CHOUTUANEAT.CO - Design Document

Group 8

Phase 2

Updated Scenario Walk-Through

After running our program, users can access the view (presenter) at localhost:8080/login (i.e. input this in any browsers) in order to log themselves in. Clients can either log in or register with a new username from the page. After successfully logging in to the website, users can view the homepage, where they can add a new dish or search for existing dishes. For adding a new dish, users will need to specify the name of their ingredients in the adding in a new ingredient page first. Then they can add a new dish by selecting the ingredients by names, specifying the name of the dish, specifying the weight per ingredient used, and filling in the instructions. For the function of searching an existing dish, clients can search for a dish by filling the name of the dish. Moreover, they can add the dish into their favourites. In the favourites page, they are able to delete dishes from their favourite list, as well as sorting their favourite dishes.

Significant Updates Since Phase 1

09/11/2021:

The practical implementation of user-related classes is now present. Including the implementation for business class userService, controllers for login, and register.

19/11/2021:

Favourite list functionality had been agreed to be reimplemented in phase 2. The conversion of the existing terminal-based functional relic had been converted to a spring-boot-based format.

20/11/2021:

The practical implementation of ViewFavouriteDishController has been written referring to services (methods) provided in favouriteService, and added to the SpringBootFavouriteBranch.

27/11/2021:

SpringBootFavouriteBranch has been merged with the main branch, with test cases for controllers written and updated.

Accessibility

Principles of Universal Design

Equitable Use

- In the future, for users with impaired visions, we could create a high-contrast theme similar to that in IntelliJ. We could also create a change-text-size feature that allows users to adjust text-size if they are too small for them to read.
- In the future, we could implement features that convert text-to-speech to further accommodate those who have vision impairment.

Flexibility in Use

- Since our program is currently only available as a website that runs locally on the user's device, in the future, we can make our website public so that users can access our program anywhere. We can also create a mobile app that allows users to access their recipes directly from their phone to provide users with more choices in method of use.

Simple and Intuitive Use

- For simple and intuitive use, we placed user-specific features, such as the logout button, into the top-navigation bar, separating them from the action-features like add dish, add ingredient, and back to homepage buttons. This is to organize information in a way that is consistent with user expectation, seeing that websites nowadays generally have user-related info on the top right corner of the page.
- Our program also prompts users when their request has been processed successfully or when something went wrong. For example, if requirements are not met for password creation, a warning shows up on the registration page. Or, when dishes have been added, a window pops up to notify them that the addition was successful.
- In the future, we could create a change language feature to accommodate users with different language abilities.

Perceptible Information

- We have currently broken down instructions for adding dishes into sections and numbered steps instead of a full paragraph telling the user what to do.
- Changing the color-scheme and text-size, as mentioned in Equitable Use, can also help in terms of Perceptible Information.

Tolerance for Error

- Our program allows users to remove dishes from their Favourites List in case they accidentally 'liked' a dish or decided to change their mind about a dish..
- In the future, we can expand this feature so users can delete ingredients and/or dishes.

- We could also add a feature that allows users to undo actions in case they accidentally removed something they didn't want to delete.

Low Physical Effort

- To help minimize repetitive actions, we created a database of ingredients so that users don't have to create the same ingredient over and over again when creating a dish that uses the same ingredient. Instead, the user can easily find the ingredient on the 'create dish' page.
- To minimize the amount of scrolling needed, we have implemented the Search feature to allow users to search for dishes. This is especially helpful as users add more and more dishes to the database.
- Furthermore, the navigation bars are also fixed, so that users don't have to scroll all the way to the top everytime they want to navigate through a website.
- Using our program for a long time can hurt the user's eyes. Hence, to be more user-friendly in the future, we could implement a feature that detects how long the user has been active on the website. After a specific amount of time, 1 hour for example, we can create a pop-up warning that reminds the user to take a rest.

Size and Space for Approach and Use.

- Since our program is entirely online and not a physical product, our program features are usable regardless of the user's body size, posture or mobility. Users may set up their laptops as they see fit when using the Choutuan website.

Target Market

Our program would be marketed to people of all ages who love to cook and bake. However, they find it hard to remember all of the recipes of the dishes they cooked or hope to try cooking someday. Choutuan.co allows them to keep a record of all the recipes and dishes so they can come back to them anytime, as well as explore new dishes shared by other users.

Unsuitable for…

Our program at its current stage is less likely to be used by those with impaired vision as there are no features that accommodate for those demographics. Our program is also less likely to be used by those who are unable to read English since English is the only language currently supported by our program. Lastly, our program is not suitable for those who do not have a device or does not have internet access since our program is run entirely online.

Progress Report

Contribution

Yuanyue Qiao: I have been working on creating CRC cards and UML and other information that need to be implemented in phase 1 (which is missing in our phase 1 report). Also, I have been working on editing all updates in phase 2 in the design document.

Emily Hu: I have been working on writing up the design document. Or, more specifically, the ‘Accessibility’ section in phase 2, and ‘Updated Specifications’, ‘Design Patterns’, and ‘Packaging Strategies’ in phase 1 portion). I have also made minor changes to frontend the html and css files.

Ruiyang Xu: I contributed mainly in aspects of major design decisions. For example the decision of reintroducing the favourite functionality. In addition, I refactored favourite related classes into spring-boot based format. Another significant contribution is the implementation of user-related functionalities. In detail, I constructed the first version of login and register controllers.

Yeming Chang: My important contribution is that I have compiled functions to add dishes, ingredients, dishes calories, and ingredients calories to the database. At the same time, there is the function of searching them in the database and displaying them on the webpage. And the first initial version of these features is displayed on the web page. In addition, I also wrote the test and docstring related to them.

Yuxuan Yang: I have been working on implementing the Dishes entity class in Phase 0, writing controller for favourites in Phase 1, and writing test cases for controller within the SpringBoot framework in Phase 2 (after we decided to use SpringBoot framework to automatically generate getters and setters for all the entities).

Yuxuan Liu: I mainly work on the development of frontend for user login(phase 1) and for favourite dish feature(phase 2). In phase 2, I also contribute to the refactor of favorite features and add some pop-ups to help users to increase accessibility.

Link to Significant Pull Requests

Yeming Chang:
https://github.com/CSC207-UofT/course-project-tut0101-choutuaneat-co-1/pull/45

This pull request is very important. First, I added some tests for entities, controllers and serviceimpl to ensure that they all run correctly and get the results we want. Secondly, related docstrings are added to controller, entities and service impl. This allows readers to know each function more clearly and quickly.

Yuanyue Qiao:

https://github.com/CSC207-UofT/course-project-tut0101-choutuaneat-co-1/actions/runs/1441357103

This is the pull request where our group decided to implement MVC instead of black box command line interface. All packages were updated to follow the clean architecture and the SOLID principles.

Ruiyang Xu:

09/11/2021

This is the pull request that constructs all user-related functionalities from the ground, including login and register.

https://github.com/CSC207-UofT/course-project-tut0101-choutuaneat-co-1/pull/26/files

19/11/2021

This is the pull request that refactors and reimplements favourite related functionalities from the relics of the terminal-based program.

https://github.com/CSC207-UofT/course-project-tut0101-choutuaneat-co-1/pull/36/files

Yuxuan Yang:

https://github.com/CSC207-UofT/course-project-tut0101-choutuaneat-co-1/actions/runs/1511207271

This is the pull request that merged SpringBootFavouriteBranch with the main branch, with important test cases for controllers and services written and updated.

Yuxuan Liu:
https://github.com/CSC207-UofT/course-project-tut0101-choutuaneat-co-1/pull/49

This pull request includes refactoring favorite feature and some pop-ups to increase accessibility. During refactoring, using dish use case class to deal with the curd operation of favorite list, reducing code smell such as Duplicate Code and Lazy Class.

Emily Hu:
https://github.com/CSC207-UofT/course-project-tut0101-choutuaneat-co-1/pull/37

This pull request consists primarily of frontend styling details. Instead of having a separate page for the search functions specifically, it was moved to all pages that displayed dish data. Minor styling details were also added to buttons. These changes allowed for better user accessibility and conforms to the Simple and Intuitive Use, Perceptible Information, and Low Physical Effort Principles.

Phase 1

Updated Specification

Since phase 0, we have added user account login functionalities so that each user has their own database of dishes and ingredients. We also implemented the Choutuan web GUI for better usage experience rather than the entire program running on the command line. In phase 1, we have also continued to implement the Favourite functionality, but due to changes to Choutuan's program design structure, the Favourite functionalities are not yet runnable. We expect to finish all Favourite implementations in the early stages of phase 2. Lastly, also due to revisions in the program design structure, Cooking Method functionalities (Fried, Boiled, Steamed, Grilled) have been taken out temporarily.

Program Structure Diagrams

CRC Cards

User (Entity)	
- Getter and setter of id - Getter and setter of username - Getter and setter of password	RegisterController (Controller) LoginController (Controller)

Ingredients (Entity)	
- Getter and setter of id - Getter and setter of ingredientsName - Getter and setter of caloriesPerGram - Getter and setter of weight - Getter and setter of dishesId	DishesIngredients (Entity) Dishes (Entity) IngredientsController (Controller)

Dishes (Entity)	
- Getter and setter of id - Getter and setter of dishes ingredient list - Getter and setter of ingredient list - Getter and setter of instructions - Getter and setter of cooking methods	Ingredients (Entity) DishesIngredients (Entity) DishesController (Controller)

DishesIngredients (Entity)	
- Getter of ingredients id - Getter of dishes id - Getter of weights	Ingredients (Entity) Dishes (Entity)

DishesIngredientsPrimaryKey (Entity)	
- Getter of ingredients id - Getter of dishes id	Ingredients (Entity) Dishes (Entity)

UserFavoriteDishes (Entity)	
- Getter of user id with dishes id	User (Entity) Dishes (Entity)

UserFavoritePrimaryKey (Entity)	
- Getter of user id and dishes id	User (Entity) Dishes (Entity)

RegisterController (Controller)	
- Register a new user - Store the new user information into the database - Connect the front-end (html) and the back-end (program)	User (Entity)

LoginController (Controller)	
- Check out the database to see if user exist - Logging users into the home page - Connect the front-end and the back-end	User (Entity)

ViewFavouriteDishController (Controller)	
- Allow users to get favourite dishes by id (return favourites) - Allow users to save or update users' favourite dishes - Allow users to get list of favourite dishes by user id (return list) - Allow users to get sorted favourite dishes by user id (return list of Dishes)	Favourites (Entity)

IngredientsController (Controller)	
- Manipulate the ingredients database. (i.e. allow users to add, get or delete ingredients from the database)	Ingredients (Entity)

DishesController (Controller)	
- Allow users to access search dishes or add dishes pages - Manipulate the dishes database (i.e. can add, get or delete dishes from the database)	Dishes (Entity)

UserRepository (Gateway) – Interface	
- Create, read, update, or delete the user database	User (Entity) RegisterController (Controller) LoginController (Controller)

IngredientsRepository (Gateway) – Interface	
- Create, read, update, or delete the ingredient database	Ingredients (Entity)

DishesIngredientsRepository (Gateway) – Interface	
- Create, read, update, or delete the dishes and ingredient database with ids of dishes and that of ingredients	DishesIngredients (Entity)

DishesRepository (Gateway) – Interface	
- Create, read, update, or delete the dishes database with dish name	Dishes (Entity)

UsersFavoriteDishesRepository (Gateway) – Interface	
- Create, read, update, or delete the favourite database with favourite list for each user	Dishes (Entity)

DishesService (Use Case Interface)	
- Get dishes by id of dishes or name of dishes - Save or update dishes - Assemble dishes - Delete dishes - Adding dishes into favourite list	DishesRepository (Gateway) UsersFavoriteDishes (Entity) Dishes (Entity)

IngredientService (Use Case Interface)	
- Get ingredients by name - Save or update ingredients - Delete ingredients	IngredientServiceImpl (Use Case)

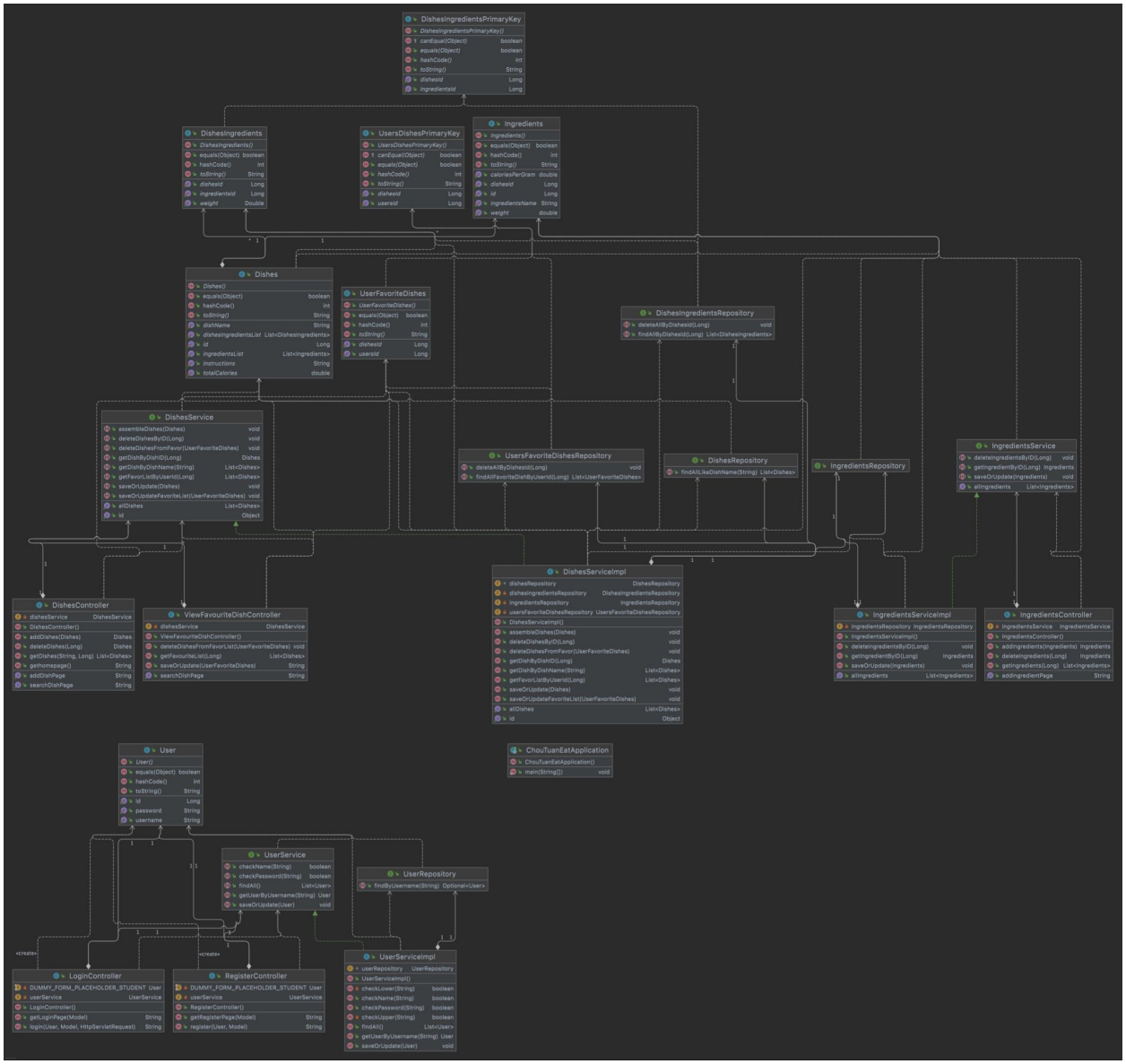
User Service (Use Case Interface)	
- Get users by usernames - Save or update users - Delete users - Check the usernames and passwords	UserServiceImpl (Use Case)

DishesServiceImpl (Use Case)	
- Search for dishes by dishes name - Save or update dishes - Delete dishes	Dishes (Entity) DishesService (Interface) DishesRepository (Gateway) DishesController (Controller)

IngredientsServiceImpl (Use Case)	
- Get ingredients by ingredient names - Save or update ingredients - Delete ingredients	Ingredients (Entity) IngredientsService (Interface) IngredientsRepository (Gateway) IngredientsController (Controller)

UserServiceImpl (Use Case)	
- Modify the user indicated by the given username - Return all the users in database - Check availability for username and password	User (Entity) UserService (Interface) UserRepository (Gateway) LoginController (Controller) RegisterController (Controller)

UML


DishesIngredientsPrimaryKey
DishesIngredients
UsersDishesPrimaryKey
Ingredients
Dishes
UserFavoriteDishes
DishesIngredientsRepository
DishesService
UsersFavoriteDishesRepository
DishesRepository
IngredientsRepository
IngredientsService
DishesServiceImpl
DishesController
ViewFavouriteDishController
IngredientsServiceImpl
IngredientsController
User
ChouTuanEatApplication
UserService
UserRepository
UserServiceImpl
LoginController
RegisterController

Major Design Decisions

- Converting the program from terminal-based to a web app:

Although our program's functionality is useful in a broad spectrum, the aspect of user-friendliness continues to be a concern. Under the unified agreement of all team members, we decided to convert our existing program to a spring-boot-based web app in pursuit of user-friendliness. We understand such moves would imply refactoring the whole structure, reconsidering each component's functionality, and adding external burdens for ourselves. However, the most important factor of our move is to force ourselves to consume experiences of composing a good application. Hence, all of us consider such a move to be, though, difficult, yet meaningful.

Design Patterns

In phase 0, our command line interface had too many responsibilities, such as presenting frontend displays, receiving user input, and calling Use cases to manipulate data. Therefore, we decided to implement the Model-View-Controller (MVC) design pattern in this phase. The MVC design pattern will allow us to effectively decouple frontend GUI displays, data manipulations and the data itself.

This design pattern is composed of three main interconnected components: the model consisting of Dishes, Ingredients and User entities, the view consisting of the user GUI, and the controller consisting of controller classes for Dishes, Ingredient, Login, and Registration.

The model is responsible for receiving user input from the controllers, and storing the data in the database. The view specific to our program presents the model data on an html webpage. The controllers connect view and model functionalities by being notified on user actions and updating model data accordingly via the Use case methods.

Clean Architecture

- From our packaging, the entity package indicates Enterprise Business Rules, the use case package indicates application business rules, the Controllers and Repositories (Gateways) as well as the presenters indicate the interface adapters. Hence, our program design strictly follows the clean architecture.

SOLID Design Principles

- Single Responsibility Principle: each class in our program is only responsible for one job
- Open-closed Principle: entity of our program can be adapted and remain unchanged.
- Liskov substitution principle: our classes can be replaced by its subclasses without any error.
- Interface Segregation Principle: our classes are only performing behaviors which achieves its own functionality.
- Dependency Inversion Principle: the interfaces in the use case classes can be substituted by any other implementations.

Packaging Strategies

We decided to package by layer since it illustrates the idea that implementations within classes of one layer should not know the specific implementations of files from another layer. More importantly, separating by layer also allows for a better view of dependencies across layers so that we can continuously check if our program adheres to Clean Architecture as we revise, implement, and add design patterns to our program.