Software Test Plan:

CSUN Dashboard

Prepared by:

Christian Jarmon

12th November 2022

1.0 INTRODUCTION

A brief summary of the product being tested. Outline all the functions at a high level.

The product being tested, named CSUN Dashboard, is a set of search tools made for students and advisors to allow the exploration of class catalogs, professors, and majors offered by the university.

2.0 OBJECTIVES AND TASKS

2.1 Objectives

Describe the objectives supported by the Software Test Plan, eg., defining tasks and responsibilities, the vehicle for communication, the document to be used as a service level agreement, etc.

To identify deviations of functionality from its specified requirements for the functional

portions of the frontend made in ReactJS and deviations of data performance and accuracy from API.

2.2 Tasks

List all tasks identified by this Test Plan, i.e., testing, post-testing, problem reporting, etc.

3.0 SCOPE

General

This section describes what is being tested, such as all the functions of a specific product, its existing interfaces, and the integration of all functions.

Verify that all frontend components function as required and that API returns data expected and required for such components to function properly.

- Planner
    - Verify correct API endpoints are used.
    - Verify responses for every possible user action
        - If a user picks a class, it responds by listing it.
        - If a user removes a class, it delists it.
        - If a user picks a conflicting class, it lets the user know and prevents the addition of said conflicting class/
        - Shows the cost of selected classes after every addition or removal
- Professor Search
    - Verify correct API endpoints are used
    - Verify correct data is returned
    - Ratings
        - Verify each rating hyperlink leads to the correct rating page.
        - Verify the post endpoint is functional

4.0 TESTING STRATEGY

Describe the overall approach to testing. For each major group of features or feature combinations, specify the approach which will ensure that these feature groups are adequately tested. Specify the major activities, techniques, and tools which are used to test the designated groups of features.

The approach should be described in sufficient detail to permit the identification of the major testing tasks and the estimation of the time required to do each one.

- For the front end, there exist two layers of responsiveness, the proper function calls for every user action and the visual response that the user sees as a result of those function calls. Therefore, what has to be tested is that the proper functions are being called at all times and that they return what they are supposed to given certain inputs and situations.

- For all backend API GET and POST requests, the application Postman will be used to verify data integrity separate from the actual integrated product to avoid the UI from mystifying results

- API
    - Manual granularity checks must occur to identify what the API is supposed to return according to the specification
    - Automated scripts will be written to compare the two string outputs, one manual, and one from the API.
        - This will be written in Bash by making use of out-the-box bash commands ‘curl’ and ‘diff’
            - ‘curl’ documentation: https://curl.se/docs/manpage.html
            - ‘diff’ documentation: https://www.man7.org/linux/man-pages/man1/diff.1.html

4.1 Unit Testing

Definition:

Specify the minimum degree of comprehensiveness desired. Identify the techniques used to judge the comprehensiveness of the testing effort (for example, determining which statements have been executed at least once). Specify any additional completion criteria (for example, error frequency). The techniques to be used to trace requirements should be specified.

- Due to the uniformity of the API and the UI/UX components, a high-granularity testing structure is not required but rather a general approach. For the API, every endpoint just has to be confirmed against one set of parameters, which would then confirm the functionality of all other parameters, such as the Subjects for the Catalog or Emails for the Professor endpoints. For the front end, most interactions follow a GET request and then a Display procedure for what was returned.

Participants:

List the names of individuals/departments who would be responsible for Unit Testing.

- Nima Shafie (API)
- David Huezo (UI/UX)

Methodology:

Describe how unit testing will be conducted. Who will write the test scripts for the unit testing, what would be the sequence of events of Unit Testing and how will the testing activity take place?

- API
  - Manually record the intended results of each endpoint and then compare it against a simulated GET request
  - Use an automated Bash script and make use of the 'diff' command to determine equality between the manual output and GET output
- UI/UX
  - Verify that every interaction calls the correct GET endpoints.
  - Using the same approach as the API, an automated Bash script will be used to determine correct page routing (as a result of the interaction), using the 'curl' bash command.

4.2 System and Integration

Testing Definition:

List what is your understanding of System and Integration Testing for your project.

Exiting the development environment, a build process with be performed with NPM's build tool for ReactJS verifying the results of the separate unit tests in the build

Participants:

Who will be conducting System and Integration Testing on your project? List the individuals that will be responsible for this activity.

- David Huezo
- Micheal Balian
- Nima Shafie

Methodology:

Describe how System & Integration testing will be conducted. Who will write the test scripts for the unit testing, what would be the sequence of events of System & Integration Testing, and how will the testing activity take place?

Using the results from the UNIT tests, a build process will take place using NPM's Build tool. The same interactions and the same GET requests will occur and will be verified using the results from the UNIT tests.

4.3 Performance and Stress Testing

IF APPLICABLE, IF NOT PUT N/A

Definition:

List what is your understanding of Stress Testing for your project.

- Flood the server with a bunch of API GET requests to test response time and identify any bottlenecks

Participants:

Who will be conducting Stress Testing on your project? List the individuals that will be responsible for this activity.

Christian Jarmon

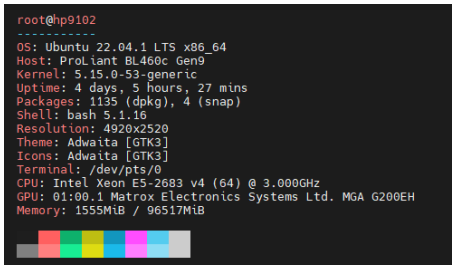
Methodology:

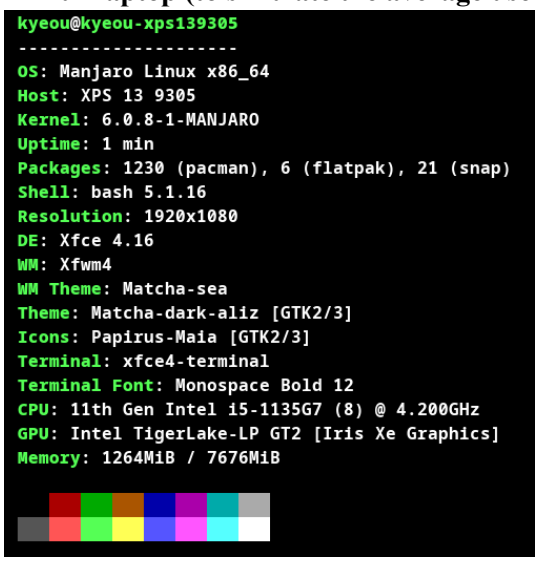
Describe how Performance & Stress testing will be conducted. Who will write the test scripts for the testing, what would be the sequence of events of Performance & Stress Testing, and how will the testing activity take place?

- A multithreaded Python script will be written to perform a GET Request on all the endpoints some set amount of times.
    - Quantity of requests will be as follows
        - 10
        - 50
        - 100
        - 500
        - 1000
        - 5000
        - 10000
        - 50000
- After each test run, the time it took for each request to complete client-side will be analyzed as such.
    - These times will be averaged and analyzed as such. (Example Output)
        - Test Start Time: 2022-12-03 09:13:31.654382+00:00
        - ------
        - Processes finished 500
        - Average Time: 1.207428 secs
        - Best: 0.085 secs
        - Worst: 4.263 secs
        - ------
    - The server-side logs will be recording in the same way
        - They will be separate to measure for certain bottlenecks should they happen either in Database queries, parsing, frontend parsing, or connection bottlenecks.

5.0 ENVIRONMENT REQUIREMENTS (TOOLS)

List test tools and environment in which those tools will be used (i.e. operating system, PC specs.)

- **All tests will be conducted in a linux environment with automated bash scripts running on a centralized server with only one outside machine for control metrics**
- **The tests will be conducted with the use of 2 machines.**
  - **A centralized Server**
    - 

  - **A linux laptop (to simulate the average user)**
    - 

6.0 TEST SCHEDULE

Include test milestones identified in the Software Project Schedule as well as all item transmittal events.

Define any additional test milestones needed. Estimate the time required to do each testing task. Specify the schedule for each testing task and test milestone. For each testing resource (that is, facilities, tools, and staff), specify its periods of use.

- N/A

7.0 CONTROL PROCEDURES

Problem Reporting

Document the procedures to follow when an incident is encountered during the testing process. If a standard form is going to be used, attach a blank copy as an "Appendix" to the Test Plan. In the event you are using an automated incident logging system, write those procedures in this section.

- Any and all errors will be logged during runtime.

- The test scripts will be have a log of successful checks and an error log of any issues during execution (i.e. connection error, I/O Errors, etc)

Change Requests

Document the process of modifications to the software. Identify who will sign off on the changes and what would be the criteria for including the changes to the current product. These modules need to be identified if the changes affect existing programs.

- Lead Christian Jarmon is responsible for all sign offs to changes while David Huezo and Micheal Balian will be responsible for requesting such changes

8.0 RESOURCES/ROLES & RESPONSIBILITIES

Specify the staff members who are involved in the test project and what their roles are going to be (for example, Mary Brown (User) compiles Test Cases for Acceptance Testing). Identify groups responsible for managing, designing, preparing, executing, and resolving the test activities as well as related issues. Also, identify groups responsible for providing the test environment. These groups may include developers, testers, operations staff, testing services, etc.

- Test Design: Christian Jarmon, Nima Shafie
- Test Execution: Michael Balian, David Huezo

9.0 RISKS/ASSUMPTIONS

Identify the high-risk assumptions of the test plan. Specify contingency plans for each (for example, a delay in delivery of test items might require increased night shift scheduling to meet the delivery date).

- Testing with the assumptions that remedies to identified bottlenecks can be produced
    - Any possible issue with the system that arises should have naive solutions and optimal solutions.
        - In the case the optimal solution can’t be implemented in time, the naive solution will be deployed