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 3

Due day: 4/10/2024

History Table		
Version	Date	feedback
1.0	April 8	Good UI. Continue to work on unique feature

Appendix I – Rubrics and checklist for Part 1 Milestone 3 review: Project Status and UI Review.

Section: 04 Team: 05 Date: 4/10
Number of students present: 6

1. UI and functionality feedback (P1 functions only)

- Instructors comments on UI/functionality during demo

1) UI design is good
2) unique feature needs to be implemented
3) Front-end team and back-end team continue work simutaneously.

- Our plan for the comments

Front-end:
1) Fix bug in create-page bug:
   01. missing instruction steps in recipe
   02. difficulty level selector not working
   03. Upload image not working
2) Implement profile page, including unique feature “Refrigerator”
3) Implement nav-bar

Back-end:
1) Implement API in profile page:
   01. Get user’s info
   02. Get user’s posted recipes
   03. Get user’s liked recipes
   04. Search by ingredients in refrigerator

Whole project:
1) Code Structure and File Structure
   01. Back-end: routes should have one input → the data will be extracted inside the service function
   02. Front-end: currently each tsx file connects with its own css file. Styles should be stored in a css file.

2) Consider boundary conditions → what do we do if someone searches something that is not in the db how do we handle it?
3) Continue in-person meetup on Monday and Wednesday

2. List of P1 features committed for delivery

Requirement 1 (ID: 64805001): Search recipes
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.

Requirement 2 (ID: 64805002): Access to various cultural recipes
2.1. Provide a “cuisines” category that includes some predefined 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

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.

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

Requirement 5 (ID: 64805005): 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.

Requirement 6 (ID: 64805006): 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.

Requirement 7 (ID: 64805007): 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.

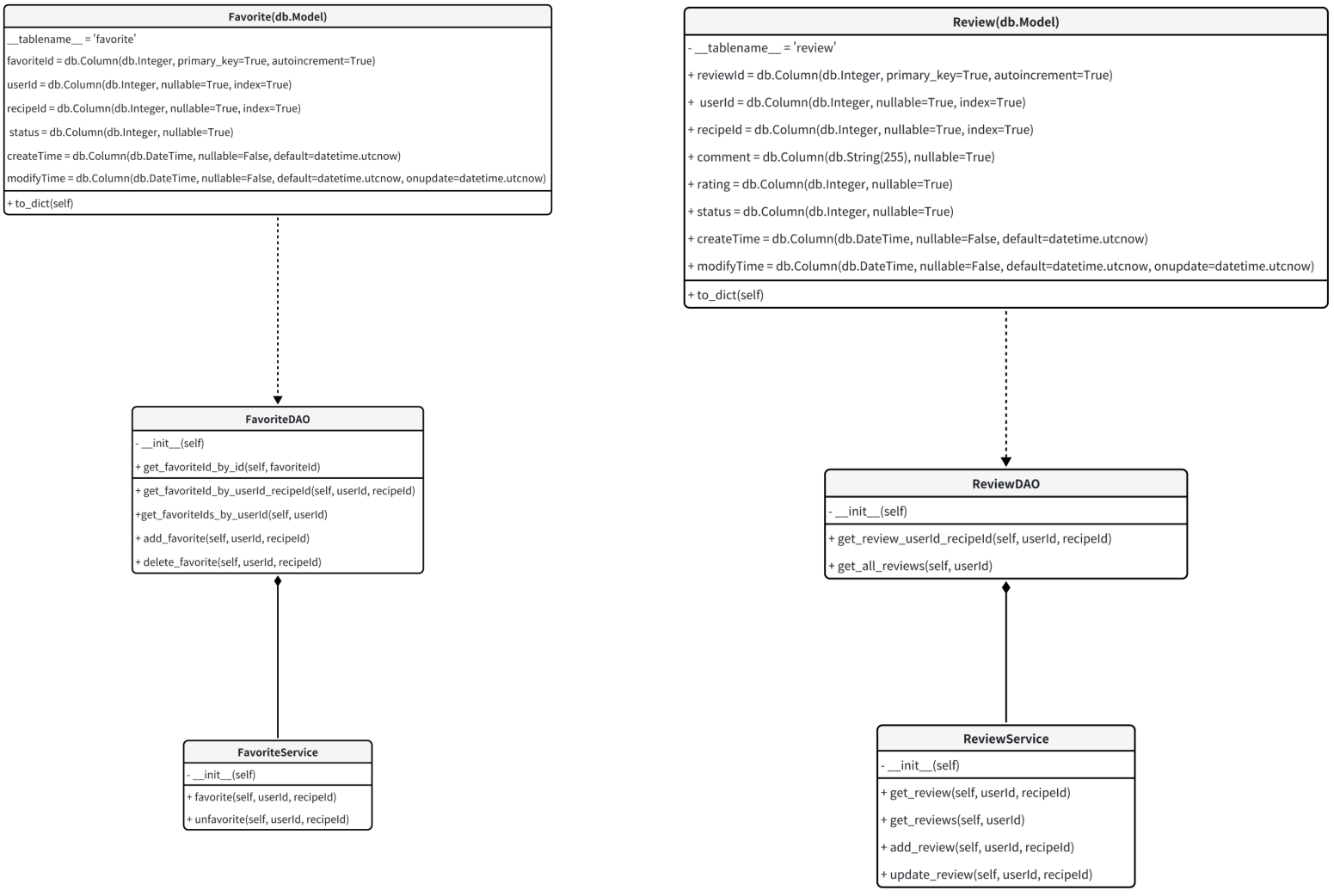
Requirement 8 (ID: 6480508): Recipe rating system
11.1. Implement a rating system where users can rate recipes on a scale (e.g., 1 to 5 stars).
11.2. Allow users to leave reviews or comments along with their ratings.

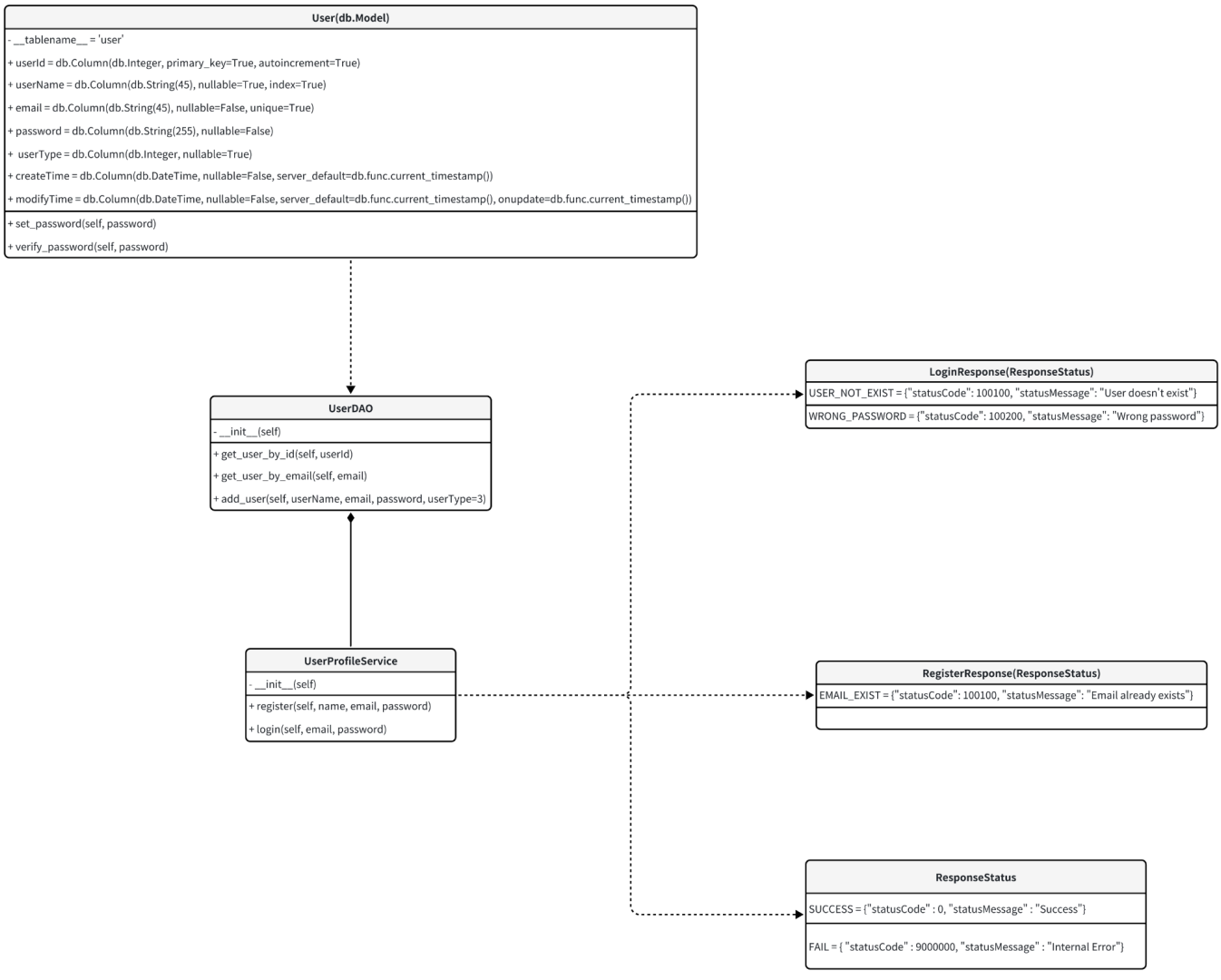
Requirement 9 (ID: 6480509): Recipe creation template
13.1. Provide a standardized template for users to create and submit their recipes, ensuring consistency in the presentation.
13.2. Include fields for ingredients, measurements, steps, cooking time, and photos in the template.

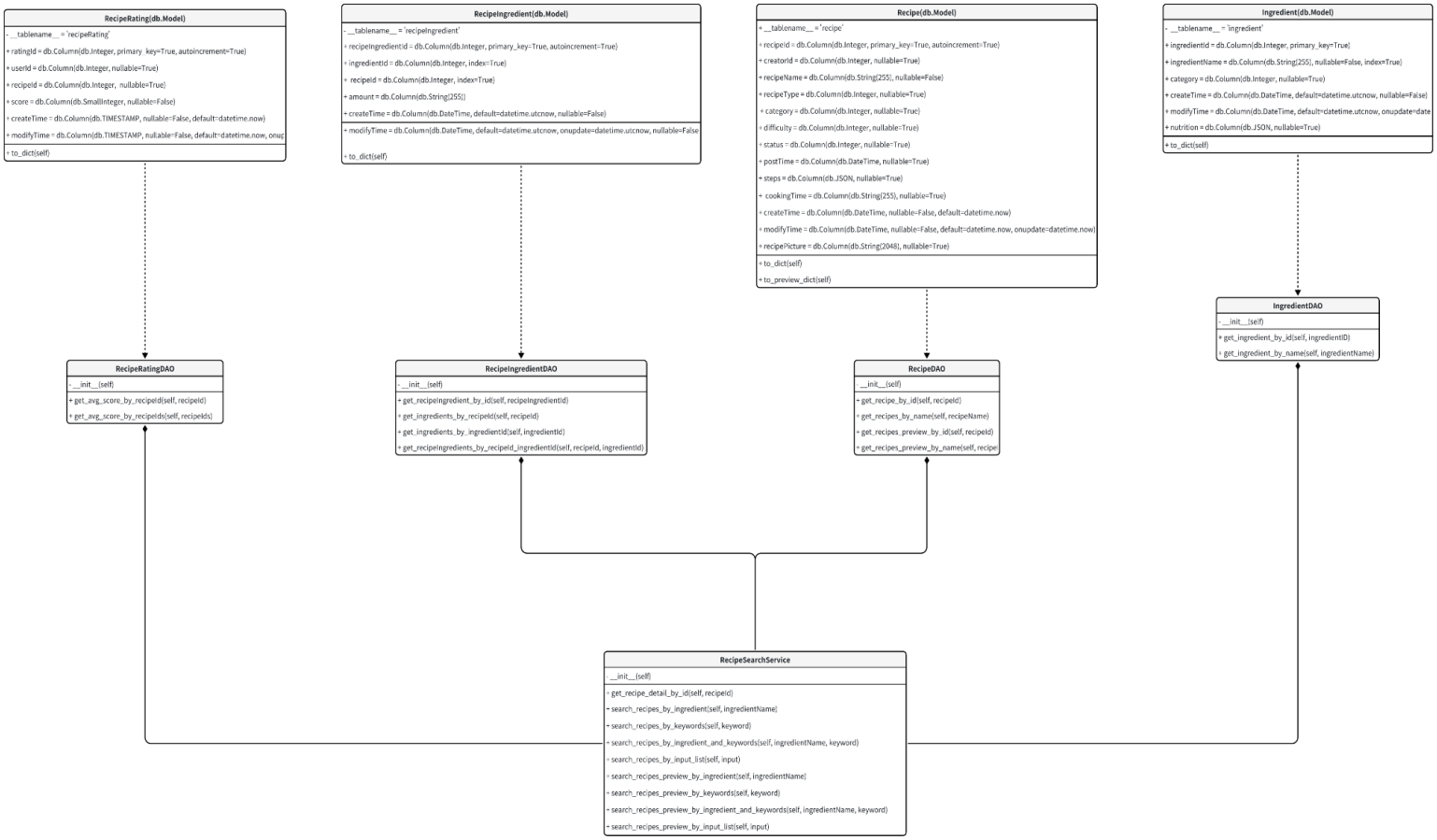
Requirement 10 (ID: 64805010): Recipes based on available ingredients
17.1. Create a feature allowing users to input their available ingredients and receive recipe suggestions that utilize those ingredients, prioritizing recipes that require the fewest additional ingredients.
17.2. Implement an "ingredient match" percentage to show how well the user's available ingredients align with the recipe requirements.

3. Architecture

- High-level UML class diagrams for implementation classes of core functionality.




User(db.Model)
- __tablename__ = 'user'
+ userId = db.Column(db.Integer, primary_key=True, autoincrement=True)
+ userName = db.Column(db.String(45), nullable=True, index=True)
+ email = db.Column(db.String(45), nullable=False, unique=True)
+ password = db.Column(db.String(255), nullable=False)
+ userType = db.Column(db.Integer, nullable=True)
+ createTime = db.Column(db.DateTime, nullable=False, server_default=db.func.current_timestamp())
+ modifyTime = db.Column(db.DateTime, nullable=False, server_default=db.func.current_timestamp(), onupdate=db.func.current_timestamp())
+ set_password(self, password)
+ verify_password(self, password)
UserDAO
- __init__(self)
+ get_user_by_id(self, userId)
+ get_user_by_email(self, email)
+ add_user(self, userName, email, password, userType=3)
UserProfileService
- __init__(self)
+ register(self, name, email, password)
+ login(self, email, password)
LoginResponse(ResponseStatus)
USER_NOT_EXIST = {"statusCode": 100100, "statusMessage": "User doesn't exist"}
WRONG_PASSWORD = {"statusCode": 100200, "statusMessage": "Wrong password"}
RegisterResponse(ResponseStatus)
EMAIL_EXIST = {"statusCode": 100100, "statusMessage": "Email already exists"}
ResponseStatus
SUCCESS = {"statusCode" : 0, "statusMessage" : "Success"}
FAIL = { "statusCode" : 9000000, "statusMessage" : "Internal Error"}



IngredientType(Enum)

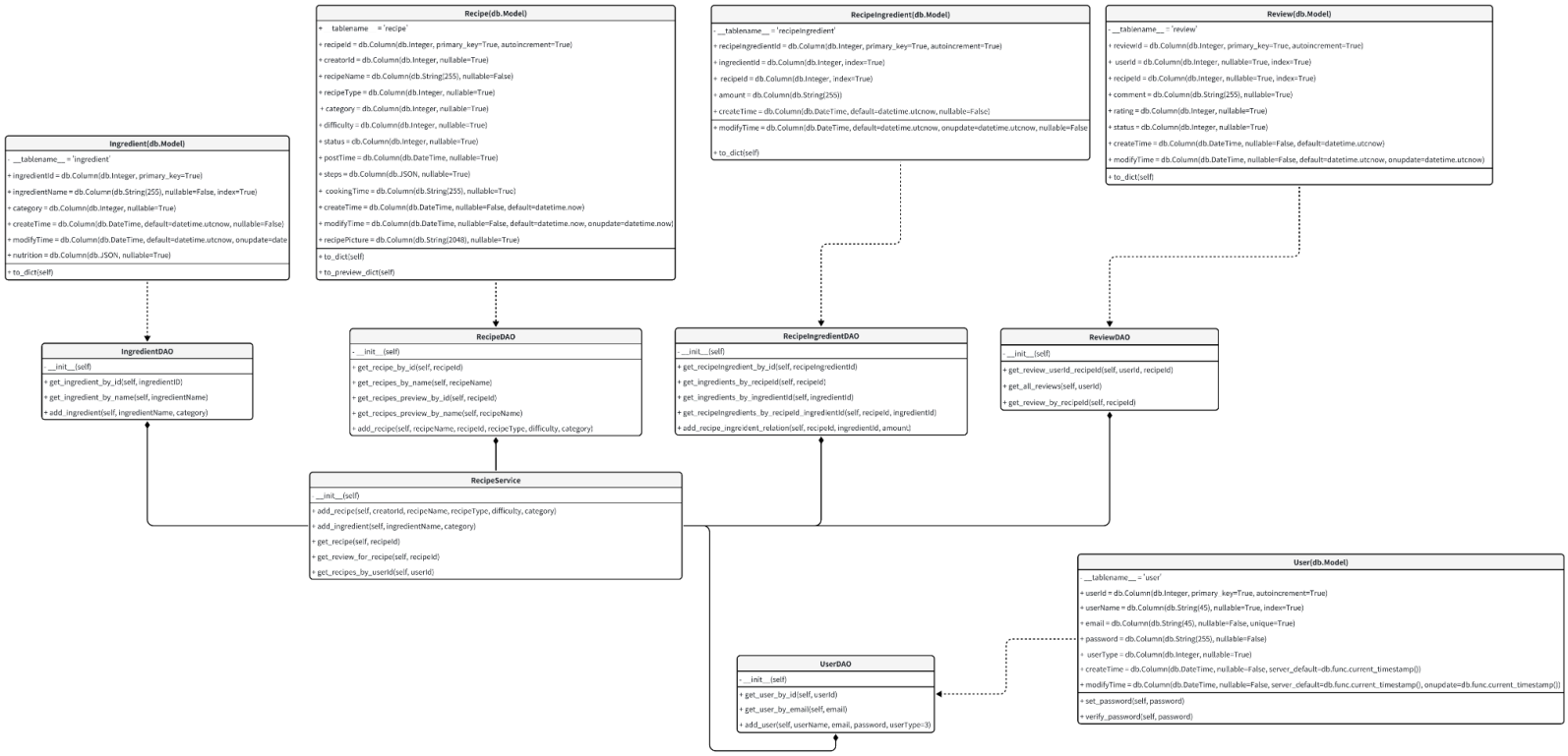
MEAT = 1
VEGETABLE = 2
FRUIT = 3
GRAIN = 4
DAIRY = 5
OIL = 6
SPICE = 7
SWEETNER = 8
LIQUID = 9

+ get_category_name_by_code(code)

RecipeCategory(Enum)

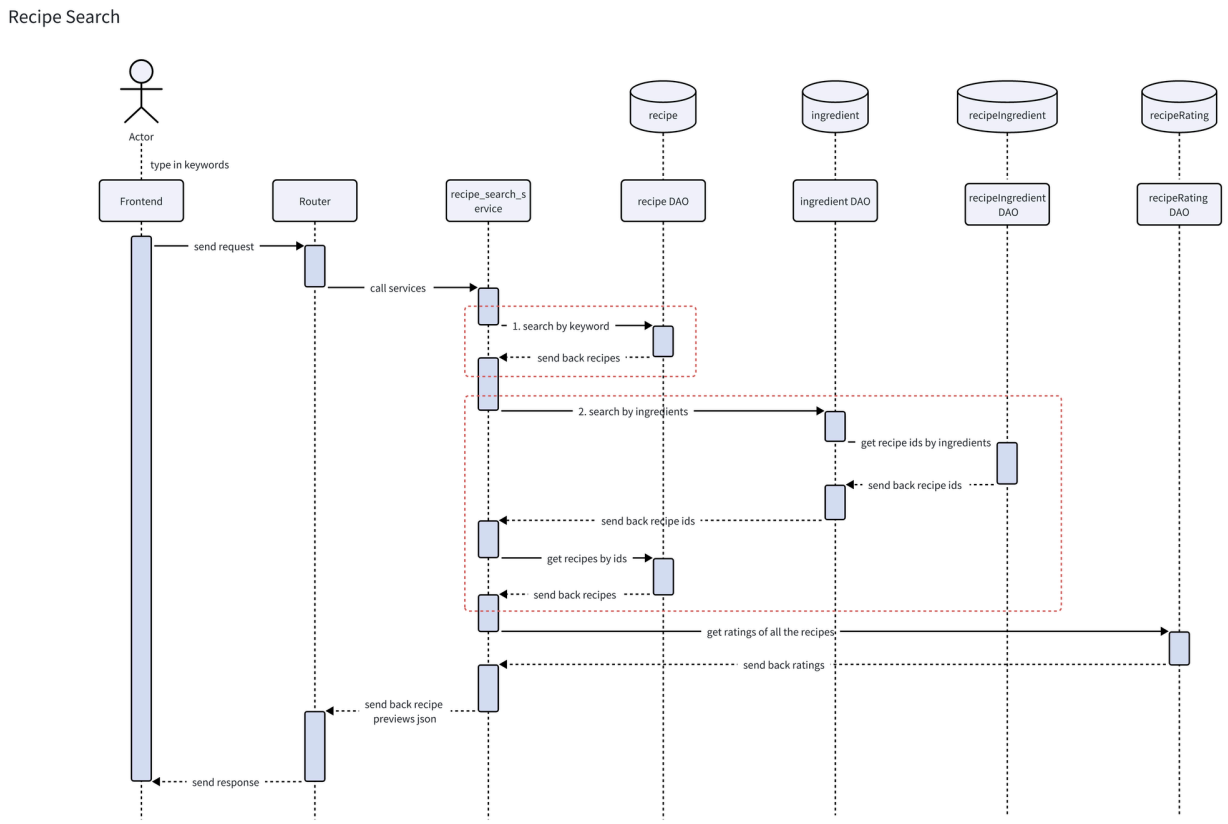
BREAKFAST = 1
LUNCH = 2
DINNER = 3
DESERT = 4
SNACK = 5
DRINK = 6
OTHER = 10000

+ get_category_name_by_code(code)


Recipe(db.Model)
+ tablename = 'recipe'
+ recipeId = db.Column(db.Integer, primary_key=True, autoincrement=True)
+ creatorId = db.Column(db.Integer, nullable=True)
+ recipeName = db.Column(db.String(255), nullable=False)
+ recipeType = db.Column(db.Integer, nullable=True)
+ category = db.Column(db.Integer, nullable=True)
+ difficulty = db.Column(db.Integer, nullable=True)
+ status = db.Column(db.Integer, nullable=True)
+ postTime = db.Column(db.DateTime, nullable=True)
+ steps = db.Column(db.JSON, nullable=True)
+ cookingTime = db.Column(db.String(255), nullable=True)
+ createTime = db.Column(db.DateTime, nullable=False, default=datetime.now)
+ modifyTime = db.Column(db.DateTime, nullable=False, default=datetime.now, onupdate=datetime.now)
+ recipePicture = db.Column(db.String(2048), nullable=True)
+ to_dict(self)
+ to_preview_dict(self)
RecipeIngredient(db.Model)
- __tablename__ = 'recipeingredient'
+ recipeingredientId = db.Column(db.Integer, primary_key=True, autoincrement=True)
+ ingredientId = db.Column(db.Integer, index=True)
+ recipeId = db.Column(db.Integer, index=True)
+ amount = db.Column(db.String(255))
+ createTime = db.Column(db.DateTime, default=datetime.utcnow, nullable=False)
+ modifyTime = db.Column(db.DateTime, default=datetime.utcnow, onupdate=datetime.utcnow, nullable=False
+ to_dict(self)
Review(db.Model)
- __tablename__ = 'review'
+ reviewId = db.Column(db.Integer, primary_key=True, autoincrement=True)
+ userId = db.Column(db.Integer, nullable=True, index=True)
+ recipeId = db.Column(db.Integer, nullable=True, index=True)
+ comment = db.Column(db.String(255), nullable=True)
+ rating = db.Column(db.Integer, nullable=True)
+ status = db.Column(db.Integer, nullable=True)
+ createTime = db.Column(db.DateTime, nullable=False, default=datetime.utcnow)
+ modifyTime = db.Column(db.DateTime, nullable=False, default=datetime.utcnow, onupdate=datetime.utcnow)
+ to_dict(self)
Ingredient(db.Model)
- __tablename__ = 'ingredient'
+ ingredientId = db.Column(db.Integer, primary_key=True)
+ ingredientName = db.Column(db.String(255), nullable=False, index=True)
+ category = db.Column(db.Integer, nullable=True)
+ createTime = db.Column(db.DateTime, default=datetime.utcnow, nullable=False)
+ modifyTime = db.Column(db.DateTime, default=datetime.utcnow, onupdate=date
+ nutrition = db.Column(db.JSON, nullable=True)
+ to_dict(self)
IngredientDAO
- __init__(self)
+ get_ingredient_by_id(self, ingredientID)
+ get_ingredient_by_name(self, ingredientName)
+ add_ingredient(self, ingredientName, category)
RecipeDAO
- __init__(self)
+ get_recipe_by_id(self, recipeId)
+ get_recipes_by_name(self, recipeName)
+ get_recipes_preview_by_id(self, recipeId)
+ get_recipes_preview_by_name(self, recipeName)
+ add_recipe(self, recipeName, recipeId, recipeType, difficulty, category)
RecipeIngredientDAO
- __init__(self)
+ get_recipeingredient_by_id(self, recipeIngredientId)
+ get_ingredients_by_recipeId(self, recipeId)
+ get_ingredients_by_ingredientId(self, ingredientId)
+ get_recipeIngredients_by_recipeId_ingredientId(self, recipeId, ingredientId)
+ add_recipe_ingredient_relation(self, recipeId, ingredientId, amount)
ReviewDAO
- __init__(self)
+ get_review_userId_recipeId(self, userId, recipeId)
+ get_all_reviews(self, userId)
+ get_review_by_recipeId(self, recipeId)
RecipeService
__init__(self)
+ add_recipe(self, creatorId, recipeName, recipeType, difficulty, category)
+ add_ingredient(self, ingredientName, category)
+ get_recipe(self, recipeId)
+ get_review_for_recipe(self, recipeId)
+ get_recipes_by_userId(self, userId)
User(db.Model)
__tablename__ = 'user'
+ userId = db.Column(db.Integer, primary_key=True, autoincrement=True)
+ userName = db.Column(db.String(45), nullable=True, index=True)
+ email = db.Column(db.String(45), nullable=False, unique=True)
+ password = db.Column(db.String(255), nullable=False)
+ userType = db.Column(db.Integer, nullable=True)
+ createTime = db.Column(db.DateTime, nullable=False, server_default=db.func.current_timestamp())
+ modifyTime = db.Column(db.DateTime, nullable=False, server_default=db.func.current_timestamp(), onupdate=db.func.current_timestamp())
+ set_password(self, password)
+ verify_password(self, password)
UserDAO
- __init__(self)
+ get_user_by_id(self, userId)
+ get_user_by_email(self, email)
+ add_user(self, userName, email, password, userType=3)

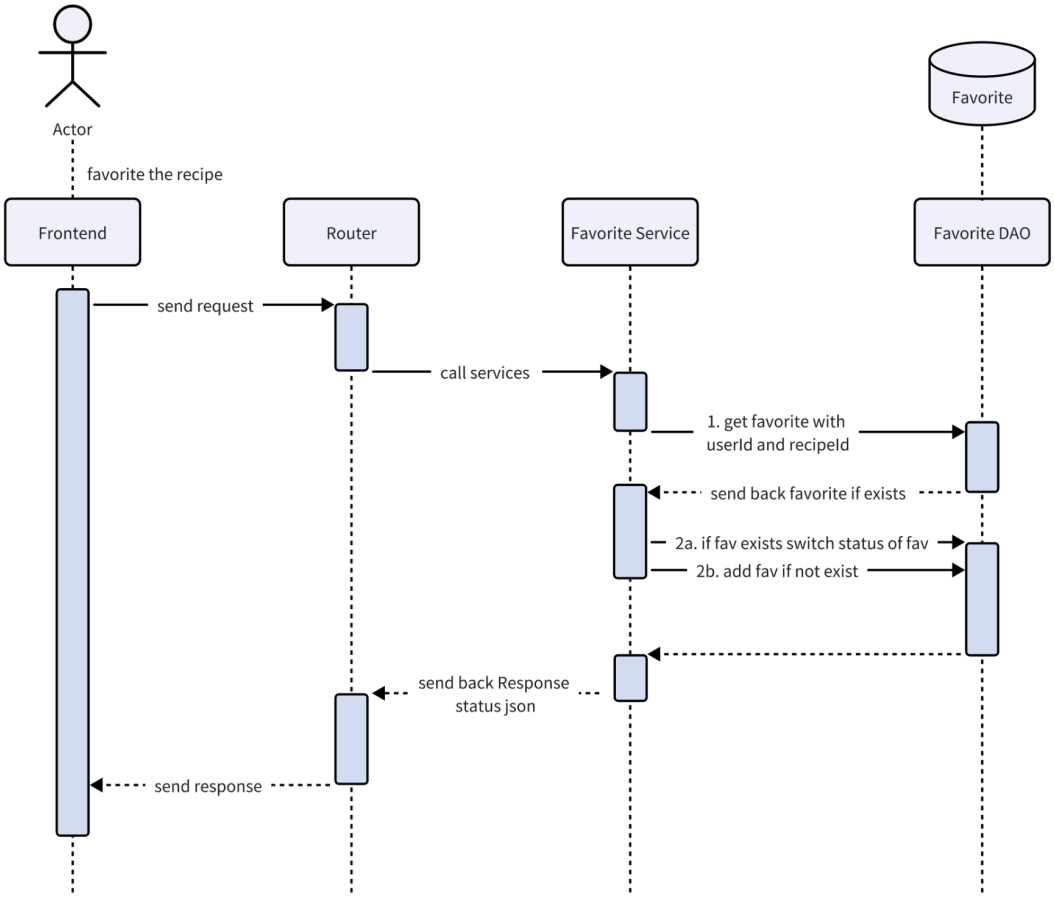
- High-level sequence diagrams: for ~5~6 functional requirements, please develop UML sequence diagrams.

Feature #1 Search Recipe



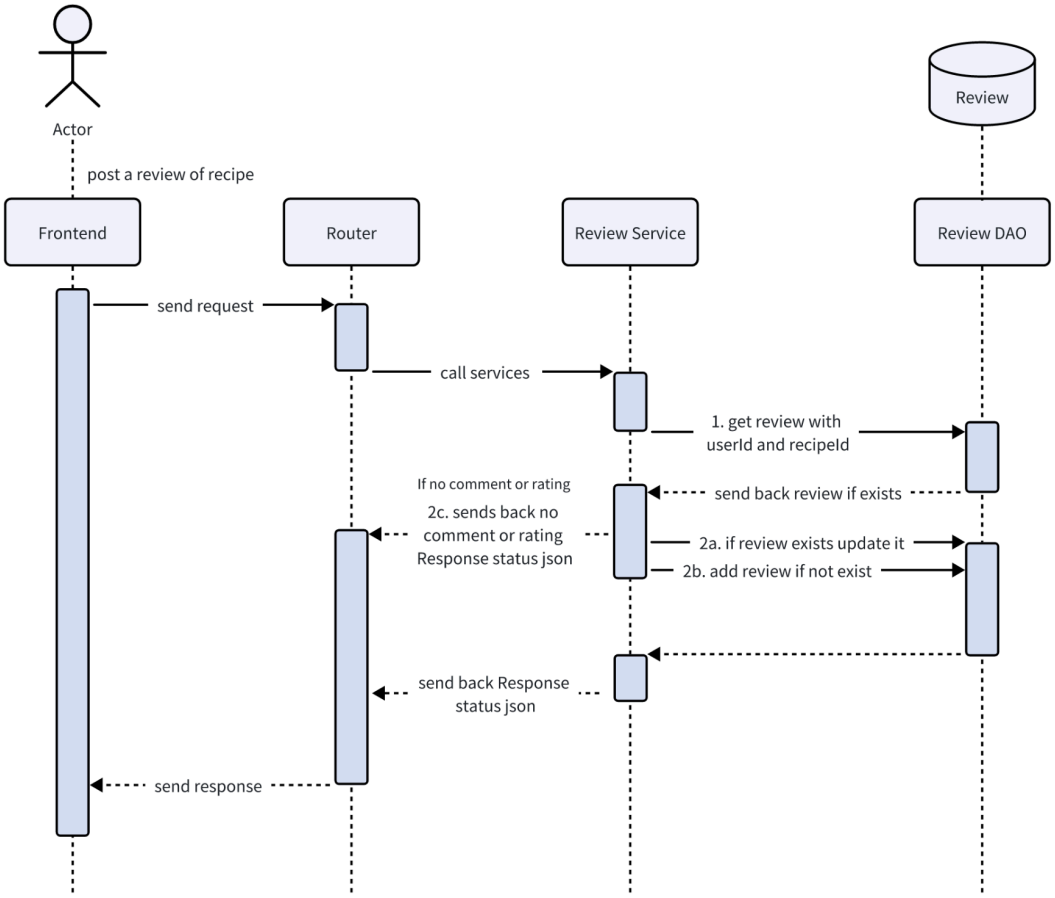
Feature #2 Favorite Recipe

Favorite Recipe



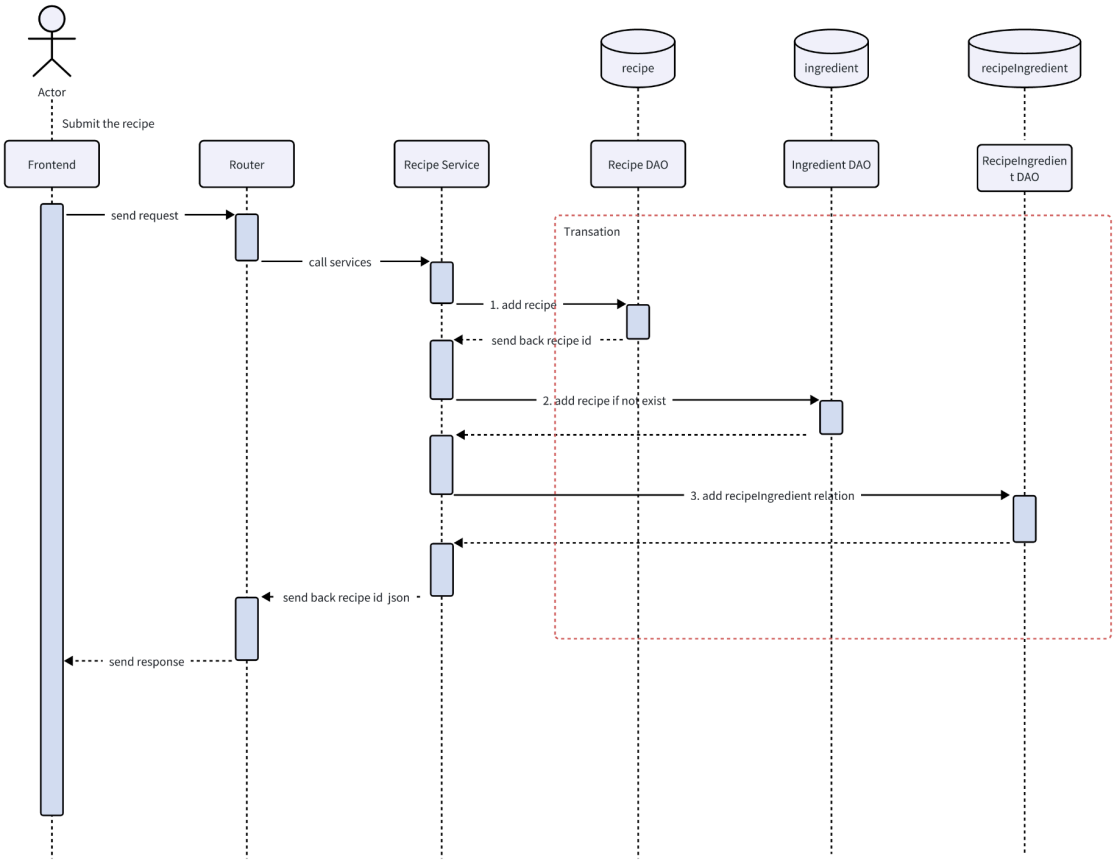
Feature #3 Review Recipe

Review Recipe



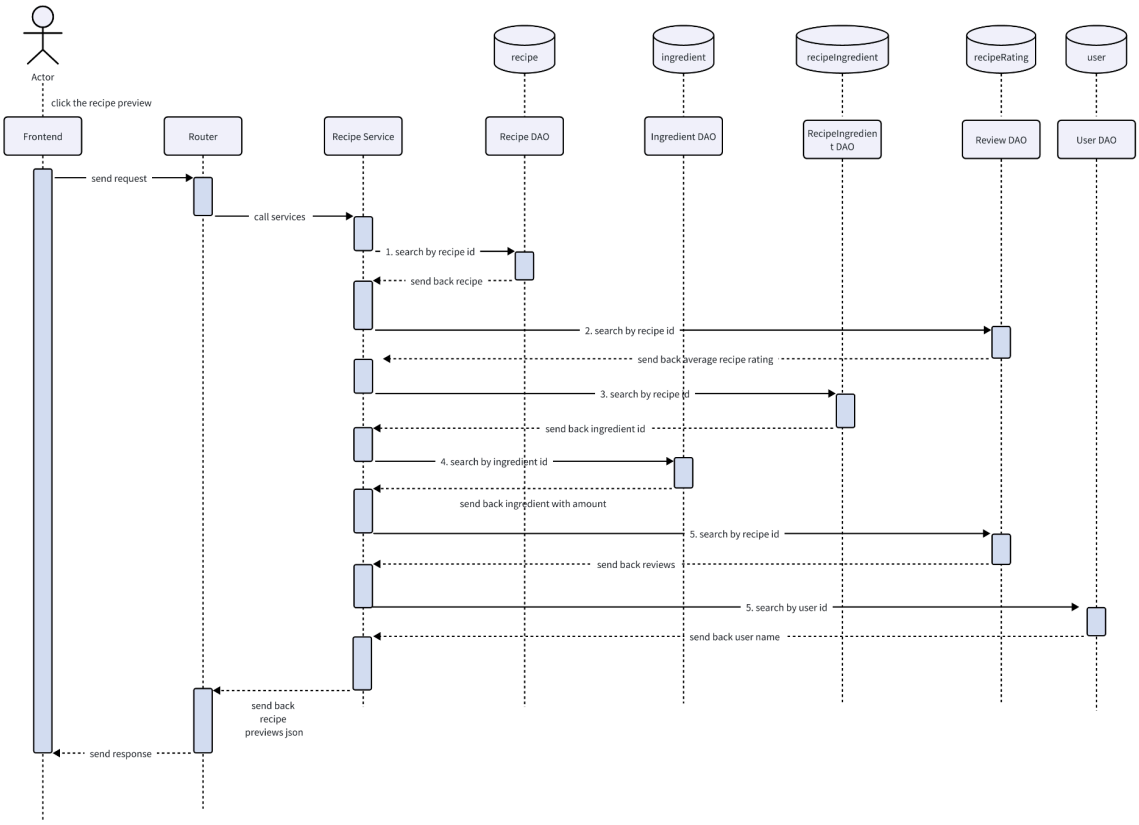
Feature #4 Create Recipe

Create Recipe



Feature #5 Recipe Detail

Recipe Detail



4. Project status

Risks: all actual (not hypothetical) risks (schedule, team work, technical, skills etc.) should be identified and either resolved or plans should be made to resolve them

1. Technical(front-end): Front-end team are facing difficulty designing uniformal and professional web pages and calling the API.
   Plans: Find a recipe website UI for reference. Communicate and pair coding with the back end team.
2. Technical(front-end): Create recipe page, upload image feature for the recipe and each instruction step is not working. The process of inserting the image into the database fails.
   Plans: Try to store the image in the front-end folder.
3. Teamwork: We are still trying to find the best way to improve teamwork. We had a once a week zoom meeting, but found out it did not help..
   Plans: During Milestone 3, we had in-person coding in the library and that really worked. So we can use the online meeting just to sync the progress and meet in-person to solve problems and write codes.
4. Technical(back-end): We are having a hard time working with the FDC api for nutrition information.
   Plans: member(s) that are having problems are working on improving their skills with online resources and are reaching out to appropriate team members that may be able to help. If we still cannot solve this, we may consider removing this feature.
5. Technical: the most important feature(unique feature) has not been implemented yet.