**Major Learnings**

In the process of developing a chat webpage using React, The members of the group had several discussions and discussed possible techniques to be used:

**ReactJS:** This is a JavaScript library for building user interfaces. With React, you can create reusable components, making your code more readable and clean.

**Node.js and Express:** Node.js is a JavaScript environment that runs on the server side, and Express is a web application framework built on top of Node.js. You can use Node.js and Express to create a backend server to handle requests from the frontend.

**WebSocket:** This is a protocol for full-duplex communication on a single TCP connection. In a chat application, WebSocket can be used to send and receive messages in real time.

**Database Technologies**: For example, MongoDB or PostgreSQL. In a chat application, you might need a database to store user information and chat history.

**Redux or Context API:** These are state management libraries for managing the state within the application. In a chat application, you might need to manage states like user login status, the current chat room, etc.

Below is a brief overview of how each technology is implemented, and how data is transmitted between the frontend and backend:

**ReactJS:** In React, you create components to represent different parts of your application. Each component has its own state and properties, which it uses to render the corresponding UI. When the state or properties change, React automatically updates the UI.

**Node.js and Express:** In an Express application, you can define routes to handle different HTTP requests. For example, you might have a /messages route to handle requests related to chat messages. When the frontend sends a request, the Express server receives the request and processes it according to the defined routes.

**WebSocket:** In WebSocket, the client and server can send and receive data simultaneously on the same TCP connection. This is different from the traditional HTTP request/response model, where only the client sends requests and the server responds to requests. In a chat application, you can use WebSocket to implement real-time chat. When a user sends a message, the server can immediately push this message to all other online users.

**Database Technologies:** You can use a database to store and retrieve data. For example, when a user sends a message, you can store this message in the database. When another user logs in and opens the chat room, you can retrieve this message from the database and display it to the user.

**Redux or Context API:** In a React application, you might need to manage many different states. For example, whether the user is logged in, which chat room is currently open, etc. These states might need to be shared across multiple components. Using Redux or Context API, you can store these states in a unified place, and then access these states in the components that need them.

In the process of data transmission between the frontend and backend, the general steps are as follows:

**User Operation on the Frontend:** For example, the user enters a message in the chat box and clicks send.

**Frontend Sends Request to Backend:** After the user clicks send, the frontend will send the user's input message as request data, using methods such as Ajax or Fetch, sending an HTTP request to the backend server. If using WebSocket, this step would be sending the message over the WebSocket connection.

**Backend Processes the Request:** After receiving the request, the backend server will process the request according to its content. For example, if the request is to send a chat message, the server might store this message in the database, and push this message to other users via WebSocket.

**Backend Returns Response to Frontend:** After processing the request, the backend server will return a response to the frontend. This response might contain the processing result, such as whether the message was sent successfully.

**Frontend Processes the Response:** After receiving the response, the frontend will update the UI according to the content of the response. For example, if the response from the backend indicates that the message was sent successfully, the frontend might clear the input box and display the new message in the chat history.

**Future Work**

After the team members had completed most of the locally running web pages, we next wanted to deploy the web pages on top of AWS servers and prepare them for testing against the high concurrent chat requirements of a large range of users.

This will be followed by the launch of the mobile app and the PC app, and the addition of file transfer capabilities.