Demo Script

SmartXPlorers

Contents

Introduction

Good morning everyone. My name is ____, and my team members are _____. Our group name is Smart XPlorers.

Project Overview

Today, we will be presenting to you our web app called ProfitPath, which empowers users to both earn money through deliveries and carpooling and request these services as well.

Before we begin the live demo, let's briefly go through our main features and expected users.

Now, assuming that you have already registered and have reached the home screen. You will see 3 options, delivery, ride and earn.

Delivery is where a user can request to have something delivered. They can use the feature by entering the pick up and drop off locations, description of the item and lastly the amount they would like to pay for the delivery. They can then click on the request to send in a request which will be viewed by any driver near the pick up location. When their request is accepted the user will then be able to view the details of the deliverer and chat with them through the chat function, on the on route page. Lastly when the delivery has been made, they will receive a pop up to show that the delivery has been completed and will be able to rate and give feedback to the deliverer through our feedback function.

That brings us to our second feature, ride. This feature is very similar to the delivery feature, especially when having to request a ride.

Lastly we have our earn function, where users can enter their location to view the delivery and ride requests available. They can then choose to take up a request and click confirm to proceed with the request. Similar to the previous features, they can also use the chat function to communicate with the requesting users. After their ride is completed they can ride the completed button and proceed back to the home screen.

Expected Users

Expected users include users who are looking for an on-the-fly delivery service, users who want to quickly grab a ride and users who are looking to earn money by providing these services on the way to their main destination.

With that, we will now proceed to the live demo.

Live Demo

Registration

- <Begin from home screen>
- Click the profile icon to go to the login screen
- Click "Register"
- Show basic validation for username and email
    - Enter invalid email (without ".com")
    - Enter username
    - Enter all valid personal details + car details

Delivery

- Click the delivery feature
- Enter pickup and drop off location
- Enter price and description
- Click "Request"
    - Will show "Request submitted successfully"
    - Click "OK" to proceed
- Show the Awaiting Acceptance page
- Show the On Route page
- Show Ride has been completed pop up
- Submit feedback

Earn

- Click the earn feature
- Enter your location
    - Will show the request available
- Click on a request
- Click "Choose"
    - Will show "Ride confirmed"
    - Click "OK" to proceed
- Click "Confirm"

- Chat:
    - Type in something from the driver side
    - Receive reply from the riders side
- **Show video for live transmission**
- Click “Complete Ride”
- Will show “Ride completed”
- Click “OK” to proceed

Profile

- Click on the profile picture
- Click “Edit”
- Edit the phone number
- Click on My Wallet
    - Click on Add and enter an amount
    - Click “Confirm”
    - Show the changes in the available balance
    - Click on Withdraw and enter an amount
    - Click “Confirm”
    - Show the changes in the available balance
- Click on Car Details
    - Click “Edit”
    - Edit the model
- Click on History
    - Click on one of the request
        - Will show the details of the request
        - Click “Close”
    - Click on Requester
    - Click on Responder

Exception Handling

Now we will be going through some potential exceptions that may occur in this application.

Good SE practices

One of the software engineering practices we used is modular design, where we broke down our code into smaller, reusable components or modules.This makes the code more flexible, easier to maintain and test. For example, we had separate modules for user interface components and API calls. Each module was responsible for a specific aspect of the project, which made it easier to identify and fix bugs.

Secondly, API documentation. We also noted down detailed guidelines on API endpoints to allow easy integration between frontend and back end components.

Lastly, we used version control, by using GIt to collaborate and keep track of changes over time. Using version control allowed us to work on different features of the app independently, without worrying about conflicts or overwriting each other’s work, since there is a commit history to fall back on. This led to a more efficient development process and reduced the risk of introducing bugs or errors.

System Design

For this application, we have employed the model-view-controller approach towards design, where the model constitutes the django functions and the PostgreSQL database, the view refers to the HTML views which users can see and interact with and the controller is the logic implemented using JavaScript with React, which connects the HTML view and the model.

React and Django form the backbone of the frontend and backend respectively in a traditional server-client architecture. These components interface at the endpoints provided at the backend for retrieval of HTML elements, data, etc. for display to the user.

Traceability of a use case

We will focus on a particular use case, which you can refer to in the main document by the identifier, ID 7 - request a ride.
This use case is called upon whenever the user chooses the carpooling service. The prerequisite of user authentication must be fulfilled.
The flow is as follows: User chooses the service -> User inputs his/her pick-up and drop-off location -> User schedules a pick-up time (can be ASAP or otherwise) -> System registers the request that drivers can choose to accept.

However, if the request has not been accepted by any driver within some set period of time, the user may cancel the request, in which he/she will be redirected out of the service/page. The user may, alternatively, “refresh” the request.
Here, we performed some simple functionality testing, for example, for invalid cases such as missing fields, invalid scheduling times, etc.