Test Plan Document

Prepared by:

William Cross, TC Prater, Joshua Coombs

07/08/2022

OBJECTIVES AND TASKS

Objectives

We aim to ensure stability and functionality for our website through unit testing. Each team member will be assigned a webpage or two (Joshua: userPage, loginPage, and homePage(Selenium); William: inventory and marketplace; TC: petPage), and will be responsible for the frontend, backend, and testing for that page and its required functionality. Communication will be held through Zoom and Slack. The final product will be on the training batch's GitHub under the p2-Team-5 repository.

Tasks

As required, we first create at least five feature files, which in turn create at least 20 unit tests. Then we will write our code, doing manual testing throughout this process to make sure the code functions as intended. Once the code is done, we will return to the feature files and update their unit tests' code as necessary. Each test case to be placed within the feature files has been written in the team's

Notion workspace, and will be included in a Test Cases document alongside our final results thereof.

SCOPE

General

We will be running frontend tests using Cucumber and Selenium to guide our overall development, and using JUnit tests to ensure proper functionality of our Service layer towards the end of the project.

Tactics

Talking over Zoom and, if necessary, Slack, will be used to ensure communication between teammates. Monday and Tuesday were spent writing feature files, and the tests will be expanded and adapted on Friday to adjust for new implementations in the code.

TESTING STRATEGY

Once our code is written, we will run unit and Cucumber tests to ensure functionality. Untested parts of the Service layer will have new unit tests written to test them, and manual testing will be done to sanity-check the additional frontend features.

Unit Testing

Definition:

Eclipse contains a Coverage tool that allows the developer to see which lines of code have actually ran during a given batch of unit tests. This will be used to measure the amount of our project that has been directly tested during service-layer unit tests. 100% coverage is, of course, ideal, but full coverage of branching methods is the minimum target.

Participants:

William, Joshua, and TC are all responsible for Unit Testing

Methodology:

Unit testing will be conducted through Eclipse using JUnit, organized with Cucumber and Selenium as necessary. Each member will be responsible for the unit tests of their own code (Joshua for Users, William for Items, and TC for Pets).

System and Integration Testing

Definition:

Testing the project using only the frontend, in a way that requires the backend to be active (ex: logging in).

Participants:

William, Joshua, and TC are all responsible for System and Integration Testing

Methodology:

We will run the feature files from the start of this project after the website and server have been coded and properly hosted. Any failure or error among the unit tests will be examined, with either the unit test or the code being changed as is appropriate for the situation.

ENVIRONMENT REQUIREMENTS

Our tests are all running on Windows operating systems, editions 10 and 11.

Our tests are written using in the Eclipse IDE, and our project is written using Eclipse and Visual Studio Code.

Our WebDrivers utilize the Edge, Chrome, and Firefox browsers

TEST SCHEDULE

We opted not to use test milestones in this project, as the initial required suite of required unit tests ensured we had clear and concrete coding goals to towards with from the start.

CONTROL PROCEDURES

Problem Reporting

If we discover a problem, we first will do manual testing and/or re-coding to fix it. Failing that, it will be mentioned to the Zoom call. If help cannot be gathered to fix it by the end of the day, it shall be entered as an item on the project's Kanban board.

Change Requests

The division of labor chosen allows each developer to have full update authority for their own code. Any required bleed-through (ex: needing to list pets on the user page) shall be discussed in the Zoom call, and pushed alongside other code

updates. No formal update criteria was discussed, but updates usually proceeded whenever a particular feature had been implemented, or enough thereof for another developer to adapt code to or from it (ex: pushing a hard-coded login so the Users developer can connect it to the backend)

FEATURES TO BE TESTED

- ▼ Top Bar
  - ▼ Given the user is on the site
  - ▼ When the user clicks on a link
  - ▼ Then the user is taken to the correct page
- ▼ Home Page
  - ▼ Create a pet
    - ▼ Positive
      - ▼ Given the user is on the homePage
      - ▼ When the user clicks create a pet and fills out the form
      - ▼ Then the pet shows up in the user pets list
    - ▼ Negative (unnecessary, submit is disabled in frontend)
  - ▼ Search
    - ▼ Positive
      - ▼ Given the user is on the homePage
      - ▼ When they type in a query in the search bar
      - ▼ Then the query result is displayed
    - ▼ Negative (unnecessary, submit is disabled in frontend)
      - ▼ Given the user is on the homePage
      - ▼ When they type in an incorrect query in the search bar
      - ▼ Then the query result is not displayed
- ▼ Login Page

- Login (positive)
  - Given the user is on the login page
  - When the user inputs a correct user/password combo
  - Then the user becomes logged in
- Login (negative)
  - Given the user is on the login page
  - When the user inputs an incorrect user/password combo
  - Then the user is given an error message

- Marketplace
  - No functionality here

- Inventory
  - Pet Inventory Subheadings
    - Given the user has pets
    - When the user loads the inventory page
    - Then subheadings dividing the inventory should appear
  - Item click popup/dropdown
    - Given the user is on the inventory page
    - When the user clicks on an item
    - Then a dropdown or popup should appear that allows the item to be assigned to places
  - Items in pet inventories/Item dropdown option submitted
    - Given the pet exists
    - And the user is on the inventory page
    - When an item is assigned to the pet’s inventory
    - Then that item appears in the pet’s section of the inventory

- User Profile

- Basic Loading
  - Given the user exists
  - When the user page is loaded
  - Then the user’s data is displayed
- User Data Editing
  - Given the user is logged in
  - And is looking at their own profile
  - When the user changes data fields
  - And presses submit
  - And refreshes
  - Then the user data should be updated
- Show Pets
  - Given the user exists
  - And has pets
  - When the user profile loads
  - Then pets should be displayed

- Pet Profile
  - Basic pet display
    - Given the pet exists
    - When the user navigates to that pet page
    - Then the pet data is displayed
  - Edit pet data
    - Given the pet exists
    - And the user is logged in
    - And the user is on the relevant pet page
    - When the user inputs new data into the open fields

▼ And presses submit

▼ And refreshes

▼ Then the new data is displayed

FEATURES NOT TO BE TESTED

In general, stretch goals do not have feature files associated with them.

RESOURCES/ROLES & RESPONSIBILITIES

Responsibility for running and creating tests falls to the developer in charge of that object/page, as discussed above.

SCHEDULES

Deliverables

Identify the deliverable documentation. Examples include:

- Test Plan
- Test Cases
- Test Summary Reports

DEPENDENCIES

All testing done was functional in nature, as we do not have the resources for comprehensive non-functional tests within this project's scope.

RISKS/ASSUMPTIONS

No risks were estimated, due to the simplicity and small scope of this project.

TOOLS

JUnit, Cucumber, and Selenium were used to automate tests, and bugs were tracked with the Kanban board mentioned above.

APPROVALS

Who must approve this plan? Provide space for the signatures and dates below for your record.

- [x] ~~Joshua: 07/16/2022~~
- [ ] William:

- [x] ~~TC: 07/15/22~~