Ingredient Tracker

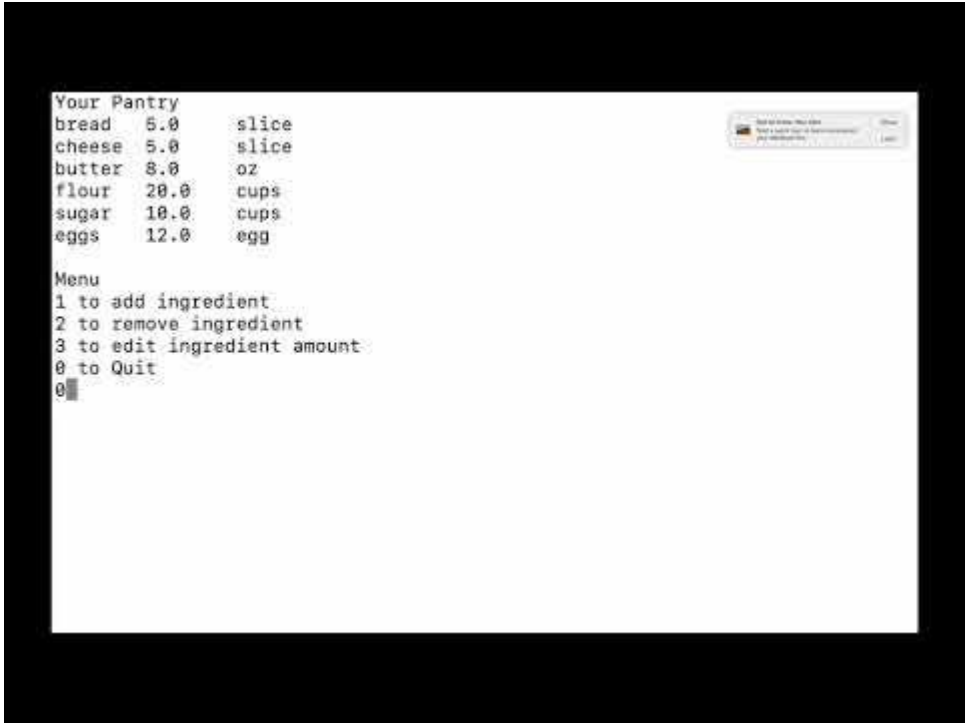
Xiaolan Cai, Chi Chen, Andrew Guttman

Project summary

A application that allows users to keep a digital pantry of ingredients, create and share recipes and search through recipes based on available ingredients and recipes names.

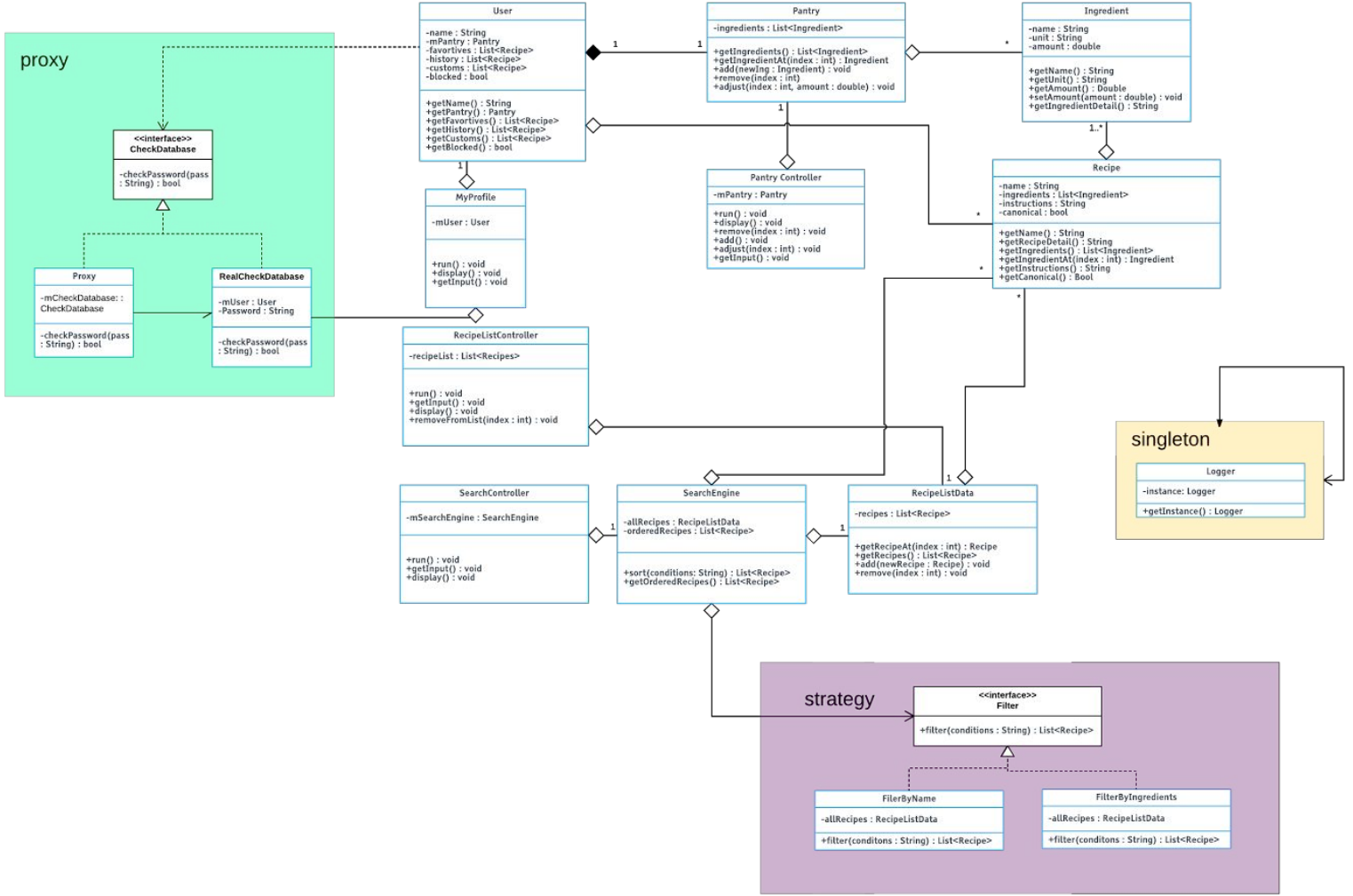
Demo time

https://youtu.be/cFeC9kh-r9I

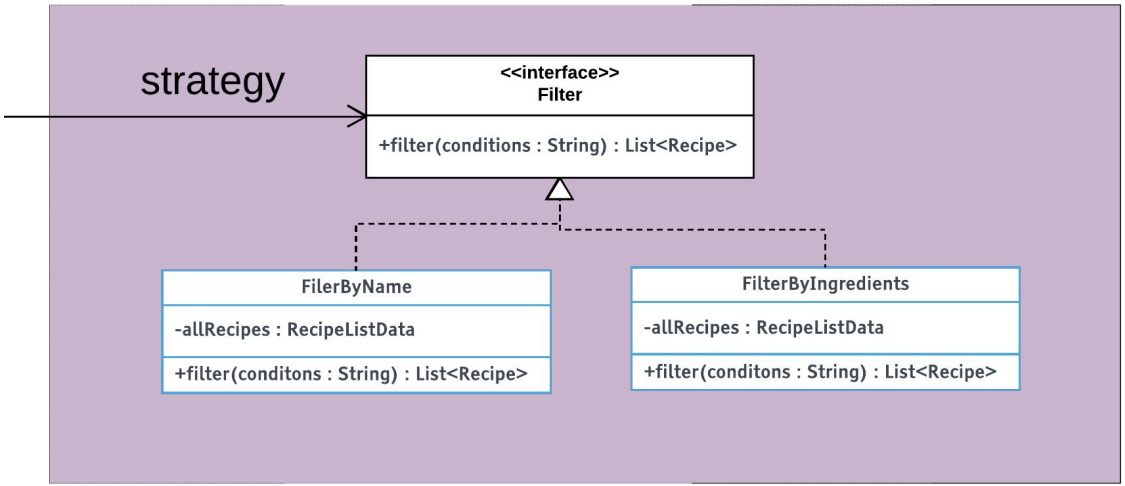


How we program based on diagram

- Programming after design is easy
- Less opportunity to mess up
- Figuring out what to build was a more involved problem than actually building it

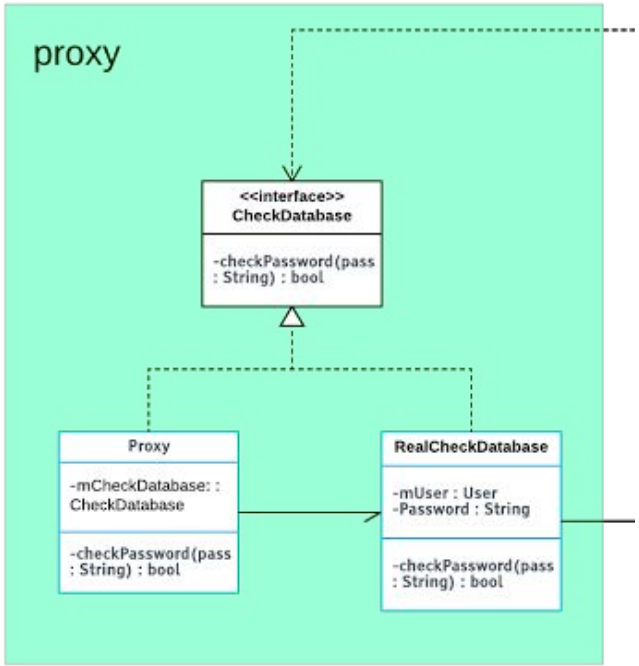

proxy
User
-name : String
-mPantry : Pantry
-favortives : List<Recipe>
-history : List<Recipe>
-customs : List<Recipe>
-blocked : bool
+getName() : String
+getPantry() : Pantry
+getFavortives() : List<Recipe>
+getHistory() : List<Recipe>
+getCustoms() : List<Recipe>
+getBlocked() : bool
Pantry
-ingredients : 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
<<interface>>
CheckDatabase
-checkPassword(pass : String) : bool
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
Proxy
-mCheckDatabase: : CheckDatabase
-checkPassword(pass : String) : bool
RealCheckDatabase
-mUser : User
-Password : String
-checkPassword(pass : String) : 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>
1
*
1..*

Strategy



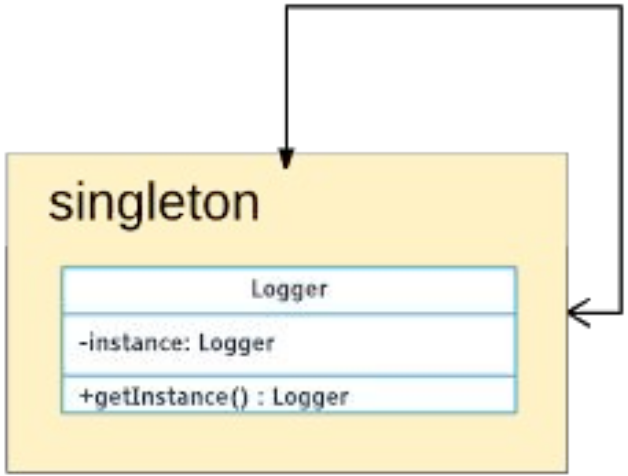
We use strategy design pattern in giving our different filtering method a common interface

Proxy



We use proxy design pattern to protect user's information in database

Singleton



We use singleton design pattern in making the Logger class

What we learn from this project

- How to design and analyze **user requirements**, **use cases**, etc. User requirements should be concise and easy to read from users' perspective
- **UML** diagrams make it easy to model the behavior of each component and visualize the interactions between them
- **Design patterns** make software engineering effective. We have used several design patterns for our software
- How to use **Maven**, **Spring MVC**, **hibernate** and other technologies to create a web application