Personal Sprint Review

In this sprint, we aimed focused on integration of frontend and backend. The goal of the sprint was to resolve synchronization issues while communicating with the backend. We also aimed to add multimedia features in conversations and live translation in this sprint.

At the end we could build a chat application which is accessible from any browser including desktop computers and mobile devices. The application can perform following features.

1. Register and sign-in
2. Find out the online users
3. Create groups, add users or group into it, add admin to it
4. Conversation with individual user
5. Conversation with a group
6. threads inside conversation
7. Audio video and gif in conversations and threads
8. Emoji
9. Live translation in 13 languages
10. Profile pictures
11. Broadcasting to all conversations
12. Private groups and users and search for them

In this sprint I focused on defining/developing and optimizing the frontend architecture to create a seamless experience for the end user. I also focused on developing frontend components to handle communication, threads, emojis and multimedia. On backend I worked on developing endpoints for multimedia communication, online users and resolve few bugs. I worked with other team members to figure out how to incorporate google translation feature to support live translation and how to mock the database for testing.

Overall I was involved in architectural designing, frontend development, few backend api development and integration of frontend and the backend.

Following are the detailed issues.

1. Enable multimedia communication and profile change [MSD208SP19-100, MSD208SP19-68]
    - ✦ We are using AWS S3 service to store multimedia content. To support multimedia we are uploading the file to our S# bucket from the frontend. After uploading this, the public url of that file is sent to the Prattle server which records it into the database. Frontend is responsible for deciding the type of multimedia message (Audio, Video, Gif).
    - ✦ I developed a react component to handle multimedia uploading and returning a public url. This component can be integrated in any page/component just by writing a single line. Thus, the same code is used for handling image, video and gif uploading from user-user conversation, group conversations and threads to send multimedia messages. This same component is also used for uploading profile picture.

2. Enable emoji
    - ✦ Since emoji is highly dependent on the UI and user ethnicity, we have developed this component in such a way that image of any emoticon is processed on the frontend. Backend stores an associated tag. For a example, if we send :-(or :sad: to the backend, it will store it as it is. However, when we are showing that on front end

such tag would be changed to an emoji (🙁 in this case). This kind of approach will allow us to provide user specific emoji sets in future.

3. Conversations [MSD208SP19-96, MSD208SP19-98, MSD208SP19-99]
    - ✦ DB architecture is very crucial for the keeping conversations very simple. DB is designed in such a way that conversation is an independent entity which consists of threads. Thread is an independent entity which consists messages. Each message has a sender associated with it. Because of this hierarchy, we can easily identify all threads inside a conversations and all messages inside threads, thus all messages inside conversations as well.
    - ✦ This same conversation entity is linked with either user-user conversation relation or with a group entity.
    - ✦ The above simplicity makes sure that same frontend component can be used to handle both group conversations and user-user conversation.
    - ✦ Till last sprint, we were using pull mechanism to fetch new messages in a conversation. Saying that, we were sending a request to fetch new messages every three seconds. This approach was causing a leg and a lot of synchronization issues.
    - ✦ I worked on solving the above mentioned issues.
        - ➡ Solution: send a notification message from backend to frontend whenever a new message is received for that particular user. On frontend, whenever this push notification is received, show it as a browser notification and fetch messages for that particular conversation id.
        - ➡ This resolved 90% synchronization issues and reduced unnecessary api calls.

4. Integration of backend and frontend [MSD208SP19-86, MSD208SP19-97, MSD208SP19-92]
    - ✦ **Why we need frontend?**
        - ➡ As a team we believed that if there is no frontend or user interface, the product is not finished. No client would be happy to use CLI to see a demo of the product. More than that, CLI can not give a good experience to showcase all the functionalities. Keeping this in mind, since the very first we developed the backend in such a way that we want to deliver a react frontend application at the end.
    - ✦ **Why it was a very difficult job?**
        - ➡ Prattle works on tcp tunnels. For each client we have a thread to communicate over the network using TCP connection.
        - ➡ Browsers does not support communication over TCP protocol.
        - ➡ Prattle doesn't support asynchronous calls within a thread.
        - ➡ Any java architecture to handle REST requests was undesirable to use.
    - ✦ **What I did to overcome the above mentioned limitations on the backend.**
        - ➡ I specifically worked on designing and developing an architecture to support API calls. In backend I added a MessageType called NOTIFICATION and API to support push messages and API requests respectively.
        - ➡ From the ClientRunnable.java API message is redirected to Route class where we fetch the endpoint URL and method type from the message. Below is an example message for obtaining messages in a conversation.

            API 10 srsanghavi 53 getMessagesInConversation/::GET::{conversation_id:12}
        - ➡ In Route class, we redirect the request to relevant Controller using Factory method and from there to a relevant model using ModelFactory. Only models can talk to the database.
    - ✦ **What I did on the frontend?**
        - ➡ Host a proxy server, which creates a tcp link between client browser and prattle sever. Send all messages using this link.

- ➡ For this, I developed a frontend gateway library which will dispatch only one api request at a time and will wait for the reply until a response it received. To handle deadlock or livelock situation, any call will timeout after 10 seconds. To handle this cascaded api calls, I introduced a sem lock which would be locked when an API call is dispatched and unlocked when response is received. Any new API call has to get hold of this lock if it goes out.
- ➡ Thus my gateway library will dispatch API calls in queue fashion rather than asynchronous calls.
- ➡ Push notifications are sent from the server without the client involvement and whenever client receives any push notification, it executes the associated method.

5. Online users [MSD208SP19-105]
    - ✦ Iterate over active threads to obtain a list of online users. Send it to as a parameter of user information to the client in a JSON object when getUser/ or getUsers/ is called.

Apart from this, we met once in two days and almost communicated over messenger or phone to resolve each others hurdles. I worked with John to make a stable frontend. I also worked with Himanshu and Ram to resolve any doubts regarding the backend architecture. John has been a very good help when it comes to organize UI elements on the frontend.

Name	Score	Comments
Himanshu Budhia (Scrum Master)	8	This have been the most fruitful sprint. As a scrum master Himanshu could motivate all the team members to deliver a complete system. He made sure that team is making progress every single day. Apart from his scrum master role, Himanshu developed the live translation feature which can support 13 languages.
John Goodcre	8	John did a fantastic job in developing frontend. His skills in developing appealing UI elements is noteworthy. To make the user experience seamless, John invested a lot more than desired working hours this sprint. He could finish all the tasks assigned to him and he also managed to make few more components to show on the frontend.
Ram Prakash	8	Ram was a crucial team member for optimizing the code. He managed to test a lot of methods with dependencies by learning and using Mockito. Ram also worked on rewriting tests in a better way. Apart from that he handled S3 integration on the backend. He also managed was ec2 infrastructure for the project.
Shashwat Sanghavi	8	I mainly worked on rewriting the frontend to resolve synchronization issues. My main job for this sprint was to integrate the frontend and the backend. However, after finishing that, I worked on developing conversations, threads and group conversations on the frontend, supporting multimedia in both frontend and the backend. I enjoyed helping out other team members with react related queries and doubts regarding web architecture on the backend.

Overall, everyone invested equal efforts to deliver the promised target.