SHOPBLOCK

Affordable, Convenient, and Community-Driven Shopping – Made for You!

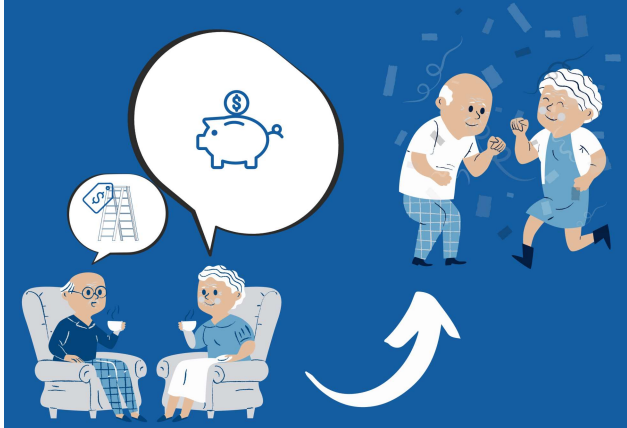
Group 45	Matriculation No.
Kua Li Min	U2322513H
Hudzaifah Bin Muhammad Taufiq	U2320600F
Cheng Hui Wen	U2322123G
Ng Hoe Ping	U2321991F
Seow Jia Xian Jackson	U2322995F
Chua Ze Ming	U2321797B
Zhang Yuxuan	U2321475L

1 Introduction

Problem statement 1.1

- **Low income families struggle with one-time purchases**
- **Purchasing rarely used items (e.g., tent) would be a pure wastage of money resources**

Case Studies 1.2

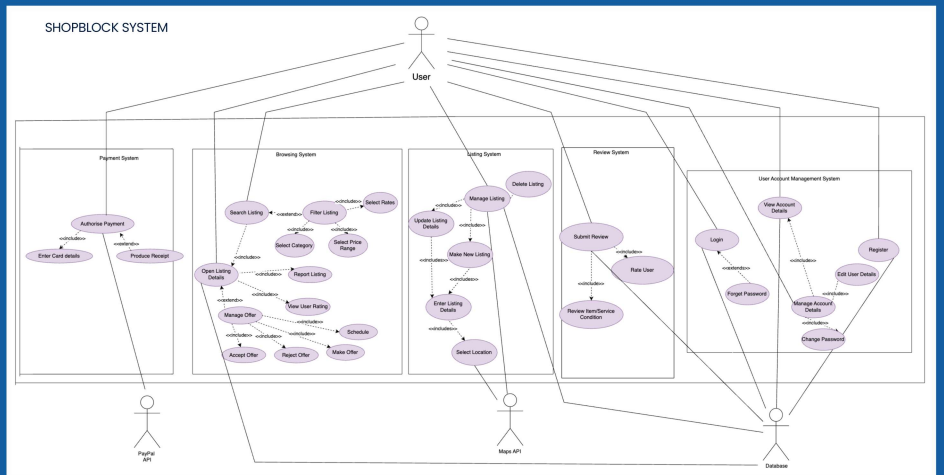


What if we could **borrow** instead of buy

How We Are Tackling This 1.3

- **Browsing through Rental Items**
- **Searching, Sorting, and Filtering**
- **Offers**
- **Reviews**

Use Case Diagram 1.4



Quick Demo! 1.7

Demonstration of the various usage scenarios by the different end-users.

2 Software Engineering Practices and Design Patterns

2.1 Tech Stack

2.2 Documentation & Good Coding Practices

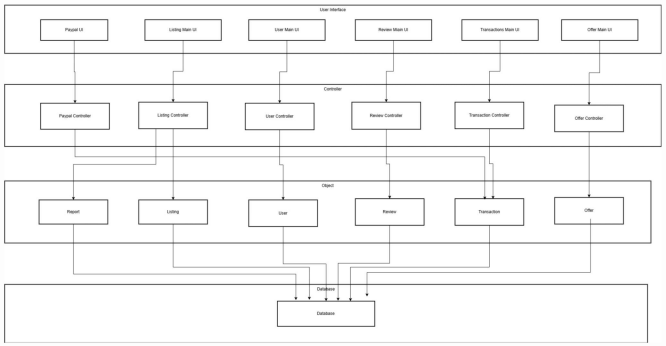
Tech Stack 3.1

Front End

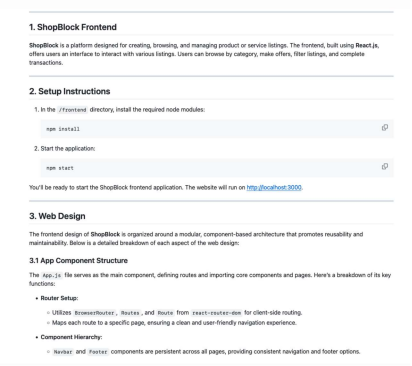
- JavaScript
- Material UI
- React
- CSS

Back End

- Python
- Django
- Open API



Good Coding Practices - Documentation 3.1



1. ShopBlock Frontend

ShopBlock is a platform designed for creating, browsing, and managing product or service listings. The frontend, built using **React.js**, offers users an interface to interact with various listings. Users can browse by category, make offers, filter listings, and complete transactions.

2. Setup Instructions

1. In the `/Frontend` directory, install the required node modules:

```
npm install
```

2. Start the application:

```
npm start
```

You'll be ready to start the ShopBlock frontend application. The website will run on http://localhost:3000.

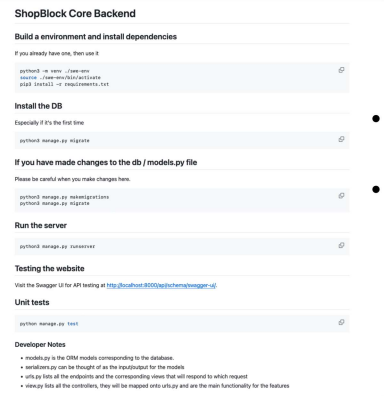
3. Web Design

The frontend design of **ShopBlock** is organized around a modular, component-based architecture that promotes reusability and maintainability. Below is a detailed breakdown of each aspect of the web design:

3.1 App Component Structure

The `App.js` file serves as the main component, defining routes and importing core components and pages. Here's a breakdown of its key functions:

- Router Setup:
  - Utilizes `BrowserRouter`, `Routes`, and `Route` from `react-router-dom` for client-side routing.
  - Maps each route to a specific page, ensuring a clean and user-friendly navigation experience.
- Component Hierarchy:
  - `Navbar` and `Footer` components are persistent across all pages, providing consistent navigation and footer options.



ShopBlock Core Backend

Build a environment and install dependencies

If you already have one, then use it

```
python3 -m venv ./swe-env
source ./swe-env/bin/activate
pip3 install -r requirements.txt
```

Install the DB

Especially if it's the first time

```
python3 manage.py migrate
```

If you have made changes to the db / models.py file

Please be careful when you make changes here.

```
python3 manage.py makemigrations
python3 manage.py migrate
```

Run the server

```
python3 manage.py runserver
```

Testing the website

Visit the Swagger UI for API testing at http://localhost:8000/api/schema/swagger-ui/.

Unit tests

```
python manage.py test
```

Developer Notes

- models.py is the ORM models corresponding to the database.
- serializers.py can be thought of as the input/output for the models
- urls.py lists all the endpoints and the corresponding views that will respond to which request
- view.py lists all the controllers, they will be mapped onto urls.py and are the main functionality for the features

- Setup guides for any incoming developer
- Clear instructions on how to set up the project

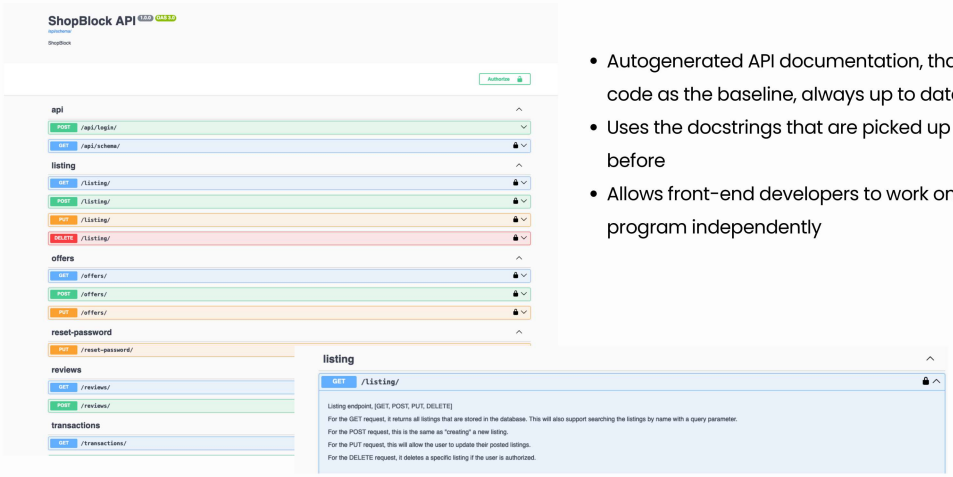
Good Coding Practices - Comments 3.2

```python
class ListingController(GenericAPIView):
    """
    Listing endpoint, [GET, POST, PUT, DELETE]

    For the GET request, it returns all listings that are stored in the database.
        This will also support searching the listings by name with a query parameter.

    For the POST request, this is the same as "creating" a new listing.

    For the PUT request, this will allow the user to update their posted listings.

    For the DELETE request, it deletes a specific listing if the user is authorized.
    """
```

```python
class TransactionController(GenericAPIView):
    """
    Transactions endpoint, [GET, POST]

    For the GET request, it returns the all transactions for the user

    For the POST request, once a user completes their payment, it will post to this endpoint to store for transaction history
    """
```

- Consistent and clear comments on every major component
- Follows the PEP convention on writing docstrings, which then gets picked up

Good Coding Practices - API Docs 3.3



ShopBlock API 1.0.0 OAS 3.0

ShopBlock

Authorize

api
- POST /api/login/
- GET /api/schema/

listing
- GET /listing/
- POST /listing/
- PUT /listing/
- DELETE /listing/

offers
- GET /offers/
- POST /offers/
- PUT /offers/

reset-password
- PUT /reset-password/

reviews
- GET /reviews/
- POST /reviews/

transactions
- GET /transactions/

listing

GET /listing/

Listing endpoint, [GET, POST, PUT, DELETE]

For the GET request, it returns all listings that are stored in the database. This will also support searching the listings by name with a query parameter.

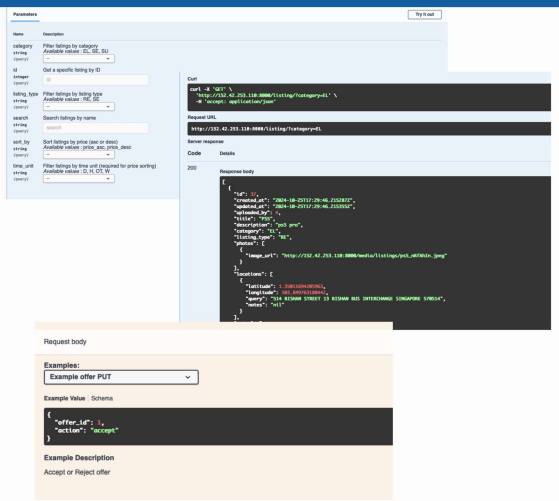
For the POST request, this is the same as "creating" a new listing.

For the PUT request, this will allow the user to update their posted listings.

For the DELETE request, it deletes a specific listing if the user is authorized.

- Autogenerated API documentation, that uses the code as the baseline, always up to date
- Uses the docstrings that are picked up from before
- Allows front-end developers to work on the program independently

Good Coding Practices - API Docs 3.4



- Friendly interface to play around with query params, and different request bodies
- A front-end to inspect the data that is being passed back and forth
- Examples generated from code, always up to date that can be used as reference for integrating the API

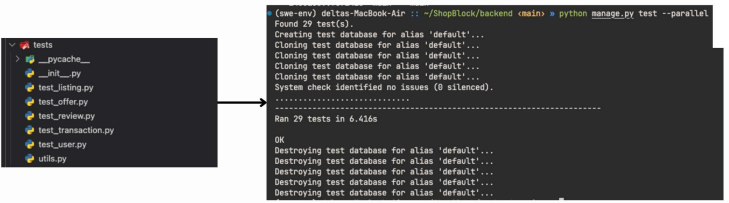
Good Coding Practices - Dev Tooling 3.5

- Consistent formatting with Python's Black
- Ensures all code is familiar

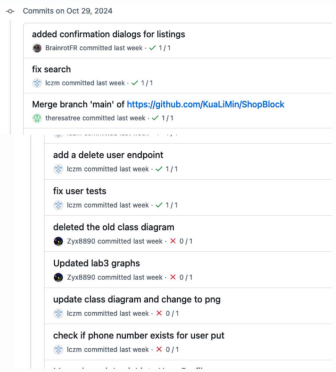
```
def validate(self, data):
    email = data.get("email")
    phone_number = data.get("phone_number")

    # case 1: both email and phone number are not found, return error message "Both email and phone number not found"
    if ( not User.objects.filter(email=email).exists() and not User.objects.filter(phone_number=phone_number).exists()):
        raise serializers.ValidationError("Both email and phone number not found")

    # case 2: email is found but phone number is not found, return error message "Phone number not found"
    if ( User.objects.filter(email=email).exists() and not User.objects.filter(phone_number=phone_number).exists()):
        raise serializers.ValidationError("Phone number not found")

    # case 3: email is not found but phone number is found, return error message "Email not found"
    if ( not User.objects.filter(email=email).exists() and User.objects.filter(phone_number=phone_number).exists()):
        raise serializers.ValidationError("Email not found")
    return data
```

```
def validate(self, data):
    email = data.get("email")
    phone_number = data.get("phone_number")

    # case 1: both email and phone number are not found, return error message "Both email a
    if (
        not User.objects.filter(email=email).exists()
        and not User.objects.filter(phone_number=phone_number).exists()
    ):
        raise serializers.ValidationError("Both email and phone number not found")

    # case 2: email is found but phone number is not found, return error message "Phone nu
    if (
        User.objects.filter(email=email).exists()
        and not User.objects.filter(phone_number=phone_number).exists()
    ):
        raise serializers.ValidationError("Phone number not found")

    # case 3: email is not found but phone number is found, return error message "Email no
    if (
        not User.objects.filter(email=email).exists()
        and User.objects.filter(phone_number=phone_number).exists()
    ):
        raise serializers.ValidationError("Email not found")
    return data
```

Good Coding Practices - Tests & CI 3.6

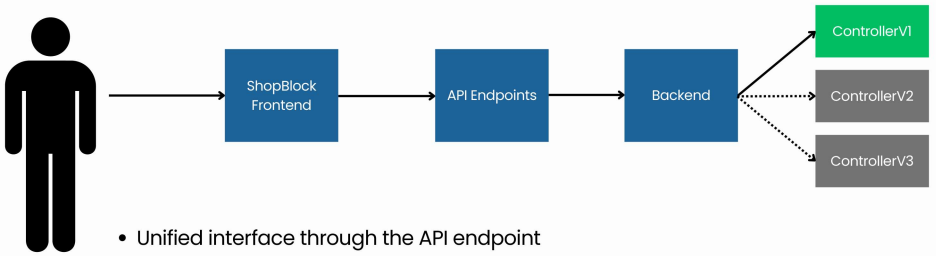



```
(swe-env) deltas-MacBook-Air :: ~/ShopBlock/backend <main> » python manage.py test --parallel
Found 29 test(s).
Creating test database for alias 'default'...
Cloning test database for alias 'default'...
Cloning test database for alias 'default'...
Cloning test database for alias 'default'...
Cloning test database for alias 'default'...
System check identified no issues (0 silenced).
.............................
----------------------------------------------------------------------
Ran 29 tests in 6.416s

OK
Destroying test database for alias 'default'...
Destroying test database for alias 'default'...
Destroying test database for alias 'default'...
Destroying test database for alias 'default'...
Destroying test database for alias 'default'...
```



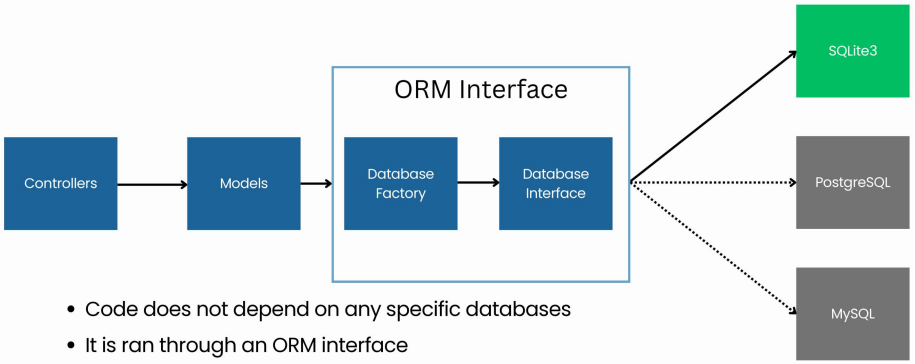
- Tests for all endpoints
- Tests run on a separate database, replicates production setup
- Ran on Github's CI/CD, ensures all endpoints are working
- There are times that it fails, and we'll know immediately

Design Patterns - Facade Pattern 3.7



- Unified interface through the API endpoint
- Frontend is not dependent on the changes the backend makes
- Backend can version and make appropriate changes when necessary, decouples the frontend from backend complexity

Design Patterns - Factory Pattern 3.8



- Code does not depend on any specific databases
- It is ran through an ORM interface
- Can easily swap out and change to a more performant database for specific needs in the future

5 Traceability

Use Case Description 5.1

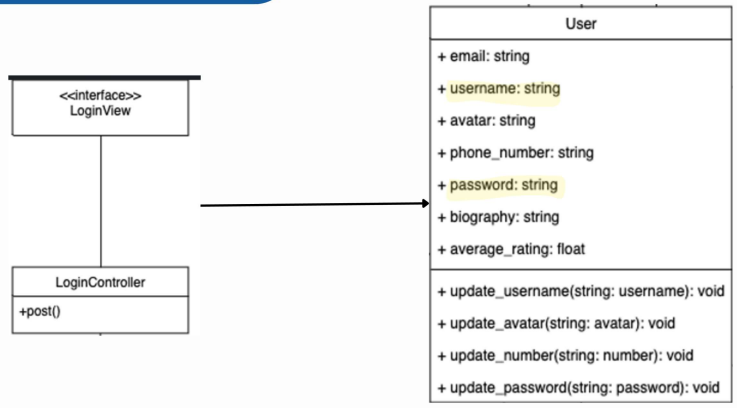
Case 1: Login in

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:	**1/9/2024**

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: username and password.** 2. **System validates the information with existing information in the database.**

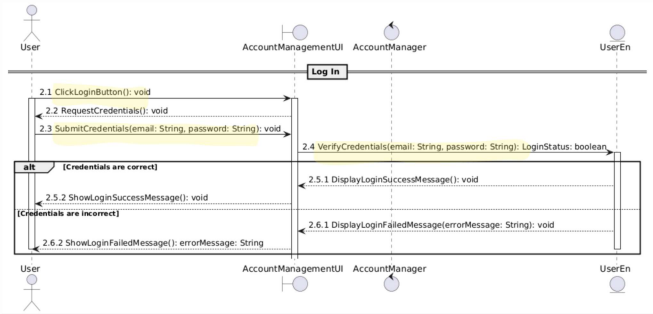
Class Diagram 5.2

Case 1: Login in



Sequence Diagram 5.3

Case 1: Login in



Tests 5.4

Case 1: Login in

1.2.1.2 Login

Test Case ID	1-1.2.1.2	**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	Displays "Your account has been temporarily locked for 5 mins. Please try again later or	Pass

Use Case Description 5.1

Case 2: Create Listing

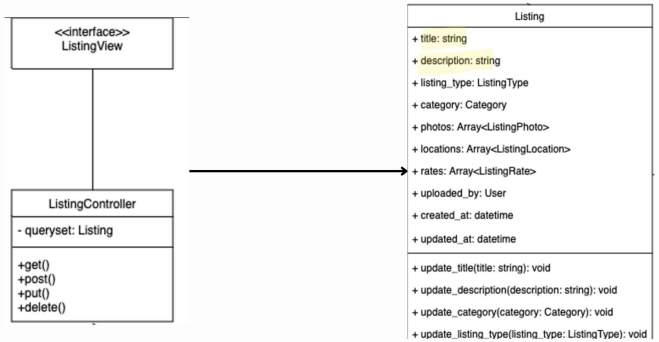
Use Case ID:	LIST_UC_2		
Use Case Name:	Make New Listing		
Created By:	Hui Wen	Last Updated By:	Hui Wen
Date Created:	1/9/24	Date Last Updated:	1/9/24

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 "Must include Title/description/rental price/location " 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 leave and delete 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.

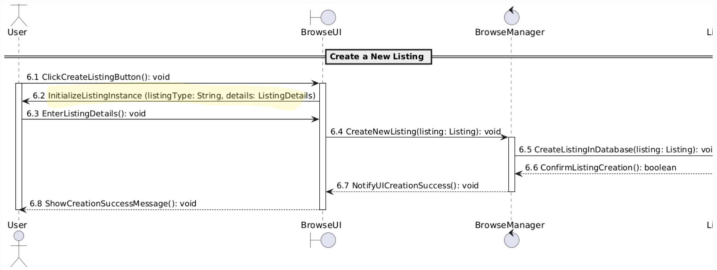
Class Diagram 5.2

Case 2: Create Listing



Sequence Diagram 5.3

Case 2: Create Listing



Tests 5.4

Case 2: Create Listing

Create Listing

Test Case ID	T-1.2.3	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 Description: Test case 3 Photo: test.jpeg Location: Jurong Point Additional Notes: Meet be at the main entrance	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
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