SW Engineering CSC648-848 Spring 2025
ThriftAtSFSU

Milestone 4

Team 15

Hilary Lui (hlui@sfsu.edu)	Team Lead
Annison Van	Frontend Lead
Sid Padmanabhuni	Backend Lead & GitHub master
Joseph Alhambra	Team Member Frontend
Joseph Shur	Team Member Backend

Date Submitted	5/20/2025
Date Revised	

1) Product Summary: ThriftAtSFSU

Name of the product:
ThriftAtSFSU

What is ThriftAtSFSU?
ThriftAtSFSU is a campus-exclusive marketplace web app built by and for SFSU students and faculty. The platform allows users to buy, sell, and trade items within the university community, helping them save money, declutter, and find affordable, second-hand goods in a safe and trusted environment. With no shipping, no scams, and no strangers, ThriftAtSFSUis unique because it builds on-campus trust and accessibility while providing a simple, streamlined experience focused entirely on local, student-to-student transactions.

Final list of committed P1 (Priority 1) functions:

- Users of all levels of authority and privileges shall be able to access the app and see listings and reviews of associated listings.
- Unregistered users shall be allowed to create an account and become a registered user.
- Unregistered users shall be able to search based on keywords and filter by category.
- Registered users shall agree to the terms and conditions upon account creation.
- Registered users can create and delete listings and make reviews for other users.
- Registered users can add items to a wishlist (favorites)
- Registered users can send and receive messages with registered users.
- Admin shall be able to delete listings and users who are reported

URL: https://linux-vm-1.tailce85ff.ts.net/

2) Usability Test Plan – Function: Posting a New Listing

1. Test Objectives

This usability test evaluates how easily users can post a new item for sale on the ThriftAtSFSU platform. This function is central to the app's value and must be intuitive, efficient, and error-free. We want to measure whether users can complete the process without confusion, what barriers they face, and how we can improve their experience.

---

2. Test Background and Setup

System Setup:
The ThriftAtSFSU website is deployed on a live server accessible via IP address. Testers will access the system through a web browser. While the site is optimized for use on a laptop or desktop computer, it is also mobile-responsive and can be accessed on smartphones or tablets if preferred.

Starting Point:
Each tester will begin the test already logged into a designated test account and landed on the homepage. From there, they will be instructed to navigate to the "Post a Listing" page to complete the task.

Hardware Requirements:

- A device running macOS, Windows, or Linux (mobile devices are also supported)
- A working internet connection
- A modern web browser such as Chrome, Firefox, Safari, or Edge

Intended Users:
The intended testers are SFSU students and faculty, particularly those who are comfortable with typical online platforms such as Craigslist, eBay, or Facebook Marketplace. Testers are assumed to be non-technical users, simulating typical user behavior.

System URL:
https://linux-vm-1.tailce85ff.ts.net/

Test Environment:
Testing will take place remotely, with participants completing tasks from home. There will be no live monitoring, screen recording, or video conferencing. All instructions will be delivered via a Google Form, and participants will submit feedback through a follow-up survey.

No pre-training will be provided. Testers are expected to rely on their own intuition to complete the task, in order to better reflect realistic user experience conditions.

---

3. Usability Task Description

Instructions to the tester:

Thank you for helping us test the ThriftAtSFSU platform! Please follow the steps below as if you were genuinely trying to sell an item on campus. Your feedback will help us improve the platform's ease of use and functionality.

Task: Post an item for sale on ThriftAtSFSU.

1. From the homepage, locate and click on the "Post a New Item" option.

2. Fill in the form with the following details:
   - Title: "Gently used microwave"
   - Price: "25"
   - Category: "Electronics"
   - Description: "Used microwave, works great, pickup only near campus."
   - Upload a sample image of a microwave (any image file you have on your device).

3. Submit the listing.

4. Navigate to your dashboard and check whether the item appears under your posted listings

5. After completing the task, please fill out the brief follow-up questionnaire provided

---

4. Plan for Evaluation of Effectiveness

Effectiveness will be assessed based on the tester's ability to complete the core task without assistance. A task will be considered effective if the tester can:

- Successfully navigate to the post listing page
- Submit a valid listing with all required fields completed
- View the newly submitted post on their dashboard after submission

We will log and review whether the tester:

- Completed the task (Yes/No)
- Faced any errors (e.g., validation issues, image upload failure)
- Required external help or clarification

---

5. Plan for Evaluation of Efficiency

Efficiency will be evaluated by how quickly and smoothly testers are able to complete the task, without unnecessary confusion or navigation errors.

- Time to Completion:
  - The total time taken by the tester to complete the task — from landing on the homepage to successfully submitting the listing.
- Missteps or Backtracks:
  - The number of times the tester:
  - Navigates to the wrong page
  - Has to go back or re-edit form field.
- Perceived Task Clarity:
  - Whether the tester felt the task was easy to complete or encountered moments of confusion, as self-reported in the post-task survey.

Data Collection Method

Efficiency data will be captured through:

- A self-reported time estimate included in the follow-up questionnaire
- Observation-based notes from testers' feedback (e.g., mentions of going to the wrong page, redoing steps, or being unsure what to do next)

---

6. Plan for Evaluation of User Satisfaction (Likert Scale Questionnaire)

After completing the task, users will be asked to rate the following statements on a 5-point scale:

Q1: I found it easy to find how to post an item for sale using this website.
1 – Strongly Disagree | 2 – Disagree | 3 – Neutral | 4 – Agree | 5 – Strongly Agree

Q2: I found the listing form easy to complete
1 – Strongly Disagree | 2 – Disagree | 3 – Neutral | 4 – Agree | 5 – Strongly Agree

Q3: I would feel comfortable using this feature to sell something on campus.

1 – Strongly Disagree | 2 – Disagree | 3 – Neutral | 4 – Agree | 5 – Strongly Agree

3) QA Test Plan and QA Testing – Feature: Post a Listing

1. Test Objectives

This QA test aims to verify that the "Post a Listing" function works as expected across different user inputs and browser environments. The focus is on ensuring the listing form handles inputs correctly, uploads images, and displays listings in the user's dashboard without error.

---

2. Hardware and Software Setup (Including URL)

Hardware:

- A desktop, laptop, or smartphone running Windows, macOS, Linux, iOS, or Android
- A stable internet connection
- A functional device capable of uploading an image

Software:

- A modern web browser such as Google Chrome, Mozilla Firefox, Safari, or Edge
- Web app deployed at: https://linux-vm-1.tailce85ff.ts.net/

---

3. Feature to Be Tested

The focus of this test is the "Post a Listing" feature. Specific aspects being evaluated include:

- Form Accessibility: All input fields and buttons should be easily navigable and usable on both desktop and mobile devices
- Input Field Validations: Title, price, category, and description fields should be required and validated correctly
- Image Upload: Users must be able to upload a valid image file from their device without error
- Form Submission: On valid input, the form should submit smoothly
- Listing Display: Once submitted, the listing should appear correctly in the user's dashboard view

---

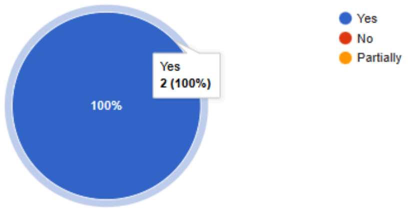
4. Results

Section 1: Task Instructions (Read-Only)

1. Were you able to complete the task successfully?

Copy chart

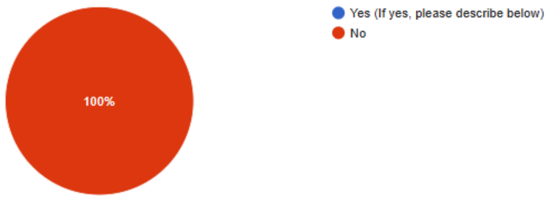
2 responses



2. Did you encounter any bugs or issues?

Copy chart

2 responses



3. Please describe any bugs or issues you experienced (if any):

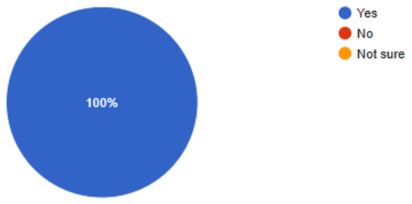
0 responses

No responses yet for this question.

4. Did your item appear in your dashboard after submission?

Copy chart

2 responses

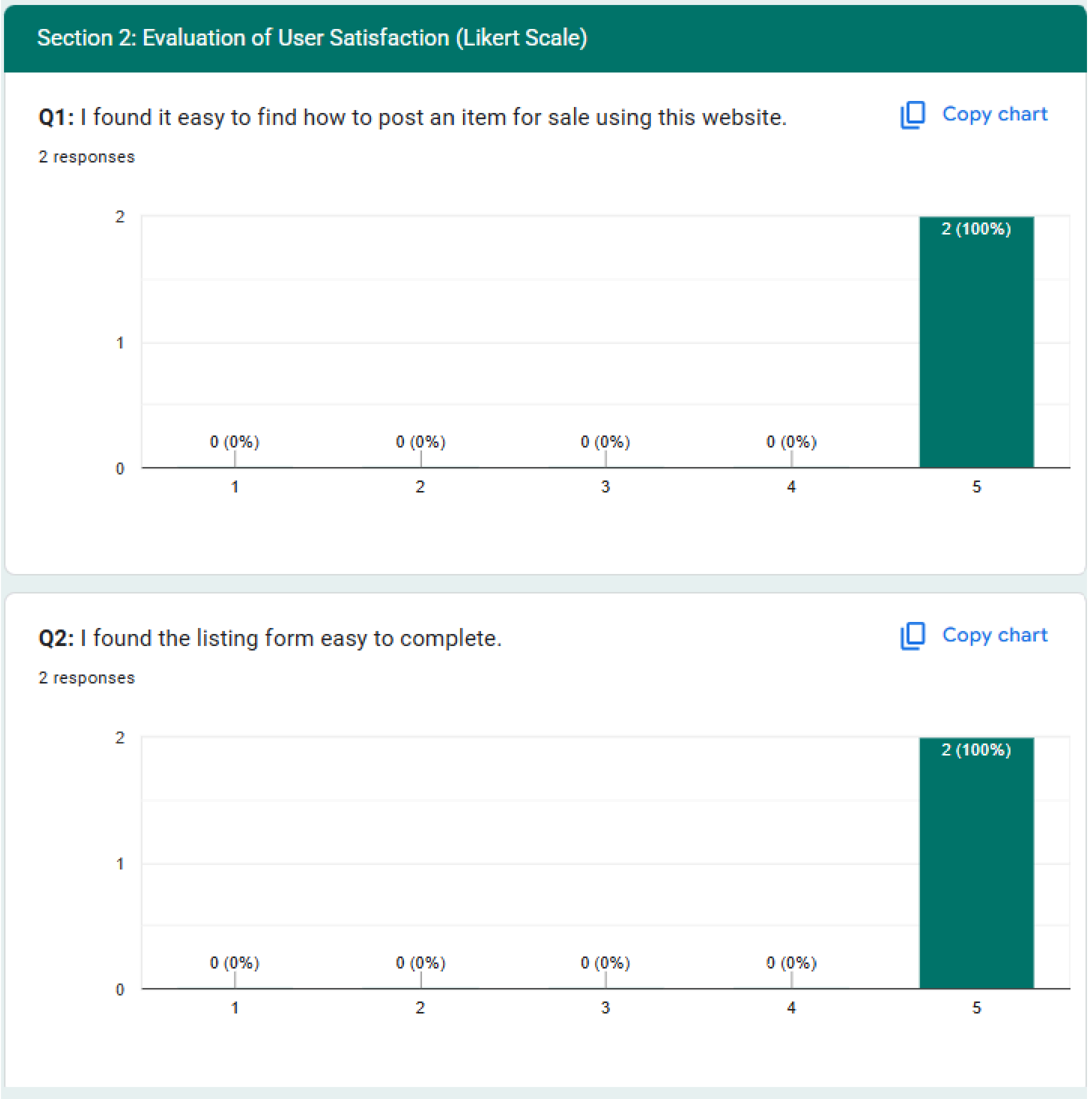


5. Any other feedback or suggestions?

2 responses

The sign-up page kept deleting either my password or image if I forgot either, leading to an increase in blood pressure. I submitted a listing before reading the instructions here, so I am now selling my motorcycle :(

Let users change their name, let users delete their account

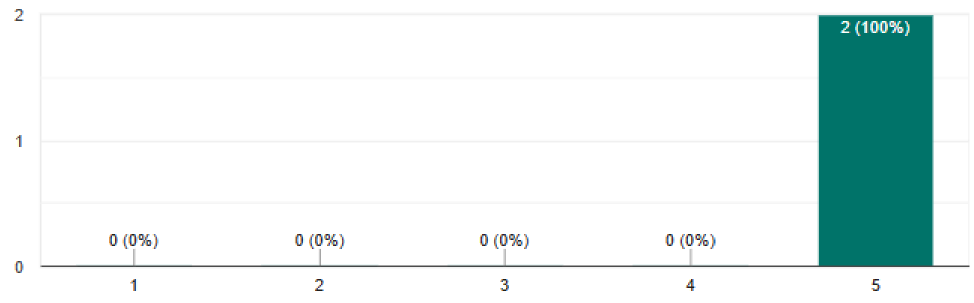


Section 2: Evaluation of User Satisfaction (Likert Scale)

Q1: I found it easy to find how to post an item for sale using this website.

Copy chart

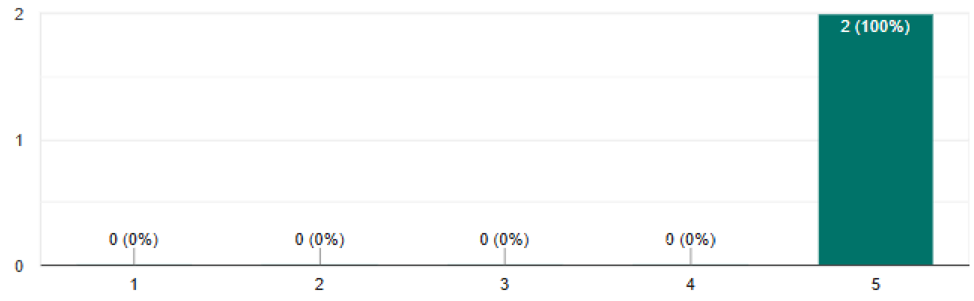
2 responses

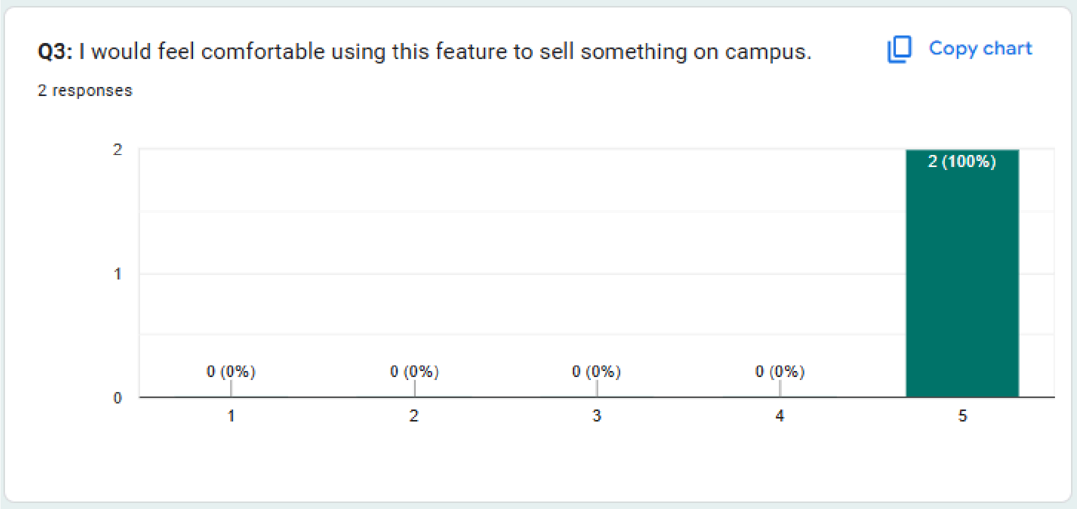


Q2: I found the listing form easy to complete.

Copy chart

2 responses

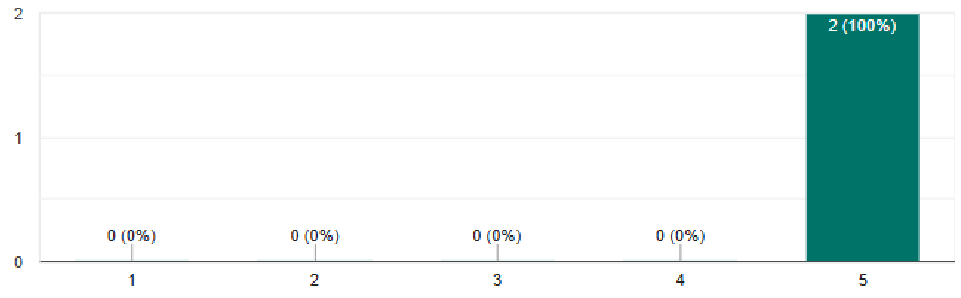




Q3: I would feel comfortable using this feature to sell something on campus.

Copy chart

2 responses



5. QA Test Plan Table

Test #	Test Scenario	Test Steps	Expected Result	Pass/Fail Criteria	Result (Chrome)	Result (Firefox)
1	Navigate to Post a Listing page	From homepage, click “Post a New Item”	User is taken to the listing form page	Page loads without error	PASS	PASS
2	Submit listing with all valid inputs	Fill in title, price, category, description, and upload image; click Submit	Listing is successfully posted and confirmation is shown	Listing is visible in dashboard	PASS	PASS
3	Validate required fields	Try submitting form with missing required fields (e.g., no title or price)	Error message is displayed, and submission is blocked	Error displayed, form not submitted	PASS	PASS
4	Upload image file	Select and upload a valid image (e.g., JPG or PNG)	Image is previewed or attached to the listing properly	Image upload succeeds	PASS	PASS
5	Check listing in dashboard	After submission, go to user dashboard	New listing appears with correct details	Listing is visible and accurate	PASS	PASS
6	Mobile usability	Perform entire task on a mobile browser (e.g., Chrome on Android/iOS)	All features function correctly; form and layout adjust for mobile	No UI or functional issues on mobile	PASS	PASS
7	Error handling for unsupported image	Upload a non-image file (e.g., .txt or .exe)	Form displays error and does not allow submission	Upload is blocked with error shown	PASS	PASS

8	localStorage handling (filter toggle retained)	Reload page after toggling filters	Filter state persists as saved in localStorage	State retained correctly	PASS	PASS

---

6. Cross-Browser Test Results

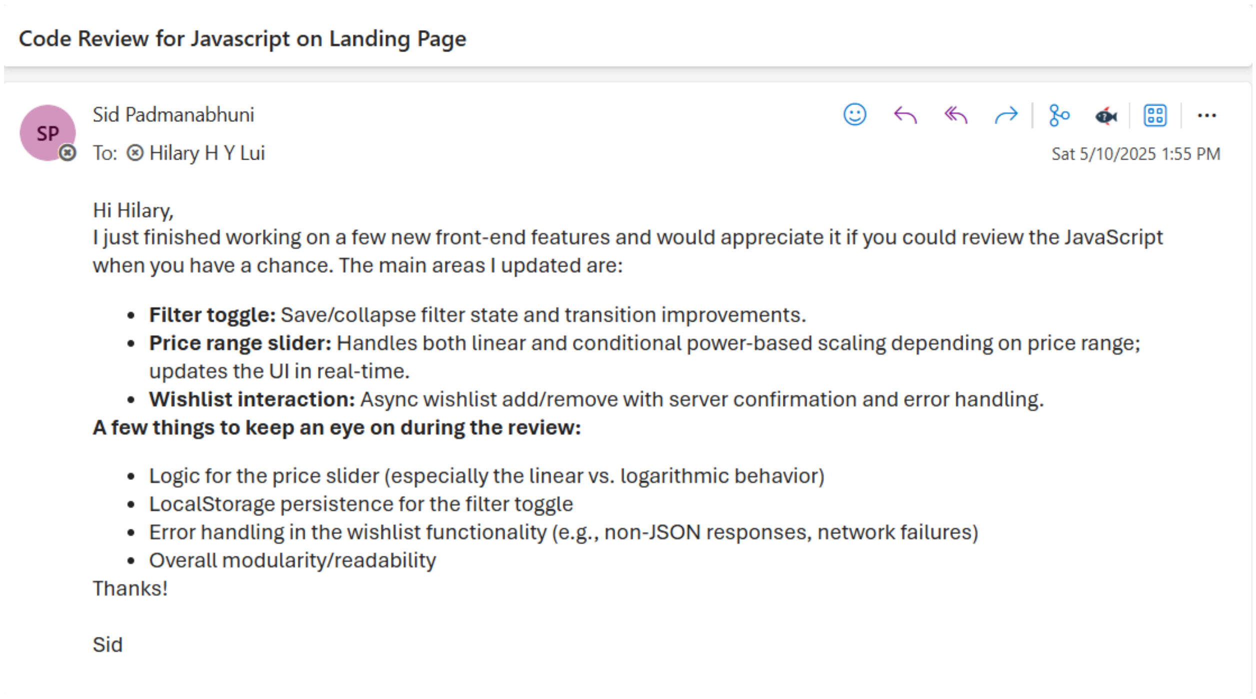
We ran the tests above on:

- **Google Chrome** – All test cases **PASSED**
- **Mozilla Firefox** – All test cases **PASSED**

This confirms that the "Post a Listing" feature behaves consistently across major modern browsers.

4) Peer Code Review

Emails:



Code Review for Javascript on Landing Page

SP Sid Padmanabhuni
To: Hilary H Y Lui
Sat 5/10/2025 1:55 PM

Hi Hilary,
I just finished working on a few new front-end features and would appreciate it if you could review the JavaScript when you have a chance. The main areas I updated are:

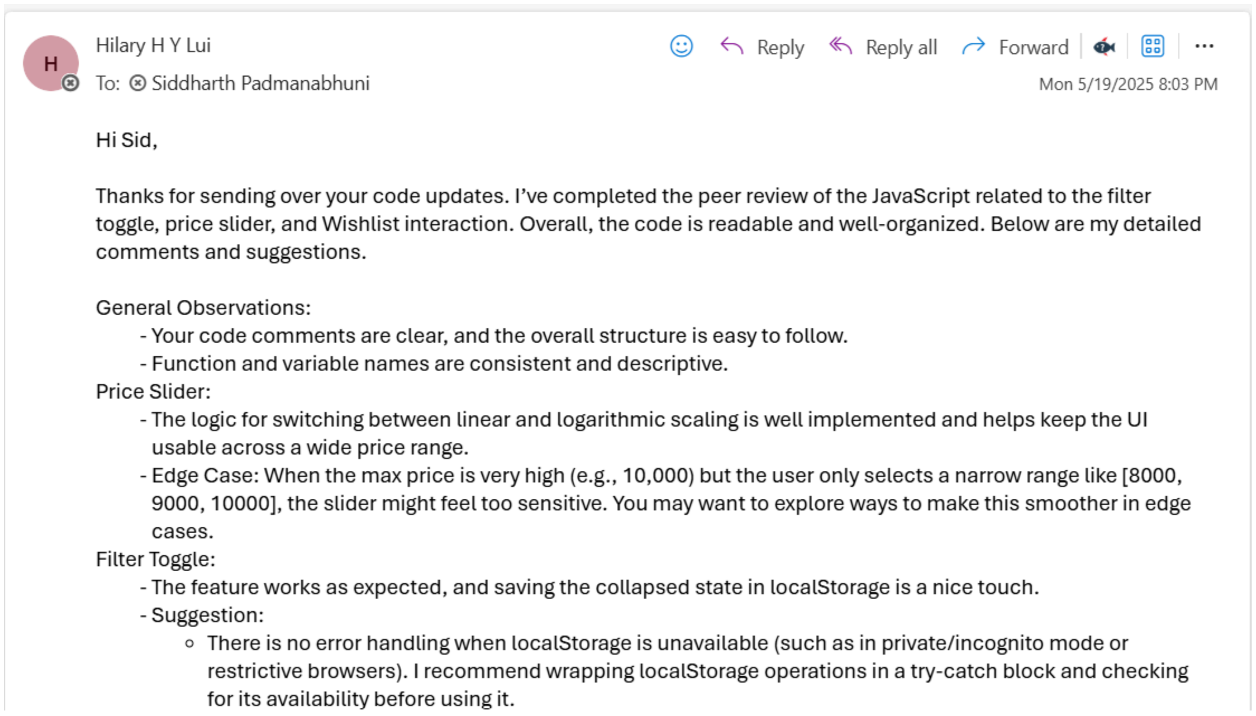
- **Filter toggle:** Save/collapse filter state and transition improvements.
- **Price range slider:** Handles both linear and conditional power-based scaling depending on price range; updates the UI in real-time.
- **Wishlist interaction:** Async wishlist add/remove with server confirmation and error handling.

A few things to keep an eye on during the review:

- Logic for the price slider (especially the linear vs. logarithmic behavior)
- LocalStorage persistence for the filter toggle
- Error handling in the wishlist functionality (e.g., non-JSON responses, network failures)
- Overall modularity/readability

Thanks!

Sid



H Hilary H Y Lui
To: Siddharth Padmanabhuni
Mon 5/19/2025 8:03 PM

Hi Sid,

Thanks for sending over your code updates. I've completed the peer review of the JavaScript related to the filter toggle, price slider, and Wishlist interaction. Overall, the code is readable and well-organized. Below are my detailed comments and suggestions.

General Observations:
- Your code comments are clear, and the overall structure is easy to follow.
- Function and variable names are consistent and descriptive.

Price Slider:
- The logic for switching between linear and logarithmic scaling is well implemented and helps keep the UI usable across a wide price range.
- Edge Case: When the max price is very high (e.g., 10,000) but the user only selects a narrow range like [8000, 9000, 10000], the slider might feel too sensitive. You may want to explore ways to make this smoother in edge cases.

Filter Toggle:
- The feature works as expected, and saving the collapsed state in localStorage is a nice touch.
- Suggestion:
  - There is no error handling when localStorage is unavailable (such as in private/incognito mode or restrictive browsers). I recommend wrapping localStorage operations in a try-catch block and checking for its availability before using it.

for its availability before using it.

Wishlist Interaction:

- The async handling is well done, and the use of aria-label for accessibility is great.
- Suggestions for Improvement:
    - The code assumes all server responses will be valid JSON. If a non-JSON response (like an HTML error page) is returned, response.json() could throw.
    - There is no user-facing feedback for network failures or server errors, which might leave users confused. Please consider showing a toast or alert to let the user know when an error occurs.

Commit Messages:

- The commit message for the filter toggle was labeled as "bug fixes". For clarity and better tracking, it would be more helpful if it were more specific (e.g., "Added filter collapse toggle with persistent state").
- The commit that includes the price slider and Wishlist interaction was titled "Implemented product listing, fixed zig zag." Since these are distinct features, it might help future maintainability to separate them and name them more descriptively in future commits.

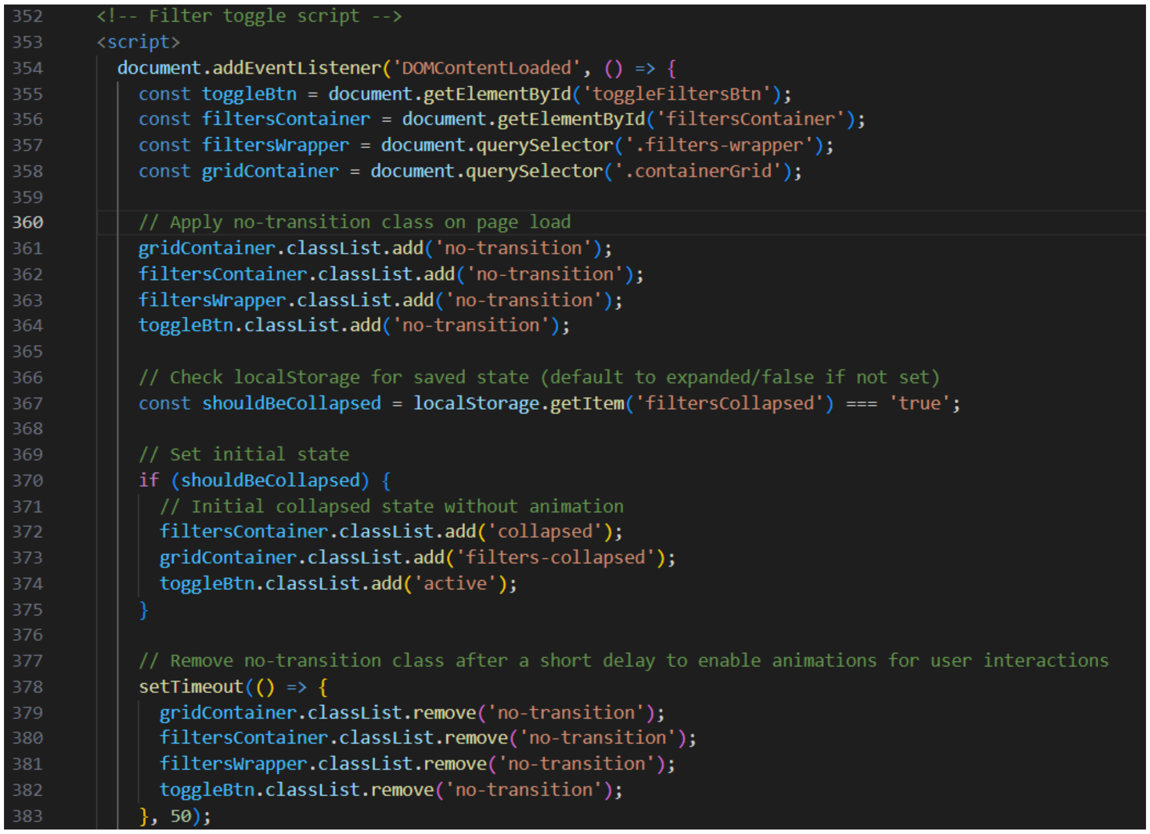
Feel free to let me know if you have any questions. Overall, great job on these features!

Best,
Hilary

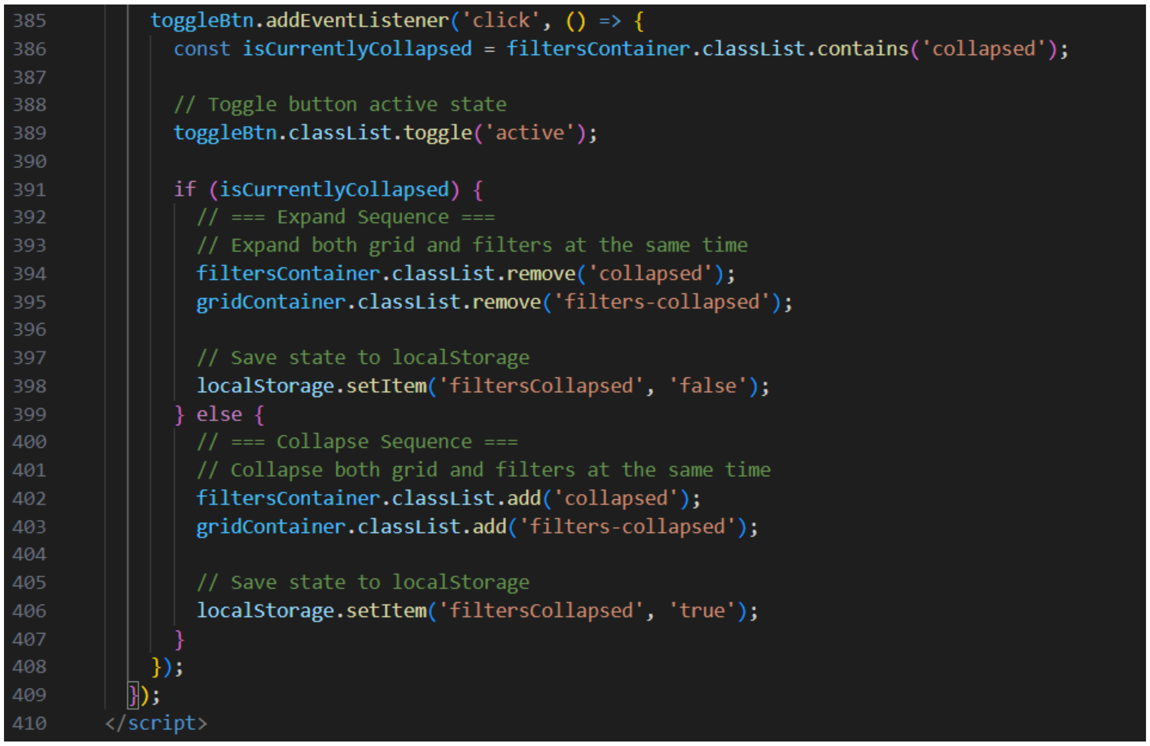
...

Reply Forward

Filter Toggle:

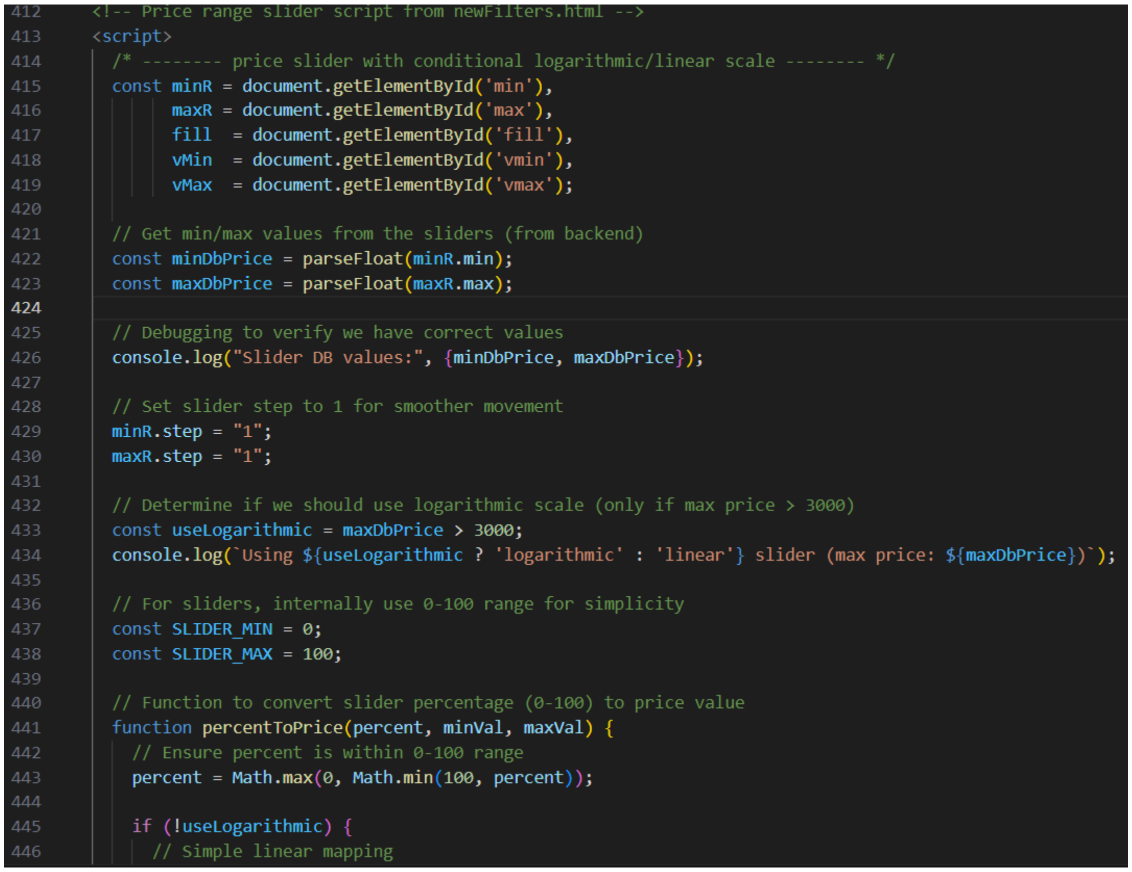


```
<!-- Filter toggle script -->
<script>
  document.addEventListener('DOMContentLoaded', () => {
    const toggleBtn = document.getElementById('toggleFiltersBtn');
    const filtersContainer = document.getElementById('filtersContainer');
    const filtersWrapper = document.querySelector('.filters-wrapper');
    const gridContainer = document.querySelector('.containerGrid');

    // Apply no-transition class on page load
    gridContainer.classList.add('no-transition');
    filtersContainer.classList.add('no-transition');
    filtersWrapper.classList.add('no-transition');
    toggleBtn.classList.add('no-transition');

    // Check localStorage for saved state (default to expanded/false if not set)
    const shouldBeCollapsed = localStorage.getItem('filtersCollapsed') === 'true';

    // Set initial state
    if (shouldBeCollapsed) {
      // Initial collapsed state without animation
      filtersContainer.classList.add('collapsed');
      gridContainer.classList.add('filters-collapsed');
      toggleBtn.classList.add('active');
    }

    // Remove no-transition class after a short delay to enable animations for user interactions
    setTimeout(() => {
      gridContainer.classList.remove('no-transition');
      filtersContainer.classList.remove('no-transition');
      filtersWrapper.classList.remove('no-transition');
      toggleBtn.classList.remove('no-transition');
    }, 50);
```

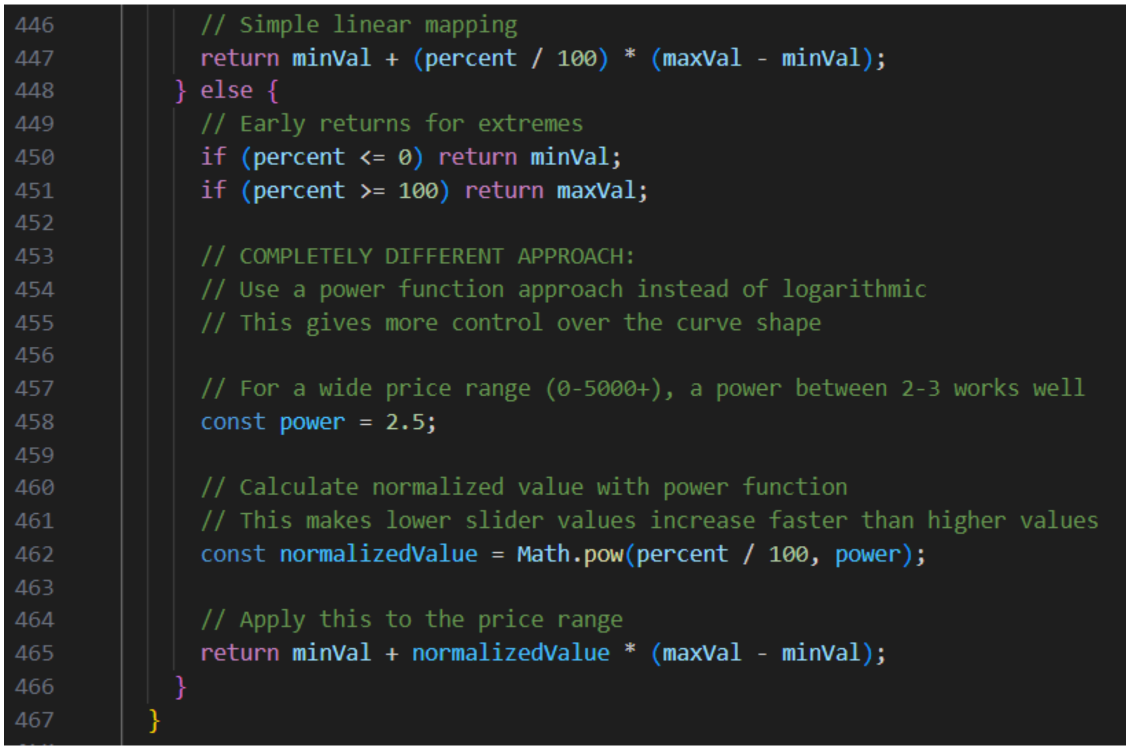


```
    toggleBtn.addEventListener('click', () => {
      const isCurrentlyCollapsed = filtersContainer.classList.contains('collapsed');

      // Toggle button active state
      toggleBtn.classList.toggle('active');

      if (isCurrentlyCollapsed) {
        // === Expand Sequence ===
        // Expand both grid and filters at the same time
        filtersContainer.classList.remove('collapsed');
        gridContainer.classList.remove('filters-collapsed');

        // Save state to localStorage
        localStorage.setItem('filtersCollapsed', 'false');
      } else {
        // === Collapse Sequence ===
        // Collapse both grid and filters at the same time
        filtersContainer.classList.add('collapsed');
        gridContainer.classList.add('filters-collapsed');

        // Save state to localStorage
        localStorage.setItem('filtersCollapsed', 'true');
      }
    });
  });
</script>
```

Price Range Slider:

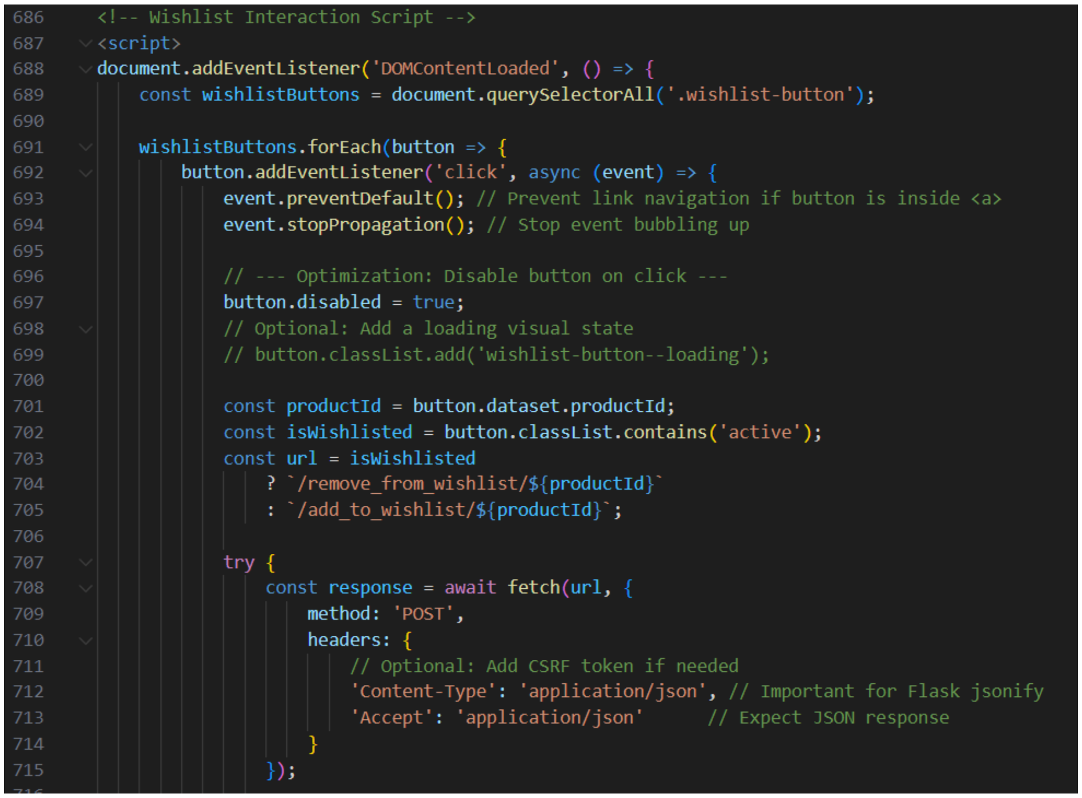


```
<!-- Price range slider script from newFilters.html -->
<script>
  /* -------- price slider with conditional logarithmic/linear scale -------- */
  const minR = document.getElementById('min'),
        maxR = document.getElementById('max'),
        fill  = document.getElementById('fill'),
        vMin  = document.getElementById('vmin'),
        vMax  = document.getElementById('vmax');

  // Get min/max values from the sliders (from backend)
  const minDbPrice = parseFloat(minR.min);
  const maxDbPrice = parseFloat(maxR.max);

  // Debugging to verify we have correct values
  console.log("Slider DB values:", {minDbPrice, maxDbPrice});

  // Set slider step to 1 for smoother movement
  minR.step = "1";
  maxR.step = "1";

  // Determine if we should use logarithmic scale (only if max price > 3000)
  const useLogarithmic = maxDbPrice > 3000;
  console.log(`Using ${useLogarithmic ? 'logarithmic' : 'linear'} slider (max price: ${maxDbPrice})`);

  // For sliders, internally use 0-100 range for simplicity
  const SLIDER_MIN = 0;
  const SLIDER_MAX = 100;

  // Function to convert slider percentage (0-100) to price value
  function percentToPrice(percent, minVal, maxVal) {
    // Ensure percent is within 0-100 range
    percent = Math.max(0, Math.min(100, percent));

    if (!useLogarithmic) {
      // Simple linear mapping
```

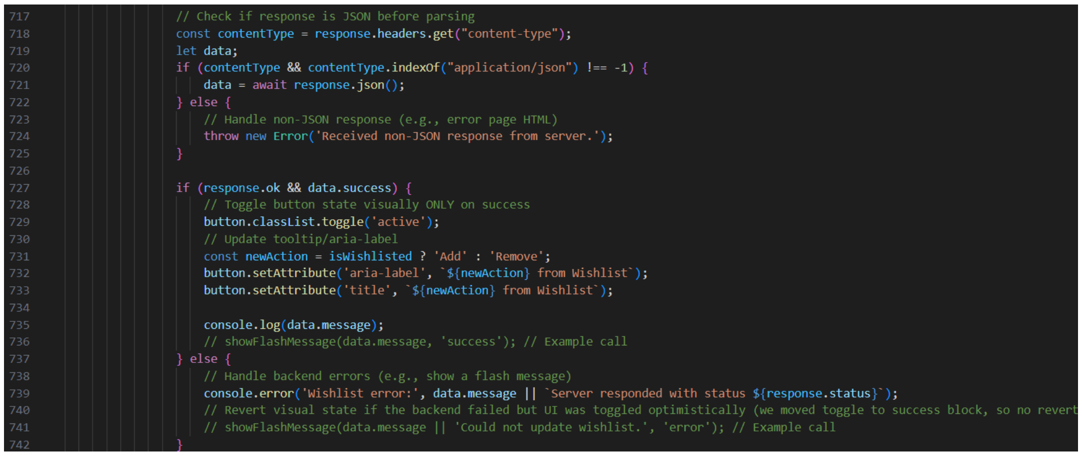


```
      // Simple linear mapping
      return minVal + (percent / 100) * (maxVal - minVal);
    } else {
      // Early returns for extremes
      if (percent <= 0) return minVal;
      if (percent >= 100) return maxVal;

      // COMPLETELY DIFFERENT APPROACH:
      // Use a power function approach instead of logarithmic
      // This gives more control over the curve shape

      // For a wide price range (0-5000+), a power between 2-3 works well
      const power = 2.5;

      // Calculate normalized value with power function
      // This makes lower slider values increase faster than higher values
      const normalizedValue = Math.pow(percent / 100, power);

      // Apply this to the price range
      return minVal + normalizedValue * (maxVal - minVal);
    }
  }
```

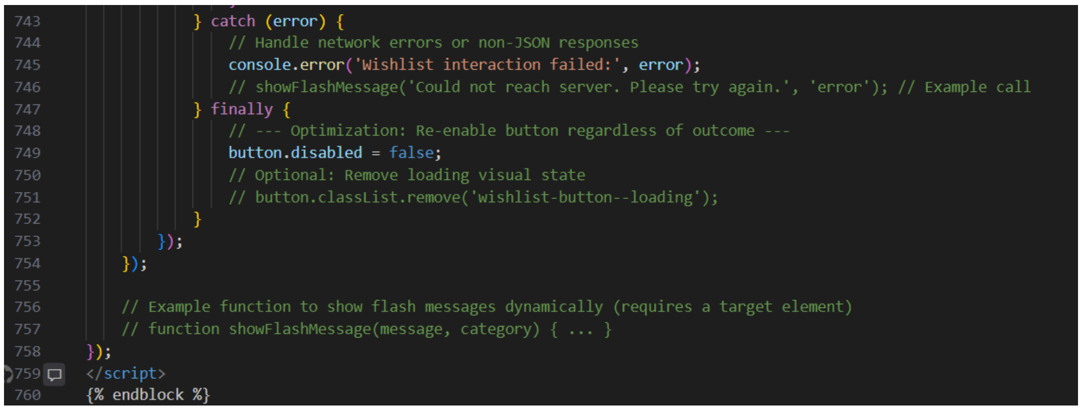
Wishlist Interaction:



```
<!-- Wishlist Interaction Script -->
<script>
document.addEventListener('DOMContentLoaded', () => {
    const wishlistButtons = document.querySelectorAll('.wishlist-button');

    wishlistButtons.forEach(button => {
        button.addEventListener('click', async (event) => {
            event.preventDefault(); // Prevent link navigation if button is inside <a>
            event.stopPropagation(); // Stop event bubbling up

            // --- Optimization: Disable button on click ---
            button.disabled = true;
            // Optional: Add a loading visual state
            // button.classList.add('wishlist-button--loading');

            const productId = button.dataset.productId;
            const isWishlisted = button.classList.contains('active');
            const url = isWishlisted
                ? `/remove_from_wishlist/${productId}`
                : `/add_to_wishlist/${productId}`;

            try {
                const response = await fetch(url, {
                    method: 'POST',
                    headers: {
                        // Optional: Add CSRF token if needed
                        'Content-Type': 'application/json', // Important for Flask jsonify
                        'Accept': 'application/json'      // Expect JSON response
                    }
                });
```



```
                // Check if response is JSON before parsing
                const contentType = response.headers.get("content-type");
                let data;
                if (contentType && contentType.indexOf("application/json") !== -1) {
                    data = await response.json();
                } else {
                    // Handle non-JSON response (e.g., error page HTML)
                    throw new Error('Received non-JSON response from server.');
                }

                if (response.ok && data.success) {
                    // Toggle button state visually ONLY on success
                    button.classList.toggle('active');
                    // Update tooltip/aria-label
                    const newAction = isWishlisted ? 'Add' : 'Remove';
                    button.setAttribute('aria-label', `${newAction} from Wishlist`);
                    button.setAttribute('title', `${newAction} from Wishlist`);

                    console.log(data.message);
                    // showFlashMessage(data.message, 'success'); // Example call
                } else {
                    // Handle backend errors (e.g., show a flash message)
                    console.error('Wishlist error:', data.message || `Server responded with status ${response.status}`);
                    // Revert visual state if the backend failed but UI was toggled optimistically (we moved toggle to success block, so no revert
                    // showFlashMessage(data.message || 'Could not update wishlist.', 'error'); // Example call
                }
```



```
            } catch (error) {
                // Handle network errors or non-JSON responses
                console.error('Wishlist interaction failed:', error);
                // showFlashMessage('Could not reach server. Please try again.', 'error'); // Example call
            } finally {
                // --- Optimization: Re-enable button regardless of outcome ---
                button.disabled = false;
                // Optional: Remove loading visual state
                // button.classList.remove('wishlist-button--loading');
            }
        });
    });

    // Example function to show flash messages dynamically (requires a target element)
    // function showFlashMessage(message, category) { ... }
});
</script>
{% endblock %}
```

GenAI Interaction:

Prompt used:

"Please review this JavaScript code related to front-end UI features (filter toggle, price range slider, wishlist). Focus on logic clarity, error handling, modularity, and browser compatibility. Suggest improvements as needed."

1. Filter Toggle: Save/Collapse Filter State

Review:

✅ Clear structure and DOM handling. Easy to follow.

💡 Suggestion: Add inline comments explaining the localStorage logic and UI state. For example:

    // Save the opposite of the current state to persist toggle state on next load

⚠️ Edge case: If localStorage is disabled in the browser, there should be a fallback or error catch.

📦 Modular idea: Wrap toggle logic in a function (e.g., initializeFilterToggle()) to improve reusability/testability.

2. Price Range Slider

Review:

✅ Great logic for dual-mode price calculation (linear/logarithmic).

📌 Tip: Add a comment explaining why a power of 2.5 is used. This helps future devs understand the UX reasoning.

💡 Consider renaming percentToPrice to calculatePriceFromPercent for clarity.

✅ Event listener on both sliders is effective.

3. Wishlist Interaction

Review:

✅ Async handling and UI update logic are well-done.

✅ Good use of aria-label for accessibility!

⚠️ Edge Case Tip: If the response is not valid JSON (e.g., HTML error page), response.json() may throw.

Suggested improvement:

```
const contentType = response.headers.get('content-type');
if (contentType && contentType.includes('application/json')) {
  const data = await response.json();
  // ...
} else {
  throw new Error('Unexpected response format');
}
```

🔧 Error logging: Consider adding user-facing error messages in addition to console.error.

✅ Summary Feedback

Area	Rating	Notes
Code Clarity	⭐⭐⭐⭐☆	Modular, easy to follow, minor comment additions needed.
Naming Consistency	⭐⭐⭐⭐⭐	Matches common conventions and M2 standards.
Error Handling	⭐⭐⭐☆	Mostly good, wishlist logic could use stricter parsing checks.
Accessibility	⭐⭐⭐⭐☆	Use of `aria-label` is a great touch.
Modularity	⭐⭐⭐⭐☆	Could be improved by wrapping blocks into named functions.

GenAI Reflection Summary:

Tool Used:

ChatGPT (GPT-4, May 2024 version)

How You Used It:

Pasted in the JavaScript snippets from index.html containing filter toggle, price slider, and wishlist features

Asked for a code review focused on error handling, readability, and modularity

Benefits:

Quickly surfaced edge cases I hadn't considered (e.g., network failure in fetch requests)

Suggested more descriptive comments and clearer function splitting

Sample Prompt:

"Please review this JavaScript code related to a price slider and wishlist feature. Check for logic clarity, error handling, and consistency with front-end best practices."

Utility Rating:

HIGH – Helped validate the quality of my review and provided suggestions I hadn't thought of

5) Self-check on best practices for security

Security self-check plan:

Asset to be protected	**Possible/Expected Attacks**	**Consequences of Breach**	**Strategy to Mitigate/Protect**
Sensitive user data (e.g., email, password, reviews)	Unauthorized access, data breach	Compromised accounts, privacy violations	- Encrypt passwords (e.g., bcrypt) - Use HTTPS for secure transport
User authentication and authorization	Privilege escalation, session hijacking	Users gaining admin rights or accessing others' data	- Role-based access control (RBAC) - Token/session expiration
POST requests (e.g., creating listings)	Script injection, spam, fake listings	Data integrity issues, flooding with bad data	- Input validation & sanitization - CSRF protection
Server resources (e.g., image uploads, page requests)	DDoS, bloating via excessive requests or uploads	Server slowdown, crashes, denial of access to real users	- Limit file sizes - Rate limiting - Efficient request handling
Admin functions (e.g., delete/edit listings)	Unauthorized use of admin APIs	Loss or misuse of platform content	- Admin route protection - Audit logs for admin actions

Security Checklist & Other Notes

- [x] ~~Password is encrypted in the database~~
- [x] ~~Input data is validated~~
- [x] ~~Search bar input limited to 40 alphanumeric characters~~
- [x] ~~Registration requires SFSU email domain (ends with "@sfsu.edu")~~
- [x] ~~Acceptance of terms is required in registration form~~

6) Self-check of adherence to original Non-functional specs – performed by team leads

1. Application shall be developed, tested and deployed using tools and cloud servers approved by Class CTO and as agreed in M0
    a. 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
    a. DONE
3. All or selected application functions shall render well on mobile devices (no native app to be developed)
    a. DONE
4. Posting of sales information and messaging to sellers shall be limited only to SFSU students
    a. DONE
5. Critical data shall be stored in the database on the team's deployment server.
    a. DONE
6. No more than 50 concurrent users shall be accessing the application at any time
    a. DONE
7. Privacy of users shall be protected
    a. DONE
8. The language used shall be English (no localization needed)
    a. DONE
9. Application shall be very easy to use and intuitive
    a. DONE
10. Application shall follow established architecture patterns
    a. DONE
11. Application code and its repository shall be easy to inspect and maintain
    a. DONE
12. Google analytics shall be used
    a. DONE
13. No e-mail clients or chat services 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
    a. DONE
14. Pay functionality, if any (e.g. paying for goods and services) shall not be implemented nor simulated in UI
    a. DONE
15. Site security: basic best practices shall be applied (as covered in the class) for main data items
    a. DONE

16. Media formats shall be standard as used in the market today
    a. DONE
17. Modern SE processes and tools shall be used as specified in the class, including collaborative and continuous SW development and GenAI tools
    a. DONE
18. The application UI (WWW and mobile) shall prominently display the following exact text on all pages "SFSU Software Engineering Project CSC 648-848, Spring 2025. For Demonstration Only" at the top of the WWW page Nav bar. (Important so as to not confuse this with a real application). You have to use this exact text without any editing.
    a. DONE

7) Use of GenAI tools like ChatGPT and Copilot

GenAI Use

Tool Used:
ChatGPT-4 (OpenAI)

How the Tool Was Used:
ChatGPT was used to help draft the QA test plan structure, generate sample test cases, and reformat the test table for easy readability. It helped simulate edge-case inputs and clearly define expected vs. actual behavior.

Key Prompts Used:

- "Create QA test cases for a form submission feature with validations and image uploads."
- "Help format a QA test plan for a web project with test title, input, and results columns."
- "Explain how to QA test cross-browser compatibility for form features."

Benefit:
Using GenAI saved time and helped ensure thorough test coverage by suggesting common edge cases and how they would typically fail or succeed. It also helped format our table for readability by non-technical reviewers.

Utility Rank:
HIGH – It provided strong templates, realistic test ideas, and helped us focus on what matters most in cross-browser QA.