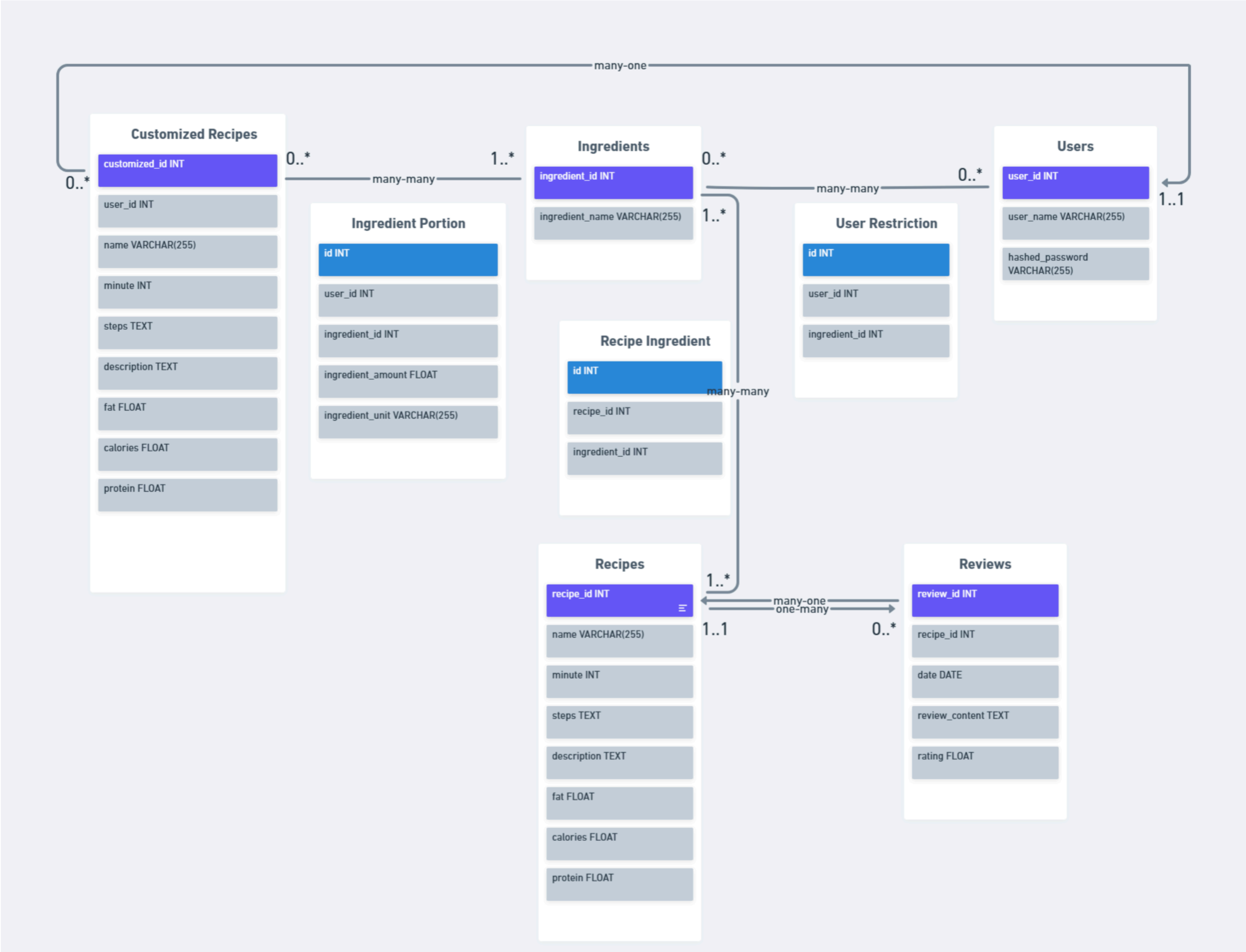
CS 411 Stage 2 Report

UML Diagram



Entity explanations

Recipes:
Recipe is the primary data table of our application and the data is imported from the Food.com dataset. It stores the attributes like time to prepare, steps and macro-nutrients that are essential for users to do meal planning. It is our primary data source, so it takes up an entire data table.

Ingredients:
Ingredients hold all the ingredients used in the recipes. Although it has only two columns: ingredient_id and ingredient_name, we have this table to allow changes to ingredients effectively cascades to all recipes. It has the **many-many** relation to the Recipes table, for one ingredient can be used in different recipes, and one recipe cna have different ingredients.

Users:
The user entity has three attributes: user_id, user_name, and hashed_password. The user entity is created by this application rather than imported from Food.com dataset. The user information is kept in a separate table because the user account information is irrelevant with other attributes in other tables. The user account is essential for saving the customized recipe based on the original recipe. The Users table has **many-many** relation to the Ingredients table, for one user may have many food restrictions, and one ingredient can be the restriction for many users. The relation table is shown in User Restriction.

Reviews:
The review entity has four attributes: recipe_id, date, review_comment and rating. It is imported from the Food.com dataset. It shows how past users review the particular recipe, including the text comment and rating on a scale of 5. The review table has a **many-one** relation with the Recipes entity because many reviews can correspond to one recipe.

Customized_recipes
The customized_recipes saves information similar to recipes table, but the content is customized by the users to better fit their needs. In addition, we want to allow the user to customize how much of each ingredient and the nutrition content. This entity has a **many-one** relation to the Users entity, for multiple customized recipes can belong to the same user. The entity also has a **many-many** relation to the Ingredients entity, and we will model this relationship in an ingredient_portion table.

Normalization Process

In the normalization process, for each of our entity tables, we have the following functional dependencies.
Recipes:
recipe_id -> name, minutes, description, steps, fat, calories, protein
Reviews:
review_id -> recipe_id, review_content, rating, date
Ingredients:
ingredient_id -> name
Users:
user_id -> user_name, hashed_password
Customized_recipes:
customized_id -> user_id, recipe_id, name, minutes, description, steps, fat, calories, protein

Since each of these entities have only one functional dependency, and the left side is a superkey. Our entity tables conform to the BCNF normalization form, and no further decomposition is needed.

Relational schema:

Recipes(recipe_id: INT [PK], name: VARCHAR(255) , minutes: INT, description: TEXT, steps: TEXT, fat: FLOAT, calories: FLOAT, protein: FLOAT)

Reviews(review_id: INT [PK], recipe_id: INT [FK to Recipes.recipe_id], review_content: TEXT, rating:FLOAT, date: DATE)

Ingredients(ingredient_id:INT [PK], name: VARCHAR(255))

Recipe_ingredients(id:INT [PK], recipe_id: INT [FK to recipes.recipe_id], ingredient_id: INT [FK to ingredients.ingredient_id])

Users(user_id: INT [PK], user_name: VARCHAR(255), hashed_password: VARCHAR(255))

Customized_recipes(customized_id: INT [PK], user_id: INT [FK to Users.user_id], recipe_id: INT [FK to recipes.recipe_id] , name: VARCHAR(255), minutes: INT, description: TEXT, steps: TEXT, fat: FLOAT, calories: FLOAT, protein: FLOAT)

Ingredient_portion(id: INT [PK], user_id: INT [FK to Users.user_id], ingredient_id: INT [FK to ingredients.ingredient_id], ingredeint_amount: FLOAT, ingredient_unit:VARCHAR(255))

User_restrictions(id: INT [PK], user_id: INT [FK to Users.user_id], ingredient_id: INT [FK to ingredients.ingredient_id])