Sequence diagrams

Strategy pattern

As there are more ways users can search recipes, strategy pattern could be used exactly for this purpose. As an example, users can search by the type of meal, the ingredients the user has, the ingredients the recipe uses, or by the difficulty of making the meal. This pattern could be implemented by adding a new attribute "strategy" to our "Recipe" class, and by adding a new "IStrategy" interface which would implement the ways the user can filter recipes.

State pattern

State pattern is used when dividing the users on Guest and Registered users. We need this division so we can give privileges to registered users. For this implementation, we added the "IMode" interface which can tell the application whether the user can use some methods or not. The interface implements methods "leaveFeedback" and "orderIngredientFromStore". Other needed methods can be easily added.

Template method

This pattern can be used in the "Feedback" class. It divides feedbacks based on if the feedback has a comment, or not. For this implementation, we added the template method "leaveFeedback" which will call on other methods. Classes "FeedbackWithComment" and "FeedbackWithoutComment" have been added, and these classes will use the implementation based on the needed logic.

Observer pattern

Observer pattern can be used in the way to divide the users on registered and guest users (registered users can be mailed "daily recipes of the day" and updates on newly added recipes). A new interface "ISubscribe" would need to be added which will implement the method "mailRecipes". The method would first check if the registered user chose to have subscription mails, after which the method would continue with its function.

Iterator pattern

This pattern could be used when searching recipes. The user can choose how he wants the recipes to be listed (alphabetically, by the number of ingredients, the number of ingredients the user is missing...). New interface "IIterate" would be needed which would call the method "giveNextRecipe". The interface would inherit the iterators specified above. We would also need a new interface "IChangeIterator" which would change the way user sees the recipes (based on what the user chooses).

Chain of Responsibility

There is no implementation in our system that would use this pattern.

Mediator pattern

The mediator pattern could be used if we added the "chat" option to our application (between an user and the administrator). There would be a mediator between these objects which would control the forwarding of the messages. This way, we would have control over everything in one place.