Report on the Website's Security, Quality and Usability

Name: Anying Hu

Student Number: 23017542

URL on OpenShift:

https://c23017542-cmt-120-cw-2-git-annlab.apps.containers.cs.cf.ac.uk

Cardiff University

10th January 2024

Contents

1 Introduction

In this web application development project, my website encompasses multiple features, one of which is the Projects page which enables users to dynamically filter projects based on various criteria such as industry, location, and year. Additionally, the website is equipped with a secure user registration and login system. In this report, I will detail the implementation of these two pivotal features, assess their performance, and discuss potential areas for improvement.

2 Filter Functionality on the Projects Page

2.1. Implementation Logic of the Filter

On my Projects page, the filter is a core functionality that permits users to dynamically display projects according to specific standards, such as industry, location, and year.

This feature's implementation relies on JavaScript and AJAX in interaction with the backend Flask service. In JavaScript, by capturing the submit event, I store the filter parameters in currentFilterParams and pass them to the loadProjects function. Subsequently, this function constructs an AJAX request using the Fetch API, and finally, the project list on the page is updated in real-time with the returned project data. This means users can seamlessly filter projects without interruption from page reloads. This instant response mode of interaction aligns with the expectations of modern web applications. The use of AJAX makes the page updates faster and provides immediate feedback.

However, this method essentially hands over a large amount of data for the frontend to render. If the actual project database is substantial, it will consume significant memory and CPU resources. This could potentially lead to a slow response or even a temporary non-response for devices with limited performance.

For this risk, I believe that the code logic can be optimized to minimize the amount of data returned by the backend each time. Alternatively, an infinite scrolling approach could be adopted, similar to Google's image page, where more data is loaded as the user scrolls to the bottom of the page, rather than loading all data at once.

2.2. Dynamism

The filter options are not hard-coded in the frontend; they are fetched in real-time from the database. This means that the filter content can reflect the database's real-time state, greatly facilitating database updates and maintenance. For example, when new project types or locations are added to the database, the filter automatically updates, eliminating the need for manual frontend code changes.

However, this places high demands on database entry accuracy. For instance, if the project year 2022 were mistakenly entered as 20222, the filter options would automatically include an incorrect option of 20222, which cannot be manually adjusted. Therefore, how to cooperate with database functionality is a direction for future improvement.

2.3. Interface Logic Between the Filter and the Paginator

Another significant feature on this page is the pagination function. It is interconnected with the filter functionality, and I have handled it as follows:

(1) Pagination Logic Reset: When filter conditions change, I need to reset the pagination state. Therefore, I call the loadProjects function with 1 as a parameter to ensure the filtered results are displayed starting from the first page.

(2) Maintaining Filter State: When users click the pagination buttons, the current filter conditions are passed as arguments to the loadProjects function. This ensures that even when switching between different page numbers, the chosen filter conditions remain applied.

However, in actual operation, I discovered a flaw in the interface with the pagination logic. After filtering projects, although the projects correctly started displaying from the first

page according to the filter conditions, there were still four buttons for switching pages below. This indicates that the code quality is not yet optimal and further refinement of logic is needed to allow the pagination component to regenerate page buttons based on the new project data.

3 Security Validation and Protection During User Registration and Login

3.1. Password Hash Processing

To protect user passwords, I have hashed the passwords to ensure that even if the data is accessed by unauthorized individuals, they cannot decipher the original passwords. In the Flask application, when users submit the registration form, I use the generate_password_hash function provided by werkzeug.security to generate the hash value of the password and store it in the database.

But even with such measures, compared to websites in real life, my project still needs to elevate its security, for instance, by adding account lock mechanisms to prevent brute force attempts or providing two-factor authentication.

3.2. Flask-WTF Form Validation

I have utilized Flask-WTF to handle user form submissions. It has built-in validation capabilities that streamline the process of validating user inputs and avoid repetitive code. Besides, it also offers Cross-Site Request Forgery (CSRF) protection. Whenever I create a form, Flask-WTF automatically incorporates a hidden CSRF token field. This ensures that each form submission is initiated from my site, thus preventing external attackers from forging requests.

Appendix A

1. **User Authentication and Account Management:**

- Flask Sessions: Used for managing user sessions to keep track of logged-in users.
- Werkzeug Security: Utilized for hashing and verifying user passwords for secure authentication.
- SQLite Database: Stores user account information and handles queries for registration and login.
- Flask-WTF: Used for creating and validating user registration and login forms.

2. **Dynamic Content Rendering with Flask and Jinja2 Templates:**

- Flask with Jinja2 Templating: Renders dynamic HTML content and injects server-side data into web pages.
- HTML and CSS: Structuring and styling the web pages.

3. **Database Interaction:**

- SQLite database integration for storing and retrieving data (users, projects, comments).
- Use of SQL queries to interact with the database, including insertion, deletion, and selection of records.
- Establishing relationships between different database tables (e.g., users and comments).

4. **Advanced Web Form Handling (Flask-WTF):**

- Implementation of web forms using Flask-WTF, which includes CSRF protection.
- Custom validation for form fields (e.g., username validation with regular expressions).

5. **Dynamic Filtering and Pagination for Projects:**

- AJAX requests handling for dynamic content loading without page refresh.
- Implementation of filters (industry, location, year) for the projects section.
- Pagination system for projects to manage the display of a large number of entries.

6. **File Handling and Image Management:**

- Python OS Module: Manages file paths and directories for project images.
- Functionality to handle and serve image files related to projects.
- Sorting and selecting specific file types from directories (e.g., filtering .jpg and .png files).

7. **Front-End Interaction with JavaScript:**

- Use of JavaScript to enhance the interactivity of the web pages (e.g., handling filter form submission, pagination).
- AJAX for asynchronous data fetches, improving the user experience by avoiding full page reloads.

8. **Responsive Web Design Considerations:**

- JavaScript: Handling client-side logic, form submissions, and interaction with the Flask backend.

- Flask: Seamlessly integrates with front-end components to ensure functionality and user experience.

9. Security Measures (Hashing):

- Flask-WTF CSRF Protection: Protects forms against Cross-Site Request Forgery attacks.
- Implementation of basic security measures like password hashing and session management.

10. Rating and Comment System for Projects:

- Ability for users to post comments and rate projects.
- Calculation and display of average ratings for projects.

11. Environment and Dependency Management

- Python and Flask Environment Setup: Configures the development and runtime environment for the application.
- Pip and Requirements.txt: Manages and installs project dependencies.

Data Analytics and Visualization

- Python's Built-in Functions: For aggregating and calculating average ratings and other data analytics tasks.