SW Engineering CSC648/848 Fall 2020

Application: SFSU Buy and Sell Website

Name: SFSU CONNECT

Team: 1

Robert Clarkson - team lead - rclarkso@mail.sfsu.edu

Kevin Fung - gitmaster

Cody Camp - backend lead

Taylor Luke - frontend lead

Gerardo Ochoa - developer

Hong Li - developer

Milestone 4

October 20, 2020

September 23, 2020	First Revision Done
October 13, 2020	Added to first revision
October 20, 2020	Made changes based on CEO feedback
December 8th, 2020	M4 changes done

Content

1. Product summary

Product name: SFSU Connect

- Any user shall be able to browse through listings
- Any user shall be able to search for listings based on categories and text
- Any user shall be able to view listings details
- Any user shall be able to filter listings by price
- Unregistered users shall be able to create an account
- Registered users shall be able to login
- Registered users shall be able to send a message to listing owner
- Registered users shall be able to create their own listings
- Registered users shall be able to see their own listings
- Registered users shall be able to see messages sent to them
- Passwords shall be encrypted
- Registered users shall be able to upload images when creating a listing

Product site: http://ec2-54-151-54-31.us-west-1.compute.amazonaws.com

2. Usability test plan

Major Function: Search

Test Objective:

For the usability test plan we will be looking at our search function. The search function is being tested because it is a key component in the website. It allows the users to quickly find what they are looking for without having to go through every item listed. By giving a keyword into the search bar and/or by choosing a category and pressing the search button, it will help organize all the items from the database and give back only listings that are related.

Test Background and Setup:

For the system setup we have MySQL for our database. This is where we store all the data that we get from the website. This includes user's account information and the items that are being listed. For the server of the website we are using Amazon Web Service to keep the website running. People with permission will be able to visit the website that is connected to our database. The coding of this project is done using Visual Studio Code and Node.js

The starting point of this function would be at our home page. Since we don't require the user to be logged in to search, they can get started right away. The search bar is located on the top of the page, the user can input a keyword relating to a listing they are looking for. On the left of the search there is a category selection that allows the user to choose what type of listing, this way it will help narrow down the search results.

The intended users are students and staff of San Francisco State University. In order to make listings or contact buyers, users need to make an account which requires an sfsu email from the school. The purpose of this website is to allow people to search

for things that are being offered nearby. This could include things such as textbooks, clothing, kitchenware, and tutoring service.

The link below will bring the user to the homepage of our website. For this test plan we are testing the search function only. We will be measuring the user's satisfaction by using Likert test.

URL of the site: http://ec2-54-151-54-31.us-west-1.compute.amazonaws.com/

Usability Task Description:

The instructions for users would be :

1. Click on the URL to go to the homepage.
2. Use categories to find clothes.
3. Input keyword to find shirt.
4. Click on a listing to view.

To measure effectiveness we want to see if the users are able to find what they are looking for. We would ask users to find a very specific item such as crayola crayons and the seller. Then we can mark down the information of how many users were able to complete the task, and the ones that fail what went wrong along with everyone's feedback.

To measure efficiency we will look at how much clicks and time it took for the user to find the listings. We will ask users to find a very specific item such as dark grey shorts and the seller. We will measure each user's clicks, and the time it took to find the information.

For the Lickert Scale questions I would put the following statements and ask for the user to check one of the answers ranging from strongly agree to strongly disagree. The statements I would put are:

1. The search function was easy to understand and use.
2. The search function returned results related to my topic/category.
3. The overall use of the search function was satisfying.

3. QA test plan:

Our QA test plan is to help users navigate through test scenarios. This is to ensure we get the correct output intended and to show the tester how to use the sight. There are simple steps given in a table that need to be followed. All tests are labeled pass or fail based on the output received.

Test objectives:

The test objectives include making sure the search function is responsive to input. Once we see it responsive to input we make sure that input is correct. We also want to make sure that nothing is misplaced, so no out of sorts search results should be showing up. If there is nothing in that category we also check to make sure that it shows up that way.

What is being tested:

Testing includes, searching for products not using the whole product name, making sure all options show up for a category, test for a specific product within a category, and search based on one key word in all categories.

HW and SW setup (including URL):

Test algorithm SW implementation for search, on http://localhost:3000/ as well as http://ec2-54-151-54-31.us-west-1.compute.amazonaws.com/ using Chrome, Safari and Firefox.

Feature to be tested:

Search function

QA Test plan:

Using Chrome:

Test #	Title	Description	Test Input	Expected output	PASS/FAIL
1	Alarm clock	Test for objects not including all keywords.	Type “alarm clock” into the search field and click the search icon.	Get one result of “Projecting alarm clock”	PASS
2	Tutoring	Test for all available tutoring to show up when using the search field.	Click on drop-down, scroll and click on “tutoring” then click on the search icon.	3 results of available tutoring show up, “CSC 415, 510, and 413.”	PASS
3	Clothing	Test for a specific item of clothing typed into the search bar.	Type “Dark grey shorts” into the search field and click the search icon.	Get one result of “Dark grey shorts”	PASS
4	Color search	Test for the term, “red”	Select “all” from the	Get three results, one	PASS

		in the category “all.”	dropdown, type “red” into the search bar. Select the search icon.	has the word “red” in the product title, two have “red” in the project description.	

Using Safari: GUI was less response unresponsive. All test PASS

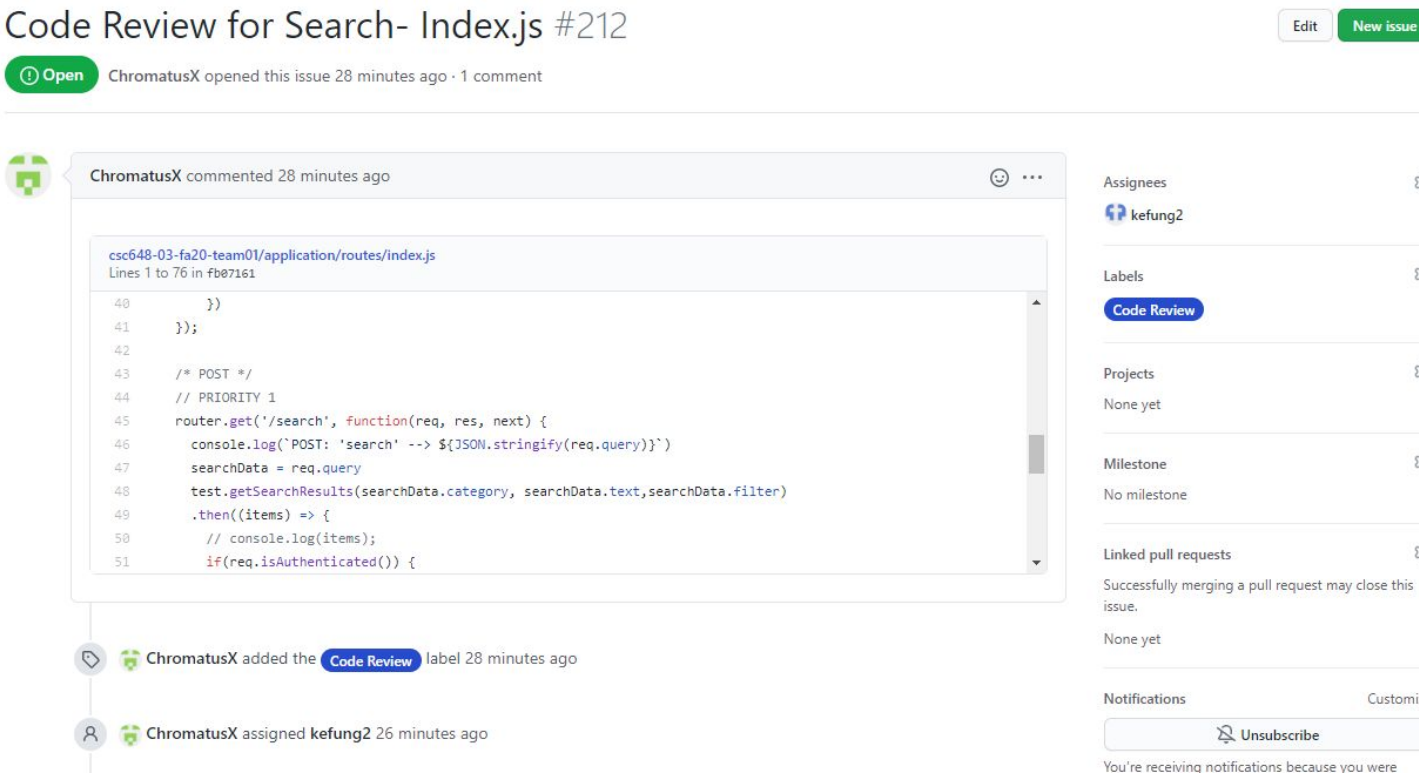
Using Firefox: GUI was as responsive as Chrome. All test PASS

4. Code Review

Function: Search

Full Detail Link in Github:

https://github.com/CSC-648-SFSU/csc648-03-fa20-team01/issues/212



kefung2 commented 2 minutes ago

csc648-03-fa20-team01/application/routes/index.js
Lines 1 to 92 in aad07b4

```
46    /**
47     * REVIEW:
48     * It would be nice to have some comment
49     * to briefly describe the function
50     *
51     * The search should be display on a new page instead of the landing
52     * page
53     *
54     */
55    router.get('/search', function(req, res, next) {
56      console.log(`POST: 'search' --> ${JSON.stringify(req.query)}`)
57      searchData = req.query
```

Write Preview

Leave a comment

Attach files by dragging & dropping, selecting or pasting them.

You're receiving notific
assigned.

2 participants

Lock conversation

Pin issue

Transfer issue

kefung2 commented now


```
var express = require('express');
var router = express.Router();
var db = require('./../db/index');
var test = require('../db/helpers/test');

/* GET */
// router.get('/', function(req, res, next) {
//     res.render('index', { title: 'Home' });
// });

router.get('/', function(req, res, next) {
  Promise.all([
    test.getSearchResults('All', ""),
    db.numItems_N_days()
  ]).then(([items, numItems]) => {
      // console.log(items)
      if (req.isAuthenticated()) {
        res.render('landing', {
          title: 'Home',
          numItems: numItems[0].number,
          items: items,
          category: 'All',
          user: req.user
        });
      } else {
        res.render('landing', {
          title: 'Home',
          numItems: numItems[0].number,
          items: items,
          category: 'All'
        });
      }
    })
    .catch((err) => {
      console.log(err)
      res.render('landing', {
        title: 'Home',
        category: 'All'
      });
    })
});

/* POST */
// PRIORITY 1
```

```
/* POST */
// PRIORITY 1

/**
 * REVIEW:
 * It would be nice to have some comment
 * to briefly describe the function
 *
 * The search should be display on a new page instead of the landing
 * page
 *
 */
router.get('/search', function(req, res, next) {
  console.log(`POST: 'search' --> ${JSON.stringify(req.query)}`)
  searchData = req.query
  test.getSearchResults(searchData.category, searchData.text,searchData.filter)
  .then((items) => {
    // console.log(items);
    if(req.isAuthenticated()) {
      res.render('landing', {
        title: 'Home',
        items: items,
        user: req.user,
        category: searchData.category,
        search: searchData.text
      });
    } else {
      res.render('landing', {
        title: 'Home',
        items: items,
        data:searchData.text,
        category: searchData.category,
        search: searchData.text
      });
    }

  })
});

/**
 * REVIEW:
 * What is this for? if this is not being use let remove it
 *
 *
 */
router.get('/inboxv2', function(req, res, next) {
  res.render('inboxv2', { title: 'About Us' });
});

module.exports = router;
```

5. Self-check on best practices for security

We are protecting

1. User passwords
2. User messages

Major threats to each asset

1. User passwords
    a. SQL injection to obtain the database list of users
    b. Brute force attacks on login screen
2. User messages
    a. SQL injection to obtain the database list of messages

Code injection

1. We make sure that every SQL query is parameterized. The use of jQueries within SQL is strictly prohibited.
2. All fields that allow user input are limited to a maximum of 40 characters.

Login practices

1. All users must use an sfsu email. Therefore all user emails must end with “sfsu.edu”.
2. Login failure feedback is indicated as “user or password incorrect”. Developers may not add more specificity.
3. All fields that allow user input are limited to a maximum of 40 characters.

Storage

1. All passwords are encrypted before being inserted into the database using bcrypt utilizing a special unique encryption keyword.

6. Self-check: Adherence to original non-functional specs

1. (ON TRACK) 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).
2. (DONE) Application shall be optimized for standard desktop/laptop browsers e.g. must render correctly on the two latest versions of two major browsers
3. (ON TRACK) All or selected application functions must render well on mobile devices
4. (DONE) Data shall be stored in the database on the team's deployment server.
5. (ON TRACK) No more than 50 concurrent users shall be accessing the application at any time
6. (DONE) Privacy of users shall be protected and all privacy policies will be appropriately communicated to the users.
7. (DONE) The language used shall be English (no localization needed)
8. (DONE) Application shall be very easy to use and intuitive
9. (DONE) Application should follow established architecture patterns
10. (DONE) Application code and its repository shall be easy to inspect and maintain
11. (ON TRACK) Google analytics shall be used
12. (DONE) No e-mail clients shall be allowed. Interested users can only message to sellers via in-site messaging. One round of messaging (from user to seller) is enough for this application
13. (DONE) Pay functionality, if any (e.g. paying for goods and services) shall not be implemented nor simulated in UI.
14. (DONE) Site security: basic best practices shall be applied (as covered in the class) for main data items
15. (DONE) Media formats shall be standard as used in the market today

16. (DONE) Modern SE processes and practices shall be used as specified in the class, including collaborative and continuous SW development
17. (ON TRACK) The application UI (WWW and mobile) shall prominently display the following exact text on all pages *"SFSU Software Engineering Project CSC 648-848, Fall 2020. For Demonstration Only"* at the top of the WWW page. (Important so as to not confuse this with a real application).