Software Requirements Specification

for

FitHub

Version 1.0 approved

Prepared by
Arwa Mater , Bosy Ayman , Farha Ahmed

Zewailcity

9 November 2024

Table of Contents

1. Introduction

1.1 Purpose

This SRS defines the requirements for the FitHub platform, a health and fitness management system designed for trainees and coaches. The system provides functionalities such as user management, post sharing, recipe management, and interaction.

1.2 Document Conventions

- **Font Style**: Section headings are bold and underlined, while standard text is in regular font for clarity.
- **Highlighting**: Functional requirements are prefixed with "FR" and non-functional requirements with "NFR" for easier identification.
- **Numbering**: All requirements are numbered hierarchically (e.g., FR1.1, FR1.2) to indicate relationships.
- **Priorities**: Requirements are assumed to inherit the priority level of their parent sections unless explicitly stated.

1.3 Intended Audience and Reading Suggestions

- This document is intended for:
    - **Developers**: To understand functional and non-functional requirements for implementation.
    - **Project Managers**: To track project goals and ensure alignment with client needs.
    - **Testers**: To validate the application meets all requirements.
- **Reading Suggestions:**
    - Start with the **Introduction** for an overview of the project.
    - Developers and testers should focus on the **Functional Requirements, Non-Functional Requirements**, and **System Features.**
    - Project Managers may review the **Scope, Constraints**, and **System Features.**
    - Documentation Writers should read the **Database Design** and **References** sections.

1.4 Project Scope

FitHub is a fitness-focused web application enabling social interaction and role-based features for Trainees, Coaches, and Admins. It supports user registration, post sharing, commenting, recipe sharing, and personalized workout plans. Admins manage coach verifications, ensuring credibility. The application emphasizes scalability, efficient media storage via Binary Encoding, and a user-friendly interface for fitness enthusiasts.

1.5 References

Flask Documentation: https://flask.palletsprojects.com/
SQLite Documentation: https://sqlite.org/docs.html

2. Overall Description

2.1 Product Perspective

FitHub is a standalone web application that consists of:

- **Frontend**: Developed using HTML, CSS and JS for rendering dynamic content.
- **Backend**: Powered by Flask to handle user interactions, business processes, and communication with the SQLite database.
- **Database**: SQLite is used to store user-generated content, posts, comments, recipes, and profile information.

2.2 Product Features

The core functions of FitHub include:

- **Signup/Login**: Allows users (coaches and trainees) to create accounts and securely log in.
- **Profile Management**: Enables users to view and update personal information.
- **Password Management**: Provides users with the ability to change their passwords.
- **Forums (Posts & Comments)**: Users can create posts, comment, and participate in community discussions.
- **Notifications**: Delivers alerts for important activities, such as new messages, posts, and exercise reminders.
- **Coach Verification by Admin**: Admins can verify coach accounts to ensure authenticity.
- **Plan Management**: Coaches can assign and modify personalized workout plans for trainees.
- **Progress Tracking**: Trainees can track their performance in workouts and nutrition over time.
- **Recipes Catalogue**: A collection of food recipes categorized by nutrition, goal, and availability for all users.
- **Direct Messages** (Trainee & Assigned Coach): Private messaging between trainees and their assigned coaches for personalized support.
- **Exercises Catalogue**: Provides a list of exercises with detailed information, allowing trainees to familiarize themselves and add them to their workout plans.

2.3 User Classes and Characteristics

FitHub will serve the following user types:

- **Admin**: Verifies coaches.
- **Coaches**: Modify workout plans, communicate with trainees, and track progress.
- **Trainees**: Track progress, interact with coaches, and share posts.

2.4 Operating Environment

The system operates in the following environments:

- **Client Side**: Accessible through modern web browsers
- **Server Side**: Hosted on a web server with Flask as the backend and SQLite as the database engine.

2.5 Design and Implementation Constraints

The development of FitHub will be influenced by the following constraints:

- **Technologies**: The system must be developed using Flask as the backend framework, SQLite as the database, and HTML, CSS and JS for the frontend.
- **Hardware**: The system must be optimized to run on standard web browsers and servers, ensuring it operates efficiently without requiring high hardware specifications.
- **Security**: User data must be handled securely with proper authentication, using Flask sessions to maintain secure user sessions.
- **Design Standards**: The system must follow established Flask development conventions and best practices to ensure maintainability and scalability.

2.6 Assumptions and Dependencies

The development and functionality of FitHub depend on the following assumptions and external factors:

Assumptions:

- Users will have access to modern web browsers (Chrome, Firefox, Safari) for an optimal experience.
- The SQLite database will perform efficiently for the scale of the platform, and future scalability will be manageable.

Dependencies:

- The success of the project relies on a stable and secure hosting environment for deploying the Flask application.

3. System Features

3.1 User Registration

Description:
This use case describes the process where new users (trainees or coaches) create an account in the FitHub system. Users provide personal information such as name, age, and weight. They select one of two available roles (trainee or coach) and gain access to role-specific functionalities after successful registration.

Actors:
Trainee, Coach, System

Preconditions:

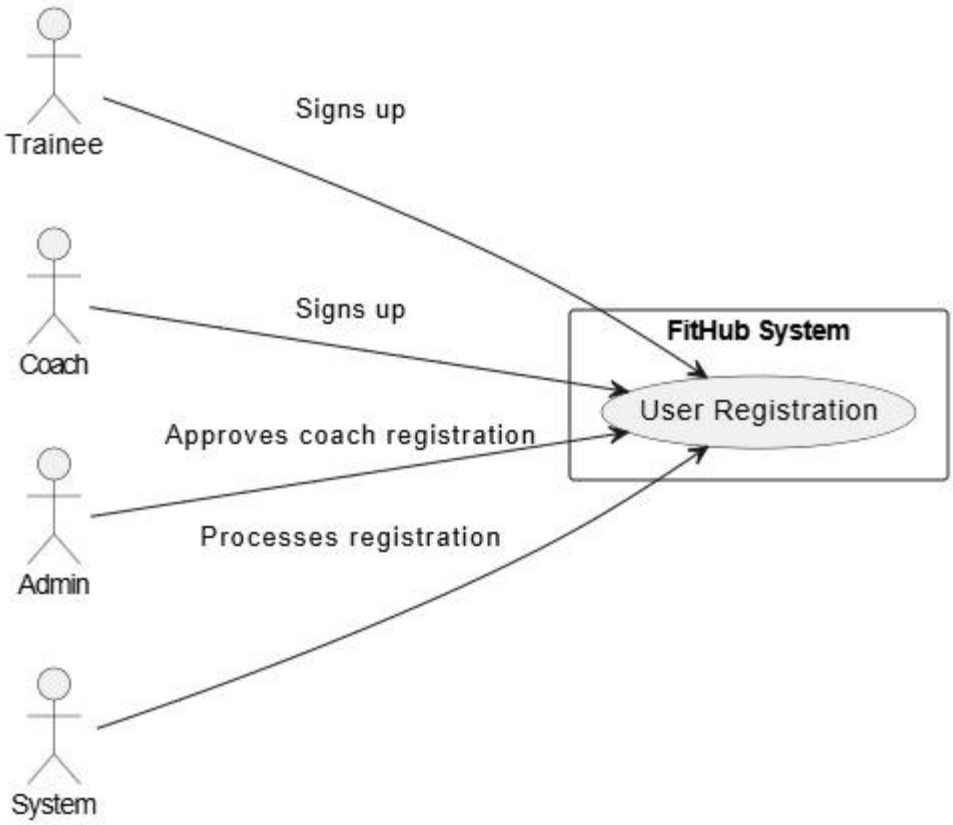
- The user must have an internet connection and navigate to the FitHub webpage.
- The user must be new and not already registered in the system.

Postconditions:

- The system registers the user and redirects them to their respective dashboard.
- Trainees receive a basic training program, while coaches await admin approval for their account.

Steps:

1. The user navigates to the FitHub main page.
2. The user clicks "Sign Up."
3. The user selects a role as either 'Trainee' or 'Coach.'
4. The user provides requested details (e.g., name, age, weight, and role).
5. If the user selects 'Coach,' their registration is subject to admin approval.
6. The user submits the form.
7. The system captures the user details and redirects them to their respective dashboard.



3.2 Trainee Selecting a Coach

Description:
This use case explains how a trainee can select and interact with a coach for personalized guidance. The trainee can search for coaches by specialization, initiate private chats, and have plans customized by the coach.

Actors:
Trainee, Coach, System

Preconditions:

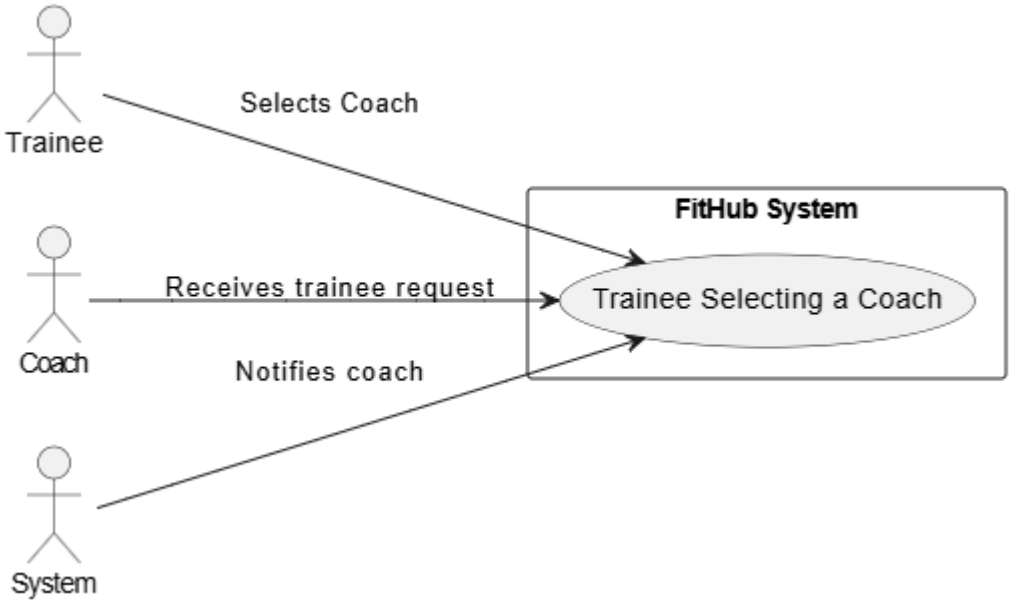
- The trainee must log in before accessing this feature.
- The coach must be vetted by the admin and listed as available.

Postconditions:

- The trainee selects a coach and views their profile.
- The system notifies the coach, enabling them to modify the trainee's plan.

Steps:

1. The trainee navigates to the "Coach Catalog" page.
2. The trainee searches for a coach based on specialization or preference.
3. The trainee selects a coach from the list.
4. The system sends a notification to the selected coach.
5. The coach accepts the trainee's request.
6. The trainee and coach exchange messages, and the coach customizes the plan.



---

4. External Interface Requirements

4.1 User Interfaces

Login Page

- **Purpose:** Provides an entry point for users to log in or sign up.
- **Elements:**
    - **Username/Email Field:** Text box for the user to enter their registered username or email.
    - **Password Field:** Password input box with masking for privacy.
    - **Login Button:** A button to submit credentials.

- **Sign Up Option:** A link or button labeled *"Sign up"* to allow new users to create an account as a coach or a trainee.
- **Forgot Password Link:** Provides an option to recover account credentials.

Sign-Up Page (Trainee/Coach)

- **Purpose:** Allows new users to register as a trainee or coach.
- **Elements:**
  - **User Role Selection:** A toggle or buttons to choose between *Trainee* or *Coach*.
  - **Input Fields:**
    - Full name
    - Email
    - Password (with a strength indicator)
    - Confirm Password
    - Additional information based on role:
      - *Trainee:* Goals, fitness level, availability.
      - *Coach:* Expertise, years of experience, certifications.
  - **Sign Up Button:** A button to submit the information.

Profile Page

- **Purpose:** Displays user-specific information and allows updates.
- **Elements:**
  - **Profile Information:**
    - Name, role (trainee/coach), and basic details.
    - Posts of (trainee/coach)
    - Statistical information of the trainee
    - Current plan of the trainee.
    - Editable fields with an "Edit" button.

Show Recipes Page

1. **Purpose:** Displays a list of all saved recipes for browsing and interaction.
2. **Elements:**
   - **Search Bar:**
     - Allows users to filter recipes by name or category.
     - Includes a "Search" button or executes a search on Enter keypress.
   - **Recipe Cards/Rows:**
     - Each recipe is displayed as a card or in a list format with:
       - Recipe name.
       - image .
     - **"See Details" Button:**
       - A button on each recipe card/row to open a detailed view of the recipe.

Show Exercises Page

3. **Purpose:** Displays a list of all saved exercises for browsing and interaction.
4. **Elements:**
    - **Exercises Cards/Rows:**
        - Each exercises is displayed as a card or in a list format with:
            - exercise name.
            - image .
        - **"See Details" Button:**
            - A button on each recipe card/row to open a detailed view of the recipe.

Trainee Profile page

Purpose: how the trainee information

Elements:

Statistics of trainee progress such as weight variation ,calories count

Show coaches

Purpose: show the coaches information

Elements:

current trainee , added posts

4.2 Hardware Interfaces

- **Client Devices:**
    - Types: Desktops, laptops, smartphones, tablets.
    - Interaction: Web browsers send HTTP requests to the server for user interactions.
    - Protocol: HTTP with JSON data.

- **Backend Server:**
    - Types: Web server
    - Interaction: Manages data requests, authentication, and business logic.
    - Protocol: HTTP, SQLite for database communication.

4.3 Software Interfaces

- **Backend (Flask):**
    - **Purpose:** Handles business logic and communicates with the SQLite database.

    - **Data Sharing:** Shares user data with HTML, CSS, JS and SQLite.
- **Frontend (**HTML, CSS, JS**):**
    - **Purpose:** UI interaction via REST APIs.
    - **Data Sharing:** Sends requests to Flask and displays user data.
- **Database (SQLite):**
    - **Purpose:** Stores data
    - **Data Sharing:** Flask queries and updates data.

4.4 Communications Interfaces

- **Web Communication:**
    - **Protocol:** HTTP
    - **Data Transfer Rate:** Standard HTTP request/response rates; optimized for low latency.
- **Email Communication:**
    - **Protocol:** SMTP (for sending notifications and alerts)
    - **Message Formatting:** HTML and plain-text formats for notifications.

5. Other Nonfunctional Requirements

5.1 Performance Requirements

- **User Request Response Time:** All requests (e.g., login, profile updates) should be processed within 2 seconds.
- **Page Load Time:** Main dashboard must load within 3 seconds.
- **Real-time Notifications:** Notifications should be delivered within 1 second of the triggering event.
- **Database Query Performance:** Queries should complete within 1 second for up to 1000 concurrent users.
- **File Upload/Download Speed:** Media files should upload/download at a minimum of 5 MB/s.
- **Simultaneous User Load:** The system must support 1000 concurrent active users.
- **Backup and Synchronization:** Data backups every 24 hours with no more than 30 minutes of downtime.

5.2 Safety Requirements

- **Data Protection:** Ensure all user data is encrypted both in transit and at rest to prevent unauthorized access or data breaches.
- **Access Control:** Implement role-based access control to restrict unauthorized users from accessing sensitive information (e.g., trainee or coach data).
- **Backup and Recovery:** Ensure automatic daily backups and provide a recovery plan for restoring data in the event of a system failure.
- **Error Handling:** Implement error logging to alert system administrators of critical failures that could impact user safety or data integrity.
- **Third-Party Compliance:** Adhere to GDPR and other data privacy regulations to ensure user data is handled securely and ethically.
- **Secure File Uploads:** Implement safeguards to prevent malicious files from being uploaded by users, including virus scanning and file type restrictions.
- **User Authentication:** Enforce strong password policies and multi-factor authentication for users accessing the system.

5.3 Security Requirements

- **User Authentication:** Implement multi-factor authentication (MFA) for secure access.
- **Access Control:** Apply role-based access control (RBAC) to restrict data access.
- **Data Encryption:** Encrypt sensitive data in transit (TLS) and at rest (AES-256).
- **Compliance:** Follow GDPR and HIPAA standards for data protection.
- **Secure APIs:** Protect APIs with authentication tokens and prevent vulnerabilities.
- **Data Anonymization:** Anonymize data where applicable to enhance privacy.
- **Security Audits:** Conduct regular security audits and vulnerability assessments.
- **Incident Response:** Implement a plan for handling and mitigating security breaches.

5.4 Software Quality Attributes

- **Usability:** User interface should be intuitive, with 85% user satisfaction.
- **Reliability:** System uptime must be at least 99.5% per month.
- **Maintainability:** Code must be modular, with 80% test coverage.
- **Performance:** Handle 500 concurrent users with a page load time under 3 seconds.
- **Scalability:** Support up to 10,000 concurrent users.
- **Security:** Adhere to OWASP security guidelines.
- **Flexibility:** Easy to add new features with minimal code changes.
- **Correctness:** Ensure 99% accuracy in calculations (e.g., BMI, calorie burn).
- **Testability:** 80% code coverage with automated unit and integration tests.

6. Other Requirements

- **Database Requirements:**
  - Use SQLite database for storage.
  - Ensure data backup every 24 hours.
- **Internationalization Requirements:**
  - The app should support multi-language support in the future (English initially).
- **Legal Requirements:**
  - Comply with GDPR for user data protection and privacy.
  - Ensure fitness-related content adheres to local health regulations.
- **Reuse Objectives:**
  - Utilize open-source libraries for authentication and data handling.
  - Encourage reusability of code modules for future fitness-related applications.

Appendix A: Glossary

API: Application Programming Interface, a set of protocols for interacting with software.
UI: User Interface, the means through which users interact with the app.
Trainee: A user role within the system focused on fitness tracking and training.
Coach: A user role within the system who guides and mentors trainees.
OAuth: Open standard for authentication.
GDPR: General Data Protection Regulation, a European Union law on data protection.