MIAMI

CSC 431

Glow Getter Web Application

System Architecture Specification (SAS)

Team 8

Talia Berler	Project Manager
Aaliyah Brown	Developer
Nimmi Suri	Developer

Version History

Version	Date	Author(s)	Change Comments
1.0	4/5/24	Talia Berler, Aaliyah Brown, Nimmi Suri	First Draft

Table of Contents

Table of Figures

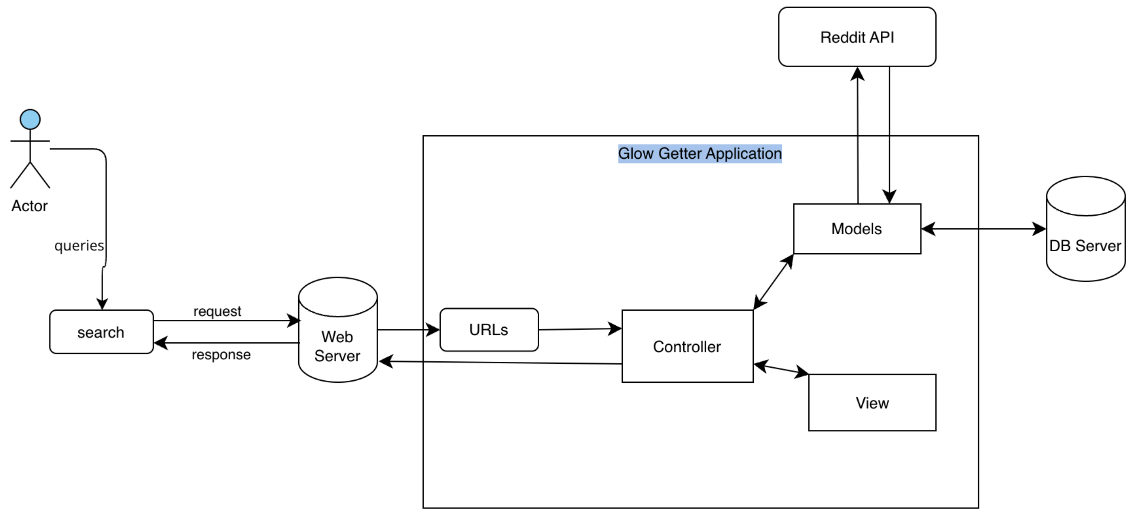
1. System Analysis

1.1. System Overview

GlowGetter, employing the Model-Controller-View (MCV) architectural style, caters to companies in the commercial beauty products sector, particularly beauty brands and firms focused on customer satisfaction. The Model encapsulates backend data and logic, utilizing Python alongside web scraping libraries like Matplotlib and PRAW for data extraction, sentiment analysis through VADER, and MySQL for data storage. The Controller manages application logic such as brand monitoring, customer feedback aggregation, and competitor analysis. Meanwhile, the View provides an intuitive web interface accessible via any internet-connected device, featuring CSS-designed UI for enhanced user experience. GlowGetter enables companies to monitor brand performance, gather customer feedback, and analyze market trends, while providing valuable insights into beauty products for both companies and consumers.

1.2. System Diagram

Figure 1: Glow Getter System Diagram



1.3. Actor Identification

There are two types of human actors: registered users and unregistered users. Registered users may upload makeup product or brand reviews directly to the site and access the search functionality. Unregistered users may access only the search functionality. The system will also support interaction with non-human actors, specifically the Reddit API and external Python library PRAW, which will be used to scrape information from Reddit threads and transfer that information to the database server.

1.4. Design Rationale

1.4.1. Architectural Style

This system is a web application with a frontend UI and a backend logic and server. As such, we are designing the application using the Model-View-Controller architectural style. This style will allow us to separate the user view from the data behind the make-up product/brand reviews.

Model: Represents the underlying data and business logic of the application. This includes data related to makeup products and brands, as well as the algorithms and processes for managing reviews and feedback.
View: the user interface components of the application. Through the view, users interact with the system, accessing and manipulating makeup product and brand information.
Controller: Acts as an intermediary between the model and the view. It processes user input, triggers appropriate actions within the model, and updates the view accordingly. By encapsulating the application's logic in controllers, we ensure a clear separation of concerns and maintainability of the codebase.

1.4.2. Design Pattern(s)

We predict that the Factory pattern will be primarily used in our project. In our application, the Model component is responsible for representing the underlying data and business logic. This includes managing makeup products, brands, reviews, and feedback. Using the Factory Method pattern within the Model allows us to encapsulate the creation of these objects, providing a centralized mechanism for creating an object class while keeping the details of object creation hidden from the client code.

By defining a factory method within the Model component, we can delegate the responsibility of creating specific types of objects (e.g., makeup products, brands) to subclasses or specialized factories.

Additionally, by adhering to the Factory Method pattern, we are adhering to the principle of "separation of concerns," where the process of object creation is decoupled from the client code that uses these objects. This enhances maintainability and allows for easier modification or extension of the object creation process in the future.

1.4.3. Framework

The web application will run on the Django framework. The Python-based framework will facilitate access to the Reddit API and implementation of the MVC structure of the application. Django supports features for frontend development, so the UI can be built without integrating another frontend software. Additionally, Django allows for seamless integration with the MySQL database for storage of information in the backend.

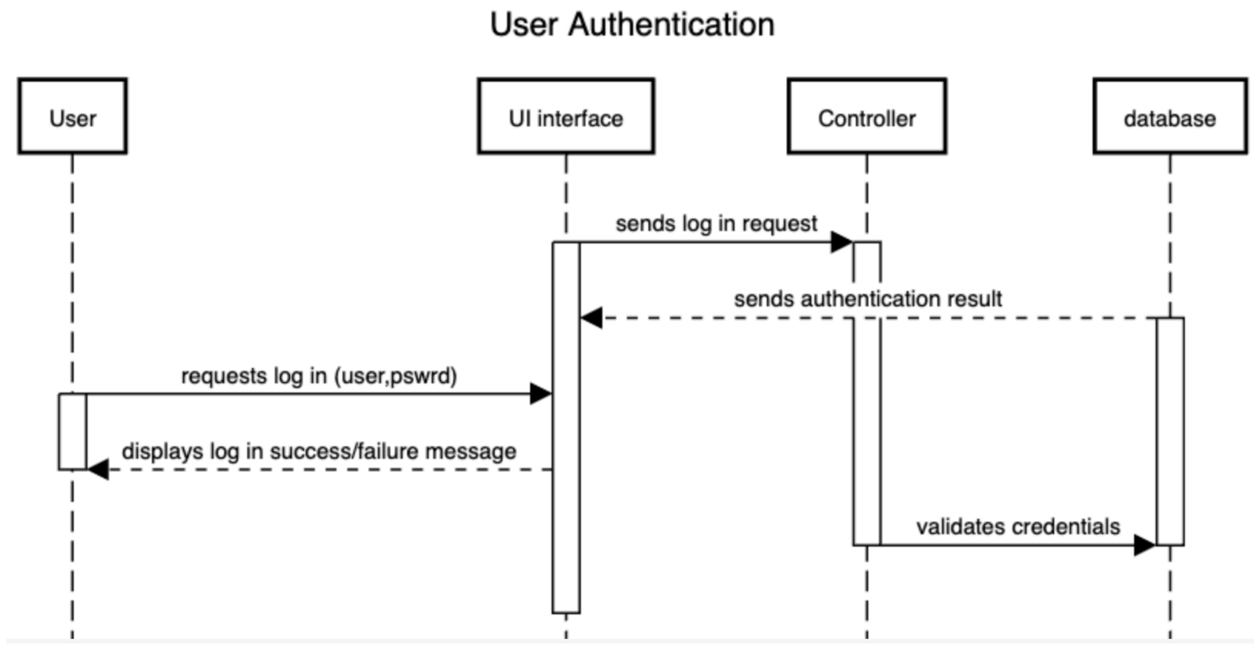
Links:

- https://www.djangoproject.com/

- https://www.mysql.com/

2. Functional Design

2.1. User Authentication


User Authentication
User
UI interface
Controller
database
sends log in request
sends authentication result
requests log in (user,pswrd)
displays log in success/failure message
validates credentials

User Requests Login: The sequence begins when the User requests to log in to the system by interacting with the UI interface component.

UI interface Sends Login Request: The UI interface forwards the login request to the Controller, indicating that the User wants to authenticate.

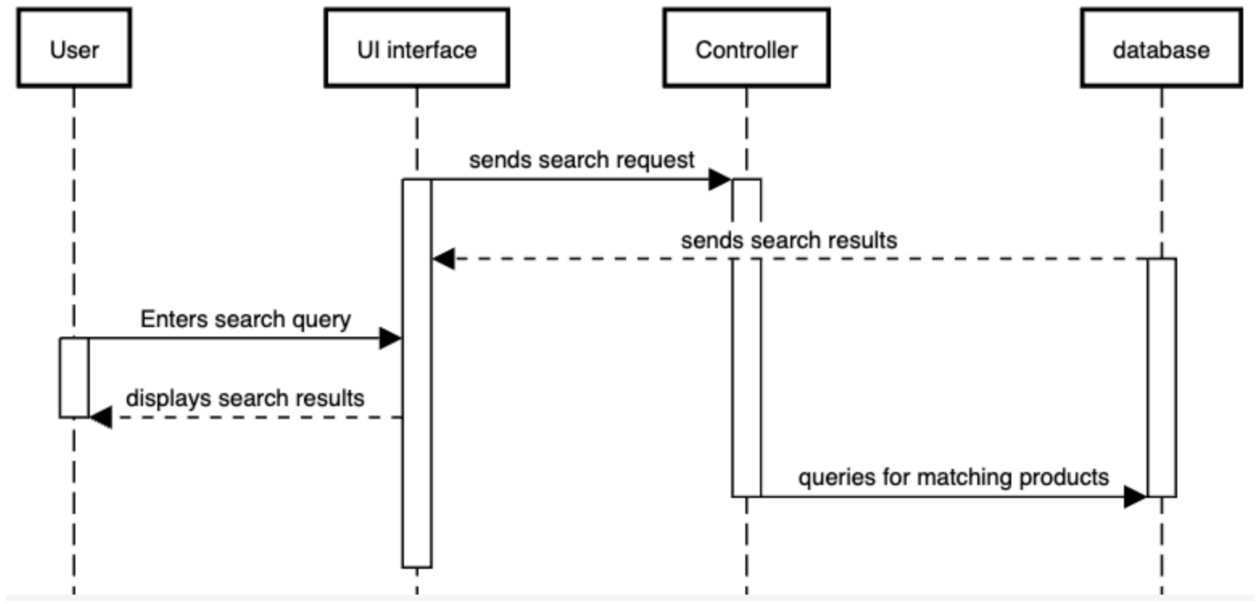
Controller Validates Credentials: Upon receiving the login request, the Controller interacts with the Database to validate the User's credentials against the stored data.

Database Sends Authentication Result: The Database processes the authentication request and sends the result (success or failure) back to the Controller.

Controller Sends Result to UI interface: The Controller receives the authentication result from the Database and forwards it to the UI interface.

UI interface Displays Result to User: Finally, the UI interface displays a message to the User indicating whether the login attempt was successful or not.

2.2 Searching for products



User Enters Search Query: The sequence begins when the User enters a search query into the UI interface component, indicating the makeup products they are looking for.

UI interface Sends Search Request: The UI interface forwards the search query to the Controller, signaling the Controller that a search operation is requested.

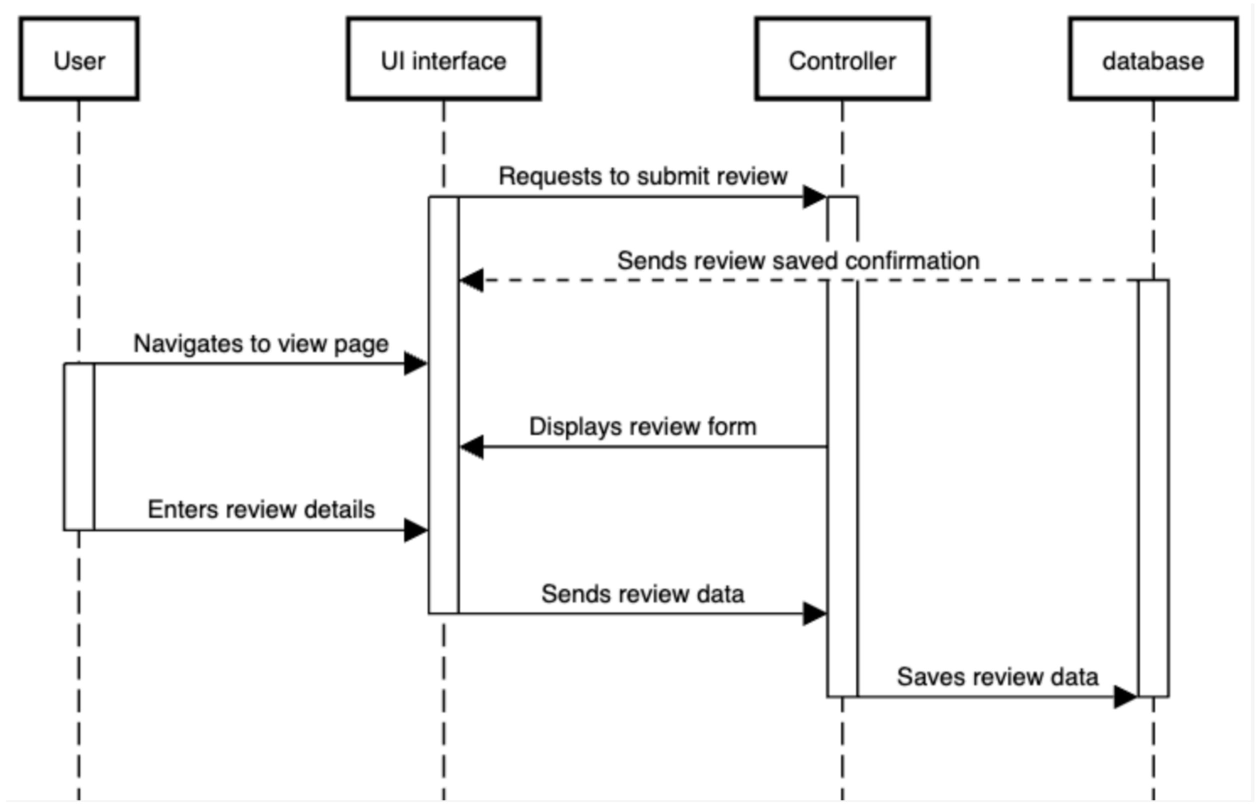
Controller Queries Database: Upon receiving the search request, the Controller interacts with the Database to query the database for makeup products matching the search query.

Database Sends Search Results: The Database processes the search query, retrieves matching makeup products from the database, and sends the search results back to the Controller.

Controller Sends Results to UI interface: The Controller receives the search results from the Database and forwards them to the UI interface .

UI interface Displays Results to User: Finally, the UI interface presents the search results to the User, displaying the makeup products that match the search query.

2.3 Submitting a product review



User Navigates to Product Review Page: The sequence starts as the User navigates to the product review page within the UI interface component, indicating the intention to submit a review for a particular product.

UI interface Requests to Submit Review: The UI interface sends a request to the Controller, requesting to submit a review for the product.

Controller Displays Review Form: Upon receiving the request, the Controller displays the review form within the UI interface, allowing the User to input their review details.

User Enters Review Details: The User enters the review details into the form presented by the UI interface, providing feedback and ratings for the product.

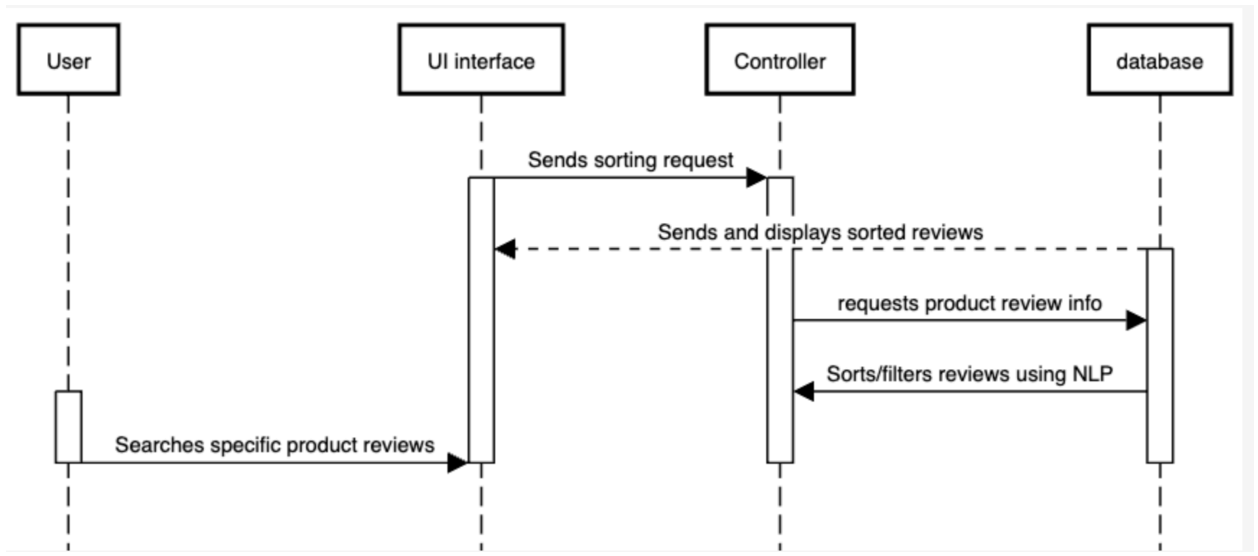
UI interface Sends Review Data: After the User submits the review details, the UI interface sends the review data to the Controller for processing.

Controller Saves Review to Database: The Controller interacts with the Database to save the review data to the database, storing the User's feedback and ratings for the product.

Database Sends Confirmation: After successfully saving the review data, the Database sends a confirmation message back to the Controller.

Controller Displays Confirmation Message: Finally, the Controller receives the confirmation from the Database and displays a confirmation message within the UI interface, notifying the User that their review has been successfully submitted.

2.4 Sorting reviews


User
UI interface
Controller
database
Sends sorting request
Sends and displays sorted reviews
requests product review info
Sorts/filters reviews using NLP
Searches specific product reviews

User Requests Sorting: The sequence begins as the User requests to sort the reviews of makeup products, specifying whether they want to view good or bad reviews.

UI interface Sends Sorting Request: The UI interface forwards the sorting request to the Controller, indicating the User's preference for sorting.

Controller Initiates Data Retrieval: Upon receiving the sorting request, the Controller interacts with the Database to retrieve the reviews from the database.

Database Filters Reviews: The Database filters the retrieved reviews based on the User's sorting preference (good or bad), selecting only the relevant reviews.

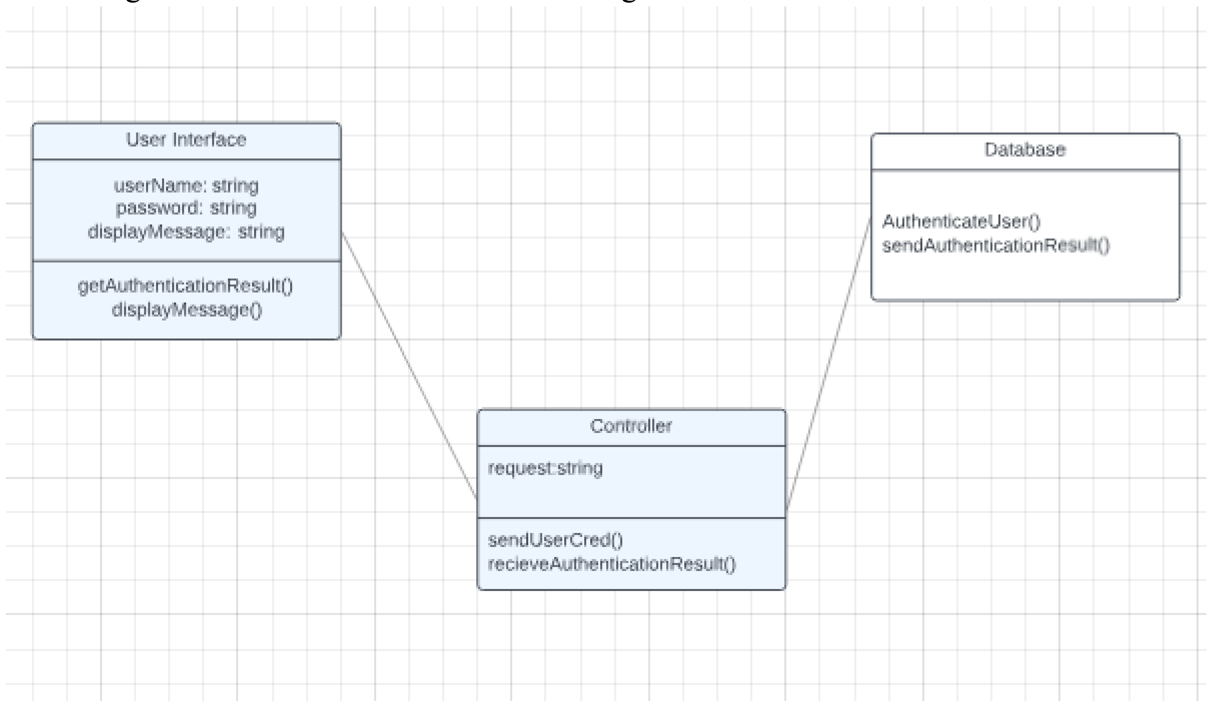
Database Sends Sorted Reviews: After filtering the reviews, the Database sends the sorted reviews back to the Controller.

Controller Sends Results to UI interface: The Controller receives the sorted reviews from the Database and forwards them to the UI interface for display.

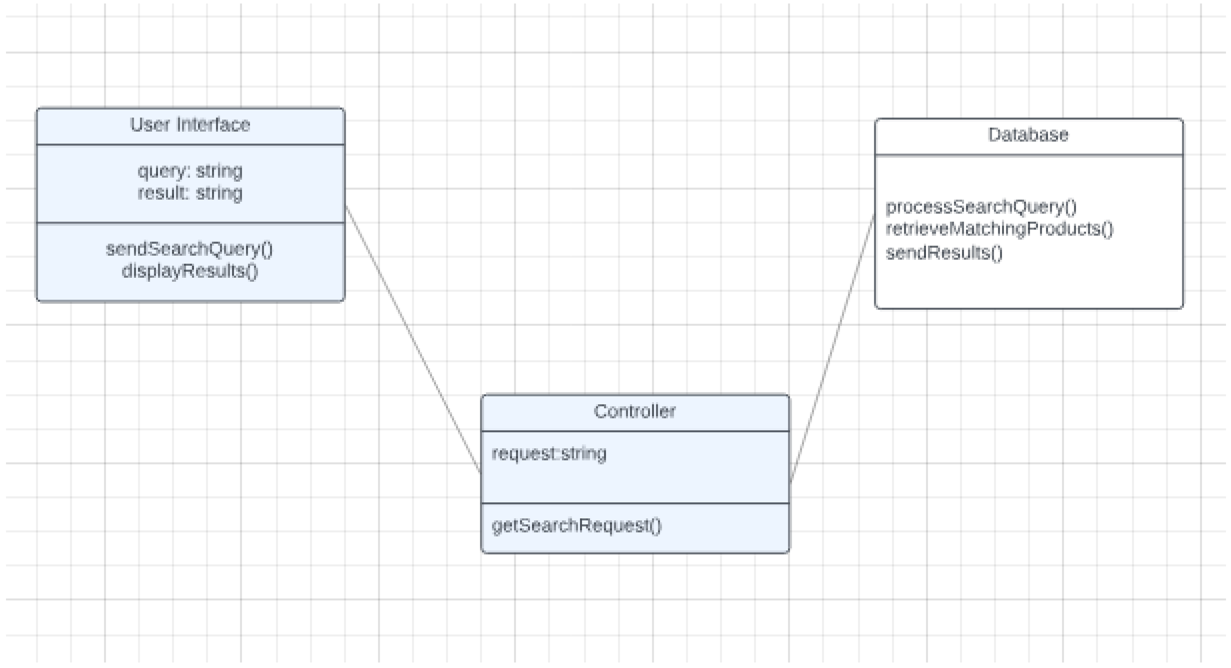
UI interface Displays Sorted Reviews: Finally, the UI interface presents the sorted reviews to the User, displaying either the good or bad reviews based on the User's preference.

3. Structural Design

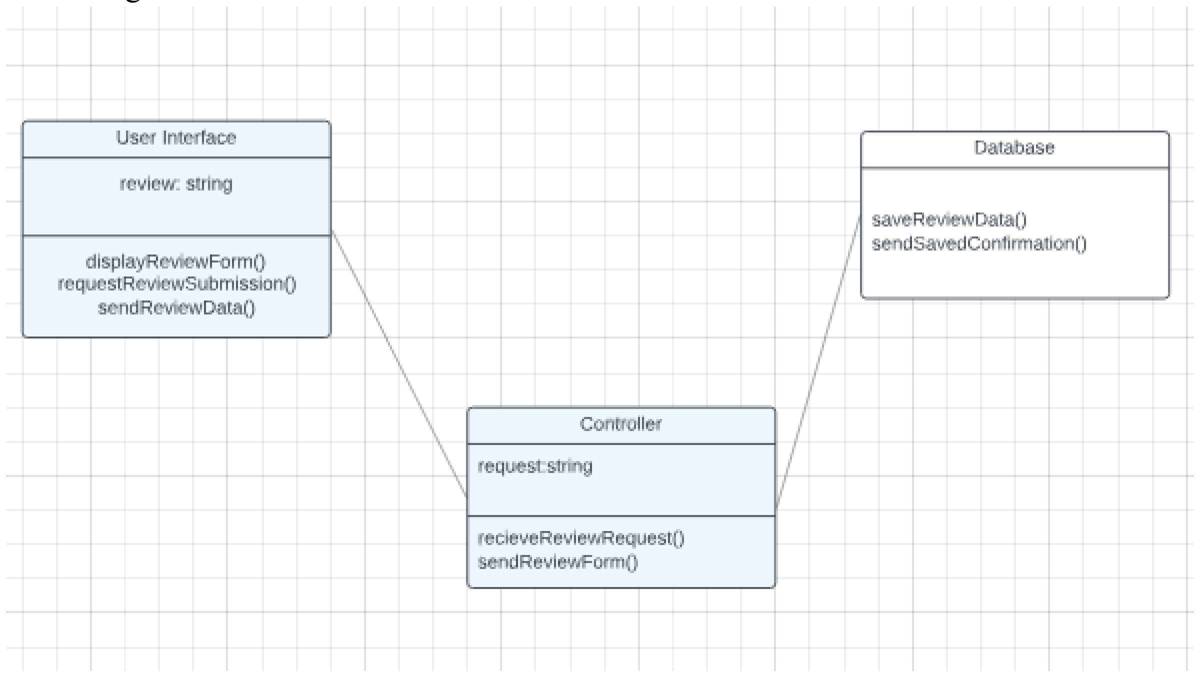
Class Diagram for the user authentication and login use case



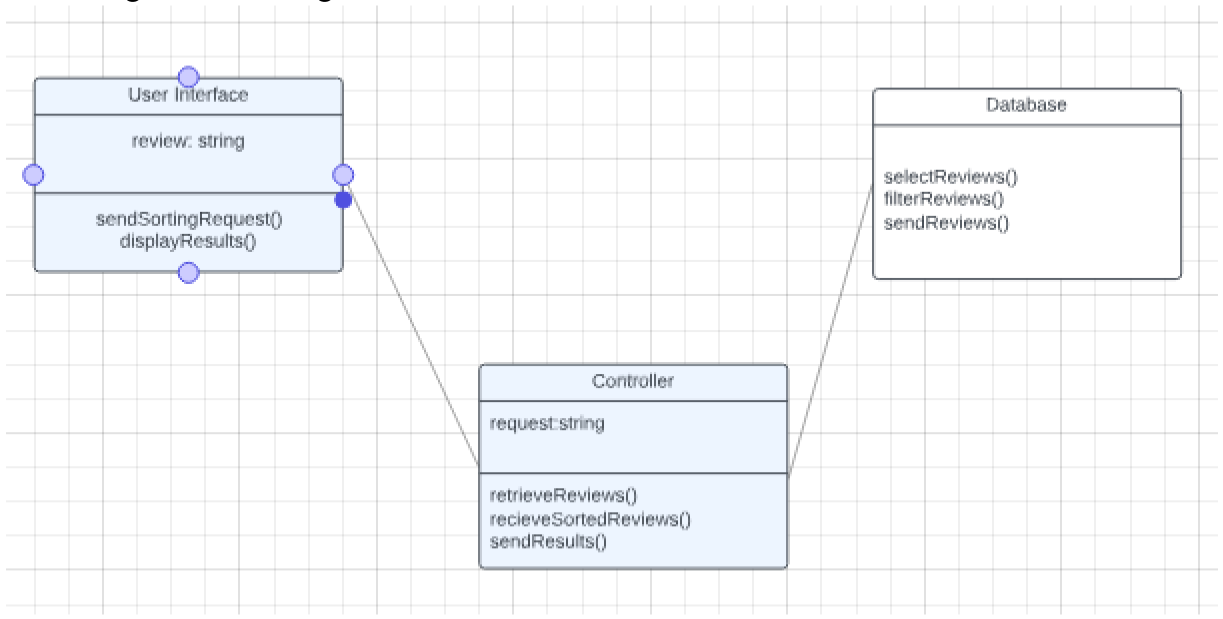
Class Diagram for product search use case



Class Diagram for review Submission



Class Diagram for sorting results


User Interface
review: string
sendSortingRequest()
displayResults()
Database
selectReviews()
filterReviews()
sendReviews()
Controller
request:string
retrieveReviews()
recieveSortedReviews()
sendResults()