**CS673 Software Engineering** 

**Team 1 - Cheffy**

**Software Design Document**

| Team Member | Role(s) | Signature | Date |
| --- | --- | --- | --- |
| Justin Fanning | Team Leader | *Justin Fanning* | 9/18/21 |
| Kyle Mabry | Backup Team Leader | *Kyle Mabry* | 9/18/21 |
| Andrew Fish | Requirement Leader | *Andrew Fish* | 9/18/21 |
| Jean Shalenkova | Design and Implementation Leader; Configuration Leader | *Jean Shalenkova* | 9/18/21 |
| Yanru Zhu | QA Leader | *Yanru Zhu* | 9/18/21 |
| Aidan Duffy | Security Leader | *Aidan Duffy* | 9/18/21 |

**Revision history**

| **Version** | **Author** | **Date** | **Change** |
| --- | --- | --- | --- |
| 1 | Team 1 | 9/27/2021 | First Draft |
| 2 | Team 1 | 10/12/2021 | Second Draft |
| 3 | Team 1 | 10/18/2021 | Final Draft |

[Introduction](#_heading=h.gjdgxs)

[Software Architecture](#_heading=h.17dp8vu)

[Design Patterns](#_heading=h.3znysh7)

[Key Algorithms](#_heading=h.tyjcwt)

[Classes and Methods](#_heading=h.3dy6vkm)

[References](#_heading=h.4d34og8)

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

# Introduction

*In this section, give an overview of this document, and also address the design goals of your software system.*

This document is used to describe the high-level design goals of the cheffy web application. The backend system of cheffy will use Python, Flask, Flask-related libraries and the frontend will use HTML5 and Jinja templates. SQLite will be used to build the database, which will be the storage of user data. The Spoonacular API will be used as our source of recipes.

Two of our main design goals are maintainability and reusability, which can be achieved by using the MVC architecture (discussed in the next item), because the front-end templates are separated from the backend business logic, keeping everything easier to be maintained, and the model is separated from views and controllers, making it easier to be used with other controllers if we want.

# Software Architecture

*In this section, you will describe the decomposition of your software system, which includes each component (which may be in terms of package or folder) and the relationship between components. You shall have a diagram to show the whole architecture, and class diagrams for each component. The interface of each component and dependency between components should also be described. If any framework is used, it shall be defined here too. Database design should also be described if used.*

Cheffy is using a MVC architecture that decomposes the system into three main logical components: the model, the view, and the controller.

The view contains an [index.html](https://github.com/BUMETCS673/BUMETCS673OLF21P1/blob/main/src/templates/index.html) page which serves as the homepage and the user will be directed to the [recipe.html](https://github.com/BUMETCS673/BUMETCS673OLF21P1/blob/main/src/templates/recipe.html) page once clicks on the “search” button. The [templates](https://github.com/BUMETCS673/BUMETCS673OLF21P1/tree/main/src/templates) folder contains all html pages that are serving as “View”.

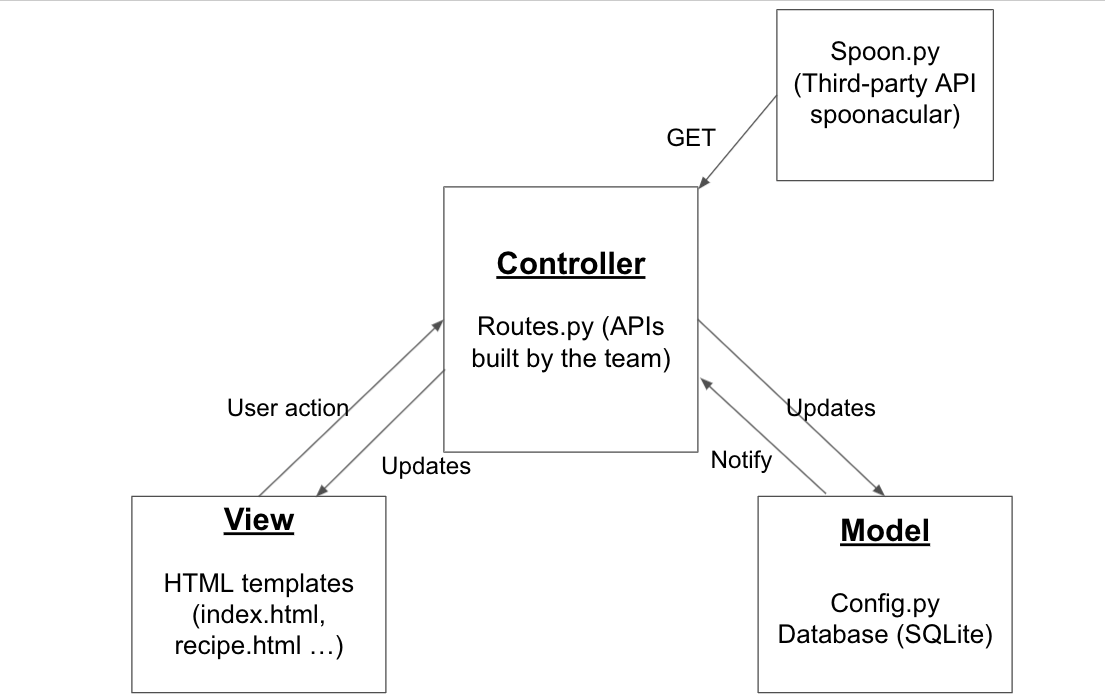
The [routes.py](https://github.com/BUMETCS673/BUMETCS673OLF21P1/blob/main/src/routes.py)is the controller that serves as a bridge between the view and model. The APIs built by the team for cheffy are defined in routes.py by using @app.route("/XYZ") serving as the API interface and def XYZ() being the implementation. Below is a detailed summary of the RESTful routes and their functions.

* @app.route("/"): Renders the homepage/[index.html](https://github.com/BUMETCS673/BUMETCS673OLF21P1/blob/main/src/templates/index.html) (if the user clicks on the logo/home button)
* @app.route("/about"): Renders the about page[/about.html](https://github.com/BUMETCS673/BUMETCS673OLF21P1/blob/main/src/templates/about.html) (if the user clicks on the button button)
* @app.route("/recipe"): Renders the list of recipe results/[recipe.html](https://github.com/BUMETCS673/BUMETCS673OLF21P1/blob/main/src/templates/recipe.html) (if the user enters ingredients and clicks on the “search” button)
* @app.route("/recipe/<recipe\_id>"): renders the recipe detail page/[recipe\_detail.html](https://github.com/BUMETCS673/BUMETCS673OLF21P1/blob/main/src/templates/recipe_detail.html) of a specific recipe (if the user clicks on a specific recipe image or the “see recipe…” button)
* @app.route("/profile"): renders the profile page/[profile.html](https://github.com/BUMETCS673/BUMETCS673OLF21P1/blob/main/src/templates/profile.html) which includes the pantry and favorite recipes (user needs to be logged in to see this)
* @app.route("/pantry/add", methods=['POST']) & @app.route("/pantry/del", methods=['POST']) & @app.route("/pantry/del\_all/", methods=['POST']): Stays in the profile page, but adds functions to add/remove/clear pantries.
* @app.route('/favoriteThisRecipe', methods = ['POST']) & @app.route('/remove\_recipe', methods = ['POST']) & @app.route('/profile\_favRemoveAll', methods = ['POST']): Stays in the profile page, but adds functions to add/remove/clear favorite recipes
* @app.route("/detailDelFav", methods = ['POST']): Stays in the recipe detail page, but adds a function to remove the recipe from favorites (if the user clicks on a filled heart button)
* @app.route('/login') & @app.route('/logout'): Used for user login and logout

The model is being built using SQLite which is located under [config.py](https://github.com/BUMETCS673/BUMETCS673OLF21P1/blob/main/src/config.py), some features are making use of the database for example the pantry feature, which is created to be [a Pantry class](https://github.com/BUMETCS673/BUMETCS673OLF21P1/blob/main/src/models.py) using the database.

Here is an example scenario of how these three components interact: The user clicks on the “save it” button in the recipe page (View) -> run.py gets that user action (Controller) -> run.py asks the Database to store that recipe into the user’s account (Model) -> Database (Model) notifies run.py (Controller) the recipe is saved successfully -> run.py (Controller) updates the recipe page (View) by notifying the user the success of saving the recipe.

The APIs defined in routes.py are mostly calling a third-party API called “[spoonacular](https://spoonacular.com/)” to get recipes and that is how the whole app is built upon. An API key is used to get different functions of the recipe work, for example to get recipe results with ingredients entered, or get similar recipes based on a recipe ID. The calling of this third-party API is created under [spoon.py](https://github.com/BUMETCS673/BUMETCS673OLF21P1/blob/main/src/spoon.py)**.**

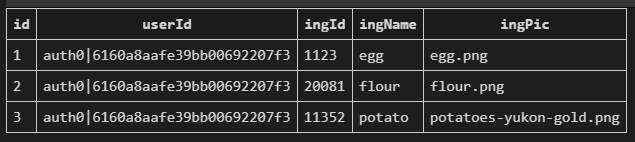
****

# Database Design (if applied)

*In this section, you shall describe any database if used in your software system.*

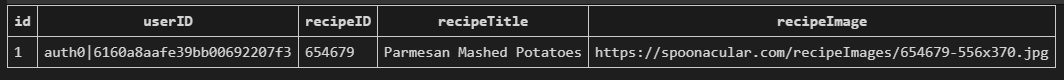
Cheffy will implement a SQLite database. A database has been created to track the contents of a user’s pantry. It connects a user ID with information about each item in the pantry. This information includes the ingredient name, the ingredient ID for use with our recipe API Spoontacular, and an image name again to be used with our recipe API. This database allows the user to keep a store of persistent ingredients, and use them as an input for a recipe search.

Database Sample Entry:



A similar database model is being used to track a user’s favorite recipes. The fields in this database are user name/id, recipe id, recipe title, and the address to an image for the recipe. This database allows the user to quickly refer to recipes they have liked in the past and pull up their information.

Sample Entry:



# Security Design

*In this section, you shall describe any security design in your software system.*

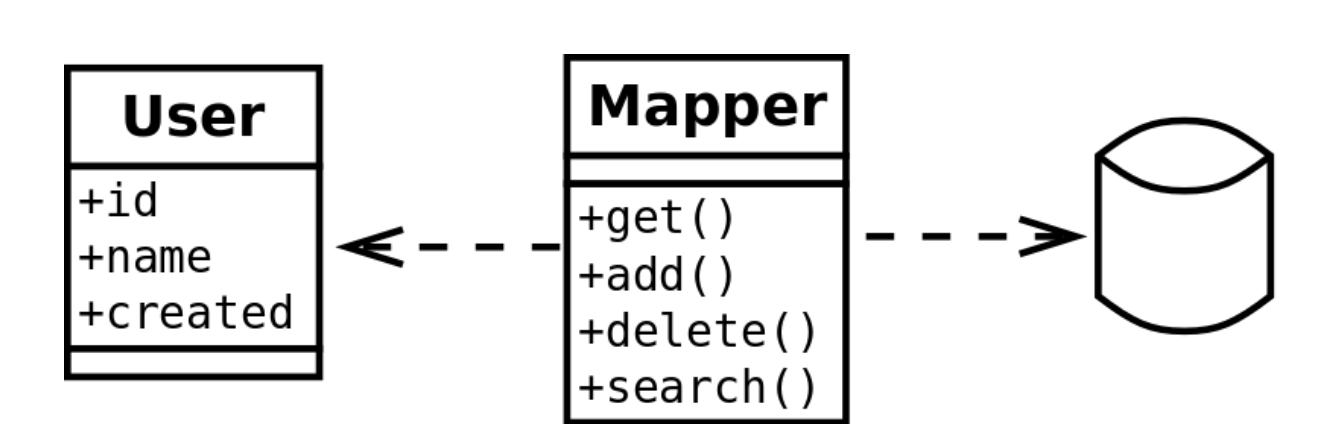
In terms of security, Cheffy will utilize the API Auth0 for page authentication, for account creation and verification, as well as user login data storage. This relieves much of the security weight off of our databases as user passwords will not be stored on our machines. Auth0 will help in generating access tokens and signatures. In terms of storing Cheffy-specific user information, such as a user’s pantry or recipes, we will tie a user’s data to their userID, supplied by Auth0. When stored on our databases, we will use PBKDF2 hashing to obfuscate a user’s authentication information. Finally, we need to integrate requiring authorization via tokens into our design for the purpose of verifying who can access specific pages (ex: a pantry), whether or not the session stores any data, etc. This information will also be done through the Auth0 API paired with the Flask-login package. Additionally, as all recipe queries will be done through Spoonacular, that API handles all input sanitization to avoid SQL injections.

# Design Patterns

*In this section, you shall describe any design patterns used in your software system.*

MVC: The Model View Controller that mentioned before is one of the design patterns we used for Cheffy. The usage of this pattern allows us to separate the database, the file that controls everything, and the front end files to different objects, giving the team much flexibility to work simultaneously on each object.

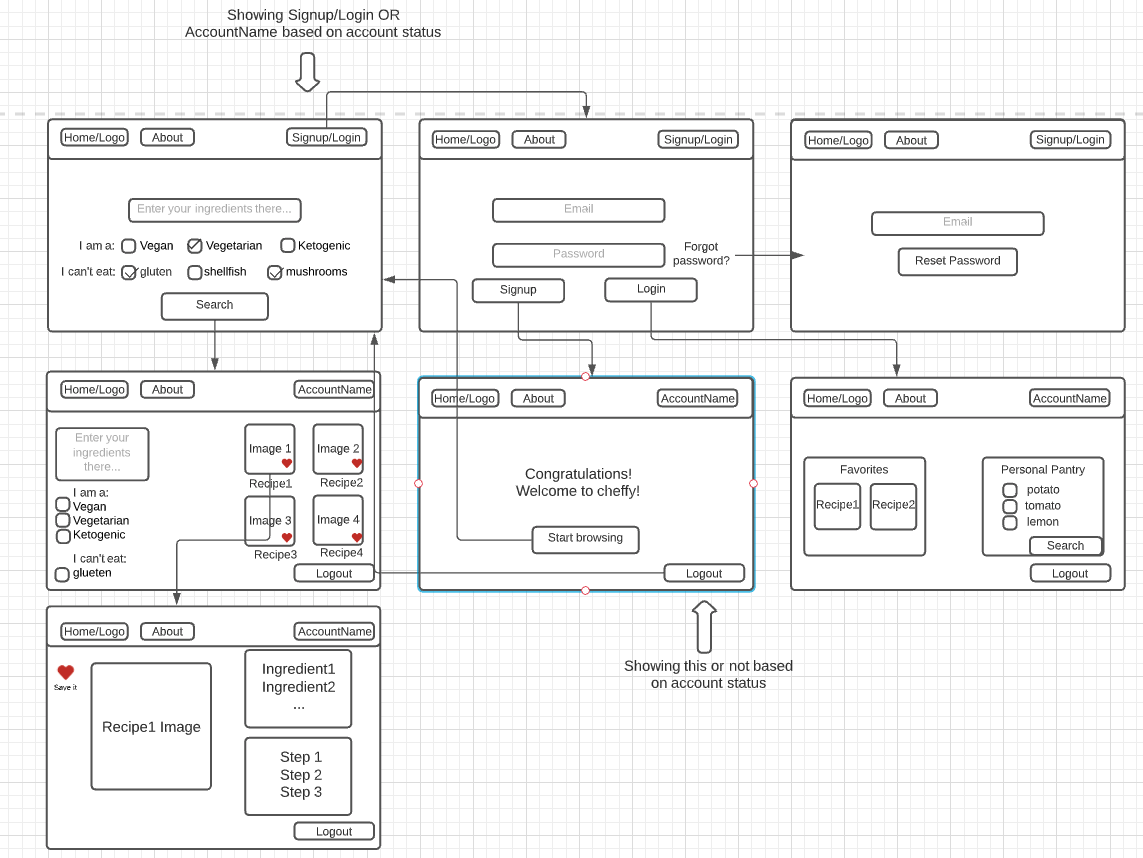
Data mapper design pattern: A specialist class, aka the mapper class, handles the interaction between a user and the relational database. See illustration below:



# UI Design

*In this section, you can describe your UI design.*

Below is the UI wireframe of cheffy. To take a look at it in a larger view please click [here](https://lucid.app/lucidchart/e43bb18e-998e-412d-8a71-ebcdbb20a884/edit?viewport_loc=-830%2C-463%2C3465%2C1799%2C0_0&invitationId=inv_3a0b20f5-932b-495a-878b-23f26f5b59da). Basically all of the pages will have a navigation bar that includes items home, about, login/register/accountname (if logged in), and logout. There will be an index/home page that has the search bar. Then the pages will be divided into two main sections based on if the user clicks on the login/register button or the search button: 1. Account-related pages: Account registration/login and account information, which will include the saved favorite recipes and personal pantry. 2. Recipe-related pages: The recipe list and a specific recipe page.



# Classes and Methods

*Please provide a link to your application API document which should be automatically generated.* **Click** [**here**](https://github.com/BUMETCS673/BUMETCS673OLF21P1/tree/page-permissions/doc/UML%20Diagrams) **for .uml files.**

# References

-https://spectralops.io/resources/how-to-choose-a-secret-scanning-solution-to-protect-credentials-in-your-code/

# Glossary