Software Requirements Specification

for

ShopBlock

Version 1.5 approved

Prepared by Group 45 NTU 13/11/2024

Group Members (Group 45)	
Name	Matriculation No.
Kua Li Min	U2322513H
Hudzaifah Bin Muhammad Taufiq	U2320600F
Cheng Hui Wen	U2322123G
Ng Hoe Ping	U2321991F
Seow Jia Xian Jackson	U2322995F
Chua Ze Ming	U2321797B
Zhang Yuxuan	U2321475L

Table of Contents

Revision History

Name	Date	Reason For Changes	Version
Kua Li Min	24/9/24	Set up document, ported use cases and input information about ShopBlock	1.0

Hud & Jackson	26/9/24	Updated external interface requirements, system features, and non functional requirements	1.1
Ze Ming	27/9/24	Added API documentation and testing, backend and interfacing configurations and constraints, and updated external interface requirements	1.2
Hui Wen & Hoe Ping	29/9/24	Edited External interface requirements and system features	1.3
Yuxuan	1/11/24	Updated product perspective, information about databases, and diagrams in document	1.4
Hoe Ping	7/11/24	Added all entire test case (Black Box and White Box)	1.5

1. Introduction

1.1 Purpose

We want to create a web application to help the community with their daily necessities by looking for rental items or services. This can include but not limited to: rental items like, one-off items that you may not want to buy upfront, for example, power drills for DIY projects; and as a one size fits all service platform to source for services such as plumbing services or dog walking services.

We aim to help both parties, the renter and rentee, by creating a website where they can advertise their services or rental items. The website will also help manage their listings, and also help the filter and sort listing details. We want to create functions that can help minimise the friction for both parties.

1.2 Document Conventions

1. Priorities for high-level requirements are assumed to be inherited by detailed requirements unless stated otherwise.
2. Default font is size 12 Arial 1.5 line space
3. **Bolded** words represent headers

1.3 Intended Audience and Reading Suggestions

This paper is intended for those who are involved in the building and maintenance of ShopBlock. Among them are:

Developers: They should concentrate on the precise specifications and technological requirements when developing ShopBlock.

Project managers are responsible for overseeing the project's resources and timing. They should begin by outlining the project's goals, scope, and high-level overview.

Marketing Staff: The interface portions would be most relevant to them as they could be concerned with the application's goal and target audience.

Users: Renters and rentees, who are end users who will engage with the site can concentrate on the sections on functionality and user interface.

Testers: Carefully examining the functional and non-functional requirements and making sure the application is tested thoroughly.

Documentation Writers: These people should focus on emulating the conventions for consistency in future writings.

1.4 Product Scope

ShopBlock is for anyone who wants to rent out items or services or look for these items or services. We are dedicated to supporting low-income families, who may face challenges purchasing items upfront - by providing cheap rental solutions for short-term usage. It is also built to combat white elephant products, giving them a second life in multiple households.

We have added negotiable rates for flexibility, allowing users to pay/sell for what they feel comfortable. We also recognize that people do not like the inconvenience of travelling often or do not have the capability to do so frequently. Thus, we committed to transparency and clarity with listing locations in our system.

We can foresee a future where we strategically display non-intrusive advertisements in order to offset the cost and maintain the site, allowing it to serve its purpose free of charge. We might also collaborate with charity shops and host virtual rent parties to promote our goal for reducing waste while increasing outreach to the underprivileged. This way, the business can stay sustainable.

1.5 References

1. **Code Complete.** (2023, February 22). *Make a search bar with React (with API calls) | Beginners tutorial* [Video]. YouTube. https://www.youtube.com/watch?v=sWVgMcz8Q44
2. ***CodeSandbox Contributor.*** *(n.d.). ReactJS image slider example. CodeSandbox. https://codesandbox.io/p/sandbox/reactjs-image-slider-h628v?file=%2Fsrc%2Fcomponents%2Fcustom.slider.css*
3. ***MUI****. (n.d.). React data grid: Filtering. MUI X. Retrieved (2024, Nov 12), from https://mui.com/x/react-data-grid/filtering/*
4. ***ReqBin. (n.d.).*** *Online API testing tool. Retrieved November 12, 2024, from https://reqbin.com/*
5. ***Singapore Land Authority****. (n.d.). OneMap API documentation. OneMap. Retrieved November 12, 2024, from https://www.onemap.gov.sg/apidocs/*
6. ***The Web Dev.*** *(2023, June 22). How to use the `useeffect` hook in React | Beginners guide [Video]. YouTube. https://www.youtube.com/watch?v=D0mxApGK-EQ*

<List any other documents or Web addresses to which this SRS refers. These may include user interface style guides, contracts, standards, system requirements specifications, use case documents, or a vision and scope document. Provide enough information so that the reader could access a copy of each reference, including title, author, version number, date, and source or location.>

Author, A. A. (Year). Title of the article. Title of the Journal, volume number(issue number), page range. https://doi.org/xx.xxx/yyyy

2. Overall Description

2.1 Product Perspective

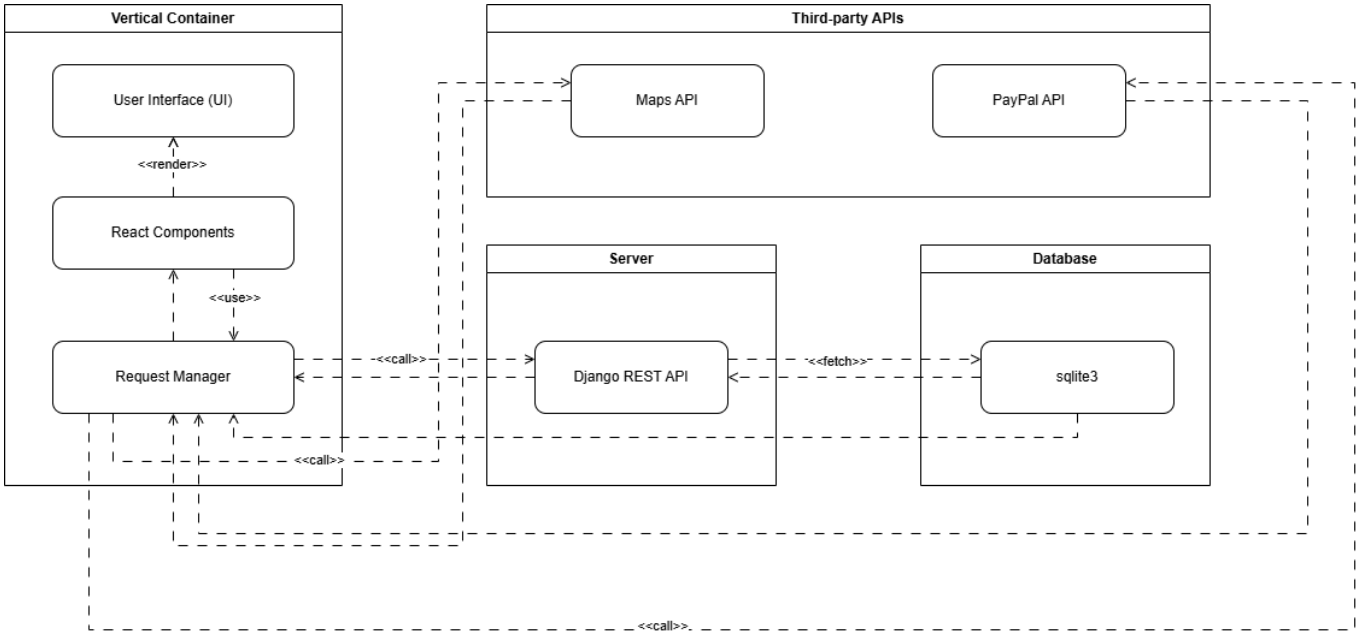
This web application is a new, self-contained platform designed to facilitate community-driven rentals and services, offering a space for users to either rent out items or provide services. The

platform is inspired by similar existing sites, such as Carousell and Facebook Marketplace, which enable users to sell items or services. However, unlike these platforms, this product focuses specifically on rental items (e.g., one-off tools or equipment) and service-based listings (e.g., plumbing or dog walking).

Unlike general online marketplaces that emphasise purchasing items, this product aims to address the needs of users who prefer short-term rental solutions or service-based exchanges. The system is built to minimise friction in the user experience by offering functions such as listing management, filtering, and sorting of items and services based on specific needs.

It also has a comprehensive bartering system which simplifies the whole process down to users submitting their offer price to a renter who would accept or reject it. This reduces the clutter which is in a lot of modern applications.

Relation to Similar Systems While platforms like Carousell and Facebook Marketplace allow users to sell, buy, or offer services, our product fills a gap by focusing specifically on rental-based exchanges and service offerings. This approach aims to create a more streamlined and efficient experience for both the renter and rentee.



2.2 Product Functions

- **User Account Management**
  - Log in, Sign Up, Reset Password, Change Password, Delete Account, Edit User Account Details
- **Listing**
  - Browse, View Listing, Create/Edit/Delete Listing, Filter Listing, Report Listing
- **Offer**
  - Make Offer, Accept/Reject Offer
- **Transaction**
  - View Transactions
- **Review**
  - Submit Review

2.3 User Classes and Characteristics

Rentee class

Aspect	**Description**
Frequency	High

Subset of product functions used	Browse Listings, Make Offer, View Transactions, Review
Technical expertise	Able to use typical shopping sites
Privilege level	Low
Characteristics	Looks for good deals, browses the site as a way to relax.

Renter class

Aspect	**Description**
Frequency	Medium
Subset of product functions used	Create/Edit/Delete Listing, Accept/Reject Offer
Privilege level	Able to use typical shopping sites
Privilege level	Low
Educational level	Finds an item or a service they would like to rent out on occasion, conscious about their waste

2.4 Operating Environment

The application can be used on any platform that has a browser. We do not use any specific technologies that are very modern. The entire application is just using the standard CRUD interfaces that have existed for many years. Any relatively modern device will be able to access it.

The development environment will be to use a relatively modern version of Node, as well as Python 3.10+. The instructions to set up both the front-end and the back-end will be in the README.md files of each folder.

2.5 Design and Implementation Constraints

Our application uses the OneMap API and the Paypal API. Both of them have client secrets that will expire in a few months. We will need to rotate these secrets in time.

We do not have any specific technologies that will limit the options to our developers. The developers can develop on both Windows and Mac, and Linux if they wish to. The technologies we chose are all very well supported, and battle-tested for all operating systems.

2.6 User Documentation

There will be a youtube video explaining our application and how to use it. There will also be OpenAPI backend documentation that will be included for the developers. This is included in Appendix D.

2.7 Assumptions and Dependencies

The main assumption is that the user will need to have an internet connection. Without the internet connection, the user will not be able to interface with our application.

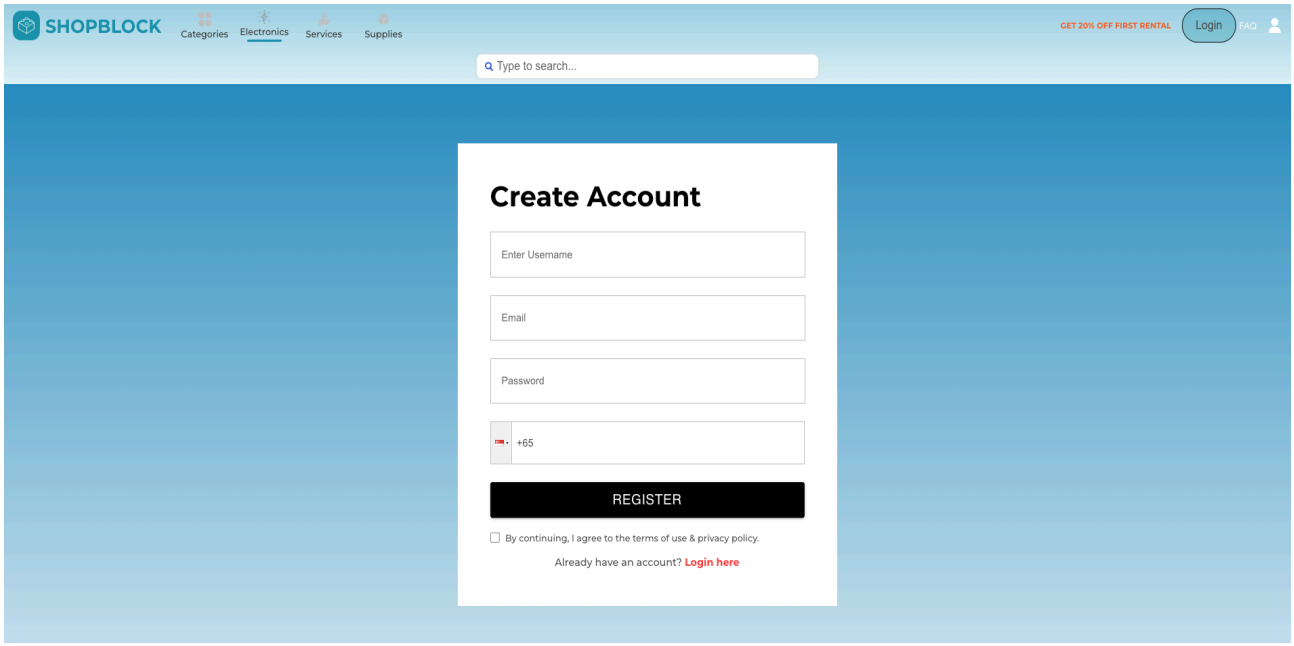
There are quite a few dependencies. The uptime of the application will depend on the cloud provider that we have chosen. Of course, it is best to choose one that has a high uptime, such as AWS or Digital Ocean. But we will be dependent on the reliability of these services. We will also be dependent on our maps and payment apis. If the OneMaps API were to go down, the users will not be able to see the listing locations. If PayPal were to go down, the users will not be able to pay. We do our best in choosing services that are well known such that we minimise the risk of this happening, but we will still be dependent on these services.

3. External Interface Requirements

3.1 User Interfaces

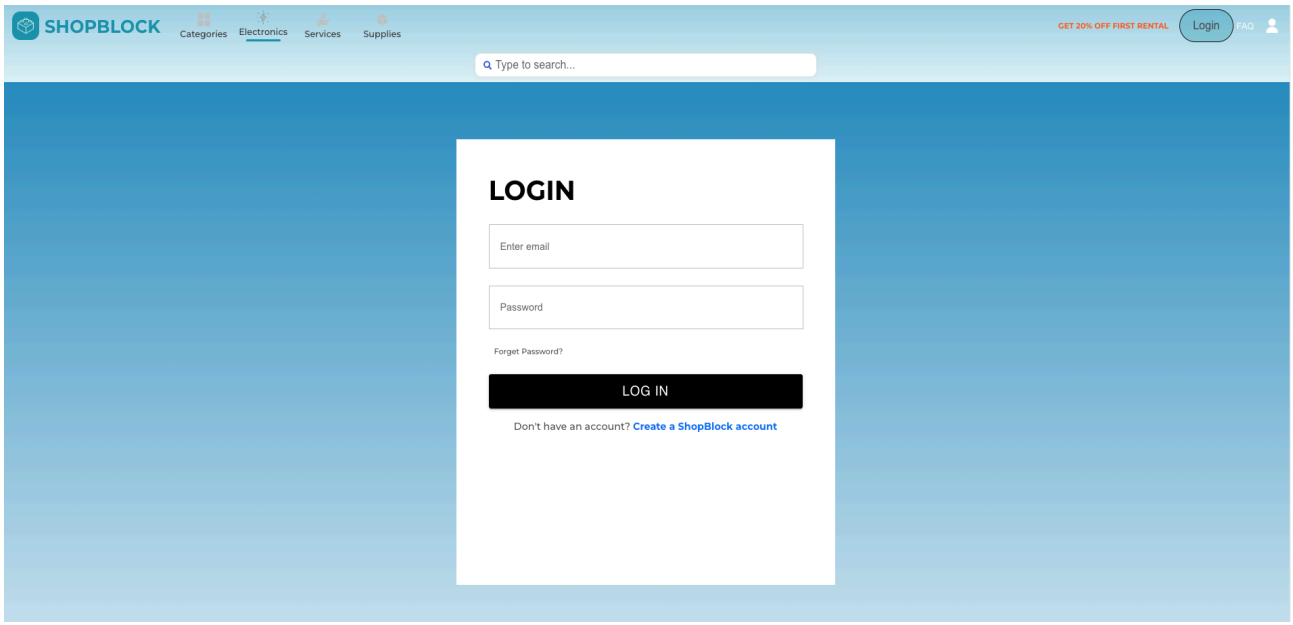
User Account Management

Create Account Page



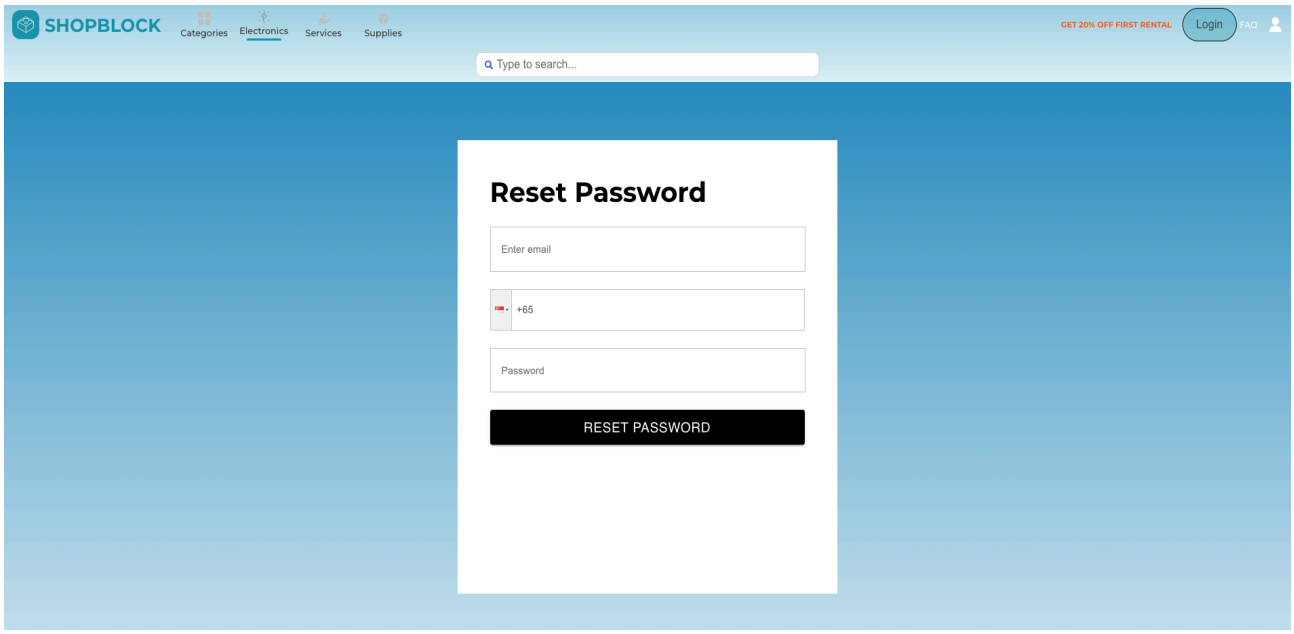
To get started, users will be prompted to create an account by providing their username, email, password, and phone number. The system will validate each field upon submission, displaying specific error messages for any discrepancies or unmet requirements. Only when all information is correctly entered will the form be successfully submitted, and the account created.

Login Page



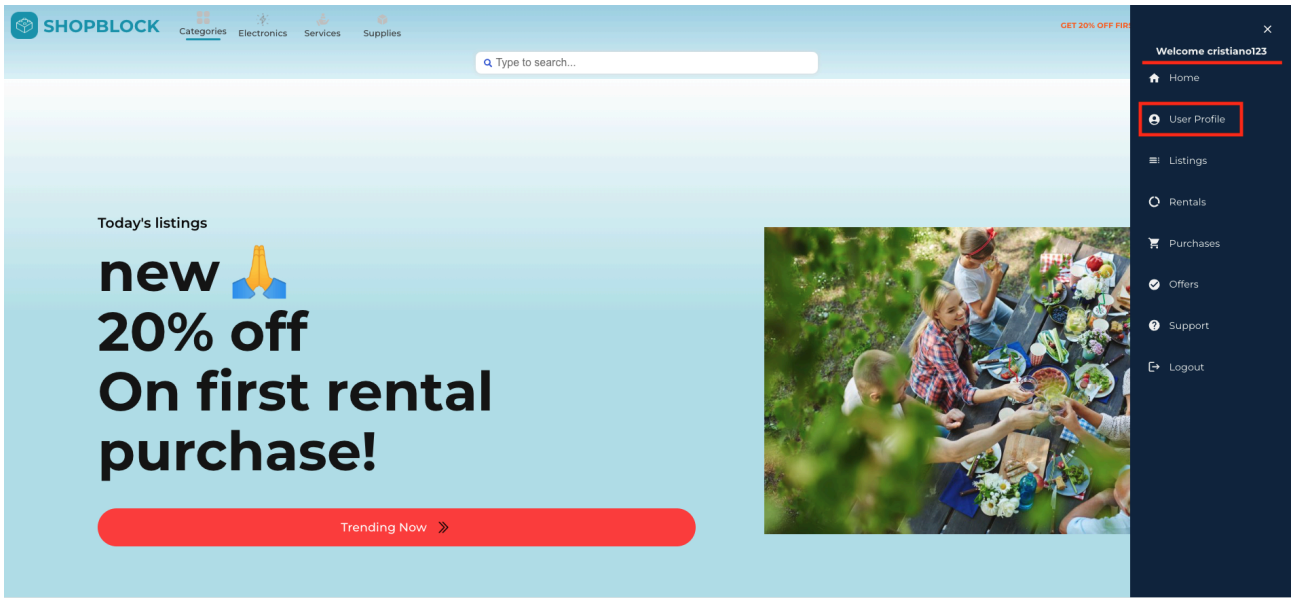
After creating an account, the user will be redirected to the login page. To access their account, they must enter both their email and password correctly. If the credentials are incorrect or do not exist, appropriate error messages will be displayed. Once valid details are provided, the login process will complete successfully. Alternatively, if the user has forgotten their password, they can click the "Forgot Password" link to be directed to the reset password page.

Reset Password Page



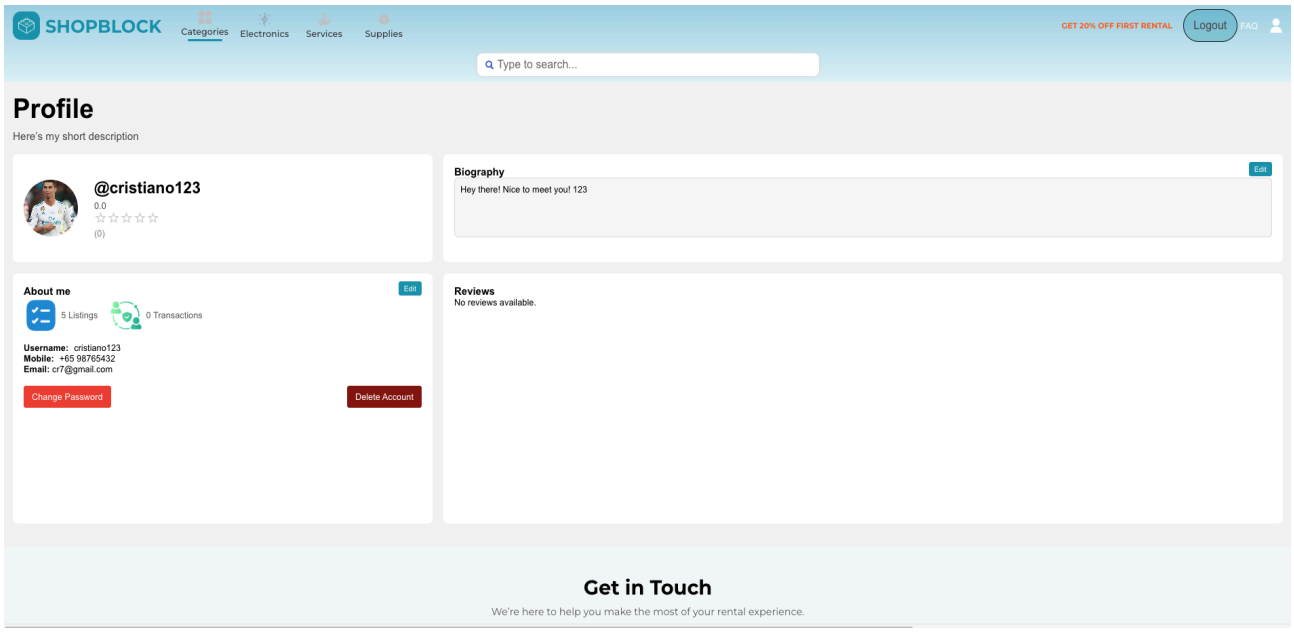
If the user has forgotten their password, they can reset it on the password reset page. Here, they will need to enter their registered email and phone number. If either detail is missing or incorrect, appropriate error messages will be displayed when they attempt to submit the form. Once both details are verified as correct, the user will then be able to input a new password, which will then become their account password.

User Profile Page



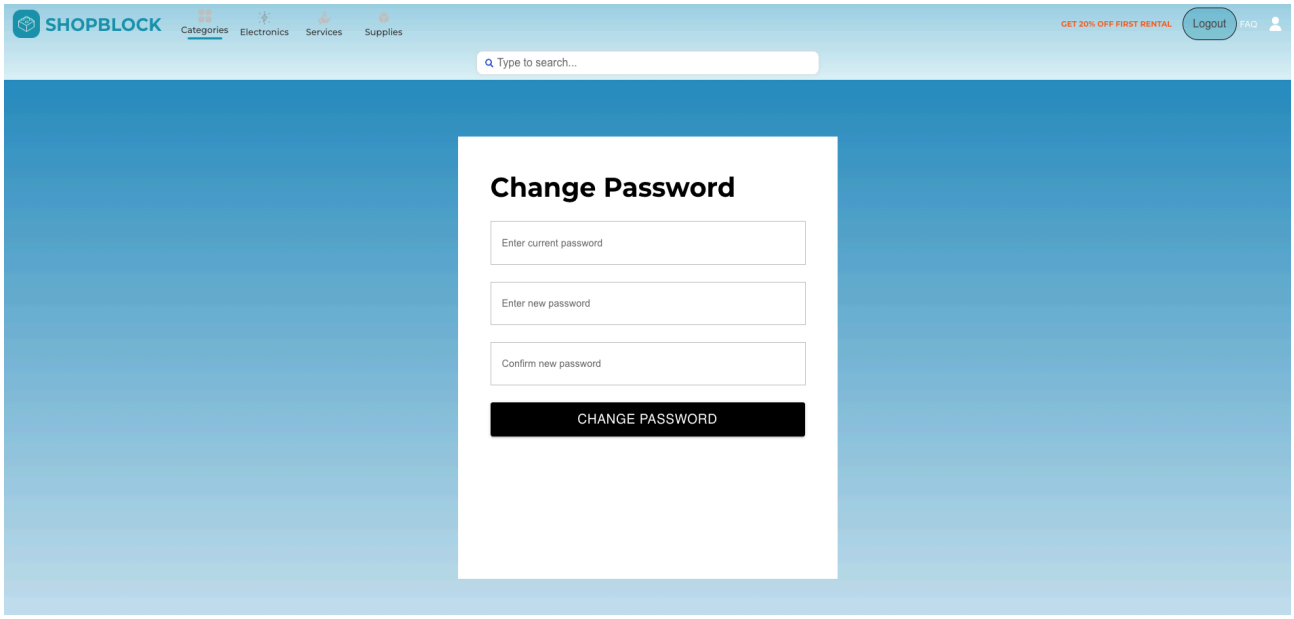
Once logged in, the user can access the sidebar by clicking the account icon in the top-right corner. The sidebar displays their username and provides a direct link to the user profile page.

Edit Account Details



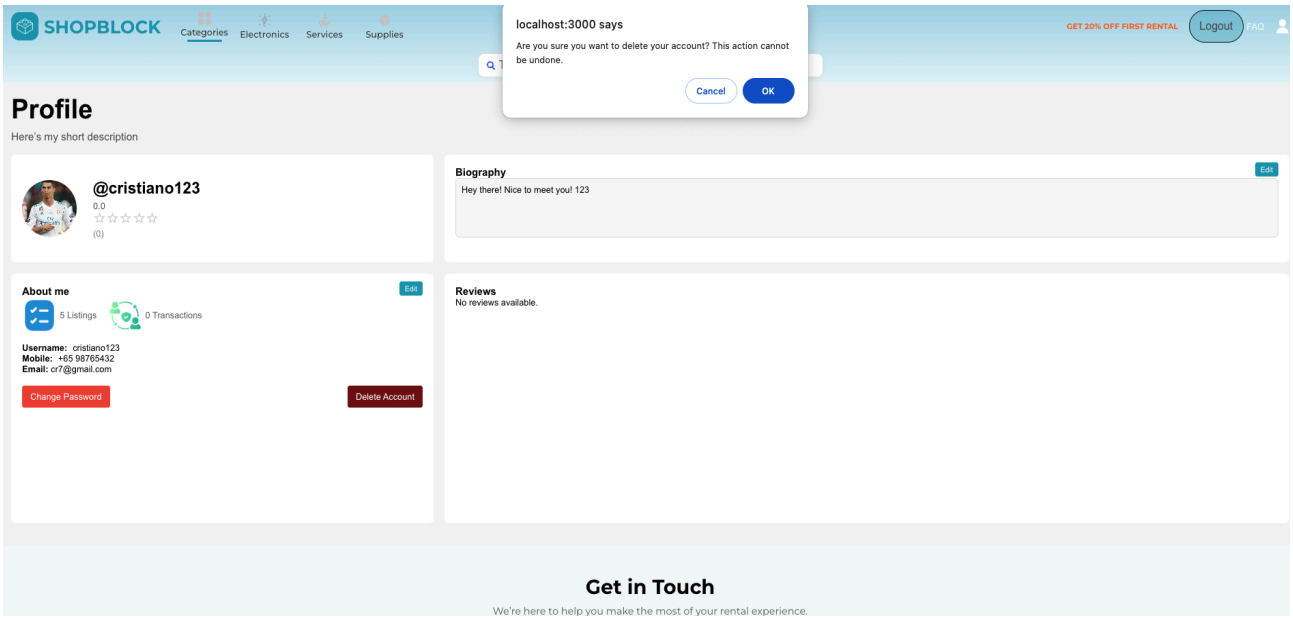
On the user profile page, users can view their account details, including username, average rating, biography, total listings and transactions, phone number, email, and past reviews. Additionally, users can edit specific account information and save changes to update their profile. The profile page also provides options for actions such as "Change Password," which navigates the user to a dedicated password change page, and "Delete Account," which permanently removes the user's account.

Change Password



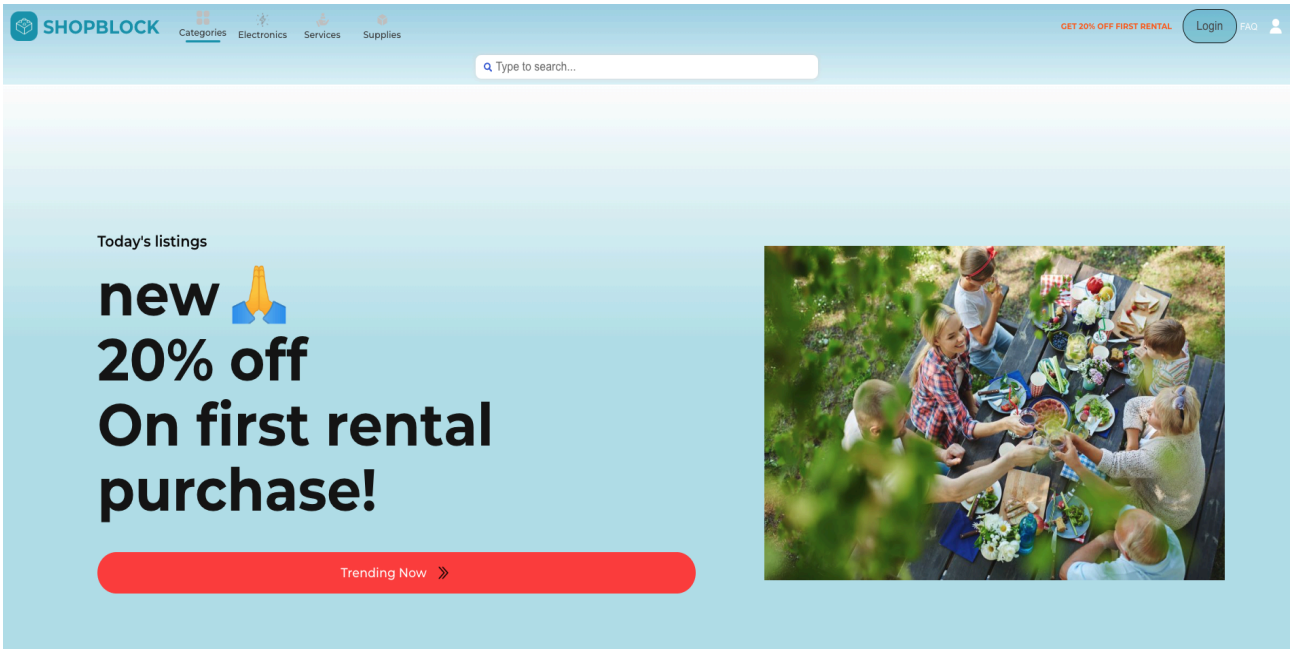
If the user selects the "Change Password" option on their profile page, they will be directed to the password change page. Here, they will be prompted to enter their current password, a new password, and a confirmation of the new password. If any information is entered incorrectly, appropriate error messages will be displayed. Once all information is correctly entered, the account password will be successfully updated.

Delete Account (1)



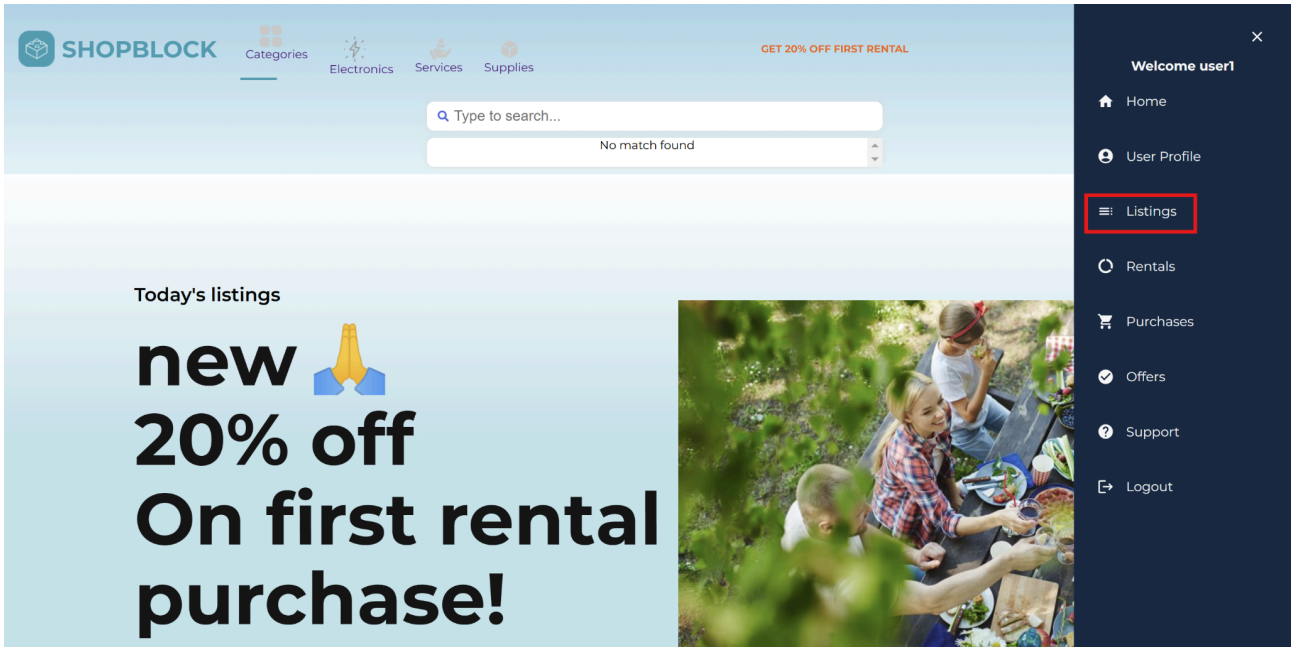
If the user clicks "Delete Account" on their profile page, a confirmation pop-up will appear, asking them to confirm the account deletion. If the user selects "OK," the account is permanently deleted from the database, and the system automatically logs the user out.

Delete Account (2)



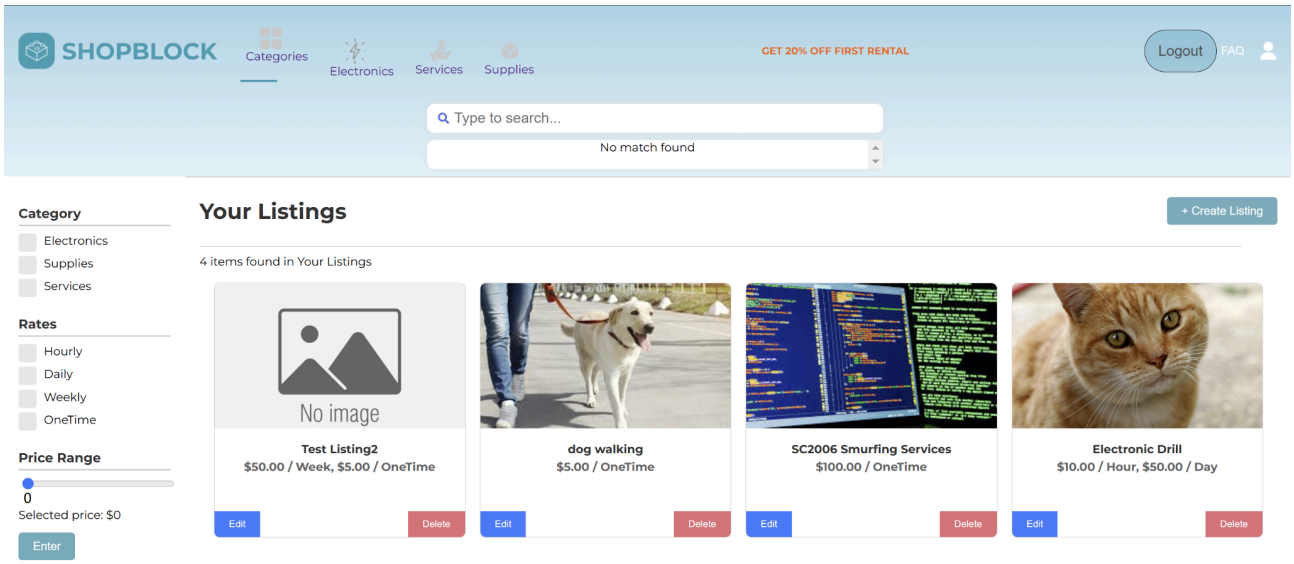
The user is logged out and can choose to create a new account or log in again.

Listing



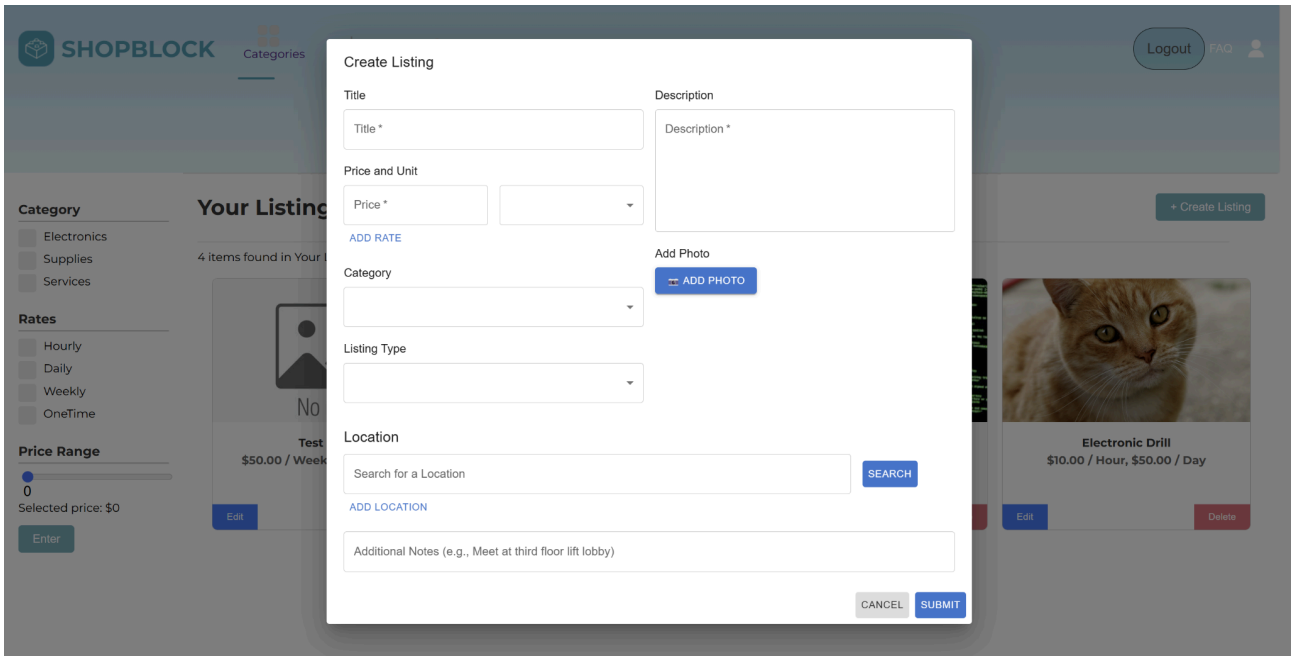
When the user is logged in, they will gain access to the navigation sidebar. There they can access their listings by clicking on the Listings tab.

User's Listing Page

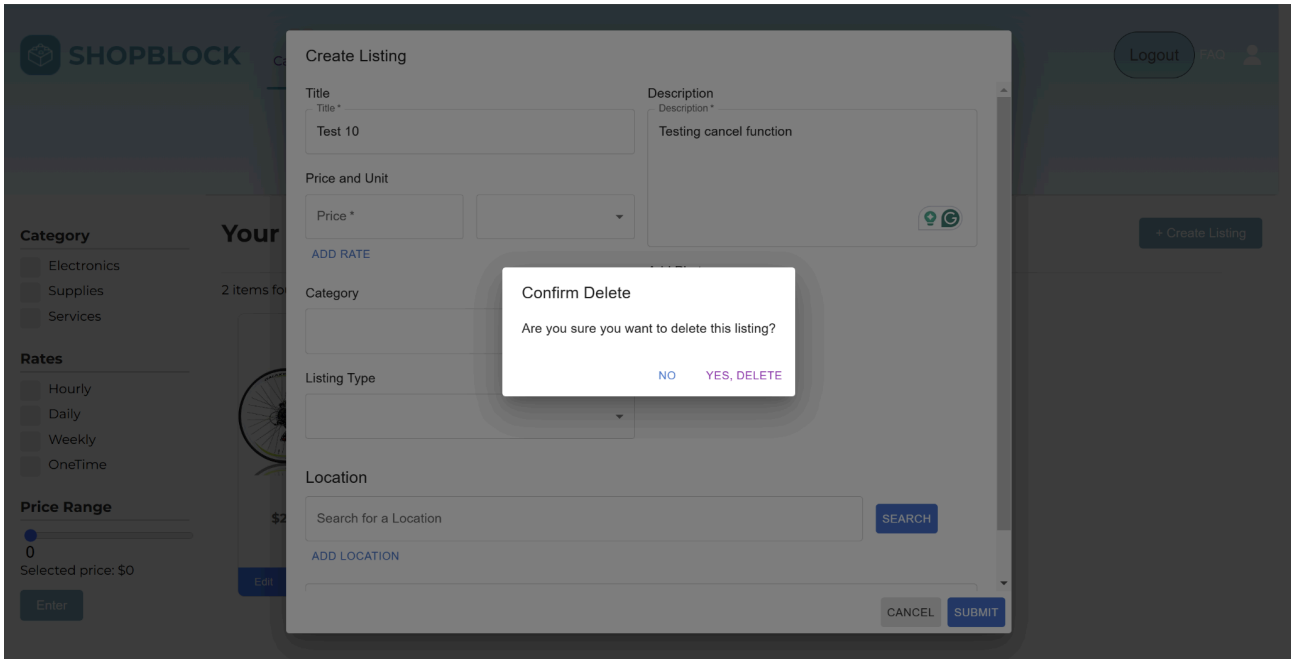


Upon clicking on Listings, they will be brought to the page above containing all the listings created by the user. There they will be able to manage their Listings and create more listings.

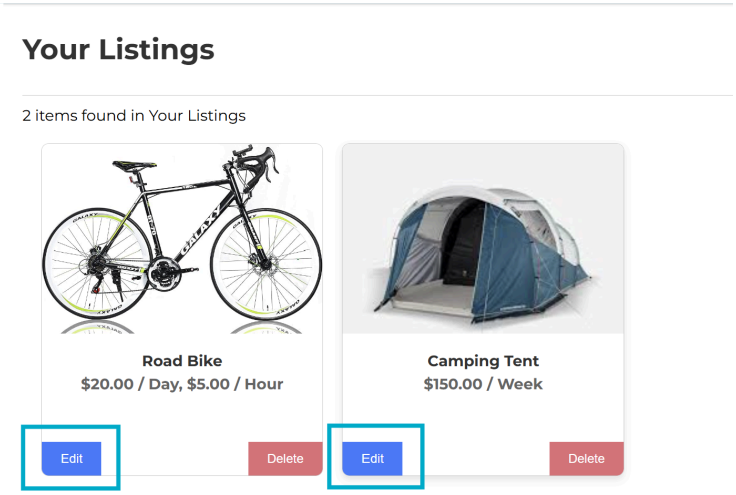
Create Listing

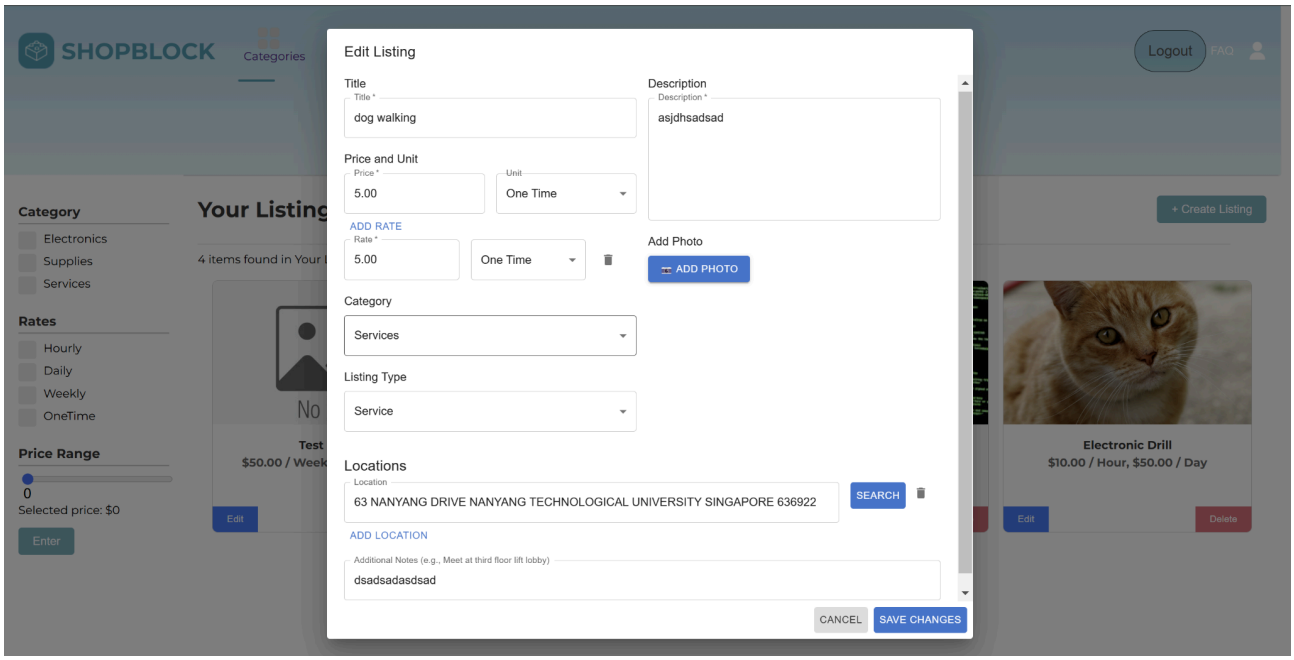


The user can create a new listing by clicking on the create Listing button at the top right corner of the page. There, they will be required to fill in all the fields and attach at least one image as well. They will not be allowed to submit the form if any of the fields are not filled in. If they try to submit the form with any of the fields not filled in, a dialog would appear informing the user to fill in all the fields. They may enter multiple rates or locations.



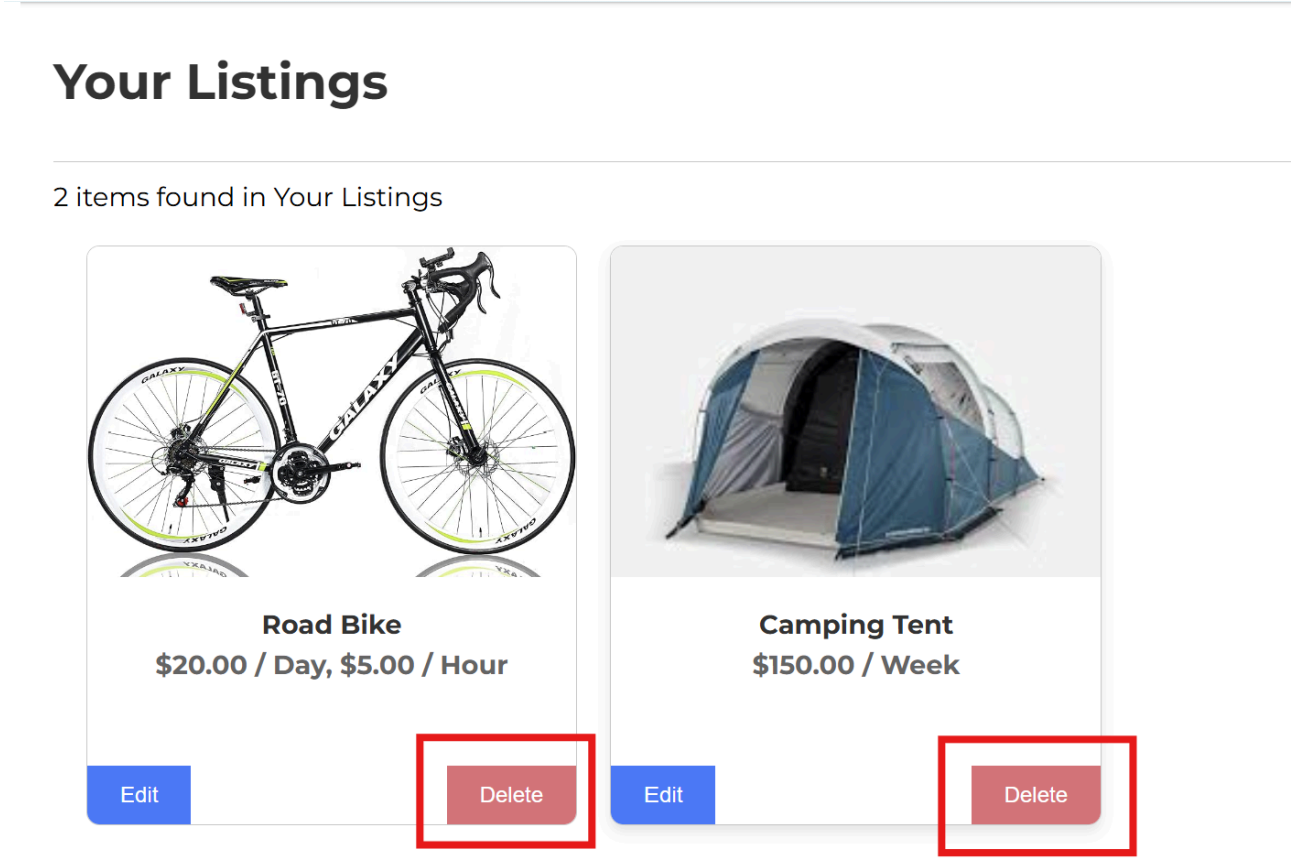
Edit Listing

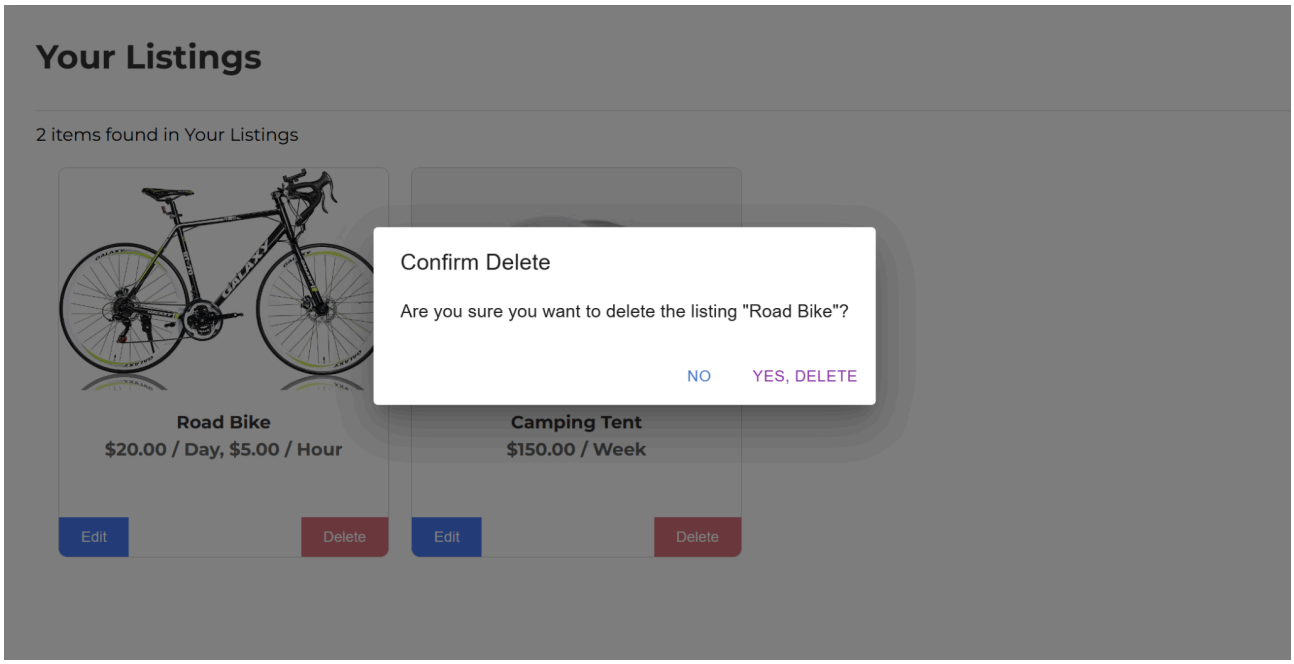




The user can choose to edit the Listings they created by clicking on the edit button in the listing card. They will be presented with a similar interface to create listing where they can make edits to the listing details.

Delete Listing

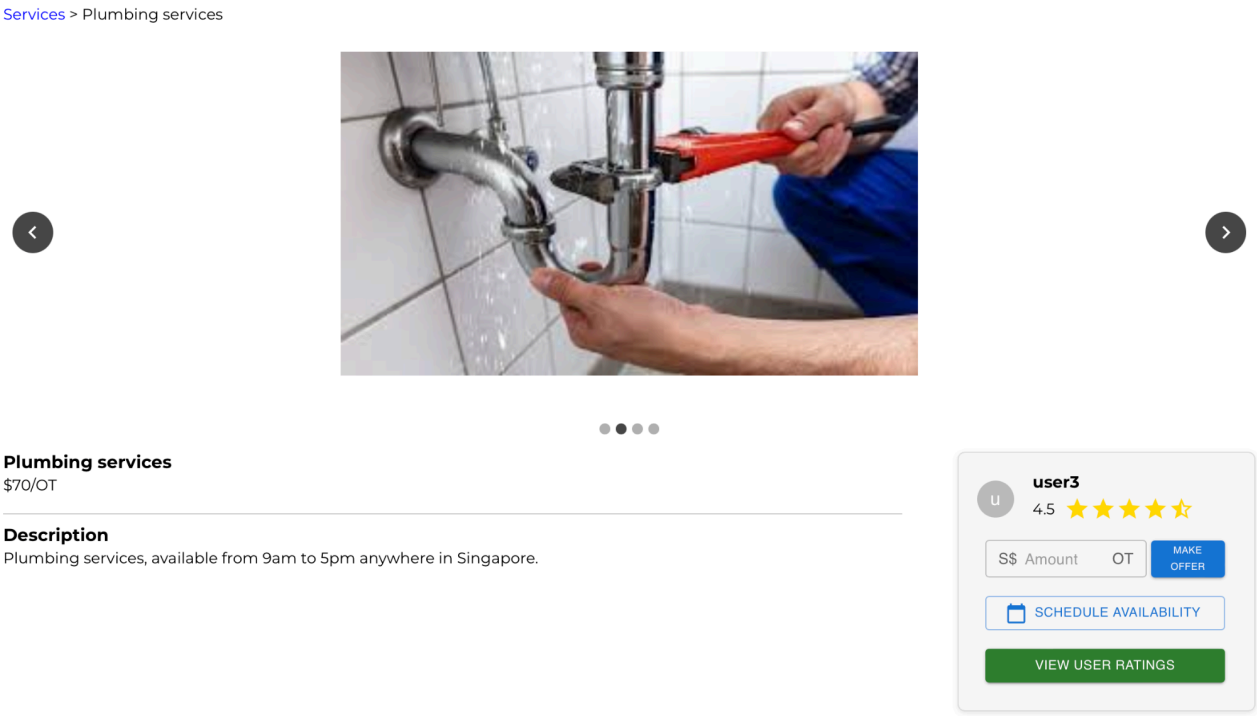




A user can also choose to delete a listing they created by clicking on the delete button at the bottom right of each listing. The user will receive a confirmation prompt in which they will have to click ok to delete the listing.

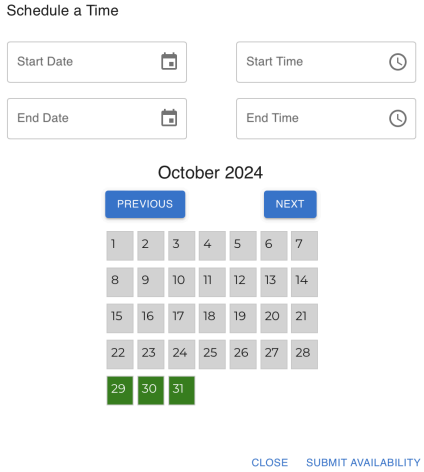
Offer

Make Offer (Renter's Perspective)

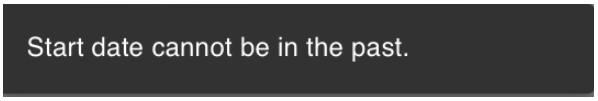
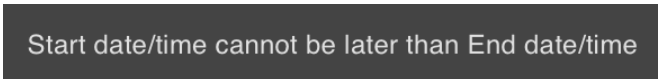


After a user has selected a listing of choice from the browsing page, the user can make an offer to the renter on the listing itself. Two inputs are required in this section to successfully make an offer: "Availability" and "Amount".

To set "Availability" of the user making the offer, they can simply click the "Schedule Availability button", which will bring up the UI as shown on the right. From here, the user is able to see which dates are available, and which dates are unavailable to be chosen.



There will be testing conditions to ensure the user has selected an appropriate date and time period, to ensure it maintains time continuity and avoids scheduling conflicts. Listed below are some conditions that prevent availability from being set due to such conflicts

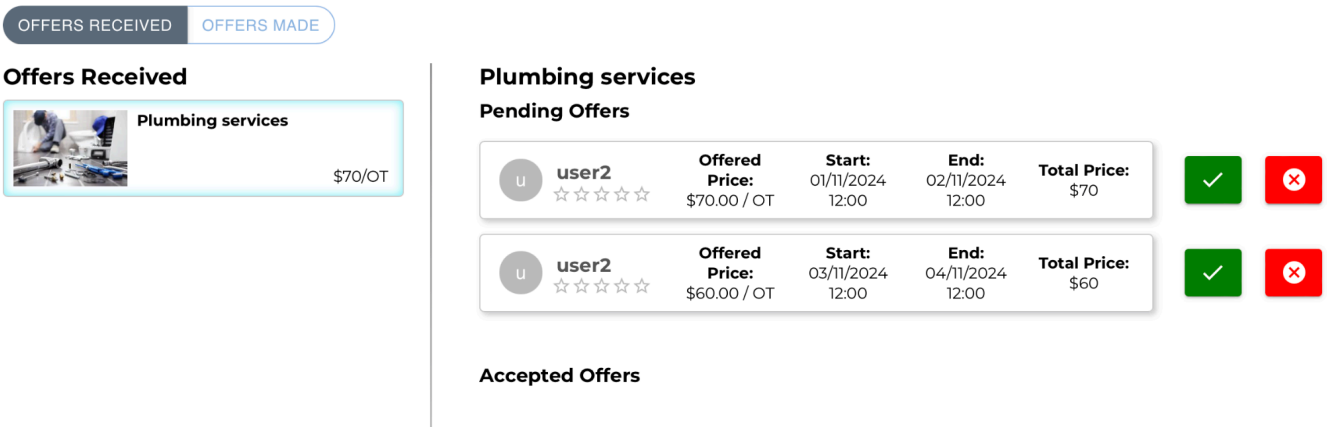
If the starting date is located in the past, there will be an error in setting dates.	
Either start date or start time can not be later than end date or end time.	
If the user forgets to set availability, there will be a warning given.	
If the user forgets to set an offered amount, there will be a warning given	

To set the amount, the user simply has to enter any desired amount to pay per time unit.

When the user successfully input a desired amount to pay, and has set an availability; clicking on the “make offer” button will successfully generate the prompt as shown below.

(E.g., “Amount” = $60)

Accept/Decline Offer (Renter's Perspective)

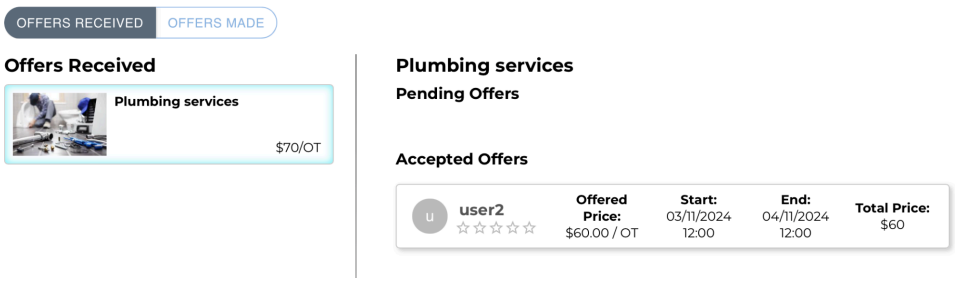


In the "Offers" page, under "Offers received", the owner of the listing, the renter, is able to view offers that are offered to them. They are able to view listings that have been previously or currently offered (In this case, only plumbing services have offers). When the listings are clicked, it shows pending offers and accepted offers that they have accepted previously.

The offer includes details such as: rentee's username and ratings, offered price, start and end date/time, as well as total price after calculating the time unit multiplied by time delta. This information will allow the listing owner to make an informed decision to accept or decline the offer. The original price is also shown on the card of the image for comparison purposes.

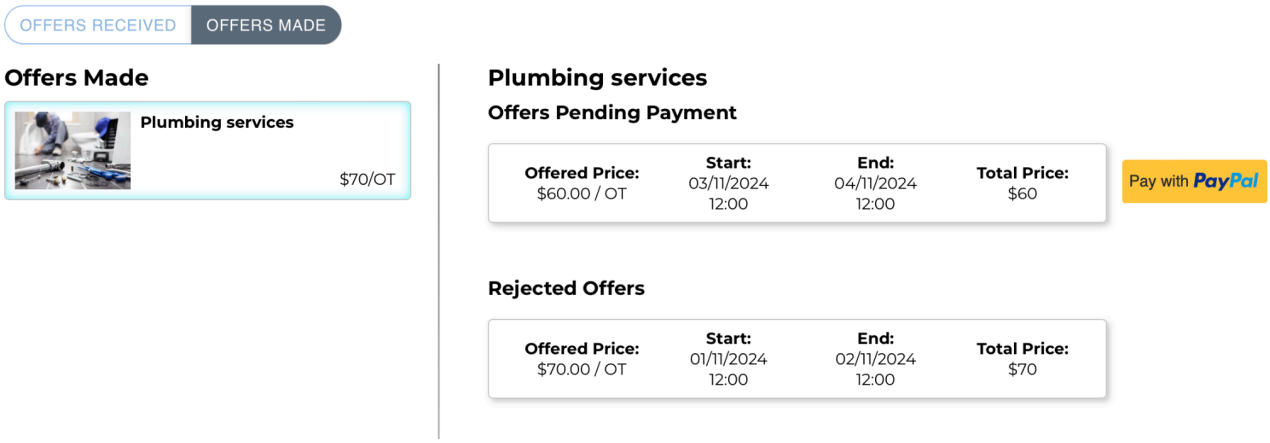
To accept, the user simply has to click the "Accept" button as shown in a tick format.
To decline, the user simply has to click the "Decline" button as shown in a cross format.

The image below shows what the page would look like, if we reject the first offer, and accept the second offer as illustrated in the image above.



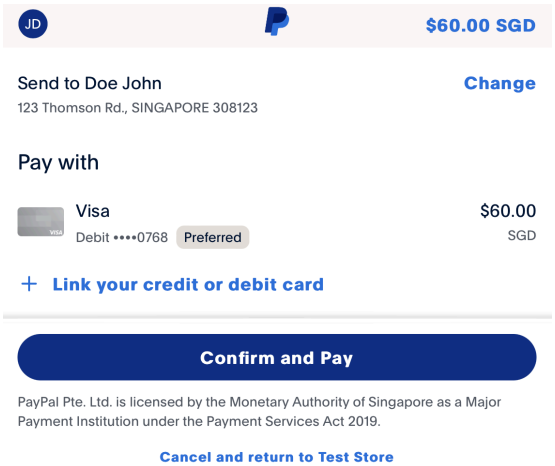
Payment

Paypal (Rentee's perspective)

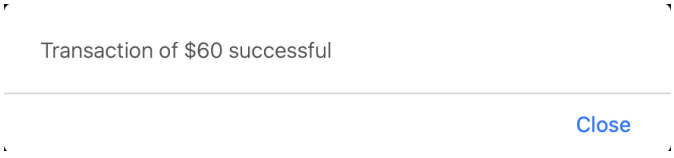


In the "Offers" page, under "Offers Made", from the perspective of the user that has made the offer, they will be able to view the outcome of the offer they made on the listing.(Whether it has been rejected or accepted).
If the offer has been accepted by the owner of the listing, it would show under the "Offers Pending Payment" section, which brings them paypal for payment.
If the offer has been declined by the owner of the listing, it would show under the "Rejected Offers" section, to update them on the outcome of the offer.

To make payment, the user simply has to click on the "Paypal" button, which will redirect them to the paypal login page. After logging in with the correct credentials, they will be asked to make payment as shown in the image on the right.

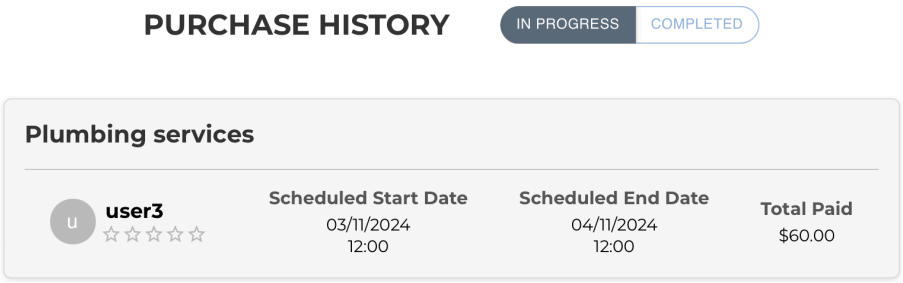


Upon successful payment, a prompt showing successful transaction will be shown as seen on the image on the right. The offer that has been paid will then subsequently be removed from the "Offers Pending Payment" section.



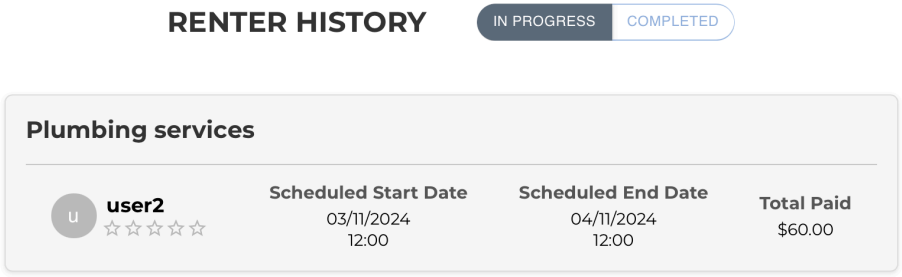
Transaction

Purchases History (Rentee's Perspective)



Under the "Purchases" page, it will show a list of all transactions the rentee has purchased and paid. Upon successful payment, the "offer" will be termed as a "transaction" to be fulfilled by both parties. It will list the schedule start date, and schedule end date for the transaction to be fulfilled.
"In progress" shows transactions that have yet to start, or have already started.
"Completed" shows transactions that have already been completed.

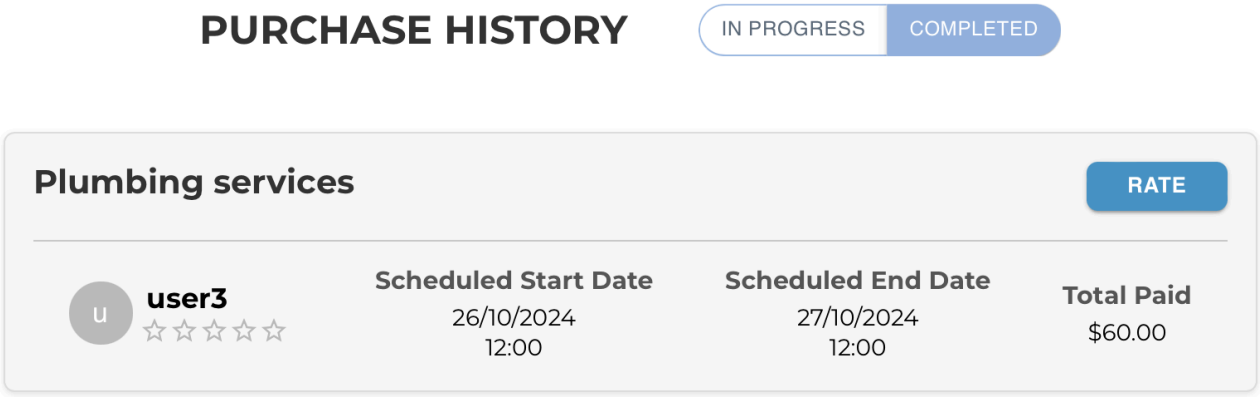
Purchases History (Renter's Perspective)



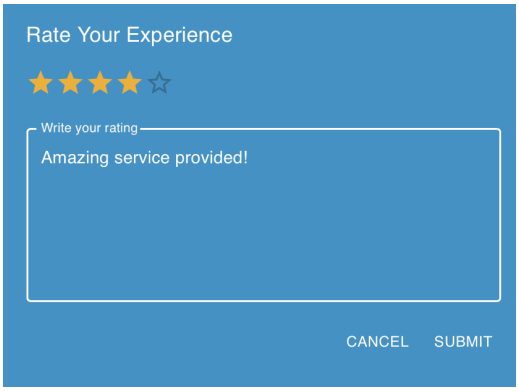
In the "Rentals" page, likewise, it will show a list of all transactions that a renter has to fulfil after a rentee has paid on the offer that has been accepted. It will list the schedule start date, and schedule end date for the transaction to be fulfilled.
"In progress" shows transactions that have yet to start, or have already started.
"Completed" shows transactions that have already been completed.

Review

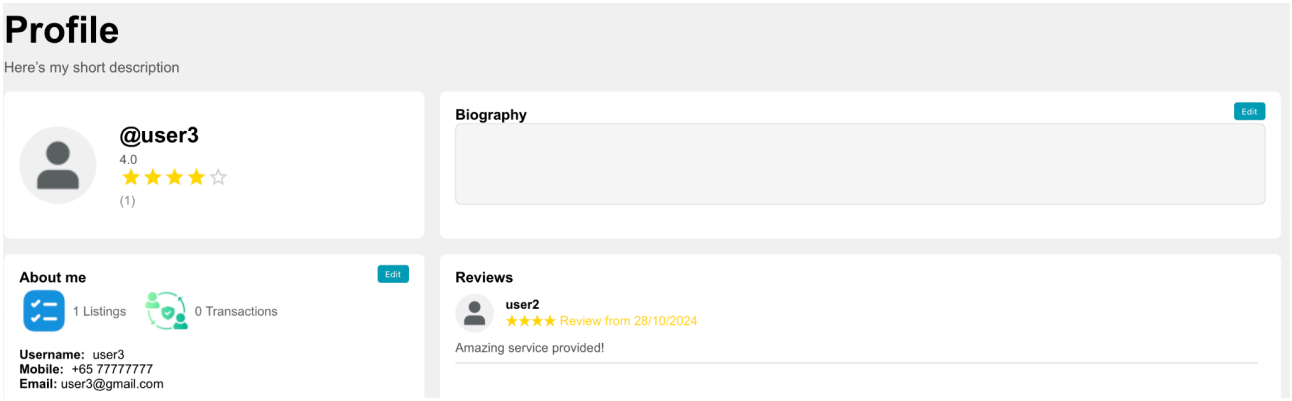
Transaction Review



After a transaction has been completed, the transaction will be moved from “In Progress” to “Completed”. This completion also allows the rentee to rate his/her experience with the transaction by clicking the “Rate” button. The image on the right shows the dialog that will appear after clicking on the “Rate” button. All inputs (Star rating and description) are required to be filled for the review to be successfully submitted.



Review reflected on renter’s profile



After a transaction has been completed, the rentee can choose to leave a review on the renter’s page. This is reflected on their profile page that can be viewed by all users. By doing so allows the renter to build credibility and legitimacy, leading to more offers.

3.2 Hardware Interfaces

Requirement	**Descriptions**
Operating System with a browser as well as a pointer and a keyboard	Any operating system that is able to run a browser for the front-end. A pointer and a keyboard to access the interface with the website and fill in the details.
A mobile phone with a browser	A mobile phone will also be able to access the website and use all of its functionality.

3.3 Software Interfaces

Since our program is split into the front-end and the back-end. We will list down the software interfaces for each separately.

3.3.1 Frontend Software Interfaces

React

We use React as it is a JavaScript library for building fast, interactive user interfaces. It lets us create reusable components and efficiently updates only parts of the UI that change. Components created as a separate .jsx file can be imported and implemented in any other .jsx file efficiently. React also offers hooks such as useEffect and useState that allow us to manage data from APIs, which allows us to fetch from backend to display on the UI.

Material UI

Material UI is a popular React component library that provides pre-designed and customizable UI components. It accelerates frontend development by providing components such as buttons, sliders, snackbar, which can be quickly implemented. This allows us to quickly create a professional look without needing to design everything from scratch.

PayPal API

The PayPal API provides a secure way to integrate payment processing on the frontend by connecting the app to PayPal's payment processing system, to allow transacting accepted offers. It allows payment without storing their private payment details by accessing a reputable payment gateway, allowing for hassle-free and smooth transactions.

OneMaps API

The OneMaps API enables location-based services in the frontend, allowing users to visualise locations on the map by the exact location through pins on the map. It also allows additional description through interaction of the pins. Thus, simplifying the process and providing hassle-free experience without having to search on where the items can be collected without having to search the location online.

3.3.2 Backend Software Interfaces

3.3.2.1 Frameworks and Libraries

- Framework: Django + Django-Rest + Django-Spectacular + Django-SimpleJWT
- Tooling: Mypy

In our application, we chose to use Django as the supporting backend, as there were many existing libraries that can support what we need from the application. Django by itself also comes with a lot of built in features that we can use.

One specific one is the Object Relational Mapper (ORM) that comes with Django. This is what we used to write our models and also map it to our database that will be mentioned below. This helps us keep our models (in code) in sync with our table relations. Django will also help us automatically generate database migrations. This helps us maintain backward compatibility.

We use Django-Rest to help us map our controllers to our endpoints. This makes it so that we limit our backend to only serving APIs, and media files. This fits with our needs, so we have a clean interface that our react front-end can use to interface with.

Django-Spectacular is what we use to follow the OpenAPI standards and to automatically generate documentation. In this case, the backend will automatically look at the models, and endpoints, and automatically generate the respective OpenAPI format. It will then host the swagger documentation on one of our routes. This ensures that we have up to date documentation at all times, and we are able to test the end points independently.

Since this application also requires authentication, we decided to use Json Web Tokens to help implement that. Django-SimpleJWT is a package that helps us generate JWTs with certain time outs and map them to specific routes. We can then set it such that certain routes can only be accessed by people with a valid token.

Since python is untyped, there can naturally be errors that come up when we are developing. We use mypy to introduce a stronger type checker that enforces our types to be correct. This is important as our ORM models are what we will be dealing with the most. With mypy, this helps us guarantee that what we are accessing and requiring from these models, will be there when we need them.

3.3.2.2 Database

In our application, the database implementation is up to the developers. Since we are using the Django ORM to interface between all database interactions, we do not have to worry about which database to use. Any database will work.

We are able to use PostgreSQL, MySQL, MSSQL and all other mainstream databases without issue. For ease of development, we have been using SQLite. But there are no implementation changes if we were to move databases. This can be a further consideration for scalability, if it reaches a state that SQLite is unable to cope with the demand, we can easily scale by switching to a database that can deal with more connections.

3.4 Communications Interfaces

An internet connection will be required for the application to function. The user will have to use their browser, to access our website, which will then be used to fetch data using the API from the backend. The communication protocol will be the HTTP protocols. HTTP GET, POST, PUT, DELETE specifically. Along with the HTTP protocols, we also use Json Web Token (JWT) authentication. Certain routes are only available to users that are logged in. This JWT will serve as the identifier for a user that has logged in with the right credentials.

4. System Features

4.1 Stimulus/Response Stimulus Use Cases

User Account Management System

Use Case ID:	**USER_UC_1**		
Use Case Name:	**Register**		
Created By:	**Ng Hoe Ping**	Last Updated By:	**Ng Hoe Ping**
Date Created:	**1/9/2024**	Date Last Updated:	**25/10/2024**

Actor:	**User**
Description:	Allows a new user to create and register an account.
Preconditions:	**1. User must not be an existing registered user in the system.** **2. User must have the necessary personal information to create an account.**
Postconditions:	**1. Account details are stored in the database to be retrieved when prompted**
Priority:	
Frequency of Use:	**1**
Flow of Events:	**1. User inputs required information: Username, Email, Password and Phone Number.** **2. System validates the information with existing information in the database.** **3. System creates the account if no conflicting email/phone number is found.** **4. System updates account information in database.** 5. **System directs the user to the login page.**
Alternative Flows:	**AF-S1-a. User account already exists: Phone number taken.** **1. System displays "An account with this phone number exists!"** **2. System prompts user to "Create New Account", "Log In" or "Forget Password"** **3. If "Create New Account", then continue from main flow step 1.** **4. If "Log In", then system directs to log in page to continue from Login (USER_UC_2)** **5. If "Forget Password", then system will direct to "forget password page" (USER_UC_3)** **AF-S1-b. Username does not meet requirements**

	1. System displays "Username must be at least 4 characters long." **2. System prompts user to input a longer username.** **3. Continue from Main Flow Step 1.** **AF-S1-c. Email does not meet requirements** **1. System displays "Please enter a valid email address"** **2. System prompts user to input a valid email address that contains an "@" symbol and ".".** **3. Continue from Main Flow Step 1.** **AF-S1-d. Password does not meet requirements** **1. System displays "Password does not meet requirements!"** **2. System displays "Please input a password of at least 1 special character, mixed case and at least 8 characters long" message.** **3. Continue from Main Flow Step 1.**
Exceptions:	
Includes:	
Extends:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

Use Case ID:	**USER_UC_2**		
Use Case Name:	**Login**		
Created By:	**Ng Hoe Ping**	Last Updated By:	**Ng Hoe Ping**
Date Created:	**1/9/2024**	Date Last Updated:	**25/10/2024**

Actor:	**User**
Description:	Allows an existing registered user to log in to the website with their personal information.
Preconditions:	**1. User must be an existing registered user in the database.** **2. User must have the necessary personal information to log in to the account.**
Postconditions:	
Priority:	
Frequency of Use:	**3**
Flow of Events:	**1. User inputs required information: Email and Password.** **2. System validates the information with existing information in the database.** **3. System logs in to the account if the inputted information matches the database.** **4. System directs the user to the main page.**
Alternative Flows:	**AF-S1-a. Email not found** **1. System displays "No account found with this email address."** **2. System prompts user to input a valid email address.** **3. Continue from Main Flow Step 1.** **AF-S1-b. Email found but incorrect password** **1. System displays "Incorrect Password! Please try again"** **2. System prompts the user to input the password again.** **3. Continue from Main Flow Step 1.** **4. If User chooses to "Forget Password" User will be directed to the "Forget Password" (USER_UC_3) Page to reset their password** **AF-S1-c. Ignoring the "Forget Password" prompt** **1. System displays "Incorrect Password! Please try again"**

	2. System prompts the user to input the password again. **3. Continue to Main Flow Step 1.** **4. After three unsuccessful attempts, the system prompts the user with "Your account will be locked after two more attempts. Forget password?".** **5. If User chooses to ignore the prompt, the user will be allowed to re-attempt to log in to the account.** **6. If User successfully input the correct corresponding password, the user will be directed to the main page.** **7. If User has two more failed attempts (total of 5 unsuccessful tries), the user will be locked out from the account.** **8. The system prompts the user with "Your account has been temporarily locked for 5 mins. Please try again later or reset your password".** **9. If User chooses to "Forget Password" User will be directed to the "Forget Password" (USER_UC_3) Page to reset their password** **10. User can choose to try again later in 5 mins, continue from Main Flow Step 1**
Exceptions:	
Includes:	
Extends:	Forget Password **(USER_UC_3)**
Special Requirements:	
Assumptions:	
Notes and Issues:	

Use Case ID:	**USER_UC_3**		
Use Case Name:	**Forget Password**		
Created By:	**Ng Hoe Ping**	Last Updated By:	**Ng Hoe Ping**
Date Created:	**1/9/2024**	Date Last Updated:	**25/10/2024**

Actor:	**User**
Description:	Allows an existing registered user to change/edit their existing password in the database.
Preconditions:	**1. User must be an existing registered user in the database.** **2. User must have the corresponding email to the registered account.**
Postconditions:	**1. Account details are stored in the database to be retrieved when prompted**
Priority:	
Frequency of Use:	**1**
Flow of Events:	**1. User inputs required information: Username and Phone Number** **2. System validates the information with existing information in the database.** **3. System prompts user to input new password.** **4. User successfully resets password.** **5. User is directed to the Login page (USER_UC_1)**
Alternative Flows:	**AF-S1-a. Password does not meet requirements.** **1. Starting from Main Flow Step 3.** **2. System displays "Please input a password of at least 1 special character, mixed case and at least 8 characters long" message.** **3. User inputs valid password and is direct to the Login Page (USER_UC_1)** **AF-S1-b. Email not found!** **1. System displays "Email not found!"** **2. System prompts the user to input email again.** **3. Continue from Main Flow Step 1**

	AF-S1-c. Phone number not found! 1. **System displays "Phone number not found!"** 2. **System prompts the user to input phone number again.** 3. **Continue from Main Flow Step 1**
Exceptions:	
Includes:	
Extends:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

Use Case ID:	**USER_UC_4**		
Use Case Name:	**View Account Details**		
Created By:	**Ng Hoe Ping**	Last Updated By:	**Ng Hoe Ping**
Date Created:	**2/9/2024**	Date Last Updated:	**25/10/2024**

Actor:	**User**
Description:	**Allows a user to view their own individual account details**
Preconditions:	1. **User must be logged in**
Postconditions:	
Priority:	
Frequency of Use:	**2**
Flow of Events:	1. **User selects the "User Profile" option from the top nav bar.** 2. **Users are able to view basic information such as username, email, phone number, biography, total ratings, total listings & rentals and reviews.** 3. **User can choose "Manage Account Details" (USER_UC_5), or choose to go back to the main page.**

Alternative Flows:	
Exceptions:	
Includes:	Manage Account Details (**USER_UC_5)**
Extends:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

Use Case ID:	**USER_UC_5**		
Use Case Name:	**MANAGE ACCOUNT DETAILS**		
Created By:	**Ng Hoe Ping**	Last Updated By:	**Ng Hoe Ping**
Date Created:	**2/9/2024**	Date Last Updated:	**25/10/2024**

Actor:	**User**
Description:	Allows a user to read/modify their own individual account details.
Preconditions:	**1. User must be logged in**
Postconditions:	
Priority:	
Frequency of Use:	**2**
Flow of Events:	**1. User can choose to edit their individual data by accessing "Change Password" (USER_UC6) or "Edit User Data" (USER_UC_7) or delete their account via "Delete Account" (USER_UC_8)**
Alternative Flows:	
Exceptions:	
Includes:	Change Password (**USER_UC_6)** Edit User Data (**USER_UC_7)**

	Delete Account (**USER_UC_8)**
Extends:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

Use Case ID:	**USER_UC_6**		
Use Case Name:	**Change Password**		
Created By:	**Ng Hoe Ping**	Last Updated By:	**Ng Hoe Ping**
Date Created:	**2/9/2024**	Date Last Updated:	**25/10/2024**

Actor:	**User**
Description:	Allows a user to change their existing password
Preconditions:	**1. User must be logged in**
Postconditions:	**1. Account details are stored in the database to be retrieved when prompted**
Priority:	
Frequency of Use:	**1**
Flow of Events:	**1. System prompts the user to input current account password.** **2. System validates the password with existing information in the database.** **3. System then prompts the user to input a new password twice.** **4. User inputs a new password twice** **5. System prompts "Your password has been changed!"** **6. System updates the new password, overwriting the existing information in the database.**

Alternative Flows:	**AF-S1-a. Current Password does not match** **1. Starting from main flow step 2.** **2. System prompts "Password does not match!"** **3. Continue from main flow step 1.** **AF-S1-b. Password does not meet requirements.** **1. Starting from main flow step 4.** **2. System displays "Please input a password of at least 1 special character, mixed case and at least 8 characters long" message.** **4. User inputs a new password** **5. System displays "Your password has been changed!"** **6. System updates the new password, overwriting the existing information in the database** **AF-S1-b. "Confirm Password does not match new password"** **3. Starting from main flow step 4.** **4. System displays "Please make sure your passwords match." message.** **7. User re-inputs the password.** **8. System displays "Your password has been changed!"** **9. System updates the new password, overwriting the existing information in the database.**
Exceptions:	
Includes:	
Extends:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

Use Case ID:	**USER_UC_7**
Use Case Name:	**Edit User Details**

Created By:	**Ng Hoe Ping**	Last Updated By:	**Ng Hoe Ping**
Date Created:	**2/9/2024**	Date Last Updated:	**30/10/2024**

Actor:	**User**
Description:	Allows a user to read/modify their own individual account details.
Preconditions:	**1. User must be logged in**
Postconditions:	**1. Account details are updated successfully in the database and displayed on the front end**
Priority:	
Frequency of Use:	**3**
Flow of Events:	**1. Users can choose to edit their username, phone number, biography and profile picture.** **2. After User edit and confirm changes** **3. System will prompt "Changes saved!"** **4. System will direct back to "Manage Account Details" (USER_UC_5)** **AF-S1-a. Phone number has been taken** **1. User chooses a new phone number and confirms changes.** **2. System displays "Phone number has been taken!"** **3. System prompts User to input a new phone number** **4. Continue from main flow step 2.**
Alternative Flows:	
Exceptions:	
Includes:	
Extends:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

Use Case ID:	**USER_UC_8**		
Use Case Name:	**Delete Account**		
Created By:	**Ng Hoe Ping**	Last Updated By:	**Ng Hoe Ping**
Date Created:	**2/9/2024**	Date Last Updated:	**30/10/2024**

Actor:	**User**
Description:	Allows a user to delete their own account
Preconditions:	**1. User must be logged in**
Postconditions:	**1. Account is deleted from the database and user is logged out from current page** **2. User is redirected to home page**
Priority:	
Frequency of Use:	**1**
Flow of Events:	**1. System prompts the user to delete account** **2. User clicks on "OK" button** **3. System deletes the account from database** **4. System logs user out** **5. User is redirected to home page**
Alternative Flows:	**AF-S1-a. User clicks on "Cancel" button** **1. Continue from main flow step 1**
Exceptions:	
Includes:	
Extends:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

Review System

Use Case ID:	**REVIEW_UC_1**		
Use Case Name:	**Submit a Review**		
Created By:	**Li Min**	Last Updated By:	**Hudzaifah**
Date Created:	**31/8/24**	Date Last Updated:	**26/10/24**

Actor:	**User, Database API**
Description:	**This use case allows a user to submit a review for a service or product. The user provides feedback on the item/service condition and optionally rates the service/item.**
Preconditions:	**1. User must be logged in** **2. User must have completed the product or service** **3. The system displays the "Rate Service/Product" button**
Postconditions:	**1. Feedback on the item/service condition is included in the review** **2. Displays a "success message" to a user**
Priority:	**High**
Frequency of Use:	**Whenever a user completes a product/service, they must submit a rating and review.**
Flow of Events:	**1. The system displays the "Rate Service/Product" button.** **2. The user clicks the button to initiate the review process.** **3. The user selects a rating between 1 to 5 stars.** **4. The user enters a text review.** **5. The user clicks "Submit Review."** **6. The system validates the input fields and saves the review.** **7. The system displays a success message to the user.**
Alternative Flows:	**AF-S1-a. User leaves ratings blank** **1. System prompts user to "Please select a rating!"** **2. Continue from main flow step 1**

	AF-S1-b. User leaves comments blank **1. System displays "Please enter some comments!"** **2. Continue from main flow step 5**
Exceptions:	**If user fails to submit the required fields (rating/text) the system will highlight the missing fields.**
Includes:	**Review Item/Service Condition (REVIEW_UC_2)** **Rate Service/Product (REVIEW_UC_3)**
Extends:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

Use Case ID:	**REVIEW_UC_2**		
Use Case Name:	**Review Item/Service Condition**		
Created By:	**Li Min**	Last Updated By:	**Li Min**
Date Created:	**31/8/24**	Date Last Updated:	**26/10/24**

Actor:	**User**
Description:	**This use case allows the user to provide feedback on the condition of the item or service being reviewed. This is part of the broader "Submit Review" use case.**
Preconditions:	**1. User must be logged in** **2. User must have had rented the product**
Postconditions:	**1. Feedback on the item/service condition is included in the review**
Priority:	
Frequency of Use:	**3**

Flow of Events:	**1. The user is prompted to describe the condition of the item/service.** **2. The user provides feedback on the condition**
Alternative Flows:	
Exceptions:	
Includes:	
Extends:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

Use Case ID:	**REVIEW_UC_3**		
Use Case Name:	**Rate Service/Product**		
Created By:	**Li Min**	Last Updated By:	**Hudzaifah**
Date Created:	**31/8/24**	Date Last Updated:	**23/9/24**

Actor:	**User**
Description:	**This use case allows the user to rate the service or product out of 5 stars. This is included as part of the broader "Submit Review" use case.**
Preconditions:	**1. User must be logged in** **2. User must have had rented the product**
Postconditions:	**1. Review is stored in the system and available for others to view**
Priority:	**High**
Frequency of Use:	**3**
Flow of Events:	**1. The system prompts the user to provide a rating out of 5.** **2. The user selects a rating.**

	3. The system saves the rating. **4. The system allows user to proceed with review submission.**
Alternative Flows:	**If no rating, then system prompts user to choose rating.**
Exceptions:	
Includes:	
Extends:	
Special Requirements:	
Assumptions:	**System will display a rating scale (stars format)**
Notes and Issues:	

Listing System

Use Case ID:	**LIST_UC_1**		
Use Case Name:	**Manage Listing**		
Created By:	**Hui Wen**	Last Updated By:	**Hui Wen**
Date Created:	**1/9/24**	Date Last Updated:	**29/10/2024**

Actor:	**User, Maps API, Database**
Description:	**Allows user to create, update, or delete a listing.**
Preconditions:	**1. User must be logged in.** **2. User must enter all the necessary information.**
Postconditions:	**1. The listing is successfully created, updated, or deleted.** **2. Any changes made to a listing are reflected in the app and visible to all users.** **3. The system ensures the listing complies with all necessary validation rules before saving.**
Priority:	
Frequency of Use:	**4**

Flow of Events:	**1. The renter navigates to the "Listings" section in the side bar.** **2. The renter selects the option to create a new listing, update an existing listing, or delete a listing.** **3. The renter provides or modifies details such as the title, description, category, rates, location, and images.** **4. The renter submits the changes.** **5. The system saves the changes and updates the listing information.** **6. The renter receives a confirmation that the listing has been successfully managed.**
Alternative Flows:	**AF-S1-a. User cancels the operation before submission** **1. System exits form and returns to the main listing page, no changes will be saved.** **AF-S1-b System encounters an error upon submission(e.g., database connectivity issues)** **1. Notifies the user with a prompt telling them to resubmit.**
Exceptions:	
Includes:	Update Listing Details Make New Listing Delete Listing
Extends:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

Use Case ID:	**LIST_UC_2**
Use Case Name:	**Make New Listing**

Created By:	**Hui Wen**	Last Updated By:	**Hui Wen**
Date Created:	**1/9/24**	Date Last Updated:	**29/10/2024**

Actor:	**User**
Description:	**Allows user to create a new listing to showcase the item/service they want to rent out.**
Preconditions:	**1. User must be logged in.** **2. User must enter all the necessary information.**
Postconditions:	**1. Listing is stored in the system and available for other users to view**
Priority:	
Frequency of Use:	**3**
Flow of Events:	**1. User selects create listing option** **2. System prompts user to input details (Covered in Enter Listing Details)** **3. User submits listing.** **4. System saves the listing and displays it on the user's profile. The listing is made available for other users to view.**
Alternative Flows:	**AF-S1-a. User leaves Title/description/rental price/location blank and tries to submit listing** **1. System displays "Please fill in all fields"** **2. continue from main flow step 2** **AF-S1-b. User leaves the create listing page before submitting** **1. System prompts user "Are you sure you want to delete this listing?"** **2. If user selects yes, all information entered is deleted and user will return to Manage Listing page** **3. If user selects no continue main flow from step 2.**
Exceptions:	
Includes:	Enter Listing Details

Extends:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

Use Case ID:	**LIST_UC_3**		
Use Case Name:	**Update Listing Details**		
Created By:	**Hui Wen**	Last Updated By:	**Hui Wen**
Date Created:	**1/9/24**	Date Last Updated:	**29/10/2024**

Actor:	**User**
Description:	**Allows user to edit their existing listings.**
Preconditions:	**1. User must be logged in.** **2. User must select an existing listing that they had previously created.**
Postconditions:	**1. Edits to listing is updated and stored in the system and available for other users to view**
Priority:	
Frequency of Use:	**3**
Flow of Events:	**1. User selects update listing option** **2. User edits the information in their listing such as changing the title, description, rating, photo.** **3. User confirms listing update.** **4. System saves the edits and displays it on the user's profile. The updated listing is made available for other users to view.**
Alternative Flows:	**AF-S1-a. User leaves Title/description/rental price/location blank and tries to confirm listing update** **1. System displays "Please fill in all fields "** **2. continue from main flow step 2**

	AF-S1-b. User leaves the update listing page before submiting but doesn't change anything **1. Main flow from step 1.**
Exceptions:	
Includes:	Enter Listing Details
Extends:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

Use Case ID:	**LIST_UC_4**		
Use Case Name:	**Delete Listing**		
Created By:	**Hui Wen**	Last Updated By:	**Hui Wen**
Date Created:	**1/9/24**	Date Last Updated:	**29/10/2024**

Actor:	**User**
Description:	**Allows the user to delete their own existing listing.**
Preconditions:	**1. User must be logged in.** **2. User must select an existing listing that they had previously created.**
Postconditions:	**1. Listing is removed from the system and users are no longer able to view it.**
Priority:	
Frequency of Use:	**3**
Flow of Events:	**1. User selects delete listing option** **2. System prompts user to confirm deletion.** **3. User confirms listing deletion.** **4. Listing is deleted from the system and is no longer available for other users to view.**

Alternative Flows:	**AF-S1-a. User selects "cancel" when being prompted to confirm deletion** **1. System cancels the deletion, listing is not deleted.** **2. User is returned to the original listing page.**
Exceptions:	
Includes:	
Extends:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

Use Case ID:	**LIST_UC_5**		
Use Case Name:	**Select Location**		
Created By:	**Hui Wen**	Last Updated By:	**Hui Wen**
Date Created:	**1/9/24**	Date Last Updated:	**29/10/2024**

Actor:	**User, Maps API**
Description:	**Allows user to select their location that a rentee can use when deciding to rent.**
Preconditions:	**1. User must be logged in.** **2. User must allow locations to be used**
Postconditions:	**1. The selected location is stored as part of the listing and is visible to all users**
Priority:	
Frequency of Use:	**3**
Flow of Events:	**1. User search for location in the search bar** **2. The user clicks on the location that they wish to set.** **3. User enters any other additional notes**

	4. System saves the selected location as part of the rental request.
Alternative Flows:	**AF-S1-a. Location Not Found:** **1. If the system is unable to find the location entered by the user, an error message is displayed which prompts the user to re-enter the address** **2. Continue from main flow 1.**
Exceptions:	
Includes:	
Extends:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

Use Case ID:	**LIST_UC_6**		
Use Case Name:	**Enter Listing Details**		
Created By:	**Ze Ming**	Last Updated By:	**Ze Ming**
Date Created:	**1/9/2024**	Date Last Updated:	**29/10/2024**

Actor:	**User**
Description:	**Enter Listing Details allows user to enter details so that they are able to find related items to borrow**
Preconditions:	**1. User must be logged in** **2. A listing must have been clicked**
Postconditions:	**1. User selects Listing location**
Priority:	
Frequency of Use:	**1**

Flow of Events:	**1. User clicks on listing name and inputs listing name** **2. User clicks on listing details and inputs listing details** **3. System validates that the input** **4. User can then select listing location**
Alternative Flows:	**AF-S1-a. User cancels enter listing details:** **1. User can click back button on the browser tab at any time** **2. The system will exit the page and brings the user back to the manage listing page** **3. Continue from Manage Listing**
Exceptions:	
Includes:	**Select Location (LIST_UC_5)**
Extends:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

Browsing System

Use Case ID:	**BROWSE_UC_1**		
Use Case Name:	**Search Listing**		
Created By:	**Yuxuan**	Last Updated By:	**Yuxuan**
Date Created:	**3/9/24**	Date Last Updated:	**29/10/24**

Actor:	**User**
Description:	**The Search Listing function allows the user to search for goods and services currently available on the web app. Users input keywords to search the listings, and the system filters and displays results that match the search criteria.**

Preconditions:	
Postconditions:	**1. System will display the relevant items or services that match the keywords entered by the user.**
Priority:	
Frequency of Use:	**5**
Flow of Events:	**1. System prompts the user to enter the keyword into the search bar.** **2. User clicks the search button to initiate the search.** **3. System processes the search request and retrieves listings that match the entered keywords.** **4. System displays a list of relevant listings based on the search results.**
Alternative Flows:	**AF-S1-a. No search results found** **Trigger: The system finds no listings that match the entered keywords.** **1. The system displays a message: "No listings found matching your search criteria."** **2. The system suggests related search terms or prompts the user to broaden their search.** **3. Users can modify the keywords and initiate a new search.** **4. Continue from main flow step 3**
Exceptions:	
Includes:	**Open Listing Details (BROWSE_UC_6)**
Extends:	**Filter Listing (BROWSE_UC_2)**
Special Requirements:	
Assumptions:	
Notes and Issues:	

Use Case ID:	**BROWSE_UC_2**		
Use Case Name:	**Filter Listing**		
Created By:	**Yuxuan**	Last Updated By:	**Yuxuan**

Date Created:	**3/9/24**	Date Last Updated:	**29/10/24**

Actor:	**User**
Description:	**The filter listing functionality allows users to apply multiple filters to refine the scope of listings and narrow down the search results. This helps users find relevant items or services efficiently based on their specific needs. Available filters may include category, price range, location, and sorting order.**
Preconditions:	**.**
Postconditions:	**1. The system displays listings that match all the selected filters.** **2. If no listings match the applied filters, the system informs the user accordingly.**
Priority:	
Frequency of Use:	**4**
Flow of Events:	**1. The system displays a "Filter" button on the web page.** **2. System prompts the user to click the "Filter" button, and the system presents a list of available filter options.** **3. System prompts the user to select multiple filters.** **4. User clicks the “Apply” button to apply the filters.** **5. System processes the filter selections and refreshes the listing page to show only the items or services that meet the selected criteria.**
Alternative Flows:	
Exceptions:	
Includes:	**Select Category (BROWSE_UC_3)** **Select Price Range (BROWSE_UC_4)** **Sorting order (BROWSE_UC_5)**
Extends:	
Special Requirements:	
Assumptions:	

Notes and Issues:	

Use Case ID:	**BROWSE_UC_3**		
Use Case Name:	**Select Category**		
Created By:	**Yuxuan**	Last Updated By:	**Yuxuan**
Date Created:	**3/9/24**	Date Last Updated:	**29/10/24**

Actor:	**User**
Description:	**Users can select multiple categories from a predefined list to filter and search for relevant items or services. The system will display items or services based on the selected categories, allowing users to narrow down their search.**
Preconditions:	
Postconditions:	**1. The system displays a list of items or services that match the categories selected by the user.** **2. If no listings match the categories, the system notifies the user and prompts for changes to the search.**
Priority:	
Frequency of Use:	**3**
Flow of Events:	**1. System prompts the user to tick multiple category boxes.** **2. System prompts the user to confirm the category selection by clicking the "Apply" button.** **3. System retrieves listings that match the selected categories and displays them to the user.** **4. System display listings that match the filters.**
Alternative Flows:	**AF-S1-a. No available listings for current filters applied**

	1. **System will display the warning message "No available listing and please enter again" if filters applied have filtered out all listings available** 2. **User must clear the current filters** 3. **Continue from main flow step 1**
Exceptions:	
Includes:	
Extends:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

Use Case ID:	**BROWSE_UC_4**		
Use Case Name:	**Select Price Range**		
Created By:	**Yuxuan**	Last Updated By:	**Yuxuan**
Date Created:	**3/9/24**	Date Last Updated:	**29/10/24**

Actor:	**User**
Description:	**The user specifies a price range for the listings by entering minimum and maximum price values and selecting a relevant time unit (hourly, daily, weekly and one-time payment). The system will filter and display listings that match the specified price range and time unit.**
Preconditions:	
Postconditions:	1. **The system displays listings that fall within the specified price range and time unit.**
Priority:	
Frequency of Use:	**3**
Flow of Events:	1. **System prompts the user to select the time unit of pricing.**

	2. System prompts the user to key in minimum and maximum price values **3. User enters minimum and maximum price values.** **4. Users click the "Apply" button to apply the filters.** **5. The system processes the request and displays listings that match the price range and time unit criteria.** **6. System redirects the user to the landing page with all the listings that match the price range filters.**
Alternative Flows:	**AF-S1-a. No listing found** **Trigger: No listings are available that match the specified price range and time unit.** **1. The system displays a message: "No listings found."** **2. Users modify the filters to search again.** **3. Continue from main flow step 4**
Exceptions:	
Includes:	
Extends:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

Use Case ID:	**BROWSE_UC_5**		
Use Case Name:	**Sorting order**		
Created By:	**Yuxuan**	Last Updated By:	**Yuxuan**
Date Created:	**3/9/24**	Date Last Updated:	**29/10/24**

Actor:	**User**
Description:	**System allows users to sort listings based on various sorting metrics, such as distance, price or user rating. Users can choose**

	to sort listings in ascending or descending order based on the selected metric.
Preconditions:	
Postconditions:	**1. System will display the listings in sorted order based on the sorting metrics chosen**
Priority:	
Frequency of Use:	**3**
Flow of Events:	**1. System prompts the user to select rate (per hour, day, week or one-time)** **2. User selects the rate** **3. System prompts the user to select either ascending or descending** **4. User enters their choices** **5. User confirms the selection by clicking "Apply".** **6. System displays listings sorted according to the selected metrics and order.**
Alternative Flows:	**AF-S1-a. No listing to sort** **Trigger: There are no listings available to sort when the user attempts to apply sorting** **1. The system displays a message: "No listings available."** **2. System prompts the user to either modify their search criteria or return to the search page to generate listings.** **3. Continue from main flow step 3**
Exceptions:	
Includes:	
Extends:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

Use Case ID:	**BROWSE_UC_6**
Use Case Name:	**Open Listing Details**

Created By:	**Seow Jia Xian**	Last Updated By:	**Seow Jia Xian**
Date Created:	**1/9/24**	Date Last Updated:	**28/10/24**

Actor:	**User**
Description:	**Open Listing Details allows the user to view listing details regarding a product**
Preconditions:	
Postconditions:	**1. User is able to view the details of the product**
Priority:	
Frequency of Use:	**4**
Flow of Events:	**1. User clicks on a product** **2. System redirects user to a new page displaying listing details of the product as well as actions which includes Report Listing, View User Rating and Make Offer (UC_7 & UC_8 & UC_9)**
Alternative Flows:	
Exceptions:	
Includes:	**Report Listing, View User Rating, Make Offer**
Extends:	**Manage Offer**
Special Requirements:	
Assumptions:	
Notes and Issues:	

Use Case ID:	**BROWSE_UC_7**		
Use Case Name:	**Report Listing**		
Created By:	**Seow Jia Xian**	Last Updated By:	**Seow Jia Xian**
Date Created:	**1/9/24**	Date Last Updated:	**28/10/24**

Actor:	**User**

Description:	**Report Listing allows users to report a listing that is deemed suspicious or illegal**
Preconditions:	**1. User must have clicked on the listed product that they wish to report**
Postconditions:	**1. Listed product is removed from the shop and the account associated with it may be banned**
Priority:	
Frequency of Use:	**1**
Flow of Events:	**1. User clicks on the report button** **2. Report Listing pop-up window appears** **3. User enters details on violations/community guidelines that the listed product has invoked** **4. User submits the information** **5. System will do a review of the report and log the event** **6. Review team will remove the listed product and ban the account associated with it if necessary**
Alternative Flows:	**AF-S1-a.** **User clicks on the cancel button on the Report Listing pop-up window which causes it to be closed**
Exceptions:	
Includes:	
Extends:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

Use Case ID:	**BROWSE_UC_8**		
Use Case Name:	**View User Ratings**		
Created By:	**Seow Jia Xian**	Last Updated By:	**Seow Jia Xian**
Date Created:	**1/9/24**	Date Last Updated:	**28/10/24**

Actor:	**User**
Description:	**View User Ratings allows users (customer) to view the user ratings/reviews of the seller. User ratings will include ratings ranging from 1-5 stars from past deals/transactions completed by the seller**
Preconditions:	**1. User must have clicked on the view user ratings button on the product page**
Postconditions:	**1. Past deals completed by the seller with the user ratings and comments is displayed to the user (customer)**
Priority:	
Frequency of Use:	**3**
Flow of Events:	**1. User clicks on the view user ratings button** **2. System redirects the user to the renter's profile page which contains the reviews** **3. System displays the information to the user**
Alternative Flows:	
Exceptions:	
Includes:	
Extends:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

Use Case ID:	**BROWSE_UC_9**		
Use Case Name:	**Manage Offer**		
Created By:	**Seow Jia Xian**	Last Updated By:	**Seow Jia Xian**
Date Created:	**1/9/24**	Date Last Updated:	**28/10/24**

Actor:	**User**
Description:	**Manage offer allows the user to make/accept/reject offers. Typically, the renter is the only party allowed to accept or reject offers while the rentee is only allowed to make offers.**
Preconditions:	**1. User must have an User account**
Postconditions:	**1. User is able to make/accept/reject offers** **2. Database will be updated with the information**
Priority:	
Frequency of Use:	**4**
Flow of Events:	**1. User clicks on a product** **2. System identifies if user is a renter or rentee** **3. System will allow user to make offer if user is a rentee; accept/reject offer if user is a renter** **4. User will make a decision** **5. System will perform the corresponding tasks according to user's decision**
Alternative Flows:	
Exceptions:	
Includes:	**Make Offer, Accept Offer, Reject Offer**
Extends:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

Use Case ID:	**BROWSE_UC_10**		
Use Case Name:	**Make Offer**		
Created By:	**Seow Jia Xian**	Last Updated By:	**Seow Jia Xian**
Date Created:	**1/9/24**	Date Last Updated:	**28/10/24**

Actor:	**User**

Description:	**Make Offer allows the rentee to make an offer to the renter.**
Preconditions:	**1. User must have an User account** **2. User must have clicked on the make offer button on the product page**
Postconditions:	**1. The renter will be able to view the offer that was made by the user**
Priority:	
Frequency of Use:	**3**
Flow of Events:	**1. User enters an offer price** **2. User selects on the calendar to indicate availability of the renting (includes date and time)** **3. User clicks on the make offer button and submits the price to the system** **4. System informs the renter that an offer has been made**
Alternative Flows:	
Exceptions:	
Includes:	
Extends:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

Use Case ID:	**BROWSE_UC_11**		
Use Case Name:	**Accept Offer**		
Created By:	**Seow Jia Xian**	Last Updated By:	**Seow Jia Xian**
Date Created:	**1/9/24**	Date Last Updated:	**28/10/24**

Actor:	**User**
Description:	**Accept Offer allows the renter to accept an offer that was made by rentee.**

Preconditions:	**1. User must have an User account** **2. User has received an offer on their product listing**
Postconditions:	**1. The rentee is informed that the offer has been accepted by the renter**
Priority:	
Frequency of Use:	**3**
Flow of Events:	**1. User clicks on "Offers" under account sidebar** **2. User (renter) clicks on the product with offers** **3. User (renter) clicks on the accept offer button** **4. System will notify the other party (rentee) that the offer has been accepted by the renter** **5. Rentee will need to login to their account and proceed with payment**
Alternative Flows:	
Exceptions:	
Includes:	
Extends:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

Use Case ID:	**BROWSE_UC_12**		
Use Case Name:	**Reject Offer**		
Created By:	**Seow Jia Xian**	Last Updated By:	**Seow Jia Xian**
Date Created:	**1/9/24**	Date Last Updated:	**28/10/24**

Actor:	**User**
Description:	**Reject Offer allows the renter to reject an offer that was made by the rentee**
Preconditions:	**1. User must have an User account**

	2. User must have clicked on the reject offer button on their product listing page
Postconditions:	**1. The rentee is informed that the offer has been rejected by the seller**
Priority:	
Frequency of Use:	**3**
Flow of Events:	**1. User clicks on "Offers" under account sidebar** **2. User (renter) clicks on the product they want to reject offers** **3. User (renter) clicks on the reject offer button** **4. System will notify the other party (rentee) that the offer has been rejected by the seller**
Alternative Flows:	
Exceptions:	
Includes:	
Extends:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

Payment System

Use Case ID:	**PAY_UC_1**		
Use Case Name:	**Authorize Payment**		
Created By:	**Ze Ming**	Last Updated By:	**Ze Ming**
Date Created:	**1/9/2024**	Date Last Updated:	**27/10/2024**

Actor:	**User, Payment API**
Description:	**Authorize Payment allows the user to pay for the service**
Preconditions:	**1. User must be logged in**

	2. User must have gotten an offer they made accepted
Postconditions:	
Priority:	
Frequency of Use:	**2**
Flow of Events:	**1. User will select to pay for the listing** **2. System will redirect the user to the payment gateway** **3. User will input their payment methods and pay** **4. Paypal will verify that payment has been made**
Alternative Flows:	**AF-S1-a. User cancels authorize payment:** **4. User can click back button on the browser tab at any time** **5. The system will exit the page and brings the user back to the offer page** **6. Continue from offers**
Exceptions:	
Includes:	**Enter Card Details (PAY_UC_2)** **Produce Receipt (PAY_UC_3)**
Extends:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

Use Case ID:	**PAY_UC_2**		
Use Case Name:	**Enter Credentials**		
Created By:	**Ze Ming**	Last Updated By:	**Ze Ming**
Date Created:	**1/9/2024**	Date Last Updated:	**27/10/2024**

Actor:	**User, Payment API**
Description:	**Authorize Payment allows the user to pay for the service**
Preconditions:	**1. User must be logged in**

	2. User must have gotten an offer they made accepted
Postconditions:	
Priority:	
Frequency of Use:	**3**
Flow of Events:	**1. System Redirects User to Payment API page** **2. Paypal prompts user to input credentials** **3. User enters credentials and submits** **4. Paypal verifies credentials** **5. Paypal successfully verifies credentials and redirects User back to Offer page**
Alternative Flows:	**User clicks the back button on the browser tab, the system will then show the initial payment page** **AF-S1-a. Credentials are wrong** **5. Paypal notifies user of incorrect credentials** **6. Continue from Step 2**
Exceptions:	
Includes:	
Extends:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

Use Case ID:	**PAY_UC_3**		
Use Case Name:	**Produce Receipt**		
Created By:	**Ze Ming**	Last Updated By:	**Ze Ming**
Date Created:	**1/9/2024**	Date Last Updated:	**27/10/2024**

Actor:	**User, Payment API**

Description:	**Authorize Payment allows the user to pay for the service**
Preconditions:	**1. User must be logged in** **2. User must have successfully completed payment**
Postconditions:	
Priority:	
Frequency of Use:	**2**
Flow of Events:	**1. Paypal API generates an e-receipt and sends it to the User's email.**
Alternative Flows:	**User clicks the back button on the browser tab, the system will then show the initial payment page**
Exceptions:	
Includes:	
Extends:	
Special Requirements:	
Assumptions:	
Notes and Issues:	

4.2 Functional Requirements

1. User Account Management System

1.1. Users must be able to register for a new account

1.1.1. System must check that all requirements are fulfilled before making a new account

1.1.1.1. User must click on the username input text box and enter a username

1.1.1.2. User must click on the email input text box and enter a email

1.1.1.2.1. If it is taken, there will be a red message that says "Email already taken!" that will be displayed to the user

1.1.1.3. User must click on the password input text box and enter a password

1.1.1.4. The system will check that the password is at least of 8 characters and consists of a upper, lower, number and a special character

1.1.1.4.1. If it does not fulfil the password requirements, there will be a red message that says "Password does not fulfil requirements, please have at least 8 characters, with at least 1 number" that will be displayed to the user

1.1.1.5. User must click on the phone number input text box and enter a phone number

1.1.1.6. The system will check that the phone number contains only digits

1.1.1.6.1. If it does not fulfil the phone number requirements, there will be a red message that says "Phone number does not fulfil requirements" that will be displayed to the user

1.2. User must be able to login to his account

1.2.1. User must click email and password input textbox and enter their details and submit

1.2.2. The system must validate the user's email and password

1.2.2.1. If it is not valid, it will show "Invalid" credentials in red, to be displayed to the user

1.2.3. System will show the home page after the user has logged in

1.2.4. User must be able to reset his password in the case it is forgotten

1.2.4.1. System must verify the user details with their email and phone number

1.2.4.2. System must verify that both email and phone number are in the database, only then user can input a new password

1.2.4.3. System must show a success message to the user when his password is changed

1.3. User must be able to view their user account data

1.3.1. User must click view user data

1.3.2. System will show the user their username, phone, email, password, ratings, listings, rentals, reviews, and bio.

1.4. User must be able to manage his account details

1.4.1. User must be able to update user details

1.4.1.1. User must click on their username and enter their new username

1.4.1.2. System must show a success message when the username of the user is updated

1.4.1.3. User must click on their phone number and enter their new phone number

1.4.1.4. The system will check that the phone number is not taken

1.4.1.4.1. If it is taken, there will be a red message that says "Phone number already taken" that will be displayed to the user

1.4.1.5. System must show a success message when the phone number of the user is updated

1.4.1.6. User must click on their biography and enter a new text

1.4.1.7. System must show a success message when the biography of the user is updated

1.4.1.8. User must click on their profile picture icon and upload a new image

1.4.1.9. System must show a success message when the profile picture of the user is updated

1.4.2. User must be able to change their password

1.4.2.1. User must click on the old password input text box and enter his old password

1.4.2.2. System must verify that the old password exists in the database

1.4.2.3. User must input both new password and confirm password

1.4.2.4. System must verify that both new password and confirm password matches

1.4.2.5. System must show a success message once the password has been changed

1.4.3. User must be able to delete their account

1.4.3.1. User must click on the delete account button

1.4.3.2. System must display a pop-up window to prompt user to confirm their deletion

1.4.3.3. System will perform a deletion of the account in the database

1.4.3.4. System will log user out of their account

1.4.3.5. User will be redirected back to home page

2. Review System

2.1. User must be able to submit a review

2.1.1. User must click on "Submit" review on the product listing or service that has been completed

2.1.2. User must click on a rating between 1 to 5 stars

2.1.3. The user must click on the text box to enter a review of the product listing or service

2.1.4. The user must then click "Submit Review"

2.1.5. The system must verify that the user has submitted a rating between 1 to 5 stars

2.1.6. The system will verify that it has received the review, and show a success message to the user

2.6 System must verify and accept the review

2.6.1 The system verifies that a rating between 1 and 5 stars has been selected.

2.6.2 The system verifies the review text has been submitted.

2.6.3 The system saves the review in the database.

2.6.4 The system displays a success message confirming the submission.

3. Listing System

3.1. User must be able to manage his listings

3.1.1. User must click on "Create Listing" button on the home page to create a listing

3.1.1.1. The user must enter the details of the items he/she wants to list in the text box

3.1.1.2. The user must upload at least one photo of their choice

3.1.1.3. System will register the item and send the details of the listing to the database

3.1.2. User may update the details of their listed items under the "Your Listings" page which is found under account information tab

3.1.2.1. The user must click on the edit button of the listed item that they want to update

3.1.2.2. User must re-enter/change the details of the listed item if need be

3.1.2.3. User must press the "Submit" button to make the necessary changes

3.1.2.4. System will send the updated details of the listed item to the database to overwrite the old details

3.1.3. User may delete their item listings under the "Your Listings" page which is found under account information tab

3.1.3.1. The user must click on delete button of the listed item that they want to delete

3.1.3.2. The system will display a pop-up box to verify if user wants to delete the selected item

3.1.3.2.1. System will not delete the listing if the user clicks on the "No" button in the pop-up box

3.1.3.3. The system will delete the selected item if the user clicks on the "Yes" button and remove the details from the database

4. Browsing System

4.1. User must be able to search listing

4.1.1. User must be able to type in keywords to find items in the search bar

4.1.2. System will redirect the user to the landing page with all relevant items

4.2. User must be able to filter listings by applying various filters to find their desired items.

4.2.1. User must be able select one or more categories by ticking the relevant category boxes.

4.2.2. System will redirect users to show items that fall under the categories that user chooses

4.2.3. User must be able to key in price range or use the slider to filter the items that fall within this range

4.2.4. System will redirect users to show items that fall within this price range

4.2.5. System must have various time-measuring units: by hours, by day, by week or one-time basis for users to choose from

4.2.6. System will display the items in categories, price ranges, order of display in terms of different time units once the user make their choices

4.3. User must be able to open listing details when clicking on the listing profiles

4.3.1. System must display seller's overall rating in terms of a star rating out of 5 based on customer reviews

4.3.2. System must display past customer reviews on various other listings of that particular seller

4.3.3. User must be able to manage offers

4.3.3.1. User must be able to make an offer

4.3.3.2. User must be able to accept offer if they are satisfied with the offer

4.3.3.3. User must be able to reject offer if they are not satisfied with the offer

4.3.4. User must be able to review report listing if they realise the listing is a scam

4.3.4.1. System must have an option to report listing

4.3.4.2. System will redirect the user if they want to report listing

4.3.4.3. User must be able to key in the text messages in the description box to describe the suspected scam case they encountered.

4.3.4.4. System will review the report and make necessary refunds arrangement if the report is deemed as truthful

5. Payment System

5.1. User must be able to make payment for selected item via PayPal

5.1.1. User must enter their bank card details on the payment page

5.1.2. User must click on the payment button

5.1.3. System will redirect the user to PayPal for payment processing

5.1.4. PayPal will deduct the balance from the user's account if there is sufficient balance

5.1.4.1. Error message will pop-up if user has insufficient balance in their account

5.1.4.2. User will be redirected back to product page with a pop-up message indicating that the purchase has failed

5.1.5. System will display a message informing users that the purchase has been successful

5.1.6. PayPal will generate a receipt/invoice containing details of the purchase

5.1.7. PayPal will send the receipt/invoice to the user's email

5.1.8. User may also check their purchase history under the account information tab

5. Other Nonfunctional Requirements

5.1 Performance Requirements

5.1.1 The web application shall take within 3 seconds to load in and exit

5.1.2 The account registration and login page shall take within 3 seconds to load

5.1.3 Verification for the validity of username, email, password, phone number shall take within 3 seconds

5.1.4 Any success message should take 3 second to display

5.1.5 New account must be updated in the system within 3 seconds

5.1.6 Any account information updates shall be reflected in the system within 3 seconds

5.1.7 System shall redirect the user to landing page within 3 seconds

5.1.8 Any review submission shall be stored in the system within 3 seconds

5.2 Safety Requirements

5.2.1 User must be able to launch a report of any suspicious listings/profiles to the system

5.2.2 Users to agree on a clear rental agreement that outlines the terms and conditions for each transaction

5.2.3 Developers of the team will perform consistent app maintenance

5.2.4 Adequate testing will be done by the developers

5.3 Security Requirements

5.3.1 User must be logged in to make an offer and to schedule an appointment

5.3.2 The web page will adhere to web security standards and use an updated security stack

5.3.3 System will use tried and tested industry standards of authenticating via JWT (Json Web Tokens) to authorise logged in users

5.4 Software Quality Attributes

5.4.1 Scalability

The system should be able to scale to fit the demands of the users at the point in time. The system is designed to separate the frontend and the backend. Both of these can be hosted separately on different servers to fit the needs of the users.

The system is also scalable in that it is able to swap out the database easily without issues. It is currently using SQLite as there is low demand, but if demand were to increase, this can be changed to PostgreSQL, MySQL or MSSQL. The database can then host more connections.

5.4.2 Maintainability

The code is modular and well separated. Front-end code does not fully depend on the backend code. But they communicate through interfaces and protocols. These protocols are automatically generated by the backend as well. In the case of attribute or model changes, the front-end developers will be able to know instantly.

Our backend database will also automatically generate database migrations. This is crucial in the case that a database migration was done that was not intended. This allows us to roll back the database migration and fix the issues. As well as maintain backwards compatibility.

5.4.3 Testability

The codebase is highly testable. There is an extremely high test coverage for the back-end apis. This ensures that the APIs will function and do exactly and precisely what they are supposed to do. This in turn lets the front-end developers work on the front-end code while depending on the back-end APIs to function.

5.5 Business Rules

5.5.1 Guest

- Guests are free to browse, filter, sort and search listings that are created on the platform

5.5.2 User

- A logged in user is able to do everything a guest can do and more.
- Users can create, delete listings, give reviews, make offers and make transactions as well

6. Other Requirements

6.1 Legal Requirements

The platform does not store any of the personal information that is not needed for the application to continue functioning. We do not take in the user's personal NRIC for example. For sensitive information such as payment details, we use PayPal to handle the transaction for us instead of keeping the user payment details.

6.2 Test Cases

6.2.1 Black Box Testing

6.2.1.1 User Account Management System

6.2.1.1.1 Sign Up / Register

Test Case ID	**T-6.2.1.1.1**	**Test Case Priority**	High
Test Case Description	Create an account for user		
Prerequisite	1. User should have their credentials ready for registering for an account	**Postrequisite**	1. User will be redirected to login page

Test Execution

#	Description	Action	Input	Expected Output	Actual Output	Test Result
1	Creating a new account	User enters username, email, password and phone number and tick the terms and conditions box User clicks on Create Account button	Username: user1 Email: user1@gmail.com Password: P@ssword1 Phone number: 98765432 Terms & Conditions: ✅	New account is created	New account is created	Pass
2	Submitting an empty form	User clicks on Create Account button without any inputs	NIL	Displays “All fields are required.”	Displays “All fields are required.”	Pass
3	Entering a username with less than 4 characters	User enters a username with less than 4 characters, email, password and phone number and	Username: 123 Email: 123@gmail.com	Displays “Username must be at least 4	Displays “Username must be at least 4 characters long.”	Pass

		tick the terms and conditions box User clicks on Create Account button	Password: P@ssword1 Phone number: 98765432 Terms & Conditions: ✅	characters long.”		
4	Entering an invalid email	User enters a username, invalid email, password and phone number and tick the terms and conditions box User clicks on Create Account button	Username: 1234 Email: 1234@gmail Password: P@ssword1 Phone number: 98765432 Terms & Conditions: ✅	Displays “Please enter a valid email address.’”	Displays “Please enter a valid email address.’”	Pass
5	Entering a password with does not meet the requirements of 1 lower, 1 upper, 1 digit and 1 special character	User enters a username, email, basic password and phone number and tick the terms and conditions box User clicks on Create Account button	Username: 1234 Email: 1234@gmail.com Password: password Phone number: 98765432 Terms & Conditions: ✅	Displays “Password does not meet the password requirements.’ ”	Displays “Password does not meet the password requirements.’”	Pass
6	Did not check terms and conditions box	User enters a username, email, password and phone number but does not tick the terms and conditions box User clicks on Create Account button	Username: 1234 Email: 1234@gmail.com Password: P@ssword1 Phone number: 98765432	Displays “You must agree to the terms and conditions.”	Displays “You must agree to the terms and conditions.’”	Pass

			Terms & Conditions:			
7	Enter a phone number that belongs to another account	User enters a username, email, password and phone number and tick the terms and conditions box User clicks on Create Account button	Username: 1234 Email: 1234@gmail.com Password: P@ssword1 Phone number: 99999999 Terms & Conditions: ✅	Displays “An account with this phone number exists!”	Displays “An account with this phone number exists!”	Pass
8	Enter an email that belongs to another account	User enters a username, email, password and phone number and tick the terms and conditions box User clicks on Create Account button	Username: 1234 Email: user1@gmail.com Password: P@ssword1 Phone number: 98765432 Terms & Conditions: ✅	Displays “An account with this email already exists.”	Displays “An account with this email already exists.”	Pass
Test Case Status		Success				

6.2.1.1.2 Login

Test Case ID	**T-6.2.1.1.2**	**Test Case Priority**	High
Test Case Description	Login as a user		
Prerequisite	1. User should have their login credentials ready.	**Postrequisite**	1. User will be redirected to home page with his account logged in

Test Execution						
#	**Description**	**Action**	**Input**	**Expected Output**	**Actual Output**	**Test Result**
1	Login with valid account details	Enter email and password User clicks on Log In button	Email: user1@gmail.com Password: P@ssword1	Redirected to home page with account logged in	Redirected to home page with account logged in	Pass
2	Submitting an empty form	User clicks on Log In button without any inputs	NIL	Displays “All fields are required.”	Displays “All fields are required.”	Pass
3	Enter a email that is not registered	Enter a unregistered email and a password User clicks on Log In button	Email: qwerty@gmail.com Password: qwerty	Displays “No account found with this email address.”	Displays “No account found with this email address.”	Pass
4	Enter a registered email with incorrect password	Enter a registered email with incorrect password User clicks on Log In button	Email: user1@gmail.com Password: password	Displays “Incorrect Password! Please try again.”	Displays “Incorrect Password! Please try again.”	Pass
5	Enter a registered email with incorrect password and submits the form 3 times	Enter a registered email with incorrect password User clicks on Log In button 3 times	Email: user1@gmail.com Password: password	Displays “Your account will be locked after two more attempts.”	Displays “Your account will be locked after two more attempts.”	Pass
6	Enter a registered email with incorrect password and submits the form 4 times	Enter a registered email with incorrect password User clicks on Log In button 4 times	Email: user1@gmail.com Password: password	Displays “Your account will be locked after one more attempt.”	Displays “Your account will be locked after one more attempt.”	Pass

7	Enter a registered email with incorrect password and submits the form 5 times	Enter a registered email with incorrect password User clicks on Log In button 5 times	Email: user1@gmail.com Password: password	Displays "Your account has been temporarily locked for 5 mins. Please try again later or reset your password Time of Login: 02:12:00 **Account will unlock at: 02:17:00**"	Displays "Your account has been temporarily locked for 5 mins. Please try again later or reset your password Time of Login: 02:12:00 **Account will unlock at: 02:17:00**"	Pass
8	Navigate user to reset password page (1.2.1.3)	Clicking on reset your password hyperlink from Test case 6	NIL	Navigate user to reset password page	Navigate user to reset password page	Pass
9	Clicking on Forget Password? (1.2.1.3)	Click on Forget Password? hyperlink	NIL	Navigate user to reset password page	Navigate user to reset password page	Pass
10	Navigate user to sign up page (1.2.1.1)	Click on Create a ShopBlock account hyperlink	NIL	Navigate user to sign up page	Navigate user to sign up page	Pass
Test Case Status		Success				

6.2.1.1.3 Reset Password

Test Case ID	**T-6.2.1.1.3**	**Test Case Priority**	High
Test Case Description	Reset Password for a user account		

Prerequisite	1. User should have their account credentials ready.	**Postrequisite**	1. User will be redirected to login page

Test Execution

#	Description	Action	Input	Expected Output	Actual Output	Test Result
1	Reset password with registered account	Enter email and phone number and a new password User clicks on Reset Password button	Email: user1@gmail.com Phone number: 98765432 Password: P@ssword123	Snackbar appears with a message displaying “Password reset successfully!” User is redirected to Log In page after a 2.5s delay	Snackbar appears with a message displaying “Password reset successfully!” User is redirected to Log In page after a 2.5s delay	Pass
2	Submitting an empty form	User clicks on Reset Password button without any inputs	NIL	Displays “All fields are required.”	Displays “All fields are required.”	Pass
3	Attempt to reset password with registered account and input basic password	Enter email and phone number and a new password that does not meet the password requirements User clicks on Reset Password button	Email: user1@gmail.com Phone number: 98765432 Password: password	Displays “New password does not match the password requirements. ”	Displays “New password does not match the password requirements.”	Pass
4	Enter unregistered email and phone number	Enter an unregistered email and phone number with a new password User clicks on Reset Password button	Email: qwerty@gmail.com Phone number: 88888888 Password: P@ssword123	Displays “Both email and phone number not found!”	Displays “Both email and phone number not found!”	Pass

5	Enter a valid email with unregistered phone number	Enter a valid email with unregistered phone number and a new password User clicks on Reset Password button	Email: user1@gmail.com Phone number: 45674567 Password: P@ssword123	Displays “Phone number not found!”	Displays “Phone number not found!”	Pass
6	Enter a unregistered email with valid phone number	Enter a unregistered email with valid phone number and a new password User clicks on Reset Password button	Email: qwerty@gmail.com Phone number: 98765432 Password: P@ssword123	Displays “Email not found!”	Displays “Email not found!”	Pass
Test Case Status		Success				

6.2.1.1.4 Change Password

Test Case ID	**T-6.2.1.1.4**	**Test Case Priority**	High
Test Case Description	Change password for a registered account		
Prerequisite	1. User must be logged in 2. User must have their account credentials ready	**Postrequisite**	1. User will be redirected to home page

Test Execution

#	Description	Action	Input	Expected Output	Actual Output	Test Result
1	Change password with logged in account	Enter old password, new password and confirm password	Password: P@ssword1 New Password: P@ssword123	Displays “Your password has been changed!”	Displays “Your password has been changed!” User is redirected to	Pass

		User clicks on Change Password button	Confirm Password: P@ssword123	User is redirected to home page after a 2.5s delay	home page after a 2.5s delay	
2	Submitting an empty form	User clicks on Change Password button without any inputs	NIL	Displays “All fields are required.”	Displays “All fields are required.”	Pass
3	Entering a password not tied to your account	Enter incorrect old password, new password and confirm password User clicks on Change Password button	Password: password New Password: P@ssword123 Confirm Password: P@ssword123	Displays “Password does not match!”	Displays “Password does not match!”	Pass
4	Entering a new password which does not meet password requirements	Enter old password, new password and confirm password User clicks on Change Password button	Password: P@ssword1 New Password: password Confirm Password: password	Displays “New password does not meet the password requirements. ”	Displays “New password does not meet the password requirements.”	Pass
5	Confirm password does not match with new password	Enter old password, new password and confirm password User clicks on Change Password button	Password: P@ssword1 New Password: P@ssword123 Confirm Password: P@ssword456	Displays “Please make sure your passwords match.”	Displays “Please make sure your passwords match.”	Pass
Test Case Status		Success				

6.2.1.1.5 Edit User Account Details

Test Case ID	**T-6.2.1.1.5**	**Test Case Priority**	High
Test Case Description	Allows the user to edit their account information under user profile page that can be accessed in sidebar		
Prerequisite	1. User must be logged in	**Postrequisite**	1. Updated details will be reflected on frontend and backend

Test Execution

#	Description	Action	Input	Expected Output	Actual Output	Test Result
1	Upload an image file to be profile picture	User clicks on profile picture icon User is prompted to upload an image file of any format (jpg, jpeg, png, svg) User submits the image of their choice	Image: User's selected image	Profile picture icon is updated with the newly uploaded image Snackbar appears and displays the message "Avatar updated successfully!"	Profile picture icon is updated with the newly uploaded image Snackbar appears and displays the message "Avatar updated successfully!"	Pass
2	Edit and Save Biography content on biography text box	User clicks on Edit button to enter editing mode Edit button is changed to Save button User enters new biography content User clicks on Save button to save changes	Old biography content: "Hey there! Nice to meet you!" New biography content: "Hey there! Nice to meet you! 12345"	Biography text box is replaced with the new content: "Hey there! Nice to meet you! 12345" Snackbar appears and displays the message "Biography saved!"	Biography text box is replaced with the new content: "Hey there! Nice to meet you! 12345" Snackbar appears and displays the message "Biography saved!"	Pass

3	Edit and Save username and phone number in About me section	User clicks on Edit button to enter editing mode Edit button is changed to Save button User enters new username and phone number User clicks on Save button to save changes	Old username: Hecarim Old phone number: 98765432 New username: Swain New phone number: 83453453	Username field is replaced with: “Swain” Phone number field is replaced with: “83453453” Snackbar appears and displays the message “Username and Phone Number saved!”	Username field is replaced with: “Swain” Phone number field is replaced with: “83453453” Snackbar appears and displays the message “Username and Phone Number saved!”	Pass
4	Edit and Save username but enter a phone number that is tied to another account in About me section	User clicks on Edit button to enter editing mode Edit button is changed to Save button User enters new username and a phone number belonging to another account User clicks on Save button to save changes	Old username: Swain Old phone number: 83453453 New username: Swain New phone number: 99999999	Snackbar appears and displays the message “Phone number has been taken!”	Snackbar appears and displays the message “Phone number has been taken!”	Pass
5	Allow user to change password	User clicks on Change Password button in About Me section	NIL	Navigate user to Change Password page	Navigate user to Change Password page	Pass
6	Allow user to delete account	User clicks on Delete Account button in About Me section	NIL	Delete account for user in the backend and	Delete account for user in the backend and log user out of the account	Pass

		Pop-up window appears to prompt user to delete account User presses “OK” button on window		log user out of the account Redirect user to home page	Redirect user to home page	
7						Pass
Test Case Status		Success				

6.2.1.2 Browsing

Test Case ID		**T-6.2.1.2**		**Test Case Priority**	High	
Test Case Description		To test if browsing categories show correct listings and behaviour based on category selection.				
Prerequisite		1. Listings exist in the system for various categories (Electronics, Supplies, Services), including at least one empty category.		**Postrequisite**	1. Listings display behaviour based on the selections.	
Test Execution						
#	**Description**	**Action**	**Input**	**Expected Output**	**Actual Output**	**Test Result**
1	Open listing categories from NavBar.	User clicks on a category in NavBar.	Select category (e.g., Electronics).	Page displays listings filtered by category, electronics	Listing shows category electronics	Pass
2	Verify empty category behaviour.	Click on a category with no listings.	Empty category selected.	"Showing 0 products in [Category]" displayed.	"Showing 0 products in [Category]" displayed.	Pass
3	Verify populated category behaviour.	Click on a populated category.	Populated category selected.	"Showing X products in [Category]" displayed.	"Showing X products in [Category]" displayed.	Pass

4	Click on a listing to confirm category correctness.	User selects a product.	Select any listed product.	Product matches the selected category.	Product matches the selected category.	Pass
5	Navigate back to browse listings via breadcrumb.	User clicks the breadcrumb link.	Click breadcrumb.	Returns to category listings view.	Returns to category listings view.	Pass
7	Select filtering options.	User selects filter options from the sidebar.	Choose rate types (e.g., hourly, daily, weekly, one time payment).	Filters are displayed with selected options highlighted.	Filters are displayed with selected options highlighted.	Pass
8	Verify rate selection behaviour.	User checks the desired rate boxes.	Select multiple rate types (e.g., hourly + daily).	Selected boxes reflect user's choice.	Selected boxes reflect user's choice.	Pass
9	Click the filter button to apply filters.	User clicks the "filter" button.	NIL	Listings filtered by selected rates are shown.	Listings filtered by selected rates are shown.	Pass
10	Verify filtered listings.	User clicks on a filtered listing.	Select a product.	Verify listing matches the filtered criteria.	Verify listing matches the filtered criteria.	Pass
11	Open the search bar.	User clicks on the search bar.	NIL	Search bar becomes active and ready for input.	Search bar becomes active and ready for input.	Pass
12	Input search letter.	User types a letter in the search bar.	“A”	Drop down menu appears showing listing titles with “A”	Drop down menu appears showing listing titles with “A”	Pass
13	Display search result.	Dropdown with hyperlink appears.	Matching title exists	Dropdown with hyperlink to the product is shown.	Dropdown with hyperlink to the product is shown.	Pass

14	Handle no match scenario.	User types any lettter in the search bar	“Anaconda” - No listing with this title	“no match found”	“no match found”	Pass
Test Case Status		Success				

6.2.1.3 Create Listing

Test Case ID	**T-6.2.1.3**	**Test Case Priority**	High
Test Case Description	Create Listing as a user who is logged in		
Prerequisite	3. User must be logged in 4. User must be at their Listing page	**Postrequisite**	

Test Execution

#	Description	Action	Input	Expected Output	Actual Output	Test Result
1	User cancels without submitting	1. User clicks on “Create Listing” 2. User clicks on “cancel” button	Nil	Form exits with no listing created. All inputs cleared.	Form exits with no listing created. All inputs cleared.	Pass
2	Submit empty form	1. User clicks on “Create Listing” 2.User clicks on “submit” without filing in the form	Nil	Display alert: "Please fill in all fields”	Display alert: "Please fill in all fields”	Pass
3	Submit form with only some required fields filled	1. User clicks on “Create Listing” 2. User fills in 4 out of the 8 fields 3. User clicks on “submit” button	Title: Test Listing Rate: 5 OneTime Category: Supplies Listing Type: Rental	Display alert: "Please fill in all fields”	Display alert: "Please fill in all fields”	Pass
4	Submit form with all required fields	1. User User clicks on “Create Listing” 2. User fills in all the required fields 3. User clicks on “submit” button	Title: Test Listing Rate: 5 OneTime Category: Supplies Listing Type: Rental	Form exits upon submission and new listing is automatically displayed on user’s listing page.	Form exits upon submission and new listing is automatically displayed on user’s listing page.	Pass

			Description: Test case 3 Photo: test.jpeg Location: Jurong Point Additional Notes: Meet be at the main entrance			
5	Submit form with all required field and multiple rates and locations	1. User User clicks on “Create Listing” 2. User fills in all the required fields, including adding 2 rates and 2 locations 3. User clicks on “submit” button	Title: Test Listing2 Rate: 5/ OneTime 50/ weekly Category: Supplies Listing Type: Rental Description: Test case 3 Photo: test.jpeg Location: Jurong Point westgate Additional Notes: Meet me at the main entrance	Form exits upon submission and new listing is automatically displayed on the user's listing page.	Form exits upon submission and new listing is automatically displayed on the user's listing page.	Pass
6	Submit Listing with Multiple Photos					Pass
Test Case Status		Success				

6.2.1.4 Offer

6.2.1.4.1 Make Offer

Test Case ID	**T-6.2.1.4.1**	**Test Case Priority**	High
Test Case Description	Make Offer on a selected listing		
Prerequisite	1. User must be logged in 2. User must select a listing from the browsing page	**Postrequisite**	
Test Execution			

#	Description	Action	Input	Expected Output	Actual Output	Test Result
1	Successfully made offer to a listing with time unit (D)	1. In the listing with “D” available, , user clicks on selects Time unit (D) 2. User clicks on “View Availability” 3. User selects start date, and end date 4. User confirms by clicking “Submit Availability” 5. User inputs desired amount 6. User clicks on “Make Offer”	Start Date: 3/11/2024 End Date: 3/11/2024 Amount: 300	Displays “Offer of S$ 300/D given. Total Price: S$300”	Displays “Offer of S$ 300/D given. Total Price: S$300”	Pass
2	Successfully made offer to a listing with time unit (H)	1. In the listing with “H” available, user clicks on selects Time unit (H) 2. User clicks on “View Availability” 3. User selects start date and start time, and end date and end time	Start Date: 1/11/2024 Start Time: 12 End Date: 1/11/2024 End Time: 18	Displays “Offer of S$ 10/H given. Total Price: S$60”	Displays “Offer of S$ 10/H given. Total Price: S$60”	Pass

		4. User confirms by clicking “Submit Availability” 5. User inputs desired amount 6. User clicks on “Make Offer”	Amount: 10			
3	Successfully made offer to a listing with time unit (W)	1. In the listing with “W” available, user clicks on selects Time unit (W) 2. User clicks on “View Availability” 3. User selects start date, and end date 4. User confirms by clicking “Submit Availability” 5. User inputs desired amount 6. User clicks on “Make Offer”	Start Date: 16/11/2024 End Date: 22/11/2024 Amount: 150	Displays “Offer of S$ 150/W given. Total Price: S$150”	Displays “Offer of S$ 150/W given. Total Price: S$150”	Pass
4	Successfully made offer to a listing with time unit (OT)	1. In the listing with “OT” available, user clicks on selects Time unit (OT) 2. User clicks on “View Availability” 3. User selects start date and start time, and end date and end time 4. User confirms by clicking “Submit Availability” 5. User inputs desired amount 6. User clicks on “Make Offer”	Start Date: 10/11/2024 Start Time: 12 End Date: 11/11/2024 End Time: 12 Amount: 70	Displays “Offer of S$ 70/OT given. Total Price: S$70”	Displays “Offer of S$ 70/OT given. Total Price: S$70”	Pass
5	Start date is later than end date	1. User selects any of the Time Unit available (H/D/W/OT)	Start Date: 10/11/2024 End Date: 5/11/2024	Displays “Start date/time cannot be later than End date/time”	Displays “Start date/time cannot be later than End date/time”	Pass

		2. User clicks on "View Availability" 3. User selects start date and end date 4. User confirms by clicking "Submit Availability"				
6	Start time is later than end time	1. In the listing with "H" or "OT" available, user clicks on selects any of the Time unit 2. User clicks on "View Availability" 3. User selects start date and start time, and end date and end time 4. User confirms by clicking "Submit Availability"	Start Date: 1/11/2024 Start Time: 18 End Date: 1/11/2024 End Time: 12	Displays "Start date/time cannot be later than End date/time"	Displays "Start date/time cannot be later than End date/time"	Pass
7	Start date is in the past	1. User selects any of the Time Unit available (H/D/W/OT) 2. User clicks on "View Availability" 3. User selects start date and end date 4. User confirms by clicking "Submit Availability"	Start Date: 15/10/2024 End Date: 17/10/2024 *Current Date: 20/10/2024	Displays "Start date cannot be in the past"	Displays "Start date cannot be in the past"	Pass
8	User cannot make offer to his own listing	1. User selects any of the Time Unit available in his listing (H/D/W/OT) 2. User clicks on "View Availability" 3. User selects start date/time and end date/time 4. User confirms by clicking "Submit Availability"	Start Date: 1/11/2024 Start Time: 12 End Date: 1/11/2024 End Time: 18	Displays "Error. You can not make an offer on your own listing"	Displays "Error. You can not make an offer on your own listing"	Pass

		5. User inputs desired amount 6. User clicks on “Make Offer”	Amount: 50			
9	Offer made with time unit “OT” must be less than 24 hours	1. In the listing with “OT” available, user clicks on selects Time unit (OT) 2. User clicks on “View Availability” 3. User selects start date and start time, and end date and end time 4. User confirms by clicking “Submit Availability”	Start Date: 10/11/2024 Start Time: 12 End Date: 11/11/2024 End Time: 13	Displays “Error. Must be less than or equals to 24 hours”	Displays “Error. Must be less than or equals to 24 hours”	Pass
10	Offer made with time unit “W” must be in multiple of 7	1. In the listing with “W” available, user clicks on selects Time unit (W) 2. User clicks on “View Availability” 3. User selects start date, and end date 4. User confirms by clicking “Submit Availability”	Start Date: 16/11/2024 End Date: 23/11/2024	Displays “Error. Days must be in multiple of 7”	Displays “Error. Days must be in multiple of 7”	Pass
11	Missing Start Date	1. User selects any of the Time Unit available in his listing (H/D/W/OT) 2. User clicks on “View Availability” 3. User selects Start time and end date/time 4. User confirms by clicking “Submit Availability”	Start Time: 12 End Date: 1/11/2024 End Time: 18	Displays “Missing input, please input Start Date”	Displays “Missing input, please input Start Date”	Pass
12	Missing Start Time	1. In the listing with “H” or “OT” available, user	Start Date: 1/11/2024	Displays “Missing input,	Displays “Missing input,	Pass

		clicks on selects any of the Time unit 2. User clicks on “View Availability” 3. User selects Start date and end date/time 4. User confirms by clicking “Submit Availability”	End Date: 1/11/2024 End Time: 18	please input Start Time”	please input Start Time”	
13	Missing End Date	1. User selects any of the Time Unit available in his listing (H/D/W/OT) 2. User clicks on “View Availability” 3. User selects Start date/time and end time 4. User confirms by clicking “Submit Availability”	Start Date: 1/11/2024 Start Time: 12 End Time: 18	Displays “Missing input, please input End Date”	Displays “Missing input, please input End Date”	Pass
14	Missing End Time	1. In the listing with “H” or “OT” available, user clicks on selects any of the Time unit 2. User clicks on “View Availability” 3. User selects Start date/time and end date 4. User confirms by clicking “Submit Availability”	Start Date: 1/11/2024 Start Time: 12 End Date: 1/11/2024	Displays “Missing input, please input End Time”	Displays “Missing input, please input End Time”	Pass
15	Missing Availability	1. User selects any of the Time Unit available in his listing (H/D/W/OT) 2. User inputs desired amount 3. User clicks on “Make Offer”	Amount: 50	Displays “Error: Please set availability!”	Displays “Error: Please set availability!	Pass

16	Missing Amount	1. User selects any of the Time Unit available in his listing (H/D/W/OT) 2. User clicks on “View Availability” 3. User selects Start time and end date/time 4. User confirms by clicking “Submit Availability” 5. User clicks on “Make Offer”	Start Date: 10/11/2024 Start Time: 12 End Date: 11/11/2024 End Time: 12	Displays “Please enter an amount”	Displays “Please enter an amount”	Pass
Test Case Status		Success				

6.2.1.4.2 Accept and Decline Offer

Test Case ID	**T-6.2.1.4.2**	**Test Case Priority**	High
Test Case Description	Allows user to accept or decline offer on a selected listing		
Prerequisite	1. User must be logged in. 2. User must have been offered by another user on any of the logged in user’s listing.	**Postrequisite**	

Test Execution

#	Description	Action	Input	Expected Output	Actual Output	Test Result
1	Accept Offer	1. User goes to “Offers page” 2. User selects any of the available listings under “Offers received” 3. User accepts an offer by clicking on accept button	N/A	Offer accepted and displayed from “Pending Offer” to under “Accepted Offers”	Offer accepted and displayed from “Pending Offer” to under “Accepted Offers”	Pass
2	Decline Offer	1. User goes to “Offers page”	N/A	Offer declined and displayed removed from	Offer declined and displayed	Pass

		2. User selects any of the available listings under "Offers received" 3. User declines an offer by clicking on decline button		"Pending Offers"	removed from "Pending Offers"	
3	No listings to select	1. User goes to "Offers page"	N/A	Displays "No listings available"	Displays "No listings available"	Pass
4	No offers to accept or decline	1. User goes to "Offers page" 2. User selects any of the available listings under "Offers received"	N/A	Nothing displayed under "Pending Offers"	Nothing displayed under "Pending Offers"	Pass
Test Case Status		Success				

6.2.1.5 Payment

Test Case ID	**T-6.2.1.5**	**Test Case Priority**	High
Test Case Description	User makes payment for an offer that he made, and has been accepted by the user that listed the service/product.		
Prerequisite	1. User must be logged in 2. User must have made an offer to another user's listing 3. Said offer must have been accepted by the user who made the listing	**Postrequisite**	1. Transaction created after successful payment
Test Execution			

#	Description	Action	Input	Expected Output	Actual Output	Test Result
1	User makes payment successfully via Paypal	1. User goes to "Offers" page 2. User selects any of the available listings under "Offers made" 3. User chooses an offer under "Offers Pending Payment" 4. User makes payment for an offer by clicking on the "Paypal" button 5. User inputs valid credentials into the pop-up and clicks "Login" 6. User selects a payment option and makes payment by clicking "Pay"	Email: sb-yw3r232752961@personal.example.com Password: m&0IKVb= *Payment amount is $70	Displays "Transaction of S$70 successful"	Displays "Transaction of S$70 successful"	Pass
2	User enters invalid Paypal credentials	1. User goes to "Offers" page 2. User selects any of the available listings under "Offers made"	Email: sb-yw3r232752961@personal.example.com Password:	Paypal pop-up displays "Please check your entries and try again"	Paypal pop-up displays "Please check your entries and try again"	Pass

		3. User chooses an offer under “Offers Pending Payment” 4. User makes payment for an offer by clicking on the “Paypal” button 5. User inputs credentials into the pop-up and clicks “Login”	aaaa			
3	User makes payment with valid paypal credentials, but has insufficient balance	1. User goes to “Offers” page 2. User selects any of the available listings under “Offers made” 3. User chooses an offer under “Offers Pending Payment” 4. User makes payment for an offer by clicking on the “Paypal” button 5. User inputs credentials into the pop-up and clicks “Login” 6. User selects a payment option and makes payment by clicking “Pay”	Email: sb-yw3r232752961@personal.example.com Password: m&0IKVb= *Bank balance is 0	User is prompted to retry payment with different payment methods.	User is prompted to retry payment with different payment methods.	Pass
4	User exits paypal popup halfway through	1. User goes to “Offers” page 2. User selects any of the available listings under “Offers made” 3. User chooses an offer under “Offers Pending Payment”	Email: sb-yw3r232752961@personal.example.com Password: m&0IKVb=	Offer selected still remains under “Offer Pending Payment”	Offer selected still remains under “Offer Pending Payment”	Pass

		4. User makes payment for an offer by clicking on the “Paypal” button 5. User exits the pop-up page				
Test Case Status		Success				

6.2.1.6 Review

Test Case ID	**T-6.2.1.6**	**Test Case Priority**	Medium
Test Case Description	Allow users to make review on completed transactions		
Prerequisite	1. User must be logged in 2. Transaction must have been completed	**Postrequisite**	1. Review is displayed on the profile of the user who provided the service

Test Execution

#	Description	Action	Input	Expected Output	Actual Output	Test Result
1	Successful review	1. User goes to “Purchases” page 2. User clicks on toggle switch to select “Completed” 3. User chooses a transaction to review by clicking “Rate” button 4. User fills in description and star rating on the pop-up dialog. 5. User clicks submit.	Star rating: 5 Description: Awesome seller!	Displays “Review submitted”	Displays “Review submitted”	Pass
2	Unsuccessful due to missing star rating input	1. User goes to “Purchases” page 2. User clicks on toggle switch to select “Completed” 3. User chooses a transaction to review by clicking “Rate” button	Description: Awesome seller!	Displays “Please input a star rating”	Displays “Please input a star rating”	Pass

		4. User fills in the description on the pop-up dialog. 5. User clicks submit.				
3	Unsuccessful due to missing description	1. User goes to “Purchases” page 2. User clicks on toggle switch to select “Completed” 3. User chooses a transaction to review by clicking “Rate” button 4. User fills in the star rating on the pop-up dialog. 5. User clicks submit.	Star rating: 5	Displays “Please input a review!”	Displays “Please input a review!”	Pass
4	User exits halfway through review	1. User goes to “Purchases” page 2. User clicks on toggle switch to select “Completed” 3. User chooses a transaction to review by clicking “Rate” button 4. User fills in description and star rating on the pop-up dialog. 5. User clicks cancel.	Star rating: 5 Description: Awesome seller!	Dialog box is closed and review is not submitted	Dialog box is closed and review is not submitted	Pass
5	No transactions available	1. User goes to “Purchases” page 2. User clicks on toggle switch to select “Completed”	N/A	Displays: “No transactions completed found!”	Displays: “No transactions completed found!”	Pass
Test Case Status		Success				

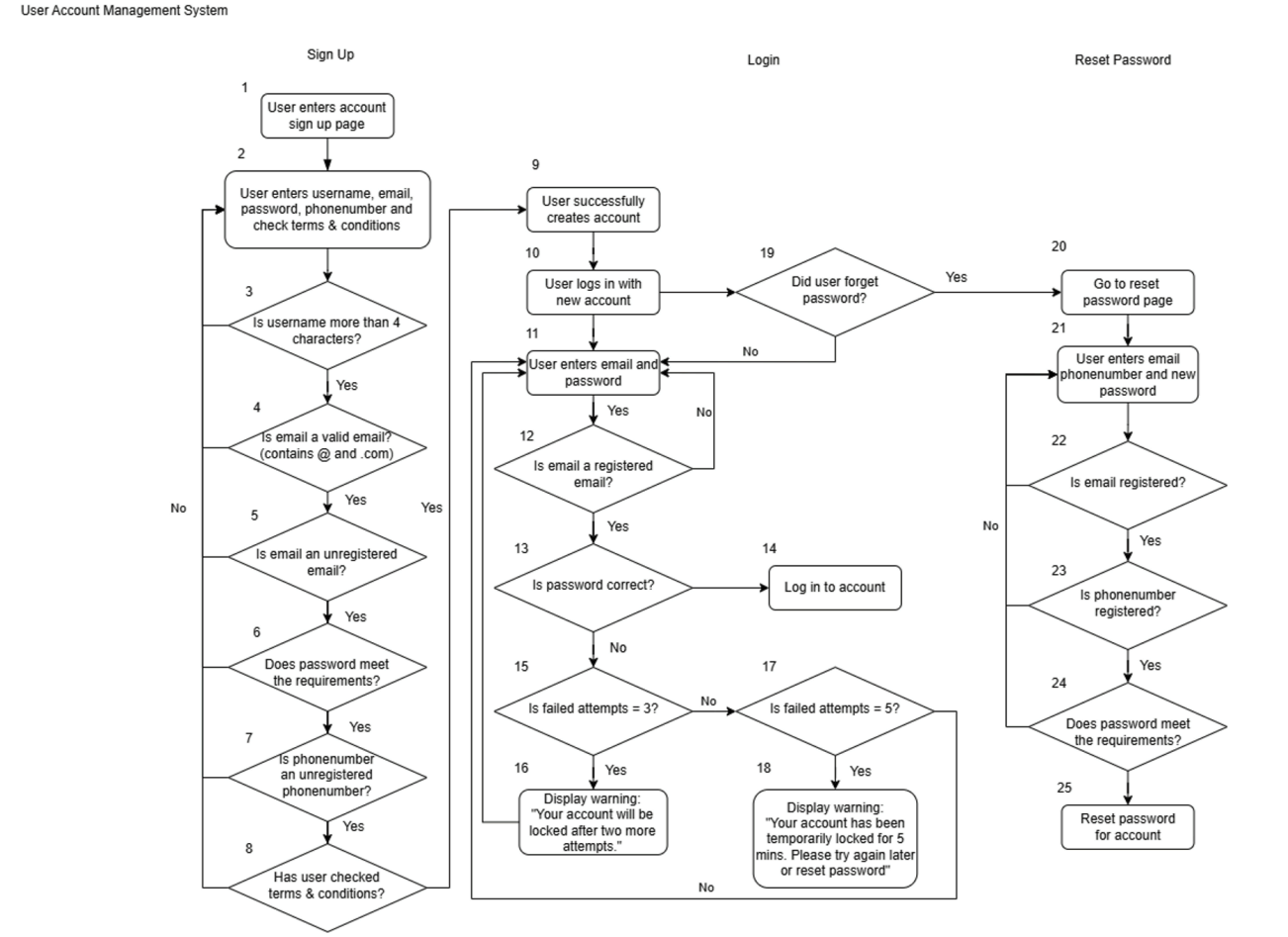
6.2.2 White Box Testing

6.2.2.1 User Account Management System

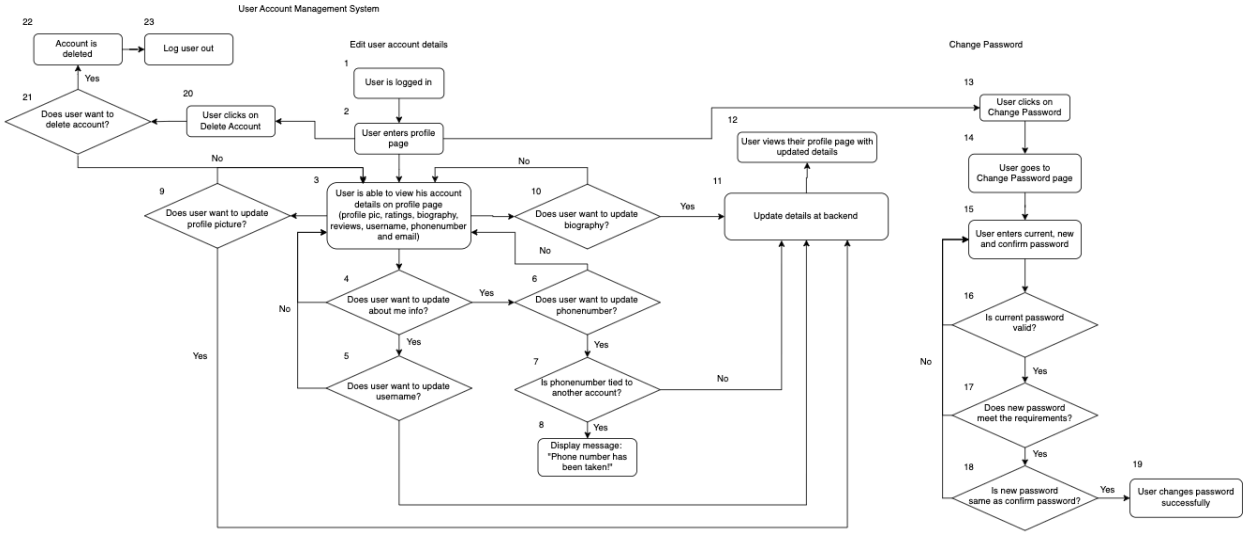
A fundamental feature for any system is comprehensive user account management, adhering to CRUD principles. In Shopblock, users will have access to essential account functionalities, including account registration, login, password reset (for forgotten passwords), password change (when logged in), and the ability to update or edit their account information (such as username, phone number, and other profile details). The system will incorporate robust validation checks to ensure entered details meet specified requirements and to prevent duplicate accounts from being created with existing information.

6.2.2.1 Control Flow Graph

6.2.2.1.1 Sign Up 6.2.2.1.2 Login 6.2.2.1.3 Reset Password



6.2.2.1.4 Edit Account Details 6.2.2.1.5 Change Password



6.2.2.1 Basic Path Testing

Cyclomatic Complexity = | decision points | + 1
Sign Up = 7
Login = 6
Reset Password = 4
Edit User Account Details = 8
Change Password = 4

Basic Paths

Sign Up:

Baseline path 1:
1 → 2 → 3 → 4 → 5 → 6 → 7 → 8 → 9

Login:

Baseline path 1:
10 → 11 → 12 → 13 → 14

Reset Password:

Baseline path 1:

19 → 20 → 21 → 22 → 23 → 24 → 25

Edit User Account Details:

Baseline path 1: (Update biography)

1 → 2 → 3 → 10 → 11 → 12

Baseline path 2: (Update profile picture)

1 → 2 → 3 → 9 → 11 → 12

Baseline path 3: (Update username)

1 → 2 → 3 → 4 → 5 → 11 → 12

Baseline path 4: (Update phone number)

1 → 2 → 3 → 4 → 6 → 7 → 11→ 12

Baseline path 5: (Delete account)

1 → 2 → 20 → 21 → 22 → 23

Change Password:

Baseline path 1:

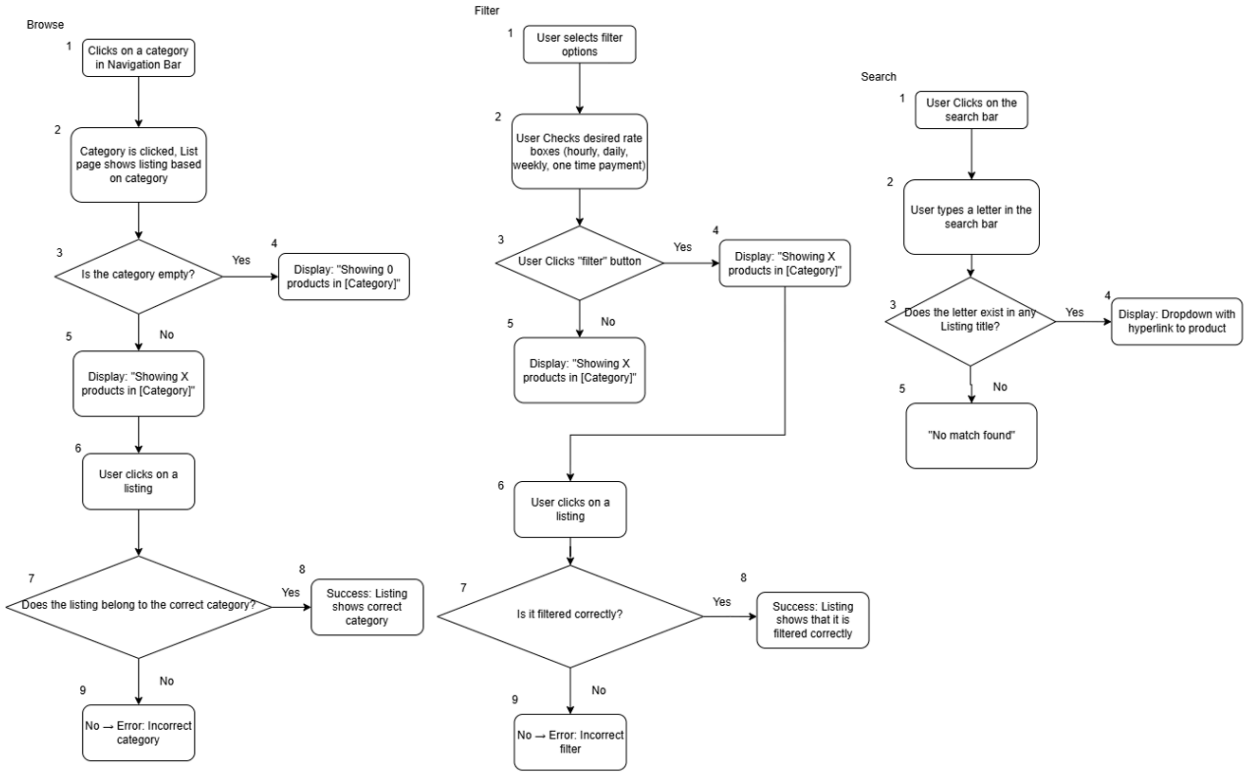
13 → 14 → 15 → 16 → 17 → 18 → 19

6.2.2.2 Browsing

The browse feature is a key component of the ShopBlock application, serving as users' first interaction with the platform. It allows users to explore various listings created by other users. A search bar is available to help customers quickly find specific listings of interest. Additionally, the navigation bar provides category-based filtering, while a filter sidebar further refines searches based on rates and price, enhancing the browsing experience.

6.2.1.2 Control Flow Graph

6.2.1.2.1 Browse 6.2.1.2.2 Filter 6.2.1.2.3 Search



6.2.1.2 Basic Path Testing

Cyclomatic Complexity = | decision points | + 1
Browse: 3
Filter: 3
Search: 2

Basic Paths

Browse:

Baseline Path 1:
1 → 2 → 3 → 5 → 6 → 7 → 8
Baseline Path 2:
1 → 2 → 3 → 4
Baseline Path 3:
1 → 2 → 3 → 5 → 6 → 7 → 9

Filter:

Baseline Path 1:
1 → 2 → 3 → 4 → 6 → 7 → 8
Baseline Path 2:
1 → 2 → 3 → 5 → 6 → 7 → 9
Baseline Path 3:
1 → 2 → 3 → 4 → 6 → 7 → 9

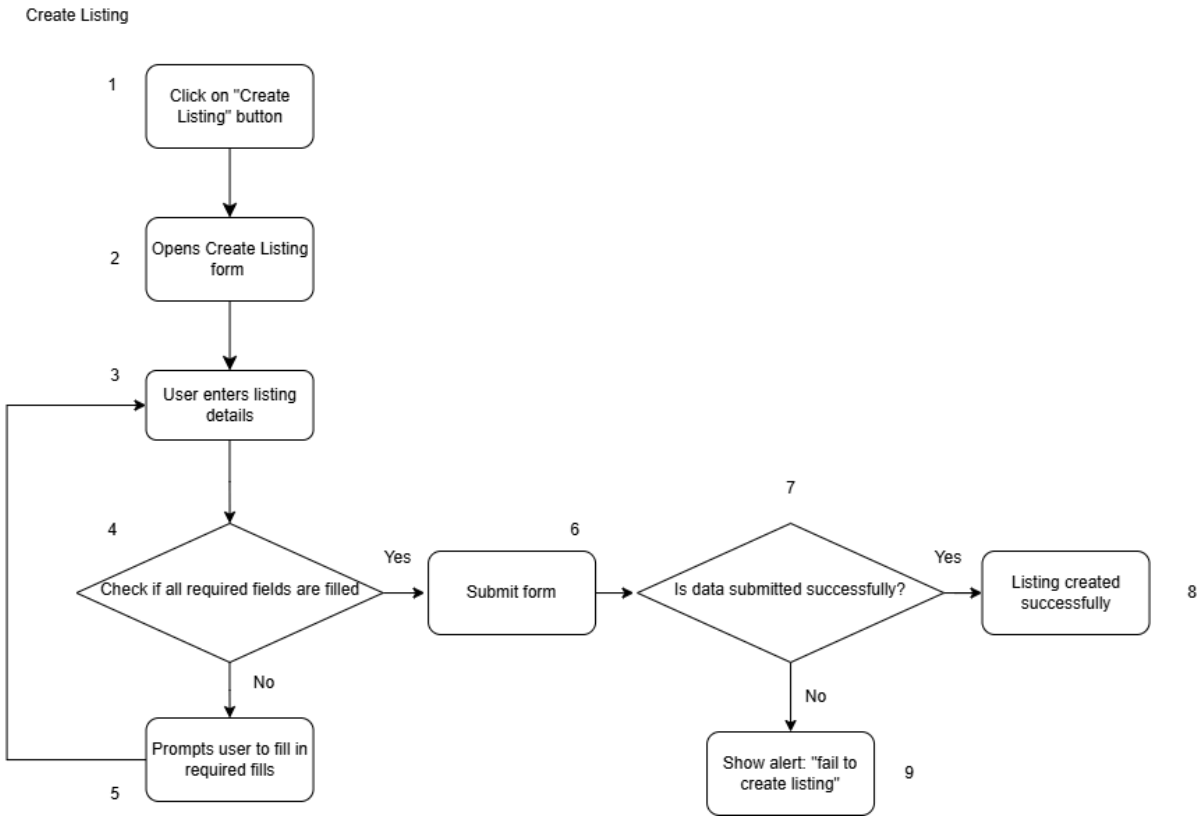
Search:

Baseline Path 1:
1 → 2 → 3 → 4
Baseline Path 2:
1 → 2 → 3 → 5

6.2.2.3 Listing

The listing feature plays a vital role in the ShopBlock application, enabling users to create and publish their own listings. When posting a listing, users are required to provide key details such as the title, price with its unit, category, listing type, location, description, and an accompanying photo.

6.2.2.3 Control Flow Graph



6.2.2.3 Basic Path Testing

Cyclomatic Complexity = | decision points | + 1 = [fill me]
Cyclomatic Complexity=2+1=3

Basic Paths

Baseline path 1:
1 → 2 → 3 → 4 → 6 → 7 → 8
Baseline path 2:
1 → 2 → 3 → 4 → 5 → 3

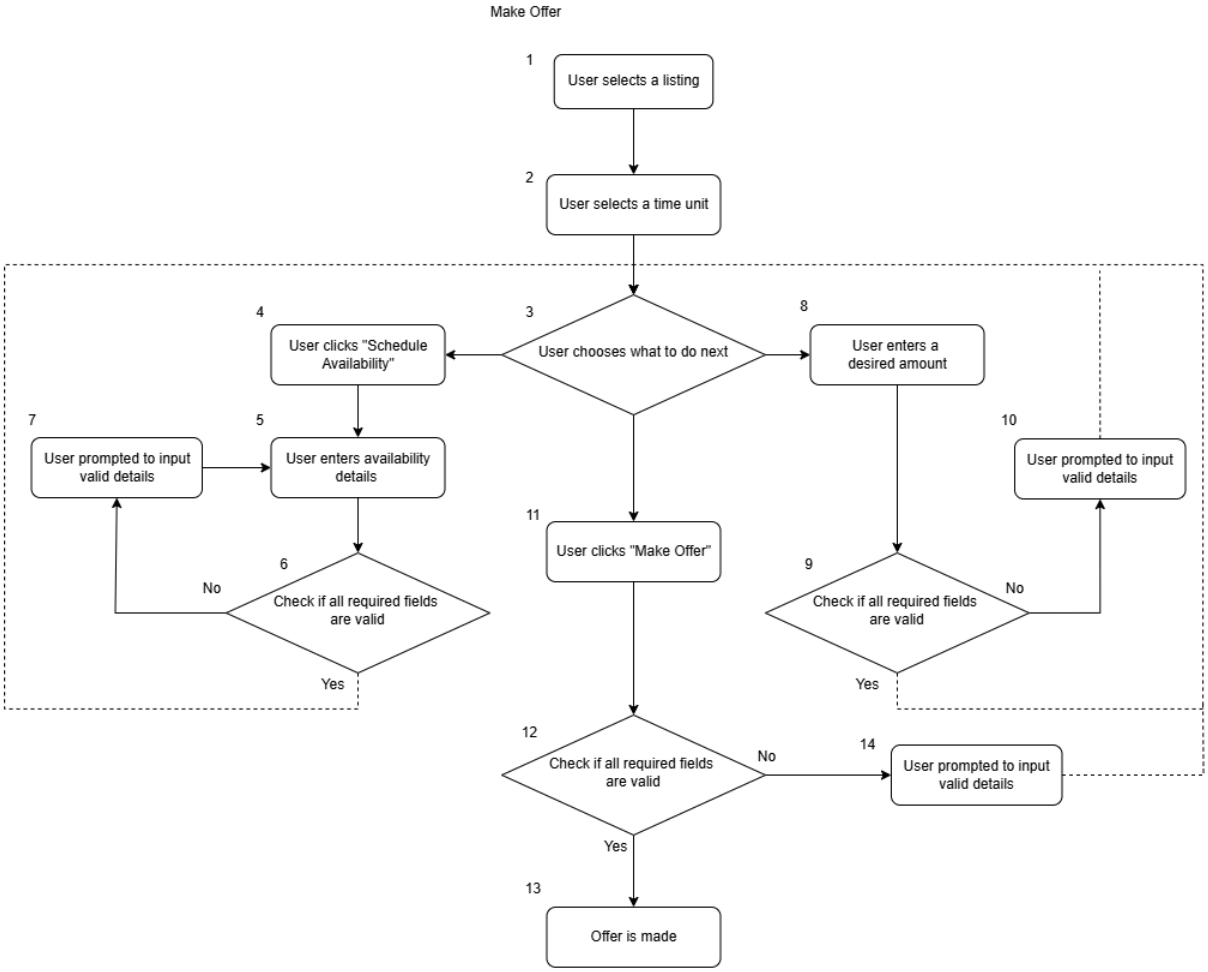
Baseline path 3:
1 → 2 → 3 → 4 → 6 → 7 → 9

6.2.2.4 Offer

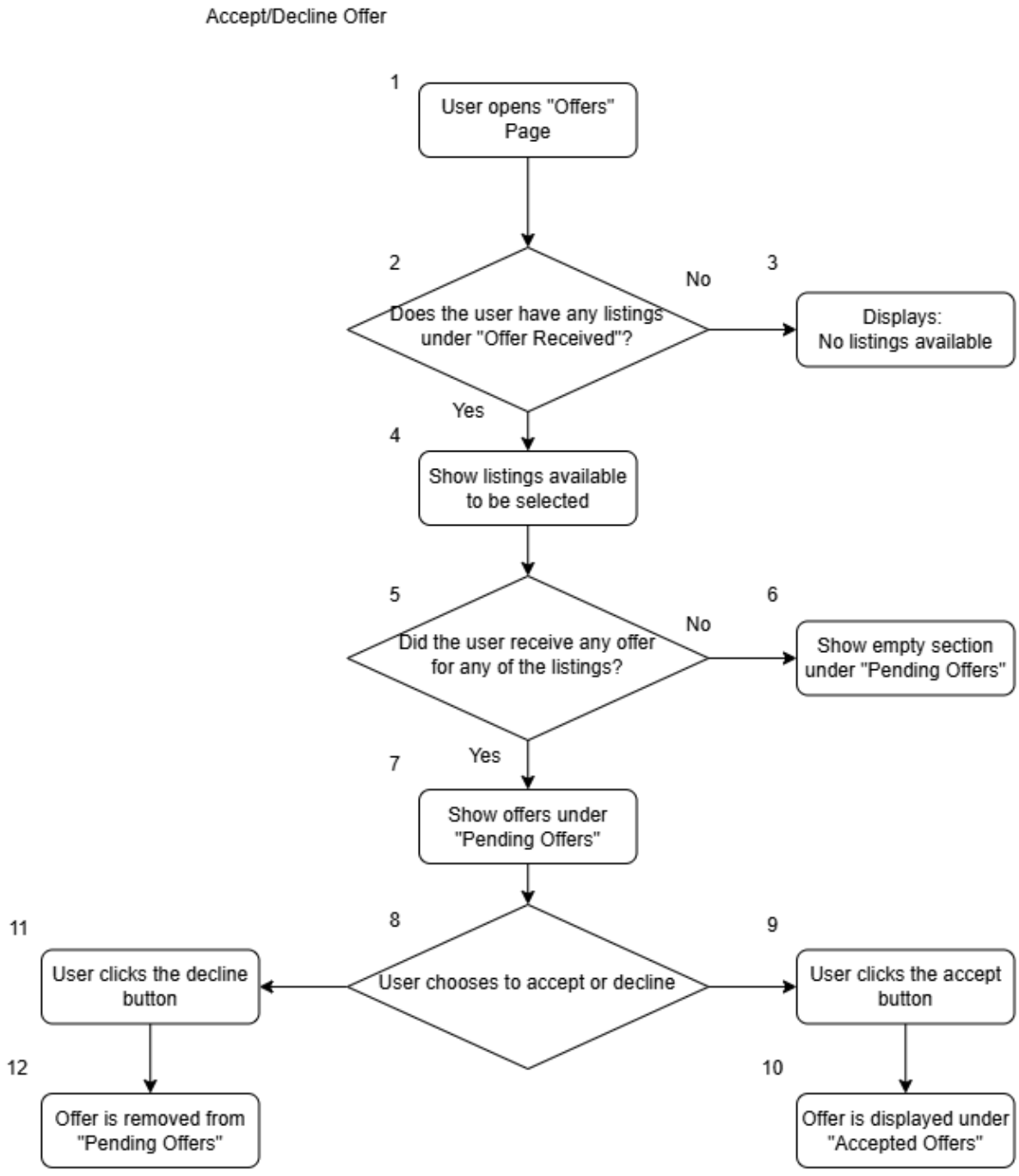
After browsing the listings, ShopBlock provides a "Make Offer" feature designed to help low-income families negotiate better deals. Customers can submit an offer on a listing, which will then be sent directly to the listing owner for review and consideration.

6.2.2.4 Control Flow Graph

6.2.2.4.1 Make Offer



6.2.2.4.2 Accept/Decline Offer



6.2.2.4 Basic Path Testing

Cyclomatic Complexity = | decision points | + 1 = [fill me]
Cyclomatic Complexity for “Make Offer” = 4+1 = 5
Cyclomatic Complexity for “Accept/Decline Offer” = 3+1 = 4

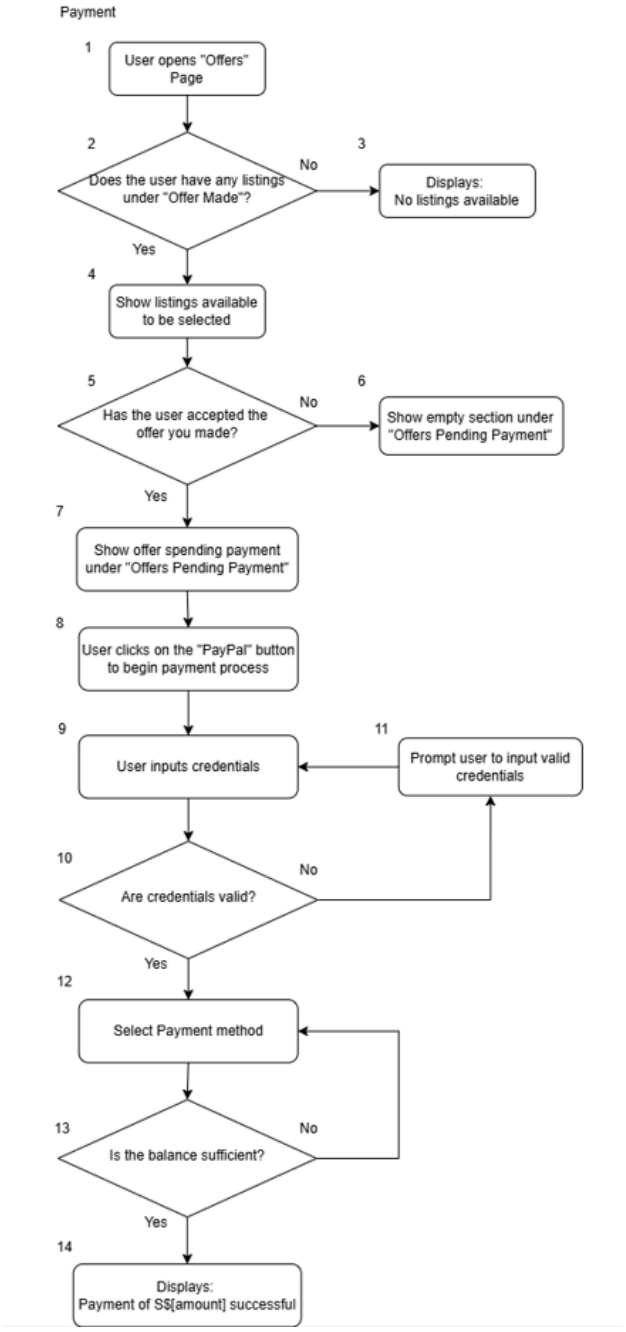
Basic Paths
Make Offer:
Baseline path 1:
1 → 2 → 3 → 4 → 5 → 6 → 3 → 8 → 9 → 3 → 11 → 12 → 13
Baseline path 2:
1 → 2 → 3 → 4 → 5 → 6 → 5
Baseline path 3:
1 → 2 → 3 → 8 → 9 → 10 → 3
Baseline path 4:
1 → 2 → 3 → 11 → 12 → 14 → 3

Accept/Decline Offer:
Baseline path 1:
1 → 2 → 3
Baseline path 2:
1 → 2 → 4 → 5 → 6
Baseline path 3:
1 → 2 → 4 → 5 → 7 → 8 → 9 → 10
Baseline path 4:
1 → 2 → 4 → 5 → 7 → 8 → 11 → 12

6.2.2.5 Payment

Allows a user to make payment on an accepted offer that he has made previously. This allows the offer to be confirmed and valid, making it a transaction both renter and rentee has to uphold.

6.2.2.5 Control Flow Graph



6.2.2.5 Basic Path Testing

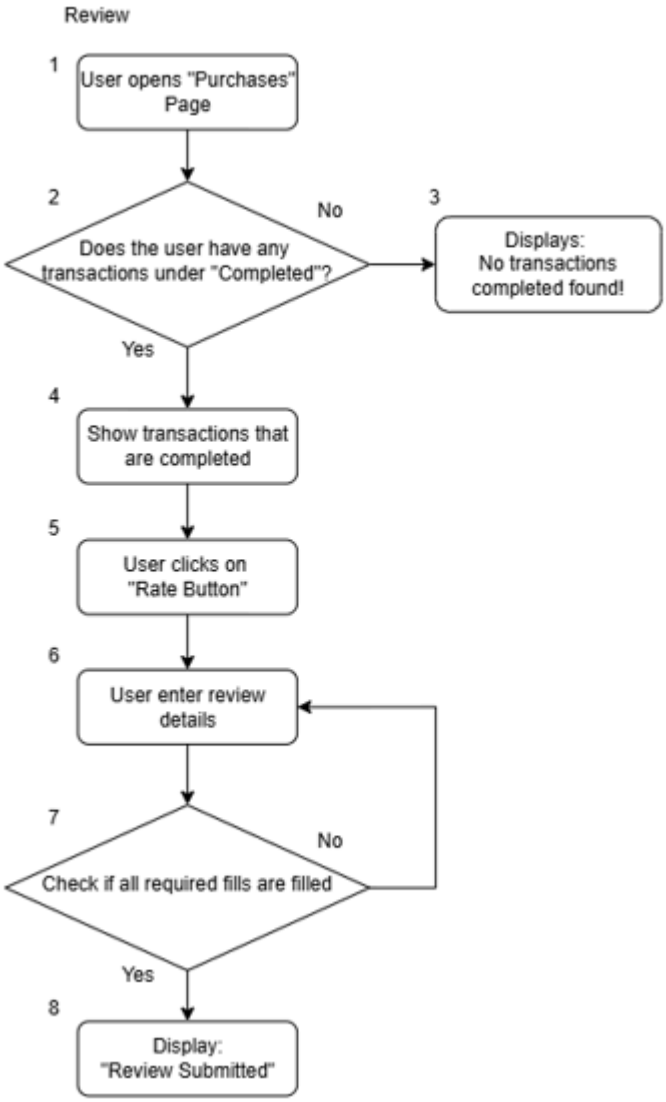
Cyclomatic Complexity = | decision points | + 1 = [fill me]
Cyclomatic Complexity = 4 + 1 = 5

Basic Paths
Baseline path 1:
1 → 2 → 3
Baseline path 2:
1 → 2 → 4 → 5 → 6
Baseline path 3:
1 → 2 → 4 → 5 → 7 → 8 → 9 → 10 → 11 → 9
Baseline path 4:
1 → 2 → 4 → 5 → 7 → 8 → 9 → 10→ 12 → 13 → 12
Baseline path 5:
1 → 2 → 4 → 5 → 7 → 8 → 9 → 10→ 12 → 13 → 14

6.2.2.6 Review

The Review feature enables users to provide both a rating and written feedback for another user from whom they have purchased services or products.

6.2.2.6 Control Flow Graph



6.2.2.6 Basic Path Testing

Cyclomatic Complexity = | decision points | + 1 = [fill me]
Cyclomatic Complexity = 2 + 1 = 3

Basic Paths

Baseline path 1:

1 → 2 → 3

Baseline path 2:

1 → 2 → 4 → 5 → 6 → 7 → 6

Baseline path 3:

1 → 2 → 4 → 5 → 6 → 7 → 8

6.2.3 API Testing

The software design of ShopBlock is heavily dependent on our RESTful backend API. Almost all functionality is done with a request to the backend in some form. This backend is crucial to the application functioning for the users as intended.

To ensure that the application functions as intended, we made use of unit testing to ensure that the APIs will take in the inputs as intended and return the outputs as intended as well. We tested all the endpoints that are exposed to the front-end.

The tests here are automated. These tests are also continuously run in our repository's continuous integration pipeline to ensure that anytime the tests fail - the developers are notified and can fix them as required.

6.2.3.1 User

We will test all functionality related to the user here.

6.2.3.1.1 User Registration

Test Case ID	**T-6.2.3.1.1**		
Endpoint	POST `/user`		
Endpoint Description	The POST endpoint will register a new user.		
Test Execution			

#	HTTP Method, Description	Header	Body	Params	Expected Status	Pass/Fail
1	POST, Register a new user	nil	{ "username": "testuser", "email": "test@example.com", "password": "testpass123", "phone_number": "88888888", "avatar": Image, "biography": "Test biography",	nil	HTTP 201 (Created)	Pass

			}			

2	POST,
	
	Register with an existing email in the system
		nil	{
			"username": "newuser",
			"email": "existing@example.com", # Using existing email
			"password": "newpass123",
			"phone_number": "99999999",
			}
				nil	HTTP 400 (Bad Request)	Pass

3	POST,
	
	Register with incomplete data
		nil	{
			"username": "logintest",
			"email": "login@example.com",
			"password": "loginpass123",
			"phone_number": "88888888",
			"biography": "",
			}
				nil	HTTP 400 (Bad Request)	Pass

Test Case Status		Success				

6.2.3.1.2 User Login

Test Case ID	**T-6.2.3.1.2**		
Endpoint	POST `/api/login`		
Endpoint Description	The POST endpoint will take in the details of the user, and return a JWT to the user if the details are correct and return an error if not.		
Test Execution			

#	HTTP Method, Description	Header	Body	Params	Expected Status	Pass/Fail

1	POST, Correct Credentials	nil	{"email": "login@example.com", "password": "loginpass123"}	nil	HTTP 200 (Ok)	Pass
2	POST, Wrong password	nil	{"email": "nonexistent@example.com" , "password": "somepass"}	nil	HTTP 401 (Unauthorized)	Pass
3	POST, Non existent user	nil	{"email": "nonexistent@example.com" , "password": "somepass"}	nil	HTTP 401 (Unauthorized)	Pass
Test Case Status		Success				

6.2.3.1.3 Get User Details

Test Case ID	**T-6.2.3.1.3**		
Endpoint	GET `/user`		
Endpoint Description	The GET endpoint will return the details of a user.		
Test Execution			

#	HTTP Method, Description	Header	Body	Params	Expected Status	Pass/Fail
1	GET, Get details of the JWT user	Authorization : Bearer Token	nil	nil	HTTP 200 (Ok)	Pass
2	GET, No JWT Token	nil	nil	nil	HTTP 401 (Unauthorized)	Pass
3	GET,	nil	nil	?id=3	HTTP 200 (Ok)	Pass

	Get details of the user in the query param					
4	GET, Get details of a non-existent user	nil	nil	?id=999	HTTP 404 (Not Found)	Pass
Test Case Status		Success				

6.2.3.1.4 Update User Details

Test Case ID	**T-6.2.3.1.4**		
Endpoint	PUT `/user`		
Endpoint Description	The PUT endpoint allows an authenticated user to update their details.		
Test Execution			

#	HTTP Method, Description	Header	Body	Params	Expected Status	Pass/Fail

1	PUT,
	
	Update user with new details
		Authorization : Bearer Token	{
			"username": "updateduser",
			"phone_number": "44444444",
			"avatar": new_avatar,
			"biography": "Updated bio",
			"password": "update123", # Current password
			"new_password": "newpass123",
			}
				nil	HTTP 200 (Ok)	Pass

2	PUT,
	
	Update user details with no authorization token
		nil	{
			"username": "updateduser",
			"phone_number": "44444444",
			"avatar": new_avatar,
			"biography": "Updated bio",
			"password": "update123", # Current password
			"new_password": "newpass123",
			}
				nil	HTTP 401 (Unauthorized)	Pass

3	PUT,
	
	Update password, but input the wrong current password
		Authorization : Bearer Token	{
			"password": "wrongpass",
			"new_password": "newpass123",
			}
				nil	HTTP 400 (Bad Request)	Pass

4	PUT,
	
	Missing fields for updating password
		Authorization : Bearer Token	{
			"username": "updateduser",
			"new_password": "newpass123", # Missing current password
			}
				nil	HTTP 400 (Bad Request)	Pass

5	PUT, Partial update of profile, only update username	Authorization : Bearer Token	{"username": "newusername"}	nil	HTTP 200 (Ok)	Pass
6	PUT, Partial update of profile, only update phone number	Authorization : Bearer Token	{"phone_number": "66666666"}	nil	HTTP 200 (Ok)	Pass
7	PUT,	Authorization : Bearer Token	{"biography": "New bio"	nil	HTTP 200 (Ok)	Pass

	Partial update of profile, only update biography					
Test Case Status	Success					

6.2.3.1.5 Delete User

Test Case ID	**T-6.2.3.1.5**				
Endpoint	DELETE `/user`				
Endpoint Description	The DELETE endpoint allows an authenticated user to delete their account.				
Test Execution					
#	**HTTP Method, Description**	**Header**	**Body**	**Params**	**Expected Status**	**Pass/Fail**
1	DELETE, Delete user details with authorization	Authorization : Bearer Token	nil	nil	HTTP 200 (Ok)	Pass
2	DELETE, Delete user details without authorization	nil	nil	nil	HTTP 401 (Unauthorized)	Pass
Test Case Status	Success					

6.2.3.2 Listing

We will test all functionality related to the listing here.

6.2.3.2.1 Get Listing Details

Test Case ID	**T-6.2.3.2.1**		
Endpoint	GET `/listing`		
Endpoint Description	The GET endpoint will return details related to the listing.		
Test Execution			

#	HTTP Method, Description	Header	Body	Params	Expected Status	Pass/Fail
1	GET, Retrieve all listings	nil	nil	nil	Returns an array of Listings HTTP 200 (Ok)	Pass
2	GET, Retrieve a single listing	nil	nil	?id=2	Returns the details of listing with id 2 HTTP 200 (Ok)	Pass
3	GET, Retrieve a non existing listing	nil	nil	?id=999	Listing not found HTTP 404 (Not Found)	Pass
4	GET, Retrieve listing with a search parameter	nil	nil	?search=Drill	Returns the listing with a Drill in the title HTTP 200 (Ok)	Pass
5	GET, Retrieve listing with a category parameter	nil	nil	?category=Services	Returns services listings only HTTP 200 (Ok)	Pass

6	GET, Retrieve listing with a listing type parameter	nil	nil	?listing_type=Rental	Returns rental listings only HTTP 200 (Ok)	Pass
7	GET, Retrieve listing while sorting by price	nil	nil	?time_unit=Hourly&sort_by=price_asc	Returns listings with only the hourly time unit and sorted by price HTTP 200 (Ok)	Pass
8	GET, Retrieve listing while sorting by price, but did not provide time unti	nil	nil	?sort_by=price_asc	Time unit must be specified when sorting by price HTTP 400 (Bad Request)	Pass
Test Case Status		Success				

6.2.3.2.2 Create Listing

Test Case ID	**T-6.2.3.2.2**		
Endpoint	POST `/listing`		
Endpoint Description	The POST endpoint will allow a user to create a listing		
Test Execution			

#	HTTP Method, Description	Header	Body	Params	Expected Status	Pass/Fail
1	POST, Create a listing with single	Authorization: Bearer Token	{ "title": "New Test Listing", "description": "Test description",	nil	HTTP 201 (Created)	Pass

	location and single rates		"category": Category.SUPPLIES, "listing_type": ListingType.RENTAL, "photos": [get_blank_photo()], "rates": '[{"time_unit": "D", "rate": "25.00"}]', "locations": '[{"latitude": 1.35160, "longitude": 103.87119, "query": "Nex", "notes": "Test location"}]', "uploaded_by": self.user1.id, }			

2	POST,
	
	Create a listing with incomplete details
		Authorization : Bearer Token	{
			"title": "Incomplete Listing",
			# missing other required fields
			}
				nil	HTTP 400 (Bad Request)	Pass

3	POST, Create a listing with multiple location and single rates	Authorization : Bearer Token	{ "title": "Multi-Location Test", "description": "Test listing with multiple locations", "category": Category.SUPPLIES, "listing_type": ListingType.RENTAL, "photos": [get_blank_photo()], "rates": '[{"time_unit": "D", "rate": "25.00"}]', "locations": '[{"latitude": "1.35160", "longitude": "103.87119", "query": "Nex", "notes": "Location 1"}, {"latitude": "1.31745", "longitude": "103.80704", "query": "Farrer Road", "notes": "Location 2"}]',	nil	HTTP 201 (Created)	Pass

			"uploaded_by": self.user1.id, }			

4	POST,
	
	Create a listing with single location and multiple rates
		Authorization : Bearer Token	{
			"title": "Multi-Rate Test",
			"description": "Test listing with multiple rates",
			"category": Category.SUPPLIES,
			"listing_type": ListingType.RENTAL,
			"photos": [get_blank_photo()],
			"rates": '[{"time_unit": "H", "rate": "10.00"}, {"time_unit": "D", "rate": "50.00"}, {"time_unit": "W", "rate": "250.00"}]',
			"locations": '[{"latitude": "1.35160", "longitude": "103.87119", "query": "Nex", "notes": "Single location"}]',
			"uploaded_by": self.user1.id,
			}
				nil	HTTP 201 (Created)	Pass

5	POST, Create a listing with multiple locations and multiple rates	Authorization : Bearer Token	{ "title": "Multi-Location-Rate Test", "description": "Test listing with multiple locations and rates", "category": Category.SUPPLIES, "listing_type": ListingType.RENTAL, "photos": [get_blank_photo()], "rates": '[{"time_unit": "H", "rate": "15.00"}, {"time_unit": "D", "rate": "75.00"}, {"time_unit": "W", "rate": "350.00"}]', "locations": '[{"latitude": "1.35160",	nil	HTTP 201 (Created)	Pass

			"longitude": "103.87119", "query": "Nex", "notes": "Location 1"}, {"latitude": "1.31745", "longitude": "103.80704", "query": "Farrer Road", "notes": "Location 2"}, {"latitude": "1.42953", "longitude": "103.83503", "query": "Yishun", "notes": "Location 3"}]', "uploaded_by": self.user1.id, }			
Test Case Status	Success					

6.2.3.2.3 Create Listing

Test Case ID	**T-6.2.3.2.3**		
Endpoint	PUT `/listing`		
Endpoint Description	The PUT endpoint will allow the user who created the listing to update the listing details.		
Test Execution			

#	HTTP Method, Description	Header	Body	Params	Expected Status	Pass/Fail
1	PUT, Update the listing details with valid data	Authorization : Bearer Token	{ "id": self.listing1.id, "title": "Updated Drill", "description": "Updated description", "category": Category.ELECTRONICS, "listing_type": ListingType.RENTAL, "rates": '[{"time_unit": "OT", "rate": "15.00"}]', "locations": '[{"latitude": 1.35160, "longitude": 103.87119,	nil	HTTP 200 (Ok)	Pass

			"query": "Nex", "notes": "Test location"}]', "uploaded_by": self.user1.id, }			

2	PUT,
	
	Trying to update another user's listing
		Authorization : Bearer Token	{
			"id": self.listing1.id,
			"title": "Updated Drill",
			"description": "Updated description",
			"category": Category.ELECTRONICS,
			"listing_type": ListingType.RENTAL,
			"rates": '[{"time_unit": "OT", "rate": "15.00"}]',
			"locations": '[{"latitude": 1.35160, "longitude": 103.87119, "query": "Nex", "notes": "Test location"}]',
			"uploaded_by": self.user1.id,
			}
				nil	HTTP 403 (Forbidden)	Pass

Test Case Status		Success				

6.2.3.2.4 Delete Listing

Test Case ID	**T-6.2.3.2.4**		
Endpoint	DELETE `/listing`		
Endpoint Description	The DELETE endpoint will allow the user who created the listing to remove the listing.		
Test Execution			

#	HTTP Method, Description	Header	Body	Params	Expected Status	Pass/Fail
1	DELETE,	Authorization : Bearer Token	nil	id=1	HTTP 200 (Ok)	Pass

	The user deletes their own listing					
2	DELETE, Trying to delete another user's listing	Authorization : Bearer Token	nil	id=2	You don't have permission to delete this listing HTTP 403 (Forbidden)	Pass
2	DELETE, Trying to delete a non-existent listing	Authorization : Bearer Token	nil	id=999	Listing not found HTTP 404 (Not Found)	Pass
Test Case Status		Success				

6.2.3.3 Offer

We will test all functionality related to the offer here.

6.2.3.3.1 Get Offers Details

Test Case ID	**T-6.2.3.3.1**		
Endpoint	GET `/offers`		
Endpoint Description	The GET endpoint will return offer details		
Test Execution			

#	HTTP Method, Description	Header	Body	Params	Expected Status	Pass/Fail
1	GET, Retrieve all offers related to the listing	Authorization : Bearer Token	nil	?listing_id=1	Returns an array of Offers related to the Listing id provided HTTP 200 (Ok)	Pass
2	GET, Retrieve all offers by received	Authorization : Bearer Token	nil	?type=received	Returns an array of offers that the owner of the JWT token has received HTTP 200 (Ok)	Pass
3	GET, Retrieve all offers by made	Authorization : Bearer Token	nil	?type=made	Returns an array of offers that the owner of the JWT token has made HTTP 200 (Ok)	Pass
Test Case Status	Success					

6.2.3.3.2 Create Offer

Test Case ID	**T-6.2.3.3.2**		

Endpoint	POST `/offers`
Endpoint Description	The POST endpoint will allow an authenticated user to create an offer for a listing.

Test Execution

#	HTTP Method, Description	Header	Body	Params	Expected Status	Pass/Fail

1	POST,
	
	Create a new offer
		Authorization : Bearer Token	{
			"listing_id": self.listing.id,
			"price": 15.00,
			"scheduled_start": (timezone.now() + timedelta(days=2)).isoformat(),
			"scheduled_end": (timezone.now() + timedelta(days=2, hours=2)).isoformat(),
			"time_unit": TimeUnit.HOURLY,
			"time_delta": 2,
			}
				nil	HTTP 201 (Created)	Pass

2	POST,
	
	Create a new offer, but the scheduled time has a collision with an existing offer for the same listing
		Authorization : Bearer Token	{
			"listing_id": self.listing.id,
			"price": 20.00,
			"scheduled_start": self.offer1.scheduled_start.isoformat(),
			"scheduled_end": self.offer1.scheduled_end.isoformat(),
			"time_unit": TimeUnit.HOURLY,
			"time_delta": 2,
			}
				nil	HTTP 400 (Bad Request)	Pass

3	POST, Create a new offer, but the	Authorization : Bearer Token	{ "listing_id": self.listing.id, "price": 20.00,	nil	HTTP 400 (Bad Request)	Pass

	scheduled end time is before the start time		"scheduled_start": timezone.now().isoformat(), "scheduled_end": (timezone.now() - timedelta(hours=1)).isoformat(), "time_unit": TimeUnit.HOURLY, "time_delta": 1, }			
4	POST, Create a new offer, but the listing does not exist	Authorization : Bearer Token	{ "listing_id":999, "price": 20.00, "scheduled_start": timezone.now().isoformat(), "scheduled_end": (timezone.now() - timedelta(hours=1)).isoformat(), "time_unit": TimeUnit.HOURLY, "time_delta": 1, }	nil	HTTP 400 (Bad Request)	Pass
	Test Case Status	Success				

6.2.3.3.3 Update Offer

Test Case ID	**T-6.2.3.3.3**		
Endpoint	PUT `/offers`		
Endpoint Description	The PUT endpoint will allow the user to update their offers received. Practically this refers to accepting and rejecting offers.		
Test Execution			

#	HTTP Method, Description	Header	Body	Params	Expected Status	Pass/Fail
1	PUT, Accept offer	Authorization : Bearer Token	{"offer_id": self.offer1.id, "action": "accept"}	nil	HTTP 200 (Ok)	Pass

2	PUT, Reject offer	Authorization : Bearer Token	{"offer_id": self.offer2.id, "action": "reject"}	nil	HTTP 200 (Ok)	Pass
3	PUT, Unauthorized user trying to accept offer	nil	{"offer_id": self.offer2.id, "action": "accept"}	nil	HTTP 404 (Not FOUND)	Pass
Test Case Status		Success				

6.2.3.4 Transaction

We will test all functionality related to the transactions here.

6.2.3.4.1 Get Transaction Details

Test Case ID	**T-6.2.3.4.1**		
Endpoint	GET `/transactions`		
Endpoint Description	The GET endpoint allows the user to see their past transactions.		
Test Execution			

#	HTTP Method, Description	Header	Body	Params	Expected Status	Pass/Fail
1	GET, Get all transactions related to the user	Authorization : Bearer Token	nil	nil	Returns an array of transactions related to the owner of the JWT token HTTP 200 (Ok)	Pass
2	GET, Unauthorized user	nil	nil	nil	HTTP 401 (Unauthorized)	Pass
Test Case Status	Success					

6.2.3.4.2 Create Transaction

Test Case ID	**T-6.2.3.4.2**		
Endpoint	POST `/transactions`		
Endpoint Description	The POST transaction request is used when the user finishes the payment flow. Then this endpoint will be used to log a record of the transaction.		
Test Execution			

#	HTTP Method, Description	Header	Body	Params	Expected Status	Pass/Fail

1	POST,
	
	Create a new transaction
		Authorization : Bearer Token	{
			"offer_id": new_offer.id,
			"amount": 45.00, # 3 hours at $15/hour
			"status": Transaction.COMPLETED,
			"payment_id": "TEST_PAYMENT_2",
			}
				nil	HTTP 201 (Created)	Pass

2	POST,
	
	Create a new transaction, but with missing fields
		Authorization : Bearer Token	{
			"offer_id": new_offer.id,
			# missing amount
			"status": Transaction.COMPLETED,
			}
				nil	HTTP 400 (Bad Request)	Pass

3	POST,
	
	Create a new transaction with pending status
		Authorization : Bearer Token	{
			"offer_id": new_offer.id,
			"amount": 30.00,
			"status": Transaction.PENDING,
			"payment_id": "PENDING_PAYMENT",
			}
				nil	HTTP 201 (Created)	Pass

4	POST,
	
	Create a new transaction with failed status
		Authorization : Bearer Token	{
			"offer_id": new_offer.id,
			"amount": 30.00,
			"status": Transaction.FAILED,
			"payment_id": "FAILED_PAYMENT",
			}
				nil	HTTP 201 (Created)	Pass

5	POST, Create a new transaction with invalid status	Authorization : Bearer Token	{ "offer_id": new_offer.id, "amount": 30.00, "status":"X",	nil	HTTP 400 (Bad Request)	Pass

			"payment_id": "INVALID_STATUS_PAYMENT", }			
6	POST, Unauthorized user creating a transaction	nil	{ "offer_id": self.offer.id, "amount": 20.00, "status": Transaction.COMPLETED, "payment_id": "TEST_PAYMENT", }	nil	HTTP 401 (Unauthorized)	Pass
Test Case Status		Success				

6.2.3.5 Review

6.2.3.5.1 Get Review Details

Test Case ID	**T-6.2.3.5.1**		
Endpoint	GET `/reviews`		
Endpoint Description	The GET endpoint gets all reviews for the given user		
Test Execution			

#	HTTP Method, Description	Header	Body	Params	Expected Status	Pass/Fail
1	GET, Gets all reviews for the user	Authorization : Bearer Token	nil	?user_id=1	Returns an array of reviews for user1 HTTP 200 (Ok)	Pass
2	GET, Gets all reviews for a	Authorization : Bearer Token	nil	?user_id=3	Empty array HTTP 200 (Ok)	Pass

	user with no reviews					
3	GET, Get reviews for user of the token	Authorization : Bearer Token	nil	nil	Returns an array of reviews for the owner of the JWT token HTTP 200 (Ok)	Pass
4	GET, Get reviews of a specific user without a token. Users can view reviews of other users without a token	nil	nil	?user_id=1	Returns an array of reviews for user1 HTTP 200 (Ok)	Pass
5	GET, Get reviews of a user without a specified parameter and without a token	nil	nil	nil	HTTP 401 (Unauthorized)	Pass
6	GET, Get the reviews of the user and ensure that the average rating is calculated correctly	nil	nil	?user_id=1	Asserts that the average rating is the same as the one calculated. HTTP 200 (Ok)	Pass
Test Case Status		Success				

6.2.3.5.2 Create Review

Test Case ID	**T-6.2.3.5.2**		

Endpoint	POST `/reviews`
Endpoint Description	The POST endpoint allows a user to create a review for another user

Test Execution

#	HTTP Method, Description	Header	Body	Params	Expected Status	Pass/Fail

1	POST,
	
	Create a review for another user
		Authorization : Bearer Token	{
			"user_id": self.user3.id, # Reviewing user3
			"rating": 4,
			"description": "Good communication and reliable",
			}
				nil	HTTP 201 (Created)	Pass

2	POST,
	
	Create a review for yourself user
		Authorization : Bearer Token	{
			"user_id": self.user2.id, # Reviewing user3
			"rating": 4,
			"description": "Good communication and reliable",
			}
				nil	Cannot review yourself
					
					HTTP 400 (Bad Request)
						Pass

3	POST,
	
	Create a review with invalid ratings, as ratings only go from 0.5 to 5
		Authorization : Bearer Token	{
			"user_id": self.user3.id, # Reviewing user3
			"rating": 6,
			"description": "Good communication and reliable",
			}
				nil	HTTP 400 (Bad Request)	Pass

3	POST,
	
	Create a review with missing fields
		Authorization : Bearer Token	{
			"user_id": self.user3.id,
			# missing rating
			"description": "Incomplete review",
			}
				nil	HTTP 400 (Bad Request)	Pass

Test Case Status	Success

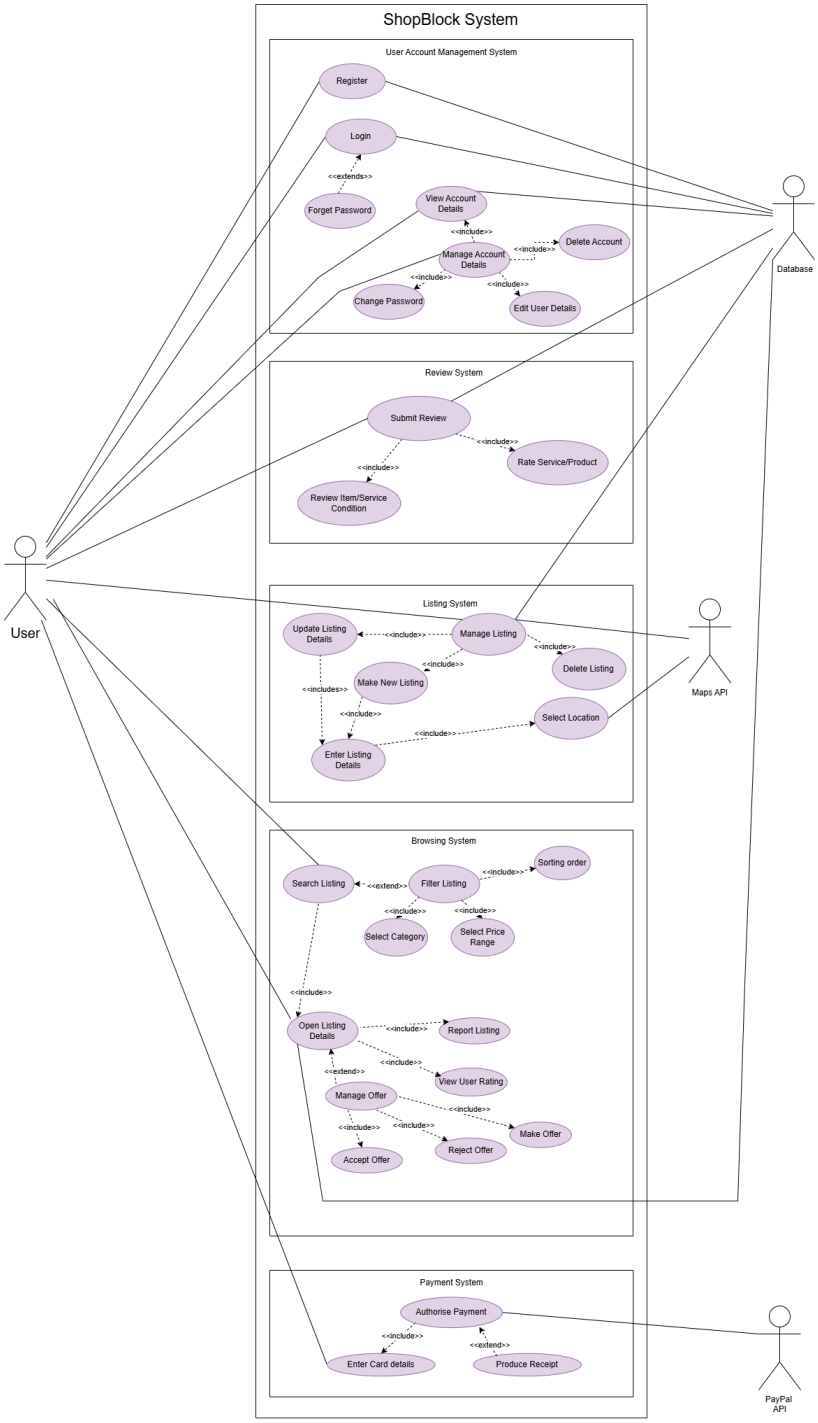
Appendix A: Glossary

Term	Definition
User	A person who has a registered account and is using the services provided by the application.
Account	Profile of registered user of the application containing their details such as their username, phone number, rating etc…
Job	A service that is rented out by a Renter (see below)
Item	An object that is rented out by a Renter (see below)
Renter	A user on the platform renting our their goods/services
Rentee	A user on the platform looking to rent goods/ services
Listing	An entry within the application posted by a renter. Each listing contains information about the good/service including its description, rental price, availability, location, and any associated media such as images or videos.
Offer	A proposal submitted by a rentee to the renter within the application. The offer includes a rental price and rental period that the rentee wishes to propose.
Rating	A numerical score provided by a user (either a rentee or renter) to assess the quality of their experience during a rental transaction. Ratings are given on a scale of 1 to 5 stars with 1 being the lowest rating and and 5 being the highest..
Review	A written evaluation provided by renter/rentee after a rental transaction has been completed. The review allows renters to

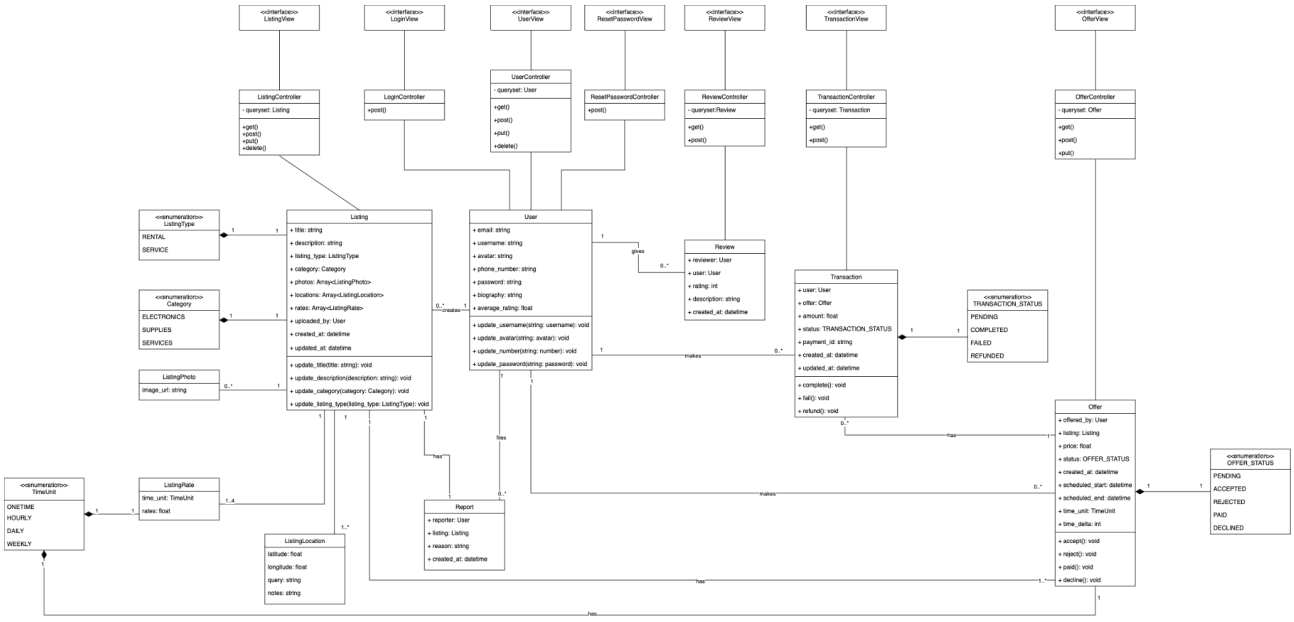
	share experiences and feedback about the rental process, including the quality of the good or service, communication, and overall satisfaction with the interaction.

Appendix B: Analysis Models

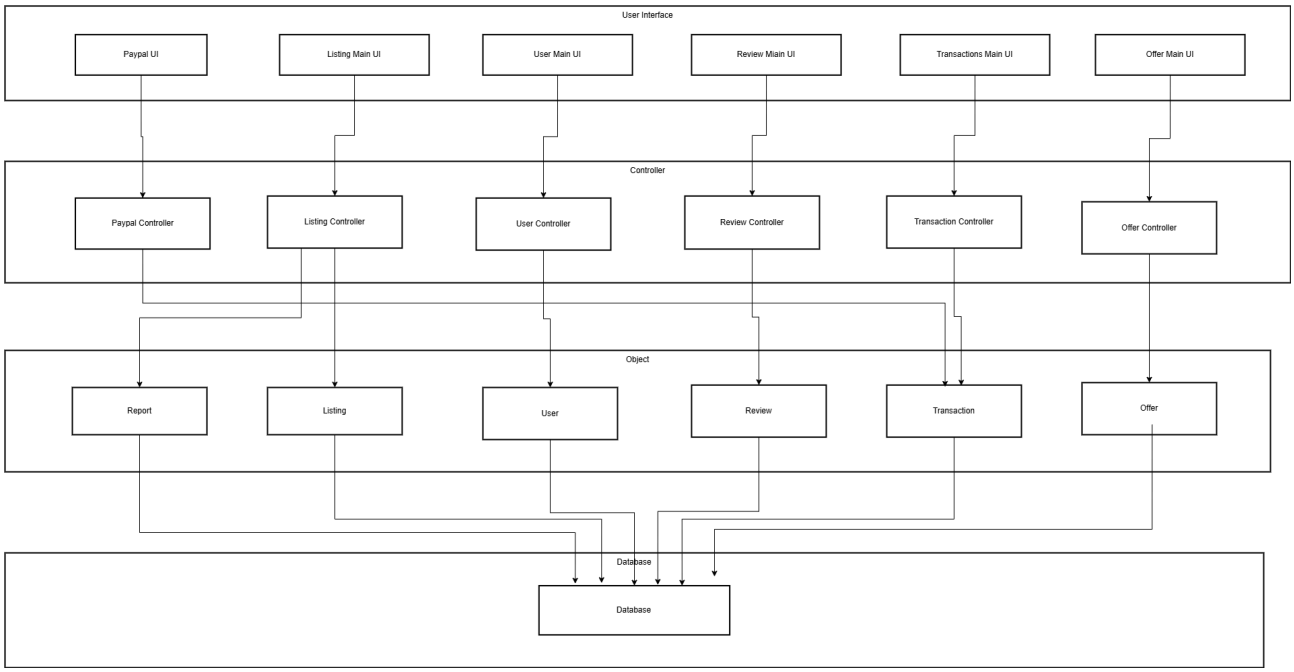
Use Case Diagram



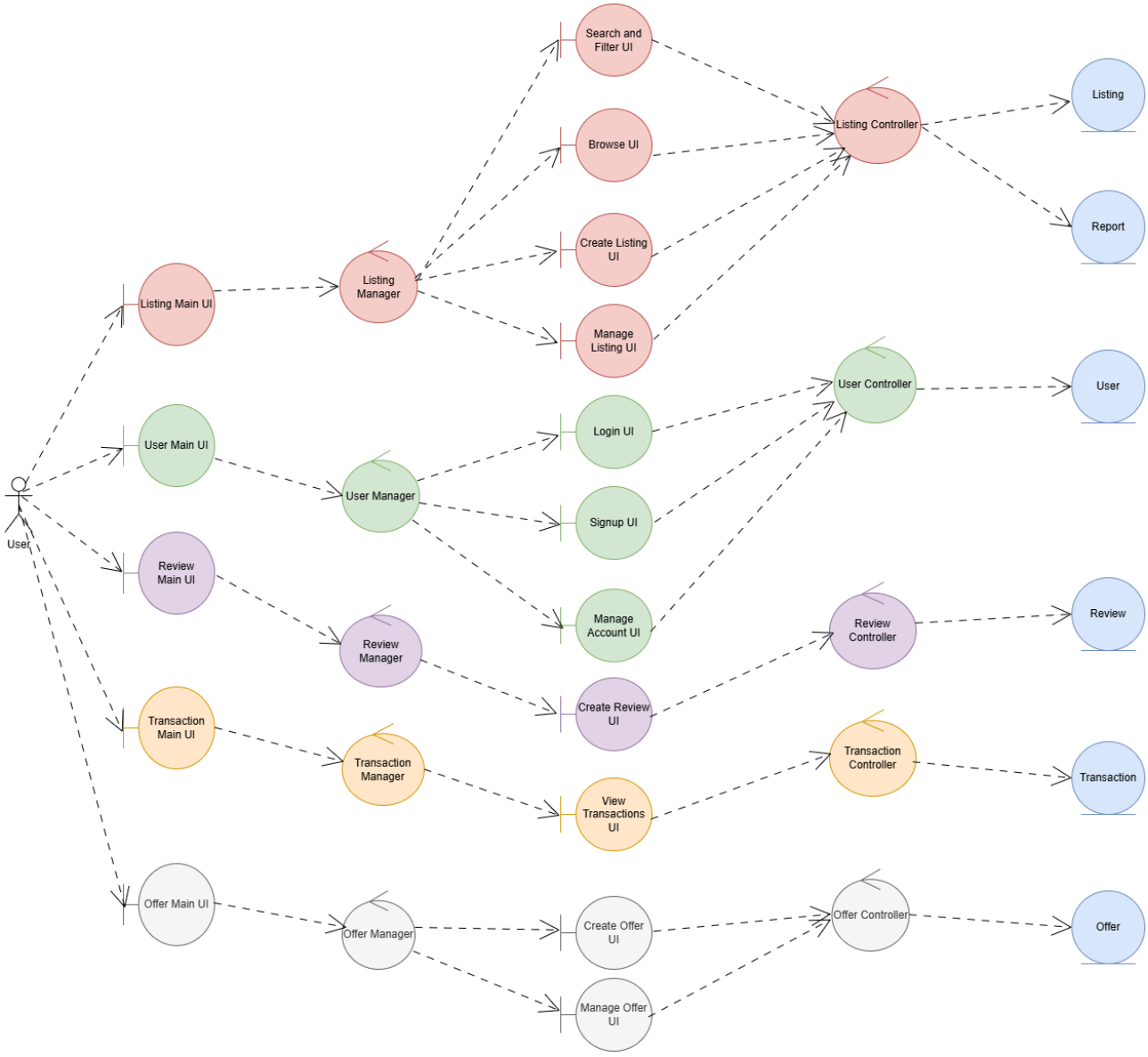
Class Diagram



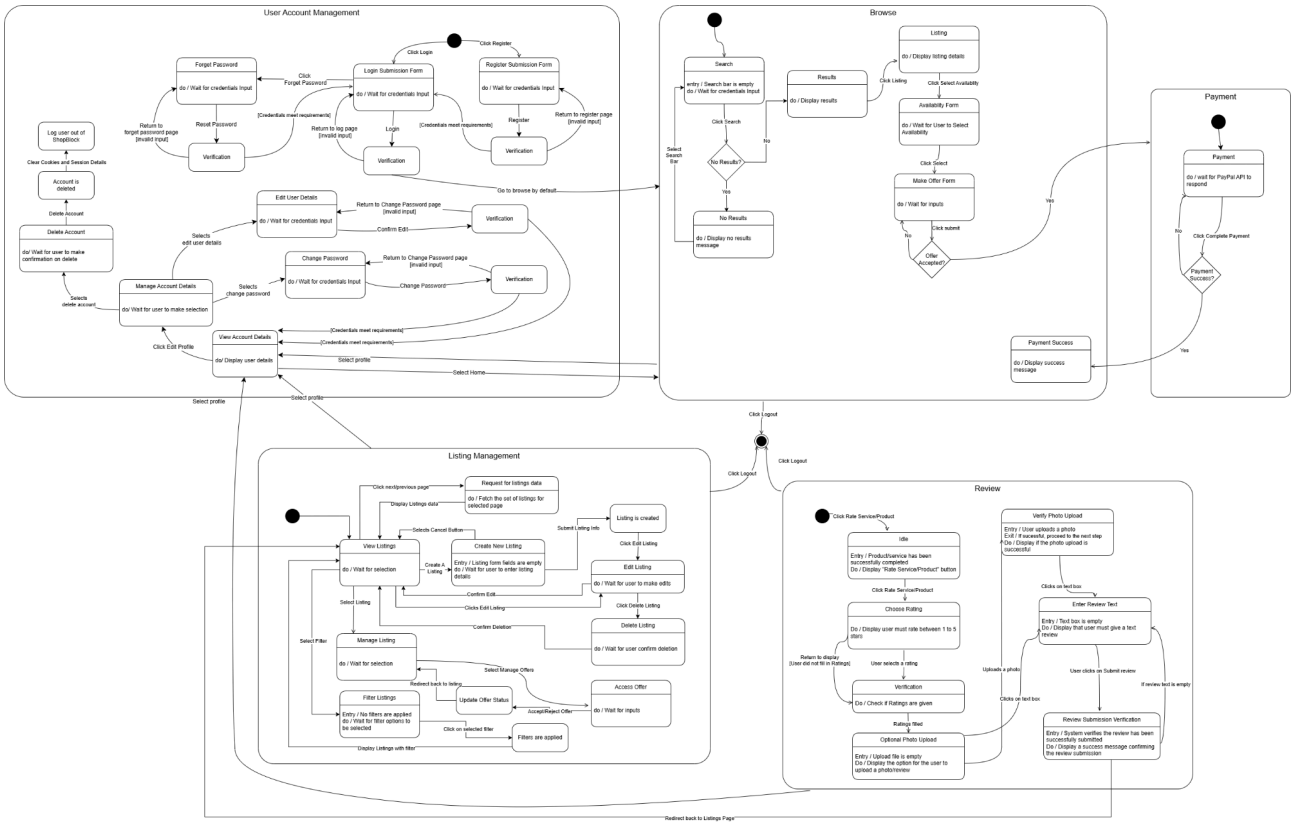
Architecture Diagram



Class Boundary Diagram

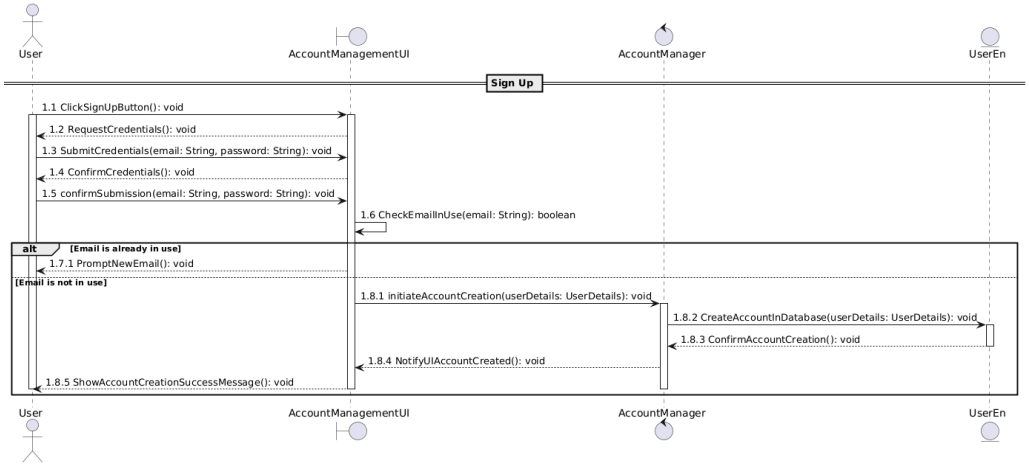


Dialog Map Diagram

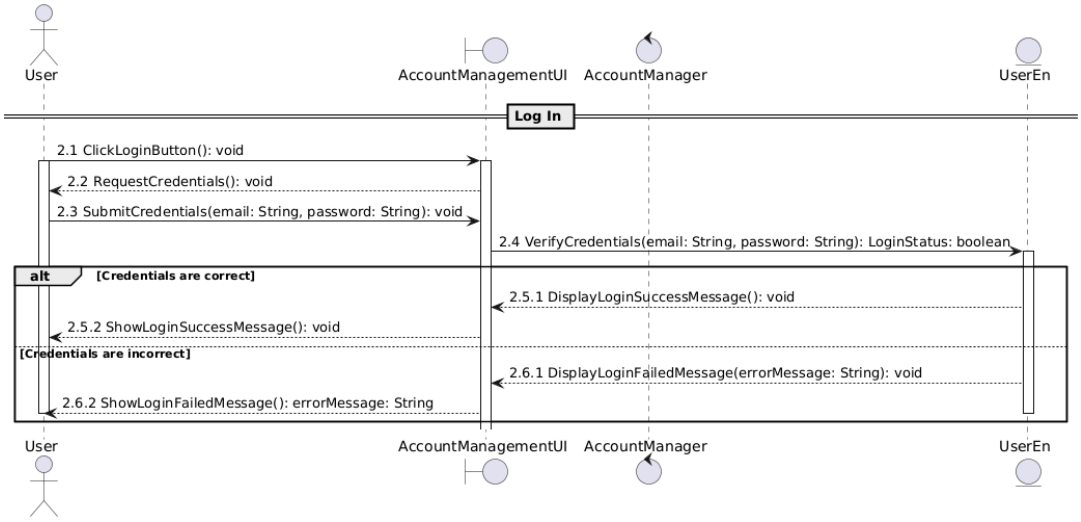


Sequence Diagrams

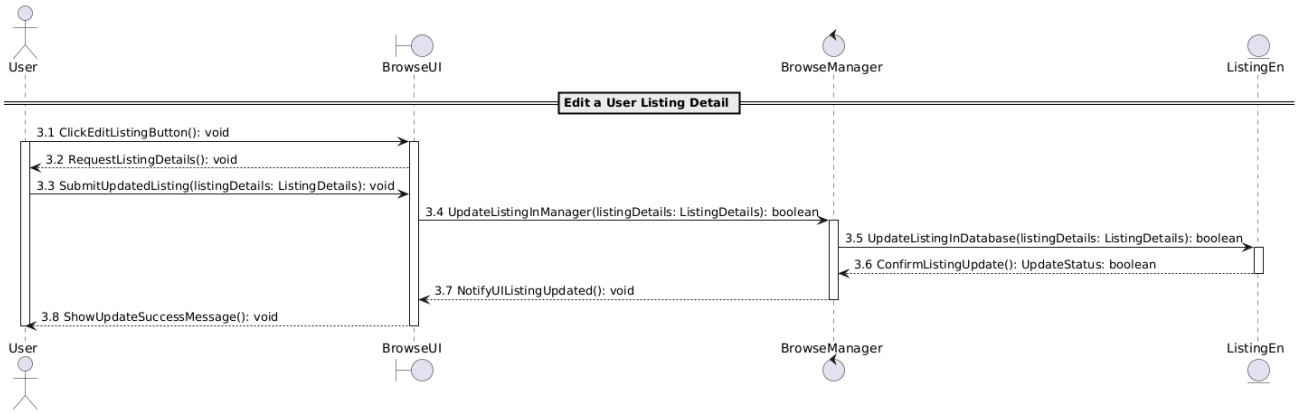
1. **Sign Up**



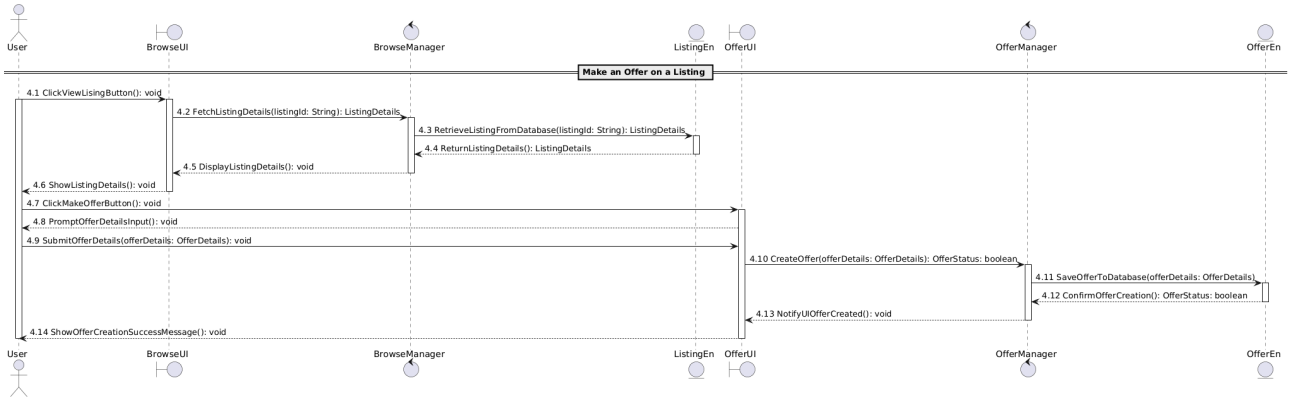
2. Login in



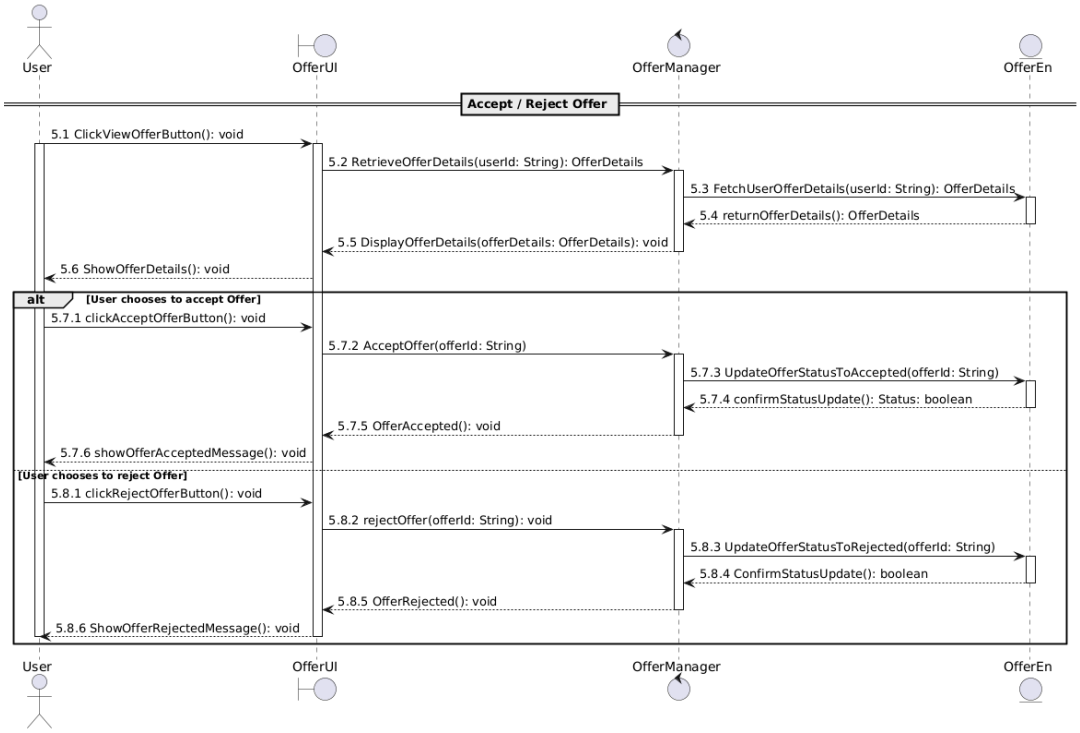
3. Edit User Listing



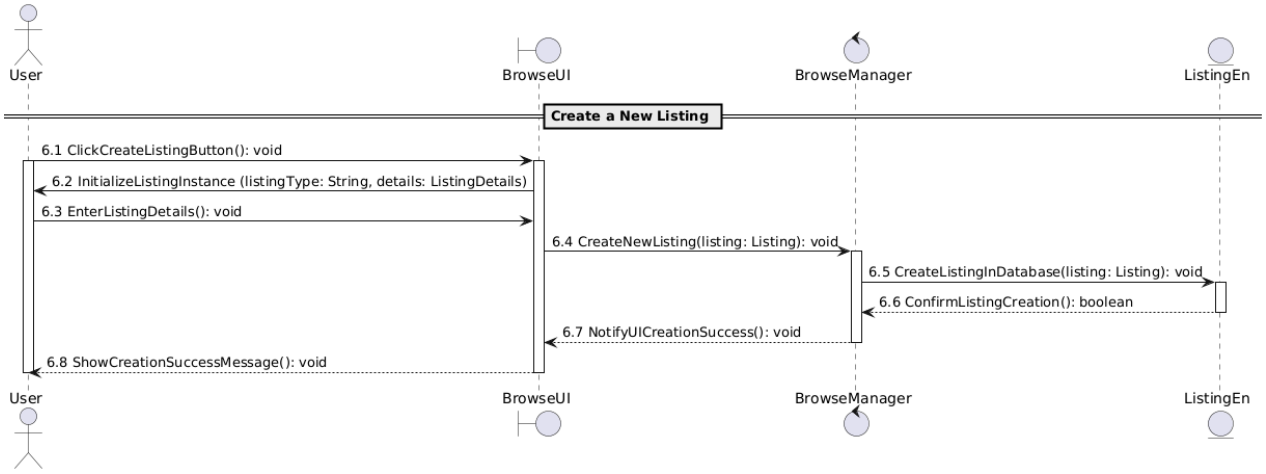
4. Make an offer



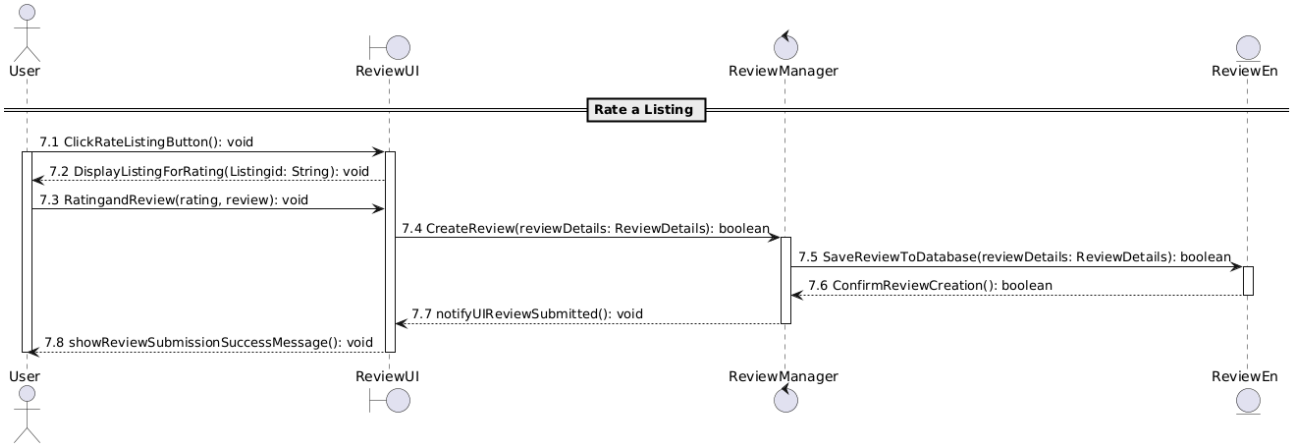
5. Accept or Reject offer



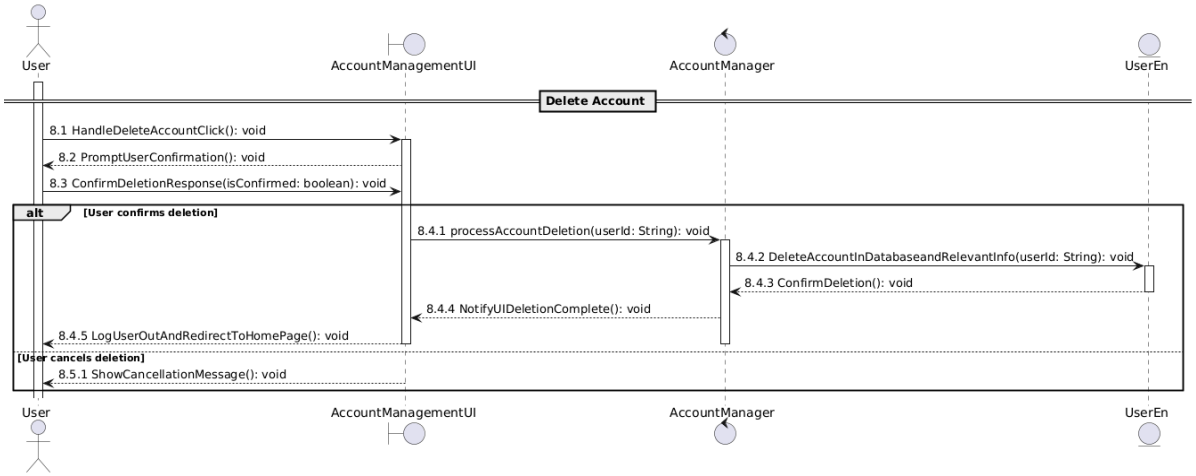
6. Create new listing



7. Rate a listing



8. Delete account



AppendixC C: API Documentation