Milestone 2 - Meal Master

Information Management & Systems Engineering

WS 2023

Nadezhda Tsvetkova - 11942924

Lucas Enzi - 12029628

Main Idea

MealMaster is a user-friendly meal planning tool that helps you create meal plans for any number of days. With MealMaster, you can store your favorite recipes, each with a name, cooking instructions, a rating, and links to the original recipe and images of the dish.

Recipes are categorized using a color-coded tagging system for quick and intuitive searching based on user preferences or dish types. MealMaster's ingredient feature allows for the cataloging of food items, complete with alternative options, offering flexibility in meal preparation.

MealMaster also lets you manage ingredients. Each recipe can include various ingredients, which can be used in multiple recipes. Ingredients can be linked to alternatives, giving you flexibility in your cooking.

Ratings help you to keep track of your favorite dishes with the breeze of a klick. Each rating contains a score (1 to 5 Stars), a review and a name of the author. A recipe can contain multiple ratings.

MealMaster also includes a feature for generating random meal plans, helping users to decide what to cook over a set number of days. It simplifies meal preparation by standardizing ingredient units, ensuring consistency across recipes.

In order to simplify the adding of ingredients to a recipe, the used units are also provided for each component.

MealMaster is crafted to be a reliable tool for individuals seeking an organized approach to meal planning and culinary exploration.

1. RDBMS Part

1.1 Configuration of Infrastructure

Our application is built on the Spring Boot framework, leveraging Java 17 for robust backend development. We use Thymeleaf for server-side HTML views, Spring Data JPA for relational database interactions, and connect to MongoDB for NoSQL data handling. The app is containerized with Docker, simplifying deployment and ensuring consistent environments across development and production.

To run the app:

- Install Docker and Docker Compose if not already installed.
- Navigate to the standalone directory, which is in the root of the project.
- Execute *docker-compose up* in the terminal to build and start the services.

- Once the services are up, access the application at http://localhost

This setup ensures a streamlined workflow from development to deployment, with Docker handling service orchestration.

Important: A file called generatedContent.txt is automatically created during the data generation process. If you want to generate the data again you must first delete this file.

1.2 Data Import

Upon launching our application and accessing it via the localhost URL, the homepage presents users with a simple yet vital feature - the data generation functionality. With a single click, users can populate the database with a diverse range of recipes. This process is not just about filling the database; it's about setting the stage for a rich, interactive user experience. After the import is complete, users are seamlessly redirected to the search page, where they can explore and discover recipes to their heart's content, engaging with our application's core functionality. The recipes and the data itself are absolutely random and it could happen that they don't have any meaning.

1.3 Implementation of Web System

Each member of the team implemented his **main use-case** and the **report**.

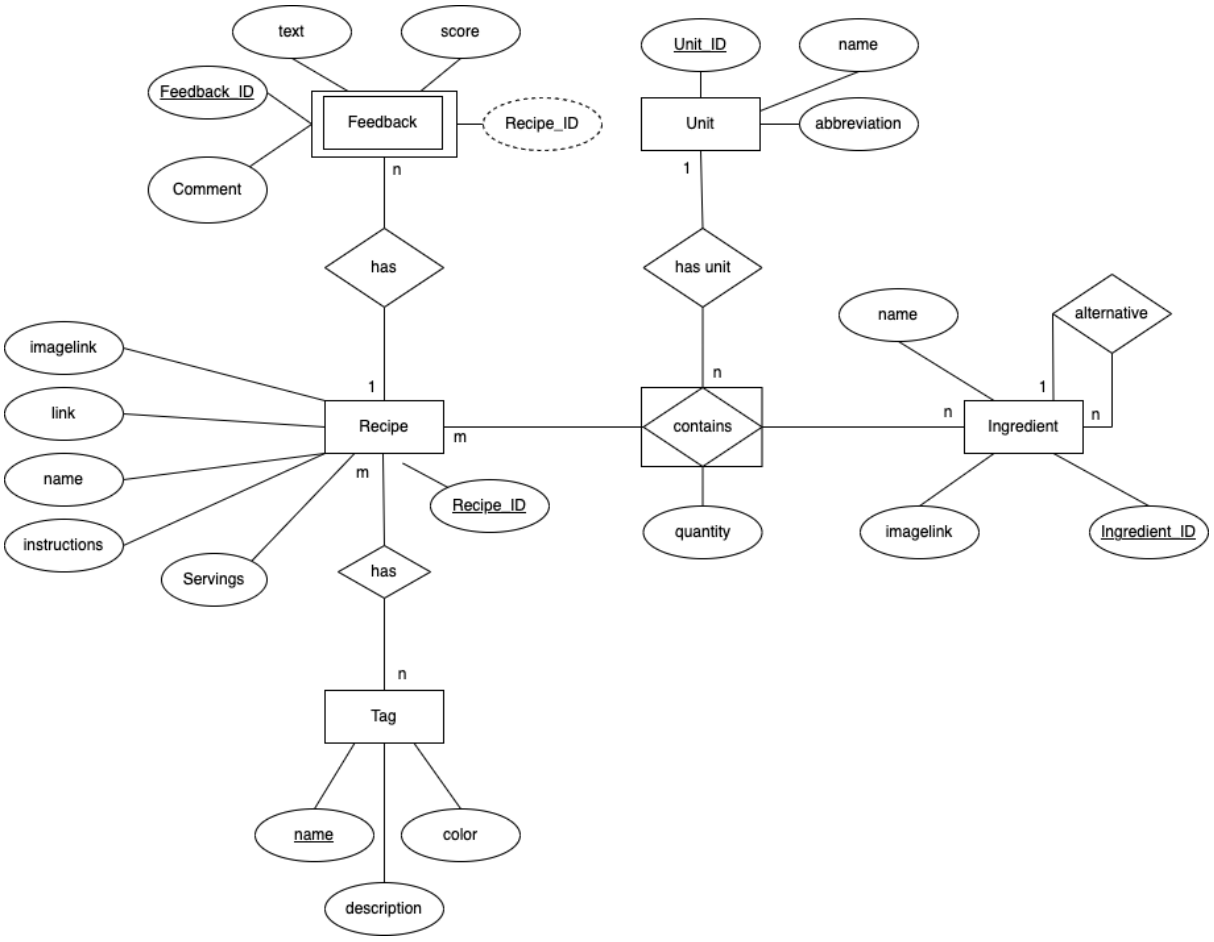
- Lucas Enzi:
  Main use-case is adding a recipe and report about the top 5 ingredients for the most common tag.
- Nadezhda Tsvetkova:
  Main use-case is adding a feedback to a specific recipe and report about the most used ingredient from top rated recipes.

2. NoSQL Design

Logical database design

In our project's progression, we found that some changes to the ER diagram from Milestone 1 are necessary. Enriching it with additional attributes to capture more detailed recipe and feedback information. During the transition from the logical

model to the physical database design, we have tried to apply all necessary normalization principles. This ensured the elimination of many-to-many relationships, leading to the creation of associative tables such as RecipeContainsIngredient and RecipeHasTag, which facilitate the representation of complex relationships through simpler one-to-many associations. This normalization not only optimizes our database's integrity and efficiency but also simplifies the data management processes.



For the relational database we used MySQL.

Schemes

The NoSQL schema for our project implements two collections: Recipe and Tag. This ensures the most efficient way to access an individual recipe, but also the tags, so that the data can be filtered in an efficient way. It is following the general rules laid out by MongoDB: “Favor embedding unless there is a compelling reason not to.” and “Needing to access an object on its own is a compelling reason not to embed it.” (Karlsson, 2022). Compared to the relational design, there are just two

collections instead of 6 tables (plus one table for the relationship between Recipe and Tag).

- Each Recipe document contains comprehensive details including name, imagelink, link, instructions, servings, and arrays of tags and ingredients.
- The ingredients subdocument includes name, amount, and unit details, with unit being another embedded document containing name and abbreviation.
- The feedback subdocument includes user feedback details like user, rating, and description.
- The Tag collection is simpler, with each document containing an _id, name, and color.

Recipe:

```
{
        _id: ObjectId,
        name: str,
        imagelink: str,
        link: str,
        instructions: str,
        servings: int,
        tags: [ str ],
        ingredients: [
                {
                        name: str,
                        amount: int,
                        unit: [
                                {
                                name: str,
                                abbreviation: str
                                }
                        ]
                }
        ],
        feedback: [
                {
                        user: str,
                        score: int,
                        description: str
                }
        ]
}
```

Tag:

```
{
        _id: ObjectId,
        name: str,
        color: str
}
```

NoSQL Indexing

For our MongoDB setup, strategic indexing is crucial to enhance query performance. Given our use case, creating indexes on fields like **recipe.name, tag.name**, and **feedback.score** in the Recipe collection will significantly improve search efficiency. A text index on **recipe.instructions** could also be beneficial for full-text search capabilities. Additionally, considering a TTL (Time-To-Live) index on **feedback.date** (assuming this field is added) can be advantageous for automatically removing outdated feedback, keeping the database optimized.

Comparison of SQL and MongoDB Queries

- Main Use Case - Add Recipe:
    - **SQL:**
      INSERT INTO recipe (image_link, instruction, link, name, servings) VALUES (?, ?, ?, ?, ?);
    - **NoSQL:**
      db.recipes.insertOne({ name: "...", imagelink: "...", link: "...", instructions: "...", servings: ... });

- Main Use Case - Add Feedback:
    - **SQL:**
      INSERT INTO feedback (comment, score, user_name, recipe_id) VALUES (?, ?, ?, ?);
    - **NoSQL:**
      db.recipes.updateOne({ _id: ObjectId("...") }, { $push: { feedback: { user: "...", score: ..., description: "..." } } });

- Use Case - Search Recipe:
    - **SQL:**
      SELECT * FROM recipe WHERE name LIKE '%query%';
    - **NoSQL:**
      db.recipes.find({ name: /query/ });

- Use Case - Filter by Tag:
    - **SQL:**
      SELECT * FROM recipe JOIN recipe_has_tag ON recipe.id = recipe_has_tag.recipe_id WHERE tag_name = 'tagName';
    - **NoSQL:**
      db.recipes.find({ tags: "tagName" });

- Report 1: top 5 ingredients for most common tag
    - **SQL:**
      SELECT i.name, COUNT(*) as ingredient_count
      FROM ingredient i
      JOIN recipe_ingredient ri ON i.id = ri.ingredient_id
      JOIN recipe_has_tag rht ON ri.recipe_id = rht.recipe_id
      WHERE rht.tag_name = (
        SELECT tag_name
        FROM recipe_has_tag
        GROUP BY tag_name
        ORDER BY COUNT(*) DESC
        LIMIT 1
)
      GROUP BY i.name
      ORDER BY ingredient_count DESC
      LIMIT 5;

    - **NoSQL:**
      db.recipes.aggregate([
       { $unwind: "$tags" },
       { $group: { _id: "$tags", commonIngredients: { $addToSet: "$ingredients.name" } } },
       { $unwind: "$commonIngredients" },
       { $group: { _id: "$commonIngredients", count: { $sum: 1 } } },
       { $sort: { count: -1 } },
       { $limit: 5 }
      ]);

- Report 2: the most used ingredient from top rated recipes
    - **SQL:** SELECT ingredient.name
      FROM ingredient
      JOIN recipe_ingredient ON ingredient.id = recipe_ingredient.ingredient_id
      JOIN (
       SELECT recipe_id
       FROM feedback
       GROUP BY recipe_id
       ORDER BY AVG(score) DESC

```
  LIMIT 5
) AS top_recipes ON recipe_ingredient.recipe_id top_recipes.recipe_id
GROUP BY ingredient.name
ORDER BY COUNT(*) DESC
LIMIT 1;
```

- **NoSQL:**

```
db.recipes.aggregate([
   { $unwind: "$feedback" },
   { $group: { _id: "$_id", avgScore: { $avg: "$feedback.score" },
ingredients: { $first: "$ingredients.name" } } },
   { $sort: { avgScore: -1 } },
   { $limit: 5 },
   { $unwind: "$ingredients" },
   { $group: { _id: "$ingredients", count: { $sum: 1 } } },
   { $sort: { count: -1 } },
   { $limit: 1 }
]);
```

Work Protocol

For this part we used a version control tool, and you can see and check all of our work and commits here: https://git01lab.cs.univie.ac.at/lucase00/mealmaster

References

Karlsson, J. (2022, January 10). *MongoDB Schema Design Best Practices*. MongoDB. Retrieved January 16, 2024, from https://www.mongodb.com/developer/products/mongodb/mongodb-schema-design-best-practices/