**CS673 Software Engineering** 

**Team 3 - Project Name**

**Project Proposal and Planning**

| Team Member | Role(s) | Signature | Date |
| --- | --- | --- | --- |
| Zuowen Tang | Security Leader | *Z.T.* | 5/12/23 |
| Jasmine Hughes | QA Leader | *Jasmine Hughes* | 5/12/23 |
| Alekhya Koppineni | Requirement leader | [Alekhya Koppineni](mailto:alekhya.koppineni@gmail.com) | 5/12/23 |
| Edara Umamaheswar | Team Leader | *Mahesh* | 5/15/23 |
| Adithya Prakash | Design and Implementation Leader | *Adithya Prakash* | 5/15/23 |
| Corydon Wood | Configuration Leader | *Corydon Wood* | 5/15/23 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Revision history**

| **Version** | **Author** | **Date** | **Change** |
| --- | --- | --- | --- |
| **1.0** | **Edara Umamaheswar** | **5/16/23** |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

[Overview](#_g6igqliy7rm)

[Related Work](#_bf21eadgjj29)

[Proposed High level Requirements](#_rgyo4hi9stmq)

[Management Plan](#_ts358bsdtbcv)

[Objectives and Priorities](#_nxeeppkjxgn4)

[Risk Management (need to be updated constantly)](#_tk7yixobah8p)

[Timeline (need to be updated at the end of each iteration)](#_iksrndohvx29)

[Configuration Management Plan](#_j5uvivmxqcsp)

[Tools](#_dzly5b9kz982)

[Deployment Plan if applicable](#_sd8zu6r3jisd)

[Quality Assurance Plan](#_vra5ptwu59qx)

[Metrics](#_vwjduhc9wuah)

[Code Review Process](#_hx3eaiwb8v3m)

[Testing](#_l9xnpmd6hh0y)

[Defect Management](#_5amsh8h9f0c7)

[References](#_pd9euov6m4du)

[Glossary](#_ty3i2nqffhtc)

# Overview

Our proposed software system is a Calorie Tracking Application designed to help users monitor their daily calorie intake and make informed dietary choices. The motivation behind this project stems from the increasing importance of maintaining a healthy lifestyle and the need for a convenient tool to track nutritional information.

The purpose of the Calorie Tracking Application is to provide users with a user-friendly interface to log their food intake and calculate their calorie consumption. By tracking their calories, users can better understand their eating habits, set goals, and make adjustments to achieve their desired health targets.

The potential users of this software system are individuals who are conscious about their health and fitness. This includes people looking to manage their weight, athletes monitoring their nutritional intake, and individuals with specific dietary requirements.

The basic functionality of the proposed software system includes user registration and authentication, food database management, calorie tracking, meal planning, and reporting and analytics. Users can create accounts, log their food consumption, view nutritional information, set calorie goals, and access reports and insights to monitor their progress.

The technology stack for this project could include Python as the primary programming language, the Django framework for web development, and a relational database management system like PostgreSQL to store user data and the food database.

# Related Work

During our online research, we had come across several similar software systems that focus on calorie tracking and nutrition management. Some notable examples include MyFitnessPal, Lose It!, and FatSecret.

While these software systems share the common goal of helping users track their calorie intake, there are distinct differences between our proposed software system and these existing solutions:

1. Customization and Flexibility: Our software system aims to provide users with a high level of customization and flexibility in tracking their calorie intake. We plan to implement a feature that has the ability to add custom food items. This level of customization sets our software system.
2. User Experience and Interface: We prioritize providing a user-friendly and intuitive interface for a seamless user experience. Our focus is on creating a clean and visually appealing interface that makes it easy for users to log their food, view nutritional information. We aim to simplify the process and ensure a positive user experience throughout the application.

# Proposed High level Requirements

* + 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).)
    - Essential Features (the core features that you definitely need to finish):

(For each essential features, please give a rough estimation in terms of person hours or an range of person hours)

* + - Desirable Features (the nice features that you really want to have too):
    - Optional Features (additional cool features that you want to have if there is time):

High-Level Requirements for Calorie Tracking Application:

Functional Requirements:

1. User Registration and Authentication:
   * As a user, I want to be able to create an account and securely log in, so that application can be accessed and track calorie intake securely.
2. Daily Food Logging:
   * As a user, I want to be able to log my daily food intake by selecting food items from a database or adding custom entries, so that calorie consumption is tracked accurately.
3. Calorie Calculation and Tracking:
   * As a user, I want the application to calculate and display the total calorie intake for the day based on the logged food items, so that calorie intake is effectively monitored and managed.
4. Food Database Management:
   * As an administrator, I want to be able to manage the food database by adding, editing, and deleting food items, so that users can access a comprehensive list of food options.
5. Meal Planning and Goal Setting:
   * As a user, I want the ability to set my calorie intake goals based on my weight or health targets and plan my meals accordingly, so that desired goals are achieved.
6. Social Features:
   * As a user, I want to connect with other users, share my progress, and provide support, such as liking or commenting on their activities, so that community interaction is present to stay motivated and to support others.

Essential Features:

1. User Registration and Authentication - 8-12 hours
2. Daily Food Logging - 10-15 hours
3. Calorie Calculation and Tracking - 8-12 hours
4. Food Database Management - 6-16 hours
5. Meal Planning and Goal Setting - 12-18 hours

Desirable Features:

1. Integration with Fitness Trackers or Apps - 10-15 hours
2. Barcode Scanner for Food Item Input - 8-12 hours
3. Recipe Tracking and Calculation - 12-18 hours

Optional Features:

1. Social Media Integration - 6-10 hours
2. Meal and Recipe Sharing - 10-15 hours
3. Customizable Meal Plans - 12-18 hours  
   * Nonfunctional Requirements
     + Security requirements

* Authentication: secure authentication to ensure that user accounts and sensitive information are protected with appropriate access to admins and users
* Secure Transmission: encrypted communication using secure protocols such as HTTPS.
* Data Privacy: follow appropriate data protection regulations measures to ensure the privacy and confidentiality of user data.

# Management Plan

## Objectives and Priorities

1. Complete all Proposed Essential Features: The highest priority objective of our project is to ensure the successful implementation and completion of all essential features outlined in the project plan. These features include user registration and authentication, calorie tracking, meal planning, and food database management. By prioritizing the completion of these features, we aim to deliver a functional and comprehensive calorie tracking application to the client.
2. Ensure Software Quality and Reliability: Another key objective is to maintain a high level of software quality and reliability. This includes thorough testing of the application to identify and fix any potential bugs or issues. We will follow best practices in software development, including code reviews, unit testing, and continuous integration, to ensure that the software meets industry standards and provides a stable user experience.
3. User-Focused Design and User Experience: We prioritize creating a user-focused design and delivering an excellent user experience. Our objective is to develop an intuitive and user-friendly interface that makes it easy for end users to navigate through the application, log their food intake, track their calorie consumption, and access relevant information and insights. We will conduct user testing and gather feedback to iterate and improve the user interface and experience.
4. Successful Deployment: We aim to successfully deploy the software and make it accessible to end users. This includes setting up the necessary infrastructure, configuring the hosting environment, and ensuring the application runs smoothly.
5. Documentation and Knowledge Transfer: An important objective is to provide comprehensive documentation to support the software. This includes technical documentation for developers, user guides for end-users, and any necessary documentation for system administrators. Additionally, we will facilitate knowledge transfer within the team to ensure that team members are equipped with the necessary information and skills to maintain and support the software.

## Risk Management (need to be updated constantly)

Our team has identified several key risks that may impact the successful completion of the project. These risks include potential delays in development, technical challenges, team member availability, and scope creep. To effectively manage these risks, we discussed a risk management plan that involves regular monitoring, mitigation strategies, and contingency plans.

1. **Development Delays**

One of the main risks is the potential for development delays due to unforeseen technical issues or complexities. To manage this risk, we will conduct thorough research and planning in the early stages to identify potential challenges. We will also allocate sufficient time for testing and bug fixing to address any issues that arise during development.

1. **Technical Challenges**

Developing a calorie tracking application may involve integrating with external APIs, handling complex calculations, and managing large amounts of data. To mitigate this risk, we will conduct a comprehensive technical analysis to identify potential challenges and ensure that the chosen technology stack is suitable for the project's requirements. We will also allocate resources for learning and skill development to address any technical gaps.

1. **Team Member Availability**

Availability and commitment of team members can impact project progress. To manage this risk, we will maintain open communication channels to address any scheduling conflicts or personal commitments that may arise. We will also establish a contingency plan by cross-training team members to ensure that essential tasks can be carried out even in the absence of a specific team member.

1. **Scope Creep**

There is a risk of scope creep, where additional features or requirements may be added during the development process, leading to project delays and increased workload. To mitigate this risk, we will define and document the project scope clearly at the outset. Any proposed changes or additions will be evaluated against project goals and priorities, and a formal change control process will be implemented to manage scope changes effectively.

**Risk Management Sheet Link:** [team3 - CS673\_SPPP\_RiskManagement](https://docs.google.com/spreadsheets/d/1F7NgvEv7hx3lGqsx5UF1zMXdne4xvG91uIpnvxyj8Pw/edit?usp=share_link)

## Timeline (this section should be filled in iteration 0 and updated at the end of each later iteration)

| Iteration | Functional Requirements(Essential/Disable/Option) | Tasks (Cross requirements tasks) | Estimated/real person hours |
| --- | --- | --- | --- |
| 1 | 1. User Registration and Authentication  2. Daily Food Logging  3. Calorie Calculation and Tracking | Task 1.1: Implement user registration functionality  Task 1.2: Implement secure login and authentication mechanisms  Task 2.1: Design and develop the user interface for food logging  Task 2.2: Implement database integration for food item selection  Task 2.3: Develop the functionality to add custom food entries  Task 3.1: Implement algorithms to calculate calorie intake based on logged food items  Task 3.2: Display total calorie intake for the day in a user-friendly format | 50 |
| 2 | 1. Food Database Management  2. Meal Planning and Goal Setting  3. Integration with Fitness Trackers or Apps | Task 1.1: Design and develop an admin interface for managing the food database  Task 1.2: Implement functionality to add, edit, and delete food items  Task 2.1: Develop a user interface for setting calorie intake goals and planning meals  Task 2.2: Implement functionality to track progress towards goals and provide recommendations  Task 3.1: Research and identify compatible fitness trackers or apps  Task 3.2: Develop integration functionality with the selected fitness tracker or app | 50 |
| 3 | 1. Barcode Scanner for Food Item Input  2. Recipe Tracking and Calculation  3. Social Media Integration | Task 1.1: Research and select a barcode scanning library or API  Task 1.2: Integrate the barcode scanning functionality into the application  Task 2.1: Design and develop a user interface for recipe input and tracking  Task 2.2: Implement functionality to calculate total calories for recipes and individual servings  Task 3.1: Research and integrate social media APIs for sharing functionality  Task 3.2: Develop user interface components for social media interaction | 100 |

# Configuration Management Plan

## Tools

* Diagraming: Lucid Chart
* CI/CD: GitHub Actions
* Testing: Pytest
* Linting: Flake8
* Project management (requirement tracking, task management): Jira
* Version Control: GitHub
* Virtual Whiteboard: Miro
* Video Meeting: Zoom
* Chat: Discord
* Container Engine: Docker
* Container Image Repository: Docker Hub
* Hosting: Heroku
* Code Documentation: GitHub Site
* API Documentation: OpenAPI standard
* API Testing: Postman
* IDE: Developer’s preference, PyCharm, VS Code, etc.
  + Code Commit Guideline and Git Branching Strategy   
    For this application a GitFlow branching strategy will be used. Due to some members of the team being new to the framework as well as CI/CD processes. Combining this factor with the short iteration cycles GitFlow should allow the team to independently work on features, test and integrate them back into the main branch without too much configuration overhead or reliance on pre-existing experience. Feature will be developed in their own branch which will be derived from the development bench rather than the main branch. This strategy is commonly used and is described further by Atlassian [here](https://www.atlassian.com/git/tutorials/comparing-workflows/gitflow-workflow) .

## Deployment Plan if applicable

The application will be deployed using a set of Docker containers for the different components. The web application will be a Linux based Docker container so that the environment is consistent from development to production. Containers will be deployed to Heroku as the hosting platform, however AWS is an alternative depending on the performance with Heroku. Depending on the final design the frontend may be deployed using a pre-defined Apache or Nginx Docker image. In order to provide efficient deployments and reduce manual processes we will aim to develop a CI/CD pipeline in the first iteration which will complete the code quality checks then move on to the image build and finally deploy the containers to the hosting platform.

# 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 |
| --- | --- |
| Search Functionality | Tests what foods are fetched when specific foods, nutrients, ingredients, etc. A quality rating could be given to search results as a measure  I.e. Flour will fetch bread and pizza |
| Test Effectiveness | This will check how effective the tests being run are.  I.e. Number of bugs detected in 1 test / (total number of bugs found in tests + bugs found after release) X 100 |
| Saving Features | Tests the various mechanics that allow a user to save something and review it at a later time.  I.e. A user saves a meal to their weekly meal planner |
| Added Foods Detect Rate | Checks whether added food remains in the user’s database when a user tries to add the food to a different day |
| Feature Coverage | Checks how many features have been implemented compared to the number of features to be added I.e. Number of features added / Number of features to be added x 100 |
| Food Catalog Coverage | Checks how many food items/ingredients are included in the catalog for users to search  I.e. (Number of foods added / Ideal number of foods to be added x 100 |
| Time Spent Addressing Defects | Checks the amount of time (and therefore money) spent addressing/fixing/remediating reported and found defects by humans |

* + Coding Standard
    - Use Underscore case for all variables : Following a consistent naming convention, such as using underscore case for variables (e.g., my\_variable\_name), can enhance code readability. Clear and meaningful variable names make it easier to understand the purpose and usage of each variable, improving code comprehension and maintainability.
    - Commenting : It is important to add comments to our code, especially for functions or methods, to provide clarity and context. Comments help explain the purpose of the function, its parameters, return values, and any important implementation details. Well-documented code is easier to understand, maintain, and debug for developers.
    - Write readable and efficient code: Writing code that is readable and easy to understand is crucial for collaboration and long-term maintenance. Use proper indentation, whitespace, and formatting to improve code structure. Break down complex tasks into smaller functions or modules, and keep the code logic clear and straightforward.
    - Use helper methods wherever necessary: Helper methods or utility functions can encapsulate reusable pieces of code. By separating specific functionalities into helper methods, you can improve code modularity, reusability, and readability. This approach helps in reducing code duplication and enhances the maintainability and scalability of our codebase.
    - Writing test cases: Test cases are essential for ensuring the correctness and reliability of our code. Writing comprehensive unit tests helps catch bugs and regressions early in the development process, improving the overall quality of our code. Test cases validate the expected behavior of our functions and provide confidence that they work as intended. Consider using a testing framework (e.g., JUnit for Java) to automate the execution of test cases and provide clear feedback on the code's correctness.

## Code Review Process

1. Code Review Participants: Design & implementation leaders will review code submissions.
2. Pull Request Workflow: All code changes should be submitted as pull requests to the designated repository. Pull requests serve as a formal submission for review and facilitate collaboration and feedback.
3. Reviewer Roles and Responsibilities: The Reviewers should thoroughly examine the code for design coherence, adherence to coding standards, efficiency, and potential issues. Reviewers should provide constructive feedback and suggestions to help improve the code.
4. Code Review Checklist:

* Readability and Maintainability:
  + Are variable and function names descriptive and meaningful?
  + Is the code properly indented and formatted for readability?
  + Are comments clear and concise, explaining complex logic or providing context
  + where necessary?
  + Are there any code blocks that could be refactored for improved readability or maintainability?
* Coding Standards and Best Practices:
  + Does the code adhere to the established coding standards and guidelines?
  + Are naming conventions consistent throughout the codebase?
  + Are there any unused variables or redundant code that can be removed?
  + Is error handling implemented appropriately, including proper exception handling and error messages?
* Performance and Efficiency:
  + Are there any inefficient algorithms or resource-intensive operations?
  + Are database queries optimized, avoiding unnecessary calls or redundant data retrieval?
  + Are loops and iterations designed to minimize computational complexity?
  + Are there any memory leaks or resource management issues that need attention?
* Security Considerations:
  + Are there any potential vulnerabilities, such as SQL injection or cross-site scripting (XSS) vulnerabilities?
  + Are sensitive data properly protected, such as passwords or API keys?
  + Are input validations and sanitizations implemented to prevent malicious inputs?
  + Are access control and authorization mechanisms correctly implemented?
* Documentation and Comments:
  + Is the code adequately documented, providing explanations for complex logic or business rules?
  + Are there any missing or outdated comments that should be added or updated?
  + Is the documentation up to date, including API references, code usage examples, and README files?

5. Feedback Constructive and Specific:

* + Feedback should focus on identifying areas of improvement and providing actionable suggestions.
  + Comments should be specific, highlighting specific lines of code or functionality that require attention.
  + Avoid general or vague feedback that does not provide clear guidance for improvement.
* Addressing Bugs and Vulnerabilities:
  + Reviewers should actively look for potential bugs or vulnerabilities in the code.
  + If any issues are identified, reviewers should clearly communicate the problem and suggest possible solutions.
  + Reviewers have to prioritize security concerns and highlight any potential security vulnerabilities they come across.
* Attention to Critical Areas:
  + Reviewers should pay close attention to critical areas of the codebase, such as complex algorithms, sensitive data handling, or performance-critical sections.
  + Any potential performance bottlenecks or optimization opportunities should be identified and communicated.
  + Reviewers should ensure that error handling and exception scenarios are properly addressed.

## Testing

Pytest will be the primary testing tool used for the application. There is a plugin for Django that can be implemented for that specific tech stack. Pytest is well documented for its ability to cover various testing styles (Functional, unit, end-to-end, integration, etc.).

The personnel that will be involved in the testing includes everyone (since we all will be implementing some amount of code). Each lead will be responsible for their own testing of implemented features with the QA Lead assisting in the testing process as needed. The Team Lead will be responsible for the final checks and approval.

Regression testing will be used after every iteration with a focus on integration testing. Unit testing should be completed before and after every commit on GitHub. No longer than an hour should be spent on unit testing. No longer than three hours should be spent on integration testing. Functional testing will be completed by the Design and Implementation Leads supervised by the QA Lead with a final check by the Team Lead.

## Defect Management

We will use github as our defect management tool.Issues should be opened on the github branch to report a defect. Everyone can report defects they found to the QA leader. The QA leader will record the defect in Jira and assign it to the relative person to solve it. The developer addressing the defect will create a defect branch to make the necessary change, and provide updates on the Jira issues that was opened. Once the defect has been resolved and tested the defect branch will be merged back into the main branch and then removed. The issue where the defect was reported will be closed with a comment about the fix and in what version it was addressed.

# References

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

# Glossary

QA - Quality Assurance