

## Description of the Overall Architecture (2 pages, 20%)

### What is Model-View-Controller (MVC) Architecture

Model View Controller (MVC) architecture is a three layered architecture that separates all functions and components into 3 different categories. The View (AKA presentation [1]) layer consists of all the UI components that the user can see and interact with [1,2,3]. The Model (AKA data [1]) layer contains all data managing and fetching functionality, this layer retrieves information that will be eventually passed to the View layer [1,2,3].

The Controller (AKA business [1]) layer is responsible for modifying and providing data to the user, acting as a link between the Model and View layers [1,2,3].

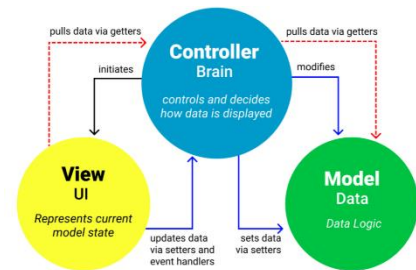


Figure 1 - MVC Diagram [3]

### Importance of Using an Architecture

Programming architecture patterns turn complex applications into manageable processes [4]. An architecture allows multiple developers to develop web and mobile based applications simultaneously by allowing each developer to work on different layers without affecting each others work [3]. Layered architectures provide more context into the processes behind each function, allowing for easier debugging, more readability and a better understanding into each layer's responsibilities [1,2,4]. By structuring functions into layers (In particular the model and view layers), components / functions can be re-used and dependencies between layers are often minimised [11].

In a layered architecture, each modular layer depends only on the layer below it [11]. Because of this, changes in one layer only tend to affect it's own layer and possibly one other layer [1, 11] (concept known as layers of isolation). This prevents errors occurring in more than one function, allowing for more readable errors and easier testing [1,2,3,11].

### How I incorporated the MVC Architecture into my Application

#### MVC in my Application In General

To incorporate the MVC architecture into my application I formatted each file into a set of folders, each folder responsible (apart from config) for a layer in the MVC architecture. The assets, components and part of the screens folder contains files related to the View layer, the persistence folder contains files related to the Model layer and the utility and screens folders contain functions related to the Controller layer. I factored out all the files into these folders to make the distinction between layers clear and allows me to import functions to files to ensure that there aren't any interdependencies between layers.

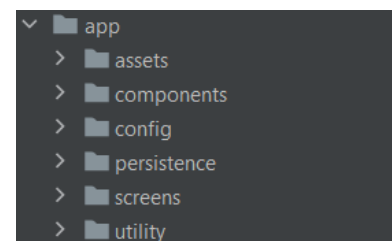


Figure 2 - File structure for my app

#### The Model Layer

In the model layer is where I contain all my functions that interact with either my static database (Firestore) or the recipes API I am using (Spoonacular). These functions either push data to the database or pull data from my database and recipes API. The model layer only

interacts with the controller layer, either through one of the screens requesting data, or by the data layer functions calling a utility function to format the fetched data appropriately.

```
export function FetchIngredients(callback){
  db.firestore().collection( collectionPath: "Ingredients").onSnapshot( onNext: (querySnapshot : QuerySnapshot<DocumentData> ) => {
    const items = [];
    querySnapshot.forEach( callback (doc : QueryDocumentSnapshot<DocumentData> ) => {
      items.push(doc.data());
    });
    callback(items);
  })
}
```

Figure 3 - Example of a function in my Model layer

### The Controller Layer

The Controller layer is where I contain the utility functions and screen functionality. The controller will fetch data from the persistence folder and parse it to the View layer to display. Whenever the user interacts with the View layer (i.e by pressing a button) a “function handler” is called and performs various tasks which either involve communicating to the Model layer to fetch / save data or to the View layer by changing whats on screen or navigating to a different page. I use these function handlers throughout as they are easy to manage and make the clear distinction that they are a part of the Controller layer.

```
const saveHandler = () => {
  SaveIngredients(user, props.login,
}

//Clears the user ingredients list
const clearHandler = () => {
  setUserIngredients( value: []);
  setLoading(!loading);
}

//User changes a filter category
const filterHandler = (filter) => {
  if(ingredients !== []){
    setFilterIngredients(QueryFilter
    if(search !== ""){ searchHandle
    else{ setLoading(!loading); }
  }
}
```

Figure 4 - Example of Function Handlers

### The View Layer

The View layer is where I contain all my UI components to display to the user. By factoring out all the components into a folder I am able to re-use frequently used components and also make the View layer easily identifiable. Assets are also easy to manage being in their own folder, only importing images to render when necessary. The View layer only interacts with the Controller layer by onClick or onPress events, these trigger handler functions that perform various tasks (i.e page navigation, querying searches, fetching data).

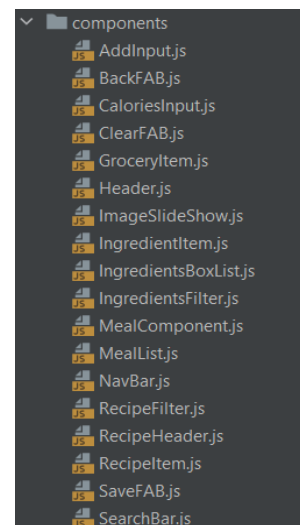


Figure 5 - View Components

## Advantages and Disadvantages of my Architecture Choice

### Advantages

MVC is the most common architecture compared to other architectures such as MVP or MVVM [2], because of this there is more resources and documentation supporting this architecture.

MVC architecture is built for front-end web and mobile applications in particular[2], making it easier to develop and maintain especially for someone like me with little experience with React.

### Disadvantages

Whilst, in general using an MVC architecture is the most ideal choice, by using this architecture there are also downsides. Stacking of layers increases file sizes and processing time [11], which in particular doesn't affect my app too heavily. Some function calls (i.e saving ingredients) can be made simpler by allowing the View layer to interact directly with the Model layer.

## Description of Utilising an External Component (1+ page, 5%)

### Description of my External Component

For my external component I chose to use a the recipe fetching API “Spoonacular” [5]. Spoonacular fetches recipes based on nutrition and calorie criteria, the API is also able to scrape the internet for links to webpages for the given recipe as well as scrape the internet for images that correlate to a given recipe [5]. In my app I have used all the features mentioned above, and used these to create my meal planner screen. Spoonacular is also able to retrieve nutritional info on ingredients, products and generate analytics for fast food chains (Although I did not use these features in my application) [5].



Figure 6 - Spoonacular Logo

### External Component Choice Justification

I was originally intending to create a slideshow component for each recipe and then populate the slideshow with images from an image scraper API. However, after consulting Jennifer, she didn't like the idea. I had a few ideas for external components which correlated with my app's purpose: An oven timer function which uses the phones system clock / external site, A meal suggestion feature which provides a recommendation based on the ingredients the user has the longest and their favoured recipes or a map component which showed nearby grocery stores for my grocery list screen.

In the end I decided to use Spoonacular's API which contains a daily meal plan generator, online recipe scrapper and an image scrapper for ingredients and recipes. I think this external component fits perfectly into my app's original concept. The app's main purpose is to provide the user with creative recipe alternatives for the ingredients they have, but this concept can suffer a problem where the user won't ever buy ingredients different from the normal. By including this external component, the user is now provided with cool recipes that they can now add ingredients for in their shopping list to go out and make later.

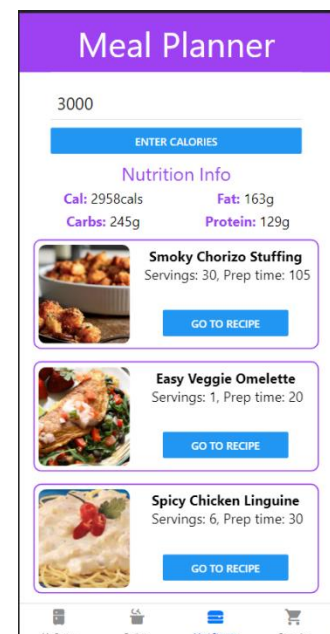


Figure 7 - Meal Planner Screen

### How I incorporated my External Component into my Code Structure

When integrating the Spoonacular API into my application, I followed the same filing structure I defined for my MVC architecture. This required me to extract UI components like the meal list and meal component (box that contains a meal and its basic info) and add them to the components folder. All API functions I seperated into its own file that is in the persistence folder (as fetching information for my external component would be classified as a model layer responsibility).

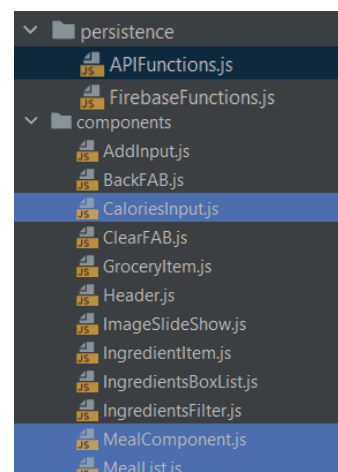


Figure 8- File Structure showing External Component files

In my meal planner screen, I contain a button and text area that the user can interact with to input calories. Once the user submits the button, I have a controller layer function that interacts with the model layer to fetch the nutrition information, 3 individual recipes, a link to a related recipe website and fetches images related to each recipe.

When adding the external component to my app there were no changes into how I structured my function calls or filing system. When inserting the external component into my app I had no issues as it was easily integrated into the architecture I had been using.

## Description of Utilising a Database (1+ Page, 10%)

### Firestore Description

Firestore is a Backend as a Service (BaaS) [6], that provides either a realtime database or static database dependent on what the developers needs are. Firestore is a flagship seervice that is developed by Google with the primary focus of providing easier functionality and deployment of web-based applications to the market [6]. Firestore utilises a cloud based framework that allows user scalability in app deployment with a pay as you go service (purchasing increased cloud storage, amount of read/write calls allocated, etc).

### Why I chose Firestore over other Alternatives

Compared to other alternatives such as AWS, Firestore is the easiest and most well known database hosting service for web applications. Due to the service being developed by Google, it is widely popular. Meaning that there are a lot of tutorials, documentation and resources online for learning and support.

Firestore allows for web-based hosting, allowing for my app to be deployed after development [6]. Meaning I can release my product easily and allow users to download my application quickly after development. Firestore allows for a variety of services such as the Firestore real time database, static database, cloud storage, authentication, machine learning, cloud messaging, remote config, real time analytics and hosting. Because of all these services, I am given the freedom to adapt and create new features for my app without having to worry about adding another dependency.

### How I used Firestore within my Application

#### My Firestore Structure

I chose to use the static database “Firestore” from the list of Firestore services as I did not require a real time database. Firestore allows me to structure data into collections, with each collection containing a

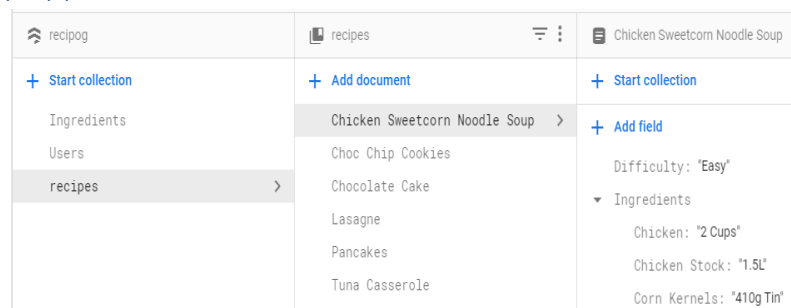


Figure 9 - Firestore structure

feature within my application (ingredients list, recipes list, ...). Each collection also contains a list of documents, this is where I defined each recipe and ingredient. Each document also contains a list of defined parameters that can be converted into a “json” like object to retrieve values from within my application.

## How Firebase is Integrated into my Architecture

When adding Firestore to my application I created a file called “FirestoreFunctions” in which all functions related to fetching data from firestore or pushing data to firestore reside. Adding these Firestore functions into my application fit perfectly into the MVC architecture I defined. As the data fetched from Firestore directly alters the View component, I define these functions as well

as the database itself the model for my application. Because this is defined as part of the model in my architecture, I added this file to correct “Persistency” folder in my file structure. The only way these Firestore functions are called is through the handler methods in the screens, which act as the Controller layer for my application, as well as the utility functions to manage and filter the data (also in the Controller layer).

```
export function SaveIngredients(user, username, userIngredients){
  db.firestore().collection( collectionPath: "Users").doc(username).set({
    username: user.username,
    ingredients: userIngredients,
    groceries: user.groceries,
  }).then(function () {
    console.log("Successfully saved ingredients!");
  }).catch(function (error) {
    console.error("Error saving ingredients: ", error)
  });
}
```

Figure 10 - Firebase function inside my Application

## Reflection on React Native as a Framework (2 Pages, 20%)

### What is React Native?

React Native is a famous Javascript-based mobile app development framework [7]. React Native is used by a lot of major companies such as Microsoft, Uber and Facebook which is used to develop their mobile and web applications [7]. The React Native framework allows the same applications to be made for different platforms (IOS, Android, Web, etc) whilst using the same code base [7].

### How React Native Helped / Hindered my Development

#### How React Helped me in Development

1. React Native allows for hot reloading and fast refreshing [12]. This is essentially allowing any changes made in development to appear in real time, avoiding the need to keep recompiling code. Allowing me to not waste time recompiling to find out that a piece of my code didn't work.
2. React Native also has a large user base, with a big community there is a lot of resources, as well as react native staying up to date with the latest technologies. With a large user base there is also many component libraries that have been released, allowing for quick and easy importing of good quality front end components [7,12]. I ended up using components from a public library “React Native Paper” which was easy to import and allowed me to spend more time developing backend functions than spend hours developing frontend components.
3. As React Native is a web-based application, it is faster to develop to multiple different platforms such as IOS or android. Therefore requiring less developers for a project and not as complicated skillsets from each developer to integrate the app into multiple frameworks [12]. This allowed me to import my app into my phone after completion but allowed me to test and develop completely in the web view.

4. React native is built on Javascript, which is easier to pick up than most other technologies like C, Java, Python, etc [7,12]. By starting this project with little to no experience in Javascript, I was able to pick up the basics relatively quickly. Certain programming features were simplified such as avoiding the type system of variables, so I didn't have to worry about type-specific errors.
5. React Native dependencies in a project are project specific. I started working on another React Native project for ENGR302, and because of this each project has their own dependencies, making it hard to know what to download and what versions to use. React Native template projects come with a package.json file which specifies the dependencies, allowing me to keep up to date by running "npm i" or "yarn install" without managing which dependencies I have installed.

#### How React Native hindered my Development Process

1. Although the Javascript language was relatively easy to learn, React Native has a steep learning curve when it comes to abstract navigation and function management concepts [12]. When I first started using React Native I spent many hours trying to get the navigation working, nesting stack navigators and combining different navigators. Concepts such as asynchronous functions, states, hooks and callback functions were all a vital part of my app, but was never taught to me properly throughout the course, where I had to learn what each concept was achieving and how to apply that to my app as I went.
2. There are some compatibility issues between functions / components used exclusively on web view that cannot be used on mobile. When developing a navigator for my app I first used a browser router, but after testing it on mobile the only navigator I managed to get working was a stack and tab navigator. Certain component libraries like Material.UI is not compatible with mobile either as well as CSS style sheets. These issues definitely made my development process a lot longer as I spent loads of time debugging and trying to find alternatives.
3. There were a few performance issues and bugs that occurred throughout the development of my app. I encountered problems where certain versions for dependencies in my package.json file did not work, or would randomly start working without any changes. There were also times where my app would start up instantly and then other times it would take 15 seconds. Although this didn't affect me too heavily, there are some projects where performance and development bugs would cause huge impacts on customers.

#### Suggestion for Improvement

If I were to suggest an improvement I would say to either optimise their framework or to create more standardised React Native library components that are compatible with mobile systems.

Optimisation is a big issue for large companies that have millions of users, use their applications and experience high traffic daily. By developing apps around low optimised frameworks, this would discourage users who experience lag, crashes or continual loading



from using that companies service. React Native experiences slight delays compared to apps created entirely natively [12], meaning users are less inclined to use React Native for apps that involve animations or extensive functions [7,12].

React Native would also improve by introducing more components to its front-end component library. Many of the components currently on the library are depreciated and no longer in service, meaning developers would have to spend more time creating their own components (or wouldn't be able to at all if they lacked the skill). If React Native were to also make the front-end components mobile compatible then I feel that the framework would definitely become more popular for mobile development.

## Description of Security Controls Used (1+ Page, 10%)

### Security Controls – Data Storage and Privacy

In my application I contain a file “EnvironmentVariables.js” for housing all sensitive information in my app such as database URL's or API keys. This correlates to both the security requirements 2.2 MSTG-STORAGE-2 “No sensitive data should be stored outside of the app container or system credential storage facilities” as well as 1.4 MSTG-ARCH-4 “Data considered sensitive in the context of the mobile app is clearly identified” [8]. By identifying that these pieces of information are sensitive, I can create threat models and plan security measures for handling this information [9]. As I separate this sensitive information into another file, I can gitignore it and therefore not push it publicly to my repository. By doing so, I reduce the risk of potential hackers getting and abusing this information. API keys in particular are important to keep private, as someone who has access to your API key can affect both the confidentiality of information stored on your databases as well as the integrity of the database [10]. Someone who gains access to the API key is given all CRUD permissions and can therefore clear the database or populate it with harmful data [10].

```
import Env from "./EnvironmentVariables";

const firebaseConfig = {
  apiKey: Env.apiKey,
  authDomain: Env.authDomain,
  databaseURL: Env.databaseURL,
  projectId: Env.projectId,
  storageBucket: Env.storageBucket,
  messagingSenderId: Env.messagingSenderId,
  appId: Env.appId,
  measurementId: Env.measurementId
};
```

Figure 11 - Showing Enviroment Variable Usage

### Security Controls – Authentication and Session Management

My application also features a login page, this allows the user to gain access to all of the apps features whilst saving data such as their grocery list to their account. This correlates to the security controls 4.1 MSTG-AUTH-1 “User / pass authentication is peformed to give user access to a remote service” and 2.11 MSTG-STORAGE-11 “The app enforces a minimum device-access-security-policy” [8]. By using an authentication feature in my app I only allow users basic permissions to my application, such as reading recipe/ingredient information and writing grocery lists / ingredient selection lists, without giving permissions to delete data [9]. Information is stored under the

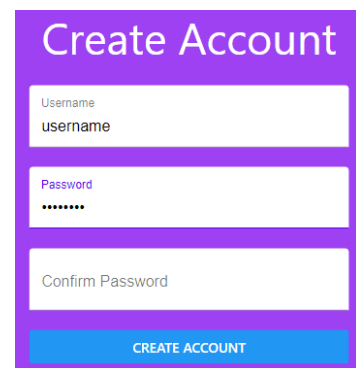


Figure 12- Showing Password Inputs

users details in Firestore, so no sensitive information would remain on the application after the user has logged out.

I have also included hidden text fields to the password areas for the login and sign up pages, as plaintext input areas would be susceptible to password spying attacks [9, 10].

### Security Controls – Code Quality and Build Setting Requirements

All third party components are from reputable sources as I only use code from Firebase, React Native documentation and the React Native Paper library. This correlates to the security requirement 7.5 MSTG-CODE-5 “All third party components used by the mobile app are identified, and checked for known vulnerabilities” [8]. If I were to use code from a non-reputable source, I would face the risk of installing harmful software such as trojans (listed on the OWASP top 10 mobile attacks [10]), which could steal information on my computer or run a denial of service attack.

I have removed all debugging code and added catches for possible exceptions in my code. These correlate to the security controls: 7.4 MSTG-CODE-4 “Debugging code and developer assistance code have been removed. The app does not log verbose errors” and 7.6 MSTG-CODE-6 “The app catches and handles possible exceptions” [8]. These security requirements are the most important as injection attacks are the most common attack for mobile applications [10]. If an attacker intentionally causes some sort of error or injects code, we must catch it to avoid giving the attacker info or potentially altering functions and data sent to the user. Sending specific errors (i.e Function “X” threw error on line 34) will also give the attacker a better knowledge of your system and therefore can plan attacks at vulnerable spots in your code.

### References

1. J.D. Meier, Alex Homer, David Hill, Jason Taylor, Prashant Bansode, Lonnie Wall, Rob Boucher Jr, Akshay Bogawat, Microsoft (2008) | Mobile Application Architecture Guide | [https://robtiffany.com/wp-content/uploads/2012/08/Mobile\\_Architecture\\_Guide\\_v1.1.pdf](https://robtiffany.com/wp-content/uploads/2012/08/Mobile_Architecture_Guide_v1.1.pdf)
2. Anh Dang (2020) | MVC vs MVP vs MVVM | <https://levelup.gitconnected.com/mvc-vs-mvp-vs-mvvm-35e0d4b933b4>
3. Martin Fowler (2006) | The evolution of MVC and other UI architectures | <https://martinfowler.com/eaaDev/uiArchs.html>
4. Rafael D. Hernandez (2021) | The Model View Controller Pattern – MVC Architecture and Frameworks Explained | <https://www.freecodecamp.org/news/the-model-view-controller-pattern-mvc-architecture-and-frameworks-explained/>
5. Crystal “coffeebean”, David Sky, Philip “qqilihq”, Allie Hill (2021) | Spoonacular Mission Statement and About me | <https://spoonacular.com/about>
6. Aman Mittal (2021) | Integrating Firebase with React Native | <https://blog.jscrambler.com/integrating-firebase-with-react-native/>
7. Maciej Budziński (2021) | What is React Native? Complex Guide for 2021 | <https://www.netguru.com/glossary/react-native>
8. Jeroen Willemsen, Sven Schleider, Carlos Holguera (2017) | Mobile Application Security Verification Standard | <https://mobile-security.gitbook.io/masvs/>



9. Jeroen Willemsen, Sven Schleider, Carlos Holguera (2017) | OWASP Mobile Security Testing Guide | <https://github.com/OWASP/owasp-mstg>
10. Jason Haddix, Daniel Miessler, Jonathan Carter, Milan Singh Thakur (2016) | OWASP Mobile Top 10 Risks | <https://owasp.org/www-project-mobile-top-10/>
11. Syed Hasan (2019) | Layered Architecture | <https://syedhasan010.medium.com/layered-architecture-5d6e500da9ec>
12. Marlena Walburg (2021) | Is React Native good? Advantages and disadvantages | <https://binarapps.com/is-react-native-good-advantages-and-disadvantages/>

## Appendix – Application Screens

Fig 1. Login Screen Images

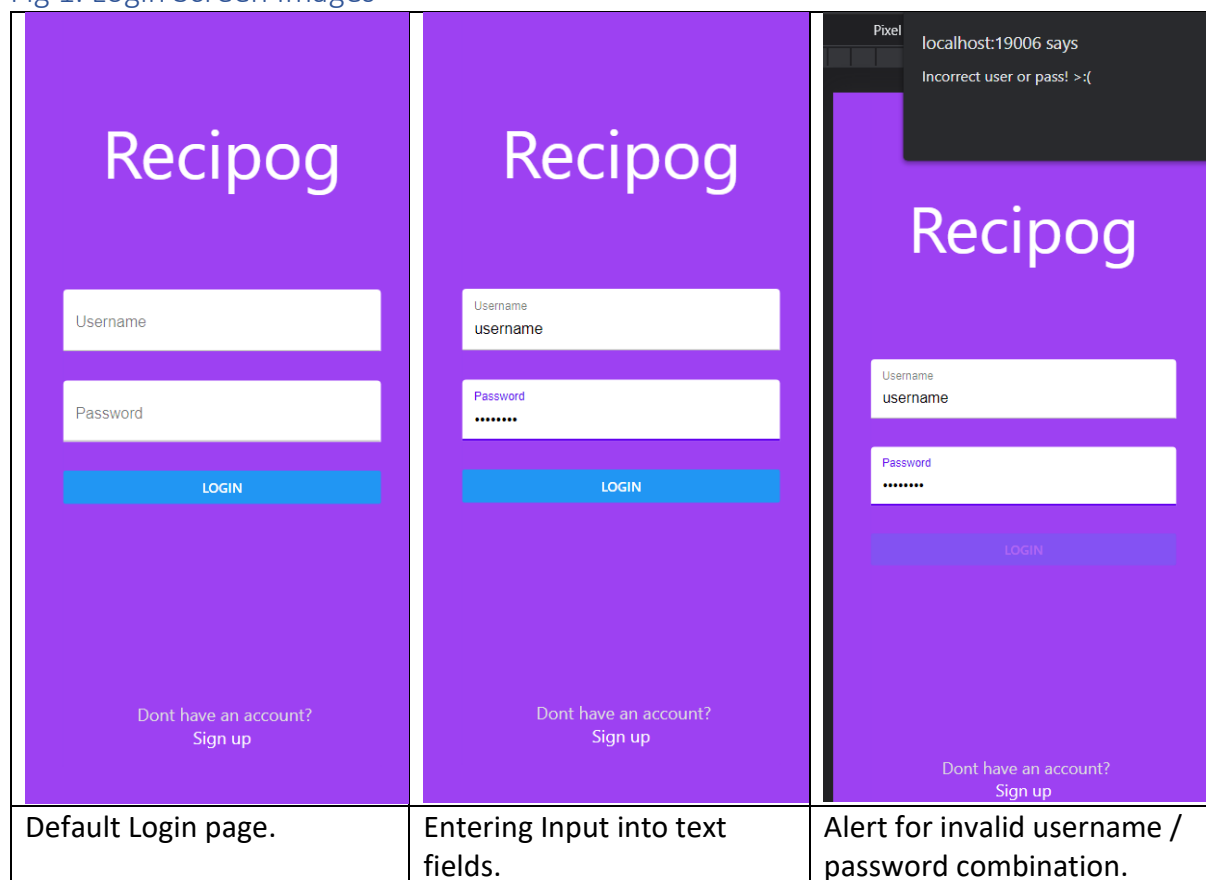


Fig 1. Description

Login page for the app. The user can enter their username and password to gain access to the app. If the user enters an incorrect username and password they are alerted that they have entered an incorrect username & password combination. The user can navigate to the signup page by pressing the “Sign up” button at the bottom of the screen.

The design features a single column, clean aesthetic using the primary blue and purple colours that features throughout the app. All elements are spaced fairly apart and padded off the edges of the screen to make these elements easier to see and interact with. All related elements are grouped together (I.e the inputs and button or the text and signup button) to show element correlation.

Fig 2. Sign up Page Images

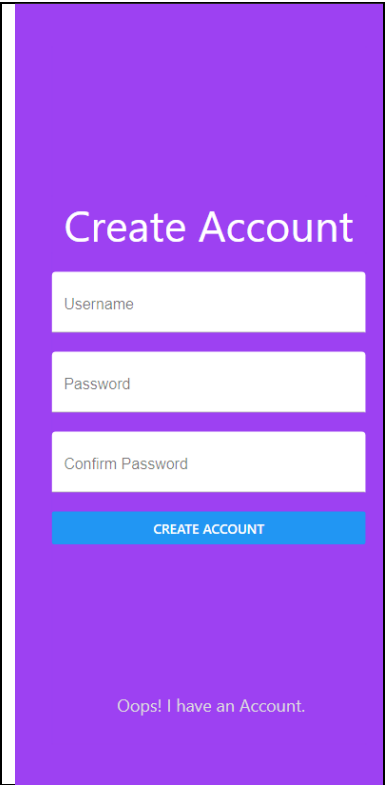
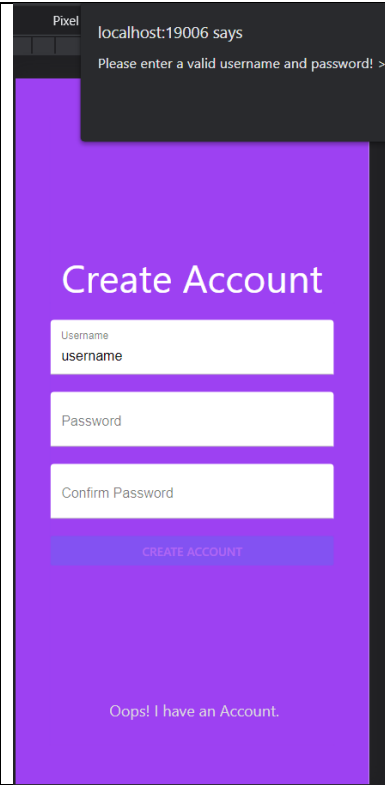
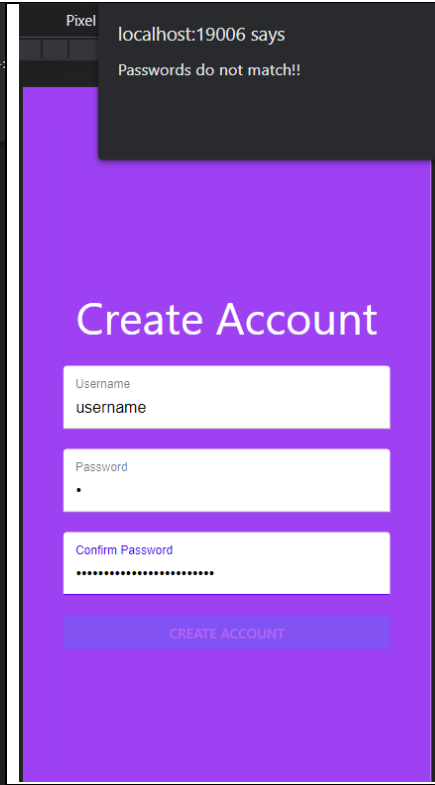
		
Default Sign up page.	Alert for having an invalid username / password.	Alert for having 2 passwords that don't match.

Fig 2. Description

Sign up page for the app. Allows the user to input a new username and password to be added to my firebase authentication feature. If the user inputs an invalid user or password (i.e “ ”) then they will get alerted to enter a valid combination. Additionally if the user inputs 2 different passwords for their password section they will be alerted that the passwords do not match and are prompted to try again. The “Oops! I have an account” button navigates the user back to the login page.

The design for the sign up page is similar to the login page, where the correlated elements are all bunched together to show relatedness. Contrasting colours are applied to direct the users attention to the elements on the screen. I decided to make the “Oops I have an account” text a duller colour to not distract the user from the primary focus of the page. However, if they do have an account they will search for the element to navigate them back to the login page.

Fig 3. Ingredients Screen Images

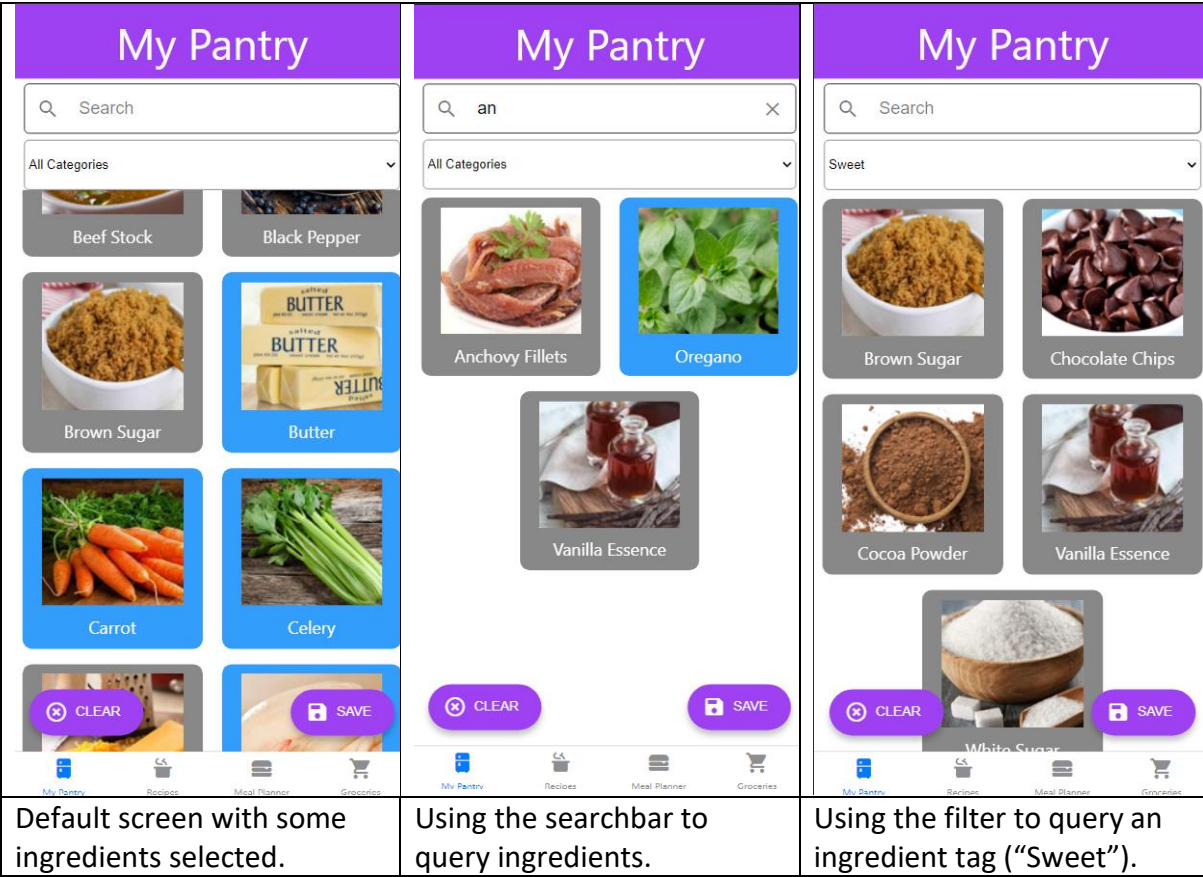


Fig 3. Description

The ingredients page shows all the available ingredients that are currently in recipes in the app. The user can scroll through the ingredients list and tap ingredients to select them. There are 2 forms of querying the ingredients list (as it can be quite large) to help the user find specific ingredients. A search bar, which can search ingredient names (queries can search in the middle of words, ie a search for “an” can retrieve “oregano”), and a filter which can select ingredients based on their tags (i.e sweet ingredients have the “Sweet tag”). The user is also presented with 2 buttons: a clear button which clears the entirety of the users selection, and a save button which stores the users ingredients on firebase, retrieving and autofilling them on app startup again.

As there are a lot of ingredients I wanted to present the user with the most ingredients possible whilst still keeping the screen clear and not cluttered. I chose to have 2 columns for ingredients and only display the image and title of each to optimize space and ease of use for the user to tap. As the user may wish to tap the clear or save buttons or query data from anywhere within the page, all these elements have absolute positioning to stay consistent on the page.

Fig 4. Recipe List Screen Images


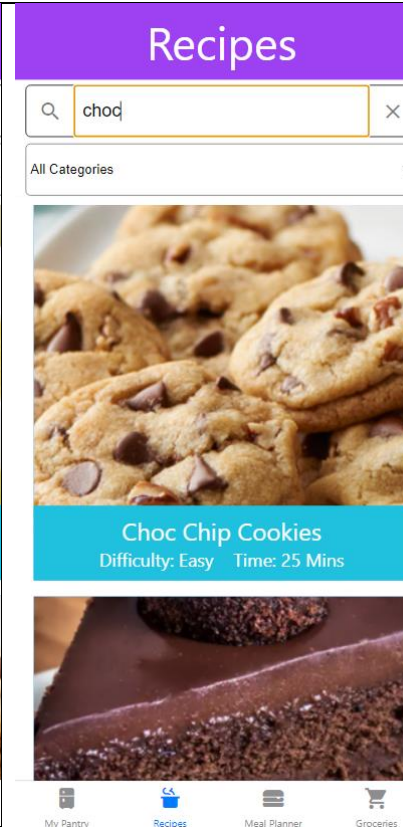
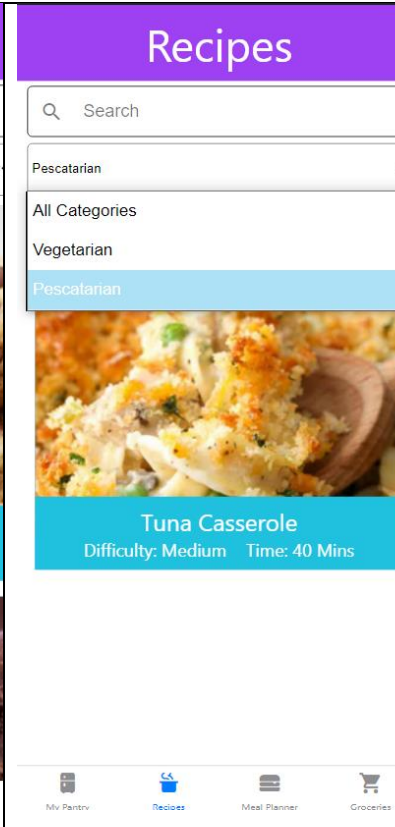
		
<p>Default Screen showing recipes the user has ingredients for.</p>	<p>Querying recipes using the search bar.</p>	<p>Using the filter to display ingredients with the "Pescatarian" tag.</p>

Fig 4. Description

The recipe screen shows all the recipes that the user can cook with, only containing recipes that the user has all the ingredients for. Like the ingredients screen, the user can also search or filter the recipes list to find items easier. Again, like the ingredients screen the searchbar in the recipes screen also picks up characters in the middle of words.

For the recipes screen I wanted to showcase each recipe and to do this I created a single column list with a large container to house each item. Only basic information is displayed to help the user come to the decision of which recipe to cook; Additional info can be provided by tapping the recipe and navigating to that recipes page.

Fig 5. Single Recipe Screen Images

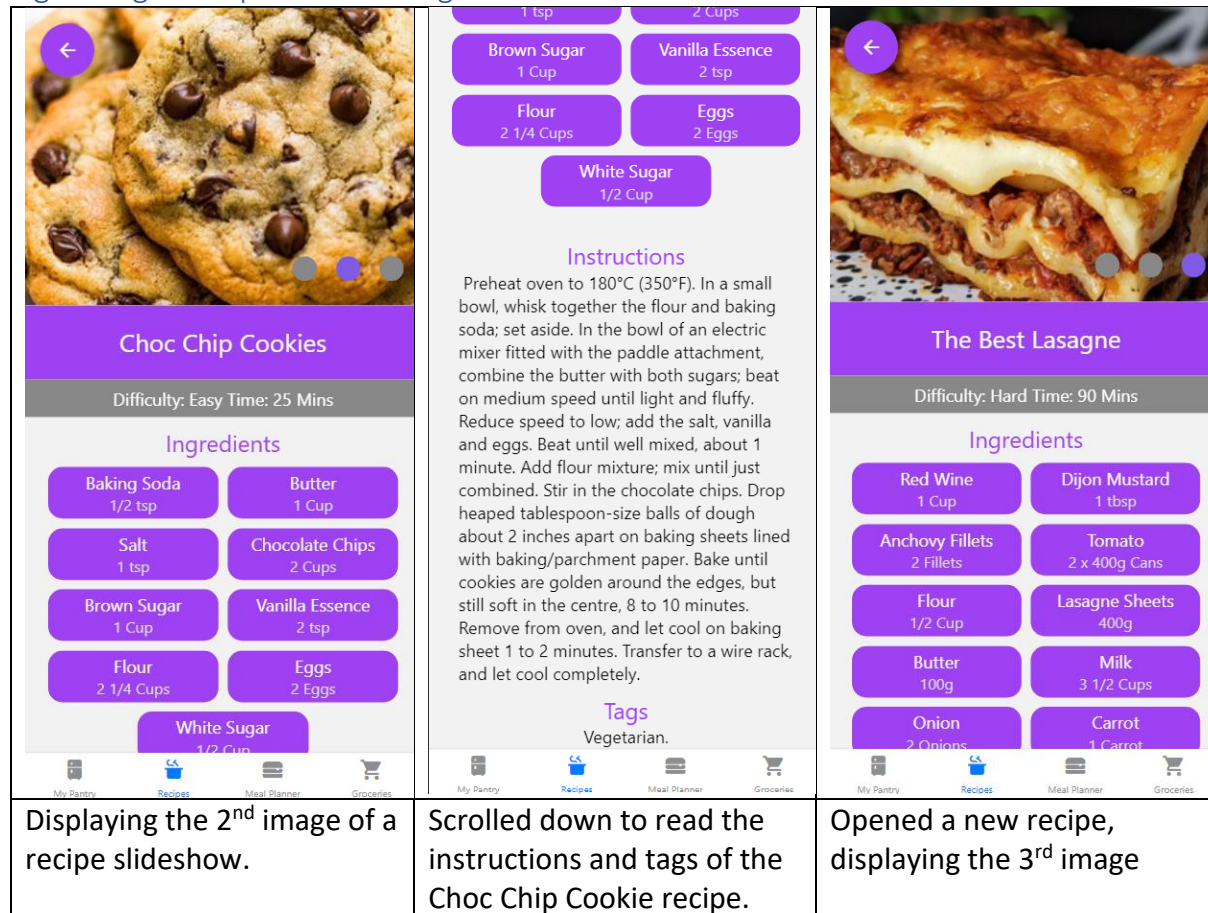


Fig 5. Description

The single recipe page is displayed when the user taps on an individual recipe to learn more about it. There is purely a page to display information about an individual recipe. Showing: the title, prep info, ingredients, instructions and additional tags. A slideshow was added, because if the user wanted to learn more about a recipe they would probably like to see more pictures as well.

I used a colour pallet of white, grey and purple, highlighting the key focus areas in purple as this would immediately attract the user to read these sections. All the information on this screen is displayed in a single column, making it easy to follow and cleaner to look at. I only displayed the ingredients in 2 columns to optimise space. All text has been padded off of the edges of the screen to increase readability and to avoid clutter.



Fig 6. Meal Planner Screen Images

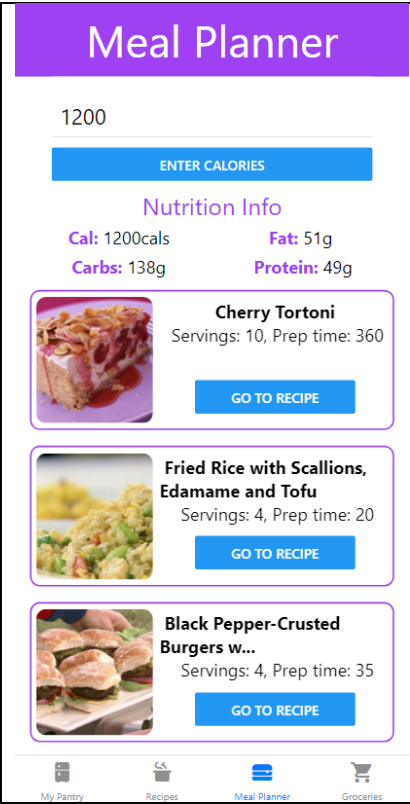
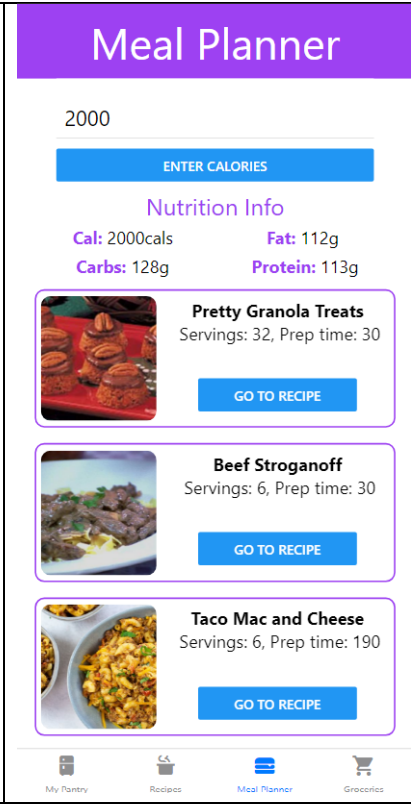
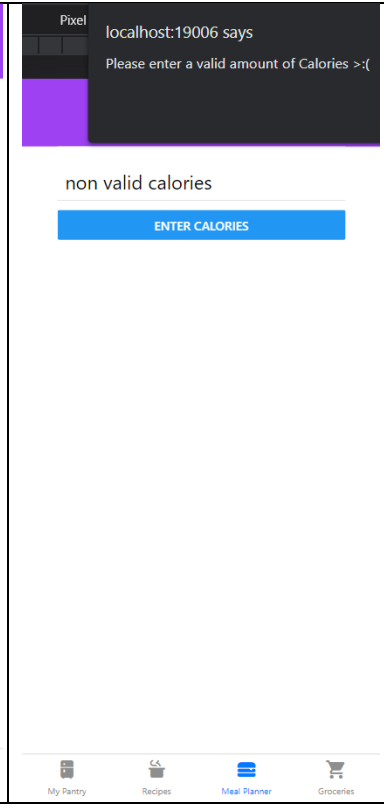
 <p>Meal Planner</p> <p>1200</p> <p>ENTER CALORIES</p> <p>Nutrition Info</p> <p>Cal: 1200cals Fat: 51g</p> <p>Carbs: 138g Protein: 49g</p> <p>Cherry Tortoni Servings: 10, Prep time: 360</p> <p>GO TO RECIPE</p> <p>Fried Rice with Scallions, Edamame and Tofu Servings: 4, Prep time: 20</p> <p>GO TO RECIPE</p> <p>Black Pepper-Crusted Burgers w... Servings: 4, Prep time: 35</p> <p>GO TO RECIPE</p> <p>My Pantry Recipes Meal Planner Groceries</p>	 <p>Meal Planner</p> <p>2000</p> <p>ENTER CALORIES</p> <p>Nutrition Info</p> <p>Cal: 2000cals Fat: 112g</p> <p>Carbs: 128g Protein: 113g</p> <p>Pretty Granola Treats Servings: 32, Prep time: 30</p> <p>GO TO RECIPE</p> <p>Beef Stroganoff Servings: 6, Prep time: 30</p> <p>GO TO RECIPE</p> <p>Taco Mac and Cheese Servings: 6, Prep time: 190</p> <p>GO TO RECIPE</p> <p>My Pantry Recipes Meal Planner Groceries</p>	 <p>localhost:19006 says Please enter a valid amount of Calories &gt;{</p> <p>non valid calories</p> <p>ENTER CALORIES</p> <p>My Pantry Recipes Meal Planner Groceries</p>
Inputting 1200 Calories and getting daily meals and their nutritional total value.	Inputting 2000 Calories and getting daily meals and their nutritional total value.	Entering in a non valid amount, getting an alert and not showing any meals.

Fig 6. Description

The meal planner screen allows the user to input a set amount of calories that they want to consume in the current day. Using the text input at the top, once pressing “enter calories” the user will be greeted with 3 meals that are planned for today. Total nutrition info is displayed to show the macronutrients. Each meal contains basic info and if the user wants to learn more about the recipe they can press the “go to recipe button” which will open a new webpage for that recipe.

I intentionally chose for the meals to all fit on one page, I felt as though if the images or text were too big then there would be a small amount of content spill over the page (which would be bad design). Again, I stuck to the app’s colour scheme of purple, white and blue. I highlight all user input functions in blue to make them clearly visible.



Fig 7. Grocery List Screen Images


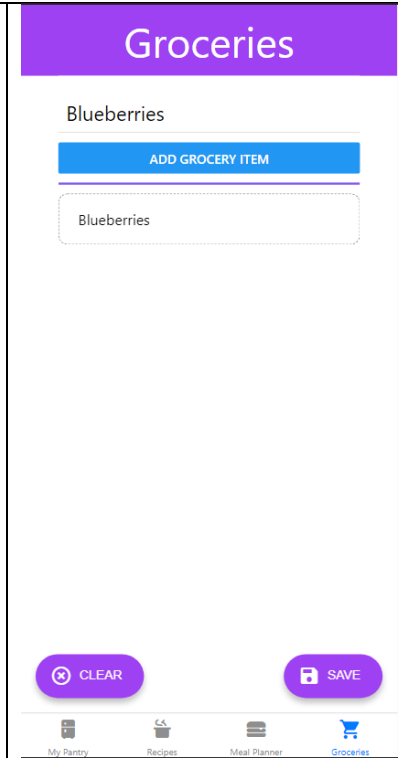
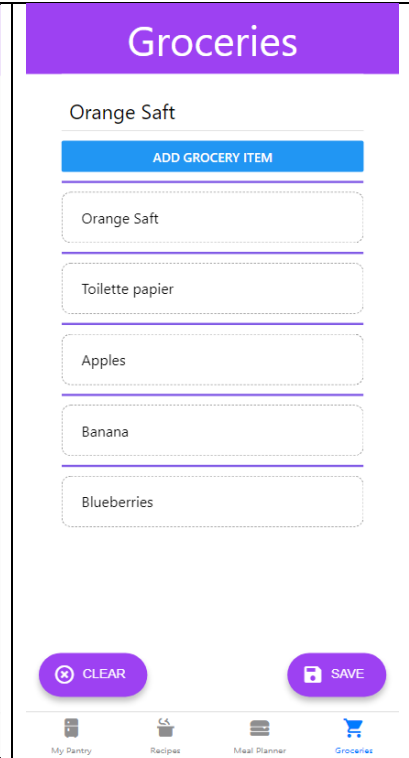
		
Default grocery list screen with no items added yet.	Adding an item, item displays in the list.	Adding more items to the list. Able to tap any to remove.

Fig 7. Description

This page allows the user to input grocery list items into the text input area and submit them into a displayable grocery list. The user is presented with the options to clear which will remove the list, or save which will push the list to firebase, storing and loading the list on-launch the next time the user logs in.

This is a very minimal one column page with not too many functions, giving the page a clean aesthetic. I originally added an “X” button to get rid of an item but later changed that to allow the user to tap anywhere on the item to clear it (as the only action the user can do is clear it, I felt it didn’t make sense to restrict that to a small, hard to press button, unless you wanted to restrict mis-tapping). Items are seperated by a purple line to clearly distinguish items. I kept to the same purple, blue and white colour pallet as the previous pages to maintain consistency within my app.