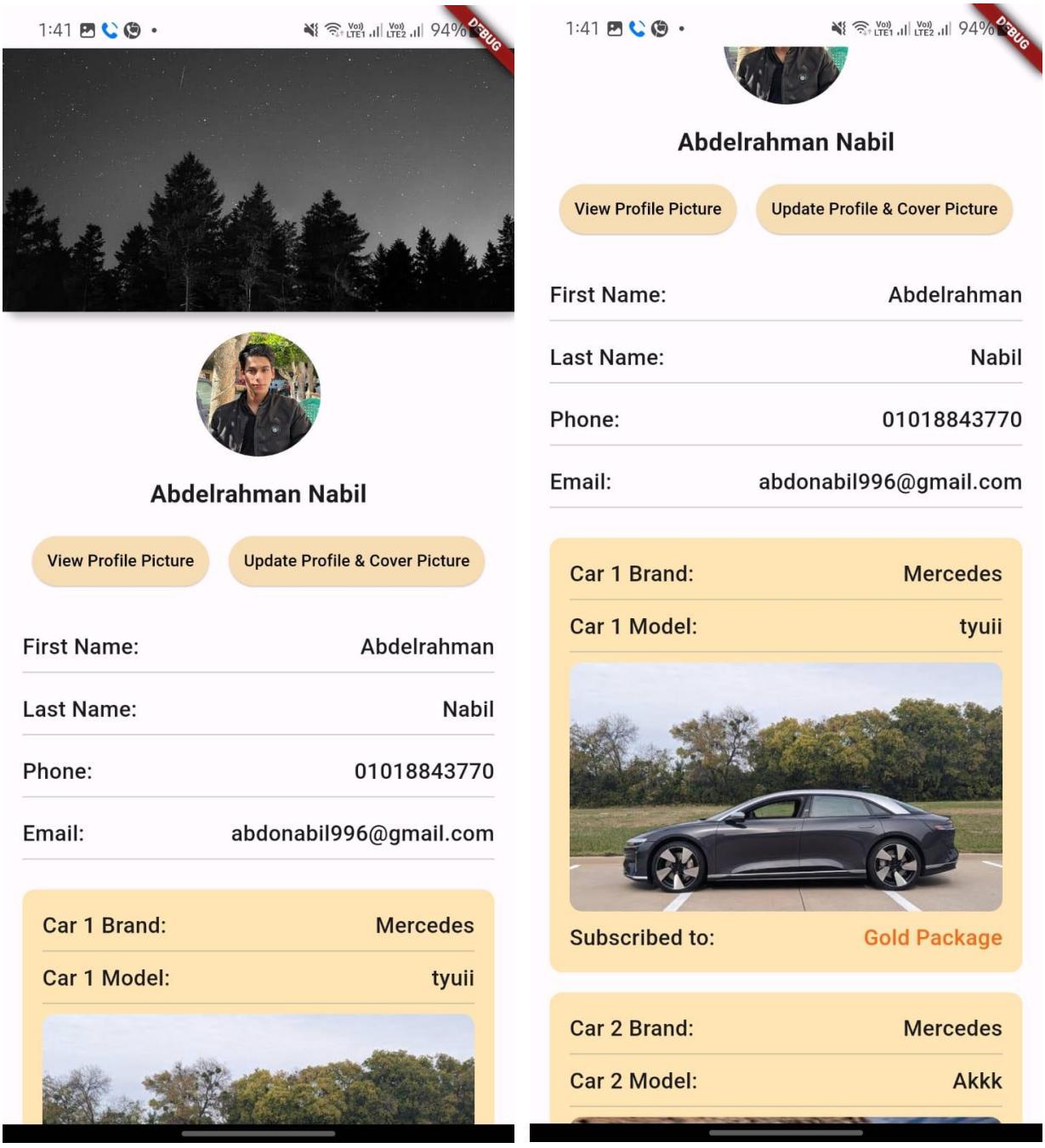
Description: Builds a text field widget for entering model and plate details.

```dart
Widget _buildImagePicker(
    String label, File? imageFile, VoidCallback onPickImage) {
  return Column(
    crossAxisAlignment: CrossAxisAlignment.start,
    children: [
      Text(
        label,
        style: const TextStyle(
          color: Colors.black,
          fontSize: 16,
          fontFamily: 'Roboto',
        ), // TextStyle
      ), // Text
      const SizedBox(height: 8),
      GestureDetector(
        onTap: onPickImage,
        child: Container(
          height: 200,
          width: double.infinity,
          decoration: BoxDecoration(
            color: Colors.grey[200],
            borderRadius: BorderRadius.circular(10),
            image: imageFile != null
                ? DecorationImage(
                    image: FileImage(imageFile),
                    fit: BoxFit.cover,
                  ) // DecorationImage
                : null,
          ), // BoxDecoration
          child: imageFile == null
```

9-Image Picker Builder in _buildImagePicker():

Description: Constructs an image picker widget for selecting and displaying images of the car and license.

4.6. Profile Page:



profile page that allows users to manage and view their profile and cover images, personal information, and associated vehicle details. The app integrates with Firebase for authentication, storage, and Firestore for data management.

```dart
import 'package:cached_network_image/cached_network_image.dart';
import 'package:firebase_storage/firebase_storage.dart';
import 'package:flutter/material.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:image_picker/image_picker.dart';
import 'dart:io';
```

Imports

The necessary packages are imported, including:

cached_network_image for efficient image caching.

Firebase packages for authentication, storage, and Firestore.

image_picker for image selection from the gallery.

dart:io for handling file operations.

```dart
Run | Debug | Profile
void main() => runApp(const Profile());
```

Main Function

The main function runs the Profile widget, which serves as the entry point of the application.

```dart
class Profile extends StatelessWidget {
  const Profile({Key? key});

  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Profile Page',
      theme: ThemeData(
        primarySwatch: Colors.blue,
      ), // ThemeData
      home: const ProfilePage(),
    ); // MaterialApp
  }
}
```

Profile StatelessWidget

The Profile class extends StatelessWidget and builds the MaterialApp with the ProfilePage as the home screen.

```dart
class ProfilePage extends StatefulWidget {
  const ProfilePage({Key? key});

  @override
  _ProfilePageState createState() => _ProfilePageState();
}
```

ProfilePage StatefulWidget

The ProfilePage class extends StatefulWidget to manage the profile page state.

```dart
@override
void initState() {
  super.initState();
  _loadingFuture = _loadProfileImage();
}
```

_loadProfileImage Method

This method retrieves the current user's profile and cover images from Firestore.

```dart
Future<void> _loadProfileImage() async {
  final user = FirebaseAuth.instance.currentUser;
  if (user != null) {
    try {
      final userData = await FirebaseFirestore.instance
          .collection('users')
          .doc(user.uid)
          .get();
      if (userData.exists) {
        setState(() {
          _profileImgUrl = userData['profileImgUrl'];
          _coverImgUrl = userData['coverImgUrl'];
        });
      }
    } catch (error) {
      print('Error loading profile data: $error');
    }
  }
}
```

_pickImage Method

This method allows users to pick an image from the gallery and updates the corresponding state variable.

```dart
Future<void> _pickImage(bool isCoverImage) async {
  final picker = ImagePicker();
  final pickedFile = await picker.pickImage(source: ImageSource.gallery);

  if (pickedFile != null) {
    setState(() {
      if (isCoverImage) {
        _coverImage = File(pickedFile.path);
      } else {
        _profileImage = File(pickedFile.path);
      }
    });
  }
}
```

_uploadProfileAndCoverImages Method

This method uploads the selected profile and cover images to Firebase Storage and updates their URLs in Firestore.

```dart
    if (_profileImage != null) {
      final profileFileName = 'profile_${user.uid}.jpg';
      final profileDestination = 'users/$profileFileName';
      final profileRef = FirebaseStorage.instance.ref(profileDestination);
      await profileRef.putFile(_profileImage!);
      final profileImageUrl = await profileRef.getDownloadURL();
      await FirebaseFirestore.instance
          .collection('users')
          .doc(user.uid)
          .update({'profileImgUrl': profileImageUrl});
      setState(() {
        _profileImgUrl = profileImageUrl;
        _profileImage = null;
      });
    }

    ScaffoldMessenger.of(context).showSnackBar(SnackBar(
      content: Text('Profile and cover pictures updated successfully'),
      backgroundColor: Colors.green,
    )); // SnackBar
  } catch (error) {
    ScaffoldMessenger.of(context).showSnackBar(SnackBar(
      content: Text('Failed to update profile and cover pictures: $error'),
      backgroundColor: Colors.red,
    )); // SnackBar
  } finally {
    setState(() {
      _isUploading = false;
    });
  }
 }
}
```

```dart
Future<void> _uploadProfileAndCoverImages() async {
  final user = FirebaseAuth.instance.currentUser;
  if (user != null && (_profileImage != null || _coverImage != null)) {
    setState(() {
      _isUploading = true;
    });

    try {
      if (_coverImage != null) {
        final coverFileName = 'cover_${user.uid}.jpg';
        final coverDestination = 'users/$coverFileName';
        final coverRef = FirebaseStorage.instance.ref(coverDestination);
        await coverRef.putFile(_coverImage!);
        final coverImageUrl = await coverRef.getDownloadURL();
        await FirebaseFirestore.instance
            .collection('users')
            .doc(user.uid)
            .update({'coverImgUrl': coverImageUrl});
        setState(() {
          _coverImgUrl = coverImageUrl;
          _coverImage = null;
        });
      }
```

_showProfilePicture Method

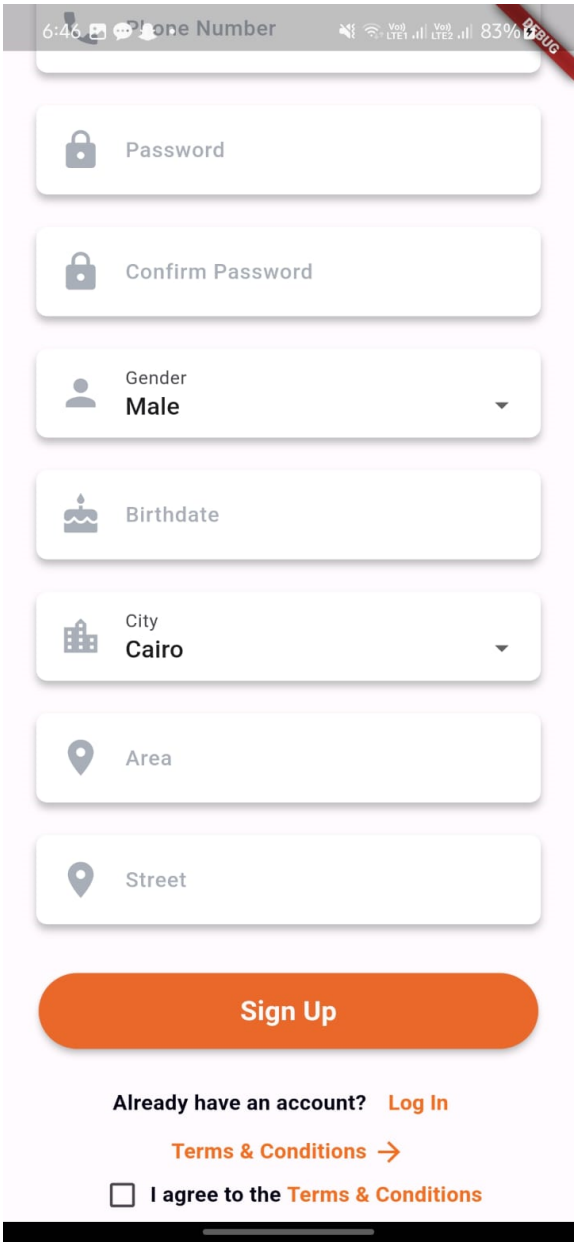
This method displays the current profile picture in a dialog.

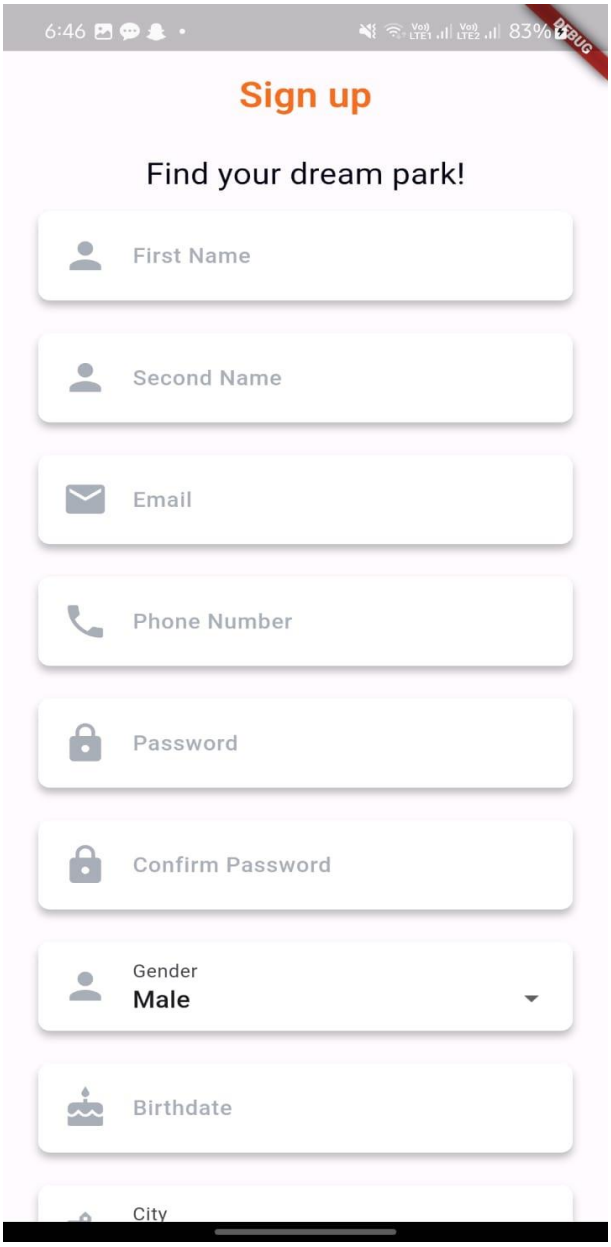
```
void _showProfilePicture() {
  if (_profileImgUrl != null) {
    showDialog(
      context: context,
      builder: (context) => Dialog(
        child: Container(
          padding: const EdgeInsets.all(16.0),
          child: Column(
            mainAxisSize: MainAxisSize.min,
            children: [
              CachedNetworkImage(
                imageUrl: _profileImgUrl!,
                placeholder: (context, url) => CircularProgressIndicator(),
                errorWidget: (context, url, error) => Icon(Icons.error),
              ), // CachedNetworkImage
              const SizedBox(height: 16),
              ElevatedButton(
                onPressed: () => Navigator.of(context).pop(),
                child: Text('Close'),
              ), // ElevatedButton
            ],
          ), // Column
        ), // Container
      ), // Dialog
    );
  } else {
    ScaffoldMessenger.of(context).showSnackBar(SnackBar(
      content: Text('No profile picture to display'),
      backgroundColor: Colors.red,
    )); // SnackBar
  }
}
```

build Method

The build method constructs the UI, including image upload and display functionalities, and user information sections.

4.7. Sign Up Page:





This page allows users to sign up for an account using Firebase Authentication and Firestore for backend services.

```dart
void main() {
  runApp(MyApp());
}

class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      home: SignUpScreen(),
    ); // MaterialApp
  }
}

class SignUpScreen extends StatefulWidget {
  @override
  _SignUpScreenState createState() => _SignUpScreenState();
}
```

```dart
class _InputFieldState extends State<InputField> {
  Widget build(BuildContext context) {
                child: TextField(
                  focusNode: _focusNode,
                  controller: widget.controller,
                  obscureText: widget.isPassword,
                  keyboardType: widget.keyboardType,
                  enabled: widget.enabled,
                  onChanged: (value) {
                    setState(() {
                      _errorMessage =
                          value.isEmpty ? 'This field is required.' : '';
                    });
                  },
                  decoration: InputDecoration(
                    labelText: widget.labelText,
                    labelStyle: TextStyle(
                      fontSize: _focusNode.hasFocus ? 16 : 14,
                      fontWeight: FontWeight.w500,
                      color: _focusNode.hasFocus
                          ? Color(0xFFF86C1D)
                          : Color(0xFFA8AFB9),
                    ), // TextStyle
                    border: InputBorder.none,
                  ), // InputDecoration
                ), // TextField
              ), // Expanded
            ],
          ), // Row
```

```dart
  bool _isInputValid() {
    if (_emailController.text.isEmpty ||
        _passwordController.text.isEmpty ||
        _confirmPasswordController.text.isEmpty ||
        _birthdateController.text.isEmpty ||
        _firstNameController.text.isEmpty ||
        _secondNameController.text.isEmpty ||
        _phoneNumberController.text.isEmpty) {
      setState(() {
        _errorMessage = 'All fields are required.';
      });
      return false;
    } else if (_passwordController.text != _confirmPasswordController.text) {
      setState(() {
        _errorMessage = 'Passwords do not match.';
      });
      return false;
    } else if (!_isChecked) {
      setState(() {
        _errorMessage = 'Please accept the terms and conditions.';
      });
      return false;
    }
    return true;
  }
}
```

```dart
            ElevatedButton(
              onPressed: () async {
                if (_isInputValid()) {
                  try {
                    final credential = await FirebaseAuth.instance
                        .createUserWithEmailAndPassword(
                      email: _emailController.text,
                      password: _passwordController.text,
                    );

                    // Log in the user immediately after sign-up
                    await FirebaseAuth.instance.signInWithEmailAndPassword(
                      email: _emailController.text,
                      password: _passwordController.text,
                    );

                    await FirebaseFirestore.instance
                        .collection('users')
                        .doc(credential.user!.uid)
                        .set({
                      'email': _emailController.text,
                      'firstName': _firstNameController.text,
                      'secondName': _secondNameController.text,
                      'phoneNumber': _phoneNumberController.text,
                      'gender': _selectedGender,
                      'birthdate': _birthdateController.text,
                      'city': _selectedCity,
```

```dart
@override
void dispose() {
  _emailController.dispose();
  _passwordController.dispose();
  _confirmPasswordController.dispose();
  _birthdateController.dispose();
  _firstNameController.dispose();
  _secondNameController.dispose();
  _phoneNumberController.dispose();
  _areaController.dispose(); // Dispose of area controller
  _streetController.dispose(); // Dispose of street controller
  super.dispose();
}
```

```dart
class _SignUpScreenState extends State<SignUpScreen> {
  bool _isChecked = false;
  late TextEditingController _emailController;
  late TextEditingController _passwordController;
  late TextEditingController _confirmPasswordController;
  late TextEditingController _birthdateController;
  late TextEditingController _firstNameController;
  late TextEditingController _secondNameController;
  late TextEditingController _phoneNumberController;
  late TextEditingController _areaController; // New controller for area
  late TextEditingController _streetController; // New controller for street
  late String _selectedGender = 'Male';
  late String _selectedCity = 'Cairo';
  String? _errorMessage;

  @override
  void initState() {
    super.initState();
    _emailController = TextEditingController();
    _passwordController = TextEditingController();
    _confirmPasswordController = TextEditingController();
    _birthdateController = TextEditingController();
    _firstNameController = TextEditingController();
    _secondNameController = TextEditingController();
    _phoneNumberController = TextEditingController();
    _areaController = TextEditingController(); // Initialize area controller
    _streetController = TextEditingController(); // Initialize street controller
  }
```

Imports:

firebase_auth.dart: Provides Firebase Authentication services.

material.dart: Flutter's material design library.

cloud_firestore.dart: Provides Firestore database services.

login.dart, home.dart, terms_and_conditions.dart: Custom modules for the login page, home page, and terms and conditions page.

Main Application:

main(): The entry point of the application that runs the MyApp widget.

MyApp: A stateless widget that sets up the MaterialApp with SignUpScreen as the home screen.

SignUpScreen:

SignUpScreen: A stateful widget that manages the sign-up form.

_SignUpScreenState: Holds the state and logic for the SignUpScreen.

State Variables:

TextEditingControllers: Controllers for handling text input for email, password, confirm password, birthdate, first name, second name, phone number, area, and street.

_selectedGender: Dropdown value for gender selection, initialized to 'Male'.

_selectedCity: Dropdown value for city selection, initialized to 'Cairo'.

_isChecked: Boolean for terms and conditions acceptance.

_errorMessage: String to display error messages.

Lifecycle Methods:

initState(): Initializes the text controllers.

dispose(): Disposes of the text controllers to free up resources.

UI Build Method:

build(): Constructs the UI using a Scaffold and SingleChildScrollView.

Contains a form with input fields for first name, second name, email, phone number, password, confirm password, gender, birthdate, city, area, and street.

An ElevatedButton to submit the form.

If there's an error message, it displays it below the submit button.

A row for navigating to the login page if the user already has an account.

A checkbox and text for agreeing to the terms and conditions.

Form Validation:

_isInputValid(): Validates the input fields to ensure they are not empty, passwords match, and terms are accepted.

Form Submission:

onPressed: The submit button triggers:

Validation of the form inputs.

Attempt to create a new user with Firebase Authentication.

On successful sign-up, log in the user and save additional user data to Firestore.

Navigate to the HomeScreen.

InputField Widget:

A custom widget to create a text input field with an icon.

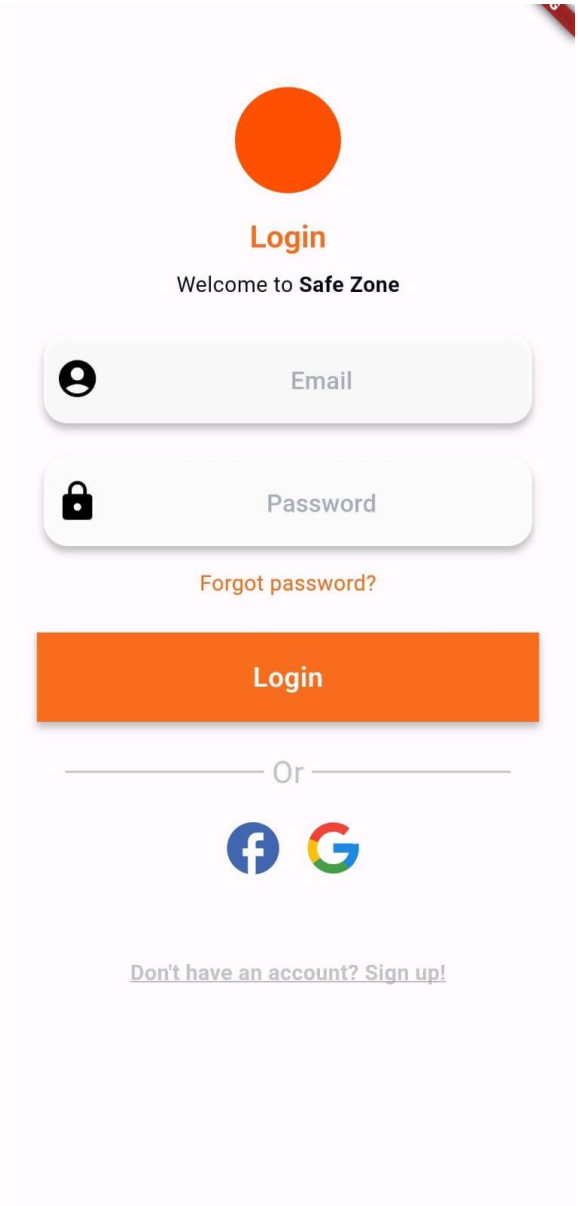
Manages focus and displays an error message if the field is empty.

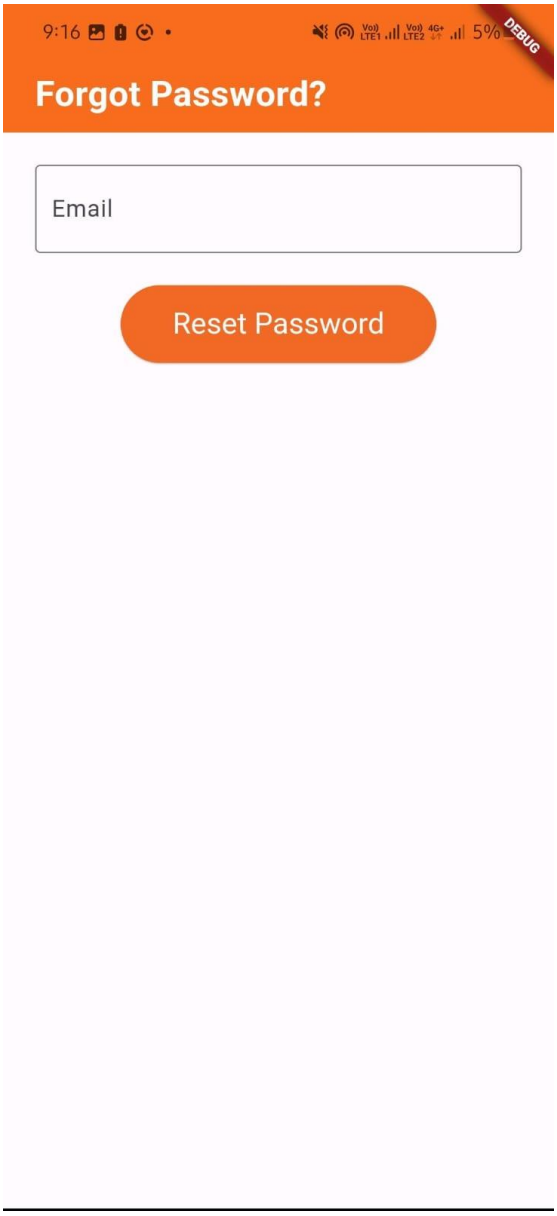
DropdownField Widget:

A custom widget for dropdown selection with an icon.

Takes parameters for the label text, icon, current value, onChanged callback, and dropdown items.

4.8. Login Page :





Login page:

```dart
import 'package:flutter/material.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'package:cloud_firestore/cloud_firestore.dart';
import 'package:safe_zone/admin_view.dart';
import 'package:safe_zone/google.dart';
import 'package:safe_zone/home.dart';
import 'package:safe_zone/signup.dart';
```

Importing necessary packages and dependencies for Flutter, Firebase authentication, Firestore, and custom views.

LoginPage StatefulWidget.

```dart
class LoginPage extends StatefulWidget {
  @override
  _LoginPageState createState() => _LoginPageState();
}
```

Defining the LoginPage as a stateful widget.

_LoginPageState Class: Controller Initialization.

```dart
class _LoginPageState extends State<LoginPage> {
  TextEditingController _emailController = TextEditingController();
  TextEditingController _passwordController = TextEditingController();

  @override
  void dispose() {
    _emailController.dispose();
    _passwordController.dispose();
    super.dispose();
  }
```

Initializing text controllers for email and password input fields and disposing of them properly.

Build Method: Scaffold and Main UI Structure.

```
Center(
  child: Column(
    children: [
      SizedBox(height: 8),
      Container(
        width: 350,
        height: 64,
        decoration: BoxDecoration(
          borderRadius: BorderRadius.circular(17),
          color: Color.fromRGBO(255, 255, 255, 0.894),
          boxShadow: [
            BoxShadow(
              color: Colors.black.withOpacity(0.25),
              offset: Offset(0, 4),
              blurRadius: 4,
            ), // BoxShadow
          ],
        ), // BoxDecoration
        child: Row(
          children: [
            SizedBox(width: 8),
            Icon(Icons.account_circle,
                size: 32, color: Colors.black), // Icon
            SizedBox(width: 8),
            Expanded(
              child: TextField(
                controller: _emailController,
                textAlign: TextAlign.center,
```

```
@override
Widget build(BuildContext context) {
  return Scaffold(
    body: SingleChildScrollView(
      child: Center(
        child: Column(
          crossAxisAlignment: CrossAxisAlignment.center,
          children: [
            SizedBox(height: 92),
            Container(
              width: 76,
              height: 76,
              decoration: BoxDecoration(
                shape: BoxShape.circle,
                color: Color(0xFFFF5000),
              ), // BoxDecoration
            ), // Container
            SizedBox(height: 16),
            Text(
              'Login',
              style: TextStyle(
                fontFamily: 'Roboto',
                fontSize: 20,
                letterSpacing: 0.0,
                color: Color(0xFFF86C1D),
                fontWeight: FontWeight.w600,
              ), // TextStyle
            ), // Text
```

```
), // Text
SizedBox(height: 8),
Row(
  mainAxisAlignment: MainAxisAlignment.center,
  children: [
    Text(
      'Welcome to ',
      style: TextStyle(
        fontFamily: 'Roboto',
        fontSize: 14,
        color: Color(0xFF040415),
        fontWeight: FontWeight.normal,
      ), // TextStyle
    ), // Text
    Text(
      'Safe Zone',
      style: TextStyle(
        fontFamily: 'Roboto',
        fontSize: 14,
        color: Color(0xFF040415),
        fontWeight: FontWeight.bold,
      ), // TextStyle
    ), // Text
  ],
), // Row
SizedBox(height: 16),
Center(
```

Building the main structure of the login page, including logo, title, and email/password input fields.

Forgot Password and Login Button

```
                    MaterialPageRoute(
                        builder: (context) => HomeScreen())); //
                }
              } else {
                ScaffoldMessenger.of(context).showSnackBar(
                  SnackBar(
                    content: Text('User document not found!'),
                    backgroundColor: Colors.red,
                  ), // SnackBar
                );
              }
            }
          } on FirebaseAuthException catch (e) {
            ScaffoldMessenger.of(context).showSnackBar(
              SnackBar(
                content: Text(e.message ?? 'An error occurred'),
                backgroundColor: Colors.red,
              ), // SnackBar
            );
          }
        },
        child: Center(
          child: Text(
            'Login',
            style: TextStyle(
              fontFamily: 'Roboto',
              fontSize: 18,
              color: Color(0xFFFBFBFB),
              fontWeight: FontWeight.w600,
            ), // TextStyle
```

```
            blurRadius: 4,
          ), // BoxShadow
        ],
      ), // BoxDecoration
      child: InkWell(
        onTap: () async {
          String email = _emailController.text;
          String password = _passwordController.text;
          try {
            final credential = await FirebaseAuth.instance
                .signInWithEmailAndPassword(
                    email: email, password: password);
            if (credential != null) {
              final user = FirebaseAuth.instance.currentUser;
              final userDoc = await FirebaseFirestore.instance
                  .collection('users')
                  .doc(user?.uid)
                  .get();

              if (userDoc.exists) {
                final bool isAdmin = userDoc.get('isAdmin') ?? false;
                if (isAdmin) {
                  Navigator.pushReplacement(
                      context,
                      MaterialPageRoute(
                          builder: (context) => AdminView())); // Mate
                } else {
                  Navigator.pushReplacement(
                      context,
                      MaterialPageRoute(
```

```
SizedBox(height: 16),
InkWell(
  onTap: () {
    Navigator.push(
      context,
      MaterialPageRoute(
          builder: (context) => ForgotPasswordPage()), // Materia
    );
  },
  child: Text(
    'Forgot password?',
    style: TextStyle(
      fontFamily: 'Roboto',
      fontSize: 14,
      color: Color(0xFFFE6610),
      fontWeight: FontWeight.normal,
    ), // TextStyle
  ), // Text
), // InkWell
SizedBox(height: 24),
Container(
  width: 360,
  height: 64,
  decoration: BoxDecoration(
    color: Color(0xFFF86C1D),
    boxShadow: [
      BoxShadow(
        color: Colors.black.withOpacity(0.25),
        offset: Offset(0, 4),
        blurRadius: 4,
```

Adding a "Forgot Password" link and a login button with the logic to authenticate the user and navigate based on the user type.

Social Login and Sign Up Link

```
                child: Text('Don\'t have an account? Sign up!',
                    style: TextStyle(
                      color: Color(0xFFC4C4C4),
                      fontSize: 14,
                      fontWeight: FontWeight.bold,
                      decoration: TextDecoration.underline,
                      decorationColor: Colors.grey,
                    )), // TextStyle // Text
                ), // InkWell
              ],
            ), // Column
          ), // Center
        ), // SingleChildScrollView
      ); // Scaffold
    }
  }
```

```
            SizedBox(height: 20),
            Text(
              '_________ Or _________',
              style: TextStyle(
                fontFamily: 'Roboto',
                fontSize: 20,
                color: Color(0xFFC4C4C4),
                fontWeight: FontWeight.normal,
              ), // TextStyle
            ), // Text
            Container(
              margin: EdgeInsets.symmetric(vertical: 16.0),
              child: Row(
                mainAxisAlignment: MainAxisAlignment.center,
                children: [
                  Icon(Icons.facebook, size: 45, color: Color(0xFF4267B2)),
                  SizedBox(
                    width: 20,
                  ), // SizedBox
                  GoogleLogo()
                ],
              ), // Row
            ), // Container
            SizedBox(height: 40),
            InkWell(
              onTap: () {
                Navigator.pushReplacement(
                  context,
                  MaterialPageRoute(builder: (context) => SignUpScreen()),
                );
              },
```

Adding options for social login and a link to the sign-up page.

```dart
Future<void> _firebaseMessagingBackgroundHandler(RemoteMessage message) async {
  // Initialize Firebase
  await Firebase.initializeApp();
  // Save the notification to SharedPreferences
  SharedPreferences prefs = await SharedPreferences.getInstance();
  List<String>? messagesJson = prefs.getStringList('notifications') ?? [];
  messagesJson.insert(0, jsonEncode(message.toMap()));
  await prefs.setStringList('notifications', messagesJson);
}
```

Defines a handler to manage background messages received via Firebase Cloud Messaging and save them to local storage.

```dart
Run | Debug | Profile
void main() async {
  WidgetsFlutterBinding.ensureInitialized();
  await Firebase.initializeApp();

  // Set up background message handler
  FirebaseMessaging.onBackgroundMessage(_firebaseMessagingBackgroundHandler);

  runApp(MaterialApp(
    home: MyApp(),
  )); // MaterialApp
}
```

Main Function

Initializes the Flutter app and Firebase, sets up the background message handler, and runs the main application.

```dart
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      title: 'Flutter Demo',
      theme: ThemeData(
        colorScheme: ColorScheme.fromSeed(seedColor: Colors.deepPurple),
        useMaterial3: true,
      ), // ThemeData
      home: AuthenticationWrapper(),
    ); // MaterialApp
  }
}
```

Defines the main app widget with a theme and sets the home screen to AuthenticationWrapper.

AuthenticationWrapper Widget

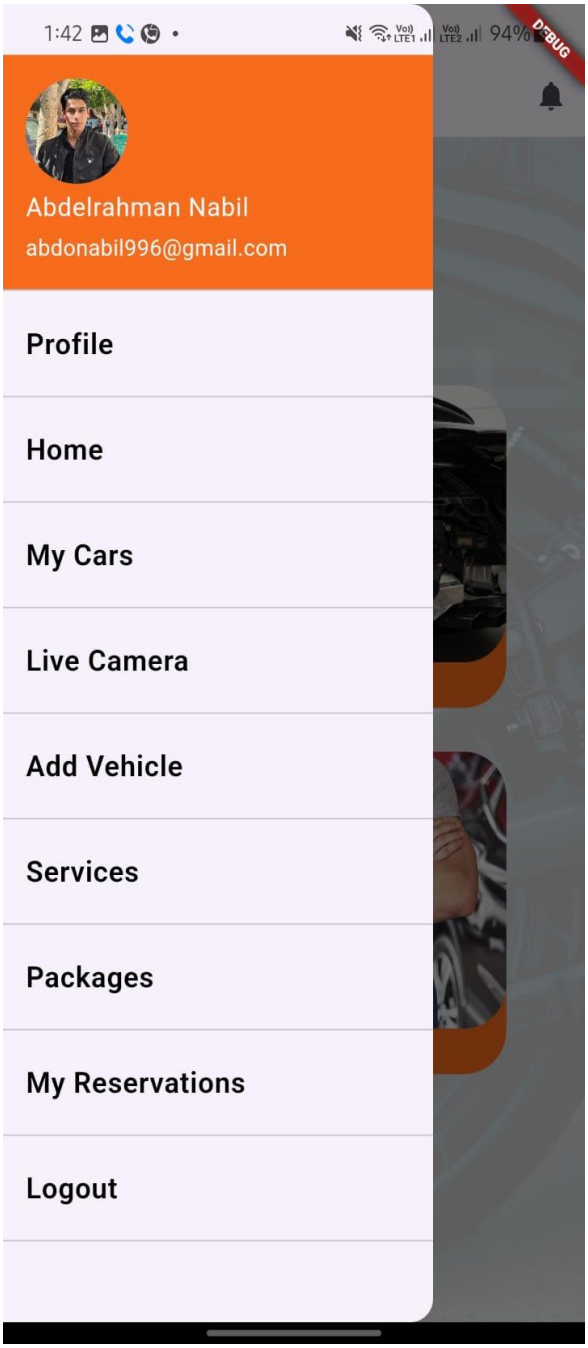
```dart
class AuthenticationWrapper extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return StreamBuilder<User?>(
      stream: FirebaseAuth.instance.authStateChanges(),
      builder: (context, snapshot) {
        if (snapshot.connectionState == ConnectionState.waiting) {
          return CircularProgressIndicator();
        } else if (snapshot.hasError) {
          return Text('Error fetching authentication state');
        } else {
          final User? user = snapshot.data;
          if (user == null) {
            // User is signed out, navigate to the onboarding screen
            return OnboardingScreen();
          } else {
            return FutureBuilder<DocumentSnapshot>(
              future: FirebaseFirestore.instance
                  .collection('users')
                  .doc(user.uid)
                  .get(),
              builder: (context, userSnapshot) {
                if (userSnapshot.connectionState == ConnectionState.waiting) {
                  return CircularProgressIndicator();
                } else if (!userSnapshot.hasData ||
                    !userSnapshot.data!.exists) {
                  return Text('User document not found!');
                } else {
                  final bool isAdmin =
                      userSnapshot.data!.get('isAdmin') ?? false;
                  if (isAdmin) {
                    return AdminView();
```

```dart
class NotificationsPage extends StatefulWidget {
  @override
  _NotificationsPageState createState() => _NotificationsPageState();
}

class _NotificationsPageState extends State<NotificationsPage>
    with AutomaticKeepAliveClientMixin {
  late FirebaseMessaging _messaging;
  List<RemoteMessage> _messages = [];

  @override
  bool get wantKeepAlive => true;

  @override
  void initState() {
    super.initState();
    _messaging = FirebaseMessaging.instance;

    FirebaseMessaging.onMessage.listen((RemoteMessage message) {
      setState(() {
        _messages.add(message);
        _saveNotifications();
      });
    });

    _loadNotifications();

    _messaging.getToken().then((token) {
      print("FCM Token: $token");
      // Send this token to your server
    });
  }
```

```dart
  void _loadNotifications() async {
    SharedPreferences prefs = await SharedPreferences.getInstance();
    List<String>? messagesJson = prefs.getStringList('notifications');
    if (messagesJson != null) {
      setState(() {
        _messages = messagesJson
            .map((json) => RemoteMessage.fromMap(jsonDecode(json)))
            .toList();
      });
    }
  }

  void _saveNotifications() async {
    SharedPreferences prefs = await SharedPreferences.getInstance();
    List<String> messagesJson =
        _messages.map((message) => jsonEncode(message.toMap())).toList();
    await prefs.setStringList('notifications', messagesJson);
  }
}
```

Manages real-time notification handling, saving and loading notifications from local storage, and obtaining the Firebase Cloud Messaging token.

4.9. Home Page:


1:42
94%
DEBUG
Abdelrahman Nabil
abdonabil996@gmail.com
Profile
Home
My Cars
Live Camera
Add Vehicle
Services
Packages
My Reservations
Logout


SAFE ZONE
Packages
Services

Home page

```dart
import 'package:flutter/material.dart';
import 'package:firebase_auth/firebase_auth.dart';
import 'my_cars.dart';
import 'profile.dart';
import 'packages.dart';
import 'add_car.dart';
import 'services.dart';
import 'notifications.dart';
import 'my_reservations.dart'; // Import MyReservations screen
import 'package:cloud_firestore/cloud_firestore.dart';
```

This snippet imports necessary Flutter and Firebase packages, as well as several custom Dart files that represent different screens in the app.

Main Function

The entry point of the application. It calls the runApp function and passes an instance of MyApp.

MyApp Class

```dart
class MyApp extends StatelessWidget {
  @override
  Widget build(BuildContext context) {
    return MaterialApp(
      // Define routes
      routes: {
        '/my_cars': (context) => MyCars(),
        // Add other routes as needed
      },
      home: HomeScreen(),
    ); // MaterialApp
  }
}
```

A stateless widget that sets up the MaterialApp, defines routes for navigation, and sets HomeScreen as the home screen.

HomeScreen Class

```dart
  Stream<DocumentSnapshot> _userInfoStream() {
    final user = FirebaseAuth.instance.currentUser;
    if (user != null) {
      return FirebaseFirestore.instance
          .collection('users')
          .doc(user.uid)
          .snapshots();
    }
    throw Exception('User not found');
  }
}
```

A helper method that returns a stream of user information from Firestore for the current authenticated user. If no user is found, it throws an exception.

```
Widget build(BuildContext context) {
        builder: (context, snapshot) {
          if (snapshot.connectionState == ConnectionState.waiting) {
            return Center(child: CircularProgressIndicator());
          } else if (snapshot.hasError) {
            return Center(child: Text('Error: ${snapshot.error}'));
          } else if (snapshot.hasData && snapshot.data != null) {
            final userData = snapshot.data!;
            final firstName = userData['firstName'] ?? '';
            final secondName = userData['secondName'] ?? '';
            final email = userData['email'] ?? '';
            final profileImgUrl = userData['profileImgUrl'] ?? '';

            return ListView(
              children: [
                UserAccountsDrawerHeader(
                  accountName: Text('$firstName $secondName'),
                  accountEmail: Text(email),
                  currentAccountPicture: CircleAvatar(
                    radius: 50,
                    backgroundColor: Colors.grey,
                    backgroundImage: profileImgUrl.isNotEmpty
                        ? NetworkImage(profileImgUrl)
                        : null,
                  ), // CircleAvatar
                  decoration: BoxDecoration(
                    color: Color(0xFFF76B1C),
                  ), // BoxDecoration
                ), // UserAccountsDrawerHeader
                // Add divider
                ListTile(
```

```
Widget build(BuildContext context) {
                      NotificationsPage(
                    userId: userId,
                  ), // NotificationsPage
                  transitionsBuilder:
                      (context, animation, secondaryAnimation, child) {
                    var begin = Offset(1.0, 0.0);
                    var end = Offset.zero;
                    var curve = Curves.easeInOutQuart;
                    var tween = Tween(begin: begin, end: end)
                        .chain(CurveTween(curve: curve));
                    var offsetAnimation = animation.drive(tween);
                    return SlideTransition(
                      position: offsetAnimation,
                      child: child,
                    ); // SlideTransition
                  },
                ), // PageRouteBuilder
              );
            }
          },
        ), // IconButton
      ],
    ), // AppBar
    drawer: Drawer(
      child: StreamBuilder<DocumentSnapshot>(
        stream: _userInfoStream(),
        builder: (context, snapshot) {
          if (snapshot.connectionState == ConnectionState.waiting) {
            return Center(child: CircularProgressIndicator());
          } else if (snapshot.hasError) {
```

```
class HomeScreen extends StatelessWidget {
  const HomeScreen({Key? key}) : super(key: key);

  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: const Text(
          'SAFE ZONE',
          style: TextStyle(
            color: Color(0xFFF76B1C),
            fontSize: 24,
            fontFamily: 'Roboto Serif',
            fontWeight: FontWeight.w600,
            height: 0.5,
            letterSpacing: -0.41,
          ), // TextStyle
        ), // Text
        centerTitle: true,
        actions: [
          IconButton(
            icon: const Icon(Icons.notifications),
            onPressed: () async {
              final user = FirebaseAuth.instance.currentUser;
              if (user != null) {
                final userId = user.uid;
                Navigator.push(
                  context,
                  PageRouteBuilder(
                    transitionDuration: Duration(milliseconds: 500),
                    pageBuilder: (context, animation, secondaryAnimation) =>
                        NotificationsPage(
```

Defines the HomeScreen widget with an app bar, a navigation drawer, and a body.

The app bar includes a title and a notifications button.

The navigation drawer contains a StreamBuilder to fetch and display user information from Firestore.

The body contains two main options (Packages and Services) that navigate to respective screens on tap.

_userInfoStream Method

Chapter (5): Conclusion and result

5.1. Conclusion:

The "Safe Zone" mobile app marks a significant milestone in the realm of automotive care, providing an all-encompassing solution for vehicle owners who prioritize convenience, security, and comprehensive service. Our primary objective was to alleviate the common concerns of car owners, especially those who frequently travel or need to leave their vehicles unattended for extended periods. By offering a variety of subscription-based packages, the app ensures not just secure parking but also regular maintenance to prevent common issues like flat tires and oil degradation.

A standout feature of the Safe Zone app is its seamless integration of cutting-edge technology to enhance user experience and vehicle security. The live cam feature, for instance, allows users to monitor their cars in real-time, delivering unparalleled peace of mind through round-the-clock surveillance. Furthermore, the app empowers users to request additional services on demand, ensuring that their vehicles receive personalized care tailored to specific needs.

The development journey of Safe Zone has been a comprehensive endeavor involving meticulous research, innovative design, and rigorous testing. Our focus has been on creating an intuitive user interface backed by a robust and secure backend system capable of managing user data, service requests, and payment processing efficiently.

In summary, the Safe Zone mobile app is a trailblazing solution in the car care industry, seamlessly blending convenience, security, and extensive service offerings. It highlights the transformative potential of mobile technology in enhancing traditional services and improving user experiences. Looking ahead, we envision numerous opportunities for further development and expansion, such as integrating with automobile manufacturers, offering more personalized services, and penetrating new markets. We believe that Safe Zone has the potential to revolutionize the way car owners manage the care and maintenance of their vehicles, setting a new benchmark in the industry

5.2. Future Work:

Another Features:

- **Loyalty Program:**

Earn points for bookings, redeemable for discounts or free services.

- **Emergency Assistance:**

Request roadside help for flat tires, dead batteries, or lockouts.

- **Eco-Friendly Options:**

Access oil recycling, eco-friendly car washes.

- **Online Payment:**

Pay from the app with a button click.

Advanced Features:

- **Predictive Maintenance:**

AI analyzes data to predict maintenance needs and send reminders.

- **In-App Navigation:**

Get guided directly to available parking spots.