University of British Columbia, Vancouver
Department of Computer Science

CPSC 304 Project Cover Page

Milestone #: 4

Date: 11.29.2024

Group Number: 107

Name	Student Number	CS Alias (Userid)	Preferred E-mail Address
Keisuke Yamamoto	39088984	g1d4h	ykei2356@gmail.com
Yuto Kikuta	32572265	v4k9h	jordan2002222@gmail.com
Seokhee Hong	91166660	t9z3e	tjrgml1207@naver.com

By typing our names and student numbers in the above table, we certify that the work in the attached assignment was performed solely by those whose names and student IDs are included above. (In the case of Project Milestone 0, the main purpose of this page is for you to let us know your e-mail address, and then let us assign you to a TA for your project supervisor.)

In addition, we indicate that we are fully aware of the rules and consequences of plagiarism, as set forth by the Department of Computer Science and the University of British Columbia

1. Description about project: The database will provide functionality to store, retrieve, and manage various recipe-related data. Users can look up histories for recipes that have been searched using AI and saved to the database, filter recipes by cuisine, category, or dietary restriction, and receive suggestions that meet their health and allergen requirements. And the user can see some stats for recipes in the history like number of recipes for each category, number of recipes for each category-restriction pair, most frequent cuisine, recipe for all restrictions.

2. Change on Schema:
   User: We added 'email', and 'password' attributes for the login system so we deleted the 'username' for simplicity.

3. SQL query
   - 3.1. Insert: insertRecipe()
     file: nutrichef/lib/actions.ts, line: 355
   - 3.2. Update: export async function updateRecipeTitle()
     file: nutrichef/lib/actions.ts, line: 404
   - 3.3. Delete: export async function deleteRecipe()
     file: nutrichef/lib/actions.ts, line: 425
   - 3.4. Selection:export async function fetchFilteredRecipes()
     file: nutrichef/lib/actions.ts, line: 444
   - 3.5. Projection:export async function fetchCustomNutritionFacts()
     file: nutrichef/lib/actions.ts, line: 502
   - 3.6. Join: export async function fetchRecipeByIngredients()
     file: nutrichef/lib/actions.ts, line: 549
   - 3.7. Aggregation with GROUP BY: numOfRecipesByCategory()
     file: nutrichef/lib/actions.ts, line: 570

```
SELECT
  c.name AS category,
  COUNT(r.id) AS recipe_count
FROM categories c
LEFT JOIN recipe_categories rc ON c.id = rc.category_id
LEFT JOIN recipes r ON rc.recipe_id = r.id
```

```
    WHERE r.user_id = ${userId} -- Filter by userId
    GROUP BY c.name
    ORDER BY recipe_count DESC; -- Optional: Order by count,
descending
```

Explanation: This SQL query retrieves a list of categories along with the count of recipes created by a specific user (`userId`), grouping the results by category name. It uses `LEFT JOIN` to include all categories, even those without associated recipes, and sorts the results in descending order of recipe count.

3.8. Aggregation with HAVING: fetchCuisineWithMostPopularRecipes()

file: nutrichef/lib/actions.ts, line: 596

```
    SELECT
      c.name AS cuisine,
      COUNT(rc.recipe_id) AS count
    FROM cuisines c
    JOIN recipe_cuisines rc ON c.id = rc.cuisine_id
    JOIN recipes r ON rc.recipe_id = r.id
    WHERE r.user_id = ${userId}
    GROUP BY c.name
    HAVING COUNT(rc.recipe_id) > 0
    ORDER BY count DESC
    LIMIT 1;
```

Explanation: The first query retrieves a list of recipe categories for a specific user, showing the count of recipes in each category and sorting them in descending order of count. The second query identifies the most popular cuisine for the same user by counting recipes associated with each cuisine, filtering out cuisines with no recipes, and returning the top cuisine based on recipe count.

3.9. Nested aggregation with GROUP BY: getCuisinesAboveGlobalAverage()

file: nutrichef/lib/actions.ts, line: 633

```
    SELECT
```

```
      category_group.category_name,
      dr.name AS restriction_name,
      COUNT(category_group.recipe_id) AS recipes_num
    FROM (
      SELECT
        cat.name AS category_name,
        r.id AS recipe_id
      FROM recipes r
      JOIN recipe_categories rc ON r.id = rc.recipe_id
      JOIN categories cat ON rc.category_id = cat.id
      WHERE r.user_id = ${userId}
      GROUP BY cat.name, r.id
      HAVING COUNT(r.id) <= ALL (
        SELECT COUNT(r2.id)
        FROM recipes r2
        JOIN recipe_categories rc2 ON r2.id = rc2.recipe_id
        JOIN categories cat2 ON rc2.category_id = cat2.id
        WHERE r2.user_id = ${userId}
        GROUP BY cat2.name, r2.id
      )
    ) AS category_group
    JOIN recipe_dietary_restrictions rdr ON category_group.recipe_id
= rdr.recipe_id
    JOIN dietary_restrictions dr ON rdr.dietary_id = dr.id
    GROUP BY category_group.category_name, dr.name
    ORDER BY category_group.category_name, dr.name;
```

Explanation:

This SQL query identifies dietary restrictions associated with recipes in the category with the lowest number of recipes for a specific user (user_id = 22). It groups the results by category name and dietary restriction name, counting how many recipes exist for each combination and ordering the output alphabetically by category and dietary restriction names.

3.10 Division

file: nutrichef/lib/actions.ts, line: 688

```
SELECT r.id AS recipe_id, r.title AS recipe_title
FROM recipes r
WHERE NOT EXISTS (
  SELECT dr.id
  FROM dietary_restrictions dr
  WHERE NOT EXISTS (
    SELECT 1
    FROM recipe_dietary_restrictions rdr
    WHERE rdr.recipe_id = r.id
    AND rdr.dietary_id = dr.id
  )
)
AND r.user_id = ${userId};
```

Explanation: This SQL query retrieves the IDs and titles of recipes created by a specific user (userId) that meet all possible dietary restrictions. It uses nested NOT EXISTS clauses to ensure that for every dietary restriction in the system, there is a corresponding record in the recipe_dietary_restrictions table for the recipe.

SQL script for initialization

```
import { db } from "@vercel/postgres";
import {
 categories,
 cuisines,
 dietaryRestrictions,
} from "../../../lib/placeholder-data";
import bcrypt from "bcrypt";

export async function GET() {
 const client = await db.connect();

 try {
   await client.sql`BEGIN`;

   // Drop existing tables to allow re-running the script
   await client.sql`
     DROP TABLE IF EXISTS ingredient_allergens;
     DROP TABLE IF EXISTS perishable_ingredients;
     DROP TABLE IF EXISTS recipe_ingredients;
     DROP TABLE IF EXISTS recipe_steps;
     DROP TABLE IF EXISTS nutrition_facts;
     DROP TABLE IF EXISTS recipe_dietary_restrictions;
     DROP TABLE IF EXISTS recipe_cuisines;
     DROP TABLE IF EXISTS recipe_categories;
     DROP TABLE IF EXISTS allergens;
     DROP TABLE IF EXISTS dietary_restrictions;
     DROP TABLE IF EXISTS cuisines;
     DROP TABLE IF EXISTS categories;
     DROP TABLE IF EXISTS ingredients;
     DROP TABLE IF EXISTS recipes;
     DROP TABLE IF EXISTS users;
   `;

   // Create tables
   await client.sql`
     CREATE TABLE IF NOT EXISTS users (
       id SERIAL PRIMARY KEY,
       email VARCHAR(255) UNIQUE NOT NULL,
       password TEXT NOT NULL
     );
   `;

   await client.sql`
     CREATE TABLE IF NOT EXISTS recipes (
       id SERIAL PRIMARY KEY,
```

```
    user_id INT REFERENCES users(id) ON DELETE CASCADE,
    title VARCHAR(255) NOT NULL,
    description TEXT NOT NULL,
    cooking_time INT NOT NULL
  );
`;

await client.sql`
  CREATE TABLE IF NOT EXISTS categories (
    id SERIAL PRIMARY KEY,
    name VARCHAR(255) UNIQUE NOT NULL
  );
`;

await client.sql`
  CREATE TABLE IF NOT EXISTS cuisines (
    id SERIAL PRIMARY KEY,
    name VARCHAR(255) UNIQUE NOT NULL
  );
`;

await client.sql`
  CREATE TABLE IF NOT EXISTS dietary_restrictions (
    id SERIAL PRIMARY KEY,
    name VARCHAR(255) UNIQUE NOT NULL,
    description TEXT
  );
`;

await client.sql`
  CREATE TABLE IF NOT EXISTS recipe_categories (
    recipe_id INT REFERENCES recipes(id) ON DELETE CASCADE,
    category_id INT REFERENCES categories(id) ON DELETE CASCADE,
    PRIMARY KEY (recipe_id, category_id)
  );
`;

await client.sql`
  CREATE TABLE IF NOT EXISTS recipe_cuisines (
    recipe_id INT REFERENCES recipes(id) ON DELETE CASCADE,
    cuisine_id INT REFERENCES cuisines(id) ON DELETE CASCADE,
    PRIMARY KEY (recipe_id, cuisine_id)
  );
`;
```

```
  await client.sql`
    CREATE TABLE IF NOT EXISTS recipe_dietary_restrictions (
      recipe_id INT REFERENCES recipes(id) ON DELETE CASCADE,
      dietary_id INT REFERENCES dietary_restrictions(id) ON DELETE CASCADE,
      PRIMARY KEY (recipe_id, dietary_id)
    );
  `;

  await client.sql`
    CREATE TABLE IF NOT EXISTS ingredients (
      id SERIAL PRIMARY KEY,
      name VARCHAR(255) UNIQUE NOT NULL,
      storage_temp INT
    );
  `;

  await client.sql`
    CREATE TABLE IF NOT EXISTS perishable_ingredients (
      id INT REFERENCES ingredients(id) PRIMARY KEY,
      shelf_life INT NOT NULL
    );
  `;

  await client.sql`
    CREATE TABLE IF NOT EXISTS allergens (
      id SERIAL PRIMARY KEY,
      name VARCHAR(255) UNIQUE NOT NULL
    );
  `;

  await client.sql`
    CREATE TABLE IF NOT EXISTS ingredient_allergens (
      ingredient_id INT REFERENCES ingredients(id) ON DELETE CASCADE,
      allergen_id INT REFERENCES allergens(id) ON DELETE CASCADE,
      PRIMARY KEY (ingredient_id, allergen_id)
    );
  `;

  await client.sql`
    CREATE TABLE IF NOT EXISTS recipe_ingredients (
      recipe_id INT REFERENCES recipes(id) ON DELETE CASCADE,
      ingredient_id INT REFERENCES ingredients(id) ON DELETE CASCADE,
      PRIMARY KEY (recipe_id, ingredient_id)
    );
  `;
```

```
await client.sql`
  CREATE TABLE IF NOT EXISTS recipe_steps (
    recipe_id INT REFERENCES recipes(id) ON DELETE CASCADE,
    step_num INT NOT NULL,
    description TEXT NOT NULL,
    PRIMARY KEY (recipe_id, step_num)
  );
`;

await client.sql`
  CREATE TABLE IF NOT EXISTS nutrition_facts (
    nutrition_id SERIAL PRIMARY KEY,
    recipe_id INT UNIQUE REFERENCES recipes(id) ON DELETE CASCADE,
    calories INT NOT NULL,
    proteins INT NOT NULL,
    fats INT NOT NULL
  );
`;

// Insert data into Categories
for (const category of categories) {
  await client.sql`
    INSERT INTO categories (name)
    VALUES (${category.name})
    ON CONFLICT (name) DO NOTHING;
  `;
}

// Insert data into Cuisines
for (const cuisine of cuisines) {
  await client.sql`
    INSERT INTO cuisines (name)
    VALUES (${cuisine.name})
    ON CONFLICT (name) DO NOTHING;
  `;
}

// Insert data into Dietary Restrictions
for (const restriction of dietaryRestrictions) {
  await client.sql`
    INSERT INTO dietary_restrictions (name, description)
    VALUES (${restriction.name}, ${restriction.description})
    ON CONFLICT (name) DO NOTHING;
  `;
```

```
  }

  const hashedPassword = await bcrypt.hash("Yk20020221", 10);
  const email = "jordan2002222@gmail.com";

  await client.sql`
    INSERT INTO users (email, password)
    VALUES
      (${email}, ${hashedPassword});
  `;

  await client.sql`
    INSERT INTO recipes (user_id, title, description, cooking_time)
    VALUES
      (1, 'Vegan Buddha Bowl', 'A healthy bowl with quinoa, roasted vegetables,
and tahini dressing.', 30),
      (1, 'Chicken Tikka Masala', 'A flavorful Indian dish with creamy tomato
curry and marinated chicken.', 45),
      (1, 'Gluten-Free Pancakes', 'Fluffy pancakes made with almond flour and a
hint of vanilla.', 20);
  `;

  await client.sql`
    INSERT INTO recipe_categories (recipe_id, category_id)
    VALUES
      (1, 1), -- Healthy Eating
      (2, 2), -- Indian Cuisine
      (3, 3); -- Breakfast
  `;

  await client.sql`
    INSERT INTO recipe_cuisines (recipe_id, cuisine_id)
    VALUES
      (1, 4), -- Fusion
      (2, 5), -- Indian
      (3, 6); -- American
  `;

  await client.sql`
    INSERT INTO recipe_dietary_restrictions (recipe_id, dietary_id)
    VALUES
      (1, 1),
      (1, 2),
      (1, 3),
      (1, 4),
```

```
    (1, 5),
    (1, 6),
    (1, 7),
    (1, 8),
    (1, 9),
    (1, 10),
    (1, 11),
    (1, 12),
    (1, 13),
    (1, 14),
    (1, 15),
    (2, 1),
    (2, 2),
    (2, 3),
    (2, 4),
    (2, 5),
    (2, 6),
    (2, 7),
    (2, 8),
    (2, 9),
    (2, 10),
    (2, 11),
    (2, 12),
    (2, 13),
    (2, 14),
    (2, 15),
    (3, 1),
    (3, 2),
    (3, 3),
    (3, 4),
    (3, 5),
    (3, 6),
    (3, 7),
    (3, 8),
    (3, 9),
    (3, 10),
    (3, 11),
    (3, 12),
    (3, 13),
    (3, 14),
    (3, 15);
`;

await client.sql`
  INSERT INTO ingredients (name, storage_temp)
```

```
  VALUES
    ('Quinoa', 25), -- Non-perishable
    ('Broccoli', 5), -- Perishable
    ('Chicken Breast', -2), -- Perishable
    ('Tomato', 8), -- Perishable
    ('Almond Flour', 25), -- Non-perishable
    ('Vanilla Extract', 25), -- Non-perishable
    ('Tahini', 25), -- Non-perishable
    ('Yogurt', 5), -- Perishable
    ('Milk', 5); -- Perishable
`;

await client.sql`
  INSERT INTO perishable_ingredients (id, shelf_life)
  VALUES
    (2, 7), -- Broccoli
    (3, 3), -- Chicken Breast
    (4, 5), -- Tomato
    (8, 7), -- Yogurt
    (9, 5); -- Milk
`;

await client.sql`
  INSERT INTO allergens (name)
  VALUES
    ('Sesame'), -- Tahini
    ('Dairy'), -- Yogurt, Milk
    ('Tree Nuts'); -- Almond Flour
`;

await client.sql`
  INSERT INTO ingredient_allergens (ingredient_id, allergen_id)
  VALUES
    (7, 1), -- Tahini -> Sesame
    (8, 2), -- Yogurt -> Dairy
    (9, 2), -- Milk -> Dairy
    (5, 3); -- Almond Flour -> Tree Nuts
`;

await client.sql`
  INSERT INTO recipe_ingredients (recipe_id, ingredient_id)
  VALUES
    (1, 1), -- Buddha Bowl -> Quinoa
    (1, 2), -- Buddha Bowl -> Broccoli
    (1, 7), -- Buddha Bowl -> Tahini
```

```
      (2, 3), -- Tikka Masala -> Chicken Breast
      (2, 4), -- Tikka Masala -> Tomato
      (2, 8), -- Tikka Masala -> Yogurt
      (3, 5), -- Pancakes -> Almond Flour
      (3, 6), -- Pancakes -> Vanilla Extract
      (3, 9); -- Pancakes -> Milk
  `;

  await client.sql`
    INSERT INTO recipe_steps (recipe_id, step_num, description)
    VALUES
      (1, 1, 'Cook quinoa according to package instructions.'),
      (1, 2, 'Roast broccoli with olive oil in the oven.'),
      (1, 3, 'Assemble the bowl and drizzle with tahini dressing.'),
      (2, 1, 'Marinate chicken in yogurt and spices.'),
      (2, 2, 'Cook chicken in a skillet until golden brown.'),
      (2, 3, 'Prepare a creamy tomato-based curry sauce and simmer chicken in
it.'),
      (3, 1, 'Mix almond flour, eggs, vanilla, and milk in a bowl.'),
      (3, 2, 'Heat a skillet and pour batter to form pancakes.'),
      (3, 3, 'Cook until golden brown on both sides.');
  `;

  await client.sql`
    INSERT INTO nutrition_facts (recipe_id, calories, proteins, fats)
    VALUES
      (1, 450, 12, 15), -- Buddha Bowl
      (2, 600, 40, 20), -- Tikka Masala
      (3, 300, 10, 10); -- Pancakes
  `;

  await client.sql`COMMIT`;

  return new Response(
    JSON.stringify({ message: "Database seeded successfully!" }),
    { status: 200 }
  );
 } catch (error) {
  await client.sql`ROLLBACK`;
  console.error("Database seeding failed:", error);
  return new Response(JSON.stringify({ error: "Failed to seed database" }), {
    status: 500,
  });
 } finally {
  client.release();
```

```
  }
}
```