**CS673 Software Engineering** 

**Team 5 - MEALit**

**Software Design Document**

| Team Member | Role(s) | Signature | Date |
| --- | --- | --- | --- |
| David Vanderschaaf | Team Lead | David Vanderschaaf | 10/19/2022 |
| Jintong Song | Security Lead | *Jintong Song* | 10/17/2022 |
| Wenxiao Zhou | Design & Implementation leader | Wenxiao Zhou | 10/19/2022 |
| Xiaohang Lin | QA lead | *Xiaohang Lin* | 10/17/2022 |
| Alaap Bharadwaj | Requirements Leader | *Alaap Bharadwaj* | 10/18/2022 |
| [Fahmida Joyti](mailto:fsjoyti@bu.edu) | Configuration Lead | *Fahmida Joyti* | 10/17/2022 |
|  |  |  |  |
|  |  |  |  |

**Revision history**

| **Version** | **Author** | **Date** | **Change** |
| --- | --- | --- | --- |
| 1.0 | Whole Team | 10/19/2022 | Initial Version |
|  |  |  |  |

[Introduction](#_87t9hln2vjz0)

[Software Architecture](#_3ipvmjgn6clp)

[Class Diagram](#_ky60nv8suxxm)

[UI Design (if applicable)](#_7ucksmkf6rzx)

[Database Design (if applicable)](#_tcmuor4nl1kz)

[Security Design](#_x18fj36s1121)

[Business Logic and/or Key Algorithms](#_mtfbusfb0eq3)

[Design Patterns](#_9zvwkmc4luo5)

[Any Additional Topics you would like to include.](#_15tmymhipvdv)

[References](#_50ojo9i46ytq)

[Glossary](#_8n34lvocupub)

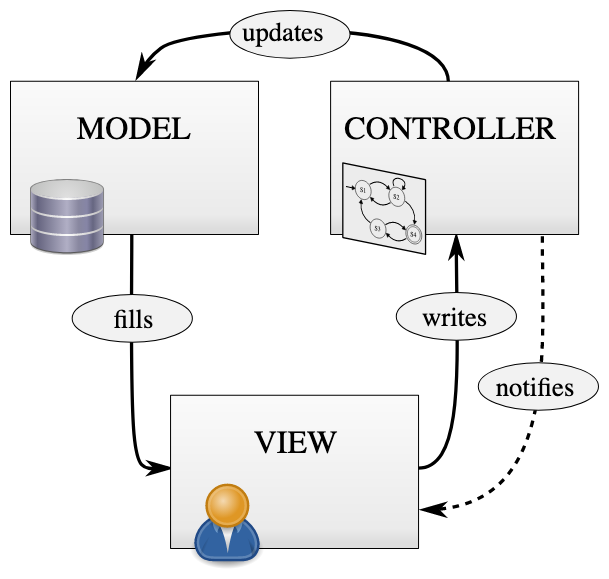
# Introduction

Our team is developing a web application that will allow users to find recipes based on the ingredients they already have on hand and to create grocery lists for the ingredients they do not yet have. Users will further be able to save their favorite recipes and previous grocery lists for easier reuse in the future.

The purpose of this document is to outline the software architecture and key design decisions that are implemented in this application.

# Software Architecture

**We are using the Spring Framework to manage component instances and providing a MVC design and architecture as shown by the diagram below**



-main

|– config

|—|-- SpringSecurity.java

|– event

|—|-- RegistrationCompleteEvent.java

|—|--listener

|—|---|-- RegistrationCompleteEventListener.java

|– dto

|—|-- UserDto

|– controller

|—|-- UserController.java

|– service

|—|-- UserService.java

|—|-- UserServiceImpl.java

|– repository

|—|-- GroceryListRepository.java

|—|-- IngredientRepository.java

|—|-- OrderRepository.java

|—|-- PreferencesRepository.java

|—|-- RoleRepository.java

|—|-- SavedRecipesRepository.java

|—|-- UserRepository.java

|– security

|—|-- CustomerUserDetails.java

|– entity

|—|-- GroceryList.java

|—|-- Ingredient.java

|—|-- Order.java

|—|-- Preferences.java

|—|-- Quantity.java

|—|-- Role.java

|—|-- SavedRecipes.java

|—|-- User.java

|—|-- VerificationToken.java

|– enums

|—|-- IngredientCategory.java

-test

We have four components called Service, Controller, Dto, Entity

The relationship is as defined:

**Controller -> Service -> Dto -> Entity**

The controller is the one that begins all the processes as it is the one that is called when the user clicks a tab which takes them to a different page.

Within the various packages we have interfaces for our implementation, for example, for our UserService we have an interface that highlights all the methods we will be using so that the UserController can call the UserService to access the methods defined by the UserServiceImpl as this follows the MVC model we chose to implement.

The basic structure of this is as follows:

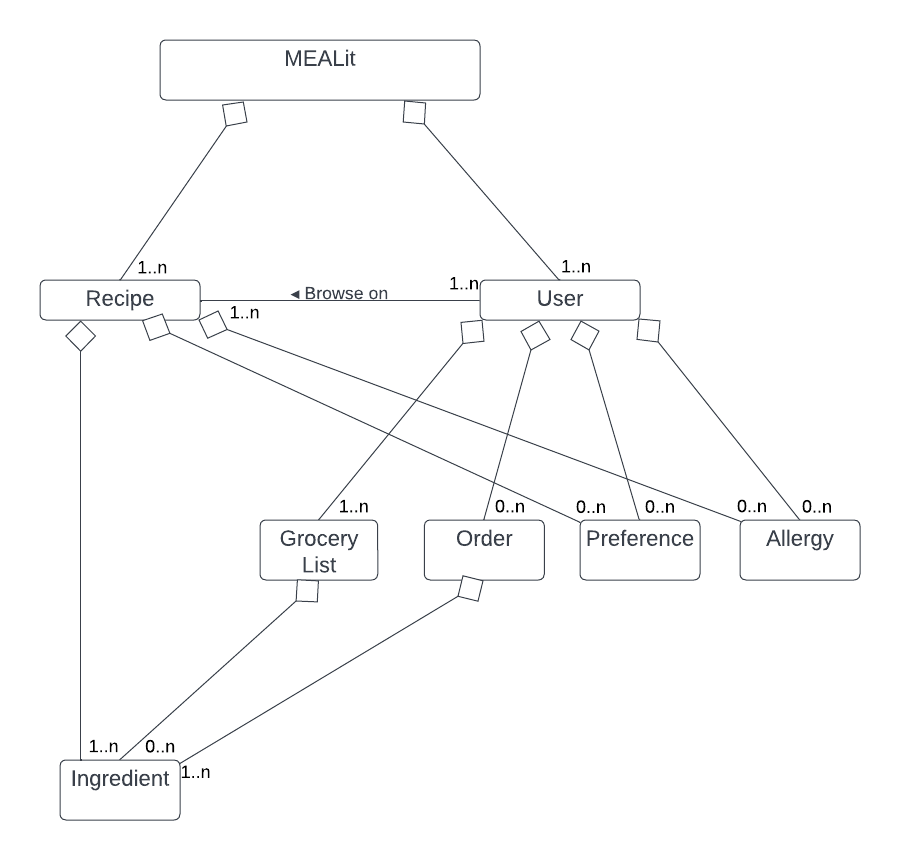
When the application is started, the templates are created for the user to be able to view. Each template is mapped to a certain api tag for example “/register”, which also corresponds to methods within the UserController class which handles the registration.

The framework we are using is the Spring framework, this is an open source framework that provides infrastructure to support the java application the developers are trying to create. The spring framework is great for this project as it helps create high performing applications using the java objects, not objects created by the framework. Spring is considered to be secure, low cost, and flexible making it desirable for Java developers.

# Class Diagram

Within the MEALit app, there are two major components: recipe and user. A general association with multiplicity of many-to-many exists between these two major components. They are the fundamental classes that build up the classification logic. The class user owns four subclasses including grocery list, order, preference, and allergy, while performing an aggregation relationship. For each user, he/she would be able to create a grocery list (ingredients needed for recipes that are not currently in the pantry)create a pantry and receive recipes according to the users filters, set recipe preferences (e.g. vegetarian), and set allergy filters (e.g. avoid recipes containing nuts). The class of preference and allergy are also two subclasses of the recipe class with an aggregation association. The ingredient class, as a subclass, has an aggregation relationship with the recipe class, the grocery list class, and the order class, since each class above contains ingredient elements.

As we start implementing more features in the coming iterations we will be updating this class diagram.



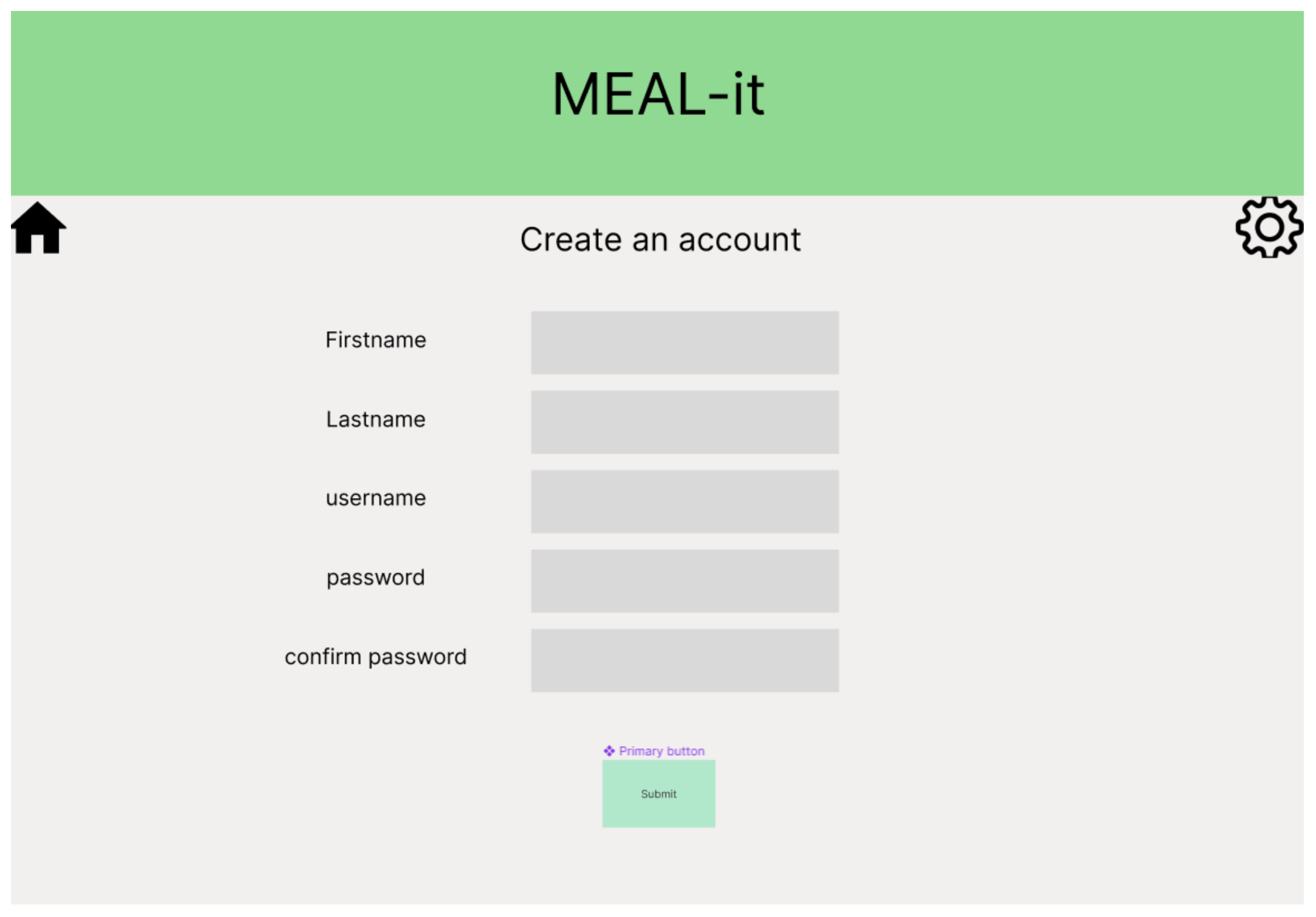
# UI Design

Our application will have several different screens to allow the user to interact with our product seamlessly.

The first webpage they will reach is the Login Page, which will enable them to put in their username and password to get access to their personalized profile.



If a user is not yet registered for the application, they can click the register link and input their information to add a user profile and access the web content.



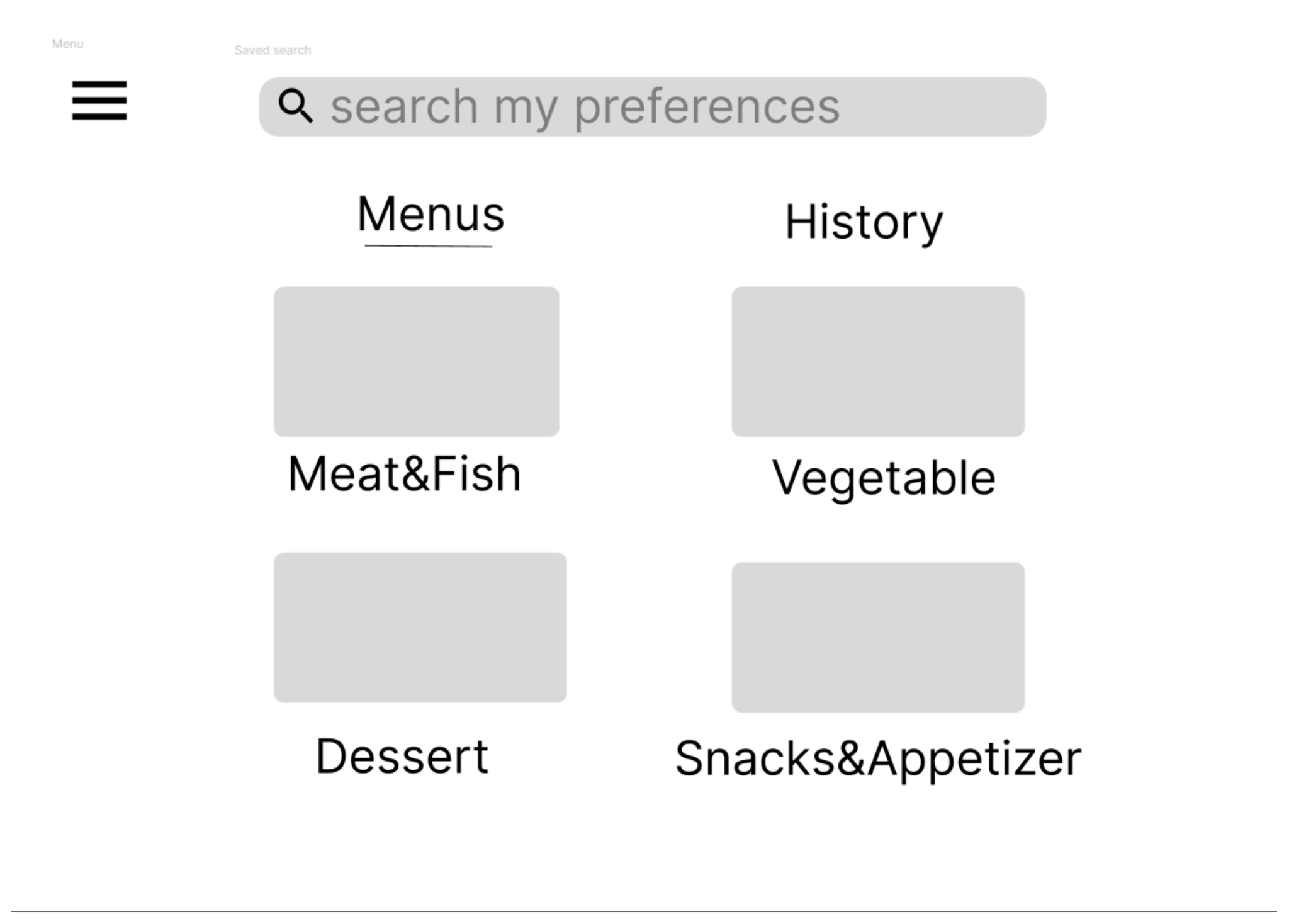
After registering and logging in, the user is taken to the main page which displays two main sections. The center section shows the recipe search results based on whatever keywords the user has entered or the ingredients they have selected. The left section is where the user can select the ingredients they have on hand and add them to their virtual pantry. Additionally on the right, there will be a collapsible preferences panel that will allow the user to add and update their preferences (like dietary restrictions or allergies).



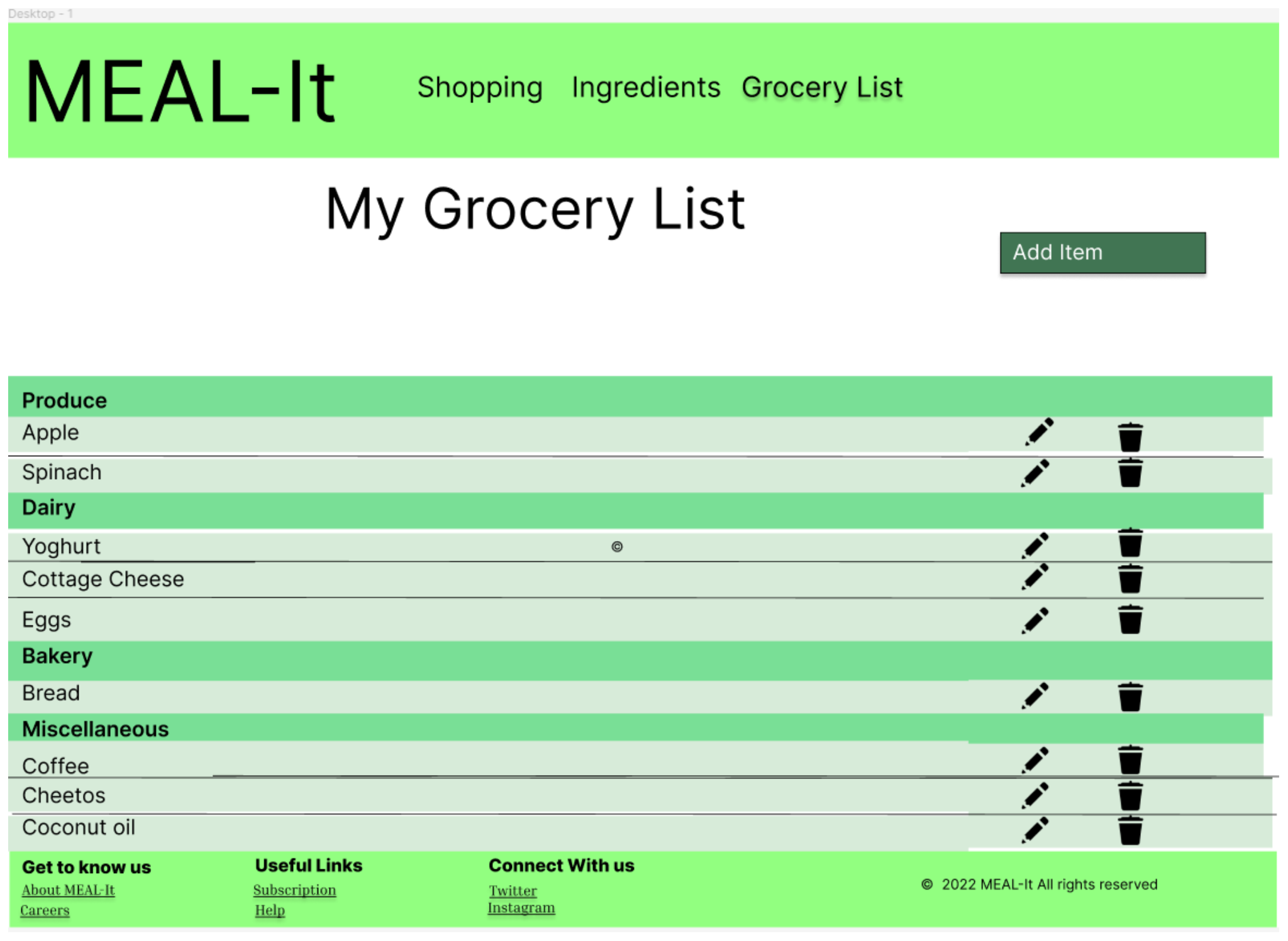
Below is a more detailed view of the center panel of the main page showing recipe results with images and needed ingredients.



Below is a detailed view of the layout of the preferences panel.



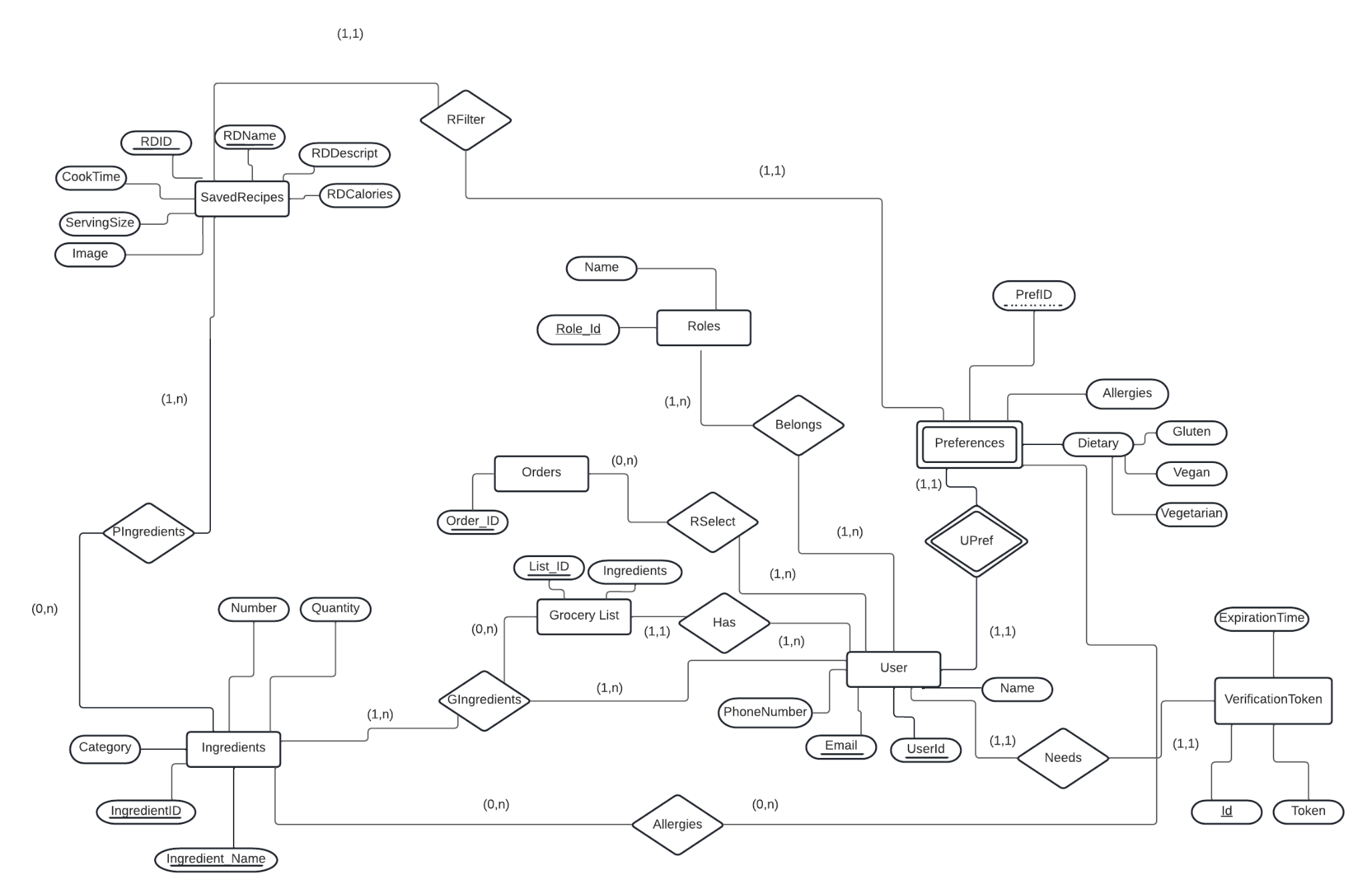
In addition to the main page, there will also be another page to display the user’s current grocery list. When a recipe is selected, the ingredients that are not currently in the pantry will automatically be added to the grocery list to make shopping for those items easier. Users can also manually add, remove, or update items to the grocery list from this page.



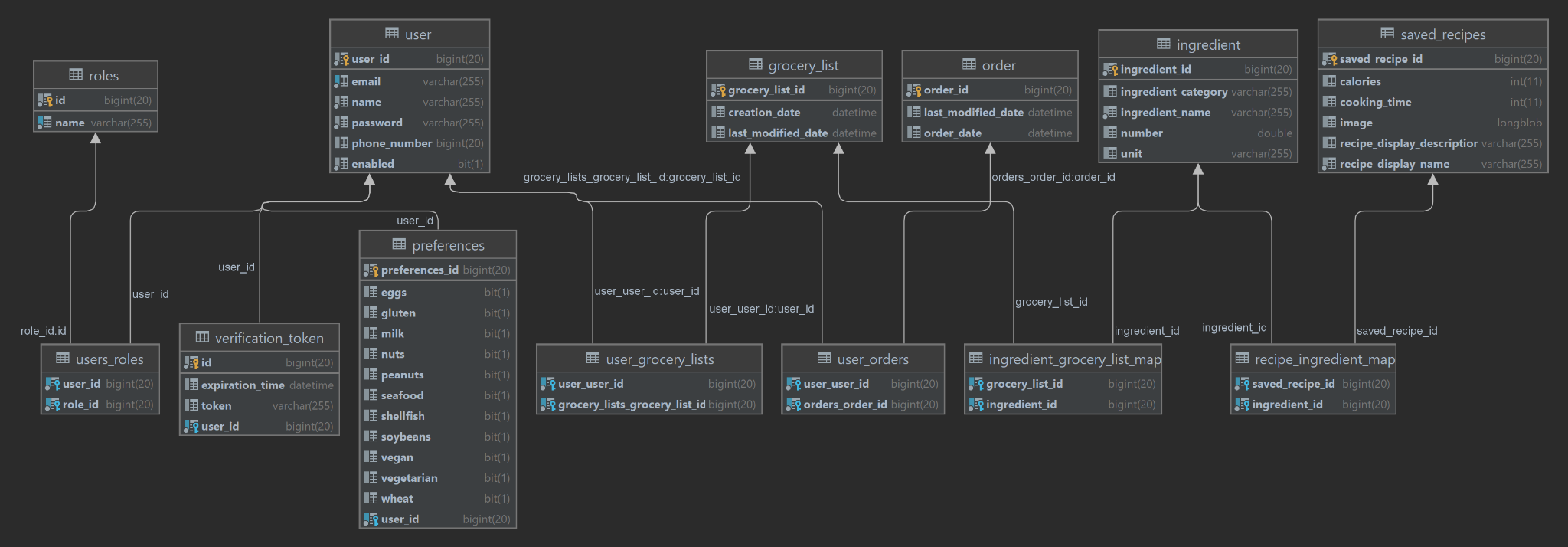
# Database Design

We first created an ER diagram to define all the entities and relationships between them. Then we mapped the ER diagram to the relational schema in MySQL database.

**ER diagram**



**ER diagram mapped to tables in the database**

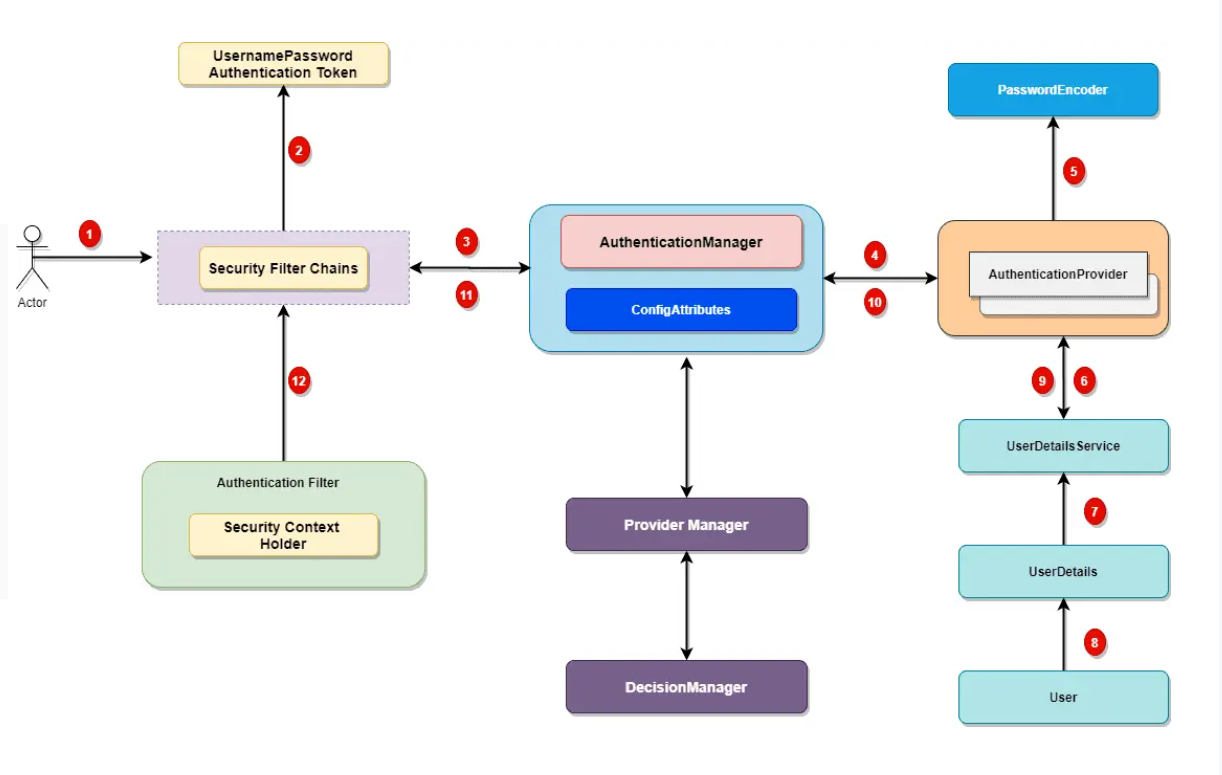


# Security Design

In this project, our team is using Spring Security as our security framework. Spring Security's support for Web security relies heavily on Servlet filters. These filters intercept incoming requests and do security processing before the application processes the request. Spring Security provides several filters that enhance security by intercepting Servlet requests and forwarding them to the authentication. We are also using Spring Security’s password encoder so the user’s actual passwords are not stored in a central database.

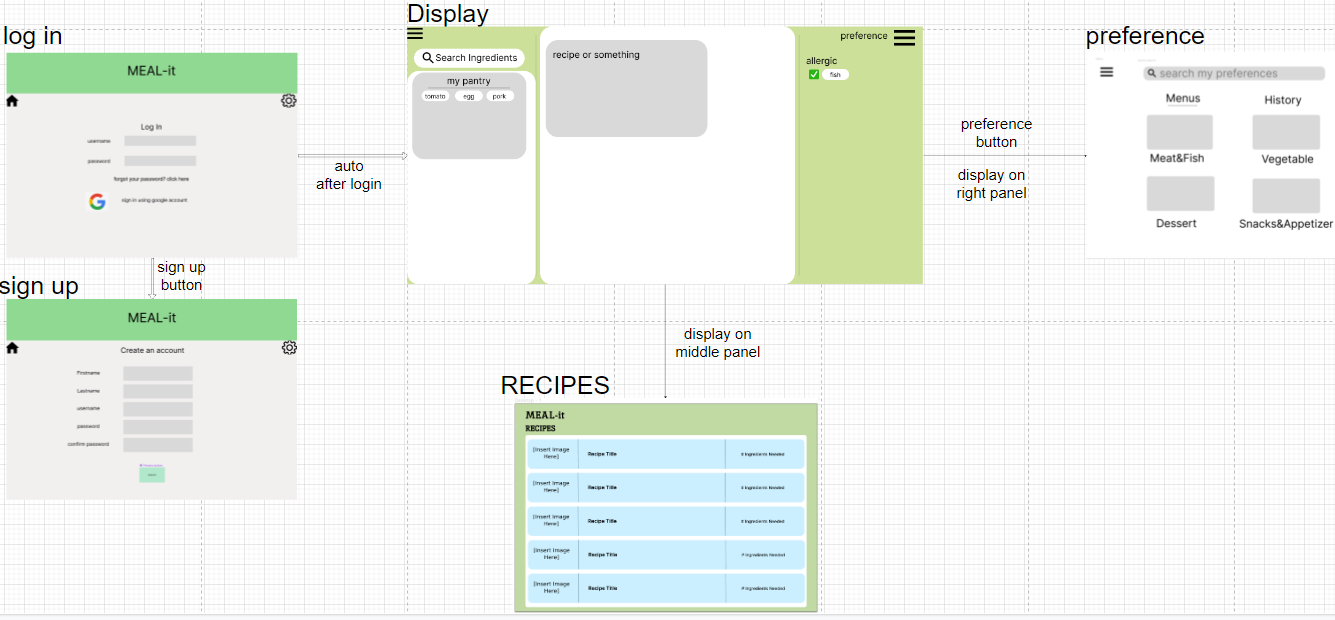
One of the core aims for any security framework is to verify the caller’s claim, the caller is who they claim to be. Authentication is the process to validate credentials and the caller’s claim.

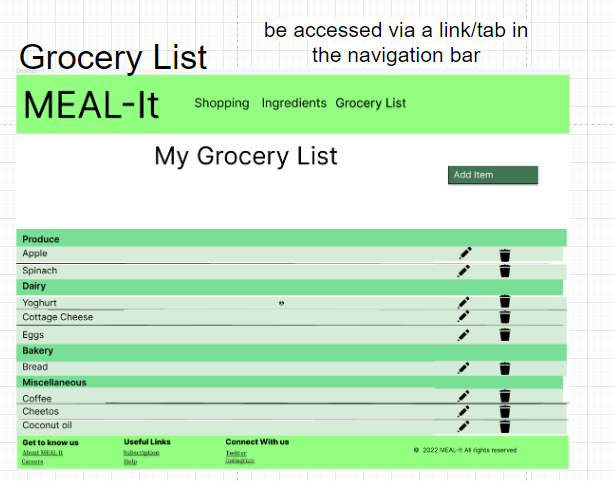
Here’s the top level workflow of the classes and filters involved in the *security authentication* process.



Spring Security comes with a set of security filters. Each request to the server is intercepted by these filters. These filters will process the request based on the logic and will pass or reject the incoming request.[1]

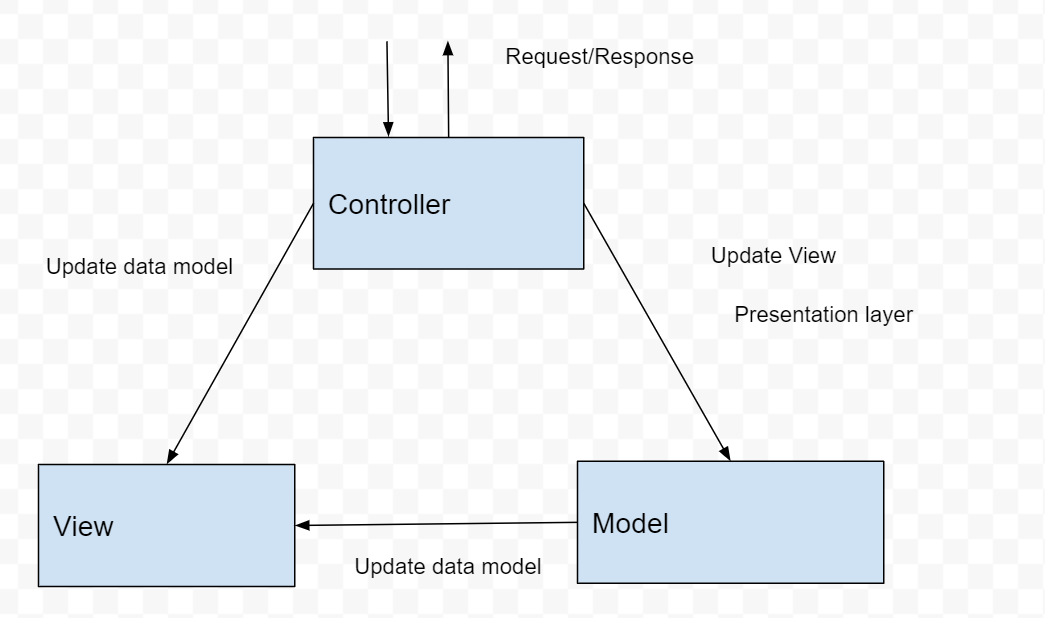
# Business Logic and/or Key Algorithms





# Design Patterns

**MVC(Model View Controller)**



MVC (Model View Controller) is a design pattern that is used to designate objects in our code to one of the three sections : data layer (model), presentation layer (view) and business logic layer (controller). These layers are separated from each other through interfaces and communicate with each other using those interfaces. MVC allows separation of concerns and enables the code to be more reusable and extensible.

# Any Additional Topics you would like to include.

# References

[1] <https://www.javadevjournal.com/spring-security/spring-security-authentication/>

[2]“Cocoa Core Competencies.” Model-View-Controller, 6 Apr. 2018, https://developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html.

[3]Pedamkar, Priya. “What Is MVC Design Pattern?” EDUCBA, 30 Mar. 2021, www.educba.com/what-is-mvc-design-pattern.

# Glossary