Software Design Specification

for

FitHub

Prepared by:

Arwa Ibrahim, Bosy Ayman, Farha Ahmed

Zewail City

9 November 2024

1. Introduction

1.1 Purpose

The purpose of this document is to describe the design, architecture, and technical specifications of FitHub. It outlines the functionality, system components, and design decisions to be followed during the development process.

1.2 Scope

This SDS covers the design and implementation details of FitHub. The software will perform the following major tasks:

- Signup/Login
- Profile management
- Password management
- Forums (posts & comments)
- Notifications
- Coach verification by admin
- Plan management
- Progress tracking
- Direct messages (trainee & assigned coach)
- Recipes catalogue
- Exercises catalogue

---

2. System Overview

FitHub is a fitness platform connecting trainees with coaches for personalized plans, progress tracking, and real-time communication. It offers the following key features:

1. **Signup/Login**: Users can create accounts and log in, with role-based access (coach, trainee).
2. **Profile Management**: Users can update personal details and fitness preferences.
3. **Password Management**: Secure password change and reset options.
4. **Forums (Posts & Comments)**: Users can share posts, ask questions, and comment in community forums.
5. **Notifications**: Real-time notifications for new messages, forum updates, and activity.
6. **Coach Verification by Admin**: Admins verify coach credentials to ensure quality.
7. **Plan Management**: Coaches create and manage personalized fitness plans for trainees.
8. **Progress Tracking**: Trainees track metrics like weight, height, and exercising.
9. **Direct Messages (Trainee & Coach)**: Private messaging for coach-trainee communication.
10. **Recipes Catalogue**: Healthy recipes for trainees and coaches to support fitness goals.
11. **Exercises Catalogue**: A collection of exercises to include in workout plans or look through.

---

3. System Architecture

3.1 Architectural Design

This project follows the client-server architecture, where:

- **Frontend** communicates with the backend using RESTAPI.
- **Backend** interacts with the database to manage and retrieve data, for efficient data storage & access.

  In this architecture, the frontend and backend are decoupled, allowing them to be developed and scaled independently. SQL queries within flask routes manage database interactions seamlessly, enabling efficient data handling.

3.2 Data Flow

1. **User Interaction**: The user interacts with the UI to perform an action (e.g., login, search for exercises, choose a coach, progress tracking).
2. **Request Processing**: The frontend sends an API request to the backend server.
3. **Data Handling**: The backend processes the request, interacts with the database, and fetches or updates the necessary data.
4. **Response**: The backend sends the response back to the frontend, updating the UI.

---

4. Database Design

The system will store data in relational (SQLite) database with the following entities and relationships:

Table 1: User

1. User_ID (PK)
2. Name
3. Email
4. Password
5. Role (admin , trainee , coach)
6. Age
7. Gender
8. Profile picture
9. Interests
10. Authentication
11. Security_Question

Table 2: Trainee

1. Trainee_ID (PK , FK)
2. Coach_ID (FK)
3. Weight
4. Height
5. BMI (Auto-calculation)
6. Exercise_Level (beginner, intermediate, advanced)
7. Trainee_Exercises
8. Trainee_Fat
9. Trainee_Calories
10. Trainee_Carbs
11. Trainee_Protein
12. Trainee_Recipes
13. Notified

Table 3: Coach

1. Coach_ID (PK , FK)
2. Verified (true, false)
3. Description
4. Experience
5. Certification

Table 4: Exercise

1. Exercise_ID (PK)
2. Coach_ID (FK)
3. Name
4. Media (Photo , Video)
5. Muscles_Targeted
6. Equipment
7. Duration
8. Description
9. More_Info

Table 5: Plan

1. Plan_ID (PK)
2. Trainee_ID (FK)
3. plan {day: list (excercise_id(FK)}

Table 6: Recipe

1. Recipe_ID (PK)
2. Coach_ID (FK)
3. Meal_Type
4. Recipe_Name
5. Media (Photo , Video)
6. Ingredients
7. Steps
8. Nutrition_Information

Table 7: Post

1. Post_ID (PK)
2. User_ID (FK)
3. Content
4. Tags
5. Time_Stamp
6. Media

Table 8: Comment

1. Comment_ID (PK)
2. Post_ID (FK)
3. User_ID (FK)
4. Content
5. Time_Stamp

Table 9: Chat

1. Chat_ID (PK)
2. User1_ID (FK)
3. User2_ID (FK)

Table 10: Message

1. Message_ID (PK)
2. Chat_ID (FK)
3. Sender_ID (FK)
4. Content
5. Time_Stamp

Table 11: Interest

1. Interest_ID
2. Name

Table 12: Trainee_Exercise

1. Trainee_ID (FK)
2. Trainee_Exercises (Comma-separated Exercise_IDs)
3. Timestamp

Table 13: Trainee_Recipes

1. Trainee_ID (FK)
2. Timestamp
3. Trainee_Fat
4. Trainee_Calories
5. Trainee_Carbs
6. Trainee_Protein

Relationships:

1. **User - Posts**: One-to-Many
2. **User - Comments**: One-to-Many
3. **Coach - Trainee**: One-to-Many
4. **Posts - Comments**: One-to-Many
5. **Chat - Message**: One-to-many
6. **Coach - Chat**: One-to-Many
7. **Trainee - Chat**: One-to-One
8. **Plan - Exercise:** Many-to-Many
9. **Trainee - Plan:** Many-to-Many

---

5. Technology Stack

FitHub's technology picks are intended to leverage our team's Python experience. The system consists of the following components:

- **Frontend**: html, css, js
  - HTML, CSS, and JavaScript are the foundation of the web. They work across all modern browsers and platforms without requiring additional software or plugins. This ensures a broad audience can access our application.

- **Backend**: Flask
  - A strong framework with tools simplifying complex processes such as API creation and authentication, keeping the reliability and security of FitHub.

- **Database**: SQLite
  - SQL integration allows for better tools for query drafting, and SQLite, being lightweight, is ideal for the initial releases due to its simplicity and speed.

6.Design Patterns

6.1 Overview

The system utilizes the following design patterns to ensure modularity, scalability, and maintainability:

1. **Observer Pattern:** Notifications and updates are managed dynamically, where users (observers) are notified of events or changes (e.g., forum updates or new messages).
2. **Strategy Pattern:** Authentication strategies (Google OAuth and traditional email/password) implement encapsulation and allow flexibility in authentication methods.

3. **Template Method Pattern:** Recipe and exercise structures follow a specific template for creation and rendering.
4. **Singleton Pattern:** Database connections use the Singleton pattern to ensure only one connection instance is active.
5. **Command Pattern:** Commands like creating posts and comments are encapsulated as separate entities and executed through a defined interface.

---

7. Testing Plan

7.1 Unit Testing

Objective: To test individual modules and components of the system to ensure they function correctly in isolation.

Approach:

- Test the frontend components (html, css, js) to verify they render correctly and handle user inputs as expected.
- Test backend functionalities by writing unit tests for individual functions and classes, such as recipe cataloging, progress tracking, and exercise uploading.
- Use a Python testing framework like unittest or pytest to automate unit tests for the backend.

7.2 Integration Testing

Objective: To test the interaction between different components of the system (frontend, backend, and database).

Approach:

- Verify that the html, css, js frontend communicates correctly with the backend through API calls.
- Ensure that the backend interacts properly with the database, retrieving and saving data as expected, using SQL queries.

7.3 User Acceptance Testing (UAT)

Objective: To validate the system with real-world user scenarios and ensure it meets the project's requirements.

Approach:

- Create test cases based on typical user interactions, such as signing up, logging progress, or viewing exercises.
- Involve actual users or client representatives to test the system and provide feedback on its functionality and usability.

7.4 Performance Testing

Objective: To ensure that FitHub performs efficiently under various loads and stress conditions.

Approach:

- Test how the system handles multiple concurrent users interacting with the system, such as searching for recipes or showcasing progress.
- Measure response times for key actions (e.g., making a post or private messaging) under different loads to ensure the system can handle expected traffic.

7.5 Security Testing

Objective: To ensure the system's security and safeguard sensitive information like user data and transaction history.

Approach:

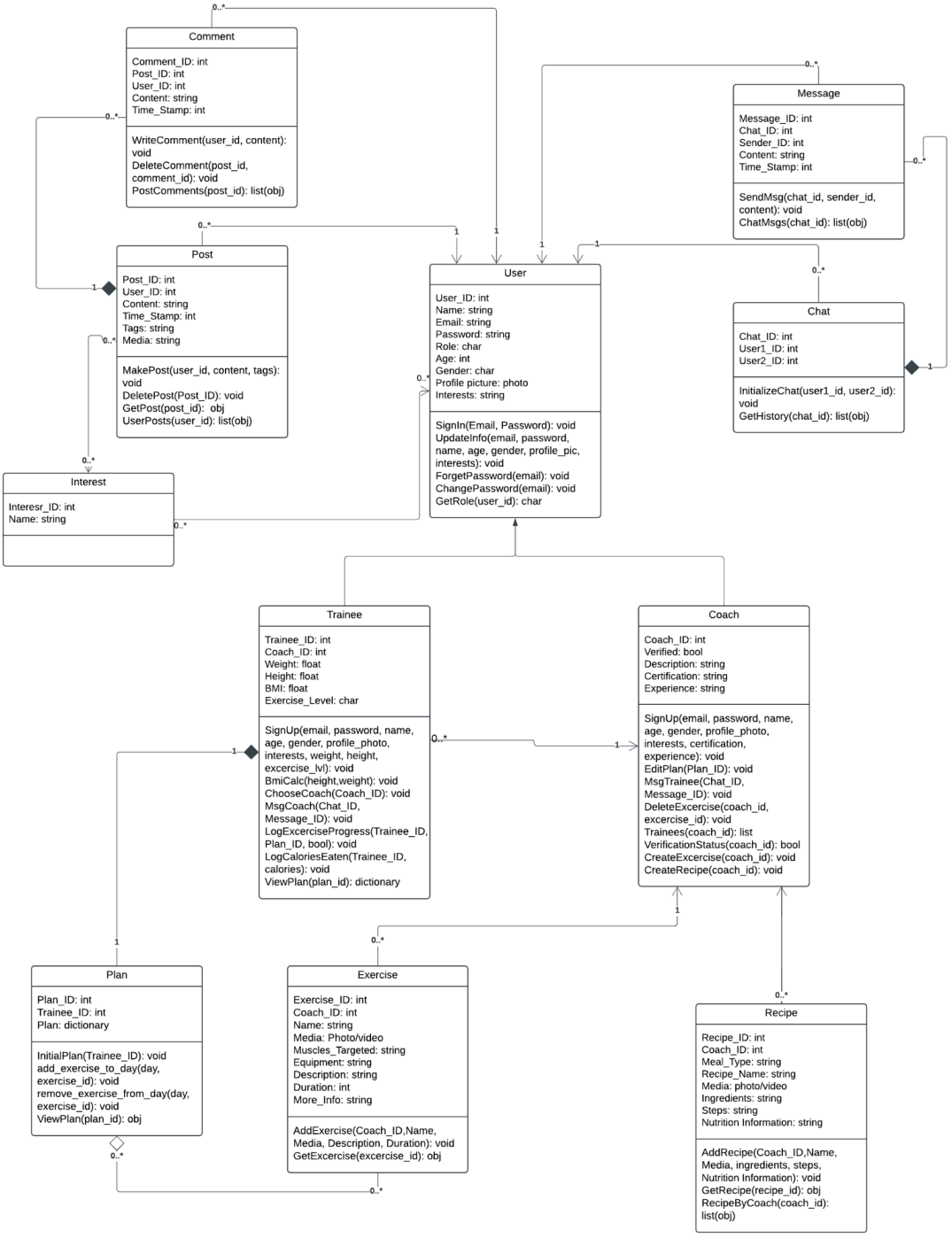
- Test user authentication and authorization mechanisms to prevent unauthorized access.
- Check for potential vulnerabilities like Cross-Site Scripting (XSS).

---

8. Diagrams

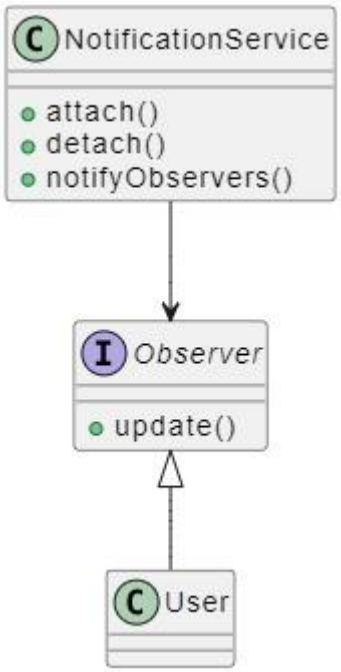
8.1 Class Diagram

Link: FitHub_Class_Diagram.pdf



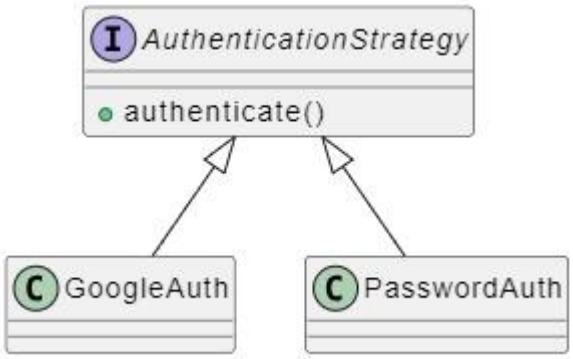
Observer Pattern

Notifications and updates are managed dynamically, where users (observers) are notified of events or changes (e.g., forum updates or new messages).



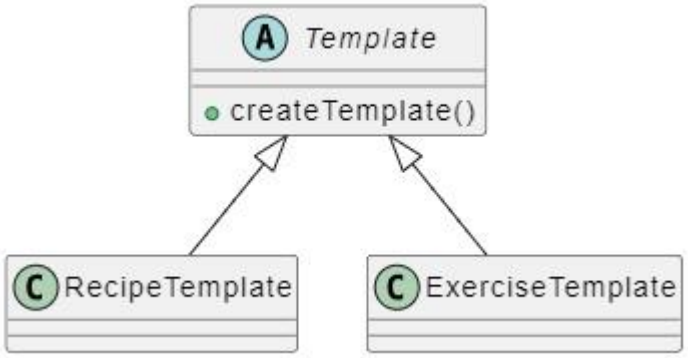
Strategy Pattern

Authentication strategies (Google OAuth and traditional email/password) implement encapsulation and allow flexibility in authentication methods.



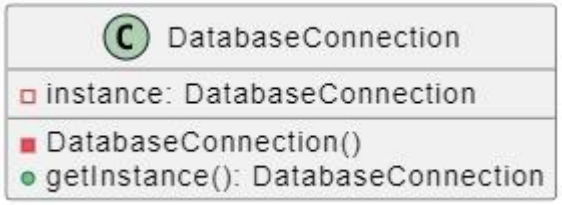
Template Method Pattern

Recipe and exercise structures follow a specific template for creation and rendering.



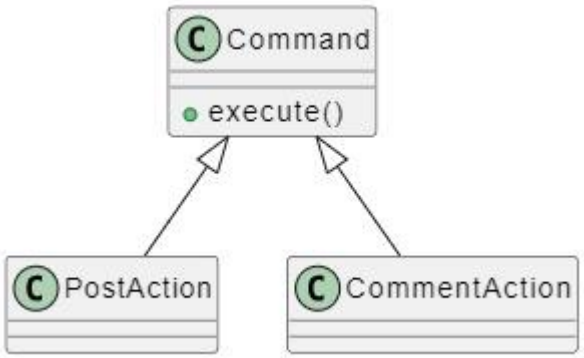
Singleton Pattern

Database connections use the Singleton pattern to ensure only one connection instance is active.

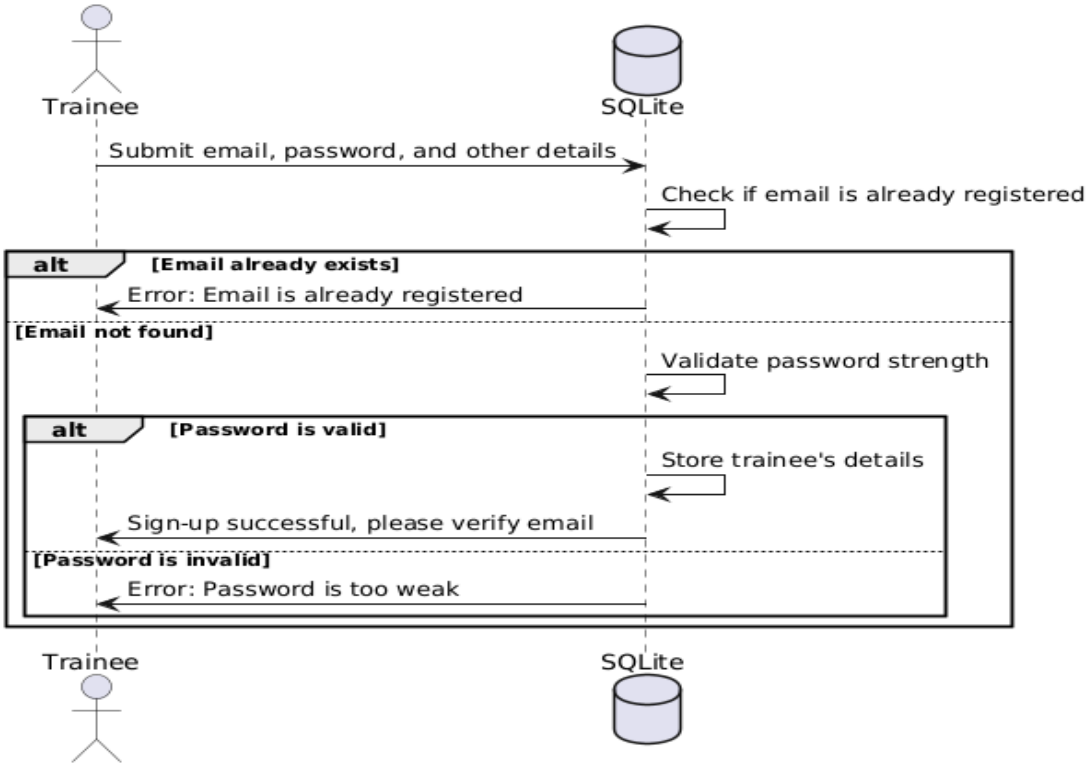


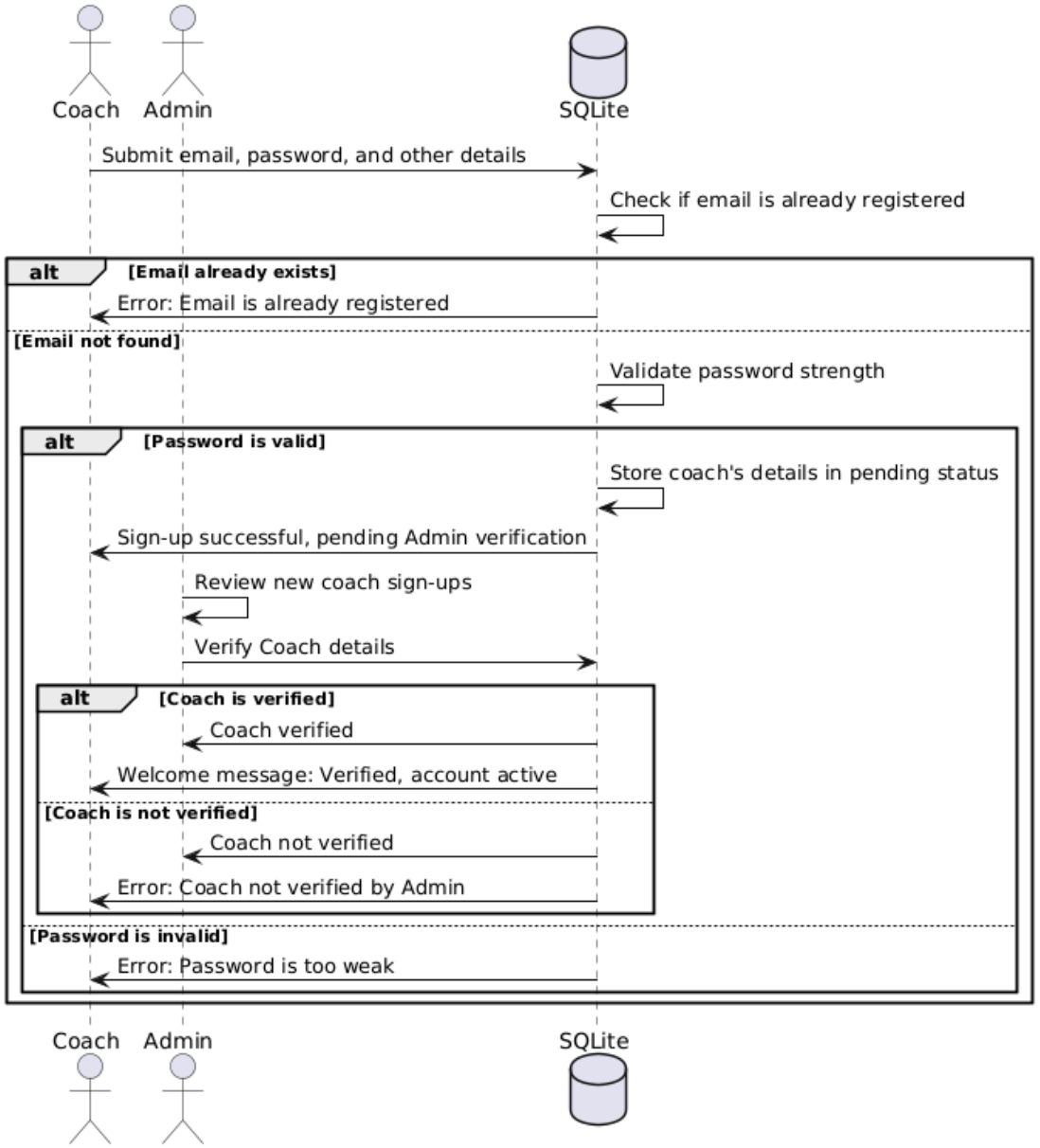
Command Pattern

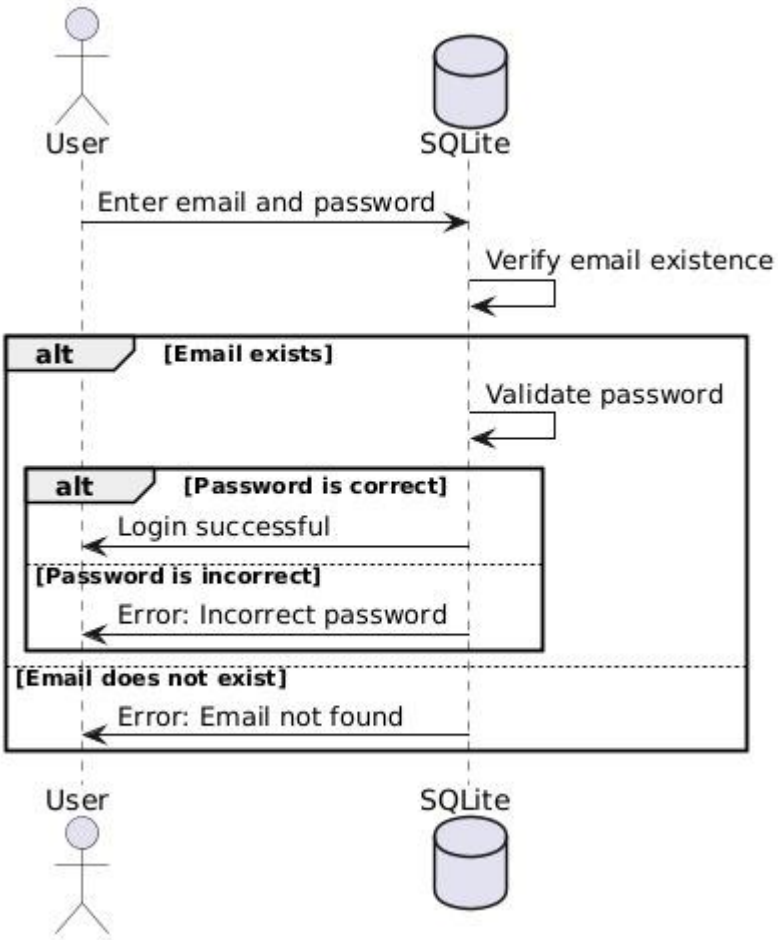
Commands like creating posts and comments are encapsulated as separate entities and executed through a defined interface.

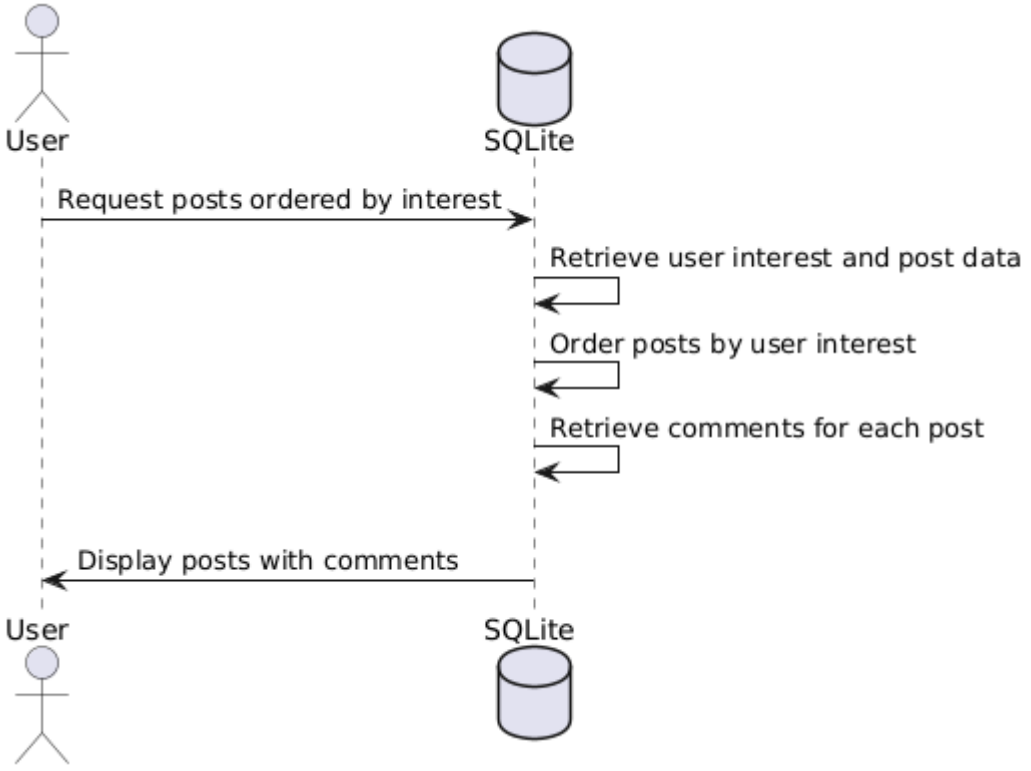


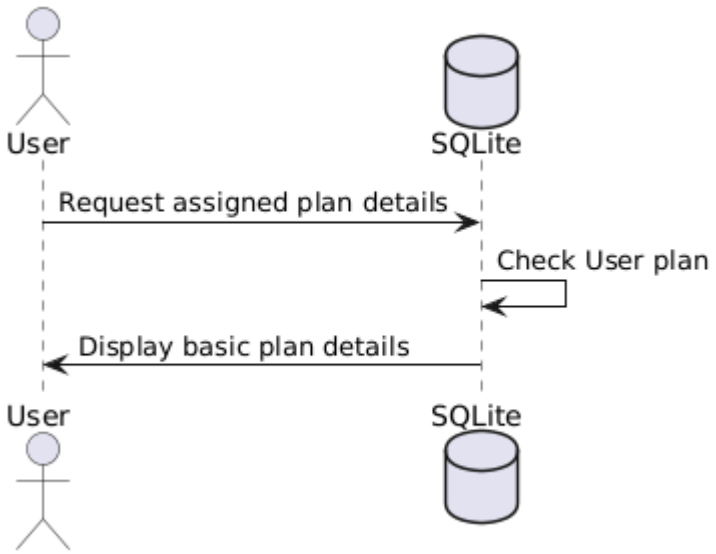
8.2 Sequence Diagram


Trainee
SQLite
Submit email, password, and other details
Check if email is already registered
alt
[Email already exists]
Error: Email is already registered
[Email not found]
Validate password strength
alt
[Password is valid]
Store trainee's details
Sign-up successful, please verify email
[Password is invalid]
Error: Password is too weak
Trainee
SQLite


Coach
Admin
SQLite
Submit email, password, and other details
Check if email is already registered
alt
[Email already exists]
Error: Email is already registered
[Email not found]
Validate password strength
alt
[Password is valid]
Store coach's details in pending status
Sign-up successful, pending Admin verification
Review new coach sign-ups
Verify Coach details
alt
[Coach is verified]
Coach verified
Welcome message: Verified, account active
[Coach is not verified]
Coach not verified
Error: Coach not verified by Admin
[Password is invalid]
Error: Password is too weak
Coach
Admin
SQLite


User
SQLite
Enter email and password
Verify email existence
alt
[Email exists]
Validate password
alt
[Password is correct]
Login successful
[Password is incorrect]
Error: Incorrect password
[Email does not exist]
Error: Email not found
User
SQLite


User
SQLite
Request posts ordered by interest
Retrieve user interest and post data
Order posts by user interest
Retrieve comments for each post
Display posts with comments
User
SQLite



---

9. Conclusion

FitHub's Software Design Specification (SDS) showcases a client-server architecture which supports key functionalities such as user management, fitness plans, and social interaction between trainees and coaches. The relational database design allows effective data management across entities, which improves scalability and usability. This SDS serves as a basis for ensuring efficient & continuous development that aligns with FitHub's objectives.