Group 192 iKunnect CSC207 Project Blueprint

Hannah Liu, Tylor Meng, Fangshi Du, Justin Liu, Yuanyuan Mu

Messaging App

- We aim to develop a chat app similar to Snapchat, supporting voice messages, text messages, and files.
- It will feature group chat creation and private messaging.
- To complement our front-end application, we will also develop our own back-end system. The system will include a database and will register for HTTP API calls so that it can respond to front-end requests.
- **In addition, we will integrate an existing translation API (DeepL) to translate messages across different languages, too.**

•User Stories 1

Send Text/Image/File/Voice Messages

- As an individual user, Bob wants to send multi-type messages to his friends, so that he can maintain secure and convenient communication without registering on major chat platforms (e.g., Instagram) which is often a ram killer and cause distraction.
- Acceptance Criteria:

1. After the local client presses the send button, it determines the type of message, organizes the send time, sender, sender's account, and content (non-text types are converted to bytecode and encoded) into a JSON, and then calls the corresponding message type API.

2. A thread is created to await a backend response when requesting the API. Once a response from the backend is received, it invokes propertyChanges to update the View and terminate the thread. The UI layer will mark the message as sent.

3. Upon receiving the JSON, the backend first verifies the legitimacy of the user, then determines the type of message, and creates a corresponding Message object to store the information. After classification, the Message object is stored in the respective Chat's chat_history[Message].

•User Stories 2

Chat History Synchronization

- As an individual user, Bob wants to synchronize historical chat records from the database, So that he won't lose precious memories with friends when he switches devices.
- Acceptance Criteria:

  1. Every chat record cached in the client has a corresponding ID. When the "Synchronize Messages" button is pressed, we will organize the local message IDs into a list, embed them in a JSON along with our user information, and send a request to the backend via HTTP GET.

  2. Upon receiving the request, the backend, through the DataAccess Object, queries the database and compares the list of chat record IDs in the database with the received list. It then compiles the differing records into a JSON file and sends it back to the client.

  3. After decoding the chat records received from the server, the client categorizes them by chat record ID. They are then stored in the local cache using the FileChatHistoryDataAccessObject. Once caching is complete, the UI layer refreshes the chat record display based on the local cache.

•User Stories 3

Translation with API

- As an individual user, Bob wants to translate chat messages in his chat system using the DeepL translation API, so that he can understand and communicate with friends who speak different languages.
- Acceptance Criteria:

  1. Chat messages are translated by clicking a "Translate" button next to a message, sending an API request to DeepL with the message text and the target language for translation.

  2. The chat system displays the translated message below the original, clearly indicating that it's a translation.

  3. Translated messages are retained and included in the chat history for synchronization with the user's other devices. When synchronizing, the chat system organizes both the original and translated versions into a JSON file, associating them with translation IDs for storage and retrieval.

•Task distribution

- **Hannah Liu (Code Quality Supervisor & Tester):**
  - Responsible for ensuring that all member submissions adhere to CA (Clean Architecture) and SOLID principles. Oversees members in correcting code that doesn't meet the standards. In charge of testing client/server programs to ensure there are no major bugs. Manages the rhythm of presentations.
- **Tylor Meng (Server-side Developer):**
  - Responsible for writing server-side code and ensuring each API aligns with our designed features. Handles certain challenges encountered during client/server development.
- **Fangshi Du (Client-side Business Logic Developer & Communicator):**
  - Responsible for defining and refining functionalities. In charge of writing the Application Business Rule section for the client-side. Coordinates with the backend, troubleshoots communication issues between client and backend, and collaborates with team members to determine the structure of data transmission.
- **Justin Liu (Client-side UI Designer & Developer):**
  - Responsible for designing and coding the client-side UI. Collaborates with Hannah and Fangshi to ensure the client UI displays correctly.
- **Yuanyuan Mu (Utility Developer):**
  - Responsible for writing utility classes and implementing functions needed by other parts of the development process.

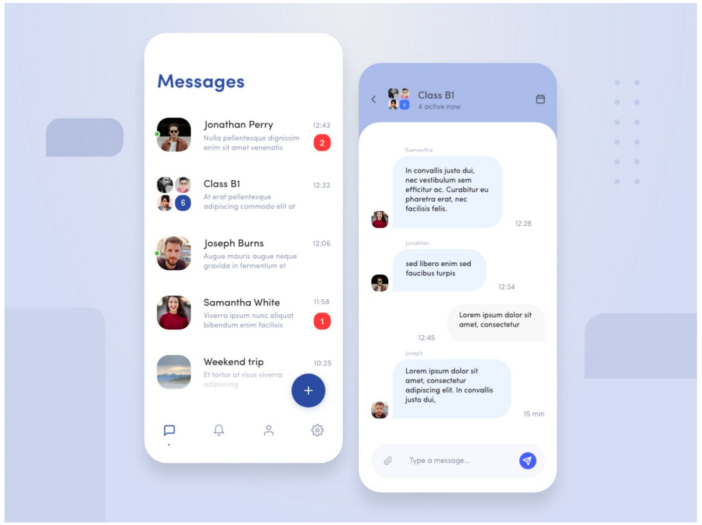
Specification

- The messaging app enables users to send text messages, images, video messages, and files. It also features integrated translation using API. Users can create individual or group conversations and receive notifications for new messages and translation suggestions.

1. **User**: The program should support multiple users, each with a unique identity.
2. Chat Room: Users should be able to create, join, leave, and manage chat rooms.
3. Message: Users can send, receive, edit, and delete messages within chat rooms. Messages can include text, images, files, and videos.
4. **Translation**: The program integrates with a translation API to provide translation suggestions.

1. **Create**: Users can create new chat rooms and messages.
2. Join: Users can join chat rooms to participate in conversations.
3. Leave: Users can leave chat rooms when they no longer wish to participate.
4. Send: Users can send messages to chat rooms or other users.
5. **Receive**: Users receive messages in real-time from chat rooms and other users.
6. Delete: Users have the option to delete their own messages.

Proposed Entities



Messages

Jonathan Perry 12:43
Nulla pellentesque dignissim enim sit amet venenatis
2

Class B1 12:32
At erat pellentesque adipiscing commodo elit at

Joseph Burns 12:06
Augue mauris augue neque gravida in fermentum et

Samantha White 11:58
Viverra ipsum nunc aliquot bibendum enim facilisis
1

Weekend trip 10:25
Et tortor at risus viverra adipiscing

Class B1
4 active now

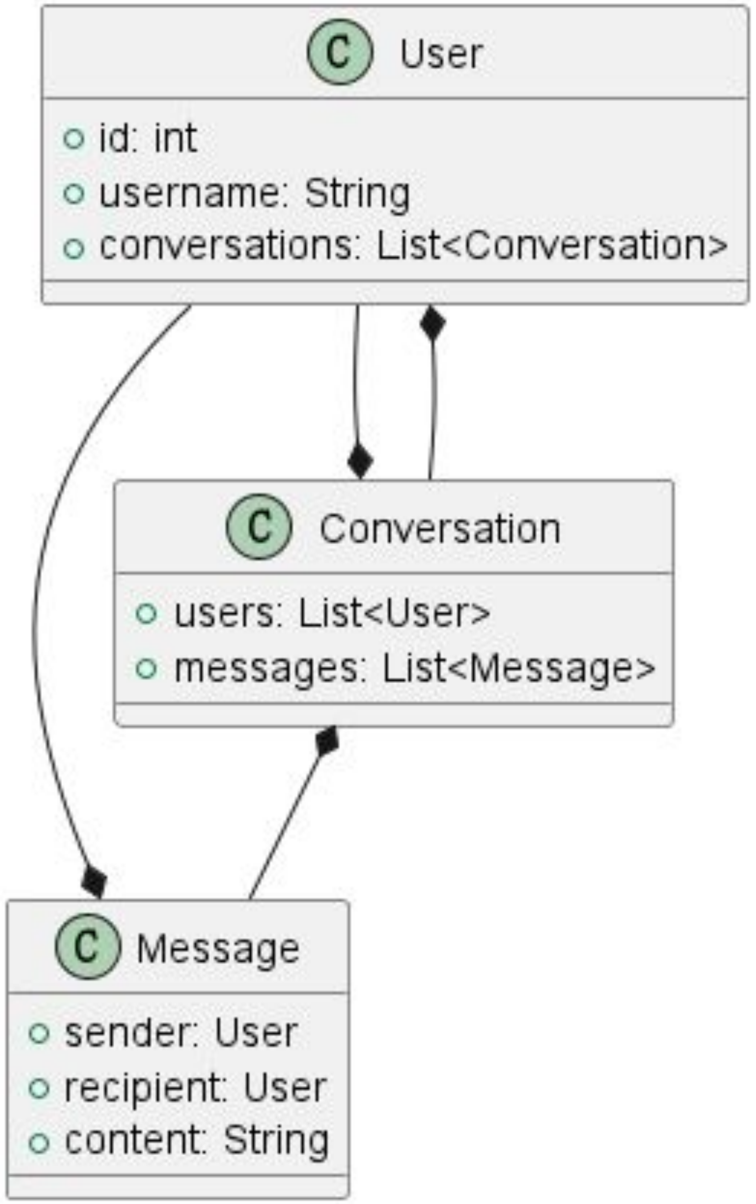
Samantha
In convallis justo dui, nec vestibulum sem efficitur ac. Curabitur eu pharetra erat, nec facilisis felis.
12:28

Jonahtan
sed libero enim sed faucibus turpis
12:34

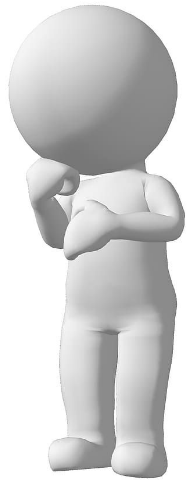
Lorem ipsum dolor sit amet, consectetur
12:45

Joseph
Lorem ipsum dolor sit amet, consectetur adipiscing elit. In convallis justo dui,
15 min

Type a message...



Beyond the blueprint!



- Some APIs we plan to use:
  - Translation API(From internet)
  - Self-designed Chat Backend(For store message, send and receive message)
- Some questions:
  - How to preserve clean architecture if we plan to use server/client/common structure?

    We will check recent commits at the end of every week to make sure that we are on the right track. When facing a CA validation, we will open up a zoom meeting to discuss and solve.

  - How to deploy our Java backend project on the server?

    Due to their prior interests and experiences, Fangshi and Tylor both possess expertise in deploying projects on servers to ensure reliable service. They also happen to have available domain names and servers. Therefore, they are confident in deploying the project on the server after debugging and testing, and integrating it with CloudFlare DNS resolution.

Thank you!