**SW Engineering CSC648/848**

**Project name: Zesty**

**Section 04, Team 05**

| **Name** | **Role** |
| --- | --- |
| Xu Gu | Team Leader |
| Pasang Sherpa | Back-End Lead |
| Raymond Liu | Git Master |
| Dante Vercelli | Back-End Lead |
| Yonatan Leake | Scrum master |
| Junghyun Song (Katie) | Front-End Lead |
| Ruxue Jin | Front-End Lead |

**Milestone 2**

**Due day: 3/20/2024**

| History Table | | |
| --- | --- | --- |
| Version | Date | feedback |
| 1.0 | Mar 12 |  |

# Data Definitions V2

**User Entity -** An entity that stores information about registered user accounts

* userId - user’s globally unique id, assigned to a user by system
* userName - user-defined name, illustrated on the webpage
* email - user’s email, used for login, should be globally unique
* password - password for user, used for login, and will be hashed
* userType - code of enumerate type, 0 - system, 1 - admin, 2 - test, 3 - normal user
* createTime - creating time of this record
* modifyTime - modifying time of this record

**Follow Entity -** An entity that stores following relationship between 2 users

* followId - unique id of a relationship record
* followerId - follower’s user id
* followeeId - followee’s user id
* status - code of enumerate type, 0 - active, 1 - inactive
* createTime - creating time of this record
* modifyTime - modifying time of this record

**Recipe Entity -** An entity that stores information of recipe

* recipeId - recipe’s globally unique id, assigned to a recipe by system
* creatorId - user id of the recipe creator
* recipeName - recipe’s name
* recipeType - code of enumerate type, 0 - system-inclusion, 1 - user-defined
* categories - a list of recipe categories, can include occasion, flavor, preference, allergy, region
* difficulty - difficulty rating of making this meal
* status - code of enumerate type, 0 - created, 1 - posted, 2 - unposted
* postTime - posting time of recipe
* steps - a json formatted string containing recipe’s image, video, steps
* cookingTime - minutes required to make that meal
* createTime - creating time of this record
* modifyTime - modifying time of this record

**Rating** Entity - User’s rating on a recipe

* ratingId - rating’s globally unique id, assigned to a favorite by system
* userId - id of a user who favorite a recipe
* recipeId - id of a recipe favorited by a user
* score - 0 - 5
* createTime - creating time of this record

**Favorite** Entity - User’s favorite recipe

* favoriteId - favorite’s globally unique id, assigned to a favorite by system
* userId - id of a user who favorite a recipe
* recipeId - id of a recipe favorited by a user
* status - 0 - active, 1 - inactive
* createTime - creating time of this record
* modifyTime - modifying time of this record

**Review** Entity - User’s review on a recipe

* reviewId - review’s globally unique id, assigned to a favorite by system
* userId - id of a user who comments a recipe
* recipeId - id of a recipe commented by a user
* comment - user’s comment on a recipe
* rating - user’s rating on a recipe, from 0 to 5 points
* status - 0 - active, 1 - inactive
* createTime - creating time of this record
* modifyTime - modifying time of this record

**Ingredient** Entity - An entity that stores information of ingredient

* ingredientId - ingredient’s globally unique id, assigned to a ingredient by system
* ingredientName - ingredient’s name
* category - 0 - basic, 1 - vegetable, 2 - meat, 3 - condiment, 4 - others
* unit - ingredient’s unit
* createTime - creating time of this record
* modifyTime - modifying time of this record

**RecipeIngredient** Entity - An entity that store the relationship of a recipe and an ingredient that the recipe need

* recipeIngredientId - globally unique id
* recipeId - recipe’s id
* ingredientId - id of a required ingredient
* amount - amount of the required ingredient to make the meal
* createTime - creating time of this record
* modifyTime - modifying time of this record
* nutrition - nutrition info for this ingredient

**UserIngredient** Entity - An entity that store the relationship of a user and an ingredient that the user have

* userIngredientId - globally unique id
* userId - user’s id
* ingredientId - id of a ingredient that the user have
* amount - amount of the required ingredient to make the meal
* createTime - creating time of this record
* modifyTime - modifying time of this record

**MealPlan** Entity - An entity that store a user’s meal plan for a period

* mealPlanId - globally unique id
* userId - user’s id
* recipeIds - id list of recipes the user add
* startTime - start time of the meal plan
* endTime - end time of the meal plan
* createTime - creating time of this record
* modifyTime - modifying time of this record

# 

# Functional Requirements V2

Requirement priorities are represented by 1(must-have) ,2 (desired), 3 (opportunistic)

Requirement IDs are represented by 8-digit numeric values, where digit 1-3 represent class number 648, 4-5 represent group number 05, 6-8 are requirement numbers

**Requirement 1 (ID: 64805001): Search recipes with user input**

1.1. Develop a search bar that allows users to input the recipe features including recipes name, ingredients, region, difficulty level and returns related recipes.

1.2. Ensure the search results are returned quickly, optimizing the database queries for performance.

Priority: 1 (must-have)

**Requirement 2 (ID: 64805002): Access to various cultural recipes**

2.1. Provide a “cuisines” category that includes some pre-defined regions for use to select

2.2 Allow user to set the region of recipe when creating a recipe

2.3. Users can search recipes with regions

Priority: 1 (must-have)

**Requirement 3 (ID: 64805003): Detailed recipe information**

3.1. Ensure each recipe entry includes a list of required ingredients with type and amount.

3.2. Provide clear cooking steps and estimated preparation and cooking time for each recipe.

Priority: 1 (must-have)

**Requirement 4 (ID: 64805004): Indicate the difficulty level of recipes**

4.1.Provide a “level” category that includes various difficult levels for use to select

4.2 Allow user to set the recipe difficulty when creating a recipe

4.3. Users can search recipes with difficulty levels

Priority: 1 (must-have)

**Requirement 5 (ID: 64805005): Nutrition information in recipes**

5.1. Include detailed nutrition information for each recipe, such as calories, protein, and vitamins.

5.2. Calculate the nutrition of each meal automatically without user input.

Priority: 1 (must-have)

**Requirement 6 (ID: 64805006): Comprehensive user account functionality**

6.1. Enable users to edit their profile information, including dietary preferences and allergies.

6.2. Allow users to manage a virtual pantry with ingredients they currently have.

6.3. Provide features for users to post, edit, and manage their recipes.

6.4. Allow users to manage their following list.

Priority: 1 (must-have)

**Requirement 7 (ID: 64805007): Community engagement through recipes**

7.1. Develop a section for users to post and browse recipes.

7.2. Implement commenting and liking features for community interaction.

Priority: 1 (must-have)

**Requirement 8 (ID: 64805008): Recipe popularity ranking**

8.1. Create a dynamic ranking system based on recipe views, likes, and user ratings.

8.2. Display a leaderboard or trending section featuring the most popular recipes.

Priority: 1 (must-have)

**Requirement 9 (ID: 64805009): Recipe recommendations**

9.1. Develop a recommendation engine that suggests recipes based on user preferences, dietary restrictions, and past interactions.

9.2. Include an option for users to receive personalized recipe suggestions.

Priority: 2 (desired)

**Requirement 10 (ID: 64805010): User-generated meal plans**

10.1. Allow users to create and customize meal plans, selecting recipes for specific days and meals.

10.2. Enable users to save and modify their meal plans.

Priority: 3 (opportunistic)

**Requirement 11 (ID: 64805011): Recipe rating system**

11.1. Implement a rating system where users can rate recipes on a scale (e.g., 1 to 5 stars).

11.2. Allow users to leave reviews or comments along with their ratings.

Priority: 1 (must-have)

**Requirement 12 (ID: 64805012): Sharing recipes and posts**

12.1. Enable sharing functionality for recipes and blog posts to external platforms (e.g., social media, email).

12.2. Ensure that shared content includes a link back to the recipe or post on the website.

Priority: 2 (desired)

**Requirement 13 (ID: 64805013): Recipe creation template**

13.1. Provide a standardized template for users to create and submit their recipes, ensuring consistency in the presentation.

13.2. Include fields for ingredients, measurements, steps, cooking time, and photos in the template.

Priority: 1 (must-have)

**Requirement 14 (ID: 64805014): Nutrition-based recipe search**

14.1. Implement a search feature that allows users to find recipes based on specific nutritional parameters (e.g., calorie count, protein content).

14.2. Ensure accurate and up-to-date nutrition data for all recipes.

Priority: 3 (opportunistic)

**Requirement 15 (ID: 64805015): Meal plan with aggregated information**

15.1. Display aggregate nutritional information and total cooking time for recipes included in a meal plan.

15.2. Provide a summary view of the meal plan with key nutritional highlights and time requirements.

Priority: 3 (opportunistic)

**Requirement 16 (ID: 64805016): Missing ingredient list for meal plans**

16.1. Generate a list of missing ingredients needed for the selected meal plan based on the user's virtual pantry.

16.2. Offer an option to print or export the shopping list.

Priority: 3 (opportunistic)

**Requirement 17 (ID: 64805017): Recipes based on available ingredients**

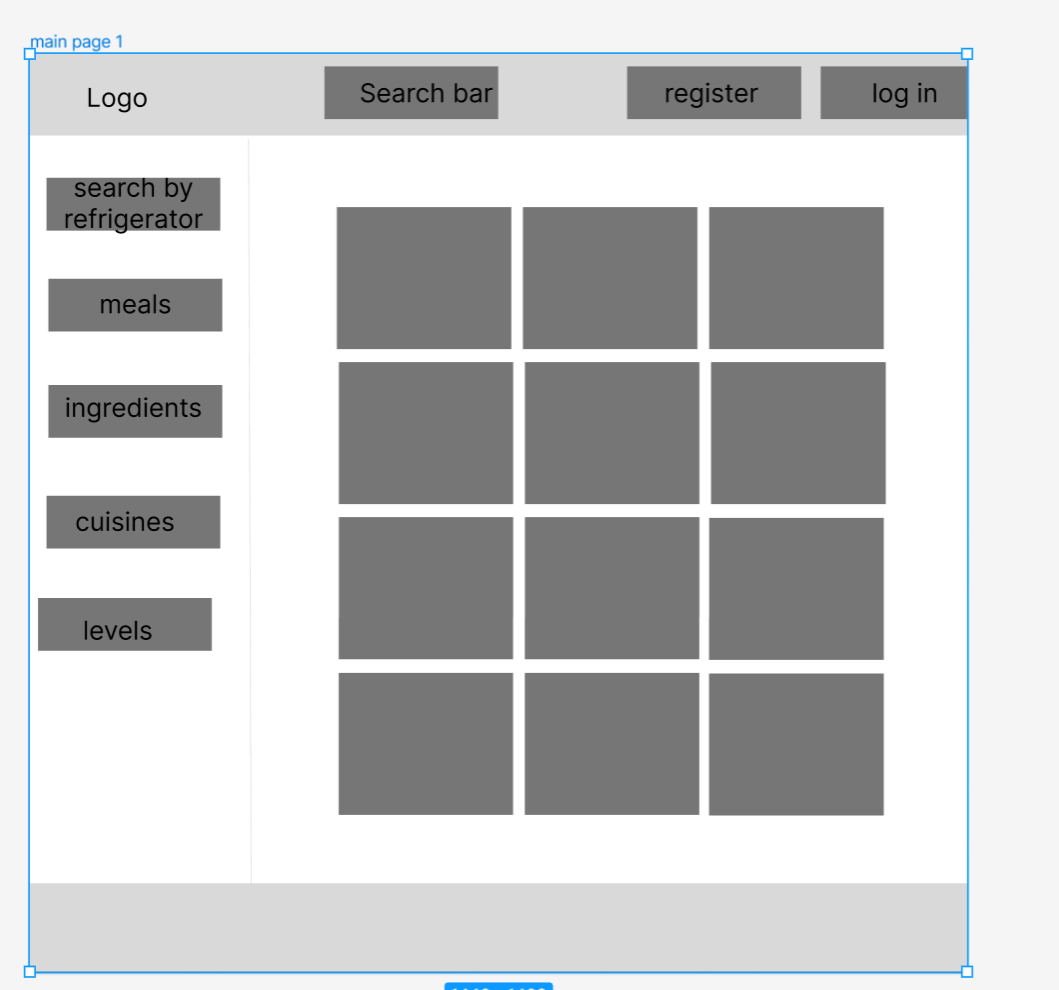
17.1. Create a feature allowing users to input their available ingredients and receive recipe suggestions that utilize those ingredients, prioritizing recipes that require the fewest additional ingredients.

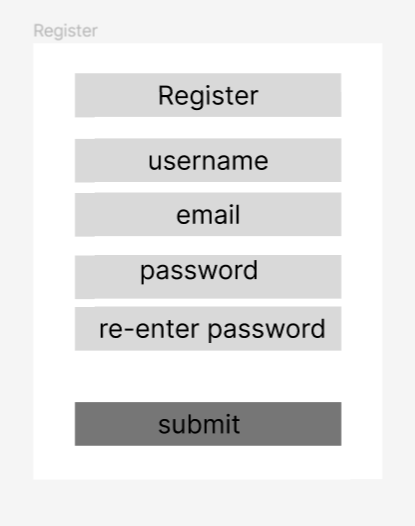
17.2. Implement an "ingredient match" percentage to show how well the user's available ingredients align with the recipe requirements.

Priority: 1 (must-have)

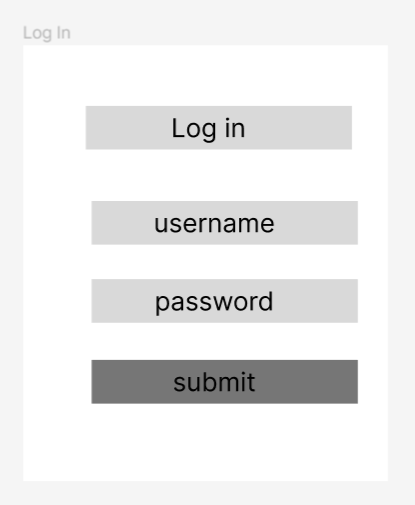
3. UI Mockups and UX Flows

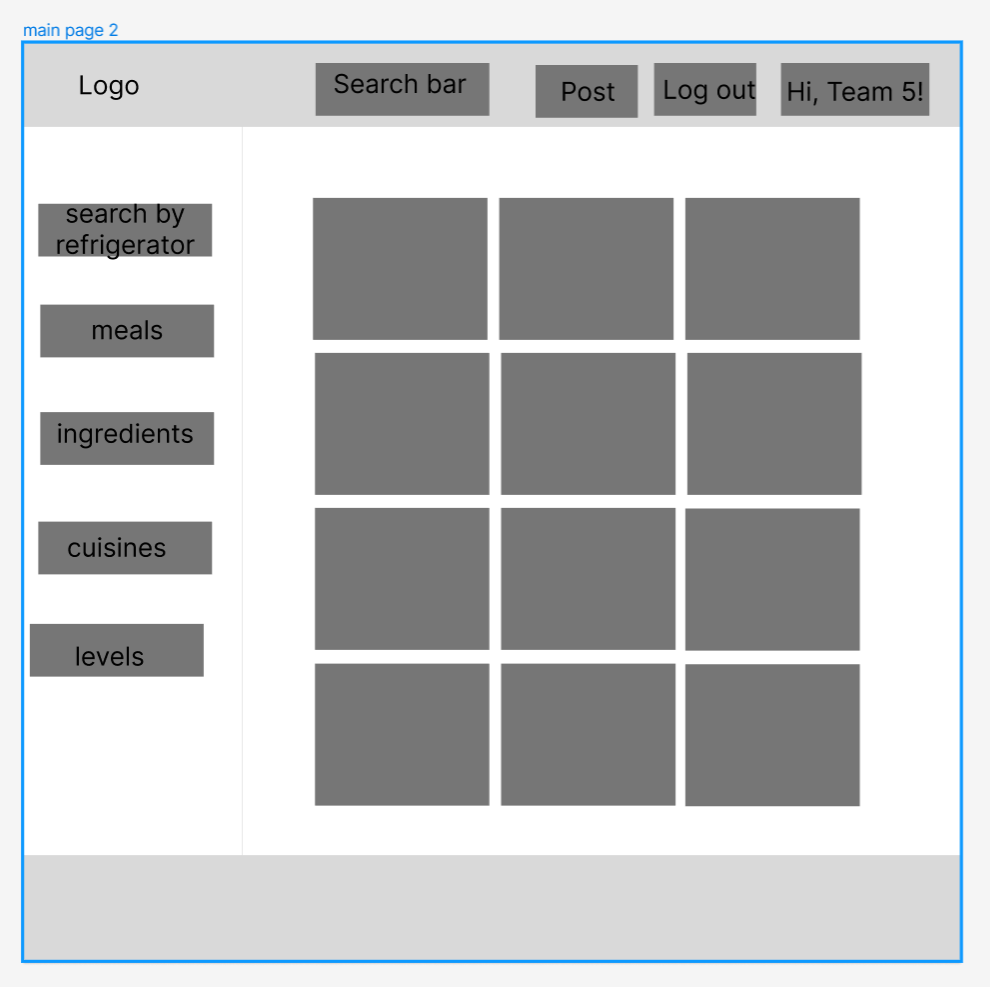
Page 1: Main page before user login

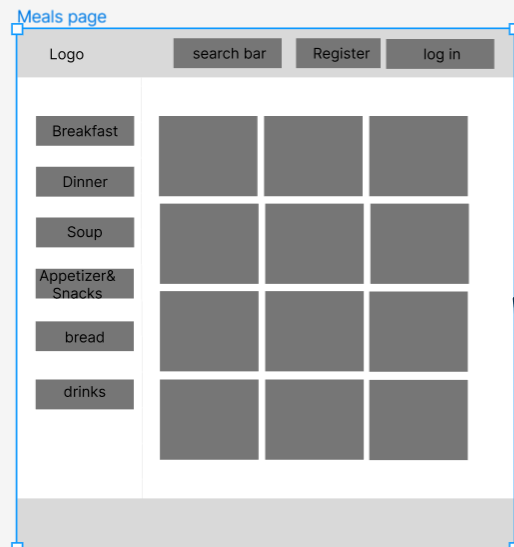


Page 2: When a new user clicks “Register” on page 1, trying to register.

Page 3: When an existing user clicks “login” on page 1, trying to register.



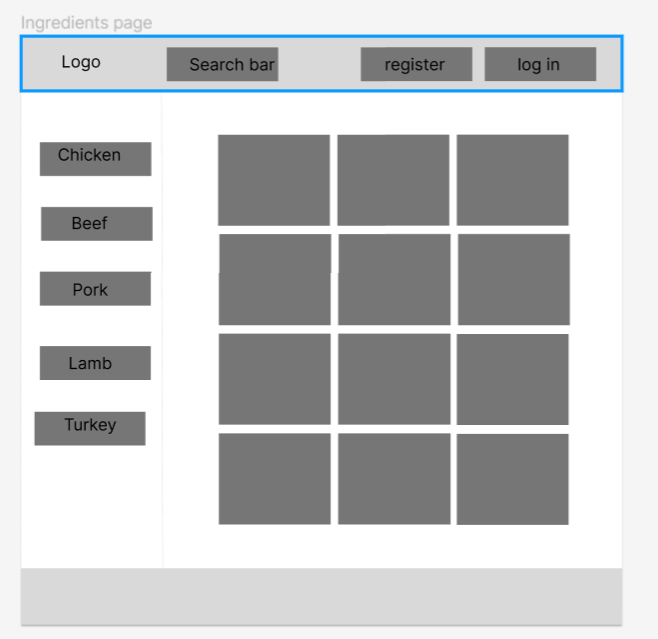
Page 4. When user clicks “submit”successfully log in 

5. When user click “meals” button in page 4

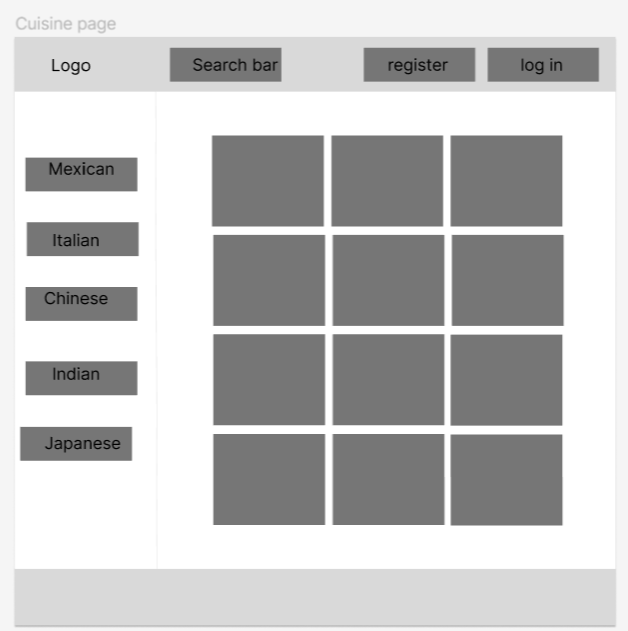
6. When the user clicks the “Breakfast button” in page 5, shows all the valid recipes.



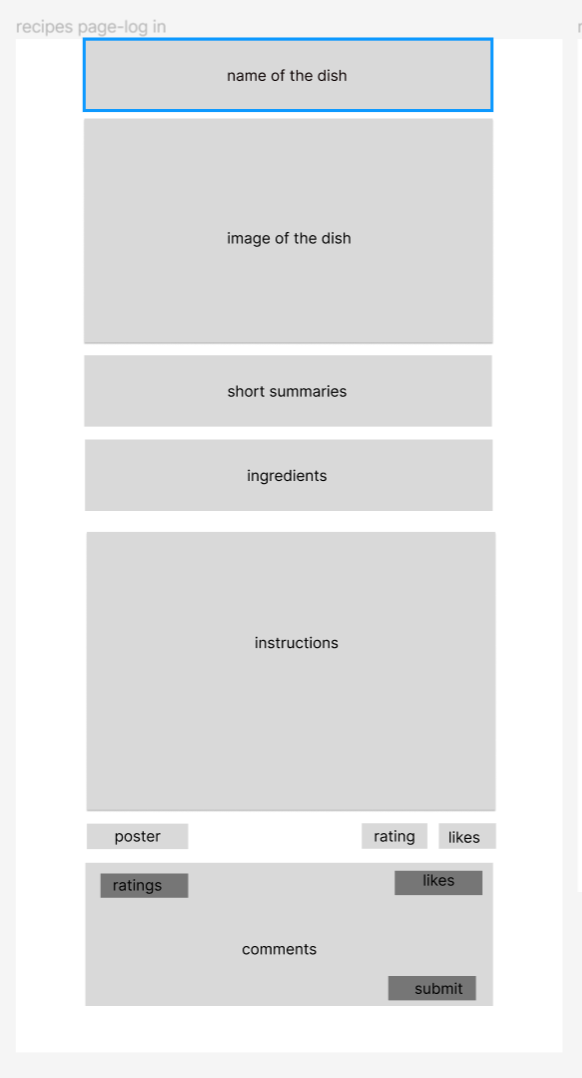
7. When the user clicks the “Ingredients” button in page 4



8. When the user clicks “Cuisine” button in page 4

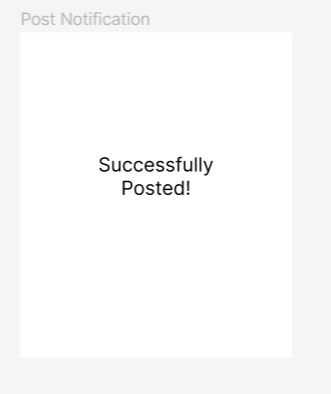


9.. When user clicks “level”button in the page 4

10. When the user clicks the recipe cards in page 1, 3-9, 

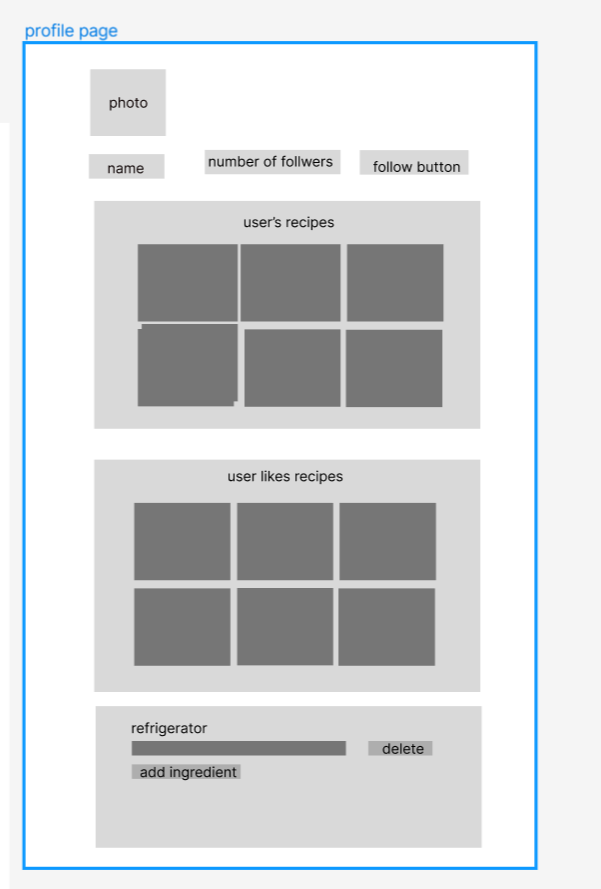
11. If the user clicks “submit” without log in, go to that log in page 3.

12. If the user clicks “submit” successfully,

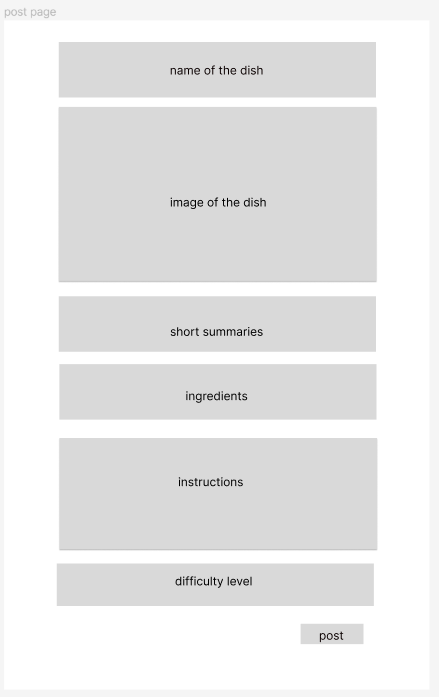


Still remain on page 10.

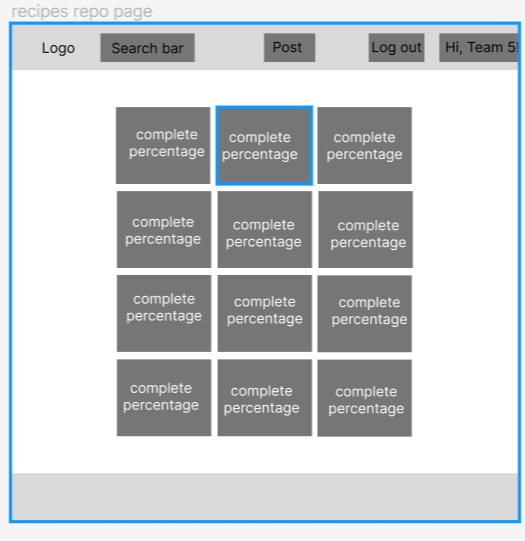
13. If user clicks “Profile” button in page 4



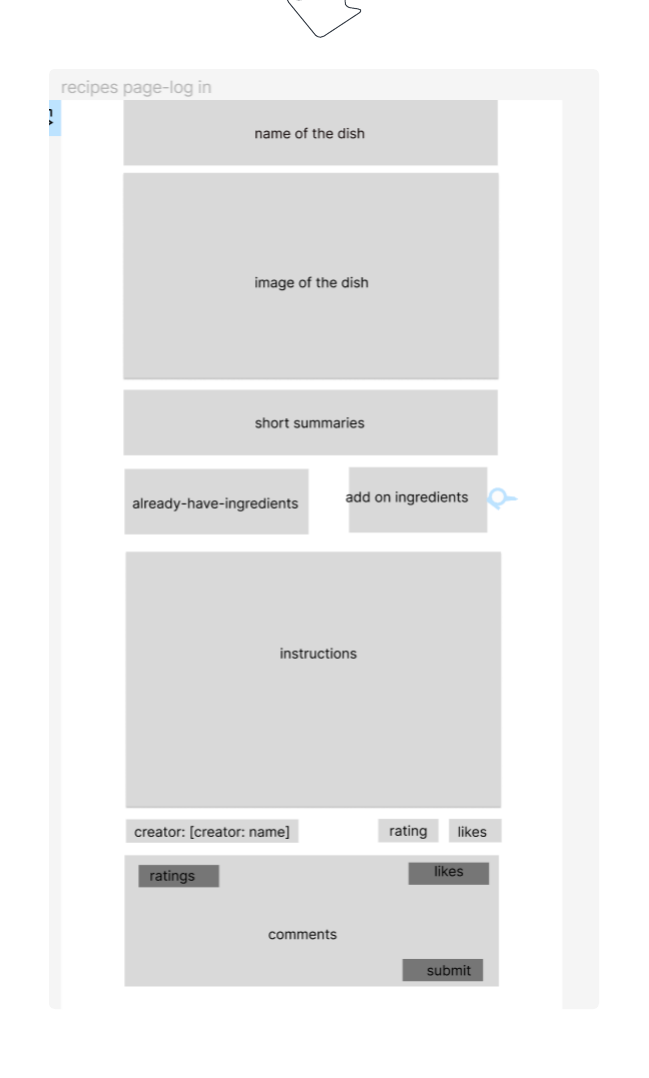
14. If the user clicks “Post” button in the page 4



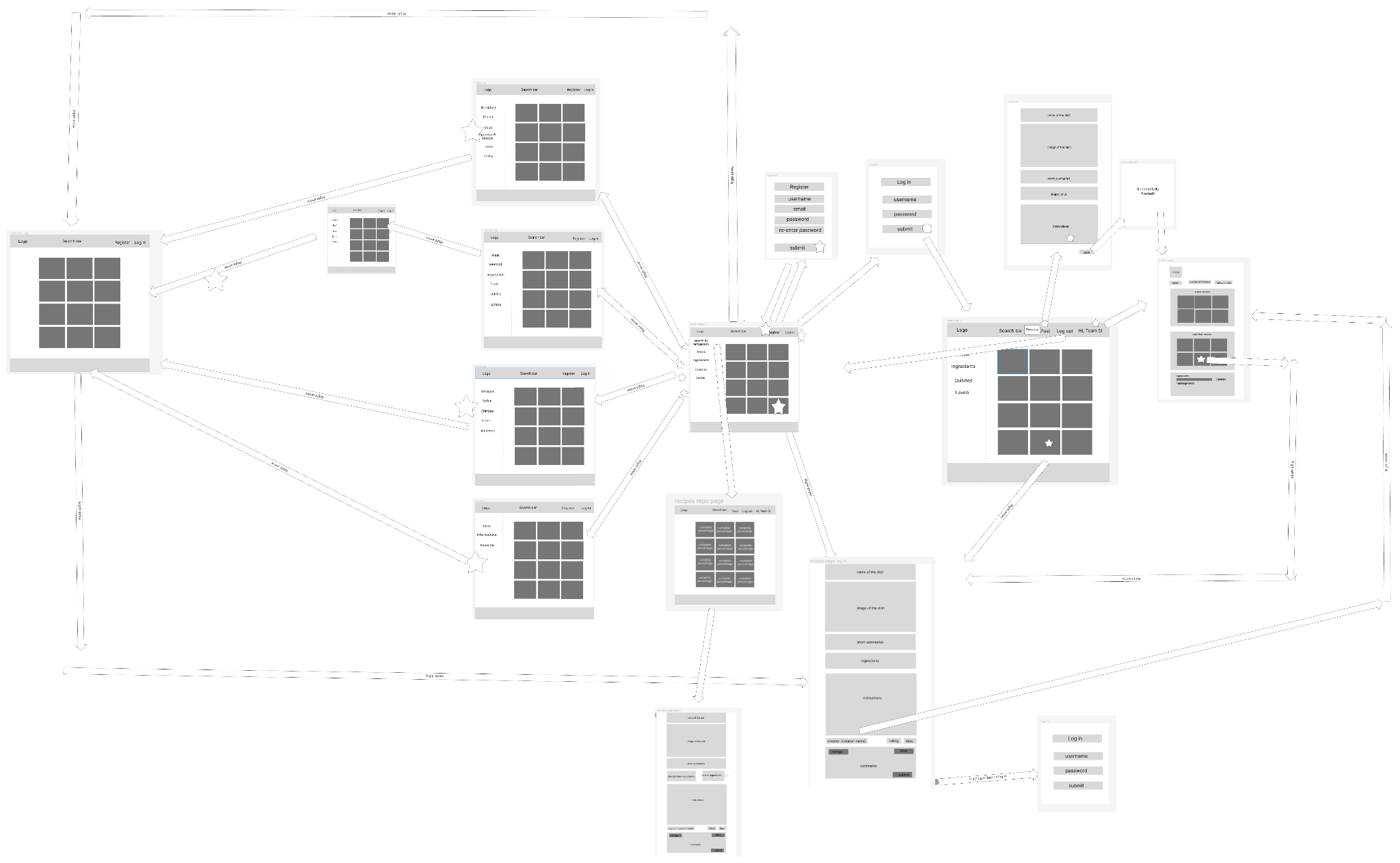
15. When the use clicks “search by refrigerator”



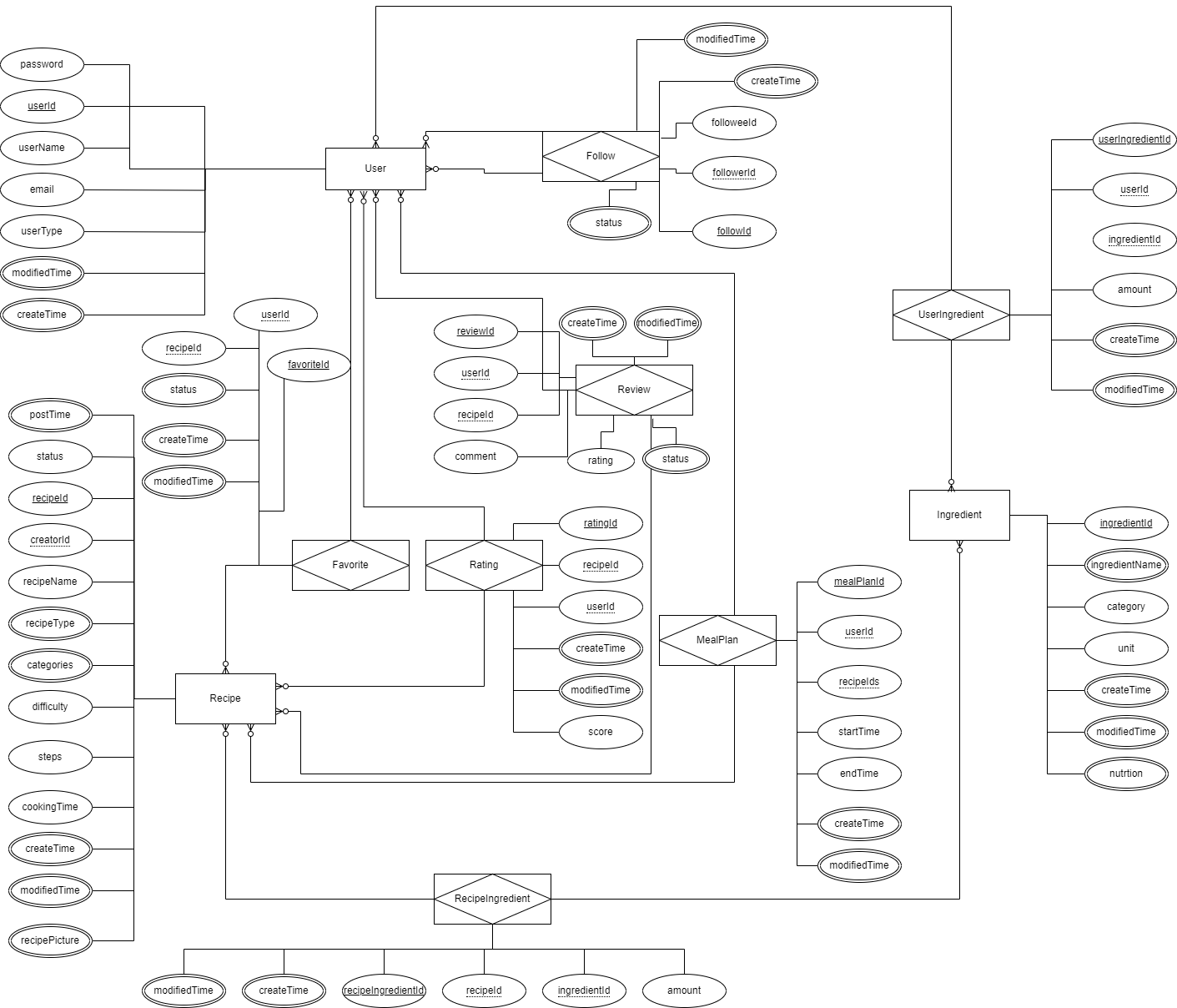
16. If the user clicks one recipes with complete percentage



UI Flow



4. High level Architecture, Database Organization



**SQL DB Tables**

**User: ADD DELETE SEARCH UPDATE**

* userId INT
* userName VARCHAR(45)
* email VARCHAR(45)
* password VARCHAR(45)
* userType INT
* createTime DATETIME
* modifyTime DATETIME

**Recipe: ADD DELETE SEARCH UPDATE**

* recipeId INT
* creatorId INT
* recipeName VARCHAR(45)
* recipeType INT
* category INT
* difficulty DECIMAL(12,2)
* status INT
* postTime DATETIME
* steps JSON
* cookingTime VARCHAR(45)
* createTime DATETIME
* modifyTime DATETIME
* recipePicture BLOB

**RecipeIngredient: ADD DELETE SEARCH**

* recipeIngredientId INT
* ingredientId INT
* recipeId INT
* amount DECIMAL(12,2)
* createTime DATETIME
* modifyTime DATETIME

**Ingredient: ADD DELETE SEARCH**

* ingredientId INT
* ingredientName VARCHAR(45)
* category INT
* unit VARCHAR(45)
* createTime DATETIME
* modifyTime DATETIME
* nutrition VARCHAR(45)

**UserIngredient: ADD DELETE UPDATE SEARCH**

* userIngredientId INT
* userId INT
* ingredientId INT
* amount DECIMAL(12,2)
* createTime DATETIME
* modifyTime DATETIME

**MealPlan: ADD DELETE UPDATE SEARCH**

* mealPlanId INT
* userId INT
* recipeId INT
* startTime DATETIME
* endTime DATETIME
* createTime DATETIME
* modifyTime DATETIME

**Favorite: ADD DELETE SEARCH UPDATE**

* favoriteId INT
* userId INT
* recipeId INT
* status INT
* createTime DATETIME
* modifyTime DATETIME

**Follow: ADD DELETE SEARCH UPDATE**

* followId INT
* followeeId INT
* followerId INT
* status INT
* modifyTime DATETIME
* createTime DATETIME

**Review: ADD DELETE UPDATE SEARCH**

* reviewId INT
* userId INT
* recipeId INT
* comment VARCHAR(255)
* rating INT
* status INT
* createTime DATETIME
* modifyTime DATETIME

**Rating: ADD SEARCH DELETE**

* ratingId INT
* userId INT
* recipeId INT
* score TINYINT(1)
* createTime DATETIME
* modifyTime DATETIME

**Add/Delete/Search architecture:**

**User Entity:**

Add: Create new user records upon registration with userId, userName, email, password, userType, createTime, and modifiedTime.

Search: Look up users by userName or email for login purposes or profile searches.

Delete: Remove user records if an account is closed.

Update: Change userName, email, or password when a user updates their profile; update userType for role changes; update modifiedTime on each change.

**Follow Entity:**

Add: Establish a new follow relationship between two users with followId, followerId, followeeId, status, createTime, and modifiedTime.

Search: Query follow relationships to display followers/followees for a user.

Delete: Remove a follow relationship if a user unfollows another user.

Update: Change status to inactive and update modifiedTime when a follow is deactivated.

**Recipe Entity:**

Add: Insert new recipes with details such as recipeId, creatorId, recipeName, recipeType, categories, difficulty, rating, status, postTime, steps, cookingTime, createTime, and modifiedTime.

Search: Retrieve recipes based on different criteria like recipeName, categories, difficulty, or rating.

Delete: Delete recipes that are removed by the creator or admin.

Update: Modify recipe details and update modifiedTime when changes are made.

**Rating Entity:**

Add: Input new ratings for a recipe with ratingId, userId, recipeId, score, and createTime.

Search: Fetch ratings to calculate the average rating for a recipe.

Delete: Erase a rating if a user decides to withdraw their rating.

**Favorite Entity:**

Add: Create a record when a user favorites a recipe with favoriteId, userId, recipeId, status, createTime, and modifiedTime.

Search: Look up all favorite recipes of a user.

Delete: Remove a recipe from a user's favorites.

Update: Set status to inactive when a user unfavorites a recipe and update modifiedTime.

**Review Entity:**

Add: Post a new review with reviewId, userId, recipeId, comment, rating, status, createTime, and modifiedTime.

Search: Display reviews for a recipe.

Delete: Remove a review if the user deletes it or if it's flagged as inappropriate.

Update: Edit a review and update modifiedTime when changes are made.

**Ingredient Entity:**

Add: Insert new ingredient information with ingredientId, ingredientName, category, unit, createTime, and modifiedTime.

Search: Retrieve ingredients for recipe creation and editing.

Delete: Remove ingredients no longer used or available.

**RecipeIngredient Entity:**

Add: Associate ingredients with a recipe using recipeIngredientId, recipeId, ingredientId, amount, createTime, modifiedTime, and nutrition.

Search: Fetch ingredients for a specific recipe.

Delete: Disassociate an ingredient from a recipe.

**UserIngredient Entity:**

Add: Add ingredients to a user's pantry with userIngredientId, userId, ingredientId, amount, createTime, and modifiedTime.

Search: Display ingredients a user has in their virtual pantry.

Delete: Remove ingredients from a user's pantry.

Update: Adjust amount when a user uses some ingredients or adds more to their pantry.

**MealPlan Entity:**

Add: Create a meal plan for a user with mealPlanId, userId, recipeIds, startTime, endTime, createTime, and modifiedTime.

Search: Display the meal plan for a user.

Delete: Remove a meal plan if a user decides to discard it.

Update: Modify a meal plan when a user makes changes to it.

**APIs**

**Backend endpoint APIs**

**Recipe APIs:**

**Get Recipes previews randomly**

Endpoint: GET /api/recipe/random

Description: get random recipe previews to fill the page when a user visit our website

Response: a json response including an array of recipe previews that match the filters, array length

**Get Recipes previews from user input**

Endpoint: GET /api/recipe/search?q={keywords}

Description: Retrieves recipes filtered by meal type, ingredient, region, or difficulty level.

Response: a json response including an array of recipe previews that match the filters, array length

**Get Recipes preview with specific type**

Endpoint: GET /api/recipe/preview?meal={}&ingredient={}&cuisine={}&level={}

Description: get a list of recipe previews with specific type

Response: a json response including an array of recipe previews that match the filters, array length

**Get Recipe Details by recipe id**

Endpoint: GET /api/recipe/{recipeId}

Description: Retrieves detailed information for a specific recipe.

Response: Detailed information of a recipe, including ingredients, cooking steps, and user ratings.

**Create Recipe**

Endpoint: POST /api/recipe

Description: Allows a user to create a new recipe.

Request: Recipe details including name, description, ingredients, steps, meal type, region, difficulty level, etc.

Response: Confirmation of creation with the newly created recipe details.

**Update Recipe**

Endpoint: PUT /api/recipe/{recipeId}

Description: Updates information for an existing recipe.

Request: Updated recipe details.

Response: Confirmation of update with the updated recipe details.

**Delete Recipe**

Endpoint: DELETE /api/recipe/{recipeId}

Description: Deletes a specific recipe.

Response: Confirmation of deletion.

**User Authentication APIs:**

**User Registration**

Endpoint: POST /api/user/register

Description: Registers a new user.

Request: User details including username, email, and password.

Response: Confirmation of registration with user details.

**User Login**

Endpoint: POST /api/user/login

Description: Authenticates a user.

Request: Username and password.

Response: Authentication token and user details.

**Delete User Account**

Endpoint: POST /api/user/delete

Description: delete a user by id.

Request: userId.

Response: Confirmation code and message.

**User Info**

**User basic profile**

Endpoint: GET /api/user/{userid}/basic

Description: get a user’s basic information including username, email, age, etc

Response: a json array with user basic info

**User following list**

Endpoint: GET /api/user/{userid}/following

Description: get a user’s following list

Response: a json array including followings

**User follower list**

Endpoint: GET /api/user/{userid}/follower

Description: get a user’s follower list

Response: a json array including followers

**User favorite recipe list**

Endpoint: GET /api/user/{userid}/recipes

Description: get a user’s favorite recipes

Response: a json array with user’s favorite recipes

**User Interaction APIs:**

**Follow Another User**

Endpoint: POST /api/user/follow

Description: Allows a user to follow another user.

Request: userId and follower’s user id.

Response: Confirmation of the action.

**Favorite a Recipe**

Endpoint: POST /api/recipe//favorite

Description: Allows a user to favorite a recipe.

Request: userId and recipe id.

Response: Confirmation of the action.

**Rate a Recipe**

Endpoint: POST /api/recipe/rate

Description: Allows a user to rate a recipe.

Request: userId, recipe id, rating points

Response: Updated average rating of the recipe.

**Comment on a Recipe**

Endpoint: POST /api/recipe/comment

Description: Allows users to post comments on a recipe.

Request: userId, recipe id, comment text.

Response: Confirmation of the posted comment.

**3rd party APIs**

FoodData Central (FDC) API:

Description: It is intended primarily to assist application developers wishing to incorporate nutrient data into their applications or websites.

API doc: <https://app.swaggerhub.com/apis/fdcnal/food-data_central_api/1.0.1#/info>

5 .Identify actual key risks for your project at this time

RISKS:

* Recommendations- we are thinking about ways to implement a recommendation algorithm however we do not have a lot of experience working with these types of algorithms including a ML approach to recommendations and collaborative filtering. We’re now doing some research on this problem and find out what 3rd-party APIs we can use.
* Nutrition - the main problem with adding nutrition info to our app is how we will get the nutrition info. One approach is we get the nutrition info for each ingredient and for a recipe the total nutrition info will be the sum of all the nutrition info of each ingredient. An api we can use for this approach is <https://fdc.nal.usda.gov/api-guide.html>. We are putting this as a risk for now because we are not confident we will be able to implement this feature yet.
* Skills risks and mitigation plan - We have a study plan for the frontend group and the backend group. The frontend is learning React and bootstrap while the backend is learning Python flask and MySQL. However we don’t have a person who’s experienced in frontend technologies so frontend development could be a little bit hard. To solve this, we’ll collect the hard points in frontend development and then launch meeting to solve those problems together.
* Schedule and Teamwork risks: The attendance is not satisfying in the past two weeks, mainly because the tasks were not assigned to everyone clearly. To solve this, our scrum master is working on creating task tracing using Notion/Jira.
* legal/content risks: Using others’ recipes data may cause copyright issues. We need to do more research on the recipe data to find out data we can legally use, and to better understand the laws we have to navigate.
* Also, the pace on the project/classes has started to catch up. Having spring break should help ease the load.

6. Project management

On M2’s meetings, our front end group shared their UI design with us and clarify each figure they

design and the whole flow. Then the rest of our team gave some suggestions on that. For backend

group, we shared the database design and API design at each meeting.

Starting on M3, we will be using Notion to help organize all of our work and keep it so we are all on

the same page. During Scrum Meetings on Wednesdays, we have started to work in smaller groups to

help maximize our team productivity, and we discuss our work at the beginning and end of each meeting.

On our outside of class meetings, we discuss over an hour-long session where everyone is at and what

needs to be finished and by what members.