Product Requirement Document

Goal

Our project is to develop a chatbot application that can provide emotional support to users. The companionship will be illustrated in two properties of our chatbot: long-term memory and customization. Long-term memory indicates that the chatbot tends to remember users' history conversations and when a new conversation, the chatbot can act like recognizing the user. For the customization, we attempt to conclude user profiles from history conversations and add them to the prompt. The prompt then would be fed to the backbone model to generate more personal responses. Also, several preset chatbots are available to users and they can imitate the tone of particular virtual characters to provide spiritual pleasure.

Model

Long-term Memory

To implement the long-term memory capability of the chatbot, we plan to store history conversations with the user. Then when a new request comes, we will join the history conversations together with the current request to generate a prompt and feed the prompt to the model. For this backbone model, ChatGPT is our choice and we will call OpenAI api to get each response.
Besides, there are a few risks that we should pay attention to. As conversations will grow longer, it's essential to monitor the delay as well as the persistence of the response, which are all the evaluation indicators of our project.

Customization

User Profiles

To enhance the capability of the model to provide emotional support, we decide to import user profiles into the prompt and make the response more personal. It should be noted that user profiles are invisible to users because they only enhance the emotion perception of the model. For the implementation, two approaches are considered.

1. First, we would try to feed ChatGPT with history conversations and let ChatGPT generate user profiles directly.
2. If this approach doesn't work effectively, we may consider asking users to fill in a preference form when they register the application. Then user profiles will be concluded from this form.

Preset Virtual Characters

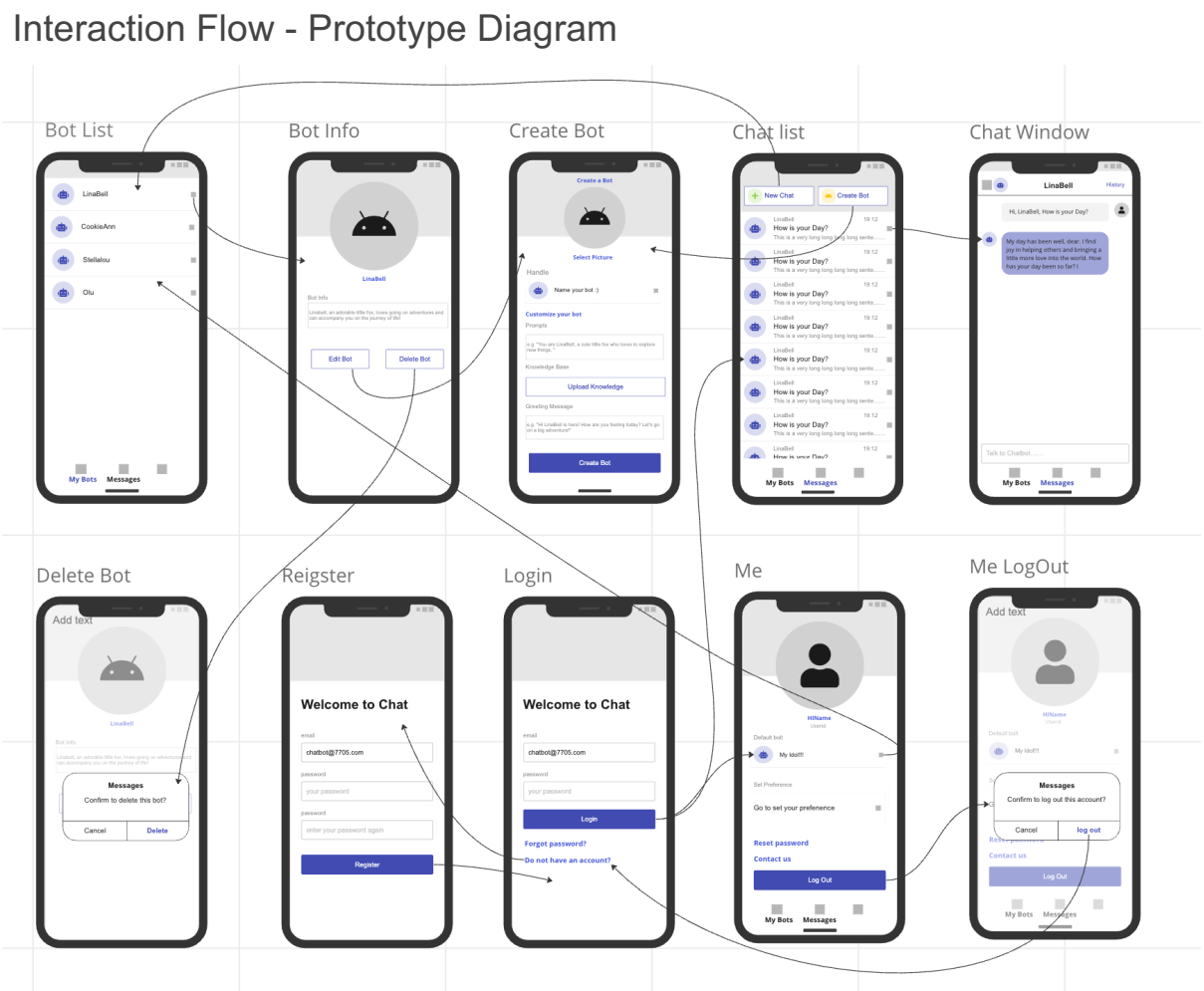
Apart from the personal chatbot above, imitation chatbots can also inspire spirits due to the influence of the virtual characters, such as Forrest Gump and Andy Dufresne. The selection of

virtual characters to imitate will be decided before implementation. And these available choices will be represented to users at the beginning. Then, two possible implementations for these imitation chatbots are discussed:

1. We can insert key words of the virtual characters into the prompt and feed the prompt to ChatGPT. Then the ChatGPT will act like the exact virtual character and generate the response to users.
2. If the above method isn't as efficient as expected. We may consider train a small-scale language model with the corpus of that virtual character and get responses from our trained model.

Client

Interface

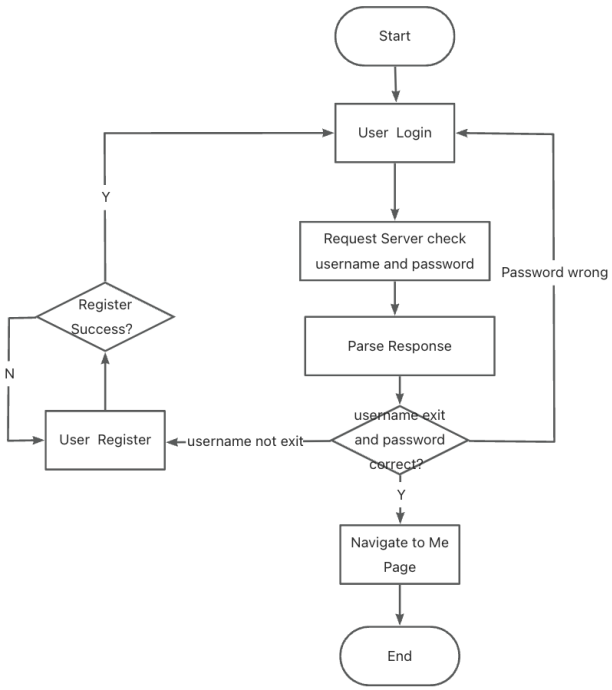


User Module

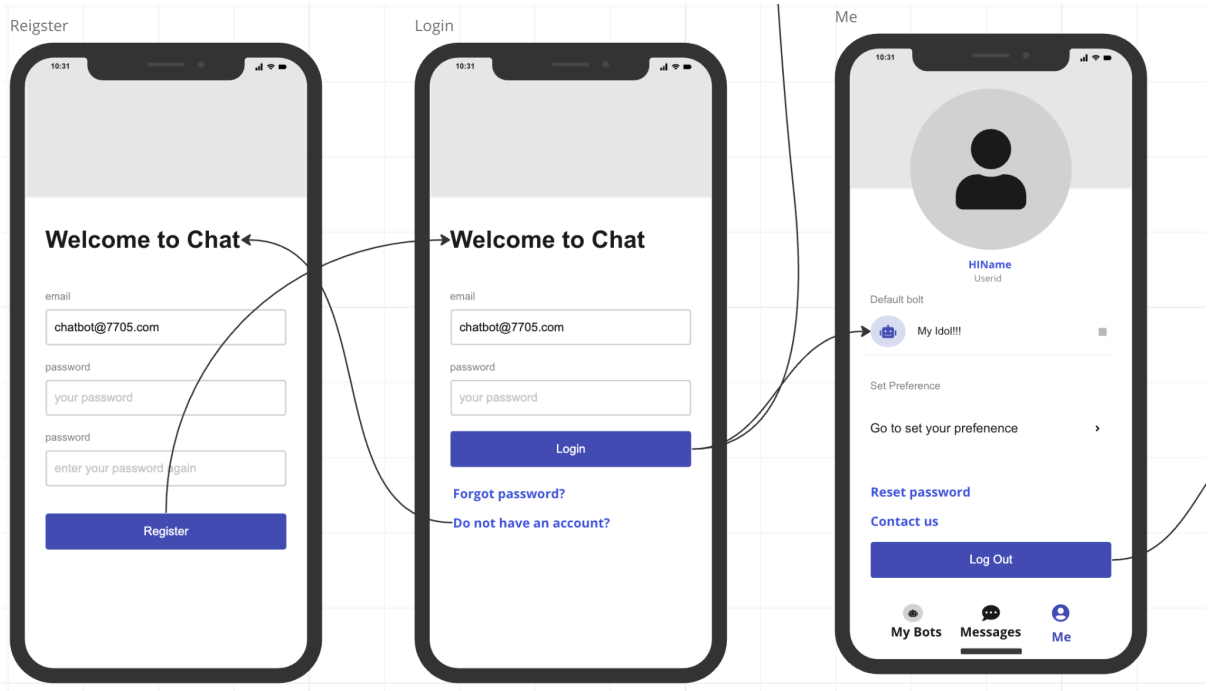
UserLogin and UserRegister

Description: User enters a username and password, and the frontend hashes the password before sending it to the backend server. The server checks if the username exists and if the password is correct. It then sends the response back to the frontend. If the username does not exist, it displays a message indicating that the username does not exist and prompts the user to proceed with the registration process. If the password is incorrect, it displays a message indicating that the password is incorrect and prompts the user to re-enter it. If the username exists and the password is correct, the page redirects to "me" Page.

Parameters: userId, username, password



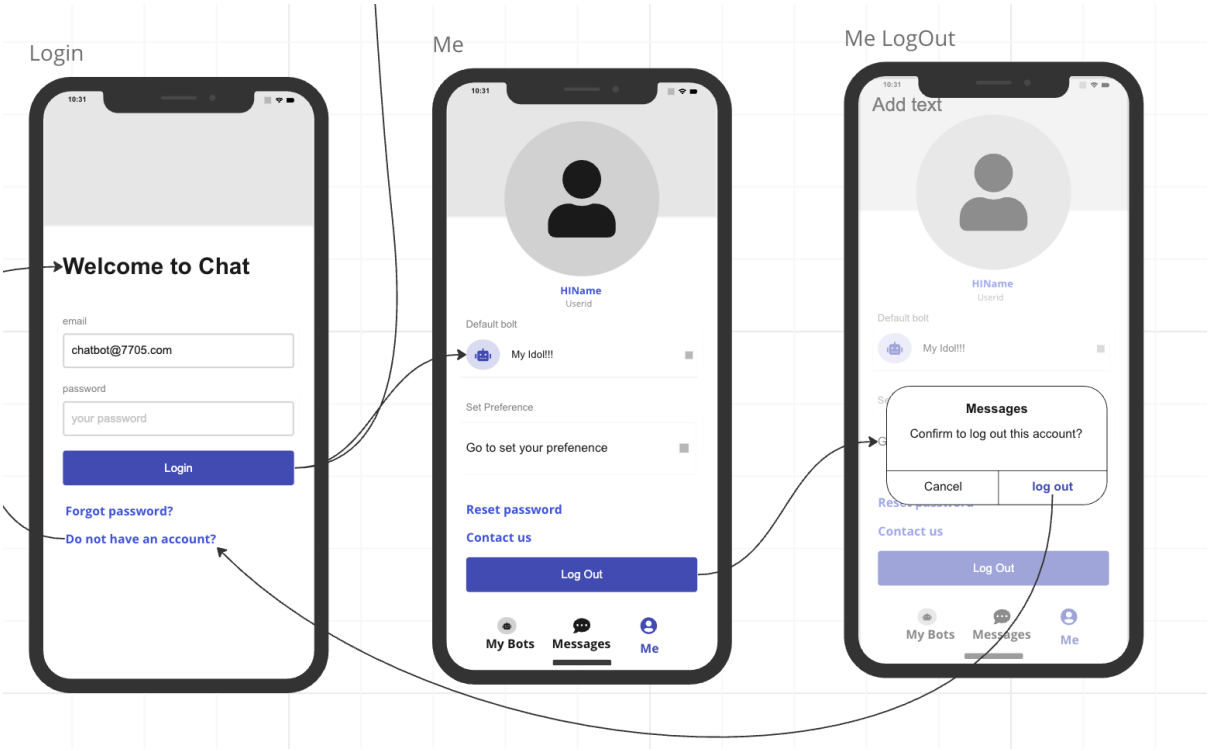
Img. Flow Chart of UserLogin and User Register



Img. Prototype of UserLogin and User Register

UserLogout

Description: When the user is on the "me" page and clicks on "logout," a confirmation dialog box appears asking if they are sure they want to logout. If they confirm the logout, they will be redirected back to the login page. Otherwise, they will remain on the "me" page.

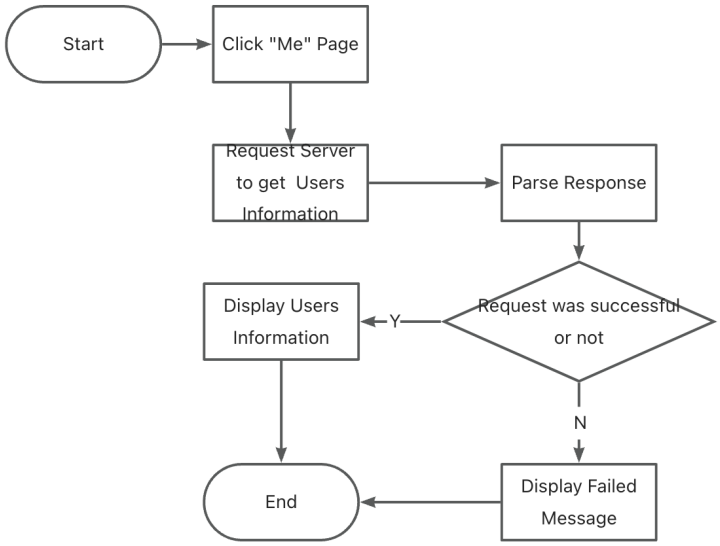


Img. Prototype of UserLogOut

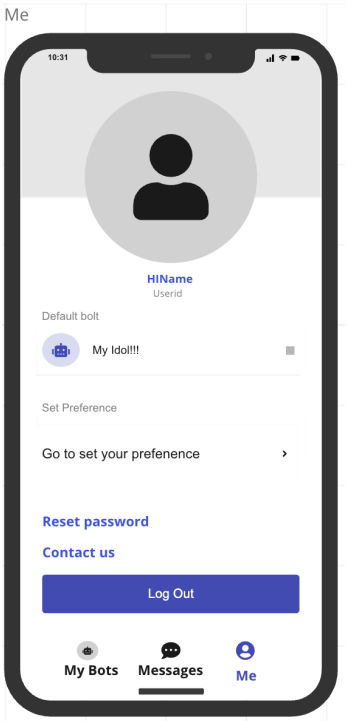
UserInformation

Description: The user clicks on the "Me" tab, the frontend sends a request to the backend for the user's userID. The backend responds by providing the user's preferences and preconfigured information, such as the default robot. If the request is successful, the information is displayed on the "Me" page.

Parameters: userId, default robot, preferenceId



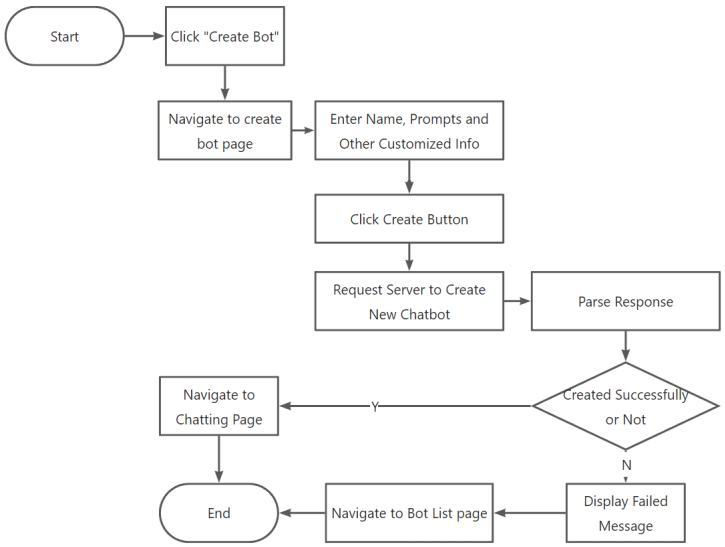
Img. Flow Chart of UserInformation



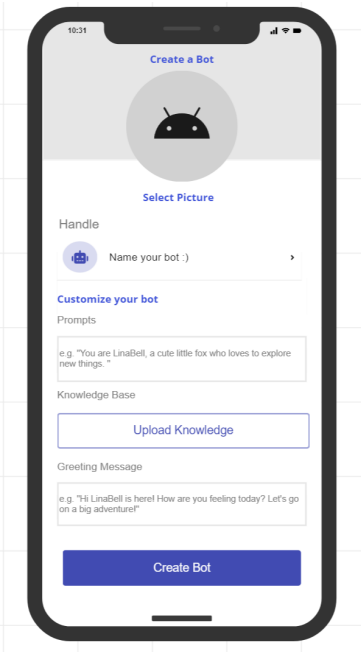
Img. Prototype of UserInformation

Bot Customized Module (5.4 P0)

Bot Creation



Img. Flow Chart of Creating Bot

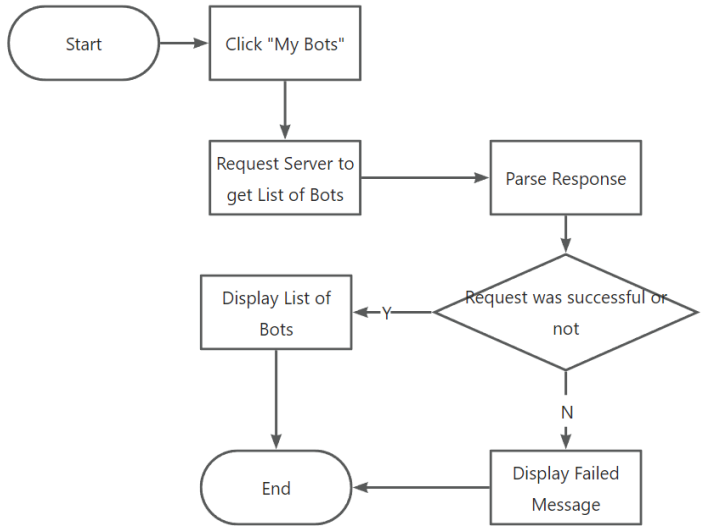


Img. Prototype of Creating Bot Page

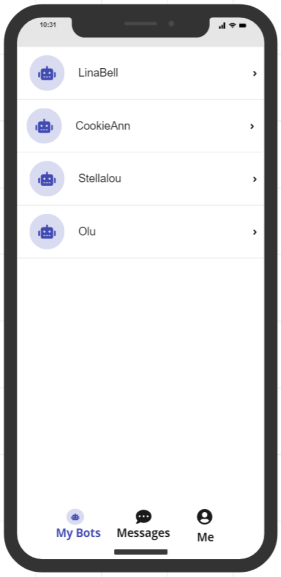
Description: To create a bot, the user should click on the 'Create Bot' button, which will take them to the 'Create Bot' page. On this page, the user should enter the bot name, prompts, and any other necessary customized information. Once the input is complete, the user should click on the 'Create' button. This will send a request to the backend to create the bot. After receiving a response from the backend, the client side will analyze the return value to determine if the bot was created successfully. If the bot was created successfully, the user will be navigated to a chat window with the bot. If the bot was not created successfully, the user will be prompted with an error message.

Parameters: userId, botName, prompts, knowledge, greetingMessage

Bot List



Img. Flow Chart of Bot List

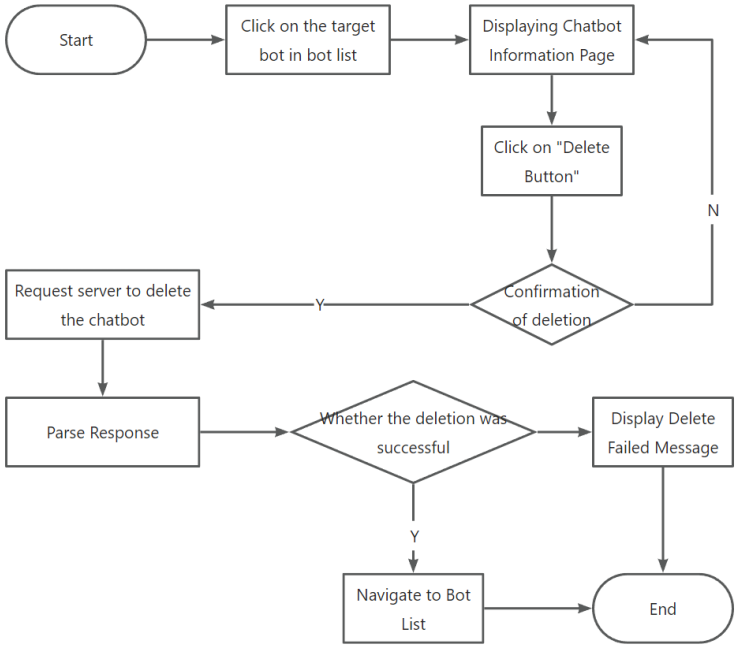


Img. Prototype of Bot List

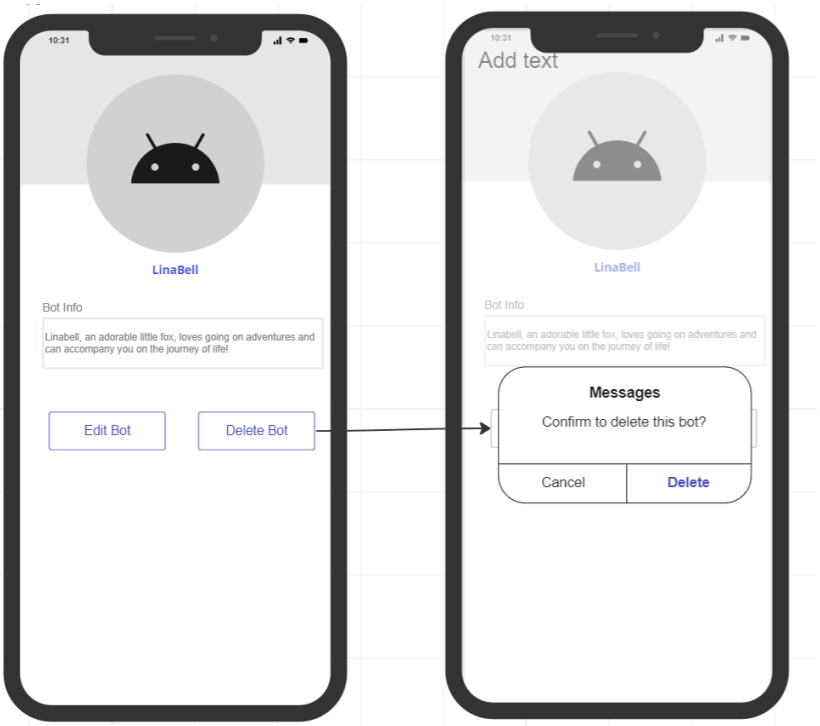
Description: By clicking ‘My Bots’ tab at the bottom of the screen, the client sends a request to the backend to retrieve the corresponding list of bots. Once the response is received, the client checks it to confirm the successful fetch of the robot list. If unsuccessful, the user is prompted with a message indicating the failure to get the bot list. If successful, the robot list is displayed.

Parameter: userId

Bot Delete



Img. Flow Chart of Deleting Bot



Img. Prototype of Deleting Bot

Description: To access the details page of a robot, simply click on the target bot in the bot list. From there, users can delete the robot by clicking the delete button located at the bottom of the page. Once users click the delete button, a pop-up window will appear to confirm the deletion. If users click the confirm button, the client will send a deletion request to the backend. When the client receives the response from the backend, it will verify the response code to check whether the bot has been successfully deleted. After successfully deleting the bot, the bot list will be displayed. If the robot deleted failed, an error message will appear indicating

the failure. If the user clicks 'Cancel' on the pop-up window, they will be navigated back to the Bot Details page.

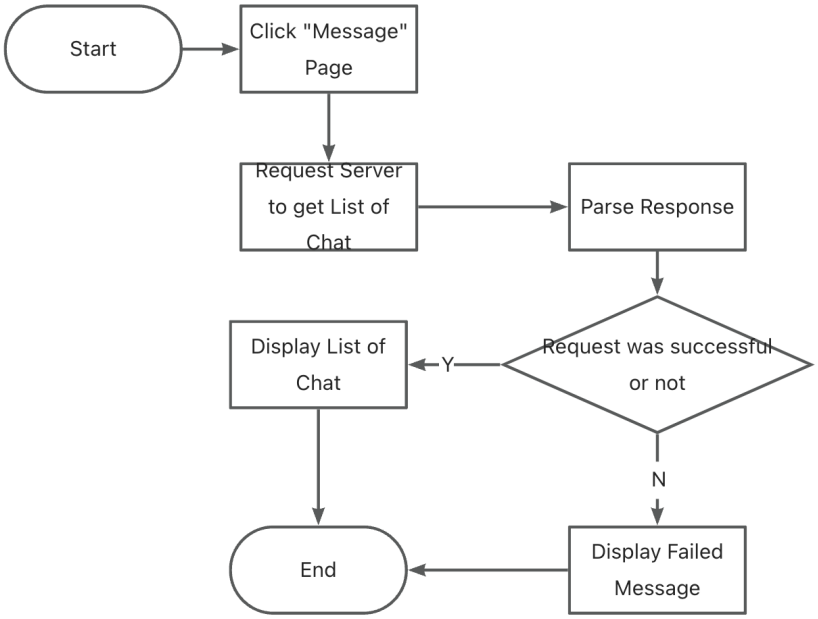
Parameter: botId

Chat Module

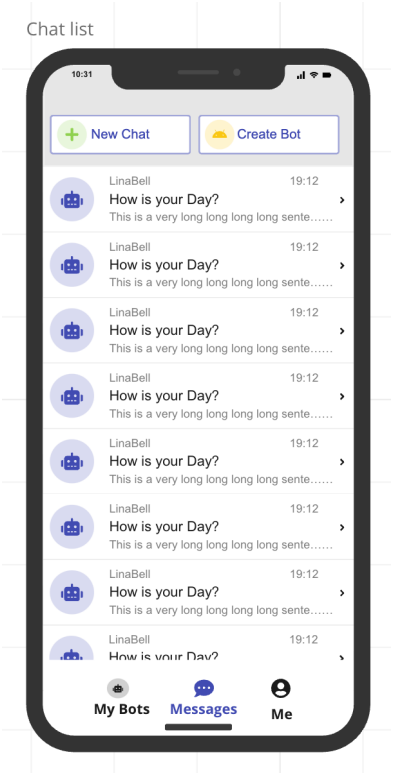
Chat List

Description: By clicking the 'Messages' tab located at the bottom of the screen, the client initiates a request to the backend in order to retrieve the associated list of chats. After receiving the response, the client verifies its contents to ensure the successful retrieval of the chat list. If the attempt is successful, the chat list is displayed on the screen.

Parameter: userId,Chat list correspond to useId



Img. Flow Chart of Chat List

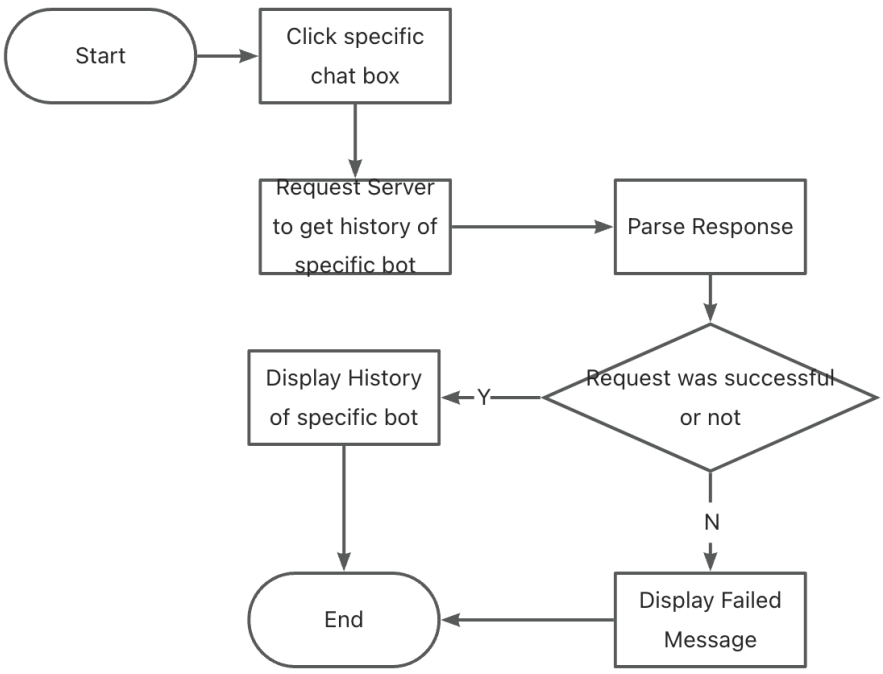


Img. Prototype of Chat List

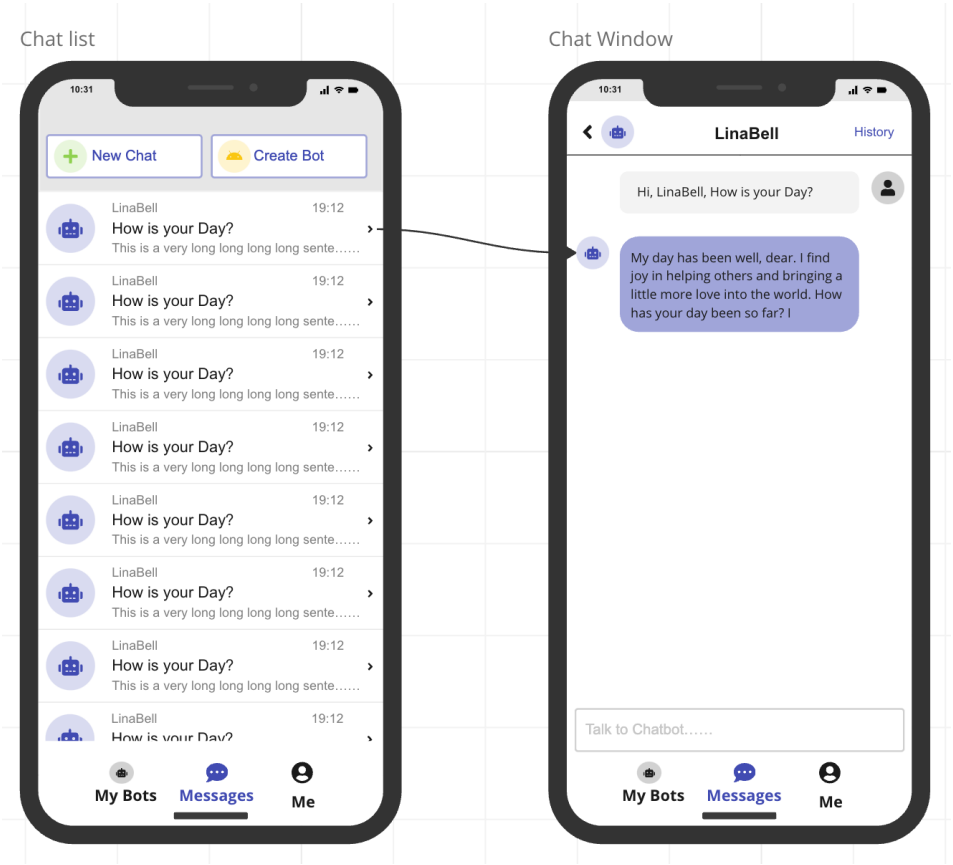
Chat Window

Description: By clicking on a specific chat box, the client navigates to the chat window interface and sends a request to the server to retrieve the corresponding conversation history. Upon receiving the response, the client displays the retrieved chat records on the interface for the user to view.

Parameter: userId,ChatId list, Chat history



Img. Flow Chart of Chat Window



Img. Prototype of Chat Window

Service

User Module

Login

- Description: Look up for the username input in the database. If the user already exists, compare the hash value of the password. If the password is correct, generate a corresponding token. In addition, can check the user's token, if matches, skip the input operation.
- Return: generated token; login result (1=succeed; -1=fail)

Registration

- Description: Look up for the username input in the database. If the user does not exist, create the account and add a new entry in the database.
- Return: registration result (1=succeed; 0=user already exist; -1=other failure)

Logout

- Description: Cut off the connection between the client and server.
- Return: none

Bot Module

Bot Creation

- Description: Create an instance of chatbot and load the corresponding model or prompts. Add an entry to the bot list.
- Return: result (1=succeed; -1=fail)

Bot Delete

- Description: Delete an instance of chatbot and clear corresponding conversation history. Delete the related entry in the bot list.
- Return: result (1=succeed; 0=bot do not exist; -1=other failure)

Chat Module

Conversation Processing

- Description: Send the text from the user to Chatgpt's API or pre-trained model and get the reply.
- Return: reply (text); result (1=success; 0=timeout;-1=connection error)

Database

key-value:
Use the usernames as keys, and corresponding values are users' basic information.
For the conversation history, there are two plans: Plan A is to use a hashmap (dictionary) for each value and the value consists of user's password, conversation history and other potential information; Plan B is to create new keys for each user's conversation history, and the key names may be the username with a tail like '_conv_history'. Other information can also be recorded in this way.