Requirement Analysis Document

Chapter 1-User Scenario

Below, we will imagine the user's usage scenario and draw a user scenario diagram.

1.Homepage Case:

1. The homepage of the emoji pack platform is displayed on the user screen, which includes recommended popular emoji packs and search boxes.
2. A user is browsing the homepage, scrolling the screen, and seeing thumbnails of various emojis.

2.Search Case:

1. Users enter keywords such as "cat" in the search box, and the search results page displays multiple emojis related to "cat".
2. Users click on a search result to view detailed information.

3.Emoji Detail Page Case:

1. The user opens a detailed page of the emoji package, where they can see a large image, name, author, and related information of the emoji package.
2. Users can click the button to add emojis to their favorites or share them on social media. Upload emoji case:
3. Users may need to fill in relevant information such as the name, label, and author information when uploading their own created emojis.
4. During the upload process, users can preview the emoji package and choose whether to share it publicly.

4.Personal Favorites Use Case:

1. Users view their personal favorites, which include their favorite emojis.
2. Users can manage their favorites, add and remove emojis.

5.Social Sharing Case:

1. The user selects an emoji pack, clicks the share button, and shares it with friends or followers on social media platforms.
2. On social media, friends can view emojis and interact with users.

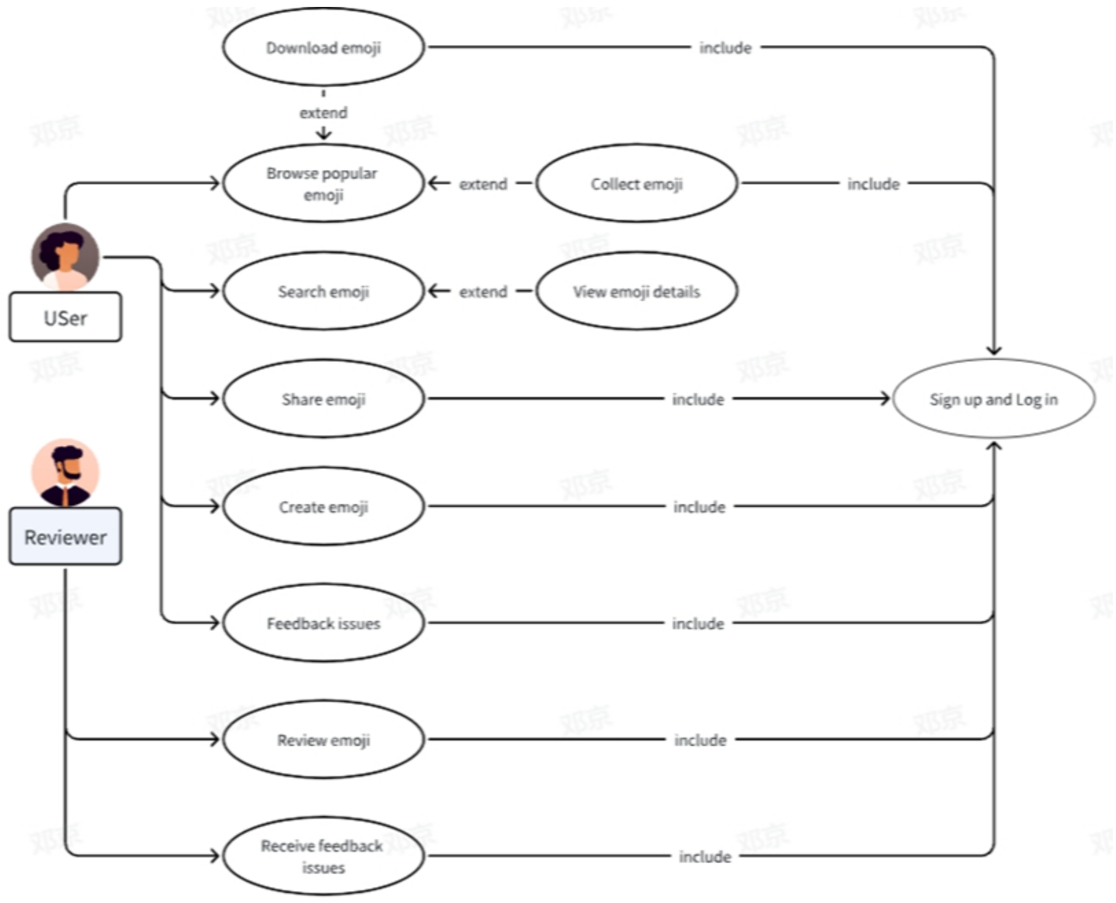
6.Emoji Creation Case:

1. Users use built-in tools or third-party applications to create their own emojis.
2. Users can add elements such as text, graphics, and stickers to personalize emojis.

7.Feedback and Support Case:

1. If users discover problems or have suggestions, they can contact the support team and fill out the feedback form.
2. The support team may respond to user questions or provide assistance.

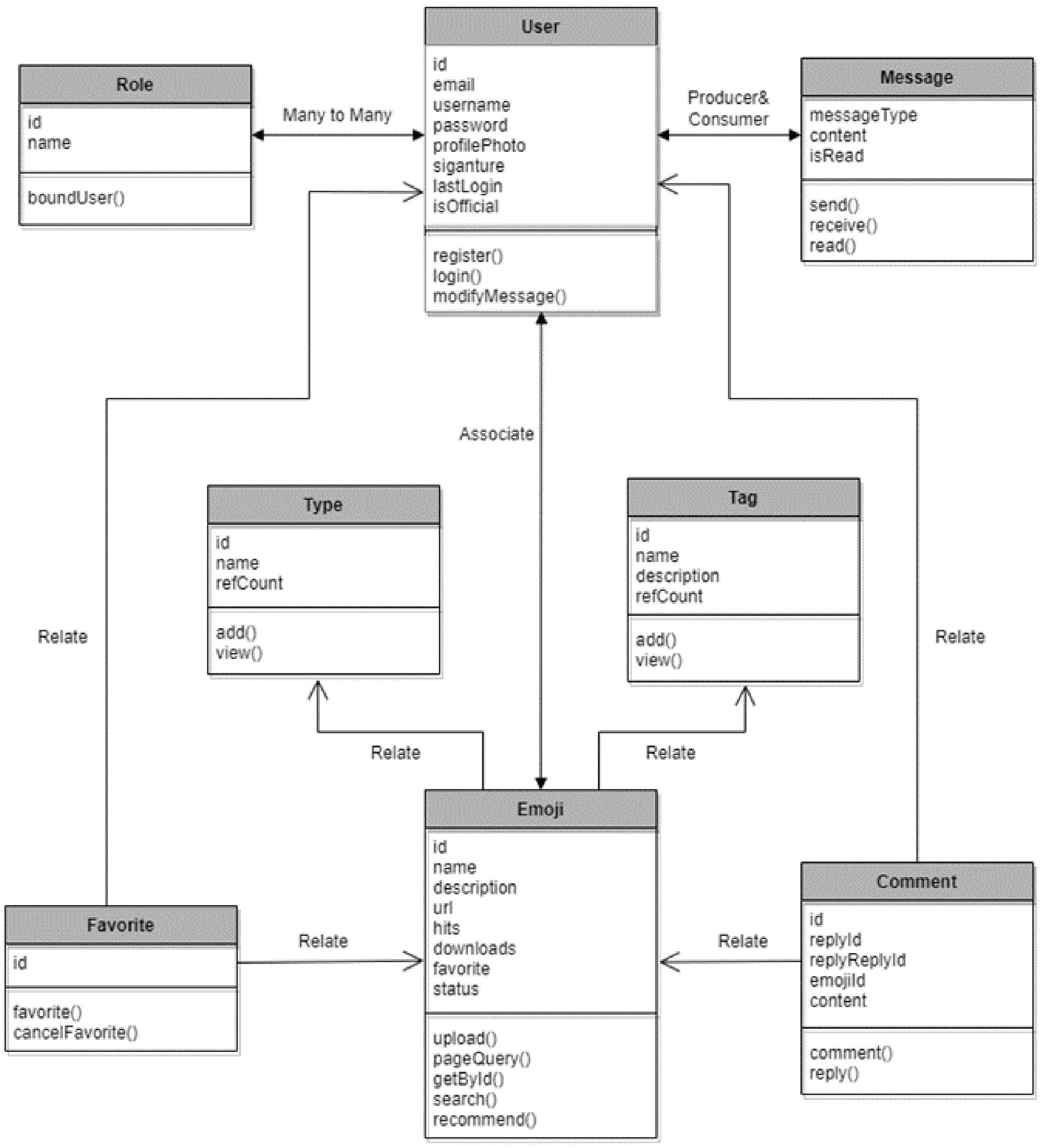
8.Draw a Scene Diagram Below:



Chapter 2-Class-Based Modeling

1.Class Diagram

We isolated the nouns in the requirements, abstracted some classes, and defined the relevant properties and operations, drawing the following class diagram:



2.CRC Modeling

Next, we perform CRC modeling for each class to get the CRC model index card:

Class:User	
Explaination:userinfo	
Responsibility:	**Collaborator:**
Define user information	Role
User registration	Emoji
User Login	Favorite
User modify presonal information	Comment
	Message

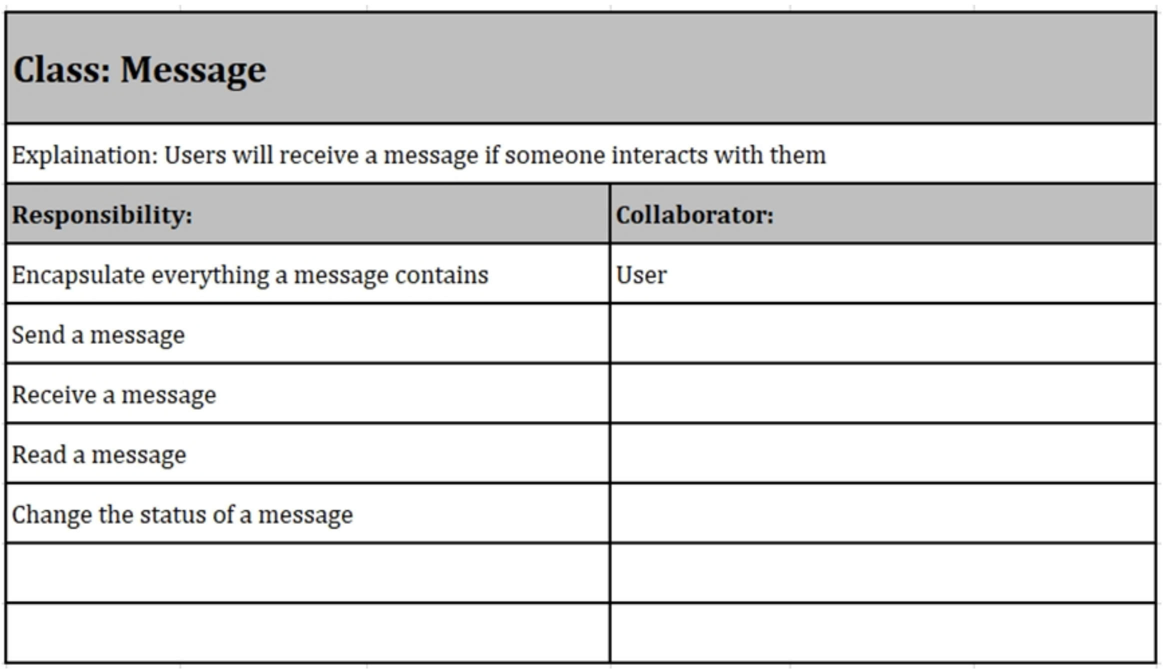
Class: Emoji	
Explaination: The abstraction of the message about the emoticons	
Responsibility:	**Collaborator:**
Encapsulate the emoji message	User
Upload a emoji	Type
Browse all emoticons	Tag
View details of one emoji	Favorite
Users search for the emoticons they want	Comment
Show popular emoticons in the home page	
Push emoticons to users based on a specific algorithm	

Class: Tag	
Explaination: Every emoji can have several tags	
Responsibility:	**Collaborator:**
Define name, description and count of references	Emoji
View all tags	
Add a tag	
Bind to emoji	

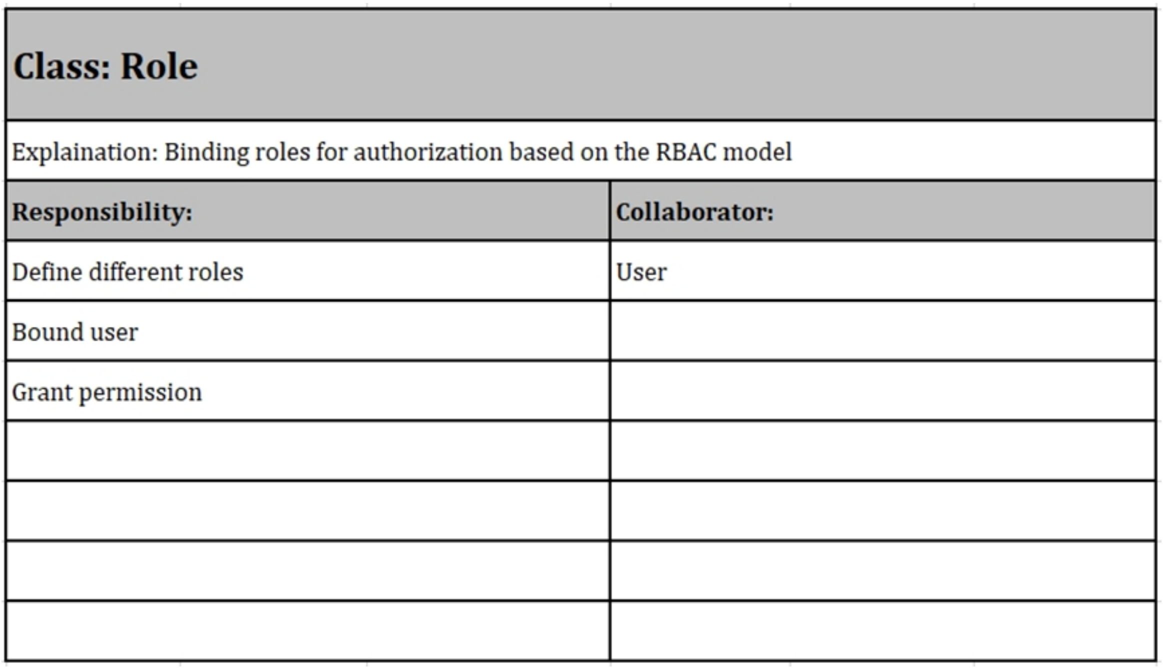
Class: Favorite	
Explaination: Every user has their own list of favorites	
Responsibility:	**Collaborator:**
Define a many-to-many relationship	User
Users bookmark a emoji	Emoji
Users unfavorite	

Class: Comment	
Explaination: Users can comment on emoticons and interact with each other	
Responsibility:	**Collaborator:**
An abstraction of the comments section	User
Users comment on a emoji	Emoji
Users reply to each other	

Class: Type	
Explaination: Every emoji belongs to a type	
Responsibility:	**Collaborator:**
Define the name and reference count of a type	Emoji
View all types	
Add a type	



Class: Message	
Explaination: Users will receive a message if someone interacts with them	
Responsibility:	**Collaborator:**
Encapsulate everything a message contains	User
Send a message	
Receive a message	
Read a message	
Change the status of a message	

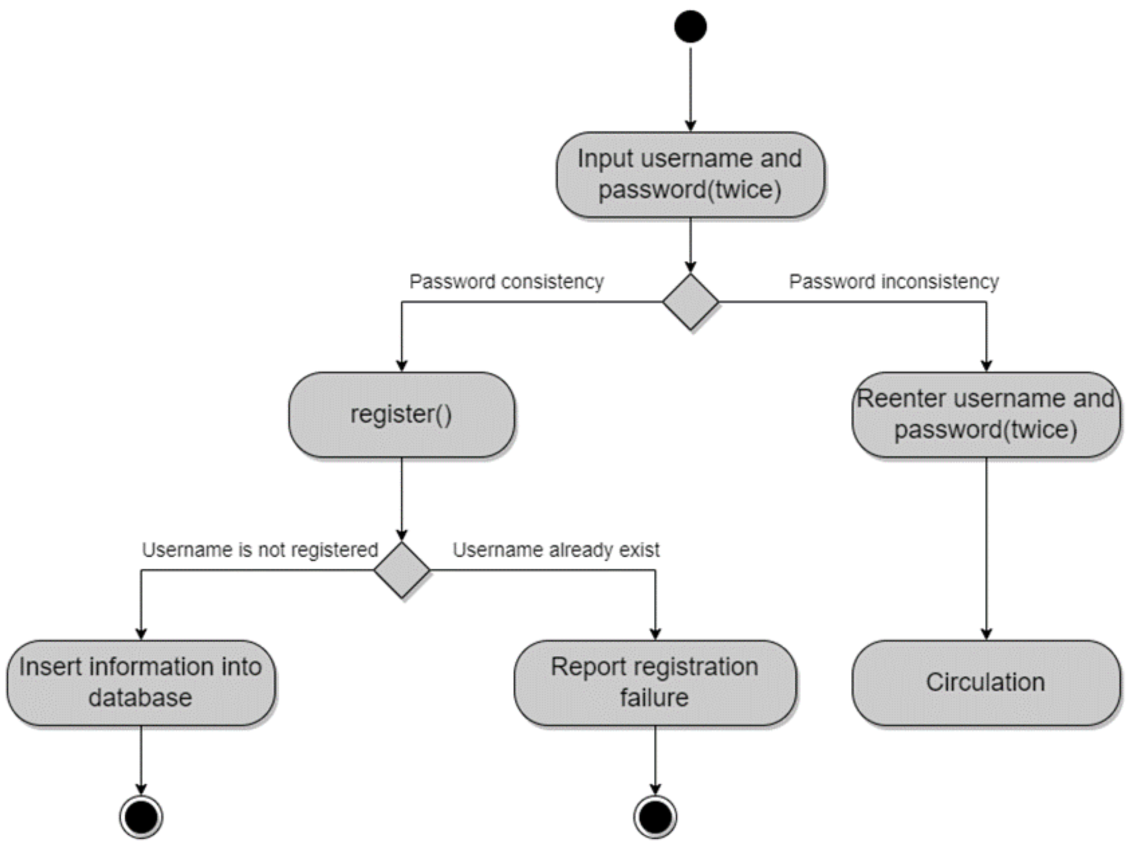


Class: Role	
Explaination: Binding roles for authorization based on the RBAC model	
Responsibility:	**Collaborator:**
Define different roles	User
Bound user	
Grant permission	

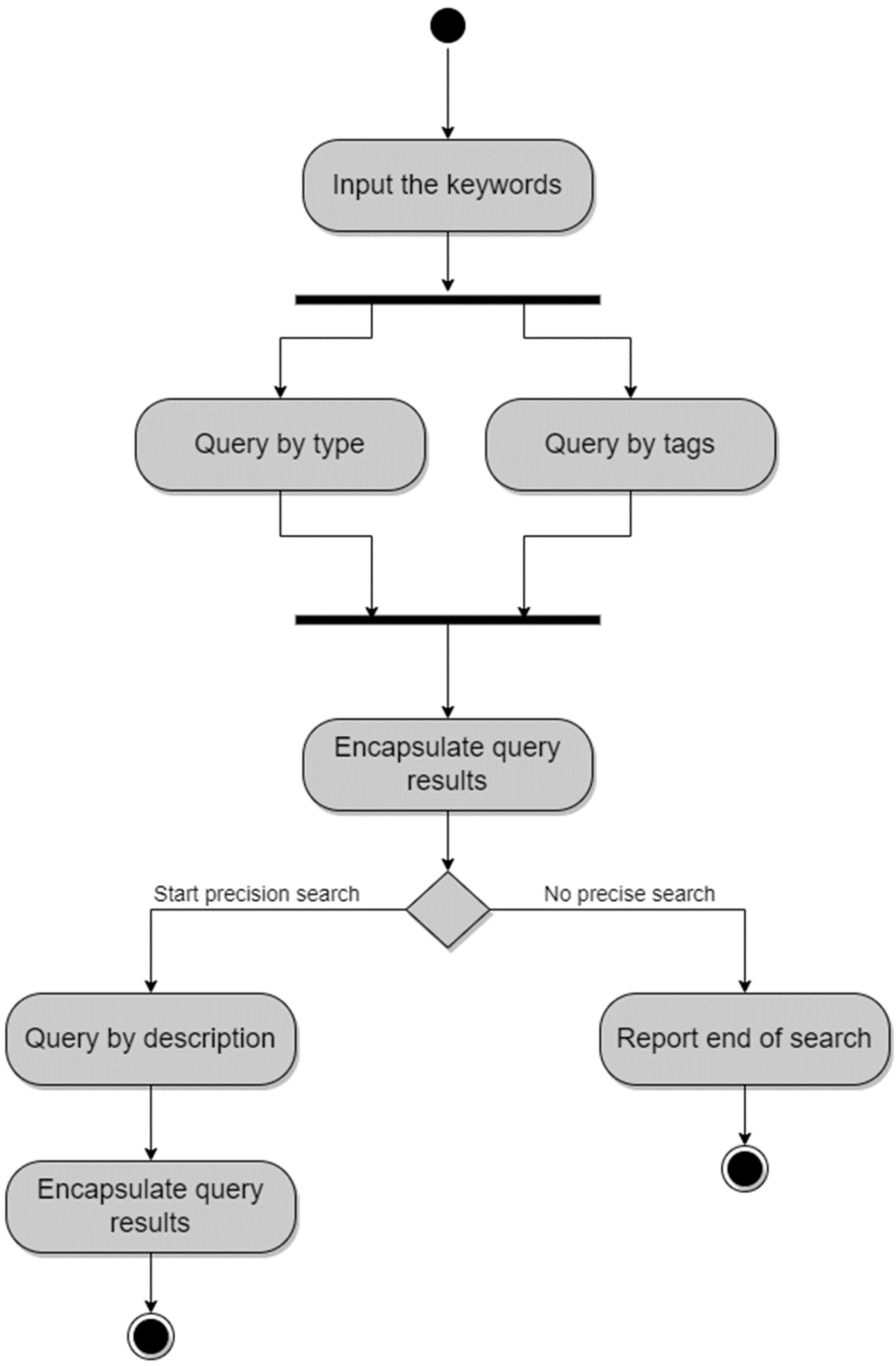
3.UML Diagrams

We utilize UML diagrams to describe the core requirements.

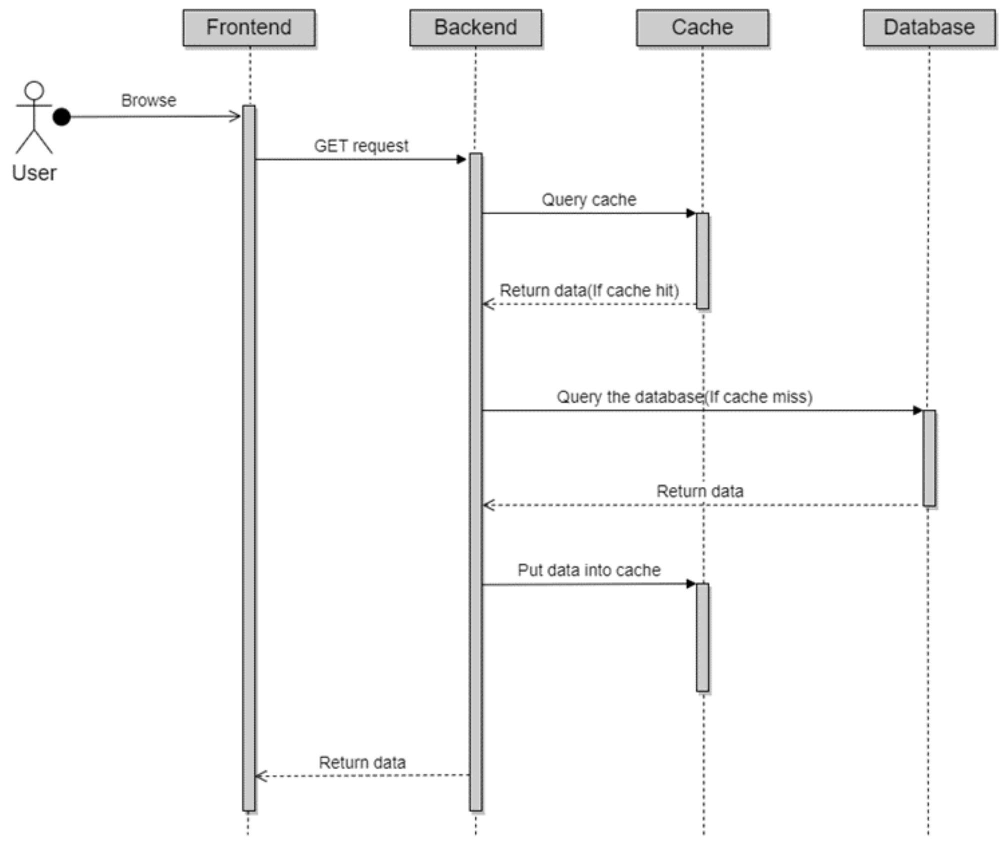
- Use UML activity diagrams to describe user register requirements:


Input username and password(twice)
Password consistency
Password inconsistency
register()
Reenter username and password(twice)
Username is not registered
Username already exist
Insert information into database
Report registration failure
Circulation

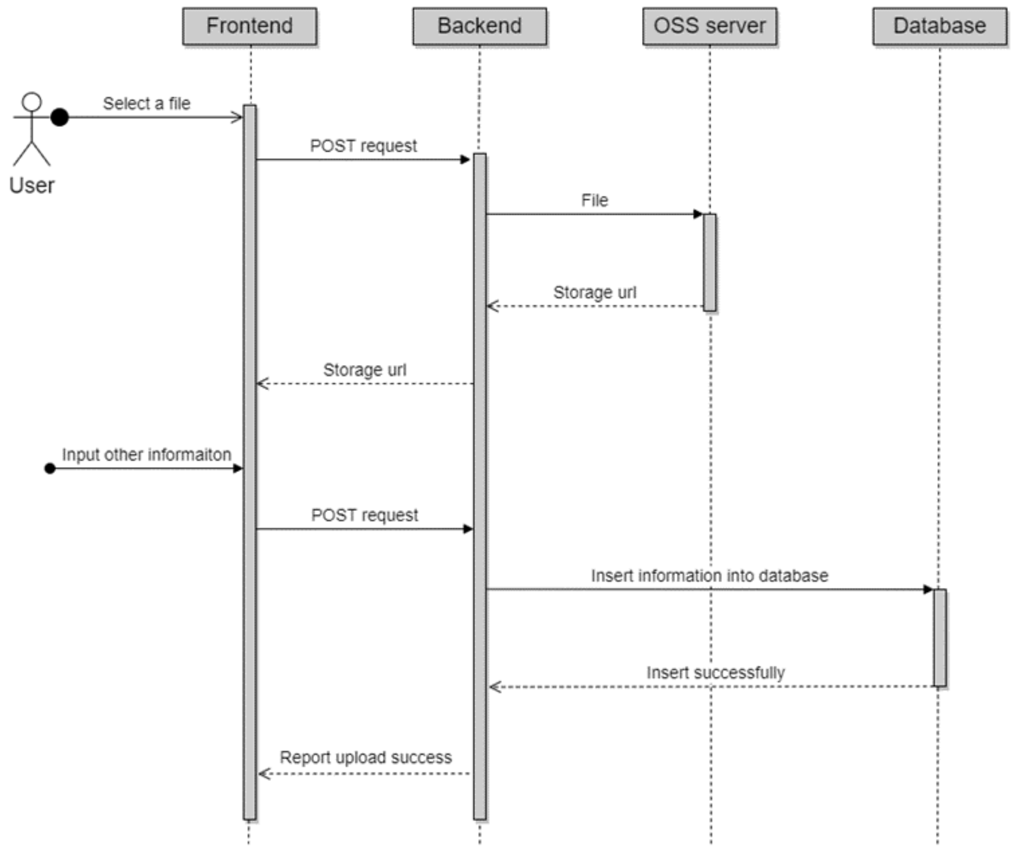
- Using UML activity diagrams to describe user requirements for searching for emojis:



- Using UML Sequence Diagrams to Describe User Requirements for Browsing Emojis:



- Use UML Sequence Diagrams to Describe User Requirements for Uploading Emojis:

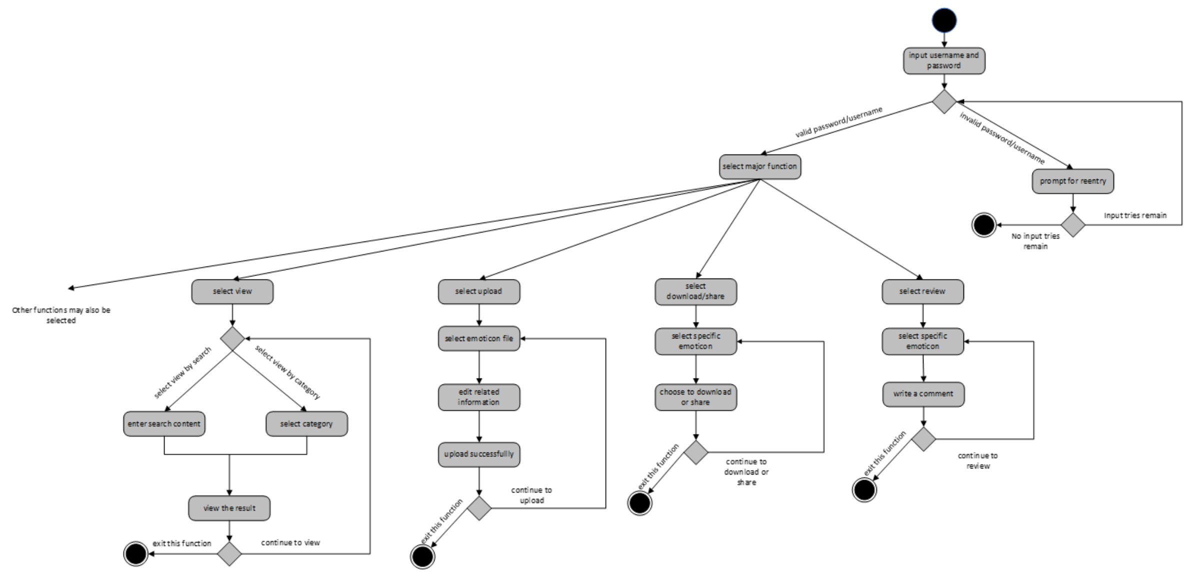


Chapter 3-Behavioral Modeling

In this part we focus on the functional aspects of our system and explain how it responds to events, user interactions, and system states.

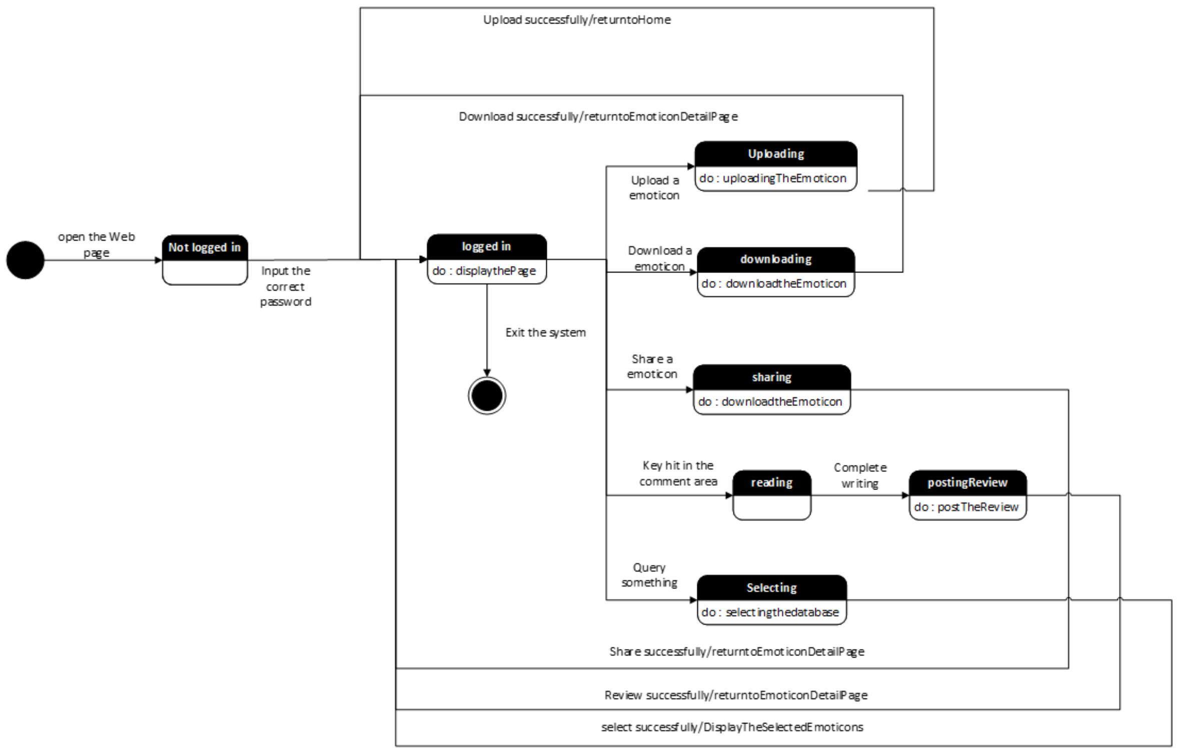
1. UML Activity Diagram

- We use UML activity diagram to help capture and communicate the dynamic behavior of our software system



2.UML State Diagram

- We use UML state diagram to document the state transitions and how they affect the system's behavior.


Upload successfully/returntoHome
Download successfully/returntoEmoticonDetailPage
open the Web page
Not logged in
Input the correct password
logged in
do : displaythePage
Exit the system
Upload a emoticon
Uploading
do : uploadingTheEmoticon
Download a emoticon
downloading
do : downloadtheEmoticon
Share a emoticon
sharing
do : downloadtheEmoticon
Key hit in the comment area
reading
Complete writing
postingReview
do : postTheReview
Query something
Selecting
do : selectingthedatabase
Share successfully/returntoEmoticonDetailPage
Review successfully/returntoEmoticonDetailPage
select successfully/DisplayTheSelectedEmoticons

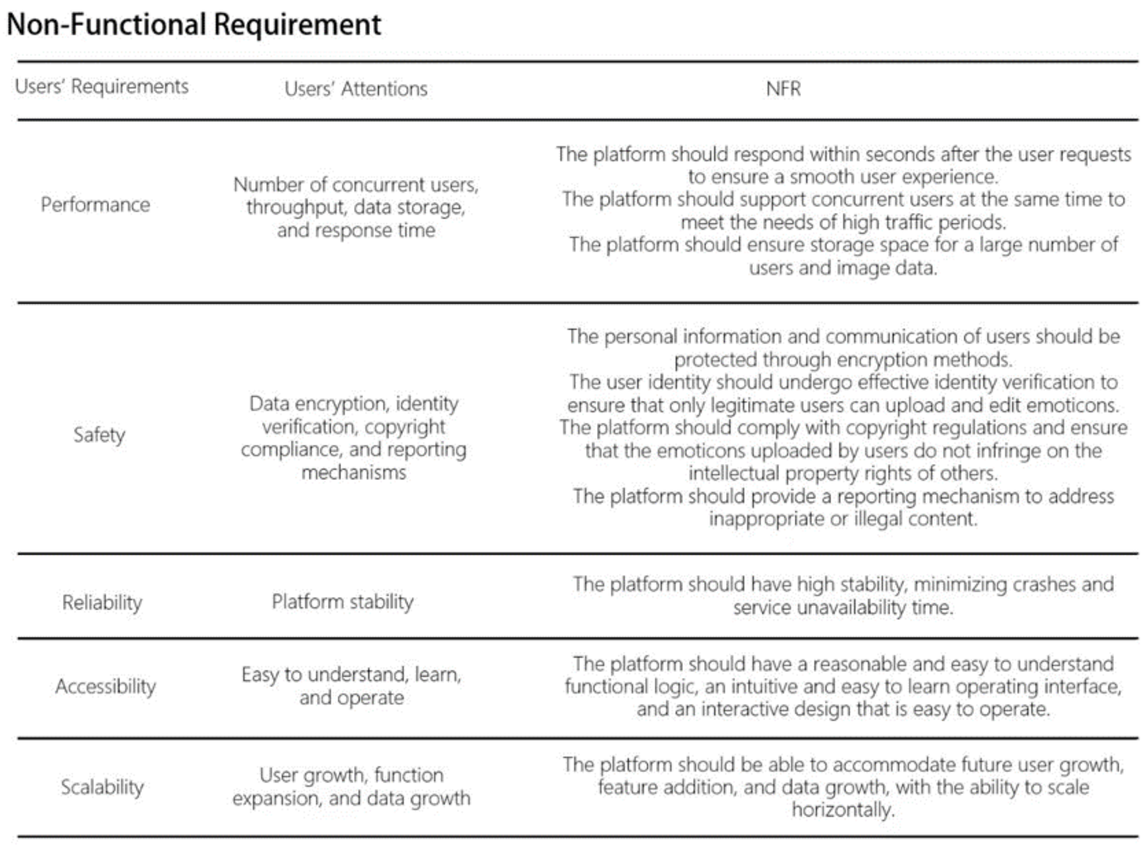
Chapter 4-Non-functional Requirements

1.Performance Requirements:

1. Response time: The platform should respond within seconds after the user requests to ensure a smooth user experience.
2. Number of concurrent users: The platform should support simultaneous concurrent users to meet the needs of high traffic periods.
3. Stability: The platform should have high stability to minimize crashes and service unavailability time.
4. User growth: The platform should be able to accommodate future user growth and have the ability to expand horizontally.

2.Security Requirements:

1. Data encryption: Users' personal information and communication should be protected through encryption methods.
2. Identity verification: The user's identity should undergo valid identity verification to ensure that only legitimate users can upload and edit emojis.
3. Copyright compliance: The platform should comply with copyright regulations and ensure that the emojis uploaded by users do not infringe on the intellectual property rights of others.
4. Reporting mechanism: Provide a reporting mechanism to address inappropriate or illegal content.



Non-Functional Requirement

Users' Requirements	Users' Attentions	NFR
Performance	Number of concurrent users, throughput, data storage, and response time	The platform should respond within seconds after the user requests to ensure a smooth user experience. The platform should support concurrent users at the same time to meet the needs of high traffic periods. The platform should ensure storage space for a large number of users and image data.
Safety	Data encryption, identity verification, copyright compliance, and reporting mechanisms	The personal information and communication of users should be protected through encryption methods. The user identity should undergo effective identity verification to ensure that only legitimate users can upload and edit emoticons. The platform should comply with copyright regulations and ensure that the emoticons uploaded by users do not infringe on the intellectual property rights of others. The platform should provide a reporting mechanism to address inappropriate or illegal content.
Reliability	Platform stability	The platform should have high stability, minimizing crashes and service unavailability time.
Accessibility	Easy to understand, learn, and operate	The platform should have a reasonable and easy to understand functional logic, an intuitive and easy to learn operating interface, and an interactive design that is easy to operate.
Scalability	User growth, function expansion, and data growth	The platform should be able to accommodate future user growth, feature addition, and data growth, with the ability to scale horizontally.