CSCI 4448 Spring 2016 Project Write-up 6

Ethan Wright
Julio Lemos
Nhan Nguyen

---

1. **Features Implemented**

ID	**Feature**
BR-003	Replication factor of 3 for data consistency
UR-001	User can search ingredients by category
UR-002	User can search ingredients by query
UR-003	User can enter dietary restrictions
UR-004	User can enter an optimal price range
UR-005	User can save a dietary preference/restriction
UR-007	User can generate recipe list at any point
UR-008	User can remove an ingredient from their list
NFR-002	Return at least 3 hits in under one second
NFR-004	Database results cached locally
NFR-005	Set partition size for Database

2. **Features Not Implemented**

ID	**Feature**
BR-001	Ingredients must be able to include sponsor links
BR-002	Collect user data for later processing
UR-006	User can select importance value of dietary restriction
NFR-001	Show 10 recipes upon search
NFR-003	Less than one second load time for initial website

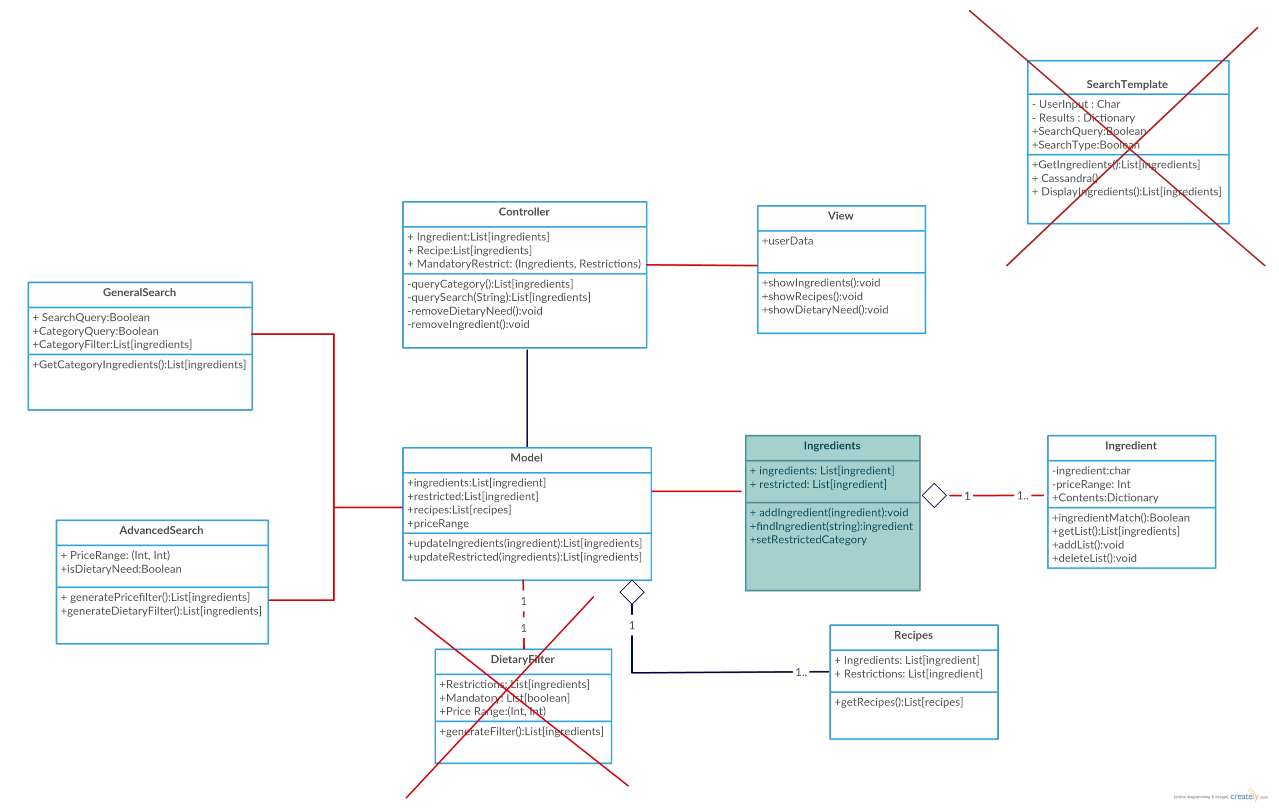
SearchTemplate
- UserInput : Char
- Results : Dictionary
+SearchQuery:Boolean
+SearchType:Boolean
+GetIngredients():List[ingredients]
+ Cassandra()
+ DisplayIngredients():List[ingredients]
Controller
+ Ingredient:List[ingredients]
+ Recipe:List[ingredients]
+ MandatoryRestrict: (Ingredients, Restrictions)
-queryCategory():List[ingredients]
-querySearch(String):List[ingredients]
-removeDietaryNeed():void
-removeIngredient():void
View
+userData
+showIngredients():void
+showRecipes():void
+showDietaryNeed():void
GeneralSearch
+ SearchQuery:Boolean
+CategoryQuery:Boolean
+CategoryFilter:List[ingredients]
+GetCategoryIngredients():List[ingredients]
Model
+ingredients:List[ingredient]
+restricted:List[ingredient]
+recipes:List[recipes]
+priceRange
+updateIngredients(ingredient):List[ingredients]
+updateRestricted(ingredients):List[ingredients]
Ingredients
+ ingredients: List[ingredient]
+ restricted: List[ingredient]
+ addIngredient(ingredient):void
+findIngredient(string):ingredient
+setRestrictedCategory
1
1..
Ingredient
-ingredient:char
-priceRange: Int
+Contents:Dictionary
+ingredientMatch():Boolean
+getList():List[ingredients]
+addList():void
+deleteList():void
AdvancedSearch
+ PriceRange: (Int, Int)
+isDietaryNeed:Boolean
+ generatePricefilter():List[ingredients]
+generateDietaryFilter():List[ingredients]
1
1
1
DietaryFilter
+Restrictions: List[ingredients]
+Mandatory: List[boolean]
+Price Range:(Int, Int)
+generateFilter():List[ingredients]
1..
Recipes
+ Ingredients: List[ingredient]
+ Restrictions: List[ingredient]
+getRecipes():List[recipes]

Figure 1: New Class Diagram

3. Design Patterns Used

i. *Mode View Controller*
For this design pattern we attempted to implement a class structure that followed a similar pattern. We didn't use the actual Spring MVC that we worked on in class, but instead tried to implement our own version of it in the Class Diagram as well as in the actual implementation of our program.

ii. *Chain of Command*
The chain of command pattern was much more difficult to actually implement in our software. Our initial class diagram implemented it fairly well (after we added it during refactoring) but we found that this was the most difficult to implement once we actually started designing. This was because once we started having to make changes to our design while coding it was hard to accurately maintain the chain of command without stepping back and remapping our core design. As a software

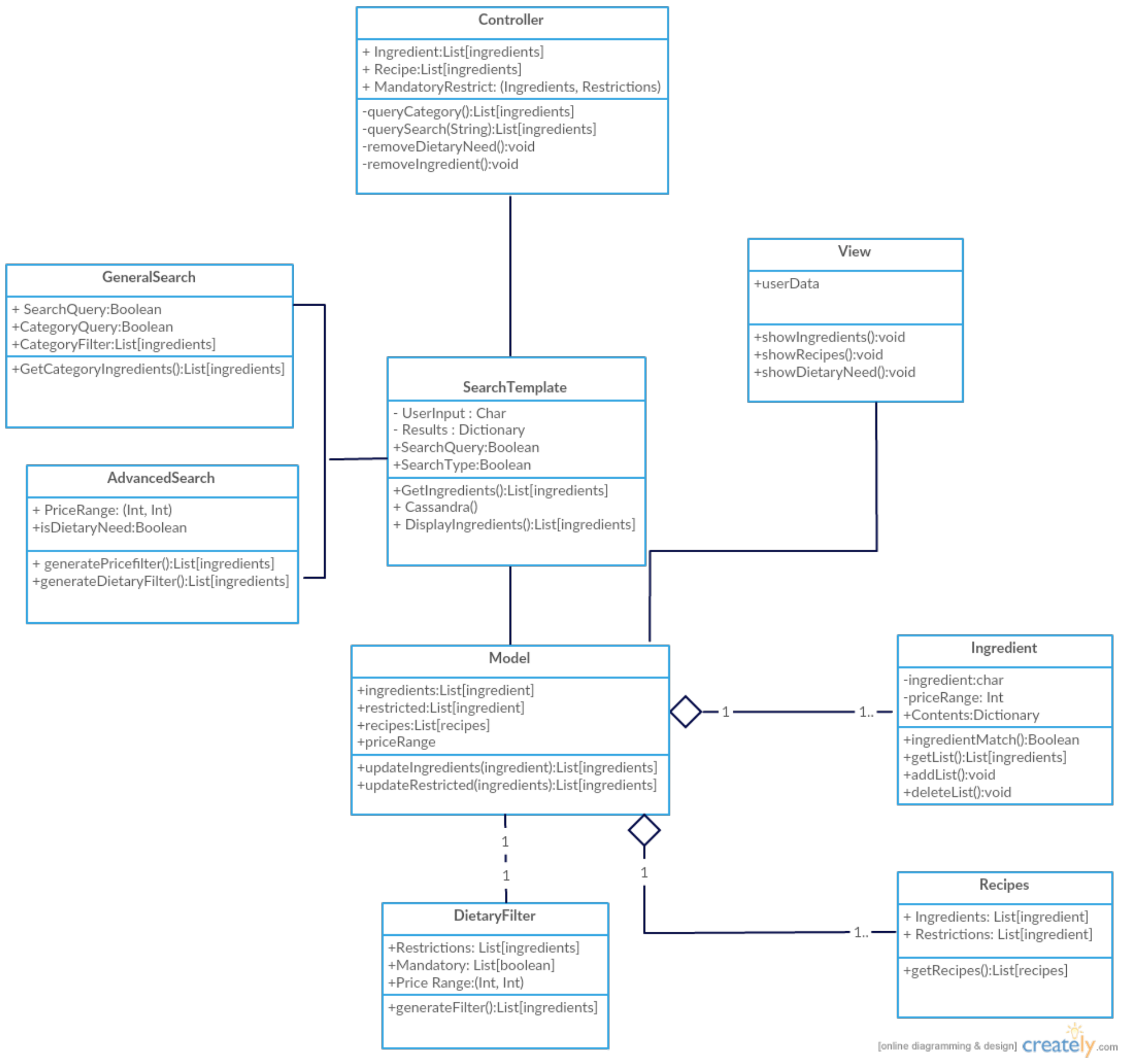
Controller
+ Ingredient:List[ingredients]
+ Recipe:List[ingredients]
+ MandatoryRestrict: (Ingredients, Restrictions)
-queryCategory():List[ingredients]
-querySearch(String):List[ingredients]
-removeDietaryNeed():void
-removeIngredient():void
GeneralSearch
+ SearchQuery:Boolean
+CategoryQuery:Boolean
+CategoryFilter:List[ingredients]
+GetCategoryIngredients():List[ingredients]
AdvancedSearch
+ PriceRange: (Int, Int)
+isDietaryNeed:Boolean
+ generatePricefilter():List[ingredients]
+generateDietaryFilter():List[ingredients]
SearchTemplate
- UserInput : Char
- Results : Dictionary
+SearchQuery:Boolean
+SearchType:Boolean
+GetIngredients():List[ingredients]
+ Cassandra()
+ DisplayIngredients():List[ingredients]
View
+userData
+showIngredients():void
+showRecipes():void
+showDietaryNeed():void
Model
+ingredients:List[ingredient]
+restricted:List[ingredient]
+recipes:List[recipes]
+priceRange
+updateIngredients(ingredients):List[ingredients]
+updateRestricted(ingredients):List[ingredients]
Ingredient
-ingredient:char
-priceRange: Int
+Contents:Dictionary
+ingredientMatch():Boolean
+getList():List[ingredients]
+addList():void
+deleteList():void
1
1..
1
1
1
1..
DietaryFilter
+Restrictions: List[ingredients]
+Mandatory: List[boolean]
+Price Range:(Int, Int)
+generateFilter():List[ingredients]
Recipes
+ Ingredients: List[ingredient]
+ Restrictions: List[ingredient]
+getRecipes():List[recipes]
[online diagramming & design] creately.com

Figure 2: Old Class Diagram

developer, it's certainly hard to take some time when coding to plan things out. As they say, days of coding save you hours of design. This was certainly the case for the implementation of chain of command.

iii. *Template*
Template proved to be highly beneficial for us during this project. We used template initially to split up our search into multiple levels, but then we used similar structures to help us implement our MVC. With our model class we were able to create set parameters for the user and the database that we could then extend to the given classes such as ingredients and recipe, and it was incredibly usefull to have the template design pattern in the back of our head when doing this construction.

4. **What we learned about process of analysis and design after implementing a system**

   The biggest take away about the process of analysis and design is the fact that it has to be an ongoing process. While we greatly enjoyed the aspects of the class that focused on the architecture of the project, we found that once we got to coding things started to shift that we weren't able to account for. Had we been using an agile approach that appropriatley accomodated for creating design patterns every day that we were going to do a programming sprint we likely would have been more sucessful. Instead we would spend a week or two doing our design and reworking it, and then we would dive into the programming agnostic to any change in design and powering through. This waterfall method of software development certainly hindered our prospect of sucess because we weren't able to shift our patterns and architecture of the project as we went.