Project Part Six

Team: Xiaolan Cai
Chi Chen
Andrew Guttman

Title: Ingredient Tracker

Project Summary: A web application that allows users to keep a digital pantry of ingredients, create and share recipes and search/filter/sort through recipes based on available ingredients and user metrics.

1. Use Case implemented:

For User

- UC - 001 -> Signup
- UC - 002 -> Login
- UC - 003 -> Manage pantry
  - UC - 004 -> Add items
  - UC - 005 -> Remove items
  - UC - 006 -> Adjust amounts
- UC - 007 -> Find a recipe
  - UC - 008 -> Filter recipes
    - String match
    - No new ingredients
- UC - 010 -> Manage custom recipes
  - UC - 011 -> Create recipe
  - UC - 012 -> Edit recipe
- UC - 013 -> Manage favorites
  - UC - 014 -> View favorites
  - UC - 015 -> Add recipe to favorites
  - UC - 016 -> Remove recipe from favorites
- UC - 017 -> Manage history
  - UC - 018 -> View history
  - UC - 019 -> Mark recipe as made
  - UC - 020 -> Remove recipe from history
- UC - 021 -> Get a shopping list to make recipe

2. Use Case not implemented:

For User

- UC - 009 -> Sort recipes

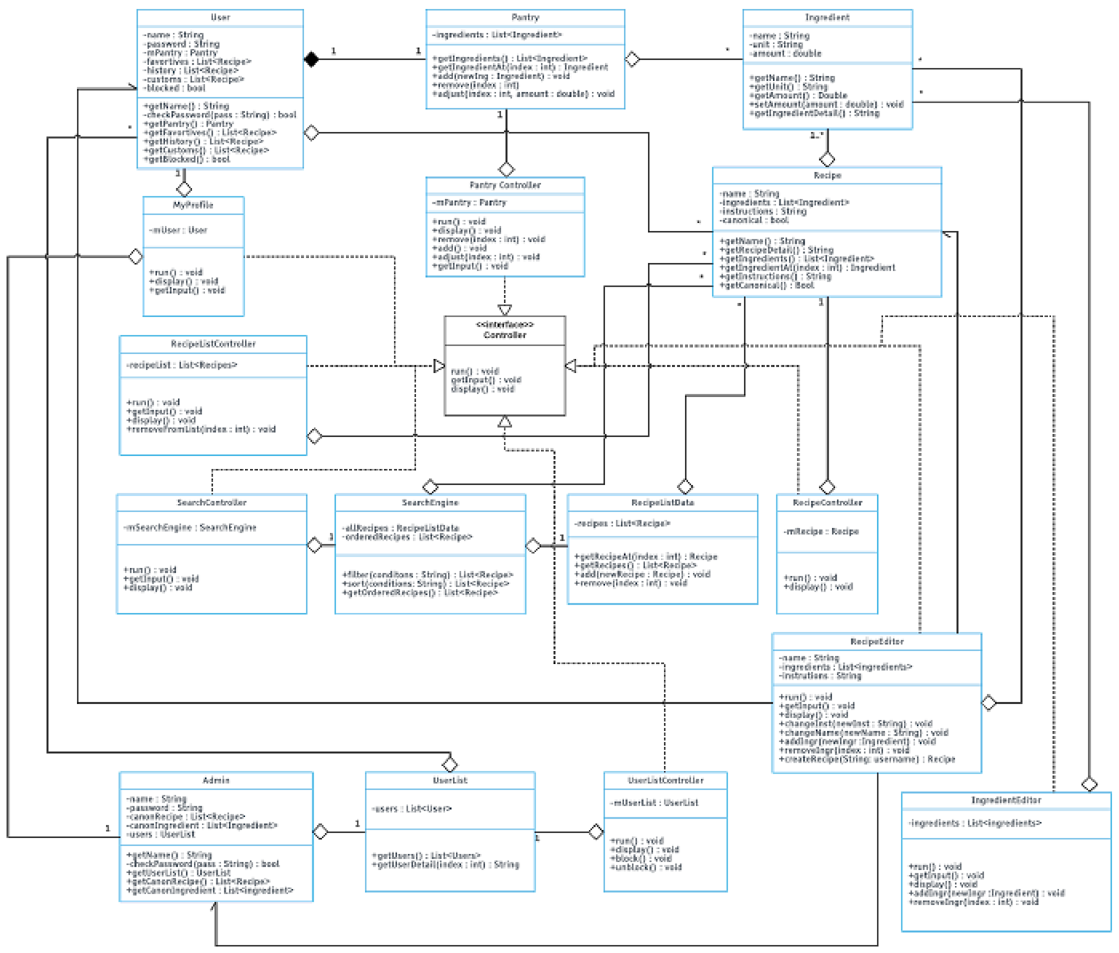
        - # of ingredients
        - Rating

For Admin

- UC - 022 -> Manage canonical recipes
    - UC - 023 -> Edit canonical recipe
    - UC - 024 -> Create canonical recipe
- UC - 025 -> Manage canonical ingredients
    - UC - 026 -> Add ingredient
- UC - 027 -> Manage users
    - UC - 028 -> View list of all users w/ their custom recipes

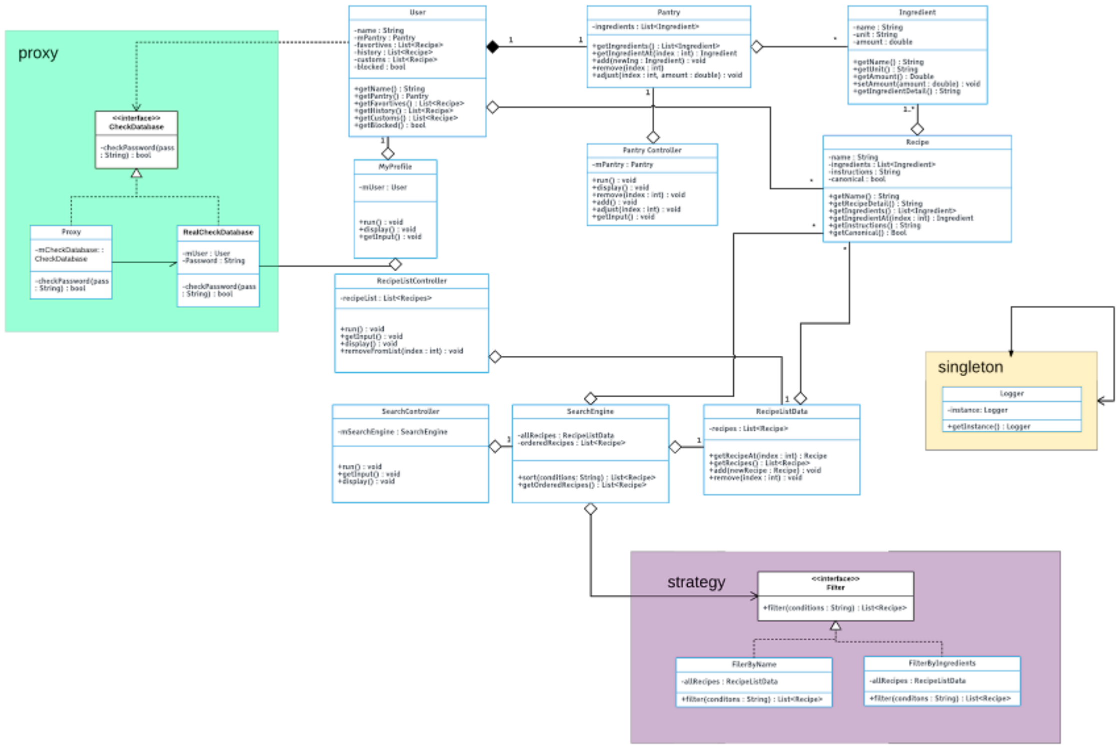
3. Show your Part 2 class diagram and your final class diagram. What change?

Original Class diagram:

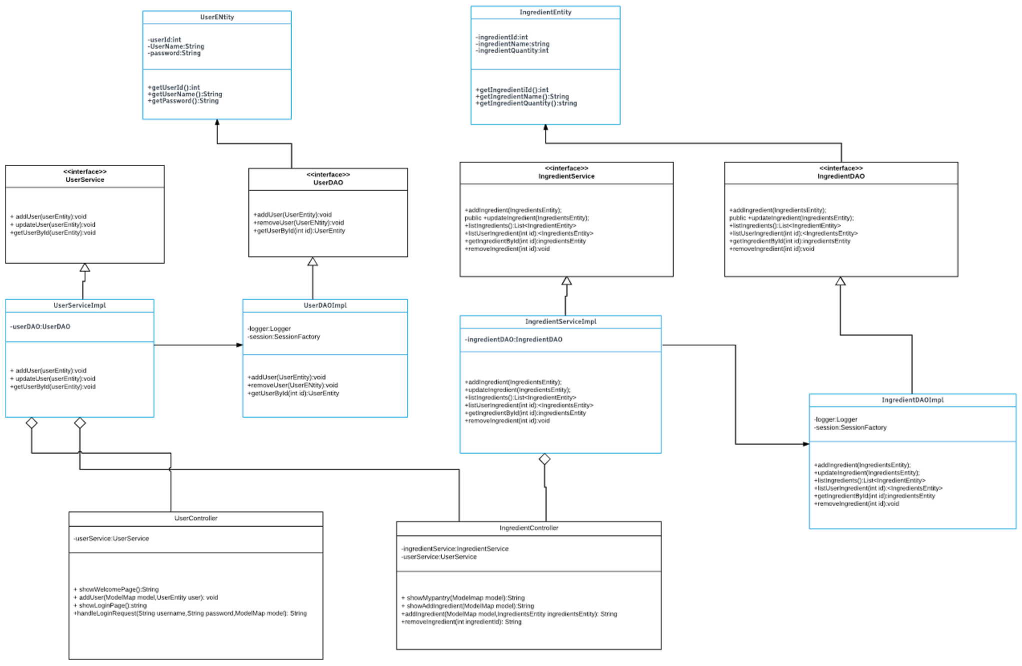

User
-name : String
-password : String
-mPantry : Pantry
-favortives : List<Recipe>
-history : List<Recipe>
-customs : List<Recipe>
-blocked : bool
+getName() : String
-checkPassword(pass : String) : bool
+getPantry() : Pantry
+getFavortives() : List<Recipe>
+getHistory() : List<Recipe>
+getCustoms() : List<Recipe>
+getBlocked() : bool
Pantry
-ingredients : List<Ingredient>
+getIngredients() : List<Ingredient>
+getIngredientAt(index : int) : Ingredient
+add(newIng : Ingredient) : void
+remove(index : int)
+adjust(index : int, amount : double) : void
Ingredient
-name : String
-unit : String
-amount : double
+getName() : String
+getUnit() : String
+getAmount() : Double
+setAmount(amount : double) : void
+getIngredientDetail() : String
MyProfile
-mUser : User
+run() : void
+display() : void
+getInput() : void
Pantry Controller
-mPantry : Pantry
+run() : void
+display() : void
+remove(index : int) : void
+add() : void
+adjust(index : int) : void
+getInput() : void
Recipe
-name : String
-ingredients : List<Ingredient>
-instructions : String
-canonical : bool
+getName() : String
+getRecipeDetail() : String
+getIngredients() : List<Ingredient>
+getIngredientAt(index : int) : Ingredient
+getInstructions() : String
+getCanonical() : Bool
<<interface>>
Controller
run() : void
getInput() : void
display() : void
RecipeListController
-recipeList : List<Recipes>
+run() : void
+getInput() : void
+display() : void
+removeFromList(index : int) : void
SearchController
-mSearchEngine : SearchEngine
+run() : void
+getInput() : void
+display() : void
SearchEngine
-allRecipes : RecipeListData
-orderedRecipes : List<Recipe>
+filter(conditons : String) : List<Recipe>
+sort(conditions: String) : List<Recipe>
+getOrderedRecipes() : List<Recipe>
RecipeListData
-recipes : List<Recipe>
+getRecipeAt(index : int) : Recipe
+getRecipes() : List<Recipe>
+add(newRecipe : Recipe) : void
+remove(index : int) : void
RecipeController
-mRecipe : Recipe
+run() : void
+display() : void
RecipeEditor
-name : String
-ingredients : List<ingredients>
-instrutions : String
+run() : void
+getInput() : void
+display() : void
+changeInst(newInst : String) : void
+changeName(newName : String) : void
+addIngr(newIngr :Ingredient) : void
+removeIngr(index : int) : void
+createRecipe(String: username) : Recipe
Admin
-name : String
-password : String
-canonRecipe : List<Recipe>
-canonIngredient : List<Ingredient>
-users : UserList
+getName() : String
-checkPassword(pass : String) : bool
+getUserList() : UserList
+getCanonRecipe() : List<Recipe>
+getCanonIngredient : List<ingredient>
UserList
-users : List<User>
+getUsers() : List<Users>
+getUserDetail(index : int) : String
UserListController
-mUserList : UserList
+run() : void
+display() : void
+block() : void
+unblock() : void
IngredientEditor
-ingredients : List<ingredients>
+run() : void
+getInput() : void
+display() : void
+addIngr(newIngr :Ingredient) : void
+removeIngr(index : int) : void
1
1
1
1
1..*
1
1
1
1
1
1

Final Class diagram:

1.The **prototype** class diagram:


proxy
<<interface>> CheckDatabase
-checkPassword(pass : String) : bool
Proxy
-mCheckDatabase : CheckDatabase
-checkPassword(pass : String) : bool
RealCheckDatabase
-mUser : User
-Password : String
-checkPassword(pass : String) : bool
User
-name : String
-mPantry : Pantry
-favortives : List<Recipe>
-history : List<Recipe>
-customs : List<Recipe>
-blocked : bool
+getName() : String
+getPantry() : Pantry
+getFavortives() : List<Recipe>
+getHistory() : List<Recipe>
+getCustoms() : List<Recipe>
+getBlocked() : bool
1
1
Pantry
-ingredients : List<Ingredient>
+getIngredients() : List<Ingredient>
+getIngredientAt(index : int) : Ingredient
+add(newIng : Ingredient) : void
+remove(index : int)
+adjust(index : int, amount : double) : void
*
Ingredient
-name : String
-unit : String
-amount : double
+getName() : String
+getUnit() : String
+getAmount() : Double
+setAmount(amount : double) : void
+getIngredientDetail() : String
1..*
MyProfile
-mUser : User
+run() : void
+display() : void
+getInput() : void
Pantry Controller
-mPantry : Pantry
+run() : void
+display() : void
+remove(index : int) : void
+add() : void
+adjust(index : int) : void
+getInput() : void
Recipe
-name : String
-ingredients : List<Ingredient>
-instructions : String
-canonical : bool
+getName() : String
+getRecipeDetail() : String
+getIngredients() : List<Ingredient>
+getIngredientAt(index : int) : Ingredient
+getInstructions() : String
+getCanonical() : Bool
RecipeListController
-recipeList : List<Recipes>
+run() : void
+getInput() : void
+display() : void
+removeFromList(index : int) : void
singleton
Logger
-instance: Logger
+getInstance() : Logger
SearchController
-mSearchEngine : SearchEngine
+run() : void
+getInput() : void
+display() : void
SearchEngine
-allRecipes : RecipeListData
-orderedRecipes : List<Recipe>
+sort(conditions: String) : List<Recipe>
+getOrderedRecipes() : List<Recipe>
RecipeListData
-recipes : List<Recipe>
+getRecipeAt(index : int) : Recipe
+getRecipes() : List<Recipe>
+add(newRecipe : Recipe) : void
+remove(index : int) : void
strategy
<<interface>> Filter
+filter(conditions : String) : List<Recipe>
FilerByName
-allRecipes : RecipeListData
+filter(conditons : String) : List<Recipe>
FilterByIngredients
-allRecipes : RecipeListData
+filter(conditons : String) : List<Recipe>

2.The **MVC architecture** class diagram:


UserENtity
-userId:int
-UserName:String
-password:String
+getUserId():int
+getUserName():String
+getPassword():String
IngredientEntity
-ingredientId:int
-ingredientName:string
-ingredientQuantity:int
+getIngredientId():int
+getIngredientName():String
+getIngredientQuantity():string
<<interface>> UserService
+ addUser(userEntity):void
+ updateUser(userEntity):void
+getUserById(userEntity):void
<<interface>> UserDAO
+addUser(UserEntity):void
+removeUser(UserENtity):void
+getUserById(int id):UserEntity
<<interface>> IngredientService
+addIngredient(IngredientsEntity);
public +updateIngredient(IngredientsEntity);
+listIngredients():List<IngredientEntity>
+listUserIngredient(int id):<IngredientsEntity>
+getIngredientById(int id):ingredientsEntity
+removeIngredient(int id):void
<<interface>> IngredientDAO
+addIngredient(IngredientsEntity);
public +updateIngredient(IngredientsEntity);
+listIngredients():List<IngredientEntity>
+listUserIngredient(int id):<IngredientsEntity>
+getIngredientById(int id):ingredientsEntity
+removeIngredient(int id):void
UserServiceImpl
-userDAO:UserDAO
+ addUser(userEntity):void
+ updateUser(userEntity):void
+getUserById(userEntity):void
UserDAOImpl
-logger:Logger
-session:SessionFactory
+addUser(UserEntity):void
+removeUser(UserENtity):void
+getUserById(int id):UserEntity
IngredientServiceImpl
-ingredientDAO:IngredientDAO
+addIngredient(IngredientsEntity);
+updateIngredient(IngredientsEntity);
+listIngredients():List<IngredientEntity>
+listUserIngredient(int id):<IngredientsEntity>
+getIngredientById(int id):ingredientsEntity
+removeIngredient(int id):void
IngredientDAOImpl
-logger:Logger
-session:SessionFactory
+addIngredient(IngredientsEntity);
+updateIngredient(IngredientsEntity);
+listIngredients():List<IngredientEntity>
+listUserIngredient(int id):<IngredientsEntity>
+getIngredientById(int id):ingredientsEntity
+removeIngredient(int id):void
UserController
-userService:UserService
+ showWelcomePage():String
+ addUser(ModelMap model,UserEntity user): void
+ showLoginPage():string
+handleLoginRequest(String username,String password,ModelMap model): String
IngredientController
-ingredientService:IngredientService
-userService:UserService
+ showMypantry(Modelmap model):String
+ showAddIngredient(ModelMap model):String
+addIngredient(ModelMap model,IngredientsEntity ingredientsEntity): String
+removeIngredient(int ingredientId): String

We removed all classes related to admin and its functions. We thought the project was already too large without admin's part, so we did not implement it, but it is possible that we will extend the project and implement that part in the future. There were also a few parts that were overbuilt, and we realized while implementing that some parts were unneeded or could be reused, eliminating their classes from the final prototype.

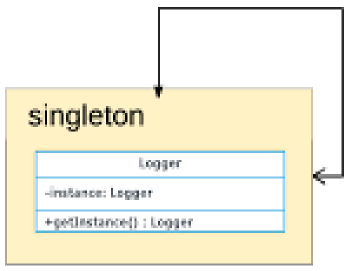
In User's part, lots of the structure didn't change a lot, but we removed the "controller" interface. Though we still make use of many "controller" classes, they started to diverge in functionality. If we had that interface, the code might be more organized in some sense, but we thought the use of an interface would imply a stronger relationship between those classes than they actually had in the implementation.

Programming after healthy design is really easier. There was less opportunity to mess up by naively going down an unproductive path because everything was thought out ahead of time. If there were changes to be made in implementation because of some unforeseen issue, it was still easy to adjust because the design process still forced us to consider so much, we could simply make the adjustment we needed while still aiming to fulfill the system as designed, ensuring that the program was still coherent. Most of the programming went as expected however, and moved quickly because figuring out what to build was a more involved problem than actually building it.

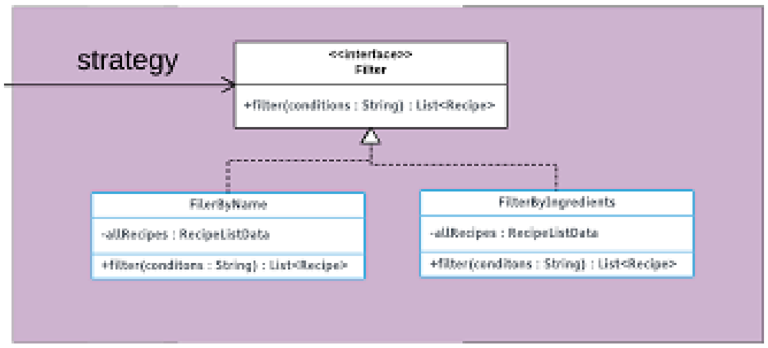
4. Did you make use of any design patterns in the implementation of your final Prototype?

4.1 Building the prototype:

Yes, we use the "strategy" and "singleton" patterns in the implementation of our final prototype.

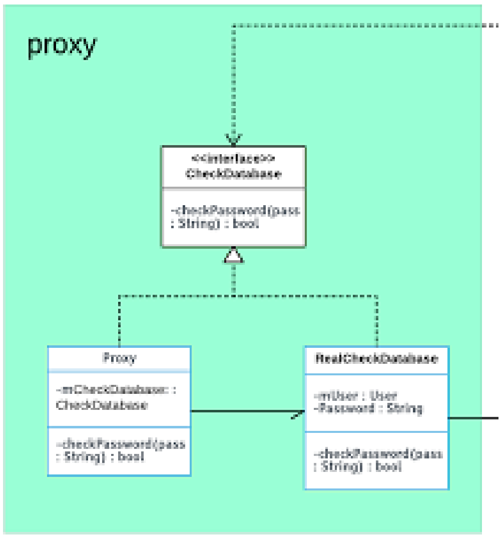


Singleton was used to make a logger.


strategy
<<interface>>
Filter
+filter(conditions : String) : List<Recipe>
FilerByName
-allRecipes : RecipeListData
+filter(conditons : String) : List<Recipe>
FilterByIngredients
-allRecipes : RecipeListData
+filter(conditons : String) : List<Recipe>

Strategy was used to give our different filtering methods a common interface to be accessed through.

However, we didn't have enough time to implement "proxy" pattern, but we really think about using "proxy" to make our program better.

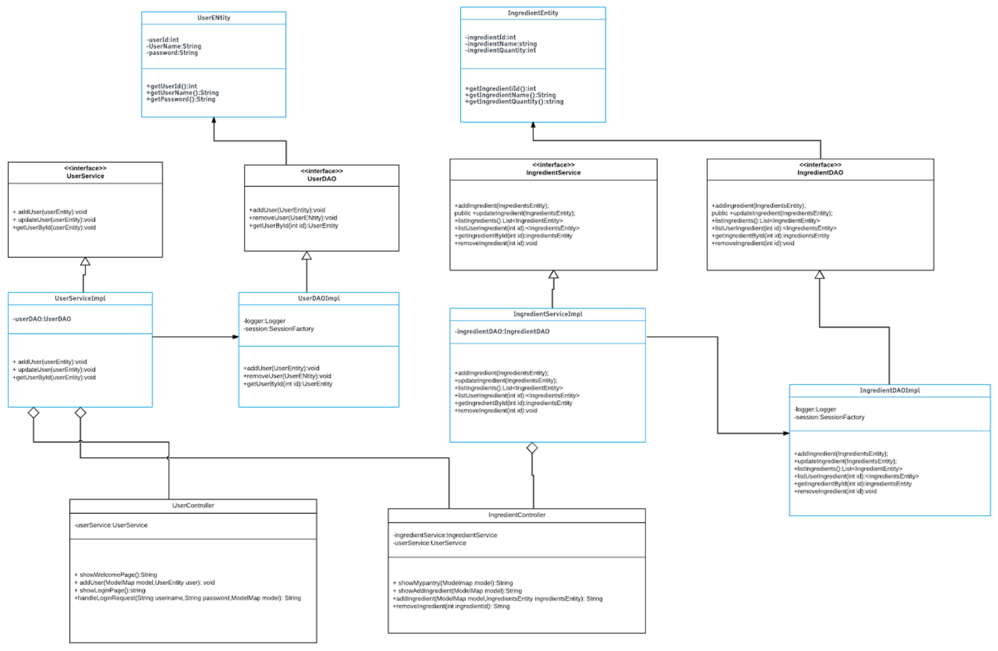

proxy
<<interface>>
CheckDatabase
-checkPassword(pass : String) : bool
Proxy
-mCheckDatabase : CheckDatabase
-checkPassword(pass : String) : bool
RealCheckDatabase
-mUser : User
-Password : String
-checkPassword(pass : String) : bool

When user logs in, the proxy pattern can protect user's information, improving the security of our program, which is really important to a website. Instead of just getting user's password in database and checking that with password typed in by user, we send the password typed in by user into database, then proxy will check password for us and return the result.

4.2 Building web application with MVC architecture:

The implementation includes model, controller, and view. Model is the data entity, which access and manage the user data and ingredients data , while view is the representation of the operations and controller takes the input from the user, convert it, and send command to the model and represent to the view. So, we have deployed strategy, observer and composite design patterns in the software engineering with the MVC architecture.

The class diagram is as follows:


UserENtity
-userId:int
-UserName:String
-password:String
+getUserId():int
+getUserName():String
+getPassword():String
IngredientEntity
-ingredientId:int
-ingredientName:string
-ingredientQuantity:int
+getIngredientiId():int
+getIngredientName():String
+getIngredientQuantity():string
<<interface>>
UserService
+ addUser(userEntity):void
+ updateUser(userEntity):void
+getUserById(userEntity):void
<<interface>>
UserDAO
+addUser(UserEntity):void
+removeUser(UserENtity):void
+getUserById(int id):UserEntity
<<interface>>
IngredientService
+addIngredient(IngredientsEntity);
public +updateIngredient(IngredientsEntity);
+listIngredients():List<IngredientEntity>
+listUserIngredient(int id):<IngredientsEntity>
+getIngredientById(int id):ingredientsEntity
+removeIngredient(int id):void
<<interface>>
IngredientDAO
+addIngredient(IngredientsEntity);
public +updateIngredient(IngredientsEntity);
+listIngredients():List<IngredientEntity>
+listUserIngredient(int id):<IngredientsEntity>
+getIngredientById(int id):ingredientsEntity
+removeIngredient(int id):void
UserServiceImpl
-userDAO:UserDAO
+ addUser(userEntity):void
+ updateUser(userEntity):void
+getUserById(userEntity):void
UserDAOImpl
-logger:Logger
-session:SessionFactory
+addUser(UserEntity):void
+removeUser(UserENtity):void
+getUserById(int id):UserEntity
IngredientServiceImpl
-ingredientDAO:IngredientDAO
+addIngredient(IngredientsEntity);
+updateIngredient(IngredientsEntity);
+listIngredients():List<IngredientEntity>
+listUserIngredient(int id):<IngredientsEntity>
+getIngredientById(int id):ingredientsEntity
+removeIngredient(int id):void
IngredientDAOImpl
-logger:Logger
-session:SessionFactory
+addIngredient(IngredientsEntity);
+updateIngredient(IngredientsEntity);
+listIngredients():List<IngredientEntity>
+listUserIngredient(int id):<IngredientsEntity>
+getIngredientById(int id):ingredientsEntity
+removeIngredient(int id):void
UserController
-userService:UserService
+ showWelcomePage():String
+ addUser(ModelMap model,UserEntity user): void
+ showLoginPage():string
+handleLoginRequest(String username,String password,ModelMap model): String
IngredientController
-ingredientService:IngredientService
-userService:UserService
+ showMypantry(Modelmap model):String
+ showAddIngredient(ModelMap model):String
+addIngredient(ModelMap model,IngredientsEntity ingredientsEntity): String
+removeIngredient(int ingredientId): String

5. What have you learned about the process of analysis and design now that you have stepped through the process to create, design and implement a system?

I think it will be very useful to do some analysis and design before you start to program, especially when you are starting a new large project. If you are careful enough in creating and designing part, you won't meet many problems when you are implementing the system.

However, it sometimes also takes lots of time to design the architecture of the whole program in the beginning, you need to imagine all the things you might meet in the middle. It really needs an amount of experience to do that properly, and if you think of something wrong in the beginning, it would cause bigger problems down the line.

But I still think it worth time to do some analysis and design before programming, lots of engineers said that the hardest part of programming wasn't in implementing but in debugging. If we can reduce the probability to produce bugs, we should try our best to do that!

We have learned how to design user requirements, user cases, how to draw UML diagrams to make the following implementation more efficient, and how to deploy design pattern solutions in the software development to make the implementation more effective. We have also learned new knowledge on Maven, Spring MVC framework with hibernate, HTML, Bootstrap, etc. while stepping through the process of implementing the software with the help of course materials from the class and guidance from the professor.