Narrative:
Give five functional and non-functional requirements for a web app that gives an individual's contributions to a project over time if given their github repository and username. The user can log into the web app to save their previous search history into a database that can be later used to quickly use the search feature on the web app. Use HTML front-end, flask, SQLAlchemy and the Github API.

Functional Requirements:

Authentication: The web app should provide a secure login and registration system to allow users to create and manage their accounts. This can be achieved by implementing OAuth or any other secure authentication method.

Repository and User Search: The web app should allow users to search for a specific Github repository and username. The search function should make use of the Github API to fetch relevant information about the specified user and their contributions to the selected repository.

Contribution Visualization: The web app should display the user's contributions to the specified repository over time, using an intuitive chart or graph to visualize the data. This may include information like number of commits, lines of code added/removed, issues created/closed, and pull requests submitted.

Search History: The web app should save the search history of logged-in users in the database using SQLAlchemy. This history should be easily accessible to the user, allowing them to quickly view and re-run previous searches.

Export Functionality: The web app should provide an option to export the user's contribution data in various formats, such as CSV or JSON, for further analysis or reporting.

Non-Functional Requirements:

Usability: The web app should have an intuitive and easy-to-use user interface, with clear navigation and well-organized content. HTML and CSS should be used to create a responsive design that works well on various devices and screen sizes.

Performance: The web app should be optimized for fast load times and smooth performance, ensuring that user interactions are responsive and that API calls are efficiently handled.

Scalability: The web app should be built using a modular approach, allowing for easy expansion of features and functionality in the future. The Flask framework should be utilized to ensure that the application is scalable and can handle increased traffic and demand as needed.

Security: The web app should follow industry best practices for web security, including protecting user data and ensuring secure communication between the client and server. This includes proper handling of authentication, session management, and data storage.

Maintainability: The web app should be developed using clean, well-structured code with proper documentation and comments. This will make it easier for developers to maintain, update, and troubleshoot the application in the future.