**CS673 Software Engineering** 

**Team 5 - Meal It**

**Project Proposal and Planning**

| Team Member | Role(s) | Signature | Date |
| --- | --- | --- | --- |
| David Vanderschaaf | Project Lead | *David Vanderschaaf* | 9/21/2022 |
| Alaap Bharadwaj | Requirement Lead | *Alaap Bharadwaj* | 9/21/2022 |
| Fahmida Joyti | Configuration Lead | *Fahmida Joyti* | 9/18/2022 |
| Xiaohang Lin | QA Lead | *Xiaohang Lin* | 9/21/2022 |
| Wenxiao Zhou | Design and Implementation Lead | *Wenxiao Zhou* | 9/21/2022 |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

**Revision history**

| **Version** | **Author** | **Date** | **Change** |
| --- | --- | --- | --- |
| 1.0 | Whole Team | 9/21/2022 | Initial |
|  |  |  |  |
|  |  |  |  |
|  |  |  |  |

[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 team is developing and deploying a meal preparation web application. This app will allow users to figure out their meals in two different ways: by finding a recipe based on ingredients they already have on hand, or by dynamically creating a grocery list of ingredients they don’t have to be sent to a third party grocery delivery service. This recipe web application will also be integrated into a calorie counting system for our calorie conscientious users.

We plan to use Java and Spring Boot to create this web application that will then be deployed using Heroku.

# Related Work

There are several applications out there that do portions of proposed app, but none that do all of them

* My Fridge Food - <https://myfridgefood.com/>
  + Find recipes based on specific ingredients
  + No ability to push missing items to a grocery list
  + No calorie tracking
* Super Cook - <https://www.supercook.com/#/desktop>
  + Find recipes based on specific ingredients
  + No ability to push missing items to a grocery list
  + No calorie tracking
* MyFitnessPal - <https://www.myfitnesspal.com/>
  + Track calories
  + Can find recipes
  + No ability to find recipes based on ingredients

# Proposed High level Requirements

**Functional Requirements**

***Essential Features***

1. **Log In/Sign up Feature**

As a user, I want to be able to log in to my account, so that my information can be secure and my preferences saved

*Acceptance Tests:*

* User is able to open application and enter a username and a password
* User is able to sign up with new credentials

Time: 25 person-hours

1. **Recipe Filter**

As a user, I want to be able to filter recipes by ingredients, so that I can narrow down my search easily

This feature will be used in two places, one to retrieve recipes from ingredients inputted, and two to retrieve recipes which the users choose from to create a grocery list

*Acceptance Tests:*

* User is able to add dietary restrictions and cuisine selections to filter out the results for recipes

Time: 30 person-hours

1. **Saving previous orders**

As a user, I want to be able to save previous orders, so that I can save previous recipes, previous grocery lists, etc to make it easier to access my history.

*Acceptance Tests:*

* User will be able to view their previous choices regarding either the ingredients or recipes
* User will be able to view their account information and update if necessary

Time: 15 person-hours

1. **Saving preferences such as dietary restrictions**

As a user, I want to be able to save my preferences, so that I do not have to select these preferences every single time I look for recipes.

*Acceptance Tests:*

* Able to view recipes without inputting the preferences every time

Time: 15 person-hours

***Desirable Features***

1. **Google Sign-In**

As a user, I want to be able to use my Google Sign-in to access my account, so that I don’t have to create or re-enter credentials when logging in.

*Acceptance Tests:*

* User is successfully able to sign up to the application using their google account

Time: 15 person-hours

1. **Calorie Counting**

As a user, I want to be able to save the calories I consume in my recipes, so that I can track my daily calorie intake.

*Acceptance Tests:*

* User is able to add food they have eaten that day which then adds a certain calorie amount to their total

Time: 15 person-hours

***Optional Features***

1. **Sending Grocery List to third party store to purchase groceries**

As a user, I want to be able to push my grocery list to a grocery delivery website, so that I can have my groceries delivered to my door.

*Acceptance Tests:*

* User is able to go to another site with all the ingredients in the cart and purchase

Time: 50 person-hours

**Nonfunctional Requirements**

1. **Security Requirements**

This requirement covers areas such as no memory leaks, and making the information displayed to the user as minimal as possible and only what is necessary, this will help reduce the vulnerability of the application.

Time: Throughout the project

# Management Plan

## Objectives and Priorities

1 - Maintain High Quality

2 - Complete all essential features

3 - Become proficient creating Java Web Applications using Spring Boot Framework

4 - Deploy the software successfully

5 - Complete all Desirable Features

6 - Complete all Optional Features

## Risk Management (need to be updated constantly)

The largest risks identified for this project are lack of familiarity with the Spring Boot framework, improper design, and not enough testing.

**Risk Management Sheet Link:** [CS673\_SPPP\_RiskManagement\_Team5](https://docs.google.com/spreadsheets/d/1twigujAgIk4K7zWFkQnpr3FnwbDjCQ1509r3RfrdUEs/edit?usp=sharing)

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

| Iteration | Functional Requirements (Essential/Desirable/Option) | Tasks (Cross requirements tasks) | Estimated/real person hours |
| --- | --- | --- | --- |
| 1 | * Design/Code Recipe Filter | * Create UML design of application * Create Spring Boot Skeleton * Integrate Git repository with IDE * Create wireframes for the user interface and do initial setup of the UI * Configure Database (mySQL) * Presentation creation | 100 person-hours |
| 2 | * Design/Code Sign-In * Design/Code saving previous orders * Design/Code Save Preferences | * Setting up CI/CD * Design and implement feature specific classes/methods * UI Development * Unit Test cases * Presentation creation | 95 person-hours |
| 3 | * Design/Code Calorie Counting * Design/Code Push to 3rd Party * Design/Code Google Sign-In | * Unit Test cases * UI Development * Presentation creation * Design and implement feature specific classes/methods | 110 person-hours |

# Configuration Management Plan

## Tools

* Git/Github will be used for version control.
* IntelliJ for IDE
* Maven will be used for managing all the dependencies in the project.
* Pivotal tracker will be used to track the progress of the project.
* Google docs will be used to keep track of any changes to the project documentation.
  1. **Code Commit Guideline and Git Branching Strategy**

## Git branching strategy - Branches [1]

* + - 1. *Main:* This will be the main release branch for the project. When it is time for a release, the dev branch will be merged into the main branch.
      2. *Dev:* This will be the stable branch. Once the members are done working on their branch the changes will be merged to the dev branch.
      3. *Feature branches:* Team members will work on separate feature branches. The branches will use the following naming convention: fahmida\_authentication\_flow, where “fahmida” is the name of the team member working on the feature and “authentication\_flow” is the feature the member is currently working on. When team members work on new features they will create the new feature branch from the dev branch.
      4. When a team member is done working on the feature the team member will make a pull request that will be reviewed by at least two members of the team.
      5. Once the pull request is approved by the reviewers it will be merged into the dev branch.

## Code commit guidelines:

* + - 1. Code changes need to be approved by the reviewers before it can be committed to the dev branch.
      2. Code can only be committed to the dev branch after testing that the code is working as intended or other members’ of the team require that code to test the functionality they are working on. Reviewers will check that the new code has sufficient testing.
      3. Pull request (PR) should describe the feature the member has been working on, motivations for introducing the feature and link to the story in pivotal track.
      4. Once PR has been approved it should be merged with an appropriate commit describing the changes along with the team member name who worked on the feature.
      5. Code should be readable and have adequate comments so that other team members can understand it.

## Deployment Plan

* The main branch code will be deployed to Heroku using github actions for each iteration.
* The CI/CD pipeline gets triggered when there is a push to the main branch.
* It will run maven clean install and perform the steps required to deploy the application to Heroku.
* If there is an error during deployment, the deployment will be rolled back to the previous version.

# Quality Assurance Plan

## Metrics

| Metric Name | Description |
| --- | --- |
| LINE OF CODE | Number of LOC(lines of code). This shows the readability and complexity of the software (10,000-50,000). |
| DEFECT RATE | Number of defects per KLOC. This measures the software’s reliability(less than 10 per iteration) |
| TEST CASES | Number of test cases. This helps evaluate how extensive the testing is for the software（more than 20) |
| PASS RATE | Percentage of tests that pass. This indicates the reliability of the software (more than 90%) |
| COST | Number of person hours. This indicates coding efficiency, number of features and difficulty. (305 person hours in total). |

* 1. Coding Standard[2]

## Notes

* + - 1. Class annotations, including Controller layer, Service layer, etc.
      2. Method annotations, all interfaces and public and private methods
      3. Try to have some comments in the code logic block

## Submission Specifications

* + - 1. Delete redundant classes, methods, and code blocks as much as possible
      2. Invalid import, do not use import xxx.xxx.\* for idea settings
      3. Submit notes to refer to the company's specification requirements

## Coding Style

* + - 1. Try to decouple the interfaces, do not share request and response classes, and each interface should have a request-response body
      2. Constants should be defined in the constant class as much as possible
      3. For the update of multiple tables, consider using transactions according to the situation, but do not abuse @Transactional transactions, as transactions will affect the QPS of the database
      4. Non-essential business logic is extracted separately, consider decorating with asynchronous annotation @Async
      5. Use the tool method in a unified manner for judging empty space;
      6. The same code is given priority to extract public methods;

## Naming Conventions

* + - 1. The method of obtaining a single object is prefixed with get, for example: "getXXX"
      2. The method of obtaining multiple objects is prefixed with list, for example: "getXXXList"
      3. The method of obtaining statistical values ​​is prefixed with count, for example: "getXXXCount"
      4. The insert method is prefixed with insert, for example: "insertXXX"
      5. The delete method is prefixed with delete, for example: "deleteXXX"
      6. The modification method is prefixed with update, for example: "updateXXX"
      7. Methods and classes will be named using Camel Case
      8. Constants will be named using All Caps

## Log Specifications

* + - 1. The entry and exit of some critical or frequently reported errors or complex logic methods, or some business code blocks can be properly added with some logs to avoid errors caused by adding logs.For example: entry and exit plus info level log(), debug level in the code block
      2. The exception log in the catch should be printed out, and the error level log should be added

## Code Review Process

## Pull Requests will be used for the code review

## Team members will be responsible for reviewing each other’s code

## There will be a review checklist to ensure consistency in the review process

## The reviewer should record their recommendations and comments on github

## Testing

## Testing tools/framework: The official test framework spring-boot-test-starter or spring-test-dbunit

## Unit Test: Testing object is a Program module or function module

## Unit testing is dominated by white box technology and supplemented by black box technology. It is done by the developers themselves and under the supervision of the QA leader

## Integration testing: Testing object is system, subsystem or modules assembled by software units

## The integration testing phase is dominated by black-box testing and supplemented by white-box testing. The primary integration tester is the QA leader and the integration testing is carried out under the supervision of the team.

## Defect Management

## Github issues: We will primarily use GitHub issues to track defects or enhancements identified during reviews or testing. GitHub issues are used to track various ideas, enhancements, tasks, bugs, etc. Github issues can also be implemented to collect user feedback

## The types of defects to look at[3]：

## A - Assignment: A small amount of code needs to be modified, such as initialization or control blocks. Defects such as declarations, duplicate names, scopes, and limitations.

## I - Interface: Defects that interact with other components, modules or device drivers, call parameters, control blocks or parameter lists.

## C - Checking: Prompted error messages, inappropriate data validation and other defects.

## B - Build/package/merge: Errors due to configuration repositories, change management, or version control.

## D - Documentation: Affects release and maintenance, including comments.

## G - Algorithm: Algorithm error.

## U - User Interface: Human-computer interaction characteristics: defects in screen format, confirmation of user input, functional validity, page layout, etc.

## P - Performance: It does not meet the measurable property values of the system, such as: execution time, transaction processing rate, etc.

## N - Norms: Does not meet the requirements of various standards, such as coding standards, design symbols, etc.

## The actions or personnel for defect management

* + - 1. Prevent the Defect
      2. Early Detection
      3. Minimize the impact
      4. Resolution of the Defect
      5. Process improvement

# References

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

1. Haddad, Rowan. “What Are the Best Git Branching Strategies.” Flagship.io, 13 July 2022, [www.flagship.io/git-branching-strategies](http://www.flagship.io/git-branching-strategies).
2. CSDN blogger "Huifeng and Changyou", "Summary of Java Coding Standards”

<https://blog.csdn.net/wennuan135/article/details/122032202>

1. Baidu Encyclopedia “Software Defects”

<https://baike.baidu.com/item/%E8%BD%AF%E4%BB%B6%E7%BC%BA%E9%99%B7/1940019>

# Glossary

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