CSUN Pantry Pal

Testing Plan

Group 6

Ziaur Chowdhury

Mark Kozlov

Esteban Maciel

Brian Melgar

Sheran Morais

CSUN CALIFORNIA STATE UNIVERSITY NORTHRIDGE

Date: 2021-07-11

1. Non-execution Testing

Documentation will be tested using an informal walkthrough approach. The documentation includes the SRS, the final report, all diagrams, and any other group assignments. During this process, documentation should be cross-referenced with other documentation. This should be an iterative process that will happen several times during the development of the product. It will also have to be performed in response to changes to the design, requirements, or even changes in the implementation.

The following aspects shall be tested:

- Spelling/grammar errors
- Accuracy in describing and representing the product
- Measurability - statements should be measurable for the purpose of validation
- Consistency with itself, other documentation, and the code
- Ambiguities
- Completeness
- Redundancies

Some documentation may be directly updated during testing to correct discovered issues. In other cases, comments shall be left and addressed later. Larger problems may warrant discussion with the team to determine an appropriate fix.

Non-executable testing of software shall refer to the documentation for both validation and verification. Among the documentation, the use cases will likely be the most useful in assessing the correctness and completeness of the software. For completeness, testing shall ensure each use case was implemented. For correctness, testing shall ensure the implementations follow the written main/alternative flows and adequately handle the listed exceptions.

The software also has opportunities to be tested during pair programming and during the review cycle for pull requests submitted on GitHub. In addition to testing for correctness and completeness, the review cycle analyses the changes' design to ensure it was maintainable. Reviews involve leaving annotations on the code with comments or suggestions, which would then be addressed by the author of the pull request. Those changes would then be subject to another review until the pull request was approved.

The security policy shall be reviewed for the AWS user which is used by CI to push the Docker image. It should be confirmed that the user only had the minimum permissions necessary to perform its duties i.e. push the image to the ECR registry.

2. Execution Testing

A majority of executable testing will be done manually due to the complexity of automating front-end/user interface tests. Testers shall use the website through various browsers to confirm the user interface appears and functions as intended. The use case diagram and descriptions shall be referenced during this process to ensure the testers go through the functionality of every requirement. The activity diagram, system-level sequence diagram, and the state-machine diagram will also be helpful in this regard.

Issue tickets shall be created on GitHub to keep track of any issues found during testing. These tickets will be labelled with a priority depending on the severity of the issue. For example, issues directly impeding the functionality of a use case should be given the highest priority.

The performance requirements specified in the SRS shall be tested informally by using the website with the browser's development tools open. The tools show the time taken for various operations. Each operation shall be repeated several times on different browsers. The times provided by the browser shall be used to ensure the software performs within specifications.

Security headers shall be tested by providing the website's URL to https://securityheaders.com/ for scanning. It will report any unset or misconfigured headers. If the grade is not A+, testers shall create an issue ticket detailing what needs to be changed in order to achieve a better grade.

Some executable testing was already automated; the backend code has 91% branch coverage. Testing was implemented using the *pytest* framework and coverage was reported using the *coverage* library. However, this figure is a bit misleading. Actually, most of the assertions in the tests focus on the API code. The tests are all integration tests which simulate actual HTTP requests to the endpoints with some data and check the responses returned. The tests are mostly black box rather than white box to avoid tightly coupling tests to the code. There are a few exceptions to this e.g. testing the authentication key is passed to the Spoonacular request.