SW Engineering CSC648/848
Fall 2019

Aardvark

Section 01

Team 01

Ida Hui (Team Lead) - ihui@mail.sfsu.edu
Alan Nguyen (Front-End Lead)
Russell Wong (Back-End Lead)
Ryan Shu (GitHub Master)
Sunminder Sandhu (Back-End Developer)
Jon Fontejon (Back-End Developer)
Daisy Sanchez (Back-End Developer)

Milestone 4

December 11, 2019

1) Product summary

We are offering our app (Aardvark) as a shopping solution to SFSU students. Our app provides an easy, smooth experience in getting students the things they need in order to have a successful school year. On this platform, students will be able to buy and sell items that fall under the categories of homegoods, supplies, and textbooks. Below are features students can expect during their experience.

1. Home page displays once the user types in our URL in browser
2. Search field allows students to find item(s) they need
3. Filter results by price for an optimal experience
4. Result page displays items smoothly
5. Access a dashboard which displays inbox and inventory (togglable)
6. Students can create post for items they wish to sell
7. Request to buy items they're interested in
8. Sell and manage items no longer need

What really differentiates our product from our competitors is our meetup feature. Student's go straight to the seller to complete their purchase in person. Our app is clear and simple to use with a direct goal of getting items students need, fast.

Website: ec2-3-136-156-1.us-east-2.compute.amazonaws.com

2) Usability Test Plan

Test objectives:

- Determine the user friendliness when searching or displaying items that are being sold on the website.
    - This feature is important because it's the main focus of this site - buying items provided by students.

Test background and setup

System setup:

- Test on two browsers (Chrome and Firefox).

Starting Point:

- ec2-3-136-156-1.us-east-2.compute.amazonaws.com

Who are the intended users:

- SFSU students

URL of the system to be tested and what is to be measured:

- 3.136.156.1

Usability Task description:

- Open browser
- Go to link
- Perform a category search
- Perform keyword search
- Perform

Questionnaire and Assessment

Measure Effectiveness

- User was able to perform category search
- User was able to perform specific word search
- User did not need assistance when performing task

Measure Efficiency

- User was able to perform categorical search in projected time (or less)
- User was able to perform specific word search in projected time (or less)
- Visuals did not cause any confusion

Lickert subjective test:

I was able to search for the item I wanted:

Strongly Disagree	Disagree	Undecided	Agree	Strongly Agree

I was quickly able to find what I need:

Strongly Disagree	Disagree	Undecided	Agree	Strongly Agree

I knew how to display categories without much difficulties:

Strongly Disagree	Disagree	Undecided	Agree	Strongly Agree

3) QA test plan - max 2 pages

Test objectives (what is being tested):

- The search feature producing desired results.

HW and SW setup (including URL):

- Test on two browsers (Chrome and Firefox).
    - Start at homepage, visit ec2-3-136-156-1.us-east-2.compute.amazonaws.com

Feature to be tested

- The search feature: searching by category, search word, and both will display correct items.

QA Test plan:

Test (Chrome):

#	Title	Description	Input	Expected Output	Results Pass/Fail
1	Launch app	Go to Homepage by opening a browser and entering the URL	Ec2-3-136-156-1.us-east-2.compute.amazonaws.com in browser	Homepage	Pass
2	Display Items in Categories	Display items filtered by category using the drop down menu	Category: texts	5 text books	Pass
3	Search	Perform search by using specific keyword	Keyword: “wooden”	Chair Pencil	Pass
4	Specified Search	Perform specific search using categories AND keywords	Category: Homegoods Keyword: “wooden”	Chair	Pass

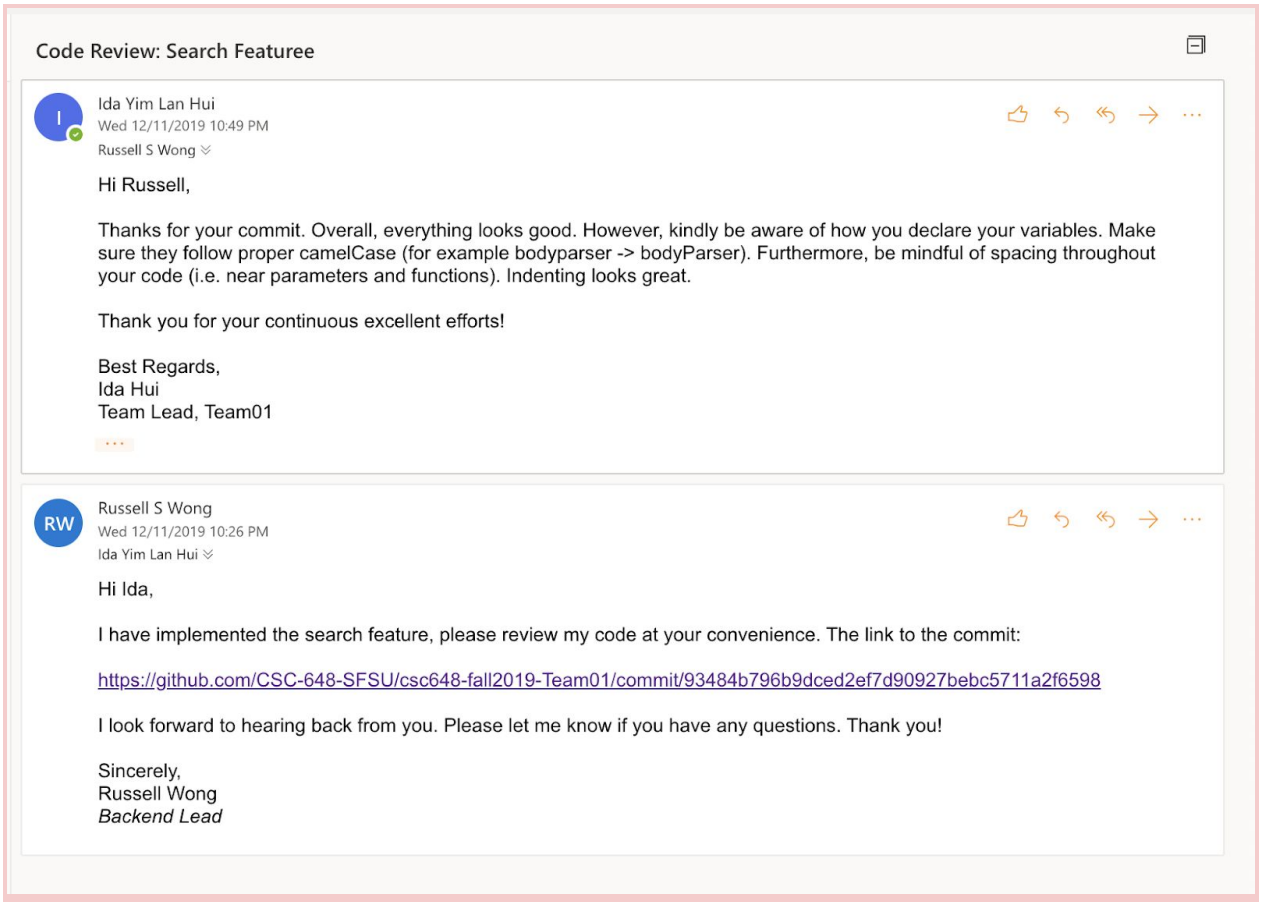
Test (Firefox):

#	Title	Description	Input	Expected Output	Results Pass/Fail
1	Launch app	Go to Homepage by opening a browser and entering the URL	Ec2-3-136-156-1.us-east-2.compute.amazonaws.com in browser	Homepage	pass
2	Display Items in Categories	Display items filtered by category using the drop down menu	Category: texts	5 text books	pass
3	Search	Perform search by using specific keyword	Keyword: “wooden”	Chair Pencil	pass
4	Specified Search	Perform specific search using categories AND keywords	Category: Homegoods Keyword: “wooden”	Chair	pass

4) Code Review:

1. By this time you should have chosen a coding style. In the report say what coding style you chose.
    - K&R style (Stroustrup)
        - Space between parameters
        - No space between the function name and parentheses between the parentheses and the parameter
        - Indentation on nested values
        - Empty line between logical blocks
        - Space around operators
        - Mandatory semicolon
    - Variables
        - camelCase
        - Meaningful (no x, y, z)
    - Comments shall be placed on top of correlating code
    - The module size should be uniform.
    - The name of the function must describe the purpose of the function clearly and briefly.

Message:



Code Review: Search Featuree

Ida Yim Lan Hui
Wed 12/11/2019 10:49 PM
Russell S Wong

Hi Russell,

Thanks for your commit. Overall, everything looks good. However, kindly be aware of how you declare your variables. Make sure they follow proper camelCase (for example bodyparser -> bodyParser). Furthermore, be mindful of spacing throughout your code (i.e. near parameters and functions). Indenting looks great.

Thank you for your continuous excellent efforts!

Best Regards,
Ida Hui
Team Lead, Team01

Russell S Wong
Wed 12/11/2019 10:26 PM
Ida Yim Lan Hui

Hi Ida,

I have implemented the search feature, please review my code at your convenience. The link to the commit:

https://github.com/CSC-648-SFSU/csc648-fall2019-Team01/commit/93484b796b9dced2ef7d90927bebc5711a2f6598

I look forward to hearing back from you. Please let me know if you have any questions. Thank you!

Sincerely,
Russell Wong
Backend Lead

Code before

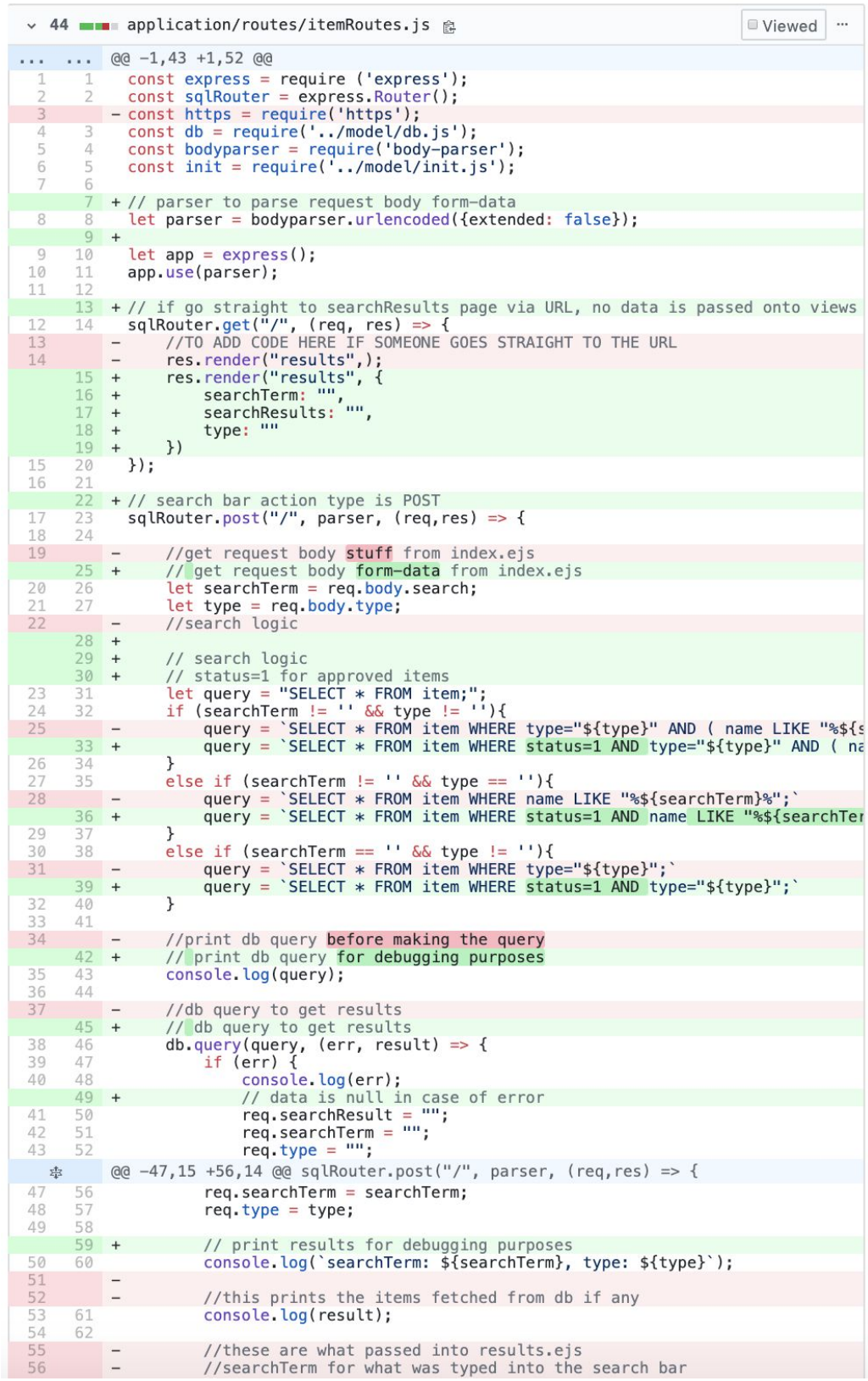
63 application/routes/itemRoutes.js

```
@@ -2,22 +2,67 @@ const express = require ('express');
  const sqlRouter = express.Router();
  const https = require('https');
  const db = require('../model/db.js');
+ const bodyparser = require('body-parser');
  const init = require('../model/init.js');

+ let parser = bodyparser.urlencoded({extended: false});
+ let app = express();
+ app.use(parser);
+
  sqlRouter.get("/", (req, res) => {
+     //TO ADD CODE HERE IF SOMEONE GOES STRAIGHT TO THE URL
      res.render("results");
  });

- sqlRouter.post("/", (req,res) => {
- /*     db.query("SELECT * FROM item WHERE status = 1")
-     .then(([items, _]) => {
+ sqlRouter.post("/", parser, (req,res) => {
+
+     //get request body stuff from index.ejs
+     let searchTerm = req.body.search;
+     let type = req.body.type;
+
+     //search logic
+     let query = "SELECT * FROM item;";
+     if (searchTerm != '' && type != ''){
+         query = `SELECT * FROM item WHERE type="${type}" AND ( name LIKE "%${se
+     }
+     else if (searchTerm != '' && type == ''){
+         query = `SELECT * FROM item WHERE name LIKE "%${searchTerm}%";`
+     }
+     else if (searchTerm == '' && type != ''){
+         query = `SELECT * FROM item WHERE type="${type}";`
+     }
+
+     //print db query before making the query
+     console.log(query);
+
+     //db query to get results
+     db.query(query, (err, result) => {
+         if (err) {
+             console.log(err);
+             req.searchResult = "";
+             req.searchTerm = "";
+             req.type = "";
+         }
+
+         req.searchResult = result;
+         req.searchTerm = searchTerm;
+         req.type = type;
+
+         console.log(`searchTerm: ${searchTerm}, type: ${type}`);
+
+         //this prints the items fetched from db if any
+         console.log(result);
+
+         //these are what passed into results.ejs
+         //searchTerm for what was typed into the search bar
+         //type for the type selected, null if All Types
+         //searchResults is the array of items.
          res.render("results", {
```

Code After



44 application/routes/itemRoutes.js

Viewed

```
@@ -1,43 +1,52 @@
 const express = require ('express');
 const sqlRouter = express.Router();
-const https = require('https');
 const db = require('../model/db.js');
 const bodyparser = require('body-parser');
 const init = require('../model/init.js');

+// parser to parse request body form-data
 let parser = bodyparser.urlencoded({extended: false});
+
 let app = express();
 app.use(parser);

+// if go straight to searchResults page via URL, no data is passed onto views
 sqlRouter.get("/", (req, res) => {
-    //TO ADD CODE HERE IF SOMEONE GOES STRAIGHT TO THE URL
-    res.render("results",);
+    res.render("results", {
+        searchTerm: "",
+        searchResults: "",
+        type: ""
+    })
 });

+// search bar action type is POST
 sqlRouter.post("/", parser, (req,res) => {

-    //get request body stuff from index.ejs
+    // get request body form-data from index.ejs
     let searchTerm = req.body.search;
     let type = req.body.type;
-    //search logic
+
+    // search logic
+    // status=1 for approved items
     let query = "SELECT * FROM item;";
     if (searchTerm != '' && type != ''){
-        query = `SELECT * FROM item WHERE type="${type}" AND ( name LIKE "%${s
+        query = `SELECT * FROM item WHERE status=1 AND type="${type}" AND ( na
     }
     else if (searchTerm != '' && type == ''){
-        query = `SELECT * FROM item WHERE name LIKE "%${searchTerm}%";`
+        query = `SELECT * FROM item WHERE status=1 AND name LIKE "%${searchTer
     }
     else if (searchTerm == '' && type != ''){
-        query = `SELECT * FROM item WHERE type="${type}";`
+        query = `SELECT * FROM item WHERE status=1 AND type="${type}";`
     }

-    //print db query before making the query
+    // print db query for debugging purposes
     console.log(query);

-    //db query to get results
+    // db query to get results
     db.query(query, (err, result) => {
         if (err) {
             console.log(err);
+            // data is null in case of error
             req.searchResult = "";
             req.searchTerm = "";
             req.type = "";
@@ -47,15 +56,14 @@ sqlRouter.post("/", parser, (req,res) => {
         req.searchTerm = searchTerm;
         req.type = type;

+        // print results for debugging purposes
         console.log(`searchTerm: ${searchTerm}, type: ${type}`);

-        //this prints the items fetched from db if any
         console.log(result);

-        //these are what passed into results.ejs
-        //searchTerm for what was typed into the search bar
```

5) Self-check on best practices for security – ½ page

- List major assets you are protecting
    - Username
    - Password
    - Messages
    - Posts
    - Personal data
    - Images

- Say how you are protecting each asset
    - Password
        - We use the Bcrypt (v3.07) node module to hash our passwords, so even if our database is compromised, our users' passwords cannot be used to login.
    - SQL injection
        - For PW: Exploit security vulnerabilities when credentials are incorrect
    - Cross-scripting
        - For PW: Tries to find malicious attacks to prevent further attacks.
    - Images
        - We store our images as BLOBs, which provides a safer storage and a faster retrieval rate.
    - All Assets
        - We use the Passport module (with a LocalStrategy) for our authentication. This module provides a much more secure authentication flow, making it harder for others to gain user access.
    - Express-validator
        - Validates request so proper information are allowed to be presented if credentials are correct.

- Confirm that you encrypt PW in the DB. **Confirmed**
- Confirm Input data validation (list what is being validated and what code you used) – we request you validate search bar input for up to 40 alphanumeric characters. **Confirmed**

6) Self-check: Adherence to original Non-functional specs – performed by team leads

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).	Done
2. Application shall be optimized for standard desktop/laptop browsers e.g. must render correctly on the two latest versions of two major browsers	Done
3. Selected application functions must render well on mobile devices	In Progress
4. Data shall be stored in the team's chosen database technology on the team's deployment server.	Done
5. No more than 50 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.	In Progress
7. The language used shall be English.	Done
8. Application shall be very easy to use and intuitive.	In Progress
9. Google analytics shall be added	In Progress
10. No email clients shall be allowed	In Progress
11. Pay functionality, if any (e.g. paying for goods and services) shall not be implemented nor simulated in UI.	Done
12. Site security: basic best practices shall be applied (as covered in the class)	Done
13. Modern SE processes and practices shall be used as specified in the class, including collaborative and continuous SW development	In Progress
14. The website shall prominently display the following exact text on all pages "SFSU Software Engineering Project CSC 648-848, Fall 2019. For Demonstration Only" at the top of the WWW page. (Important so as to not confuse this with a real application).	On Track - Change (verbatim)