Digital Cookbook

Course: SWENG 421: Software Architecture

Group members: Jason Cross & Dominick Carlucci

I. Motivation

1. What are the current problems?

Whatever happened to that recipe for grandma's cookies that everyone loves? Chances are, since it was written on an index card 50 years ago, it is either lost or unreadable. What are good meal options for someone on the Keto diet? Well, you will probably have to look it up online to know for sure. Cooking is something that nearly every person must do at some point. Some people love it, some people loath it. Making an edible meal can be as easy or difficult as we make it, let alone a healthy meal. Most people that make a meal that does not come out of a box might follow a recipe, whether it be one handed down over generations, one someone saw on the internet, or one that was just created by a happy accident. It can be a chore to scour the internet or wade through a mountain of cookbooks to find what you are looking for. Having all of those recipes in one place would be great, and it would assure us that grandma's cookie recipe will always be available.

2. What problems will be resolved?

We would like to create a digital cookbook that allows us to enter in any recipe we wish. Not only will this program allow us to enter recipes, but it will also allow us to design meals using certain criteria, such as a diet plan or meals with chicken as a main course. The app will store the recipes in categories, such as entrees, sides, desserts, etc. If you choose, you can enter the number of servings you need, and the app will adjust the ingredient amounts accordingly. The goal is for this app to make meal preparation as easy and organized as possible, and hopefully preserve recipes that may potentially end up lost to time.

II. Functional Requirements

Functional Requirements:

1. The system should be able to store as many recipes as needed per user.
2. The system should allow users to search the cookbook for recipes, by various criteria.
3. The system should allow users to tag certain recipes as favorites for faster retrieval.
4. The system should be able to recommend meals determined by user input.
5. The system should be able to adjust recipes depending on the number of servings needed.

III. Non-Functional Requirements

Non-Functional Requirements:

1. The system will be designed for Windows platforms with possible ports to other operating systems and mobile phones in the future.

2. The system should be intuitive, allowing users instant usability.
3. The system should handle potential errors effectively.
4. The system should be able to handle concurrent read access to recipes.
5. The system should be laid out in a logical manner.

IV. Pattern Choices

Creational pattern (Factory Method, Builder, **Abstract Factory** or Prototype).

Abstract Factory is chosen for the creational pattern and will piggy-back off of the decorator pattern for the user to customize diet plans. Doing so will be able to create different meals with a certain diet plan based off user parameters such as caloric intake and ingredients the user wants to use.

Partitioning patterns (Filter or **Composite**).

Composite is chosen for the partitioning pattern to categorize different recipes from all sections by ingredient (Example: I want all recipes that contain chicken in them). This implementation would be beneficial to someone looking to make certain recipes based on the ingredients they currently have and not have to go to the store if it is not required.

Behavioral patterns(Chain of Responsibility, Observer, **State**, or Visitor)

State is chosen as the behavioral pattern because the app will have different functions depending on user choices. Similar to the calculator, different choices in the app will change the state of the app to meet those needs. For example, choosing “Make A Meal” will have the app act as a meal planner. Choosing a large amount of servings for a recipe will have the app act as a calculator adjusting the ingredient amounts.

Structural patterns (Bridge, **Decorator**, or Dynamic Linkage).

Decorator is chosen for the structural pattern for the ease that a user can customize and make diet plans easily based on the caloric value that recipes yield. The Decorator pattern will delegate certain recipes to be selected based on the user inputted values of what their caloric intake for the day is.

Concurrency patterns (Scheduling, **Read/Write Lock**, Two-Phase Termination, or Future).

Read/write lock is chosen for the concurrency pattern to prevent a recipe from being modified at the same time as it is being accessed. This would cause a synchronization issue that may prevent the recipe from being accessed properly or functioning as it should.

V. Programming Language

The programming language chosen for this project is C#. There are a few reasons this language was chosen. The biggest reason for this choice is the ease with which a GUI can be created using C# and Visual Studio. This program will have multiple screens for the GUI and C# is the most efficient known language to do this. Additionally, we have used C# multiple times, and there are many patterns and/or algorithms that have already been developed using this language, and this will make incorporating any of them easier and more efficient.