

SW Engineering CSC648/848 Spring 2021

PipeWave

"The Fast-Track from Student to Professional"

Team Lead/Front End: Jennifer Finaldi,
Team Lead Email: jfinaldi32@gmail.com
Database/Back End: Robert Cacho Ruiz
Front End: Kevin Danh
Back End: Jahir Hernandez
Github: Anthony Nguyen

Section 02, Team 04

Milestone 4-- 04/19/2021

History Table (Revision Summary):

04/19/2021: Document created

1) Product Summary

Pipewave

- Diverse signup form depending on occupation
- Professors able to rate a student based on their talents
- Users of industry occupation are granted Alerts
- Customize and edit profile for:
 - Students
 - Professors
 - Industry individuals

Our product includes a unique feature that allows industry users and professors to filter their search results of students' cultural backgrounds and identity. This grants more opportunity for people of cultural backgrounds or unique identities to be widely represented within the workspace of a company. While our regular search function returns results that match to what keywords you enter, the advanced search with selectable filters provides more depth to what cultural background, identity, and major you are looking for.

URL: 35.235.77.107

2) Usability test plan

Test Objectives:

For this usability test plan, we are assessing our search function to verify it's working correctly and showcase our unique feature of our filters provided on our website. Since our website is focused on allowing professors or industry professionals to search for talents and possible recruits for their company, it's imperative that our search function works without a hitch to attract more attention to potential students and our site. Although the filters in the advanced search may be an optional alternative to utilize the search feature, it's still an important task of having this feature work correctly to guarantee to bring up the appropriate students they are interested in and leave a good impression of our unique feature provided on our site.

❖ Test Background and Setup:

For the tester to have the appropriate setup to execute the test plan, they must first have Node installed to their machine. The tester must then git clone our project code to their machine to prepare testing it. Using whatever IDE they have already installed or available to them, open the cloned project folder and use the terminal to cd into applications. Once they enter this directory, they execute npm install to retrieve the dependencies where they can finally execute nodemon to launch the server.

To begin the test, we place the tester's starting point at the homepage where all the registered users are displayed. This position is where all users are greeted when launching our site, so running tests at this point should yield more accurate results to how users would theoretically handle and navigate the website.

Regarding testing these particular functions, our intended users to complete the tasks provided would be professors and/or industry professionals. Since students are responsible for making their profile and portfolio appealing as well as choosing to share any additional details like their identity and background, our intended users are the ones who primarily will utilize our search and filter features.

The tester is to enter the URL localhost:3000 upon activating our server to be directed to our website.

What we will be measuring from the results of these tests are how effective our search function is in returning the appropriate results to the user, the tester's level of satisfaction towards how well the site handled the tasks given, and how the tester responds to the Lickert questionnaire regarding the testing.

❖ Usability Task Description:

1. Search for Computer Science majors
2. Search for users named "Robert"
3. Apply filters to your advanced search
4. Search for invalid characters

To measure effectiveness, we would check the average of accurate search results returned to the tester's search queries. To do this, we verify how many results should be returned to a search query and compare to the output results.

To measure efficiency, we determine the average time it takes from having all the tasks completed. This is reliant on how fast our site can pull up the results to the tester's search query and return it to them.

❖ Lickert Subjective Test:

The search feature delivered me the results quickly.

- ☐ Strongly Agree
- ☐ Agree
- ☐ Neutral
- ☐ Disagree
- ☐ Strongly Disagree

The advanced search filters are a helpful alternative to my search queries.

- ☐ Strongly Agree
- ☐ Agree
- ☐ Neutral
- ☐ Disagree
- ☐ Strongly Disagree

My search results that were returned to me were what I was looking for.

- ☐ Strongly Agree
- ☐ Agree
- ☐ Neutral
- ☐ Disagree
- ☐ Strongly Disagree

3) QA test plan

QA Test Plan

Test Objectives:

For the QA Test objectives, we will be looking for bugs and errors when applying the search filter, how the search filter results show results in correct order, and for all the information to show up. We will be making sure the website performs the search results within the given specs. When given very specific tasks to the user, the website must be tested to show all the search results related to the search entry and filters. Test will report PASS or FAIL. We will be checking the SE process and documentation.

HW and SW Setup(Including URL):

The user will begin by turning on the PC. They will then start their browser (chrome, safari, firefox) that is supported. They will enter in the URL 35.235.77.107 into the address bar. From there the user will type the desired entry into the search bar located in the upper half side of the screen. The user will click on the button 'advanced' and click on the bubbles alongside the categories in order to filter the results. The user may then click on the 'search' button.

Features to be tested:

- Major Search Results
- Name Search Results
- Filter Success
- Invalid Character Search Results

QA Test Plan: - table format:

Safari

Test #	Test Title	Test Description	Test Input	Expected Correct Output	Test Results
1	Name Search + Major	Test for name field	"Robert", "Computer Science" Selected	Display all "Robert" users or names containing "Robert" from Database.	FAIL
2	Filter	Test for advanced filter	"asdf", Male Selected	Display all "asdf" users in the database that are male	FAIL
3	Invalid Characters	Test to see error handling for invalid characters	"!@#"	Return to home page	PASS

Chrome

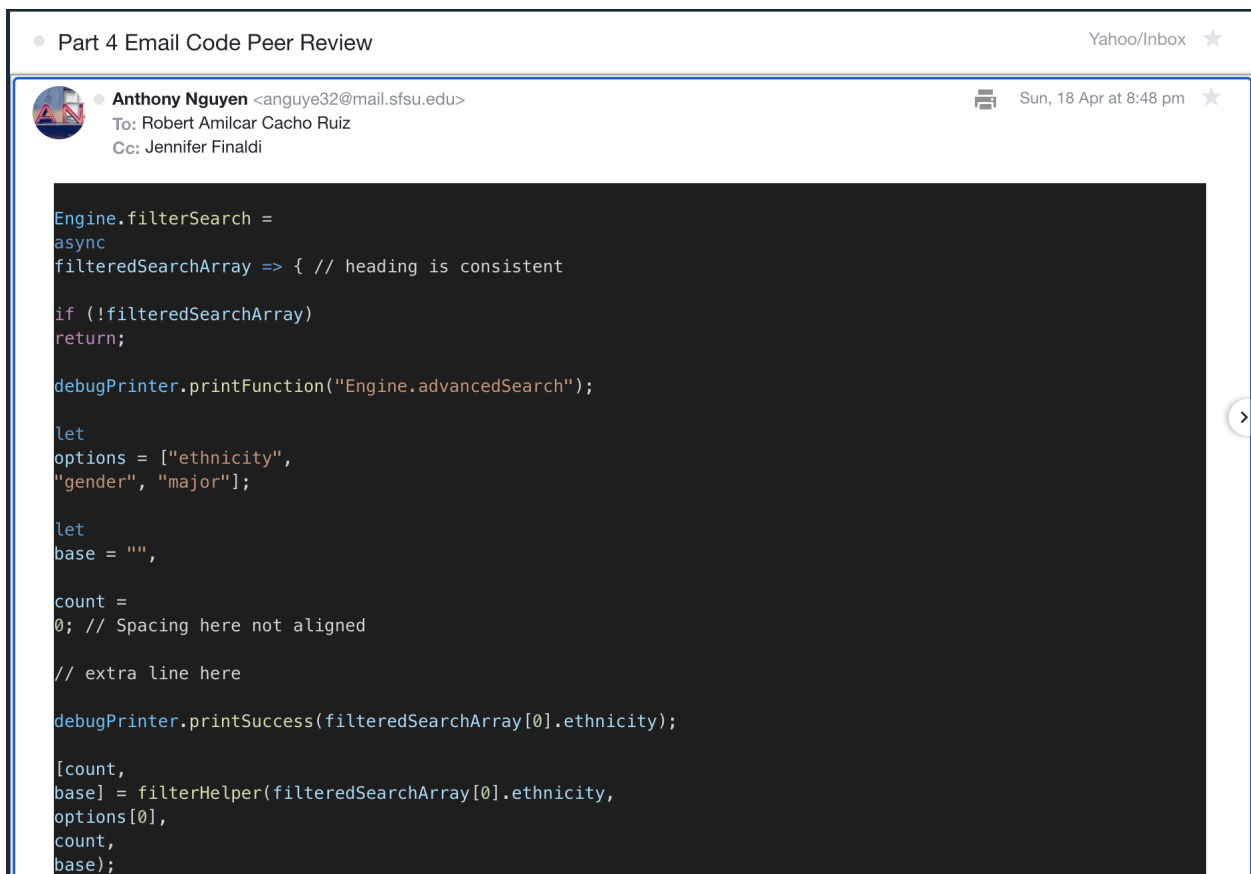
Test #	Test Title	Test Description	Test Input	Expected Correct Output	Test Results
1	Name Search + Major	Test for name field	"Robert", "Computer Science" Selected	Display all "Robert" users or names containing "Robert" from Database.	FAIL
2	Filter	Test for advanced filter	"asdf", Male Selected	Display all "asdf" users in the database that are male	FAIL
3	Invalid Characters	Test to see error handling for invalid characters	"!@#"	Return to home page	PASS

4) Code Review

A.

The style coding practices we used consisted of several things. First we declared all the variables in the beginning of the class. The variable declarations are separated by spaces except the parenthesis, quotation, and semicolons. Every function implementation has the parenthesis on the same line, as well as the if statements. After every comma, is a new line. Tabs are used to indent further between a function and objects.

B.



The screenshot shows an email interface with the title "Part 4 Email Code Peer Review" and the sender "Anthony Nguyen <anguye32@mail.sfsu.edu>". The email is dated "Sun, 18 Apr at 8:48 pm". The email body contains a code review of JavaScript code. The code is displayed in a dark-themed editor with syntax highlighting. The code includes variable declarations, a function call, and a conditional statement. The code is as follows:

```
Engine.filterSearch =
async
filteredSearchArray => { // heading is consistent

if (!filteredSearchArray)
return;

debugPrinter.printFunction("Engine.advancedSearch");

let
options = ["ethnicity",
"gender", "major"];

let
base = "",

count =
0; // Spacing here not aligned

// extra line here

debugPrinter.printSuccess(filteredSearchArray[0].ethnicity);

[count,
base] = filterHelper(filteredSearchArray[0].ethnicity,
options[0],
count,
base);
```

5) Self-check on best practices for security

- Database of registered users' information
 - The database belongs to a cloud server, and in order to access it, you require a ppk private key
- Database of sessions
 - Similar to the database for users, the database that holds a user's current session to our website is secured behind requiring the ppk private key
- Validating incoming data towards the backend
 - Search bar, we validate data to make sure no sql queries are escaped by certain characters which would allow for malicious code injections.
- Root access to cloud server
 - The team members are using an alternative account with limited permissions to access the server to keep root access disabled
- Authentication
 - From the registered users created and stored in our database, users must authenticate that they are a registered user when signing in

Password Encryption in DB:

5	coolbeans2	Not Robert Cacho	coolbeans2@gmail.com	\$2b\$15\$0s2poKtA/dez0a.vx/Z/OupuIaRU8aue...
6	coolbeans4	Cooler McGee	coolbeans4@gmail.com	\$2b\$15\$6OLDaj1dOljyC4lXkkFQy.Ux/OMRvw1t...
7	coolbeans5	Richard Hendricks	coolbeans5@gmail.com	\$2b\$15\$fMKJU/iE7PIthWVGextBbOzd9Z9VefaP...
8	coolbeans6	Barack Obama	coolbeans6@gmail.com	\$2b\$15\$Mta87XpuWjjjER0Q4QTzb.MjvGHCe/bP...

To validate our data we are using front-end javascript along with backend javascript. We are using two forms since only using the front end validation wouldn't be enough, since programs like curl or postman can easily send data to our backend without any interaction with our frontend. In our backend we make sure the limit length is 40 characters as long with it all being alpha-numeric. This sort of validation is implemented in our search bar, our registration and login also has validation.

6) Self-check: Adherence to original Non-functional specs

1. Application shall be developed, tested and deployed using tools and servers approved by Class CTO and as agreed in M0 (some may be provided in the class, some may be chosen by the student team but all tools and servers have to be approved by class CTO).

ON TRACK

2. Application shall be optimized for standard desktop/laptop browsers e.g. must render correctly on the two latest versions of two major browsers

ON TRACK, as of right now this is true

3. Selected application functions must render well on mobile devices

ISSUE, there is currently no plans to port this application to mobile devices at this time.

4. Data shall be stored in the team's chosen database technology on the team's deployment server.

DONE

5. No more than 100 concurrent users shall be accessing the application at any time

DONE

6. Privacy of users shall be protected, and all privacy policies will be appropriately communicated to the users.

ON TRACK

7. The language used shall be English.

DONE

8. Application shall be very easy to use and intuitive.

ON TRACK

9. Google maps and analytics shall be added

ISSUE, not sure the team has time to implement this, nor are we clear on why this application would require maps.

10. No e-mail clients shall be allowed. You shall use webmail.

ISSUE, while we aren't using e-mail clients, our webmail aka in site messaging feature is a P2 that may not get completed.

11. Pay functionality, if any (e.g. paying for goods and services) shall not be implemented nor simulated in UI.

DONE, we are not implementing any payment features at this time

12. Site security: basic best practices shall be applied (as covered in the class)

ON TRACK

13. Modern SE processes and practices shall be used as specified in the class, including collaborative and continuous SW development

ON TRACK

14. The website shall prominently display the following exact text on all pages "SFSU Software Engineering Project CSC 648-848, Spring 2021. For Demonstration Only" at the top of the WWW page. (Important so not to confuse this with a real application).

ON TRACK