Implementation Plan

Jitsi Challenge and Response Feature

Released on 19.11.2021

Doganbros

Ümit Doğan
Hacımimi Mh.
Kapıkulu Sk. No: 5A
Istanbul Turkey
+90 532 286 56 64
umit@doganbros.com

Table of Contents

1. Summary

When a moderator presses the chat button, the ChatInitialized message will be broadcasted to all the lobby room participants. This will ensure that other moderators will not have the ability to chat with that attendee. When the attendee receives this event, the chattingWith key will be set and the chat widget shown. Now both parties can chat using sendPrivateMessage method in the conference and the messages will be populated in the lobbyMessages array. When the attendee is accepted or rejected, the state is reset to the default.

2. Backend

The *ChatRoom* class, used to create rooms for conferences, is the same class used to create lobby rooms. This means we are able to use the *sendMessage* and *sendPrivateMessage* methods to send messages to other participants in the (lobby) room. Currently, the mod_muc_lobby_rooms' filter_stanza method filters out these types of messages. We will add that to the filter_stanza to ensure the messages are sent to participants successfully.

3. Frontend

3.1 lib-jitsi-meet Library

Most of the lobby room functionalities are implemented in lib-jitsi-meet. It hides the implementation details of most of the lobby room features from jitsi-meet. I think it will be better if we also follow the same approach. We could implement most of the utility functions that we will be using in the main jitsi-meet app. Below are some of the functionalities we will need to consider.

3.1.1 Utility Methods in Lobby.js

- **sendMessage**

This is the method that is going to be responsible for sending messages to all participants in the lobby room. (This will be needed to broadcast information to the other moderators about the chat flow in the lobby room so that they would be informed about the attendee who is yet to be paid attention to)

```
sendMessage(message: Object);
```

- **sendPrivateMessage**

This is the method that is going to be responsible for sending private messages to only a participant. It could be from an attendee to a moderator or from a moderator to an attendee. Note that, messages from one attendee to the other will be forbidden as it is not necessary.

```
sendPrivateMessage(
      id: string, // paticipant id
      message: Object
);
```

- **setMessageListener**

This is the method that will be called when a new message is received from the lobby room to a particular member (moderator or attendee). This method will be used primarily for the chat events in this lobby room.

```
setMessageListener(listener: (message: Object, from: string,
privateMessage: boolean) => void);
```

3.1.2 Exposed Lobby.js Methods in JitsiConference.js

We are exposing these methods in the JitsiConference.js because this seems to be the standard convention of the contributors. For example, the joinLobby method of the JitsiConference class calls the join method in the Lobby.js class.

- **sendLobbyMessage**

This is the method that will call the sendMessage method in the Lobby.js class.

```
sendLobbyMessage(message: string | Record<string , any>);
```

- **sendPrivateLobbyMessage**

This is the method that will call the sendPrivateMessage in the Lobby.js class.

```
sendPrivateLobbyMessage(
      id: string,
      message: Object
);
```

- **setLobbyMessageListener**

This is the method that will call the setMessageListener in the Lobby.js class.

```
setLobbyMessageListener(listener: (
      message: Object, from: string, privateMessage: boolean
) => void);
```

3.1.3 Message Types

- **ChatInitialized**

This is the type of message broadcasted to all lobby room members when a moderator starts a chat with an attendee in the lobby room. When other moderators receive this event, their chat button for that attendee will be disabled. When the attendee receives this event their lobby room loader will be swapped with a chat widget. It will be sent using the sendMessage method.

```
interface ChatInitialized {
      type: 'CHAT_INITIALIZED', // message type
      attendee: {
      name: string; // attendee name
      id: string; // attendee participant id
      },
      moderator: { // moderator who started the chat
      name: string; // moderator name
      id: string; // moderator participant id
      }
}
```

- **ChatMessage**

  This is the type of message that a moderator sends to an attendee or vice versa. The privateMessage api will be used to ensure the confidentiality of the message.

```
interface ChatMessage {
      type: 'CHAT_MESSAGE', // message type
      message: string; // the message sent
      participantId: string; // the participant id
}
```

3.2 Jitsi Meet App

The main objective of this project is to extend the existing lobby room feature. Due to this, the already available lobby module in the jitsi-meet app will be the center of attention. Below are some of the features that will be added to the jitsi-meet app.

3.2.1 config.js

A new key 'allowChatInLobby' will be added to the config.js file. This will help users decide if they want to allow chat in the lobby feature. The default value will be true.

3.2.2 features/lobby Reducer

This reducer is going to host the state of whatever will be going on between a moderator and an attendee.

We already have some helpful states such as knockingParticipants, and knocking states for the moderator and the attendee respectively already. The knockingParticipants state will be extended to have a chatStarted key to help other moderators to know which participants are already in a chat. When we receive the ChatInitialized message type we will be able to decide that accordingly. Additional key in this reducer will be chattingWith. This is the state that will keep track of the current moderator or attendee that a participant is in chat with. When this key is null no chat widget will be shown. The last but not least key in this reducer will be lobbyMessages. All these states will be reset to

their defaults after the attendee is accepted or rejected by a moderator. The lobbyMessages key type is specified below.

```
interface LobbyMessage {
      message: string;
      date: Date;
      participantId: string;
}
lobbyMessages: Array<LobbyMessage>
```

3.2.3 Chat Message Handler

The events needed for the chat flow in this implementation will be set when the user joins the lobby room for the first time. The moderator joins a lobby room when the lobby feature is enabled. The attendee joins the lobby room when he tries to join the main room for the first time. We can listen to the messages and update the lobby reducer accordingly by making use of our setLobbyMessageListener method.

3.2.4 Jitsi Conference Access

A moderator can access the conference at features/base/conference reducer's conference key. The attendee can be able to access the conference from the same reducer's membersOnly key. This means the setLobbyMessageListener, sendLobbyMessage and sendPrivateMessage methods will be accessible from this state value.

3.2.5 UI Components

To prevent to code duplication, as stated in the contribution guide, chat components in the react/features/chat directory for the main chat widget, and base components in the react/features/base for the buttons and other UI elements be used; which is sufficient for the presented initial UI wireframe and there is no need to create any custom component.
Any changes to the existing code and any added function will be provided with a JSDoc comment, as stated in the contribution guide. All the text for the UI elements will be translated, English and Turkish texts for the components will be provided in the JSON files. All the UI components will be rendered with all the

necessary condition checks, so if the user disables the challenge response feature, everything will work as expectedly.

3.2.6 Mobile

Since jitsi-meet uses react native, most of the UI code for the mobile is shared with the web. Mobile specific UI components are placed in the native folder under each react/features folder. Similar to web components, chat components in the react/features/chat/native directory for the main chat widget, and base components in the react/features/base for the buttons and other UI elements be used.