**CS673 Software Engineering**

**Team 6 - FitFusion**

**Project Proposal and Planning**

| Team Member | Role(s) | Signature | Date |
| --- | --- | --- | --- |
| Haoran Zheng | Team Leader | *Haoran Zheng* | 09/20/2024 |
| Jiankun Dong | Design and Implementation Leader | *Jiankun Dong* | 09/20/2024 |
| Chengqin Li | QA Leader | *Chengqin Li* | 09/20/2024 |
| Yuhan Pan | Requirement Leader | *Yuhan Pan* | 09/20/2024 |
| Yunrui Huang | Configuration Leader | *Yunrui Huang* | 09/20/2024 |
| Hangqi Wu | Security Leader | *Hangqi Wu* | 09/20/2024 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Revision history**

| **Version** | **Author** | **Date** | **Change** |
| --- | --- | --- | --- |
| **V1.0** | **Whole Team** | **09/23/2024** | **Updated Team names and roles. Finished baseline version.** |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

[Overview](#_heading=h.30j0zll)

[Related Work](#_heading=h.1fob9te)

[Proposed High level Requirements](#_heading=h.3znysh7)

[Management Plan](#_heading=h.2et92p0)

[Objectives and Priorities](#_heading=h.tyjcwt)

[Risk Management (need to be updated constantly)](#_heading=h.3dy6vkm)

[Timeline (need to be updated at the end of each iteration)](#_heading=h.1t3h5sf)

[Configuration Management Plan](#_heading=h.4d34og8)

[Tools](#_heading=h.2s8eyo1)

[Deployment Plan if applicable](#_heading=h.17dp8vu)

[Quality Assurance Plan](#_heading=h.3rdcrjn)

[Metrics](#_heading=h.26in1rg)

[Code Review Process](#_heading=h.lnxbz9)

[Testing](#_heading=h.35nkun2)

[Defect Management](#_heading=h.1ksv4uv)

[References](#_heading=h.44sinio)

[Glossary](#_heading=h.2jxsxqh)

# Overview

The proposed software system aims to help users efficiently track, plan, and optimize their workouts, catering to fitness enthusiasts, athletes, and beginners. Its key features include workout tracking, routine building, progress visualization, goal setting, and AI-driven exercise recommendations as a stretch goal. The technology stack could include **Vue** for the front end, **Node.js** for the backend, **MongoDB** for data storage, and **ChatAPI** or **Llamda API** for AI suggestions.

# Related Work

Popular fitness tracking softwares includes: Keep, Fitbit, Strava, Apple and Google fitness etc. Most of those softwares are Android/IOS device based, and have limitations due to the portable devices’ limited display capabilities. Our software is web based, providing a more powerful user interface and functionalities. Our website can provide a more dynamic, personalized workout routine that users can adjust based on user performance, while other apps are more static. It is easier to add new functionalities to the site such as AI suggestions and plannings. Other apps have to provide separate implementations for IOS and Android systems.

# Proposed High level Requirements

* 1. Functional Requirements  
     (For each functional requirement, please give a feature title and a brief description using the following format: As (a role), I want to (action), so that (value).)
     1. Essential Feature
        1. **User Registration and Login**:

1. As a user, I want to have a secure and easy-to-use registration and login function, so that I can access my fitness tracking data and progress.
   * + 1. **Personalized Fitness Plan**:
2. As a user, I want to view and manage my personalized fitness plan, so that I can keep track of my past workouts and plan for future exercises.
   * + 1. **Exercise Tracking**:
          1. As a user, I want to log my exercise duration and intensity, so that I can monitor my fitness progress and make adjustments to my workout schedule.
       2. **Push Notifications**:
          1. As a user, I want to receive daily reminders for my workouts, so that I can stay on track with my training schedule and maintain consistency.
     1. Desirable Features (the nice features that you really want to have too):
        1. **Video and Illustration Support**:
           1. As a user, I want access to instructional videos and illustrations for workouts, so that I can improve my exercise techniques and overall performance.
        2. **Social Interaction**
3. As a user, I want to connect with a workout community, so that I can share progress, view others’ performances, and communicate with fellow users.
   * 1. Optional Features (additional cool features that you want to have if there is time):
        1. **Integration with Wearable Devices (e.g., Fitbit, Apple Watch)**:
           1. As a user, I want to sync data from my wearable devices, so that I can consolidate all my fitness data in one place and avoid data loss.
        2. **Workout Suggestions**
           1. As a user, I want access to data-driven workout recommendations, so that I can receive personalized suggestions to enhance my fitness routine.
        3. **Fitness Challenges and Competitions**:
           1. As a user, I want to participate in fitness challenges, so that I can push myself further and compete with others for motivation.
   1. Nonfunctional Requirements
      1. Security requirements:
         1. **Data retention:** User data should be stored securely. Users should have the option to request data deletion.
         2. **Role-Based Access Control (RBAC)**:The platform should support role-based access control, allowing users with specific roles (e.g., moderators, maintenance staff) to have administrative privileges over regular users.
         3. **Authentication & Authorization**: Secure login functionality should be implemented, allowing users to authenticate and access their stored data. OAuth support is recommended to enhance security.
      2. Performance requirements:
         1. **Response Time**: The website should respond to user requests in a reasonable time.
         2. **User traffic handling**: The website should be able to handle a reasonable amount of users at the same time.
      3. Usability requirements:
4. **Intuitive User Interface**: The website should have an intuitive user interface.
5. **User Onboarding and Help**: The website should provide a straightforward and quick onboarding for users.
   * 1. Reliability requirements:
        1. **Backup**: The website should have a way to backup user data in case of critical failures.
        2. **Uptime**: The website should have a reasonable percentage of uptime, considering the resources available.
     2. Maintainability requirements:
        1. **Modular Code**: The codebase for the website should be modular, allowing for easy updates and debugs.
        2. **Testability**: The codebase should be tested and testable. Tests should cover a reasonable percentage of the codebase.
        3. **Documentation**: The codebase should have sufficient and clear documentation to ensure easy maintenance.
     3. Scalability requirements:
        1. **Cloud Infrastructure**: The codebase should be designed with integration of AWS in mind, for potential scaling.

# Management Plan

## Objectives and Priorities

Main objectives: User-friendly interface; Goal setting function; unique tracking and analyzing on multiple training programs;

Priorities to do: Keep user’s data safe; Ensure user experience;

Update periodically to keep our product’s accessibility and scalability.

## Risk Management (need to be updated constantly)

**Main risks and methods:** As some of the team members are not familiar with certain tech stacks and the whole workflow of creating a project from the very beginning, much time is required to learn the lifecycle of a SE project. Also, with many different parts and components(frontend, backend, container, database, connection and possible mobile development), we decide to meet more frequently outside classes and between iterations, and keep personal progress updates on Github branches and Jira task management.

**Risk Management Sheet Link: https://docs.google.com/spreadsheets/d/1tlETEcsiV0036xuAtBX1AIFRGq4aJeGM/edit?pli=1&gid=1924349204#gid=1924349204**

## Timeline

Iteration due dates:

Iteration 1: Otc. 17

Iteration 2: Nov. 7

Iteration 3: Dec. 5

| Iteration | Functional Requirements(Essential/Disable/Option) | Tasks (Cross requirements tasks) | Estimated/real person hours |
| --- | --- | --- | --- |
| 1 | Essential: Login UI; Exercise tracking.  Target: Design workout suggestions and personalized fitness plans. | Frontend design, Training plan survey, User story design. | Estimated hours:  6-8 hours/person.  Real hours: |
| 2 | Essential: Fitness plan builder; User registration; Data storage and security.  Target: Calling AI Api to give training suggestions; More user-friendly function | Frontend design and improvement, Building data-storage database, Backend initiating, Function testings. | Estimated hours:  8-10 hours/person.  Real hours: |
| 3 | Essential: Push notification for recording;  Future: Design Apps or integration with wearable devices; Adding social interaction functions. | Connecting front/backend + DB, Practical service deployment. | Estimated hours:  8-10 hours/person.  Real hours: |

# Configuration Management Plan

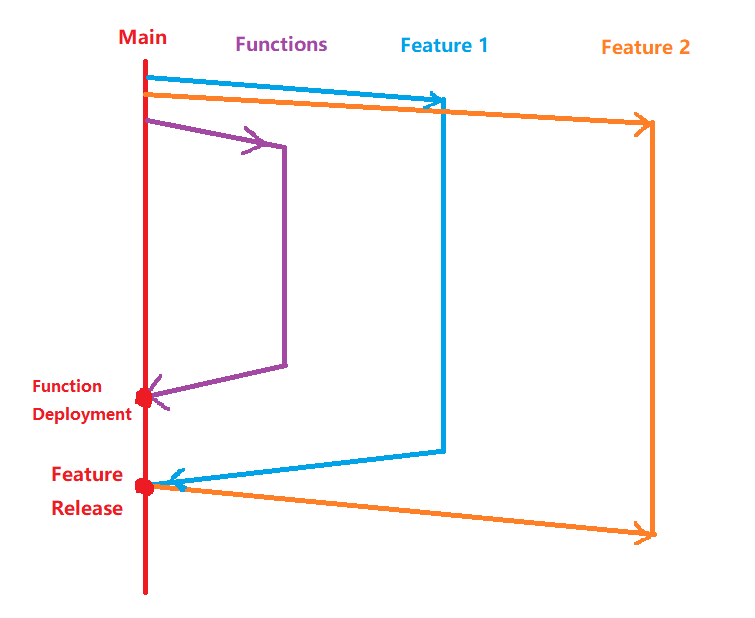
## Tools

(In this project, we will use Git and Github as the version control tools. Please also specify any other tools to be used, e.g. IDE tools, CI/CD tools, container tools, SAST or DAST tools, and any other DevOps tools)

* + 1. **Version Control**: **Git & GitHub**
    2. **IDE Tool: Visual Studio Code**
    3. **Containerization: Docker**
    4. **Database: MangoDB**
    5. **DevOps Tools: Jira, AWS**
  1. Code Commit Guideline and Git Branching Strategy

We are to use a trunk based development strategy, as shown in the figure below. We shall have a **Main** branch to keep the released code, a **develop** branch for integration of features, and **Feature** branch for each feature or task development. We shall add **hotfix**  branches if those are necessary.

Code should be committed to feature branches often, and a descriptive comment should be included for each commit. When feature development is finished and testing done, a peer review session should be completed before a pull request is merged. We shall use squash and merge for the merge strategy.



## CI/CD Plan if applicable

N/A

# Quality Assurance Plan

## Metrics

(Describe the metrics to be used in the project to measure the quality of your software. Each metric should be measurable and quantifiable. Examples of metrics include product complexity (LOC, # of files, # of classes, # methods, cyclomatic complexity, etc.) , defect rate (# of defect per KLOC), # of test cases, test case pass rate, cost (# of person hours used), # of user stories completed, etc. **The result of these metrics should be reported in the progress report/ iteration summary sheet.**)

| Metric Name | Description |
| --- | --- |
| Lines of Code (LOC) | Measures the size of the codebase in terms of the number of lines written. Helps to monitor complexity and track the growth of the project. |
| Cyclomatic Complexity | Measures the complexity of the code by counting the number of independent paths through the code. Lower values indicate simpler, more maintainable code. |
| Defect Rate (Defects/KLOC) | The number of defects found per 1,000 lines of code. This will help track code quality and identify areas of the system that need improvement. |
| Test Case Pass Rate | The percentage of test cases that pass during automated and manual testing phases. Indicates the overall stability of the application. |
| Cost (Person-Hours) | The total number of person-hours spent on the project, helping to assess effort and productivity. |

* 1. Coding Standard

(Describe any coding standard to be used)

**Languages**: JavaScript/TypeScript (for frontend and Node.js)

## Code Review Process

(Everyone should review all documents to be submitted. Here you will mainly describe how the code review will be done. Who will review the code, e.g. design or implementation leader will review all code or team members review each other’s code. Do you use pull requests for the code review? Is there a checklist to help review? What feedback should the reviewer provide?)

**Who Reviews the Code**:

For each function implementation , we will select out a main code reviewer to examine the whole process and results. Other remaining developers are responsible for checking the consistency and quality of other peer’s code sections and works.

The person that develops the test cases for said functionality should not be the main reviewer.

**Pull Request Process**：

All code changes will be submitted through **Pull Requests (PRs)** on GitHub.

At least **one peer** must review each pull request before merging into the develop branch.

PRs must pass all automated tests and static analysis checks before merging.

**Review Checklist**:

Does the code follow the coding standard?

Are there sufficient unit/integration tests for the new code?

Is the code well-structured and easy to understand?

Does the code introduce any potential security vulnerabilities?

Are all edge cases considered?

**Feedback**:

Reviewers will provide constructive feedback through GitHub PR comments.

If major changes are needed, the code will be revised and resubmitted for review.

## Testing

(Both manual testing and automated testing should be considered. Both unit testing and integration testing should be considered. Briefly describe the testing tools/framework to be used, the personnel involved (e.g. the QA leader will focus on the integration testing and each developer will unit test their own code), when and what types of testing will be performed, the testing objectives, etc)

We will mainly focus on the automation testing for GUI and REST-Api. We will test the database via manually.

Testing Tools and Function:

1. For the Web GUI testing, we will use Selenium to implement the automated testing
2. For the REST API, we will use the Postmen for manual testing and Jmeter for automated testing
3. For the Unit test, we will use Jest to implement the automated testing
4. For the Database, we will manually create the testing data and testing

## Defect Management

We will use **Github Issues** to manage defects. **Github Issues** offers an integrated platform where team members can report detects, track progress, assign responsibilities, collaborate and integrate with codebase.

Types of defects:

1. Functional Defects:
   1. Feature bugs: When a feature doesn’t work as intended.
   2. Missing Functionalities: Absence of features specified in the requirements
2. Performance Defects:
   1. Slow Response Times: Actions taking longer than the acceptable threshold.
   2. High Resource Usage: CPU or memory consumption impacting user experience.
3. Security Vulnerabilities:
   1. Authentication Flaws: Issues with login, OAuth integration or session management.
   2. Data Exposure: Unencrypted data/improper data handling
   3. Attack vulnerability: other security vulnerabilities that make the website weak to attacks.
4. UI/UX Defects:
   1. Layout issue: Misaligned elements, unreadable text, inconsistent styles and any other similar issues.
   2. Navigation issues: Dead links and bad menus that hinders user flow.
5. Compatibility Errors:
   1. Browser Compatibility: Feature not working across different web browsers.
6. Integration Defects:
   1. API failure: Problems with API.
   2. Data Synchronization: Inconsistencies between server and client-side data
7. Data Handling Defects:
   1. Data Loss: User data not being saved
8. Documentation Errors:
   1. Missing information: Lack of necessary guidance or help content.
   2. Inaccurate information: Outdated or incorrect documentation.

**Action and personnel:**

**Defect Reporting:**

Personnel Involved: Everyone

Action:

1. Identify and document defects in GitHUB Issues with clear titles and detailed descriptions.
2. Include steps to reproduce, with screenshots of videos that shows the defects.

**Monitoring and planning:**

Personnel Involved: QA Lead, Team Leader

Action:

1. Continuous monitor and be alert to new defects.
2. Understand defects’ severity and impacts
3. Assign new defects to SPRINT for implementation

**Defect Assignment:**

Personnel Involved:Design/Implementation Lead, Team Leader

Action:

1. Assign defects to developer familiar with the affected component
2. Set due dates aligned with project timelines and defect severity.
3. Assign Test engineer for testing of the defect resolution.

**Defect Resolution:**

Personnel Involved: Assigned Developers

Action:

1. Analyze the root cause of the defect.
2. Create a new branch to implement code fixes.
3. Reference the Github issue number in the commit message for traceability.

**Test of resolution:**

Personnel Involved: Test Leader, Test engineer

Action:

1. Test the fixed defect.
2. Execute regression tests to make sure the rest of the system are unaffected.
3. Update Github issue to reflex and confirm that defect is resolved.

**Code Review and Merge:**

Personnel Involved: All developers

Action:

1. Review the code changes submitted for defect fixes.
2. Ensure the solution is effective and does not introduce new issues.
3. Verify compliance with coding standards and security guidelines.
4. Merge the branch.

**Defect Closure:**

Personnel Involved: Test Leader, Project Manager

Action:

1. Close the Github issue once the fix is verified and merged into the main codebase
2. Update related documentation
3. Let the team know that defect is closed

(Describe the tool to be used to manage the defect (e.g github issues). The types of defects to look at. The actions or personnel for defect management. )

# References

(For more details, please refer to the encounter example in the book or the software version of the documents posted on blackboard. )

* 1. <https://www.flagship.io/git-branching-strategies/>
  2. https://github.com/cfilipov/MuscleBook.net

# Glossary

(Any acronym used in the document should be explained here)

UI: User Interface

UX: User Experience

VSC: Visual Studio Code