FitHub Final Documentation

Development Process

Process Chosen: Scrum Methodology

Why? Scrum was selected for its iterative and incremental nature, which allowed flexibility and adaptability to evolving requirements. By using sprints and a product backlog, we maintained structured progress while incorporating user feedback efficiently.

Product Backlog and Sprint Management

Product Backlog

The product backlog consisted of the following prioritized tasks, organized across three sprints:

1. **Sprint 1: Database Schema Setup and Basic UI**
    - Design and implement the database schema to support core entities.
    - Create basic UI components for user interaction.
2. **Sprint 2: Implementation of Core Functionalities**
    - Develop user authentication, including signup, login, and OAuth integration.
    - Implement functionalities for posts, recipes, and exercises.
    - Ensure smooth integration of these features with the database and UI.
3. **Sprint 3: Implementation of Remaining Functionalities and Deployment**
    - Complete profile management and progress tracking.
    - Finalize forum functionalities for posts and comments.
    - Integrate real-time notifications to enhance user engagement.
    - Conduct comprehensive testing, including unit, integration, and end-to-end tests.
    - Prepare and deploy the application using Docker.

Object-Oriented Design Rationale

OO Design Principles

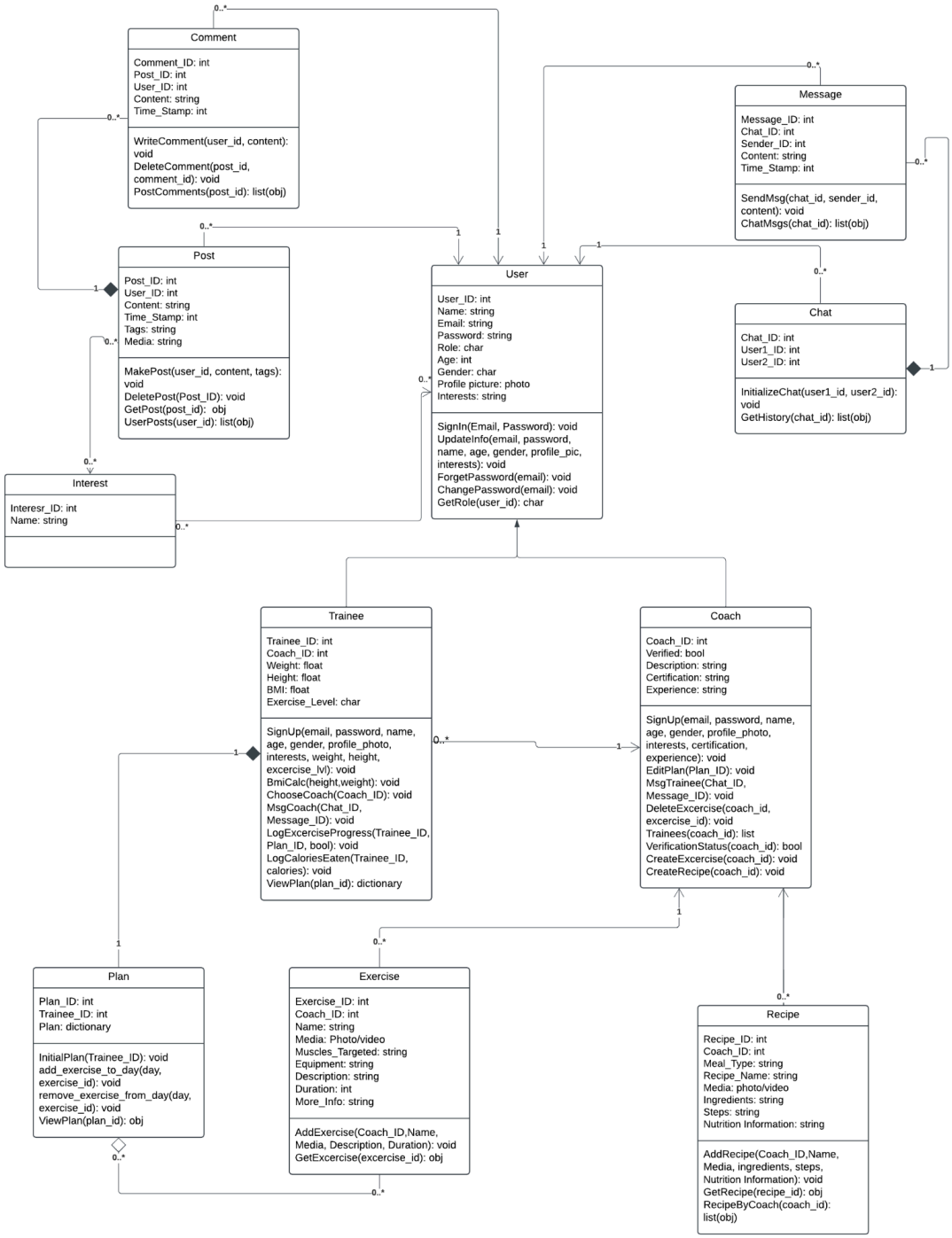
We adhered to the following **SOLID principles**:

1. **Single Responsibility Principle**: Each class has a single, well-defined responsibility.
2. **Open-Closed Principle**: The system was designed to accommodate extensions without modifying existing code.

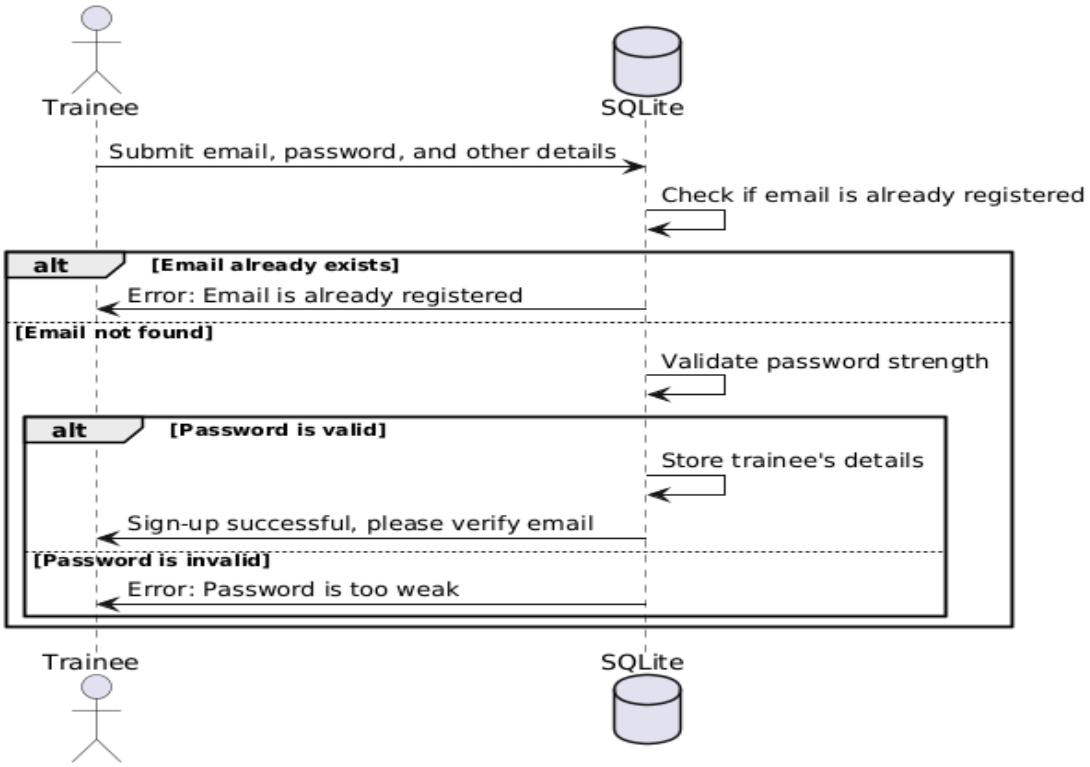
UML Diagrams

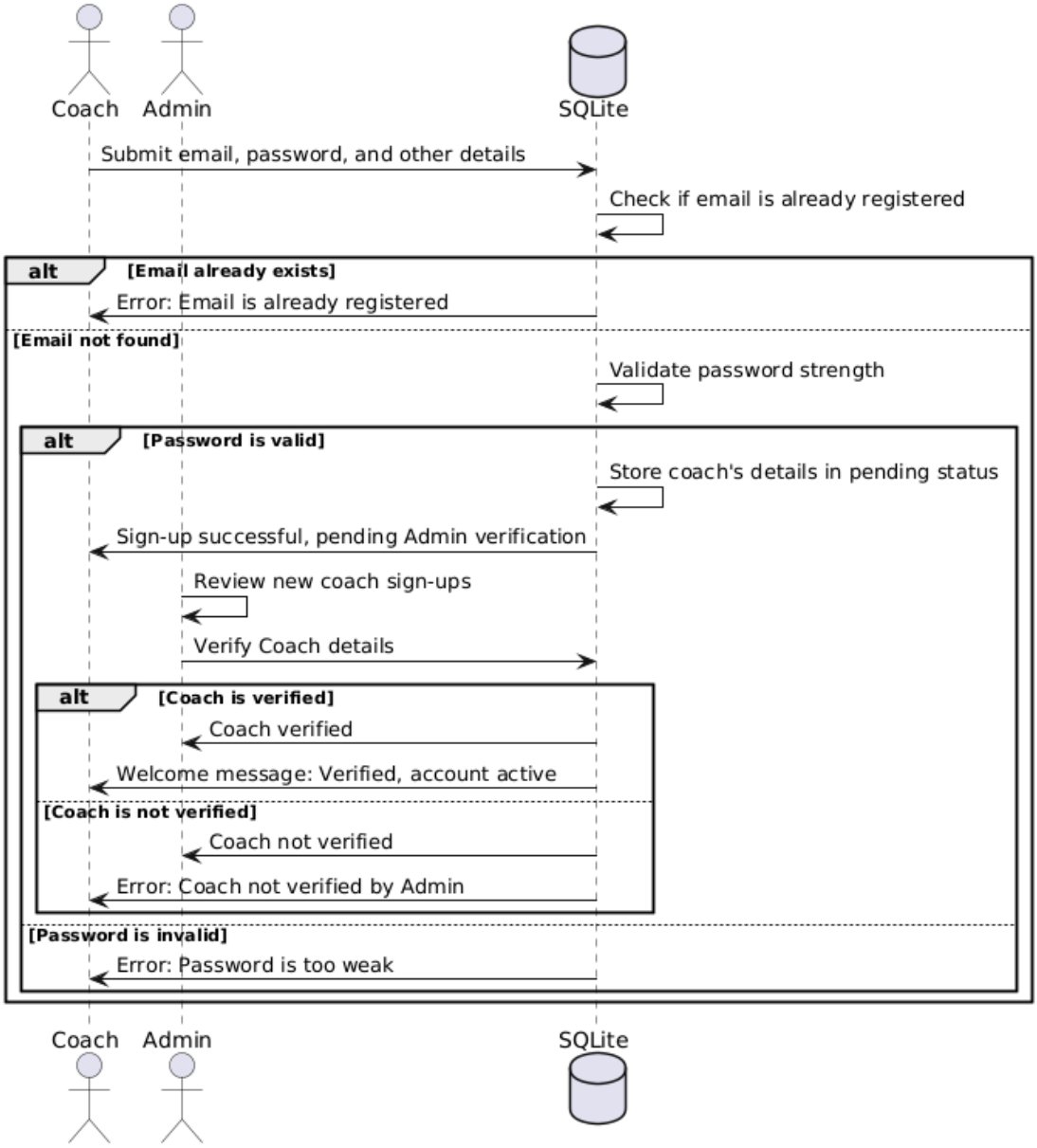
Below are the UML diagrams that represent the system's design:

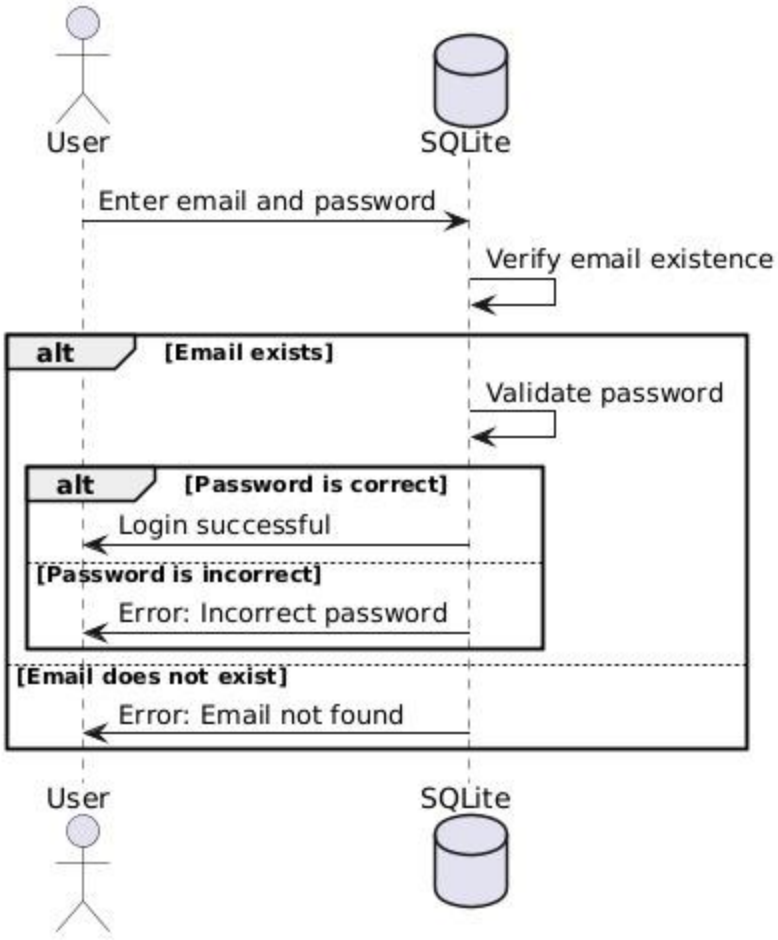
1. **Class Diagram**: link: [class diagram]



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

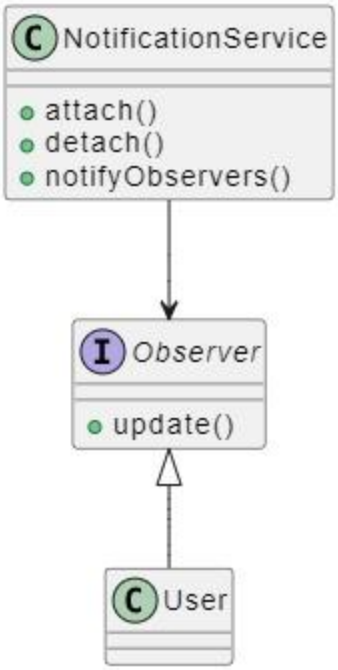


Design Patterns

1. Observer Pattern

Rationale: This pattern was used to implement real-time notifications and updates. It allowed for dynamic updates to multiple users whenever an event occurred (e.g., new post or message).
Implementation:

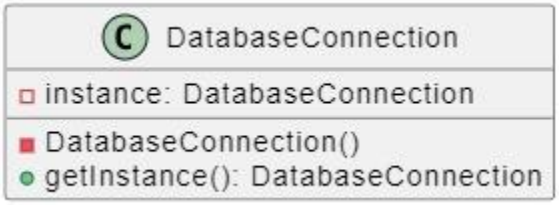
- **Subject**: NotificationService
- **Observers**: Users
- **UML**:



2. Singleton Pattern

Rationale: Ensured a single instance of the database connection, optimizing resource usage and preventing conflicts. **Implementation**:

- **Singleton Class**: DatabaseConnection
- **UML**:



Testing Plans and Scripts

Testing Plans

- **Unit Testing**: Each module was tested independently using pytest.
- **Integration Testing**: Verified interactions between modules.
- **End-to-End Testing**: Ensured the system functions as expected from login to workout plan updates.

Sample Testing Script

```
# Test for user login functionality
import pytest

def test_login():
    response = app.test_client().post('/login', data={'email': 'test@example.com', 'password': 'password123'})
    assert response.status_code == 200
    assert b"Welcome" in response.data
```

Deployment Model

Using Docker

A Docker image was created for seamless deployment. It encapsulated the application along with its dependencies, ensuring consistency across environments.

Dockerfile Example:

```
FROM python:3.9-alpine

WORKDIR /app

COPY . .

RUN pip install flask numpy Authlib flask-mail requests flask_bcrypt

EXPOSE 4000

CMD [ "python", "app.py" ]
```

Deployment Steps

Build the Docker image:

```
 docker build -t fithub-app .
```

Run the container:

```
 docker run -p 5000:5000 fithub-app
```

Performance Measures

Monitoring Performance

- Used **Postman** for load testing.
- Monitored response times, throughput, and error rates.

Evaluation Results

- Average Response Time: ~900ms
- Max Concurrent Users Supported: 100
