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

Detailed Design Specification
CSE 4317: Senior Design II
Spring 2022

UTA STeam
Groco

Patrick Faulkner
Andrew Hands
Hozefa Tankiwala
Karki Kiran
Uyen Do

Revision History

Revision	Date	Author(s)	Description
0.1	1.30.2022	AH	document creation

CONTENTS

List of Figures

List of Tables

1 INTRODUCTION

Groco aims to streamline the way we find recipes, make shopping lists, and find the best locations to buy ingredients. We accomplish this buy having a centralized place for people to add, search, and view recipes. Whenever a given user decides they like a recipe, the recipe can be added to their shopping list by the click of a button. Whenever a user decides to to go shopping, by the click of a button they can be given a list of stores and items to shop at which is optimized by their shopping preferences. Additionally, we add customization in the form of meal plans, which users can create and use as common collections of recipes that they can add to their shopping list when needed.

For additional information, consider looking at the architectural design and system requirement specifications, which can be found at https://tinyurl.com/2p8psxc8 and https://tinyurl.com/2p8snvxf, respectively.

2 SYSTEM OVERVIEW

The applications main interfaces are the front-end and the data collector. A given user will load front-end pages and make requests to read/write/modify information. These requests will go through the back-end to be processed, then proceed to the database. However, in the case where the user decides to map their route, the data collector will be used to search various external websites for various products to aggregate the user's shopping plan. The bulk of the processing in the back-end will be devoted to the routing algorithm which computes the optimal route for the user to take. Similarly, the presentation of this information on the front-end will be one of the more complicated screens. The database will store information about users, user recipes, meal plans, and ingredients/brands. Thus, if the users are just looking for items in the database, these requests will just go from the front-end to the database through the back-end. Finally, the data-collector will search various store websites on demand given requests from the back-end, and more specifically the routing/shop algorithm.

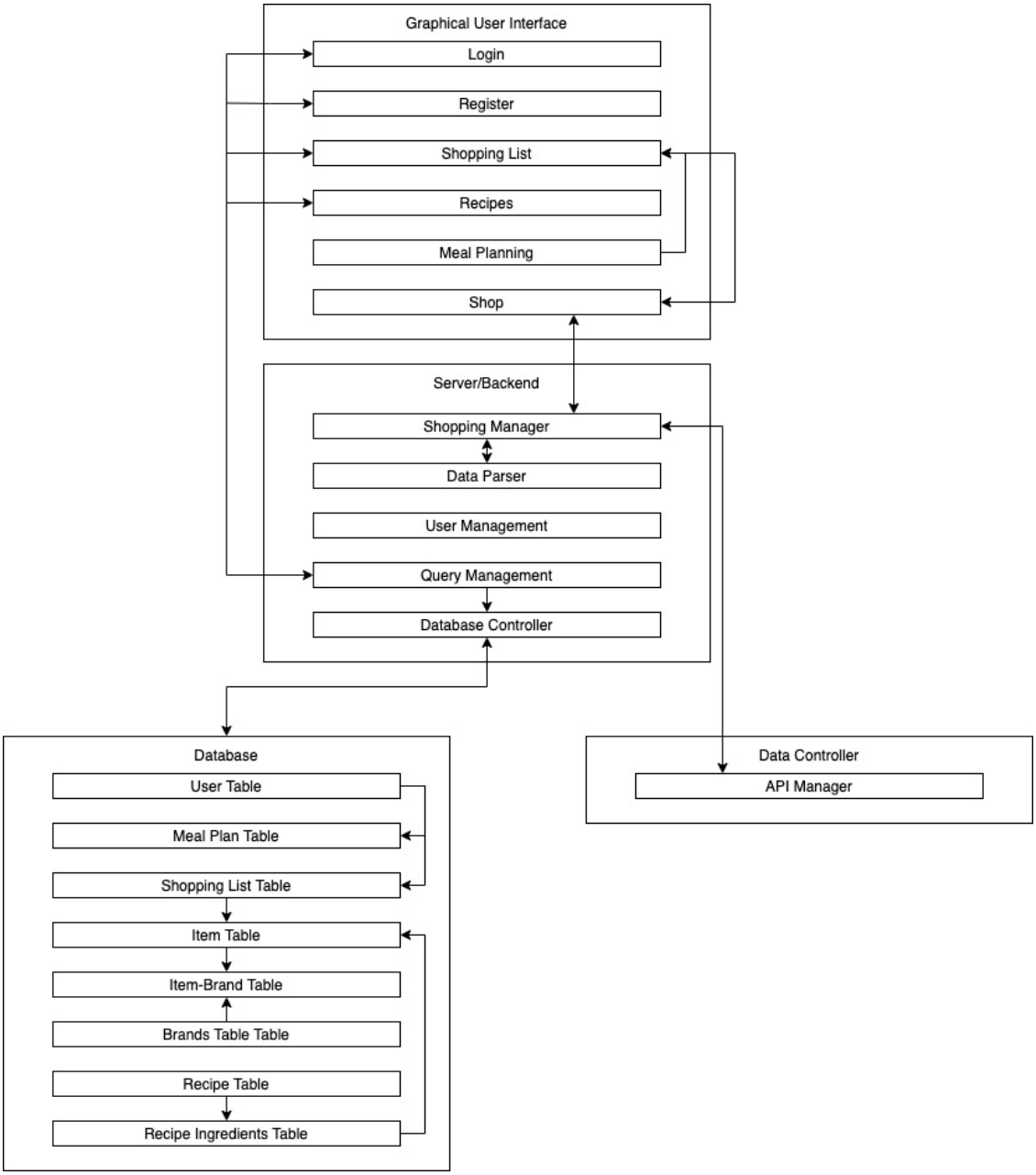

Figure 1: System architecture

3 Front-End Layer Subsystems

3.1 Layer Software Dependencies

The entire front-end will depend on Bootstrap and the React library. The front-end will also depend on a consistent connection to the Query Manager on the Server layer.

3.2 Login Subsystem

The Login Subsystem will provide an interface for users to login into the application. Users will be able to log in with their Google accounts.

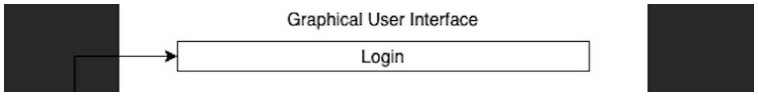

Figure 2: Login Subsystem diagram

3.2.1 Subsystem Software Dependencies

For a user to log in with their Google accounts, the Login Subsystem will depend on the GoogleLogin library.

3.2.2 Subsystem Programming Languages

The Login Subsystem will be developed using, React.js, HTML, and CSS.

3.2.3 Subsystem Data Processing

When the user logs in with their Google account, they will select the Google login button. The Login Subsystem will then call on the GoogleLogin API which users will log in with. If the login is successful, the Login Subsystem will redirect the user to the Home Page. If the login fails, the user will be given an error message and prompted to re-enter their credentials.

3.3 Shopping List Subsystem

The Shopping List Subsystem will provide an interface for users to view all the items on their current shopping list. It will also allow users to add and remove items.

Figure 3: Shopping List Subsystem diagram

3.3.1 Subsystem Software Dependencies

The Shopping List Subsystem has no additional dependencies.

3.3.2 Subsystem Programming Languages

The Shopping List Subsystem will be developed using, React.js, HTML, and CSS.

3.3.3 Subsystem Data Processing

The Shopping List Subsystem will retrieve the current user's shopping list from the Query Manager and display all relevant information.

3.4 Recipes Subsystem

The Recipes Subsystem will provide an interface for users to view and shop for all recipes in the system's database. It will also allow users to create and edit their recipes.

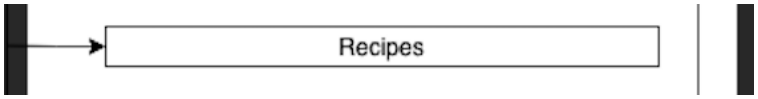

Figure 4: Recipes Subsystem diagram

3.4.1 Subsystem Software Dependencies

The Recipes Subsystem has no additional dependencies.

3.4.2 Subsystem Programming Languages

The Recipes Subsystem will be developed using, React.js, HTML, and CSS.

3.4.3 Subsystem Data Processing

The Recipes Subsystem will display all recipes retrieved by the Query Manager.

3.5 Meal Plan Subsystem

The Meal Plan Subsystem will provide an interface for users to view and edit their Meal Plans.

Figure 5: Meal Plan Subsystem diagram

3.5.1 Subsystem Software Dependencies

The Meal Plan Subsystem has no additional dependencies.

3.5.2 Subsystem Programming Languages

The Meal Plan Subsystem will be developed using, React.js, HTML, and CSS.

3.5.3 Subsystem Data Processing

The Meal Plan Subsystem will display the current user's Meal Plans that are retrieved by the Query Manager.

3.6 Shop Subsystem

The Shop Subsystem will provide an interface for users to view the results of the shopping trip being planned.

Figure 6: Shop Subsystem diagram

3.6.1 Subsystem Software Dependencies

The Shop Subsystem depends on a consistent connection to the Shop Manager on the Server layer.

3.6.2 Subsystem Programming Languages

The Shop Subsystem will be developed using, React.js, HTML, and CSS.

3.6.3 Subsystem Data Processing

The Shop Subsystem will display all information that is calculated by the Shopping Manager.

4 Back-End Layer Subsystems

4.1 Layer Operating System

Back-end will work on Google Cloud Platform (GCP).

4.2 Layer Software Dependencies

The back-end will be hosted using JavaScript's NodeJS Framework.

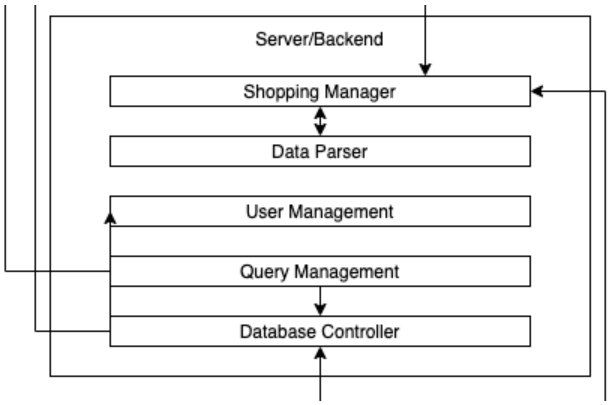

Figure 7: Back-End layer subsystem

4.3 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.

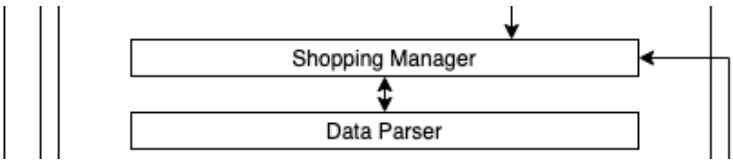

Figure 8: Shopping manager Subsystem

4.3.1 Subsystem Software Dependencies

The subsystem will use the axios library to communicate with the Data Controller Layer. It will also use Google Maps library to display and calculate the optimum route to the user.

4.3.2 Subsystem Programming Languages

JavaScript (NodeJS)

4.3.3 Subsystem Data Structures

It will provide the front-end with the Prices and Availability of grocery items. Also the data required to display the optimum route on the front end.

4.3.4 Subsystem Data Processing

The shopping optimization algorithm will seek to find the best route given the users criteria. To start with, define $\mathbf{S} = \{S_1, S_2, ..., S_n\}$ to be an ordered set of n stores, to be visited, from S_1 to S_n in order

and define S_0 to be the starting location of the user. Next, define w_0 and w_1 to be non-negative weights defined by the user to give the importance of distance and item price, respectively. Define $d(S_{i-1}, S_i)$ to be the distance from S_{i-1} to S_i for all $i = 1, 2, ..., n$. Define (S_i, h) to be the cost of item h at store S_i. Finally, define $\mathbf{H}(S_i)$ to be the set of items the user is planning to buy at store S_i. Thus, we are able to define our cost function as:

$$COST(\mathbf{S}, \mathbf{H}) := w_0 \sum_{i=1}^{n} d(S_{i-1}, S_i) + w_1 \sum_{i=1}^{n} \sum_{h \in \mathbf{H}(S_i)} \lambda(S_i, h) \tag{1}$$

Hence, if **A** represents the shopping list of the user, then it should follow that

$$\mathbf{A} =: \bigcup_{s \in \mathbf{S}} \mathbf{H}(s) \tag{2}$$

provided all of the items on the shopping list are able to be found. Thus, we can see that if **S** is known, then **H** follows by choosing the cheapest store for each item in **A**. It also follows that $COST$ is monotonic with respect to the number of items in **A**. Therefore, A-Star can be applied and shall be applied to this measure time, and should be of quadratic time complexity.

4.4 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.

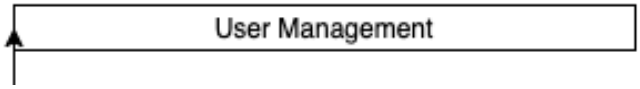

Figure 9: User Management Subsystem

4.4.1 Subsystem Software Dependencies

This subsystem will Handle GoogleLogin authentication to make sure a user is logged in and connects their Google ID to the ID stored in the database. This subsystem will also use Axios library.

4.4.2 Subsystem Programming Languages

JavaScript (NodeJS) and SQL.

4.4.3 Subsystem Data Structures

This subsystem will create a data structure for the user's profile using the Google ID and send it to the front-end.

4.4.4 Subsystem Data Processing

It will take in the Google ID for the user, search for that stored Google ID in the database and pull all the data associated to that user.

4.5 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.

4.5.1 Subsystem Software Dependencies

It will use the npm pg library to communicate with the PostgreSQL Database stored in Google Cloud Platform.

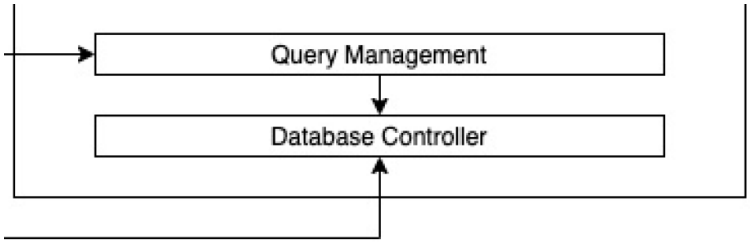

Figure 10: Query Management Subsystem

4.5.2 Subsystem Programming Languages

JavaScript (NodeJS) and SQL.

4.5.3 Subsystem Data Structures

This subsystem will be responsible for creating and managing all the queries to the database.

4.5.4 Subsystem Data Processing

Depending on the request, this layer will create a database query and send it to the Database Controller Layer.

4.6 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.

4.6.1 Subsystem Software Dependencies

It will use the npm pg library to communicate with the PostgreSQL Database stored in Google Cloud Platform using Axios.

4.6.2 Subsystem Programming Languages

JavaScript (NodeJS) and SQL.

4.6.3 Subsystem Data Structures

This subsystem will take queries from the Query Management Layer and send those queries to the database.

4.6.4 Subsystem Data Processing

This subsystem will send the queries to the database and get the result and send it back to the Query Management Layer.

5 Database Layer Subsystem

Groco will use a relational database to keep back-end information uniform and well structured. There will be five different table subsystems. They are User Table Subsystem, Meal Plan Table Subsystem, Brand Table Subsystem, Ingredients Table Subsystem and Recipe Table Subsystem.

5.1 Layer Operating System

The Database Layer will run on Google Cloud Platform.

5.2 Layer Software Dependencies

The Database Layer will depend upon Postgres for the creation of tables and execution of queries. Also, the Database Layer will depend upon Google Cloud Platform for hosting the database on the cloud.

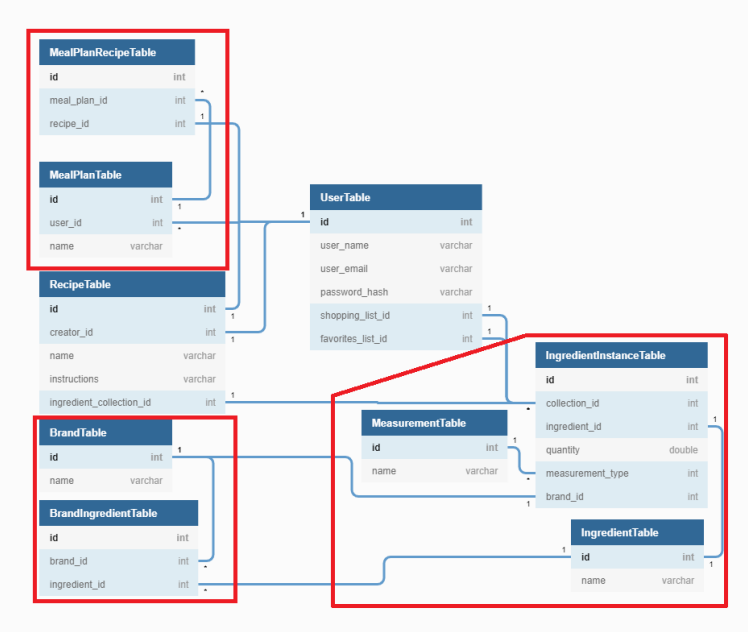

Figure 11: Database Diagram

5.3 User Table Subsystem

The User Table Subsystem will be used for storing identifiable information about each user and holding information to help the user. It has only one table called User table.

5.3.1 Subsystem Software Dependencies

User Table Subsystem does not have additional dependencies.

5.3.2 Subsystem Programming Languages

The User Table Subsystem will be created using SQL and queried using SQL.

5.3.3 Subsystem Data Processing

Whenever a user is added to the system, their username and ID will be added to this table. A global ID will also be used to improve data efficiency and ensure data integrity.

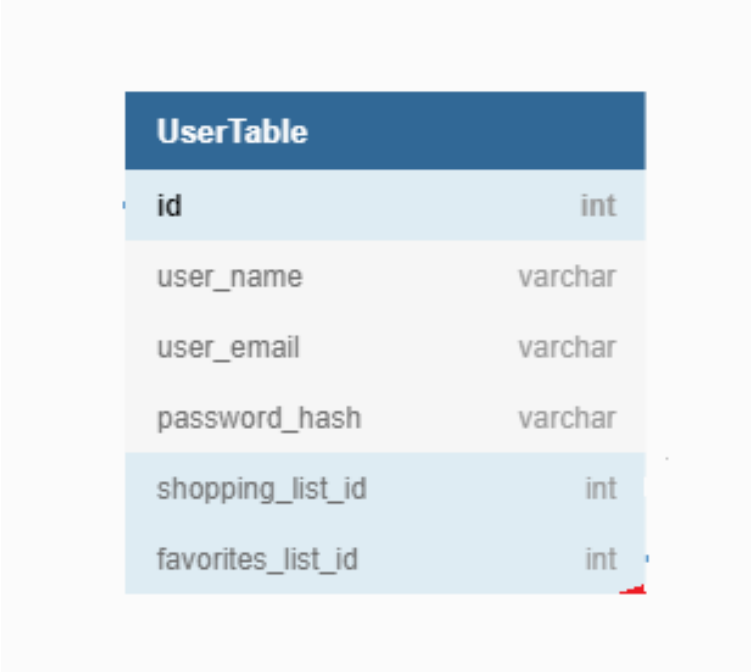

Figure 12: User Table subsystem

5.4 Meal Plan Table Subsystem

The Meal Plan Subsystem will be in charge of complying with customer requirements regarding having meal plans and related features. It has two tables called meal plan recipe table and meal plan table.

5.4.1 Subsystem Software Dependencies

Meal Plan Subsystem does not have additional dependencies.

5.4.2 Subsystem Programming Languages

The Meal Plan Subsystem will be created using SQL and queried using SQL.

5.4.3 Subsystem Data Processing

Meal plan recipe table will has mealplanid as a foreign key and it will use that foreign key when a join operation is required between meal plan recipe table and meal plan table.

5.5 Ingredient Table Subsystem

The Ingredients Subsystem will be used for storing ingredients required in the construction of a meal plan. This subsystem has three tables and they are called measurement table, ingredient table, and ingredient instance table.

5.5.1 Subsystem Software Dependencies

Ingredient Table Subsystem does not have any additional dependencies.

5.5.2 Subsystem Programming Languages

The Ingredient Table Subsystem will be created using SQL and queried using SQL.

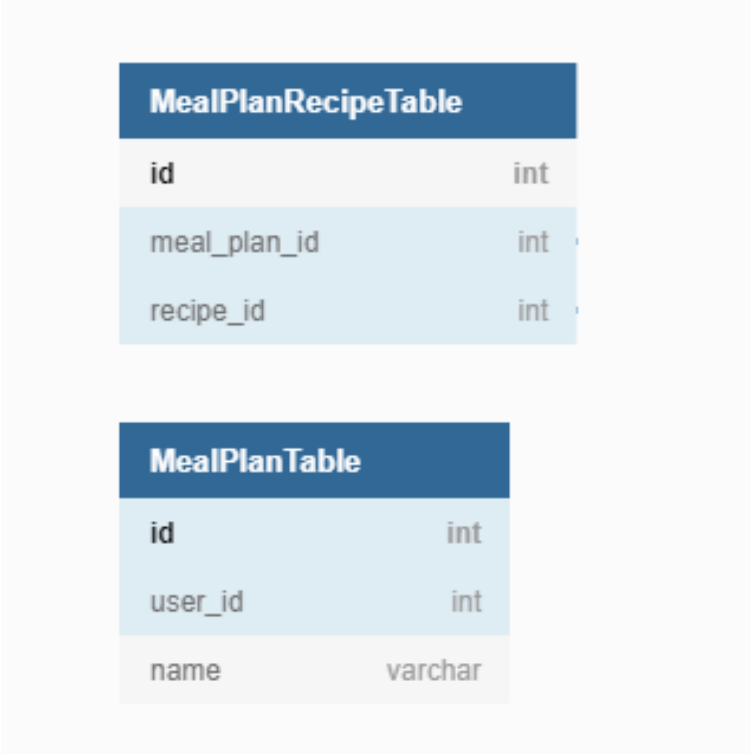

Figure 13: Meal plan table subsystem diagram

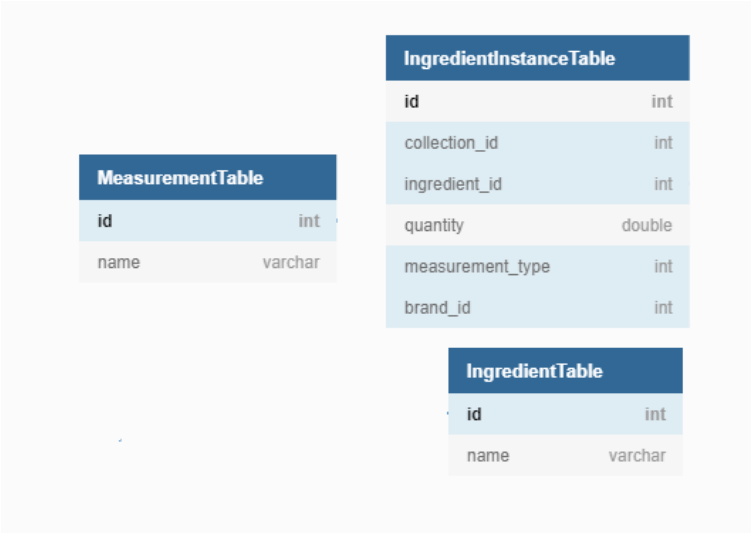

Figure 14: Ingredient Table subsystem

5.5.3 Subsystem Data Processing

The ingredients instance table will use measurement type and ingredient id foreign keys when the join operation is required between measurement table, ingredient table and ingredient instance table.

5.6 Recipe Table Subsystem

The Recipe Table Subsystem will be responsible for maintaining lists of all recipes, along with associated ingredient collection IDs. It has only one table called recipe table.

Figure 15: Recipe Table subsystem diagram

5.6.1 Subsystem Software Dependencies

Recipe Table does not have any additional dependencies.

5.6.2 Subsystem Programming Languages

The Recipe Table Subsystem will be created using SQL and queried using SQL.

5.7 Brand Table Subsystem

The Brand Table Subsystem will be responsible for maintaining lists of all brands, along with associated ingredient of the brand. This subsystem has two tables: Brand Table and Brand Ingredient Table.

5.7.1 Subsystem Software Dependencies

Brand Table Subsystem does not have any additional dependencies.

5.7.2 Subsystem Programming Languages

The Brand Table Subsystem will be created using SQL and queried using SQL.

5.7.3 Subsystem Data Processing

Brand Ingredient Table has brandId as foreing key and it will be used when a join operation is required between brand table and brand ingredient table. It also has ingredient id as a foreign key which will be used when a join operation is required between brand ingredient table and ingredient table.

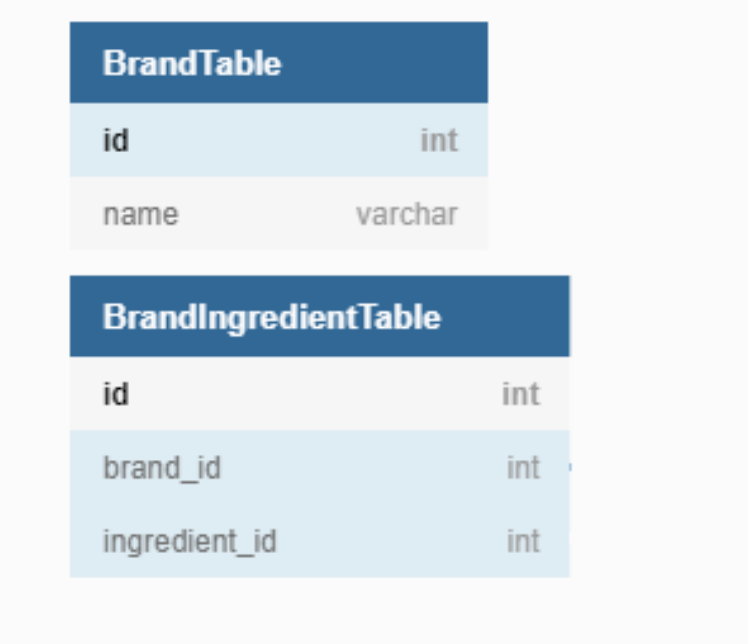

Figure 16: Recipe Table subsystem diagram

6 Data Controller Layer Subsystem

The Data Controller Layer is responsible for getting data about grocery items including data about prices, stock, and store location from different stores. It then transfers the data to the Shopping Manager Subsystem in the Server/Back-end layer. There is one subsystem in the data controller layer, the API manager. Since many stores do not allow web scraping, the developer team decided to use public APIs from stores to gather grocery data.

6.1 Layer Software Dependencies

The Data Controller Layer depends on API's that will be managed by other companies.

6.2 API Manager

The API manager is a core component of the data collector. The API manager provides one central point for data collection through web APIs across different stores.

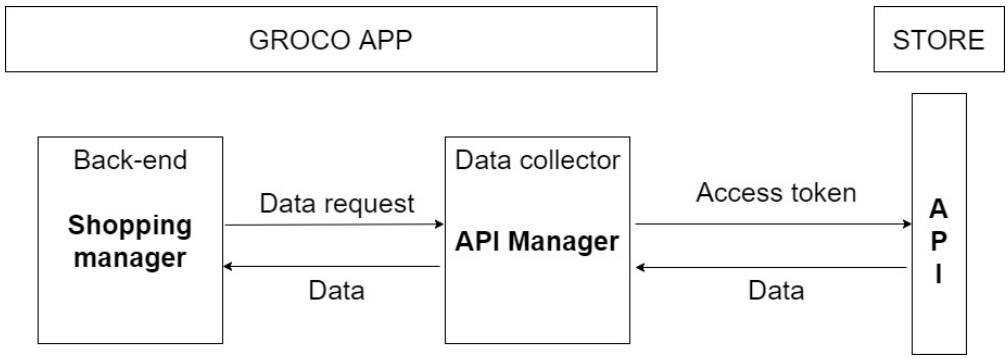

Figure 17: Data Controller Subsystem

6.2.1 Subsystem Software Dependencies

The API Manager has no additonal dependencies.

6.2.2 Subsystem Programming Languages

The API Manager is written in JavaScript.

6.2.3 Subsystem Data Processing

The API Manager is responsible for storing access tokens to get data from different APIs. The retrieved data will be sent to the Shopping Manager Subsystem in the Back-end layer to fulfil the request. The API Manager is also responsible for refreshing the access tokens when they are expired.

7 APPENDIX A

Include any additional documents (CAD design, circuit schematics, etc) as an appendix as necessary.

REFERENCES