Software Design Specification

for

FitHub

Prepared by:

Arwa Ibrahim, Bosy Ayman, Farha Ahmed,

Habiba Arafa, Nourhan Deif

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. The Django ORM manages 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. Email
3. Password
4. Role (admin , trainee , coach)
5. Name
6. Age
7. Gender
8. Profile picture
9. Interests

Table 2: Admin

1. Admin_ID (PK , FK)
2. coaches_awaiting_verification

Table 3: Trainee
1. Trainee_ID (PK , FK)
2. Coach_ID (FK)
3. Weight
4. Height
5. BMI (Auto-calculation)
6. Exercise_Level (beginner, intermediate, advanced)

Table 4: Coach
1. Coach_ID (PK , FK)
2. Verified (true, false, awaiting)
3. Certification
4. Experience

Table 5: Exercises
1. Exercise_ID (PK)
2. Coach_ID (FK)
3. Name
4. Media (Photo , Video)
5. Description
6. Duration

Table 6: Plan
1. Plan_ID (PK)
2. Trainee_ID (FK)
3. plan {day: list (excercise_id(FK)}

Table 6: Recipes
1. Recipe_ID (PK)
2. Coach_ID (FK)
3. Name
4. Media (Photo , Video)
5. Ingredients
6. Steps
7. Nutrition Information

Table 7: Posts
1. Post_ID (PK)
2. User_ID (FK)
3. Content
4. Time_Stamp
5. Tags

Table 8: Comments
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

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

---

5. Technology Stack

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

- **Frontend**: React
  - o A widely used framework, meaning it has plenty of resources and 3rd party tools to help in ensuring easy and proper implementation for the project. It also has fast rendering features which allows for instant notifications and social interactions.

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

- **Database**: Django ORM, SQLite
  - o Django ORM will be easier to integrate with Django as the backend, as well as, allowing for better query drafting. SQLite being lightweight is perfect for the initial releases since it's simple & quick.

6. Testing Plan

6.1 Unit Testing

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

Approach:
- Test the frontend components (React) to verify they render correctly and handle user inputs as expected.
- Test backend functionalities (Django) by writing unit tests for individual functions and classes, such as recipe cataloging, progress tracking, and exercise uploading.
- Use Django's built-in testing framework to automate unit tests for the backend.

6.2 Integration Testing

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

Approach:
- Verify that the React frontend communicates correctly with the Django backend through API calls.
- Ensure that the Django backend interacts properly with the database via Django ORM, retrieving and saving data as expected.

6.3 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.

6.4 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.

6.5 Security Testing

Objective: To ensure the system's security and safeguard sensitive information like user data and transaction history.
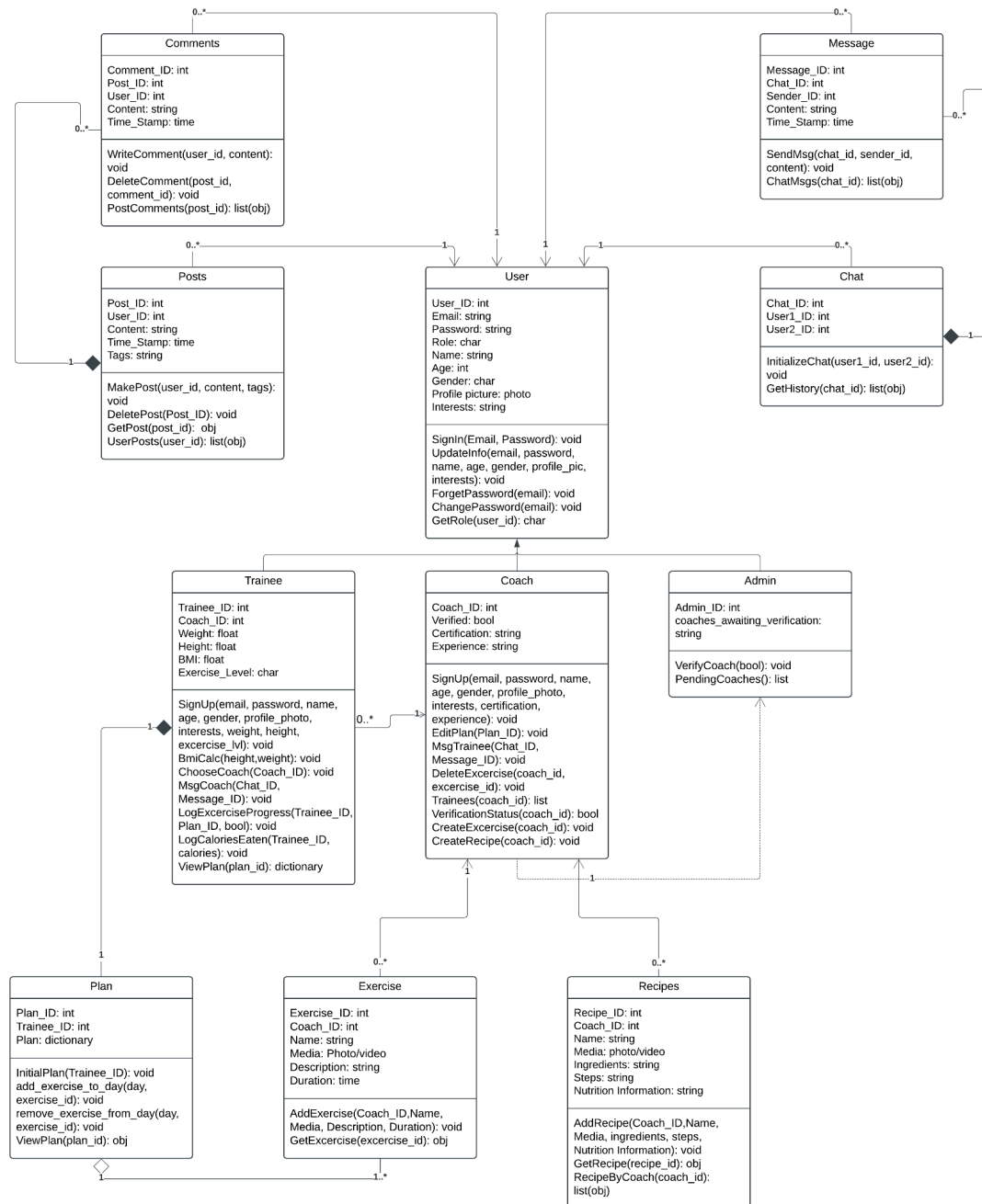
Approach:
- Test user authentication and authorization mechanisms to prevent unauthorized access.
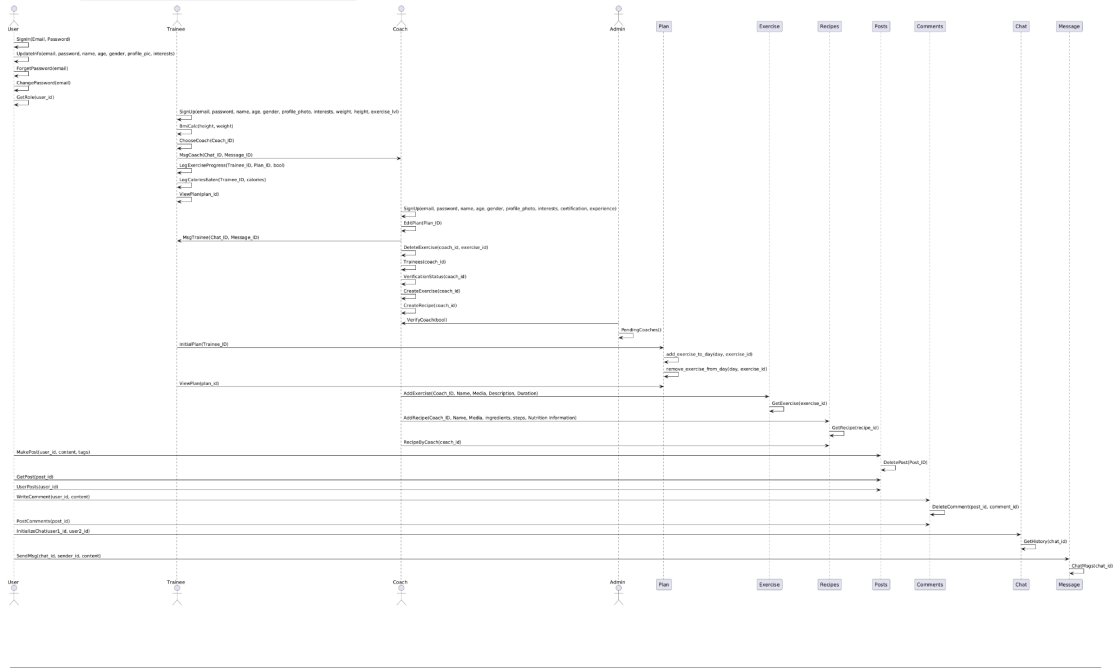- Check for potential vulnerabilities like Cross-Site Scripting (XSS).

7. Diagrams

7.1 Class Diagram

Link: 📄 FitHub_Class_Diagram.pdf

Comments

Comment_ID: int
Post_ID: int
User_ID: int
Content: string
Time_Stamp: time

WriteComment(user_id, content): void
DeleteComment(post_id, comment_id): void
PostComments(post_id): list(obj)

Message

Message_ID: int
Chat_ID: int
Sender_ID: int
Content: string
Time_Stamp: time

SendMsg(chat_id, sender_id, content): void
ChatMsgs(chat_id): list(obj)

Posts

Post_ID: int
User_ID: int
Content: string
Time_Stamp: time
Tags: string

MakePost(user_id, content, tags): void
DeletePost(Post_ID): void
GetPost(post_id): obj
UserPosts(user_id): list(obj)

User

User_ID: int
Email: string
Password: string
Role: char
Name: string
Age: int
Gender: char
Profile picture: photo
Interests: string

SignIn(Email, Password): void
UpdateInfo(email, password, name, age, gender, profile_pic, interests): void
ForgetPassword(email): void
ChangePassword(email): void
GetRole(user_id): char

Chat

Chat_ID: int
User1_ID: int
User2_ID: int

InitializeChat(user1_id, user2_id): void
GetHistory(chat_id): list(obj)

Trainee

Trainee_ID: int
Coach_ID: int
Weight: float
Height: float
BMI: float
Exercise_Level: char

SignUp(email, password, name, age, gender, profile_photo, interests, weight, height, excercise_lvl): void
BmiCalc(height,weight): void
ChooseCoach(Coach_ID): void
MsgCoach(Chat_ID, Message_ID): void
LogExcerciseProgress(Trainee_ID, Plan_ID, bool): void
LogCaloriesEaten(Trainee_ID, calories): void
ViewPlan(plan_id): dictionary

Coach

Coach_ID: int
Verified: bool
Certification: string
Experience: string

SignUp(email, password, name, age, gender, profile_photo, interests, certification, experience): void
EditPlan(Plan_ID): void
MsgTrainee(Chat_ID, Message_ID): void
DeleteExcercise(coach_id, excercise_id): void
Trainees(coach_id): list
VerificationStatus(coach_id): bool
CreateExcercise(coach_id): void
CreateRecipe(coach_id): void

Admin

Admin_ID: int
coaches_awaiting_verification: string

VerifyCoach(bool): void
PendingCoaches(): list

Plan

Plan_ID: int
Trainee_ID: int
Plan: dictionary

InitialPlan(Trainee_ID): void
add_exercise_to_day(day, exercise_id): void
remove_exercise_from_day(day, exercise_id): void
ViewPlan(plan_id): obj

Exercise

Exercise_ID: int
Coach_ID: int
Name: string
Media: Photo/video
Description: string
Duration: time

AddExercise(Coach_ID,Name, Media, Description, Duration): void
GetExcercise(excercise_id): obj

Recipes

Recipe_ID: int
Coach_ID: int
Name: string
Media: photo/video
Ingredients: string
Steps: string
Nutrition Information: string

AddRecipe(Coach_ID,Name, Media, ingredients, steps, Nutrition Information): void
GetRecipe(recipe_id): obj
RecipeByCoach(coach_id): list(obj)

7.2 Sequence Diagram

Link: 📄 Sequence_Diagram.png

8. 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.