University of Nevada, Reno

Department of Computer Science and Engineering

CS 425 Software Engineering

Project Part 3: Acceptance Criteria & Testing Strategy/Plan

Drift - Team 12

Jordan Rood, Fiorina Chau, John Christian Jackson

Instructors: Dave Feil-Seifer, Devrin Lee, Sara Davis, Vinh Le, Zach Estreito

External Advisor(s) w/ Affiliation:

Brittany N Avila - Psychology Department, University of Nevada - Reno

Araam Zaremehrjardi - Grad Student, University of Nevada - Reno

February 22, 2024

1 - Abstract

Drift is a thrifting mobile application as well as a software service specifically for public users to buy and sell thrifting goods (I.e., an e-commerce app for thrifting specifically). Our project is important because it will pave the way for a more sustainable thrifting experience while making it easier to buy/sell, connect with other thrifters, and discover all types of second hand items. The major features that we intend to design and implement are user login/signup authentication, item posting, item discovery and querying, cart use, checkout, and chat functionality. Our thrifting and e-commerce centralized application will provide a virtual thrift shopping experience for browsing for items, posting items to sell on one's profile, and perusing saved items.

2 - Project Updates and Changes

Drift's progress so far includes the implementation of the login and signup authentication flow with frontend to backend connection, post item, saved folder, profile, and Discover page UI. On the Discover page, we have the search bar querying working along with item selection for further details. For authentication, all those pages UI are done with login, signup, and password encryption and validation with that working. We also have rendering the users orders to the orders page up and running with that frontend to backend connection as well as post items working. Additionally, we had a meeting where we got our database all on the same page locally so that we can mitigate the problems occurring with differing database instances and API development. We have our stored procedures and database structure all finished up with the only changes needed to be any tweaks we may have for the rest of the software development lifecycle. Some tasks currently in the development process are the adding and removal of items from the cart, rendering the items to the discover page from the database, profile and saved items functionality.

Drift changes thus far include the plans on instead of creating our own fully fledged chat feature where two or more Drift users can communicate with each other through messages, we will be utilizing the TalkJS API and library for helping to implement this feature. Therefore, we still plan to implement such a feature, but found it to be unrealistic to implement such a big feature (having a multitude of subfeatures) in the scope of this class alongside all the other features and main functionalities that we have on the table for Drift. This change will hopefully make it easier to navigate the implementation of our chat feature and in result have more time for the development of other use cases and features.

The main reason for this change as said prior is scoping and planning to make a more realistic workload for the time constraint that our team of three has.

3 - User Stories and Acceptance Criteria

User stories are what describe the multitude of use cases and situations that need to be able to occur due to such user requirements that Drift shall encompass. Tasks and plans for development and design of some software are usually built off of the prospective user stories, requirements, and use cases that are expected to be within such software. Moreover, acceptance tests and criteria are those which well describe the expected behavior to be accepted by the system being made both by the stakeholders and developers themselves; in result, these points help to plan and document the expected behavior and output that is needed for development to be complete on such features. Therefore, below lies a list of user stories and the corresponding acceptance criteria for each.

1. As a Drift user, I want to be able to make an account through signing up with my name, email, and a new password and then be authenticated and directed to Drifts's main app directory.

   Acceptance Criteria:
   - As a user, I can enter my first and last name, email address, phone number (optionally), and desired password on the sign up page to register a new account with the Drift app.
   - A user will be directed to Drift's main pages upon signup and will be logged into my account that they have just registered.

2. As a Drift user, I want to be able to log in and log out of the Drift mobile application with the credentials made prior upon first signup.

   Acceptance Criteria:
   - From the main splash screen, a user can continue to the login page and input their username and password that they created previously from signing up.
   - When input correctly, the user will be promptly redirected to the main pages and navigation with the app being custom to their account.

- When a user inputs the wrong username and/or password, the user will observe a popup alert that states “The username and/or password input are incorrect. Please try again!”.

3. As a Drift user, I want to be able to post an item for sale so that it can be rendered to both my Profile postings and the main Discover page.

   Acceptance Criteria:
   - From the Discover page, users can press on the Post page button in the bottom navigation bar to be directed there.
   - The user can add photos of the item they’re posting along with text into the input fields the name, description, quality, and other details. Once done, the user presses the post button which opens a confirmation popup upon posting of the item to the public
   - The item should then be viewable from the Discover page by others and the profile page of the user who posted the item.

4. As a Drift user, I expect to be able to click on a displayed item and its own item page to pop up with its picture(s), details, etc.

   Acceptance Criteria:
   - The user can view the rendered items from the database that are up for sale by other users on the Discover page.
   - The user should be able to then peruse the page and scroll through items until they find one they want more details on. Once found, the user can click on its card and will be redirected to the item’s specific page with a range of details below the main pictures of the item.
   - There should be a button to add the item to cart and a button to navigate back to the Discover page. When pressing on the add item to cart button, the icon will switch to a checkmark to indicate the item has been added to the cart.

5. As a user of the Drift application, I want to be able to view the publicly posted items for sale by other users on the main Discover page.

   Acceptance Criteria:
   - The Drift thift-er should be able to login with a previously registered user account to ensure that they will have access to the main Discover page for viewing.
   - The discover page will be composed of all the items that are up for sale by other users.

- The Discover page should be reachable by both the button on the drawer and the bottom navigation bar.

6. As a Drift user, I expect to be able to add and/or remove items to and from the cart.

   Acceptance Criteria:

   - A user should be able to login to a registered account and therefore have access to the discover page and all other pages where items may be depicted.
   - The user should be able to find a desired item, click on it, and be directed to the item page where they will find an 'Add to cart' button. If clicked, the process will execute and confirm addition by popup saying item has been added.
   - The item should be in the users cart on the cart page and there should be a 'Remove item' button there as well. If clicked on the 'Remove item' button the item will no longer be in the cart. The total price should be updated accordingly.

7. As a Drift user, I want to be able to proceed to checkout and purchase the items that are in my cart.

   Acceptance Criteria:

   - Users should be able to view all the items added to the cart in the Cart page.
   - There should be a 'Proceed to Checkout' button at the bottom of the Cart page which when pressed starts the checkout process. This process starts with the billing and orders form sliding up on top of the checkout page. After submitting the order, an order confirmation page will pop up with a button for navigating back to the cart that should be empty.

8. As a Drift user, I want to be able to make modifications and updates to my own profile page such as changing my profile picture, updating my bio, or adding any other possible information.

   Acceptance Criteria:

   - Users will be able to login to their own account and therefore have a customizable profile which is viewable on the profile page.
   - Users should be able to edit their profile from the profile or settings page. To make edits, a form popup with the current information input should be shown upon clicking on the edit button.

- Once the user makes the desired edits to their profile (bio, profile pic, etc.), they should be able to press on the save button which redirects them to their profile page with the updates taking place and viewable to confirm.

9. As a basic user of the Drift application, I expect to be able to search for items from the main Discover page by keywords and categories.

   Acceptance Criteria:
   - Users are expected to be able to login with valid credentials in the Login page which will properly redirect them to the main application pages. The page it lands on should be the Discover page.
   - A search bar should be at the top in which a user can click on it to start a search. Filtering should start as soon as any characters are input, and this filtering queries based on keywords and categories of items. The items being displayed under a search should only be those that match what is being searched for by the user.

10. As a Drift user, I want to be able to view saved items from the saved items page. These saved items should be those which I have saved for later and assorted into respective folders for future reference.

    Acceptance Criteria:
    - The users should be able to save items in their specific item pages through pressing on the save icon button which will fill in to indicate to the user that the item has been saved.
    - The saved items should then be viewable from the saved items page which can be navigated to by the drawer or bottom navigation bar. Users should be able to organize their saved items into folders as well. These folders can hold from 0 to several items within for future reference for the user.
    - When on the item page and deselecting the saved item, the item should then not be rendered or viewable in the saved items page at all.

4 - Testing Workflow

The testing workflow below will outline both happy path and unhappy path workflows through the Drift application. Happy paths are those that are regular and common paths and functionality uses that would occur and unhappy paths are those that are moreso corner cases and ones which are not usually paths visited. The following depicts those paths and how they will be validated below.

Happy Path Workflows

1. Chat Feature:
    - Log into an account
    - Navigate to chat inbox
    - Send a message to another user account
    - Log out of current account, log into the account of the message recipient
    - Verify message is delivered and displayed correctly

We plan to validate the chat feature as soon as we have finished implementing messaging and integrate it with user accounts. As described above, we will perform manual testing to verify that users can successfully send and receive messages by messaging an account, logging into that account, verifying the message was received, and then repeating the same process with the original account.

2. Post Item For Sale:
    - Log into an account
    - Navigate to ‘Post’ page
    - Fill out the description, brand, price, category, and image details
    - Post the item
    - Verify the item appears in ‘Profile’ page as well as ‘Discover’ page

We plan to validate item posting immediately, as all of the necessary supporting features are in place and the backend is connected. We will perform both manual and automated testing to validate this workflow. As described above, after going through the process of posting an item, we will then navigate to the ‘Discover’ page and ‘Profile’ page to verify the item is displayed there. We will also implement a unit test to verify that the routes for posting items to the backend maintains data consistent with what was input by the user.

3. Profile Management:
    - Log into an account
    - Navigate to user ‘Profile’ page
    - Update profile information such as bio and profile picture
    - Log out of current account, log into account of another user

- Navigate to 'Profile' page of previous account, verify the updated information is displayed

We plan to validate profile management as soon as we have finished implementing the user profile page. As described above, after going through the process of updating user information on the 'Profile' page, we will then log into another user account, and navigate to the 'Profile' page of the original user. Once on that page, we will confirm that the information displayed is consistent with the changes made.

Unhappy Path Workflows

1. Failed Sign Up Attempt
    - Navigate to 'Sign Up' page
    - Attempt to sign up with username and/or email that is already associated with another user account
    - Verify that users are informed about the invalid credentials and prompted to modify their input accordingly

We plan to test this workflow immediately, as the routes for sign up are in place and the other supporting features have been implemented. As described above, we will test this manually by attempting to sign up with invalid details. In the case that the appropriate safeguards are not implemented, we will add those to avoid a scenario in which users with different accounts can have the same username or email.

2. Posting Invalid Items
    - Navigate to 'Post' page
    - Attempt to post an item with missing or invalid information (e.g. blank description, paste a negative number into the price field)
    - Verify that users are prevented from posting invalid items and are prompted to modify their input accordingly

We plan to test this workflow immediately, as the interface and necessary features for posting items have been implemented. As described above, we will utilize manual testing to validate this workflow. The keyboard that is currently opened upon clicking the 'Price' field does not have a '-' symbol on it, but it would be possible to paste a number or even a string of letters into the field. We will also attempt to post an item that is missing necessary information. In the case that the appropriate safeguards are not implemented, we will add those to avoid a scenario in which an item can have a negative price or undefined descriptor field.

3. Purchase Failure
    - From ‘Discover’ page, save an item to ‘Saved’ folder
    - Log into other user account
    - From ‘Discover’ page, add the same item to cart, and proceed to checkout
    - Log back into original user account
    - From ‘Saved’ page, if item still present, add to cart, and attempt to checkout
    - Verify that user is prevented from purchasing previously sold product and descriptive error message is displayed

We plan to test this workflow as soon as the ‘Saved’ items functionality has finished being implemented. As described above, we will utilize manual testing to validate this workflow. Upon logging into a second account and finding and purchasing the item that was saved in the first user account, we will then log back into the first user account, and if the item is still displayed we will then add it to cart and attempt to check out. In the case the appropriate safeguards have not been implemented, we will add those to avoid a scenario in which a user pays for a product that is not available.

5 - Testing Strategy

For our testing, we plan to conduct the following tests with the ideal timeline and to the assigned people:

- Unit Tests (Week of March 4th) -
    - All of us will be conducting unit tests for the individual systems we are responsible for.

- Acceptance Tests (Week of March 18th) -
    - All of us will be conducting acceptance tests. We will ensure that the person developing the story’s functionality is not testing the story for the acceptance test. Jordan will be conducting acceptance testing for the parts Fiorina is developing (e.g. profile, saved, discovered features). Fiorina will conduct acceptance testing for the parts Christian is developing (e.g. cart, post features). Christian will conduct acceptance testing for the parts Jordan is responsible for (e.g. user authentication, chat features). We will also be working with our advisor(s) on the acceptance testing. Some acceptance criteria and workflow items include:
    - Users should be able to create an account or login.
    - Users should be able to search for different items

        - Users should be able to buy items or save items to folders
        - Users should be able to message each other
        - Users should be able to complete a form to sell their own clothes
        - Users should be able to manage their profile

- <u>User Tests (Week of April 1st)</u> -
    - All of us will be gathering feedback from our peers and friends on their experience with the app for the user tests.

Any defects found can be reported to our team chat. The person originally assigned to develop the feature will be responsible for handling the defect. However, if one person has too much workload, we can distribute the defect to team members with less workload for the project to stay on track. Once there are no defects found and all tests are passed, the project will be completed.

Test Plan:

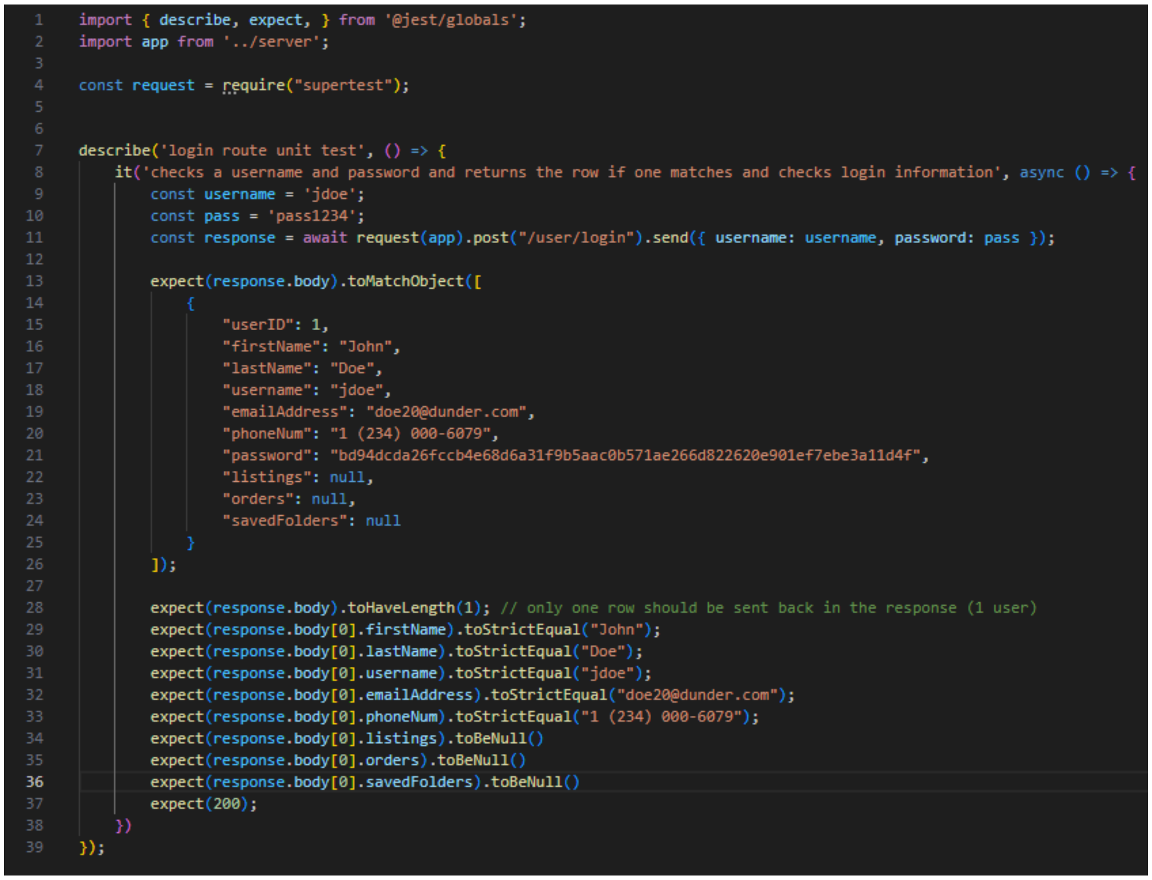
Test No.	Test Type	Target File or Screen	Test Name	Purpose of Test	Test Data or Situation	Expected Result	Actual Result	Outcome and Actions required
1	Login	Users.ts	Users logging in	Takes in username and password and if found, returns a matching user object	UN: jdoe Pass: 1234	One user object Column Sequence: userID, firstName, lastName, username, emailAddress, phoneNum, password, listings, orders, savedFolders Username and password values should match user input	To Be Determined	If a match is found, the screen should lead the user to the discover page. If not found, user should remain on login screen
2	Post Item	items.ts	Users selling a new item	Check if an item can be added to the item table	Takes in attributes from completed form for selling an item to create an item	One item object Column sequence: itemID, userID. name, description, price, size, quality, brand, color, hashtag, category, subcategory, photoURL, soldStatus	To Be Determined	If item is successfully added to the table, form should have a successful indication and item should show up on user's profile
3	Search items	items.ts	Users searching through items	Given a keyword, check if the items were correct parsed through by brand, description, name, and color	Users will search through the app for items they're interested in with the search bar	Returns an item or list of items One item object Column sequence: itemID, userID. name, description, price, size, quality, brand,	To Be Determined	Takes in keyword from search bar and screen refreshes with item cards based on the search query

						color, hashtag, category, subcategory, photoURL, soldStatus At least one of the columns: name, brand, color, or description should match the keyword		
4	saveIt em	savedFold ers.ts	User saving item	Confirm an item can be added to the savedItem table	Users may want to save items for inspiration	An item object should be added to the savedFolderTable via itemID	to be determined	once a user saves an item, the item should appear in the corresponding saved folder in the saved tab
5	change bio	users.ts	user changing profile	Confirm columns in a user row can be updated	Users may want to update their profiles	user row with corresponding userID to the user logged in should have updated information based on input. In this case, the column bio should change.	To be determined	Once a user saves their changes, the profile should rerender to reflect their new bio.
6	messa ge	chat.ts	user messaging	Confirms users can message each other. Messages should be in order of time.	Users may send messages to each other	New message row should be added to the messages table. Columns include message, timestamp, receiver, sender	to be determined	Once a user sends a message, the conversation should refresh with their message in it along with all the previous messages.

6 - Unit Testing

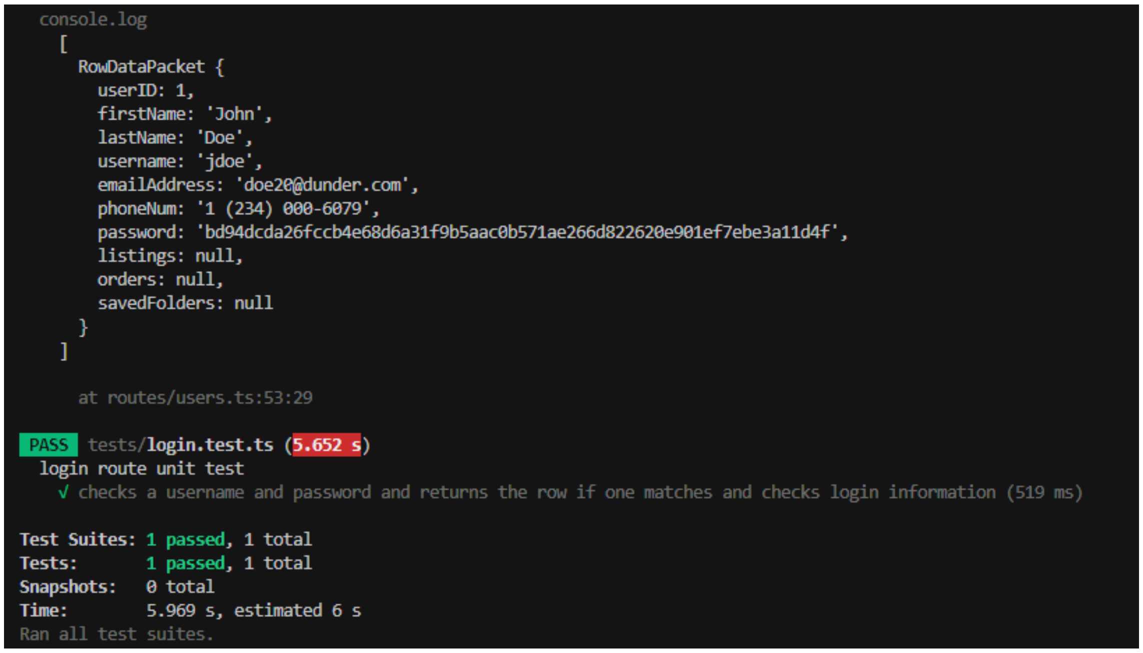
Each team member has written a unit test for a certain functionality path that they have developed within Drift. Unit tests are those that test the smallest units of projects and create a sort of documentation and confirmation of functionality to be behaving as we are expecting them too. Below are screenshots of the unit test code for each test and who wrote them.

Unit Test for Login Functionality (Written by - Jordan Rood)
Test Code:



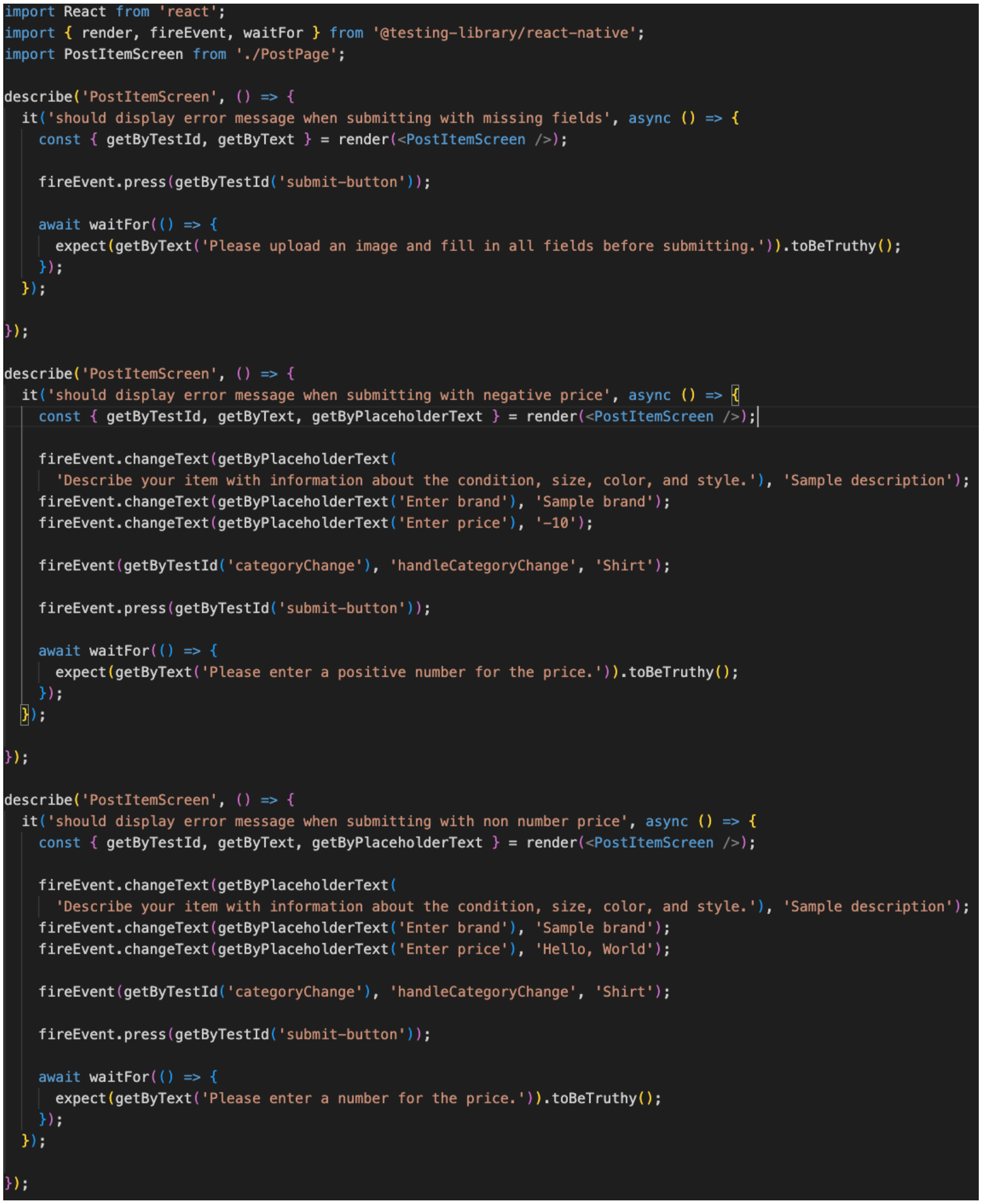
```
import { describe, expect, } from '@jest/globals';
import app from '../server';

const request = require("supertest");


describe('login route unit test', () => {
    it('checks a username and password and returns the row if one matches and checks login information', async () => {
        const username = 'jdoe';
        const pass = 'pass1234';
        const response = await request(app).post("/user/login").send({ username: username, password: pass });

        expect(response.body).toMatchObject([
            {
                "userID": 1,
                "firstName": "John",
                "lastName": "Doe",
                "username": "jdoe",
                "emailAddress": "doe20@dunder.com",
                "phoneNum": "1 (234) 000-6079",
                "password": "bd94dcda26fccb4e68d6a31f9b5aac0b571ae266d822620e901ef7ebe3a11d4f",
                "listings": null,
                "orders": null,
                "savedFolders": null
            }
        ]);

        expect(response.body).toHaveLength(1); // only one row should be sent back in the response (1 user)
        expect(response.body[0].firstName).toStrictEqual("John");
        expect(response.body[0].lastName).toStrictEqual("Doe");
        expect(response.body[0].username).toStrictEqual("jdoe");
        expect(response.body[0].emailAddress).toStrictEqual("doe20@dunder.com");
        expect(response.body[0].phoneNum).toStrictEqual("1 (234) 000-6079");
        expect(response.body[0].listings).toBeNull()
        expect(response.body[0].orders).toBeNull()
        expect(response.body[0].savedFolders).toBeNull()
        expect(200);
    })
});
```

Test Pass:



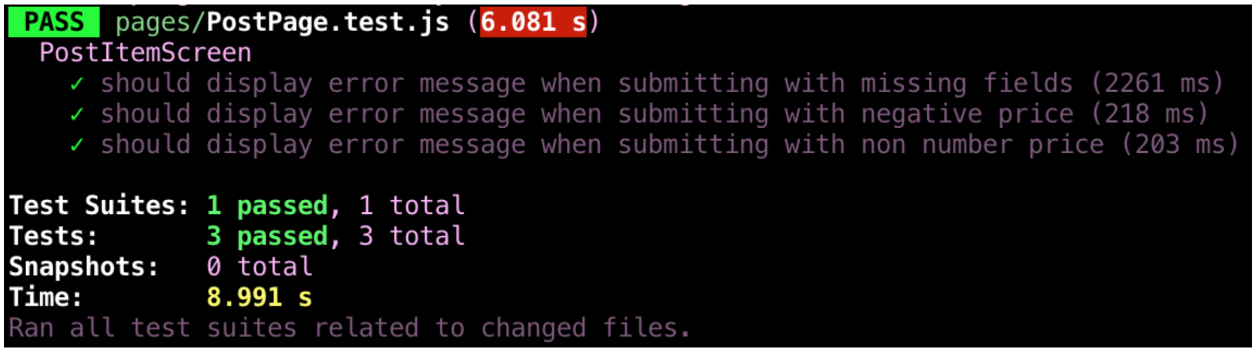
```
  console.log
    [
      RowDataPacket {
        userID: 1,
        firstName: 'John',
        lastName: 'Doe',
        username: 'jdoe',
        emailAddress: 'doe20@dunder.com',
        phoneNum: '1 (234) 000-6079',
        password: 'bd94dcda26fccb4e68d6a31f9b5aac0b571ae266d822620e901ef7ebe3a11d4f',
        listings: null,
        orders: null,
        savedFolders: null
      }
    ]

      at routes/users.ts:53:29

 PASS  tests/login.test.ts (5.652 s)
  login route unit test
    √ checks a username and password and returns the row if one matches and checks login information (519 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        5.969 s, estimated 6 s
Ran all test suites.
```

Unit Test for Error Handling with Post Item (Written by - John Jackson)
Test Code:



```
import React from 'react';
import { render, fireEvent, waitFor } from '@testing-library/react-native';
import PostItemScreen from './PostPage';

describe('PostItemScreen', () => {
  it('should display error message when submitting with missing fields', async () => {
    const { getByTestId, getByText } = render(<PostItemScreen />);

    fireEvent.press(getByTestId('submit-button'));

    await waitFor(() => {
      expect(getByText('Please upload an image and fill in all fields before submitting.')).toBeTruthy();
    });
  });

});

describe('PostItemScreen', () => {
  it('should display error message when submitting with negative price', async () => {
    const { getByTestId, getByText, getByPlaceholderText } = render(<PostItemScreen />);

    fireEvent.changeText(getByPlaceholderText(
      'Describe your item with information about the condition, size, color, and style.'), 'Sample description');
    fireEvent.changeText(getByPlaceholderText('Enter brand'), 'Sample brand');
    fireEvent.changeText(getByPlaceholderText('Enter price'), '-10');

    fireEvent(getByTestId('categoryChange'), 'handleCategoryChange', 'Shirt');

    fireEvent.press(getByTestId('submit-button'));

    await waitFor(() => {
      expect(getByText('Please enter a positive number for the price.')).toBeTruthy();
    });
  });

});

describe('PostItemScreen', () => {
  it('should display error message when submitting with non number price', async () => {
    const { getByTestId, getByText, getByPlaceholderText } = render(<PostItemScreen />);

    fireEvent.changeText(getByPlaceholderText(
      'Describe your item with information about the condition, size, color, and style.'), 'Sample description');
    fireEvent.changeText(getByPlaceholderText('Enter brand'), 'Sample brand');
    fireEvent.changeText(getByPlaceholderText('Enter price'), 'Hello, World');

    fireEvent(getByTestId('categoryChange'), 'handleCategoryChange', 'Shirt');

    fireEvent.press(getByTestId('submit-button'));

    await waitFor(() => {
      expect(getByText('Please enter a number for the price.')).toBeTruthy();
    });
  });

});
```

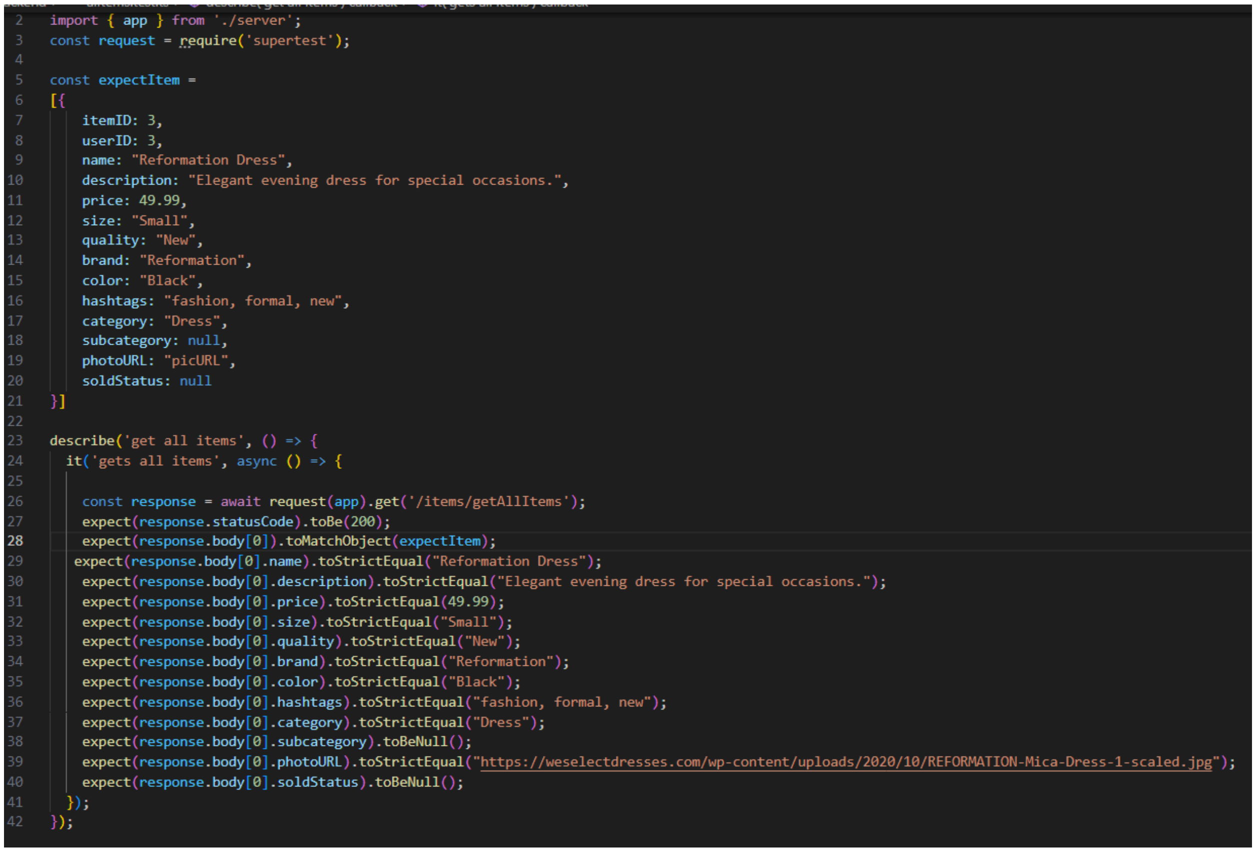
Test Pass:



```
 PASS  pages/PostPage.test.js (6.081 s)
  PostItemScreen
    ✓ should display error message when submitting with missing fields (2261 ms)
    ✓ should display error message when submitting with negative price (218 ms)
    ✓ should display error message when submitting with non number price (203 ms)

Test Suites: 1 passed, 1 total
Tests:       3 passed, 3 total
Snapshots:   0 total
Time:        8.991 s
Ran all test suites related to changed files.
```

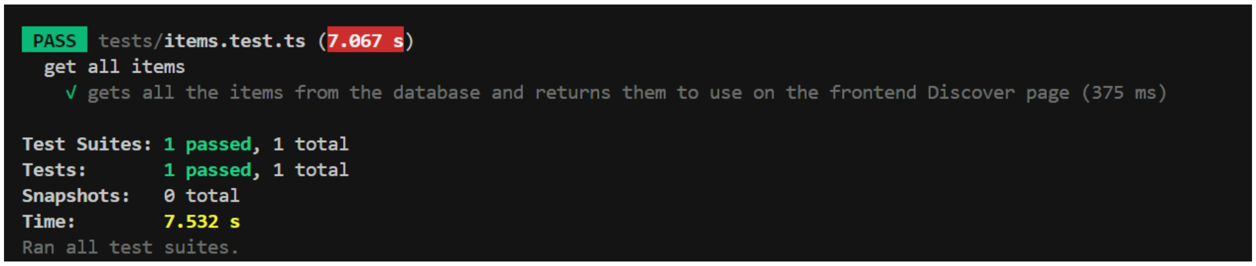
Unit Test for Get All Items Functionality (Written by - Fiorina Chau)

Test Code:



```
import { app } from './server';
const request = require('supertest');

const expectItem =
[{
    itemID: 3,
    userID: 3,
    name: "Reformation Dress",
    description: "Elegant evening dress for special occasions.",
    price: 49.99,
    size: "Small",
    quality: "New",
    brand: "Reformation",
    color: "Black",
    hashtags: "fashion, formal, new",
    category: "Dress",
    subcategory: null,
    photoURL: "picURL",
    soldStatus: null
}]

describe('get all items', () => {
  it('gets all items', async () => {

    const response = await request(app).get('/items/getAllItems');
    expect(response.statusCode).toBe(200);
    expect(response.body[0]).toMatchObject(expectItem);
   expect(response.body[0].name).toStrictEqual("Reformation Dress");
    expect(response.body[0].description).toStrictEqual("Elegant evening dress for special occasions.");
    expect(response.body[0].price).toStrictEqual(49.99);
    expect(response.body[0].size).toStrictEqual("Small");
    expect(response.body[0].quality).toStrictEqual("New");
    expect(response.body[0].brand).toStrictEqual("Reformation");
    expect(response.body[0].color).toStrictEqual("Black");
    expect(response.body[0].hashtags).toStrictEqual("fashion, formal, new");
    expect(response.body[0].category).toStrictEqual("Dress");
    expect(response.body[0].subcategory).toBeNull();
    expect(response.body[0].photoURL).toStrictEqual("https://weselectdresses.com/wp-content/uploads/2020/10/REFORMATION-Mica-Dress-1-scaled.jpg");
    expect(response.body[0].soldStatus).toBeNull();
  });
});
```

Test Pass:



```
 PASS  tests/items.test.ts (7.067 s)
  get all items
    √ gets all the items from the database and returns them to use on the frontend Discover page (375 ms)

Test Suites: 1 passed, 1 total
Tests:       1 passed, 1 total
Snapshots:   0 total
Time:        7.532 s
Ran all test suites.
```

7 - Contributions of Team Members

Team Members	Time Worked on Project Part	Specific Activities Worked On
Jordan Rood	5 hours	- Abstract / Project Updates & Changes - User Stories and Acceptance Criteria - Login Unit Test
Fiorina Chau	4 hours	- Testing Strategy - Search Items by Keyword Unit Test
John Christian Jackson	4 hours	- Testing Workflow - Post Item Unit Test