(B8IT150) ADVANCED PROGRAMMING

Lecturer: Kanza Ali Manzar

Student: Ismael Carayol (20038258)

Advanced Programming CA

Recipe Book WebApp

High Diploma in Science in Computing

December 2025

Contents

1. Introduction

The goal of Recipe Book webapp project for Advanced Programming CA is to create a system that allows users to search, filter cooking recipes and once authenticated, create and manage their own recipes.

The project is clearly defined in a two-layer architecture, Business Logic Layer and Data Access Layer. It is ready to implement a Presentation Layer that would communicate with the rest of the system through an API with, now, 15 endpoints that will allow the user to experience all the Recipe Book webapp functionalities.

The motivation to do a project to search, filter, create, update and delete cooking recipes comes from my personal passion for food and cooking and the need I recognised of digital recipe systems that would allow users to share their cooking recipes between friends, family members or to the public.

2. System design

The Recipe book system uses an n-layer architecture and SOLID principles for which were key to separate the classes following the Single Responsibility Principle. This principle promotes that a class should only have a single, well-defined job or responsibility.

2.1. Entity Relationship Diagram (ERD)

The ERD shows the system database design, its entities and how their relationships are established.

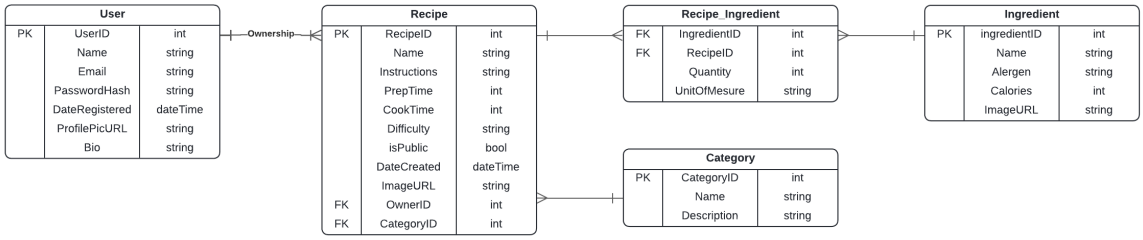

Figure 1. Entity Relationship Diagram

The RecipeBook system uses five entities to structure its data:

- User
- Recipe
- Category
- Ingredient
- RecipeIngredient (junction table)

These entities relationships are:

- One User can own many Recipes and a recipe can have only one owner.
- A Recipe can be listed only in one Category, but a Category can comprehend many Recipes.
- A Recipe can have many Ingredients and an Ingredient can be used in many Recipes. This creates a many to many relationship that is solved by creating the junction table RecipeIngredient.

RecipeIngredient is key on the system design since it allows to store specific data from an Ingredient in each Recipe, such as Quantity and UnitOfMeasure, while avoids duplicating Ingredient data and makes Ingredient reuse possible.

2.2. UML

As previously mentioned, classes are separated to hold single responsibilities in a two-layer architecture.

Model classes represent entities that match the database tables' structure. It doesn't contain logic, only properties. They act as a data carrier between layers making models reusable between them.

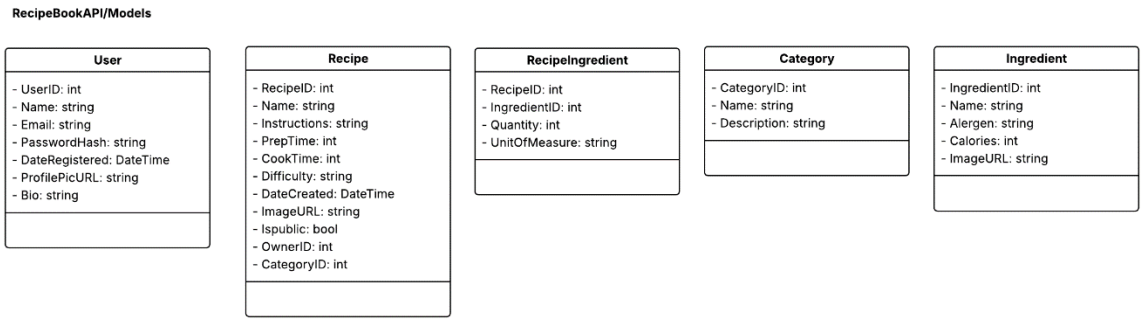

Figure 2. UML diagram for classes. Models

Repository classes handle all communication with the database, main function of the Data Access Layer. They abstract data access logic to exchange data between layers without affecting the business logic by encapsulating SQL queries from other parts of the application.

They are responsible for creating tables, insert, update, delete records and retrieve data from the database.

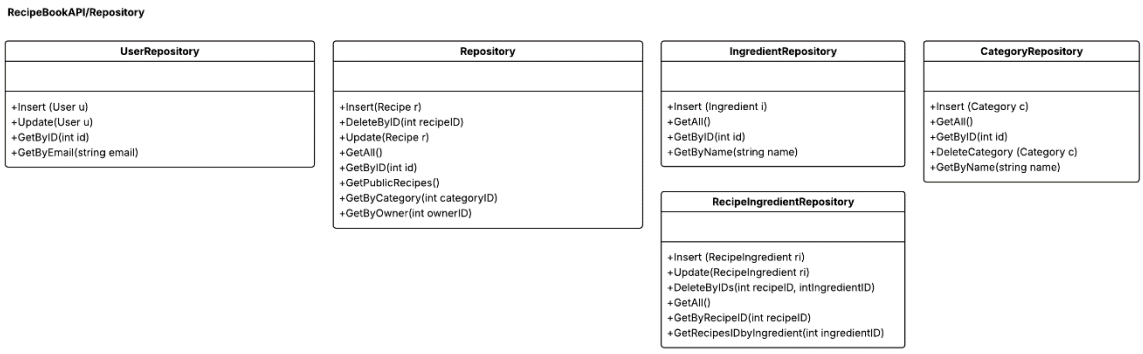

Figure 3. UML diagram for Data Access Layer. Repositories

Service classes contain the methods that implement the business rules and application features. They compose the Business Logic Layer and on this system are responsible for validating user input, enforcing rules such as ownership and permissions and applying searching and filtering algorithms while coordinating multiple repositories.

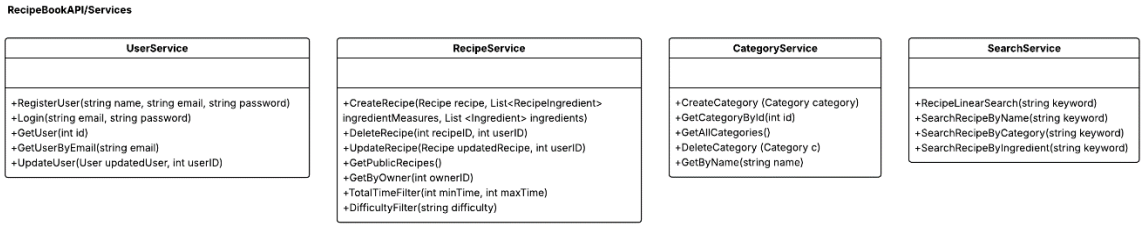

Figure 4. UML diagram for Business Logic Layer. Services

By separating models, repositories, and services, the system applies core OOP principles such as encapsulation, abstraction, and separation of concerns. This design improves maintainability, readability and scalability.

2.3. Flowchart

The following flowchart shows requests lifecycle from the moment a user interacts with the webapp creating a data request, flowing through all separated layers until the data is back being displayed to the user.

When a user creates a recipe through the UI (in the case of the system at the moment, through Swagger), the request will be sent, the controllers will receive the input, services will validate the information and apply the logic to create the recipe, the repositories will interact with the database creating the needed rows in Recipe, Ingredients and RecipeIngredients tables and finally a response will be returned.

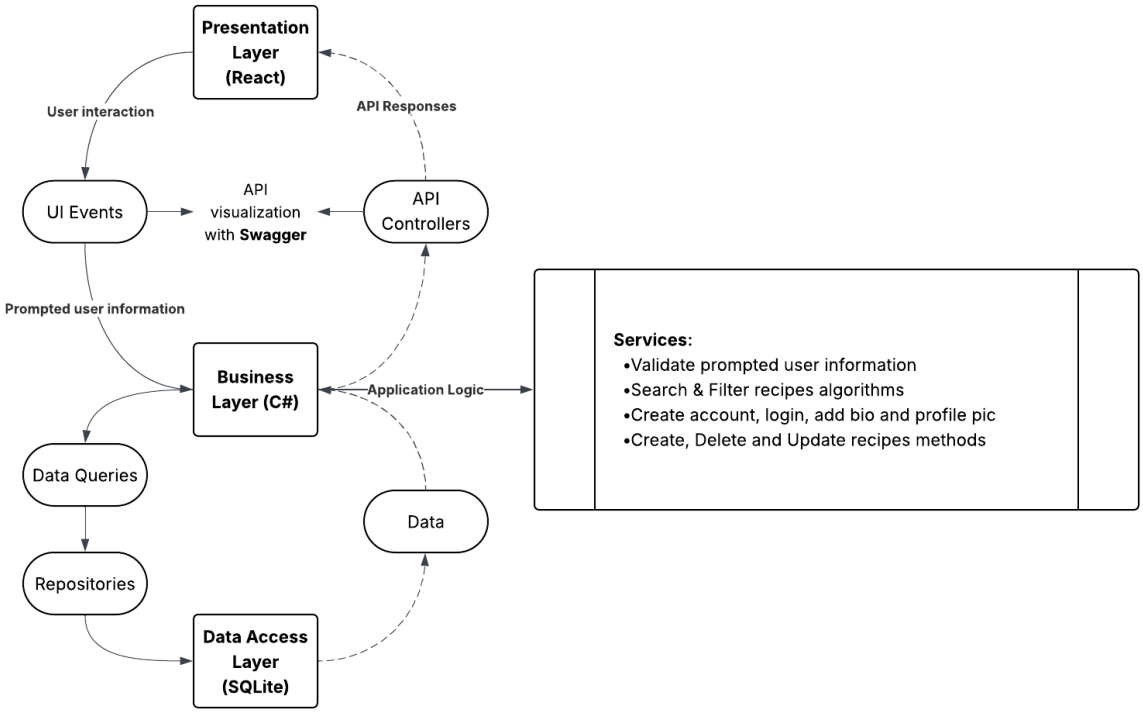

Figure 5. WebApp flowchart

3. Technical design decisions and performance considerations

This section covers key technical decisions taken during the design and development of the RecipeBook webapp project, including selection of data structures, algorithms and database design. It also evaluates the impact of these choices on performance, including time complexity and space usage.

3.1. Data structures used

Since RecipeBook system works with variable number of recipes, ingredients and dynamic search and filter results, the main specifications that are leveraged from Lists are:

- **Dynamic and flexible**. Lists can grow dynamically, there's no need to define a size beforehand.
- **Efficient iteration and manipulation.** This data structure supports operations such as adding, removing, sorting and modifying elements.
- **Integration with the project's searching and filter algorithms.**
- **.NET framework reliability**. This framework contains many Lists specific methods that when used means having to create less custom code and having fewer bugs.

3.2. Custom algorithms and data validation applied

Different custom algorithms were created on the Business Logic Layer per brief requirements that with the CRUD methods, compose the project's main functionality.

3.2.1. Searching algorithm

Given a keyword from user input, a linear search algorithm looks for a Recipe, Ingredient or Category that includes the keyword on their respective tables. Then, it returns a list of distinct recipes that had a match on their recipe name, ingredient name or category name.

Linear search was the searching algorithm design choice made based on the project's small dataset and implementation complexity.

3.2.2. Filter algorithms

Filtering was implemented by iterating through a list of public recipes and applying conditional checks by:

- Difficulty
- Total preparation and cooking time

Once the filter iterated through all the recipes, it returns a filtered list of recipes.

3.2.3. Data validation

Most features have different data validation depending on which layer or part of the request lifecycle is:

- **Controllers** check the request structure and provides different status answers if the body is missing, or queries have invalid parameters.
- **Services** ensure that business rules are applied by confirming that only owners can update or delete recipes, allowing users to login if the credentials are correct among other logical constraints.
- **Repositories** protect database integrity by doing null checks.

Multilayer validation improves security, prevents propagation of invalid data and makes the system more robust.

3.3. Time complexity, performance and space usage

The algorithms implemented on the project rely on linear iteration meaning that the average case will have a time complexity of O(n).

3.3.1. Searching recipes

Linear search starts at the beginning of the list, checks each element one by one and stops once a match is found.

The time complexity for this algorithm is **O(n)** for the average case. Based on the project's small dataset, this performance is acceptable.

3.3.2. Filtering

Filters iterate through all public recipes applies the filter condition and adds to a filteredRecipes list those recipes that match.

Time complexity is always **O(n)** since it has to go through all recipes every time. Space complexity is **O(k)** where k is the number of matched recipes.

4. Conclusion

This project successfully implements a structured cooking recipes management system using object-oriented principles and a 2-layer architecture.

By separating the application into Models, Services Repositories and Controllers, the system follows SOLID principles and improves maintainability and scalability.

Future planned implementations are:

- Responsive frontend design with React,
- Recipe Books functionality to share recipes between users,
- Safer authentication with third party sign in and more secure credential handling
- Like recipes functionality to allow users to save their favourite recipes.

Bibliography

Alaftekin, S. (n.d.). Retrieved from Medium: https://medium.com/@serhatalftkn/domain-driven-design-a-comprehensive-guide-to-validation-across-layers-8955d6854e7d

GeeksForGeeks. (2025, Aug 23). *GeeksForGeeks*. Retrieved from https://www.geeksforgeeks.org/system-design/solid-principle-in-programming-understand-with-real-life-examples/

Kumavat, S. (n.d.). *Medium*. Retrieved from https://medium.com/@sakshikumavat01/lists-in-coding-a-fundamental-data-structure-bd6d7dce0c4f

Quora. (n.d.). Retrieved from https://www.quora.com/What-is-model-class-in-Java