María Alejandra Estupiñán Murilo

***PRUEBA Técnica***

***Parte Teórica***

1. ***Diferencias entre WebSockets y SSE (Server-Sent Events)***

La principal diferencia entre WebSockets y SSE es en el tipo de comunicación puesto que en el WebSockets la comunicación es bidireccional es decir existe un intercambio de mensajes entre el servidor y el cliente, en cambio en SSE es unidireccional en donde la comunicación solo se da entre el servidor que envía los datos al cliente pero este no puede responder.

Otra diferencia que se encuentra es que el envío de datos que soporta WebSocket es de binarios y texto, en cambio SSE solo admite texto.

***¿En que situaciones utilizarías cada tecnología?***

* ***WebSocket:*** La utilizaría en aplicaciones o sitios donde sea necesario la comunicación bidireccional continua y en tiempo real, esto permite una disminución en el ancho de banda y velocidad de la conexión, como por ejemplo los chats en tiempo real.
* ***SSE:*** La utilizaría en aplicaciones donde el contenido se debe estar actualizando periódicamente como es el caso de aplicaciones de visualización de noticias.

1. ***Manejo de Estado en Aplicaciones en Tiempo Real***

El correcto funcionamiento de los estados en aplicaciones en tiempo real hace que el usuario tenga una interacción amena con la aplicación y la forma en la que llevaría acabo esta labor seria contar con tres estados:

* ***Estado de mensajes:*** Este estado se encarga de mantener el diseño y la estructura general del chat pero que si vaya cambiando o actualizando el mensaje nuevo en pantalla
* ***Estado de conexión:*** En este estado se encarga de que la conexión con el ciberespacio siempre se encuentre presente mientras la aplicación se encuentre en uso
* ***Estado de cantidad de usuario:*** Estado que se encarga de limitar la cantidad de usuarios que pueden estar al tiempo conectados con el fin de no sobrepasar el ancho de banda dado para la aplicación.

Para la implementación de lo anterior utilizaría librerías como Reactcontext que permite un flujo de datos claro y centralizado en toda la aplicación y redux la cual es una librería que almacena todo el estado de la aplicación en un único lugar llamado \*store\*. Esto hace que sea fácil acceder y modificar cualquier parte del estado desde cualquier componente de la aplicación.

1. ***Optimización de una Aplicación Frontend en Tiempo Real***

La estrategia que aplicaría para optimizar el rendimiento de una aplicación de chat sería la de no renderizar las cosas que el usuario no está viendo para así evitar cuellos de botella con la constante comunicación entre el servidor y lo que se va presentando en la página web, hacer esto también repercute en que los tiempos de respuesta disminuyan y a medida que baje se va renderizando, de igual manera si el usuario quiere ver mensajes anteriores se le presenta la opción de hacerlo pero con una cantidad de información limitada, es decir la cantidad de mensajes que se renderizan estarán dados por bloques o grupos de mensajes.

1. ***Accesibilidad y Diseño de Interfaz de Usuario***

Para contar con todos los usuarios se realiza una interfaz accesible para todos en la cual se implementa un chat tipo texto en tiempo real o también se podría tener un chat tipo audio para aquel que cuente con alguna discapacidad pueda acceder sin ningún inconveniente, al igual que contar con diferentes propiedades de personalización, como lo sería el cambio en tamaño de letra, color, tipo de letra, que cuente con un motor de búsqueda para buscar palabras en específico, entre otras propiedades que provoquen una sensación de acogida en cada usuario.

1. ***Manejo de errores y reconexiones***

La forma en la que manejaría los errores de conexión en una aplicación de chat en tiempo real seria realizando un llamado de reconexión en la parte de código y por otro lado en la parte de la visualización del usuario por medio de un texto informar que se produjo un error de conexión y se está solucionando para seguir hablando con el chat de forma optima pero que lo que han estado hablando no se pierde.

1. ***Análisis de código en React***

***Código en React***

***// Importa React y los hooks useState y useEffect desde la biblioteca de React.***

***import React, { useState, useEffect } from 'react';***

***// Define el componente funcional ChatApp.***

***const ChatApp = () => {***

***// Declara el estado messages para almacenar la lista de mensajes en el chat.***

***// setMessages es la función que se usará para actualizar este estado.***

***const [messages, setMessages] = useState([]);***

***// Declara el estado newMessage para almacenar el mensaje que el usuario está escribiendo.***

***// setNewMessage actualiza el valor del mensaje actual en el campo de entrada.***

***const [newMessage, setNewMessage] = useState('');***

***// useEffect ejecuta una acción cuando el componente se monta.***

***// Este useEffect simula la llegada de un nuevo mensaje cada 3 segundos.***

***useEffect(() => {***

***// Configura un intervalo que ejecuta el código dentro de él cada 3 segundos.***

***const interval = setInterval(() => {***

***// Crea un mensaje simulado. El número se basa en la longitud actual de la lista de mensajes.***

***const simulatedMessage = Mensaje ${messages.length + 1};***

***// Actualiza el estado messages, agregando el nuevo mensaje simulado a la lista existente.***

***setMessages([...messages, simulatedMessage]);***

***}, 3000); // 3000 milisegundos = 3 segundos.***

***// La función de limpieza se ejecuta cuando el componente se desmonta,***

***// eliminando el intervalo para evitar actualizaciones no deseadas.***

***return () => clearInterval(interval);***

***}, []); // El array vacío indica que esto solo se ejecuta una vez al montar el componente.***

***// Función para manejar el envío de un nuevo mensaje por el usuario.***

***const handleSendMessage = () => {***

***// Verifica si newMessage no está vacío (después de eliminar espacios en blanco).***

***if (newMessage.trim() !== '') {***

***// Si hay texto en newMessage, agrega el mensaje actual a la lista de mensajes.***

***setMessages([...messages, newMessage]);***

***// Limpia el campo de entrada de mensaje, reiniciando newMessage a una cadena vacía.***

***setNewMessage('');***

***}***

***};***

***// Renderiza la interfaz de usuario de la aplicación de chat.***

***return (***

***<div>***

***{/\* Contenedor para mostrar los mensajes en la lista \*/}***

***<div>***

***{/\* Mapea sobre la lista de mensajes para renderizarlos en el orden de llegada \*/}***

***{messages.map((msg, index) => (***

***// Muestra cada mensaje en un párrafo. key={index} se usa para dar una clave única a cada mensaje.***

***<p key={index}>{msg}</p>***

***))}***

***</div>***

***{/\* Campo de entrada para que el usuario escriba su mensaje \*/}***

***<input***

***type="text" // Define el tipo de entrada como texto.***

***value={newMessage} // Vincula el valor del input con el estado newMessage.***

***onChange={(e) => setNewMessage(e.target.value)} // Actualiza newMessage con el texto ingresado.***

***/>***

***{/\* Botón para enviar el mensaje \*/}***

***<button onClick={handleSendMessage}>Enviar</button>***

***</div>***

***); };***

***// Exporta el componente ChatApp para que pueda ser importado y utilizado en otros archivos.***

***export default ChatApp;***

***/\* Dependencias de useEffect:***

***Error: messages no está en las dependencias del useEffect, lo que significa que el interval no se actualiza correctamente cuando cambia el estado.***

***Solución: Añadir messages como dependencia o usar la función setMessages(prevMessages => [...prevMessages, simulatedMessage]).***

***Uso de índices como claves:***

***Error: Utilizar índices (key={index}) como claves puede causar problemas de renderizado si los mensajes cambian de orden o se eliminan.***

***Solución: Usar identificadores únicos, como un timestamp o UUID.***

***Accesibilidad del botón de enviar:***

***Error: El botón de "Enviar" carece de atributos de accesibilidad (aria-label) y no permite su activación con el teclado.***

***Solución: Agregar aria-label="Enviar mensaje" y soporte de accesibilidad con un evento onKeyDown.***

***Desempeño de la lista de mensajes:***

***Mejora: Con una gran cantidad de mensajes, la renderización puede volverse lenta. Se recomienda usar virtualización de listas con librerías como react-window.***

***\*/***

***Código en Javascript***

//----------------------------Codigo en javascript----------------

// Se declara una función llamada fetchData para obtener datos desde una API.

function fetchData() {

/\*Se declara una variable data cuyo valor de inicio es null, se crea con el fin

de que almacene los datos que se van obteniendo de la API\*/

let data = null;

fetch('https://api.example.com/data')

// Llama a la función fetch con la URL de la API como argumento.

// fetch es una función asíncrona que envía una solicitud HTTP y devuelve una promesa.

.then(response => {

// Usa el método .then para manejar la respuesta cuando la promesa se resuelve exitosamente.

// La respuesta HTTP obtenida es pasada como argumento a esta función response.

data = response.json();

// Intenta asignar el resultado de response.json() a la variable data.

// response.json() es una función asíncrona que convierte la respuesta en formato JSON,

// pero devuelve una promesa, no los datos directamente, lo que genera un error.

})

.catch(error => {

// Usa el método .catch para manejar cualquier error que ocurra durante la solicitud fetch.

// Si ocurre un error, el mensaje se registra en la consola.

console.log('Error:', error);

// Muestra el mensaje de error en la consola junto con la palabra "Error:" para facilitar su identificación.

});

return data;

// Retorna el valor de data. Sin embargo, dado que response.json() devuelve una promesa,

// data no contendrá los datos JSON deseados al ejecutar esta línea.

}

const result = fetchData();

// Llama a la función fetchData y asigna su valor de retorno a la constante result.

// Sin embargo, debido a la naturaleza asincrónica de fetch, result será null.

console.log(result);

// Muestra el valor de result en la consola, que probablemente será null en este contexto.

/\* Problema con la Asincronía:

Error: La función fetchData intenta asignar data usando una llamada asincrónica (fetch), pero retorna data antes de que la promesa se resuelva, lo cual resultará en null.

Solución: Usar async/await para esperar el resultado antes de retornarlo, o retornar la promesa. \*/

// solución

async function fetchData() {

// Declara una función asincrónica llamada fetchData.

// Al usar async, esta función devuelve automáticamente una promesa,

// lo que permite manejar el valor devuelto cuando se resuelva la operación asincrónica.

try {

// Inicia un bloque try para manejar posibles errores en el proceso de obtención de datos.

const response = await fetch('https://api.example.com/data');

// Usa await para pausar la ejecución hasta que la promesa devuelta por fetch se resuelva.

// fetch envía una solicitud HTTP GET a la URL especificada y espera la respuesta del servidor.

// El resultado se almacena en response cuando la solicitud finaliza

const data = await response.json();

// Usa await para pausar la ejecución hasta que response.json() se resuelva.

// response.json() es una función que convierte la respuesta en formato JSON y devuelve una promesa.

// Al usar await, data contendrá los datos JSON resultantes directamente en lugar de una promesa.

return data;

// Retorna los datos obtenidos. Como fetchData es asincrónica, esto devuelve una promesa que se resolverá con data.

} catch (error) {

// Si ocurre un error en cualquier parte del bloque try, este será atrapado aquí en el bloque catch.

console.log('Error:', error);

// Muestra el mensaje de error en la consola, facilitando el seguimiento y la depuración.

return null;

// Retorna null en caso de error para indicar que no se pudieron obtener los datos correctamente.

} }

fetchData().then(result => console.log(result));

// Llama a fetchData y usa .then para manejar el valor cuando la promesa se resuelve.

// result contendrá los datos JSON obtenidos o null si ocurrió un error.

// Muestra result en la consola.