FACH HOCHSCHULE LÜBECK

University of Applied Sciences

Software Engineering II Project

Digital Cookbook

Part task: Display anticipated taste by tags

Group Currywurst

Bing, Guanqi

Song, Yuchao

Ma, Qiang

Zhang, Xiaoyue

Shan, Xue

Content

1. Specification

1.1. Description

1.1.1. Digital cookbook

There has been a growing number of people who are not only satisfied with their local dishes and meantime have an increasingly strong desire to taste exotic dishes cooked by themselves.

However, the traditional cook books which are only able to provide a few choices and are impossible to include a huge selection of recipes due to the limited physical size, can no longer meet the demand of the public.

Therefore, we hereby would like to introduce our new digital cook book which will no longer occupy any physical space, only some bytes in your digital equipment instead. Thanks to the vast variety, this digital cook book enables you to cook a table of fantastic dishes easily if you follow our user-friendly guidance. We are confident in providing you with an even further horizon of delicacies.

1.1.2. Additional Task

Our digital cook book can not only demonstrate certain procedures, the flavour information will be also written in tags along with the name, such as sweet, spicy, sour, salty, which is intended for avoiding the circumstance where our customers attempt to make a new dish which was never heard of before, but it consequently failed to match the anticipated taste.

Furthermore, the user can also utilize the filter to select a certain flavour to see all the relevant dishes, which can boost efficiency to a great extent, saving much time for the user.

1.2. Product functions

It shows:

- The name of a dish
- The region where the dish is originated
- The preparing time and cooking time
- The amount of served people
- Ingredients and their amount
- The procedures of cooking
- Anticipated taste by tags

The user can search a dish with:

- A certain name
- A certain ingredient
- A certain flavour

The user can also:

- Add new recipes
- Dynamically revise serving number
- Edit recipes
- Delete recipes

1.3. User characteristics

There are majorly three kinds of users who will benefit from our project: the ordinary people who want to search for recipes, the users like cooks who need special and classic recipes and the users who wish to store their new recipes conveniently and efficiently.

1. The ordinary people are the major users of our software, who find the exact recipes so that they can cook with the help of our cookbook. To meet their demands, we need to design a quick and effective searching engine so that they can find out the target recipes as soon as possible.
2. The advanced users generally focus on the specific descriptions of some recipes. They need to get the details of the particular dish and judge the dishes

by all the information they want. In order to optimize the details, our cookbook will provide various taste tags to distinguish different kinds of recipes.

3. The third kind of users are the people who would like to store their own recipes in our digital cookbook. They can save all their recipes into the digital cookbook. For the sake of their special needs, we have provided the GUI for the users so that they can fill in the information related to the recipes by several steps.

1.4. Functional requirements

The functional requirements of this application will be divided into a number of aspects.

1. The application should provide those who are hesitating to decide what to cook with various recipes as solutions, which should include at least one option to fulfil the customer's expectation. This means the application is required to contain a significant number of recipes almost of every flavour, from every country, with plenty of ingredients. Every recipe should begin with its name and be described in detail including preparing time, cooking time, ingredients, tools needed and the steps.
2. Based on the numerous recipes embodied in the application, all the cuisines should be labelled with several tags like flavour, ingredients needed and so on, and then sorted according the labels.
3. When all the recipes are well labelled, the search engine should be designed for customers to find the cuisines related to the entry they input. Three ways can be used: searching by name, ingredient and flavour.

1.5. Non-functional requirements

Our digital cookbook does not only focus on providing specified functions for the customers but also concentrate on how to keep the system work stably and smoothly. Non-functional requirements are divided into four parts.

1. The first part is usability. It is important to ensure that the most frequently used functions should be written correctly. Our customers are not willing to see the book always crashing with different kinds of errors. Meanwhile, the accuracy of the digital book should also be promised. This is the base of our product.
2. Then it is the part of reliability. Users include customers and programmers have to trust our system after using it for a long time. When using our digital book, users are willing to use our cook book every time. So, we need to regularly refresh our database and update our information. The data stored in the database should also be protected from being deleted or modified by accident.
3. How our system can perform is also an essential issue. For example, the time of response should not be more that 3 seconds when users want to find the recipe at peak time. Considering the experience of users, we are obliged to keep the response time almost the same under any circumstances.
4. Maintainability is also a critical aspect for developers. A good product has to be written in a way easy to maintain. It should also include some test documentation to prevent the product from being illegally operated.

2. UML Specification

2.1. Use Cases

Use cookbook app to manage recipes

Primary actor: The user

Precondition:

- Cookbook app is activated
- MySQL database is activated

Basic flow of events:

1. The user adds recipes and inputs information.
2. The user inputs key words of recipe or ingredient to search recipes.
3. A list of results will be demonstrated to the user.
4. The user can check the profiles of recipes, click one of the them and skip to the view page.
5. The user sees all the information of the recipe in detail.
6. The user can click “Edit” button and edit the information of the recipe.

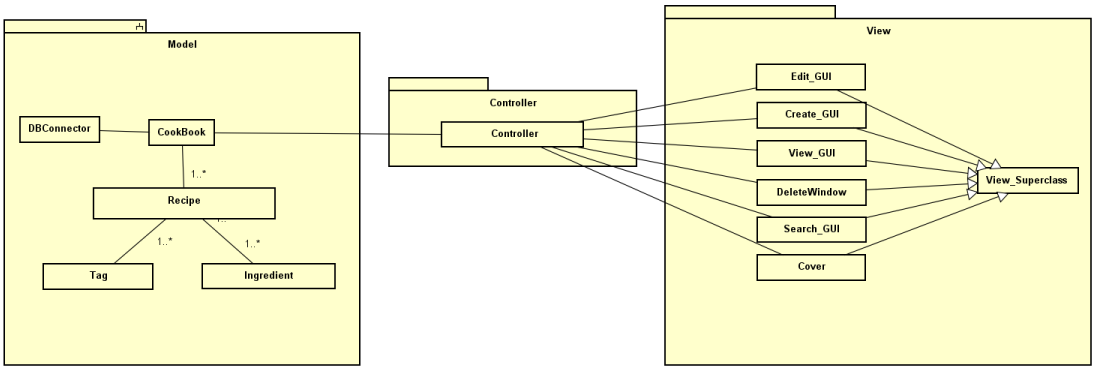
Alternative flows:

2a. The user can select different taste tags to specify the target recipes.

5a. The user can change the number of served people and the amount of ingredients will change dynamically.

5b. The user can delete the current recipe.

2.2. Class Diagrams



The overview of our class diagram

In the application, we use MVC pattern to separate Model, Controller and View, which improves the safety of the information from database. It can also help us to manage each part of codes with ease.

In the "View" part, we configured a superclass for the six GUI classes to extend. The GUI classes manage the static frames and the words demonstrated to users.

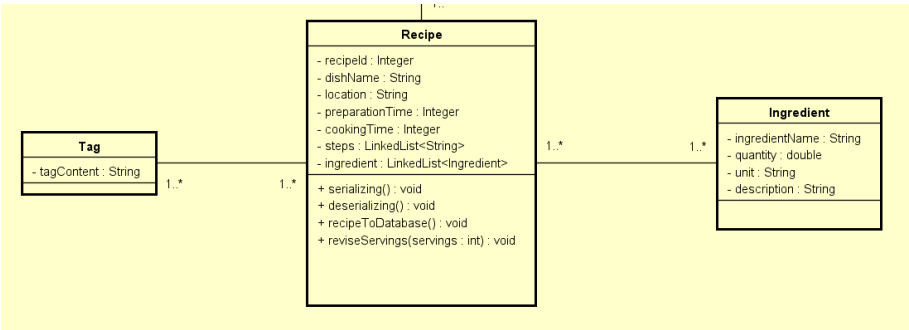
In the "Controller" part, we configured the Controller class which maintains the listeners from the GUI classes and derives information for the CookBook class.

In the "Model" part, we configured the entity of our data structure-- the Recipe class, the DBConnector class which communicates with database and the CookBook class which takes charge of all the recipes while communicating with the DBConnector class.

Model

Our data structure of recipe is built in a way that data can be easily retrieved and easy for us to set new values. It is also confirmed with the logic structure of a recipe.

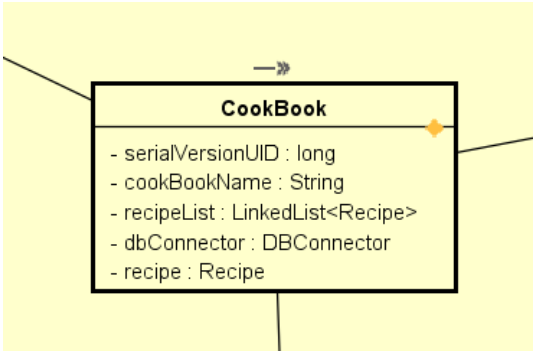
"Recipe", "Ingredient" and "Tag" class:



The "Tag" class and the "Ingredient" class both have "many-to-many" relationship with one recipe. The instances of these two classes are stored into two "LinkedList" respectively in the recipe.

With the help of the structure, we can retrieve any tags or ingredients by using only one "Recipe" object.

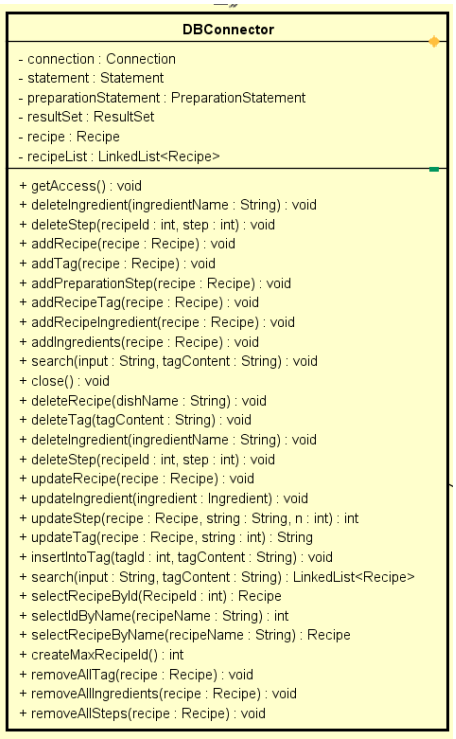
"CookBook" Class:



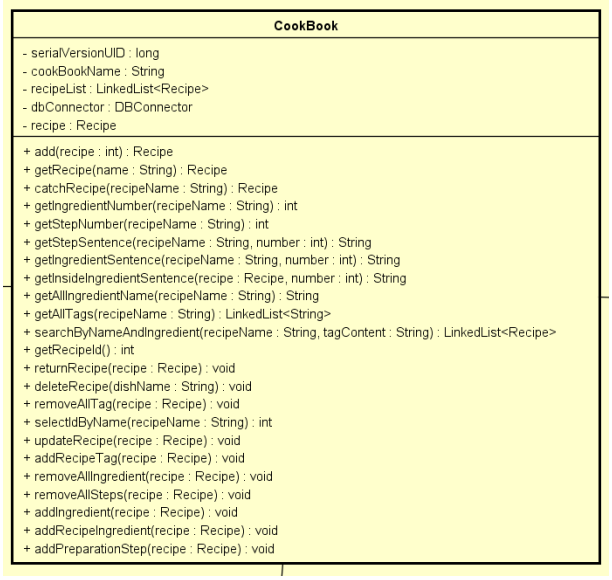
"CookBook" class which manages the "Recipe" class

There's a "recipeList" in this class which contains a list of recipes. A private attribute "dbConnector" is configured to exchange information with our database.

“DBConnector” class:

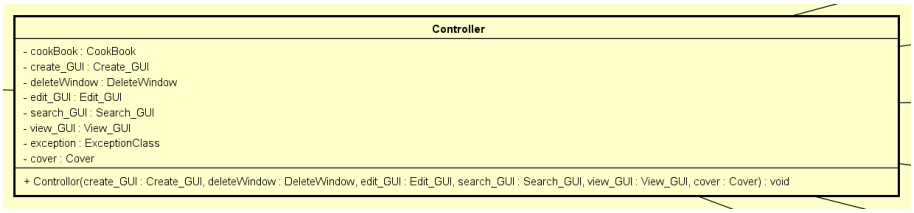


In the “DBConnector” class, we have attributes acting as containers. In this class, all the methods communicate with MySQL server and our database, in which connections are built with MySQL and responsible for conveying specific SQL sentence to get desired results.



By initializing the "DBConnector" object in the "CookBook" class, all the methods dealing with data processing in this class are set to call the methods from the “DBConnector” class. In order to serve the representing data on the GUI, the methods in the “DBConnector” class has been improved and adjusted to fit the requirements of the “Controller” class.

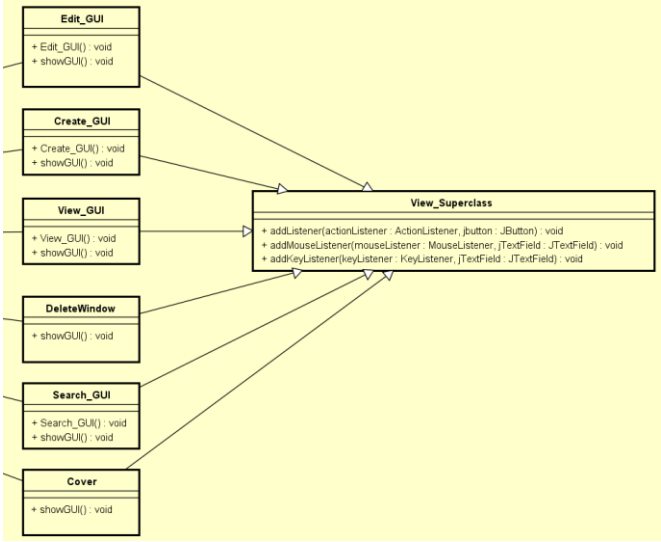
Controller



The class “Controller” shares the attributes from the “CookBook” class and all the GUI classes. In this class, all the GUI objects are initialized and all the listeners are equipped for them. By this way, we can manage data performance of GUI and since we have the attribute of the “CookBook” class, all the needed data can be retrieved by calling these methods from the “CookBook” class.

The “Controller” class does not have access to the “DBConnector” class, but it can use the methods from “CookBook” in every listener in GUI classes. Therefore, our GUIs can provide dynamic information to users.

View



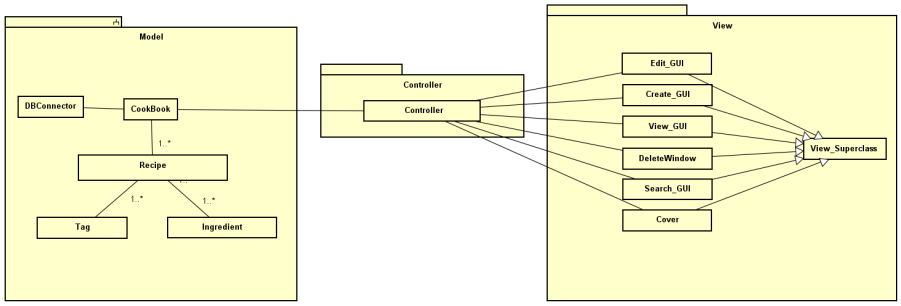
In the “View” part, all the GUI classes are configured and extended by the “View_Superclass” class in which methods for adding listeners are written.

Every GUI class initializes their static components in their constructors. In the “showGUI” methods, every class defines the size of the interface and visualize it to our users.

The listeners of GUI classes are established in the “Controller” class handling the exchange of dynamic information with our database.

3. GUI Design

3.1. Structure



MVC pattern is adopted to design our structure which we can divide work responsibility and maintenance more easily. It also keeps the data safe.

In the Model part, the “Recipe”, “Tag” and “Ingredient” classes are to describe and store our most important object--recipe. The “Recipe” class can have the access to its own tags or ingredients by “LinkedLists” which are the attributes in the “Recipe” class.

Therefore, these three classes are mostly used for data storage and data retrieving. Next, our “DBConnector” class is responsible for exchanging data with MySQL server. Specific SQL sentences are executed to realize the data inserting, selecting, updating and deleting action.

In the class “CookBook”, specific methods are built to deal with data. Then our “Controller” class can get or send data by the “CookBook” class. The “DBConnector” class is a private attribute in the “CookBook” class which means that only the “CookBook” class can have the access to our database and our “Controller” cannot “see” anything in our database, which has improved the safety of our data in the database.

The Controller part only has one class “Controller”. The controller has all the attributes of GUI classes and we initialize GUI and their listeners in “Controller” class. In this class we can use an object of “CookBook” to call the methods and therefore we can exchange data. The “Controller” has no idea what the database is. It can only use these methods in “CookBook” class to execute operations related to data. Thus we can hide our database to Controller and View and protect our data.

The “Controller” class also holds all the listeners from GUI classes which operate with dynamic data flows. By using the “Controller” class we can have a clear idea of all the listeners and manage them easily.

Usability heuristic

1. Visibility of system status:

Our GUIs have different styles for each interface, so our user will know clearly which interface they are using and what operations they can activate. Immediate feedbacks will pop up when the user's operations are successful or rejected.

2. Match between system and real world:

In our application, plain words are demonstrated to instruct our user. After the "Cover" GUI, our user will see the "Search" GUI which contains an entrance to "Create" GUI. After clicking one of the result, they can see and edit the recipe. These sequences make the best efforts to show the right information and provide convenience for our user.

3. Error prevention:

In the application, if the user inputs incorrect words or letters the app will pop up a warning notice and reject the operation of "Save". When the user wants to delete a recipe, a warning notice will be shown in order to prevent accidental operation. Thus, errors could be significantly reduced.

4. Recognition rather than recall:

In our GUI, different colours are utilized to show our main functions so that our users can see the place and the function efficiently. Four different colours are intended to differentiate the taste of tags in every recipe. For example, we use "red" to show the taste "spicy" because "red" usually makes the user think of chilli.

5. Flexibility and efficiency:

In our main GUI "Search", we allow the user to input words of a recipe or ingredient name and our search engine will automatically provide the best results for users. What's more, users can choose their favorite recipes with specific tastes with the help of four different taste tags, which could improve the flexibility and the efficiency of our search function.

6. Aesthetic and minimalist design:

Our program has reduced as much irrelevant information as possible and the rest is the essential information relevant to our recipes. A simple “back or forward” structure enables the users to know exactly which interface they are dealing with.

7. Help users recognize, diagnose, and recover from errors:

Error notice windows will pop up when the user has made an illegal operation. For example, if the user inputs letters in a quantity box, they will be advised to correct it and input only numbers before clicking the “Save” button.

8. Help and documentation:

In the report, we have included the instructions and a ReadMe file so that users will know how to use our software after reading these materials. We have also added some specific pictures to describe some important operations.

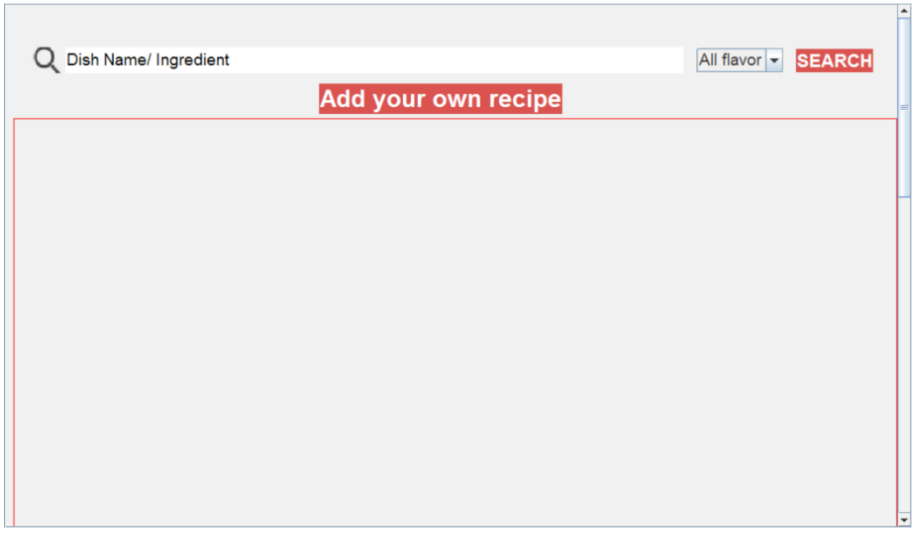
3.2. Screenshots

There are totally five main interfaces that will interact with the user, which are the cover interface, the searching interface, the viewing interface, the recipe-creating interface and the recipe-editing interface. These five major interfaces are described in detail as below.

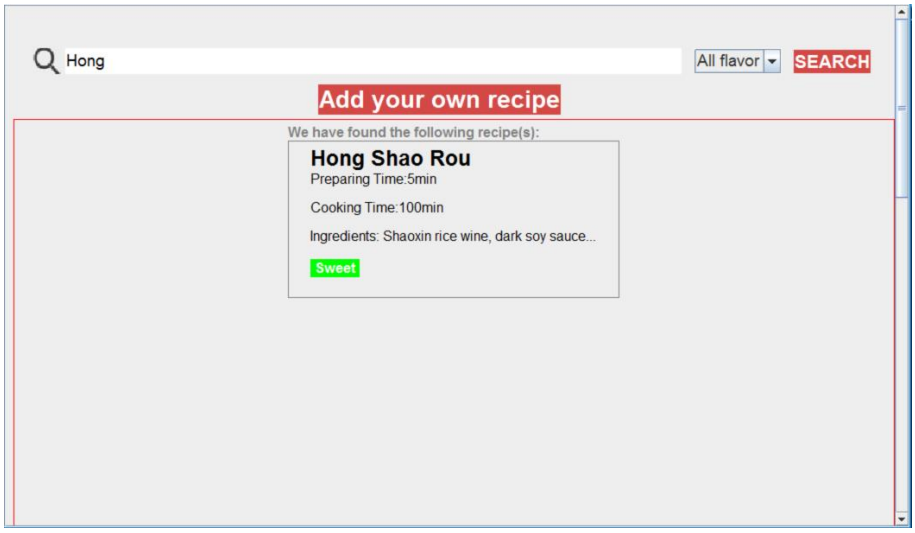
1. The cover interface:

This interface is the interface that will appear at first when the user starts our digital cookbook. In order to leave a good impression at the first glimpse, an attractive welcome interface must play an important role. We searched our album and finally selected this photo as our background picture after being beautified in the Photoshop. The reason why we chose red as our entering button is that the red colour is not only easy to be recognized but also can arouse the user's interest to click.

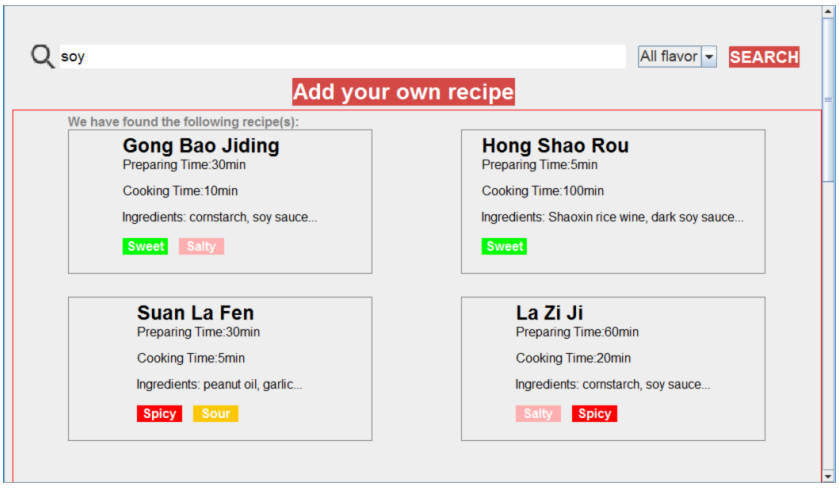
2. The searching interface:



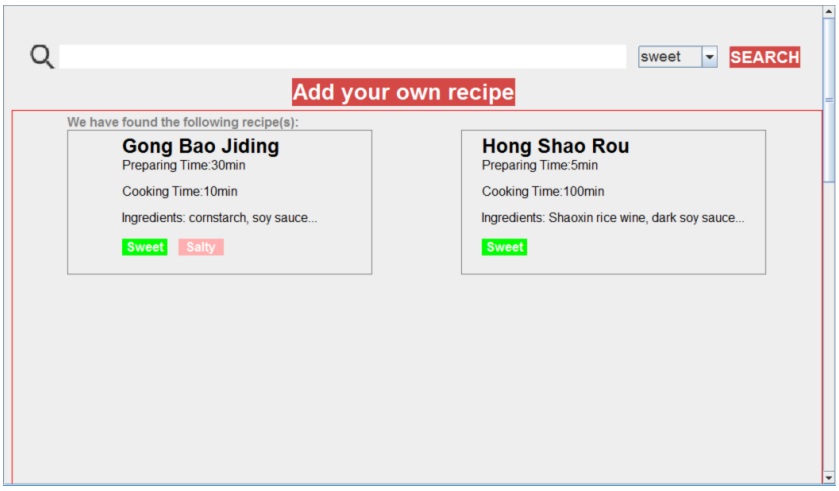
After clicking the button, the current interface will follow. This is one of the core interfaces of the whole software due to its highest using frequency. Here the user can search his/her target recipe by typing the recipe name into the text area and selecting the flavor he/she wants. We put an icon of a magnifying glass to make it more indicative to the user. The words "Dish Name/Ingredient" in the text area are filled as default to inform and direct the user to fill in the right key words. After clicking the "Search" button, the results will appear accordingly.



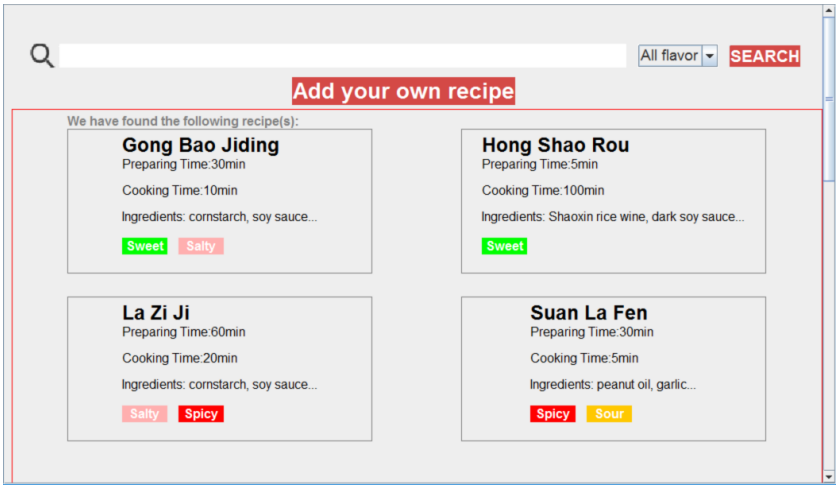
The result if the user searches by a key word from a recipe name



The results if the user searches by a certain ingredient name



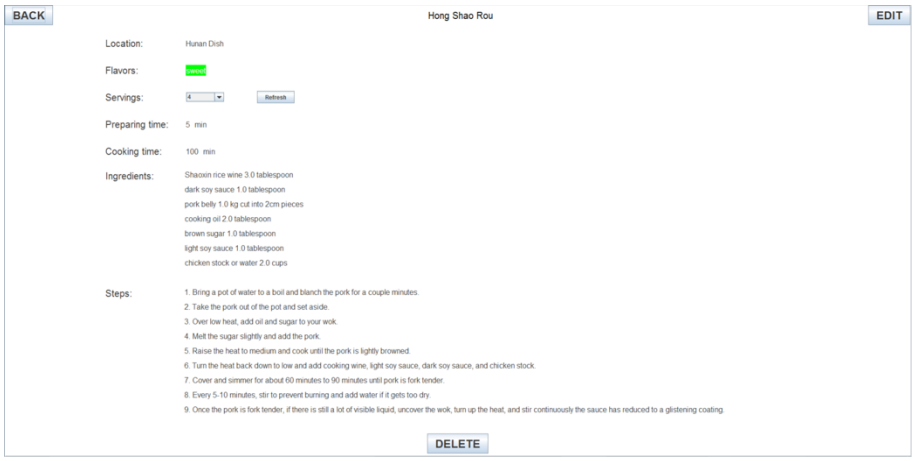
The results if the user searches by a certain flavor



If nothing is typed in and the flavor is “All flavor”, all recipes in this app will be shown.

If the user decides to see the detailed information of a dish, he/she only needs to click on the box of a certain recipe and the specific information interface will follow.

3. Viewing interface:



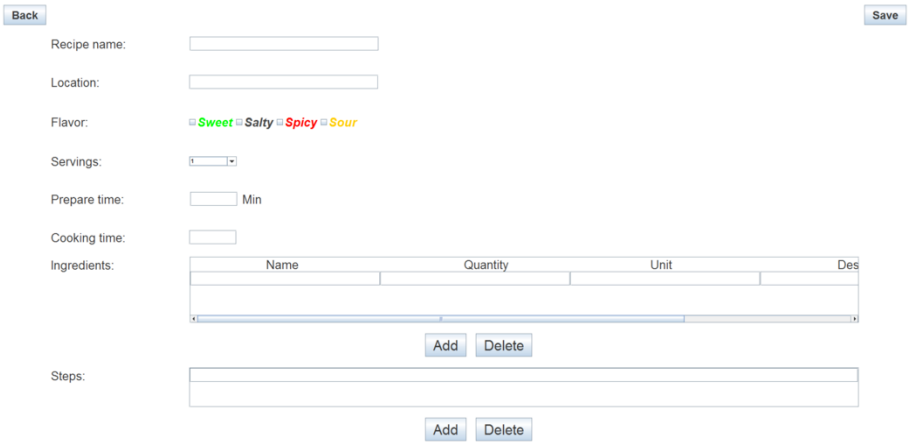
After clicking a certain result box, the viewing interface will be called to demonstrate all the information of the recipe. This interface is set to an even larger size in order to contain as much information as possible without the user rolling the scrollbar. The recipe name is placed at the top to be more apparent to the user. There are four flavours in our cook book, which are the additional task of our application. The four different flavour tags are set with four different colours. Green for sweet; Red for spicy; Grey for salty; Orange for sour. The reason for that is if the user becomes familiar with the colour, the user will know how it tastes without reading the word on it because people are more sensitive to colours than words.

The feature on this interface is that the user can change the number of serving people on this interface and after clicking the “Refresh” button, the amount of the ingredients will be changed instantly. If the user would not like to derive this recipe again, the user can click the “Delete” button to delete this recipe for good. If the user would like to customize this recipe, simply clicking the “Edit” button will lead him/her to the editing interface. If the user wants to go back to the previous page, the “Back” button can do the job.

4. The recipe-editing interface:

On this interface, the user is able to modify almost all the information he/she sees in the viewing interface. The "Min" labels after the preparing time and the cooking time text box suggest the user that he/she should type numbers in it. If more ingredients or steps need to be added, the "Add" button will add a row with blank text boxes for the user to fill in. After modifying the recipe, the "Save" button at the top right corner will be responsible for saving it into the database.

5. The recipe-creating interface:



If the user cannot find his/her recipe, he/she can click the red button "Add your own recipe" on the searching interface to create the user's personal recipe. Here all of the information can be customized by the user. The configuration of the editing, viewing and the creating interface has been modified to look as similar as possible to reduce the difficulty of usability.

4. Test

4.1. Boundary Test

Search window

1. **Search_GUI.Search Textfield:**

Value: String; English letters or blanks

Valid EC:

V1= { value | value= Hong }

V2= { value | value= “ ” }

Invalid EC: numbers, symbols and other language

V3= { value | value= 123 }

V4= { value | value= * }

V5= { value | value= ü }

Edit window

1. **Edit_GUI.LocationTextfield:**

   Value: String; length<= 45

   Valid EC:

   V1= {value | value= Xi’an }

   V2= {value | value= Lübeck }

   V3= { value | value= [A-Za-z]*45 }

   Invalid EC:

   V4= {value | value= null }

   V5= { value | value= [A-Za-z]*46 }

2. **Edit_GUI.PreparingTimeTextfield:**

   Value: Range of integer (0<= value<=2147483647)

Valid EC:

V1= {value | 0<= value<= 2147483647 }

Invalid EC:

V2= {value | value< 0 }

V3= {value | value> 2147483647 }

V4= {value | value= a }

Test cases for: 0; 2147483647; 2147483648; -1; a

3. Edit_GUI.CookingTimeTextfield:

Value: Range of integer (0<= value<=2147483647)

Valid EC:

V1= {value | 0<= value<= 2147483647 }

Invalid EC:

V2= {value | value< 0 }

V3= {value | value> 2147483647 }

Test cases for: 0; 2147483647; 2147483648; -1; a

4. Edit_GUI.Ingredient.NameTextfield:

Value: String; English letters or blanks; length<= 80

Valid EC:

V1= { value | value= Hong }

V2= { value | value= " " }

V3= { value | value= [A-Za-z]*80 }

Invalid EC:

V4= { value | value= 123 }

V5= { value | value= * }

V6= { value | value= ü }

V7= { value | value= [A-Za-z]*81 }

5. Edit_GUI.Ingredient.QuantityTextfield

Value: Range of integer (0<= value<= 2147483647)

Valid EC:

V1= {value | 0<= value<= 2147483647 }

Invalid EC:

V2= {value | value< 0 }

V3= {value | value> 2147483647 }

Test cases for: 0; 2147483647; 2147483648; -1; a

Test cases for: 0; 2147483647; 2147483648; -1; a

6. **Edit_GUI.Ingredient.UnitTextfield:**

Value: String; English letters or blanks; length<=45

Valid EC:

V1= { value | value= g }

V2= { value | value= " " }

V3= { value | value= [A-Za-z]*45 }

Invalid EC:

V4= { value | value= 123 }

V5= { value | value= * }

V6= { value | value= ü }

V7= { value | value= [A-Za-z]*46 }

7. **Edit_GUI.Ingredient.DescriptionTextfield:**

Value: String; length<= 80

Valid EC:

V1= { value | value= [A-Za-z]*80 }

V2= { value | value= * }

V3= { value | value= " " }

V4= { value | value= null }

Invalid EC:

V5= { value | value= [A-Za-z]*81 }

8. Edit_GUI.StepTextfield:

Value: String; length<= 255

Valid EC:

V1= { value | value= [A-Za-z]*255 }

V2= { value | value= * }

V3= { value | value= “ ” }

V4= { value | value= null }

Invalid EC:

V5= { value | value= [A-Za-z]*256 }

Create window

1. Create_GUI.DishName Textfield:

Value: String; English letters or blanks; length<= 80

Valid EC:

V1= { value | value= Hong }

V2= { value | value= “ ” }

V3= { value | value= [A-Za-z]*80 }

Invalid EC:

V4= { value | value= 123 }

V5= { value | value= * }

V6= { value | value= ü }

V7= { value | value= [A-Za-z]*81 }

2. Create_GUI.LocationTextfield:

Value: String; length<= 45

Valid EC:

V1= {value | value= Xi’an }

V2= {value | value= Lübeck }

V3= { value | value= [A-Za-z]*45 }

Invalid EC:

V4= {value | value= null }

V5= { value | value= [A-Za-z]*46 }

3. **Create_GUI.PreparingTimeTextfield:**

Value: Range of integer (0<= value<= 2147483647)

Valid EC:

V1= {value | 0<= value<= 2147483647 }

Invalid EC:

V2= {value | value< 0 }

V3= {value | value> 2147483647 }

V4= {value | value= a }

Test cases for: 0; 2147483647; 2147483648; -1; a

4. **Create_GUI.CookingTimeTextfield:**

Value: Range of integer (0<= value<=2147483647)

Valid EC:

V1= {value | 0<= value<= 2147483647 }

Invalid EC:

V2= {value | value< 0 }

V3= {value | value> 2147483647 }

Test cases for: 0; 2147483647; 2147483648; -1; a

5. **Create_GUI.Ingredient.NameTextfield:**

Value: String; English letters or blanks; length<= 80

Valid EC:

V1= { value | value= Hong }

V2= { value | value= " " }

V3= { value | value= [A-Za-z]*80 }

Invalid EC:

V4= { value | value= 123 }

V5= { value | value= * }

V6= { value | value= ü }

V7= { value | value= [A-Za-z]*81 }

6. **Create_GUI.Ingredient.QuantityTextfield**

Value: Range of integer (0<= value<=2147483647)

Valid EC:

V1= {value | 0<= value<= 2147483647 }

Invalid EC:

V2= {value | value< 0 }

V3= {value | value> 2147483647 }

Test cases for: 0; 2147483647; 2147483648; -1; a

7. **Create_GUI.Ingredient.UnitTextfield:**

Value: String; English letters or blanks; length<=45

Valid EC:

V1= { value | value= g }

V2= { value | value= “ ” }

V3= { value | value= [A-Za-z]*45 }

Invalid EC:

V4= { value | value= 123 }

V5= { value | value= * }

V6= { value | value= ü }

V7= { value | value= [A-Za-z]*46 }

8. **Create_GUI.Ingredient.DescriptionTextfield:**

Value: String; length<= 80

Valid EC:

V1= { value | value= [A-Za-z]*80 }

V2= { value | value= * }

V3= { value | value= “ ” }

V4= { value | value= null }

Invalid EC:

V5= { value | value= [A-Za-z]*81 }

9. Create_GUI.StepTextfield:

Value: String; length<= 255

Valid EC:

V1= { value | value= [A-Za-z]*255 }

V2= { value | value= * }

V3= { value | value= “ ” }

V4= { value | value= null }

Invalid EC:

V5= { value | value= [A-Za-z]*256 }

4.2. Usability Test

4.2.1. Description

User Analysis

This application contains a number of recipes as reference for users. Therefore, users are supposed to be those who are not so capable of cooking, or those who wonders a certain recipe. Hence, this app might be more suitable for young people.

Young people usually are not so good at cooking. Some of them even have no idea about it. As beginners, they will turn to recipes for help. Of course, there might also be users who are proficient in cooking. However, sometimes they cannot decide what to cook. Recipes in the application can be the inspiration for this group.

To conclude, in terms of age, young people are supposed to be in the majority of users of this application.

In the case of gender, both males and females have the demand of cooking. So, the testers will consist of both men and women.

Students in university mostly live away from their family. Cooking on their own could be their only way out. Many graduates who begin to work are also independent from their family. In addition, parents are also probable to use our software. In conclusion, the user group majorly consists of students, graduates and parents.

User Profile

User	Gender	Age	Job
user1	male	21	student
user2	male	21	student
user3	male	21	student
user4	female	21	student
user5	male	21	student
user6	male	23	graduate
user7	female	22	graduate
user8	male	23	graduate
user9	female	39	teacher
user10	male	53	housekeeper

Task

Step 1: You want to cook a dish named “Hong Shao Rou”.

Aim: To test whether user can search a recipe by dish name.

Step 2: And there will be 6 people having this dish.

Aim: To test the function of changing servings.

Step 3: One of them love boiled eggs together with pork.

Aim: To test the function of add new ingredients.

Step 4: You want to create a new recipe on your own.

Instruction:

1. Input “Pommes” into “Recipe name” textfield.
2. Input “Germany” into “Location” textfield.
3. Tick “Salty” checkbox. Click “1” in combobox.
4. Input “30” in “Prepare time” textfield.
5. Input “30” in “Cooking time” textfield.
6. Input “Potato” into “Name” textfield, “1.0” into “Quantity” textfield, “kg” into “Unit” textfield.
7. Then click “Save” button.
8. Input “Cut potatoes into pieces” into “Step” textfield.
9. Then click “Back” button. Search “Pommes” in “Dishname/ Ingredient” textfield.)

Aim: To test the function of creating new recipe.

Step 5: 3 days later you are bored of this dish and you want to delete this recipe.

Aim: To test the function of deleting recipe.

Step 6: You love spicy food very much. Today you want to cook a dish with chicken.

Aim: To test the function of searching a recipe by both ingredient and flavor.

Step 7: You find that the quantity of chilly in this dish is not enough.

Aim: To test the function of editing ingredients of a recipe.

Step 8: When you are cooking this dish, your 3-year-old son begin to edit the ingredients. He inputs “123” into “Ingredient name” textfield and try to save.

Aim: To test the function of inputting invalid values.

4.2.2. Results

Step 1:

1. Began to type directly without clicking the "Search" textfield (user4, user7);
2. Pressed "Enter" key on keyboard to search (user6, user10);
3. Input "hong shao rou", so the result did not appear. Did not click the preview window of "Hong Shao Rou" to see the steps and read directly (user2);
4. Input "hong shao rou", so the result did not appear. Then input "Hongshaorou", so the result still did not appear (user5);
5. Thought for a while then click the preview window of "Hong Shao Rou" (user3);

Step 2:

1. Chose "6" in combobox successfully, but did not click "Refresh" button (user4, user5);
2. Clicked "Refresh" button accidentally, and did not see the change of quantity, did not find "Edit" button (user5);

Step 3:

1. Did not find "Edit" button (user1, user2, user3);
2. When finished editing, did not click "Save" button (user1, user9);
3. Complained the words are too big to see it completely while writing a new ingredient (user6);
4. Mistook "Edit" to "Exit", so user 7 did not know how to edit the ingredient (user7);
5. Kept clicking "Add" button because that the new empty line did not appear without clicking scrollbar (user10);

Step 4:

1. Did not find that "Add your own recipe" is a button (user6, user8, user9);
2. Did not know that before creating a new recipe, one should click "Back" button to return Search window first; when inputting "Cooking time" and "Preparing Time", user1 wrote "10 min" in the textfield, which only accept number; when writing items into "Ingredient" table, user1 did not see "Unit" column, and input "2kg" into "Quantity" textfield (user1);
3. Input the unit "L" into "Description" textfield (user2);

4. Input numbers in “Ingredient Name” and “Unit” textfield (user5);

Step 5:

1. Did not know how to delete a recipe when in “Edit” window (user2);

Step 6:

1. When in “Edit” window of “La Zi Ji”, had difficulty in using scrollbar (user2, user10);
2. Input “Ji” instead of “chicken” (user5);
3. Did not find that “Flavor” combobox is used to choose flavor and input “chicken” (user7);
4. Did not find that “Flavor” combobox is used to choose flavor, and input “spicy chicken” (user8);

Step 7:

1. Tried to clicked “Add” button to save (user1, user2);
2. Complained the words are too big to see it completely while writing a new ingredient (user3);

Step 8:

1. User input “123” into “Ingredient Name”, and the warning of invalid value appeared successfully, but still saved successfully (all).

4.2.3. Modification after the test:

Problem	Description	Solution
Step1.1	"Search" text field cannot be cleared by typing directly.	Modifying the class "Listener_Search_SearchMouse"
Step3.1	Users cannot see "Edit" button without using scrollbar.	Moving "Edit" button
Step3.3	Words are bigger than those in lines above.	Minifying the size of input
Step4.1	Users don't know that "Add your own recipe" is a button.	Enlarging the distance of between the button "Search" textfield; changing the color from blue to red
Step4.2	Users input "min" into "Preparing time" textfield.	Adding labels "min" behind textfield
Step7.2	Words are bigger than those in lines above.	Minifying the size of input
Step8.1	Invalid values can also be saved.	Restricting the input to only English letters and blanks

5. Evaluation

5.1. Group Work

During the time when we develop our digital cook book working as a team, every member in the group has tried their best to make our software even better. It is cheerful that everyone is willing to help each other. When someone has problems on his/her own task, the rest of the members are always ready to help.

Furthermore, everyone never hides his/her genuine opinions from others, which means we are all willing to share what we are thinking about to the whole team, although sometimes our ideas do not necessarily arrive at a consensus. The final decision will eventually be made after our heated discussion. Therefore, in general, we believe that we all have a good sense of team spirit.

However, since it is the first time that we did such a big project, it is inevitable that we have encountered quite a lot of problems.

The major problem often arises when we communicate with each other, which is the very key part during every group work. Sometimes we think that online discussing could save us time, but it is not always the case. The limited typing speed and the inconvenience in showing the codes with error are the two major barriers. We eventually found that sitting together and speaking face-to-face is always the most effective and efficient way of solving complicated problems, which can eliminate all the misunderstandings that would happen on the Internet.

The second problem is the code synchronization among the group, which has not been solved even until we finished our project. It got even worse when we finally integrated all the java classes together. To put it simply, if everyone is modifying the codes, but the progress on every computer must be synchronized. It has happened many times that we have to spend a lot of time in collecting all the updates into one final version. Otherwise, only one group member can modify the codes. If another member is to modify other part in the project simultaneously, he/she has to wait until the first one has finished, which has significantly reduced the team efficiency. We tried to conquer this by reporting every updating details and uploading only some segments. However, it didn't help too much. Initially we tried to use the website "GitHub", but due to the expense of learning how to use it, we abolished this method later. Maybe next time we will try to learn how to use it to boost our efficiency.

Up till now, we have learned a lot from this project. We become familiar with the process of developing a software and how to do a group work. In conclusion, we all appreciate to have the chance to develop our own software and we hope to learn more in the future!

5.2. Task Responsibilities

Bing Guanqi:

1. Code the classes “Recipe”, “Ingredient”, “Tag” in the model
2. Code the classes “ExceptionClass” and “InputException” (Self define exception) to catch the exception when user input the information we don't want them to input.
3. Design and code the classes “Cover”, and “Search_GUI” in the view.
4. Design and code the class “DeleteWindow” in the view with Shan Xue.
5. Code the relevant Listeners about “Cover” and “Search_GUI” in the controller.
6. Code the class “DBConnectorTest” in Junit_Test, to test the methods in the DBConnector.
7. Write the report of Product functions in the specification.
8. Conclude the task responsibilities of all the members and Write the paper of evaluation.

Zhang Xiaoyue:

1. Design the UML diagram, synchronizing with the developing process.
2. Code the class “DBConnector”, establish the connection between database and model, write SQL methods.
3. Write the method “search” in class “DBConnector”, used to search recipe from database
4. Establish and connect the “Controller” and view model.
5. Write some methods in class “CookBook” to connect with “DBConnector”.
6. Code the class “View_Superclass”.
7. Fix the BUGs after relevant testing.
8. write the parts of UML and MVC in the report.

Ma Qiang:

1. Code the controller about the class “Edit_View” and “Create_View” with Song Yuchao assisting
2. Code the method “Edit_Save” and “Create_Save” function in the controller.
3. In the class “DBConnector”, do the remove and delete Recipe, getMaxId, delete, part of update
4. Fix the BUGs after we did relevant testing.

5. Code the classes of “CookBook” in the model.
6. Write the “Read me” file.
7. Database dump.

Shan Xue:

1. Code the “View_GUI” class;
2. Drew the initial drafts of GUI of the whole application;
3. Wrote “showView” method in “Controller” class;
4. Helped the coding of the method “reviseServings”, to change the quantity of ingredients when changing the servings.
5. Modified “Listener_View_Refresh” class in “Controller” class;
6. Modified “rebuild_View_GUI” method in “Controller” class;
7. Conduct the usability test, including “designed the task of usability test of the application”, “Conducted usability test”, “Recorded users’ behaviours” and “Wrote the report of usability test”.
8. Conduct the boundary test of the application and Write the report of boundary test.

Song Yuchao:

1. Code the Serialization and deserialization
2. Code the method “reviseServings”, to change the quantity of ingredients when changing the servings in CookBook class
3. Design the final GUI Mockup and the Cover interface
4. Code the classes “Create_GUI” and “Edit_GUI” and the “add ingredient” and “add steps” listeners
5. Code all the Exit-Program-Confirmation Dialog windows in “showGUI”
6. Code Junit test for all the models and two self-define exception class
7. Write the “Description”, “Screenshots” and “Group evaluation” part in the report.
8. Write all the comments and Javadoc and code tidying
9. Grammatical correction, semantic optimization, overall integration and formatting of the report.