Changes in Project Direction

Initially, our proposal focused on generating comprehensive weekly meal plans based on users' calorie, nutritional needs, and dietary restrictions. However, during implementation, we shifted to a more flexible approach by introducing recipe templates that users can save and customize according to their preferences. Despite this shift, we retained the feature enabling users to search and retrieve recipes that align with their calorie and nutritional requirements.

Achievements and Usefulness

Our application meets its primary objective of allowing users to find and customize recipes within their calorie range and dietary restrictions. This ensures that users can effectively access recipes tailored to their needs.

However, we did not achieve our initial goal of automating weekly meal planning. We thought this can give users more autonomy and avoid rigid meal plans.

Data Schema and Source

No changes were made to the data schema or the source of data for the application during the development process.

ER Diagram and Table Implementation

The ER diagrams and table implementations remained consistent with the original design. Upon evaluation, we concluded that the initial design effectively supported the application's requirements, and no modifications were necessary.

Added and Removed Functionalities

- **Added:**
    - Functionality for users to customize ingredient quantities in recipes. This enhancement improved user engagement and provided more flexibility.
- **Removed:**

- ○ The feature to generate a full weekly meal plan. We removed this to avoid imposing rigid structures on users, opting instead to allow them to customize individual recipes as needed.

Advanced Database Programs

Indexing the database table allowed other query to run more efficiently when operations need to scan through the long tables.

Stored procedures made our backend code concise that we do not have to enter to complete long sql queries. Instead, we can invoke the save procedures. Also, we wrote a query that displays the calories difference between 10 most popular dishes in 2 time periods and put that in the stored procedure.

For triggers, we wanted to include the log of the history change of the amount so that the user can see how they edited the amount of the ingredients overtime.

```
CREATE TABLE IngredientModificationLog (
   log_id INT AUTO_INCREMENT PRIMARY KEY,
   customized_id INT,
   ingredient_id INT,
   old_amount DECIMAL(10,2),
   new_amount DECIMAL(10,2),
   old_unit VARCHAR(20),
   new_unit VARCHAR(20),
   modification_date TIMESTAMP,
   FOREIGN KEY (customized_id) REFERENCES CustomizedRecipes(customized_id),
   FOREIGN KEY (ingredient_id) REFERENCES Ingredients(ingredient_id)
);
CREATE TRIGGER after_ingredient_update
AFTER UPDATE ON ingredient_portion
FOR EACH ROW
BEGIN
   IF NEW.amount != OLD.amount OR NEW.unit != OLD.unit THEN
      INSERT INTO IngredientModificationLog (
         customized_id,
         ingredient_id,
         old_amount,
         new_amount,
         old_unit,
         new_unit,
         modification_date
      ) VALUES (
         NEW.customized_id,
         NEW.ingredient_id,
         OLD.amount,
            NEW.amount,
            OLD.unit,
            NEW.unit,
            CURRENT_TIMESTAMP
        );
    END IF;
END;
```

We wanted to implement Table-Level constraints that ensure that the detailed information in Recipes are correct, especially for some values to be non-negative (eg. minutes INT CHECK (minutes>0)). Additional constraints that can be added after table creation such as check the unit name is correct and amount does not exceed a certain value, like CHECK ((unit IN (‘g’, ‘ml’) AND amount <= 1000). Since it would take too long to recreate the table, we have to postpone this plan.

Technical Challenges Encountered

Hongyi Yang:
Frequent merge conflicts arose during the integration of features. To address this, we recommend maintaining separate branches for each feature and merging into the main branch only after thorough testing and review.

Yiwei Wang:
Dependency management posed significant challenges, as code functioning correctly on one system often failed on another. Standardizing environment configurations and using containerization tools like Docker would mitigate such issues in the future.

Peiyang Wu:
JavaScript’s permissive typing led to frequent mismatches between frontend data and database responses. Incorporating stricter type definitions, such as using TypeScript, could help prevent such issues.

Kaiwen Zhu:

Not applicable.

Other Changes from Original Proposal

There are no other changes we made to the original proposal.

Future Improvements

Beyond interface enhancements, the application could benefit from:

1. **Stronger Type Definitions:** Introducing stricter typing, especially with TypeScript, to reduce runtime errors.
2. **Improved Code Modularization:** Refactoring the codebase for better organization and maintainability.
3. **Optimized Dependency Management:** Employing tools to streamline and standardize configurations across development environments.

Division of Labor and Teamwork

- **Hongyi Yang:** User interface development, registration/login functionality, backend coordination.
- **Yiwei Wang:** Customized recipe interface, recipe recommendation feature.
- **Peiyang Wu:** Backend CRUD operations for users, recipes, and dietary restrictions.
- **Kaiwen Zhu:** Backend CRUD operations for users, recipes, and dietary restrictions.

The workload was divided among team members based on features. Overall, most of the team worked effectively, but we could benefit more from active participation of each teammate.