

“SW Engineering CSC648-848 Spring 2025”

Gator Savvy

Team 5

Yu-Ming Chen (Team Lead) - ychen88@sfsu.edu

David Cabanela (GitHub Master) - dcabanel@mail.sfsu.edu

David (Jun) Chang (Backend Lead) - dchang6@mail.sfsu.edu

Alexis Perez (Frontend Lead) - aperez75@sfsu.edu

Athan Cheung (Frontend) - acheung4@mail.sfsu.edu

Milestone 4

History

	Date Submitted	Date Revised
Group meeting & started doc		5/1/2025
Group meeting		5/4/2025
Group meeting		5/11/2025
Group meeting + Demo prep		5/15/2025
Group meeting + Demo prep		5/17/2025
Group meeting + Final wrap up	5/18/2025	5/18/2025
Presentation day	5/19/2025	5/19/2025

1) Product Summary

Name of our product: Gator Savvy

Product Description: Gator Savvy is a marketplace web application for SFSU students and faculty to buy, sell, rent items, and get exclusive student deals. Whether it's textbooks, clothes, tech, or everyday essentials, students can find what they need at affordable prices from other students or SFSU-affiliated groups/services.

Itemized List of All Major Committed Functions

1. Guests shall browse and search for items.
2. Guests shall filter item listing search results by category, price, and more.
3. Guests shall sign up for a Gator Savvy account (SFSU email required).
4. Registered users shall post item listings for sale or rent.
5. Registered users shall filter item listing search results by category, price, etc.
6. Registered users shall be verified and receive a verification badge.
7. Registered users shall set up a profile with a username and picture.
8. Registered users shall message each other inside the app.
9. Admins shall review and remove item listings.

URL to Gator Savvy: <http://35.94.172.233>

2) Usability test plan for a selected function

ONE major function that will be tested for usability is the **SEARCH** function.

1. Test objectives:

We will be testing the usability of the search function of our app. The goal is to observe whether users can effectively apply filters, navigate search results, and complete searches without confusion. We also want to measure user satisfaction and identify any usability issues that make the search difficult or frustrating (e.g., unclear filter options, complicated workflows).

2. Test the background and setup

System setup:

The app can be tested on our deployment server IP or locally on one of the developers' computers. Users/Testers can receive the IP address for testing. Our app runs on standard web browsers like Google Chrome or Firefox.

Starting point:

Users/Testers will start at our home page (/). From there, they can go to the search bar located on top of the home page and in the center of the navigation bar. Users do not have to be logged in to search for items.

Hardware requirements:

Testers will need access to internet connections and a web browser, either on a computer or a mobile device (if accessing the app via the deployment IP).

Intended users:

The intended users are students or faculty from San Francisco State University, with email addresses ending in (@sfsu.edu or @mail.sfsu.edu), who are interested in posting or browsing textbooks, clothing, tech items, or everyday essentials.

URL of the system to be tested:

<http://35.94.172.233/>

Test environment:

The tests can be conducted at home (a typical environment where Gator Savvy would be used). No cameras are necessary. There will be no monitoring since testers will provide feedback via a questionnaire after the test is complete. Training will not be provided since we want to test whether users can use the search function naturally without any guidance.

3. Usability Task Description:

Tasks (Specific & Measurable):

1. Search for an item you would normally look for (like a textbook or a calculator).
2. Apply a category filter (like "Electronics" or "Books") to your search.
3. Change the filter and try searching for a different item.

Feedback after performing the required task:

- Did the user find something? Something relevant?
- Did the filter work properly?
- Was the change in filter/search successful?

Task	Description
Task	Search for an item on the website with the search function
Machine State	On the main page, not logged in
Successful completion criteria	Results display relevant items matching the search query
Benchmark	Completed in under 1 minute

4. Plan for evaluation of Effectiveness

We will measure effectiveness by checking if users can complete the search tasks. A task is considered completed and effective if:

1. The user finds an item or items that match their search
2. The category filter returns the correctly filtered results
3. The user shall switch filters and search again without any issues

5. Plan for Evaluation of Efficiency:

We will measure efficiency by tracking how long it takes users to complete each task and how easily they navigate the process.

- Measure how long it takes users to finish tasks.
- Track the time spent searching, applying filters, and changing them.
- Observe if users make extra or unnecessary clicks or look confused.
- Expect tasks to be completed in under 1 minute with minimal confusion.

We expect users to complete each task quickly (less than 1 minute). If the user experiences any delays or issues, that would mean that the function is not very efficient.

6. Plan for Evaluation of user satisfaction (Likert scale questionnaire):

The search function was easy to use.

1. Strongly Disagree
2. Disagree
3. Neutral
4. Agree
5. Strongly Agree

Using a filter made it easier to find what I was looking for.

1. Strongly Disagree
2. Disagree
3. Neutral
4. Agree
5. Strongly Agree

It was easy to switch between different filters.

1. Strongly Disagree
2. Disagree
3. Neutral
4. Agree
5. Strongly Agree

The search results were relevant to what I was looking for.

1. Strongly Disagree
2. Disagree
3. Neutral
4. Agree
5. Strongly Agree

Overall, I am satisfied with my experience using the search function.

1. Strongly Disagree
2. Disagree
3. Neutral
4. Agree
5. Strongly Agree

7. GenAI use:

ChatGPT - HIGH

Task: I asked ChatGPT about the usability task description and Likert Scale Questionnaire. It helped with wording, structuring/organizing the sections, and gave feedback on what to include and improve. Very helpful.

3) QA test plan and QA testing

1. Test objectives: - What is being tested

We will test whether the search function works properly. This includes checking whether users can search for items, apply filters like category, and get correct results. We also want to make sure it doesn't break or show errors.

2. HW and SW setup (including URL):

HW: Any laptop or desktop with internet

SW: Google Chrome or Firefox browser (Latest Version)

URL: <http://35.94.172.233/>

3. Feature to be tested

We will be testing the **search** feature. Search bar, filters, and results.

4. QA Test plan

The 2 major WWW browsers we chose were Google Chrome and Firefox. Both passed the test with flying colors. Everything worked as expected.

Test Title	Description	Input	Expected	Chrome	Firefox
Blank search	Searching with nothing	Query: ""	Show all items	PASS	PASS
Search w/o filter	Searching for something without a filter	Query: "Camera" Filter: n/a	Show related items to the search query	PASS	PASS
Search w/ filter	Searching for something with a filter	Query: "Camera" Filter: "Electronics"	Show filtered plus matching items to search	PASS	PASS
No match	Searching for something we don't have	"testingtesting"	Display "Results (0)"	PASS	PASS

4) Peer Code Review

Jim is requesting code review of the search functionality via the GitHub pull request link in the email:

[M4 Code Review Request] Search Feature

 **Yu-Ming Chen**
To:  David B Cabanela

Thu 5/15/2025 10:23 AM

Hi David,

I've pushed code changes for the search feature (search.py, Searchbar.js, and Results.js) to a new branch: `search_m4_code_review`.

Here's the GitHub pull request link:

<https://github.com/CSC-648-SFSU/csc648-fa25-0104-team05/pull/4>

Can you review it on GitHub and send me a quick summary email after? We'll use the screenshots for the M4 submission.

Thanks!

Jim

David, the reviewer, responded to the email with a summary of code review feedback:

[M4 Code Review Request] Search Feature

 **David B Cabanela**
To:  Yu-Ming Chen

Sat 5/17/2025 8:17 PM



Hi Jim,

I have completed the code review and reviewed your pull request on GitHub. I left a comment on GitHub for you to check out. Below is a summary of the code review feedback. I have also attached screenshots with more detailed comments for your reference.

Code Review Summary

Overall, great job on this search functionality feature. Here is a summary of feedback including kudos and improvements you can make to the code for this feature:

What's working well:

- Clear, readable code with intuitive variable names, proper indentation, and Python method docstrings.
- You properly follow the naming convention established in the Data section in Milestone 2.
- Good separation of concerns by separating search logic into dedicated .py and .js files.

Recommendations for improvement:

- Consider removing some of the in-line comments explaining obvious logic to reduce redundancy.
- Add a brief header comment to the top of each code file to briefly summarize its purpose.
- In the future, we can dynamically search for categories in the Category database table instead of hardcoding it.
- We can more intuitively group and organize import statements.

Hope this helps! Let me know if you have any questions.

Best regards,
David

David's code review comments within the GitHub pull request:

Prepared search functionality code for peer review #4

[Open](#) JimboRumbleCad... wants to merge 1 commit into `dev` from `search_m4_code_review`

Conversation 2 Commits 1 Checks 0 Files changed 3

JimboRumbleCadumbo commented 3 days ago

M4 Peer Code Review: Prepared search.py, Searchbar.js, and Results.js for review.

davidcabanela reviewed 15 hours ago [View reviewed changes](#)

davidcabanela left a comment • edited ▾

Overall, great job on this search functionality feature. Here is a summary of feedback including kudos and improvements you can make to the code for this feature:

What's working well:

- The code is very readable with intuitive variable names and proper indentation.
- The docstrings at the beginning of each Python method are also helpful.
- You properly follow the naming convention established in the Data section in Milestone 2 (e.g. Item_Listing for marketplace items).
- It's great that you separated out the search logic into dedicated.py and .js files instead of combining everything into a single, giant Flask app.py. This really improves maintainability and modularity.
- Git commit comments are concise and informative.

Recommendations for improvement:

- Consider removing some of the in-line comments sprinkled across the code that explain straightforward code. It would be better to only include comments to explain non-obvious or unconventional logic.
- Maybe we can add a header comment to the top of the file to briefly summarize the overall purpose of the code for that file at a glance.
- In the future, we can enhance flexibility by dynamically search for categories in the Category database table instead of hardcoding it at the top of search.py.
- Something minor, but we can more intuitively group the import statements at the top of each Python file more intentionally (e.g. by standard library, third-party packages, local modules).

...

Screenshots of more specific code review comments from David are included in the email reply to Jim, which is included in the next few pages.

Code review comments for search.py:

Prepared search functionality code for peer review #4

JimboRumbleCade... wants to merge 1 commit into `dev` from `search_m4_code_review`

Conversation 0 → Commits 1 ↗ Checks 0 ↗ Files changed 3 ↗

Changes from all commits ↗ File filter ↗ Conversations ↗ 0 / 3 files viewed Ask Copilot ↗ Review in codespace ↗ Review changes ↗ Viewed ↗ ⋮

Filter changed files

application/main/search.py

```
from flask import Flask, request, jsonify
from flask_sqlalchemy import SQLAlchemy
from dotenv import load_dotenv
import MySQLdb.cursors
import os
import base64
# Category constants for better readability and maintainability
CATEGORY_IDS = {
    'Electronics': 1,
    'Books': 2,
    'Clothing': 3,
    'Furniture': 4,
    'Sports Equipment': 5
}
def init_search_routes(app):
    """
    Initialize search routes and MySQL configuration for the Flask application.
    Args:
        app: Flask application instance
    """
    # Get the shared MySQL connection
    mysql = app.config.get('MYSQL_CONNECTION')
    if not mysql:
        # Fallback if shared connection is not available
        load_dotenv()
    # Configure MySQL connection
    app.config['MYSQL_HOST'] = os.getenv('MYSQL_HOST')
    app.config['MYSQL_USER'] = os.getenv('MYSQL_USER')
    app.config['MYSQL_PASSWORD'] = os.getenv('MYSQL_PASSWORD')
    app.config['MYSQL_DB'] = os.getenv('MYSQL_DB')
    mysql = MySQL(app)
    @app.route('/api/search', methods=['POST'])
    def api_search():
        """
        Search logic implementation
        """
        # Implement search logic here
        return jsonify({'message': 'Search API endpoint'})
```

Consider using the Category table instead of hardcoding for flexibility

Consider using the Category table instead of hardcoding for flexibility

This in-line comment
explains
straightforward code,
can remove

Proper usage of Data Entity naming from M2

Code review comments for Searchbar.js:

<https://github.com/CSC-648-SFSU/csc648-fa25-0104-team05/pull/4/files>

Prepared search functionality code for peer review #4

application/main/search.py

```

1 +// Please code review this file.
2 +
3 export default {
4     template:
5         <!-- Search Bar and Filters Dropdown in Flexbox -->
6         <div class="search-bar-container">
7             <!-- Filter Dropdown -->
8             <div class="dropdown" :class="{ open: isDropdownVisible }">
9                 <button @click="toggleDropdown" class="dropdown">{{ selectedFilter || "All" }}</button>
10                <div class="dropdown-content">
11                    <div v-for="filter" in filters" :key="filter" @click="selectFilter(filter)">
12                        {{ filter }}
13                    </div>
14                </div>
15            <!-- Search Bar -->
16            <div class="search-bar">
17                <input
18                    type="text"
19                    v-model="searchQuery"
20                    placeholder="Search..." 
21                    @keyup.enter="submitSearch"
22                />
23                <button @click="submitSearch">Search</button>
24            </div>
25        </div>
26    ,
27    setup() {
28        const searchData = Vue.inject('searchData');
29        const searchQuery = Vue.ref('');
30        const selectedFilter = Vue.ref('');
31        const isDropdownVisible = Vue.ref(false);
32        const router = VueRouter.useRouter();
33    }
34    function submitSearch() {
35
36        fetch('/api/search', {
37            method: 'POST',
38            headers: { 'Content-Type': 'application/json' },
39            body: JSON.stringify({
40                query: searchQuery.value,
41                filter: selectedFilter.value || 'All'
42            })
43        })
44        .then(res => {
45            if (!res.ok) {
46                return res.json().then(err => {
47                    throw new Error(`Server error: ${err.error || res.statusText}`);
48                });
49            }
50        })
51        .catch(error => {
52            console.error('Search error:', error);
53            alert(`Search failed: ${error.message}`);
54        });
55    }
56
57    return {
58        searchData,
59        selectedFilter,
60        isDropdownVisible,
61        filters: ['All', 'Electronics', 'Books', 'Clothing', 'Furniture', 'Sports Equipment'],
62        submitSearch,
63        toggleDropdown() {
64            isDropdownVisible.value = !isDropdownVisible.value;
65        },
66        selectFilter(filter) {
67            selectedFilter.value = filter;
68            isDropdownVisible.value = false;
69        }
70    };
71 }
72 
```

Add brief header summary comment to top of file

application/main/static/components/Searchbar.js

```

32    const router = VueRouter.useRouter();
33
34    function submitSearch() {
35
36        fetch('/api/search', {
37            method: 'POST',
38            headers: { 'Content-Type': 'application/json' },
39            body: JSON.stringify({
40                query: searchQuery.value,
41                filter: selectedFilter.value || 'All'
42            })
43        })
44        .then(res => {
45            if (!res.ok) {
46                return res.json().then(err => {
47                    throw new Error(`Server error: ${err.error || res.statusText}`);
48                });
49            }
50        })
51        .catch(error => {
52            console.error('Search error:', error);
53            alert(`Search failed: ${error.message}`);
54        });
55    }
56
57    return {
58        searchQuery,
59        selectedFilter,
60        isDropdownVisible,
61        filters: ['All', 'Electronics', 'Books', 'Clothing', 'Furniture', 'Sports Equipment'],
62        submitSearch,
63        toggleDropdown() {
64            isDropdownVisible.value = !isDropdownVisible.value;
65        },
66        selectFilter(filter) {
67            selectedFilter.value = filter;
68            isDropdownVisible.value = false;
69        }
70    };
71 }
72 
```

In future iterations, consider not hardcoding, fetch from DB

application/main/static/pages/Results.js

```

... ...
@@ -1,3 +1,5 @@

```

Code review comments for Results.js:

The screenshot shows a GitHub code review interface for a file named 'Results.js'. The file is located in the 'application/main/static/pages' directory. The code is a Vue.js component for displaying search results. A specific line of code at the bottom of the component's template is highlighted with a green background and a yellow border, indicating it has been reviewed or annotated.

```
... @@ -1,3 +1,5 @@
  + // Please code review this file.
  +
1   3   console.log("Dashboard.js Loaded!");
2   4
3   5   export default {
4   6     template: `
5       <Navbar></Navbar>
6       <div class="page-wrapper">
7         <div class="container">
8           <div class="content">
9             <h1> Results <span class="result-count">{{ filteredResults.length }}</span> </h1>
10            ...
11            <div class="filter-section">
12              <label>Price Range:</label>
13              <input type="number" v-model="minPrice" placeholder="min" />
14              <span>--</span>
15              <input type="number" v-model="maxPrice" placeholder="max" />
16            ...
17            <button @click="applyFilter">Apply Filter</button>
18            <button @click="clearFilter">Clear Filter</button>
19          </div>
20
21          <!-- Product Grid -->
22          <div class="result-product-grid" v-show="filteredResults.length > 0">
23            ...
24            <router-link v-for="result in filteredResults"
25              :key="result.id"
26              :to="'/item/:id' + result.item_id"
27              class="card-link"
28            >
29              <div class="result-card">
30                
31                <h3>{{ result.name || result.title }}</h3>
32                <p>${{ result.price || result.cost }}</p>
33              </div>
34            </router-link>
35
36            ...
37            </div>
38          </div>
39          <footer class="footer">
40            ...
41            <p>© 2025 CSC 648 Team 05. All rights reserved.</p>
42            <router-link to="/about" class="footer-link">About</router-link>
43          </footer>
44        </div>
45      </div>
46    `,
47    setup() {
48      //Added to make sure searchresults are reset if you hit home page
49      const route = useRouter().useRoute();
50      const searchData = Vue.inject("searchData");
```

A callout bubble with an orange border and white text is overlaid on the code, pointing to the footer section. The text in the bubble reads: "Consider utilizing a reusable footer component for modularity".

GenAI Use for Code Review

- **Tool:** ChatGPT, **Version:** GPT-4-turbo
- **Task:** I used this tool to perform a code review of the same 3 files for the search functionality. I provided context of what our project is about and gave instructions on what to pay attention to in the code review, such as naming conventions and code comments.
- **Benefits:** Having ChatGPT review the code was a good way to double-check our work, like having a different set of eyes in case we missed something. It also structured the code review similarly to how David did it (e.g., what's working well section, improvements section), which was a good indicator that we are on the right track in terms of performing code reviews within the team.
- **Example prompt:**
 - *"I'm working on a team project for my software engineering capstone class. This project is a marketplace web app where students from my university can buy and rent items. One of the key features of this webapp is the search functionality to allow users to browse and filter item listings. Perform a code review of the following code for this feature. Code review must include checking for basic header and in-line comments, as well as proper and consistent class/methods/variable names, and consistency with naming established in the DATA section of Milestone 2."*

Here is the naming established in milestone 2: [included bulleted list]

There are 3 code files for you to review: search.py, Searchbar.js, and Results.js. [shared code files]"

- Example response from ChatGPT:

 [search.py](#) (Flask Backend)

What's Good

- Header comment and docstrings: Present and descriptive.
- Modular route definition: `init_search_routes()` is good practice.
- Secure parameterized queries: Avoids SQL injection.
- Image handling with base64: Nice touch for frontend use.

Suggestions for Improvement

- Naming Consistency: Use `item_name` instead of `name` when filtering (Milestone 2 defines “Item Name”).
- Error Handling: Good use of `try/except`, but logging can be cleaner and centralized.
- Inline Comments: Add inline comments, especially for SQL query construction.
- Hardcoded Categories: Consider dynamically loading these from a Category table to stay DRY.
- **Rank of GenAI Utility: MEDIUM** – while it was a good exercise to have ChatGPT review the code, a lot of the review points were similar to what David had already commented (e.g., dynamically retrieve categories). The GenAI tool also didn’t have access to the full codebase, and having that fuller context might have yielded more insightful feedback. Also, some feedback was inconsistent with what we learned in class. For example, we learned to only include inline code comments for unconventional logic, but ChatGPT recommended adding inline comments for SQL query construction, which should be relatively obvious logic that doesn’t necessarily need comments.

5) Self-check on best practices for security

Major assets we are protecting in our team application:

- Password, item posting, server storage, SQL injection

The major threats to each asset above:

- Password: Easy to brute force without extra encryption
- Item posting: Bypass law enforcement, selling illegal items
- SQL injection: Having root access to the database or even the server kernel

Consequences of a security breach:

- Personal data breach; potential legal infringement; lose access to server

Methods of protection

- Password: We encrypted our password using bcrypt, so it will be impossible for hackers to brute force the password
- Item posting: Admin approval will ensure the item is legal and contains no potential harm to people
- SQL injection: Passed SQL injection detection tool - sqlmap's highest level & risk injection test on all pages.

Input data validation:

- The search bar can accept input of up to 40 alphanumeric characters.
- The customer's e-mail must include "@[sfsu.edu](#)" or "@[mail.sfsu.edu](#)" at the end.
- Acceptance of terms is provided in the registration form as a required item.

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

Our list of non-functional requirements:

1. The application shall be developed, tested, and deployed using tools and cloud servers approved by Class CTO and as agreed in M0 **DONE**
2. The application shall be optimized for standard desktop/laptop browsers, e.g., it must render correctly on the two latest versions of the two major browsers **DONE**
3. All or selected application functions shall render well on mobile devices (no native app to be developed) **DONE**
4. Posting of sales information and messaging to sellers shall be limited only to SFSU students **DONE**
5. Critical data shall be stored in the database on the team's deployment server. **DONE**
6. No more than 50 concurrent users shall be accessing the application at any time. **DONE**
7. The privacy of users shall be protected **DONE**
8. The language used shall be English (no localization needed) **DONE**
9. The application shall be straightforward to use and intuitive **DONE**
10. The application shall follow established architecture patterns **DONE**
11. The application code and its repository shall be easy to inspect and maintain **DONE**
12. Google Analytics shall be used **DONE**
13. No e-mail clients or chat services shall be allowed. Interested users can only message the sellers via in-site messaging. One round of messaging (from user to seller) is enough for this application **DONE**
14. Pay functionality, if any (e.g., paying for goods and services), shall not be implemented or simulated in UI. **DONE**
15. Site security: basic best practices shall be applied (as covered in the class) for the main data items **DONE**
16. Media formats shall be standard as used in the market today **DONE**
17. Modern SE processes and tools shall be used as specified in the class, including collaborative and continuous SW development and GenAI tools **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. (It is important not to confuse this with a real application.) You have to use this exact text without any editing.
- **DONE**

7) Use of GenAI tools like ChatGPT and Copilot

AI models used:

ChatGPT (GPT-4o-mini-high)

Task 1: Used ChatGPT for debugging messaging. There was a query error, and the error is hard to spot.

Helpfulness: Very helpful. It found the error very quick and fixed it within 5 minutes.

Task 2: I asked ChatGPT about the usability task description and Likert Scale Questionnaire. It helped with wording, structuring/organizing the sections, and gave feedback on what to include and improve.

Helpfulness: Helpful, saves everyone a lot of time. Helped with wording, gave good examples and feedback, and helped finalize and polish.

Task 3: I asked ChatGPT about adding a bold SOLD text next to the messaging page if the item is detected as sold

Helpfulness: Very helpful, saves tons of time. Even for backend people, it is easy to integrate your code into the frontend even if you don't know front-end code. Seems easier to become a full-stack developer now with the help of GenAI.

Task 4: I asked ChatGPT for assistance on getting the functionality for sign-up working.

Helpfulness: It helped me to implement the functionalities and saved a lot of time. It even helped me with many trials and errors, and how to resolve them.

Task 5: I asked ChatGPT how to connect Google Analytics to track our site's activities.

Helpfulness: It was straightforward and helpful as it told me how to implement it and get it working.