**University of British Columbia, Vancouver**
Department of Computer Science

# CPSC 304 Project Cover Page

Milestone # 4

Date: November 29, 2024

Group Number: 28

| Name | Student ID | CS Alias (Userid) | Preferred Email Address |
|---|---|---|---|
| Gagenvir Gill | 86152758 | a9g3t | gagenvirg@gmail.com |
| Preston Lai | 22541395 | q1m4a | preston1lai@gmail.com |
| Mave Hur | 55693402 | o3x3e | jmavehur@gmail.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.

# Deliverable 1 - Cover page

Please see above.

# Deliverable 2 - Repository link

- Submitted on Canvas
- Link to our GitHub Repository (Group 28): https://github.students.cs.ubc.ca/CPSC304-2024W-T1/project_a9g3t_o3x3e_q1m4a

# Deliverable 3 - Single SQL script

- In the GitHub repository, please find a file named “database_initialization.sql” (Link)

# Deliverable 4 - Project description and accomplishment

- FridgeFind is a kitchen management application that allows users to organize their kitchens, recipes, and other information for later use. It allows users to find new recipes based on those that other users have created while also being able to filter through them based on category tags that can be optionally assigned to recipes. The application also allows users to organize recipes that they or others have created by putting them in lists and allows users to track what ingredients they have and allergies they have managed by being able to store that information in the application as well. Finally, the application allows for users to add other users and add them as in app friends.

# Deliverable 5 - Changed schema

## A description of how our final schema differs from the previous schema:

User/Username Table

- The name of the Username table was changed to AppUser to better reflect its purpose as a representation of user data,
- Removed the Profile Picture attribute as our team encountered persistent technical challenges in handling and storing profile picture files, including complexities in file management
- Changed the name of the Name attribute to FullName as it was more clear that it should/would include the AppUser’s first and last name.

Recipe Table
- The foreign key reference has been updated from User(Username) to AppUser(Username) to reflect the updated table name.

RecipeIngredient Table
- The primary key was changed to also include the RecipeID, thus making a RecipeIngredient 'belong' to a Recipe, instead of having them be separate, we made a RecipeIngredient a weak entity of a Recipe.

Category Table
- The attribute Name has been renamed to CategoryName to better align with naming conventions.

RecipeHasCategory Table
- The foreign key reference has been updated from Category(Name) to Category(CategoryName) to reflect the updated table name.

RecipeListHasRecipe Table was Created
- This table was created to represent the missing many to many relationship between a Recipe and a RecipeList

AllergyList Table
- The foreign key reference has been updated from User(Username) to AppUser(Username) to reflect the updated table name
- The attribute Description has been renamed to ListDescription for clarity
- The attribute Name has been renamed to ListName to better align with naming conventions

Change to all Foreign Key References
- Changed all foreign key references to include ON DELETE CASCADE to be able to delete entities in a safe method that ensures no table has values that no longer exist. We believe this is beneficial as if for example a AppUser is deleted or a Recipe is deleted, then the relations it is related to in the Friends or RecipeStep tables should also be deleted. Further, we would also include ON UPDATE CASCADE, however as Oracle does not support that feature, we instead made sure that changing the value of a Foreign Key (AKA a Updating a tables Primary Key) was NOT possible.

## Deliverable 6 - SQL queries to satisfy rubric

(NOTE: All queries are implemented in a service methods in the 'appService.js' file)

## 6.1 INSERT

Function: ‘insertUser’ on line 92

- INSERT INTO AppUser(Username, Email, FullName, DefaultPrivacyLevel) VALUES (:Username, :Email, :FullName, :DefaultPrivacyLevel)

## 6.2 UPDATE

Function: ‘updateUser’ on line 142

- ```
  let queryParams = [];
  let querySetClauses = [];

  if (NewEmail != "") {
    querySetClauses.push(`Email =: NewEmail`);
    queryParams.push(NewEmail);
  }
  if (NewFullName != "") {
    querySetClauses.push(`FullName =: NewFullName`);
    queryParams.push(NewFullName);
  }
  if (NewDefaultPrivacyLevel != "Do Not Change") {
    querySetClauses.push(`DefaultPrivacyLevel =: NewDefaultPrivacyLevel`);
    queryParams.push(NewDefaultPrivacyLevel);
  }

  let query = `UPDATE AppUser SET `;
  query += querySetClauses.join(`, `);
  query += ` WHERE Username =: Username`;
  queryParams.push(Username);
  ```

## 6.3 DELETE

Function: ‘deleteUser’ on line 118

- DELETE FROM AppUser WHERE Username=:Username

## 6.4 SELECTION

Function: ‘areTheyFriends’ on line 274

- SELECT * FROM Friends WHERE (Username1 = :username1 AND Username2 = :username2) OR (Username1 = :username2 AND Username2 = :username1)
  ```

## 6.5 PROJECTION

Function: ‘fetchAllergyListByProjectFromDb’ on line 1208

- SELECT ${formattedColumns}
  FROM AllergyList
- This query retrieves the user-selected columns from formattedColumns and displays only the chosen columns in the table.

## 6.6 JOIN

Function: ‘fetchRecipesByRecipeListFromDb’ on line 388

- SELECT r.*
  FROM Recipe r
  JOIN RecipeListHasRecipe rlhr ON r.RecipeID = rlhr.RecipeID
  WHERE rlhr.RecipeListID=:RecipeListID

## 6.7 Aggregation with GROUP BY

Function: ‘fetchAllergyListPrivacyLevelCounts’ on line 1248

SELECT PrivacyLevel,
COUNT(*) AS PrivacyLevelCount FROM AllergyList
GROUP BY PrivacyLevel

- This query is hardcoded and will count how many AllergyLists are in each Privacy Level.

## 6.8 Aggregation with HAVING

Function: ‘fetchNumAllergiesPerUserHaving’ on line 1261

SELECT al.Username, COUNT(DISTINCT alhai.IngredientID) AS NumberOfAllergies
FROM AllergyList al
JOIN AllergyListHasAllergicIngredient alhai ON al.IngredientListID = alhai.IngredientListID
GROUP BY al.Username
HAVING COUNT(DISTINCT alhai.IngredientID) > 0

- This query is hardcoded and will count how many App Users actually have Allergies and the number of allergies that User has.

## 6.9 Nested aggregation with GROUP BY

Function: ‘fetchSimpleOrComplicatedRecipesFromDb’ on line 305

- let query = `SELECT r.*, (SELECT *COUNT*(*) FROM RecipeIngredient ri WHERE ri.RecipeID = r.RecipeID), (SELECT *COUNT*(*) FROM RecipeStep rs WHERE rs.RecipeID = r.RecipeID) FROM Recipe r WHERE (SELECT *COUNT*(*) FROM RecipeIngredient ri WHERE ri.RecipeID = r.RecipeID) `;
  if (Difficulty === 'Simple') {
  query += `< `;
  } else if (Difficulty === 'Complicated') {

```
    query += `> `;
  }
  query += `(SELECT AVG(IngredientCount) FROM (SELECT COUNT(*) AS IngredientCount
  FROM RecipeIngredient GROUP BY RecipeID)) AND (SELECT COUNT(*) FROM
  RecipeStep rs WHERE rs.RecipeID = r.RecipeID) `;
  if (Difficulty === 'Simple') {
    query += `< `;
  } else if (Difficulty === 'Complicated') {
    query += `> `;
  }
  query += `(SELECT AVG(StepCount) FROM (SELECT COUNT(*) AS StepCount FROM
  RecipeStep GROUP BY RecipeID))`;
```

- This query finds Recipes that have either an above average amount of RecipeSteps and RecipeIngredients OR have a below average amount of RecipeSteps and RecipeIngredients. The user decides whether they want above average or below average by selecting ‘Simple or Complicated in a selection bar which is passed to the service method, if the user selected simple, they get below average which is Recipes that are ‘simple’ and if they selected complicated they get Recipes that are above average, ie ‘complicated’.

## 6.10 DIVISION

Function: ‘viewUsersWhoAreFriendsWithEveryone’ on line 207

- ```
  SELECT u.Username
  FROM AppUser u
  WHERE NOT EXISTS (
    SELECT a.Username
    FROM AppUser a
    WHERE a.Username != u.Username
     AND NOT EXISTS (
      SELECT 1
      FROM Friends f
      WHERE (f.Username1 = u.Username AND f.Username2 = a.Username)
        OR (f.Username1 = a.Username AND f.Username2 = u.Username)
    )
  )
  ```
- This query is completely hardcoded, it finds AppUsers that are friends with all other AppUsers, i.e. has a tuple in the ‘Friends’ table with every other AppUser.
  ```