**CS673 Software Engineering** 

**Team 1 - Cheffy**

**Project Proposal and Planning**

| Team Member | Role(s) | Signature | Date |
| --- | --- | --- | --- |
| Justin Fanning | Team Leader | *Justin Fanning* | 9/12/2021 |
| Kyle Mabry | Backup Team leader | *Kyle Mabry* | 9/12/2021 |
| Andrew Fish | Requirement leader | *Andrew Fish* | 9/12/2021 |
| Jean Shalenkova | Design and Implementation leader; Configuration leader | *Jean Shalenkova* | 9/12/2021 |
| Yanru Zhu | QA leader | *Yanru Zhu* | 9/12/2021 |
| Aidan Duffy | Security leader | *Aidan Duffy* | 9/12/2021 |

**Revision history**

| **Version** | **Author** | **Date** | **Change** |
| --- | --- | --- | --- |
| 1 | Team 1 | 9/12/2021 | Put draft version of all items |
| 2. | Justin Fanning, Jean Shalenkova, Yanru Zhu | 9/19/2021 | Updated all items based on feedback from iter 0; put in icon |
| 3 |  |  |  |
|  |  |  |  |

[Overview](#_87t9hln2vjz0)

[Related Work](#_mps353x5ezyl)

[Detailed Description](#_fg3z0hpd4q9v)

[Management Plan](#_ds8oyr75pnh1)

[Process Model](https://docs.google.com/document/d/107bVcXdAG-ogRr90PquFB8-aWGvTwSua8pu_O4Kmz6c/edit#heading=h.27177f40uci)

[Risk Management](https://docs.google.com/document/d/107bVcXdAG-ogRr90PquFB8-aWGvTwSua8pu_O4Kmz6c/edit#heading=h.a4oqwntk3mw)

[Monitoring and Controlling Mechanism](https://docs.google.com/document/d/107bVcXdAG-ogRr90PquFB8-aWGvTwSua8pu_O4Kmz6c/edit#heading=h.ywdoc2clc9yt)

[Schedule and deadline](#_tadq5mb0pici)

[Quality Assurance Plan](https://docs.google.com/document/d/107bVcXdAG-ogRr90PquFB8-aWGvTwSua8pu_O4Kmz6c/edit#heading=h.72e1f4uawy2r)

[Metrics](#_b2haznn3yyz2)

[Standard](https://docs.google.com/document/d/107bVcXdAG-ogRr90PquFB8-aWGvTwSua8pu_O4Kmz6c/edit#heading=h.vc72k6dweldv)

[Inspection/Review Process](#_f1c69ifi68h7)

[Testing](https://docs.google.com/document/d/107bVcXdAG-ogRr90PquFB8-aWGvTwSua8pu_O4Kmz6c/edit#heading=h.r5d5mhtlf0kq)

[Defect Management](https://docs.google.com/document/d/107bVcXdAG-ogRr90PquFB8-aWGvTwSua8pu_O4Kmz6c/edit#heading=h.54a4wuncjg1c)

[Process improvement process](#_jhct37ebxxpn)

[Configuration Management Plan](https://docs.google.com/document/d/107bVcXdAG-ogRr90PquFB8-aWGvTwSua8pu_O4Kmz6c/edit#heading=h.hw41vg4ykxen)

[Configuration items and tools](https://docs.google.com/document/d/107bVcXdAG-ogRr90PquFB8-aWGvTwSua8pu_O4Kmz6c/edit#heading=h.bwlb4d4vdox2)

[code commit guidelines](#_yyauft6zr9hw)

[References](https://docs.google.com/document/d/107bVcXdAG-ogRr90PquFB8-aWGvTwSua8pu_O4Kmz6c/edit#heading=h.8mva2050iy7t)

[Glossary](#_ty3i2nqffhtc)

# Overview

Cheffy is a recipe app that allows users to enter the ingredients they have on hand and to search for meals that can be made from them. The motivation behind this application is to help users discover new recipes, and to reduce food waste by finding uses for ingredients that might otherwise go to waste. Potential users for this app include anyone who cooks at home and wants to utilize the random assortment of ingredients they may find themselves with.

# Related Work

Yummly and Half Lemons are similar in nature to the application that we want to develop. Our app seeks to create a hybrid functionality of the two. In its most basic form our app will perform similarly to Half Lemons with the added functionality of allowing dietary restrictions/preferences for recipes that require additional ingredients. One of our reach goals includes functionality similar to Yummly in that our app would be able to generate a shopping list for recipes that include items in excess of what the user has on hand.

Links:

[Yummly](https://www.yummly.com/cook-smarter?utm_source=google&utm_medium=cpc&utm_campaign=WP%20-%20YummlySub%20-%20G%20-%20Search%20-%20Brand_06242021&utm_content=textad&utm_term=yummly&gclid=CjwKCAjwp_GJBhBmEiwALWBQk8sHm2ZDDo9c8rwkyZt4dYmASMsEvA9babA4u6jwm2UY0M4agU9KdxoCedcQAvD_BwE)

[Half Lemons](https://www.halflemons.com/)

# Proposed High level Requirements

* 1. Functional Requirements
     1. 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)

* Login: as a user, I want to create an account, so that I can store a list of ingredients I have, a list of ingredients to be avoided, and a list of favorite recipes. (10h)
* Logout: as a user, I want to be logged out of my account when I close the app/window, or by choice, so my recipe/pantry data is not accessible to others (1h)
* Password: As a user, I want to have my account password protected, so that others cannot log into my account. (3h)
* Ingredient Entry: as a user, I want to create/edit a list of ingredients, so that I can maintain an inventory of what I have in my kitchen. (10h)
* Allergies, Dietary Restrictions/Preferences: as a user, I want to create/edit a list of ingredients to be avoided, so that I can avoid recipes I dislike or would cause me harm. (5h)
* Search Recipes: as a user, I want to be able to search a database of recipes using lists of desired and undesired ingredients, so that I can find recipes that fit my criteria. (5h)
* Save Recipes: as a user, I want to be able to save/tag recipes, so that I can quickly find them in the future. (10h)
  + 1. Desirable Features (the nice features that you really want to have too):
       - Shopping List Generation: as a user, I want the ability to generate a shopping list, so that I can easily gather ingredients for recipes.
       - Recipe Rating: as a user, I want the ability to rate recipes, so that I can remember what I thought of them in the future.
    2. Optional Features (additional cool features that you want to have if there is time):
       - Suggest Recipes: as a user, I want to see recipes similar to ones i’ve liked, so that I can find new things to try
       - Suggest alternative ingredients: as a user, I want to see alternative ingredients in recipes that have items I do not like or cannot have, so that I can still make them.
       - Bonus: include a machine learning component to these features
  1. Nonfunctional Requirements
     1. Security requirements
     2. Allow sorting of ingredients and recipes (alphabetically, time to prepare…)
     3. Allow the user to store up to 100 ingredients in their pantry
     4. Allow the user to save up to 100 recipes
     5. Provide a means of feedback to the user about misspelled ingredients

# Management Plan

## Process Model

We are employing an Agile methodology. Each iteration begins actually at the end of the previous iteration, where we review the previous iteration and look forward to the next iteration. We review our goals for each week and mark priority in Pivotal Tracker, then during the iteration once developers have completed their code, testers will take over and test to make sure functionality is as necessary. At end-of-iteration review, we identify and confront any issues or risks we can foresee with previous features to see if it is necessary to update to a new version. These iteration meetings will be held on Zoom, while any day-to-day communication will be in our Slack channel.

## Objectives and Priorities

Priorities as of Iter 0 are to begin programming with a strong setup (to lay solid groundwork for which our app to be built on), to identify Pivotal stories, and to document a solid MVP. Distant goals are to reach basic MVP with time to spare.

Priorities as of Iter 1 are to complete the search by ingredient feature and begin thinking about creating user accounts and how to tie them into the app.

Priorities of Iter 2 are to further expand the capabilities of the app by adding the ability for a user to create a pantry of ingredients, the ability to save recipes, the ability to recommend recipes, and a more refined UI.

Priorities of Iter 3 are to revise and refine the code to ensure that the application performs as intended. Final UI revisions will also be targeted here.

## Risk Management (need to be updated constantly)

As of Iter 0, we have identified three main risks (as they pertain to our current level of project completion):

1. Project Completion: One of the biggest challenges while working on a project with such a short turnaround time like this is choosing a project that we can viably complete within time constraints without underestimating the time it takes to complete each feature, and without overestimating how much time each team member can dedicate to coding each week. It’s easy for us to estimate completion time without taking into account some particularly nasty conflicts that may arise in the codebase and the time it takes to resolve them. To combat this, we have chosen a project with a straightforward MVP that may seem simple at first glance. Due to a straightforward MVP that only focuses on essential features, we will be able to fully complete the basic functionality of the app and depending on the time left we can integrate some icebox ideas (we chose a few of small, medium, and large time commitments).
2. A majority of our members have voted 2-3 out of 5 on Flask familiarity, but we agree that it will be a useful and necessary tool to build our app. To resolve this, Jean has procured a plethora of Flask resources for the whole team to review and reference as we work on this app.
3. We have identified that a large source of chaos and grief for the duration of this project can be insufficient communication between team members: in order for us to succeed, we must all be on the same page, and for that to happen we need to communicate rigorously and in detail, and stay up-to-date on happenings within the project group even on days we can not dedicate to project work. To solve this, we have agreed to all download the Slack mobile app in order to always have team communication at hand.

[Risk Management Sheet Link](https://docs.google.com/spreadsheets/d/11XEUzvBX6gMj3LkrX_aLvpNc3mw5pBRJR6A1fKmtTmM/edit?usp=sharing)

## Monitoring and Controlling Tools and Mechanisms

We will use the following tools to facilitate group communication and monitor the project progress.

* + 1. Pivotaltracker Link: https://www.pivotaltracker.com/n/projects/2531653
    2. Slack Link: C02DK1V2L2E
    3. Github Link: https://github.com/BUMETCS673/BUMETCS673OLF21P1
    4. Zoom meeting Link: https://bostonu.zoom.us/j/6318252261
    5. Weekly meeting time: Saturday 12:30 EST
    6. Team file sharing: Google Drive

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

| Iteration | Functional Requirements(E/D/O) | Tasks | Estimated/real person hours | Presentation Recording Link (5-10 minutes) |
| --- | --- | --- | --- | --- |
| 1 | Search recipes | Need to fix endpoints so they properly search and display results | 4 |  |
| 2 | Personal Pantry(E), Recipe Display(E), User Accounts(E), Account Security(E), Save Recipes of Interest(E), Recipe Recommendations(O) | See Pivotal Tracker for more detailed tasks | 40 |  |
| 3 | Grocery List Generation(D) | Overall app Q/A, code refinement | 20 |  |

# Quality Assurance Plan

## Metrics

* + 1. Product Metrics: We will measure below metrics to determine the quality of the product
       - Test passing rate: This is the the tests that pass based on how many tests pass and the total test numbers using this formula: Tests passed/Total tests. This metric will be measured during each iteration and the types of tests include unit test and manual test.
       - User story counts: Use Pivot Tracker to create user stories and tackle them. The number of user stories completed will be tracked in each iteration.
       - Product complexity: This includes LOC (lines of codes), classes, methods, APIs used etc.
    2. Process Metrics: We will use below metrics to measure the effectiveness of the process.
       - Defect counts and defects fix rate: Defect counts is the total number of bugs (tracked in Pivot Tracker or Github with tag “bugs”) and the defects fix rate is defects fixed/defects found.
       - Project effort: Number of total hours spent on the project, which will be tracked in each iteration.
    3. Results (to be completed at the end of each iteration).
       - Test reports will be completed at the end of each iteration which include the metrics mentioned above.
  1. Standard
     1. Document standards
        + All documents finished consistently based on the templates provided by the professor.
        + Progress report: Filled out by teammates in each iteration to keep track of everyone's effort.
        + Meeting minutes: Teammates rotate to take. Action items will be used as guidance to finish work.
        + SPPP: This document. It’s an overview of the project and should be updated in each iteration.
        + Risk management: Bring up possible risks and potential plans to address them.
     2. Coding standards
        + Python: [PEP 8](https://www.python.org/dev/peps/pep-0008/), especially below the below standard.
          - Docstring: All public modules, functions, classes, and methods should have docstrings to describe the purpose of them. For functions, docstrings should also include the name, type, and description of the argument and result.
          - Naming conventions: Class names should use the CapWords convention; Function and variable names should use camelCase.

## Inspection/Review Process

* + 1. Document review:
       - All documents will have a draft version a few days before the deadline and every teammate can review other’s parts, leave comments in the docs if there needs any adjustments.
       - Team leader will make sure all documents are completed before submission.
    2. Code review
       - Developers who write the codes will review the code line by line to make sure the program runs well before creating a pull request.
       - Each feature added needs to be created through a pull request, and the repository will be gatekept by having at least 1 reviewer approval (could be any teammate except the developer) before the pull request can be merged into main.

## Testing:

* + 1. [Test report](https://docs.google.com/document/d/1dL6UG5lnAeUSCmmfh8YZ6Q2wlW_N-XHsz7rhVJ6nNws/edit) (TBD)
    2. These types of tests will be conducted:
       - Unit test: Developers working on the code will run unit tests of each core method before creating a pull request. Unit tests should be a part of the pull request.
       - End-to-end test: Use [Selenium](https://www.selenium.dev/) to create automated end-to-end tests on business-critical feature/user story.
       - Manual test: QA leader will run manual tests of the features after each iteration finishes and records test results in the test report.

## Defect Management

* + 1. Criteria used to describe defect: severity
       - Major: defects that impact the main feature or prevent the whole program from running, need to be fixed as soon as possible, preferably within one day.
       - Medium: defects that only happen with some edge cases, could be fixed before the end of each iteration.
       - Trivial: defects that are barely noticeable, could be fixed anytime, even after the iteration ends.
    2. Defects tracking
       - The QA leader will track the defects repaired, unrepaired, and the repair rate using a defect tracking spreadsheet (or Pivot Tracker) at the end of each iteration.
    3. Process of defects finding and fixing
       - Teammates who find the defects create a ticket in Pivot Tracker and assign it to the developer responsible for that functionality with below information:
         * Mark of the severity (major, medium, or trivial)
         * Description and screenshots of the defect
         * Expected result
         * Deadline of the fix based on severity
       - Developers fix the defect and create a pull request with a “bug” tag. Developers can include the teammate who finds the defect specifically to review this pull request and approve it. The pull request will be merged as a standard pull request after approval.

# Configuration Management Plan

(For more details, please refer to SCMP document for encounter example)

## Configuration items and tools

* + 1. IDE: VS Code
    2. Requirements tracker: Pivotal
    3. Version control: Github
    4. Teamwork: Slack
    5. Defect tracking: Github
    6. Environment management: Heroku

## Change management and branch management

* + 1. Main branch will be where our finalized code lives. Nobody should be working on any code in the main branch. This is the branch that will be user-facing and will be deployed to Heroku
    2. Feature branches - We will each create a development branch for each feature we work on. This way, any and all bugs will be isolated to the feature they reside in. Develop branch will be merged with the main branch when all bugs and conflicts are resolved.

## Code commit guidelines

* + 1. Each member will *git pull* before each time they begin coding in order to stay on the same page in regards to changes in the codebase. This will prevent unnecessary conflicts.
    2. Each commit message will briefly describe what has been completed in that commit.
    3. Each commit should contain no known bugs. If there is a bug in your code, you should instead *git stash* to save your progress.
    4. Commits should be reviewed by another team member before merge.
  1. Integration and deployment plan
     1. Python/Flask
     2. Jinja (built into Flask) for templating
     3. Spoonacular API as our source of recipes
     4. Deploy to Heroku

# References

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

# Glossary