1. **Name of project:**

Recipes Database

2. **Name of team:**

Team 7 on Canvas

3. **Team members:**

Dawn Lu, Jake Gilden, Ryan Duong, Sammi Chen (all in section 3)

4. **Description of project:**

We created a database that stores recipes, the user that owns the recipe, and the ingredients used for each recipe. We also created a user interface so the users can easily add new recipes and ingredients to the database. This could be used by restaurants and chefs to keep a database of various types of recipes and easily share them with others.

5. **Link to UML diagram:** [Here](https://drive.google.com/file/d/1lhu5imBE9k_YP45EgdIJ3zB-VKQ9UIvD/view?usp=sharing)

6. **Description of user data model:**

The user data model is a table called users containing all our users, which represents the owner of each recipe. The fields we have in the user data model are user\_id (int)(pk), firstname (string), lastname (string), username (string), password (string), email (string), date\_of\_birth (date).

7. **Description of 2 domain object data models:**

Our first domain object data model is a recipe table called recipes that includes information about each recipe. This includes recipe\_id (pk), name (string), instructions (string), prep\_time (int), cook\_time (int), serving \_size (int), dietary\_restriction (enumeration), user\_id (fk). We use user\_id as a foreign key for the relationship between the user table and recipe table. We also created a portable enumeration for dietary\_restriction to limit the options that can be entered.

Our second domain object data model is a table called ingredients that contains all the ingredients for our recipes. This includes ingredient\_id (pk) and ingredient (string). This table has a many to many relationship with recipes. We implemented that relationship by creating a table called recipe\_ingredient and created a many-to-one relationship with recipes and recipe\_ingredient and ingredients and recipe\_ingredient.

8. **Description of user to domain object relationship:**

Our user to domain object relationship is the one-to-many relationship between our users table and recipes table. This relationship is achieved by the recipes table using user\_id as a foreign key to the users table. The relationship also uses on delete cascade, so if a user is deleted from the database, then the recipes they own will also be removed.

9. **Description of domain object to domain object relationship:**

Our domain object to domain object relationship is the many-to-many relationship between the recipes table and ingredients table. Multiple ingredients can be used in multiple recipes, so this relationship is valid. We implemented this relationship by creating a recipe\_ingredient table and a many-to-one relationship between recipes and recipe\_ingredient and ingredients and recipe\_ingredient. The recipe\_ingredient table includes recipe\_ingredient (int)(pk), recipe\_id (int)(fk), ingredient\_id (int)(fk), unit (string), quantity(int), and prep\_method(string).

10. **Description of portable enumeration:** Our portable enumeration is the dietary restriction for each recipe. We created the enumeration by implementing a dietary\_restriction table with the following options: vegetarian, vegan, pescatarian, nut free, and no restriction. This means that when the dietary\_restriction field is being filled out for the recipe table it must be one of those options. We decided on dietary restrictions as our enumeration because there are a fixed number of options you can choose from.

11. **Description of the user interface requirements:**

* **user-list**
  + displays a list of all the users with their first name, last name, username, and a link to the form editor for a specific user id
  + A button that allows you to create a new user
* **user-form-editor**
  + displays the fields for a specific user and allows someone to edit the fields and create or delete a user
  + displays links to the recipe editor for the recipes associated with the specific user
* **recipe-list**
  + displays a list of all the recipes with the recipe name and a link to the form editor for the specific recipe id
  + A button that allows you to add a new recipe to the database
* **recipe-form-editor**
  + displays the fields for a specific recipe and allows someone to edit the fields and create or delete a recipe
* **ingredients-list**
  + displays a list of all the ingredients with the name and a link to the form editor for the specific ingredient id
  + A button that allows you to add a new ingredient to the database
* **ingredients-form-editor**
  + displays the fields for a specific ingredient and allows someone to edit the fields and create or delete an ingredient

**Problem Statement**

The problem we are trying to solve is a restaurant or chefs trying to keep track of all the recipes they make for customers. It can also be a way for people to easily share recipes with each other or for users to explore completely new recipes. Restaurants can have multiple locations with different chefs so sharing the recipes using a database can be very useful for them. They are constantly creating, editing, and removing recipes from their menu so this allows them to stay more organized.

**Solution Statement**

We first created a database using SQL containing tables for a user object (user or chefs) and two domain objects (the recipes and ingredients). We then created a one-to-many relationship between the user table and recipe table. We also had a many-to-many relationship between the recipe table and ingredients table.

Next, we programmed the database using Object Relational Mapping (ORM) in IntelliJ. We created model classes and Dao classes for each object, so we could implement CRUD (create, read, update, delete) operations for the database.

Lastly, we created a user interface so a user could utilize any of the CRUD operations. We created a list and edit screen for the user, recipe, and ingredients table. This allows anyone to create, edit, update, or have a list of the values in our database.

**User**

A typical user for our solution would be someone who manages a restaurant or a chef. It would be someone who is looking to store data about recipes and the ingredients used for each recipe. A chef can use it to train other members working in the kitchen. If the restaurant has multiple locations, it can be used by the managers to easily share recipes across the organization. Our solution also allows the user to easily create, edit, or delete recipes from the database.

**Domain Objects**

Our first domain object data model is a recipe table called recipes that includes information about each recipe. This includes recipe\_id (pk), name (string), instructions (string), prep\_time (int), cook\_time (int), serving \_size (int), dietary\_restriction (enumeration), user\_id (fk). We use user\_id as a foreign key for the relationship between the user table and recipe table. We also created a portable enumeration for dietary\_restriction to limit the options that could be entered.

Our second domain object data model is a table called ingredients that contains all the ingredients for our recipes. This includes ingredient\_id (pk) and ingredient (string). This table has a many to many relationship with recipes. We implemented that relationship by creating a table called recipe\_ingredient and created two many- to-one relationships with recipes and recipe\_ingredient and ingredients and recipe\_ingredient.