University of St Andrews | FOUNDED 1413

CS5030-Software Engineering Principles

Matriculation Number: 200009834

Content

University of St Andrews | FOUNDED 1413

Specification of functional requirements:

The application needs to implement:

1. Users can browse the items in the app.
2. Users can register or log in to become buyers and sellers.
3. User registration requires necessary information. Fill in name, email, address, date of birth, mobile phone number.

1. Sellers can log in.

After the seller logs in:

2. Sellers can publish a list of items to be sold.
3. The seller can view the item list.
4. The seller can change the status of the item.
5. The seller can log out.

1. Buyers can log in.

After the buyer logs in:

2. Buyers can view their purchase history.
3. Buyers can bid on items.
4. Buyers can pay for the winning items.
5. Buyers can log out.

Construction of the item list system:

1. The item list needs to contain the necessary information. Provide the item's name, description, usage status (new/already used), photo, starting price, starting time, auction duration, postage price, status of listing (draft, in progress, ended).
2. The item list only displays items that are being auctioned for the user.
3. When the item auction is over, the system should notify the winner.

1. The system should be linked to the payment system
2. The application should be associated with the postage calculation system.

Specification of non-functional requirements:

Interface requirements:
The interface of the entire application should be simple and easy to understand, and support various disability modes, such as color blindness, hearing impairment, and understanding impairment.

Usability requirements and executable requirements:
The number of users supported by the application should be no less than 1000, the response time that users can accept is within 100ms, and the data scale has tens of thousands of data.

Reliability/availability/recoverability:
The application requires the system to run 7x24 hours, and the cumulative outage time of continuous operation throughout the year cannot exceed 10 hours. Data can be quickly rebuilt when errors and failures occur in the application. For example, after the transaction submission fails, it is necessary to ensure the rollback. The money that cannot be banked is not successfully paid, but the user shows that the payment is successful or the bank deducts the money. When the user changes the transaction status, the transaction is still due to redundancy or other reasons. Unpaid status.

Security requirements:
Ensure that transmission encryption, storage encryption, unbreak ability, and various unauthorized user behaviors need to be prevented and controlled. And for the control of internal users with different levels of authority.

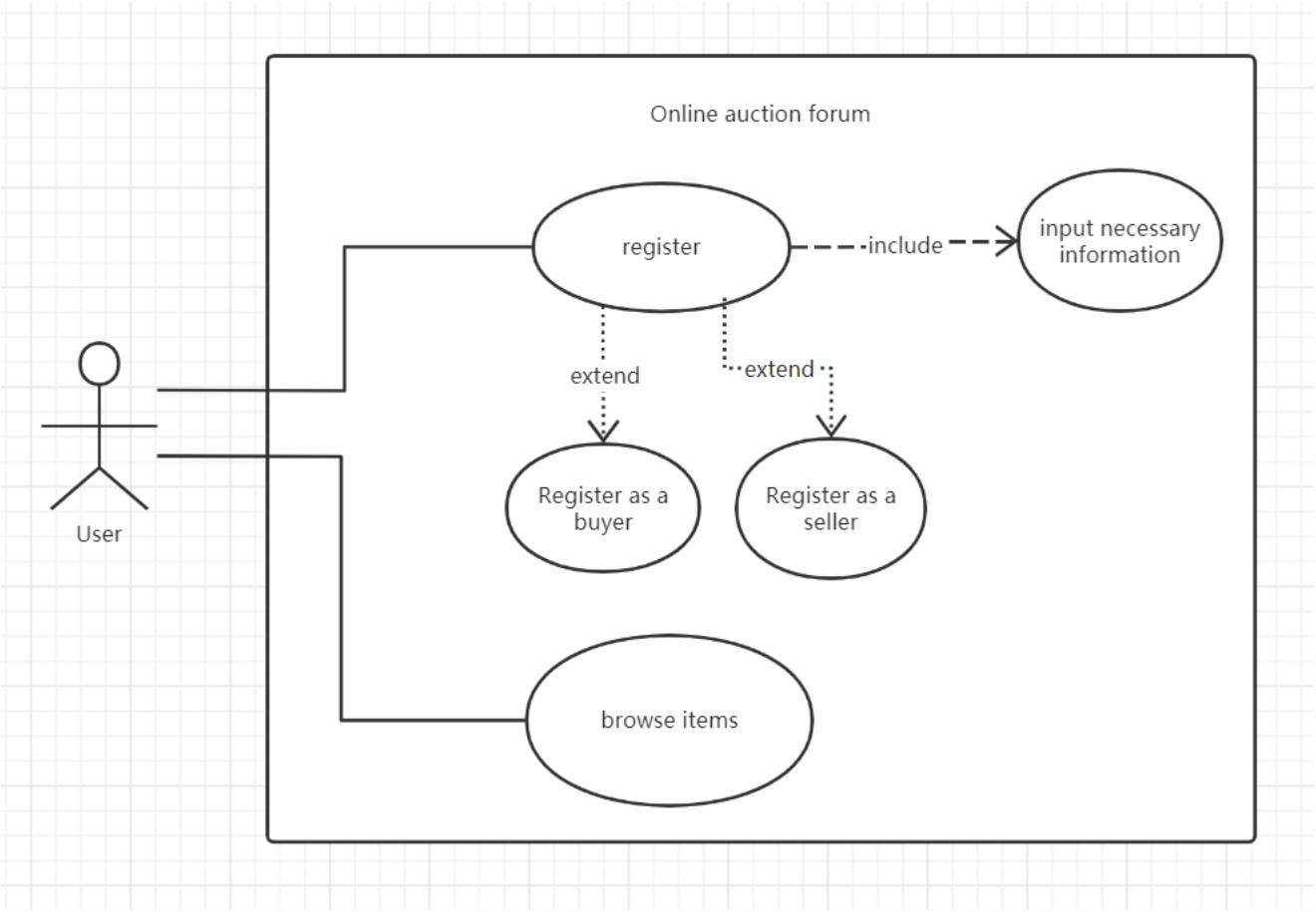
System integrity requirements:
Among them, data consistency is the most important, including data coding and language, and redundant data consistency requirements (including time requirements).

System/environmental conditions and restrictions:
Applications are mobile Internet products. Developers should consider different user network conditions, as well as terminal computing performance and capabilities, as well as the user's mobile network stability.

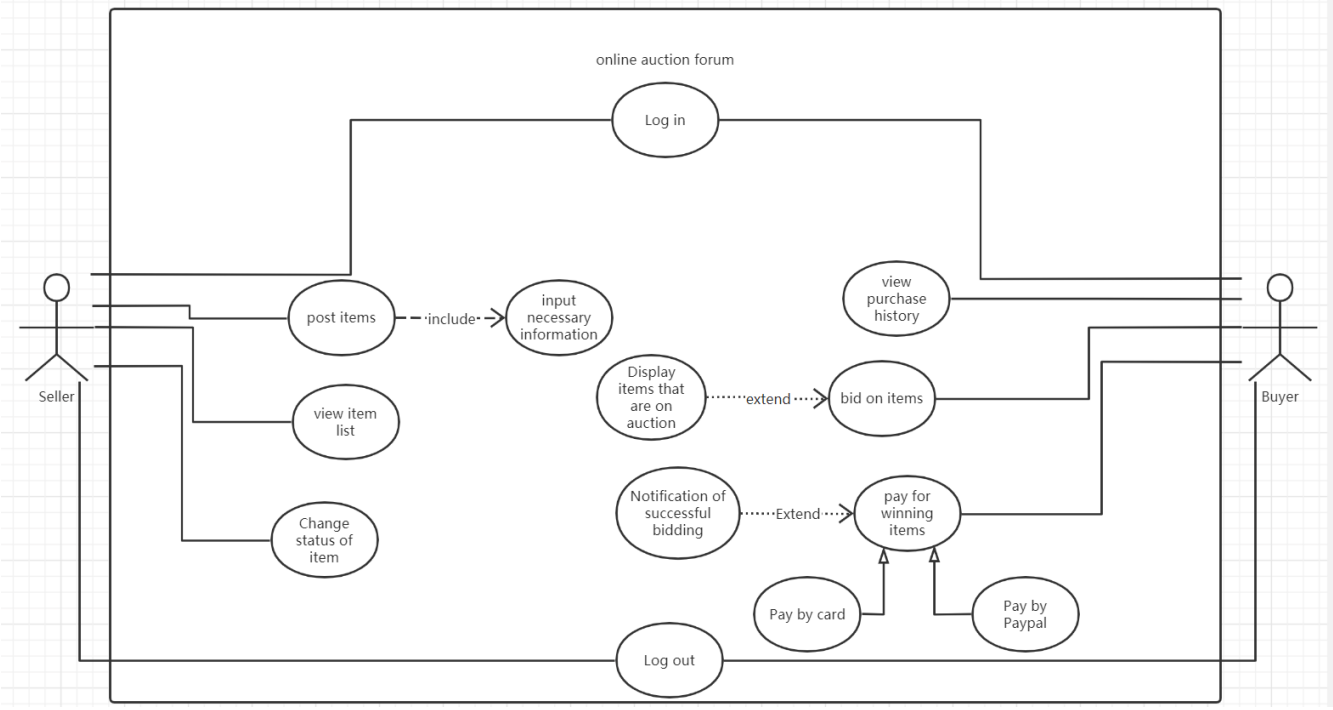
Scalability and maintainability of the system:
The system can operate stably without manual intervention and has self-debugging capabilities. The ability to troubleshoot faults, system corrections, upgrades, backups, and recovery mechanisms need to be simple and easy to debug, which can reduce system operation and maintenance costs and maintenance difficulties.

Use case diagram:

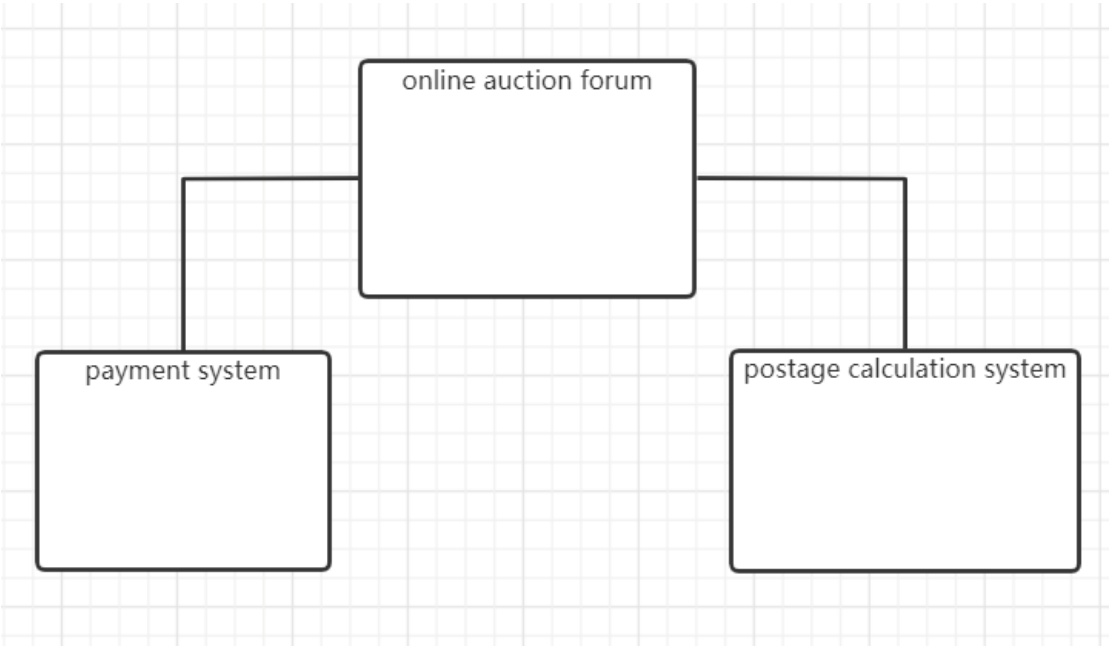
Use case 1:



Use case 2:



Use case 3:


online auction forum
payment system
postage calculation system

Use case specification:

Use Case 1	User Registration
Actor Action	System Response
1 –User chooses registration as seller/buyer.	2 – System displays registration screen.
3 –User enters his/her details (should include name, email, address, date of birth and phone number).	4 – System displays seller/buyer main menu.
Alternative route	
3 (a) – User enters invalid details.	4 (b) – Registration fails. Appropriate error message displayed.
3 (c) –User already has an existing account.	4 (d) - Registration fails. Appropriate error message displayed.

Use Case 2	Seller/Buyer Login
Actor Action	System Response
1 –Seller/Buyer enters username and password.	2 – Login details are checked. Displays main menu.
Alternative route	
1 (a) –Seller/Buyer enters incorrect username or password.	2 (b) – Login check fails. Error message is displayed.

Use Case 3	Seller/Buyer Log Out
Actor Action	System Response
1 –Seller/Buyer selects log out option.	2 – System displays log out screen.

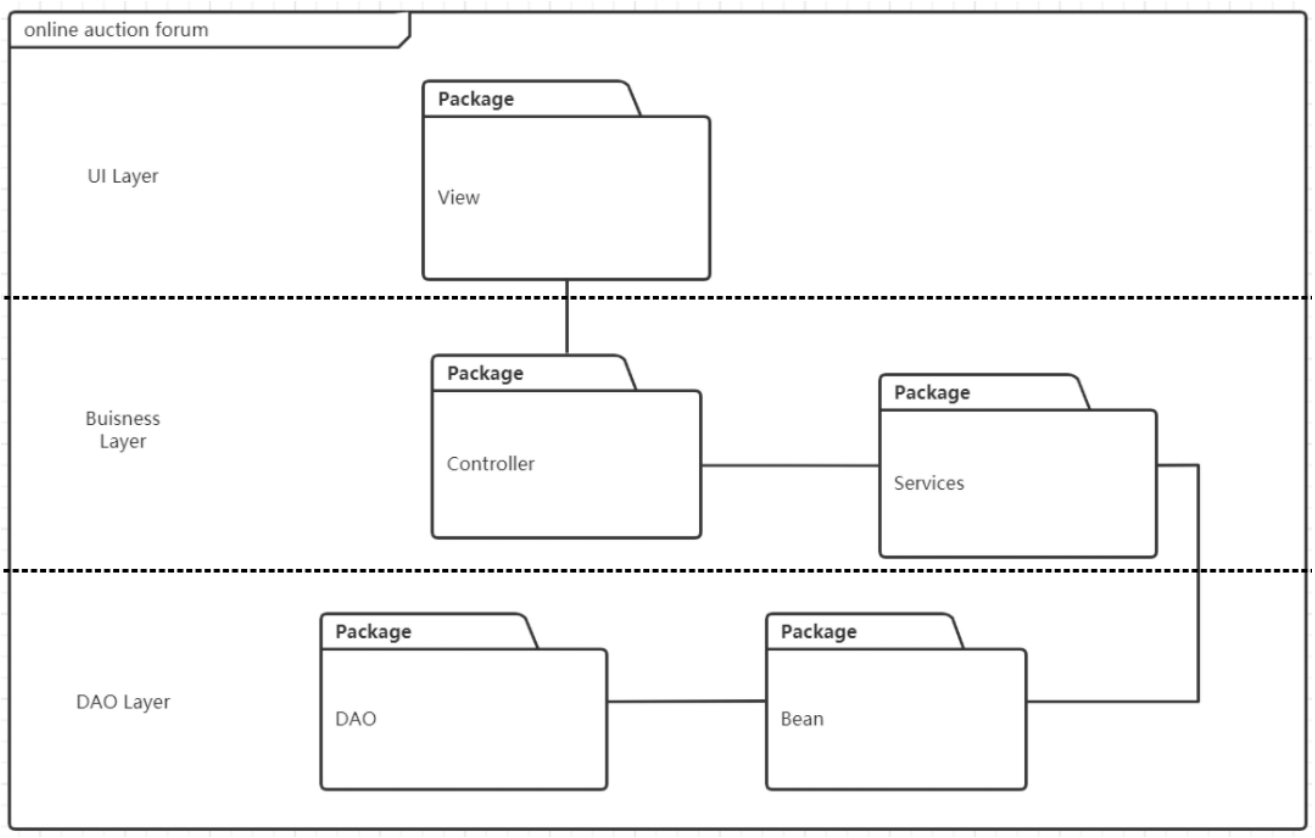
Use Case 4	User browse items
Actor Action	System Response
1 –User open the application.	2 – System displays items list.

Use Case 5	Seller post items
Actor Action	System Response
1 - Seller chooses item post button.	2 - System returns post item page.
3 - Seller chooses item and input necessary information to post to application.	4 - System returns post successful.
Alternative route	
3 (a) - User puts incorrect information.	4 (b) – System return fails, error message is displayed.

Use Case 6	Buyer paying for winning items
Actor Action	System Response
	1- System notification successful bidding.
2- Buyer pays for winning items by card or PayPal.	3 - System returns paying successful
Alternative route	
2 (a) – Buyer gives wrong information.	3 (b) - System returns error. And show the message.
2 (c) – Buyer not have enough money.	3 (d) – System returns insufficient balance

Software architecture:

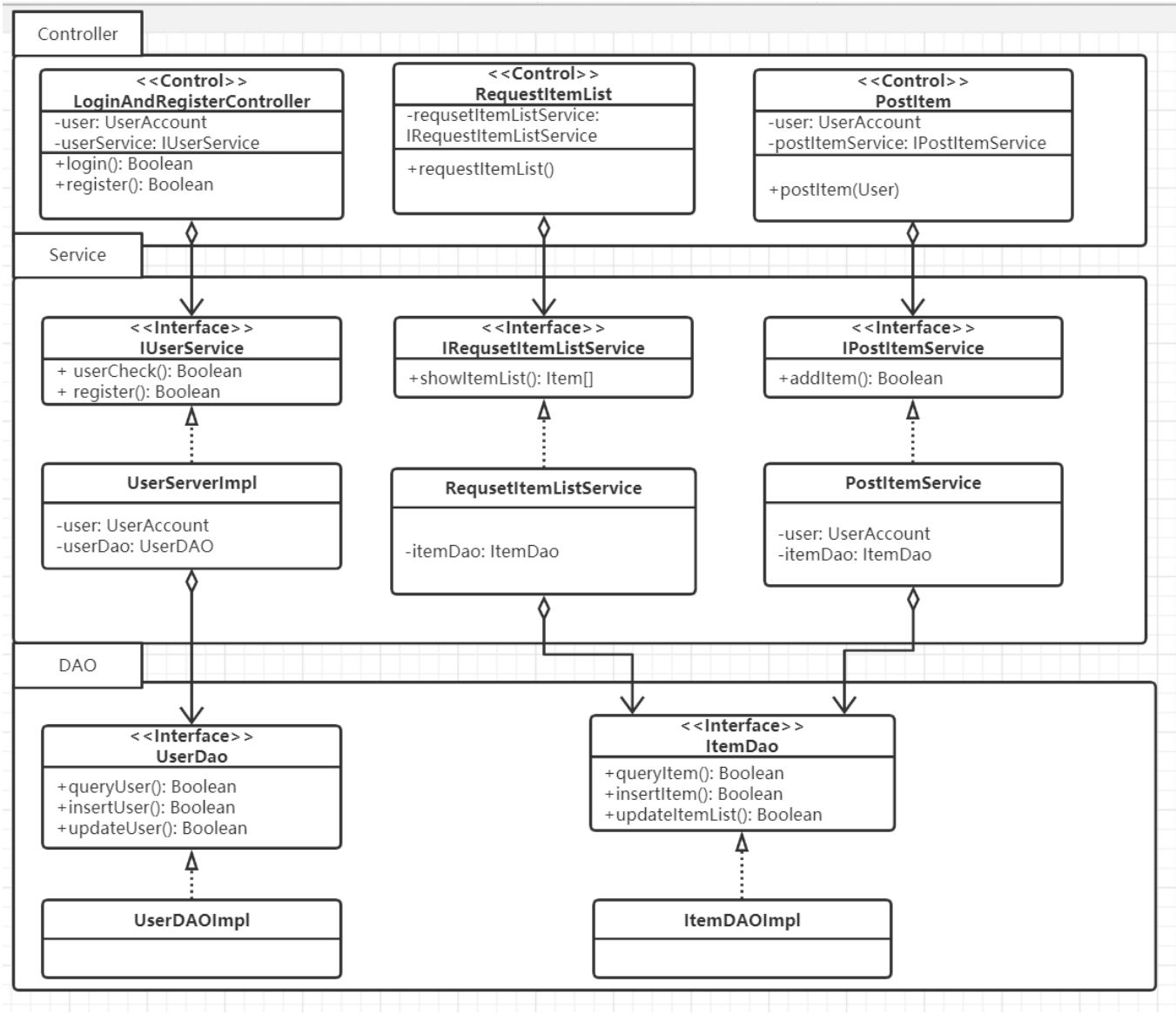
In order to reflect the logical structure of the entire program, I decided to use a package diagram



As shown in the figure, the entire program is divided into three parts, views, business logic processing, and data access. The view is mainly the processing layer that interacts with the user. When the user clicks on a function on the view, the request is transferred to the control package. The control package is distributed to different service packages according to the request, and the logical processing is performed in the service package. The data is encapsulated in a bean package. The bean package assigns data to the data access object(DAO), and finally the DAO interacts with the database. The data access here is bidirectional, data can be extracted or stored.

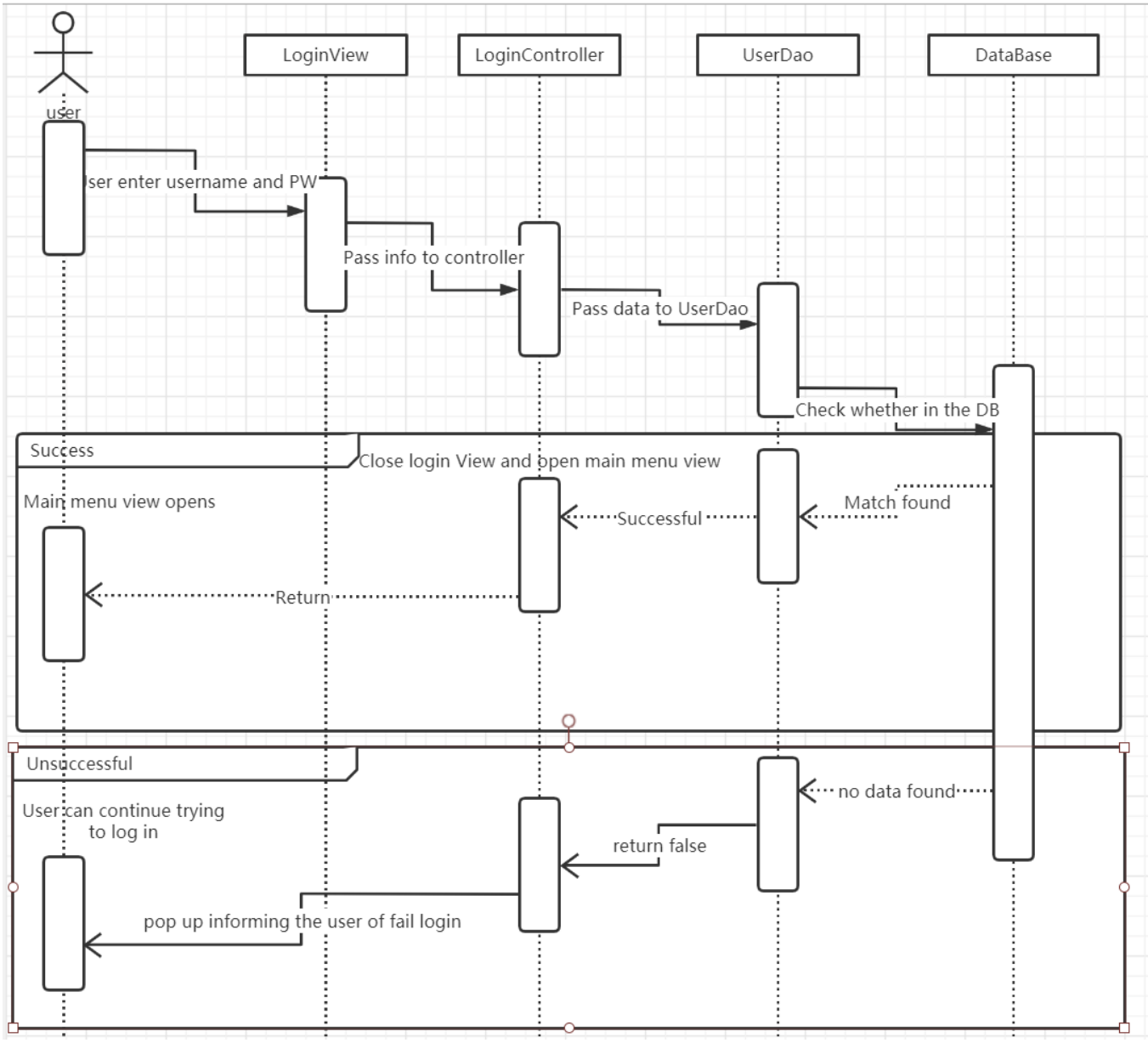
UML class diagram:

According to requirements, I chose to use class diagrams to express part of the software implementation.



UML sequence diagram:

I choose to use a sequence diagram to represent interaction sequence.

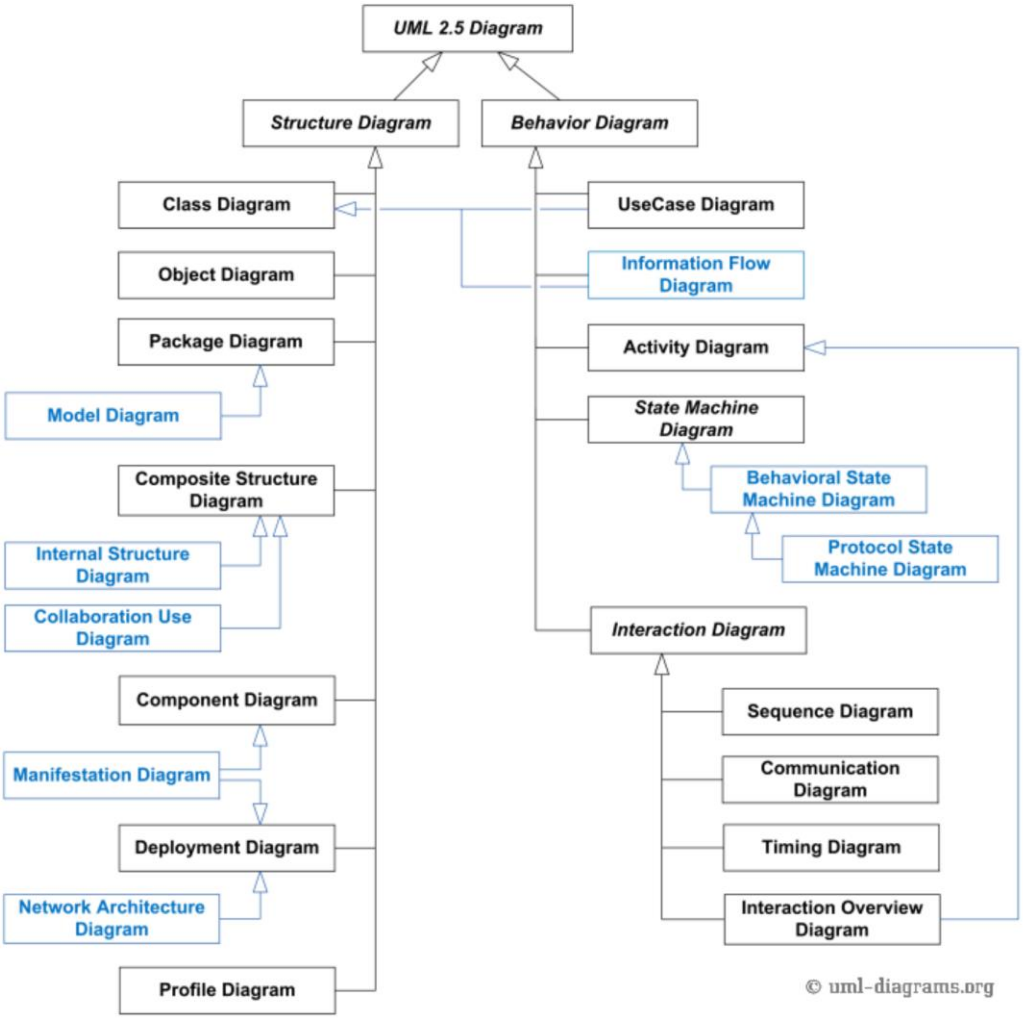


Review design:

First of all, in the first part, there are not too many functional requirements that need to be considered, just extract the relevant functions that need to be implemented from the text. However, non-functional requirements need to consider a lot of elements, this part is the focus of program design, can significantly affect the experience of the program. So, I have considered a lot in this part, such as what problems will be encountered in the development and operation of this program, and how to prevent and remedy these problems. For non-functional requirements, I spent a lot of time thinking about it from different angles.

The second part is not very difficult, just draw the use case step by step according to the functional requirements of the first part. The difficulty encountered is the relationship between each case and the logical idea of how to distribute the picture.

The third part is a detailed explanation of several key cases in the use case. The difficulty encountered is that I don’t know whether the consideration is complete. For example, when logging in, are there other aspects to consider besides denying access?



Picture1(https://www.uml-diagrams.org/uml-25-diagrams.html)

For the fourth part, the main problem is the choice of UML diagrams. After a long hesitation and according to the latest version of UML in Picture1, I decided to use UML package diagrams, because the title requires to reflect the logical structure of the entire software, and the package diagram is to observe the entire system from a macro perspective.

The main problem of the fifth part appears in the construction of the class, we must carefully consider the methods that need to be implemented and the member variables included. In fact, the UML diagram at the end did not satisfy me, and I always felt that something was missing.

There are not too many problems encountered in the sixth part, mainly in the order.

In general, this project deepened my deep impression of the software requirements and made up for many of my vulnerabilities and shortcomings.

Reflection of UML:

The diagram is clearer than the code. UML diagrams are very necessary in complex requirements. UML diagrams can help the team to efficiently identify problems, master knowledge, and solve problems efficiently.

I carefully read the process of agile development and found that if it is used improperly, there will be a lot of problems at the communication level that shouldn't be a problem, and there is no unified standard. However, UML unifies the different views of various methods on different types of systems, different development stages and different internal concepts, thereby effectively eliminating unnecessary differences between various modeling languages.

But I checked the UML2.5 standardized document which has about 800 pages, which is basically the same as the Java standard document. It is too much to read and it is very difficult to understand, not to mention my native language is not English. I have tried to draw a UML that can generate a complete code, but failed and encountered a lot of problems. It is much harder to write than Java programs in the same situation, and the workload is much larger, so I think that in most cases the next step is completely unnecessary. But different UML diagrams imply a kind of perspective thinking on modeling, which is worth learning. I think it is a great help to improve engineering ability, and can understand software systems from different angles. In my opinion, the focus of learning UML should not be on lexical grammar. Focus on understanding how UML reflects requirements, overall design, business logic design, verification and testing, specific implementation, etc., and the similarities and differences from different perspectives, and coordinate and balance different considerations in the software system.