Chat Application Report

- **Team SOAP:** Sangram Jagtap, Omkar Nagarkar, Atharva Jadhav, Purvil Patel

---

Objective

To build a simple chat application that allows users to send messages in real-time. When one user sends a message, it should appear in any other active sessions, ensuring real-time communication among users.

Implementation Approach

The chat application was developed twice, each time using different frontend and backend frameworks. The primary goal was not to create a perfect architectural masterpiece but to understand the nuances and characteristics of different frameworks.

Frameworks Used

1. Angular (Frontend) + Node.js (Backend)
2. React (Frontend) + Node.js (Backend)

Comparative Analysis

1. State Management:

a. **React:** useState hook is evident in the Chat component where currentMessage and chatList states are initialized.
const [currentMessage, setCurrentMessage] = useState("");
const [chatList, setChatList] = useState([]);

b. **Angular:** The ChatComponent uses class properties such as currentMessage and messageList to maintain state.
currentMessage: string = '';
messageList: Array<any> = [];

2. Event Handling and Data Binding:

a. **React:** The onChange event is used for the input field in the Chat component, and event data is manually updated.

```
onChange={(event) => {
  setCurrentMessage(event.target.value);
}}
```

b. **Angular:** Two-way data binding is shown with the [(ngModel)] directive in the input fields of app.component.html and chat.component.html.

[(ngModel)]="currentMessage"

3. Socket Management:

a. **React**: In App.js, the socket is initialized and passed as a prop to the Chat component. Event listeners are managed within useEffect in Chat.
b. **Angular**: The socket is initialized in app.component.ts and event listeners are set up in ngOnInit lifecycle hook in chat.component.ts.

4. UI Rendering:

a. **React**: JSX is utilized in chat.js for UI rendering with conditions embedded within JSX.
{chatList.map((messageContent) => { ... })}
b. ***Angular:*** *Angular templates are used in chat.component.html. Directives like* ****ngFor*** *handle loops.*
<div *ngFor="let messageContent of messageList" ...>

5. File Structure:

a. **React**: Flexible structure with JSX files combining logic and template.
b. **Angular**: Opinionated structure with separate files for templates (HTML), styles (CSS/SCSS), and logic (TypeScript).

MVC Architecture

1. Model (M):

- The model in the Angular application is represented by the variables used to store the chat information. For instance, in the app.component.ts file, the student, studyGroup, and displayGroupChat variables can be considered as part of the model.
- The model in the React application is represented by the state variables used to store chat information. In the App.js file, the student, studyGroup, and displayGroupChat state variables can be considered as part of the model.
- The data being transferred via the sockets (like chat messages) can be seen as data models in the context of this chat application.

2. View (V):

- The view is represented by the HTML templates that display the data to the user. For example, the app.component.html and chat.component.html files define the structure and layout of the chat application.
- The view in React is represented by the JSX code that defines the UI components. The App.js and Chat.js files contain the JSX code that renders the chat interface.

3. Controller (C):

- In the Angular application, the logic for handling chat messages, joining chat groups, and other functionalities is defined in the TypeScript files like app.component.ts and chat.component.ts.

- The controller logic in React is embedded within the functional components using hooks and event handlers. The functions like onJoinChat in App.js and onMessageSent in Chat.js handle user interactions and update the state accordingly.
- The index.js file in the Node.js backend sets up the server and handles socket events for chat functionalities.

The separation of concerns in the MVC architecture allows for modular and maintainable code. By keeping the Model, View, and Controller separate, it becomes easier to manage and scale the application.

Conclusion

React:

- **Pros**: Offers a flexible, component-centric approach. Direct integration with JS allows for easy logic embedding within the UI.
- **Cons**: Requires additional libraries for features outside its scope, like two-way data binding.

Angular:

- **Pros**: Comprehensive, offering out-of-the-box solutions. Two-way data binding simplifies UI updates, and a clear structure aids in maintainability.
- **Cons**: More verbose, especially for those familiar with a JS-centric approach.