Red P2P con Servidor Central para Registro de Clientes

👬 Integrantes del Proyecto:

- Santiago Andrés David Gómez
- Henry Guerrero Vargas

Objetivo General

Diseñar una red **Peer-to-Peer (P2P)** donde:

- Los **clientes se conectan entre sí directamente** para compartir archivos.
- Un **servidor central solo registra** a los clientes y gestiona la lista de nodos conectados.
- La **transferencia de archivos es 100% entre clientes**, sin pasar por el servidor.

💻 Instalación de Dependencias

Para ejecutar este proyecto en tu entorno local, sigue los pasos a continuación POR CADA CARPETA:

```
npm install
```

🔌 1. Servidor Central (Servidor de Registro)

Propósito:

Actúa como una libreta de direcciones para registrar y consultar clientes conectados.

Funcionalidad Clave:

- ☑️ **Registro de Clientes**: Guarda ID y dirección IP al conectarse.
- 🗃️ **Lista de Clientes Activos**: Retorna la lista completa de clientes conectados.

Rutas:

- `/register`: Registra cliente con su ID/IP.
- `/clients`: Devuelve lista de clientes activos.
- ❌ No gestiona archivos ni los almacena.

💻 2. Cliente (Cliente P2P)

Propósito:

Intercambiar archivos con otros clientes directamente.

Funciones Principales:

- ⚙️ **Configuración de Carpetas**:
  - Carpeta para compartir archivos.
  - Carpeta de descargas.
- 📂 **Escaneo de Archivos Locales**:
  - Lista los archivos disponibles para compartir.
- 📝 **Registro en el Servidor**:
  - Envía su ID/IP al servidor central al iniciar.
- 📋 **Descarga desde otros Clientes**:
  - Consulta lista de clientes.
  - Selecciona uno y obtiene su lista de archivos.
  - Solicita descarga directa (sin pasar por el servidor).
- 🖥️ **UI Dividida**:
  - Archivos descargados.
  - Archivos compartidos con info del cliente emisor.
- ⚡ **Transferencia Directa**:
  - Enlace directo entre clientes para enviar archivos.
- 🔄 **Actualización Dinámica de Archivos**:
  - Al agregar/eliminar archivos en la carpeta compartida, se actualiza localmente.
- ❌ **Manejo de Desconexiones**:
  - El servidor elimina a clientes desconectados.

⚙️ 3. Funcionamiento General

- ▶️ **Servidor se inicia**:
  - Escucha conexiones entrantes para registrar clientes.
- ▶️ **Cliente se inicia**:
  - Configura carpetas.
  - Escanea archivos.
  - Se registra en el servidor.
  - Consulta clientes activos.
- ↔ **Interacción entre Clientes**:
  - Un cliente A ve la lista de archivos de cliente B.
  - Solicita un archivo y lo descarga directamente.

- 🔄 **Actualización de Archivos**:
    - Al modificar archivos compartidos, se reflejan en la siguiente consulta entre pares.

⚠️ Consideraciones Importantes

🔐 Seguridad:

- Autenticación básica para registrar clientes.
- Cifrado durante transferencia de archivos.

📈 Escalabilidad:

- Balanceo de carga con múltiples servidores de registro.

🔧 Solución Técnica y Problemas Resueltos

Descripción de la Solución:

Este proyecto aborda la necesidad de una red de intercambio de archivos Peer-to-Peer (P2P). La solución se centra en garantizar que los archivos sean transferidos directamente entre clientes, sin que el servidor central se vea involucrado en la transmisión de los datos.

Servidor Central:

Se creó un servidor central que actúa solo como registrador de clientes. No se involucra en la transferencia de archivos, lo que garantiza eficiencia y evita cuellos de botella.

Cliente P2P:

Cada cliente puede enviar y recibir archivos directamente, gracias a la comunicación mediante WebSockets a través de la librería socket.io, asegurando transferencias rápidas y sin sobrecargar el servidor central.

Arquitectura:

Se utilizó Node.js y Express para el servidor, y Socket.io para la comunicación en tiempo real entre clientes. Además, se empleó un enfoque basado en archivos estáticos, donde se utilizaron express.static para servir los archivos estáticos del proyecto.

Interfaz de Usuario (UI):

Se diseñó una interfaz intuitiva para los clientes, permitiéndoles conectarse, seleccionar archivos y enviarlos a otros clientes con facilidad. La interfaz también muestra el estado de los archivos enviados y recibidos, como se muestra en la captura de pantalla proporcionada.

Retos Abordados:

- **Transferencia Directa de Archivos**: Se logró que los archivos sean transferidos directamente entre clientes mediante un canal WebSocket, sin necesidad de pasar por el servidor central.

- **Gestión de Conexiones y Desconexiones**: El servidor gestiona las conexiones de clientes, asegurándose de que aquellos que se desconecten sean eliminados de la lista de clientes activos.
- **Manejo de Archivos Locales**: Se implementó una funcionalidad para escaneos automáticos de las carpetas compartidas, lo que permite a los usuarios ver qué archivos tienen disponibles para enviar.

Conclusión:

Esta solución permite a los usuarios compartir archivos entre ellos de forma rápida y eficiente, respetando la privacidad de la transferencia, ya que el servidor solo se encarga de registrar las conexiones y gestionar la lista de clientes.

🛠️ Estructura del Proyecto

```
/cliente
  |-- client.js           # Lógica del cliente, manejo de conexiones y archivos.
  |-- index.html          # Interfaz de usuario (UI) del cliente.
  |-- styles.css          # Estilos para la UI.

/servidor
  |-- server.js           # Lógica del servidor de registro, manejo de conexiones.
  |-- public/             # Archivos estáticos como imágenes, CSS, JS.
```

¡Gracias por tu interés en nuestro proyecto P2P!

Este archivo README.md detalla el objetivo general, la solución técnica implementada, y explica cómo se resolvieron los desafíos encontrados durante el desarrollo del sistema de intercambio de archivos P2P con servidor central para registro.

Anexos