SOFTWARE ENGINEERING CONCEPTS

LAB MANUAL

A MINI-PROJECT REPORT

Submitted by

DIVYASHREE S	220701070
GOPINATH R	220701076
HARIHARAVISWANATHAN P	220701082
HARINI S	220701085
HARSHINI AKSHAYA A S	220701088
JAGADEESH BASKAR	220701094

in partial fulfillment of the award of the degree of

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE AND ENGINEERING

RAJALAKSHMI ENGINEERING COLLEGE, CHENNAI
An Autonomous
Institute Thandalam
Chennai - 602105

2023-2024

INDEX

Overview of the Project

Why Restaurant Analysis?

- Solve the problem of customers struggling to find good restaurants efficiently.
- Current search methods are time-consuming and lead to subpar dining experiences.

What is Restaurant Analysis?

- Process of collecting and studying customer reviews to understand the best restaurants.
- Involves gathering feedback, calculating ratings, and identifying common themes.

How Restaurant Analysis Works:

Review Collection:
- Gather feedback from customers through website and app.
- Sort feedback as positive, negative, or neutral, and store it in a structured database.

Review Analysis:
- Calculate overall ratings and track restaurant performance over time.
- Identify common themes in feedback to understand strengths and weaknesses.

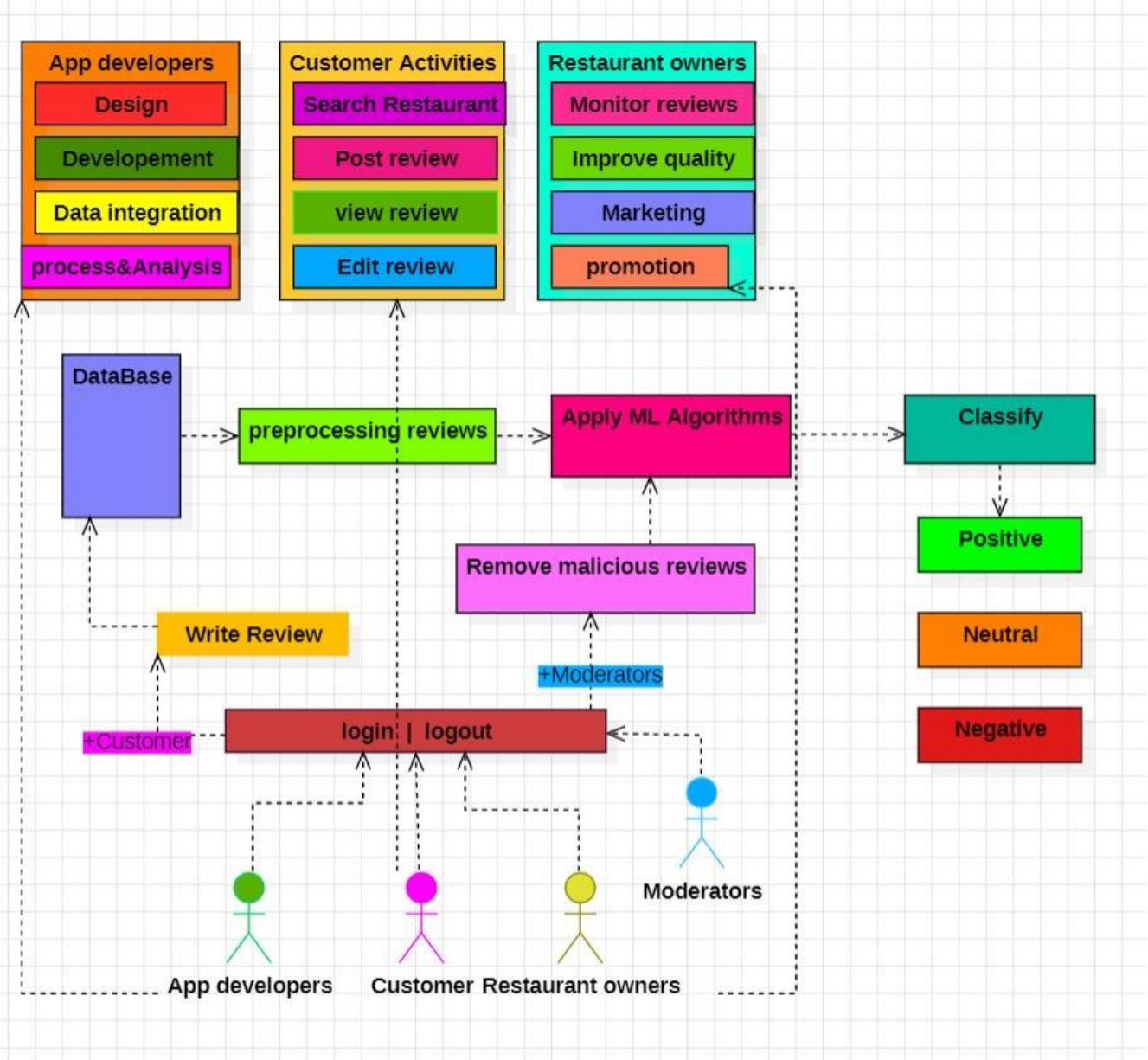
Ranking and Recommendations:
- Develop an algorithm to rank restaurants based on feedback scores, ratings, and user preferences.
- Personalize recommendations based on location and past dining history.

Efficient Search and Discovery:
- Present a ranked list of top-rated restaurants based on aggregated feedback.
- Simplify the search process, saving users time and effort.

Informed Decision-Making:
- Provide detailed restaurant profiles, including menus and recommended dishes.
- Promote restaurants with consistently high ratings for better choices.

User Engagement:
- Allow users to interact with reviews and share feedback.
- Implement loyalty programs to reward frequent users, fostering a community of food enthusiasts.

BUSINESS ARCHITECTURE



REQUIREMENTS FOR USER STORIES

FUNCTIONAL REQUIREMENTS

1. As a user,
   I want to sign up and log in
   so that I can leave reviews and manage my account.

   Acceptance criteria:
   - Users can create an account with an email and password or via social media login.
   - Users can log in with their credentials.
   - Users can reset their password if forgotten.

2. As a user,
   I want to browse through a list of restaurants
   so that I can decide where to dine.

   Acceptance criteria:
   - Users can see a list of restaurants with basic information (name, location, rating).
   - Users can filter restaurants based on criteria like cuisine, rating, distance, etc.

3. As a user,
   I want to view detailed information about a restaurant
   so that I can learn more about it.

   Acceptance criteria:
   - Users can see detailed information such as menu, opening hours, contact details, and reviews.
   - Users can view the overall rating and the count of positive and negative reviews.

4. As a user,
   I want to submit a review for a restaurant
   so that I can share my experience with others.

   Acceptance criteria:

- Users can write and submit a text review.
- Users can upload photos with the review.

5. As a user,
   I want to read reviews for a restaurant
   so that I can make an informed decision.

   Acceptance criteria:
   - Users can see a list of reviews with ratings and comments.
   - Users can filter reviews to see only positive or only negative ones.

6. As a restaurant owner,
   I want to claim my restaurant's profile
   so that I can manage the information and reviews.

   Acceptance criteria:
   - Owner can request to claim a restaurant profile by providing proof of ownership.
   - Owner can edit the restaurant's information once the claim is verified.

7. As a restaurant owner,
   I want to respond to reviews
   so that I can engage with customers and address their feedback.

   Acceptance criteria:
   - Owner can reply to individual reviews.
   - Owner's responses are visible to all users.

8. As a restaurant owner,
   I want to view statistics about my restaurant's reviews
   so that I can understand customer feedback.

   Acceptance criteria:
   - Owners can see the count of positive and negative reviews.
   - Owners can see trends over time (e.g., weekly/monthly review counts).
   - Owners can download review statistics as a report.

9. As an admin,
I want to manage users and reviews
so that I can ensure the platform remains trustworthy and high-quality.

Acceptance criteria:

- Admin can view and edit user profiles.
- Admin can delete inappropriate reviews.
- Admin can ban users who violate terms of service.

10. As an admin,
I want to access a dashboard that shows the count of positive and negative reviews for each restaurant
so that I can monitor platform activity.

Acceptance criteria:

- Admin can see a list of all restaurants with their respective counts of positive and negative reviews.
- Admin can export the dashboard data.

NON-FUNCTIONAL REQUIREMENTS

1.Performance:

- The website should load within 2 seconds for the homepage and 3 seconds for restaurant detail pages.
- The system should handle at least 1,000 concurrent users without performance degradation.

2. Availability:

- The website should have an uptime of 99.9%, ensuring it is available to users at all times.
- The system should have automated failover mechanisms in place to handle server outages.

3. Security:

- All user data should be encrypted both in transit and at rest.
- The system should implement strong authentication and authorization mechanisms, including multi-factor authentication for users and restaurant owners.
- Regular security audits and penetration testing should be conducted to identify and mitigate vulnerabilities.

4. Usability:

- The user interface should be intuitive and easy to navigate, with clear calls to action.

5. Maintainability:

- The codebase should follow best practices for readability and documentation, including comments and clear naming conventions.
- The architecture should support modularity, allowing components to be updated or replaced with minimal impact on the system.

ARCHITECTURE DIAGRAM DEPICTING USER STORIES

SOFTWARE ARCHITECTURE USED: **MVC**

MVC architecture is chosen for its clear separation of concerns, enhancing maintainability and scalability. It enables parallel development, with models handling data storage, views managing presentation, and controllers orchestrating user interactions, making it ideal for structuring the restaurant review system across MERN stack and Flask components.

MODULES:

1. User Registration
2. Review Writing
3. Review Reading
4. Backend Review Storage
5. Review Analysis

ARCHITECTURE OVERVIEW:

USER REGISTRATION:

Model:

- MongoDB collection users
- Stores user details (username, password, email)

View:

- React components for user registration form
- Form validation and user feedback

Controller:

- Express routes for handling registration requests
- Validation and encryption of user data
- Interaction with MongoDB to store user data

Flow:

1. User submits registration form (View).
2. Form data is sent to the Express server (Controller).
3. Express validates and processes the data, then stores it in MongoDB (Model).
4. Success/failure response is sent back to the React frontend (View).

REVIEW WRITING:

Model:

- MongoDB collection reviews
- Stores review details (user ID, restaurant ID, review text, rating, timestamp)

View:

- React components for writing reviews
- Text input, rating selection, submission button

Controller:

- Express routes for handling review submission requests
- Validation and processing of review data
- Interaction with MongoDB to store review data

Flow:

1. User writes a review and submits the form (View).
2. Form data is sent to the Express server (Controller).
3. Express validates and processes the data, then stores it in MongoDB (Model).
4. Success/failure response is sent back to the React frontend (View).

REVIEW READING:

Model:

- MongoDB collection reviews
- Query for fetching reviews based on restaurant/user ID

View:

- React components for displaying reviews
- List of reviews, pagination, sorting options

Controller:

- Express routes for fetching review data
- Query MongoDB for review data based on criteria (restaurant ID, user ID, etc.)

Flow:

1. User navigates to a restaurant or profile page (View).
2. React frontend sends a request to the Express server to fetch reviews (Controller).
3. Express queries MongoDB for relevant reviews (Model).
4. Reviews are sent back to the React frontend and displayed (View).

BACKEND REVIEW STORAGE:

Model:

- MongoDB collections reviews and users
- Structured storage for all review data with appropriate indexes for fast querying

Controller:

- Express middleware for CRUD operations on reviews and users

Flow:

1. Express server handles create, read, update, and delete operations for reviews and user data.
2. Interacts with MongoDB to store, retrieve, update, and delete data as requested by the frontend or analysis server.

REVIEW ANALYSIS:

Model:

- MongoDB collection analysis
- Stores analyzed data such as overall ratings, themes, and trends

View:

- React components to display analysis results
- Graphs, charts, and trend analysis

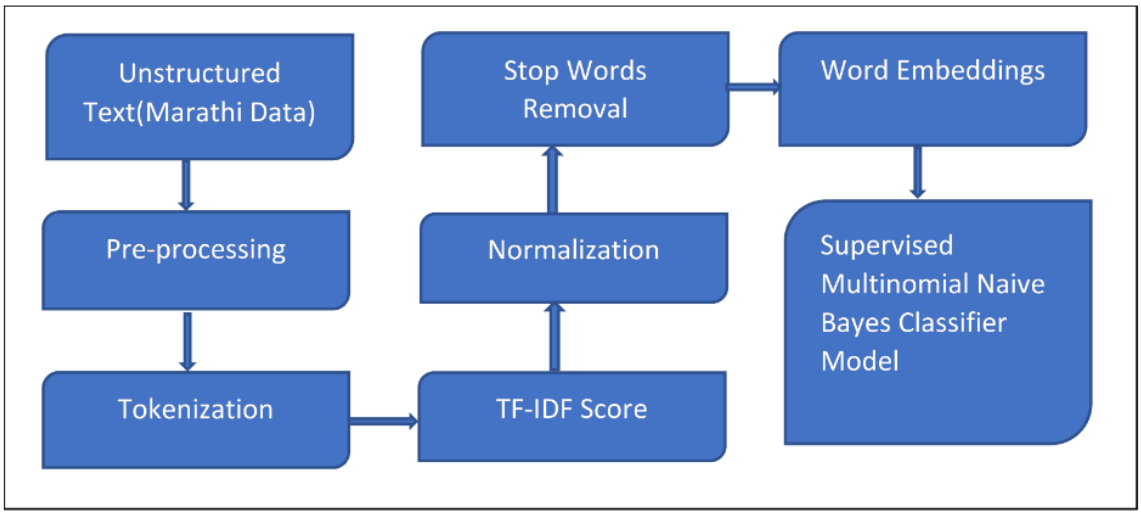
Controller:

- Express routes to fetch analysis results for the frontend
- Flask server for performing analysis on the review data

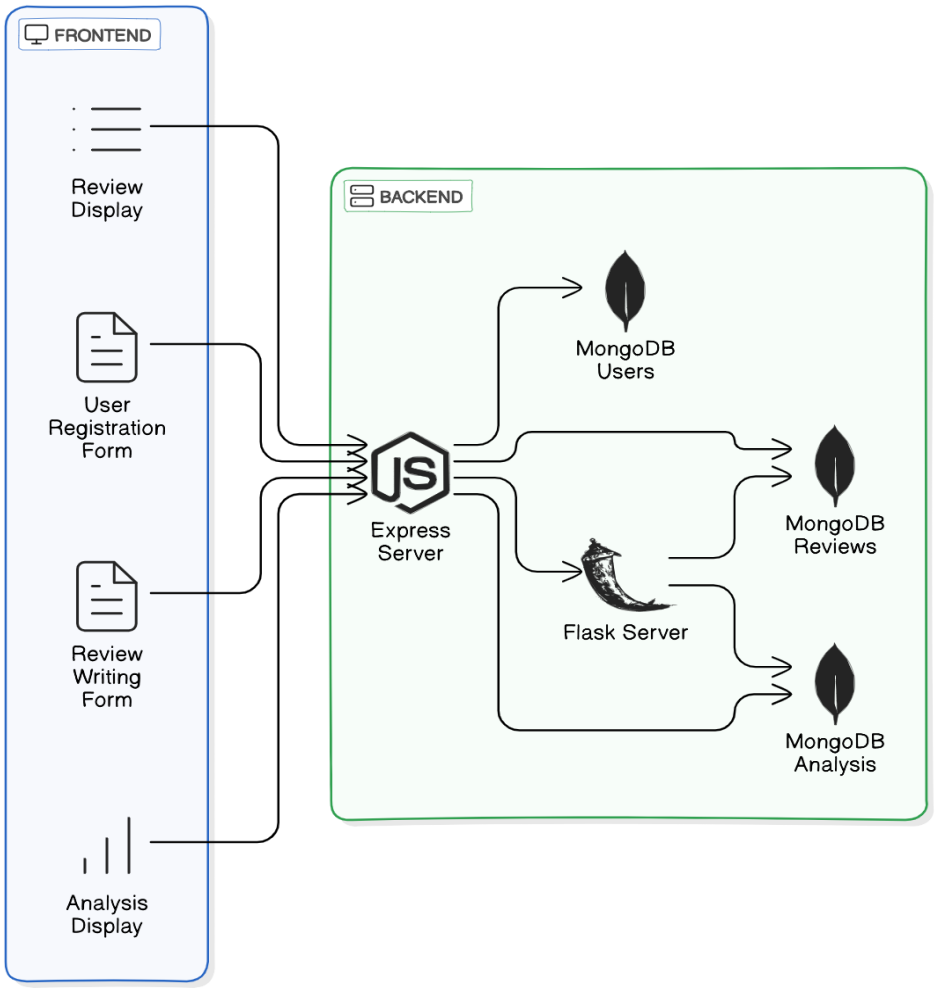
Flow:

1. Periodic or triggered data export from MongoDB (Express) to Flask server.
2. Flask server processes and analyzes the review data.
3. Analyzed data is stored back into MongoDB.
4. React frontend fetches analyzed data via Express and displays it.

Data Flow diagram:

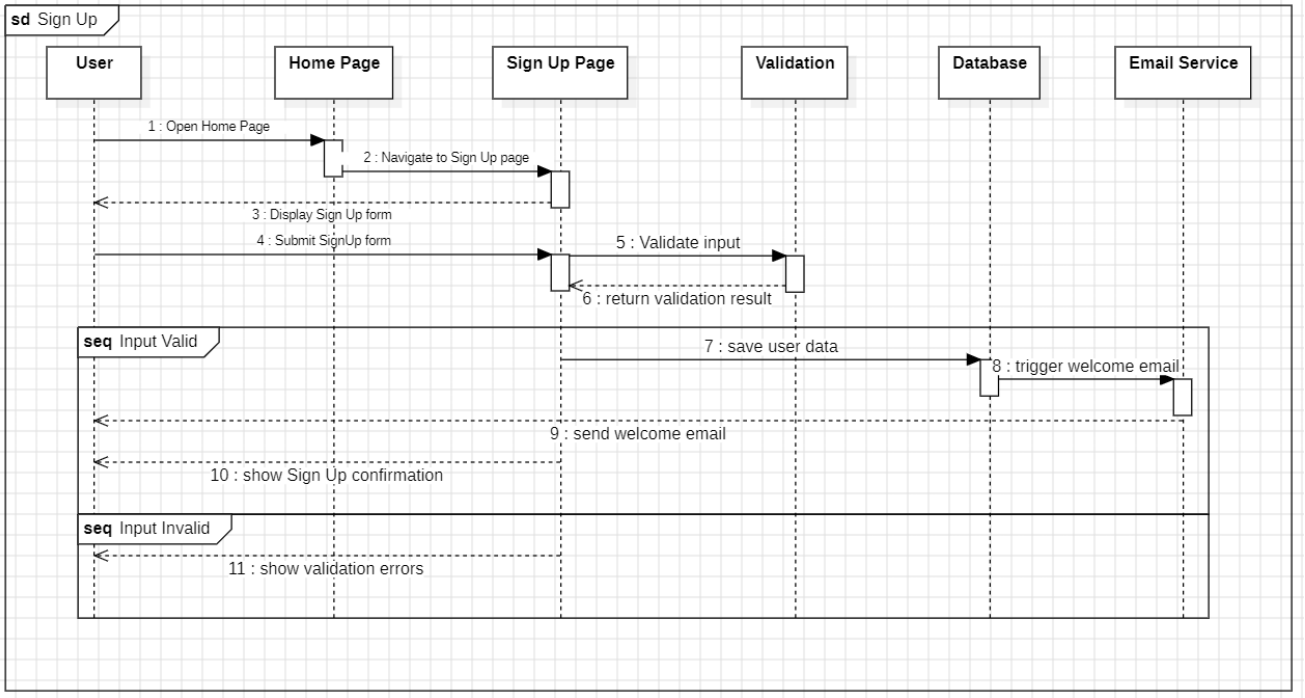


User Registration and Review System Architecture

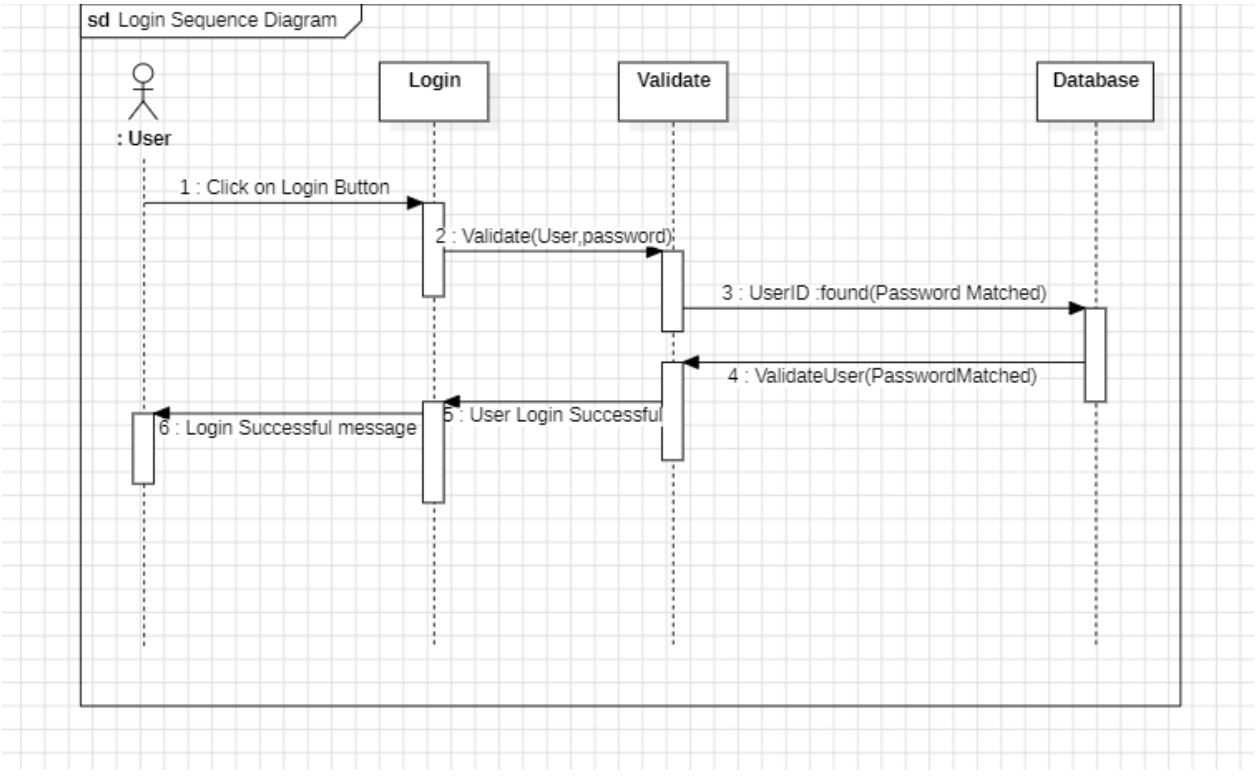


SEQUENCE DIAGRAMS :

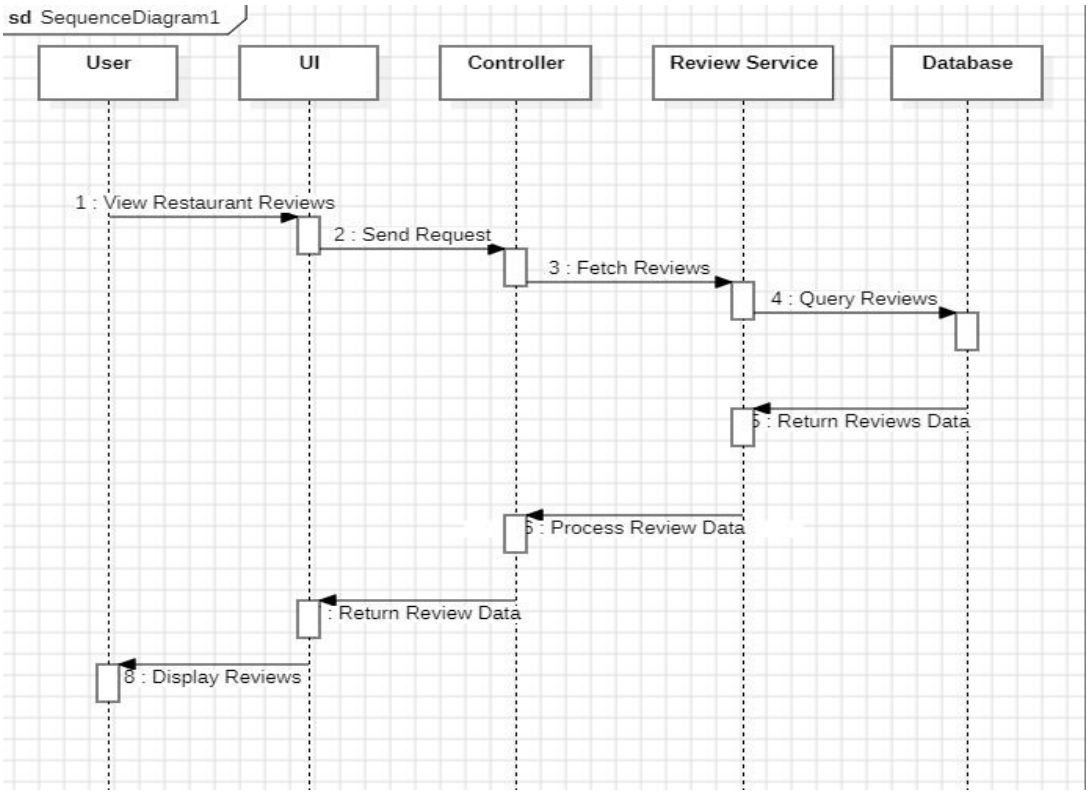
User Registration:



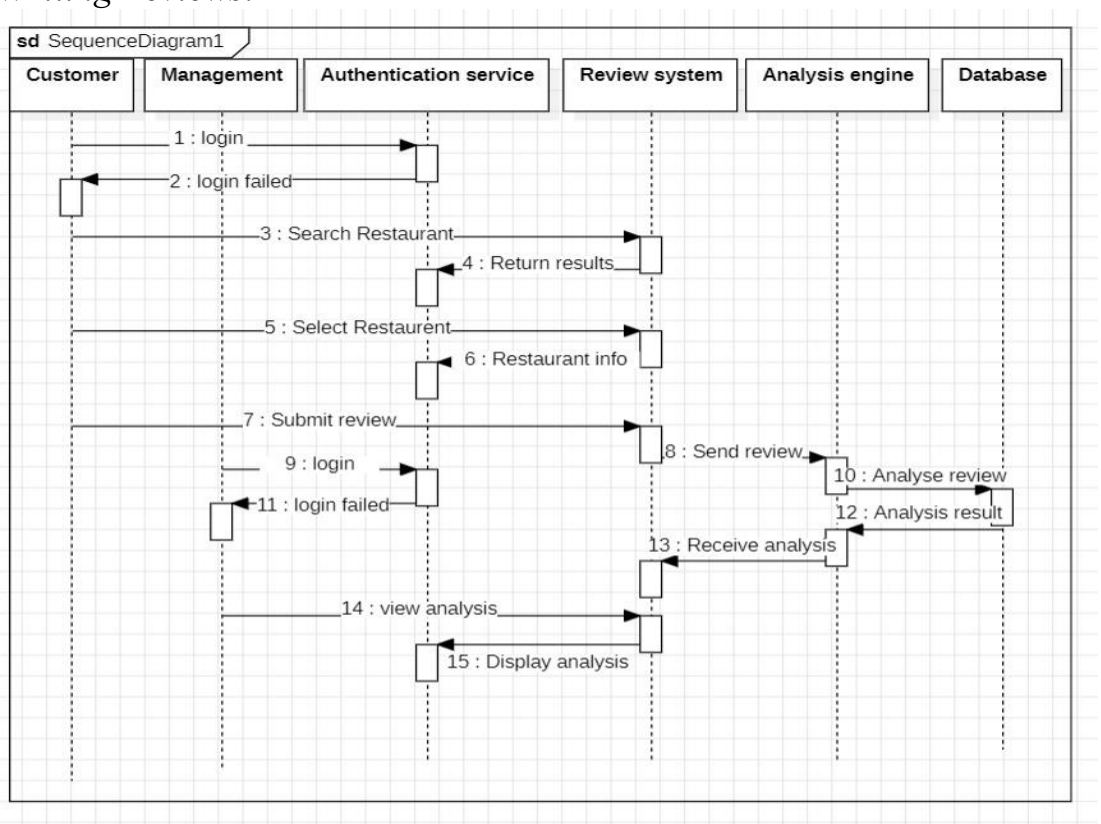
User Login:



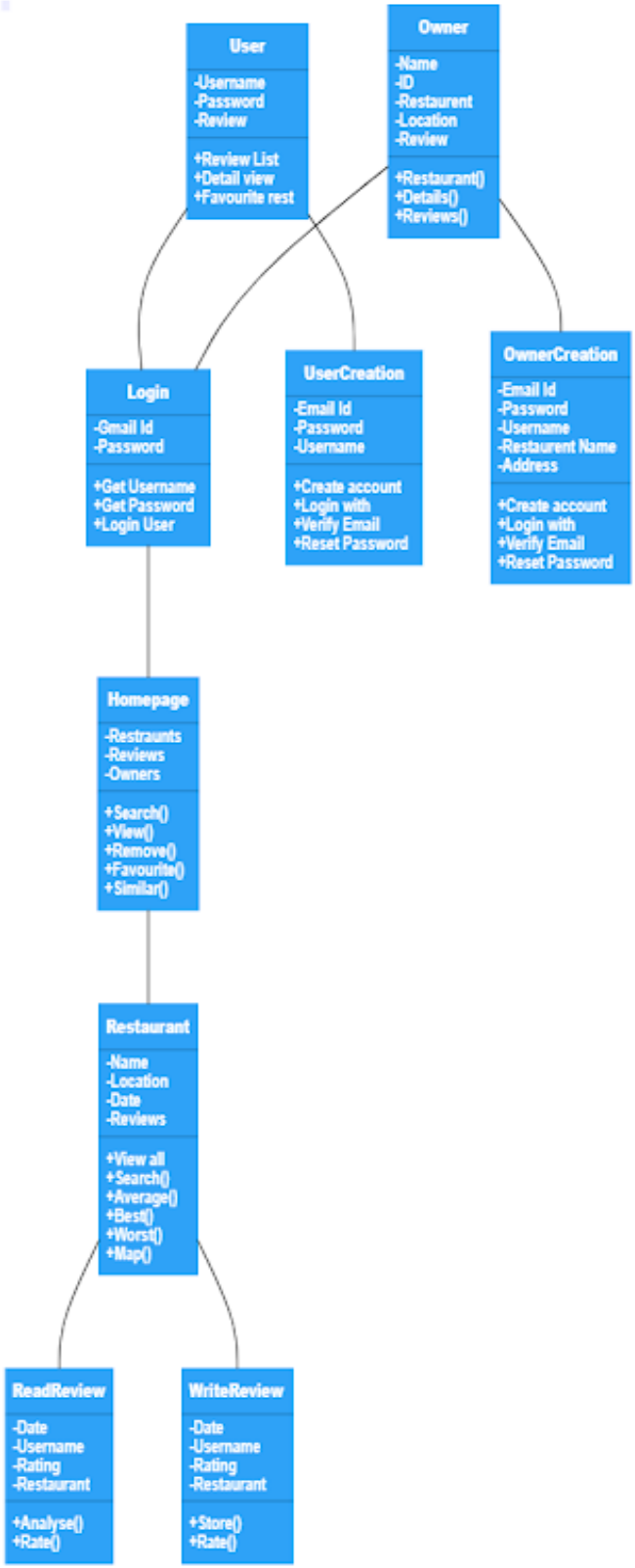
Reading Reviews:



Writing Reviews:



CLASS DIAGRAM:



TEST STRATEGY:

1. **Review Collection:**

    - Test that the system successfully collects customer reviews from various sources such as websites and apps.
    - Verify that reviews are categorized as positive, negative, or neutral based on sentiment analysis.
    - Ensure that the collected reviews are stored accurately in the database with proper metadata.

2. **Sentiment Analysis:**

    - Test the accuracy of sentiment analysis by providing a variety of review texts with known sentiments.
    - Verify that the sentiment analysis algorithm correctly classifies reviews as positive, negative, or neutral.
    - Test the handling of ambiguous or sarcastic reviews to ensure accurate sentiment classification.

3. **Rating Calculation:**

    - Test the calculation of overall ratings for restaurants based on the sentiment scores of their reviews.
    - Verify that the rating calculation algorithm considers both the quantity and sentiment of reviews.
    - Test scenarios where restaurants have varying numbers of reviews to ensure consistent rating calculation.

4. **Ranking and Recommendations:**

- Test the ranking algorithm to ensure that top-rated restaurants are presented prominently to users.
- Verify that recommendations are personalized based on user preferences and past dining history.
- Test scenarios where user preferences and dining history vary to ensure accurate recommendations.

5. Search and Discovery:

- Test the search functionality to ensure users can efficiently find restaurants based on location and cuisine.
- Verify that the search results are accurate and relevant to the user's query.
- Test scenarios with different search criteria and locations to ensure robustness.

6. Detailed Restaurant Profiles:

- Test that restaurant profiles contain accurate information including ratings, reviews, menus, and recommended dishes.
- Verify that menus are displayed correctly and are up-to-date.
- Test scenarios where restaurants have varying levels of detail in their profiles.

7. User Engagement:

- Test user interaction features such as the ability to like, comment, or share reviews.
- Verify that feedback mechanisms are functional and intuitive.
- Test loyalty programs to ensure users are properly rewarded for their engagement.

8. Error Handling:

- Test error scenarios such as invalid user inputs or system failures.
- Verify that error messages are clear, informative, and guide users towards resolution.
- Test the system's resilience to unexpected errors and its ability to gracefully recover.

9. Performance Testing:

- Conduct performance testing to evaluate system responsiveness under various loads.
- Verify that the application performs well under typical usage scenarios.
- Test scenarios with concurrent user access to assess scalability.

10. Security Testing:

- Perform security testing to identify vulnerabilities such as SQL injection or cross-site scripting.
- Verify that user data is properly encrypted and protected.
- Test authentication and authorization mechanisms to ensure secure access to user information.