SW Engineering CSC 648/848 Fall 2024

GAITOR GATE

Section 02

Team 3

Team Lead: Jared Aung (yaung2@sfsu.edu)

Front End Lead: Ulices Gonzalez

Back End Lead: Andre Dargani

Github Master & Front End Engineer: Mowtee Sailan

Front/Back End Engineer: Marco Barraza

Back End Engineer: Sergio Aguilar

Department of Computer Science, San Francisco State University

Professor. Henry Villar

April 22nd, 2025

History Table

Version	Date Submitted	Date Revised	Description
2.0	04/22/2025	4/29/2025	Initial Submission

1. Product Summary

1. **GAITOR GATE** is an intelligent and streamlined search engine designed to help users discover, filter, and engage with AI-powered software tools tailored to their needs. Unlike traditional search engines or AI directories, **GAITOR GATE** offers advanced filtering, community-based recommendations, and multimodal search input, making tool discovery faster and more inclusive. Its integration of an AI chatbot (Alli Gator) further distinguishes it by offering realtime summaries, citations, and data analysis directly within the platform. Built for students, professionals, and researchers, as well as AI developers and companies looking to showcase their tools, **GAITOR GATE** delivers a clean, customizable, and user-centric interface that simplifies the search for the right tool in a secure and efficient way.

Committed P1 Functions Deliverables

2. Users can register for an account, log in, and log out securely.
3. Users can search for tools using natural language text and keywords and filter results by category (Academic Assistance, Productivity, Career Development , platform, and publishing date.
4. Search results are displayed in a paginated format with tool thumbnails, descriptions, and links to a dedicated tool page where users can find out more information about the specific tool.
5. Unregistered users shall only be authorized to use the search functionality.
6. Logged-in users can rate a tool on a scale of 1 to 5 and leave reviews for tools and view other users' reviews.
7. The users shall be able to favorite and unfavorite tools.
8. The search query shall be added to the search history of the user. The user shall have access to their history and delete history records.
9. The search results shall be ranked by highest rated to lowest by default but the user shall be able to change the ranking in terms of the following (cost, highest reviews, recency)
10. The system shall offer similar UI/UX and different functionalities for different types of account (Student, General Public, AI Developers & Companies, Admin).

2. Usability Test Plan

- Test objectives:

Our primary objective for the usability test of our **GAITOR GATE** search engine is evaluate the satisfaction users have with its primary features in an effort to gauge its ease of use, its usefulness when filtering and categorizing results, and its effectiveness in outputting relevant results. By identifying issues within these aspects of **GAITOR GATE** we can then apply necessary changes in order to make it intuitive and enjoyable for users when using our search engine. These areas of improvement will also help inform further changes within other aspects of **GAITOR GATE** outside of the primary search capability. The findings gathered from our usability test will also help us

determine how closely **GAITOR GATE** performs compared to our use cases developed around our personas when first developing **GAITOR GATE**.

- Test background and Setup:
    - System setup

During testing, users will be using our live site which serves as our beta version of the **GAITOR GATE** search engine. The site itself is usable on all platforms whether they be PC, Laptop, and even Mobile allowing for various kinds of users to participate. The site was developed with Google Chrome in mind but is accessible using most other browsers such as FireFox, Opera, and Microsoft Edge. No software is necessary in order to run the site besides a browser like those mentioned previously. Users participating in testing will be provided direct access to the site using our URL for them to access the testing environment.

    - Starting Point

To begin testing users will begin by following the URL listed below to our **GAITOR GATE** site and arrive at our Home/Index splash page. From here users will have the capability of accessing some features granted to non-registered users which for the sake of our testing will be specified to the search. Once users arrive they will then navigate to the search page which will be empty and be set to a default size of one filter/category. During testing users will not be required to register to **GAITOR GATE** nor log in if they have a preexisting account.

    - Intended Users

Similarly to our personas the main users that will be participating in our usability test will be students part of SFSU from different backgrounds, majors, and professions.

    - URL of the system to be tested

http://18.222.76.244:8000/

    - What is to be measured

For our usability test we will be using the Likert Scale/Questionnaire in order to measure user satisfaction after they have used our Search Engine. The main aspects of the search that will be evaluated will be the filtration, categorization, and the output of search results based on a user's prompt and selected filters/categories.This will be conducted after users have used the search engine to complete set test cases for them to go through. In order to assess a user's satisfaction with the search function and its capabilities we will be measuring using a scale that ranges from very difficult to very easy gauge which will be followed by comments they wish to share.

- Usability Task description:
    - Task 1: Conducting a Search
        - Go to the search page, Enter "AI tools" into the search box, Select a filter (example: Categories) to refine the search, Select an option

from under Categories (example: “Academic Advisor”), Check the search results to see if they are relevant to your search selected filter

- Task 2: Applying Multiple Filters
  - After searching, apply an additional filter from Categories (example: "Productivity"), Next, apply the Platform filter (example: select options like web, mobile, etc.), Review how the search results change according to the filters you’ve selected
- Task 3: Rating and Reviewing a Tool
  - Choose one tool from the search results, Rate the tool on a 1 to 5 star scale, Provide a brief review of the tool, providing feedback as to whether the tool is helpful or has good features. Verify that the review and rating system is functioning/displaying correctly

Measuring Effectiveness:
Effectiveness will be measured by tracking if users can successfully complete each task (searching, filtering, rating,etc). Success will be determined by the completion rate of each task. We aim to have users complete the tasks with minimal errors or confusion.
Measuring Efficiency:
Efficiency will be measured by the time it takes to finish each task and number of steps taken. We want to reduce the task finishing time and number of total steps, which will mean a more efficient user experience and higher overall efficiency.

Questions	Very Difficult	Difficult	Average	Easy	Very Easy
How simple was it to add filters?					
Were you able to enter a search prompt without issues?					
Were you able to leave a review and rating with ease?					

3. QA Test Plan

Test Objectives:

- Validate the search engine's ability to handle various query types (keywords, specific items, non-existent items, empty queries) and return appropriate results or messages.
- Assess the registration process for successful new user creation and handling of invalid/existing user scenarios.
- Evaluate the basic responsiveness and usability of the application on mobile devices.
- Confirm pagination functionality lets users navigate through multiple result pages.

HW and SW Setup:

- **Hardware:** Macbook
- **Software:** Desktop Web Browser **(Chrome, Firefox)**, Mobile Web Browser **(Chrome, Safari)**, Server OS (**Ubuntu**).
- **URL:** http://18.222.76.244:8000/

Features to be Tested:

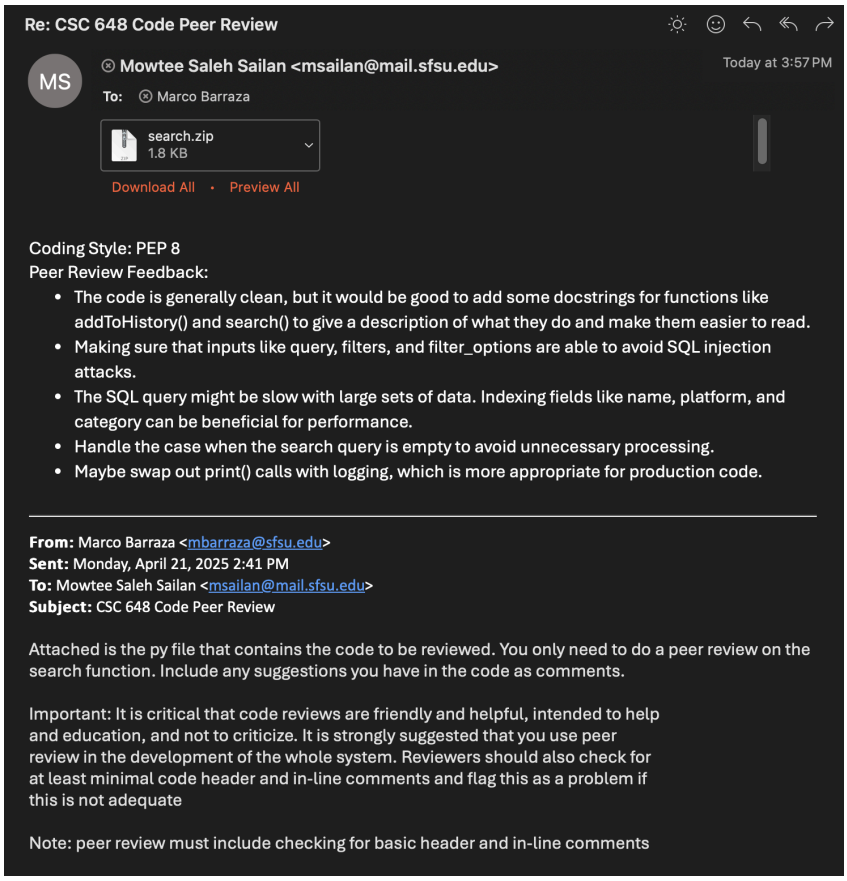
- Search Functionality (Keyword, Specific Item, Non-Existent Item, Empty Query)
- Mobile Responsiveness
- Search Results Pagination

QA Test Plan Table:

test #	test title	test description	test input	expected correct output	test results (Desktop Browser)	test results (Mobile Browser)
1	Search (Basic Keyword)	Test search with a general keyword.	Search Term: "AI"	Results related to AI should be displayed.	PASS	PASS
2	Search (Item in DB)	Test search with a specific item known to be in the database.	Search Term: "GitHub Copilot"	The specific item "GitHub Copilot" should appear in the results.	PASS	PASS
3	Search (Item Not in DB)	Test search with a term unlikely to exist in the database.	Search Term: "ItemNotInDatabaseXYZ"	A "No results found" message should be displayed.	PASS	PASS
4	Search (Empty)	Test performing a search with an empty search bar.	Search Term: "" (empty string)	Should display results (User noted it shows AI results) or a prompt to enter a search term.	PASS	PASS
5	Search Results Pagination	Verify the "Next" button on search results navigates to the subsequent page.	Perform a search yielding multiple pages. Click the "Next" pagination button.	The next page of search results is displayed correctly.	PASS	PASS
6	Mobile Responsiveness	Assess the visual layout and usability on a mobile screen size.	Load and interact with various pages (Homepage, Search, Login) on a mobile browser.	All features should be usable and layout should adapt cleanly to the smaller screen.	N/A	PASS

4. Code Review

For our project we chose PEP 8 as our coding style. PEP 8 is the official style guide for Python and is widely used in professional settings. This coding style is ideal for readability and ensures that our codebase is clean and well structured.



Re: CSC 648 Code Peer Review

MS

Mowtee Saleh Sailan <msailan@mail.sfsu.edu>

Today at 3:57 PM

To: Marco Barraza

search.zip
1.8 KB

Download All • Preview All

Coding Style: PEP 8
Peer Review Feedback:

- The code is generally clean, but it would be good to add some docstrings for functions like addToHistory() and search() to give a description of what they do and make them easier to read.
- Making sure that inputs like query, filters, and filter_options are able to avoid SQL injection attacks.
- The SQL query might be slow with large sets of data. Indexing fields like name, platform, and category can be beneficial for performance.
- Handle the case when the search query is empty to avoid unnecessary processing.
- Maybe swap out print() calls with logging, which is more appropriate for production code.

From: Marco Barraza <mbarraza@sfsu.edu>
Sent: Monday, April 21, 2025 2:41 PM
To: Mowtee Saleh Sailan <msailan@mail.sfsu.edu>
Subject: CSC 648 Code Peer Review

Attached is the py file that contains the code to be reviewed. You only need to do a peer review on the search function. Include any suggestions you have in the code as comments.

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

```python
def addToHistory(idAccount, query_text):
    # Consider adding a docstring to help explain the purpose of this function
    # and while ensuring the inputs prevent sql injection
    conn = current_app.config["MYSQL"].connection
    cursor = conn.cursor()

    result = cursor.execute(
        """
        INSERT INTO Search_History
        (idAccount, query_text)
        VALUES (%s, %s)
        """,
        (idAccount, query_text),
    )

    conn.commit()
    # Maybe replace this print with a logging statement
    print("result:", result)


@search_bp.route("/search", methods=["GET", "POST"])
def search():
    # add to search history if logged in
    # Ensure that the search query isn't empty and handle it properly
    idAccount = current_user.get_id()
    if idAccount is not None:
        query_text = request.form.get("search", "").strip()
        addToHistory(idAccount, query_text)

    with current_app.app_context():
        conn = current_app.config["MYSQL"].connection
        cursor = conn.cursor(MySQLdb.cursors.DictCursor)

        filters = request.form.getlist("filters[]")
        filter_options = request.form.getlist("filter-options[]")
        query = request.form.get("search", "").strip()
        page = request.args.get("page", 1, type=int)

        filter_pairs = sorted(list(zip(filters, filter_options)))
        print(filter_pairs) # Could replace this with a logging

        where_clauses = []
        params = []
```

```python
    for i in range(len(filters)):
        selected_filter = filters[i].strip()
        filter_option = filter_options[i].strip()

        if selected_filter == "categories" and filter_option:
            where_clauses.append("c.name = %s")
            params.append(filter_option)
        elif selected_filter == "platform" and filter_option:
            where_clauses.append("p.name = %s")
            params.append(filter_option)
        elif selected_filter == "publishing date" and filter_option:
            where_clauses.append("YEAR(d.published_date) = %s")
            params.append(filter_option)

    search_condition = """
        %s = '' OR
        MATCH(t.name) AGAINST (%s IN NATURAL LANGUAGE MODE) OR
        MATCH(k.name) AGAINST (%s IN NATURAL LANGUAGE MODE)
    """
    where_clauses.append("(" + search_condition + ")")
    params.extend([query, query, query])

    sql = """
        SELECT
            si.idIndex,
            t.idTool,
            t.description,
            t.name,
            t.company,
            t.url,
            t.thumbnail_url,
            t.published_date,
            t.pricing,
            t.version,
            c.name AS category,
            p.name AS platform,
            AVG(r.rating) AS average_rating
        FROM SearchIndex si
        JOIN Tools t ON si.idTool = t.idTool
        LEFT JOIN Category c ON si.idCategory = c.idCategory
        LEFT JOIN Platform p ON si.idPlatform = p.idPlatform
        LEFT JOIN Keywords_Indexes ki ON ki.IndexID = si.idIndex
        LEFT JOIN Keywords k ON ki.keywordID = k.idKeywords
        LEFT JOIN Rating r ON si.idIndex = r.idIndex
        WHERE {}
        GROUP BY si.idIndex, t.idTool, t.description, t.name, t.company, t.url, t.thumbnail_url, t.published_date, t.pricing, t.version, c.name, p.name
        ORDER BY si.idIndex
    """.format(" AND ".join(where_clauses))

    cursor.execute(sql, tuple(params))
    data = cursor.fetchall()
    cursor.close()
    print(data) # replace with logging
    total_results = len(data)
    total_pages = (total_results + RESULTS_PER_PAGE - 1) // RESULTS_PER_PAGE
```

```
    offset = (page - 1) * RESULTS_PER_PAGE
    page_data = data[offset : offset + RESULTS_PER_PAGE]

return render_template(
    "searchpage.html",
    data=page_data,
    current_page=page,
    total_pages=total_pages,
    title="Results",
)
```

---

5. Self Check on Best Practices for Security

Major Assets being Protected:

1. User Passwords are hashed using a bcrypt secure hashing algorithm before being stored in the database.
2. User sessions are securely managed by Flask_Login.
3. SearchIndex table acts as a reference between searchable metadata (category, platform) and Tools table itself. This design helps modularize tool data and prevents direct manipulation of sensitive information by separating core data from the search layer
4. Sensitive data such as tools data, reviews and ratings are stored in well-normalized and access-controlled tables to prevent unauthorized modifications.
5. Each user information, their search history, and their activities are tied to their authenticated session. Only logged-in users can view or delete their own records.
6. Parameters like %s are placeholders that are properly escaped and sanitized by the database driver, preventing malicious inputs from altering the SQL structure

Input Data Verification

1. Information required for registering accounts are verified by
    a. Username, password and email cannot be empty.
    b. Username has to be unique.

        cursor.execute('SELECT * FROM Account WHERE username = %s', (username,))

        check_username = cursor.fetchone()

    c. One email can create one and only one account.

        cursor.execute('SELECT * FROM Account WHERE email = %s', (email,))
        check_email = cursor.fetchone()

    d. Email is verified to have a proper format.

        re.match(r'[^@]+@[^@]+\.[^@]+',email)

    e. Username is verified to have only letters and numbers.

        re.match(r'^[A-Za-z0-9]{4,20}$',username)

    f. Passwords are required to be entered twice to prevent user mistakes.

g. All routes that require user-specific actions (such as submitting reviews, ratings, and viewing personal history) are protected using the @login_required decorator provided by Flask-Login.
h. Query is verified to have less than 40 alphanumeric characters.

```
re.match(r'^[A-Za-z0-9]{0,40}$',query)
```

6. Self Check: Adherence to Original Non-Functional Specs

- The application shall be developed, tested, and deployed using tools and servers approved by Class CTO and as agreed in M0 (some may be provided in the class, some may be chosen by the student team but all tools and servers have to be approved by Class CTO). : DONE
- The application shall be optimized for standard desktop/laptop browsers e.g., must render correctly on the two latest versions of two major browsers : DONE
- Selected application functions must render well on mobile devices (this is a plus) : ON TRACK
- Data shall be stored in the team's chosen database technology on the team's deployment server. : DONE
- The privacy of users shall be protected, and all privacy policies will be appropriately communicated to the users. : ON TRACK
- The language used shall be English. : DONE
- The application shall be very easy to use and intuitive. : DONE
- No email clients shall be allowed. You shall use webmail. : DONE
- Pay functionality, if any (e.g. paying for goods and services) shall not be implemented nor simulated in UI. : DONE