MusicPlayer React Web Application Documentation

Documentation by Sione Havili

12/14/23

Weber State University

Table of contents

SocketProvider (Interface) Component

1. Introduction

SocketProvider was designed to manage all frontend to backend ('server.mjs') communication. Initially it was meant to allow any react component in the application to have a streamlined process of communicating with the backend ('server.mjs'). It still can do that, however our team didn't take the time to properly nest the SocketProvider wrapper components properly. socketProvider currently only does communication through an initial parent React component ('Rooms') which directs a lot of socket communication and components.

2. Usage

- Import SocketProvider React Component and use it as a wrapper component. Any child component within the wrapper component of the SocketProvider should be able use SocketProvider functions.
- In any child component of the Socket Provider wrapper component, import '{useSocket}' from file path to socketProvider, UseSocket will allow you to access any of its functions.
- OnLobbyData() - Grabs Initial data for Lobby Component which is called in parent ('Rooms') Component.
- OnAudioRoomData() - Socket function that manages all room audio data and room control data. This makes a direct connection to all users to compare their roomNumber to the incoming data before applying data.
- OnReceiveAll() - Serves as a catch all basket for server communication. SendIt() will send data to the backend ('server.mjs') which will broadcast to all users. By naming your identifier you can handle this action in your react component in the receiveAll() function. This should be used for low level communication ex: general news feed.
- onRoomControlData() - directly manages data for RoomControl component
  SendIt() - single function that sends all data to backend, current implementation can be simplified. All data goes through here.

3. Props:

- n/a

4. State:

- Socket constant variable stores all the values & functions within a single useState variable before passing as useSocket();

5. Lifecycle Methods:

SocketProvider life cycle exists as long as any of its children components within the socketProvider wrapper component are still using it for communication.

6. Event Handling:

All receiving data socket functions (all but sendIt()) must be handled as an arrow function. All receiving data socket functions must also be handled inside of a useEffect() (React Requirement) ex:

function(receivedData => { console.log("Im handling this function" + receivedData);

7. CSS Classes/Styles:

- n/a

8. Performance Considerations:

- It will only work for children component within it's wrapper component

9. Contributor(s) / Contributions

- Developer : Sione Havili

10. References

- SocketIO - How to use with React | Socket.IO

Rooms Component

1. Introduction

Rooms component is the main component which maintains all UI component switching and backend logic based on received data from SocketProvider(). Upon rendering, Rooms waits for initial data for the LobbyData component which contains the data of how many rooms are open and creating a room. Once a user creates/joins a room they are given initial room audio data and room control data. Room will then switch to the AudioRoom component which handles all room audio and room control data.

2. Usage

Rooms must call useSocket() to access its communication functionality. Use SendIt() and ReceiveAll() for all communication, additionally you may adjust predefined specific socket data receiving functions. Rooms isn't necessarily a reusable react component but rather the component that links itself to SocketProvider and provides communication and UI switching.

3. Props:

n/a

4. State:

- roomState - State of the Rooms component so that it may logically switch UI's
- roomNumber - Identifies data and. ensure the data is being received by the right room.
- lobbyData - holds data specific to displaying all open/available rooms (aka lobby) lobbyData is needed so users can create/join a room.
- roomControls holds data for the RoomControls component.

5. Lifecycle Methods:

Rooms can exist as an independent web application component because it utilizes Socket Provider communication. Therefore it exists as long as its component is called. In our instance, as long as the user is connected.

6. Event Handling:

All (but SendIt()) SocketProvider Functions must be handled within a useeffect(). SendIt() however, can be called anywhere.

7. CSS Classes/Styles:

Rooms.module.css - CSS stylesheet exists in same folder as its corresponding component

8. Dependencies:

All communication between frontend and backend in done through useSocket() (which refers to socketProvider Functions that provide communication)

9. Examples:

- Send Data - sendIt("newFeed", [roomNumber, newsFeedData]);

- Receive data (called within a useeffect) - onReceiveAll((receivedDataIdentifier, recievedData) =>
```
{
        If (receivedDataIdentifer === "NewsFeed" && roomNumber ===
        "receveidDataIdentifier[0]) { setNewsFeed(recievedData)}
}
```

10. Performance Considerations:

- Any React component can use UseSocket for its communication with backend ('server.mjs')
- Room always exists once its called since it a direct child component of socketProvider()

11. Contributor(s) / Contributions

- Developer : Sione Havili

Lobby Component

1. Introduction:

Rooms component will first render Lobby component after receiving Lobby components initial data and pass it the total amount of rooms currently existing. Lobby component creates a button for each room to be selected. Lobby component notifies parent (Rooms) if a connected user creates or joins a room. Lobby component is intended to be a lightweight component.

2. Usage:

Lobby component only needs to know how many rooms in order to initially render properly. Lobby will notify parents when the user creates/joins a room.

3. Props:

- incomingLobbyData = total rooms open
- onSendLobbyData(passItStringValue[“CreateRoom”] or index value of room selected to join.)

4. State:

n/a

5. Lifecycle Methods:

Rooms component will always retrieve initial lobby data and render Lobby component first. When a user selects an audio room (from lobby selection), the Rooms component will communicate with the backend to retrieve audio data and room controls. Once Room receives room data it will render the AudioRoom component at which Lobby component’s life cycle will end. Lobby component will be re rendered should a user decide to leave its current room thus re rendering Lobby component with latest data.

6. CSS Classes/Styles:

Lobby.module.css - CSS stylesheet exists in same folder as its corresponding component

7. Dependencies:

Lobby component's lightweight design requires a parent component to initialize room count and call Lobby component.

8. Contributor(s) / Contribution

- Developer : Sione Havili

AudioRoom Component

1. Introduction:

Once a Rooms component has received a room selection from the user and received specified room's audio data and room controls, it will switch from rendering Lobby Component to rendering AudioRoom component. AudioRoom Component is the main interface of the Audio Room in which users can play, pause and skip music. Backend ('server.mjs') holds audio data for all existing rooms.

2. Usage:

AudioRoom component will take audio command input from user, parent component (Rooms) will relay current audio data and room controls. Backend ('server.mjs') will update room data on backend and notify all rooms.

3. Props:

- roomNumber - All room communication data is identified by room number
- trackPosition - Every audio command will notify the backend which will update trackPosition every time.
- trackTimeStamp - Marks the time the sender made the command, this is not in use at the moment. It was intended to adjust all user web applications to begin mp3 audio at the exact same time. This can still be done but would take a significant amount of time to dial in perfectly.

- partyCount:0 - is not currently in use. It was intended to be used for directly sending audio to a single user. However we did not finish implementing a username and login system.
- isTrackPlaying - indicates to audioRoom if they should begin playing immediately or if the audio should currently be paused.
- Host - Rooms component will hold a unique userID (socket id). If a user creates a room or takes over as host, their userID will be set to host. If the host disconnects (hard refresh), the host will be set to null and all users in the room will be notified of host availability. Anyone in the room can take over but only 1 user can be the host at a time.
- SongIndex - allows audioRoom to notify which song in the array is currently selected. It allows AudioRoom to locate the correct mp3 file and allows identification of the current song.
- skipVoteCount - keeps count of the current songs skip count, if it reaches 2, backend will notify all to move to next song.

4. State:

- audioRef - is a useRef() that references the audio component. This is how the audioRef component state is changed (‘play’,’pause’,’skip’);
-
- buttonDisabled - will disable pause and play buttons according to host settings
-
- roomControls - holds state data for RoomControl component
-
- canUserVote - holds value for whether a user has voted on the current song. This variable is handled separately because it only exists as long as the corresponding mp3 audio file. It is passed down from the Rooms component.

5. Lifecycle Methods

AudioRoom will be rendered by parent (Rooms) once the user has selected a room to join or to create a room and the backend server has returned room audio data. The AudioRoom component will exist until the server has been restarted. If user disconnects, the most recent audio room data will be passed to anyone who joins the room

6. CSS Classes/Styles:

AudioRoom.module.css - CSS stylesheet exists in the same folder as its corresponding component.

8. Dependencies:

AudioRoom component is dependent upon a parent (Rooms) component to initialize its data and communicate with backend to update all in room.

9. Performance Considerations:

AudioRoom does not always completely re-render, it will rerender when

10. Known Issues and Limitations:

AudioRoom will completely re-render if usestates aren't well managed.

11. Contributor(s) / Contributions

- Developer : Sione Havili

RoomControls Component

1. Introduction:

RoomControls is a lightweight component that displays and handles controls in the Audio Room. RoomControls will provide the host with options for play/pause controls and audio output control. Host can dynamically change these settings. For any room guest that is not the host, RoomControls will simply display the settings set by the host.

2. Usage:

If a room host has disconnected, RoomControl will display a button which allows any room guest to take over as host. Upon a user choosing to take over as host, all room guests will be updated and controls will be given to that user. RoomControl is designed to be a lightweight component that displays room controls options and updates parent (Rooms) component and parent component will update room controls once backend has received it and sent it out to all.

3. Props:

- host - this is the userID generally held in the parent (Rooms) component, if null, this tells RoomControls there is not host, which RoomControl will display an option for all users to 'be host'
-
- isHost - notifies RoomController if their specific user is the host. If they are, they will be given control input, else any other user will just see the settings set by the host.
-
- isHostControl - Determines whether the host has controls open to all room guests or is is set to host only
-
- audioOutput - Determines whether the host has audio playing through their speaker only or all room guest speakers.

4. State:

n/a

5. Lifecycle Methods

RoomControl is designed to be a lightweight component. It will re-render every time any of its data is changed/altered. Once the RoomControl receives an input, the backend ('server.mjs') will notify all users of the new settings which will re-render the component.

6. CSS Classes/Styles

RoomControls.module.css - CSS stylesheet exists in the same folder as its corresponding component.

7. Dependencies

As a lightweight component, it will always depend on a parent component to handle all communication. RoomControls is just a UI component.

8. Performance Considerations

RoomControls is a child component of AudioRoom to reduce unnecessary re-renders if audio data has not changed. RoomControls is designed to be re-rendered every time.

9. Contributor(s) / Contributions

Developer : Sione Havili

MusicList Component

1. Introduction

MusicList takes an array of strings of all mp3 files. The names of all mp3 files will be displayed in the form of a playlist. MusicList will provide a vote to skip button for the current mp3 song selection.

2. Usage:

Pass it an array of the mp3 files and it will display, MusicList component takes input from user and passes it back to parent component. Parent will notify backend (‘server.mjs’) and backend will notify room of skip vote.

3. Props:

- selectedSongIndex- The index of the given array which determines the current selection
- songList - the array of strings of mp3 file names
- hasVoted - if user has already voted on selected song, disable vote button
- currentSkipVoteCount - allows the component to display how many votes currently exist.

4. State:

n/a

5. Lifecycle Methods:

MusicList is a lightweight component and will re-render every time its data has changed

6. CSS Classes/Styles:

RoomControls.module.css - CSS stylesheet exists in the same folder as its corresponding component.

7. Dependencies:

Lightweight component will depend on a parent component to handle data.

8. Contributor(s) / Contributions

- Developer : Sione Havili

Server.mjs

1. Introduction

Socket.io can be used with react components to handle peer-to-peer communication as well as backend server communication. Rooms is an array that contains all AudioRoom data for all rooms that have been created, It is initialized with a room with standard information. All socket communication event handling must be done within the io.on("connection) function.

2. Sockets (incoming data)

- socket.on('connect') - logs once a user connects to server

- socket.on('createRoom') - Creates an initial set of audio data and pushes it to the Rooms array. It will also create an initial data set for host controls and push it to the HostControls array. A Callback function to the caller will return the caller's socket.id, room number, room data and room control data.

- socket.on("joinRoom") - receives a room index from caller, retrieves corresponding room's most recent data and uses call back function to return data to caller.

- socket.on('startStopAudio') - Incoming data should always be in the form of an array. First item in the array should be the corresponding room index. The second item in the array should be a string that serves as an identifier. Based on the identifier, all necessary will be updated. An array with room number and room data will be sent to all connected users, The frontend will compare room numbers to see if incoming data belong to their room.

- socket.on('sendAll') - is a general use socket communication. Anybody can make a command and it will be sent to all, on the front end is where logic should handle the data.

- socket.on('disconnect') - when a user disconnects, backend will check to see if it is user is a host and remove them, if they remove a host, they will notify all users

- socket.on("sendRoomControls") - updates room controls then emits room controls, this implementation works but having separate room controls from audio data was causing unexpected rerenders

- socket.on('beHost') - will update the host and send out to the room the updated room data.

3. Sockets (Outgoing Data)

- socket.emit("lobbyData") - When the Rooms component is loaded, backend will immediately send initial Lobby data in order to display all existing rooms.

4. Limitations

- Room controls are being passed with audio data. Room controls does not necessitate a reason to also update and re render audio data but current implementation has this as sending room controls separately was causing complete re-renders of parent (Rooms) component,

5. Issues

- Socket.io has specific built in room functions, however, we were unable to get a working implementation, Reasons why built in room functions didn't work are unknown.

- There is currently no implementation for when a user leaves the room. When a user leaves the room, the Room component will reset all their corresponding data on the frontend to initial values which allows the user to select another room, however the backend has not been updated. Note; leaving room is not the same as disconnecting (hard refresh).

6. Contributor(s) / Contributions

- Server.mjs has a collaboration of functions. All code in Server.mjs explained within this was done by Sione Havili.