Department of Computer Science & Engineering
The University of Texas at Arlington

# Architectural Design Specification
# CSE 4316: Senior Design I
# Fall 2021


## STEAM
## GROCO

Patrick Faulkner
Hozefa Tankiwala
Andrew Hands
Kiran Karki
Uyen Do

## Revision History

| Revision | Date | Author(s) | Description |
|---|---|---|---|
| 0.1 | 10.29.2021 | PF | document creation |
| 0.2 | 11.05.2021 | PK, HT, AH, KK, UD | first draft |

# CONTENTS

## List of Figures



## List of Tables

# 1 INTRODUCTION

Groco is a grocery shopping web application that is accessible on PCs, smartphones, and tablets. Users will be able to search for grocery items and based on the user's brand preference and location, the system will suggest the optimal grocery items. The system also allows users to search for recipes, add their recipes and meal plans into their shopping list to perform the optimization and navigation route.

The purpose of this product is to help users with grocery shopping. The key requirements are:

- **The application must allow a user to search grocery items**: A search functionality must be implemented into the application which will allow the user to search for a specific grocery item and then allows the user to add that item to their shopping/grocery list. The user must be allowed to type in the grocery item they are looking for and the search should find the matching item for the user to add to the list.
- **The application must present the user with the best grocery item**: Based on the item that the user searched for, the application should go and look for the best possible match for that item based on the price of the item and the location of the stores that item is available in. The application may list multiple options for the same item depending on the item's price and store's location.
- **The application must allow users to choose references that define optimal items**: The user should be able to choose the brand, price, distance, and maximum stores preferences for certain grocery items.
- **The application must allow users to create grocery lists**: The application must allow users to create shopping lists. The shopping list must store multiple items. The user should be able to add items to their shopping list by searching for the item
- **The application must search for all items on the shopping list**: The application must search for all items in the shopping list and return the optimal items, their stores, and their prices based on user-specified preferences
- **The application search for optimal items from more than one store**: When the user searches their entire grocery list, the application must search for the optimal results from more than one store.
- **The application must provide the most optimal route to the user**: If the user opts to visit multiple stores to get their groceries then the application should provide the user with an optimal route and the order in which the user should visit those stores. The application must consider multiple factors in deciding the route.
- **The application must allow users to view and share recipes**: Users should be able to look up recipes in the application. Users must also be allowed to create their recipes and save them. The recipes will store both the ingredients for that recipe and the procedure to follow. Users must be able to view and share recipes with other users.
- **The application must allow users to add all ingredients from a recipe to their shopping list**: The application must allow the user to add all the ingredients of a recipe to their shopping list.
- **The application must allow users to keep a list of favorite grocery items**: The application must allow users to add and remove grocery items to a favorites list. This will allow users to quickly add certain grocery items to their shopping list.

The key requirement of this product is to optimize the grocery items based on users' preferences. The user's preferences include brand, number of stores, and max traveling distance. The system will choose the optimal item by prices and distance.

The scope of the product covers grocery stores that have their products and store information online and allow third party access to request or collect data through their public API or web-scraping within the United States. The product prototype should be completed within the budget of 800 dollars and by April 22, 2022.

The main assumptions in this project are that users have internet access, the items' prices and store information collected online are accurate. Groco is not responsible for the price changes or stock-out at the stores. It is the user's responsibility to make sure that a price and discount offer is present at the stores since they are subject to change without notice.

The product contains no obscene material; therefore it is suitable for general grocery shoppers and is made available publicly and free of charge for all users.

The final product will be tested and approved by the client. The development team is free to decide which developing tools and programming languages to be used.

## 2 System Overview

Groco consists of four main layers: the front-end (the client), the back-end (the server), the database, and the data collector. To create an account and store a user's information, the font end layer will take inputs from the users, send them to the back-end layer to validate, and finally store them in the database. Similarly, the front-end layer will send a request to the back-end and the back-end can retrieve the data from the database then return the appropriate data to display on the front-end. To search for the items, the front-end will take inputs from the users, send them to the back-end layer to process, the back-end will request data from the data collector, using the collected data the back-end complete the request and return the result to display on the front-end.

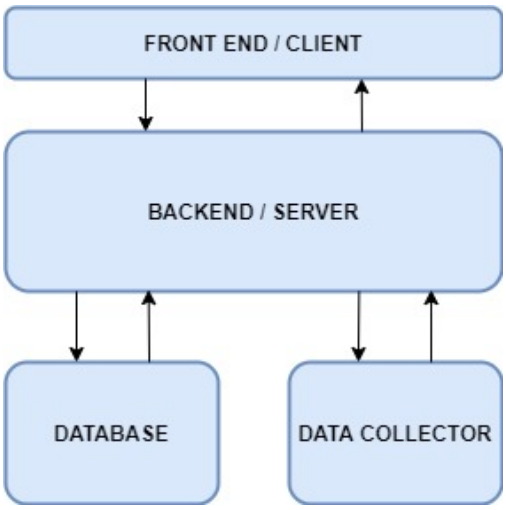


Figure 1: A simple architectural layer diagram

### 2.1 Front-end Description

The front-end layer includes all software that is part of the product interface. The software is the code that is executed on the client-side (typically HTML, CSS, and JavaScript) that runs in the user's browser to create the user interface. Users interact directly with different components of the front-end, including user-entered data, buttons, links, and other features. The front-end of this application includes subsystems such as login, register, shopping list, recipes, meal planning, and shop. The front-end is designed to be accessible, pleasant, and easy to use.

### 2.2 Server Description

The back-end layer is the code that runs on the server. The back-end receives requests from the front-end (client) and contains the logic of the application to process each request and return appropriate data to the client. The back-end can directly interact with the database and data collector to retrieve the required data to fulfill each request. The back-end layer includes subsystems such as shopping manager, data parser, user management, query management, and database controller.

### 2.3 Database Description

The database layer stores and retrieves data. The database is also responsible for managing updates. The database layer includes multiple data tables that correspond with different functionalities of the product such as a table for users, meal plan, shopping list, item, item-brand, brands, recipes, and recipe ingredients.

## 2.4 Data Collector Description

The data collector layer is responsible for retrieving data from multiple sources. There is one subsystem in the data collector layer, the API Manager. The data collector retrieves data from various stores through their respective APIs and returns it to the backend layer.

# 3 Subsystem Definitions & Data Flow

Figure 2 is the data flow diagram for Groco. The diagram shows the Graphical User Interface, Server/Backend, Database, and Data Collector along with their subsystems and the data flow between them.

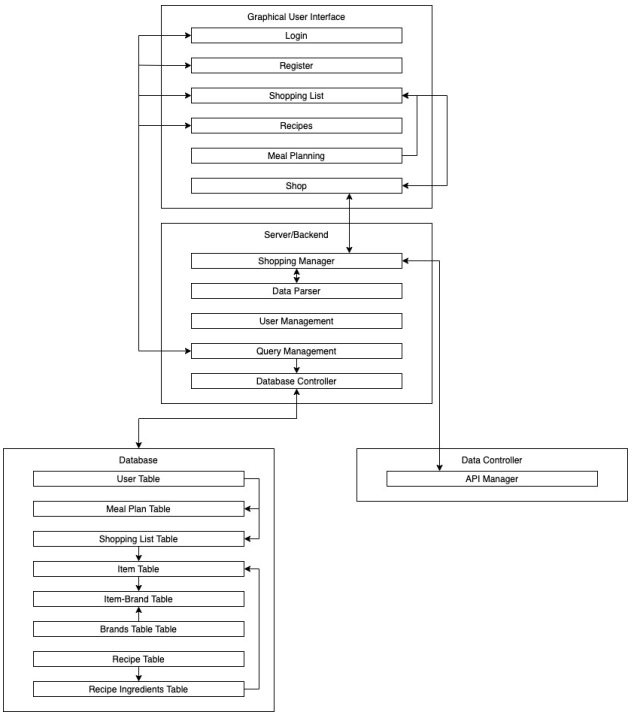


Figure 2: Groco data flow diagram

# 4 Front End Layer Subsystems

## 4.1 Login Subsystem

The Login Subsystem will be the first subsystem that users will interact with when first entering the application. This subsystem will allow users to login with their unique username or email, and their password. Once the credentials are confirmed to be correct, the system will redirect the user to the home page.

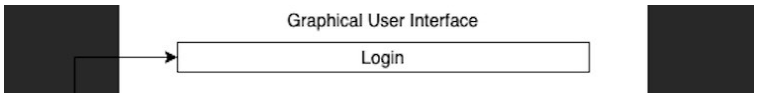


Figure 3: Login Subsystem

### 4.1.1 Assumptions

There will be the assumption that any user who attempts to login, will have already registered and has a unique username/email and set a password.

### 4.1.2 Responsibilities

The primary responsibility of the Login Subsystem will be to provide an interface for users to sign into the application. Upon receiving a user's unique username or email and password, the Login Subsystem will relay these inputs to the Query Manager subsystem to be processed. The subsystem will then wait to receive confirmation. If the user entered credentials are incorrect, then the Login Subsystem will give an error message and prompt the user to re-enter their login information. If the credentials are correct, then the Login Subsystem will redirect the user to the home page.

### 4.1.3 Subsystem Interfaces

Each of the inputs and outputs for the subsystem are defined here.

Table 2: Subsystem interfaces

| ID | Description | Inputs | Outputs |
|---|---|---|---|
| #1 | User Email or Unique Username and Password | Email or Username<br>Password | Page Redirect or Error message |

## 4.2 Register Subsystem

The Registration Subsystem will allow new users to sign up and create their account with a unique email. Users should only need to use this subsystem once.


Figure 4: Register Subsystem

### 4.2.1 Assumptions

All users who register have an email that they can access.

### 4.2.2 Responsibilities

The primary responsibility of the Register Subsystem is to provide an interface for users to sign up to use the application. Users will enter their email address, unique username, and password. Once the Register Subsystem receives these inputs, it will relay them to Query Management subsystem. If the user enters an email or username that is already being used, the Register Subsystem will display an error message and prompt the user for a new email or username. If the user gives all valid inputs, the Register Subsystem will give a confirmation and redirect the user to the login page.

### 4.2.3 Subsystem Interfaces

Each of the inputs and outputs for the subsystem are defined here.

Table 3: Subsystem interfaces

| ID | Description | Inputs | Outputs |
|---|---|---|---|
| #1 | User Email, Unique Username, and Password | Email and Username<br>Password | Page Redirect or Error message |

## 4.3 Shopping List Subsystem

The Shopping List Subsystem will provide an interface for users to search for their desired grocery item. This will also show users all the items that are currently on their shopping list.


Figure 5: Shopping List Subsystem

### 4.3.1 Assumptions

There are no assumptions for this subsystem.

### 4.3.2 Responsibilities

The Shopping List Subsystem will have two responsibilities. First it will communicate with the Query Manager subsystem to obtain the users current grocery list, it will then display this list for the user to see. The Shopping List Subsystem will also allow user to search for items to add to their list.

Users will be able to type in the name of a desired grocery item. The system will send this input to the Query Manager to retrieve the results. This subsystem will then display all results. The user can then select the item they want. The Shopping List subsystem will send this selection to the Query Manager to be added to the users Shopping List.

Finally, if a user is ready to shop for all of their grocery items, they will select the option to shop and control will be transferred to the Shop Subsystem.

### 4.3.3 Subsystem Interfaces

Each of the inputs and outputs for the subsystem are defined here.

Table 4: Subsystem interfaces

| ID | Description | Inputs | Outputs |
|---|---|---|---|
| #1 | Display Shopping List | - | User ID to Query Manager |
| #2 | Display Shopping List | Shopping List Query Manager | Display Shopping List to User |
| #3 | Grocery Item Search | Grocery Item from User | User input to Query Manager |
| #4 | Search Results | Search Results from Query Manager | Display Search Results to User |
| #5 | User Select Results | User Selected Item | Selection to Query Manager |

## 4.4 Recipes Subsystem

The Recipes Subsystem will allow users to view recipes, add a recipe's ingredients to their shopping list, and add their own recipes.


Figure 6: Recipes Subsystem

### 4.4.1 Assumptions

User will be responsible for ensuring that recipes do not conflict with their dietary restrictions. It will also be assumed that users will not add dangerous recipes.

### 4.4.2 Responsibilities

The first responsibility of the Recipes Subsystem will be to provide an interface for users to view and search recipes. Upon entering the Recipes subsystem, users will be shown recipes that were randomly selected using the Query Manager. Users will then have the ability to search for recipes by using keywords or searching the name of a known recipe. Upon receiving search text from the user, the system will relay that to the Query Manager to retrieve and then display all results matching the search.

After viewing the recipes, users will be able to select individual recipes. This will allow them to see all the ingredients and instructions. If the user likes the recipe, they will be able add it to their meal plan or add all the ingredients directly to their shopping list.

The Recipes Subsystem will also enable users to share their own recipes. If a user decides to do this, the user will first select all the ingredients and the amounts needed. After all ingredients are selected, the user will then be able to enter step-by-step instructions that describe how to create the recipe. Once this is complete, the system will send the user provided information to the Query Manager to be added to the recipe database.

### 4.4.3 Subsystem Interfaces

Each of the inputs and outputs for the subsystem are defined here.

Table 5: Subsystem interfaces

| ID | Description | Inputs | Outputs |
|---|---|---|---|
| #1 | Recipe Search | User Search Text | User Search Text to Query Manager |
| #2 | Recipe Search Results | Results of Search from Query Manager | Display Search Results to User |
| #3 | Select Recipe to View | User Recipe Selection | Recipe Selection to Query Manager |
| #4 | Display Recipe to View | Recipe Ingredients and Instruction from Query Manager | Display Recipe Ingredients and Instruction |
| #5 | Add Recipe to Shopping List | User Choice to Add Recipe to Shopping List | All Ingredients to Query Manager |
| #6 | Add Recipe to Meal Plan | User Choice to Add Recipe to Meal Plan | Recipe ID to Query Manager |
| #7 | Add Own Recipe | All Ingredients and Instructions | User Given Ingredients and Instructions to Query Manager |

## 4.5 Meal Plan Subsystem

The Meal Plan Subsystem will allow users to view, create, and delete collections of recipes called Meal Plans. Users will be able to add recipes (via the Recipes Subsystem), delete recipes, and add all ingredients from a Meal Plan to their shopping list.


Figure 7: Meal Plan Subsystem

### 4.5.1 Assumptions

There are no assumptions for this subsystem.

### 4.5.2 Responsibilities

The Meal Plan Subsystem will allow users to create new Meal Plans that will be empty. Once a user chooses to do this, the Query Manager will be responsible for the creation of the Meal Plan in the database. A user will be able to add recipes to the their Meal Plan via the Recipes System. Once there is more than one recipe in a Meal Plan, the system will allow the user to view each Meal Plan and select

one to view individually. If a user selects a Meal Plan to view, the Meal Plan subsystem will retrieve all the recipes through the Query Manager and display them to the user.

If a user decides they would like to use a Meal Plan, they will be able to add it to their shopping list. This will be done by retrieving all the ingredients from each recipe via the Query Manager and then adding them to the Shopping List.

### 4.5.3 Subsystem Interfaces

Each of the inputs and outputs for the subsystem are defined here.

Table 6: Subsystem interfaces

| ID | Description | Inputs | Outputs |
|---|---|---|---|
| #1 | View Meal Plans | - | User ID to Query Manager |
| #2 | View Meal Plans | User's Meal Plan from Query Manager | Display All of the User's Meal Plans |
| #3 | Create Meal Plan | User Selection to Create a Meal Plan | Creation Command to Query Manager |
| #4 | Select Single Meal Plan | User Meal Plan Selection | Meal Plan Selection to Query Manager |
| #5 | View Single Meal Plan | Recipes in Meal Plan from Query Manager | Display All Recipes to User |

## 4.6 Shop Subsystem

The Shop Subsystem will act as an interface to present users with the optimal grocery trip.

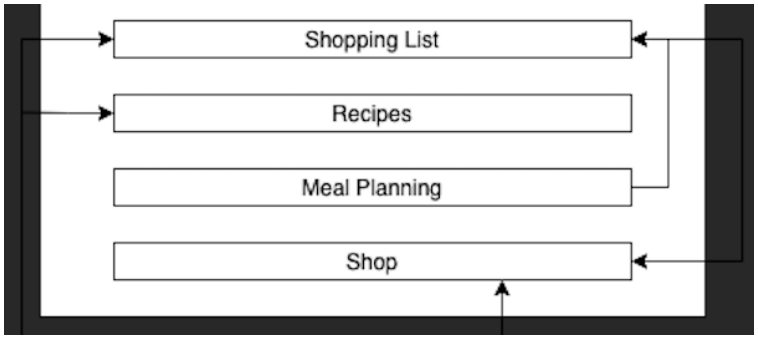


Figure 8: Shop Subsystem

### 4.6.1 Assumptions

There are no assumptions for this subsystem.

### 4.6.2 Responsibilities

The system will access the Shop Subsystem via the Shopping List Subsystem when a user decides they are ready to shop for the grocery items on their list. Once the subsystem receives the grocery items from the list, it will send that information to the Shopping Manager Subsystem along with the user defined preferences. All calculation will be done on the Server Layer. Once the Shop Subsystem receives the

results, it will display the stores the user should visit, the items they will get at each store, the sequence the stores should be visited, and a link to a map application to navigate their route.

### 4.6.3 Subsystem Interfaces

Each of the inputs and outputs for the subsystem are defined here.

Table 7: Subsystem interfaces

| ID | Description | Inputs | Outputs |
|---|---|---|---|
| #1 | Initiate Shopping | Shopping List Items<br>User Preferences | Shopping List Items and User Preferences to the Shopping Manager |
| #2 | Display Results | Optimal Shopping Trip From Shopping Manager | Display Optimal Shopping Trip to User |

# 5 BACK END LAYER SUBSYSTEMS

This will be the actual server or the back end for the application which will handle most of the heavy computations required for the application. The back end will handle requests coming in from the front end, interact with the database and the data control layer to provide the front end with the appropriate response. The back end contains the following subsystems:

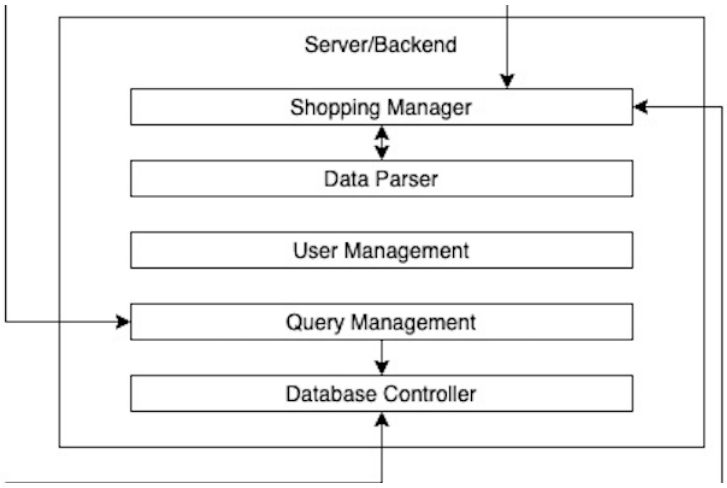


Figure 9: Back End Subsystems

## 5.1 SHOPPING MANAGER SUBSYSTEM

The shopping manager subsystem will deal with all the shopping related computation. It will get requests directly from the shopping subsystem of the front end. Then it will get the required shopping data from the Data Controller layer, use the Data parser subsystem to parse the data into standard format, and then send it as a response to the front-end.

### 5.1.1 ASSUMPTIONS

The requests from the front end are in standard format and the data is found by the data controller layer.

### 5.1.2 RESPONSIBILITIES

The shopping manager subsystem is responsible for getting prices for grocery items, entire shopping lists, and calculating the optimal route for shopping based on user preferences such as brand and maximum travel distance. The algorithm will optimize by distance and price to generate the list of items and where to buy them for the user and send this data back to the front end.

### 5.1.3 Subsystem Interfaces

Table 8: Subsystem interfaces

| ID | Description | Inputs | Outputs |
|---|---|---|---|
| #1 | Request for a single item | Standard name for that item | List of places and prices |
| #2 | Optimal route | Grocery List | Optimal Route |
| #3 | data from controller | Take data from controller | Send it to parser |

## 5.2 User Management Subsystem

The user management subsystem will handle all requests from the front end related to the user of the application and give back appropriate response. It will keep track of the currently logged in user.

### 5.2.1 Assumptions

There are no assumptions for this subsystem.

### 5.2.2 Responsibilities

The user management subsystem will be responsible for handling all user related tasks such as updating user profile and keeping track of their preferences. This will also keep track of how long the user has been logged in and determine when to automatically log them out.

### 5.2.3 Subsystem Interfaces

Table 9: Subsystem interfaces

| ID | Description | Inputs | Outputs |
|---|---|---|---|
| #1 | Login | Response from Query Manager | Keep track of the logged in user |
| #2 | Update user profile | New user data | Success or failure |

## 5.3 Query Management Subsystem

The query management subsystem will handle all requests from the front end that needs a result from a query on the database and give back the appropriate response. Also give the current logged in users info to user management subsystem to keep track of the user.

### 5.3.1 Assumptions

Email format is checked at the front end before passing it to the Query Management System.

### 5.3.2 Responsibilities

The query management subsystem will be responsible for handling all user related tasks such as account creation, logging in, and logging out. For user registration it would take username or email, password and other user details and store it into the database. For login it would take the username or email, & password and check it against the username stored in the database and determine login success and

relay the message back to the front end and also get the user data and pass it to user management to keep track of the logged in user.

The query management subsystem will also create all queries for users when they want to view their shopping lists, meal plans, and recipes. It will also create queries for users when they are searching for grocery items to add to their shopping lists.

#### 5.3.3 Subsystem Interfaces

Table 10: Subsystem interfaces

| ID | Description | Inputs | Outputs |
|---|---|---|---|
| #1 | Register | Email and password | Response from database |
| #2 | Login | Email and password | Success or Failure |
| #3 | Requests from Shopping List | Username or Email | User's Shopping List from Database |
| #4 | Requests from Recipes | Username or Email | User's stored recipes |
| #5 | Search for Grocery Items | Search text from front-end | Search Results |

### 5.4 Database Controller Subsystem

The database controller will handle all the communication of the application to the database. It will give responses to the query management system with the required data extracted from the database.

#### 5.4.1 Assumptions

The query management system will protect the database controller from attacks like SQL Injection.

#### 5.4.2 Responsibilities

The database controller subsystem will be responsible for handling all communications to and from the database. It will create the initial connection with the database on the application startup. Every query to the database will go through this controller. It will get login, registration, shopping list, meal plans, recipes, grocery items, and update queries from the query management system and provide responses from the database.

#### 5.4.3 Subsystem Interfaces

Table 11: Subsystem interfaces

| ID | Description | Inputs | Outputs |
|---|---|---|---|
| #1 | Requests from Query manager | Email and password | Response from database |

# 6 Z Layer Subsystems

Groco is to use a relational database to keep backend information uniform and well structured. The database will not maintain any information gathered from grocery searches. Since such information would become obsolete so quickly, the userbase would have to become rather large for the hit rate of such a cache to make it worth having. Furthermore, since the collection of such data is in its own subsystem, the integration of such a cache later on would require relatively little more work than would be required to have it added intially, thus, it is better to leave such a system for future analysis and upgrades. Note that the following subsections reference figure 6.1, but this figure is only shown once in subsection 6.1 to avoid being verbose.

## 6.1 UserTable Subsystem

The UserTable Subsystem will be used for storing identifiable information about each user and holding information to help the user.

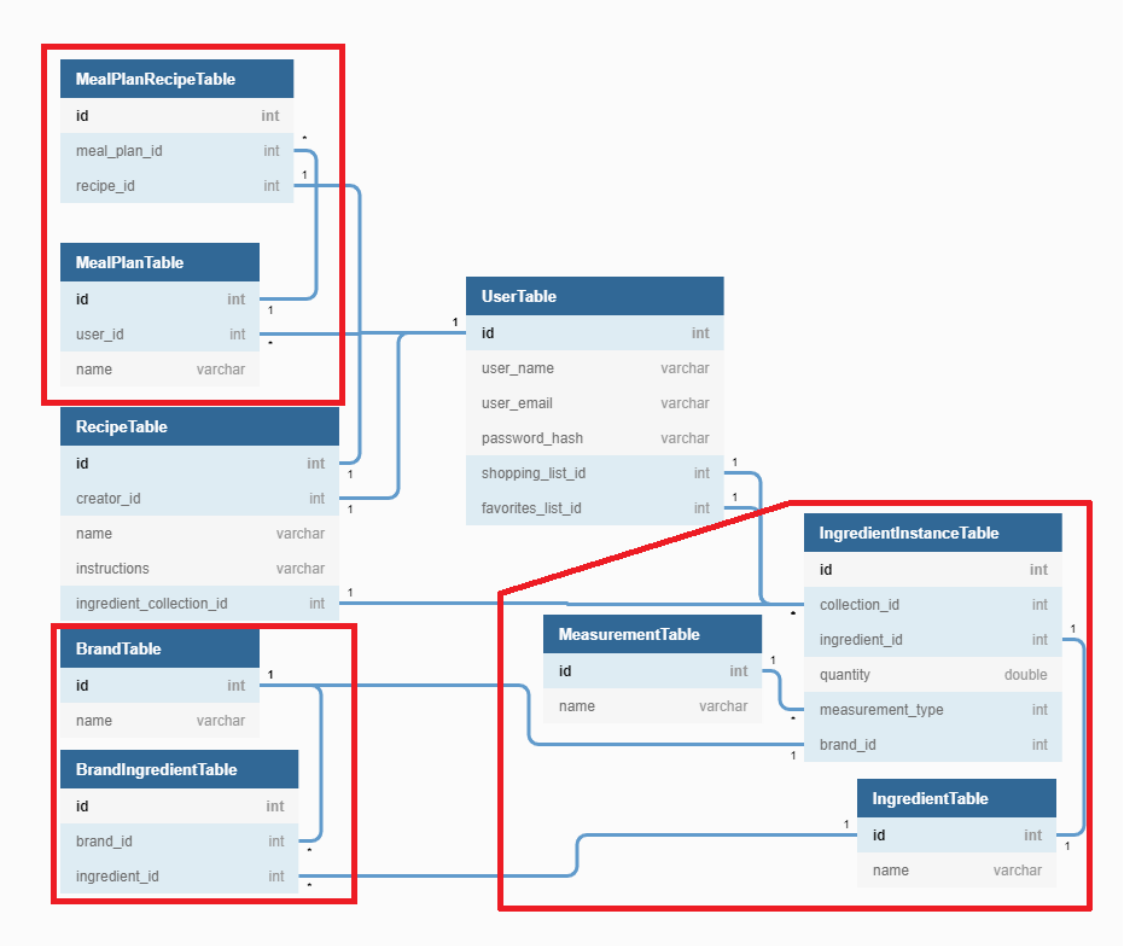


Figure 10: Database Diagram

### 6.1.1 Assumptions

The users email and username and password shall be needed and stored in the database.

### 6.1.2 Responsibilities

Whenever a user is added to the system, their email, username, and password will be added to this table. A global ID will also be used to improve data efficiency and ensure data integrity. Their password is also stored as a hash instead of plain form for better security, but this field may end up being removed if it is proven obsolete by other authentication systems such as AWS authentication. A few other noteworthy fields in this table are are "shopping_list_id" and "favorites_list_id" which point to the current shopping list and favorite items list, which will be discussed in subsection 6.3.

### 6.1.3 Subsystem Interfaces

Table 12: Subsystem interfaces

| ID | Description | Inputs | Outputs |
|---|---|---|---|
| #1 | A new user entry is created | Username, password hash, and email | success/failure |
| #2 | A user entry is updated | A field of "UserTable" is updated for a particular given user ID. | success/failure |
| #3 | A user entry is deleted | A user ID | success/failure |
| #4 | A user is queried | username/email, hashed password | user record |

## 6.2 Meal Plan Subsystem

The Meal Plan Subsystem will be incharge of complying with customer requirements regarding having meal plans and related features.

### 6.2.1 Assumptions

There are no assumptions for this subsystem.

### 6.2.2 Responsibilities

For meal plans, several layers of one to many relationships are needed. Consider the following enumeration to get an idea of those:

1. To allow the user to have multiple meal plans, "MealPlanTable" maps each user ID to zero or more meal plans, each of which have their own name.

2. To allow each meal plan to have multiple recipes, "MealPlanRecipeTable" maps each meal plan to zero or more recipes.

### 6.2.3 Subsystem Interfaces

Table 13: Subsystem interfaces

| ID | Description | Inputs | Outputs |
|---|---|---|---|
| #01 | A new meal plan is created | user ID, meal plan name | meal plan ID |
| #02 | A meal plan is updated | A field of from the meal plan and the meal plan ID | success/failure |
| #03 | A meal plan is deleted | meal plan ID | success/failure |
| #04 | A recipe is added to a meal plan | meal plan ID, recipe ID | success/failure |
| #05 | A recipe is removed to a meal plan | meal plan ID, recipe ID | success/failure |
| #06 | A list of meal plans is queried | A user ID | A list of meal plans |
| #07 | A list of meal recipes is queried | A meal plan ID | A list of recipes |

## 6.3 Ingredient Subsystem

The Ingredients Subystem will be used for storing ingredients used elsewhere in the database.

### 6.3.1 Assumptions

Whenever a shopping, favorites list, or recipe is cleared, the corrosponding collection ID will be discarded so that only ones in use will appear in "IngredientsInstanceTable".

### 6.3.2 Responsibilities

The system needs to be able to know the names of ingredients and the amounts needed for shopping lists and recipes. These two needs are met by "IngredientTable" and "IngredientInstanceTable" respectively. The former mearly holds the name of each ingredient, which can be used to comply with the grocery search requirements mentioned in the SRS. The latter is a bit more complicated, the various fields are broken down in the following list:

- id: Global id for each enty to ensure uniqueness, this may end up being removed if shown unneeded.
- collection_id: Every user shopping list, user favorites list, and recipe will have an associated collection id, which will corrospond to everything on that list. Each of those items for that "collection" will fall in this table. By constructing the database in this way, several requirements are satisfied with a single table.
- quantity: Gives the amount required of the ingredient in the units specified in the next field.
- measurement_type: Has a key corrosponding to the particular measurement type in use.

### 6.3.3 Subsystem Interfaces

Table 14: Subsystem interfaces

| ID | Description | Inputs | Outputs |
|---|---|---|---|
| #01 | An ingredient instance is added | ingredient ID, measurement ID, amount, collection ID | particular ingredient ID |
| #02 | An ingredient instance is updated | particular ingredient ID, other field value | success/failure |
| #03 | An ingredient instance is deleted | particular ingredient ID | success/failure |
| #04 | An unused collection ID is requested | none | the largest collection ID currently in use in the table |

## 6.4 RecipeTable Subsystem

The RecipeTable Subsystem will responsible maintaining lists of all recipes, along with associated ingredient collection IDs.

### 6.4.1 Assumptions

Any recipe can be written as a set of instructions and ingredients.

### 6.4.2 Responsibilities

A table containing each recipe within the application. The first few fields are fairly straightforward, giving a unique ID to each recipe, the ID of the creature of the recipe, the name of the recipe, and instructions on how to make the recipe. The last field, as discussed in subsection 6.3, gives the ID which can be used to query the particular ingredients for the recipe. This last field is to be used in constructing shopping lists.

### 6.4.3 Subsystem Interfaces

Table 15: Subsystem interfaces

| ID | Description | Inputs | Outputs |
|---|---|---|---|
| #01 | A recipe is added | all field values except ID | recipe ID |
| #02 | A recipe is updated | recipe ID, other field value | success/failure |
| #03 | A recipe is deleted | recipe ID | success/failure |

# 7 DATA CONTROLLER SUBSYSTEMS

The data controller layer is responsible for getting data about grocery items from different stores and transferring the data to the shopping manager subsystem which is in the Server/Back end layer. There is only one subsystem in the data controller layer, the API manager. Since many stores do not allow web scraping, the developer team decided to use APIs from stores and gather grocery data.

## 7.1 API MANAGER

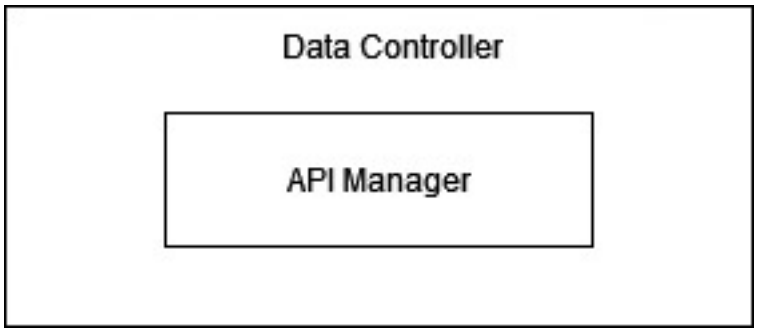


Figure 11: Data controller subsystem diagram

### 7.1.1 ASSUMPTIONS

All of the stores will return grocery descriptions in JSON format.

### 7.1.2 RESPONSIBILITIES

The API manager will take a grocery list and give the items in the list to the stores APIs. If the store API will find the given items then it will return the description of the items else it will return an error message. After the API manager gets the data it is responsible for transferring the data to the store manager subsystem.

### 7.1.3 SUBSYSTEM INTERFACES

Table 16: Subsystem interfaces

| ID | Description | Inputs | Outputs |
|---|---|---|---|
| #1 | The given grocery is found | list of grocery items | grocery item descriptions |
| #2 | The given grocery is not found | list of grocery items | not found message |

## References