SW Engineering CSC648/848

Application Name: NNSE
Section 02, Team 06

Name	Role
Anzara Ausaf	**Team Lead / Frontend Lead**
Harris Chan	**Git/Document/Instance Master**
Jiqing Liu	**Backend / Debugger / Backend Lead**
Nicolas Carrillo	**Database Engineer/ Backend**
Michael Ho	**Database Engineer / Document Organizer**
Chance Vodnoy	**Backend**
Casey Steven	**Backend**

Milestone 4

April 23, 2025

Revision Table			
Revisions	Date	Author(s)	Description of Changes
1	04/14/2025	Michael Ho	Initial Draft
2	04/15/2025	Harris Chan	Completed Product Summary
3	04/16/2025	Anzara Ausaf	Added Unique Feature
4	04/19/2025	Michael Ho	Completed Usability and QA Test plans
5	04/22/2025	Anzara Ausaf	● Team Member role changed. **Jiqing Liu**: Added Backend Lead **Harris Chan**: Added Instance Master **Michael Ho**: Added Document Organizer
6	04/24/2025	Team Members	Final Draft

1) Product summary (e.g. how would you market and sell your product)
NoName Search Engine is a smart, flexible discovery platform designed to help users—especially college students—quickly find the best online tools and resources for every need. Whether you're looking for academic assistance, creative apps, mental-health support, or career development resources, NoName Search Engine combines powerful filters, multimedia previews, and community insights to surface exactly what you need, when you need it.

Final P1 Functional Commitments

- User Login/Logout: Users can register, log in, and log out securely. Session management ensures access to personalized features.
- Users can filter results by categories: Academic Assistance, Productivity, Career Development, Mental Health Support, and Creative Applications.
- Registered users can save favorite tools and build a personal watch list.
- Registered users can submit new tool entries, including use cases, features, and pricing.
- Users can rate and write reviews for any listed application.
- Users can view their personal search history and quickly repeat past searches.
- Every saved review and new tool submission generates a unique reference ID for easy tracking.
- Admins can approve or remove user-submitted tools and reviews.

URL to product: https://www.nonamesearchengine.com/

Unique Feature: Tool of the Day

We're rolling out a new "Tool of the Day" on our homepage. Each morning you'll find one AI tool featured front and center, picked by popularity, rotating through categories, or even chosen at random. Why you'll love it:

- It gives you a fresh reason to stop by every day.
- It helps you discover something new when you're not sure what to try.
- It adds a personal touch to your homepage.

2) Usability test plan

Test objectives:
We're testing the search function because that is the core function of our product. We believe that as a search engine, our product should be able to excel at its searching capabilities more than its other functions. We want to ensure that the searching is effective, efficient, and satisfactory first before we look at other functions and do the same.

Test background and setup:
The system setup for the test is relatively simple. All that needs to be done is to goto the URL for the search engine, sign up for an account if not already done, then sign into that account. The starting point of the test is the home page that is seen after logging in. This is where the search bar can be seen and tested. The search function is intended to be used by the general public, so everyone should be able to find it usable regardless of their experience with using computers. The URL of the system to be tested is https://www.nonamesearchengine.com/. The main focus of the test is user satisfaction.

Usability Task description:
Tasks:
- Do an empty search
- Browse through all of the search results
- Do a filtered search
- Browse through all of the filtered search results
- Do a search of your choosing
- Browse through all of the results of that search

If we were to measure effectiveness we would:
- Set a time as a reasonable goal for how long we believe users should be able to complete the tasks.
- Time the testers or have the testers time themselves and report it in order for us to note what percentage of testers were able to complete the tasks within the goal time.
- Keep track of or ask testers to keep track of how many errors or obstacles they faced in completing the task and report it.

If we were to measure efficiency we would:
- Take note of the time it took for each tester to complete the task
- Take note of the amount clicks it took for each tester to complete the task
- Compare and analyze the relation between the time taken and the number of clicks of the testers

User Satisfaction:

It was easy to perform multiple searches.				
Very Hard	Hard	Neutral	Easy	Very Easy
Comments:				

It was easy to browse through search results.				
Very Hard	Hard	Neutral	Easy	Very Easy
Comments:				

It was easy to use the search filter options.				
Very Hard	Hard	Neutral	Easy	Very Easy
Comments:				

3) QA test plan
Test objectives: Test search functionalities
HW and SW setup:
Go to https://www.nonamesearchengine.com/ and create an account to log into or log into an existing account.
Feature to be tested:

- Empty search
- Lowercase/Incomplete search
- Filter search

QA Test plan:

Test number	Test description	Test input	Expected output	PASS/FAIL
1	Testing empty search on Firefox	Don’t enter any input and click on the search button	A list of 24 results	PASS
2	Testing lowercase and incomplete search on Firefox	Input “Git” and search, then input “git” and search again	Both searches show a single result for “GitHub Copilot”	PASS
3	Testing search filters on Firefox	Don’t enter any input and select an option from the “Category” dropdown	A smaller list of results showing tools of the selected category.	FAIL
4	Testing empty search on Google Chrome	Don’t enter any input and click on the search button	A list of 24 results	PASS
5	Testing lowercase and incomplete search on Google Chrome	Input “Git” and search, then input “git” and search again	Both searches show a single result for “GitHub Copilot”	PASS
6	Testing search filters on Google Chrome	Don’t enter any input and select an option from the “Category” dropdown	A smaller list of results showing tools of the selected category.	FAIL

4) Code Review:
a) By this time you should have chosen a coding style. In the report say what
coding style you chose.

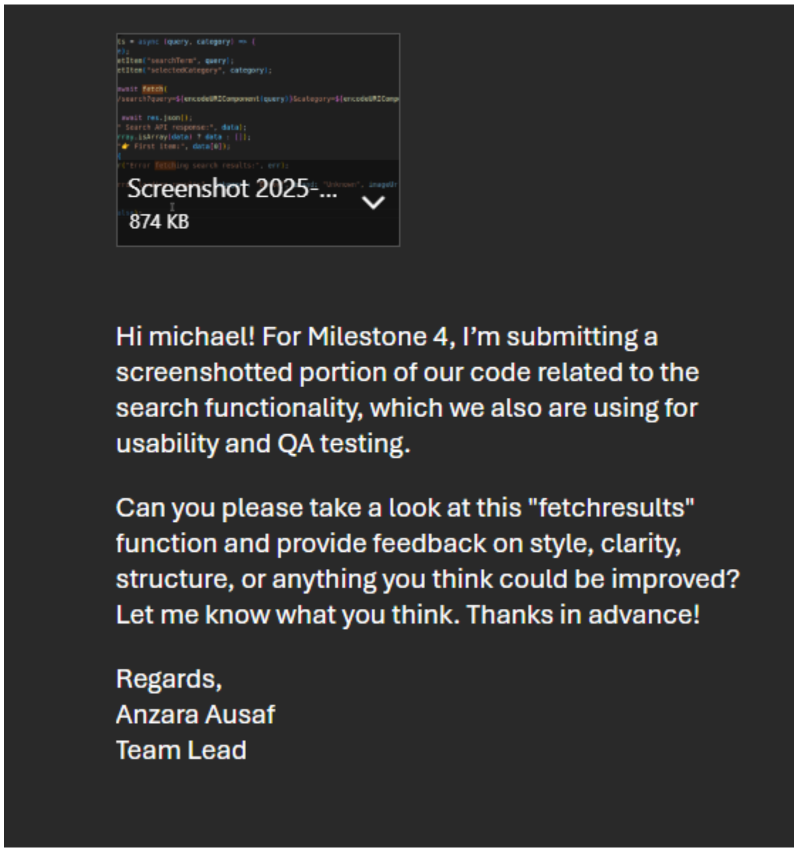
For our project, we adopted a clean, modular, and consistent coding style across both frontend and backend codebases. Specifically:

- Frontend (React + Vite + Mantine):
- Functional Components: We used modern React functional components with hooks (e.g., useState, useEffect).
- Consistent Naming: Component, function, and variable names use camelCase for clarity and readability.
- Component Reuse: Shared UI pieces like ArticleCard were modularized and reused across pages.
- Separation of Concerns: State logic (e.g., theme, user data) is handled in page components, while UI is abstracted in child components.
- Styling: We used Mantine as a component library for styling and layout, avoiding inline CSS where possible and leveraging theme-based adjustments for light/dark/custom themes.
- Backend (Fastify + Prisma):
- RESTful API Structure: Each route file handles related functionality (e.g., search.js, bookmark.js) with clear HTTP methods (GET, POST, DELETE).
- Validation: Request bodies and query strings are validated using Fastify's schema object.
- Error Handling: Meaningful HTTP status codes (e.g., 400, 404, 500) and error messages are returned for failed requests.
- Clean Queries: Prisma queries are used with readable conditions and proper logging for debugging.

b) Chose the code (substantial portion of it) related to the feature you used for
QA and usability test. You need to submit an example of the code under
review (or part of it – 2 pages or so MAX) for this function to be peer
reviewed, and document this as follows:
1. One team member should submit code to other team member(s) for peer review.
2. Peer review should be performed by other group member(s) (1 review is OK).
3. Peer review is to be done by e-mail and comments are to be included in the code
4. Submit the e-mail containing or screen shot of the peer review and commented code and
e-mail communication related to this in your Milestone 4 document
Important: It is critical that code reviews are friendly and helpful, intended to help
and education, and not to criticize. It is strongly suggested that you use peer
review in the development of the whole system. Reviewers should also check for
at least minimal code header and in-line comments and flag this as a problem if

this is not adequate

Note: peer review must include checking for basic header and in-line comments

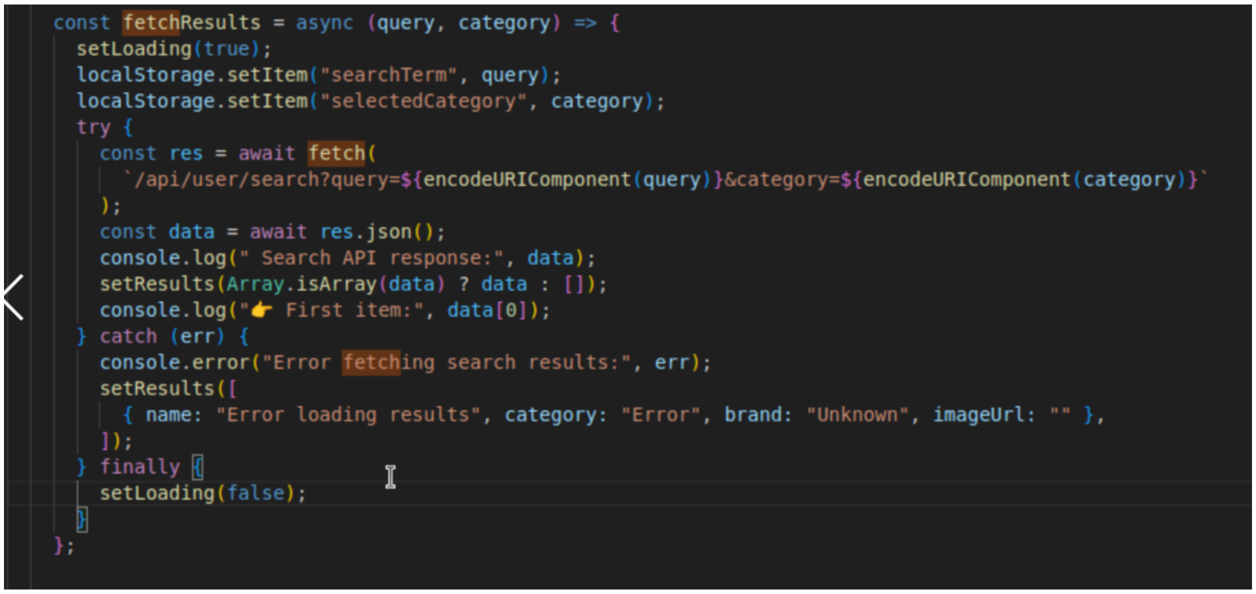


Screenshot 2025-...
874 KB

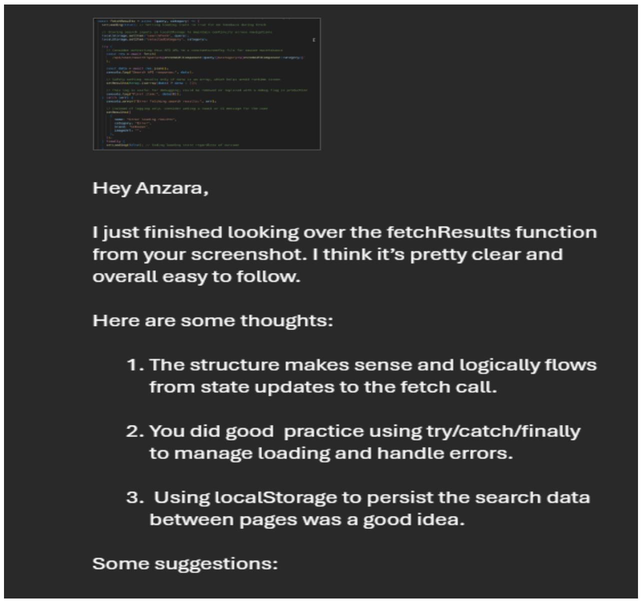
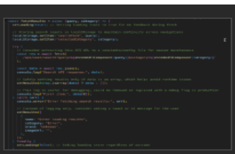
Hi michael! For Milestone 4, I'm submitting a screenshotted portion of our code related to the search functionality, which we also are using for usability and QA testing.

Can you please take a look at this "fetchresults" function and provide feedback on style, clarity, structure, or anything you think could be improved? Let me know what you think. Thanks in advance!

Regards,
Anzara Ausaf
Team Lead



```javascript
const fetchResults = async (query, category) => {
  setLoading(true);
  localStorage.setItem("searchTerm", query);
  localStorage.setItem("selectedCategory", category);
  try {
    const res = await fetch(
      `/api/user/search?query=${encodeURIComponent(query)}&category=${encodeURIComponent(category)}`
    );
    const data = await res.json();
    console.log(" Search API response:", data);
    setResults(Array.isArray(data) ? data : []);
    console.log("👉 First item:", data[0]);
  } catch (err) {
    console.error("Error fetching search results:", err);
    setResults([
      { name: "Error loading results", category: "Error", brand: "Unknown", imageUrl: "" },
    ]);
  } finally {
    setLoading(false);
  }
};
```

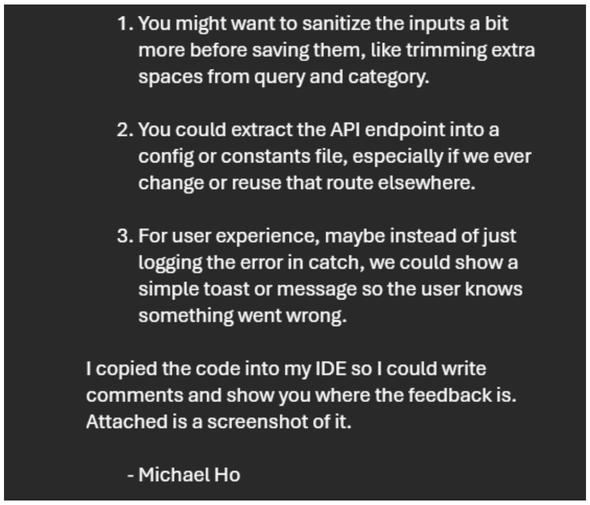
Hey Anzara,

I just finished looking over the fetchResults function from your screenshot. I think it's pretty clear and overall easy to follow.

Here are some thoughts:

1. The structure makes sense and logically flows from state updates to the fetch call.
2. You did good practice using try/catch/finally to manage loading and handle errors.
3. Using localStorage to persist the search data between pages was a good idea.

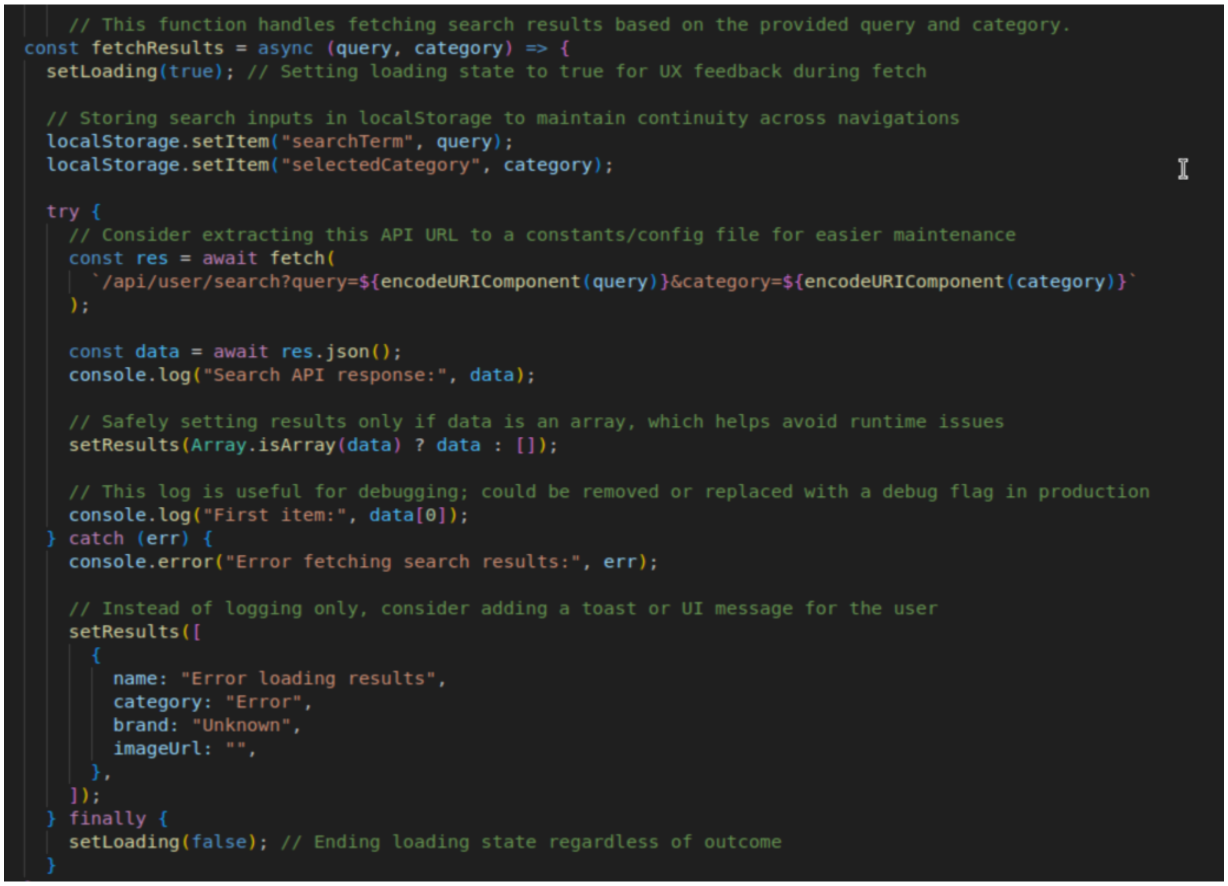
Some suggestions:



1. You might want to sanitize the inputs a bit more before saving them, like trimming extra spaces from query and category.
2. You could extract the API endpoint into a config or constants file, especially if we ever change or reuse that route elsewhere.
3. For user experience, maybe instead of just logging the error in catch, we could show a simple toast or message so the user knows something went wrong.

I copied the code into my IDE so I could write comments and show you where the feedback is. Attached is a screenshot of it.

- Michael Ho



```
    // This function handles fetching search results based on the provided query and category.
const fetchResults = async (query, category) => {
  setLoading(true); // Setting loading state to true for UX feedback during fetch

  // Storing search inputs in localStorage to maintain continuity across navigations
  localStorage.setItem("searchTerm", query);
  localStorage.setItem("selectedCategory", category);

  try {
    // Consider extracting this API URL to a constants/config file for easier maintenance
    const res = await fetch(
      `/api/user/search?query=${encodeURIComponent(query)}&category=${encodeURIComponent(category)}`
    );

    const data = await res.json();
    console.log("Search API response:", data);

    // Safely setting results only if data is an array, which helps avoid runtime issues
    setResults(Array.isArray(data) ? data : []);

    // This log is useful for debugging; could be removed or replaced with a debug flag in production
    console.log("First item:", data[0]);
  } catch (err) {
    console.error("Error fetching search results:", err);

    // Instead of logging only, consider adding a toast or UI message for the user
    setResults([
      {
        name: "Error loading results",
        category: "Error",
        brand: "Unknown",
        imageUrl: "",
      },
    ]);
  } finally {
    setLoading(false); // Ending loading state regardless of outcome
  }
```

5) Self-check on best practices for security – ½ page
Major assets we are protecting:

- **User Accounts:** Includes email, password, and name information.
- **Search Input:** The queries users enter into the search bar.
- **Bookmarked Tools:** Data tied to each user's account that must remain private and accurate.
- **Submitted Tools and Reviews:** User-generated content that is publicly displayed.

How we are protecting each asset: (1-2 lines of text per each)

- **User passwords** are hashed using bcrypt before being stored in our database to ensure secure authentication.
- **Search input** is validated on the client side to limit injection attempts and restrict unexpected characters.
- **Bookmarks** are tied to authenticated user sessions by securely storing the userId from localStorage, and server endpoints verify input data before creating/deleting records.
- **Submitted tool entries and reviews** are only accessible to registered users and are subject to basic validation (e.g., text fields, input lengths).

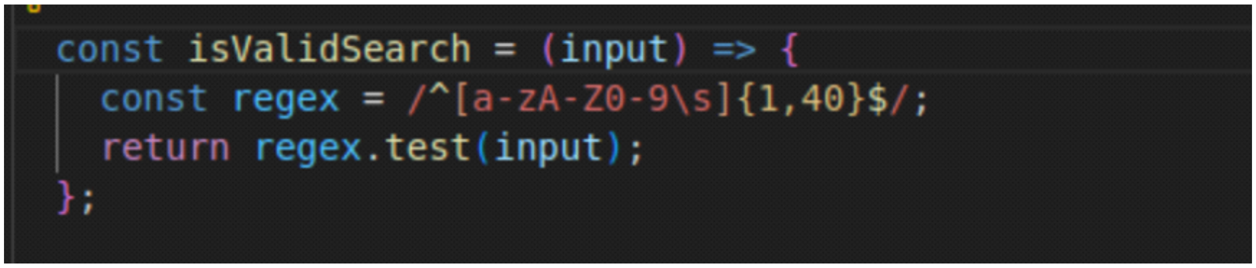
Password Encryption:

- We confirm that **passwords are encrypted using bcrypt** before being saved to the database. This is handled in our backend authentication logic (not shown here for brevity but implemented securely).

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;

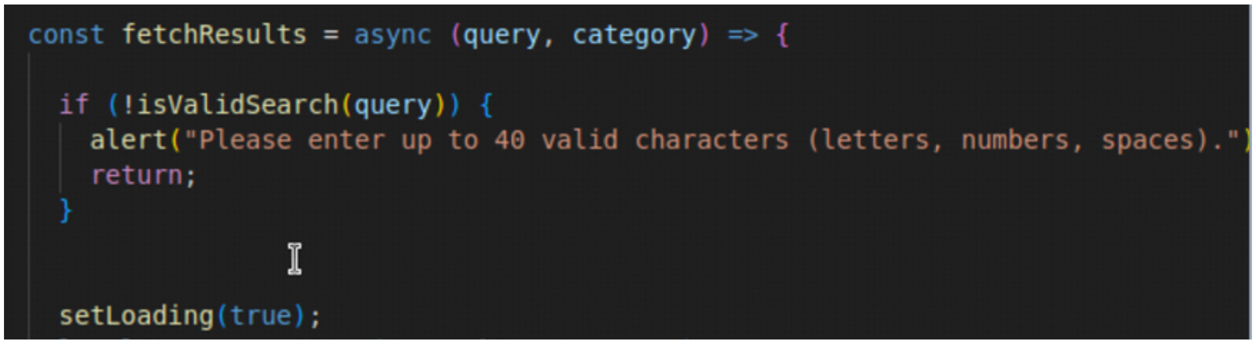
- We currently validate search bar input on the frontend by restricting it to a maximum of 40 alphanumeric characters (and spaces). This prevents accidental or malicious input from being sent to the backend.

<u>Example of our validation logic in SearchResults.jsx:</u>
//Validate search input to only allow 1–40 alphanumeric characters



```
const isValidSearch = (input) => {
  const regex = /^[a-zA-Z0-9\s]{1,40}$/;
  return regex.test(input);
};
```

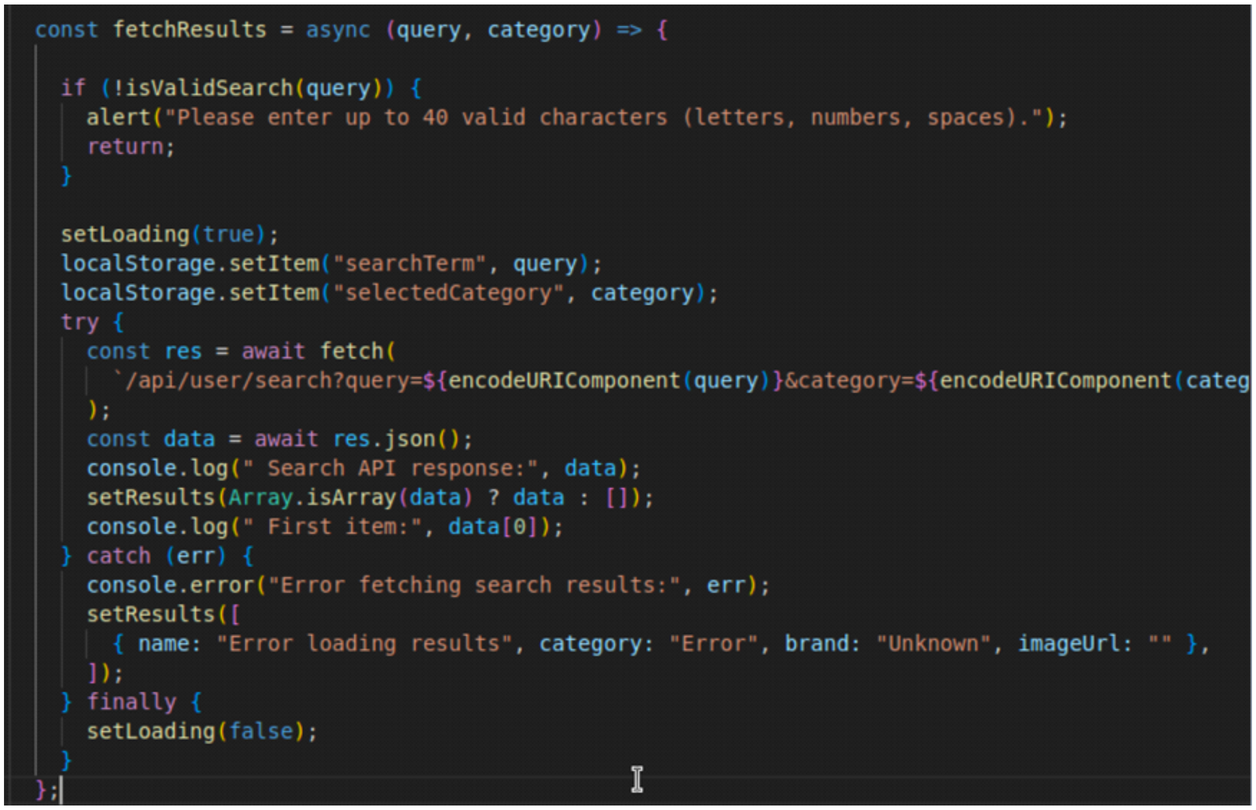
// Used before calling the search API



```
const fetchResults = async (query, category) => {

  if (!isValidSearch(query)) {
    alert("Please enter up to 40 valid characters (letters, numbers, spaces).")
    return;
  }


  setLoading(true);
```



```
const fetchResults = async (query, category) => {

  if (!isValidSearch(query)) {
    alert("Please enter up to 40 valid characters (letters, numbers, spaces).");
    return;
  }

  setLoading(true);
  localStorage.setItem("searchTerm", query);
  localStorage.setItem("selectedCategory", category);
  try {
    const res = await fetch(
      `/api/user/search?query=${encodeURIComponent(query)}&category=${encodeURIComponent(categ
    );
    const data = await res.json();
    console.log(" Search API response:", data);
    setResults(Array.isArray(data) ? data : []);
    console.log(" First item:", data[0]);
  } catch (err) {
    console.error("Error fetching search results:", err);
    setResults([
      { name: "Error loading results", category: "Error", brand: "Unknown", imageUrl: "" },
    ]);
  } finally {
    setLoading(false);
  }
};
```

We plan to expand validation to other forms and inputs across the site in future milestones, including tool submissions and reviews.

6) Self-check: Adherence to original Non-functional specs – performed by team leads
Copy all original non-functional specs as in high level application document published at the very beginning of the class. Then for each say either: DONE if it is done; ON TRACK if it is in the process of being done and you are sure it will be completed on time; or ISSUE meaning you have some problems and then explain it.
Note: you must adhere to all original non-functional specs as published in the original high level specification document. Failure to do so may cause reduced SE Product grade

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. **DONE -** Selected application functions must render well on mobile devices (this is a plus)
4. **ON TRACK -** Data shall be stored in the team's chosen database technology on the team's deployment server.
5. **ON TRACK -** Privacy of users shall be protected, and all privacy policies will be appropriately communicated to the users.
6. **ON TRACK -** The language used shall be English.
7. **ON TRACK -** Application shall be very easy to use and intuitive.
8. **ISSUE -** Google maps and analytics shall be added
9. **ISSUE -** No email clients shall be allowed. You shall use webmail.
10. **DONE -** Pay functionality, if any (e.g. paying for goods and services) shall not be implemented nor simulated in UI.
11. **ON TRACK -** Site security: basic best practices shall be applied (as covered in the class)
12. **ON TRACK -** Modern SE processes and practices shall be used as specified in the class, including collaborative and continuous SW development
13. **ON TRACK -** The website 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. (Important so not to confuse this with a real application).
14. **ON TRACK -** The AI chatbot is an optional capability (bonus requirements)

*Issue: cannot understand the reason for implementing google maps for a search engine to find AI tools or webmail over email.