

Final Year Project Report

Full Unit – Final Report

SwiftFuel - Fuel Delivery Mobile Application

Fahad Riaz (101027495)

A report submitted in part fulfilment of the degree of

BSc (Hons) in Computer Science

Supervisor: Vasudha Darbari



Department of Computer Science
Royal Holloway, University of London

April 11, 2025

Declaration

This report has been prepared based on my own individual work. Where other published or unpublished sources have been consulted, they have been appropriately acknowledged through in-text citations and referenced in the bibliography section.

Furthermore, I would like to clarify that in many parts of this report, particularly in sections involving personal reflection, project-specific development processes, testing, and discussion of practical challenges, the content is based on my own original work and experiences. As such, these sections do not contain in-text citations, since they reflect personal analysis and practical implementation rather than academic research.

I confirm that this report adheres to the university's academic integrity guidelines and that all external material has been credited where used.

Full Project Demo Video Link: <https://youtu.be/IuCqbYZZ0aE>

Word Count: 12,231 words excluding title page, declaration page, table of contents, and bibliography.

Student Name: Fahad Riaz

Date of Submission: 10th April 2025

Signature:



Table of Contents

Introduction.....	4
Project Specification	6
Chapter 1: Literature Review.....	8
1.1 Market Analysis and Existing Solutions	8
1.2 Technologies in Mobile App Development	9
1.3 Real-Time GPS Tracking (Google Maps API)	10
1.4 Conclusion	10
Chapter 2: Project Planning and Methodology	11
2.1 Development Methodology	11
2.2 Project Timeline	11
2.3 Features Prioritization: MoSCoW Method	12
2.4 Risk Assessment and Mitigation Strategies	13
Chapter 3: System Design	15
Chapter 4: Implementation Details	18
4.1 Login Screen.....	18
4.2 Register Screen	20
4.3 Fuel Ordering Screen	21
4.4 Order Tracking Screen	24
4.5 Payment Screen	25
4.6 Other Screens.....	27
Chapter 5: Testing and Evaluation	29
5.1 Integration Testing	29
5.2 Manual Testing	30
5.3 User Acceptance Testing (UAT)	31
5.4 Real Device Testing (Samsung Galaxy S24 Plus)	32
5.5 Evaluation Summary.....	33
Chapter 6: Professional Issues.....	34
Chapter 7: Results and Discussion	36

7.1	Summary and Timeline of Gitlab Diary	36
7.2	Challenges during Development.....	38
7.3	Screenshots of the Final Product (UI and Features)	39
Chapter 8:	Future of SwiftFuel.....	45
8.1	Feature Enhancements and Planned Future Work	45
8.2	Real-World Implementation as a Business.....	45
Chapter 9:	Critical Reflection on Learning	47
Chapter 10:	Conclusion	49
Bibliography	50

Introduction

There is an increasing demand for fuel because of many cars being introduced in the auto industry and this has led to a need for building fuel delivery applications that require the users to place an order for their vehicle when their fuel runs out or when it is low without the hassle of going to the petrol stations [1]. The app aims to provide the users with a convenient and efficient platform for ordering fuel to their vehicles at any given location, this can be done by using real-time GPS tracking of the delivery vehicle which will allow the users to monitor the status. Also, the app will implement secure payment integration for ordering fuel. Users will be presented with an intuitive platform that will ease the whole process of ordering and receiving fuel at their locations [2]. This will change how people approach refuelling in their day to day lives because it will be a time saving and seamless experience [2]. By using Flutter, the app will have support for both Android and iOS mobile phones making it cross platform with an attractive user interface because of Flutter. The app will also integrate with Google Maps because of its powerful navigation capabilities for real time GPS tracking. There are many benefits for this app, users will be able to order fuel from anywhere which will also provide accessibility to remote areas which will make sure fuel is available for everyone [2]. The process for the app will be simple: Users register on the app and login, share their current location, vehicle details and select the fuel type they want, order fuel, then, the fuel truck will reach the user's location at the given time. There was research conducted on this matter in which users were interviewed to find out if such an app would be successful or not, the response for this research showed that majority (68.8 percent) of the users were interested in having fuel delivered to them and they wanted such an app to be developed [3]. An in-built GPS is found in almost all mobile devices nowadays which enables such devices to accurately determine their location anywhere on Earth, this is done by receiving signals from GPS satellites [4]. Here is a simple image below to understand how the app will work [5].

How On-Demand Fuel Apps Work

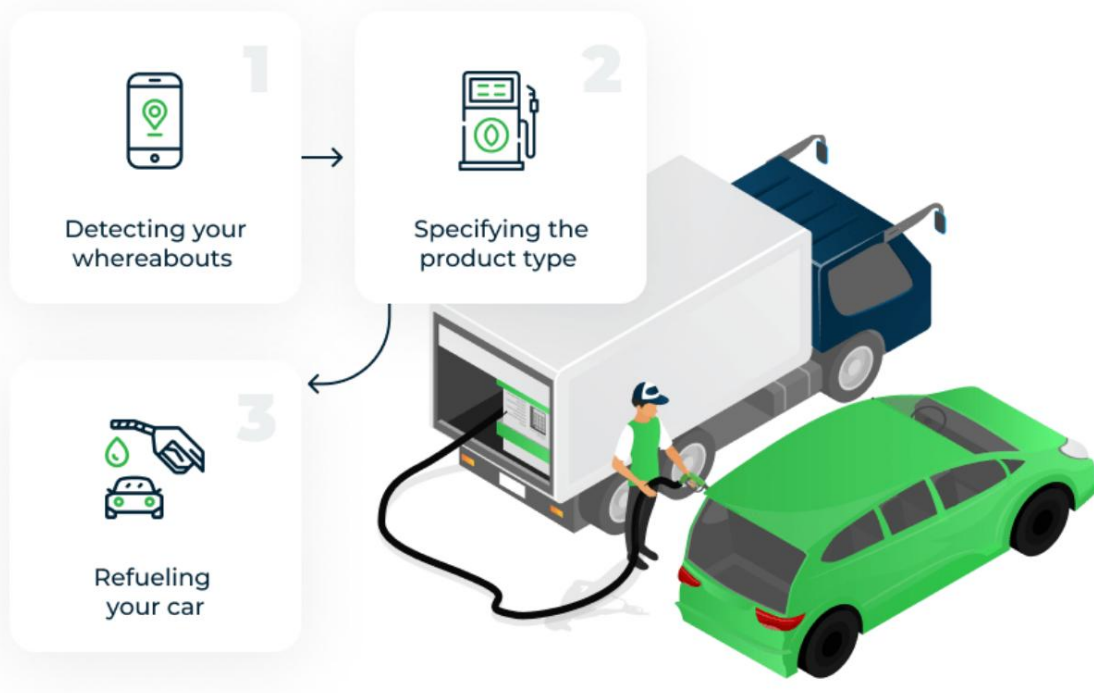


Figure 1: Workflow of the fuel delivery app.

The project's main objectives are:

1. Develop an easy to use and attractive interface/application where users can place fuel orders by selecting fuel type, and delivery location.
2. Implement real time GPS tracking to provide users with live updates on the fuel delivery status.
3. Ensure secure payment system and processing to protect user data.
4. Design and build a scalable system capable of handling multiple user orders and managing delivery status updates.
5. Develop a secure system for user registration, login, and profile management, allowing users to easily create accounts, track their order history and manage their profiles.

Project Specification

1. Project Overview

Mobile App Name: SwiftFuel – Fuel Delivery App

Summary: SwiftFuel is a mobile application for Android designed to provide on-demand fuel delivery service, which will allow customers to refuel their vehicles conveniently at their provided location, without the need for going to the gas station. The app aims to deliver a seamless experience for users by offering contactless and efficient fuel delivery services, anytime, anywhere.

2. Functional Requirements

- **User Registration and Authentication:** Users must be able to register, log in and log out safely using Firebase Authentication.
- **User Dashboard/Home:** The app should provide a user-friendly home screen where users can view and easily navigate to different parts of the app.
- **Fuel Ordering System:** Users should be able to select fuel type, enter their vehicle details, and select their location for delivery using Google Maps integration.
- **Payment:** Integration of secure payment system for user transactions.
- **Order Confirmation:** Users should see their order details for reviewing before placing their orders.

3. Non-Functional Requirements

- **Usability:** The app should be an intuitive, user-friendly interface that is easy to use and navigate.
- **Performance:** The app must be responsive, with minimum delays during different tasks.
- **Scalability:** The backend system should be able to handle multiple users and requests.
- **Security:** User data must be well protected to ensure security and privacy compliance.

4. Technical Specifications

- **Platform:** App is built for Android using Flutter.
- **Backend:** Firebase services used, which includes Firestore database and Firebase Authentication.
- **APIs:** Google Maps API for location-based features, Stripe API for secure payment system.
- **Programming Languages:** Dart.
- **Development Tools:** Android Studio, Git

5. User Requirements

- Target Users: Vehicle owners who want convenient and contactless refuelling for their vehicles without going to the petrol stations.
- User Stories:
 - *“As a user, I want to register and login easily to access my account securely.”*
 - *“As a user, I want to choose my current location so that I can have fuel delivered to me after ordering it from the app.”*
 - *“As a user, I want to see my past orders to keep track of them.”*

6. Constraints

- Time Constraints: The project needs to be completed within the academic year (2024/25).
- Budget Constraints: The app will rely on Firebase’s free tier and Google Maps API within limited use.
- Technical Constraints: Development focuses on Android only due to hardware limitations, as the project development environment is Windows-based.

7. Assumptions

- Users have a stable internet connection while using the app.
- Users provide accurate location details to receive appropriate service.
- The initial service area is restricted geographically to make logistics feasible.

8. Success Criteria

- Successful user registration, login, and logout features.
- Users can place fuel orders with selected location and fuel type etc.
- Proper integration of Google Maps.
- Proper integration of Stripe Payment system.
- Successful saving of different data to the Firestore database.
- Users can track their orders in real-time using Google Maps integration.

9. System Requirements

- Hardware: Android device.
- Software: Android OS with internet access.

Chapter 1: Literature Review

The fuel delivery industry has gone through major phases with the emergence of new digital technologies, this literature review will explore existing research, solutions, market analysis, and mobile app development. On-demand fuel delivery services have emerged as a convenient alternative to the traditional fuelling method at gas stations, the main concept in this involves delivering fuel to customers at their locations which ensures convenience and time saving.

1.1 Market Analysis and Existing Solutions

There already exist some notable competitors in the market for on-demand fuel delivery services such as Booster in the USA and CAFU in the Middle East so the market has seen some steady growth, driven by consumer demand for more convenient services. Companies like Booster and CAFU have already successfully tapped into the market by providing on-demand fuel delivery services with easy to use mobile applications.

CAFU operates in the United Arab Emirates, specifically in Dubai, and they deliver Super 98 and Super 99 fuel to the people living there in Dubai when they order it using their mobile app [6]. When a user orders fuel from the official CAFU mobile app, they send CAFU trucks to the user's provided location so that the user's vehicle can be refuelled [6]. They have multiple stakeholders in their business and even the oil and fuel companies are an important part of their supply chain [6]. CAFU's existing solution changed the landscape of oil and gas transportation and logistics with their services and reports state that the UAE market has around 3.4 million cars on the road and the annual revenue per customer for CAFU is 264 AED which makes the market's annual estimate at around 897 million AED [6]. This growth in the UAE market has caught the attention of other countries as well and the growth of this market is connected to the general trend toward service-based mobile applications. An analysis of CAFU's existing application highlights features such as real-time GPS tracking, integrated payment systems, and fuel quality guarantees. Just like other on-demand services like food delivery, delivering a product like fuel to the user is not a revolutionary concept but it does become innovative because of the use of technology to enhance customer experience and to make it efficient and convenient [7]. CAFU also does not charge their customers for deliveries, instead, they create customer loyalty by rewarding the customers with points that can be redeemed against other services [7].

The global on-demand fuel delivery market size was 0.33 billion USD in 2024 and it is projected to touch 2.04 billion USD by 2032 which shows a compound annual growth rate (CAGR) of 15.97 percent from 2024 to 2032 [8]. This shows how important this market really is and the increasing use of fuel vehicles in different countries is expected to support the expansion of worldwide on-demand fuel delivery market. (Shown in Figure 2 below)

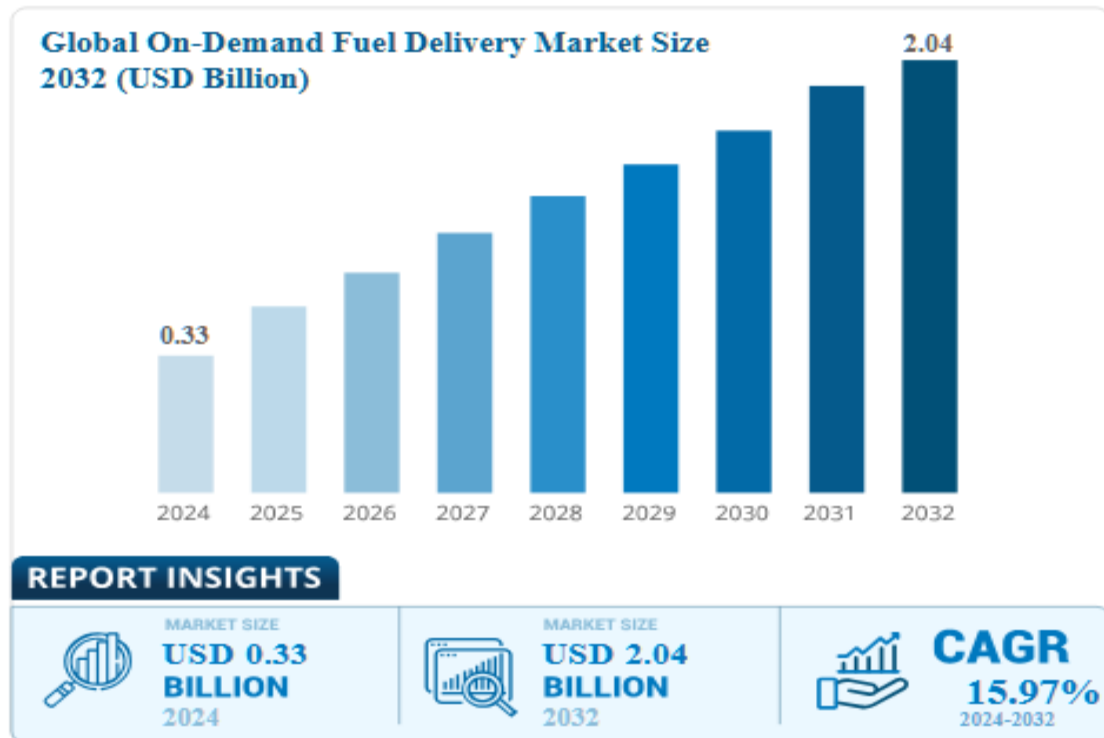


Figure 2

1.2 Technologies in Mobile App Development

The development of mobile applications for apps like fuel delivery requires robust and flexible technological solutions. Cross-platform development tools like Flutter are very popular because of their ability to reduce development costs and time. Flutter is an excellent framework for developing such applications with a single codebase for both iOS and Android.

Google has defined Flutter as a portable UI toolkit for developing beautiful looking natively compiled mobile apps from a single codebase [9]. A flutter project is written in Dart programming language, at the top level, Flutter provides widgets to work with which are used to create interface objects that we see usually on iOS and Android. Dart language is also faster as compared to other languages like JavaScript and it supports AOT and JIT compilation which is why it is perfect for the hot reload feature in Flutter [9]. Flutter comes with pre-made packages like Cupertino for iOS and Material for Android and these packages are used for creating the UI as they contain the interface elements which are specific to the themes of iOS and Android [9]. A survey was conducted in which 100 developers from different IT companies participated, these developers had 6 months to 8 years of experience and according to them in this survey, they thought that Flutter was good in terms of performance as compared to other technologies, 51.21 percent of the developers were in favour of Flutter [10]. These developers also claimed that Flutter has less learning curve than Java and Kotlin. The best feature in Flutter is the hot reload, this provides the ability to keep the app running and quickly reflect any changes made in the code at runtime without losing state on emulators or any hardware, in addition to this, Flutter also comes with a rendering engine and some development tools that help accelerate application development [11].

Firebase provides a reliable Backend-as-a-Service (BaaS) solution that eases tasks like user authentication, database management, and real-time data synchronization. Firebase is a web application platform which is designed to aid developers in building mobile applications, it stores data in JSON format which doesn't use querying for any action and it is used as the backend system with many services available. One of these services is Firebase Authentication, this is a service that can authenticate users to create accounts and log-in and this comes with a user management system as well. Another service that Firebase provides is the real-time Firestore database [12].

1.3 Real-Time GPS Tracking (Google Maps API)

Real-Time GPS Tracking is an important part of on-demand fuel delivery services. By integrating Google Maps API, mobile apps can offer accurate location tracking which will allow delivery agents to locate customers easily to deliver the fuel and the API can also be used to provide real-time updates on the delivery status.

There was a study conducted which demonstrated the use of GPS services to monitor the live locations of garbage trucks, displaying their real time locations on smartphones using Google Maps API. The app for this was developed in Android Studio and the approach was similar to what should be in a fuel delivery application. In the study, the mobile app gathers the coordinates of the garbage trucks and then displays it to the users using Google Maps API. This allows for the users to view the vehicle's location in real time. The location information is presented as latitude and longitude and it is collected through GPS services embedded within the garbage trucks. The Google Maps API plays an important role in this by displaying the vehicle's current live location and route, making it easier for the users to see where the truck is [13].

Similar to the study mentioned above, GPS services embedded in smartphones can be used to track the live location of fuel delivery trucks or their delivery agents using their smartphone's GPS capabilities and this implementation will allow the users to view the live location of the delivery vehicle using an integrated Google Maps interface.

1.4 Conclusion

The literature reveals that on-demand fuel delivery services are poised to transform the way consumers access fuel. Advancements in mobile application development, combined with technologies such as Flutter, Firebase and Google Maps API can be leveraged to develop a robust and scalable Fuel Delivery App. The analysis and comparison of existing solutions and technologies highlight that innovation in this sector relies heavily on providing the best experience through use of modern technologies.

Chapter 2: Project Planning and Methodology

This section outlines the strategic approach and timeline for developing the fuel delivery mobile application. It includes a detailed project plan, covering each key milestone, as well as the methods and practices used throughout the development process. The chosen methodology is designed to accommodate changing requirements while keeping a focus on delivering value to users and meeting project objectives within the specified time-period.

2.1 Development Methodology

The development of my Fuel Delivery Mobile Application follows an Agile methodology, which provides a flexible approach to software development. This methodology is well suited for this purpose as it allows for iterative improvements, incremental feature additions, and continuous adaptation based on user/supervisor feedback. Agile methodology was chosen for its ability to manage the changing requirements effectively, in a project like this where new features are tested and integrated regularly. Agile provides the framework to breakdown the development process into smaller sub-tasks which are manageable. Agile software development shows that developers should start with simple and predictable approximations to the final requirements and then continue to implement and increment the details of these requirements overtime [14] .

The project so far has been developed using iterative and incremental approach as the initial iterations focused on building the core functionalities of the mobile app, such as user authentication, user dashboard and the basic user interface. With time, more advanced features were implemented such as Google Maps API integration and fuel ordering process and system. All of this ensures that every feature is properly functional before moving onto the next one which helped in minimizing bugs and maintaining stability.

Frequent testing is also an important part of the development stages, every feature was properly tested on a virtual android device (emulator) to ensure a smooth user experience. The development process has also been very heavily user-centric to make sure that the user experience is intuitive and with enhanced convenience. All the features so far were implemented with the end-user in mind so that the best product is delivered.

2.2 Project Timeline

Term 1:

- Week 1: Setup the Flutter development environment for iOS and Android and research similar apps.
- Weeks 2–3: Working on the app design, finalizing the main features and interactions of the app, implementing firebase authentication, and building the UI for placing fuel orders.

- Week 4: Design database schema, some backend planning like using APIs etc.
- Weeks 5–6: Integrating Google maps for user location.
- Weeks 7–8: Testing for core features and remaining development from previous weeks.
- Weeks 9–10-11: Preparing and working on the interim report and presentation, any bug fixes or quality of life improvements.

Term 2:

- Weeks 1–2: Implementing push notifications and real time GPS updates, Integrating secure payment system.
- Weeks 3–4: Developing interface for delivery drivers/admins and developing user profiles (account management) + Implementing fuel order delivery tracking screen for users.
- Weeks 5–6: Any remaining development and testing the performance of the app.
- Weeks 7–8: User Acceptance Testing for feedback.
- Week 9: Deployment of the app if required.
- Weeks 10–11: Final testing, bug fixes, preparing and working on the final report and final submission.

2.3 Features Prioritization: MoSCoW Method

The MoSCoW method is an effective prioritization technique used in project management to determine the importance of several features and requirements. For the Fuel Delivery app, this method was used to categorize the features into two levels of importance: Must Have and Should Have. This ensures that the most important features of the application are developed first and this also contributes to the Agile methodology being used.

Must Have Features:

These are the core features that are required for the app/project to function properly as intended:

- User Registration and Login: Users must be able to create accounts and log in to place fuel orders
- Fuel Ordering System: The core feature where users select the fuel type, add all their details, select their delivery location, and place the order
- Real Time GPS Tracking: Users should be able to track the fuel delivery vehicle/driver in real time.
- Secure Payment Integration: A payment gateway must be implemented to deal with user transactions for placing orders.
- Google Maps Integration: This should be implemented to allow location selection and navigation for delivery.

- Notifications: Users should receive real time notifications about their order status.
- Driver Interface: A separate interface for the fuel provider drivers to update and manage the fuel deliveries.

Should Have Features:

These are also important but not critical features, these will enhance the user experience or add value to the project:

- Push Notifications: Implementing push notifications to notify users about their order updates.
- Order History: Users should be able to view their past orders and details related to these orders.
- User Profile Management: Allow users to manage and update their personal details, payment methods and other details.

2.4 Risk Assessment and Mitigation Strategies

- Hardware Failure: While working on my mobile app, there is a possibility that the hardware on which I am doing the project might fail for some reasons leading to data loss and the progress made being wasted. To counter this, I will make sure that I am working on the project and keeping everything related to it under version control (Gitlab) and I will make sure that I am committing and pushing regularly to the repository.
- Uneven Balance Between Report and Code: Focusing too much on either the technical part (coding) or the written reports could lead in one of them being incomplete in time or insufficient in detail. To counter this, I will regularly review progress for both to make sure that both are receiving balanced attention, and I will make sure to follow my timeline and milestones to prevent this from happening.
- Technical Issues with GPS Tracking: Real time GPS Tracking may be inaccurate or may fail sometimes, to mitigate this I will properly integrate Google Maps with error handling.
- Scope Creep: A project's goal or requirements can change beyond what was originally planned upon, this can happen by adding too many features which could lead to missing deadlines. For this, I will stick to the original plan and use agile principles to manage priorities.
- Insufficient Testing: Poor testing during or after production might result in unfound bugs or poor user interface. For this, I will regularly test my product during development and afterwards as well.
- Time Management Issues: Delays in successfully achieving the milestones mentioned above in the timeline section could push the project beyond the given deadlines, so, to reduce this risk I will strictly follow the timeline, regularly review the progress, and break tasks into smaller sub-tasks which are manageable and easier to implement.

- **Lack of Required Skills:** It is possible that while working on this project, I might need to deal with new programming languages or technologies which I have not used before. For this, I will make sure that I also schedule time for learning these new skills by researching and doing.

Chapter 3: System Design

The system design for the fuel delivery mobile app aims to balance usability, scalability, and functionality. This section outlines the design choices and UI elements, supported by a low fidelity storyboard that showcases the core features and user journey for the UX/UI of the project.



Figure 3

The user journey for the user experience is broken down into five main steps as shown in Figure 3 above:

1. User Registration/Login: The starting point for the users to register or log in with their email and password.
2. Home/Dashboard: After logging in, users access the dashboard to initiate fuel orders.
3. Fuel Ordering: Users provide details like delivery location, fuel type, and vehicle information to order fuel.
4. Payment: Users pay for their fuel order.
5. Order Tracking: After the order is successfully placed, users track their delivery using real-time GPS tracking.

The storyboard for this User Journey (Figure 4 below) showcases the low fidelity user interface design of the mobile app, highlighting the flow and functionality of each screen:

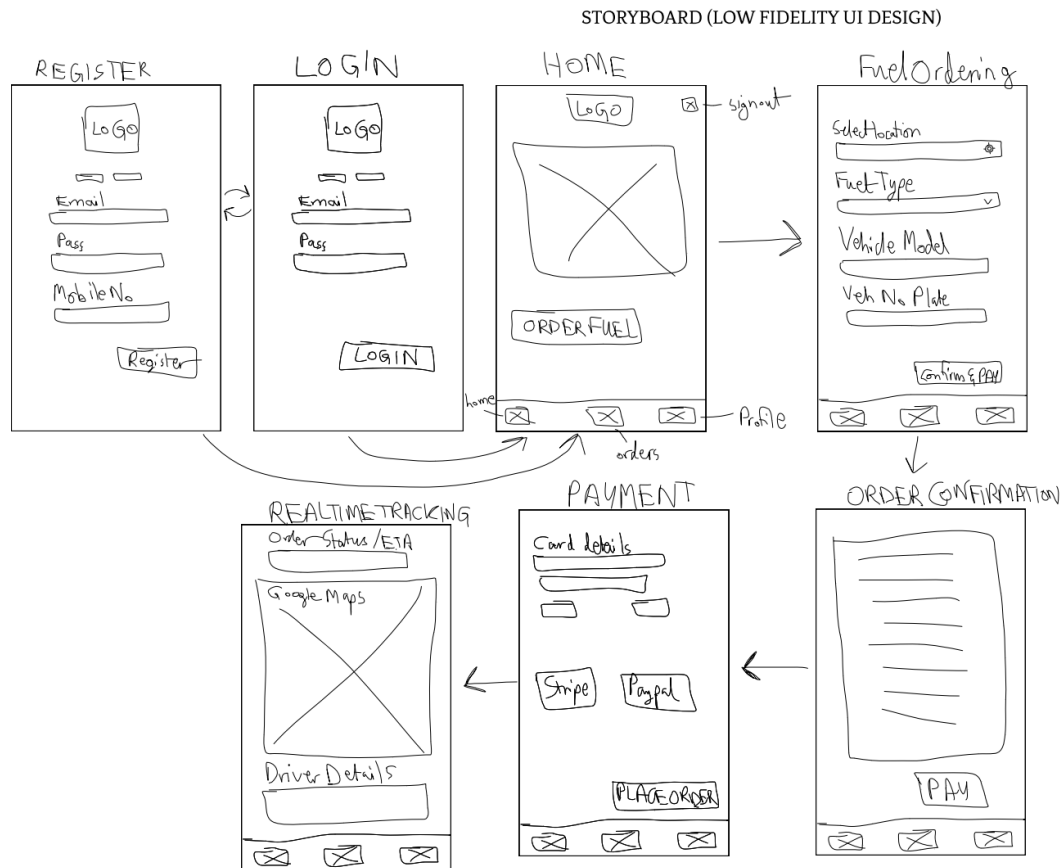


Figure 4

1. Register Screen: This allows the users to create an account by providing their email address, password, and mobile number. The screen is minimalistic, focusing on easy usage with clearly labelled input fields and prominent buttons.
2. Login Screen: Has input fields for users to log in with their registered details.
3. Home Screen: This is the central hub of the app, shows an “Order Fuel” button which allows the users to initiate the fuel ordering process. The home screen also includes navigation to other sections of the app like orders and profile.
4. Fuel Ordering Screen: This screen collects key details for the user’s order.
5. Order Confirmation Popup: This popup displays the summary of the user’s order, users can confirm then go to the next payment screen.
6. Payment Screen: This screen provides the users for using payment gateways like PayPal or Stripe to pay for their fuel orders.
7. Real Time Tracking Screen: This shows the user the status of their delivery and live location of the delivery agent/truck on a map. Users can also see ETA and driver details etc.

The storyboard provided has significantly informed the design considerations for the mobile app, key considerations include:

1. The apps UI design focuses on simplicity and clarity, which ensures that the users can navigate through the app easily. Each screen has distinct buttons and content which reduces cognitive load on the users.
2. A consistent visual and functional design across all screens.
3. Navigation is streamlined with a bottom navigation bar on the home screen which ensures the best user experience.
4. The storyboard also accounts for user feedback mechanisms such as confirmation popups and notifications.

This storyboard has played a very important role in developing and building the actual user interface of the app and by visually representing this, it has provided a clear framework for development while ensuring alignment with the project's objectives.

The system design for the mobile app implements main features of user-centred design (UCD), this emphasizes the importance of understanding user's needs, tasks, and goals to create the UI that aligns with their expectations [15] . To achieve this, the UI of the app has been designed with focus on usability and simplicity, ensuring that the users can successfully complete core tasks with minimal effort and cognitive load. By staying true to the design principles, the app aims to provide the best possible user experience, for example, the UI elements for navigation and interaction are visually consistent throughout the app to reduce confusion and build familiarity. In addition to this, the addition of real-time feedback mechanisms and order confirmation dialog enhances user trust and engagement. Hence, the app's design has been made in such way that it addresses the user's pain points and ensuring it meets both the functional requirements and aesthetic expectations by drawing on the best UCD practices in prototyping.

While this section focuses on the conceptual and low fidelity design for the app, the actual implementation of the UI with screenshots of the developed UI/app, is covered in the Results section later. This ensures a clear distinction between the design process and the actual outcome achieved during development.

Chapter 4: Implementation Details

This chapter presents a technical overview of the main implementation components of the SwiftFuel mobile application. The main screens are discussed in terms of its functionality, key logic, and implementation details. The app was built using the Flutter framework in Dart, with Firebase used for authentication and backend services. The UI was developed with a focus on simplicity, usability, and responsiveness, ensuring that the users have an intuitive experience.

Screens are categorized by their primary roles in the user journey, and only the most relevant code portions are highlighted to demonstrate the core logic used during development.

4.1 Login Screen

Purpose:

This screen is the user's entry point into the application. It allows the existing registered users to sign in securely with their email and password using Firebase Authentication. Depending on the authenticated user's role (customer or driver), the user is then redirected to the appropriate screen which is either the HomeScreen (for customers) or the DeliveryDashboardScreen (for delivery partners).

Key Features Implemented:

- Email and password login using Firebase Authentication
- Role based Access Control (RBAC)
- Password visibility toggle using a ValueNotifier
- Navigation to the RegisterScreen for new unregistered users

Main Code Snippets with Explanation: (Next Page)

```

void _handleLogin() async {
  setState(() => _isLoading = true);

  String email = _emailController.text.trim();
  String password = _passwordController.text.trim();

  if (email.isNotEmpty && password.isNotEmpty) {
    var userData = await _authService.login(email, password);
    if (userData != null) {
      if (userData['role'] == 'customer') {
        Navigator.pushReplacement(
          context, MaterialPageRoute(builder: (context) => const HomeScreen()));
      } else if (userData['role'] == 'driver') {
        Navigator.pushReplacement(
          context, MaterialPageRoute(builder: (context) => const DeliveryDashboardScreen()));
      }
    } else {
      ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(content: Text('Login failed')),
      );
    }
  }

  setState(() => _isLoading = false);
}

```

This function retrieves user credentials from text fields, passes them to the AuthService, and handles redirection based on the user's role. Error feedback is shown using a SnackBar.

```

ValueListenableBuilder(
  valueListenable: _isPasswordVisible,
  builder: (context, value, child) {
    return _buildTextField(
      _passwordController, 'Password', Icons.lock, !value,
      suffixIcon: IconButton(
        icon: Icon(value ? Icons.visibility : Icons.visibility_off),
        onPressed: () {
          _isPasswordVisible.value = !_isPasswordVisible.value;
        },
      ),
    ); // IconButton
  },
), // ValueListenableBuilder

```

This toggle improves usability by allowing users to see their password input if needed, reducing errors during login.

Challenges and Noteworthy Decisions:

- Password visibility toggle was implemented using a ValueNotifier for minimal overhead and cleaner state management compared to setState.

- A loading indicator was added during the login process to improve UX and prevent duplicate submissions.

4.2 Register Screen

Purpose:

This screen allows new users to create an account within the app. The screen captures important user information such as email, password, and a UK mobile number, and securely registers these using Firebase Authentication. After a successful registration, user data is stored in the backend Firestore database which includes a default role assignment of customer.

Key Features Implemented:

- User registration using Firebase Authentication
- Storage of user profile and data in Firestore
- Form validation for empty fields
- Password visibility toggle
- Visual feedback during registration

Main Code Snippets with Explanation:

```
void _handleRegister() async {
  setState(() => _isLoading = true);

  String email = _emailController.text.trim();
  String password = _passwordController.text.trim();
  String mobileNumber = _mobileController.text.trim();

  if (email.isNotEmpty && password.isNotEmpty && mobileNumber.isNotEmpty) {
    User? user = await _authService.register(email, password);

    if (user != null) {
      String fullMobileNumber = '+44 $mobileNumber';

      // Store User Data & Role in Firestore
      await _firestore.collection('users').doc(user.uid).set({
        'email': email,
        'mobileNumber': fullMobileNumber,
        'role': 'customer',
        'createdAt': Timestamp.now(),
      });

      Navigator.pushNamed(context, '/login');
      ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(content: Text('Registration successful')));
    } else {
      ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(content: Text('Registration failed. Try again.')));
    }
  } else {
    ScaffoldMessenger.of(context).showSnackBar(
      const SnackBar(content: Text('Please fill all fields.')));
  }

  setState(() => _isLoading = false);
}
```

This function handles the entire registration process, from field validation to calling Firebase Authentication, then storing the user provided data into Firestore.

Challenges and Noteworthy Decisions:

- The role field was hardcoded as customer at this stage because drivers/delivery persons were manually registered in the backend, but this could be extended in the future for driver or admin registrations.
- Registration automatically navigates back to the LoginScreen after success, providing a smooth transition in the user journey.

4.3 Fuel Ordering Screen

Purpose:

This screen is the most important components of the fuel delivery mobile app. It enables users to order fuel by selecting their current location on an interactive map, choosing a fuel type, and entering their vehicle number plate. This screen integrates with Google Maps, Firebase, and a PaymentScreen.

Key Features Implemented:

- Integration with Google Maps for real-time location selection
- Automatic detection of user's current location using the location package
- Dropdown menu for selecting fuel type
- Order confirmation popup summarizing user order details
- Transition to PaymentScreen and creation of a Firestore order document upon successful payment

Main Code Snippets with Explanation:

```
Future<void> _getUserCurrentLocation() async {  
  Location location = Location();  
  bool _serviceEnabled = await location.serviceEnabled();  
  if (!_serviceEnabled) {  
    _serviceEnabled = await location.requestService();  
    if (!_serviceEnabled) return;  
  }  
  
  PermissionStatus _permissionGranted = await location.hasPermission();  
  if (_permissionGranted == PermissionStatus.denied) {  
    _permissionGranted = await location.requestPermission();  
    if (_permissionGranted != PermissionStatus.granted) return;  
  }  
  
  LocationData _locationData = await location.getLocation();  
  _selectedLocation.value = LatLng(_locationData.latitude!, _locationData.longitude!);  
  
  _mapController?.animateCamera(  
    CameraUpdate.newCameraPosition(  
      CameraPosition(target: _selectedLocation.value!, zoom: 15),  
    ),  
  );  
};
```

This function checks for GPS and location permissions, retrieves the device's current location, and centres the map to that point. It makes the ordering process faster and easier for users by auto-pinning their location.

```

: GoogleMap(
  initialCameraPosition: CameraPosition(target: location, zoom: 15),
  onMapCreated: (controller) {
    _mapController = controller;
    _getUserCurrentLocation();
  },
  onTap: (LatLng loc) {
    _selectedLocation.value = loc;
  },
  markers: location != null
    ? {
        Marker(markerId: const MarkerId("selected-location"), position: location),
      }
    : {},
  myLocationEnabled: true,
  myLocationButtonEnabled: true,
), // GoogleMap

```

This interactive map allows the user to tap and select their desired location for fuel delivery. The selected point is visually marked, and location services are enabled for additional UX enhancements.

```

showDialog(
  context: context,
  builder: (BuildContext context) {
    return AlertDialog(
      title: const Text('Confirm Your Order'),
      content: Column(
        mainAxisAlignment: MainAxisAlignment.start,
        crossAxisAlignment: CrossAxisAlignment.start,
        children: [
          Text('🚗 Vehicle: ${_vehicleNumberController.text}'),
          const SizedBox(height: 8),
          Text('⛽ Fuel Type: ${_selectedFuelType.value}'),
          const SizedBox(height: 8),
          Text('📍 Location: (${_selectedLocation.value!.latitude.toStringAsFixed(6)}, ${_selectedLocation.value!.longitude.toStringAsFixed(6)})'),
        ],
      ), // Column
      actions: [
        TextButton(
          onPressed: () => Navigator.of(context).pop(),
          child: const Text('Cancel', style: TextStyle(color: Colors.black)),
        ), // TextButton
        ElevatedButton(
          onPressed: () async {
            Navigator.of(context).pop();
            await _placeOrder();
          },
          style: ElevatedButton.styleFrom(
            backgroundColor: const Color(0xFFE91E63),
            foregroundColor: Colors.white,
            shape: RoundedRectangleBorder(
              borderRadius: BorderRadius.circular(15.0),
            ), // RoundedRectangleBorder
          ),
          child: const Text('Confirm'),
        ), // ElevatedButton
      ],
    ); // AlertDialog
  },
);

```

This confirmation popup dialog ensures users review all order details before proceeding, reducing accidental or incorrect submissions.

```

Future<void> _placeOrder() async {
  if (_selectedFuelType.value == null || _selectedLocation.value == null || _vehicleNumberController.text.isEmpty) {
    ScaffoldMessenger.of(context).showSnackBar(
      const SnackBar(content: Text('Please fill all fields and select a location')),
    );
    return;
  }

  double orderAmount = 50.0;
  bool paymentSuccessful = await Navigator.push(
    context,
    MaterialPageRoute(
      builder: (context) => PaymentScreen(
        fuelType: _selectedFuelType.value!,
        vehicleNumber: _vehicleNumberController.text,
        amount: orderAmount,
      ), // PaymentScreen
    ), // MaterialPageRoute
  );

  if (paymentSuccessful) {
    User? user = _auth.currentUser;
    if (user != null) {
      DocumentReference orderRef = await _firestore.collection('orders').add({
        'userId': user.uid,
        'fuelType': _selectedFuelType.value,
        'vehicleNumber': _vehicleNumberController.text,
        'location': GeoPoint(_selectedLocation.value!.latitude, _selectedLocation.value!.longitude),
        'orderedAt': Timestamp.now(),
        'status': 'Pending',
        'assignedDriverId': null,
      });

      ScaffoldMessenger.of(context).showSnackBar(
        const SnackBar(content: Text('Order placed successfully')),
      );

      Navigator.pushReplacement(
        context,
        MaterialPageRoute(
          builder: (context) => OrderTrackingScreen(orderId: orderRef.id),
        ), // MaterialPageRoute
      );
    }
  }
}

```

Once the payment is successful, this function creates a new order in Firestore, saving user details, selected fuel type, vehicle information, and location. The user is then redirected to the OrderTrackingScreen.

Challenges and Noteworthy Decisions:

- Implementing Google Maps required careful setup of the API keys and proper handling of location updates to avoid UI lag
- The fuel order workflow was designed to be modular – breaking it into map selection, confirmation, and payment – to make future updates easier
- Hardcoded fuel price value 50 was used during development but it is planned to be dynamic according to fuel types in future updates.

4.4 Order Tracking Screen

Purpose:

This screen provides the users with real-time visibility into the status and location of their fuel delivery. By using the Firestore database, Google Maps API, and Google Directions API, the screen dynamically displays both the customer's and the delivery driver's location, screen also draws a route between them, and updates order status live as data changes in Firestore.

Key Features Implemented:

- Real-time location tracking using Firestore document listeners
- Custom route rendering using the Google Directions API and decoded polylines
- Styled google map with customer and driver markers
- Live updates for order status

Main Code Snippets with Explanation:

```
void _listenToOrderUpdates() {  
  FirebaseFirestore.instance.collection('orders').doc(widget.orderId).snapshots().listen((orderSnapshot) {  
    if (orderSnapshot.exists) {  
      GeoPoint customerGeoPoint = orderSnapshot['location'];  
      LatLng newCustomerLocation = LatLng(customerGeoPoint.latitude, customerGeoPoint.longitude);  
  
      LatLng? newDriverLocation;  
      if (orderSnapshot.data()!.containsKey('driverLocation')) {  
        GeoPoint driverGeoPoint = orderSnapshot['driverLocation'];  
        newDriverLocation = LatLng(driverGeoPoint.latitude, driverGeoPoint.longitude);  
      }  
  
      setState(() {  
        _orderStatus = orderSnapshot['status'];  
        _customerLocation = newCustomerLocation;  
        _driverLocation = newDriverLocation;  
      });  
  
      // Fetch and update route when driver location changes  
      if (_driverLocation != null) {  
        _fetchRoute();  
      }  
  
      // Move the map to the updated driver location  
      if (_mapController != null && _driverLocation != null) {  
        _mapController!.animateCamera(  
          CameraUpdate.newLatLng(_driverLocation!),  
        );  
      }  
    }  
  });  
}
```

This function listens to live changes in the order document and updates the customer and driver locations accordingly. It also refreshes the map route and camera position when needed.

```

Future<void> _fetchRoute() async {
  if (_customerLocation == null || _driverLocation == null) return;

  String url =
    "https://maps.googleapis.com/maps/api/directions/json?origin=${_driverLocation!.latitude},${_driverLocation!.longitude}&destination=${_customerLocation!.latitude},${_customerLocation!.longitude}&key=$googleMapsApiKey";

  final response = await http.get(Uri.parse(url));

  if (response.statusCode == 200) {
    Map<String, dynamic> data = json.decode(response.body);

    if (data['routes'].isNotEmpty) {
      String encodedPolyline = data['routes'][0]['overview_polyline']['points'];
      List<LatLng> polylineCoordinates = _decodePolyline(encodedPolyline);

      setState(() {
        _polyLines.clear();
        _polyLines.add(PolyLine(
          polylineId: const PolylineId("route"),
          points: polylineCoordinates,
          color: const Color(0xFFE91E63),
          width: 6,
        )); // Polyline
      });
    } else {
      print("Error fetching route: ${response.statusCode}");
    }
  }
}

```

The Google Directions API is used to fetch the optimal route between the driver and the customer. The encoded polyline is decoded and rendered on the map in real time.

Challenges and Noteworthy Decisions:

- A major challenge was using two separate location points for customer and driver and dynamically drawing polylines, which required decoding and managing async API calls efficiently
- The order status is shown in real time below the map with color-coded styling, this improves user clarity

4.5 Payment Screen

Purpose:

This screen handles secure payment processing for fuel orders using the Stripe API. It allows users to finalize their order by completing a payment through an embedded payment sheet. This integration ensures a seamless and trusted checkout experience, enabling real money transactions with actual Stripe keys (test environment for this project scope).

Key Features Implemented:

- Integration with Stripe's Payment API
- Payment amount conversion and secure request to Stripe backend
- Embedded payment experience using flutter_stripe package
- Display of branded UI elements
- Payment success/failure handling and navigation back to the order screen

Main Code Snippets with Explanation:

```
Future<Map<String, dynamic>> createPaymentIntent(String amount, String currency) async {
  try {
    // Stripe API Secret Key
    const String secretKey = 'sk_test_51QvWmxLd8eQ0UiZd848BSHgU6UU653Is4KznCFh8qxpVQ9Cbn9L5eAvNDi25uaD690QXLP0VckstwBqEwihE0IR00kVM7zi4N';

    // Convert amount to smallest currency unit (e.g., cents for USD)
    int amountInCents = (double.parse(amount) * 100).toInt();

    var response = await http.post(
      Uri.parse('https://api.stripe.com/v1/payment_intents'),
      headers: {
        'Authorization': 'Bearer $secretKey',
        'Content-Type': 'application/x-www-form-urlencoded'
      },
      body: {
        'amount': amountInCents.toString(),
        'currency': currency,
      },
    );

    return jsonDecode(response.body);
  } catch (err) {
    throw Exception(err.toString());
  }
}
```

This function securely contacts Stripe’s server to create a PaymentIntent, which is the foundation of Stripe’s payment flow. The amount is converted into the smallest currency unit.

```
// Step 2: Initialize Payment Sheet
await Stripe.instance.initPaymentSheet(
  paymentSheetParameters: SetupPaymentSheetParameters(
    paymentIntentClientSecret: paymentIntent!['client_secret'],
    merchantDisplayName: 'SwiftFuel',
  ),
);

// Step 3: Display Payment Sheet
await Stripe.instance.presentPaymentSheet();
```

These two steps initialize and show Stripe’s native payment sheet to the user. It’s a fast, secure, and intuitive payment experience that supports cards, wallets, and more.

(Next Page)

```
Future<void> makePayment() async {  
  try {  
    // Step 1: Create Payment Intent on the Server  
    paymentIntent = await createPaymentIntent(widget.amount.toString(), 'gbp');  
  
    // Step 2: Initialize Payment Sheet  
    await Stripe.instance.initPaymentSheet(  
      paymentSheetParameters: SetupPaymentSheetParameters(  
        paymentIntentClientSecret: paymentIntent!['client_secret'],  
        merchantDisplayName: 'SwiftFuel',  
      ),  
    );  
  
    // Step 3: Display Payment Sheet  
    await Stripe.instance.presentPaymentSheet();  
  
    ScaffoldMessenger.of(context).showSnackBar(const SnackBar(content: Text('Payment Successful!')));  
  
    // Navigate to Order Confirmation Page  
    Navigator.pop(context, true); // Return true for successful payment  
  } catch (e) {  
    ScaffoldMessenger.of(context).showSnackBar(SnackBar(content: Text('Payment Failed: $e')));  
  }  
}
```

This function provides the full payment flow logic with error handling and user feedback using snackbars.

Challenges and Noteworthy Decisions:

- Proper Stripe setup was critical; this required configuring the test keys, and Stripe API permissions.
- Stripe's flutter_stripe package handles the UI rendering natively but this required careful initialization steps and permission setups
- While currently using test keys, the screen is structured for production deployment with minimal adjustments needed like switching to live keys for real money transactions.

4.6 Other Screens

While the previous subsections focused on the primary and most technically major screens in the app, a number of other additional screens were also implemented to support the overall user experience and ensure smooth navigation throughout the app.

Home Screen:

This screen acts as the central hub of the entire app, providing users with access to the main features including fuel ordering, profile management, and past orders. It uses intuitive navigation elements and consistent UI styling to maintain the best experience overall.

Delivery Dashboard Screen:

This screen is developed for the delivery drivers/delivery partners, offering them a simple separate interface to view placed fuel orders in order to accept and deal with them accordingly. It ensures separation of roles between customers and service providers.

Past Orders Screen:

This screen retrieves and displays a logged in user's previous completed fuel orders from the Firestore database.

User Profile Screen:

This screen allows users to view their account information and change their passwords.

Chapter 5: Testing and Evaluation

To ensure the fuel delivery mobile application met its functional, performance, and usability requirements, a combination of testing techniques were used throughout the development lifecycle of the project. Testing plays an important role in the software development process as it allows the identification and fixing of bugs, it verifies that the project components interact correctly with each other, and it ensure that the final product behaves as expected. This section of the report discusses different approaches taken to validate the mobile application.

The main goal was to maintain a reliable and stable application while covering all of the required features, where possible, automated integration tests were written to provide repeatable and consistent validation of the main workflows. Manual testing on an emulator was done and used to supplement the automated tests. Additionally, User Acceptance Testing (UAT) was done for real user feedback and to evaluate the final product from an end-user perspective.

5.1 Integration Testing

Integration testing is an important part of any modern application, mainly when dealing with multiple moving parts such as Firebase authentication, Firestore database, and Google maps API. The main goal of integration testing in this project was to verify that individual features worked cohesively when combined in real-world usage scenarios. The test files were written using Flutter's `integration_test` package and were designed to simulate realistic user flows and interactions such as tapping buttons, entering text into fields, navigating between different screens, and verifying some outcomes.

The first integration test implemented was the user login flow. This is an important part of the application and is required to access any personalized features. The test mimicked a real user entering their email and password, tapping the login button, and waiting for authentication to complete. Upon success, the test verified that the user was redirected to the home screen. This test confirmed that the login mechanism works perfectly and that the role-based navigation is functioning as required for customers and delivery partners.

The second test focused on the user registration process. It simulated a new user providing their email address, password, and UK mobile number, and then tapping the register button. Firebase authentication and firestore were both involved in this, with user data being stored securely and in Firestore. After registration, the test validated that the user was redirected back to the login screen and this test ensured that the new user onboarding is seamless and functional.

One of the main functionalities of the application is the ability to place a fuel order from a chosen location. The integration test for this flow first logged in the user, navigated to the fuel ordering screen from the home screen, and provided the order details like fuel type, location, and vehicle number plate. Finally, it confirmed the order through the confirmation popup prompt and verified that the app navigated to the payment screen for the payment. While the actual Stripe payment sheet couldn't be automated, this test ensured that all frontend and backend order preparation logic was triggered correctly and consistently.

Another feature tested was the logout feature, this test ensured that the user could successfully log out using the icon button provided on the home screen. After tapping this button, the test validated that the user was redirected back to the login screen after successfully logging out. This test ensured that the sessions are properly terminated and that the users are securely and successfully signed out of the mobile application.

The last and final integration test focused on the account management feature, specifically updating user passwords inside the application. The test simulated navigating to the user profile tab from the home screen, entering a new password there, and submitting it via the button. After confirming a success message, the test then programmatically reset the password back to the original one to allow the test to be re-run again without any failure. This flow demonstrated not only successful password updating via the Firebase authentication but also the ability to write repeatable tests that preserve user data for future test runs.

All of these five automated integration tests form a comprehensive test suite that covers the app's main features and their workflows. They provide a high level of confidence in the correctness of feature implementations.

5.2 Manual Testing

While integration testing provides repeatable and automated validation for expected interactions, manual testing played an important role in identifying interface level issues, usability concerns, and edge cases not covered by automated flows. Manual testing was conducted continuously throughout the development of this project on the Android studio virtual devices/emulators.

One of the main goals of manual testing was to ensure a smooth and intuitive user experience. This included checking padding, spacing, button sizes, colours, colours contrast, fonts and their readability, and navigation between different screens. Animations, loading states, and feedback messages were also evaluated to ensure responsiveness. On multiple times, manual testing revealed minor UI inconsistencies like pixel overflow which were then fixed during development.

The application integrates with Firebase services, Google Maps API, and Stripe API for payments. During testing, many edge cases were explored. These included manually simulating the driver

moving on the map for the user's order tracking screen, in all cases, the app worked perfectly. The Stripe payment flow was manually tested using Stripe's test debit cards which are provided by them in their documentation, this process involved entering a valid test card number, expiry date, and CVC to simulate a successful payment, ensuring that the app behaved correctly on payment success so that the orders could then be placed successfully.

The Firestore database integration was also manually tested and validated. Fuel orders placed by the user were confirmed to appear in the Firestore database. All the data was verified to be correct and so this step ensured that the data was being correctly stored and retrieved across different sessions of the application.

Manual testing was the backbone of the development process and it was instrumental in making sure that the user experience and the app's functionality were both refined to a high standard. From the early stages of the app, all the screens and user interactions were tested manually as soon as they were implemented. Each of the screens was evaluated for layout, responsiveness, navigation flow, overall look, and alignment with the original storyboard wireframes.

5.3 User Acceptance Testing (UAT)

After the completion of the mobile application, User Acceptance testing was performed to get real users feedback on the application and to document their user experience. This was carried out with three real users to assess the usability, flow, and performance of the SwiftFuel mobile application. The main goal of this was to determine whether the system met the user's needs in realistic scenarios and provided a seamless experience across all the features implemented. The participants were allowed to use the mobile application by connecting a real Android mobile device to the development system (laptop) and using Android studio the app was run on that real device for the participant to use and give feedback. The participants were asked to use the app independently and provide feedback on a piece of paper as they used the app.

Participants:

User	Description	Role in Testing
Hayat	My sister	Evaluated the app as a new customer using the app for the very first time.
Ibrahim	My friend	Tested the app as an existing customer focused on ordering fuel, viewing past order history, and updating user profile.
Qasim	My father's friend	Used the app from the delivery partner's perspective to explore managing orders and order tracking features.

Feedback Summary:

Each of the users were asked to do their roles and carry out tasks like registering, logging in, placing a fuel order, making a test payment with the test card, viewing past order history, updating their password, and logging out. They were also asked to test the smoothness and performance of the app and also the overall UI of the app. The feedback collected from these three users revealed several key insights:

1. User A (Hayat):

Hayat found the app visually appealing and easy to use and understand. She appreciated the simple registration process and liked the use of visual cues like icons and coloured buttons. Overall, she was positive about the entire application.

2. User B (Ibrahim):

Ibrahim commented positively on the layout and responsiveness of the application. He was impressed by the UI and recommended showing a loading indicator during long transitions.

3. User C (Qasim):

Qasim focused on the delivery partner dashboard and order tracking experience. He found the Google Maps integration clear and functional. However, he recommended adding audio alerts when new orders were placed and shown on the deliver partner dashboard screen and this could be considered in a future update of the mobile application.

Overall Evaluation:

The overall feedback was constructive and very positive. All the participants successfully completed the tasks given to them without any assistance or anything. No major or critical issues/bugs were encountered during this testing phase. Hence, the UAT session confirmed that the core workflows of the application are intuitive, and that the system works as expected under normal usage scenarios. Additionally, this testing validated that the app meets its functional requirements.

5.4 Real Device Testing (Samsung Galaxy S24 Plus)

To ensure the robustness and reliability of the mobile application beyond emulators/virtual devices, I conducted final testing on a real Android device which was the Samsung S24 Plus running Android version 14. Testing on a real device provides invaluable insights into the actual performance of the app, user experience, responsiveness, and potential device-specific issues that might not be found in a virtual testing environment.

The process began by connecting the device to my development machine/laptop via USB and enabling developer options and USB debugging on the device. Android studio automatically detected my connected physical device and I was able to run the flutter app on that device. Once

the application was running on the phone, I systematically went through the entire user journey to verify that every screen, feature, and interaction worked as expected. This included testing the registration and login functionality with Firebase, the fuel ordering flow with Google maps integration, location selection, stripe payment flow, and real-time order tracking with live driver updates.

I also paid specific attention to the UI responsiveness and in terms of performance the app was extremely smooth, the animations were fluid, the google maps interactions were responsive and the real-time updates were also working perfectly. There were no app crashes or any memory related issues, and overall the app delivered a production-quality experience.

This real device testing phase was very important in finalizing the app, giving me full confidence that it performs reliably under real-world conditions. It highlighted some device-specific nuances and offered an accurate picture of how end-users will experience the application. Hence, based on this specific testing phase I consider the app stable and production ready for deployment in the future.

5.5 Evaluation Summary

The combination of automated integration tests, manual testing, and user acceptance testing provided strong validation for the core features and user journeys within the fuel delivery mobile application. Each strategy complemented the others – automated tests ensured consistent backend logic and navigation, manual testing verified all user interactions and error handling, and user acceptance testing modelled the end-user experience. Together, they established a strong foundation of quality assurance that enhances the application's readiness for real-world usage.

Chapter 6: Professional Issues

Mobile app development has become increasingly democratized thanks to powerful cross-platform tools like Flutter. However, despite the ideal of platform neutrality, developers still face major challenges when it comes to delivering inclusive, universally accessible applications. This section reflects on a professional issue encountered during the development of my mobile fuel delivery application, the challenge of building for both Android and iOS platforms, and how platform dependency and proprietary barriers shaped the outcome of this project.

From the start, my vision for this project was to build a clean, fast, and user-friendly mobile app that would be available on both Android and iOS platforms. To achieve this, I selected Flutter, a Google UI toolkit designed for cross-platform development. Flutter's promise was appealing which was that a single codebase will compile to both Android and iOS apps with native like performance and visual fidelity. This decision was not just about convenience. It was also rooted in the professional value of accessibility. A mobile app for fuel delivery could be useful to users regardless of the operating system they use. By supporting both platforms, I could make the service more widely available, reduce platform bias, and deliver a consistent experience to a diverse user base.

However, early in the development stages, I came to understand a major limitation of Flutter which was that while it supports cross-platform development, it still requires macOS to build and deploy iOS apps. This is due to Apple's proprietary requirements namely that Xcode which is their integrated development environment must be used to build mobile apps for Ios, and Xcode is only available on macOS. This requirement locked me out of Ios development because I was using a Windows-based development machine/laptop and did not have access to a Mac. The implications of this were frustrating but important from a professional perspective. Despite choosing a technology that theoretically allowed for universal reach, I was restricted by hardware barriers and vendor lock in. Without access to a Mac device, I could not compile or run an Ios version of my app and thus my app then became Android only.

Professional Considerations and Ethical Impact:

At first, all of this looked like a simple technical limitation, but then upon deeper reflection, it raises broader professional and ethical concerns which are as follows:

- Limiting my mobile application to Android only inherently restricts its accessibility. iOS users, who represent a major part of the smartphones market, are excluded from using the app. In professional terms, this undermines the principle of equal access to technology, especially for such an app that offers a practical utility like fuel delivery.
- This experience also highlighted how platform gatekeeping can exclude developers based on their resources. If a student cannot afford a Mac, they are cut off from iOS development, regardless of skill or intent.
- Apple's decision to tie its development ecosystem to macOS is a classic case of vendor lock in. While it is their idea to protect the quality and security of their app store, it also creates a monopoly like control over who gets to publish their apps. For developers like

myself who do not own an Apple hardware, this control becomes a professional barrier rather than just a technical one.

Project Impact and Justification:

Professionally, it was important to weigh these ethical considerations against practical constraints. Could I have acquired a second Mac? Possibly. Could I have used cloud-based macOS build tools or borrowed a device? Maybe. But these options would have added considerable complexity, cost, and delay to the project which was not feasible within the scope and deadline of such project.

Hence, I made the decision to proceed with Android only development while maintaining iOS support in the codebase. This means that the app is written in such a way that with minor updates and changes it can be built for Ios in the future, should access to macOS machine becomes available. This was the most responsible and scalable approach I could take under the given circumstances.

One important professional responsibility that emerged from this situation was the need to communicate platform limitations transparently. In a real-world launch scenario, it would be important to inform users that the Ios version is under development but not yet available. Ethical development isn't just about writing good code but it's about setting realistic expectations for users.

Reflection and Future Actions:

This experience has shaped my understanding of how technical decisions intersect with broader professional issues. Choosing a development technology isn't just about productivity but it affects who can use the app, who can develop it, and how the app fits into the broader software ecosystem.

In future versions of this project and in other future projects, I would consider budgeting time and resources early on for accessing required hardware like renting or borrowing a Mac.

The decision to build SwiftFuel exclusively for Android due to platform restrictions was not ideal, but it was a necessary compromise given the circumstances and constraints. This situation has shown a real and major professional issue in mobile app development and while cross-platform tools like Flutter have made multi-platform development more feasible, proprietary requirements can still limit the reach of well-intentioned projects, specifically for independent or resource constrained developers. As future software developers, it is very important to recognize how our tools and choices affect who gets to participate, who gets served, and who gets left out. This experience has been a valuable lesson in balancing technical goals with professional ethics and has shaped how I will approach platform accessibility in future projects from now onwards.

Chapter 7: Results and Discussion

7.1 Summary and Timeline of Gitlab Diary

The SwiftFuel mobile application has made some major progress and the results highlight the milestones achieved, focusing on core features, user experience, and the integration of backend system.

The project started with extensive research to determine the app's purpose, platform, and functionality. After evaluating potential markets, it was decided to build a fuel delivery app targeting Android devices initially. This decision was later expanded to include iOS as well using Flutter in the future. The development environment was setup which included the Flutter project folders and structure, Firebase, and Android studio IDE. All of this ensured that the app had a solid foundation before the development actually started.

In the early stages of development, a simple “Hello World” screen was developed using Flutter to familiarize with how Flutter works and its functionalities. After this, a low-fidelity storyboard of the app's UI was created using Figma software (as talked about in the previous chapters), this storyboard outlined the app's overall design and interactions.

With the foundational setup complete, Firebase authentication services were implemented to handle user registration and login processes. Then, login and registration screens were built and coded with proper UI and functionality. The app was officially named “SwiftFuel”, reflecting its purpose and mission.

Efforts were also made to improve the app's UI by researching Flutter Flow templates for inspiration. This led to the development of the home/dashboard screen and the integration of the register, login, and home screens was achieved after this. In addition to this, the Firebase Firestore database was setup to store the user data, including mobile numbers entered during the registration process. Snackbar notifications were also implemented across the app to enhance user feedback during the interactions.

Some graphical content was added to the home screen to enhance its visual appeal. A major milestone was the successful integration of Google Maps API into the app, enabling the fuel ordering feature. The fuel ordering screen was also developed, which allowed the users to select their current location or pin a location on the map, choose the fuel type they want, and enter their vehicle number plate to place a fuel order.

Key functionality enhancements followed, including the implementation of a back button on the fuel ordering screen for easy navigation to the home screen. The fuel ordering system was also integrated with Firebase authentication to ensure secure user data handling and proper usage of the fuel ordering system. An order confirmation popup was added which allowed users to review their fuel order details before placing the order.

To validate the app's core functionalities, rigorous testing was also conducted during the development phase for all the implemented features. The registration and login screens were tested to ensure perfect user authentication with Firebase. The Google Maps integration for location selection was also tested to verify the accuracy of the location tracking and map rendering. Functional tests were regularly conducted on the fuel ordering screen to ensure that users could pin locations, select their current location and submit their orders after confirmation without any errors. Navigation between the screens was also tested for user-friendly transitions.

Several bug fixes were done to improve app stability and performance. Additionally, a detailed README file was created to guide users on how to deploy and run the main code of the app. After this, development continued with major feature expansion in early 2025. The user profile screen was successfully implemented, this screen provided users with access to their personal details which were fetched from Firebase. For consistency across the application, the profile screen was designed to match the overall app UI, including a placeholder profile picture for visual context. A major addition at this point was the change password feature which allowed the users to securely update their passwords.

Then, the payment functionality was implemented, for this the payment screen was developed with proper UI and was integrated with Stripe, a trusted payment gateway. Using Stripe's test mode and test cards, full end to end payment testing was conducted which confirmed that the customers could complete payment for fuel orders securely. This marked a critical milestone in the apps preparation for production use. A checklist of remaining developmental tasks was also documented to ensure that the final polish could be applied towards the very end of the project.

After this, Role Based Access Control (RBAC) system was implemented for the authentication part of the application, this ensured that regular customers and delivery partners (drivers) were routed to the appropriate dashboard based on their user roles after logging in. The delivery dashboard screen was implemented to show available placed fuel orders that the drivers could accept. Upon acceptance, these orders were assigned to the driver who accepted it, this not only introduced logical flow but also dynamic content updates. The system's realtime updating capabilities were tested using three virtual devices at the same time which made sure that all data including order status, assignment, and delivery was reflected live across the devices.

Testing for responsiveness was also done, ensuring that the app maintained consistent UI and UX across different screen sizes and resolutions, this was done using multiple emulator models. Then, the delivery dashboard screen was updated so that the drivers could view their assigned orders which they had accepted and mark them as delivered after completion. After this, the past orders screen was developed which gave the customers the option to view their history of completed fuel orders. Another major feature after this was the introduction of an order tracking screen, which

used the Google Maps API to allow customers to view real-time live updates on the location of their assigned driver.

The next development sprint focused on performance tuning and UI consistency. All the screens were optimized to improve app performance and reduce memory consumption. Some quality of life improvements were also made to the screens and a display of different fuel type prices was added to the home screen, which replaced the previous unimplemented search bar. In addition to this, the logout functionality was added to the other screens as well.

Finally, integration testing was completed across the entire mobile application. This also served as a prelude to project finalization. In summary, the later phases of development were focused on completing the remaining development and refining user experience, ensuring that the features were complete. These developments helped transform the app into a production ready app with a full suite of customer facing and delivery side features.

7.2 Challenges during Development

Several challenges were also faced and resolved during the development of the app, such as Firebase rule misconfigurations which in the start prevented the app from properly saving data to the Firestore database. These problems were debugged by reviewing the Firestore security rules, making sure that only authenticated users could interact with the database. In addition to this, integration of the Google Maps API presented some difficulties at first, which included map rendering failures and incorrect location initialisation. These issues were also fixed by properly setting up the API key and verifying project configurations.

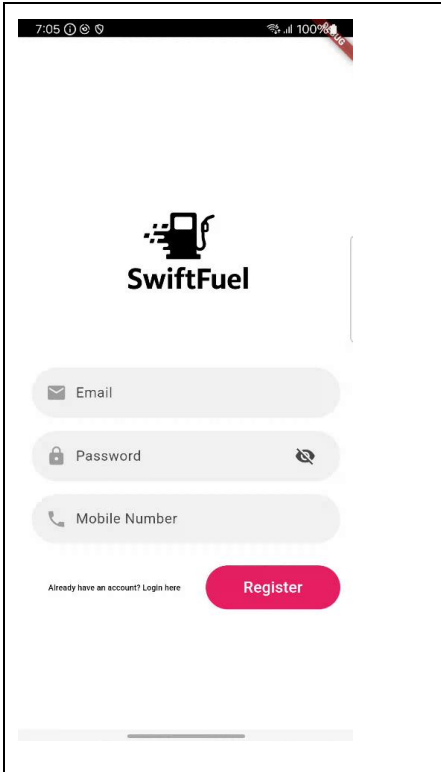
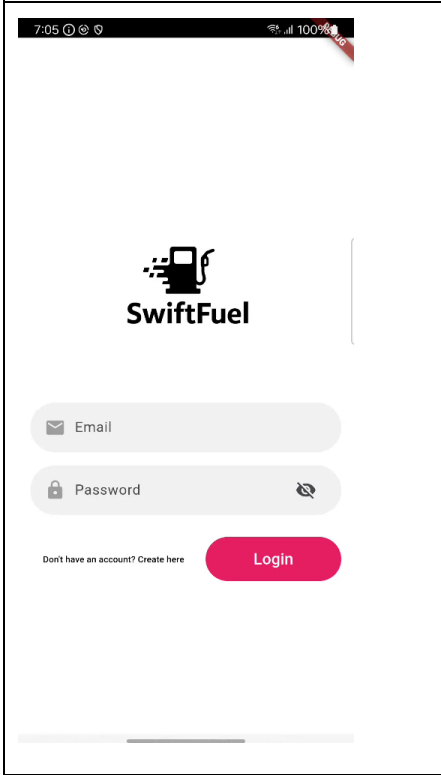
Another major challenge was the Stripe payment integration. While test mode functionality was successful, configuration of the flutter stripe package and managing secure flow between the frontend and backend proved complex. For this, stripe's documentation had to be closely followed and several iterations were required to ensure that the payment sheet loaded and completed the payment safely without breaking the flow of the application.

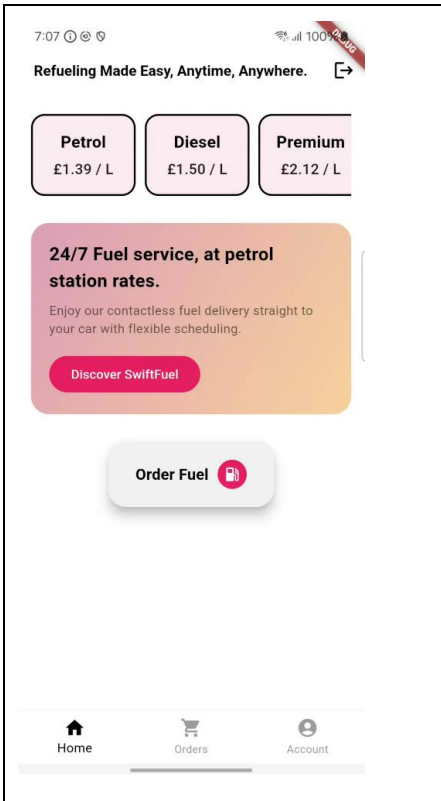
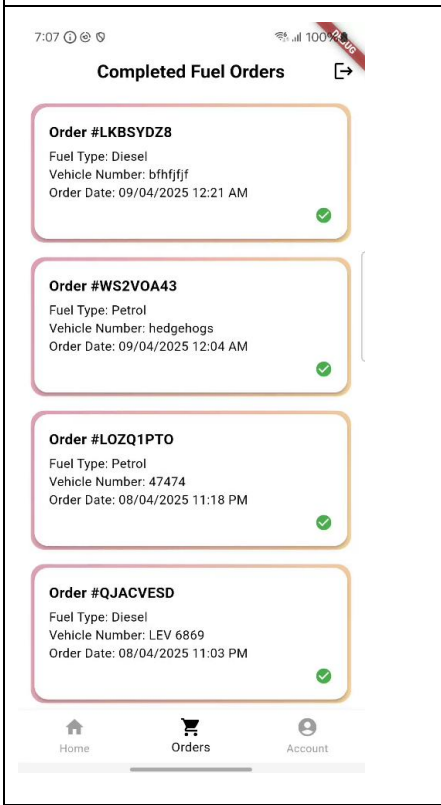
UI responsiveness also proved to be a challenge, especially when the application was tested on a real device with different screen size and resolution than the emulator device. For example, a UI layout that appeared perfect on the emulator ended up showing overflow or broken elements on the real device which was then fixed accordingly. This experience also highlighted the importance of testing on a real device in addition to emulators.

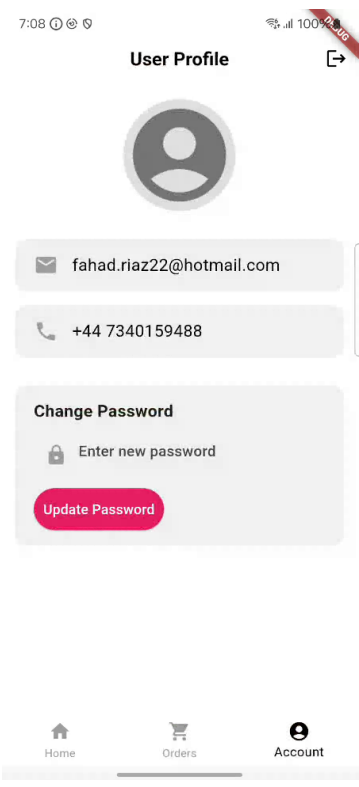
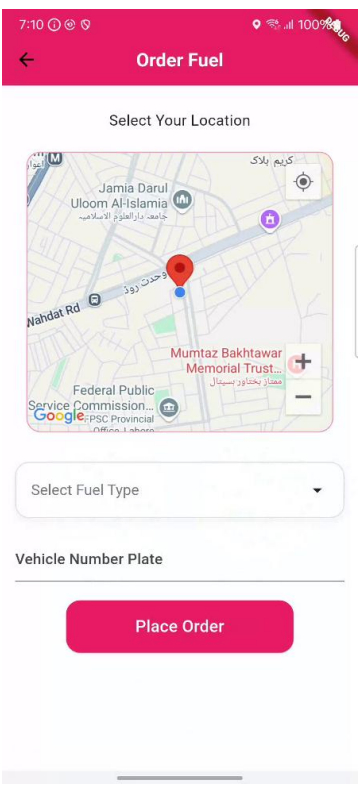
All of these challenges, ultimately provided valuable learning experiences. They reinforced good development practices such as modular code structure, constant testing, use of platform specific documentation, and debugging using both console logs and emulator logs. Solving these challenges contributed directly to improving the quality, stability, and performance of the fuel delivery app.

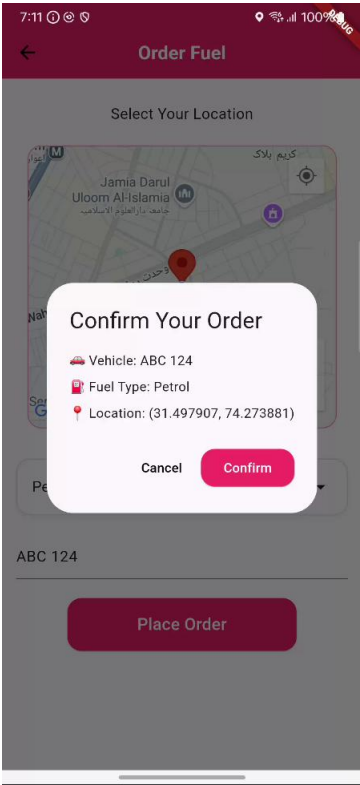
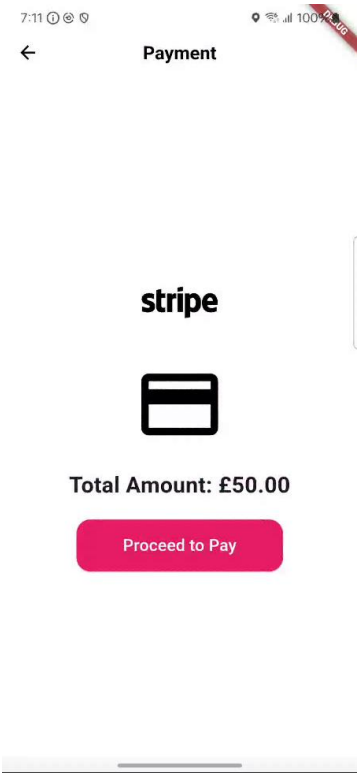
7.3 Screenshots of the Final Product (UI and Features)

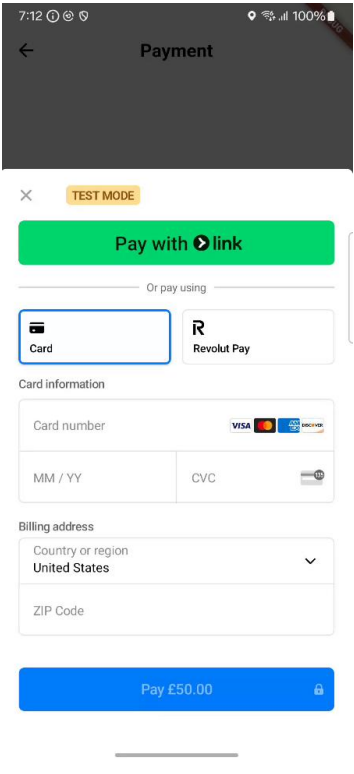
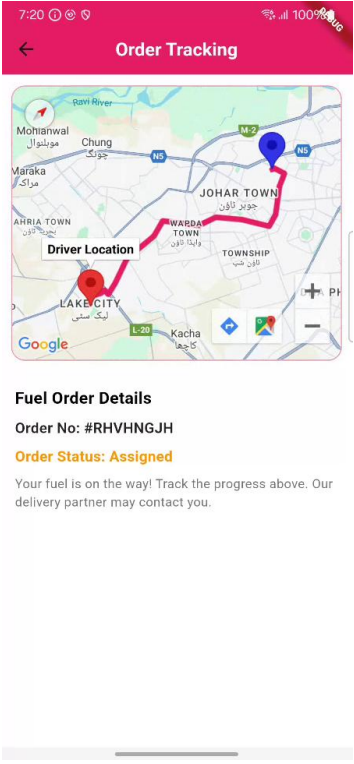
To showcase the final product, the following screenshots showcase the app’s core features and user interface design, each screen reflects the key milestones achieved:

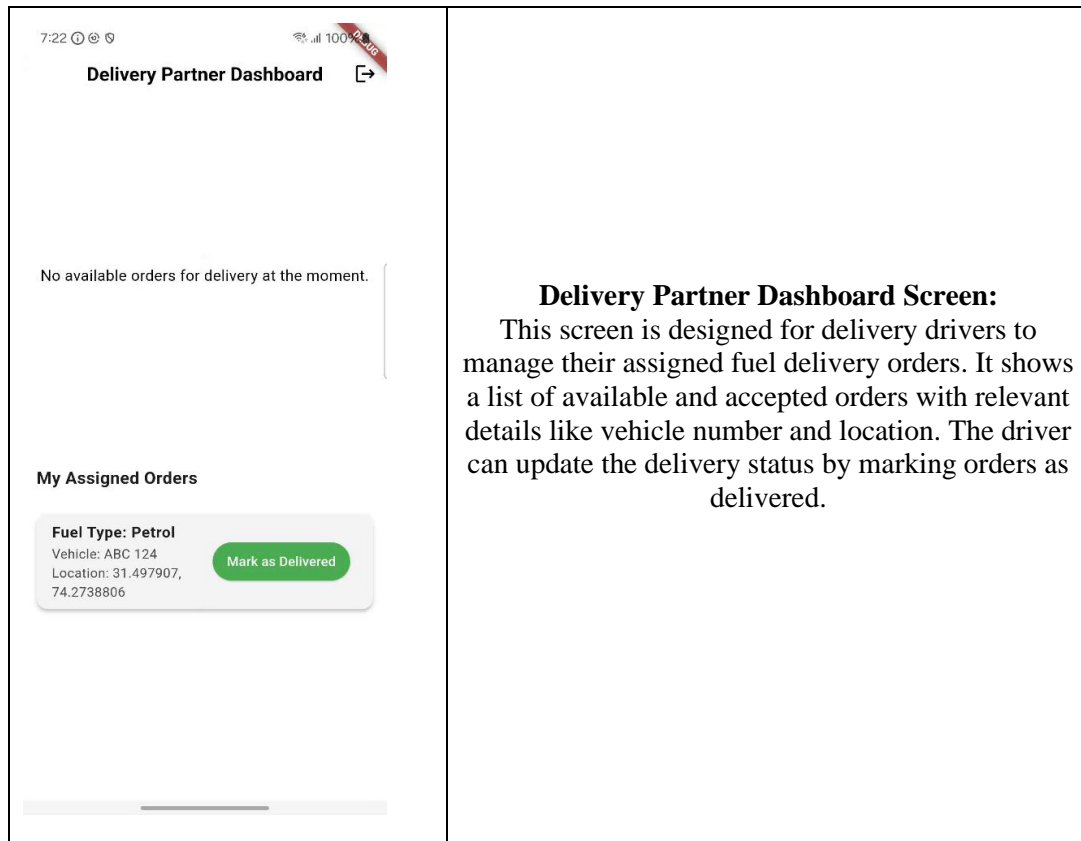
	<p>Register Screen:</p> <p>The registration screen allows new users to create an account by entering their email, password, and mobile number. The UI is clean and minimal with a focus on usability and clarity. It also includes a toggle to view the entered password and a navigation link for users who already have an account.</p>
	<p>Login Screen:</p> <p>The login screen allows existing users to securely access their SwiftFuel account using their email and password. It features a clean layout, password visibility toggle, and a quick link to the registration screen for new users. The app’s logo and branding are prominently displayed to maintain visual identity.</p>

	<p>Home Screen:</p> <p>The home screen welcomes users with fuel price cards for Petrol, Diesel, and Premium fuels. It highlights the 24/7 fuel delivery service with a brief promotional message and a call-to-action button. A prominent “Order Fuel” button allows users to quickly begin the fuel ordering process.</p>
	<p>Past Orders Screen:</p> <p>The past orders screen displays a history of all completed fuel orders for the logged-in user. Each card shows essential details such as order ID, fuel type, vehicle number, and date/time. A green check icon visually confirms that the order was successfully completed.</p>

 The screenshot shows the 'User Profile' screen. At the top, there's a status bar with time 7:08, signal, and battery icons. Below is a header 'User Profile' with a share icon. A circular profile placeholder is centered. Below it, two rows show contact info: an email 'fahad.riaz22@hotmail.com' and a phone number '+44 7340159488'. A 'Change Password' section follows, with a prompt 'Enter new password' and a red 'Update Password' button. At the bottom is a navigation bar with 'Home', 'Orders', and 'Account' icons.	<p>User Profile Screen:</p> <p>The user profile screen displays the logged-in user's email and phone number fetched from the backend. It includes a profile placeholder and a section for changing the user's password. The screen maintains a clean UI while offering essential account management features.</p>
 The screenshot shows the 'Order Fuel' screen. It has a red header with a back arrow and the title 'Order Fuel'. Below is a map section titled 'Select Your Location' showing a map with a red pin and labels like 'Jamia Darul Uloom Al-Islamia' and 'Mumtaz Bakhtawar Memorial Trust'. Below the map is a 'Select Fuel Type' dropdown menu. Underneath is a 'Vehicle Number Plate' input field. At the bottom is a large red 'Place Order' button.	<p>Fuel Ordering Screen:</p> <p>This screen allows users to place a fuel order by selecting their location on the map, choosing the type of fuel, and entering their vehicle number plate. It integrates Google Maps for real-time location selection and ensures a seamless ordering experience. The UI is simple and action-focused for quick fuel requests.</p>

	<p>Fuel Ordering Screen (Confirmation Pop up):</p> <p>Before placing an order, users are shown a confirmation popup to review their vehicle number, selected fuel type, and pinned delivery location. This feature ensures accuracy and allows users to cancel or confirm their order. It adds an extra layer of clarity and control to the ordering process.</p>
	<p>Payment Screen:</p> <p>The payment screen displays the total amount for the fuel order and provides a clear call-to-action to proceed. Integrated with Stripe, this screen allows users to securely complete their payment. Its clean UI ensures a simple and professional checkout experience.</p>

	<p>Payment Screen (Stripe payment sheet):</p> <p>This screen showcases the Stripe payment sheet integrated into the app. Users can enter their card details or choose alternative methods like Link or Revolut Pay. The interface ensures a secure, smooth, and trusted checkout experience with Stripe's hosted solution.</p>
	<p>Order Tracking Screen:</p> <p>This screen provides real-time tracking of the fuel delivery on an interactive map. It displays the driver's live location and the route to the customer. The order status and details are clearly shown, keeping the user informed throughout the delivery.</p>



The screenshots demonstrate the app's design aesthetics and provide evidence of technical implementation such as Firebase integration, Google Maps feature, Stripe integration for payments, and secure user authentication. The designs from the storyboard were instrumental in shaping the current UI, demonstrating how they were translated into actual and functional screens. This highlights the connection between the design planning and practical execution of the project.

Chapter 8: Future of SwiftFuel

The future of this project holds a wide range of opportunities for expansion, feature enhancement, and potential real-world implementation. What started as a final year university project could evolve into a full-fledged, scalable business if approached with a strong post launch strategy. The technical foundation is already in place, now the next step is to refine, extend, and commercialize this project.

8.1 Feature Enhancements and Planned Future Work

While the current version of SwiftFuel is stable and feature rich, several enhancements are envisioned that can significantly improve user experience and overall usability.

- To better support users and enhance the customer experience, a live chat support feature is planned for the home screen. This would allow users to directly interact with customer service representatives or support bots within the app in real time. This will allow the customers to seek any help during the ordering process, resolve any payment issues, or simply ask general questions.
- Currently, the user profile screen allows users to update their password, but future versions of this project will include more comprehensive account management features. This includes the ability to change emails and update mobile numbers, these additions will allow the users to have more control over their personal information.
- Currently, the fuel price in the app is fixed at a static value (50 GBP) which is not reflective of real-world market dynamics. To address this, a dynamic pricing system will be implemented in which prices will vary based on the type of fuel selected and the current market rates for each fuel type.
- While the current order tracking screen shows the driver's live location on the map, there is room to improve this experience as well. One major enhancement would be to display an ETA (estimated time of arrival) to the user. This added feature will further build the user trust and improve the efficiency of the app.

8.2 Real-World Implementation as a Business

The most exciting prospect for this fuel delivery mobile app project is turning the app into a real-world business. Fuel delivery services have started to become popular in different parts of the world as talked about in the literature review of this report.

In the future, SwiftFuel can operate on a commission based model where the users pay for the fuel and a small delivery fee. Alternatively, a subscription model could also be introduced, which will offer unlimited deliveries per month for a specific rate. Additional revenue can come from partnerships with fuel suppliers, ads, or upselling products like engine oil, car cleaning and more.

To scale the app, partnerships with local petrol stations or major fuel suppliers would be needed. These vendors would be responsible for fuel inventory and possibly employing delivery drivers. On the other hand, SwiftFuel could adopt a model similar to UberEats, where individual drivers sign up, get verified, and start delivering fuel orders.

Implementing SwiftFuel as a real world business would also require addressing legal compliance and fuel transportation regulations. This might also involve working with government officials or industry bodies to ensure all practices are lawful and safe for both customer and drivers. Initial rollout of the app could target major cities like London where demand is higher and population density favors app based delivery services. As the business expands, a custom backend might be developed to replace Firebase with more tailored data handling and other capabilities.

To gain traction in a competitive space, strategic marketing would be required. This could include referral incentives, discounts, loyalty programs, and partnerships with car brands or companies. Social media, local ads, and other such campaigns could also help with brand visibility and user retention.

The future of SwiftFuel is full of possibilities. From improving current features to launching a real world business venture, the foundation laid during this project offers enormous potential. With proper planning, continuous innovation, and commitment, this project could become a disruptive force in the industry as very few such solutions exist.

Chapter 9: Critical Reflection on Learning

Throughout the development of SwiftFuel mobile application, I experienced a major transformation in both my technical capabilities and my mindset as a future software engineer. This critical reflection serves as an opportunity to evaluate my journey, highlight key learning experiences, and explore how this project shaped my professional growth and readiness for the real-world tech industry.

One of the most important key takeaways from this project was the transition from learning programming theoretically in previous modules to applying that knowledge practically to build a full and complete functional product. Before this project, my understanding of mobile development was mainly conceptual and now working with Flutter, Firebase, Google Maps API, and Stripe API transformed that perspective entirely. I learned how to design and build user interfaces that are not only visually appealing but also intuitive and responsive. I discovered the importance of considering user journeys and reducing friction in interactions. For example, implementing order confirmation popup and error handling demonstrated the value of good UX practices. Furthermore, my experience with integrating services like Firebase taught me how modern applications rely on external services to accelerate development. I gained hands-on experience working with real-time database, authentication tokens, and secure payment gateways and these are the skills that extend beyond university and are directly applicable to industry standards.

Another important learning experience was adopting the Agile methodology and applying iterative development practices. I broke the project down into manageable sprints with each focused on specific milestones. This approach made the project less overwhelming and gave me a sense of progress with every task completed. Agile taught me the value of building iteratively, making continuous improvements, and being open to change based on feedback from the project supervisor. The gitlab diary that I maintained helped track my iterations and served as a roadmap for the project.

Problem solving was at the core of this project. I faced a few challenges during development and these tested my patience and determination but more importantly these sharpened my analytical skills. Instead of seeing bugs as roadblocks, I began to treat them as opportunities to learn deeply about the system. Each challenge required reading, testing, and sometimes even trial and error.

The process of documenting the project taught me the importance of technical communication. Explaining implementation logic, system design decisions, and evaluation strategies in a structured report enhanced my ability to convey complex ideas clearly and concisely. Writing the interim report and later this final report also improved my research skills. This experience helped me understand that good development is not just about writing good code rather its also about being able to explain it to others and I now feel more confident in both verbal and written communication within a technical context.

Developing this app also gave me the opportunity to reflect on broader professional and ethical issues. As I discussed in the previous chapters, I encountered real barriers that affected platform accessibility. Despite choosing Flutter, the inability to build an iOS version of the app highlighted issues of vendor lock in and digital accessibility. This experience made me more aware of the ethical implications of software development. It also made me consider inclusivity in my design choices and reminded me of the professional responsibility developers have to ensure their tools serve a wide and diverse user base.

Perhaps the most transformative aspect of this project was the confidence it instilled in me. Building an end to end mobile application from scratch and integrating real-world services like Google Maps and online payments was something I would not have imagined myself doing a year ago. This project required me to independently learn new technologies, solve problems with limited guidance, and make decisions based on trade offs. I also learned to rely on documentation, community forums, and Stack Overflow when stuck and the ability to independently research, test, and apply solutions is something that I will take with me into any future job role or personal project.

Throughout this project, I actively sought feedback from my supervisor during the official project meetings and also from friends and their insights helped me see things from perspectives I hadn't considered. Acting on such feedback demonstrated the value of adaptability and openness to critique, qualities I now see as important for continuous improvement in any software development role.

Moving forward, I plan to take the lessons learned from this project and apply them to more advanced projects in the future. I now understand what it takes to move from a concept to a working product, and this knowledge is very empowering. I have gained technical confidence, design intuition, and a deeper appreciation for the broader responsibilities of software developers. While SwiftFuel began as a university project, the learning that came from it extends far beyond academia. It has equipped me with a solid foundation in mobile app development, taught me how to manage a software project from start to finish, and prepared me to enter the tech industry with a proactive, responsible, and reflective mindset.

Chapter 10: Conclusion

The development of the SwiftFuel mobile application has demonstrated how modern mobile technologies can be utilized to improve real-world services like fuel delivery. What started as a university final year project has evolved into a fully functional Android application with robust features, strong backend integration, and an intuitive user interface that streamlines the process of refuelling vehicles. This project successfully met its core objectives, delivering a reliable, secure, and scalable mobile solution.

Throughout this report, each stage of the project has been explored – from literature review to planning, implementation, testing, and future work. The literature review emphasized the growing demand for on-demand services, and validated the relevance of such an app through real-world examples like CAFU. The technological choices (Flutter, Firebase, Google Maps, and Stripe) proved to be well suited for building a robust mobile application. These tools not only accelerated development but also ensured seamless integration between frontend and backend.

The system was designed with a user-centred approach, focusing on usability, accessibility, and performance. Every feature was implemented with both functional value and user experience in mind. Extensive testing reinforced the quality and stability of the application. Integration tests validated key user workflows whereas manual and user acceptance tests highlighted usability and real-world performance. Real device testing confirmed that the app behaves as expected in a live mobile environment.

From a professional perspective, the project also exposed some major industry challenges like platform exclusivity and ethical development practices. The decision to develop for Android only due to lack of access to Apple hardware reflected both practical constraints and broader issues around accessibility.

Looking forward, the project is well positioned for real-world implementation. The existing foundation allows for further future improvements and the potential for scaling this app into a fully operational business model is clear.

In conclusion, this project has not only fulfilled its academic purpose but has also laid the groundwork for something far more impactful which is a scalable, real-world solution to a modern urban problem. The experience has been both technically enriching and professionally formative, and it highlights the potential of innovative thinking combined with thoughtful execution. This project stands as a testament to the power of combining user needs with technology to create meaningful digital solutions.

Bibliography

- [1] Mishra, N., Raghuwanshi, R., Maurya, N.K. and Kumar, I., 2023, August. Efficient Fuel Delivery at Your Fingertips: Developing a Seamless On-Demand Fuel Delivery App with Flutter. In *International Conference on Cognitive Computing and Cyber Physical Systems* (pp. 134-147). Cham: Springer Nature Switzerland.
- [2] Kumar, A., Surya, S., Keerthiga, A. and Shalini, S., 2024. FUEL PULSE: Digital Fuel Logistics Revolutionizing Delivery Management. Available at SSRN 4776337.
- [3] Prabhu, M. and Sarker, Y. . (2023) “On-Demand Fuel Delivery Mobile App for OmanOil”, *Journal of Student Research*. Houston, U.S. Available at: <https://www.jsr.org/index.php/path/article/view/2317> (Accessed: 9October2024).
- [4] Gunthe, S., Sangale, A., Brahmanekar, Y., Kulkarni, P. and Baddi, P., 2023. Fuel Delivery Application. *International Journal for Research in Applied Science and Engineering Technology (IJRASET)*, 11(5), pp.1-7. Available at: <https://doi.org/10.22214/ijraset.2023.52655>.
- [5] On-Demand Fuel Delivery Apps Development Explained Blog, Available at: <https://easternpeak.com/blog/on-demand-fuel-delivery-apps-development/>.
- [6] Esferasoft Solutions. *How to build An On-Demand Fuel Delivery App Like CAFU in 2022*. Retrieved from <https://www.esferasoft.com/blog/cafu-on-demand-fuel-delivery-app-business-model-development-cost/>
- [7] Wamda. (2023). *CAFU on the road to world domination*. Retrieved from <https://www.wamda.com/2023/07/cafu-road-world-domination>
- [8] Business Research Insights (2024) 'On-Demand Fuel Delivery Market REPORT OVERVIEW', *Business Research Insights*, 18 November. Available at: <https://www.businessresearchinsights.com/market-reports/on-demand-fuel-delivery-market-102906>
- [9] Dagne, L. (2019) 'Flutter for cross-platform App and SDK development', *Bachelor's Thesis*, Metropolia University of Applied Sciences, 01 May 2019. Available at: <https://www.theseus.fi/bitstream/handle/10024/172866/Lukas%20Dagne%20Thesis.pdf>
- [10] Hussain, H., Khan, K., Farooqui, F., Arain, Q.A. and Siddiqui, I.F. (2021) 'Comparative Study of Android Native and Flutter App Development', *Proceedings of the 13th International Conference on Internet (ICONI) 2021*, December. Available at: https://www.researchgate.net/profile/Hina-Hussain-2/publication/361208165_Comparative_Study_of_Android_Native_and_Flutter_App_Development/links/635a857c12cbac6a3efff6b6/Comparative-Study-of-Android-Native-and-Flutter-App-Development.pdf
- [11] Fentaw, A.E. (2020) 'Cross platform mobile application development: a comparison study of React Native vs Flutter', *Master's Thesis*, University of Jyväskylä. Available at: <https://jyx.jyu.fi/handle/123456789/70969>
- [12] Khawas, C. and Shah, P. (2018) 'Application of Firebase in Android App Development—A Study', *International Journal of Computer Applications*, 179(46), pp. 49–53. Available at: https://www.researchgate.net/profile/Chunnu-Khawas/publication/325791990_Application_of_Firebase_in_Android_App_Development-A_Study/links/5bab55ed45851574f7e6801e/Application-of-Firebase-in-Android-App-Development-A-Study.pdf

- [13] Kassim M et al. (2019) 'Mobile Application on Garbage Collector Tracker Using Google Maps', *Proceedings of the 2019 IEEE 9th International Conference on System Engineering and Technology (ICSET)*, Shah Alam, Malaysia, 7 October. Available at: <https://ieeexplore.ieee.org/document/8906374>
- [14] Kumar, G. and Bhatia, P.K. (2012) 'Impact of Agile Methodology on Software Development Process', *International Journal of Computer Technology and Electronics Engineering*, 2(4), pp. 46–50. Available at: https://www.researchgate.net/profile/Gaurav-Kumar-175/publication/255707851_Impact_of_Agile_Methodology_on_Software_Development_Process/links/00b49520489442e12d000000/Impact-of-Agile-Methodology-on-Software-Development-Process.pdf
- [15] Stone, D., Jarrett, C., Woodroffe, M., and Minocha, S. (2005) *User Interface Design and Evaluation*. San Francisco: Morgan Kaufmann. Available at: <https://books.google.co.uk/books?id=VvSoyqPBPbMC>

THE END
