Software Engineering CSC 648 Section 4

Milestone 4 REVISION

FraGrant

Team 3 - Debuggers

Ruqaiyah Angeles - Team Lead

Aisha Asif - Scrum Master

Chris Solorzano - Back End Lead

Shahriz Malek - Front End Lead

Mandy Noto - GitHub Master

Ahmed Ballal - Universal Helper

12.05.22

Revision table (revisions)

Revision ID	Revision Date
1	12.05.22

Contents

Product Summary

The name of our product is FraGrant, an effective line of fragrances that provide many benefits to you. With each purchase of a fragrance, you are gaining the temporary ability to enhance something about yourself. From sleep deprivation and social anxiety to neutralizing a bad odor, we have got you covered. We were mainly focused on college students as these are some recurring issues we have seen. We hope that providing these temporary fixes, that it assists students to be better students.

Some of the major committed functions that were implemented within our website are: Users should be allowed to add to cart as a guest (Priority 3),Users should be allowed to browse the site as a guest. (Priority 1), Users should not be allowed to checkout without signin/signout. (Priority 1), Users should be able to view the product. (Priority 1), Users should be able to add products to the cart.(Priority 1), Users should be able to remove products from the cart. (Priority 1), Users should be able to view their order. (Priority 1), Users should be able to delete or cancel their order. (Priority 1), Users should be allowed to purchase products. (Priority 1), Users should be able to check the history of the products they were viewing, Users should be allowed to buy again. (Priority 1), Users should be able to track their order. (Priority 1), Purchase - Every purchase should have an order number. (Priority 1), Users should be allowed to receive a random product. (Priority 3)

Creating a unique function for a website was a tricky task. What would have been unique about our product features was recommending a random fragrance because this allows the customer to shop with ease. They can take the stress of choosing off of them and leave their purchase to our random fragrance generator. The way it works is that the user would need to click into a tab on the screen and click “Spin for a randomized product.” From there, one of the products would be added to their cart and the only thing left is to browse the rest of the catalog, then purchase!

The URL to our product is https://fragrantdebuggers.live/Product.
Visit our website here: Fragrantdebuggers.live

QA Testing

For the functions, write a QA unit & integration test plan and results (check class slides). You have to write a report to contain the following information :

I. Unit Test

Define, develop, write, build and run unit test cases in your chosen unit testing framework (e.g., Jest).
The unit testing framework that we chose was jest. We tested two priority 1 features.

One of these priority 1 features that we tested were related to the user stories. Specifically, one of these user stories is about being able to log in as a computer science stem student.

One of the main focuses of the test for login, at least, is validating the that the user inputted the correct format for an email address; for example, checking if there is an '@' symbol in the input. We also made sure that the login pages rendered correctly.

The other priority 1 feature that we tested was the sign-up. The steps we used to do this is identical to validating the correct log-in user input.

If possible, integrate with CI framework (e.g., Github Actions) so that your unit testing will be automatically triggered (extra points).
We integrated our unit testing framework, jest, with the continuous integration framework called GitHub actions. The trigger we used was upon the pull request from one branch to another.

GitHub URL of test cases directory

1. Directory: csc-648-project-csc648-04-fa22-team03/applications/client/src/components/__tests__ at dev · CSC-648-SFSU/csc-648-project-csc648-04-fa22-team03 (github.com)
   a. home.test.js csc-648-project-csc648-04-fa22-team03/home.test.js at dev · CSC-648-SFSU/csc-648-project-csc648-04-fa22-team03 (github.com)
   b. login.test.js csc-648-project-csc648-04-fa22-team03/login.test.js at dev · CSC-648-SFSU/csc-648-project-csc648-04-fa22-team03 (github.com)

c. signup.test.js [csc-648-project-csc648-04-fa22-team03/signup.test.js at dev · CSC-648-SFSU/csc-648-project-csc648-04-fa22-team03 (github.com)]

II. Integration Test

Define the integration testing cases according to the format.
The format should include test case id; test case description; dates tested, Test scenario with steps to follow, Prerequisites, Test data and Test results (PASS or FAIL for each tested browser). If fail, assignee should be included. Once debugging is done by the assignee, its status is also recorded including how to fix it (e.g., Github url for the bug fix).

Perform testing and record the testing results.
You can record the test results in the bug tracker system of your choice like Jira, or Github Issues.

Analyze of your test coverage for your P1 features and describe it.
Out of all your P1 features, how many are tested?
Out of all our P1 features, we tested…

III. Code Review

GitHub pull request urls

1. refactor(products): delete duplicate relax.js by aballal-source · Pull Request #18 · CSC-648-SFSU/csc-648-project-csc648-04-fa22-team03 (github.com)
2. Feat/update quantity by chrisxsolo · Pull Request #21 · CSC-648-SFSU/csc-648-project-csc648-04-fa22-team03 (github.com)
3. feat(applications): Added redirect and styling of checkout by chrisxsolo · Pull Request #22 · CSC-648-SFSU/csc-648-project-csc648-04-fa22-team03 (github.com)
4. feat(applications): Added database functionality with login and signup by chrisxsolo · Pull Request #25 · CSC-648-SFSU/csc-648-project-csc648-04-fa22-team03 (github.com)
5. feat: updated mongodb ipaddress and webserver hosting by ShahrizSchool · Pull Request #26 · CSC-648-SFSU/csc-648-project-csc648-04-fa22-team03 (github.com)

6. feat(applications): created functional order page with database by chrisxsolo · Pull Request #27 · CSC-648-SFSU/csc-648-project-csc648-04-fa22-team03 (github.com)
7. feat(m5): test the database IP on the hosted server by ShahrizSchool · Pull Request #36 · CSC-648-SFSU/csc-648-project-csc648-04-fa22-team03 (github.com)
8. feat/connect server to db by ShahrizSchool · Pull Request #35 · CSC-648-SFSU/csc-648-project-csc648-04-fa22-team03 (github.com)
9. Feat/checkout login by chrisxsolo · Pull Request #33 · CSC-648-SFSU/csc-648-project-csc648-04-fa22-team03 (github.com)
10. feat(.github): track and setup login-signup workflow by mandynoto · Pull Request #37 · CSC-648-SFSU/csc-648-project-csc648-04-fa22-team03
11. test: unit testing for login, homepage, and signup by ShahrizSchool · Pull Request #40 · CSC-648-SFSU/csc-648-project-csc648-04-fa22-team03 (github.com)

❖ Coding style description and enforcement

The coding style that we decided on was that which was standardized by Google. To begin with, the files we mainly worked with were javascript, html, and css. To be consistent we sought after a coding style that is standardized by the same company, and we chose Google. The names of these coding styles for javascript; html; and css, respectively are: the Google JavaScript Style Guide, and the Google HTML/CSS Style Guide. These styles influenced the way we structured our files, determined our bracing structures, and file naming conventions, etc.

To enforce our style description, our software development team would check with the styles from their source websites, https://google.github.io/styleguide/jsguide.html and https://google.github.io/styleguide/htmlcssguide.html, to determine if they were following it or not. Then, code reviewers of pull requests would double check again if it was being followed, rejecting the pull request if it was not.