Teleparty Extension Challenge

Hi! We're very excited that you're interviewing at Teleparty. We created this task so that we can evaluate your ability to learn and code. The challenge is expected to take 3 hours. We hope that you learn something new working on this. Good luck! ☺

Task:

Implement the front-end for a simplified real-time chat application using this library. Your application will use WebSockets to communicate with Teleparty's backend in order to facilitate real-time communication. Familiarity with Websockets is not required for this challenge, details about the Websocket connection have been encapsulated through our custom library. It will be your job to set up an event listener through our library and call the appropriate functions in order to make the application functional.

Functional Requirements:

- Users should be able to press a button in order to create a chat room
- Users should be able to enter an ID in order to join a chat room

- Users should be able to send chat messages within the room.
- Users should be able to set their nickname when joining or creating a chat room
- Users should be able to view chat messages that they have sent and others have sent in the room.
- Load all previous chat messages when someone joins a session
- Show typing presence when someone is typing in the chat room

Optional Functionality:

- Allow users to choose/upload a user icon

Technical Requirements:

- Upload solution to a public Github Repository and host the application
- Javascript

Preferred Technical Approaches:

- Create the App using React
- **Typescript**
- Deploy web app to Github Pages

Documentation

Sending Chat Messages:

- You will send chat messages through the sendMessage function on the Teleparty Client
- The server expects chat messages to have the type: SocketMessageTypes.SEND_MESSAGE
- The data for chat messages that the client sends to the server should be of the form:

```
{
  body: string;
}
```

- We have created an interface for this data called SendMessageData

Receiving Chat Messages:

- You will receive chat messages through the onMessage function of the SocketEventHandler
- The server sends chat messages to the client with the type: SocketMessageTypes.SEND_MESSAGE
- The data for chat messages that the server sends to the client is of the form:

```
{
  isSystemMessage: boolean;
  userIcon?: string;
  userNickname?: string;
  body: string;
```

```
    permId: string;
    timestamp: number;
}
```

  - Timestamp here is a Unix Timestamp in Milliseconds
  - System Messages are messages that were initiated by the server
    - i.e (when someone joins or leaves the chat room, or when the chat room is created)
    - You are expected to process system messages
- We have created an interface for this data called **SessionChatMessage**
- If you choose to load all previous messages when someone joins a chat, you will receive an object that implements the **MessageList interface.**

Updating the user's chat presence (Optional):

- In order to update a user's current presence state, send a message to the server with the type specified by the enum: SocketMessageTypes.SET_TYPING_PRESENCE
- The data for the presence message should be of the form:

```
{
    typing: boolean;
}
```

- We have created an interface for this data called **`SetTypingMessageData`**

Receiving chat presence updates (Optional):

- You will receive chat messages through the onMessage function of the SocketEventHandler
- The server sends presence updates to the client with the type: SocketMessageTypes.SET_TYPING_PRESENCE
- The data for chat messages that the server sends to the client is of the form:

```
{
    anyoneTyping: boolean;
    usersTyping: string[]
}
```

  - Here usersTyping is an array of permIds of the users currently typing in chat
  - Note: You do not need to display the names of the users who are typing, just indicating someone is typing is sufficient.

- We have created an interface for this data called **TypingMessageData**

Additional Documentation can be found on the readme for the library

Tips / Suggestions

- Wait for the Connection to be ready before attempting to create/join a room.
- You don’t need to worry about maintaining the socket connection. If you detect that the socket has closed via the onClose function in the SocketEventHandler, you can prompt the user to reload the app.

Final Notes:

- Aim to spend 3 hours on this challenge, not including the time spent reading documentation / setting up.
- Do not stress yourself trying to complete the optional requirements if you find them challenging or they take too much time
- Focus on building a solution that satisfies all the functional requirements before working on making it look pretty or trying some of the optional additions
- We appreciate a visually satisfying solution but are more concerned with your code/implementation.
- Feel free to browse the repository for our library to see how it’s implemented
- If you have any questions or run into any problems with the library while attempting the challenge, please email us.
- Feel free to add any additional bells and whistles you see fit
- Have fun!