

# CSCE 608 Database Systems

## Project 1

Cheng-Yun Cheng

### **Application Description**

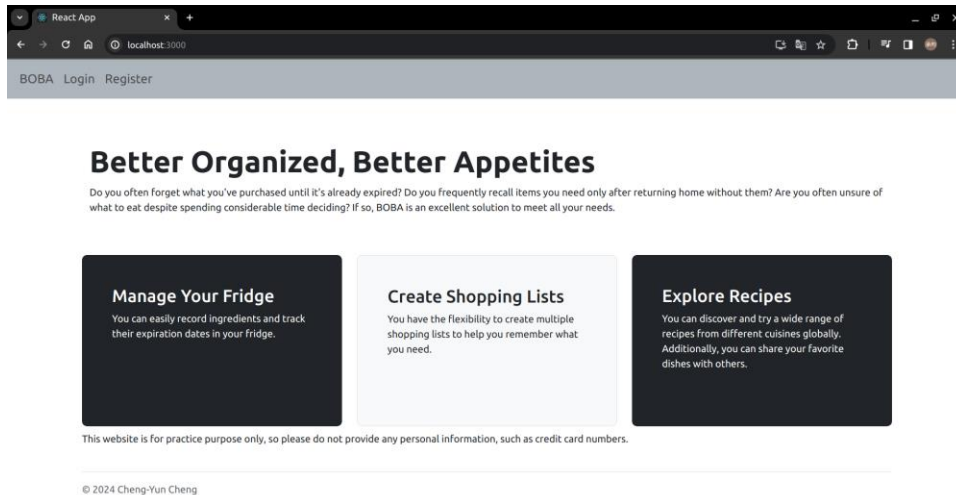
Do you often forget what you've purchased until it's already expired? Do you frequently recall items you need only after returning home without them? Are you often unsure of what to eat despite spending considerable time deciding? If so, BOBA, "Better Organized, Better Appetites," is an excellent solution to meet all your needs. BOBA provides three primary features: managing your food inventory, creating shopping lists, and discovering recipes.

Upon logging in, users are directed to the "My Fridge" section where they can input newly purchased ingredients along with their quantities. Users can also note the purchase date and set expiration dates, with BOBA suggesting recommended expiration periods for each ingredient. For instance, apples might last around ten days in the fridge, while chicken breast may only last three days. The fridge organizes items based on their expiration dates for reminder. As items are consumed, users can freely update their quantities or remove them from the fridge.

The next section is dedicated to shopping lists. Many people prefer buying certain items from different stores, such as vegetables and fruits from HEB, steaks and salmon from Costco, and herbs from Trader Joe's. BOBA enables you to create numerous shopping lists with unique names for easy management. You can include notes to remember specific details and freely add or remove items within the lists.

The final feature of BOBA is the recipe exploration section. Here, users can browse recipes contributed by other users. Upon entering the "Explore Recipe" section, the page displays a variety of recipes based on the number of likes they have received. Users can also search for recipes using specific ingredients or categories. For instance, if you only have chicken breast and onions and wish to try a new dish, simply enter these ingredients into the search bar to find recipes containing them. Furthermore, users can review all recipes by a particular user if they find their recipes appealing. Additionally, users can create their recipes in the "My Recipes" section. Here, they can write and save recipes for future reference. The liked recipes are also conveniently displayed in this section for easy access.

BOBA offers a comprehensive solution for individuals who struggle with managing their food inventory, creating shopping lists, and finding new recipes. With its user-friendly interface and innovative features, BOBA simplifies all the process. Whether you're a busy professional, a home cook looking for inspiration, or someone who wants to reduce food waste, BOBA, "Better Organized, Better Appetites," makes it a valuable tool for anyone seeking to optimize their kitchen management and meal planning.



## E/R Diagram

To support the three primary features discussed earlier—managing food inventory, creating shopping lists, and discovering recipes—we have designed five entities.

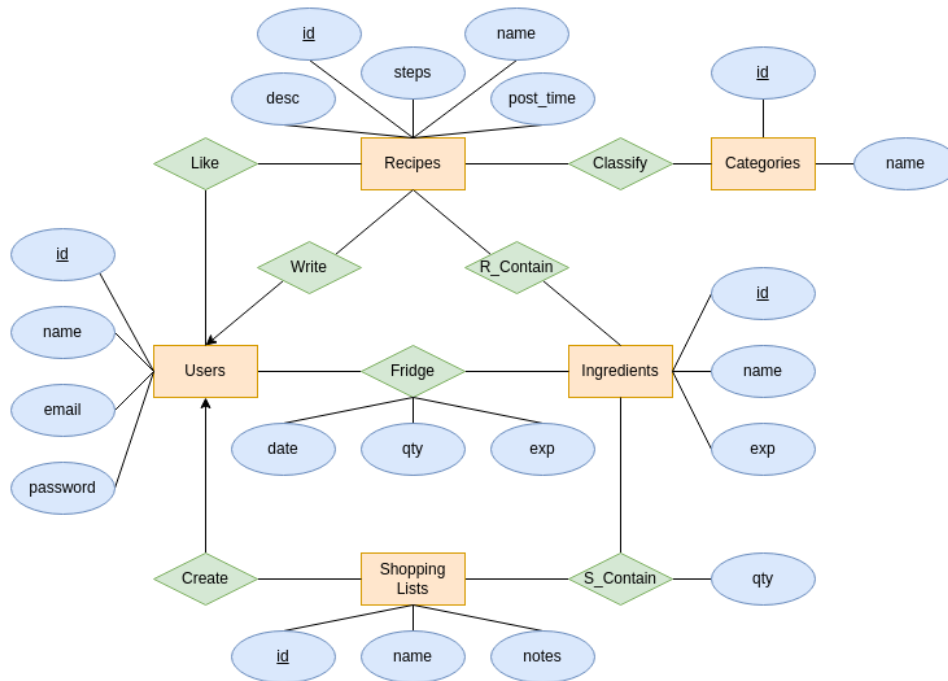
1. The user entity comprises four attributes: id, email, name, and password. It serves to store users' information, with email and password used for login purposes, and id acting as the primary key to represent a user in entity relationships.
2. The ingredient entity includes three attributes: id, name, and exp. This entity is designed to record the recommended expiration period for each ingredient, with id serving as the primary key to represent an ingredient in entity relationships.
3. The shopping list entity consists of three attributes: id, name, and notes. It is utilized to store information about shopping lists, with id functioning as the primary key to represent a shopping list in entity relationships.
4. The recipe entity encompasses five attributes: id, name, description, steps, and post time. It is structured to store information about recipes, with id serving as the primary key to represent a recipe in entity relationships.
5. The category entity contains two attributes: id and name. This entity is used to store the categories of recipes, with id acting as the primary key to represent a category in entity relationships.

There are seven relationships between these entities.

1. The “fridge” is a relationship between the user entity and the ingredient entity, signifying that a user stores an ingredient in their fridge. This relationship encompasses attributes such as quantity, date, and expiration date to indicate the status of an ingredient. It is classified as a many-to-many relationship, as a user can store multiple ingredients in their fridge, and an ingredient can be stored by multiple users.
2. The “create” is a relationship between the user entity and the shopping list entity, representing that a user creates a shopping list. This relationship is categorized as a many-to-one relationship, as a user can create multiple shopping lists, but each shopping list belongs to only one user.

3. The “shopping list contain” is a relationship between the shopping list entity and the ingredient entity, indicating that a shopping list contains an ingredient. This relationship includes the quantity attribute and is identified as a many-to-many relationship, as a shopping list can contain multiple ingredients, and an ingredient can be included in multiple shopping lists.
4. The “write” is a relationship between the user entity and the recipe entity, signifying that a user writes a recipe. This relationship is categorized as a many-to-one relationship, as a user can write multiple recipes, but each recipe belongs to only one user.
5. The “recipe contain” is a relationship between the recipe entity and the ingredient entity, indicating that a recipe contains an ingredient. This relationship is classified as a many-to-many relationship, as a recipe can contain multiple ingredients, and an ingredient can be added to multiple recipes.
6. The “like” is a relationship between the user entity and the recipe entity, representing that a user likes a recipe. This relationship is categorized as a many-to-many relationship, as a user can like multiple recipes, and a recipe can be liked by multiple users.
7. The “classify” is a relationship between the recipe entity and the category entity, indicating that a recipe belongs to a category. This relationship is classified as a many-to-many relationship, as a recipe can have multiple categories, and a category can include multiple recipes.

The E/R diagram of BOBA is shown below.



1

## Database Schema and Normalization

The relations translated from the E/R diagram and BCNF examination are shown below.

1. Users(id, email, name, password)

Nontrivial FD: { $id \rightarrow email$ ,  $id \rightarrow name$ ,  $id \rightarrow password$ ,  $email \rightarrow id$ ,  $email \rightarrow name$ ,  $email \rightarrow password$ }

The Users table is in BCNF because both the id and email attributes serve as keys. However, only one attribute, either id or email, is necessary to determine the other attributes in the table. The reason for using both attributes is that email uniquely identifies a user during login, but it is typically too large to efficiently represent a user in other relationships. Hence, email is used for login identification, while the id is used to identify the user in other relationships.

The Users schema is shown below, where the id attribute serves as the primary key, and the email attribute is set as a unique key.

Field	Type	Null	Key	Default	Extra
id	int unsigned	NO	PRI	NULL	auto_increment
email	varchar(256)	NO	UNI	NULL	
name	varchar(256)	NO		NULL	
password	varchar(512)	NO		NULL	

2. Ingredients(id, name, exp)

Nontrivial FD: { $id \rightarrow name$ ,  $id \rightarrow exp$ ,  $name \rightarrow id$ ,  $name \rightarrow exp$ }

The Ingredients table is in BCNF as both the id and name attributes serve as keys. To address the potential length issue of the name attribute when representing ingredients in other relationships, each ingredient is assigned a unique id. This id is then used to identify the ingredient in various relationships.

The Ingredients schema is shown below, where the id attribute serves as the primary key, and the name attribute is set as a unique key.

Field	Type	Null	Key	Default	Extra
id	int unsigned	NO	PRI	NULL	auto_increment
name	varchar(256)	NO	UNI	NULL	
exp	int unsigned	NO		NULL	

3. Shopping\_Lists(id, name, notes) and Create(user.id, list.id)

Because “Create” is a many-to-one relationship, it can be combined with the Shopping Lists table. Then, the Shopping Lists table would be:

Shopping\_Lists(id, user.id, name, notes)

Nontrivial FD: { $id \rightarrow user.id$ ,  $id \rightarrow name$ ,  $id \rightarrow notes$ }

The key in the table is the id attribute, so the Shopping Lists table is in BCNF.

The Shopping\_Lists schema is shown below, where the id attribute serves as the primary key. Additionally, the user attribute is linked as a foreign key to the id attribute in the Users table.

Field	Type	Null	Key	Default	Extra
id	int unsigned	NO	PRI	NULL	auto_increment
user	int unsigned	NO	MUL	NULL	
name	varchar(256)	YES		NULL	
notes	varchar(4096)	YES		NULL	

4. Recipes(id, name, desc, steps, post\_time) and Write(user.id, recipe.id)

Because “Write” is a many-to-one relationship, it can be combined with the Recipes table. Then, the Recipes table would be:

Recipes(id, user.id, name, desc, steps, post\_time)

Nontrivial FD: {id → user.id, id → name, id → desc, id → steps, id → post\_time}

The key in the table is the id attribute, so the Recipes table is in BCNF.

The Recipes schema is shown below, where the id attribute serves as the primary key. Additionally, the user attribute is linked as a foreign key to the id attribute in the Users table.

Field	Type	Null	Key	Default	Extra
id	int unsigned	NO	PRI	NULL	auto_increment
user	int unsigned	NO	MUL	NULL	
name	varchar(256)	YES		NULL	
desc	varchar(4096)	YES		NULL	
steps	varchar(4096)	YES		NULL	
post_time	date	NO		NULL	

5. Categories(id, name)

Nontrivial FD: {id → name, name → id}

The Categories table is in BCNF because both the id and name attributes serve as keys. The reason for using the id attribute is that the name attribute may not be appropriate for identifying the category in other relationships.

The Categories schema is shown below, where the id attribute serves as the primary key, and the name attribute is set as a unique key.

Field	Type	Null	Key	Default	Extra
id	int unsigned	NO	PRI	NULL	auto_increment
name	varchar(256)	NO	UNI	NULL	

6. Fridges(user.id, ingr.id, qty, date, exp)

Nontrivial FD:  $\emptyset$

There is no nontrivial functional dependency in the Fridge table, so it is in BCNF.

However, it is difficult to manipulate the data, especially when updating. Therefore, I added the id attribute. The resulting table is:

Fridge(id, user.id, ingr.id, qty, date, exp)

Nontrivial FD:  $\{id \rightarrow user.id, id \rightarrow ingr.id, id \rightarrow qty, id \rightarrow date, id \rightarrow exp\}$

Now, the key of the new table is the id attribute, so it is still in BCNF.

The Fridges schema is shown below, where the id attribute serves as the primary key. Additionally, the user attribute is linked as a foreign key to the id attribute in the Users table, and the ingr attribute is linked as a foreign key to the id attribute in the Ingredients table.

Field	Type	Null	Key	Default	Extra
id	int unsigned	NO	PRI	NULL	auto_increment
user	int unsigned	NO	MUL	NULL	
ingr	int unsigned	NO	MUL	NULL	
qty	int unsigned	NO		NULL	
date	date	NO		NULL	
exp_date	date	NO		NULL	

#### 7. S\_Contain(list.id, ingr.id, qty)

Nontrivial FD:  $\emptyset$

There is no nontrivial functional dependency in the S\_Contain table, so it is in BCNF.

However, it is difficult to manipulate the data, especially when updating. Therefore, I added the id attribute. The resulting table is:

S\_Contain(id, list.id, ingr.id, qty)

Nontrivial FD:  $\{id \rightarrow list.id, id \rightarrow ingr.id, id \rightarrow qty\}$

Now, the key of the new table is the id attribute, so it is still in BCNF.

The S\_Contain schema is shown below, where the id attribute serves as the primary key. Additionally, the list attribute is linked as a foreign key to the id attribute in the Shopping Lists table, and the ingr attribute is linked as a foreign key to the id attribute in the Ingredients table.

Field	Type	Null	Key	Default	Extra
id	int unsigned	NO	PRI	NULL	auto_increment
list	int unsigned	NO	MUL	NULL	
ingr	int unsigned	NO	MUL	NULL	
qty	int unsigned	NO		NULL	

#### 8. R\_Contain(recipe.id, ingr.id)

Nontrivial FD:  $\emptyset$

There is no nontrivial functional dependency in the R\_Cotain table, so it is in BCNF.

The R\_Contain schema is shown below, where the recipe and ingr attribute serve as the primary key. Additionally, the recipe attribute is linked as a foreign key to the id attribute in the Recipes table, and the ingr attribute is linked as a foreign key to the id attribute in the Ingredients table.

Field	Type	Null	Key	Default	Extra
recipe	int unsigned	NO	PRI	NULL	
ingr	int unsigned	NO	PRI	NULL	

#### 9. Classify(recipe.id, cat.id)

Nontrivial FD:  $\emptyset$

There is no nontrivial functional dependency in the Classify table, so it is in BCNF.

The Classify schema is shown below, where the recipe and cat attribute serve as the primary key. Additionally, the recipe attribute is linked as a foreign key to the id attribute in the Recipes table, and the cat attribute is linked as a foreign key to the id attribute in the Categories table.

Field	Type	Null	Key	Default	Extra
recipe	int unsigned	NO	PRI	NULL	
cat	int unsigned	NO	PRI	NULL	

#### 10. Like(user.id, recipe.id)

Nontrivial FD:  $\emptyset$

There is no nontrivial functional dependency in the Like table, so it is in BCNF.

The Like schema is shown below, where the user and recipe attribute serve as the primary key. Additionally, the user attribute is linked as a foreign key to the id attribute in the Users table, and the recipe attribute is linked as a foreign key to the id attribute in the Recipes table.

Field	Type	Null	Key	Default	Extra
user	int unsigned	NO	PRI	NULL	
recipe	int unsigned	NO	PRI	NULL	

## Creating Tables/Data

The majority of data in the BOBA database is sourced from the [Food.com Recipe and Interaction dataset](#), supplemented by some randomly generated entries. The recipes data from the Food.com dataset includes attributes such as name, submitted time, tags, steps, description, and ingredients.

In the Users table, I created 200 user entries with unique values for name, email, and password.

For the Recipes table, I randomly selected 500 recipes from the Food.com Recipe and Interaction dataset and assigned each recipe a unique id and a randomly chosen writer. Additionally, when a new ingredient was introduced in a recipe, it was added to the Ingredients table, and the corresponding ingredient id was recorded in the R\_Contain table along with the recipe id. Similarly, for the Categories data, I followed a similar process. Besides, recommended expiration periods for each ingredient are randomly generated.

In the Like table, I randomly assigned a number to each user to represent the number of recipes they liked and randomly selected that many recipes for each user. It's important to note that only recipes not authored by the user were added to their like list.

As a result, the amount of data in each table is shown below.

- Users: 200
- Ingredients: 1281
- Recipes: 500
- Categories: 351
- R\_Contain: 4343
- Classify: 9106
- Like: 1966

## **User Interface and Functions**

As previously mentioned, BOBA's primary features include managing food inventory, creating shopping lists, and discovering recipes. I will describe the specific functions of these features and how they work.

- **Register and Login**

1. Register: For registration, users input their email, name, and password, and this information is stored in the Users table using an INSERT INTO statement.

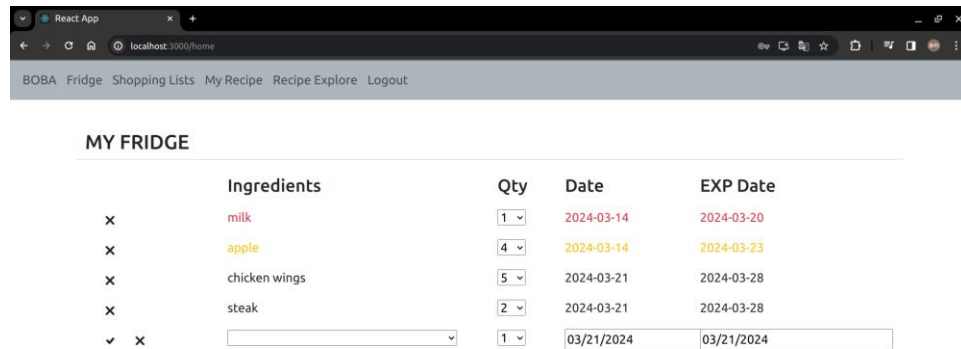


2. Login: To log in, users enter their email and password. A SELECT statement with a condition based on the email is used to retrieve the corresponding password from the database. The retrieved password is then compared with the entered password. If they match, the user is redirected to their home page.

- **Managing Fridge**

1. Examining ingredients: Upon entering the "My Fridge" section, all ingredients stored in the fridge are displayed, sorted by their expiration dates. Expired ingredients are highlighted in red, while those nearing expiration are highlighted in yellow. This functionality is achieved using a SELECT statement with a condition based on the user id to retrieve data from the Fridges table. Additionally, a NATURAL JOIN statement is employed to fetch the names of ingredients from the Ingredients table.
2. Adding new ingredients: When a user purchases a new ingredient, they can select its name, quantity, purchase date, and expiration date before pressing the OK button. Subsequently, an INSERT INTO statement is used to store this data in the Fridge table.

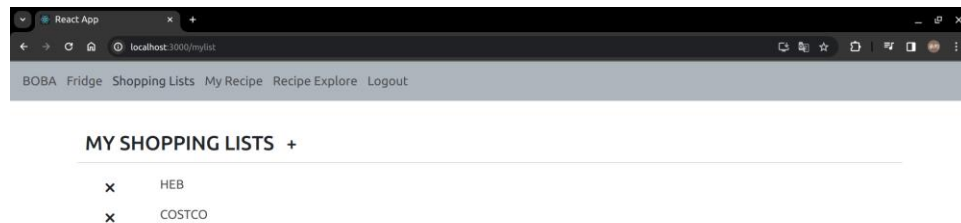
- Updating ingredient quantities: As users consume ingredients, they can update the quantity by selecting a new value. An UPDATE statement with a condition based on the fridge id is then utilized to update the data in the Fridge table.
- Deleting ingredients: Users can remove ingredients from the fridge by clicking the remove button when they no longer have the ingredient. A DELETE statement with a condition based on the fridge id is employed to delete the corresponding data from the Fridge table.



	Ingredients	Qty	Date	EXP Date
X	milk	1	2024-03-14	2024-03-20
X	apple	4	2024-03-14	2024-03-23
X	chicken wings	5	2024-03-21	2024-03-28
X	steak	2	2024-03-21	2024-03-28
✓ X	<input type="text"/>	1	<input type="text" value="03/21/2024"/>	<input type="text" value="03/21/2024"/>

- Creating Shopping List**

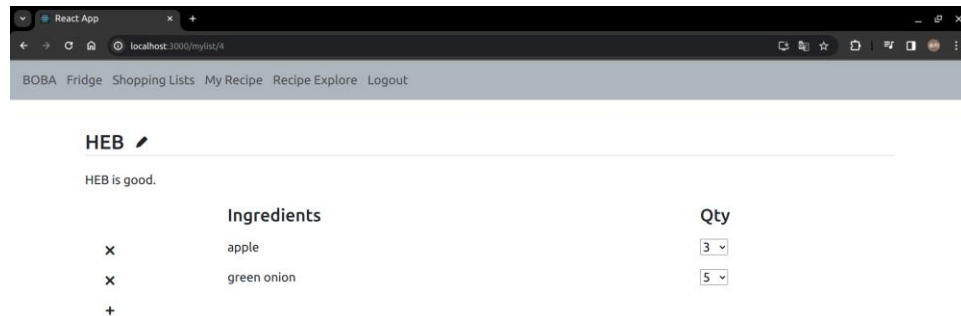
- Reviewing all shopping lists: Upon accessing the "My Shopping List" section, users will see the names of all their shopping lists. This is achieved using a SELECT statement with a condition based on the user id to retrieve data from the Shopping\_Lists table.



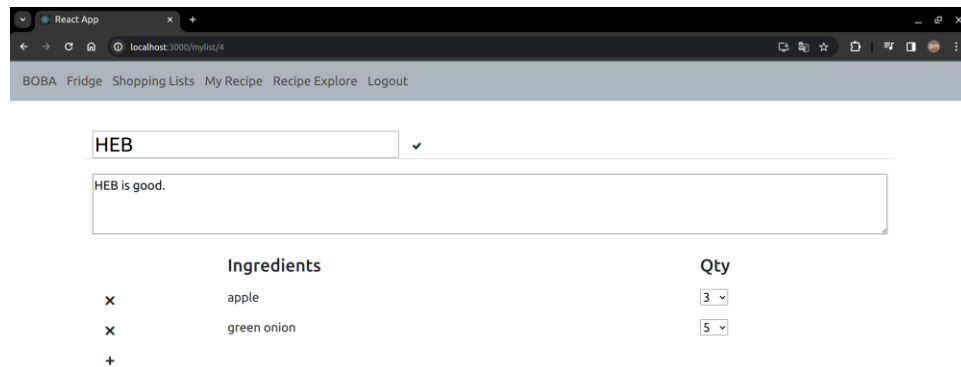
	Shopping Lists
X	HEB
X	COSTCO

- Examining a specific shopping list: Clicking on a particular shopping list redirects the user to the shopping list page, where notes and ingredients from that list are displayed. Data retrieval is accomplished through a SELECT statement with a condition based on the list id to fetch information from the Shopping\_List table. Another SELECT statement, also based on the list

id, retrieves ingredient data from the S\_Contain table, with the NATURAL JOIN statement utilized to obtain ingredient names from the Ingredients table.



3. Creating new shopping lists: Clicking the add button generates a new shopping list in the Shopping\_Lists table using an INSERT INTO statement, and the user is redirected to the shopping list page.
4. Updating shopping lists: Clicking the edit button allows users to modify the name and notes of a shopping list. Upon confirming the changes by clicking OK, an UPDATE statement with a condition based on the list id is executed to update the data in the Shopping\_Lists table.



5. Deleting shopping lists: Clicking the delete button removes the corresponding shopping list using a DELETE statement with a condition based on the list id to delete data from the Shopping\_Lists table.
6. Adding new ingredients: On a recipe page, users can add new ingredients to the shopping list by selecting the name and quantity. This information is then added to the S\_Contain table using an INSERT INTO statement.



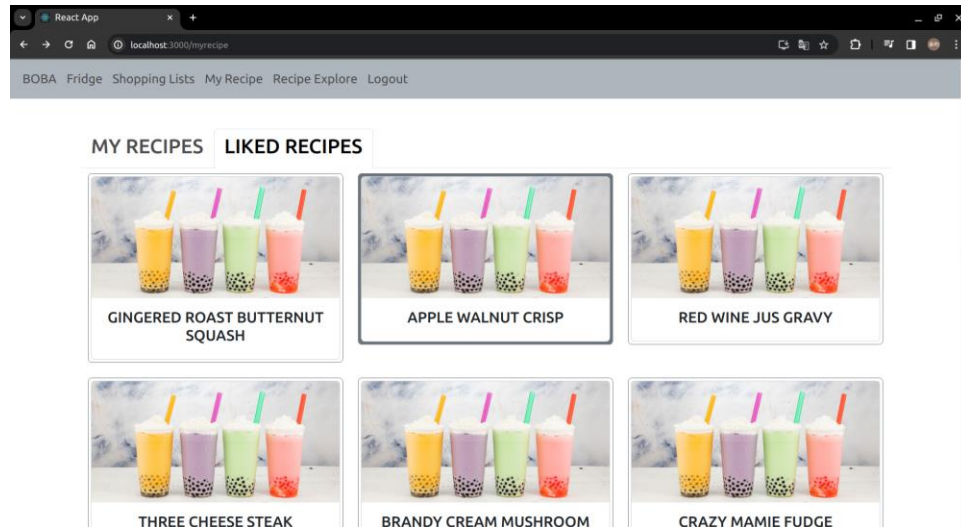
7. Updating ingredient quantities: Users can update ingredient quantities by selecting new values, triggering an UPDATE statement with a condition based on the list id to update the data in the S\_Contain table.
8. Deleting ingredients: Clicking the delete button removes the corresponding ingredient from the shopping list using a DELETE statement with a condition based on the list id to delete data from the S\_Contain table.

- **Exploring Recipes**

1. Reviewing your recipes: Upon accessing the "My Recipe" page, all recipes authored by the user are retrieved from the Recipes table using a SELECT statement with a condition based on the user id.

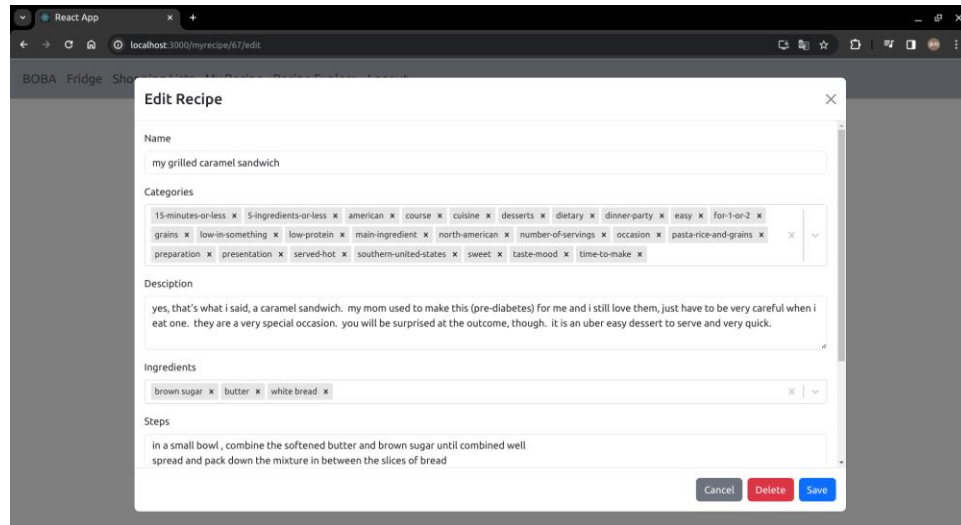


2. Reviewing your liked recipes: Clicking on "Liked Recipe" displays all recipes that the user has liked. This information is fetched using a SELECT statement with the user id from the Like table, along with a NATURAL JOIN statement to gather recipe details from the Recipes table.

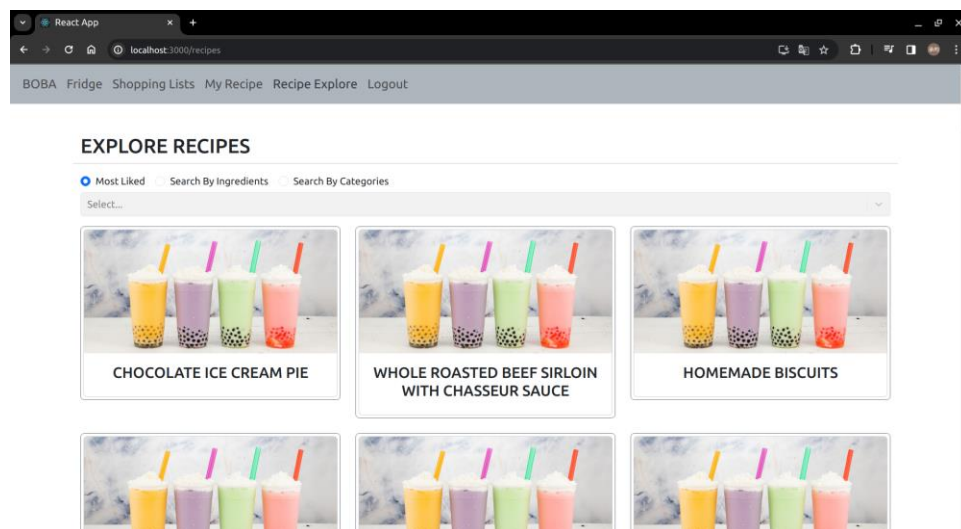


3. Adding new recipes: Clicking the add button prompts users to fill out a form to create a new recipe. Upon clicking save, three INSERT INTO statements are executed: one to add the new recipe to the Recipes table, another to add ingredients to the R\_Contain table, and a third to add categories to the Classify table.

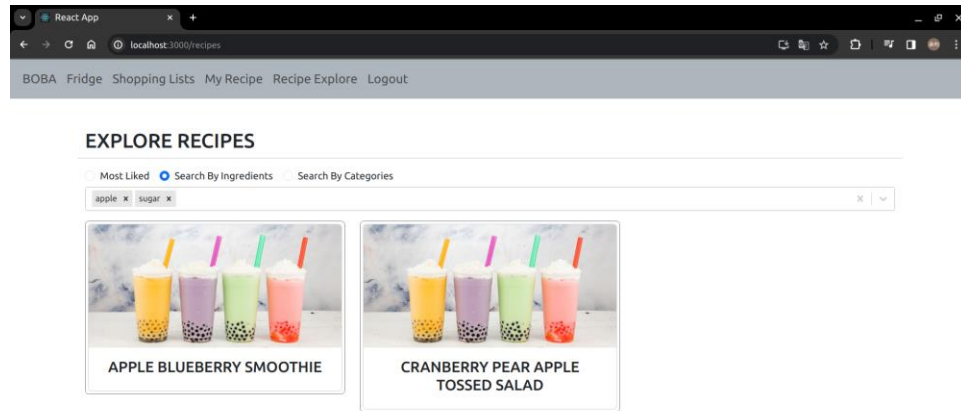
4. Updating recipes: Users can edit their recipes by clicking the edit button. After making changes and clicking save, an UPDATE statement is used for the Recipes table with a condition based on the recipe id. For updating ingredients and categories, INSERT INTO statements are used to add new ingredients or categories, while DELETE statements are used to remove existing ingredients or categories.



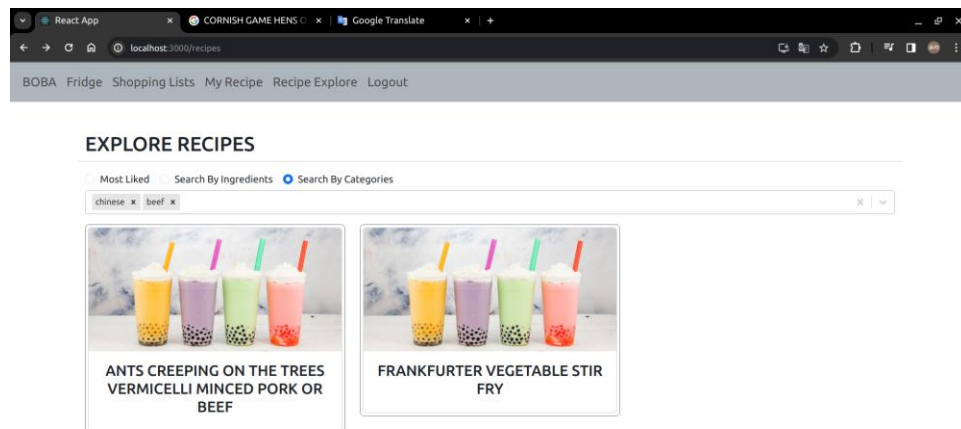
5. Deleting recipes: Clicking the delete button removes the recipe from the Recipes table using a DELETE statement with a condition based on the recipe id. Additionally, due to foreign key constraints in the R\_Contain table and the Classify table, the associated ingredients and categories are also deleted.
6. Exploring popular recipes: Upon entering the "Recipe Explore" section, users are presented with recipes authored by other users, arranged in descending order based on the number of likes they have received. Additionally, recipes liked by the user are excluded from this display. The SQL query utilizes SELECT, NATURAL JOIN, AGGREGATE, and ORDER BY statements to achieve this functionality.



7. Searching recipes by ingredients: In the "Recipe Explore" section, users have the option to search for recipes based on specific ingredients. By selecting ingredients from the search bar, recipes containing those ingredients are displayed and sorted based on the number of likes they have received. The SQL query incorporates SELECT, NATURAL JOIN, INTERSECT, AGGREGATE, and ORDER BY statements to perform this search.



8. Searching recipes by categories: Users can also search for recipes based on their categories. By selecting categories from the search bar, recipes classified under those categories are displayed and sorted based on the number of likes they have received. The SQL query utilizes SELECT, NATURAL JOIN, INTERSECT, AGGREGATE, and ORDER BY statements for this search functionality.



9. Liking recipes: Users can like recipes by clicking the like button, which adds the recipe to their "Liked Recipes" list. This action is implemented using an INSERT INTO statement to create a new like relationship in the Like table.
10. Disliking recipes: If users wish to remove recipes from their "Liked Recipes" list, they can click the like button again to remove the like. This action is executed using a DELETE statement to remove the corresponding like relationship from the Like table.

## Project Source Code

The source code is uploaded to GitHub. Here is the link: <https://github.com/kkeen699/CSCE608-Project1-BOBA>.

## **Discussion**

This project has significantly enhanced my understanding of SQL. While SQL discussions in class seemed easy, implementing an application with it is still challenging, particularly when dealing with complex queries. When developing the search function, I tried numerous attempts to achieve the desired outcome. Regarding the BOBA application, there are several areas that could be improved, such as error handling and database recovery. Incorporating these features would enhance BOBA's robustness and overall performance.