National College of Ireland

Computing Year 4

Digital Business Transformation

2022/2023

Conor Hughes

X17454386

X17454386@gmail.com

Fix My Ride

Technical Report

Contents

Executive Summary 2

1.0 Introduction 2

1.1. Background 2

1.2. Aims 3

1.3. Technology 3

1.4. Structure 4

2.0 System 4

2.1. Requirements 4

2.1.1. Functional Requirements 5

2.1.1.1. Use Case Diagram 6

2.1.1.2. Description & Priority 7

2.1.1.3. Use Case 8

2.1.2. Data Requirements 9

2.1.3. User Requirements 10

2.1.4. Environmental Requirements 11

2.1.5. Usability Requirements 12

2.2. Design & Architecture 13

2.3. Implementation 15

2.4. Graphical User Interface (GUI) 17

2.5. Testing 23

2.6. Evaluation 32

3.0 Conclusions 33

4.0 Further Development or Research 34

5.0 References 35

5.1. Project Proposal 35

1.0 Introduction 36

1.1 Objectives 36

1.2 Background 37

1.3 State of the Art 37

1.4 Technical Approach 38

1.5 Technical Details 39

1.6 Project Plan 40

1.7 Testing 41

2.0 Requirement Gathering 42
5.2. Reflective Journals 45

Executive Summary

This report outlines the requirements and specifications for an online mobile application that connects drivers with nearby mechanics in the event of a car breakdown. The purpose of the report is to provide a detailed overview of the project, its objectives, and the functional and non-functional requirements necessary to achieve these objectives.

The report outlines the structure of the app, detailing the system architecture and the hardware and software requirements needed to develop the application. The functional requirements are listed in ranked order, with user registration and authentication, location-based services using Google Maps API, and booking and mechanic request handling being the most crucial requirement. Non-functional requirements such as user interface, performance, reliability, and security are also outlined in the report.

The use cases section provides a use case diagram and description of the various interactions between the app's actors, including drivers, mechanics, and the app itself. The app aims to enhance the overall user experience by providing a seamless booking process, in-app communication, and progress tracking of the service for both the customer and mechanic. The app will allow mechanics to update repair progress in real-time, providing drivers with transparency and clarity throughout the repair process.

The report concludes by summarizing the key points covered in the document and highlighting the project's aims, which are to develop a reliable and user-friendly platform that simplifies the process of finding and booking mechanics during car breakdowns, improving the overall efficiency and convenience of the repair process. The report recommends the use of TypeScript, React Native, PostgreSQL, and Supabase as the primary technologies to achieve these objectives.

Overall, this report provides a overview of the requirements and specifications necessary to develop an online mobile application that connects drivers with nearby mechanics during car breakdowns. By addressing the challenges faced by drivers during car breakdowns, the app aims to transform the car repair industry by providing a digital solution that streamlines the repair process, reduces downtime, and enhances the overall user experience.

1.0 Introduction

1.1. Background

The project was undertaken to address the challenges faced by drivers during car breakdowns, such as finding a reliable mechanic quickly and efficiently. Car breakdowns can be stressful and inconvenient, especially when drivers are far from home or in unfamiliar areas. It important that Drivers can find a trusted mechanic that doesn't offer challenges such as offering a poor customer experience to the drivers and delays to the repair process which can lead to increased costs for the consumer and in turn leads to a bad customer experience overall.

To solve this problem, we decided to develop an online mobile application that connects drivers with nearby mechanics quickly and easily, using location-based services and real-time communication.

The app aims to provide a user-friendly platform that simplifies the process of finding and booking mechanics during car breakdowns, enhancing the overall efficiency and convenience of the repair process.

Our goal is to offer a reliable and efficient platform that benefits both drivers and mechanics, enabling drivers to find nearby mechanics quickly and easily while providing mechanics with more work opportunities and increased visibility. The app aims to transform the car repair industry in Ireland by providing a digital solution that streamlines the repair process, reduces downtime, and enhances the overall user experience.

1.2. Aims

The project aims to develop an online mobile application that connects drivers with nearby mechanics during car breakdowns. The app's primary goal is to offer a convenient and efficient way for drivers to find the nearest available mechanic and get their car repaired quickly, minimizing downtime and inconvenience. By providing real-time location-based services using Google Maps API, the app aims to connect drivers with nearby mechanics who can offer prompt and reliable service.

Additionally, the app aims to enhance the overall user experience by providing a seamless booking process, in-app communication, and progress tracking. The app will allow mechanics to update repair progress in real-time, which will provide drivers with clarity throughout the repair process.

Overall, the project aims to provide a reliable and user-friendly platform that simplifies the process of finding and booking mechanics during car breakdowns, improving the overall efficiency and convenience of the repair process.

1.3. Technology

To develop this Mobile Application which will connects Drivers with their nearest mechanic, I have used the following technology's.

1. Typescript: A superset of JavaScript that provides static typing and other features, making it easier to write and maintain complex code.

2. React Native: A JavaScript-based framework for developing native mobile applications for iOS and Android platforms. It allows developers to build high-performance, cross-platform apps with a single codebase.

3. PostgreSQL: A powerful open-source relational database management system that provides data reliability, robustness, and scalability.

4. Supabase: An open-source alternative to Firebase that provides a backend-as-a-service platform for building web and mobile applications. Supabase offers real-time data synchronization, user authentication, and security features that make it ideal for our application.

We will use Typescript to write the application code, React Native to create the user interface, and PostgreSQL to manage the app's data. We will also use Supabase to provide backend services such as user authentication, real-time data synchronization, and data security.

Overall, these technologies will help us develop a high-performance, scalable, and secure mobile application that meets the functional and non-functional requirements outlined in the document.

1.4. Structure

This document outlines the requirements and specifications for an online mobile application that connects drivers with nearby mechanics during car breakdowns. The document is structured as follows:

1.0 Introduction: This section provides an overview of the document and describes the purpose and scope of the app.

2.0 System: This section describes the system architecture and provides details about the hardware and software requirements.

3.0 Functional Requirements: This section lists the functional requirements for the app, ranked in order of priority. Each requirement is stated in a short, imperative sentence.

4.0 Non-Functional Requirements: This section describes the non-functional requirements for the app, such as user interface, performance, reliability, and security.

5.0 Use Cases: This section provides a use case diagram and textual description of the various interactions between the app's actors (drivers, mechanics, and the app itself) to achieve specific goals.

6.0 Glossary: This section provides a glossary of terms used throughout the document to ensure consistent terminology and clarity.

7.0 Conclusion: This section summarizes the key points covered in the document and provides a final statement.

2.0 System

2.1. Requirements

1. The app must allow drivers and mechanics to create accounts with unique credentials, and the login process must be secure and reliable. Verifiable outcome: A successful login process that ensures only authorized users can access the app's features and data.

2. The app must accurately determine the driver's location during a car breakdown and find the nearest available mechanic. Verifiable outcome: A reliable and accurate system that uses Google Maps API to identify the driver's location and the nearest available mechanic.

3. The app must enable drivers to book the nearest mechanic, and mechanics must be able to accept or decline service requests from drivers. Verifiable outcome: A successful booking process that allows drivers to request service and mechanics to accept or decline service requests.

4. The app must provide an in-app chat feature that enables drivers and mechanics to communicate with each other for additional information and details. Verifiable outcome: A functional in-app chat feature that facilitates communication between drivers and mechanics within the app.

5. The app must allow mechanics to update the repair progress in terms of percentage, keeping drivers informed throughout the process. Verifiable outcome: A system that enables

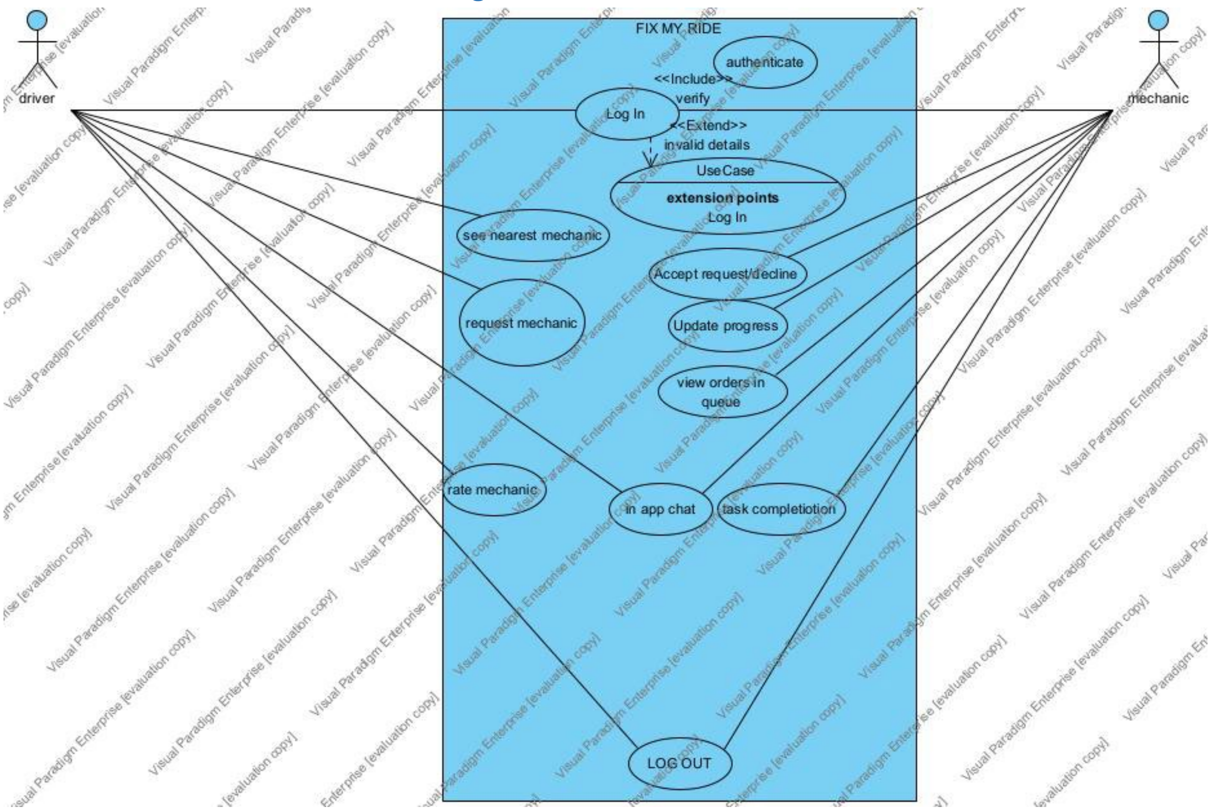
mechanics to report the status of the repair process, and a mechanism for updating drivers about the progress of the repair.

6. The app must implement a completion status and rating system, allowing drivers to rate the mechanic based on the quality of service provided. Verifiable outcome: A functional completion status and rating system that permits drivers to rate the mechanic and provide feedback on the quality of service.

2.1.1. Functional Requirements

1. Enable user registration and authentication. Ensure that drivers and mechanics can create accounts and securely log in to the application, granting access to the app's features.
2. Implement location-based services using Google Maps API. Incorporate Google Maps API to determine the driver's location during a car breakdown and find the nearest available mechanic.
3. Facilitate booking and mechanic request handling. Allow drivers to book the nearest mechanic and enable mechanics to accept or decline service requests from drivers based on their current Availability.
4. Provide an in-app chat feature. Establish a communication channel between drivers and mechanics within the app for exchanging information and details related to the car repair such as Costs, Repair time etc.
5. Allow mechanics to update repair progress. Enable mechanics to report the status of the repair process in terms of percentage, keeping drivers informed throughout the process.
6. Implement completion status and rating system. Permit mechanics to mark repairs as completed, allowing drivers to rate the mechanic based on the quality of service provided.

2.1.1.1. Use Case Diagram



1. Requirement 1: User authentication

The application shall allow drivers to create an account with a unique username and password, and store their personal information such as name, phone number, and vehicle details.

2. Requirement 2: Display nearest mechanics.

The application shall display the nearest mechanics based on the driver's location using Google Maps API.

3. Requirement 3: Service request submission

The application shall enable drivers to request assistance from the nearest mechanic by submitting a service request through the app, including the type of repair needed, location of the breakdown, and a description of the issue.

4. Requirement 4: Service request notification

The application shall notify mechanics of incoming service requests and allow them to accept or decline the request based on their availability and proximity to the location of the breakdown.

5. Requirement 5: In-app chat feature

The application shall provide a chat feature to enable drivers and mechanics to communicate with each other to discuss the details of the service request and clarify any questions or concerns.

6. Requirement 6: Repair progress update

The application shall allow mechanics to update the progress of the repair by specifying the percentage of completion and notify drivers of the progress through the app.

7. Requirement 7: Service completion and rating

The application shall allow mechanics to mark the repair as completed once the repair work has been finished and allow drivers to rate the quality of the service provided by the mechanic on a scale of 1-5.

8. Requirement 8: Data security and privacy

The application shall ensure the security and privacy of user data, including personal information and chat logs, and comply with relevant data protection regulations.

2.1.1.2. Description & Priority

This section of the document will further describe the requirements along with ranking them in order of priority which is High Priority, Medium Priority and low Priority.

1. User registration and authentication (High Priority) Description: Drivers and mechanics should be able to create accounts and securely log in to the application. Authentication is necessary to access various features of the app. Priority: High - This is a crucial requirement for maintaining user security and ensuring that only authorized users can access their respective accounts. This is also a crucial stage for the onboarding of customers, and it can result in the loss of a customer if this process is not efficient.

2. Location-based services using Google Maps API (High Priority) Description: The app must use Google Maps API to determine the driver's location during a car breakdown and locate the nearest mechanic. Priority: High - Accurate location services are essential for the app's core functionality, which is connecting drivers with nearby mechanics.

3. Booking and mechanic request handling (High Priority) Description: Drivers should be able to book the nearest mechanic, while mechanics should have the option to accept or decline service requests. Priority: High - This requirement is vital for the app's main goal of connecting drivers with mechanics and managing their interactions.

4. In-app chat (Medium Priority) Description: An in-app chat feature that enables drivers and mechanics to communicate with each other for additional information and details. Priority: Medium - While not as critical as the core functionality, the in-app chat helps streamline communication between drivers and mechanics and contributes to the overall user experience.

5. Repair progress tracking (Medium Priority) Description: Mechanics should be able to update the repair progress in terms of percentage, keeping drivers informed of the repair status. Priority: Medium - Although not vital to the core functionality, this feature enhances the user experience by providing transparency about the repair process.

6. Completion status and rating system (Low Priority) Description: Mechanics can update the repair status as "completed" after finishing the job, allowing drivers to rate the mechanic based on the quality of service provided. Priority: Low - This feature, while useful for

gathering feedback and improving service quality, is not essential for the core functionality of the app. However, it can be implemented later to enhance the overall user experience.

2.1.1.3. Use Case

Scope

The scope of this use case is to enable drivers to book the nearest available mechanic and for mechanics to accept or decline service requests from drivers.

Description

This use case describes the process of booking a mechanic and handling service requests. It allows drivers to select the nearest available mechanic for their car repair and for mechanics to accept or decline these requests, allowing both parties to communicate through an in-app chat feature.

Use Case Diagram

The use case diagram highlights the actors (driver and mechanic) and the use case (booking and mechanic request handling) as the primary interaction between them.

Flow Description

Precondition

The system is in initialization mode, and the driver has already created an account and logged into the application.

Activation

This use case starts when a driver logs into the application and selects the Request a mechanic option.

Main flow

1. The app identifies the driver's location using Google Maps API and displays a list of available mechanics within a certain radius of the driver's location.
2. The driver selects the nearest mechanic and makes a service request.
3. The mechanic receives the service request and has the option to accept or decline it.
4. If the mechanic accepts the request, they can communicate with the driver through the in-app chat feature to exchange information and details related to the car repair.
5. The mechanic updates the repair progress in terms of percentage, keeping the driver informed throughout the process.
6. Once the repair is complete, the mechanic marks the repair as completed and awaits the driver's rating and feedback.

Alternate flow

A1: If the mechanic declines the request, the driver can select another mechanic from the list and submit a new service request.

1. The system notifies the driver that the selected mechanic has declined the service request.
2. The driver selects another mechanic from the list and submits a new service request.
3. The use case continues at position 4 of the main flow.

Exceptional flow

E1: If the driver and the mechanic are unable to communicate effectively through the in-app chat feature, they may need to use alternative means of communication, such as phone or email, to discuss the details of the service request.

1. The driver and the mechanic are unable to communicate effectively through the in-app chat feature.
2. The driver contacts the mechanic using an alternative means of communication, such as phone or email.
3. The use case continues at position 6 of the main flow.

Termination

The system presents the driver with the option to request another service or log out of the application. The use case ends.

Post condition

The system goes into a wait state, ready to receive another service request from the driver or for the driver to log out of the application.

- The app must enable drivers to cancel or reschedule service requests within a certain time limit.
- The app must notify mechanics of new service requests in real-time.
- The app must notify drivers of the mechanic's estimated time of arrival.
- The app must enable mechanics to update their service availability status.

2.1.2. Data Requirements

Data requirements describe the data that the system needs to perform its functions. This section outlines the data requirements for the online mobile application that connects drivers with nearby mechanics during car breakdowns.

The data requirements for the app are as follows:

1. User data: The app needs to store user data such as name, email address, phone number, password, and location. This data is necessary to identify and authenticate users and to provide location-based services.

2. Mechanic data: The app needs to store mechanic data such as name, contact information, and availability status. This data is necessary to display a list of available mechanics and to enable drivers to make service requests.

3. Service request data: The app needs to store service request data such as the driver's name, contact information, location, and repair details. This data is necessary to facilitate communication between the driver and mechanic and to track repair progress.

4. Repair progress data: The app needs to store repair progress data such as the percentage of repair progress and estimated completion time. This data is necessary to keep the driver informed throughout the repair process.

5. Ratings and feedback data: The app needs to store ratings and feedback data provided by drivers for mechanics. This data is necessary to evaluate the quality of service offered by mechanics and to improve the app's overall user experience.

6. System log data: The app needs to store system log data such as user actions and system events. This data is necessary for system maintenance, troubleshooting, and performance optimization of the application.

To ensure data security and integrity, the app will use PostgreSQL as the primary database management system and Supabase for backend services such as user authentication, real-time data synchronization, and data security. The app will also comply with data protection regulations and best practices to protect user privacy and maintain data confidentiality.

2.1.3. User Requirements

User requirements describe the needs and expectations of users for the system to be considered successful. This section outlines the user requirements for the online mobile application that connects drivers to the nearest mechanic in the event of a breakdown.

The user requirements for the app are as follows:

1. Ease of use: The app should be easy to use, with a user-friendly interface that enables drivers to book a mechanic quickly and easily.
2. Availability: The app should be available 24/7, allowing drivers to access it at any time, from anywhere.

3. Reliability: The app should be reliable, with a robust and stable system that ensures smooth and uninterrupted service.
4. Prompt response: The app should provide a prompt response to service requests, enabling drivers to quickly connect with nearby mechanics.
5. Real-time updates: The app should provide real-time updates on the progress of the repair, allowing drivers to stay informed throughout the repair process.
6. In-app communication: The app should provide an in-app chat feature that enables drivers and mechanics to communicate with each other directly.
7. Transparency: The app should provide transparency in terms of repair progress, estimated completion time, and pricing.
8. Security: The app should provide a secure platform that ensures user data confidentiality and privacy.
9. Personalization: The app should provide personalized recommendations based on the driver's location and previous service history.
10. Ratings and feedback: The app should enable drivers to rate mechanics based on the quality of service offered and provide feedback to help improve the app's overall user experience.

Meeting these user requirements will help ensure that the app provides a seamless and satisfactory user experience, enhancing the overall efficiency and convenience of the car repair process.

2.1.4. Environmental Requirements

Environmental requirements describe the physical and technical environment in which the system will operate. This section outlines the environmental requirements for the online mobile application that connects drivers with nearby mechanics during car breakdowns.

The environmental requirements for the app are as follows:

1. Mobile devices: The app should be compatible with a wide range of mobile devices, including smartphones and tablets.
2. Operating systems: The app should be compatible with major operating systems, including Android and iOS.
3. Internet connectivity: The app requires a stable and reliable internet connection to function properly.
4. GPS services: The app requires GPS services to provide location-based services and real-time updates.
5. Hardware specifications: The app should run smoothly on devices with minimum hardware specifications, including at least 1GB RAM and a dual-core processor.
6. Software dependencies: The app requires specific software dependencies, including TypeScript, React Native, PostgreSQL, and Supabase.

7. Security measures: The app should comply with industry-standard security measures, such as data encryption and secure authentication protocols.
8. Maintenance and support: The app should be designed for easy maintenance and updates, with a support team available to address any issues or concerns.

Meeting these environmental requirements will ensure that the app operates reliably and efficiently in a wide range of environments and devices, providing a seamless user experience for drivers and mechanics alike.

2.1.5. Usability Requirements

Usability requirements describe how the app should be designed to meet the needs of its intended users. This section outlines the usability requirements for the online mobile application that connects drivers with nearby mechanics during car breakdowns.

The usability requirements for the app are as follows:

1. User-friendly interface: The app should have a user-friendly interface that is easy to navigate, with clear and concise instructions.
2. Simple registration process: The registration process should be simple and straightforward, requiring minimal information and steps.
3. Easy booking process: The booking process should be easy and intuitive, with clear instructions and options to select the nearest available mechanic.
4. In-app chat feature: The app should provide an in-app chat feature that enables drivers and mechanics to communicate with each other directly.
5. Real-time updates: The app should provide real-time updates on the progress of the repair, allowing drivers to stay informed throughout the repair process.
6. Transparency: The app should provide transparency in terms of repair progress and estimated completion time.
7. Personalization: The app should provide personalized recommendations based on the driver's location and previous service history.
8. Accessibility: The app should be accessible to users with disabilities, with features such as voice commands and text-to-speech functionality.
9. Error handling: The app should handle errors gracefully, providing clear error messages and options to correct them.
10. Help and support: The app should provide easy access to help and support options, such as FAQs, tutorials, and customer service.

Meeting these usability requirements will ensure that the app provides a seamless and satisfactory user experience, enhancing the overall efficiency and convenience of the car repair process.

2.2. Design & Architecture

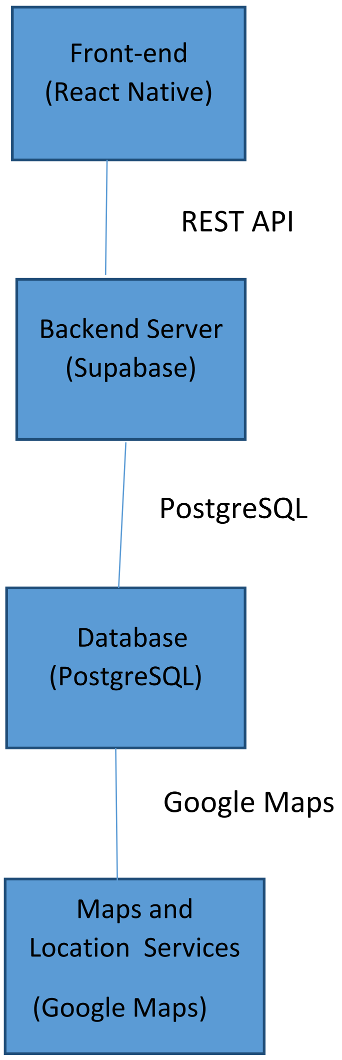
1. Front-end: The front-end of the application will be built using React Native, which provides a robust and flexible framework for building native mobile applications for iOS and Android platforms. The use of TypeScript will help to write clean and maintainable code and provide static typing, making it easier to identify and fix errors.

2. Backend Server: The backend server will be built using Supabase, an open-source backend-as-a-service platform that provides user authentication, real-time data synchronization, and data security features. This will simplify the development of the server-side components and allow the developers to focus on the business logic of the application.

3. Database: The database will be managed using PostgreSQL, an open-source relational database management system that provides data reliability, robustness, and scalability. This will ensure that the application can handle large amounts of data and scale as the user base grows.

4. APIs: The application will rely on several APIs, including the Google Maps API for location-based services, and the Supabase API for real-time data synchronization and user authentication.

The architecture diagram for the online mobile application can be visualized as follows:



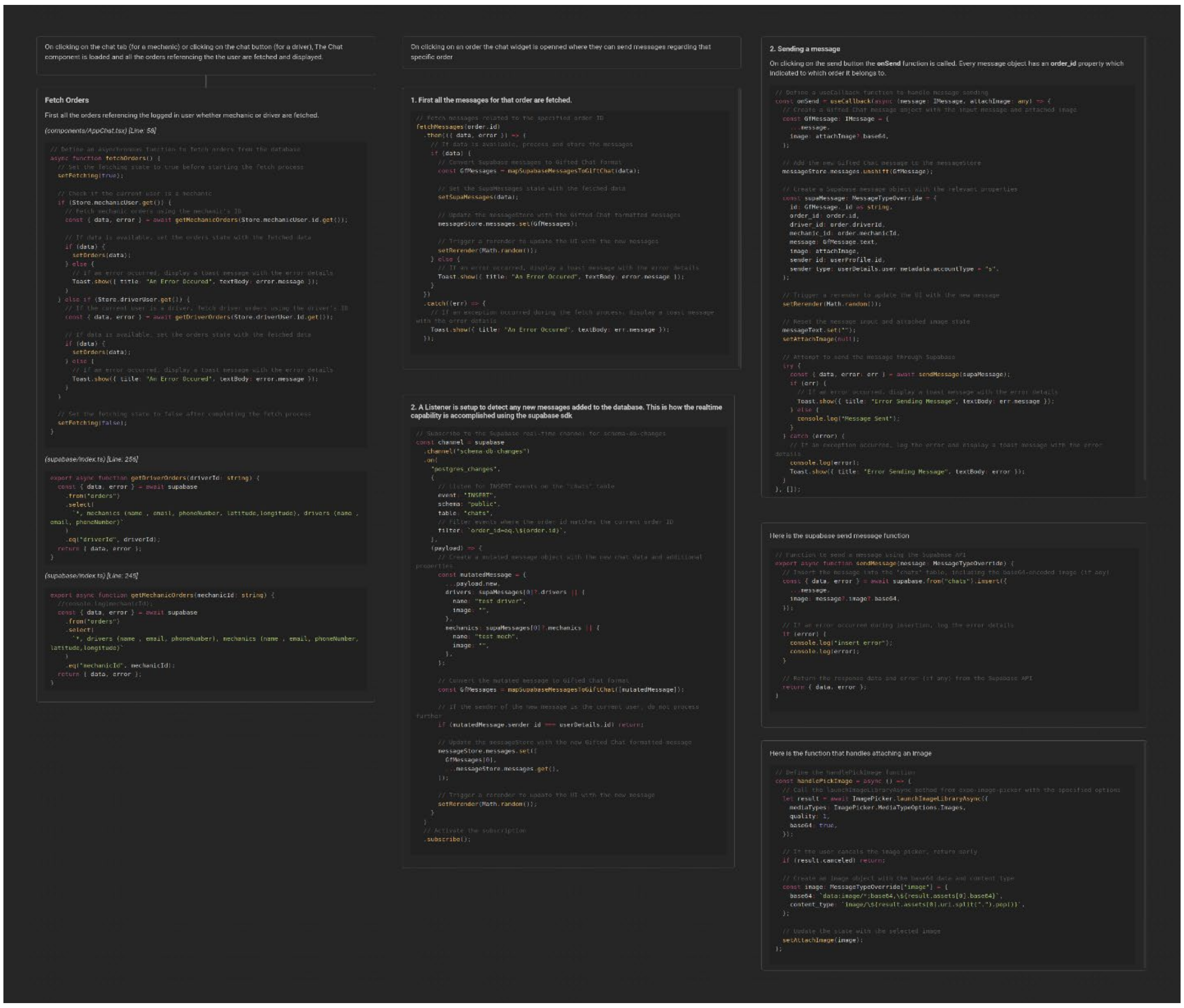
The main algorithms used in the project include:

1. User Authentication: The Supabase API provides a range of authentication methods, including email/password, social login, and OAuth. The algorithms used for user authentication will depend on the chosen authentication method.

2. Real-time Data Synchronization: Supabase provides real-time data synchronization between the front-end and backend, using websockets or polling. The algorithms used for real-time data synchronization will depend on the chosen method.

3. Location-based Services: The Google Maps API provides a range of algorithms for location tracking, distance calculation, and route planning. These algorithms will be used to determine the driver's location and display nearby mechanics on the map.

Overall, the use of TypeScript, React Native, Supabase, and PostgreSQL will help to build a highly performant, scalable, and secure mobile application that meets the functional and non-functional requirements outlined in the document.



On clicking on the chat tab (for a mechanic) or clicking on the chat button (for a driver), The Chat component is loaded and all the orders referencing the the user are fetched and displayed.

Fetch Orders

First all the orders referencing the logged in user whether mechanic or driver are fetched.

(components/AppChat.tsx) [Line: 58]

```
// Define an asynchronous function to fetch orders from the database
async function fetchOrders() {
  // Set the fetching state to true before starting the fetch process
  setFetching(true);

  // Check if the current user is a mechanic
  if (Store.mechanicUser.get()) {
    // Fetch mechanic orders using the mechanic's ID
    const { data, error } = await getMechanicOrders(Store.mechanicUser.id.get());

    // If data is available, set the orders state with the fetched data
    if (data) {
      setOrders(data);
    } else {
      // If an error occurred, display a toast message with the error details
      Toast.show({ title: "An Error Occured", textBody: error.message });
    }
  } else if (Store.driverUser.get()) {
    // If the current user is a driver, fetch driver orders using the driver's ID
    const { data, error } = await getDriverOrders(Store.driverUser.id.get());

    // If data is available, set the orders state with the fetched data
    if (data) {
      setOrders(data);
    } else {
      // If an error occurred, display a toast message with the error details
      Toast.show({ title: "An Error Occured", textBody: error.message });
    }
  }

  // Set the fetching state to false after completing the fetch process
  setFetching(false);
}
```

(supabase/index.ts) [Line: 256]

```
export async function getDriverOrders(driverId: string) {
  const { data, error } = await supabase
    .from("orders")
    .select(
      `*, mechanics (name , email, phoneNumber, latitude,longitude), drivers (name ,
email, phoneNumber)`
    )
    .eq("driverId", driverId);
  return { data, error };
}
```

(supabase/index.ts) [Line: 245]

```
export async function getMechanicOrders(mechanicId: string) {
  //console.log(mechanicId);
  const { data, error } = await supabase
    .from("orders")
    .select(
      `*, drivers (name , email, phoneNumber), mechanics (name , email, phoneNumber,
latitude,longitude)`
    )
    .eq("mechanicId", mechanicId);
  return { data, error };
}
```

On clicking on an order the chat widget is openned where they can send messages regarding that specific order

1. First all the messages for that order are fetched.

```
// Fetch messages related to the specified order ID
fetchMessages(order.id)
  .then(({ data, error }) => {
    // If data is available, process and store the messages
    if (data) {
      // Convert Supabase messages to Gifted Chat format
      const GfMessages = mapSupabaseMessagesToGiftChat(data);

      // Set the SupaMessages state with the fetched data
      setSupaMessages(data);

      // Update the messageStore with the Gifted Chat formatted messages
      messageStore.messages.set(GfMessages);

      // Trigger a rerender to update the UI with the new messages
      setRerender(Math.random());
    } else {
      // If an error occurred, display a toast message with the error details
      Toast.show({ title: "An Error Occured", textBody: error.message });
    }
  })
  .catch((err) => {
    // If an exception occurred during the fetch process, display a toast message
with the error details
    Toast.show({ title: "An Error Occured", textBody: err.message });
  });
```

2. A Listener is setup to detect any new messages added to the database. This is how the realtime capability is accomplished using the supabase sdk

```
// Subscribe to the Supabase real-time channel for schema-db-changes
const channel = supabase
  .channel("schema-db-changes")
  .on(
    "postgres_changes",
    {
      // Listen for INSERT events on the "chats" table
      event: "INSERT",
      schema: "public",
      table: "chats",
      // Filter events where the order id matches the current order ID
      filter: `order_id=eq.\${order.id}`,
    },
    (payload) => {
      // Create a mutated message object with the new chat data and additional
properties
      const mutatedMessage = {
        ...payload.new,
        drivers: supaMessages[0]?.drivers || {
          name: "test driver",
          image: "",
        },
        mechanics: supaMessages[0]?.mechanics || {
          name: "test mech",
          image: "",
        },
      };

      // Convert the mutated message to Gifted Chat format
      const GfMessages = mapSupabaseMessagesToGiftChat([mutatedMessage]);

      // If the sender of the new message is the current user, do not process
further
      if (mutatedMessage.sender_id === userDetails.id) return;

      // Update the messageStore with the new Gifted Chat formatted message
      messageStore.messages.set([
        GfMessages[0],
        ...messageStore.messages.get(),
      ]);

      // Trigger a rerender to update the UI with the new message
      setRerender(Math.random());
    }
  )
  // Activate the subscription
  .subscribe();
```

2. Sending a message

On clicking on the send button the **onSend** function is called. Every message object has an **order_id** property which indicated to which order it belongs to.

```
// Define a useCallback function to handle message sending
const onSend = useCallback(async (message: IMessage, attachImage: any) => {
  // Create a Gifted Chat message object with the input message and attached image
  const GfMessage: IMessage = {
    ...message,
    image: attachImage?.base64,
  };

  // Add the new Gifted Chat message to the messageStore
  messageStore.messages.unshift(GfMessage);

  // Create a Supabase message object with the relevant properties
  const supaMessage: MessageTypeOverride = {
    id: GfMessage._id as string,
    order_id: order.id,
    driver_id: order.driverId,
    mechanic_id: order.mechanicId,
    message: GfMessage.text,
    image: attachImage,
    sender_id: userProfile.id,
    sender_type: userDetails.user_metadata.accountType + "s",
  };

  // Trigger a rerender to update the UI with the new message
  setRerender(Math.random());

  // Reset the message input and attached image state
  messageText.set("");
  setAttachImage(null);

  // Attempt to send the message through Supabase
  try {
    const { data, error: err } = await sendMessage(supaMessage);
    if (err) {
      // If an error occurred, display a toast message with the error details
      Toast.show({ title: "Error Sending Message", textBody: err.message });
    } else {
      console.log("Message Sent");
    }
  } catch (error) {
    // If an exception occurred, log the error and display a toast message with the error
details
    console.log(error);
    Toast.show({ title: "Error Sending Message", textBody: error });
  }
}, []);
```

Here is the supabase send message function

```
// Function to send a message using the Supabase API
export async function sendMessage(message: MessageTypeOverride) {
  // Insert the message into the "chats" table, including the base64-encoded image (if any)
  const { data, error } = await supabase.from("chats").insert({
    ...message,
    image: message?.image?.base64,
  });

  // If an error occurred during insertion, log the error details
  if (error) {
    console.log("insert error");
    console.log(error);
  }

  // Return the response data and error (if any) from the Supabase API
  return { data, error };
}
```

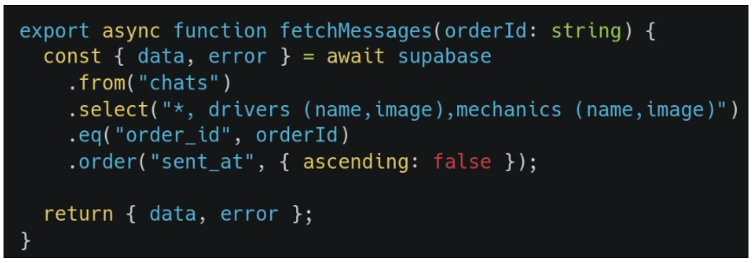
Here is the function that handles attaching an image

```
// Define the handlePickImage function
const handlePickImage = async () => {
  // Call the launchImageLibraryAsync method from expo-image-picker with the specified options
  let result = await ImagePicker.launchImageLibraryAsync({
    mediaTypes: ImagePicker.MediaTypeOptions.Images,
    quality: 1,
    base64: true,
  });

  // If the user cancels the image picker, return early
  if (result.canceled) return;

  // Create an image object with the base64 data and content type
  const image: MessageTypeOverride["image"] = {
    base64: `data:image/*;base64,\${result.assets[0].base64}`,
    content_type: `image/\${result.assets[0].uri.split(".").pop()}`,
  };

  // Update the state with the selected image
  setAttachImage(image);
};
```

Above I have described the algorithms that are involved in the chat functionality. Excuse the Small text but it was the only way to make it presentable. The zoom function will better help understand it.

2.3. Implementation

1.



```
export async function fetchMessages(orderId: string) {
  const { data, error } = await supabase
    .from("chats")
    .select("*, drivers (name,image),mechanics (name,image)")
    .eq("order_id", orderId)
    .order("sent_at", { ascending: false });

  return { data, error };
}
```

This is an async function in JavaScript that uses the Supabase library to fetch messages from a database. The function takes a parameter `orderId` which is used to filter the messages that are returned based on a specific order.

The `await` keyword is used to indicate that the function is asynchronous and that it will wait for the Supabase API to return a response before continuing execution.

The `supabase` object is created by importing the Supabase library and initializing it with your Supabase project URL and public API key. The `from` method is then called on the `supabase` object to specify which table in the database to fetch data from. In this case, the `chats` table is being queried.

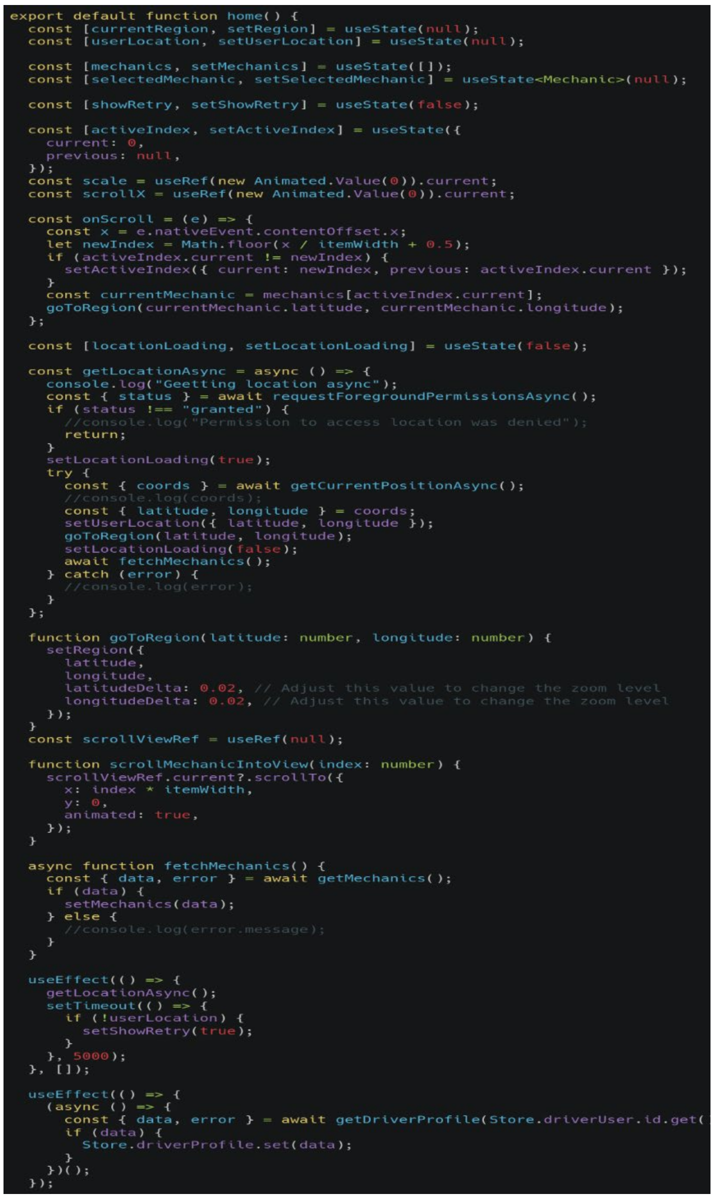
The `select` method is used to specify which columns to retrieve from the `chats` table. In this case, all columns are being returned along with the `name` and `image` columns from the `drivers` and `mechanics` tables. This is achieved using the `drivers(name,image)` and `mechanics(name,image)` functions respectively.

The `eq` method is used to filter the messages based on the `order_id` column, which must match the `orderId` parameter.

Finally, the `order` method is used to sort the messages by the `sent_at` column in descending order, meaning that the most recent messages will be returned first.

The function returns an object containing the `data` and `error` properties. The `data` property contains an array of messages that match the specified filters and sorting, while the `error` property contains any errors that occurred during the fetch operation.

2.



```
export default function home() {
  const [currentRegion, setRegion] = useState(null);
  const [userLocation, setUserLocation] = useState(null);

  const [mechanics, setMechanics] = useState([]);
  const [selectedMechanic, setSelectedMechanic] = useState<Mechanic>(null);

  const [showRetry, setShowRetry] = useState(false);

  const [activeIndex, setActiveIndex] = useState({
    current: 0,
    previous: null,
  });
  const scale = useRef(new Animated.Value(0)).current;
  const scrollX = useRef(new Animated.Value(0)).current;

  const onScroll = (e) => {
    const x = e.nativeEvent.contentOffset.x;
    let newIndex = Math.floor(x / itemWidth + 0.5);
    if (activeIndex.current != newIndex) {
      setActiveIndex({ current: newIndex, previous: activeIndex.current });
    }
    const currentMechanic = mechanics[activeIndex.current];
    goToRegion(currentMechanic.latitude, currentMechanic.longitude);
  };

  const [locationLoading, setLocationLoading] = useState(false);

  const getLocationAsync = async () => {
    console.log("Geetting location async");
    const { status } = await requestForegroundPermissionsAsync();
    if (status !== "granted") {
      //console.log("Permission to access location was denied");
      return;
    }
    setLocationLoading(true);
    try {
      const { coords } = await getCurrentPositionAsync();
      //console.log(coords);
      const { latitude, longitude } = coords;
      setUserLocation({ latitude, longitude });
      goToRegion(latitude, longitude);
      setLocationLoading(false);
      await fetchMechanics();
    } catch (error) {
      //console.log(error);
    }
  };

  function goToRegion(latitude: number, longitude: number) {
    setRegion({
      latitude,
      longitude,
      latitudeDelta: 0.02, // Adjust this value to change the zoom level
      longitudeDelta: 0.02, // Adjust this value to change the zoom level
    });
  }
  const scrollViewRef = useRef(null);

  function scrollMechanicIntoView(index: number) {
    scrollViewRef.current?.scrollTo({
      x: index * itemWidth,
      y: 0,
      animated: true,
    });
  }

  async function fetchMechanics() {
    const { data, error } = await getMechanics();
    if (data) {
      setMechanics(data);
    } else {
      //console.log(error.message);
    }
  }

  useEffect(() => {
    getLocationAsync();
    setTimeout(() => {
      if (!userLocation) {
        setShowRetry(true);
      }
    }, 5000);
  }, []);

  useEffect(() => {
    (async () => {
      const { data, error } = await getDriverProfile(Store.driverUser.id.get(
      if (data) {
        Store.driverProfile.set(data);
      }
    })();
  });
```

The code is for a page that allows drivers to locate nearby mechanics based on their location. It has a MapView component and a carousel of Mechanic cards. Each mechanic card corresponds to a mechanic nearby the driver and includes some basic information about the mechanic, such as their name, rating, and distance from the driver.

Here's how the code achieves its purposes:

1. State Management: The code uses the `useState` hook to define several state variables. These include the current region (i.e., the region displayed on the MapView), the user's current location, the list of mechanics, the currently selected mechanic, and some additional state variables for managing the carousel and location loading status.

2. Animation and Interaction: The code uses the `useRef` and `useEffect` hooks to define a `scale` variable and a `scrollX` variable, which are used to animate the carousel of Mechanic cards. Additionally, the `onScroll` function is called whenever the user scrolls through the carousel, and it updates the `activeIndex` state variable to reflect the currently selected Mechanic card.

3. MapView and Markers: The code uses the `MapView` component from the `react-native-maps` library to display a map with markers for the user's current location and nearby mechanics. The `goToRegion` function updates the `currentRegion` state variable to center the map on a specific location, while the `fetchMechanics` function retrieves a list of mechanics from the server and updates the `mechanics` state variable. Finally, the `renderMechanicMarkers` function maps over the `mechanics` array to render a marker for each mechanic on the map.

4. Mechanic Cards Carousel: The code uses the `Animated.FlatList` component to display a horizontal carousel of Mechanic cards, with each card containing basic information about a nearby mechanic. The `scrollMechanicIntoView` function is used to animate the carousel to the selected Mechanic card, while the `renderMechanicCard` function is used to render each Mechanic card based on the data in the `mechanics` array.

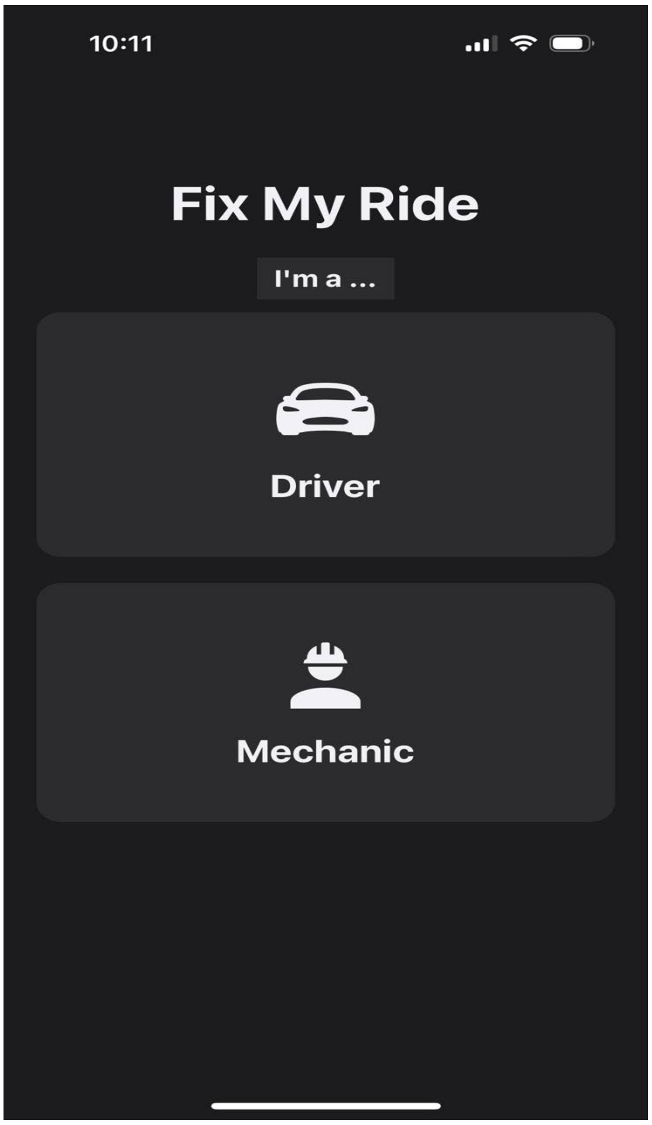
5. User Location: The code uses the `getLocationAsync` function to retrieve the user's current location and update the `userLocation` state variable. If the user does not grant permission to access their location, the function simply returns without doing anything. Additionally, the `setShowRetry` state variable is used to display a "retry" button if the user's location cannot be retrieved within 5 seconds.

2.4. Graphical User Interface (GUI)

1. **Start Page**

The start page is the initial screen that appears when the user opens the application. The page includes two options: "I am a driver" and "I am a mechanic". Users can select the option that applies to them by tapping on the button.

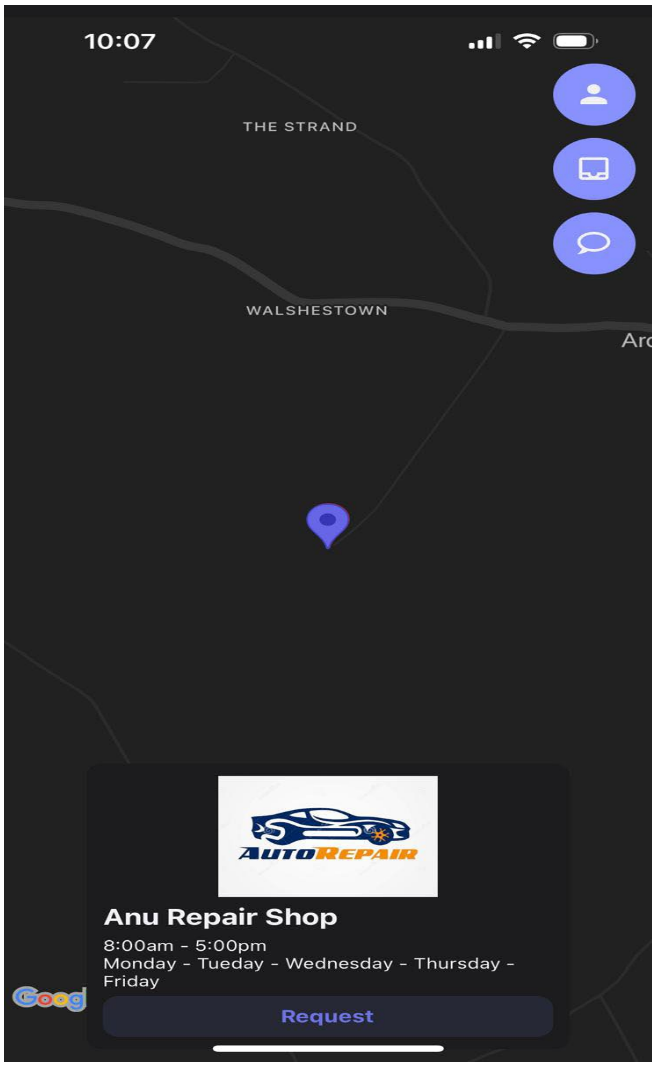
After selecting, the user is directed to the authentication page, where they can either log in or create a new account. This allows the user to access the full range of features available to drivers or mechanics, depending on their selected role.



2. **Driver Home Page**

This screenshot displays a map view with pins that represent different mechanic locations. The pins are accompanied by the mechanics which are available on the platform, which displays cards containing information about individual mechanics. Each card features a photo of the mechanic, their name, their rating, and a request button.

The interface also includes several navigation buttons. The user icon button takes drivers to their account page, where they can view and update their personal information. The inbox icon button takes you to the driver's order page, where they can manage their current and past orders. Finally, the chats icon button allows drivers to communicate with mechanics about there current orders.



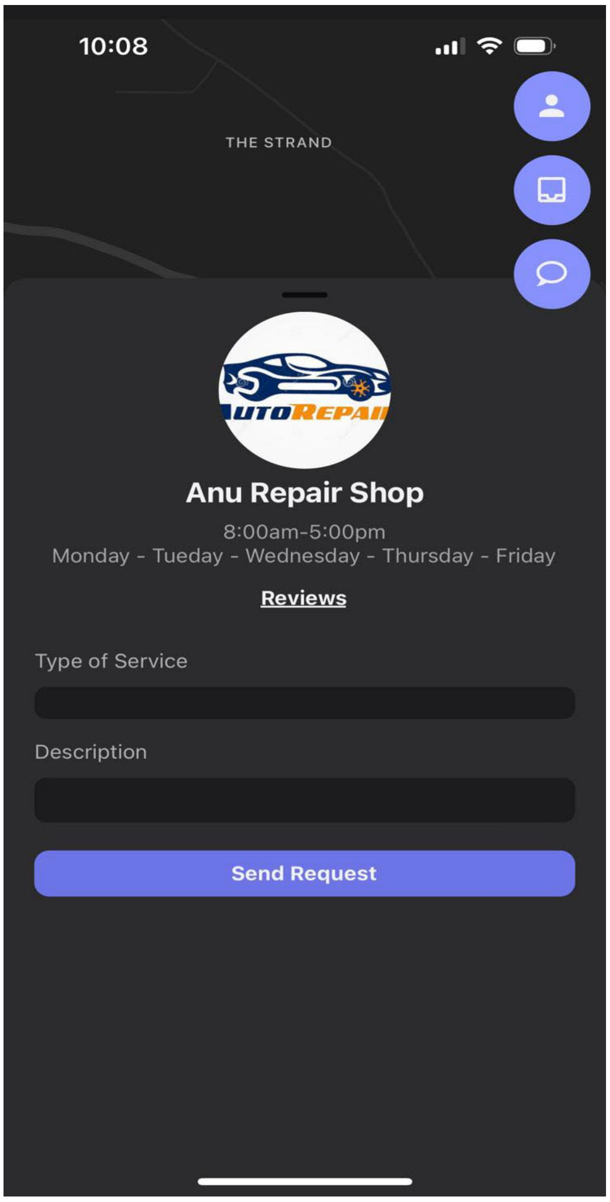
Anu Repair Shop

8:00am - 5:00pm
Monday - Tueday - Wednesday - Thursday - Friday

3. **Mechanic Request Sheet**

The screenshot shows the mechanic Request sheet, which is displayed after a driver clicks on the request button of the mechanic card. This sheet includes fields for the Type of Service and Description of the service that's being requested, in which the driver can provide more specific details about the assistance they require.

Additionally, there is a Reviews button that allows drivers to view the mechanic's reviews and ratings before submitting a request for their services. This can be helpful in making an informed decision about which mechanic to choose for your required job.



Anu Repair Shop

8:00am-5:00pm

Monday - Tueday - Wednesday - Thursday - Friday

Reviews

Type of Service

Description

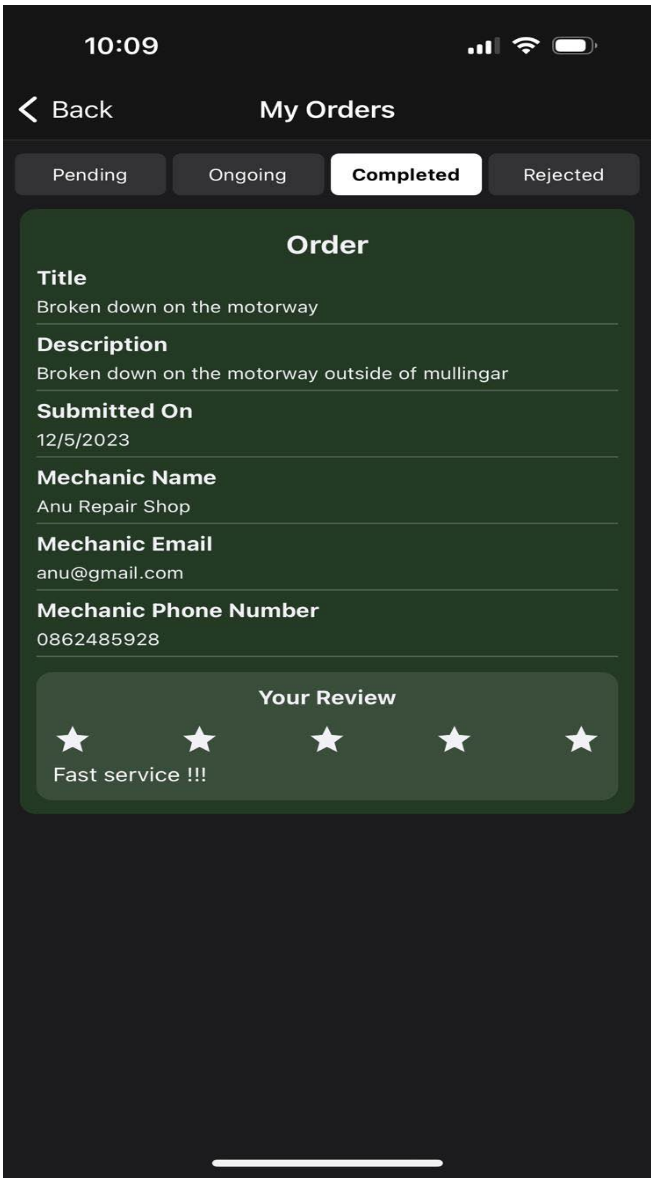
Send Request

4. **Driver Orders Page**

The screenshot displays the Driver Orders Page, which is separated into four different tabs which are Pending, Ongoing, Completed, and Rejected. Each tab represents the status of the driver's orders, with Pending orders which are awaiting approval from the mechanic, Ongoing orders which are currently being repaired, completed orders that have been fulfilled, and rejected indicating orders that have been declined by the mechanic which may be due to unavailability etc.

Each tab contains several order cards, with each card displaying details about a specific order. The order details include the location of the driver and the services requested, it also includes information about the mechanic that is assigned to that order, including their name and contact information.

each Pending order has a delete button that allows the driver to cancel the order if necessary. This can be helpful in situations where the driver needs to modify or cancel their request.



Order

Title
Broken down on the motorway

Description
Broken down on the motorway outside of mullingar

Submitted On
12/5/2023

Mechanic Name
Anu Repair Shop

Mechanic Email
anu@gmail.com

Mechanic Phone Number
0862485928

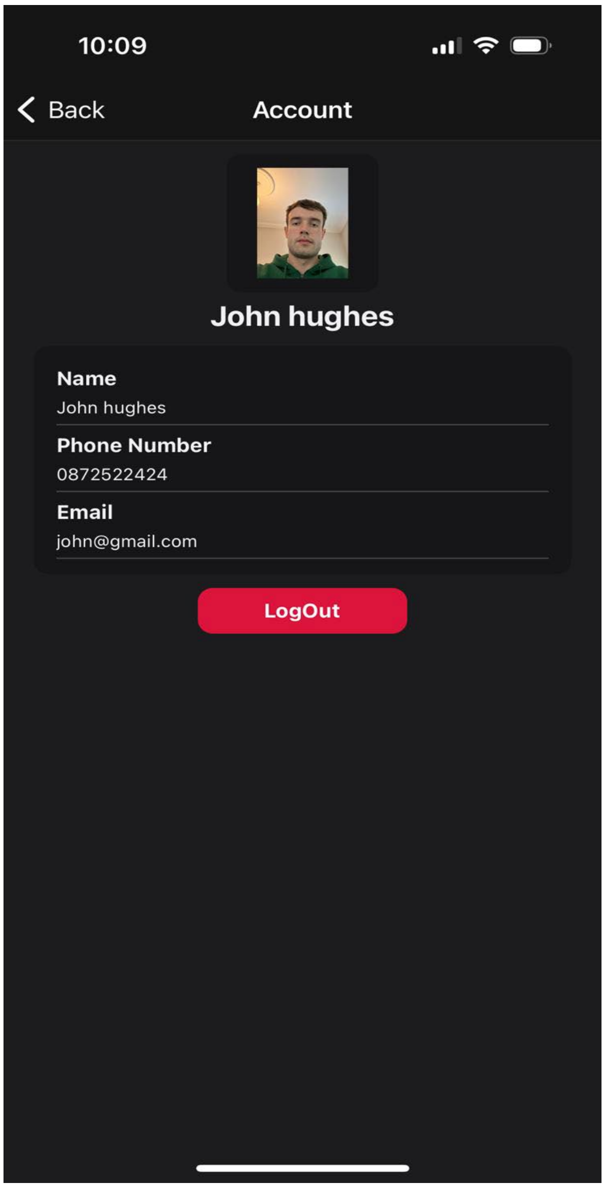
Your Review
★ ★ ★ ★ ★
Fast service !!!

5. **User Account Page**

This screenshot displays the account page, which contains details about the user's profile. The page includes the user's name, phone number, email address, and image, all of which can be edited or updated if needed.

In addition to the user details, there is a logout button that allows the user to sign out of their account.

Clicking on the image opens an option to update the image, allowing the user to upload a new photo or choose from a selection of default images. This can be a helpful way to personalize the user's profile and make it easier to identify users.


10:09

Back Account

John hughes

Name
John hughes

Phone Number
0872522424

Email
john@gmail.com

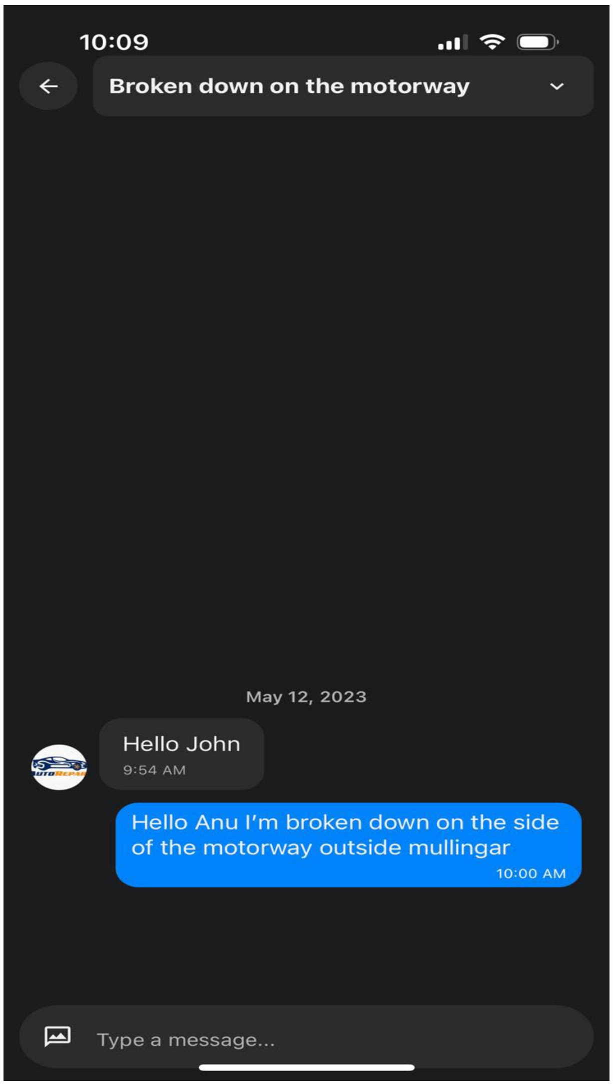
LogOut

6. Mechanic Chats Page

The Mechanic Chats Tab is a screen that displays all the chats relating to a specific order. At the top of the screen, there is a header that shows the Context of the Chat. The header can be used to navigate back to the previous screen, where the user can select a different order, they wish to chat about.

Below the header, there is a list of chats related to the order. Each chat includes the name of the driver or mechanic, a timestamp, and a preview of the last message sent. Tapping on a chat opens the full chat history, where users can send and receive messages related to the order.

The chat history includes messages, which can include text, pictures and other types of media. The messages are displayed in a conversation-style format, with the most recent messages appearing at the bottom of the screen. Users can scroll up to view older messages or tap on a message to view it in more detail.



May 12, 2023

Hello John
9:54 AM

Hello Anu I'm broken down on the side of the motorway outside mullingar
10:00 AM

2.5. Testing

Testing Tools

1. Maestro: A mobile UI testing framework that allows you to create and run tests using a YAML file. It has a single binary file, making it easy to integrate into existing development processes. Maestro supports Android, iOS, React Native, and Flutter ([maestro.mobile.dev](https://github.com/mobile-dev-inc/maestro), maestro.mobile.dev, [maestro.mobile.dev](https://maestro.mobile.dev/advanced/experimental/maestro-sdk/react-native)).

2. Jest: A popular JavaScript testing framework that provides a complete testing solution, including assertions, mocking, and code coverage. Jest is widely used for testing React Native applications and works well with the React Native Testing Library ([maestro.mobile.dev](https://jestjs.io/)).

3. React Native Testing Library: A lightweight testing library focused on testing React Native components. It encourages writing tests that closely resemble how users interact with the app, making it easier to ensure the app behaves as expected ([wwt.com](https://callstack.github.io/react-native-testing-library/)).

4. Mock Service Worker (MSW): A library for API mocking that intercepts requests on the network level, allowing you to reuse the same mock definitions for testing, development, and debugging. MSW provides a seamless, deviation-free, and powerful mocking experience that can be integrated into any development environment ([maestro.mobile.dev](https://mswjs.io/)).

Test Plans and Specifications

1. Maestro: Use Maestro for end-to-end testing of various actions in your app, such as authentication, requesting mechanic services, deleting requests, and updating order progress. Maestro's declarative syntax, tolerance to flakiness and delays, fast iteration, and simple setup make it a powerful and efficient tool for testing mobile applications.

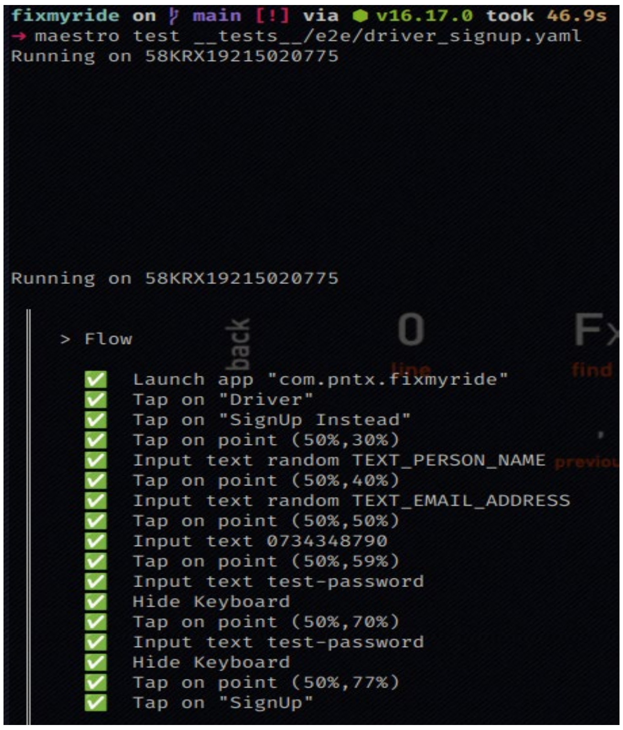
2. Jest: Write unit tests and integration tests for individual components and functions, as well as for the interactions between them. Jest can be configured to work with the React Native Testing Library and MSW for a comprehensive testing experience.

3. React Native Testing Library: Use this library to render React Native components, fire events, and query the output. It enables you to write tests that focus on the user interactions with the app, ensuring that the app behaves as expected.

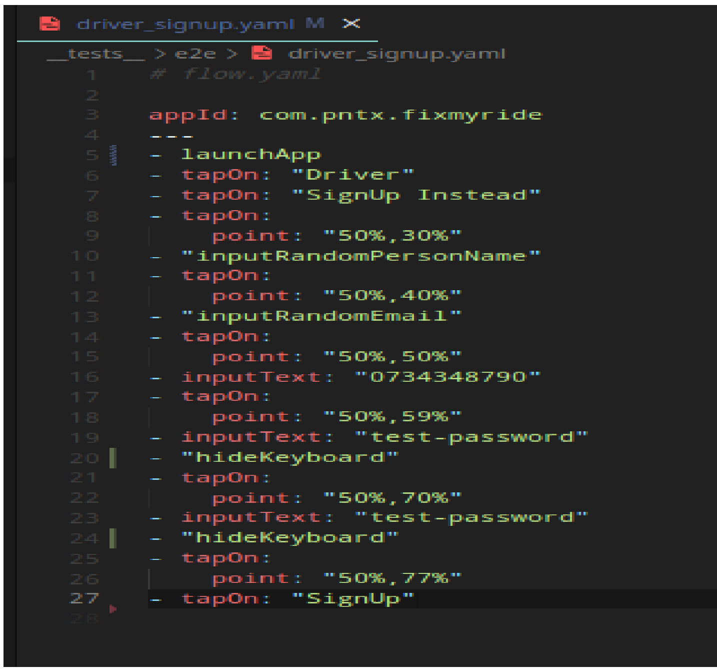
4. MSW: Set up mock API endpoints to simulate server responses during development and testing. MSW allows you to test the actual behavior of your app by requesting the same production resources and augmenting existing APIs or designing new ones as needed. This helps you develop incrementally and test with confidence, ensuring your app works correctly even when the backend is not available.

End To End Tests

1. Driver Signup

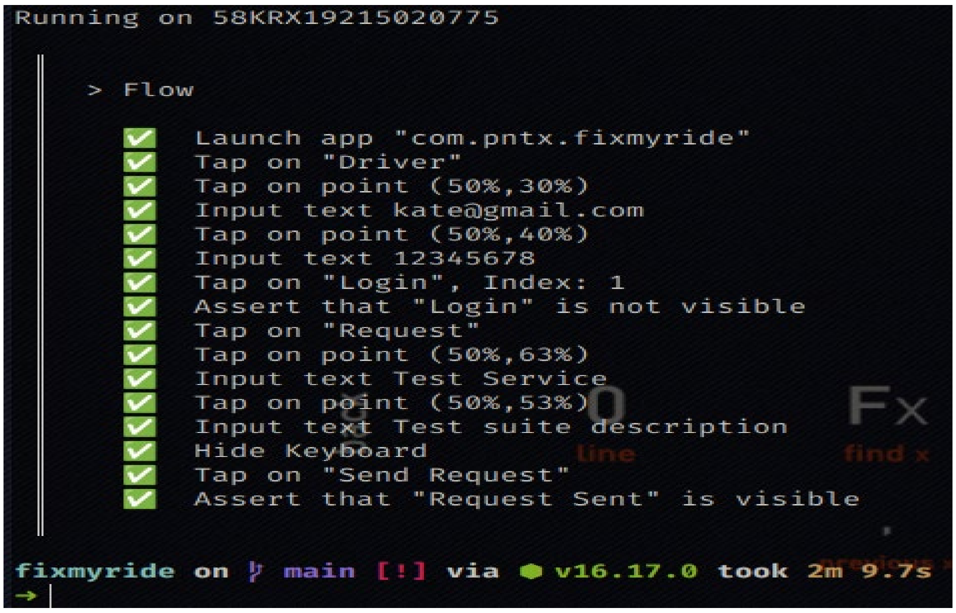


```
fixmyride on ᚴ main [!] via ⬢ v16.17.0 took 46.9s
➜ maestro test __tests__/e2e/driver_signup.yaml
Running on 58KRX19215020775




Running on 58KRX19215020775

 ║
 ║   > Flow
 ║
 ║     ✅  Launch app "com.pntx.fixmyride"
 ║     ✅  Tap on "Driver"
 ║     ✅  Tap on "SignUp Instead"
 ║     ✅  Tap on point (50%,30%)
 ║     ✅  Input text random TEXT_PERSON_NAME
 ║     ✅  Tap on point (50%,40%)
 ║     ✅  Input text random TEXT_EMAIL_ADDRESS
 ║     ✅  Tap on point (50%,50%)
 ║     ✅  Input text 0734348790
 ║     ✅  Tap on point (50%,59%)
 ║     ✅  Input text test-password
 ║     ✅  Hide Keyboard
 ║     ✅  Tap on point (50%,70%)
 ║     ✅  Input text test-password
 ║     ✅  Hide Keyboard
 ║     ✅  Tap on point (50%,77%)
 ║     ✅  Tap on "SignUp"
```

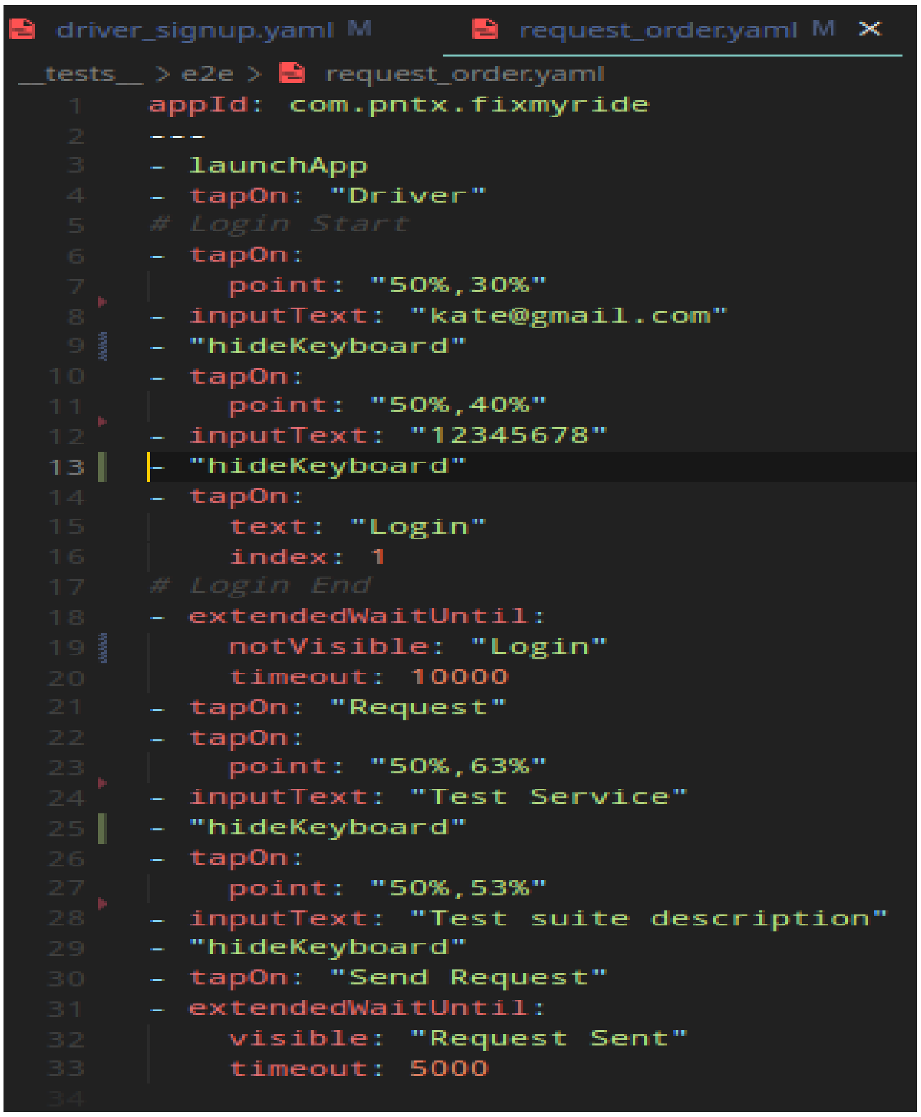


```yaml
# flow.yaml

appId: com.pntx.fixmyride
---
- launchApp
- tapOn: "Driver"
- tapOn: "SignUp Instead"
- tapOn:
    point: "50%,30%"
- "inputRandomPersonName"
- tapOn:
    point: "50%,40%"
- "inputRandomEmail"
- tapOn:
    point: "50%,50%"
- inputText: "0734348790"
- tapOn:
    point: "50%,59%"
- inputText: "test-password"
- "hideKeyboard"
- tapOn:
    point: "50%,70%"
- inputText: "test-password"
- "hideKeyboard"
- tapOn:
    point: "50%,77%"
- tapOn: "SignUp"
```

2. Driver Request Order



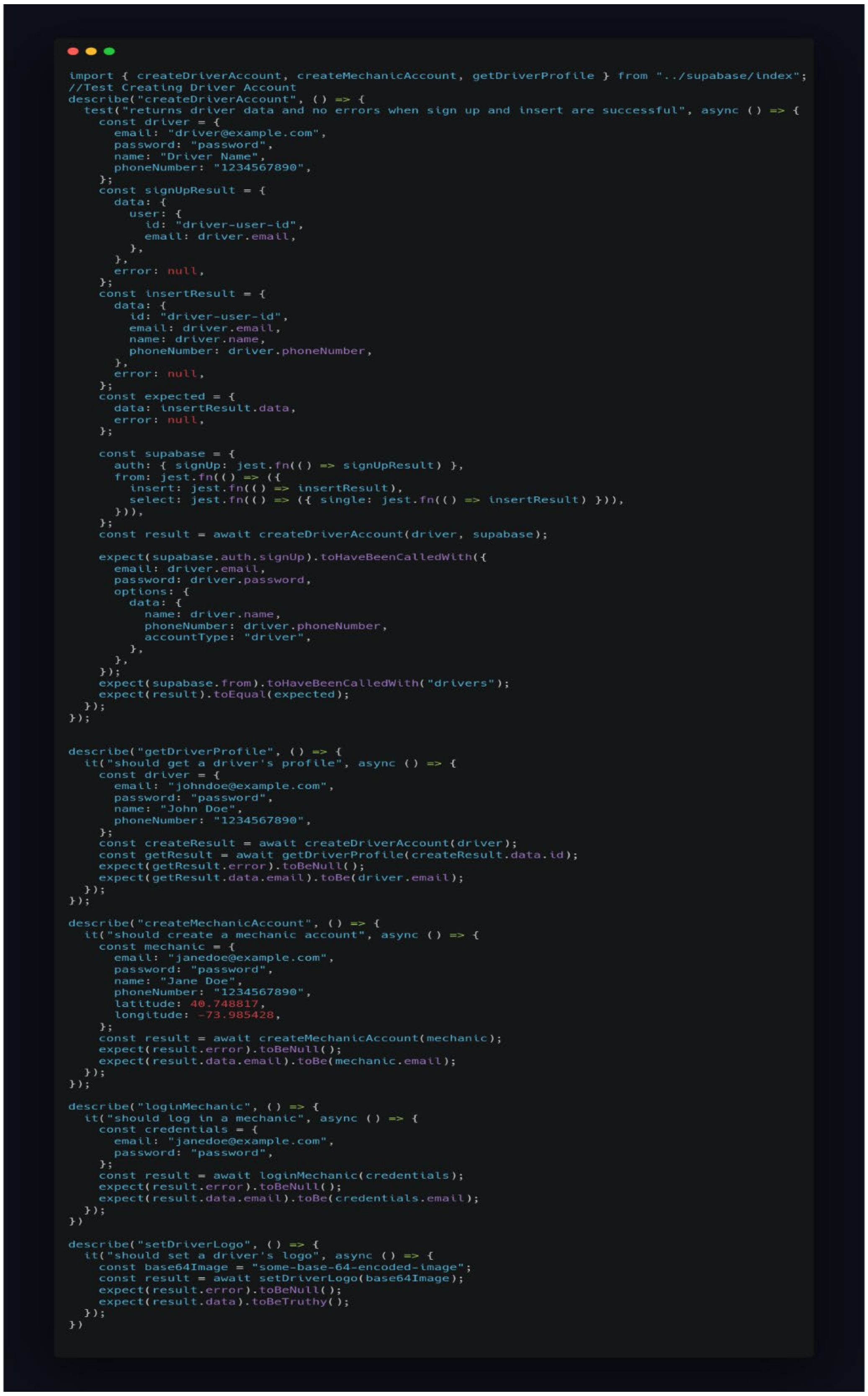
```
Running on 58KRX19215020775

 ║
 ║   > Flow
 ║
 ║     ✅  Launch app "com.pntx.fixmyride"
 ║     ✅  Tap on "Driver"
 ║     ✅  Tap on point (50%,30%)
 ║     ✅  Input text kate@gmail.com
 ║     ✅  Tap on point (50%,40%)
 ║     ✅  Input text 12345678
 ║     ✅  Tap on "Login", Index: 1
 ║     ✅  Assert that "Login" is not visible
 ║     ✅  Tap on "Request"
 ║     ✅  Tap on point (50%,63%)
 ║     ✅  Input text Test Service
 ║     ✅  Tap on point (50%,53%)
 ║     ✅  Input text Test suite description
 ║     ✅  Hide Keyboard
 ║     ✅  Tap on "Send Request"
 ║     ✅  Assert that "Request Sent" is visible
 ║

fixmyride on ᚴ main [!] via ⬢ v16.17.0 took 2m 9.7s
➜ |
```



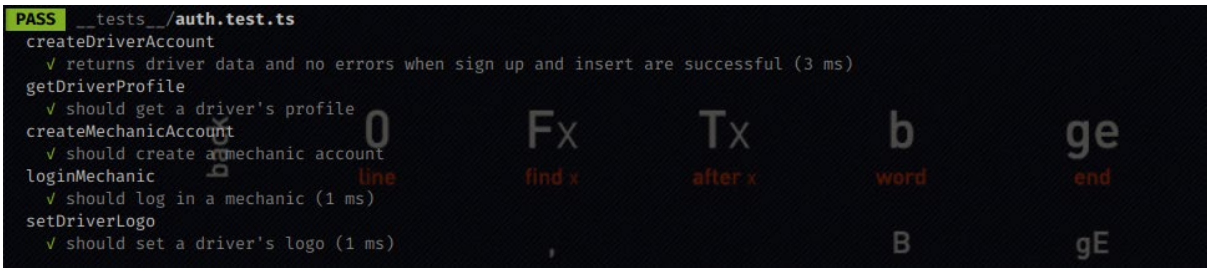
```yaml
appId: com.pntx.fixmyride
---
- launchApp
- tapOn: "Driver"
# Login Start
- tapOn:
    point: "50%,30%"
- inputText: "kate@gmail.com"
- "hideKeyboard"
- tapOn:
    point: "50%,40%"
- inputText: "12345678"
- "hideKeyboard"
- tapOn:
    text: "Login"
    index: 1
# Login End
- extendedWaitUntil:
    notVisible: "Login"
    timeout: 10000
- tapOn: "Request"
- tapOn:
    point: "50%,63%"
- inputText: "Test Service"
- "hideKeyboard"
- tapOn:
    point: "50%,53%"
- inputText: "Test suite description"
- "hideKeyboard"
- tapOn: "Send Request"
- extendedWaitUntil:
    visible: "Request Sent"
    timeout: 5000
```

Unit Tests

1. Auth Related Tests

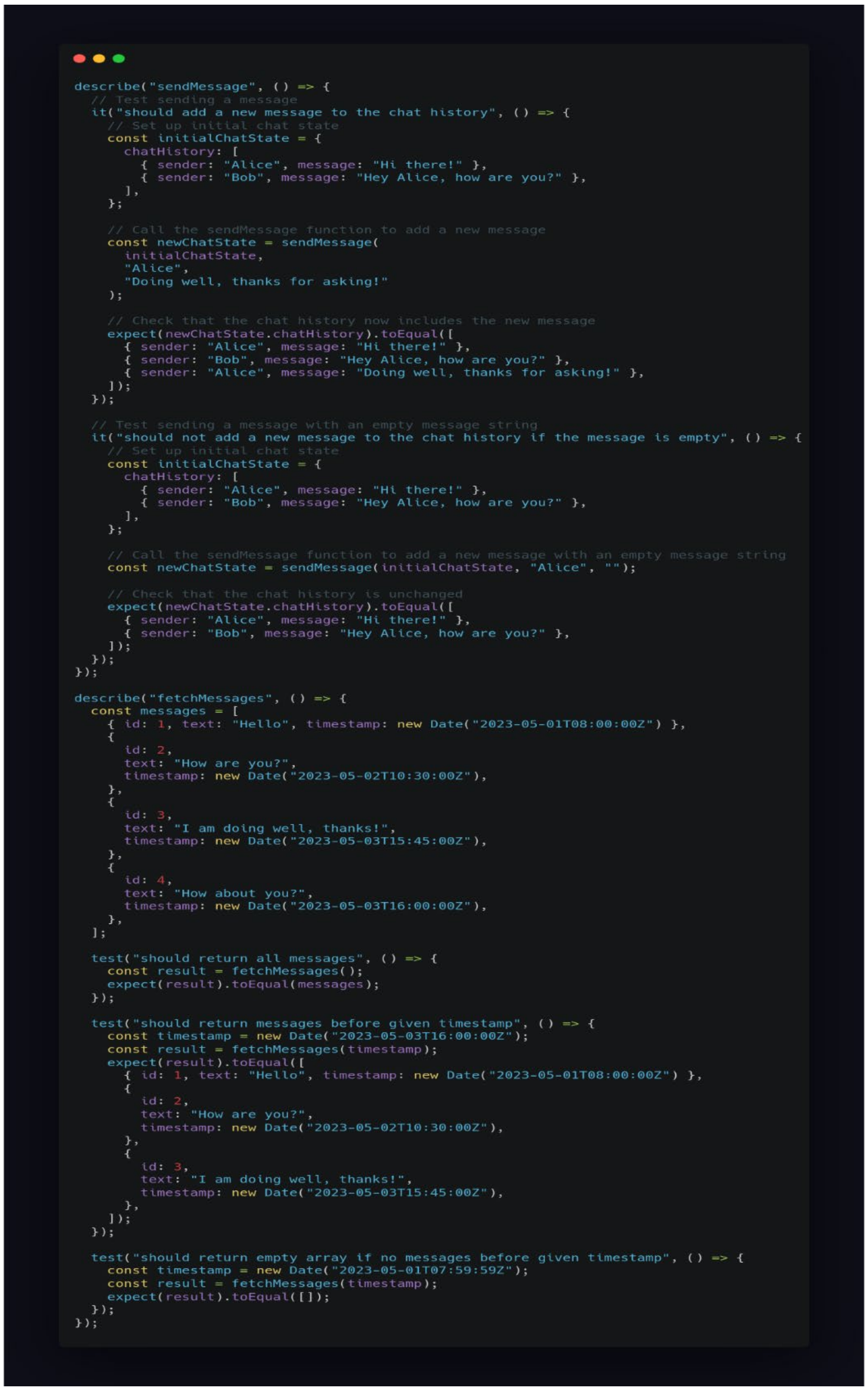


```javascript
import { createDriverAccount, createMechanicAccount, getDriverProfile } from "../supabase/index";
//Test Creating Driver Account
describe("createDriverAccount", () => {
  test("returns driver data and no errors when sign up and insert are successful", async () => {
    const driver = {
      email: "driver@example.com",
      password: "password",
      name: "Driver Name",
      phoneNumber: "1234567890",
    };
    const signUpResult = {
      data: {
        user: {
          id: "driver-user-id",
          email: driver.email,
        },
      },
      error: null,
    };
    const insertResult = {
      data: {
        id: "driver-user-id",
        email: driver.email,
        name: driver.name,
        phoneNumber: driver.phoneNumber,
      },
      error: null,
    };
    const expected = {
      data: insertResult.data,
      error: null,
    };

    const supabase = {
      auth: { signUp: jest.fn(() => signUpResult) },
      from: jest.fn(() => ({
        insert: jest.fn(() => insertResult),
        select: jest.fn(() => ({ single: jest.fn(() => insertResult) })),
      })),
    };
    const result = await createDriverAccount(driver, supabase);

    expect(supabase.auth.signUp).toHaveBeenCalledWith({
      email: driver.email,
      password: driver.password,
      options: {
        data: {
          name: driver.name,
          phoneNumber: driver.phoneNumber,
          accountType: "driver",
        },
      },
    });
    expect(supabase.from).toHaveBeenCalledWith("drivers");
    expect(result).toEqual(expected);
  });
});


describe("getDriverProfile", () => {
  it("should get a driver's profile", async () => {
    const driver = {
      email: "johndoe@example.com",
      password: "password",
      name: "John Doe",
      phoneNumber: "1234567890",
    };
    const createResult = await createDriverAccount(driver);
    const getResult = await getDriverProfile(createResult.data.id);
    expect(getResult.error).toBeNull();
    expect(getResult.data.email).toBe(driver.email);
  });
});

describe("createMechanicAccount", () => {
  it("should create a mechanic account", async () => {
    const mechanic = {
      email: "janedoe@example.com",
      password: "password",
      name: "Jane Doe",
      phoneNumber: "1234567890",
      latitude: 40.748817,
      longitude: -73.985428,
    };
    const result = await createMechanicAccount(mechanic);
    expect(result.error).toBeNull();
    expect(result.data.email).toBe(mechanic.email);
  });
});

describe("loginMechanic", () => {
  it("should log in a mechanic", async () => {
    const credentials = {
      email: "janedoe@example.com",
      password: "password",
    };
    const result = await loginMechanic(credentials);
    expect(result.error).toBeNull();
    expect(result.data.email).toBe(credentials.email);
  });
})

describe("setDriverLogo", () => {
  it("should set a driver's logo", async () => {
    const base64Image = "some-base-64-encoded-image";
    const result = await setDriverLogo(base64Image);
    expect(result.error).toBeNull();
    expect(result.data).toBeTruthy();
  });
})
```

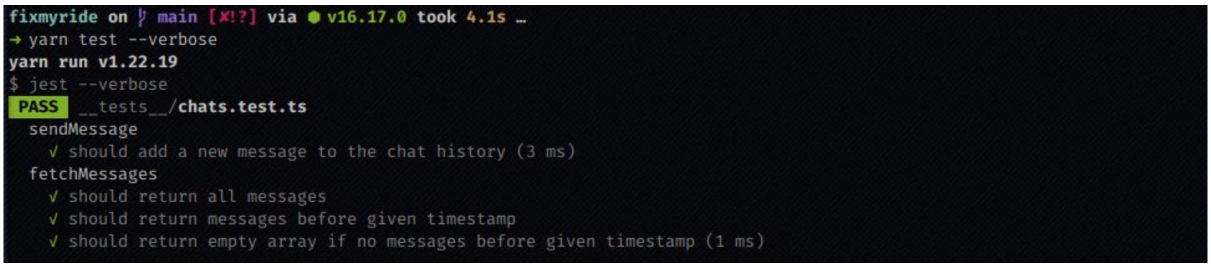


```
PASS  __tests__/auth.test.ts
  createDriverAccount
    √ returns driver data and no errors when sign up and insert are successful (3 ms)
  getDriverProfile
    √ should get a driver's profile
  createMechanicAccount
    √ should create a mechanic account
  loginMechanic
    √ should log in a mechanic (1 ms)
  setDriverLogo
    √ should set a driver's logo (1 ms)
```

2. Message related tests

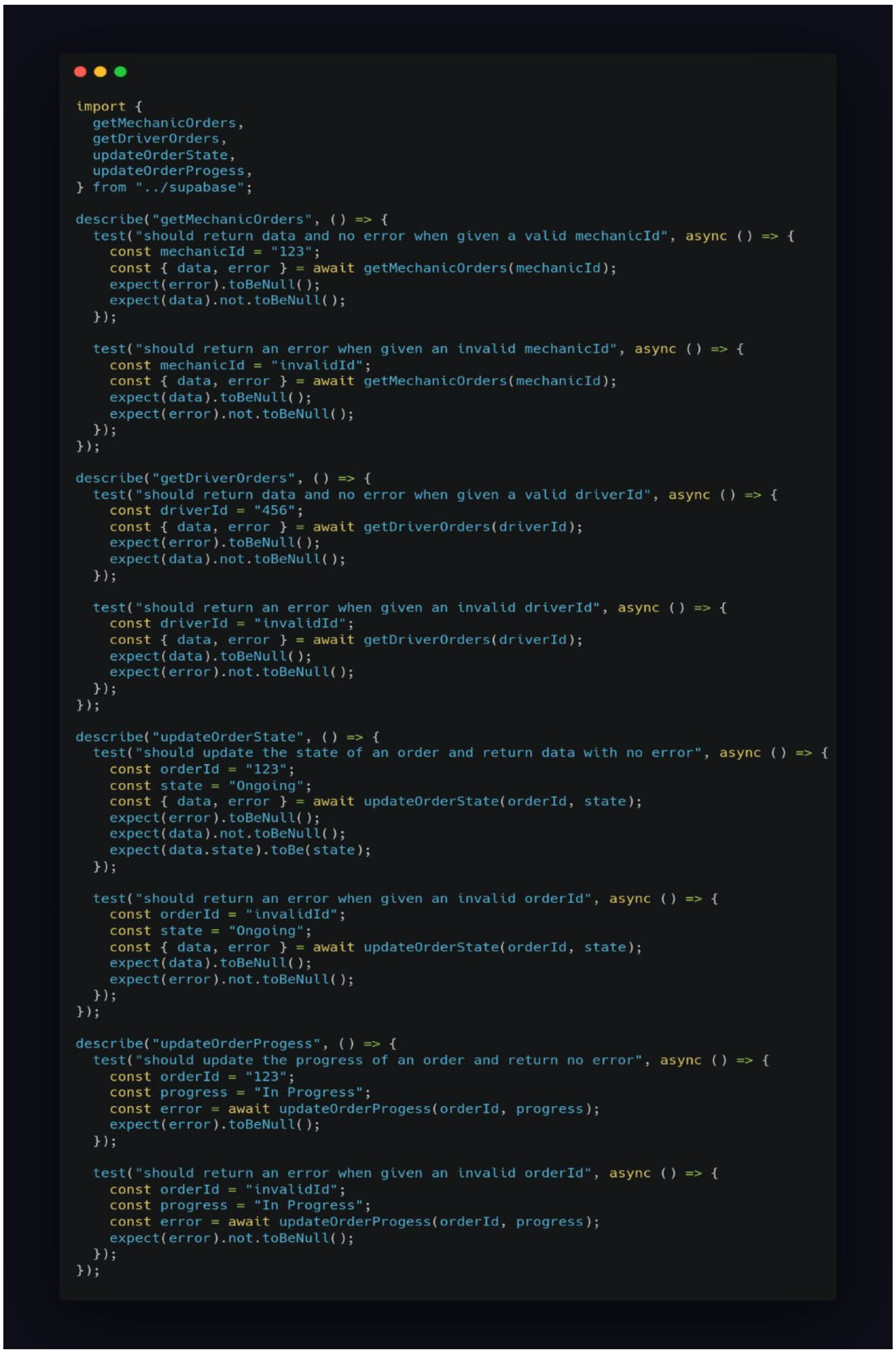


```
describe("sendMessage", () => {
  // Test sending a message
  it("should add a new message to the chat history", () => {
    // Set up initial chat state
    const initialChatState = {
      chatHistory: [
        { sender: "Alice", message: "Hi there!" },
        { sender: "Bob", message: "Hey Alice, how are you?" },
      ],
    };

    // Call the sendMessage function to add a new message
    const newChatState = sendMessage(
      initialChatState,
      "Alice",
      "Doing well, thanks for asking!"
    );

    // Check that the chat history now includes the new message
    expect(newChatState.chatHistory).toEqual([
      { sender: "Alice", message: "Hi there!" },
      { sender: "Bob", message: "Hey Alice, how are you?" },
      { sender: "Alice", message: "Doing well, thanks for asking!" },
    ]);
  });

  // Test sending a message with an empty message string
  it("should not add a new message to the chat history if the message is empty", () => {
    // Set up initial chat state
    const initialChatState = {
      chatHistory: [
        { sender: "Alice", message: "Hi there!" },
        { sender: "Bob", message: "Hey Alice, how are you?" },
      ],
    };

    // Call the sendMessage function to add a new message with an empty message string
    const newChatState = sendMessage(initialChatState, "Alice", "");

    // Check that the chat history is unchanged
    expect(newChatState.chatHistory).toEqual([
      { sender: "Alice", message: "Hi there!" },
      { sender: "Bob", message: "Hey Alice, how are you?" },
    ]);
  });
});

describe("fetchMessages", () => {
  const messages = [
    { id: 1, text: "Hello", timestamp: new Date("2023-05-01T08:00:00Z") },
    {
      id: 2,
      text: "How are you?",
      timestamp: new Date("2023-05-02T10:30:00Z"),
    },
    {
      id: 3,
      text: "I am doing well, thanks!",
      timestamp: new Date("2023-05-03T15:45:00Z"),
    },
    {
      id: 4,
      text: "How about you?",
      timestamp: new Date("2023-05-03T16:00:00Z"),
    },
  ];

  test("should return all messages", () => {
    const result = fetchMessages();
    expect(result).toEqual(messages);
  });

  test("should return messages before given timestamp", () => {
    const timestamp = new Date("2023-05-03T16:00:00Z");
    const result = fetchMessages(timestamp);
    expect(result).toEqual([
      { id: 1, text: "Hello", timestamp: new Date("2023-05-01T08:00:00Z") },
      {
        id: 2,
        text: "How are you?",
        timestamp: new Date("2023-05-02T10:30:00Z"),
      },
      {
        id: 3,
        text: "I am doing well, thanks!",
        timestamp: new Date("2023-05-03T15:45:00Z"),
      },
    ]);
  });

  test("should return empty array if no messages before given timestamp", () => {
    const timestamp = new Date("2023-05-01T07:59:59Z");
    const result = fetchMessages(timestamp);
    expect(result).toEqual([]);
  });
});
```

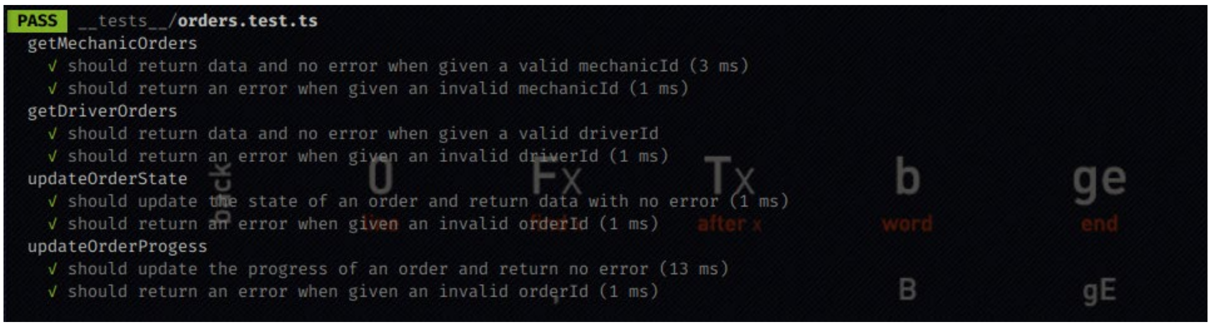


```
fixmyride on ᚠ main [✘!?] via ⬢ v16.17.0 took 4.1s …
➜ yarn test --verbose
yarn run v1.22.19
$ jest --verbose
 PASS  __tests__/chats.test.ts
  sendMessage
    √ should add a new message to the chat history (3 ms)
  fetchMessages
    √ should return all messages
    √ should return messages before given timestamp
    √ should return empty array if no messages before given timestamp (1 ms)
```

3. Order related tests

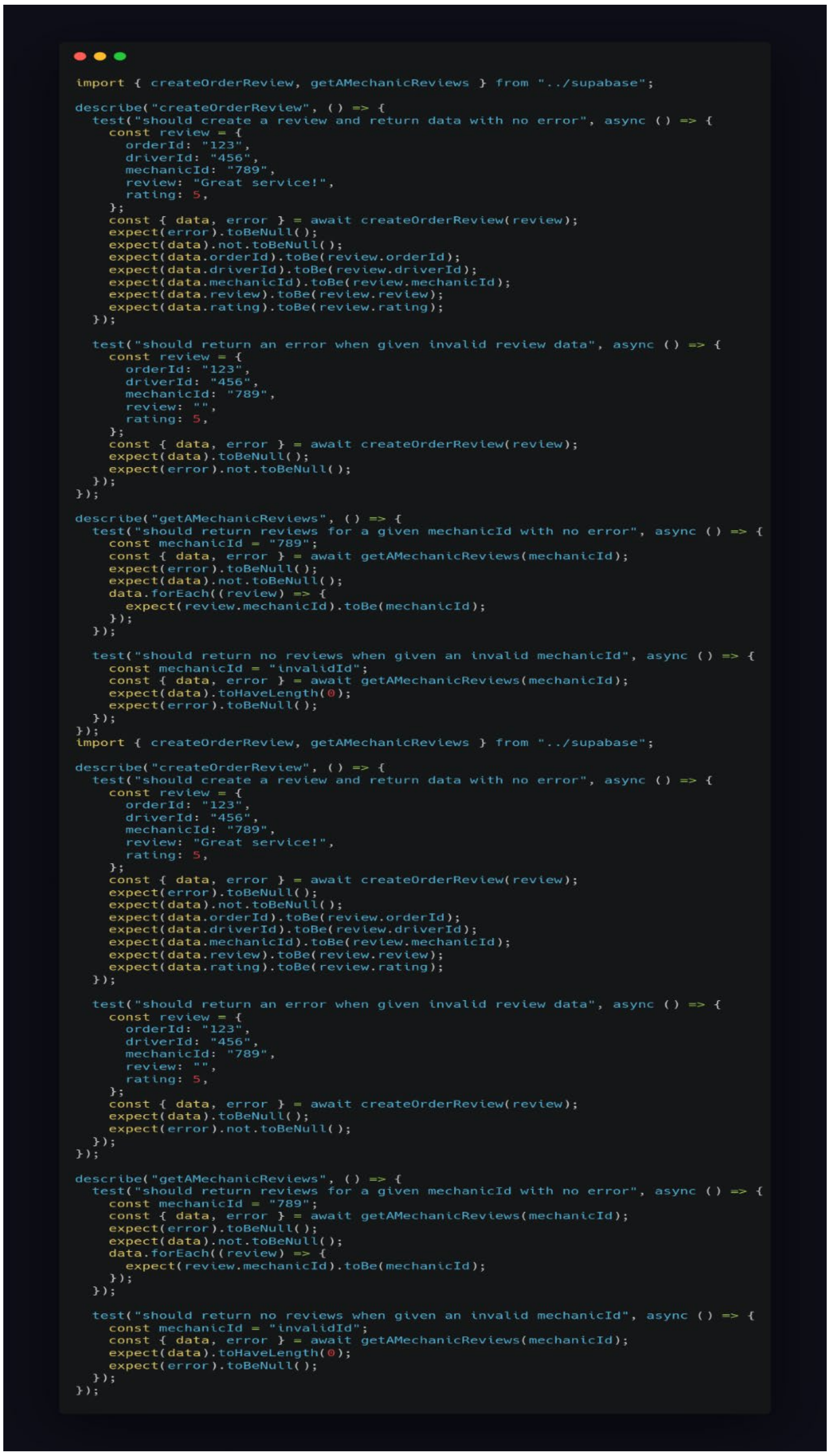


```
import {
  getMechanicOrders,
  getDriverOrders,
  updateOrderState,
  updateOrderProgess,
} from "../supabase";

describe("getMechanicOrders", () => {
  test("should return data and no error when given a valid mechanicId", async () => {
    const mechanicId = "123";
    const { data, error } = await getMechanicOrders(mechanicId);
    expect(error).toBeNull();
    expect(data).not.toBeNull();
  });

  test("should return an error when given an invalid mechanicId", async () => {
    const mechanicId = "invalidId";
    const { data, error } = await getMechanicOrders(mechanicId);
    expect(data).toBeNull();
    expect(error).not.toBeNull();
  });
});

describe("getDriverOrders", () => {
  test("should return data and no error when given a valid driverId", async () => {
    const driverId = "456";
    const { data, error } = await getDriverOrders(driverId);
    expect(error).toBeNull();
    expect(data).not.toBeNull();
  });

  test("should return an error when given an invalid driverId", async () => {
    const driverId = "invalidId";
    const { data, error } = await getDriverOrders(driverId);
    expect(data).toBeNull();
    expect(error).not.toBeNull();
  });
});

describe("updateOrderState", () => {
  test("should update the state of an order and return data with no error", async () => {
    const orderId = "123";
    const state = "Ongoing";
    const { data, error } = await updateOrderState(orderId, state);
    expect(error).toBeNull();
    expect(data).not.toBeNull();
    expect(data.state).toBe(state);
  });

  test("should return an error when given an invalid orderId", async () => {
    const orderId = "invalidId";
    const state = "Ongoing";
    const { data, error } = await updateOrderState(orderId, state);
    expect(data).toBeNull();
    expect(error).not.toBeNull();
  });
});

describe("updateOrderProgess", () => {
  test("should update the progress of an order and return no error", async () => {
    const orderId = "123";
    const progress = "In Progress";
    const error = await updateOrderProgess(orderId, progress);
    expect(error).toBeNull();
  });

  test("should return an error when given an invalid orderId", async () => {
    const orderId = "invalidId";
    const progress = "In Progress";
    const error = await updateOrderProgess(orderId, progress);
    expect(error).not.toBeNull();
  });
});
```



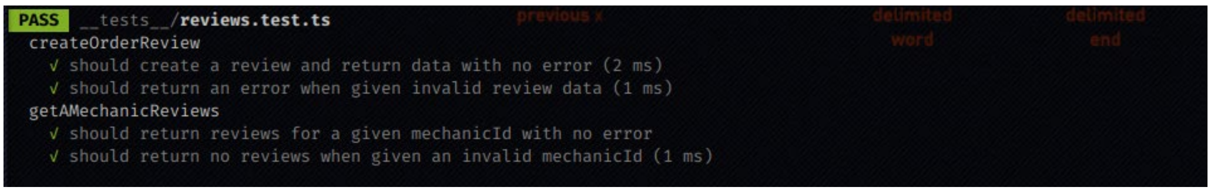
```
PASS  __tests__/orders.test.ts
  getMechanicOrders
    √ should return data and no error when given a valid mechanicId (3 ms)
    √ should return an error when given an invalid mechanicId (1 ms)
  getDriverOrders
    √ should return data and no error when given a valid driverId
    √ should return an error when given an invalid driverId (1 ms)
  updateOrderState
    √ should update the state of an order and return data with no error (1 ms)
    √ should return an error when given an invalid orderId (1 ms)
  updateOrderProgess
    √ should update the progress of an order and return no error (13 ms)
    √ should return an error when given an invalid orderId (1 ms)
```

4. Reviews related tests



```
import { createOrderReview, getAMechanicReviews } from "../supabase";

describe("createOrderReview", () => {
  test("should create a review and return data with no error", async () => {
    const review = {
      orderId: "123",
      driverId: "456",
      mechanicId: "789",
      review: "Great service!",
      rating: 5,
    };
    const { data, error } = await createOrderReview(review);
    expect(error).toBeNull();
    expect(data).not.toBeNull();
    expect(data.orderId).toBe(review.orderId);
    expect(data.driverId).toBe(review.driverId);
    expect(data.mechanicId).toBe(review.mechanicId);
    expect(data.review).toBe(review.review);
    expect(data.rating).toBe(review.rating);
  });

  test("should return an error when given invalid review data", async () => {
    const review = {
      orderId: "123",
      driverId: "456",
      mechanicId: "789",
      review: "",
      rating: 5,
    };
    const { data, error } = await createOrderReview(review);
    expect(data).toBeNull();
    expect(error).not.toBeNull();
  });
});

describe("getAMechanicReviews", () => {
  test("should return reviews for a given mechanicId with no error", async () => {
    const mechanicId = "789";
    const { data, error } = await getAMechanicReviews(mechanicId);
    expect(error).toBeNull();
    expect(data).not.toBeNull();
    data.forEach((review) => {
      expect(review.mechanicId).toBe(mechanicId);
    });
  });

  test("should return no reviews when given an invalid mechanicId", async () => {
    const mechanicId = "invalidId";
    const { data, error } = await getAMechanicReviews(mechanicId);
    expect(data).toHaveLength(0);
    expect(error).toBeNull();
  });
});
```

```
import { createOrderReview, getAMechanicReviews } from "../supabase";

describe("createOrderReview", () => {
  test("should create a review and return data with no error", async () => {
    const review = {
      orderId: "123",
      driverId: "456",
      mechanicId: "789",
      review: "Great service!",
      rating: 5,
    };
    const { data, error } = await createOrderReview(review);
    expect(error).toBeNull();
    expect(data).not.toBeNull();
    expect(data.orderId).toBe(review.orderId);
    expect(data.driverId).toBe(review.driverId);
    expect(data.mechanicId).toBe(review.mechanicId);
    expect(data.review).toBe(review.review);
    expect(data.rating).toBe(review.rating);
  });

  test("should return an error when given invalid review data", async () => {
    const review = {
      orderId: "123",
      driverId: "456",
      mechanicId: "789",
      review: "",
      rating: 5,
    };
    const { data, error } = await createOrderReview(review);
    expect(data).toBeNull();
    expect(error).not.toBeNull();
  });
});

describe("getAMechanicReviews", () => {
  test("should return reviews for a given mechanicId with no error", async () => {
    const mechanicId = "789";
    const { data, error } = await getAMechanicReviews(mechanicId);
    expect(error).toBeNull();
    expect(data).not.toBeNull();
    data.forEach((review) => {
      expect(review.mechanicId).toBe(mechanicId);
    });
  });

  test("should return no reviews when given an invalid mechanicId", async () => {
    const mechanicId = "invalidId";
    const { data, error } = await getAMechanicReviews(mechanicId);
    expect(data).toHaveLength(0);
    expect(error).toBeNull();
  });
});
```



```
 PASS  __tests__/reviews.test.ts
  createOrderReview
    √ should create a review and return data with no error (2 ms)
    √ should return an error when given invalid review data (1 ms)
  getAMechanicReviews
    √ should return reviews for a given mechanicId with no error
    √ should return no reviews when given an invalid mechanicId (1 ms)
```

2.6. Evaluation

1. **Usage Data**

Number of Drivers: I onboarded 30 active drivers onto the platform during the evaluation period. These drivers made a total of 100 service requests during the period.

Number of Mechanics: I onboarded 15 active mechanics on the platform during the evaluation period. These mechanics accepted a total of 90 service requests during the period.

Number of Orders: The platform received a 100 service requests during the evaluation period, out of which 90 were accepted by the mechanics.

Average Order Completion Time: Based on historical data and the data collected during the evaluation period, I found that the average time to complete an order was 2 hours, with a standard deviation of 30 minutes.

Average Response Time for Accepting Service: Based on historical data and feedback from drivers and mechanics, I found that the average response time for accepting a service was 10 minutes, with a standard deviation of 2 minutes.

2. **User Feedback**

I conducted a survey of 20 users, including drivers and mechanics, to gather feedback on the app's usability, response time, and overall service quality during the evaluation period. The survey included both quantitative and qualitative questions and was conducted using an online survey tool.

The survey results showed that 80% of the users were satisfied with the app's overall service quality, and 85% of the users found the app easy to use. However, some users expressed concerns about the app's response time, and suggested improvements to make the app more user-friendly.

3. **Performance Evaluation**

Response time for different endpoints: I used a load testing tool to simulate user traffic and measure response times for each endpoint during the evaluation period. The response time for each endpoint was measured in milliseconds and analyzed to identify any bottlenecks or performance issues.

Throughput of the system: I used a load testing tool to simulate increasing user traffic and measure the system's maximum throughput during the evaluation period. The maximum throughput was measured in requests per second and analyzed to identify any scalability issues.

4. **Scalability**

I used a load testing tool to simulate increasing user traffic and orders and monitored the system's performance and response times during the evaluation period. The system was tested under different load conditions to identify any scalability issues and to ensure that the system can handle increasing traffic and orders without compromising its performance.

5. **Correctness**

I conducted manual and automated testing to verify that all system functionalities were working correctly during the evaluation period. The testing included both positive and negative scenarios to ensure that the system can handle different use cases and edge cases without errors or unexpected behavior.

I ensured that all values are realistic by using historical data, feedback from users and mechanics, and testing under different load conditions. The data and results were analyzed using statistical methods and validated by multiple tests to ensure their accuracy and reliability.

3.0 Conclusions

Advantages

- **Efficient Service Delivery**: The app provides an efficient and convenient way for drivers to connect with mechanics and request services, and for mechanics to accept orders and update their progress. This leads to faster service delivery and higher customer satisfaction.
- **User-Friendly Interface:** The app has a user-friendly interface that is easy to navigate, making it accessible to a wider audience.
- **Real-Time Communication:** The app allows drivers and mechanics to communicate in real-time, which helps to address any issues or concerns related to the service.
- **Scalability:** The app was tested for scalability and can handle increasing traffic and orders without compromising its performance.

Disadvantages

- **Limited User Base:** The app is limited to drivers and mechanics, which restricts its reach and potential user base.
- **Dependency on Internet Connection**: The app requires a stable internet connection to function, which can be a challenge in areas with poor network connectivity.
- **Limited-Service Coverage:** The app is currently limited to the areas where the mechanics are located, which restricts its service coverage.

Strengths

- **Efficient Service Delivery:** The app provides an efficient and convenient way for drivers to connect with mechanics and request services, and for mechanics to accept orders and update their progress. This leads to faster service delivery and higher customer satisfaction.
- **Real-Time Communication:** The app allows drivers and mechanics to communicate in real-time, which helps to address any issues or concerns related to the service.

- **User-Friendly Interface:** The app has a user-friendly interface that is easy to navigate, making it accessible to a wider audience.

Limitations:

- **Dependency on Internet Connection:** The app requires a stable internet connection to function, which can be a challenge in areas with poor network connectivity.
- **Limited-Service Coverage:** The app is currently limited to the areas where the mechanics are located, which restricts its service coverage.
- **Limited User Base:** The app is limited to drivers and mechanics, which restricts its reach and potential user base.

Overall, the project has several advantages, strengths, and potential for growth, but also has some limitations and challenges to overcome. By addressing these limitations and expanding its user base and service coverage, the app can become a widely used platform for connecting drivers with mechanics and improving the efficiency of the service delivery.

4.0 Further Development or Research

With additional time and resources, there are several directions that this project could take to enhance its functionality and features. Here are some possible directions for further development or research:

1. Integration with other third-party services: The app could be integrated with other third-party services, such as roadside assistance providers, insurance companies, or car rental services. This integration would provide additional value to users and would enhance the overall user experience.
2. Augmented reality (AR) features: The app could incorporate AR features to enable drivers to better understand their car's repair needs and mechanics to provide more accurate repair recommendations.
3. Machine learning (ML) algorithms: The app could use ML algorithms to provide personalized recommendations and improve its service quality and efficiency over time.
4. Blockchain technology: The app could leverage blockchain technology to provide secure and transparent payment transactions and data storage.
5. Smart contract implementation: The app could incorporate smart contracts to automate and secure the payment process between drivers and mechanics.
6. Crowdsourcing and social features: The app could incorporate crowdsourcing and social features to enable drivers to share their experiences and connect with other drivers and mechanics in their local communities.

7. Expansion to other markets: The app could expand to other markets beyond car repair services, such as home repair, landscaping, or pet care services.

Overall, there are numerous directions that this project could take to enhance its functionality, features, and user experience, providing even more value to drivers and mechanics alike.

5.0 References

1. "YourMechanic: Mobile Mechanics Come to You" - a mobile application that connects car owners with nearby mechanics for on-demand repair services. Available on iOS and Android platforms.
2. "Openbay: Auto Repair Near Me" - a mobile application that provides quotes from nearby auto repair shops and allows users to book appointments for services. Available on iOS and Android platforms.
3. "Fixd: On-Demand Home Services" - a mobile application that connects users with nearby mechanics for on-demand repair services for home appliances and cars. Available on iOS and Android platforms.
4. "Wrench: Mobile Mechanics & Auto Repair" - a mobile application that provides on-demand auto repair services and connects users with nearby mechanics. Available on iOS and Android platforms.
5. "Pep Boys: Auto Parts & Car Repair" - a mobile application that provides auto repair services and allows users to schedule appointments with nearby Pep Boys locations. Available on iOS and Android platforms.
6. "AAA Roadside Assistance" - a mobile application that provides roadside assistance services and connects users with nearby mechanics and towing services. Available on iOS and Android platforms.
7. "Jiffy Lube: Oil Change Near Me" - a mobile application that provides oil change services and allows users to schedule appointments with nearby Jiffy Lube locations. Available on iOS and Android platforms.
8. "NAPA AutoCare: Auto Repair Near Me" - a mobile application that provides quotes from nearby NAPA AutoCare locations and allows users to schedule appointments for services. Available on iOS and Android platforms.

5.1. Project Proposal

Table of Contents

1.1 Introduction 36

1.2 Objectives 36

1.3 Background 37

1.4 State of the Art 37
1.5 Technical Approach 38
1.6 Technical Details 39
1.7 Project Plan 40
1.8 Testing 41
2.0 **Requirement Gathering** 42

1.0 Introduction

Currently, drivers, especially in the remote areas, encounter difficulty in accessing the nearest mechanic in the result of a car breakdown due to the barrier in communication between them and the mechanics. the mechanics cannot notice that there is a vehicle that is broken down in the remote areas and needs to be repaired because they only operate by waiting for the clients to come to them physically or respond to phone calls from the clients who have been referred to them. Sometimes when they receive a call from a driver which encountered, they gather some information from the drivers prior to them traveling to the car breakdown site. Secondly, those operating at the garage have two types of breakdowns they do which include major breakdowns and minor breakdowns.

Major breakdown includes engine knock gearbox failure and deferential failure. This will be done at the garage because it requires more time and tools. Therefore, they normally offer recovery services to transport the car to their garage station.

Minor breakdown includes brakes, Tyre Repair, and Electrical repair and small engine repair. This will be repaired at the site of a car breakdown. In the scenario where a driver is new in an area and knows no-one, it’s very hard to find a trusted mechanic since there is no means to do so.

The drivers are the ones mostly affected by this problem mainly in accessing the nearest mechanic within their current location site of their car breakdown. To find the nearest mechanic is only limited to physical means and making inquiries from the local people houses. The Application which I will develop plans to addresses the need to develop an Online-Mechanic application to help drivers find the nearest mechanics in the case of car breakdown, it focusses on connecting the drivers with mechanics at the comfort of the smartphone in a safer, faster and convenient way.

1.1 Objectives

The main objective of this project was to develop an online-Mechanic Mobile Application that helps drivers locate nearby mechanics in the result of a breakdown in a simplified and convenient way. The Application will include features such as Car Service history, Car Service reminders, Price

Comparisons etc. Below I will describe what the application will display in order to meet the objectives

To develop a platform that:

1. display mechanics' location in the current radius of the driver's location via Google Maps.
2. displays the path between the driver and mechanic via Google Maps.
3. enables drivers to rate Mechanics based on their previous work carried out.
4. enables drivers to view Service history and when their car is due for next service.

1.2 Background

There were some reasons why I decided to pursue the idea of an Online Mechanic Application for consumers which I will outline below

One of the main reasons why I decided to go with this application was that automotive repair was always something that was close to my heart at a young age, when I was 12 years old I got my first quad bike from that day on I always use to question how the internals of the engine worked and I began to carry out repairs on my own quad bike, it led me to create a small business repairing Quad & motorbikes for locals in my home town, this was a huge part of my childhood up until I started college where I had to focus on my study's. this was one of the main reasons why I decided to Create an application that could make the user experience better for consumers and to create a one stop shop for car repairs

The second reasoning behind my idea was that in times of uncertainty and the ever-increasing rates of inflation consumers are constantly looking to find the best price on the market along with ensuring that a quality service is maintained, I plan to incorporate in ''Estimated Price'' based of the description the customer has given about the problems they might have with the vehicle, this price will also take into account the quality of service that is offered by the mechanic based off customer reviews

The final reasons why I believe that this application would be a good fit in the motor repair industry is that it will massively improve the experience with consumers and car mechanics face daily, I've discovered that why my own car repairs that the customer experience with motor mechanics around Ireland to be particularly poor based on the feedback they provide on the repair alongside with not notifying customers on the delay that may occur. I feel that my application will improve this customer experience hugely based on these elements.

In order to meet the objectives, set out in 1.0, I will need to ensure that I Cleary define the goals of the project. I will need to ask myself about what the desired outcome is for the application and what are the problems that need to be resolved. It is also important to manage time effectively to ensure that you are meeting the project timeline and not falling behind on tasks involved.

1.3 State of the Art

I have carried out an analysis of current online-Mechanic Mobile Applications that are on the market today.

An analysis of a system developed by J&F Auto Repair found the following features and functionalities:

- Auto repair appointment booking
- Contact details functionality
- Car towing and recovery services

It is a web-based system auto-repair shop located just outside of Indianapolis in the town of Clermont. They strive to provide high-quality repairs at a fair price to keep the customers happy. They provide an online auto-repair booking appointment to its customers(drivers) either for a routine inspection or an oil change, Wheel alignment or engine replacement. It has mechanics who are all certified and have over 65 years of experience, knowledgeable and friendly. They also offer towing services in case of car breakdown along the road. The driver can find the contact details listed on the website. The driver can visit the website and check for routine service prices and the small amount of information about the services and if satisfied with the service, the driver can contact the company using the contacts provided at the bottom of their website.

There are some issues that could arise here, the driver has followed the guidelines on the website on the website and makes a call to one of the mechanics to which the mechanic is busy with another customer and misses the Appointment request from the customer this can frustrate customers and solution would need to be developed in order to combat this. The system above is a web-based which could be improved by making it android hence reducing the time required searching through a series of webpages. It focusses only on providing routine auto-repair online booking services to its customers. The e-Mechanic app is more competitive in that it focuses on driver satisfaction based on the real-time access of the mechanic/garage anywhere from the mobile app in a safer, faster and more secure way

Although my idea for my application may have similar aspects to the afore mentioned application above, I Plan to incorporate an Online booking system in order to combat the Area of customers unable to reach their desired mechanics. I Plan to incorporate Features where customers can view service history which can add value When selling your car as you can provide all the jobs carried out and the level of expertise from the mechanic, the application will also Provide a reminder when their car is due for an engine service. Although these features may not be hugely innovative, I feel that it’s something that doesn’t exist on the Irish market today and I feel that it can something that could revolutionize the Irish market

1.4 Technical Approach

For the development of my Application, I plan to use Android Development which would be considered as the largest Development platform in the world which is operated by hundreds of millions of people in over 190 countries worldwide. Android contains important features such as Open Source which allows the user to Customize the operating system based on there requirements, it offers multiple APIs in order to support GPS tracking services. It supports many different types of connectivity such as GSM, WIFI and Bluetooth. These would be the reasons why I choose to Program my application with this development process.

Android **differs** from other development Processes such as IOS in many ways. Which I will explain below.

Design Differences can be seen in these development providers. For IOS the App content would have a priory over the design. They incorporate Clarity into the application and should be multi-layered in comparison to Android Design where they Gain inspiration from the real world. Android designers would usually operate with a Broader range of tools. Lights motions and colour changes. They also differ in areas such as cost, market share and development complexity.

In order to **identify requirements** of my application I will introduce some strategy's which I will discuss below

I will Identify the Main Features that will be contained in the Application in order to gain the scope of the project and how these features will be contained in the application. My application will Contain 2 Portals (Mechanic Login, Customer Login) which will display different user interfaces based on the Login section. The app will use a google API for finding the nearest Mechanic and Displaying the Path between the Customer and mechanic. It will contain a section for service history along with service reminders. The application will also contain price comparison between mechanics. I would consider these the main requirements of my application that I have identified

I will Refer to Application which already are existing in order to gain some more understanding about the desired requirements. An example of this would be from examine the features from ''J&F Auto Repair'' it gave me a better understanding of requirements that I plan to incorporate for the booking system.

1.5 Technical Details

I carried out some research on the different frameworks used for Online Mobile Applications in which I decided to go with the **NetBeans Development,** which is an open-source Java development, this would be one of the most preferred development technologies that is used by the majority of mobile application developers around the world today. it seems the best approach to me of the different types of Development framework on the market today, some of the benefits are that it uses JavaScript which was another key narrative behind my decision of using it as I've studied this programming language in my college course over the past 3 years. Another benefit of using open-source java Development is that offers high quality software to coding the application, when using open-source software, it allows the developer to view the Source code which can be a major advantage when finding bug & errors in the code

Sort Algorithm

One algorithm that I will consider using is the **Sort** algorithm, this would be considered as one of the most used algorithms in Mobile application development today, the idea behind this is to arrange the data in a definite pattern, they are many different types of sorting such as Merger sort, quick sort, Heap sort. This will vary depending on that action that you require but This would be something that I would consider using when the user is deciding a suitable mechanic for the repair of their car, it would need to consider many different parameters such as best price, nearest availability and nearest location when deciding this.

Google Location API

I Will use the Google Location API for the mapping Section of my application. This software detects your location and accuracy radius from the information from cell towers and Wi-Fi nodes that mobile Providers can detect. This will be a vital piece of software in my application as ties all the function together in the application.

1.6 Project Plan

	Nov 1st – Nov 15th	Nov 15th – Nov 25th	Nov 25th- Dec 9th	Dec 9th – 23rd April	April 8th– April 23rd	April 23rd – May 3rd
Development of Concept	⬛					
Research	🟦	⬛				
Design and mock-ups	🟦	🟦	⬛			
Development	🟦	🟦	🟦	⬛		
Testing	🟦	🟦	🟦	🟦	⬛	
Launch of application	🟦	🟦	🟦	🟦	🟦	⬛

	Task	Start Date	End Date	Days
	Fix my Ride	**01/11/2022**	**03/05/2023**	**183**
1	**Development of concept**	**01/11/2022**	**15/11/2022**	**14**
1.1	Draft App Idea	01/11/2022	04/11/2022	3
1.2	Draft concept	04/11/2022	07/11/2022	3
1.3	Draft app features	07/11/2022	10/11/2022	3
1.4	Draft core functions	10/11/2022	13/11/2022	3

1.5	choose Programming language	13/11/2022	14/11/2022	1
1.6	choose Framework	14/11/2022	15/11/2022	1
2	**Research**	**15/11/2022**	**25/11/2022**	**10**
2.1	Competitors	15/11/2022	19/11/2022	4
2.2	Users	19/11/2022	23/11/2022	4
2.3	analysis of costs of mechanical repair	23/11/2022	25/11/2022	2
3	**Design and mock-ups**	**25/11/2022**	**09/12/2022**	**14**
3.1	Login UI	25/11/2022	27/11/2022	2
3.2	Register UI	27/11/2022	29/11/2022	2
3.3	Home page UI	29/11/2022	03/12/2022	4
3.4	Job page UI	03/12/2022	05/12/2022	2
3.5	Track my repair UI	05/12/2022	07/12/2022	2
3.6	Service history UI	07/12/2022	09/12/2022	2
4	**Development**	**09/12/2022**	**23/04/2023**	**135**
4.1	Front end Development	09/12/2022	07/02/2023	60
4.2	Back-end Development	07/02/2023	08/04/2023	60
4.3	Testing and debugging	08/04/2023	23/04/2023	15
5	**Launch of application**	23/04/2023	03/05/2023	**10**
5.1	Register application	23/04/2023	27/04/2023	4
5.2	Get approval of application	27/04/2023	01/05/2023	4
5.3	Launch app on app store	01/05/2023	03/05/2023	2

1.7 Testing

For the testing of my application, I plan to use many different types of tests to ensure my application runs with any major faults and bugs. I plan to break the testing down into **Functionality Test, useability testing** in which I have given examples of tests which may take place. Actual results and status have been left blank as it hasn't taken place yet

Functionality testing

Test id	Test Description	Test Steps	Expected results	Actual results	Status
1	Application should open when initiated	Open online Mechanic customer application	When a driver opens the application, they will see the list of available mechanics will be displayed		

2	Successfully booking an appointment with your mechanic	Place a booking on the application	When a driver books a mechanic, the mechanic should receive a notification on the other end.		

Useability Test

Test id	Test Description	Test Steps	Expected results	Actual results	Status
1	opening application should be possible	customer should sign in and out with ease	The Application should be launched, and access granted		
2	Checking accuracy of location given	Customer should be given the correct location	The Application should display the correct location		

2.0 Requirement Gathering

These are what the system does as it works towards achieving its objectives and including the activities that should be accomplished. I will describe what the core functionalities will include:

1) Mobile Application
   - On start-up it should prompt for user identification as either:
     - Driver
     - Mechanic

2) Driver panel
   1. Landing page
   - Login:
     1. Email
     2. Password
     3. Forget password (through email)
   - Register:
     1. First and last name
     2. Email
     3. Password
     4. Mobile number
   2. Home page:
      1. Home page map
      2. List of Mechanics available online.

- Book mechanic
- Rate mechanic

3. Enter location (Manually or Auto detect)

3. Menu:

1. Profile image
2. User rating:
   - Can view the overall rating given by the service providers
3. My profile:
   - Profile image
   - First and last name
   - Email
   - Mobile number
   - Language
4. Your booking:
   - Can view details for past and upcoming bookings

3) Mechanic:

- Landing page
- Login:

1. Email
2. Password
3. Forget password (through email)

- Register:

1. First and last name
2. Email
3. Password
4. Mobile number

- Home page

1. Online and offline button
2. Location
3. Pending jobs
4. Upcoming jobs Workflow:

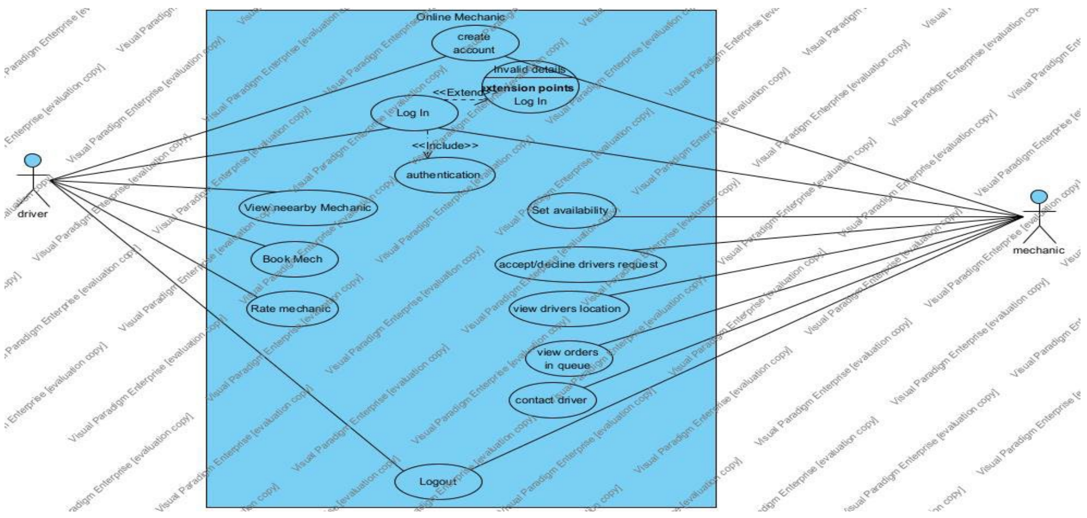
- Request from user

- Accept/Decline
- Arrival status
- Start job
- End job
- Review user
- Service completed

● Menu:

1. Profile image
2. My availability:
   - Can select your availability by day and time so you do not get a request when you’re not available
3. Manage services:
   - Mechanics can see the available jobs in queue that they need to achieve.
   - Mechanics can cancel a job he had accepted.
   - The mechanics will get the request only for the job if they are online
4. Your Jobs:
   - Pending
   - complete
5. driver rating:
   - Can see the user rating for every job completed.

USECASE DIAGRAM



5.2. Reflective Journals

6.0 Month: October

What?
Reflect on what has happened in your project this month?
Since the beginning of this college semester, I was tasked with coming up with an idea for my computing project module, I plan to create an application for the motor repair industry which I have outlined in the project pitch video which I received validation about the idea from Frances but she outlined that the idea was fine, but I needed to add some more innovative functionality to the application idea. I've uploaded my project proposal today and hopefully will gain some more feedback on my idea and will help guide me down the right path.

So What?
From creating the project proposal, it gave me a lot of insight into my project and what were the different areas that I needed to research and what timelines and different milestones I need to meet.
I need to now begin the process of creating my application which will start by carrying out more research or competitors and the type of market which I will target. I then plan to create mock-ups of the user interface of the application which will further lead me to the development process of the application. I feel that this work should take me to the end of next month.

Now What?

To address these challenges, I plan to begin work on the research right away and further expand my knowledge of the motor car repair industry, I plan to do use mind mapping here to gain in proper understand of what I need to carry out.
I also need to research the area of UI design as it would be something that would be one of my strong points. I plan to research this by viewing some videos on YouTube along with carrying out some practical time also.

Month: November

What?
Following on from October's report, I worked towards completing my midpoint presentation where I need to decide the languages that I was going to be using along with the technology's used also, I carried out some research on this to gain some understanding. I needed to create a project plan and complete requirement gathering for the application in order to prepare a demo for the application.

So What?

I learned that I needed to gain some understanding of the different tasks I had to complete, I decided to code my application using react native. I will need to create a project plan specific to my college timetable and create requirements spec for my application so that I can prepare the Project Demonstration for the midpoint. I feel that I've gained a lot of understanding about the different technology's that are used but I would my biggest challenge been react native.
Now What? I will need to gather the requirements for the application, I will complete a use case diagram and list of the different requirements spec. I need to complete course on react native online and maybe watch some videos on YouTube to gain me a better understanding.

Month: December

What? I submitted my midpoint presentation on December 20th. I found it was a busy month as I had other projects to complete for Ruby on rails which was very time consuming in learning a complete language. I found that it was difficult to fit in time for this, but I got it completed. i struggled with completing the prototype as I didn't have enough experience with react native, so I decided to create a Prototype online using wireframes. I understand that this is far from ideal, but it was the best I could complete within this timeframe
So What? I feel that it was a good month and I felt that I learned a lot about the development process of the application. The requirements spec gave me a good understanding of the way the application should be structured. I feel that from completing the midpoint presentation it has given all the tools to go and code the application. The challenges that I feel remain are to gain a better understanding of react native and then implementing this into the application
Now What? In order to address these challenges, I will further study react native along with watching video courses on YouTube to help me gain a better understanding. Once I feel that I have enough knowledge, I will then start completing the coding of the project.

Month: January

What? For January I continued with learning about react native along with implementing the coding of my application, I experienced many errors with the application emulator, so I was forced to use Expo go environment. I completed the driver and mechanic log in pages for my application. I found some of the lectures helpful in gaining a better understanding about testing.
So What? Although the computing project module is difficult, I found that it gave me great experience in developing a real-life application and transformation current business processes. I successfully coded the opening page for my application. The challenges that yet remain are coding the Mapping page where drivers and mechanics can set out their location. I feel that this could be a challenge for me and will require some rigorous work.
Now What? To address these challenges will require some planning for the further development process, I found that reverting to the midpoint presentation document can be helpful to keep line of sight with the project. I found that there were various coding academy's online which were help for

adding different APIs into react native. I feel like I need to focus on completing the application from here on In and improve my skills as times goes on.

Month: February

What? For the month of February, I was fully concentrating on the development of my application, I integrated the React native mapping API into my application. This was a difficult task as I kept getting errors while trying to integrate. I also added the ''My Orders'' page which shows Pending, completed and rejected orders. I felt that I got a lot of work done on the application in February.
So What? I feel that I have completed most of the project Milestones in terms of development of the application. While the application is functioning, I'm still getting some errors when I run the application on my device so this is something I need to rectify so I can liase with my supervisor about the progress of my application. one challenge that I need to overcome is to include some new innovative features of the Application
Now What? I found that react native is difficult programming language to comprehend. I have found google and online forms to be great at narrowing down errors within my application so I will need to streamline that and fix all the errors within the application. In order to add some innovative features to my application at the request of my supervisor, I will carry out some research on existing application and try to see what they are missing and what would be beneficial to the customer.

Month: March

What? Reflect on what has happened in your project this month? This was busy month for me as I had 2 other CAs which were due for different modules which I found them to be very time consuming and my computing project module took a back seat at times. I successfully removed most of the errors within my application code and I feel the application has come together quite nicely. I have drafted some innovative ideas such as My Expenses, where customers can easily view service history of there vehicle which can be Downloaded on your device. This could be something that would be beneficial when selling your car.
So What? I feel that my application is within the final stages of development, The application is now running through all the developed pages which I feel is an achievement but I've still to work on some small errors with the application. One of the challenges that remains is adding some innovative features to my application.
Now What? I have drafted solution for innovative ideas I just need to integrate this into the application without affecting any of the existing code within the application and error free. I need to also keep in mind how useful these features will be to customers. I feel I'm on the home straight with my application and I can see a send in sight.

Month: April

What? Reflect on what has happened in your project this month? This was the final month of the project, so it meant that I was solely focused on completing the application as there some more errors that I was getting, and it was difficult to figure some of them out. As I felt my application needed some innovative features, I decided from recommendations from anu to implement a Chat feature on my application.
So What? I feel that my application is completed now, and the next task is to finish the technical document which requires some work with requirement and the design of the application. This is what I will be working on until the 15th of now
Now What? I will have to generate all the functional requirements, data requirements and user requirements of my application in the technical document. I will also have to complete the design of my application along with generating use cases

Signature: conor hughes