Stage4: Final Report

Demo Video Link: https://youtu.be/iSZExtQTITA

1. Changes in the directions

In our initial proposal, we aimed to build a meal planning platform inspired by dating apps, featuring a swipe-based interaction along with personalized calorie tracking, recipe saving, and grocery shopping list.

However, during the development process, we shifted toward a more traditional recipe browsing system where users are able to search recipes based on ingredients and save the recipes to their account. The grocery list generation feature, the calendar, and the ability for users to add their new recipes were eventually dropped due to time constraints.

While we simplified some features, the final product still aligns with the original goal of helping users discover recipes based on available ingredients and basic health needs.

2. Application Usefulness: Achievements and Limitations

Our application successfully achieved its primary goal of helping users quickly find recipes based on the ingredients they already have at home. The ingredient search feature allows users to dynamically add available items and discover matching recipes, simplifying daily cooking decisions.

In addition to providing total calories information for each recipe, the application also enriches the user experience by recommending similar recipes based on shared ingredients, helping users explore alternative options. Moreover, we integrated external cooking videos linked to each recipe when available, offering users additional guidance during the cooking process. These features deliver both practical convenience and broader culinary discovery, supporting users in making healthier and more varied eating choices.

However, some of the goals we set initially were not fully achieved. We were not able to implement features such as generating weekly meal plans based on nutritional goals or creating personalized recommendations based on user

behavior history. These features would help users to plan healthier diets and have a more intelligent experience.

3. Schema or Data Source Changes

During development, we continued to use the originally selected data source, but have made adjustments to the schema.

We simplified the structure of food nutrition dataset since it contained rich nutritional information (such as fats, vitamins, and minerals). We ultimately retained only the calorie fields.

Similarly, we modified the Recipe schema by keeping key fields including RecipeID, RecipeTitle, Directions, and Source, in order to focus on the core functionalities we planned to make.

4. ER Diagram and Table Design Changes

1. We made important changes to the **Users** table compared to the original design.

Originally, the **Users** table was defined as:

```
CREATE TABLE Users (
    UserID INT PRIMARY KEY,
    Username VARCHAR(255) NOT NULL,
    Email VARCHAR(255) NOT NULL
);
```

Changes we made:

- We modified **UserID** to be AUTO_INCREMENT so that the database automatically generates a unique ID for each user without manually specifying it.
- We removed **Username** because our app mainly identifies users by their **Email**.
- We added two new fields: **CreatedAt** and **LastLoginAt** to track when a user registers and when they last logged in.

Final version:

```
CREATE TABLE Users (
    UserID INT PRIMARY KEY AUTO_INCREMENT,
    Email VARCHAR(255) NOT NULL UNIQUE,
    CreatedAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    LastLoginAt TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE CURRE
);
```

2. We also made adjustments to the **Review** table.

Originally, the **Review** table was designed as:

```
CREATE TABLE Review (
    ReviewID INT PRIMARY KEY,
    RecipeID INT,
    UserID INT,
    Stars INT,
    Comments VARCHAR(1000),
    FOREIGN KEY (RecipeID) REFERENCES Recipe(RecipeID),
    FOREIGN KEY (UserID) REFERENCES Users(UserID)
);
```

Changes we made:

- We modified **ReviewID** to be AUTO_INCREMENT using:

```
ALTER TABLE Review MODIFY ReviewID INT NOT NULL AUTO_INCREMENT;
```

In the original design, every new review would require manually assigning a unique ReviewID. The final version automates ID creation, making inserts safer and less error-prone.

3. We also separate **IngredientName_Recipe and Quantity**

In the original design, ingredients were stored as a single string, combining ingredient name and quantity together. We modified the structure by separating IngredientName_Recipe and Quantity into two distinct fields. This change allows for better matching with ingredients data and easier querying.

Changes we made:

```
CREATE TABLE ComposedOf (
   id INT AUTO_INCREMENT,
   RecipeID INT,
   IngredientName_Recipe VARCHAR(512),
   IngredientName_Nutrition VARCHAR(512),
   IngredientID INT,
   Quantity VARCHAR(255),
   PRIMARY KEY (id),
   FOREIGN KEY (RecipeID) REFERENCES Recipe(RecipeID),
   FOREIGN KEY (IngredientID) REFERENCES Ingredients(IngredientID)
);
```

5. Added and Removed Functionalities

Removed functionalities:

Some of the swipe-based browsing feature was removed specifically the option to dislike a recipe, because it is not critical to the recipe discovery experience and user can also skip them if they wanted.

We originally intended to create weekly meal plans to support users' dietary needs and preferences. Due to the complexity of constructing a full recommendation and scheduling system, we shifted focus toward providing basic nutritional information instead.

In addition, the grocery list generation feature, which was closely tied to meal planning, was no longer relevant after removing the meal planning functionality.

Added functionalities:

Beyond the originally planned calorie calculation feature, we added a new functionality that identifies and recommends the recipe most similar in ingredient composition to the one currently viewed. This feature helps users discover

alternative dishes they can prepare with overlapping ingredients, enriching the browsing experience and offering practical inspiration when ingredient availability is limited.

6. How Advanced Database Programs Support the Application

To better support the application's functionality, we developed both a stored procedure and a trigger that integrate directly with core features.

The stored procedure, `CalcRecipeCalories`, calculates the total calories of a recipe based on the associated ingredients and their quantities. It also handles unit conversions to ensure consistent measurements. By embedding this logic within the database, we are able to deliver real-time and accurate nutritional information without placing additional load on the application server. Additionally, it also handles the logic finding a recipe with most amount of common ingredients. This design choice improves the efficiency and responsiveness of our nutrition-related features.

In addition, the `validate_review_length` trigger enforces a minimum length for user comments before they are inserted into the database. This ensures that the review system maintains a basic standard of content quality, enhancing the overall credibility and usefulness of user feedback.

By delegating important validation and computation tasks to the database layer, these programs reduce application-side complexity and improve both the reliability and maintainability of the platform.

7. Technical Challenge

Jiajun Huo: While developing the recipe YouTube video page, we faced a challenge with YouTube API quota limits, which prevented us from fetching videos for all recipes in real time. To solve this, we designed a switch to control API usage based on quota consumption and adopted a database-first approach: when a user accesses a recipe, the system first checks if a video already exists in the database. If not, it fetches the video from YouTube, stores it locally, and serves it. This reduces API calls, lowers costs, and ensures faster access for future visits.

Yvonne Zhang: One technical challenge I addressed was enforcing a minimum length for user reviews. Initially, we handled this validation on the frontend, but to ensure stronger data integrity, I implemented a **MySQL trigger** that blocks any

review shorter than five characters at the database level. This approach ensures that validation remains secure even if frontend restrictions are bypassed.

Yiting Wang: One technical challenge we encountered was designing efficient SQL queries for multi-ingredient searches while avoiding common pitfalls like incorrect grouping and duplication. When users selected multiple ingredients, we needed to ensure that only recipes containing *all* selected ingredients were returned, not just any matching one. This required careful construction of the group by and having count clauses, and thorough testing to guarantee both correctness and scalability as the number of search ingredients varied.

Zhihao Cheng: One of the technical challenge I resolved was calculating total recipe calories when ingredient units were inconsistent. The nutrition dataset listed calories per 100 grams, but recipes used a wide range of units like ounces, cups, milliliters, or vague ones like "1 can" or "1 bar." I solved this by building a conversion function that standardizes known units into grams using fixed conversion rates. For ambiguous units, I used reasonable default estimates (e.g., one can = 300g) to enable calorie estimation. This approach allowed us to provide calorie counts for most recipes as accurately as we can.

8. Future Work

- **Weekly Meal Planning:**

  In the future, users can save different dishes and organize them into a personal meal plan.

  They can personalize the meal plan's name and save it for later use, allowing for a more customized and convenient cooking experience.

- **Grocery List Generator:**

  Once meal planning is ready, we can automatically aggregate all selected recipe ingredients into a single grocery list. Users could then check off ingredients while shopping, making the app more practical for real-life use.

9. Teamwork

Responsibilities	Team
Frontend(UI design, Client-side logic)	Yiwen (Yvonne) Zhang, Jiajun Huo

Backend(Server setup, API development)	Jiajun Huo, Zhihao Cheng
Database Management(Schema design, SQL processing)	Yiting Wang, Yiwen (Yvonne) Zhang, Zhihao Cheng
Tools (GitHub, GCP, MySQL, Notion)	All members

While each team member had clearly defined responsibilities, we maintained a collaborative workflow in which everyone actively contributed to different parts of the project as needed.
Through regular discussions and joint decision-making, team members were able to offer input beyond their assigned areas, developing a broader understanding of the system as a whole.
This approach strengthened our coordination, improved our ability to solve problems efficiently, and helped ensure the high quality of the final application.