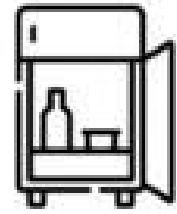


# Fridge Filler Software Final Report

4/28/21



## Team:

Shangyi Dai

Jonathan Derr

Travis Flake

Gage Gabaldon

Zhibang Qin

## Project Sponsors:

Dr. Richard Rushforth

Sean Ryan

## Mentor:

Sambashiva Kethireddy

## Overview

This document is the final document for the fridge filler project and goes over the entirety of our project and is a final send-off for the capstone project.

# Table of Contents

<b>Table of Contents</b>	<b>2</b>
<b>Introduction</b>	<b>4</b>
<b>Process Overview</b>	<b>5</b>
<b>Requirements</b>	<b>6</b>
Front End	6
Mobile App	6
Functional Requirements	6
Default App View (Pages)	6
Alternate App View	12
Non-Functional Requirements	12
Environmental Requirements	13
Admin Portal	13
Functional Requirements	13
Login Page	13
Food Box Management Page	14
Recipe Management Page	15
Default View	16
Non-Functional Requirements	16
Appearance	16
Authority	16
Environmental Requirements	16
Robustness	16
Back End	16
Database	17
Functional Requirement	17
Users	17
Pantry	17
Recipes	17
recipesSteps	18
Ingredients	18
Boxes	18
Server	18
Use Case: Administering the Database	18
Non-Functional Requirements	19
Environmental Requirements	19
Offline Ability	19

Analytics	19
Functional Requirements	19
Data Collection	19
Data Analysis	19
Viewing Data	19
Non-Functional Requirements	20
Performance	20
Response Time	20
Environmental Requirements	20
Data Usage	20
API (Application Programming Interface)	20
Functional Requirements	21
Methods	21
Server	21
Non-Functional Requirements	21
Simplicity	21
Intuitive Use	21
Environmental Requirements	21
Accessibility/Cost	21
<b>Architecture and Implementation</b>	<b>22</b>
Admin Portal	22
Web/Mobile Application	23
Recipe Browser	23
Saved Recipes	23
Digital Pantry	24
API	25
Authentication Server	26
Database	26
<b>Testing</b>	<b>29</b>
Web Application and Mobile Application Testing	29
Back End Testing	29
<b>Project Timeline</b>	<b>30</b>
<b>Future Work</b>	<b>31</b>
<b>Conclusion</b>	<b>31</b>
<b>Glossary</b>	<b>32</b>
Appendix A: Development Environment and Toolchain	32

# Introduction

Food banks have always had problems with getting food to families, and in recent times this demand is increasing even more. Even before the pandemic, Saint Mary's Food Bank and other food banks have had trouble making sure that food is not thrown away and wasted, which is caused in part by the uncertainty of what food will come through the food banks, meaning the distribution of food products is not uniform. For example, in the United States alone:

- Every year, 80 billion pounds of food are thrown away in the United States.
- Food banks across America receive different food products in varying quantities from donors throughout the country.
- Foodbank workers may be unfamiliar with products given to the food bank.

All of these factors create an inefficiency that is hard to solve with the changing supplies available to the food banks and with the lack of knowledge to best use the food.

This is where our sponsors, Richard Rushforth and Sean Ryan are looking to solve. Partnering with Saint Mary's Food Bank (SMFBA), They want to better feed the people the food bank is serving. The process of delivering and managing food for families all across the country is no easy task and one that is vital to continuing to help those in need.

The general workflow of the food bank is very reactionary. The food bank gets lots of different and unusual food from various food donors throughout the state. They then want to push the food as fast as they can to the food sites. Without this constant supply from the food bank, the food sites will undoubtedly run out of food. This is a good thing with the constant flow of food but comes with various issues for both the food bank and the user. The food donated is great but causes an issue that the food bank consumers don't know what to do with some of the more eccentric and unusual ingredients. Since the food bank gets what they get, whether it is catfish or red bean paste. This causes an issue where consumers either waste the food given or don't pick up the unusual food at the food banks. Saint Mary's Food Bank wants to help the consumers and the people working at the various distribution centers. The common problems at Saint Mary's Food Bank are:

- Food is not being used effectively at the food bank.
- Unusual food and unfamiliarity with ingredients.
- Online recipes are varied and require expensive ingredients along with too many complicated steps for those served by the food bank.
- Knowledge about what to do about the food is a mixed bag.
- Foodbank workers often can't help with this, as they often don't have enough time to plan out ways for people to use the food.

This is where our group, Fridge Filler, comes in. Fridge Filler aims to create a mobile application that can serve as a go-between for the food banks and the people and families that use the food banks. That way, families can figure out what food to cook with the ingredients they have through simple, easy-to-follow recipes. The app also can recommend places to pick up missing

ingredients and even a way to sync what you have at home with the app. The overall goal of the application is to help families effortlessly plan meals and to help ensure that the food at the distribution sites is used more effectively. The main features of the app are:

- A searchable library of recipes that will help consumers to use the food given to them.
- A Virtual Pantry to help keep track of what food the user has and helps recommend recipes.
- Navigation to both food bank sites and food retailers (preferably SNAP and EBT)
- Analytics about the way the users use the recipes and what foods are getting used where.
- Printable recipes for personal or food bank employee use.

This document will serve as the accumulation of the work we have done over the course of the school year and will help those to understand how we went into the project.

---

## Process Overview

As a team we meet weekly with our mentor and client, receiving feedback and ideas on how to improve our project. As well, each week we met as a team to discuss the completed tasks and tasks to come for each member. We began with all meetings through Zoom. We updated our repositories when we have something new. We started by getting the functionalities of the modules working as expected, considering user interface design later in the Spring 2021 semester. Once we had some progress on FridgeFiller and the admin portal, we distributed the application to our client to ensure we were developing everything as expected.

Our team distributed roles for front-end and back-end development.

The roles are:

- Jonathan Derr: Team lead and Backend developer
- Travis Flake: Frontend Sub-lead
- Shangyi Dai: Backend Sub-lead
- Gage Gabaldon: Release manager and Frontend developer
- Zhibang Qin: Frontend developer

The tools we used:

- Github
- Zoom
- Google Drive
- Google Firebase
- Monday.com
- Discord

- VSCode
  - Atom
  - Android Studio
  - JetBrains WebStorm
  - MySQL Workbench
- 

## Requirements

### Front End

The front end will serve as the basis of what the user will see when they open the app or view the website. This will include graphics, data analytics, and app design. This is the part of the app that users will interact with and is vitally important to make the design as simple and accessible as possible.

### Mobile App

The mobile app version of this project is the main front of the app. In the mobile app, all user functionality will be present, including Navigation to food distribution centers, a digital pantry system, a recipe browser, and a way to print recipes you have saved.

## Functional Requirements

### Default App View (Pages)

- **Login Page:** User Login/Registration Options.
  - From here users will be able to login to an existing account with the app or register for a new account, where they will need to provide basic information, specifically a username, email address, and password for the account. This will be authenticated and stored within our database, and the email will specifically be needed in the case of losing account access.
  - If a user does not wish to register for an account, they can proceed as a guest, and still have access to recipe browsing and navigational features. However, they will not be permitted access to saved recipes or user pantry, as specified in sponsor meetings.
- **Navigation Page:** A page that contains external resources pertaining to locating food distribution sites and SNAP/WIC retailers.

- **Digital Pantry**
  - Pantry Home: A home page for the pantry, where users can navigate to pages to add ingredients or manage their current pantry.
- **Recipe Printing**
  - Recipe Library: A page where users can browse their saved recipes, and view the full recipe or export them to a printer friendly PDF format if desired.
- **Recipe Browser**
  - Recipe Search Page: A page where users can view, search for, and rate recipes based on pantry content, popularity, rating, cuisine type, etc, as well as save recipes to the saved recipes page.

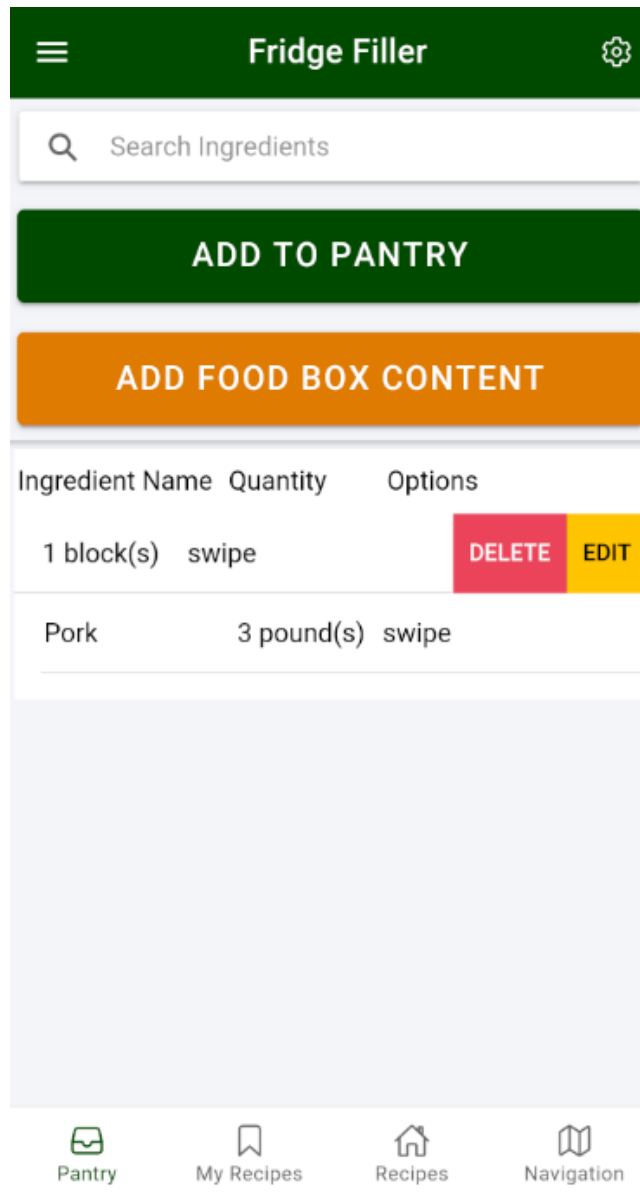


Figure X: Pantry Page on Mobile App

- The Pantry Home page will serve as more of a general navigation hub for pantry features. It will contain connections to the digital pantry, where you can browse your ingredients, and search for recipes that use them, as well as a connection to the Add to Pantry page, where users can add ingredients from their Emergency Food Box (EFB) or household to the digital pantry for recipe searching and keeping inventory of what food they have.
- Add to Pantry: A page to add food items, where users input item name, quantity, and unit of measure for an ingredient. Users can also add contents of an EFB to their pantry using the Add Food Box Content button, which opens a page with a list of food boxes, and inside each a list of common food box ingredients to add to the pantry.



The screenshot shows a mobile application interface for adding ingredients. The top section is an orange header with the text "Add Ingredients" and a close button (X). Below the header, there are three input fields: "Ingredient Name: Water", "Number of Ingredients: 1", and "The measurement" with a dropdown menu showing "gallon(s)". At the bottom, there are three green buttons: "ADD INGREDIENT", "CANCEL", and "SCAN ITEM(TENTATIVE)".

Figure X: Add to pantry page on mobile application

- **Note:** UPC scan recognition is listed as an optional feature, alongside item recognition. At a bare minimum, we expect the application to be capable of UPC input for food items.
- View Pantry: A page to view currently owned ingredients. From here users can select ingredients they own to search recipes that include them. They can also edit the quantity or delete food items from this page if they no longer have or wish to use them.

- When editing ingredients, users can modify the item name or the quantity of the item.
- Upon opting to remove an item from the digital pantry, a warning message will appear on the screen, to ensure that a user does not accidentally delete items that they intended to keep within their pantry.
- On each ingredient within the pantry, there will be a checkbox to select the ingredient. This feature can be used for either deletion of multiple items, or to select ingredients that you own to search the recipe database for. For instance, if a user selects chicken and rice, then press the Search Recipes button, the recipe database will look for recipes that contain both chicken and rice, and display them to you on the recipe browsing page.

### **Use Case: Adding Your EFB to the Pantry**

Let's say Joe has just gotten home from the food distribution center and has been given his Emergency food box. From here, Joe can then enter each food item based on UPC or item name. Joe can also input how much of each item they have received. From there, Joe can use these ingredients to search the recipe database, which is explored more below.

The digital pantry feature will be used primarily by the end-user, and not as much by SMFBA workers or administrators.

- **Recipe Management**

- Recipe Search: A page to search and view recipes by rating, ingredients, cuisine, difficulty to make, and time to make. From here you can view and "Favorite" recipes you like to print or use from the app.

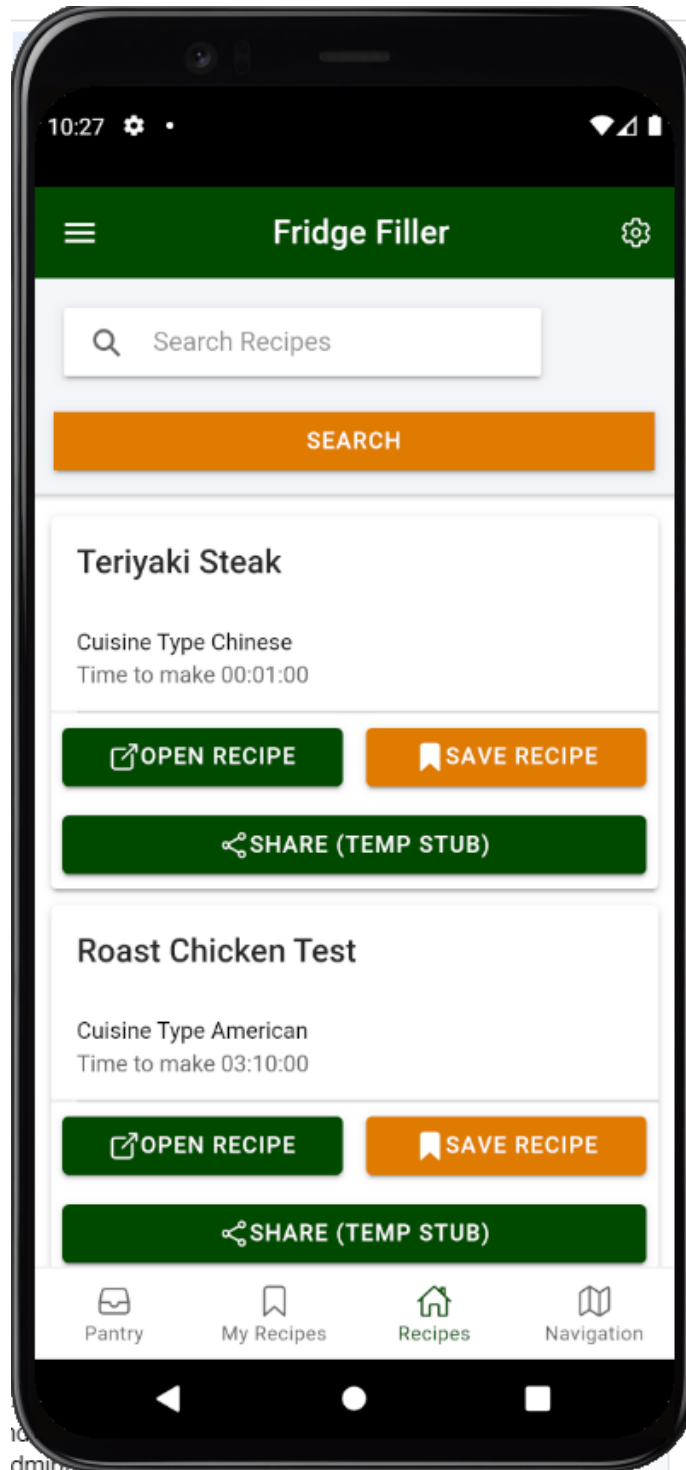


Figure x: recipe browser in the mobile application

- Recipes can be filtered by cuisine type, time to make, difficulty to make, and ingredients
- “Favorited” recipes will show as options on the recipe printing page.

- Recipes hosted within the app will be converted to plaintext, as images included within a recipe will make it consume more storage if it were to be stored locally, and will also cause it to take longer to load on slower or unstable internet connections. The sponsors wish for the app to be accessible to as many people as possible, and for that, we are trying to maximize the number of devices that can run it by keeping the required system strength and storage low.
- Recipe Print: A print recipe page, where you can queue up recipes in your bookmarked section to print.
  - The recipe print page serves to allow consumers to choose which recipes they would like to print, along with other general printing functionality (where the document will print, recipes per page). As specified in recipe search, the recipes are plaintext, without any images included, to ensure that users will be able to load them and store them on the device without needing a great internet connection or a large amount of storage available.

### **Use Case: Finding Recipes at Home**

Now that Joe has received his EFB and put the ingredients into the digital pantry, he is still not sure what he can cook using his foods. He can now use his ingredients list to search the recipe database for recipes that include ingredients that Joe has selected. Joe can now look at recipes, and search recipes by name. Recipes can be filtered by time to make, cuisine type, and rating as well to help Joe find a recipe that they will like. After Joe finds recipes that they want to try or like, they can print off the recipe, or save the PDF for offline use later.

### **Alternate Use Case: Finding Recipes as a Food Bank Worker**

Let's say our food bank volunteer, Linda needs to find recipes to provide EFB recipients. She can use the recipe library to find recipes that contain the ingredients that are in the food boxes that are being provided. She can then print the recipes off, and provide them to food bank recipients along with their food boxes.

## **Alternate app view based on the account type**

### **● Guest View**

- Those who do not sign up for an account in the app will be given a restricted version of the application. Guests will not be able to store ingredients in the digital pantry, as it will be stored on an individual, user by user basis.
- Other functionality of the app will still be fully available, but any features attached to the digital pantry will be unavailable in this view, namely the search of recipes based on the pantry and the complete use of the pantry.

## **Non-Functional Requirements**

### **Appearance**

- The User interface will be aesthetically similar to the Saint Mary's Food Bank website, utilizing a red and white color scheme, with simple shapes and sharp design.
  - This can be modified based on feedback from user testing, as it was also suggested that the orange and green appearance of the AZ Food Bank Network website or the Feeding America website could be referenced.

## **Accessibility**

- The application will use larger fonts, and provide step-by-step instructions to users to ensure that the visually impaired and tech-illiterate aren't struggling to use the app.
  - The user experience will have an intuitive, simple design such that no new users will feel intimidated by their lack of technological knowledge and experience. All search boxes will have placeholders that tell the user what is meant to be inputted, and any instruction prompts will have simple steps.

## **Environmental Requirements**

### **Cross-Platform Application**

- Our client requires an application that can work on as many systems as possible to facilitate users who only own certain devices. We must develop an Android/IOS/Web version of the application.
  - The framework we have chosen, Ionic, is based on HTML, CSS, and JS, which are the building blocks for most modern web applications. The Ionic framework will allow us to develop an application that functions on IOS, Android, and as a website with simple modifications.

---

## **Admin Portal**

The application needs an administrator portal to alter the information of food boxes and recipes provided to the food bank app users. The administrator portal will be where Food Bank officials can change the list of food boxes, number of ingredients and the content of food boxes. Administrators will also be able to add, edit or delete recipes that are shown within the app, potentially to add warnings for allergies, or delete recipes if the feedback is showing that the recipe is considered bad by the community.

For example, a manager needs to change the information of a food food box, he will open the login page, use google account to log in. Then he will open the food box management page and choose the edit button behind the site he needs to change and then edit the information.

## **Functional Requirements**

### **Login Page**

System Administrators will log in on this page to manage the database.

- Google login will be used for administrators, and admin access will be allocated to admins' accounts.
- The navigation page will not be able to be opened until the user has logged in to an admin account successfully.
- After logging in, admins will be able to enter the navigation page to manage the database.
- There is a sign-out bottom for admins to switch accounts or just log out.

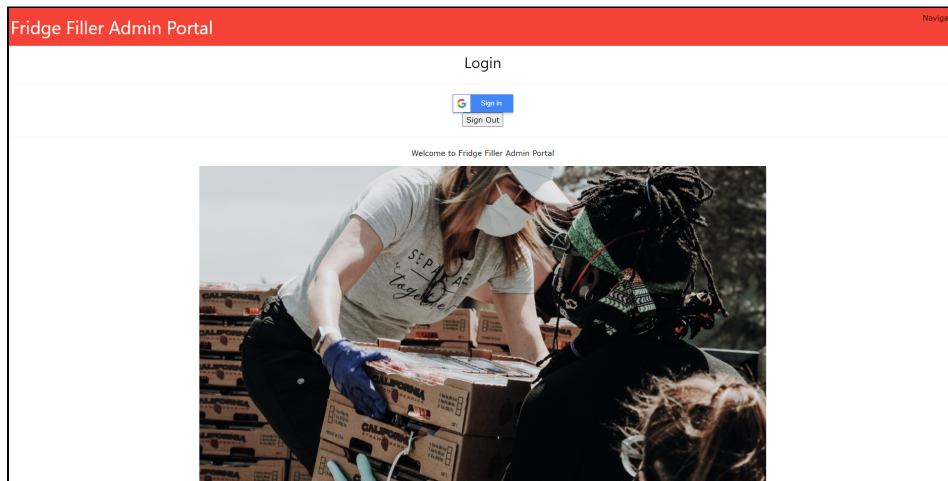


Figure: Login Page

## Food Box Management Page

This page is used for the management of the food box.

- On this page, box id, box name, number of ingredients will be displayed in a tabular form and there will be three option bottoms including show, edit and delete for the admin.
- There will be a search bar for the admin to find a certain food box.
- Admins can manage the name, number, information(such as opening time) of the food box through the edit bottom in the form, or add a new food box through the add bottom.

BoxName	NumIngredients	Option
ShowTest	3	show edit delete
puTNew1	3	show edit delete
1	0	show edit delete
122	1	show edit delete

Figure: Boxes Management Page

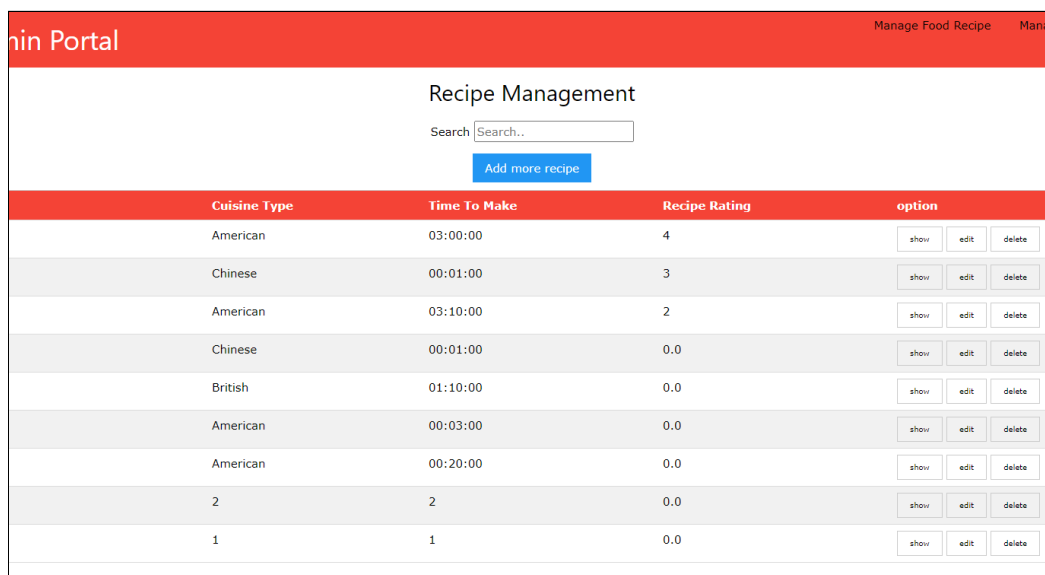
## Use Case: Managing Food Boxes

Our admin user, Mike, may eventually receive an updated list of food boxes, which is not the exact list of boxes already shown within the app. Mike can then input information about the new food boxes here, as well as remove food boxes that may not be used or have problems. If a food box needs to be changed, Mike can use the edit function of the food box to change its content.

## Recipe Management Page

This page is used for the management of the food recipe database.

- On this page, the information of the recipe will be shown like a card that contains the picture, name, description of the recipe and there will be an edit bottom for admins.
- There will be a search bar for the admin to find a certain recipe.
- Admins can view existing recipes, delete a recipe, or manage the detailed information of the recipe through the bottoms in the form or add new recipes through the added bottom.



Cuisine Type	Time To Make	Recipe Rating	option
American	03:00:00	4	show edit delete
Chinese	00:01:00	3	show edit delete
American	03:10:00	2	show edit delete
Chinese	00:01:00	0.0	show edit delete
British	01:10:00	0.0	show edit delete
American	00:03:00	0.0	show edit delete
American	00:20:00	0.0	show edit delete
2	2	0.0	show edit delete
1	1	0.0	show edit delete

Figure: Recipe Management Page

## Use Case: Changing The Recipe Database

The database for healthy recipes may need new recipes one day, and the administrator, Mike, will need to add these. Using the Add New Recipe button, Mike can input a recipe formatted to match the other recipe files, or manually input data that was submitted in an invalid format to create a new recipe listing that meets the standards of the recipes the app shows. If Mike needs to change a recipe, he could search the recipe on the recipe management page and edit the recipe data to include the updated recipe. If a recipe is deemed poor quality or too difficult to expect users to make, the recipe can also be deleted by Mike here, should SMFBA wish to.

## Default View

- Login Page: Admins need to enter a username and password to log in.
- Admin view: This is a page with a bottom related to the management page of food distribution sites or food recipes after logged in.
- Recipe Management: Admins can view the recipe list, add, view, edit, or delete recipes.
- Site management: Manage the name, address, information, and so on of the existing dist sites or add new dist sites.

## Non-Functional Requirements

### Appearance

- The admin portal will have a simple design in the login page and admin view page. On the admin site management page there will be a form to manage the detailed information. On the recipe management page, there will be cards to show the recipe. The color of the admin portal is similar to other parts which are mainly red and white color.

### Authority

- Usernames and passwords are needed on the login page.
- Except for the login page, all pages can only be opened after login.

## Environmental Requirements

### Robustness

- To help keep maintenance tasks to a minimum, we should try to make this admin interface as robust as possible to avoid having to add functionality later.

---

## Back End

The backend will encompass the parts of the app that the users will not see directly. This includes the more functional parts of the project, such as data storage, analytics gathering, and general logic. This end will be the heart of the app and serves as the backbone of all the frontend designs.



# Database

The database contains all the data we get or build. It is the base of our application and website and will store all of the information required for the service to work properly.

## Functional Requirement

Here we will show the main tables for our database.

### Users

- We need a user table to store all the information about users, we have User\_has\_Recipes and User\_rates\_Recipes tables to help it, and for administrators, we have the AdminUser table for them. The user table should be able to contain this information:
  - The ID of the user
  - The name of the user
  - The phone number of the user(Optional)
  - The email address of the user
  - The password of the user
  - The zip code of the user(Optional)

### Pantry

- We need a pantry table to store what food the users have, we have a Pantry\_has\_Ingredients table to help it. The pantry table should be able to store the following information:
  - The ID of the pantry
  - The corresponding user id for the pantry

### Recipes

- A Recipes table will be necessary to store all the recipes, we have recipeSteps and Recipe\_has\_Ingredients tables to help it. The Recipes table should be able to store the following information:
  - The ID of the recipe
  - The name of the recipe
  - The cuisine type of the recipe
  - The time of the recipe
  - The rating of the recipe
  - The number of ratings the recipe has received

## **recipeSteps**

- A recipeSteps table will be necessary to store all the recipe steps. The recipeSteps table should be able to store the following information:
  - The ID of the step
  - The corresponding recipe id for the step
  - The number of the step
  - The content of the step

## **Ingredients**

- An Ingredients table will be able to handle all the ingredients data that will be used for different tables. It should be able to handle this information:
  - The ID of the ingredient
  - The name for the ingredient
  - The number of the ingredient
  - The UPC code of the ingredient

## **Boxes**

- A Boxes table will be able to handle all the box information and we will use the Boxes\_has\_Ingredients table to help it. The Boxes table will have this information:
  - The ID of the box
  - The name of the box

## **Server**

The Database will need to be hosted on a server of some description. For initial testing, we use a Linux remote server. After that, we will move the database to a formal server.

## **Use Case: Administering the Database**

Mike needs to do some administrative work on the database, which includes manually updating some information, as well as removing some old data that isn't used anymore. The Admin portal doesn't support this functionality, so he will have to access the database directly. He can do this with the MySQL Shell, or MySQL Workbench. Using either one of these tools, he simply enters in his admin credentials, and queries for the data he's looking for, sending delete or overwrite commands as necessary. He can also manually add data here if the admin portal is down, or if he needs to edit the information that isn't shown in it.

## **Non-Functional Requirements**

### **Maintenance**

- This database should be easy to maintain, it is designed for long time use. The tables in the database can be modified and transformed quickly. The whole database system will be able to import and outport between different machines.

## **Environmental Requirements**

### **Offline Ability**

- Some of the data stored on the database will need to be stored on the device locally. The reason for this is because we need to allow our service to work during periods where there is no internet connection available, which we can achieve by storing some of the data locally.
- 

## **Analytics**

The Analytics portion of the project will collect data on how the app is used through Google Firebase Analytics.

## **Functional Requirements**

### **Data Collection**

- Data will be collected using the Google Firebase framework, which allows data points to be collected using “Event Listeners”, which “Listen” for a certain action to be performed (such as pressing a button or entering text), which triggers a response.

### **Data Analysis**

- Firebase will unpack and keep track of the various bits of information sent by the app through the event listeners, which can then be shown through a console.

### **Viewing Data**

- The collected data will be viewable in the Google Firebase Console through an administrator account, which will have to be created during the development phase.

### **Use Case: Viewing Data**

Mike has to prepare data on how the app is used for Saint Mary’s quarterly report. Fortunately, the Firebase console will give him access to everything he needs, including raw data, charts, and other ways of visualizing the data that he can use in his report. To access

this, he simply logs into the Firebase console with his admin credentials and navigates to the Analytics tab, where he will find everything he needs.

## **Non-Functional Requirements**

### **Performance**

- The logs sent out by the Event Listeners that we will be using should take no more than a few seconds to send.
- If they take longer than expected, the event will be cached on the device with a timestamp, and the contents of this cache will be sent out when network conditions allow the events to be sent out within the expected timeframe
- The event logs should not interfere with the normal functionality of the app

### **Response Time**

- The response time of the analytics system will be largely out of our hands once the data has been sent, but it is reasonable to assume that any data collected will be viewable within a few minutes.

## **Environmental Requirements**

### **Data Usage**

- In keeping with our goal to use as little internet-based data as possible, we will attempt to conserve data in both the strings used in the bundles sent with the Event Listeners as well as the number of elements contained within those bundles.
- We will also take extra care to ensure that these Event Listeners are called only when necessary; or in other words, avoiding unnecessary repeat transmissions

---

## **API (Application Programming Interface)**

The backend API will be the part of this project that does most of the heavy lifting. The API and the Server it is hosted on will handle all of the requests for information that are sent from the Mobile App, and will also allow for the information stored in the database to be modified through the Admin Interface. The API will need to be hosted on a web server, which will allow it to receive the requests sent by the app and send out Queries to the Database.

# Functional Requirements

## Methods

- The API will consist of methods that will send queries to the database depending on what the method does. For example, one method will have to search for and return a selection of recipes based on the ingredients specified by the function call, while another one will have to check the login credentials of the user.

## Server

- The API will need to be hosted on a web server, which will allow it to receive requests from the app. We host the API on Firebase.

# Non-Functional Requirements

## Simplicity

- For the sake of ease of maintenance, we will attempt to use as few functions as possible without making the API confusing or clunky to use.

## Intuitive Use

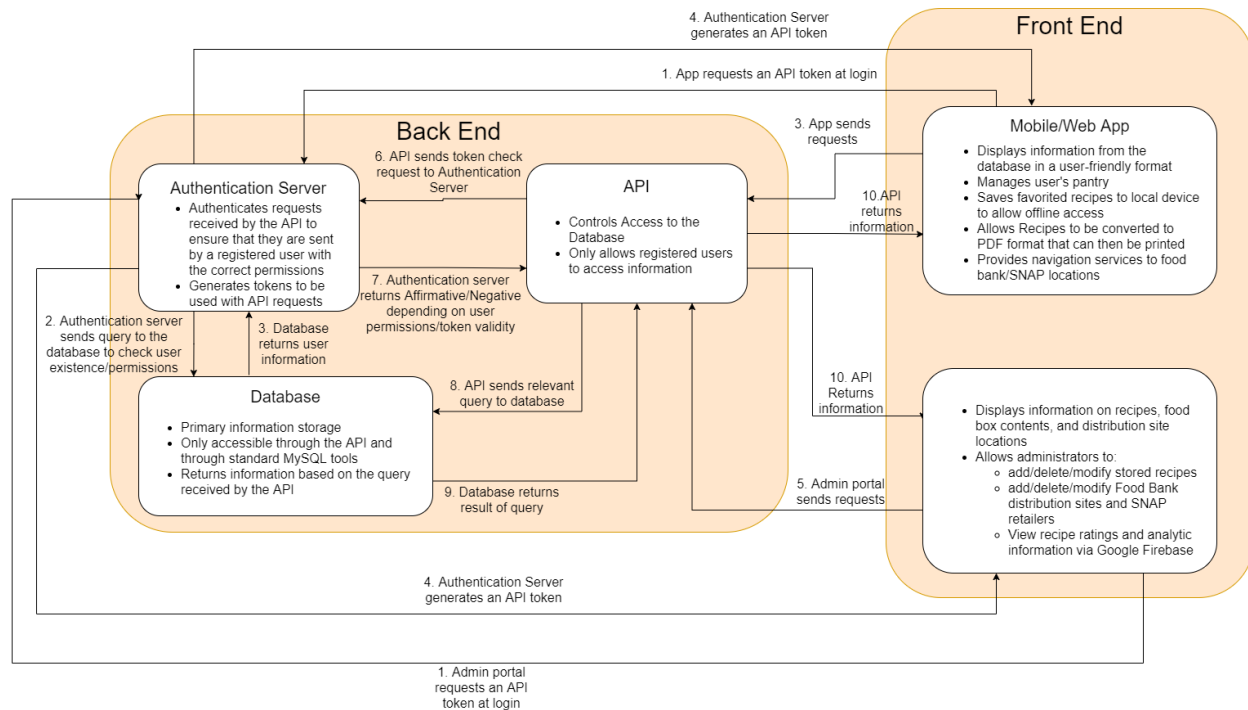
- To make the API easier to use for other programmers, we will make all of our function and field names intuitive; that is, the names will describe what the function does or what the variable is for.

# Environmental Requirements

## Accessibility/Cost

- Since this project is being developed for a Non-Profit organization, we need to keep costs down as much as possible to help make sure the organization (either Saint Mary's or Feeding America) can afford to run it for an extended period.

# Architecture and Implementation



## Admin Portal

The admin portal is the window that Saint Mary's Food Bank will use to manage recipes and food box contents that will show up within the web/mobile application of the project. The admin portal is authenticated through google authentication services, and only google accounts added by other admins to the portal are allowed to access this site. Admins can add new recipes, edit existing recipes, or delete existing recipes from the recipe database. Admins can add, edit the contents of, and delete food boxes, which are a list of ingredients made by Saint Mary's food bank that commonly appear in the emergency food boxes they provide.

### Use Case: Manipulating the Recipes

Imagine a hypothetical Saint Mary's Food Bank Admin, Martha, who wishes to add a new recipe to the recipe browser, and delete an old one that users were not reviewing very favorably. She can log into the admin portal using her google account, and then proceed to navigate to the recipes page from the navigation menu. From there, Martha can search for the recipe to remove by name, and click the delete button to remove it from the database. On the top of the recipe page, she can click the add recipe button to fill in a new recipe form, which includes recipe name, time to cook, cuisine type, number of ingredients, specific ingredients, number of steps, and specific steps. Once Martha is done, she can click confirm to post the new recipe to the database, or click cancel should Martha decide against adding the recipe. These recipe changes will be sent to our secure API, which, upon verifying the contents of the query, will change the database using the data sent with it.

### **Use Case: Adding New Food Boxes**

Martha has added new recipes to the database and now wants to add a food box with ingredients that are commonly given to food distribution sites. To do this, Martha will navigate to the Food Boxes tab of the navigation page, and select add food box, where you can add ingredients, each with a default amount and unit of measure. Once the food box is confirmed, it will be sent to the database through our secure API, much like the recipe data, and when it has been added to the database, it will be visible to end-users in the Pantry tab of the app, under "Add Food Box Content".

## Web/Mobile Application

The user-facing application is split into 3 separate modules, being the recipe browser, user pantry, and saved recipes pages. The Navigation page at its final state simply directs users to a food distribution site resource, and as such is being omitted from the list of components as it has little to no interaction with other sections of the application.

### Recipe Browser

The recipe browser will query the database through the API, and return a list of recipes to the app, which will then dynamically render the recipe listings on the page using the Angular typescript libraries "ngFor" functionality. Users can view recipes here by clicking the view recipe button, which will open up a more detailed view of each recipe. In the recipe view window, users can also rate recipes. Recipes can also be saved on this page, which will tie the recipe ID to the current user ID, effectively saving the recipe info to a user's saved recipes page. For the mobile version of the application, the saved recipes will be logged into the local version of the database to reduce data usage.

### **Use Case: Find a new Recipe**

Imagine a food box recipient, Bob, cannot find any recipes that use catfish, an ingredient that his emergency food box contained. Bob can open up the application and navigate to the recipe page, where he can search for recipes by name, cook time, cuisine type, rating, or popularity. Since Bob needs to use catfish, when he searches catfish it will show recipes that contain catfish in their ingredients list. From here, Bob can save the recipe for future use, or potentially printing.

### Saved Recipes

When a user saves a recipe from the recipe browser, it will show up on this page. Much like the recipe browser page, recipes can be viewed and rated here, and through the save API calls it will update ratings displayed on the next page load. This page also allows users to select and

print up to 3 different recipes at any given time and will produce a print-friendly PDF version of each recipe in a print efficient (low whitespace between recipes) version.

### **Use Case: Printing Recipes**

Bob will be cooking his catfish recipe tonight and wants to print it out to use in the kitchen so he does not get his phone screen messy with the ingredients he will be used to prepare it. To print the recipe, Bob opens my recipes page of the application and selects the recipe (or recipes, if he wishes to print more) to print, and then taps the print recipes button to create a pdf version of the recipes to print. On the web version of the app, it will store the recipe in the download folder of the machine used, and on mobile devices, it will notify the user where the file is to be stored. From there, Bob can print the recipe(s) using this PDF file.

## **Digital Pantry**

The user pantry is where users can keep a catalog of ingredients they have at home or have received from the food bank that they wish to use in order to make recipes. The pantry page will pull from the database on load for the web version or the local database on mobile devices. On this page, users can add, edit, or delete ingredients, the quantity of an ingredient, or unit of measure for an ingredient. Users can also view and add ingredients from food boxes defined by Saint Mary's Food Bank administration, which is a list of ingredients commonly found in specific food box types. The pantry page shares this list of ingredients with the database so that the recipe browser page can sort recipes based on the ingredients users already possess.

### **Use Case: Add ingredients to the pantry**

After starting up the app, Janet wants to add ingredients she has at home to her digital pantry, in order to keep track of how much she has without writing it down. By clicking/tapping the add ingredient button, Janet can add each ingredient from her home with quantity and unit of measure one by one, and then press submits to add them to the database. If Janet accidentally adds too much of an ingredient, perhaps too many loaves of bread, she can swipe to the left and tap edit recipe, where she can change the quantity or unit of measure for an ingredient.

### **Use Case: Add Ingredients from Food Box**

Steve has just received his emergency food box and was informed by Saint Mary's Food Bank that this app may help him find relevant recipes based on this food box content. After registering, Steve can navigate to the user pantry and tap the add food box content button, which will show a list of possible food boxes that Steve could have received. Upon selecting a food box, a list of possible ingredients will be displayed based on the food box selected, and Steve can choose from this list which ones he wishes to add to his digital pantry by tapping the add ingredient button. Upon closing the window, Steve will find all food box ingredients he chose to add are now in the digital pantry, and as such will work with the recipe browser to show recipes based on pantry content.



# API

The API we wrote for our project attempts to follow the REST guidelines. It is written in Javascript, runs using Node.js, and uses Express as a supporting framework for handling HTTP requests. It is, in effect, an HTTP server, that constructs queries to the database depending on the content of the requests sent to it. There are several defined paths that are publicly accessible which allows the requester to receive, add new, edit existing, or delete various types of information, including User information, Recipe information, Pantry contents, Food Box contents, Recipes that have been favorited by users, and more. It is a critical component of the project, as it handles all communication between the two front-end services and the database.

The API is capable of handling valid HTTP requests for specified request paths, which is how information from the database is manipulated and accessed. In lieu of going through use cases for every single logical endpoint for the API, we'll take a look at the different routes that are available and the different request types that are supported by each.

## Routes:

- /user: The user path handles user information. This includes things like the user's ID number, password, phone number, and other related information.
- /pantry: The pantry path handles the ingredients contained in each user's pantry.
- /recipes: The recipes path handles information about recipes, which is broken up into several parts for ease of storage, and needs to be assembled within the API itself.
- /box: The box path handles the different food boxes that are available for users to add ingredients from, as well as the contents of each of those boxes.
- /userfav: The userfav path takes care of keeping track of which recipes a user has favorited.
- /rating: The rating path keeps track of what recipes a user has rated, as well as the actual rating the user gave.

## Request types:

- GET: GET requests to retrieve information and send it back to the source of the request
- POST: POST requests add new information to the specified table/tables in the database
- DELETE: DELETE requests delete existing entries from the specified table/tables in the database
- PUT: PUT requests modify existing entries in the specified table/tables in the database

Each request path supports at least one of the supported request types, though in most cases, all of the list request types are supported. This allows for external manipulation of the database without allowing direct access to it, which is a crucial part of keeping the database, and the information contained within it, secure.

## Authentication Server

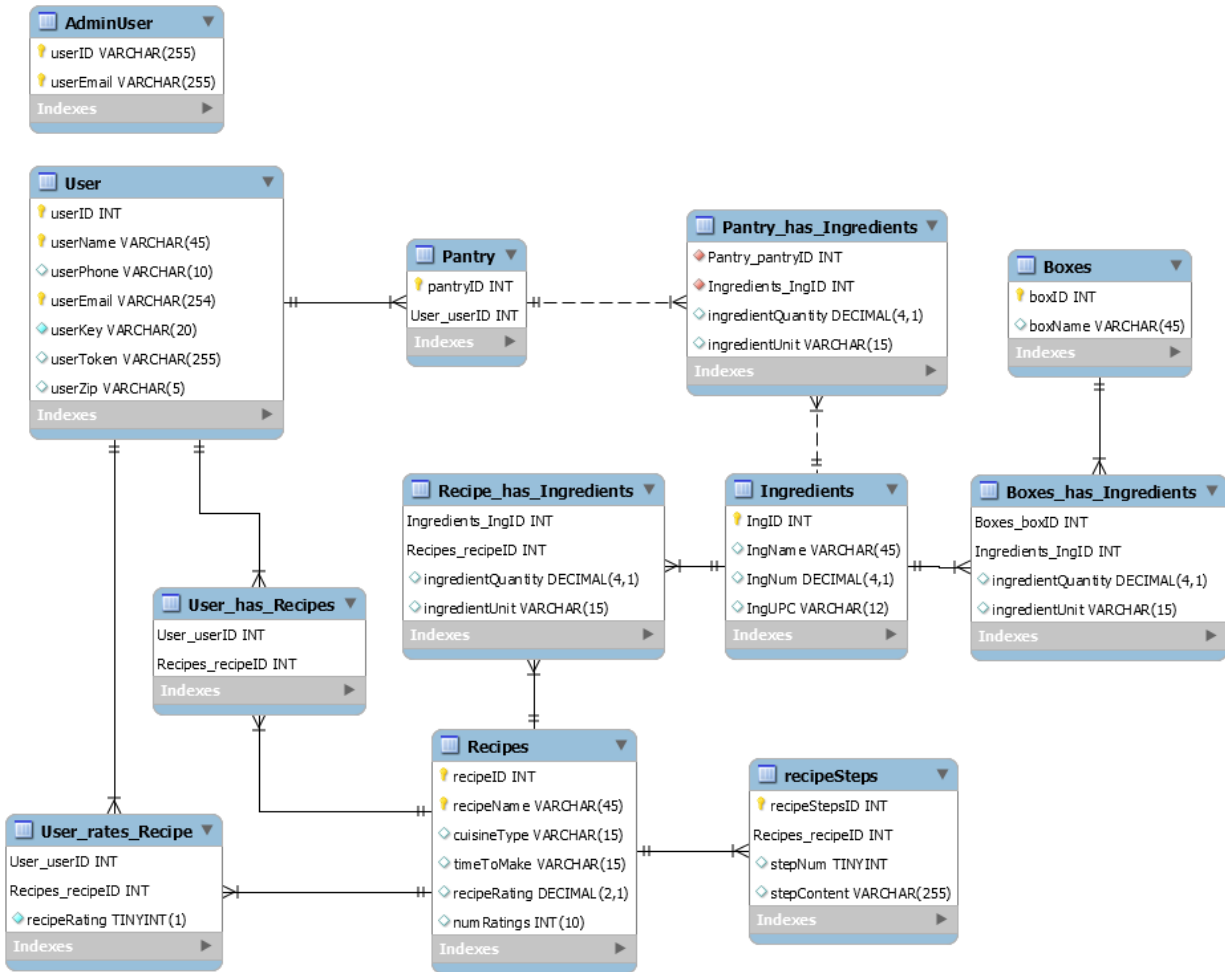
The Authentication server is written in Javascript, the same as the API it runs on Node.js and uses Express to handle different requests. All data input will be in a JSON file and posted to the authentication server. There are several defined paths for different requests, including user register, user gettoken, user authenticate, administrator register and administrator authenticate. And we have two DB models to help make connections with the database which are db.js and admindb.js.

There are several routes for the requests:

- /user/register this will register the user
- /user/gettoken this will get tokens for user
- /user/authenticate this will check the user
- /user/adminregister this will register administrators
- /user/adminauthenticate this will check the administrator

## Database

The EER Diagram for our database



## User Table

This table is a content table that contains all the information about users, it supports user signup and login features. It has a connection with the Pantry table and Recipes table.

## AdminUser Table

This table is a content table that contains all the information about administrators, it supports administrators signup and login features.

## User\_rates\_Recipe Table

This table is a relational table that is created by many to many identifying relationships between User Table and Recipes Table, it supports the recipe rating feature.

## User\_has\_Recipes Table

This table is a relational table that is created by many to many identifying relationships between User Table and Recipes Table, it supports the user's favorite recipe and recipe print feature.

The detailed information stored in this table is the connection between the User Table and Recipes table. For example, User A has three favorite recipes, and this table will store User A's id and the three recipes' ids.

Once a user adds one recipe to his/her favorite list, this table will add the corresponding recipeID.

## **Pantry Table**

This table is a content table that contains all the information about foods, it supports the UPC scanning feature. It has a connection with the Recipes table and Ingredients table.

## **Pantry\_has\_Ingredients Table**

This table is a relational table that is created by many to many identifying relationships between Pantry Table and Ingredients Table. It supports the pantry table.

The detailed information stored in this table is the connection between the pantry table and the ingredients table. For example, there are two elements in the pantry table, and this table will store the pantry id and the elements' corresponding ingredient ids.

## **Recipes Table**

This table is a content table that contains all the information about recipes. It supports recipe searching and printing features. It has a connection between the User table, recipeSteps table, and Ingredients table.

## **Recipes\_has\_Ingredients Table**

This table is a relational table that is created by many to many identifying relationships between the Recipes Table and Ingredients Table. It will support the recipe ingredients show-up feature.

## **recipeSteps Table**

This table is a content table that contains the steps of the corresponding recipe. It will show in recipe detail. It has a connection with the Recipes table. It supports the recipe detail and recipe print feature.

## **Ingredients Table**

This table is a content table that contains all the ingredients. It has a connection between the Pantry table, Boxes table, and Recipes table. It supports recipe detail and food detail features.

## **Boxes Table**

This table is a content table that represents the emergency food boxes handed out by Saint Mary's. It contains the ID for the box and the ingredient ID connected to it.

## Boxes\_has\_Ingredients Table

This table is a relational table that is created by many to many identifying relationships between the Boxes Table and Ingredients Table.

# Testing

## Web Application and Mobile Application Testing

For the web and mobile versions of the application, we used a Javascript testing suite, Jasmine. Jasmine is a testing suite that allows you to write unit tests for javascript applications, and then run them through a partner program, which for Angular applications is the Karma test runner. Due to the nature of how Ionic renders apps paired with Angular, many features are not optimally designed to be tested using Jasmine, but we still had tests to ensure correct rendering of elements on each page of the application, so no components are left out of the user experience. Some inputs were also unit test ready, but the results of these leaned more on the database side of the project than the front end, so unit testing them was somewhat less useful to ensure the correctness of the application.

Outside of unit testing, the team has tested each feature added to the project during and post-development to ensure that they are usable, intuitive, and functional in a less controlled environment. For each tab of the app, we tested how each component behaves when used alongside other features, but as each page is somewhat isolated from the others, the integration testing of this app to ensure that features like the user pantry, printing, and recipe browsing worked was rather simple, as each page only hosts a handful of features that are somewhat isolated from other pages. For usability testing, we showed our web application version hosted on Google Firebase to people of different age groups, to verify that functionality is actually simple enough for users to understand and use.

The results of testing varied. Unit testing was an ordeal to configure, and the results often were not as expected even though the product was, at least on the user end, fully functional in testing. Usability testing was productive, and we got useful UI and experience feedback from potential users that we integrated into our application, such as adding a popup warning to users to inform them where recipes are saved when you click print recipe.

## Back End Testing

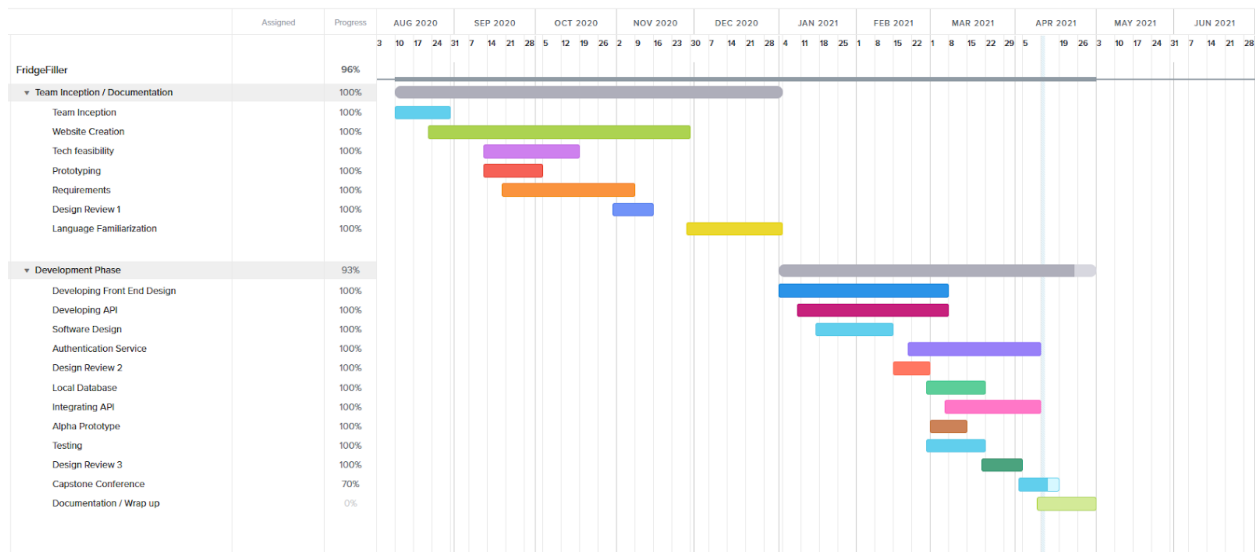
Testing the API will be a bit different due to the fact that it's a persistent application hosted on the web. Thankfully, Firebase has provided some infrastructure for running unit tests on hosted functions in their project framework, so used Mocha.js as our testing framework, and as stated previously all tests will be included with the function code that is deployed to Firebase, allowing the tests to be run externally.

And when doing the test, we found a quick way to test the functionality which is to use Postman to send test data directly to the API. By using Postman, we wrote the test data and compared the result with the data in the database to see if the API works correctly. According to the test result, we have found and repaired the broken parts like the recipe PUT in the API and unreasonable design like the need to set the foreign key to CASCADE in the database. The integration test between the frontend and backend works well. According to the test result, we found some unreasonable parts and broken points on both ends. In the backend, we found the user POST didn't do the proper work it was supposed to, the recipe order by rating and popular didn't list the highest one at the front. In the frontend, we found the user sign-up was using the old URL which will send requests to the old API.

## Project Timeline

Now going over the general timeline of the project. This has been an interesting project with two sides to the project. We have the documentation side and the development side. They have both helped with getting the project on track and both have certain semesters for them.

The first phase of the project last semester was going over the documentation and technical aspects of the project. That served as the backbone of fridge filler. This included the requirements documentation that has been the main way we have been developing the project.



This current semester was all about the development phase. As we have created and designed the software for the project. Among the many things we did this semester was creating all the

backend API. Such as the recipe API and the Authentication API. We have also created the front-end app portion as well. We then spent time Integrating them together so that the app is working for the user. The first time the app started coming together was for the alpha presentation. From alpha, we have added a few more much-needed features. Like getting the admin portal up to standards, fixing local database issues, and starting to refine all the features we have added so far. With the semester almost done the team is settling down and getting the last documentation in order. Along with any last-minute requirements done before we hand stuff over to saint mary's food bank.

## Future Work

As it stands now, the project isn't 100% ready for a public release, as there are still some features missing that would make it a more complete product. Luckily, our sponsors fully intend to develop the project further using their own development team to get it ready for release. This team will take what we were able to accomplish and refine it further, adding anything that we either missed or didn't have time to implement. Additionally, our sponsors were also thinking of sending it through the capstone cycle again, giving the project to a new team of students to add non-critical features that the full-time developer doesn't have the time to add. In any case, the project will definitely have a future once we're done with it, and we can't wait to see what happens with it.

## Conclusion

In conclusion, we believe that we've created a product that fills the client's immediate needs rather well. We were able to deliver a functional mobile and web-based application that uses external data sources, a database for storing information offsite, user authentication, a RESTful web API, and administrative tools to modify the data the app uses, all within the time constraints and fulfilling the requirements outlined by our clients. It isn't perfect, but it's a very good foundation for their developers to work off of, and our sponsors are very excited about the project's future.

# Glossary

## Appendix A: Development Environment and Toolchain

### Hardware:

We developed on Windows, IOS platforms with many different tech specs and nothing concrete for the hardware. The development cycle for the app isn't very hardware intensive just a computer that can run a decent environment and has a text editor will work for most of the development. With testing on android with android studio, you may need to have a machine that is capable of running an emulator.

### Toolchain:

We have a few tools that had made our life easier and/or were required for the project to work. Look below at the tools we used and how they might help for development in the future.

- Ionic: A framework to work on the app that hosts many different helpful commands and design elements. Our framework for the project and has a lot of useful documentation.
- Firebase: A way to help push to the firebase platform. It is useful for the deployment of our hosting solutions.
- Node: A package manager that helps install different packages and other useful components like npm. Useful to install and run packages.
- Github and Git: A way to manage version control of the code and manage the code repositories. We used it to keep our code in a cloud repository and to manage version control.
- Postman: API testing software that helps to test the API calls.
- Android Studio: An IDE that we use to run the android emulator and to get APK's and other android functionality.
- An IDE: Types we used were Atom, VScode, and JetBrains WebStorm. Needed to edit the code
- Express: Allows for easier routing and HTTP request handling. Simplified the coding process for the back end.
- My SQL Workbench: This is a way to easily look at a MySQL database graphically and to see what is in the tables along with anything else that has to do with a database. We used this to check if elements were

### Setup:

We split the setup into two sections based on what part of the app the user wants to work on. The front end will be the actual application the user will be interacting with and houses all the



design elements and other integrations that the user will use. The backend will be the place where all the recipes and other important information will be stored.

## Front End Setup:

This is the setup for the front-end environment of the application. It requires downloading Node, installing a few packages, and then getting the repository, and then from there, you're ready to start developing.

- Download the installer for Node LTS.
- Download the firebase tools with `npm install -g firebase-tools`
- Install the Ionic CLI globally: `npm install -g @ionic/cli`
- Clone this repository: `git clone https://github.com/fridge-filler/CSCapstone.git`
- Run `npm install` from the project root.
- Run `ionic serve` in a terminal from the project root.

## Back End Setup:

This is the setup for the back-end development.

When cloning the repo for the first time, follow these steps:

1. Clone the repo. There are two links for the repository: <https://github.com/fridge-filler/API.git> and <https://github.com/fridge-filler/Authentication.git>

The API code is located in `functions/index.js`. the public directory contains some generic html pages and isn't relevant to the API itself.

2. Run `npm install` in the functions directory to get all of the dependencies for the project. These aren't included in GitHub because GitHub doesn't like how big the dependencies folder is.
3. Return to the root directory, run `firebase login` and log into whichever account of yours is linked to the Firebase project.
4. IN THE ROOT DIRECTORY, run `firebase init hosting`. Use an existing project, and select `fridgefiller-296102`. Set `public` as the public directory, and say "No" when it asks if you want to configure the project as a single-page app. Don't set up Github, and don't overwrite any files.
5. IN THE ROOT DIRECTORY, run `firebase init functions`. Select Javascript as your language, install any dependencies you need to, *DO NOT* overwrite any files, and do not install ESLint.

## Production Cycle:

The production cycle of the application has been split up into two sections: the Front End and the Back End.

## Front End Cycle:

The front-end cycle of the application is somewhat simple.

Once you made your changes to the application you are going to want to rebuild the project. You can do this by doing an **ionic build --prod**.

Once this is done you are going to want to sync the changes you made to the android version of your application. By using the **ionic cap sync** command, once this is done you can test the web app by doing an ionic serve command in the command line to see if it works correctly.

For android, you want to run it on an emulated device or phone to test your changes.

The last thing you want to do is to deploy it to firebase and push it to GitHub. To deploy to firebase you need to do declare a target to deploy to. So **firebase target:apply hosting TARGET\_NAME RESOURCE\_IDENTIFIER**. Once you have a target and the necessary file is in the firebase.json you can then deploy using the following command: **firebase deploy --only hosting:<target\_name>**.

Lastly, You want to push all your changes to the Github repository. Most preferably need to push to a Dev branch before merging with the main repository.

## Back End Cycle:

Once the project has been initialized, you are free to make any changes necessary to add new features or to repair/maintain existing ones. And because of the languages and systems used in the Back End, no compilation is necessary once those changes have been made.

For the development of the API and Authentication Service, any changes made should be tested locally before they are deployed. This can be done using an emulator and a tool that is capable of sending HTTP requests. Firebase provides the ability to emulate Javascript code locally as if it were being hosted live, and Postman is capable of sending requests to that locally-hosted code as if it were being sent to the live version, so these tools should be used together to test any changes.

Once the changes have been tested and are ready to be deployed, you will need to run a command provided by the Firebase CLI to deploy it to the hosting environment:

```
firebase deploy --only hosting:<hosting target>,functions:<function name>
```

Where `<hosting target>` is the name of the hosting space you want to deploy to, and `<function name>` is the name of the function you want to create or overwrite. Once the command has been executed, the CLI takes care of everything else. Ensure that you don't overwrite any other functions when executing this command.

Modifying the Database is done through either MySQL Shell or MySQL Workbench. To do this, you'll need to log into the database instance with valid credentials. Once logged in, any changes made will be applied in real-time, so take care when editing or removing fields, as they may break certain functional elements and changes to the schema cannot be automatically rolled back. It is recommended that you create a backup of the database whenever changes are made so that rollbacks can be made if necessary.

**Any Other Questions Consult the User Manual.**