Microsoft Engage ‘21

The Challenge
Build a Microsoft Teams clone.

Project Documentation

Developed By :-
Name : Shridhar Thakur
Email : thakur.shridhar3093@gmail.com
College : Birla Institute of Technology, Mesra
Branch : Information Technology (B.Tech.)
Graduation : 2023

Contents :

Features :

A. **Minimum Requirement :** A minimum of two participants are able to connect with each other using my product to have a video conversation.

B. **Group Video Conversation :** By creating a room mapped with a particular room ID my application is able to support group video conversation. On current deployment 4 people can have video conversation with negligible latency. More than 4 people can also join but it may create some latency.

C. **Lobby Page :** On the lobby page, the user is able to create new rooms or join existing rooms with room ID. It also contains a **copy to clipboard button** to copy the newly created room ID.

D. **Room Page :** On the room page, the user is able to connect to other peers in the same room. This is seen as new video elements appends to the video grid when a new user joins.

E. **Mute/Unmute Button :** It is available on the room page bottom bar, and it is used by a peer to mute/unmute themselves in the meeting.

F. **Stop/Play Video Button :** It is also available on the room page bottom bar, and it is used by a peer to stop/show their video feed in the meeting.

G. **Leave Meeting Button :** It is used to redirect back to the lobby page.

H. **Chat Feature ("Adopt" Feature) :** This can be used by a peer to send text messages to others on the same room ID. The chat is displayed with the name of sender, message body and send time of message.

I. **Persistence of Chat ("Adopt" Feature) :** This means that the chat persists in a room even when the user has left the meeting. The user can access chats again by rejoining with the same room ID.

J. **Join/Leave Alert** : Whenever a peer joins or leaves, the chat div of other peers on the same room ID is updated with a join or leave alert.

Technologies Used :

Frontend :-

1. **EJS :** I used EJS to dynamically render HTML views with the data sent from the server. EJS is an easy to use and lightweight templating engine. Moreover it fits well with NodeJS and Express.
   Link : https://ejs.co/

2. **CSS3 + Bootstrap v5 :** I used CSS combined with Bootstrap v5 to create lightweight, fast and responsive views for medium to large screens. Bootstrap provides easily customisable CSS classes that speeds up UI development.
   Link : https://getbootstrap.com/

3. **JavaScript + jQuery :** I used modern JavaScript combined with jQuery for UI events. jQuery provides a cleaner and faster way of DOM manipulation and event handling.
   Link : https://jquery.com/

4. **Font Awesome :** I used font awesome to get icons for different event buttons in the UI.
   Link : https://fontawesome.com/

5. **WebRTC** : With WebRTC, we can add real-time communication capabilities to our applications. It supports video, voice, and generic data to be sent between peers, allowing us to build powerful voice- and video-communication solutions. I used PeerJS to implement this.
   Link : https://peerjs.com/

Backend :-

1. **NodeJS :** I used NodeJS to create the server for my application because it has an event-driven architecture capable of asynchronous I/O. Moreover, It is also based on JavaScript which is consistent with frontend.
   Link : https://nodejs.org/en/

2. **Express :** This node module takes care of a lot of boilerplate code in NodeJS and speeds up server-side development.
   Link : https://www.npmjs.com/package/express

3. **Nanoid :** This node module generates a unique ID for different rooms.
   Link : https://www.npmjs.com/package/nanoid

4. **WebSockets :** With this API, we can send messages to a server and receive event-driven responses without having to poll the server for a reply. I used Socket.IO to implement this.
   Link : https://socket.io/

5. **Twilio :** I used twilio to generate credentials for its TURN (Traversal Using Relays around NAT) server that is required by webRTC to connect peers behind different networks.
   Link : https://www.npmjs.com/package/twilio

6. **Serve-favicon :** Node.js middleware for serving a favicon.
   Link : https://www.npmjs.com/package/serve-favicon

Deployment/ Version Control :-

1. **Heroku :** I used heroku free dynos to deploy my nodeJS server. Since I am using the free tier of heroku, there is a limitation that my server, after 30 minutes of inactivity (no server calls), will go to sleep which creates some latency on the next server call. If possible, please ignore this initial latency.
   Link : https://www.heroku.com/

2. **Git + GitHub :** I used git and github for version control.
   Link : https://github.com/

Text Editor/ IDE :-

1. **Visual Studio Code :** https://code.visualstudio.com/

Dev Dependencies :-

1. **Nodemon :** Nodemon is a tool that helps develop node.js based applications by automatically restarting the node application when file changes in the directory are detected.
   Link : https://www.npmjs.com/package/nodemon

2. **Dotenv :** Dotenv is a zero-dependency module that loads environment variables from a .env file into process.env.
   Link : https://www.npmjs.com/package/dotenv

Agile Methodology :

I used the SCRUM subset of Agile Methodology. SCRUM promotes an iterative model where the planning is performed on a very short term making it quicker to achieve small goals. It also helped me to reject or adopt changes frequently without them breaking the project workflow.

Development Cycle :
1 sprint = 4 days

I used this magnitude of sprint because it gave me enough time to implement the new features. And in the remaining three days of the week I was able to test these features and get ready for the next sprint.

Finally in the last few days I was busy documenting my project and creating the video demo.

Project Workflow :

1. User gets to the Lobby page and either creates or joins a room with a Room ID and their Name.

2. They are redirected to their required Room page.

3. Here, the client emits a “join-room” event to the server’s web socket passing in their room ID and peer ID (generated by PeerJS).

4. The server then emits this peer ID through “user-connected” event to every client on the given room ID.

5. Every client on the room hence receives this new peer ID and calls this new peer through PeerJS (implementation of WebRTC) passing in their video and audio streams.

6. Then, this new peer answers the incoming calls by passing his/her video and audio streams.

7. Therefore this new peer is connected to all in the room through video and audio.

8. Chat events are handled in a similar way through web sockets. The sender emits “message” event to the server's web socket passing in the room ID and message body which the server then emits to all clients on the same room ID as “create-message” event.

9. After emitting this event server also saves the chat so that chat persistence is achieved i.e., chat can also be seen on rejoining with the same room ID.

Server Routes :

1. **Route :** “/”
   Method : GET
   Response : It renders the lobby.ejs page.

2. **Route :** “/room”
   Method : GET
   Query Params : It requires the name and room ID.
   Response : It renders the room.ejs page with TURN server credentials that is used by Peer constructor (included in PeerJS) and it also provides chats of that room ID.
   Error Response : If either name or room ID is NULL or invalid then it redirects to “/”.

3. **Route :** “/createRoom”
   Method : POST
   Request body : {}
   Response : It responds with a unique new room ID.

4. **Route :** “/joinRoom”
   Method : POST
   Request body : { name : <user_name>, roomId : <room_ID> }
   Response : It responds with the required room URL which the client javascript uses to redirect the user to their required room.
   Error Response : If the room ID is invalid then it responds with “Room ID invalid” message.

WebSocket Events :

1. **“join-room”** - This event is emitted by the new user to the server socket passing in the userID and roomID. On this event the server responds with a “user-connected” event.

2. **“user-connected”** - When the server receives the “join-room” event, it responds with “user-connected” event to all clients on the required roomID by passing in the userID of the client which created the “join-room” event.

3. **“message”** - The sender client emits this event to the server socket by passing in the message object with required information such as sender’s name, message body. On this event the server appends the current date and time to the message object and emits a “create-message” event. The server also saves this message object mapped to the room ID.

4. **“create-message”** - This event is emitted by the server to all clients on the required room ID passing in the new message object. The clients on this event append the new message on their view.

5. **“disconnect”/ “user-disconnected”** - On “disconnect” event the server emits a “user-disconnected” event to all clients on the room ID so that they can clean their video feed by removing the disconnected userID.

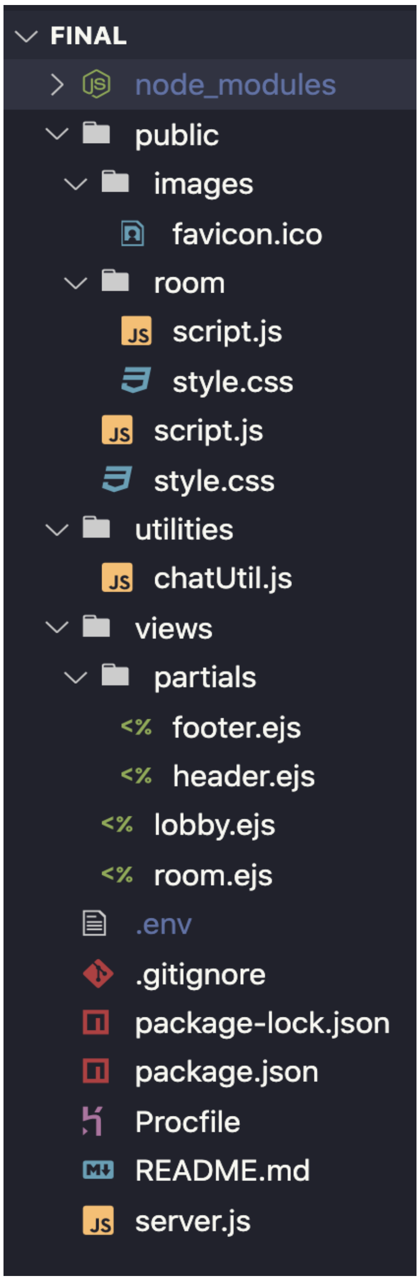
   Note : On connect/disconnect events the server also emits a “create-message” event to send a join/leave alert to all clients on the room ID.

Utility Functions :

1. **getUTCDate() :** This function is used in the backend to get UTC date for the chat message objects. Defined in **chatUtil.js**

2. **chatCleanup(chatThread) :** This function is used in the backend to remove older message objects of the provided chatThread if the chat thread exceeds a particular length (500 message objects). Defined in **chatUtil.js**

Project Directory :



```
FINAL
  node_modules
  public
    images
      favicon.ico
    room
      script.js
      style.css
    script.js
    style.css
  utilities
    chatUtil.js
  views
    partials
      footer.ejs
      header.ejs
    lobby.ejs
    room.ejs
  .env
  .gitignore
  package-lock.json
  package.json
  Procfile
  README.md
  server.js
```

X---END---X