CS 348 - Fall 2022 – Project Milestone 1

1.Introduction

We will use the recipe dataset (https://www.kaggle.com/datasets/shuyangli94/food-com-recipes-and-user-interactions) to make an application for generating recipes from ingredients. We will also provide features to add more recipes from certain ingredients. The users of the application can be anyone who wants to cook from their available ingredients, or people who wants to share their recipes. The administrators of the database systems will be the application administrator.

2.System support

We will use our local machine to build the web application. We will be using Golang (Backend), Postgresql (Database System), and Svelte (Frontend) for our tech stack. We chose to learn new stack to make this project more valuable. Golang helps us to write unit tests along with writing the project, and provides a robust library and tools to increase the performance of running the application. Then, PostgreSQL is one of the most well-developed open-source databases in the world. It has better performance and more functionalities over MySQL, and most importantly it is more friendly in supporting complicated data structures like arrays. Svelte runs at build time, converting the code into highly efficient imperative code that updates DOM, which also provides a better performance for writing applications.

3.Design Schema

List of assumptions:

1. Each recipe should have the following: a recipe id, a name, a list of cooking steps, a prepare time, an author id from its author, a description from the author, a list of tags id from the tags describing the recipe, a list of ingredients id from the ingredients it needs, and a list of nutrition it contains.

2. Each user should have an user id, a password for login, a name, and a list of recipe id he likes.

3. Each ingredient should have an ingredient name, an ingredient id.

4. Each tag should have a tag id, and its name (ex. Fried, crispy)

5. Each recipe id should be unique.

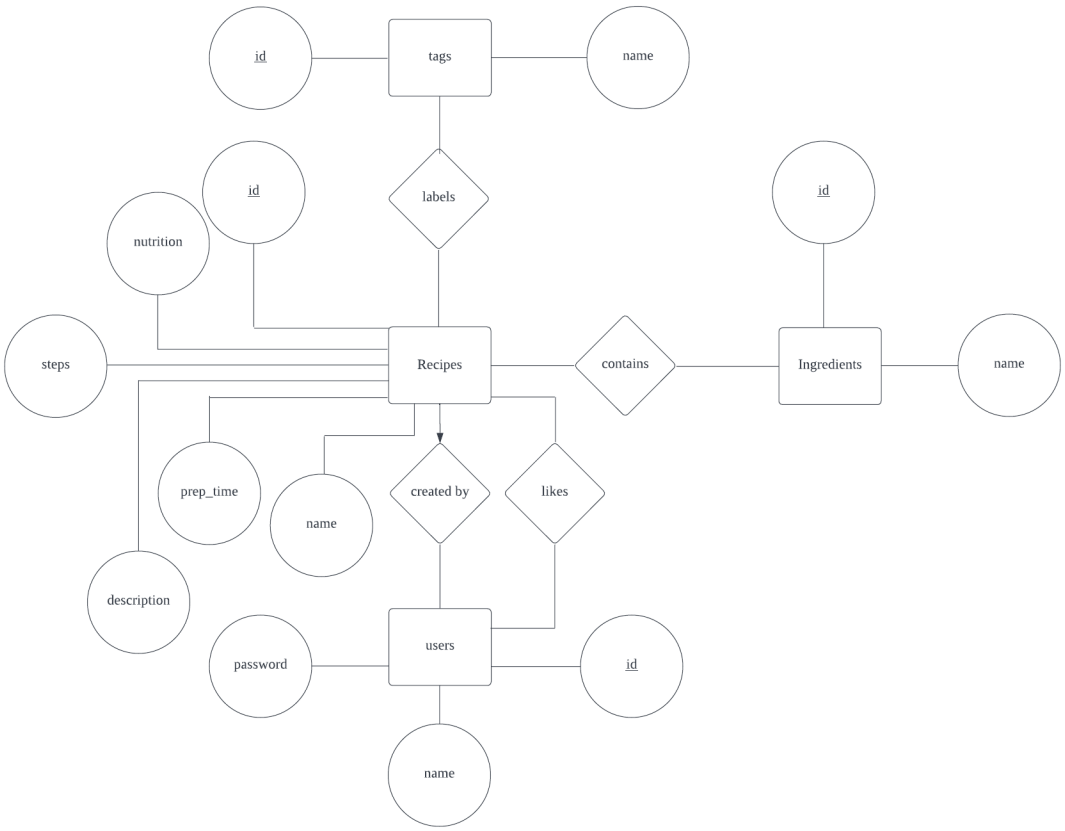
6. Each user id should be unique.

7. Each ingredient id should be unique.

8. Each tag id should be unique.

9. Each recipe should have only one author.

E\R Diagram:



Relational model:

Entity sets are translated into:

Recipes(id int, name text, steps list of text, prep_minutes int, description text, nutrition list of float)

Users(id int, password int, name text)

Ingredients(id int, name text)

Tags(id int, name text)

Relationship sets are translated into:

User_favourites(user_id int, recipe_id int)

Recipe_ingredient(recipe_id int, ingredient id int)

Recipe_tag(recipe_id int, tag_id int)

Recipes_Authors(recipe_id int, user_id int)

Then we can merge the Recipes_Authors table and the recipes table since this is a many-to-one relation, and we can just create a column named author as a foreign key referencing the user_id in users table. The rest of the relationship set tables cannot be merged with the entity set tables because they are many-to-many relationship, and SQL does not allow to define a column as an array of foreign keys.

All tables after merged:

Recipes(id int, name text, steps list of text, prep_minutes int, description text, nutrition list of float)

Users(id int, password int, name text)

Ingredients(id int, name text)

Tags(id int, name text)

User_favourites(user_id int, recipe_id int)

Recipe_ingredient(recipe_id int, ingredient id int)

Recipe_tag(recipe_id int, tag_id int)

4.Sample data

The sample data is from the dataset we mentioned in the introduction part, we used the top 22 entries of their CSV dataset as our example to test the feature we plan to implement. The sample data is stored in \demo\data, each file corresponds to one table we used in the database.

5.Production data

The production data is from the dataset we mentioned above, just download it from the website. The production data is stored at /data directory.

6.Features

We implemented 4 features so far, all the query templates are stored in \server\database\queries, and all the sample data, queries, and output are stored in the \demo\data directory. Simply run the *'milestone1.sh'* and *'featureN.sh'* files to see all the query definitions and expected output.

Feature 1: We allow users to sign up for accounts by creating their user names and passwords. Then, they can use the names and passwords they created to log in. On the main page, users can click on the 'signup' button to create their accounts. Later, after completing signup, they can type their names and passwords on the main page to log in their accounts.

Template:

```
create or replace function userSignup(text, text)
  returns void
  as $$
  insert into users(name, password) values ($1, $2);
  $$ language sql;
create or replace function userLogin(text)
  returns text
  as $$
  select password from users
    where name = $1;
  $$ language sql;
```

Sampled queries:

```
select userSignup('testuser', 'testpass')
select userLogin('testuser')
```

Sample output:

```
testpass
```

The first query 'userSignup' won't produce any output as it returns void.

The second query will produce output—testpass—the password of username testuser.

Implementation:

The sql code is at directory /server/database/queries, file UserLogin and UserSignup.

```
select * from users where name = @username;
insert into users(name, password) values (@username, @password);
```

By executing the curl command:

```
curl -d '{"username": "testfeature", "password": "testpass"}'
http://localhost:8080/api/signup/
```

```
curl -d '{"username": "testfeature", "password": "testpass"}'
http://localhost:8080/api/login/
```

The expected output:

```
{ "status": "success" }// User registered successfully
{ "status": "success" }//username and password matches
```

Feature 2: We implemented autocomplete in the search bar to help users to type ingredients quickly. If users type part of the name of an ingredient, then the autocomplete function will help them to complete the names of the ingredients which helps the user create better searches, get better results, avoid type error, and save a lot of time.

Template:

```
create or replace function autocompleteSearch(text)
  returns ingredients
  as $$
  SELECT * FROM ingredients WHERE name LIKE $1;
  $$ language sql;
```

Sampled queries:

```
select autocompleteSearch('bac%')
```

Sample output:

```
(11,bacon)
```

Implementation:

The sql code is at directory /server/database/queries, file AutocompleteSearch.

```
SELECT * FROM ingredients WHERE name LIKE @substring;
```

By executing the curl command:

```
curl -d 'substring=bac' http://localhost:8080/api/ingredients/autocomplete/
```

The expected output (sample data):

```
{"data":[{"id":11,"name":"bacon"}],"message":"List of Similar
Ingredients","status":"success"}
```

The expected output (production data):

Feature 3: We allow users to use the ingredients they have to search for any recipe that includes these ingredients. The user could simply type what ingredients they have at hand to search for what food they can do, and the step-by-step recipe for that food.

Template:

```
create or replace function matchRecipes(int[])
  returns recipes
  as $$
  SELECT * FROM recipes AS r
  WHERE r.id NOT IN (
    SELECT recipe_id FROM recipe_ingredient AS ri
      WHERE ri.ingredient_id NOT IN (
        SELECT * FROM UNNEST($1)
      )
  );
  $$ language sql;
```

Sampled queries:

```
select matchRecipes('{38,102,29,33,91,82,39,10}')
```

Sample output:

```
(1,"arriba   baked winter squash mexican style","autumn is my favorite time of
year to cook! this recipe \r
can be prepared either spicy or sweet, your choice!\r
two of my posted mexican-inspired seasoning mix recipes are offered as
suggestions.",55,1,"{""'make a choice and proceed with recipe'"",""'depending on
size of squash"",""cut into half or fourths'"",""'remove seeds'"",""'for spicy
squash"",""drizzle olive oil or melted butter over each cut squash
piece'"",""'season with mexican seasoning mix ii'"",""'for sweet
squash"",""drizzle melted honey"",butter,""grated piloncillo over each cut squash
piece'"",""'season with sweet mexican spice mix'"",""'bake at 350
degrees"",""again depending on size"",""for 40 minutes up to an hour"",""until a
fork can easily pierce the skin'"",""'be careful not to burn the squash
especially if you opt to use sugar or butter'"",""'if you feel more
comfortable"",""cover the squash with aluminum foil the first half hour"",""give
```

```
or take"","of baking'"","'if desired"","season with
salt'""}","{51.5,0,13,0,2,0,4}"
```

The output includes name of the recipes and the step-by-step instructions about how to cook the food.

Implementation:

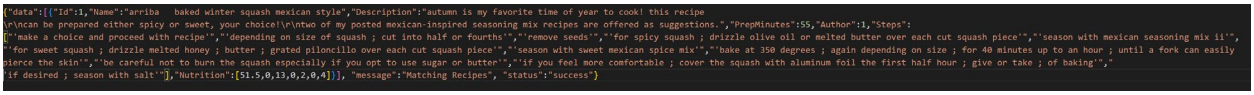
The sql code is at directory /server/database/queries, file MatchRecipes.

```
SELECT * FROM recipes
WHERE id IN ((SELECT id FROM recipes)
  EXCEPT
  (SELECT DISTINCT recipe_id FROM recipe_ingredient
    WHERE NOT (ingredient_id = ANY (CAST(@ingredient_ids AS INT[])))
  );
```

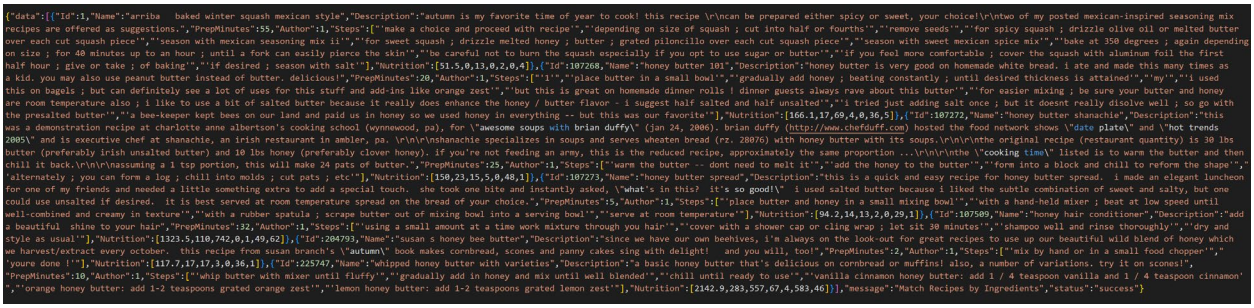
By executing the curl command:

```
curl -H 'Content-type: application/json' \
-d '{"ingredientIds": [4621,4684,6557,8899,10583,12245,14832 ]}' \
http://localhost:8080/api/recipes/match/
```

The expected output (sample data):



The expected output (production data):



Feature 4: We allow users to filter the result of the recipe they searched by tags. Users can click on the 'filter' button and select the tags they want, then a new page will display all recipes that have the tag.

Template:

```
create or replace function filterByTags(int)
  returns recipes
  as $$
  select * from recipes
    where id in (select recipe_id from recipe_tag
      where tag_id = $1);
  $$ language sql;
```

Sample queries:

```
select filterByTags(124)
```

Sample output:

```
(1,"arriba   baked winter squash mexican style","autumn is my favorite time of
year to cook! this recipe \r
can be prepared either spicy or sweet, your choice!\r
two of my posted mexican-inspired seasoning mix recipes are offered as
suggestions.",55,1,"{""'make a choice and proceed with recipe'"",""'depending on
size of squash"","cut into half or fourths'"",""'remove seeds'"",""'for spicy
squash"","drizzle olive oil or melted butter over each cut squash
piece'"",""'season with mexican seasoning mix ii'"",""'for sweet
squash"","drizzle melted honey"",butter,""grated piloncillo over each cut squash
piece'"",""'season with sweet mexican spice mix'"",""'bake at 350
degrees"","again depending on size"","for 40 minutes up to an hour"",""until a
fork can easily pierce the skin'"",""'be careful not to burn the squash
especially if you opt to use sugar or butter'"",""'if you feel more
comfortable"","cover the squash with aluminum foil the first half hour"",""give
or take"","of baking'"",""'if desired"",""season with
salt'""}","{51.5,0,13,0,2,0,4}"
```

Feature 5: We will show 5 recipes that are marked favorite by users the most in the last week. Users can see it in the main page. By clicking on it, the users will be directed to the detailed page of that recipes.

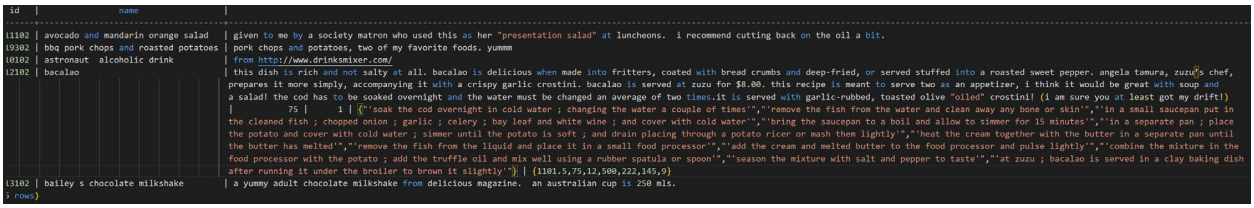
Template:

```
SELECT * FROM recipes WHERE id IN ( SELECT recipe_id FROM (
  SELECT recipe_id, COUNT(*) AS favCount FROM user_favorites
  WHERE liked_at >= @week_timestamp
  GROUP BY recipe_id
  ORDER BY favCount DESC
) AS top_recipes
LIMIT 5 )
```

Sample queries:

```
SELECT * FROM recipes WHERE id IN ( SELECT recipe_id FROM (
  SELECT recipe_id, COUNT(*) AS favCount FROM user_favorites
  WHERE liked_at >= '2022-11-16 21:50:56.44249'
  GROUP BY recipe_id
  ORDER BY favCount DESC
) AS top_recipes
LIMIT 5)
```

Sample output:



Feature 6: On the page of one recipe, besides its ingredients, nutrition, and cooking steps, there are also recommendations for recipes that use similar ingredient.

Template:

```
((SELECT id FROM recipes)
  EXCEPT
  (SELECT DISTINCT recipe_id FROM recipe_ingredient
    WHERE NOT (ingredient_id = ANY (CAST(@fridge1 AS INT[])))))
EXCEPT
((SELECT id FROM recipes)
  EXCEPT
  (SELECT DISTINCT recipe_id FROM recipe_ingredient
    WHERE NOT (ingredient_id = ANY (CAST(@fridge2 AS INT[])))));
```

Sample queries:

```
((SELECT id FROM recipes)
  EXCEPT
  (SELECT DISTINCT recipe_id FROM recipe_ingredient
    WHERE NOT (ingredient_id = ANY (CAST('{4621,4684,6557,8899,10583,12245,14832,
12149}' AS INT[])))))
EXCEPT
((SELECT id FROM recipes)
  EXCEPT
  (SELECT DISTINCT recipe_id FROM recipe_ingredient
    WHERE NOT (ingredient_id = ANY
(CAST('{4621,4684,6557,8899,10583,12245,14832}' AS INT[])))));
```

Sample output:

```
   id
--------
 107465
(1 row)
```

Feature 7: Based on the favorite recipes of users, we will recommend some recipes that are similar to those she/he like. Users can access them by clicking the ‘recommend recipes’ button.

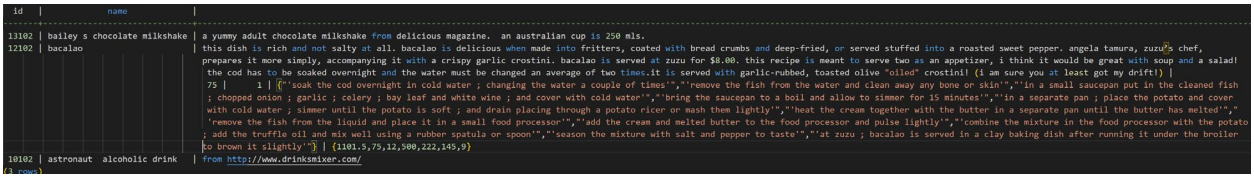
Template:

```
SELECT * FROM recipes WHERE id IN ( SELECT recipesResult.recipe_id FROM (
  SELECT recipe_id, COUNT(*) AS favCount FROM user_favorites
  GROUP BY recipe_id
  HAVING COUNT(*) >= (
    SELECT AVG(favCount2) FROM (
      SELECT uf.recipe_id, COUNT(*) AS favCount2 FROM user_favorites as uf
      GROUP BY recipe_id
    ) as favRecipeWithCounts, user_favorites as uf2
    WHERE uf2.user_id = @user_id AND uf2.recipe_id =
favRecipeWithCounts.recipe_id
  )
) AS recipesResult
ORDER BY RANDOM()
LIMIT 3)
```

Sample queries:

```
SELECT * FROM recipes WHERE id IN ( SELECT recipesResult.recipe_id FROM (
  SELECT recipe_id, COUNT(*) AS favCount FROM user_favorites
  GROUP BY recipe_id
  HAVING COUNT(*) >= (
    SELECT AVG(favCount2) FROM (
      SELECT uf.recipe_id, COUNT(*) AS favCount2 FROM user_favorites as uf
      GROUP BY recipe_id
    ) as favRecipeWithCounts, user_favorites as uf2
    WHERE uf2.user_id = 2 AND uf2.recipe_id = favRecipeWithCounts.recipe_id
  )
) AS recipesResult
ORDER BY RANDOM()
LIMIT 3)
```

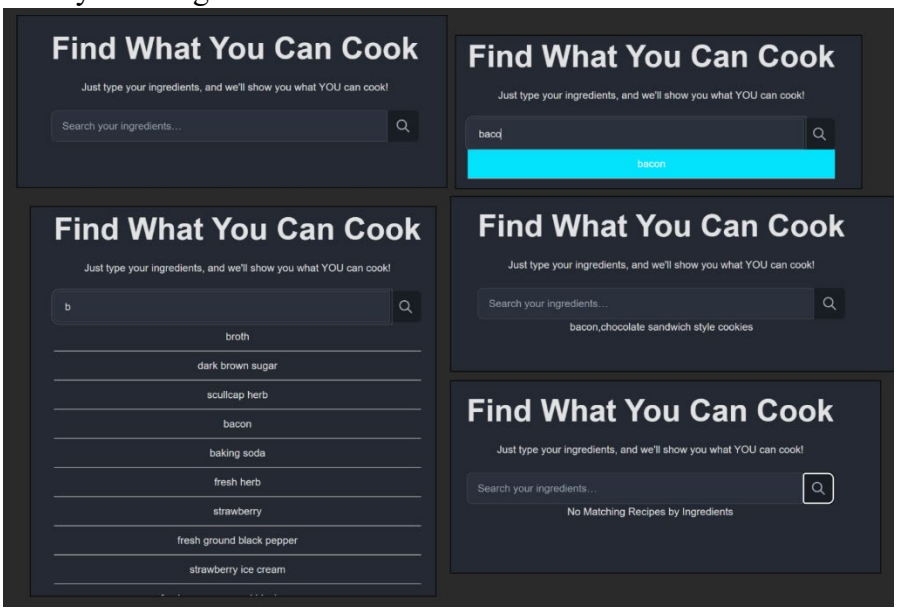
Sample output:



Future Features:

- We allow users to add and query their favorite recipes. Users can click on the ‘like’ button to add the recipe to their favorite recipe lists, then access these recipes conveniently later.
- We allow users to filter recipes based on nutrition and prep time, same as feature 4.

- We allow users to order recipes by prep time, # of ingredients, # of favorites from users/likes by simply click on the ‘order’ button, then choose ‘decreasing order’ or ‘increasing order’.
-Fancy UI design



7.Members

- Edward Ryan
  - Wrote the code for SQL queries and applications.
- Kevin Xu
  - Wrote sample SQL files, tested it with sample data.
  - Wrote the report.
- Nicholaus Suprapto
  - Wrote the code for SQL queries and applications.
- Shuyao Shi
  - Wrote sample SQL files, tested it with sample data.
  - Wrote the report.
- Warren Elbert
  - Wrote the code for SQL queries and applications.

We will be using a **private** repository.

Github Link: https://github.com/chic27/FindMyRecipe