



EBU6304–Software Engineering Group Project

--The report of *Project Ramen*

Group: 54

Members:

Student Name	BUPT Student ID	QM Student ID
Tianyi Liu	2017212958	171049170
Xiaotian Wang	2017213053	171049550
Xiyue Liu	2017213058	171049240
Ruisi Han	2017212914	171048690
Shuming Fan	2017212970	171048586
Yang Li	2017213043	171049022

1. Introduction

This report shows the development process of the project. This software is a self-service ordering program running on the self-service ordering machine in the Totoro Ramen restaurant, including a friendly ordering system developed for restaurant customers, as well as an independent management system developed for managers. The name of this Project is: Project Ramen.

Using the system, users can easily order food, pay for it, or even sign up for a discount. And managers can modify pricing, view user orders and do a variety of management operations.

We use agile approaches to participate in the whole software development process. In the process of developing this system, our team worked together, applied the skills we learned in the software engineering class, and tried our best to meet the needs of users. Finally, we completed the development of this software. Next, this article will describe our development process by introducing project management, requirements, analysis & design, and implementation & testing.

1.1 Roles and Responsibilities

In agile development, everyone is involved in all aspects of the project. Therefore, in our team, everyone's role is uncertain, but each members' responsibility still has a focus. Tianyi Liu is the team leader, he is responsible for integrating projects, making decisions and organizing meetings to move the project forward. He also did a lot in the control and entity classes. Xiaotian Wang is responsible for the design and implementation of the management system modules. Ruisi Han is responsible for the structural design and implementation of the user interface. Xiyue Liu focuses on testing and bug fixing. Shuming Fan is mainly responsible for writing the user story and developing a portion of the control classes. Yang Li is our artist and she is responsible for all of our prototypes and the design of the user interface.

The above is just a brief introduction to the team members. Please refer to the weekly report for specific responsibilities.

2. Project Management

2.1 Scrum Approach with Leangoo

In the agile development process, we use Scrum to help us to work. We use Leangoo, which is a powerful tool for agile development, to manage the agile development process.

On the first Tuesday after each iteration, the Scrum master (Tianyi Liu) organizes a meeting. In the meeting, you will discuss the user story that needs to be implemented for this iteration as planned and break the user story into tasks. Depending on the area of expertise, each person will receive a certain number of tasks. Next, during the development process, if nothing unexpected happens, the developer just needs to focus on completing his own tasks. This way of working is very efficient in **Figure 2.1.1**.

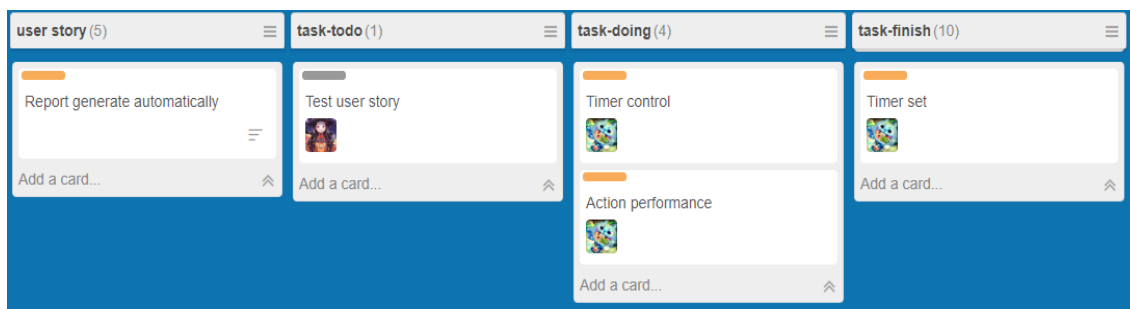


Figure 2.1.1 A line of tasks in Leangoo

For better agile development, we use Leangoo. As shown in the figure above, on this platform, each user story and its corresponding tasks will be visually displayed as a card. When a developer starts a task, he needs to move the task from "to" column to "doing" column. When he finishes the task, he needs to move the task to "finish" column. Therefore, the completion of each task can be intuitively monitored. In addition, we can get the user story details, the person in charge of the task, and the module to which the task belongs from the card. Leangoo also has some stats on its own, such as the ability to see how tasks are finished over time (burndown chart). As shown below, this is our burndown chart for the first iteration in **Figure 2.1.2**:

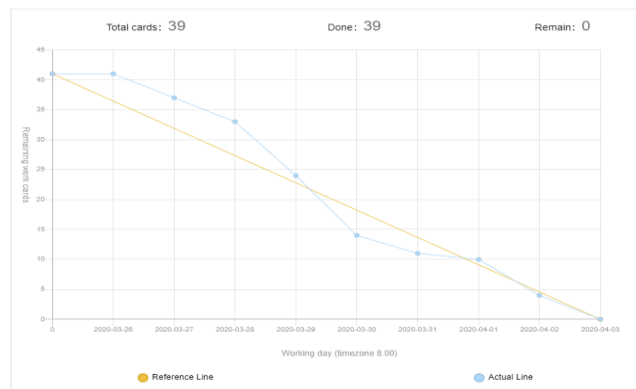


Figure 2.1.2 Burndown Chart of Iteration 1

The Scrum played a big role in our development process and greatly improved our development efficiency. The Scrum methodology played a big role in our development process and greatly improved our development efficiency. For each iteration, we have a Leangoo page, and you can find them in the appendix. All charts generated by Leangoo can be found in the QM hub Journals.

2.2 Code Version Control with Git

In many companies, it is very popular to use Git to do the code management. With Git, your team's code can be easily shared without fear of damage or loss. And the branch function of git gives us a lot of traversal. Git has also played a big role in the establishment and delivery of milestones. Over the course of the project, the team completed a total of 100 commits and six releases. In each iteration we create about six or seven branches, which are merged and deleted at the end of the iteration in **Figure 2.2**.

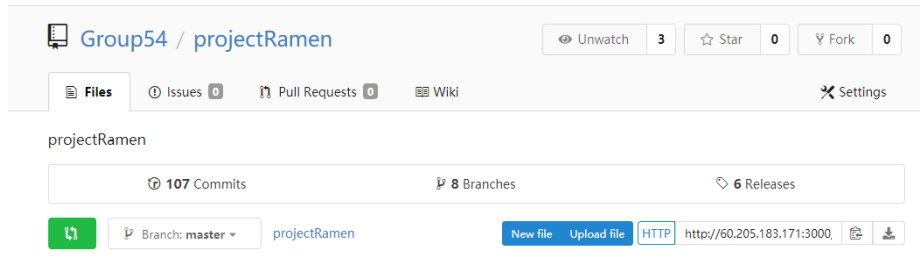


Figure 2.2 Our Git page

Our Git is built on our own server and can be safely accessed at any time. The URL is: <http://60.205.183.171:3000/Group54/projectRamen>

2.3 Time Estimate and Plans

We made a complete plan for the whole project before all tasks started. We have clear goals and assumptions for each iteration. Through time management, we successfully completed the whole task. Because the software is flexible, the project plan is constantly changing. Instead of using traditional Task chart and Activity network, our project uses Leangoo to demonstrate the project plan. The weekly plan will be decided at the regular meeting. Because we also don't know what kind of difficulties and problems we will encounter later in development. Therefore, before an iteration begins, we discuss the timeline in that iteration. Through Leangoo, we can easily see which tasks are not being done at each point in time, and which tasks are consuming a lot of time, so as to continuously improve the plan and schedule during the development process.

2.4 Decision Making

Because of our adoption of agile development, our team's decisions are very democratic. If there is a problem in the design pattern, it is not required to report it to the scrum master, and it can be implemented by the developer. If there is a design problem, it will be exposed in the later testing and implementation. If there is an issue about the new function, the scrum master should be notified and the function will be discussed at the weekly meeting. All in all, team members have enough autonomy, but there are still limitations.

2.5 Adapting to Change in Management

During our weekly meetings, we would summarize possible changes in requirements, and then the developers could discuss together to find the best way to design and implement. Because of our adoption of agile development, the team is very adaptable to change. If any changes are needed, they can be implemented only in the next iteration.

2.6 Risk Handling

In this project, we may encounter project risks and product risks. For the project risks, the approach we took was to give the team members enough time. If one person can't get things done on time, others have plenty of time to take over and get things done. Therefore, our schedule is slightly different from the recommended schedule. For the product risks, we take the approach of multiple iterations + multiple evaluations. At the end of each iteration, we would discuss the shortcomings of this iteration in the QQ group and evaluate our design and implementation. Once a problem is identified, it will be improved in the next iteration. If a serious problem is found during the iteration, we do not change the plan in the middle of the iteration, but choose to fix the problem in the next iteration. In fact, we did encounter a serious problem and solved it using this method. Please see our weekly journals for details.

3. Requirements

3.1 Requirements Finding Techniques

In the process of requirements identification, we took a variety of approaches and wrote the analysis report(on QM hub).

Background Reading and Brainstorming: Because the handout outlines what restaurant need for the software, we started with a detailed analysis of the entire handout. Then, before the first iteration, we summarized many requirements. More than 80% of high-priority requirements come from brainstorming.

Interviewing: We interviewed a restaurant manager and a classmate who had been to a ramen restaurant in Japan. From this we knew the real requirements of restaurants and customers.

Observation: We visited a number of restaurants and tried out their vending machines and found many positives and negatives. About 10% of requirements comes from this.

3.2 User stories

In terms of user stories, we produce a large number of user stories, delete a batch through meetings, and then evaluate and priorities the remaining parts. From the product backlog, you can see that our user stories numbers are discontinuous because many user stories have been removed due to project changes. For the compilation of user story, our principle is: innovation needs to be limited to the scope. At the beginning, the discussion atmosphere of the team was very lively, and we came up with a lot of interesting user stories. For example, there is a user story about a stamp exchange. Implementing the function of this user story means that users can give away tickets to each other. However, the user story was quickly rejected because the focus of the ordering machine was not focus on social interaction and sharing. Then we consulted the teaching assistant. He also supported deleting the user story. During the whole development process, we continuously generated new user stories under the guidance of the teaching assistant, which continuously improved the quality of the project.

For the estimation of user stories, we combined the Scrum and came up with an approach that worked for us. At each meeting, we looked together at the latest product backlog to find the user stories that were planned to be completed for this iteration. These user stories themselves have points that were previously estimated. Team members review these user stories and break each user story into multiple tasks, each with a similar amount of work. At this point, the Scrum master will arrange tasks based on the number of tasks. After completing the user story, the team members re-evaluated the story points based on their own experiences, and estimate the newly generated user stories. Over time, the estimation of user stories will become more and more accurate.

We have our own understanding of the priority of user story. It is not necessary to complete the user story according to the priority. In each iteration, we first select the user stories with high priority, and then we select some user stories with small according to the specific situation. This will make development more efficient. The way to prioritize is shown in the figure below in **Figure 3.2**:

Conditions	priority
Appears in the handout, and affects the normal use of the machine	Very High
Appears in the handout, and not affects the normal use of the machine	High or Very High
Not appears in the handout, and affects the normal use of the machine.	High
Appears in the handout, and is very important in requirements surveys.	High or Medium
Not appears in the handout, and is very important in requirements surveys	Medium
Not appears in the handout, and is not important in a requirements survey	Medium or Low
Not appears in the handout, and not affects the normal use of the machine.	Low or Very Low
Fancy little features	Very Low

Figure 3.2 The way to prioritize

3.3 Prototypes

In our group, there is a student with professional art knowledge: Yang Li. As an artist and designer on our team, she did almost all the prototyping. As development progressed, our requirements changed, so the prototype was modified or redone many times. In the whole project, we have made more than 15 prototypes, including high-fidelity and low-fidelity prototypes. These prototypes can be roughly divided into three categories in **Figure 3.3**:

Prototype Vesion	Characteristic
v1.x	Low precision main system prototype
v2.x	High precision main system prototype
Manage	A prototype designed to beautify management system

Figure 3.3 Prototype characteristic

Prototypes of the main system were built using the Adobe XD. This software can easily help designers design UI and UX. Therefore, these prototypes are interactive. The prototype of the management system was made using procreate. You can find these prototypes at the QM hub. Note that not all prototypes have been uploaded. We only uploaded the key prototypes.

With these prototypes, developers can feel comfortable knowing what the software should be designed to look like. Interactive prototypes allow developers to understand the logical structure of the user interface. Moreover, these prototypes contain art materials and color schemes. This makes our software more beautiful.

3.4 Iterations planning and time planning

We modified the recommended schedule to divide the entire development period into two preparation periods and five iterations. In the preparation periods, we did the brainstorm and wrote product backlog, glossary and prototype. During the iterations, we design and implement the software and improve what is written in the preparation periods. Each iteration has a corresponding theme. The theme for the first three iterations was determined in the preparation phase, and the themes for the last two iterations was determined in the third iteration. The theme of each iteration corresponds to the tasks of the iteration, which are formulated according to the theme before the iteration begins. In this way, we can not only have a complete plan, but also change our plan according to the actual situation. Details of the plan are as follows in **Figure 3.4**:

Time	Theme	Tasks	Outcomes
9-15 March	Preparation before starting the project	Group QQ group established QM hub group established Development environment configuration Learn to use tools like Git	QQ and QM hub groups Private Git platform
16-20 March	Indicate the direction of development	The weekly meeting Product requirements Identification Brainstorming User story writing salon	Product Backlog v1.0 Glossary v1.0 Prototype v1.0 Prototype v1.5 Requirements Identification report
21 March -3 April	Create a simple runnable version	The weekly meeting UI structure redesign Logical implementation of the system Management system build start up Modify the Product backlog Unit testing	A lot of milestones Project Ramen v0.1 Prototype v2.1 Prototype v2.2 Prototype v2.3 Product backlog v2.0
4-20 April	Create a version of the functionality that is basically available	The weekly meeting File operating system implementation Add member module Establish the management system UI Unit testing	Prototype-manage A lot of milestones Project Ramen v0.2 Project Ramen v0.21
21 April - 3 May	Management system embellished and debug	The weekly meeting Redesign the management system UI Fixed bugs in the connection between the main system and the management system Add new features Unit testing	A lot of milestones Project Ramen v0.3 Glossary v2.0
4-15 May	Creativity and innovation	The weekly meeting Fix a lot of bugs Add a lot of creative new features Unit testing	A lot of milestones Project Ramen v0.4 Product backlog v3.0
16-28 May	Complete development and prepare for release	The weekly meeting Fix a lot of bugs Document writing System testing Revision of the product backlog	A lot of milestones Project Ramen v1.0 Product backlog v4.0

Figure 3.4 Iteration planning

3.5 Adapting to Change in requirements

As mentioned above, our team has adopted a very flexible approach to change. In the process of the development, user story has been modified several times according to the tips of the teaching assistant and our reinvestigation. To be able to quickly fit to the new requirements, we categorize the requirements and assign different requirements to different modules. Each module is designed to be reusable. In modules, new requirements can be quickly added as the corresponding classes have been subdivided. In addition, in the numbering of user story, we also adopted the method of discontinuous numbering. Therefore, we are free to delete the old user story or add a new one. Classification and numbering can be viewed from the user backlog.

In addition, we specifically address the need for change in the of non-functional requirements. Over many iterations, our code has been refactored many times. The current design is generic, flexible and extendable to adapt to change.

4. Analysis and Design

4.1 Design of the main system

When designing the master system, our goal is to make the system as layered and modular as possible. In each layer, different modules are clustered together for use in the next layer. The advantage of this design is that the communication between the different layers is very convenient and the different modules in the same layer are decoupled from each other.

In the design, we use a pattern called "agent" to layer. In our early UML concept diagrams, we represented " agent " in an abstract way (Please see appendix). The agent is like a connector that connects classes between different layers. Due to the existence of agent, the boundary class, control class and entity class are completely decoupled. To better illustrate the layering and modularity of the design, we drew a number of boxes in the UML diagram to help understand. Please see the attachment for the original UML diagram. The following figure is a UML diagram of the main system(Next page) in **Figure 4.1**.

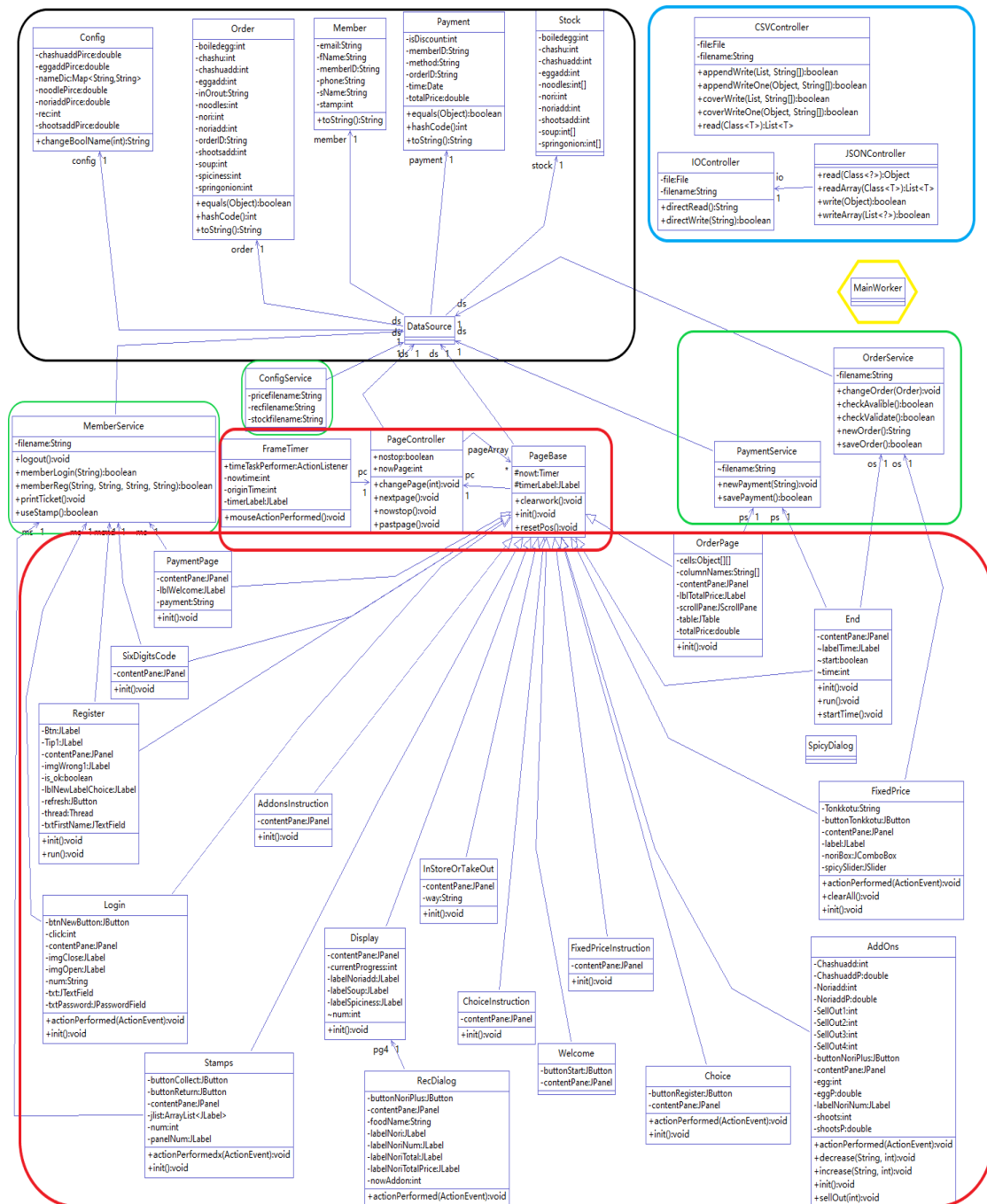


Figure 4.1 UML diagram of main system

In the figure above, the entity classes are in the black box, control classes are in the green boxes, and the red boxes contain boundary classes. The utility classes are in the blue box, and the yellow hexagon is the main class used to launch the entire program.

All entity classes are JavaBean objects. Based on the features of JavaBeans and Java reflection mechanism, our team designed two storage utility classes, CSV and JSON. CSV storage class specializes in storing frequently changing content, such as order information. JSON specializes in storing content that doesn't change very often, such as product prices. To decouple, we made an

aggregate class for all entity classes: DataSource, which is the “agent” mentioned earlier. All boundary classes have permission to communicate with the agent, giving them flexible access to system data.

We take a "service control" approach in control classes. For all transactions that occur in the system, we can put them into one or more services and invoke the corresponding service class to handle the transactions. The boundary class realizes various functions through the combination of services. For example, an order can be created through the OrderService, and a payment can be made through PaymentService. Due to the small number of services in this project, we did not add agent to the control class. However, we have reserved the relevant code so that developers can easily add the agent as the number of services increases.

For boundary classes, we use a “message-action” mechanism. Except for simple dialogs, all boundary classes are inherited from PageBase. We call these classes that inherit from PageBase “Page”. PageBase specifies the various underlying properties of the Page, including the location of the window, the DataSource, and a timer. Decoupling between all pages is achieved through the PageController class.

For each Page, PageController creates and "registers" it when the program starts. But at this point the page is not fully initialized. PageController calls its corresponding method to initialize the page only before the page is displayed. When a page needs to jump to another page, the current page will send a message to the PageController. This message contains only the page number or page relationship, not the object or class for the new page. PageController will take some actions, such as closing the old page or initializes and displays the new page according to the message. This design provides great flexibility, allowing developers to easily add or remove pages. And the aggregated DataSource avoids direct data exchange between different pages. Therefore, on the boundary class, we have achieved complete decoupling.

In terms of reusability, our design takes into account as many reusable components as possible. For the storage of information, such as user information and order information, files are read and written through the CSVController class and the JSONController class. These two classes are so versatile that almost all services can read and write files through them. In addition, inheritance and aggregation in boundary class design also improve reusability. The DataSource and PageController are used repeatedly.

4.2 Design of the management system

For the management system, we used a similar design to the main system, but more lightweight. We believe that the management system should be completely separate from the primary system, so the management system has its own independent master class. The exchange of information between the management system and the main system is carried out through files. This means you can use another computer to log in to the automatic ordering machine while the main system is running. This design also improves security. The following UML diagram shows

the design of the management system. As with the main system, we annotated the UML diagram with boxes in different colors. The original UML can be found in the appendix in **Figure 4.2**.

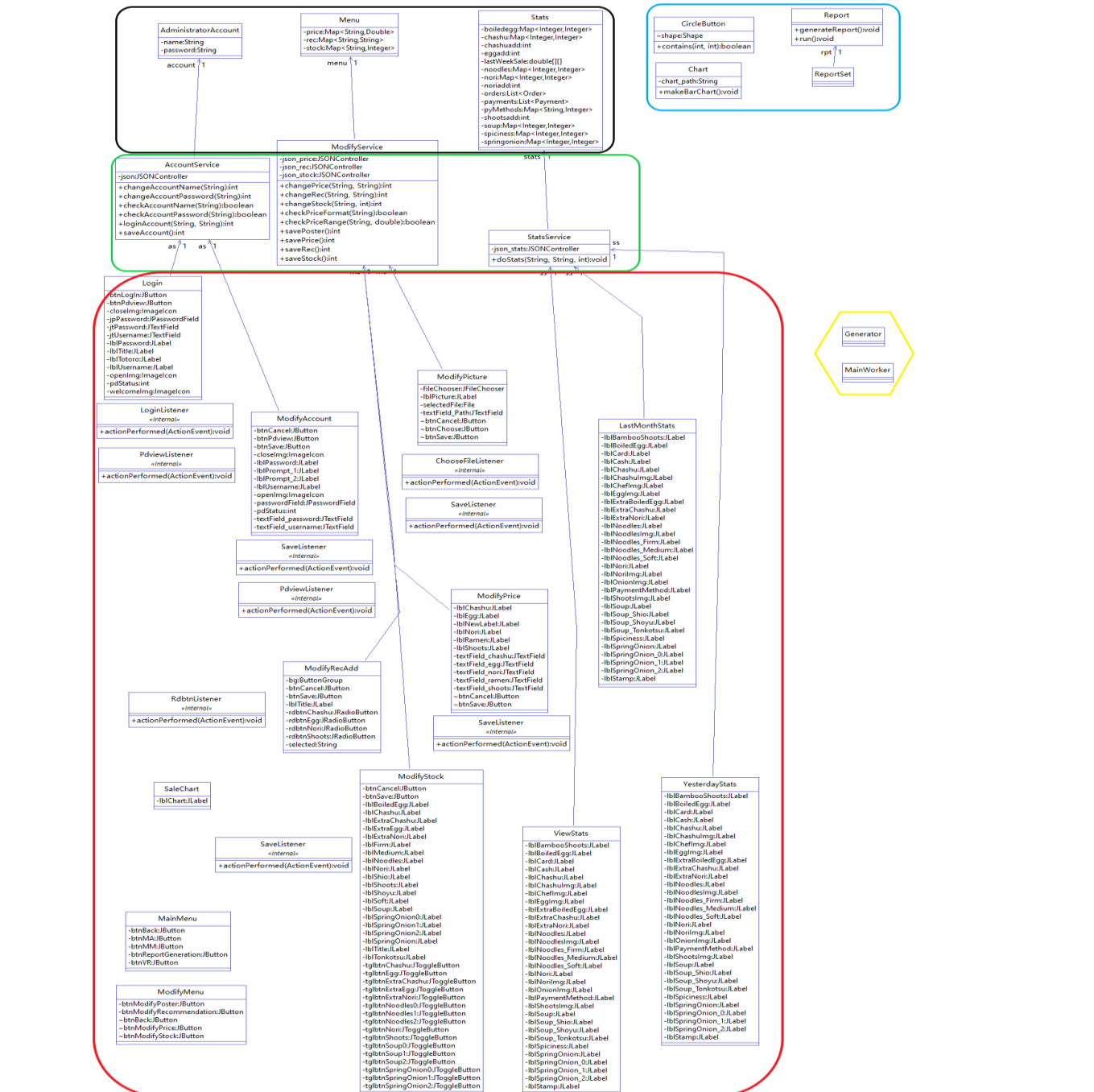


Figure 4.2 UML diagram of management system

In the figure above, like the main system, utility classes are in the blue box, the entity classes are in the black box, control classes are in the green boxes, and the red boxes contain boundary classes. In the yellow hexagon are the two main classes of the management system. MainWorker is used to start the management system, and the Generator is used to generate the report.

A closer look reveals that there is no agent and PageController in the management system. This is because the exchange of information in the management system is simpler. Almost all information exchange is a system and file operation. In order to achieve rapid development and iteration, the

management system is designed to be as light as possible, so these designs such like agent are also eliminated.

As a subsystem of project ramen, the design of the management system pays attention to reusability and extensibility. The management system and the main system share the same IO module. Therefore, the management system can read the information directly from the JSON file and CSV file of the main system. This greatly improves reusability. In addition, the statistics module is reused by multiple interfaces and functions. Lightweight design also increases reusability.

4.3 Design principles

Single Responsibility Principle(SRP): In order to comply with SRP, we tried to distinguish classes by modules and services from the very beginning of the design. In particular, in the control class, each control class corresponds to only one service and only modifies the entity class associated with the service. Design decoupling also helps us conform to SRP.

Dependency-Inversion Principle(DIP): This principle requires that high-level modules are less dependent on low-level modules. Our design is very much in line with this principle. Based on the “agent”, there are almost no dependencies between classes at different levels.

For other design principles, such as Don't Repeat Yourself(DRY), Open-Closed Principle(OCP) and Liskov Substitution Principle(LSP), we are working hard to implement. The vast majority of classes conform to these principles, but a small number of classes still do not.

5. Implementation and Testing

5.1 Implementation strategy

In our group's opinion, the implementation should be flexible. We will first have an overall implementation plan and then consider the detailed plan for the week before each iteration. Because the team adopted scrum and used Leangoo and Git, our implementation process became very flexible. Early in each iteration, team members are encouraged to feel free to explore implementation methods and techniques and upload their code to git. The team will then test the initially completed code. At the end of the iteration, team members check each other's code for code review to improve code quality. Thanks to Leangoo, everyone can see the team's overall progress. And the branch function of Git make this implementation strategy possible.

5.2 Iteration/built plan

Because of Scrum development, our minimal implementation unit is the task. Team members are encouraged to synchronize their code to Git as a build after completing 2-3 tasks. Git can number builds automatically. Because of our flexible implementation strategy, we did not specify requirements for each build. Requirements are constantly changing and it doesn't make much sense to plan for each build. Therefore, before the whole project started, we only made a brief plan for the important milestones, and the plan was improved over time. As shown in the figure below in **Figure 5.2**.

Iterations	Prototype	Milestone	Content
1	v1.0	v0.1	Minimum executable program No file operation is required Not implement management system
2	v2.1	v0.21	Software that barely meets most of the requirements The management system does not use files Member systems can retain some bugs
3	v2.2	v0.3	A fully functional but unattractive program The appearance of the management system should be redone System bugs should be kept to a minimum
4	v2.3	v0.4	Beautiful and functional program Detail changes to the appearance of the main system Fixed bugs in the management system System Testing
5	v2.3	v1.0	Deliverable software Preparation of related documents System Testing

Figure 5.2 Iteration/built plan

All builds and milestones are recorded on our Git. In particular, the deliverable version has also been tagged.

5.3 Test strategy

During the testing process, we used a black box test and a white box test.

● Black box test:

For boundary and entity classes, we use the black-box test method. In addition, we use a large and extensive set of automated tests. Before we started coding, we compared many test frameworks and found out JUnit is the most suitable test framework for us to test the code since it has many advantages:

1. We can write a series of test methods and unit test all of the project's interfaces or methods.
2. After start up, automate the tests and judge the execution results without human intervention.
3. Just look at the end result to see if the method interface for the entire project is unobstructed.
4. Each unit test case is relatively independent, started by Junit, and invoked automatically. No additional call statements need to be added.
5. Add, remove, and mask test methods without affecting other test methods. Open source frameworks all support JUnit.

Although we decided to use JUnit, group members had different ideas about which version to use. Some of the members prefer to use JUnit4 and the other prefer JUnit5. After comparing the difference of JUnit4 and JUnit5, combining the code we need to test, we ended up using JUnit4 to test our project.

Also, we separated the test classes from the other classes and put them into the same package. Finally, we wrote the test report.

● White box test:

For all user interfaces, we used a white box test. For efficiency, white-box testing is done in conjunction with user manual writing. Since we need to cover all of the program's functions when writing a user's manual, white-box testing is appropriate in this process. We made sure that all statements were executed at least once. In order to improve efficiency, we did not write the report in the white box test, but wrote

the user manual. Because we think this white box test report is redundant for the user manual.

When we test the functions of the pages, we press the button one by one to make sure the customers can have a fluent operation. What's more, we also test the function of the management on the front end to find out whether the information can be shown on the computer normally. The csv file generation is also an essential part of our testing. Our group members need to check out each line of the list to guarantee that the data sent by the system was stored correctly and completely.

In addition, we conducted performance testing and reliability testing. However, due to the limitation of space, I won't go into details here.

5.4 The using of TDD

When we use TDD in our project, we discuss what we want function the class to have, and then we design a complete test classes first in order avoiding the imprecise testing. We require the class can input the similar data given in the test classes and can let test classes run correctly, as well as giving out the expect data. When the test classes go wrong, we will record the mistakes and redesign the function classes until the test classes can run correctly. We need to write enough test cases to make the test fail, and write enough code to make the test successful. This can make we code for a much more specific purpose.

Specifically, we used TDD in the Menu class of the management system. Menu class is the entity class responsible for modifying the product price, availability, and other attributes. It contains information about the product. At the beginning, the team designed a series of test data to test the saving function of the price. The tests were then run and all failed. Then, we designed the Menu class against the test results. We run the tests again and they all pass. Finally, we redesigned the test to make it more concise and clear.

6. Conclusion

In the software engineering class, we learned a lot of things and used them to create the successful software we have today. In this process, all the team members worked together to face the difficulties and complete the project. At the same time, teachers and teaching assistants have been helping us, we are very grateful. In the days to come, we will use what we have learned to put into production and life. We will use software engineering to create value.

7. Appendix

Appendix-1 Reference List

Leangoo:

Iteration1: <https://www.leangoo.com/kanban/board/go/3349874>

Iteration2: <https://www.leangoo.com/kanban/board/go/3386999>

Iteration3: <https://www.leangoo.com/kanban/board/go/3429839>

Iteration4: <https://www.leangoo.com/kanban/board/go/3460039>

Iteration5: <https://www.leangoo.com/kanban/board/go/3486365>

Git:

Group54/projectRamen: <http://60.205.183.171:3000/Group54/projectRamen>

Journals | EBU6304-2020 Software Engineering Group 54:

Week1- Product Backlog and Prototype

<https://hub.qmplus.qmul.ac.uk/artefact/blog/view/index.php?id=693060>

Week2- Preparation and iteration 1

<https://hub.qmplus.qmul.ac.uk/artefact/blog/view/index.php?id=696417>

Week3- Iteration 1 completed

<https://hub.qmplus.qmul.ac.uk/artefact/blog/view/index.php?id=700231>

Week4- Iteration 2 started

<https://hub.qmplus.qmul.ac.uk/artefact/blog/view/index.php?id=704267>

Week5- Iteration 2 finished

<https://hub.qmplus.qmul.ac.uk/artefact/blog/view/index.php?id=708094>

Week6-7- Iteration 3: Management embellished and debug

<https://hub.qmplus.qmul.ac.uk/artefact/blog/view/index.php?id=712767>

Week8-9- Iteration 4: Creativity and innovation

<https://hub.qmplus.qmul.ac.uk/artefact/blog/view/index.php?id=718808>

Week10-11: Iteration 5-Final delivery

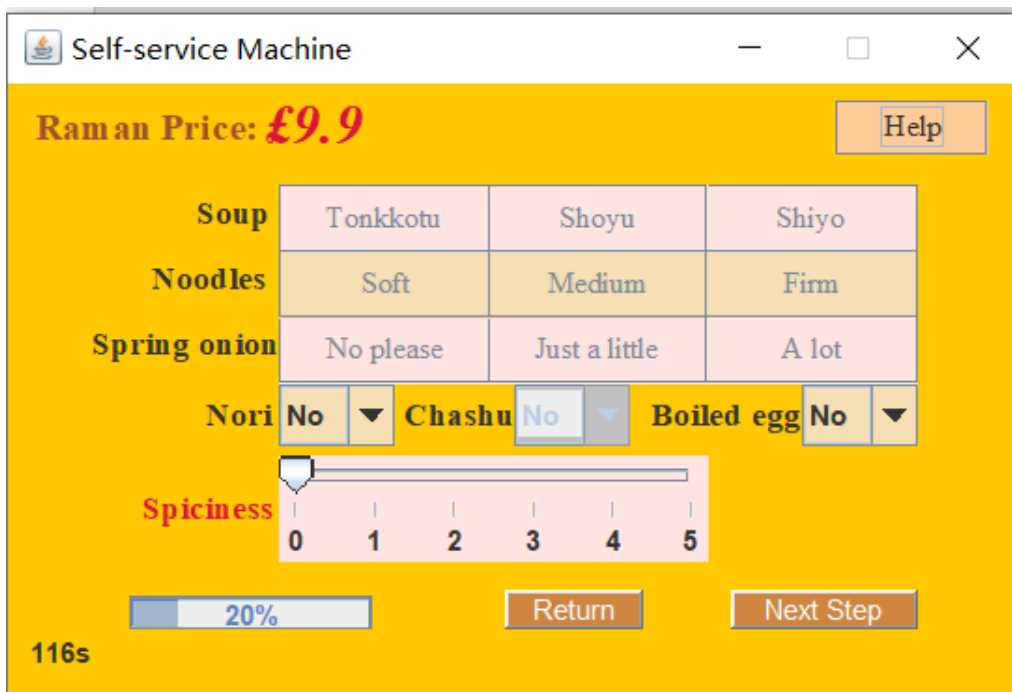
<https://hub.qmplus.qmul.ac.uk/artefact/blog/view/index.php?id=723281>

Appendix-2 Main Screenshots

Main system:



This is the welcome page. Customers click the “start” button to start ordering.



This page allows the customers to personalize the ramen by choosing the options.

Self-service Machine

Add-ones Help

	Extra Nori(£1.0)	£0.0	-	0	+
	Extra boiled egg(£1.0)	£0.0	-	0	+
	Bamboo shoots(£1.0)	£0.0	-	0	+
	Extra Chashu(£1.0)	£0	-	0	+

112s 40% Return Next Step

This page allows the customers to choose add-ons.

Order display

Your chooses:

Soup: Shoyu

Noodles: Medium More Nori: 0

Spring onion: Just a little More eggs: 0

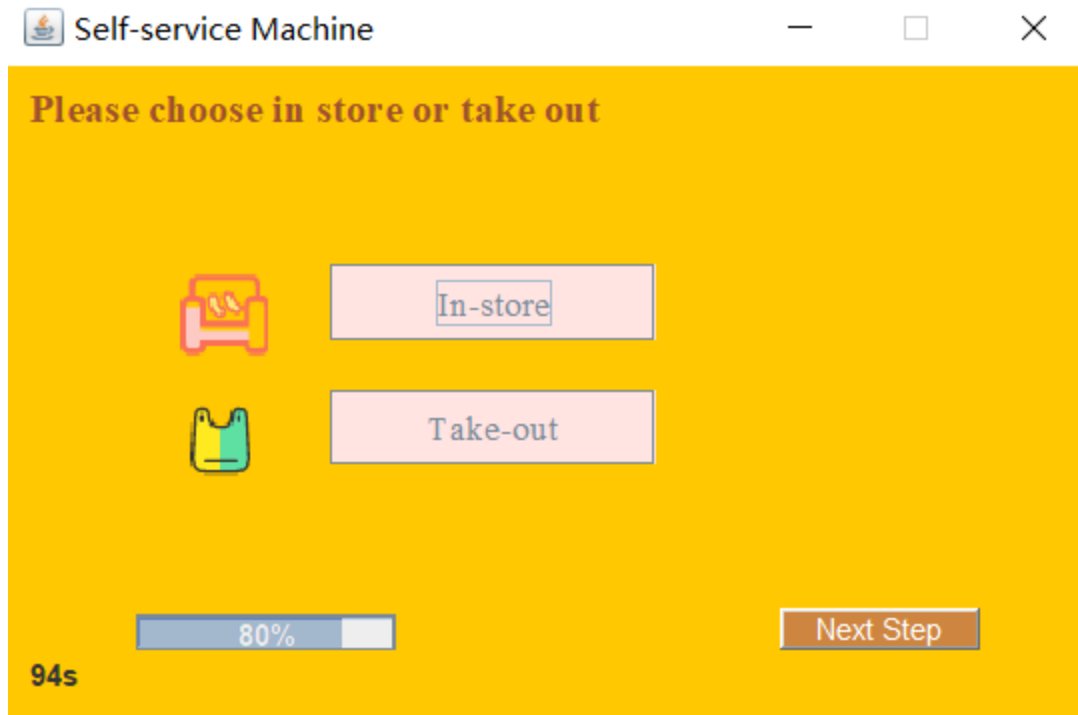
Nori: No More shoots: 1

Chashu: No More Chashu: 0

Boiled egg: No **Spiciness: 0**

118s 60% Return Next Step

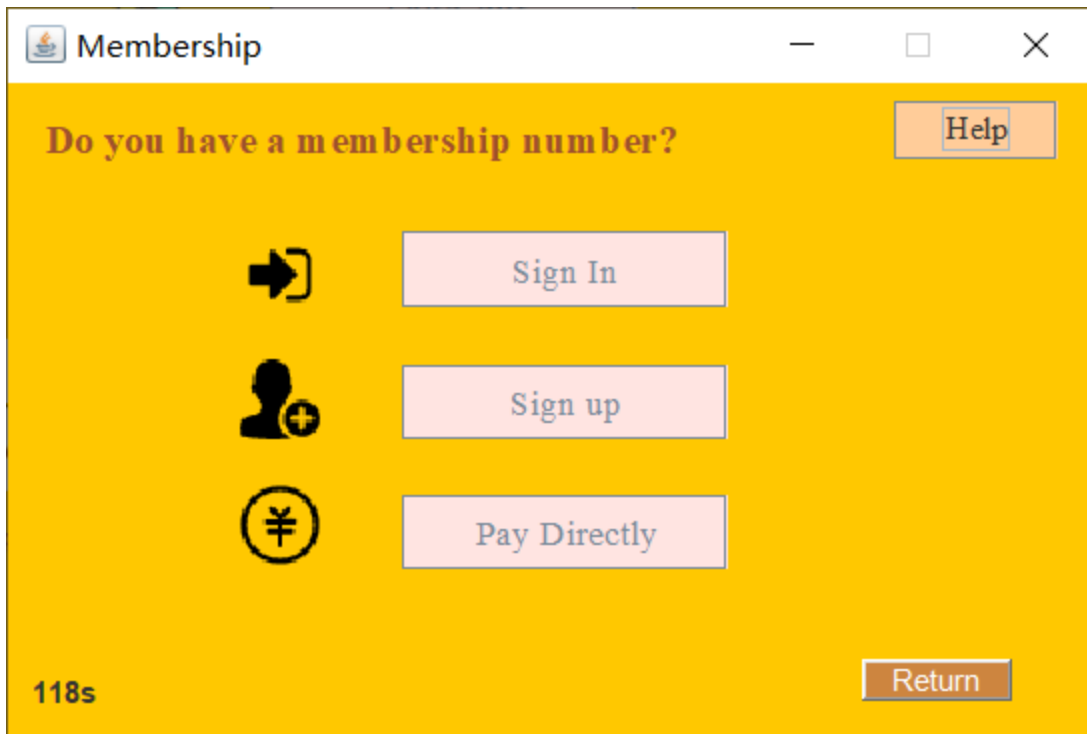
This page displays your order information.



Customers can choose in store or take out on this page.

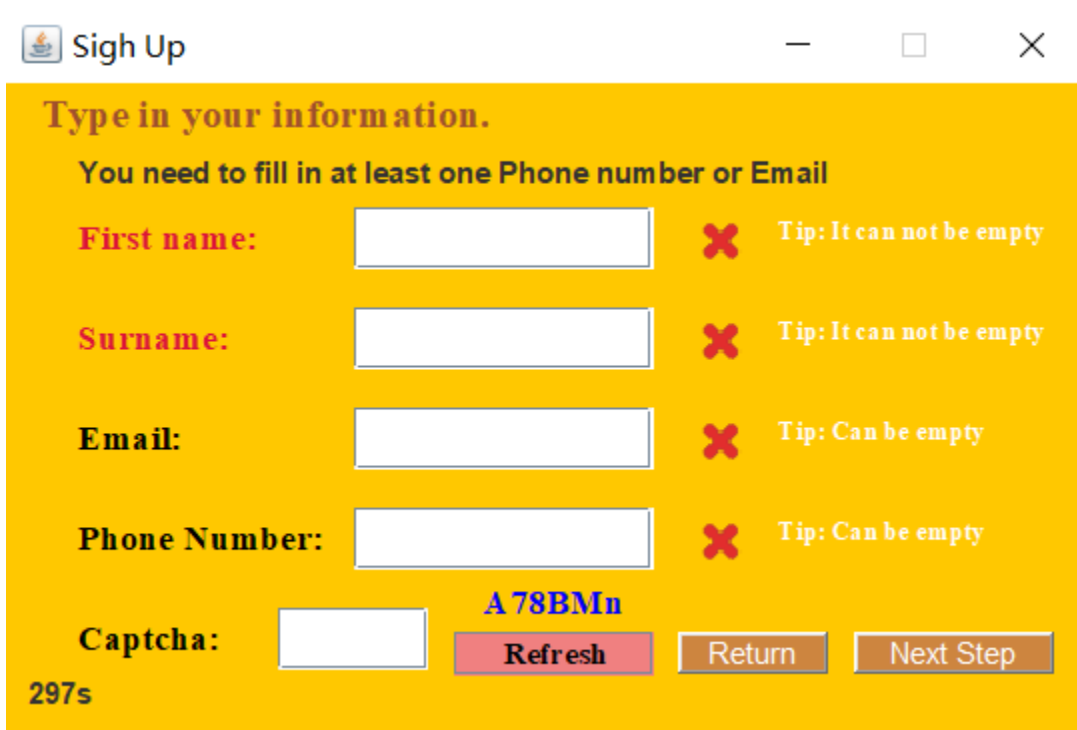


Order can be rechecked on this page.



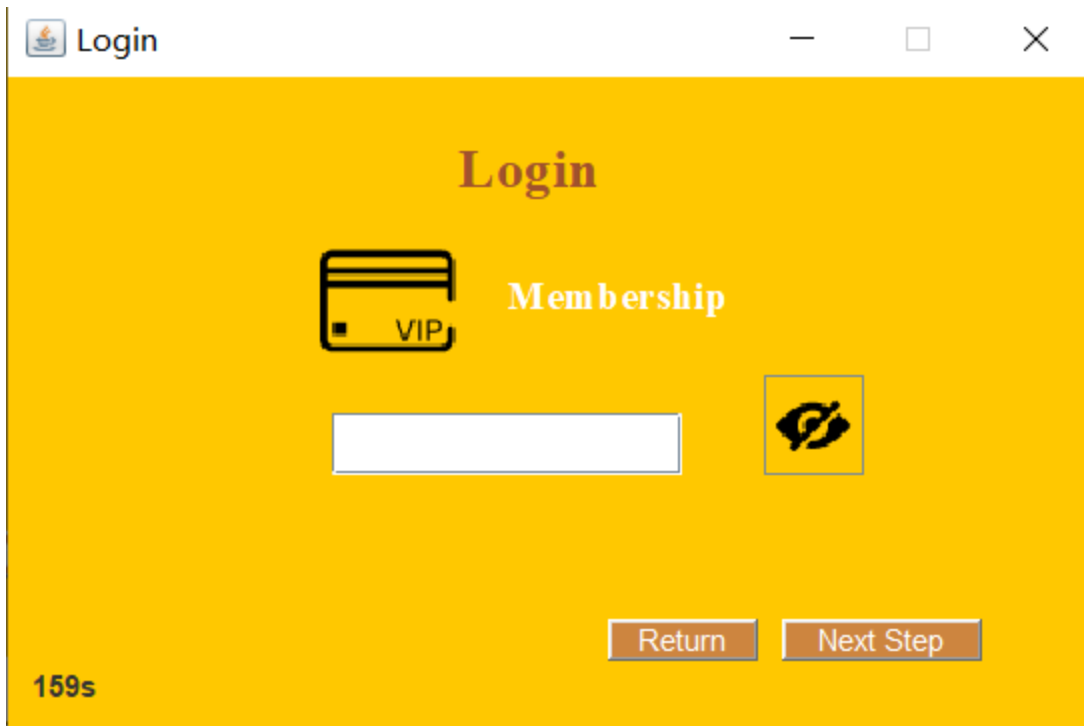
The screenshot shows a window titled "Membership" with a yellow background. At the top, it asks "Do you have a membership number?" with a "Help" button. Below this, there are three options, each with an icon and a button: a right-pointing arrow icon with a "Sign In" button, a person icon with a plus sign and a "Sign up" button, and a yen symbol icon with a "Pay Directly" button. At the bottom left, it says "118s", and at the bottom right, there is a "Return" button.

Customer can choose to sign in their member account, register as a member or pay directly.



The screenshot shows a window titled "Sign Up" with a yellow background. It asks the user to "Type in your information." and states "You need to fill in at least one Phone number or Email". There are four input fields: "First name:", "Surname:", "Email:", and "Phone Number:". Each field has a red "X" icon and a tip: "Tip: It can not be empty" for First name and Surname, and "Tip: Can be empty" for Email and Phone Number. Below the input fields, there is a "Captcha:" field with the text "A 78BMn" above it. To the right of the captcha field are three buttons: "Refresh", "Return", and "Next Step". At the bottom left, it says "297s".

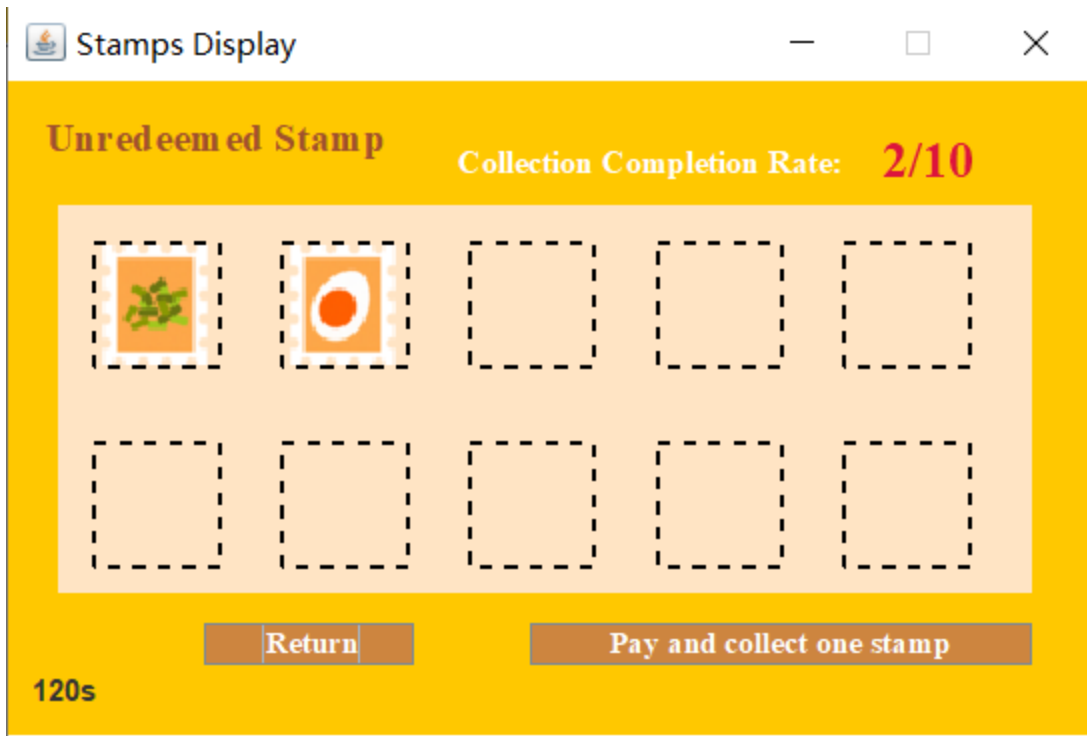
Customer can join the loyalty scheme by entering information name, email, phone number, etc.



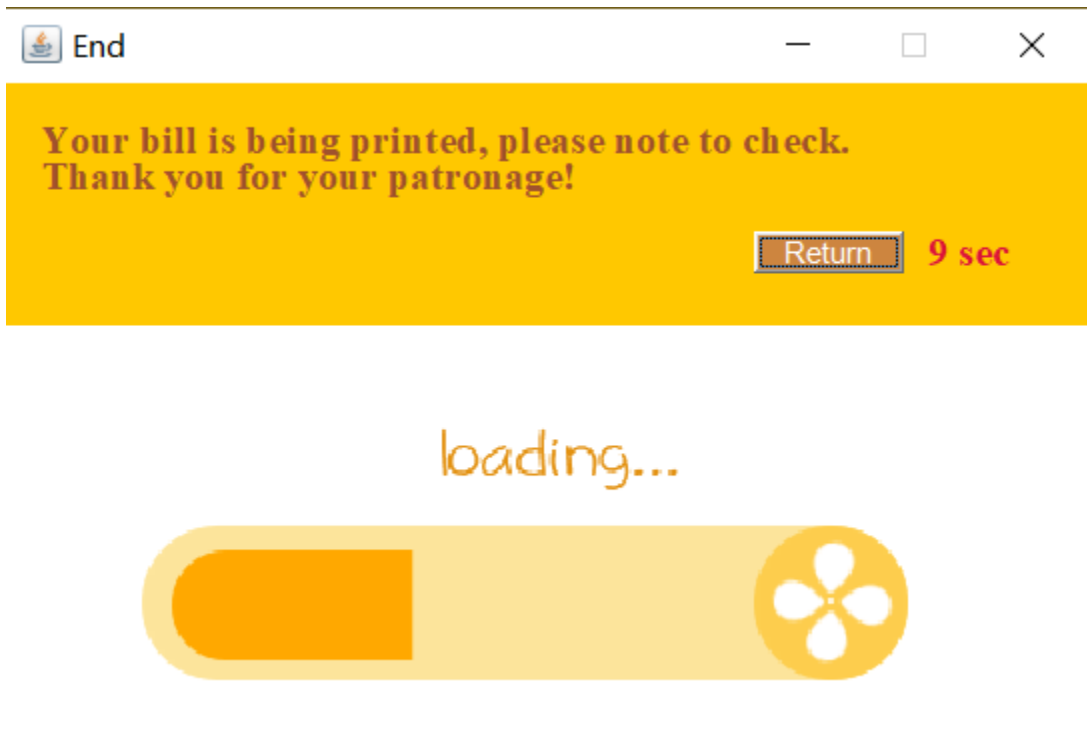
Customer can enter their membership number to log in.



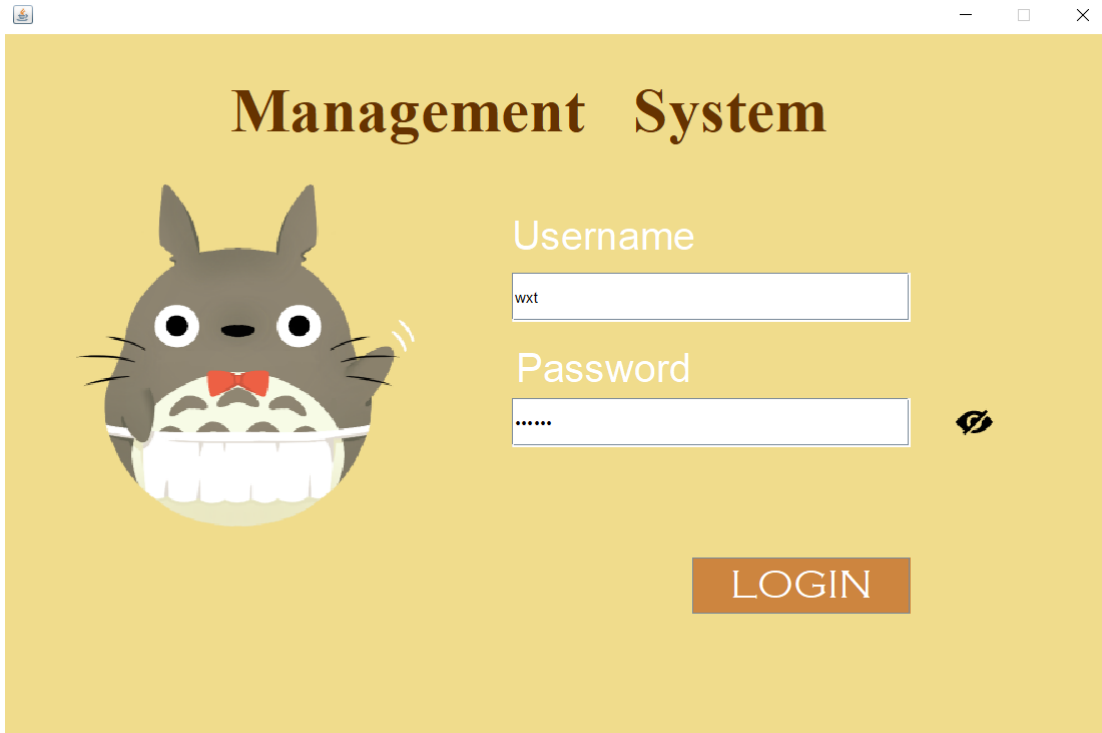
Customer can choose payment method on this page.



This page shows how many stamps has the customer collected. If the customer has collected 10 stamps, this meal will be free and the number of stamps will become 0.

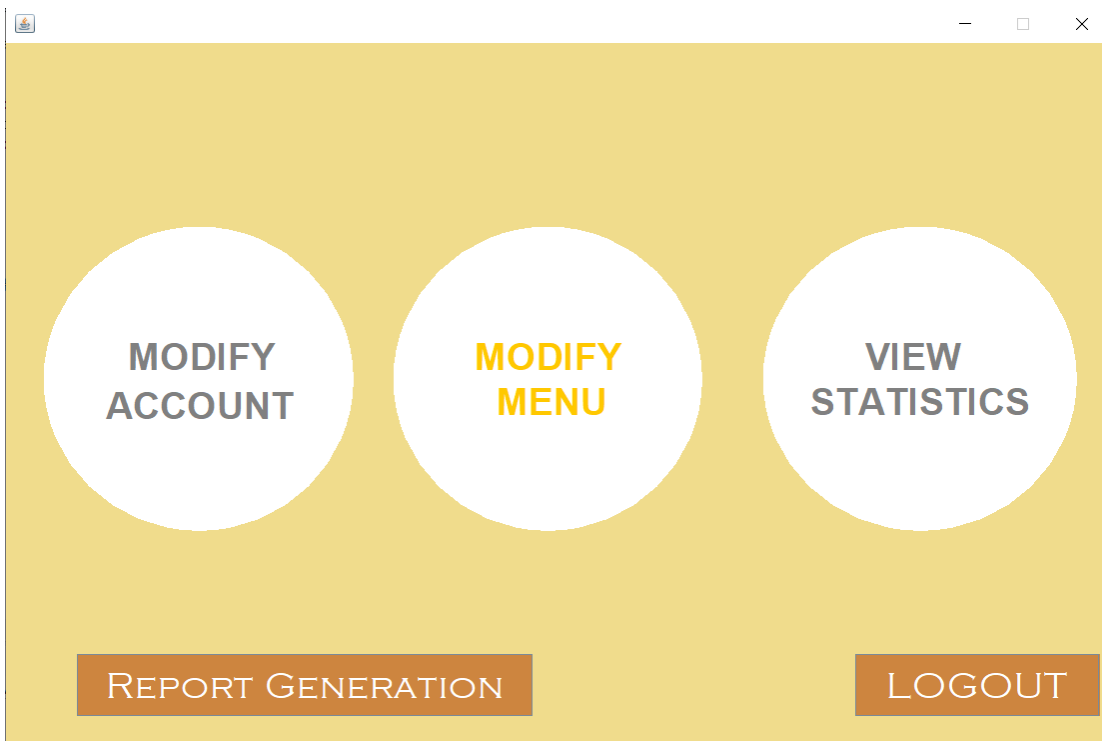


This page shows the customer has paid successfully and the ticket has been generated.

Management system:

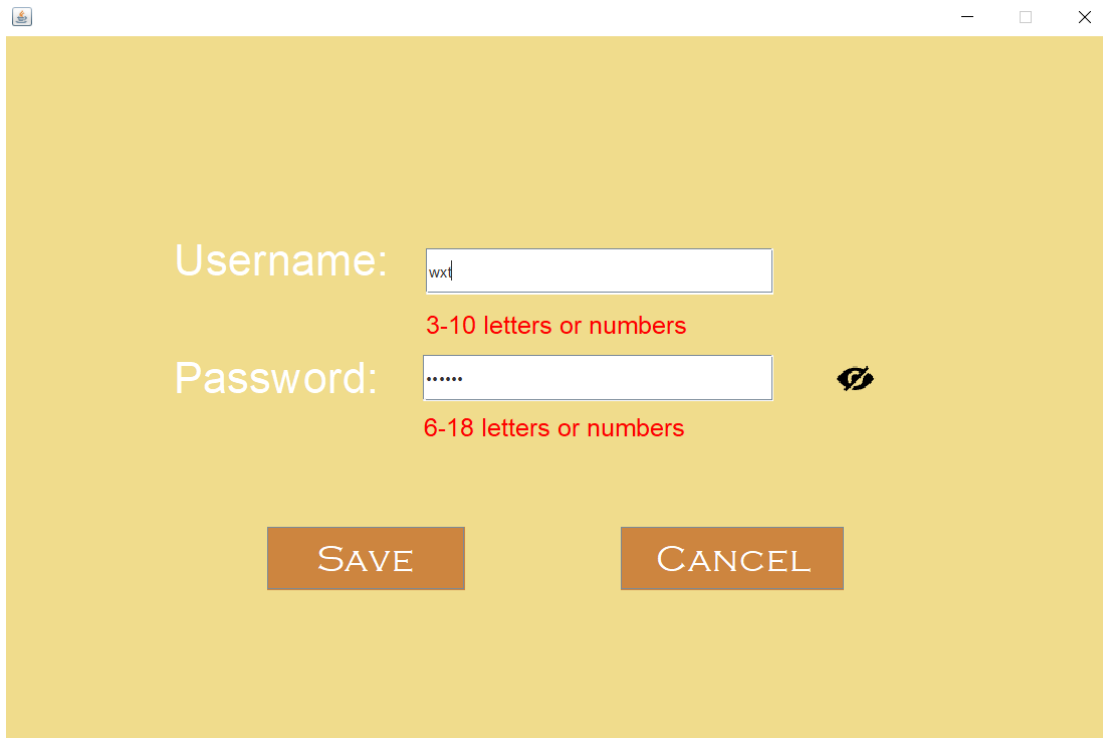
A screenshot of a web application window titled "Management System". The window has a yellow background. On the left side, there is a cartoon illustration of Totoro's face. To the right of the illustration, there are two input fields: "Username" with the text "wxt" and "Password" with masked characters ".....". A small eye icon is next to the password field. Below the input fields is a brown button labeled "LOGIN".

Restaurant manager should enter username and password of the administrator account to log in.



A screenshot of a web application window showing the main menu. The window has a yellow background. There are three large white circles arranged horizontally. The first circle contains the text "MODIFY ACCOUNT". The second circle contains the text "MODIFY MENU" in yellow. The third circle contains the text "VIEW STATISTICS". At the bottom of the window, there are two brown buttons: "REPORT GENERATION" on the left and "LOGOUT" on the right.

This is the main menu page. Restaurant manager can choose to modify account, modify menu, view statistics and change report generation config.



Username:

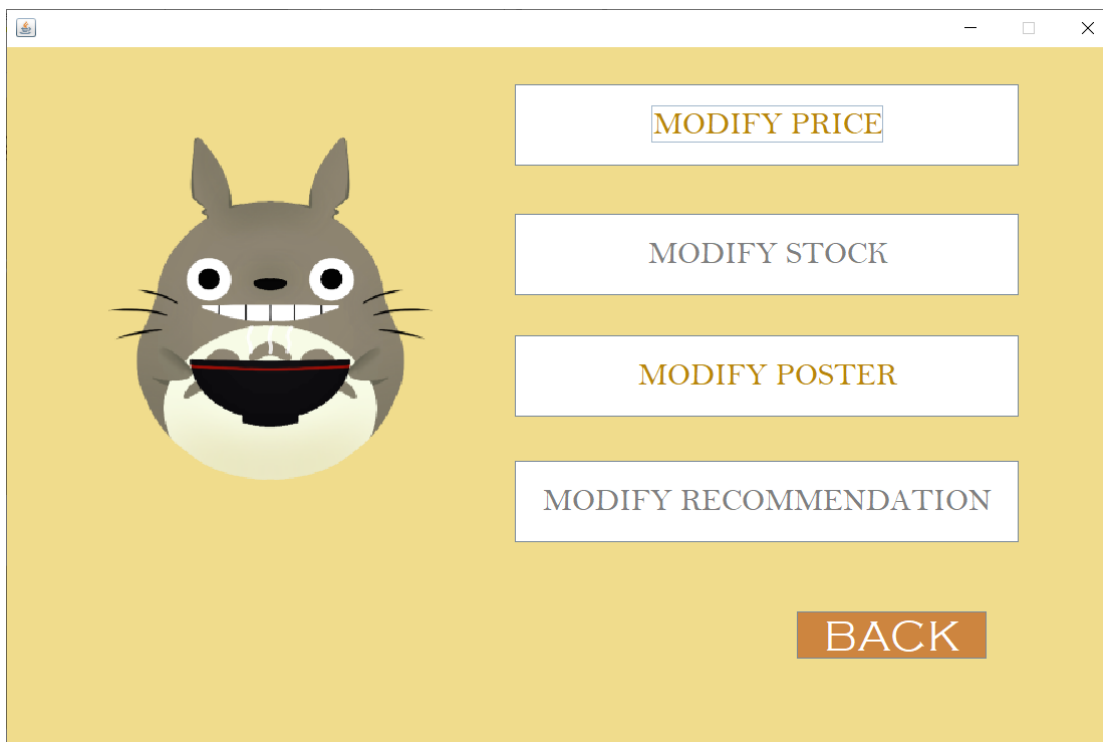
3-10 letters or numbers

Password:

6-18 letters or numbers

SAVE CANCEL

In this page, restaurant manager can change username and password of the administrator account.



MODIFY PRICE


MODIFY STOCK

MODIFY POSTER

MODIFY RECOMMENDATION

BACK

This is modification menu page. Restaurant manager can choose modify price, stock, poster and recommended add-on.

 — □ ×

MODIFY PRICE

Tip: Price should be a positive number with a maximum of two decimal places

<i>Ramen</i>	<input type="text" value="9.9"/>	<i>Extra Nori</i>	<input type="text" value="1.0"/>
<i>Extra Boiled Egg</i>	<input type="text" value="1.0"/>	<i>Extra Chashu</i>	<input type="text" value="1.0"/>
<i>Bamboo Shoots</i>	<input type="text" value="1.0"/>		

SAVE

CANCEL

Price can be changed in this page.

 — □ ×

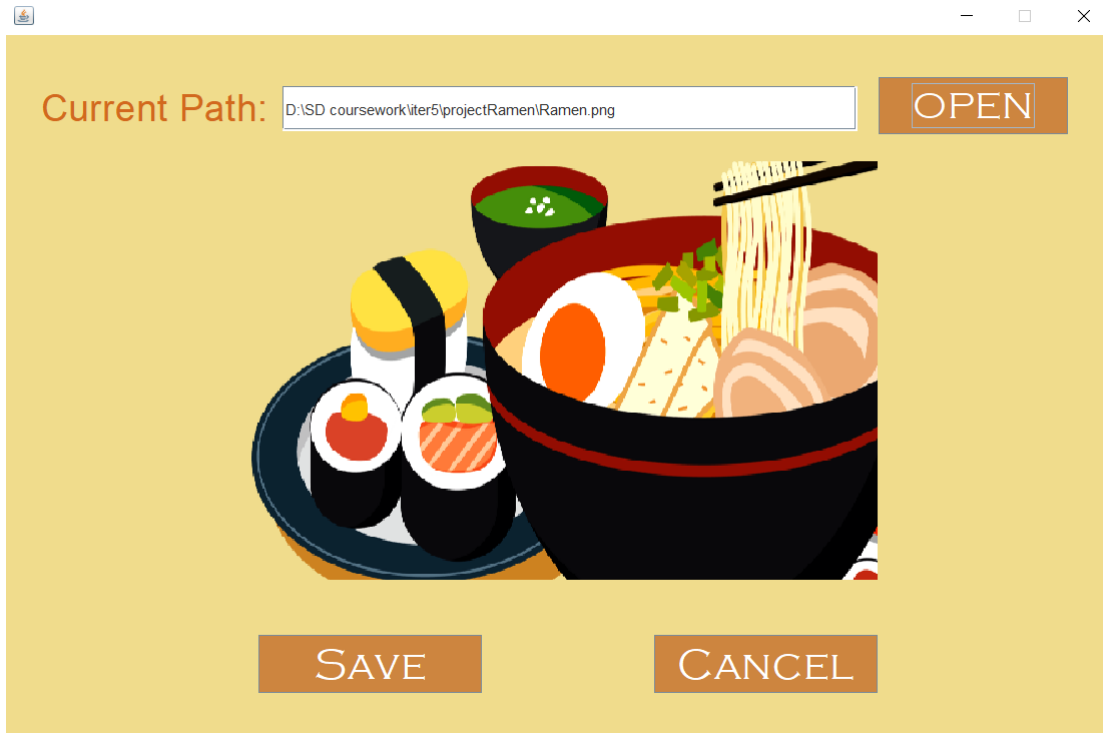
MODIFY STOCK

<i>Soup:</i>	<i>Tonkotsu</i>	<input checked="" type="checkbox"/>	<i>Shoyu</i>	<input checked="" type="checkbox"/>	<i>Shio</i>	<input checked="" type="checkbox"/>
<i>Noodles:</i>	<i>Soft</i>	<input checked="" type="checkbox"/>	<i>Medium</i>	<input checked="" type="checkbox"/>	<i>Firm</i>	<input checked="" type="checkbox"/>
<i>Spring onion:</i>	<i>No please</i>	<input checked="" type="checkbox"/>	<i>Just a little</i>	<input checked="" type="checkbox"/>	<i>A lot!</i>	<input checked="" type="checkbox"/>
<i>Nori</i>	<input checked="" type="checkbox"/>	<i>Chashu</i>	<input type="checkbox"/>	<i>Boiled egg</i>	<input checked="" type="checkbox"/>	
<i>Extra Nori</i>	<input checked="" type="checkbox"/>	<i>Extra Chashu</i>	<input type="checkbox"/>			
<i>Extra boiled egg</i>	<input checked="" type="checkbox"/>	<i>Bamboo shoots</i>	<input checked="" type="checkbox"/>			

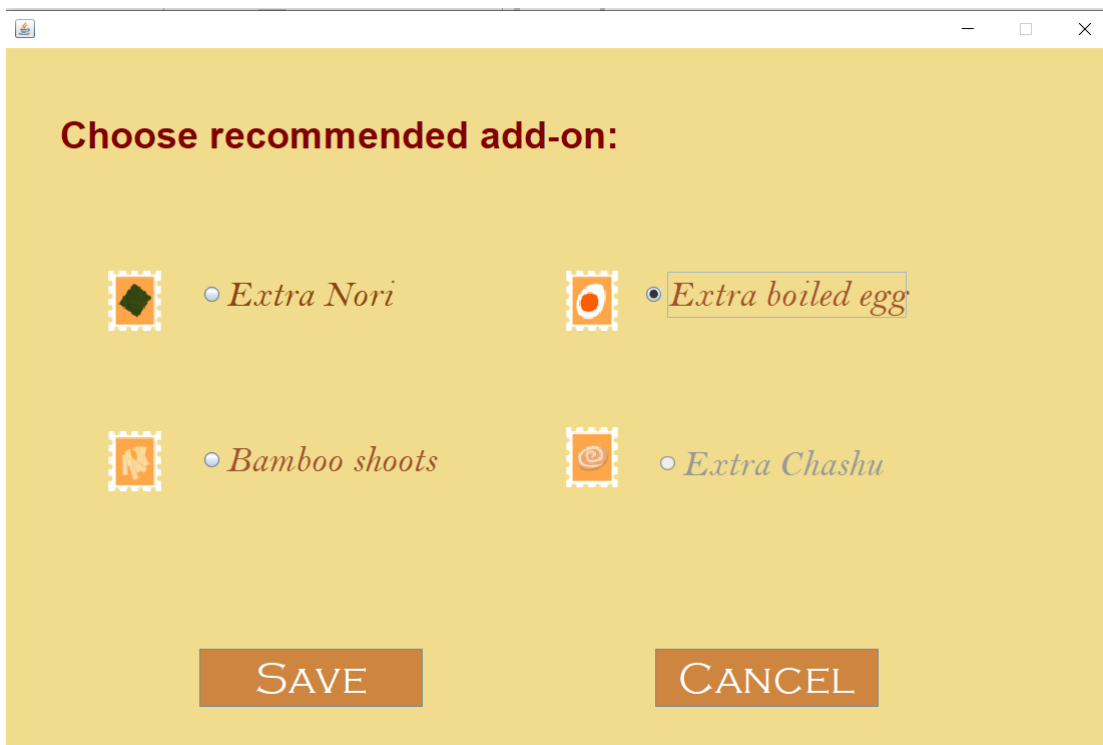
SAVE

CANCEL

Stock can be changed in this page.



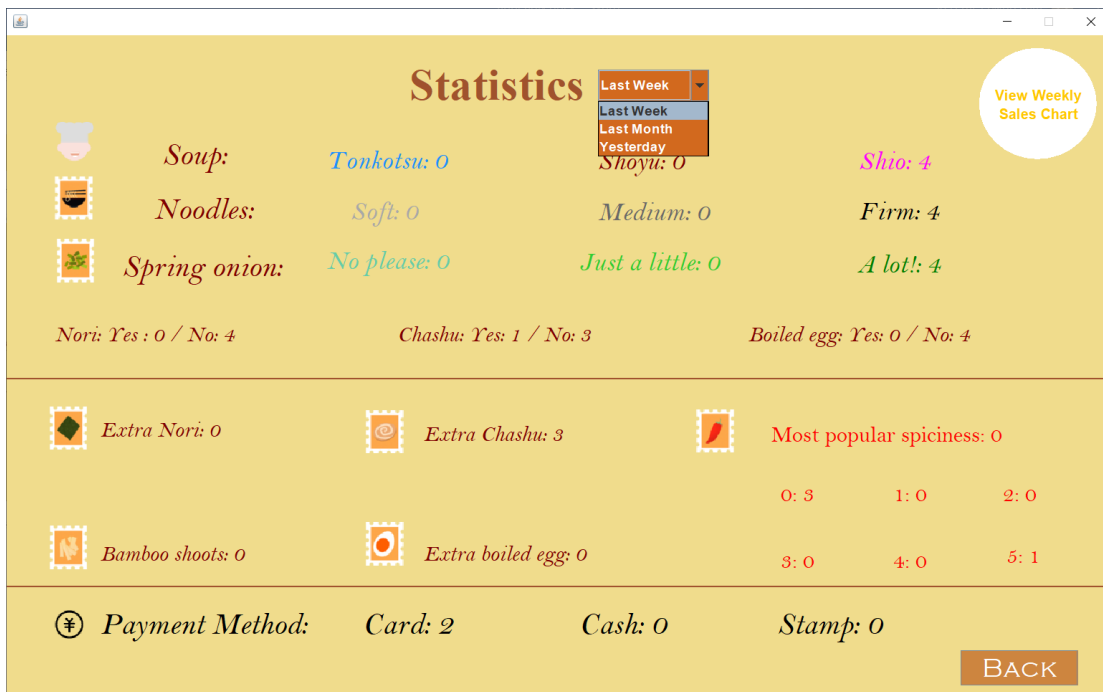
Poster can be changed in this page.



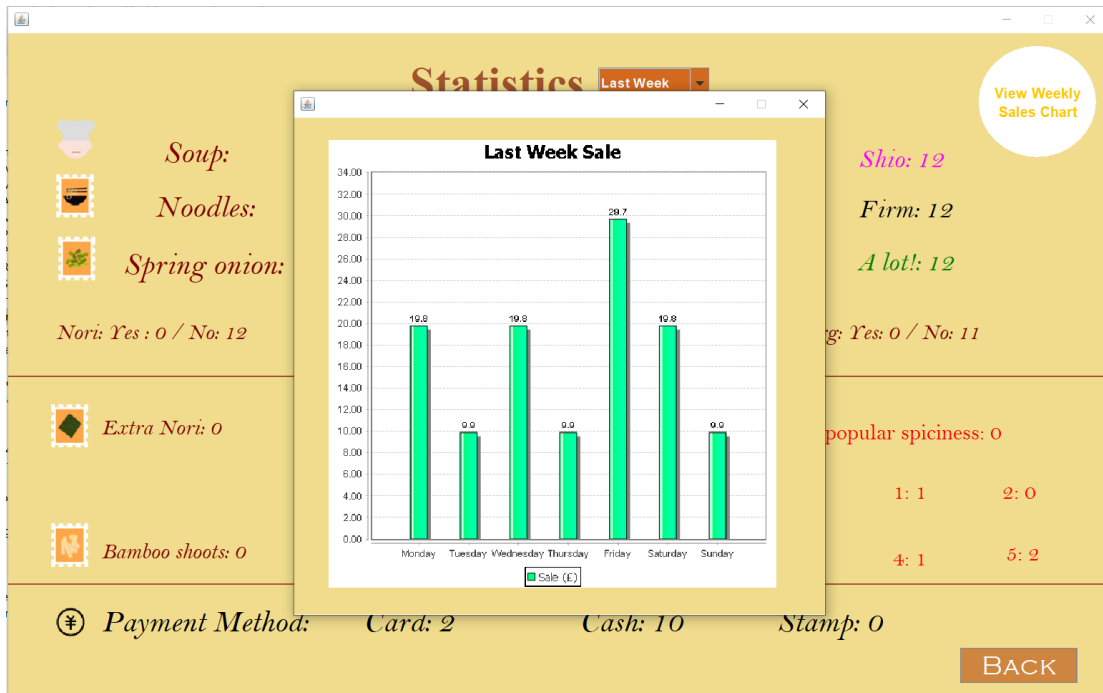
Recommended add-on can be changed in this page.



Report generation config can be changed in this page.



In this page, manager can view how many times each item on the menu has been sold in the past week and generate sales chart. Manager can also choose to view statistics of last month or yesterday.



This is an example of last week sales chart.

Appendix-3 Test Cases

All the tests below are based on junit. Please read the junit code.

AccountTest

AccountTest:

Input(String,String)		Expect Output	
boss	ramen123	boss	ramen123
Big	666	Big	666
Cat	123	Cat	123
YYY	yyy	YYY	yyy
Tom	cat	Tom	cat
lala	555	lala	555
lty	666	lty	666
Auto	man	Auto	man
♂	♀	♂	♀
&&&	@@@	&&&	@@@

AccountServiceTest

AccountService:

Input(String,String)		Expect Output
wxt	123456	1
wsdadd	123456	0
wxt	1521123.121	0
null	null	0
好 123	asdasd	0
♂ 21345	♀ ashofl	0

SaveAccount:

Input: The user input his or her information and save it

Expect Output: 1,the information can be stored correctly

PaymentServiceTest

Default Data	Boiledegg	Chashu	Noodles	Nori	Soup	Spiciness	Springonion	Chashuadd
	1	1	1	2	1	5	1	1

getPrice:

Input: Each quantity of the dish, is discounted or not, the method to enjoy the food

Expect Output: The original price of the order which has discount

Input		Expect Output:
"Take-out"	no discount	7.1
"Take-out"	is discount	0

getNoDisPrice:

Input: The order which has already have discount

Expect Output: The original price of the order which has discount

savePayment:

Input: The information of each order

Expect Output: The information of each order can be stored in a csv file correctly

MemberServiceTest

memberRegTest:

Input(String,String,String,int)				Expect Output
<u>Li</u>	Hua	123@qq.com	1234	000000001
Lsdasfasfasf	Hua	123@qq.com	1234	000000002
Ls 陈哦哦哦 fasf	afasf 好	121113@qq.com	1234	000000003
Ls♥++sfasf	Hu+++~!@a	123q@q.com	1234	000000004
来来来	略略略	121113@qq.com	1111114	000000004

memberLoginTest:

Input	Expect Output
88888888	FALSE
99999999	FALSE
00000001	TRUE
00000002	TRUE
00000003	TRUE
82456879	FALSE
45678135	FALSE
23594684	FALSE
23176282	FALSE
95324841	FALSE

ModifyServiceTest

savePrice:

Input: Different dishes with different price

Expect Output: 1, the price can be stored in the system correctly

Input(String,double)		Expect Output:
Egg	1	1
Ramen	6.6	1
Chashu	0.5	1
Bintang	1	1
Xiangcong	1	1
Zhacai	1	1
Hongshu	2	1
Tudou	1.5	1
Xiangchang	1	1
Luobo	1	1

saveStock:

Input		Expect Output
Noodles1	100	1
Spring onion2	100	1
Noodles0	200	1
Spring onion0	400	1
Chashu	100	1

savePicture:

Input: A path with the new picture need to be stored and its name

Expect Output: 1, the new picture with name can be stored correctly

OrderServiceTest

ChooseAndBuy:

	Boliedegg	Chashu	Noodles	Nori	Soup	Spiciness	Springonion	Chashuadd	Expect Output
Default	1	1	1	2	1	5	1	0	True
Test1							4		False
Test2						6	0		False
Test3						0			True
Test4								4	True
Test5			2					6	True
Test6		2			5				False
Test7	7								False
Test8				6					False
Test9			2			3			True
Test10	4						2		False

saveOrder:

Input:The information of each order

Expect Output: The information of each order can be stored correctly in a csv file.

Appendix-4 TDD in Menu.java

MenuTest TDD

Disciplined techniques: Black-box testing

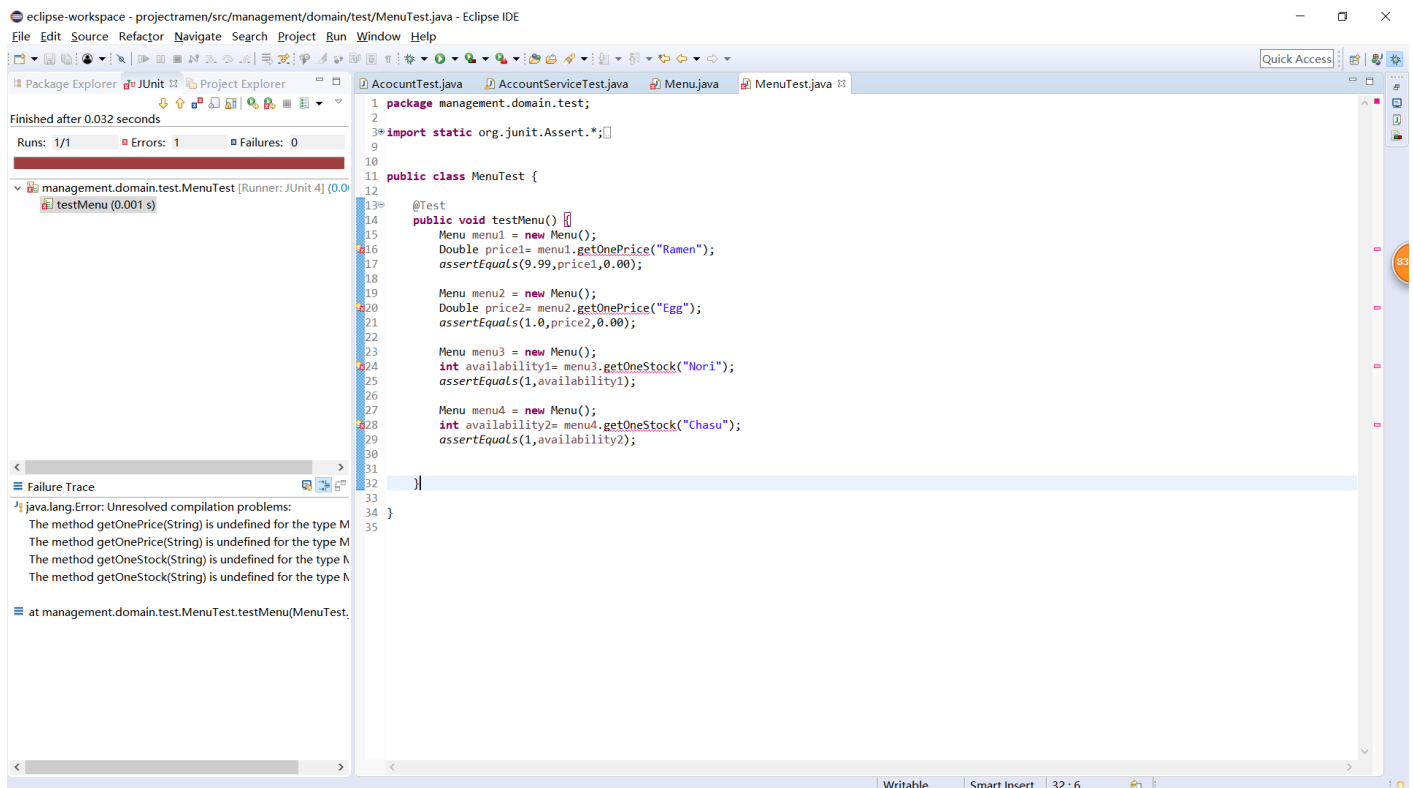
MenuTest:

Input(String): The price and the stock of different dishes

Expect Output: The correct price of each dish

Input	Expect Output
Ramen	9.99
Egg	1
Chasu	1
Nori	1

Fail:



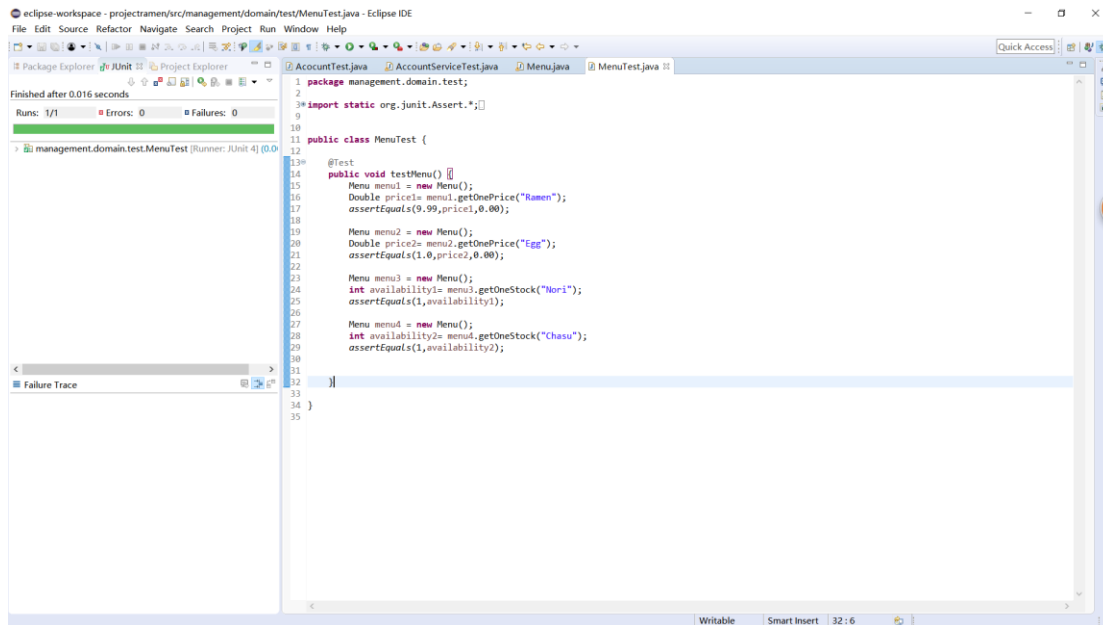
Then write the code, and get passed:

```

1 package management.domain;
2
3 import java.math.BigDecimal;
4
5
6
7 public class Menu {
8     private Map<String, Double> price = new HashMap<>();
9     private Map<String, Integer> stock = new HashMap<>();
10    private Map<String, String> rec = new HashMap<>();
11
12
13    public Menu() {
14        //read data from file
15
16        // 测试用
17        price.put("Ramen", 9.99);
18        price.put("Egg", 1.0);
19        stock.put("Nori", 1);
20        stock.put("Chasu", 1);
21    }
22
23
24
25    public Map<String, Double> getDishesPrice() {
26        return price;
27    }
28
29    public Map<String, Integer> getStock() {
30        return stock;
31    }
32
33    public void setDishesPrice(Map<String, Double> price) {
34        this.price = price;
35    }
36
37    public void setStock(Map<String, Integer> stock) {
38        this.stock = stock;
39    }
40
41    public double getOnePrice(String name) {
42        return price.get(name);
43    }
44
45    public int getOneStock(String name) {
46        return stock.get(name);
47    }
48
49
50

```

Refactor1:

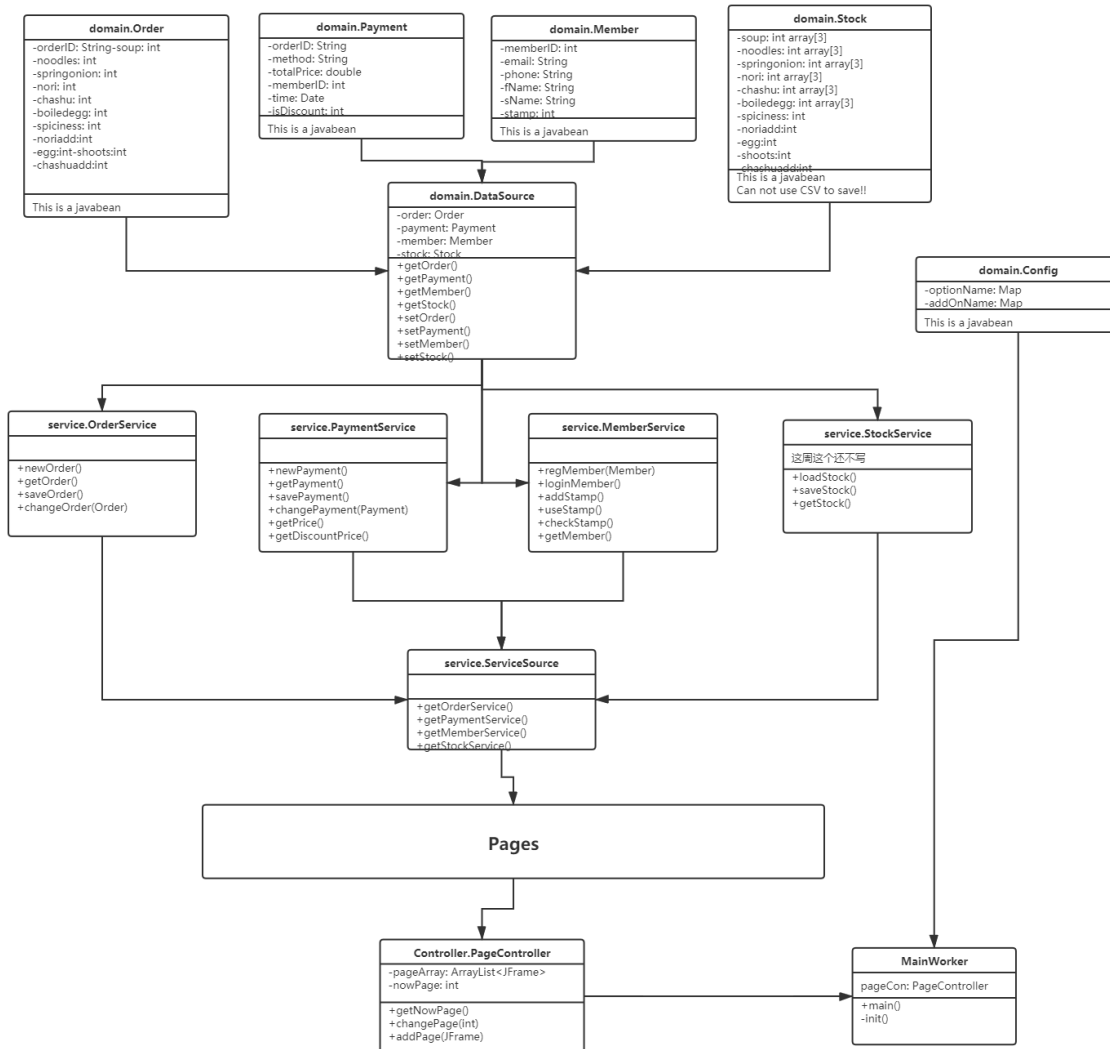


Refactor2:

```
33= public void setDishesPrice(Map<String, Double> price) {
34     this.price = price;
35 }
36
37= public void setStock(Map<String, Integer> stock) {
38     this.stock = stock;
39 }
40
41= public double getOnePrice(String name) {
42     return price.get(name);
43 }
44
45= public int getOneStock(String name) {
46     return stock.get(name);
47 }
48
49= public void setOnePrice(String name, double price) {
50     this.price.put(name, price);
51 }
52
53= public void setOneStock(String name, int status) {
54     this.stock.put(name, status);
55 }
56
57
58= public Map<String, String> getRec() {
59     return rec;
60 }
61
62
63= public void setRec(Map<String, String> rec) {
64     this.rec = rec;
65 }
66
67
68
69
70= public static void main(String[] args) {
71
72     Menu menu = new Menu();
73     menu.getDishesPrice().put("Ramen", 9.99);
74     System.out.println(" "+menu.getOnePrice("Ramen"));
75 }
76 }
77
```

Appendix-5 UML diagram

Earlier versions of UML diagrams:



The figure above is just a schematic of the software structure, which was developed in the early days of software development.

