SW Engineering CSC648/848 Fall 2019
Gator Trader - SFSU Buy and Sell Website
Section 1
Team 3

Steve Rentschler - Team Lead (srentsch@mail.sfsu.edu)
Johnson Wong - Back-end Lead / GitHub Master
Sergei Katakhov - Front-end Lead
Ho Yin Mak (Tevis)
Tsun Ming Lee (Matthew)
Tim Chan
Osbaldo Martinez

Milestone 4
12/3/19

History Table

Date Submitted	12/11/2019
Date Revised	TBD

1) Product summary:

Our product name is Gator Trader. The following functions are what our team will deliver when our product is complete:

1. Unregistered users will able to register for an account
2. Unregistered users will able to browse home page
3. Unregistered users will able to search for a product by name and/or category
4. Unregistered users will able to browse a product search results page
5. Unregistered users will able to browse a product listing page
6. Unregistered users will able to view the form for listing a product
7. Unregistered users will able to sort by price
8. Unregistered users will able to search by class name and instructor's name for class materials
9. Registered users will be able to do 1-6
10. Registered users will be able to login and logout
11. Registered users will be able to purchase a product
12. Registered users will be able to list a product for sale
13. Registered users will be able to view active sales items on the dashboard page
14. Registered users will be able to remove their product listing(s)
15. Registered users will be able to contact the seller of a listing
16. Registered users will be able to see a map that confirms their safe pickup location.
17. The Administrator will be able to see listing request(s)
18. The Administrator will be able to approve and disapprove listing request(s)
19. The Administrator will be able to remove any active listing
20. The Administrator will be able to delete users

Our product is unique in that it is made for SFSU students to buy and sell items, they will be able to pick out a safe location where they can pick up the product they are purchasing and will be able to search for class specific materials that they might need for their class at SFSU.

Here is the URL for our product**:** https://sfsugatortrader.com

2) Usability test plan:

Test objectives:

The feature being tested in this usability test is the “Post an item” feature. Posting an item is an important part of the website since the users (SFSU students) need this to sell and display the item to other users.

Test background and setup:

1. System setup:
   A 2014 13 inch Macbook Pro with 8GB of RAM. Running browsers on Safari and Google Chrome.

2. Starting point:
   First, the user clicks on the URL which will lead to the Home Page, then Sign In, and find the Sell button.

3. Intended users:
   The intended users are SFSU students who have a basic knowledge on using computers.

4. URL: https://sfsugatortrader.com/sell

5. What is to be measured:
   This test is to measure how satisfied the user is about posting an item. The users will be asked to post an item, then will fill out a Likert questionnaire where they rate the questions asked.

Usability Task description:

1. Sign in
2. Click on Sell
3. Fill out Information
4. Click Submit
5. You should see your item on dashboard waiting to be approved

Lickert subjective test: Questionnaire

		Strongly Disagree	Disagree	Neutral	Agree	Strongly Agree
	Questions	1	2	3	4	5
1	I found the steps to post an item are straightforward.					
2	I found filling out the posting form simple to do.					
3	I found the website very user-friendly.					

Notes:	

3) QA test plan:

Test Objectives: Post an item and be able to see the item card of the posted item with the details provided before the item was posted to the database.

HW Setup: A laptop with the current Windows 10 update.

SW Setup: Go to https://www.sfsugatortrader.com/. Login in with the following credentials: SFSU email: w@mail.sfsu.edu and Password: password . Then, click on the sell button located in the navbar near the search bar and fill out the input fields. After submitting the inputs user will be redirected to their dashboard, click on the name of the created product listing and check that the information entered was preserved.

Feature to be Tested: Posting an Item

Test plan:

Test #	Test Title	Test Description	Test Input	Test Output	PASS/FAIL
1	Chrome Test	Test that making a sale post stores the information provided to be stored on the current Chrome build.	Name: “Nintendo Switch” Price: 220 Category: Electronics Is this a specific class material?: No Condition: Used-Good Quantity: 1 Delivery Method: Shipping Description: “A cheap Nintendo Switch.”	Item card with the input information that displays the selected images. And under the product name, it should say by w(testing user). There should be no pickup location. Delivery Method: Shipping Only	PASS

			Add Image Here: Select “PictureA.jpg” and “PictureB.jpg” images from the laptop.		
2	FireFox Test	Test that making a sale post stores the information provided to be stored on the current FireFox build.	Name: “Nintendo Switch(Game Included)” Price: 240 Category: Electronics Is this a specific class material?: No Condition: Used-Excellent Quantity: 1 Delivery Method: Shipping and Library(A) Description: “A cheap Nintendo Switch with the game Xenoblade Chronicles 2.” Add Image Here: Select “PictureA.jpg”, “PictureB.jpg” and	Item card with the input information that displays the selected images. And under the product name it should say by w(testing user). There should be pickup location. Delivery Method: Shipping and Pickup	PASS

			“PictureC.jpg” images from laptop.		

4) Code Review:

a) Coding Style:

The coding style that we chose for the back-end is functional programming with modular-based structure. Each function performs a certain job and is separated into individual JavaScript files. These files may contain multiple functions depending on its responsibility (for instance, files in the “controllers” folder). For the language itself, we used JavaScript ES6 coding style conventions. We also chose to use “Prettier” code formatter which is a Visual Studio Code extension that allowed us to make the code more readable and consistent on the overall project scale.

b) Code Review:

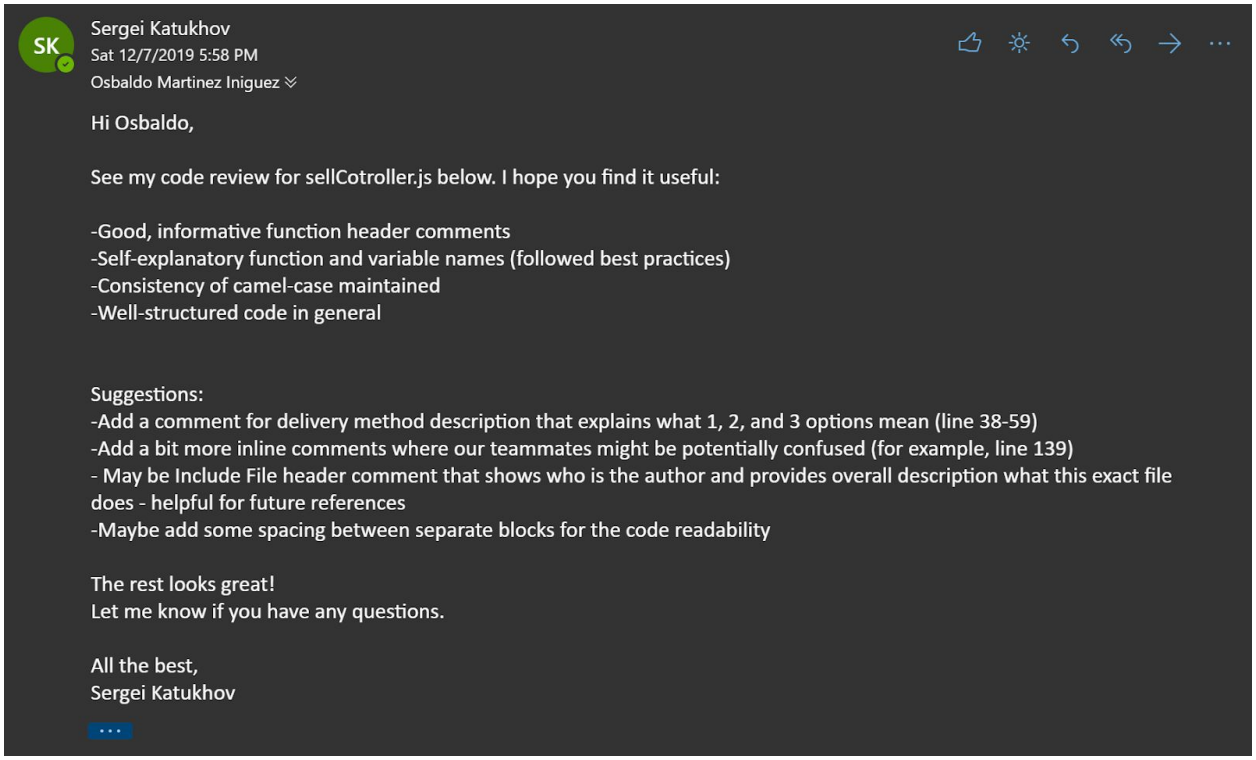
The chosen code is from sellController.js which contains logic for “Post an Item” feature used in usability and QA testing. The part of the code reviewed is shown below:

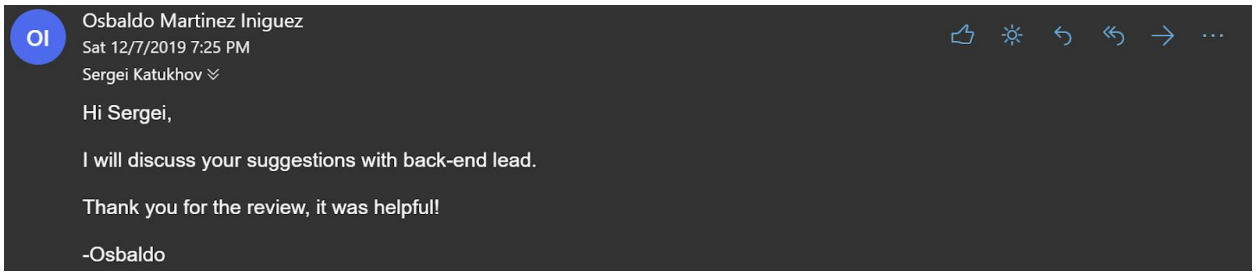
Code review for item post code


```
// Handle submitting sales item for sell on POST
exports.sell_post = (req, res, next) => {
    let productId = uuidv1();
    let { productName, price, category, classMaterialSection, condition, quantity, deliveryMethod, description } = req.body;
    let seller = req.user.sid;
    let salesItemImages = req.files;
    let sql = "";
    let placeholders = [];
    let tempDeliveryMethod = deliveryMethod;
    let salesItemPlaceholders = [];

    // Check if delivery method is shipping only, pickup only, or both
    if (Array.isArray(tempDeliveryMethod)) {
        // Shipping and pickup
        if (tempDeliveryMethod.includes("shipping")) {
            deliveryMethod = 3;
        }
        // Pickup only (multiple locations)
        else {
            deliveryMethod = 2;
        }
    }
    else {
        // Shipping only
        if (tempDeliveryMethod == "shipping") {
            deliveryMethod = 1;
        }
        // Pickup only (single location)
        else {
            deliveryMethod = 2;
        }
    }

    // Intepret and store newline
    description = description.replace(/\r\n|\r|\n/g, "<br>");

    // Check if class material section field is empty
    if (classMaterialSection != '') {

        // Create a new sales item
        sql += "INSERT INTO SalesItem (pid, seller, category, name, price, `condition`, quantity, description, deliveryMethod, classMaterialSection
) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, ?);";

        salesItemPlaceholders = [productId, seller, category, productName, price, condition, quantity, description, deliveryMethod, classMaterialSe
ction];
```

```
        // Create a new sales item
        sql += "INSERT INTO SalesItem (pid, seller, category, name, price, `condition`, quantity, description, deliveryMethod, classMaterialSection) VALUES (?, ?, ?, ?, ?, ?, ?, ?, ?, NULL);";

        salesItemPlaceholders = [productId, seller, category, productName, price, condition, quantity, description, deliveryMethod];
        placeholders.push(...salesItemPlaceholders);
    }

    // Store pickup location in DB
    // Pickup only (multiple locations) or shipping & pickup
    if (Array.isArray(tempDeliveryMethod)) {
        for (let i = 0; i < tempDeliveryMethod.length; i++) {
            // Make sure "shipping" is not included as a location
            if (tempDeliveryMethod[i] != "shipping") {

                // Create a new pickup location for a sales item
                sql += "INSERT INTO PickupLocation (product, location) VALUES (?, ?);";

                placeholders.push(productId);
                if (tempDeliveryMethod[i] == "library") {
                    placeholders.push(1);
                }
                else if (tempDeliveryMethod[i] == "student-center") {
                    placeholders.push(2);
                }
                else {
                    placeholders.push(3);
                }
            }
        }
    }
    // Single pickup location
    else {

        // Create a new pickup location for a sales item
        sql += "INSERT INTO PickupLocation (product, location) VALUES (?, ?);";

        placeholders.push(productId);
        if (tempDeliveryMethod == "library") {
            placeholders.push(1);
        }
        else if (tempDeliveryMethod == "student-center") {
            placeholders.push(2);
        }
        else {
            placeholders.push(3);
        }
    }

    // Save filename of uploaded sales item photo(s)
    for (let i = 0; i < salesItemImages.length; i++) {

        // Create a new sales item photos (filename) for a sales item
        sql += "INSERT INTO SalesItemPhoto (product, fileName) VALUES (?, ?);";
        placeholders.push(productId);
        placeholders.push(salesItemImages[i].filename);
    }

    db.query(sql, placeholders, (err, result) => {
        if (err) {
            req.flash('error', 'Error listing item');
            res.render('sell');
        }

        if ((typeof result !== 'undefined')) {
            req.flash('success', 'Successfully listed item for sale');
            res.redirect('/user/dashboard');
        }
        else {
            req.flash('error', 'Error listing item');
            res.redirect('/sell');
        }
    });
}
```

Comments on the code:





5) Self-check on best practices for security:

Major assets:

1. Registered user's credentials
2. Administrator's credentials
3. Revise sales item action route
4. End sales item action route
5. Relist sales item action route
6. Approve sales item action route
7. Disapprove sales item action route

Protecting each asset:

1. Registered user's credentials: Hashed password, and utilized placeholders in raw SQL queries to prevent SQL injection
2. Administrator's credentials: Hashed password, and utilized placeholders in raw SQL queries to prevent SQL injection
3. Revise sales item action route: Validate that the logged in user is also the seller of that item, else redirect to error page
4. End sales item action route: Validate that the logged in user is also the seller of that item, else redirect to error page
5. Relist sales item action route: Validate that the logged in user is also the seller of that item, else redirect to error page
6. Approve sales item action route: Validate that the logged in user has administrative privileges, else redirect to admin sign-in page
7. Disapprove sales item action route: Validate that the logged in user has administrative privileges, else redirect to admin sign-in page

***Confirmed all passwords are encrypted in database.**

Input data validation:

1. Search bar input field
    a. Max characters is 40
        - Client-side validation: Added maxlength="40" attribute to search bar input tag

2. Register page: SFSU email input field
    a. Max characters is 40
        - Client-side validation: Added maxlength="40" attribute to SFSU email input tag
        - Server-side validation: "username.length <= 40"
    b. Must end in @mail.sfsu.edu or @sfsu.edu
        - Server-side validation: Using regex, /^[-!#$%&'*+\/0-9=?A-Z^_a-z`{|}~](\.?[-!#$%&'*+\/0-9=?A-Z^_a-z`{|}~])*@(mail.sfsu.edu|sfsu.edu)$/

3. Register page: SFSU ID input field
    a. Max characters is 9
        - Client-side validation: Added maxlength="9" attribute to SFSU ID input tag
        - Server-side validation: "sid.length == 9"
    b. Contains only numbers
        - Server-side validation: Using regex, /^\d+$/.test(sid)
    c. Starts with the number 9
        - Server-side validation: "sid.startsWith("9")"

4. Register page: password input field
    a. Must be 8-20 characters
        - Client-side validation: Added minlength=“8” and maxlength="20” attributes to password input tag
        - Server-side validation: “(password.length >= 8) && (password.length <= 20)”

5. Register page: confirm password input field
    a. Must match password input field
        - Client-side validation: “$("#password").val() == $("#confirmPassword").val()”
        - Server-side validation: “(password == confirmPassword)”

6. Sell page: product name input field
    a. Max characters is 70
        - Client-side validation: Added maxlength=“70” attribute to product name input tag

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

1. **DONE:** 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. **DONE:** Selected application functions must render well on mobile devices
4. **DONE:** Data shall be stored in the team's chosen database technology on the team's deployment server.
5. **DONE:** 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.
8. **DONE:** Application shall be very easy to use and intuitive.
9. **DONE:** Google analytics shall be added
10. **DONE:** No email clients shall be allowed
11. **DONE:** Pay functionality, if any (e.g. paying for goods and services) shall not be implemented nor simulated in UI.
12. **DONE:** Site security: basic best practices shall be applied (as covered in the class)
13. **DONE:** Modern SE processes and practices shall be used as specified in the class, including collaborative and continuous SW development
14. **DONE:** 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).