BIM 492 Design Patterns
Term Project

Recipe Management System

Emre Karabulut
Semih Kocadere
Ayberk Karakaş

The four types of design patterns we used in our project, why they were used, and how they were implemented are as follows:

1. Factory Pattern

- **Why used:** Factory Pattern was used in the Recipe Creation module because it delegates object creation to subclasses and generalizes the object creation process. In this case, it's a suitable choice for allowing users to create different types of recipes and managing this process with a factory class.
- **How used:** In the Recipe Creation module, classes representing different types of recipes were created, each implementing the Recipe interface. The FactoryRecipe class was used to instantiate the appropriate Recipe subclass based on the type input by the user.

2. Command Pattern

- **Why used:** The Command Pattern was employed in the Recipe Modification module because it encapsulates requests as objects, thereby allowing clients to parameterize operations with different requests. In this context, it's a suitable choice for representing operations like modifying recipes and undoing modifications.
- **How used:** The RecipeModificationCommand interface defines the commands that modify a Recipe. Classes implementing this interface, such as IngredientModificationCommand and InstructionModificationCommand, alter specific properties of a Recipe. Additionally, the Command Pattern was used to manage undo operations. Each modification command was stored in a data structure, and these objects were used when undoing operations.

3. Strategy Pattern

- **Why used:** The Strategy Pattern was utilized in the Recipe Search module because it defines a family of algorithms, encapsulates each one, and makes them interchangeable. In this context, it's a suitable choice for searching recipes based on different criteria such as tags, categories, or ingredients.
- **How used:** The SearchingRecipe class holds a strategy object and executes the search operation. This strategy object implements the SearchStrategy interface and is realized by classes representing different search strategies. For instance, classes like SearchingByTags

and SearchingByIngredients implement different strategies for searching recipes based on specific tags or ingredients.

4. Observer Pattern

- **Why used:** The Observer Pattern was applied in the Recipe Rating module because it defines a one-to-many dependency between objects so that when one object changes state, all its dependents are notified and updated automatically. In this context, it's a suitable choice for updating and displaying recipe ratings across different modules.
- **How used:** The Recipe class acts as a Subject, handling rating operations. Each user, upon rating a recipe, is added as an Observer. When a rating change occurs, all Observers are notified and updated with the new rating. This ensures that ratings made for any recipe are visible to all users.

Throughout our project, we aimed to use the most fitting design patterns to make our system work smoothly. We carefully picked and applied these patterns to ensure our system could easily adapt and stay organized. The Factory Pattern helped us create different recipe types easily, while the Command Pattern made it simple to undo changes. With the Strategy Pattern, searching for recipes became efficient, and the Observer Pattern ensured that ratings updated smoothly across the system. Overall, these patterns worked together to create a strong and user-friendly recipe management system.