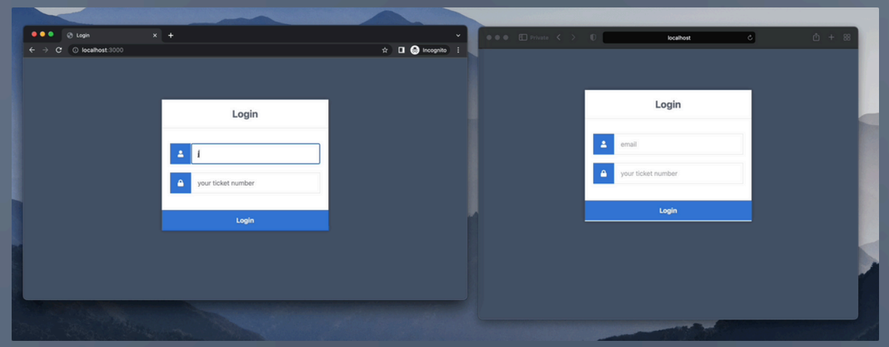


Fundamentos de Node.js para un chat en tiempo real

Aprenderemos los conceptos fundamentales de Node.js y cómo utilizarlos para construir un chat en tiempo real. Exploraremos herramientas como Express.js, Socket.io, Morgan y Turso para crear una aplicación de chat robusta y funcional.

por Eleomar Callejas Dorantes

¿Qué es Node.js y por qué es útil?

Entorno de Ejecución

Node.js es un entorno de ejecución de JavaScript basado en el motor V8 de Google, lo que permite ejecutar código JavaScript fuera del navegador.

Arquitectura Asíncrona

Node.js utiliza un modelo de programación asíncrona y sin bloqueo, lo que lo convierte en una herramienta ideal para aplicaciones en tiempo real como chats.

Ecosistema de Paquetes

Node.js cuenta con un amplio ecosistema de paquetes y bibliotecas que facilitan el desarrollo de aplicaciones web y de red.

Introducción a Express.js: creación de un servidor web

Servidor Web Rápido

Express.js es un framework web minimalista y de alto rendimiento que simplifica la creación de servidores web en Node.js.

Enrutamiento Flexible

Express.js proporciona una forma sencilla y eficiente de manejar solicitudes HTTP y definir rutas para la aplicación.

Middleware Versátil

Express.js permite agregar funcionalidad adicional a través de una amplia gama de middleware, como el manejo de sesiones y el análisis de datos.

Socket.io: comunicación en tiempo real entre cliente y servidor

1

Comunicación Bidireccional

Socket.io permite una comunicación en tiempo real entre el cliente y el servidor, lo que es esencial para un chat en vivo.

2

Manejo de Eventos

Socket.io facilita el envío y recepción de eventos personalizados entre el cliente y el servidor, lo que simplifica la lógica del chat.

3

Transporte Adaptativo

Socket.io se encarga de la detección y uso del mejor mecanismo de transporte disponible, como WebSockets o Polling HTTP.

Manejo de solicitudes HTTP con Express.js

1 Rutas y Verbos HTTP

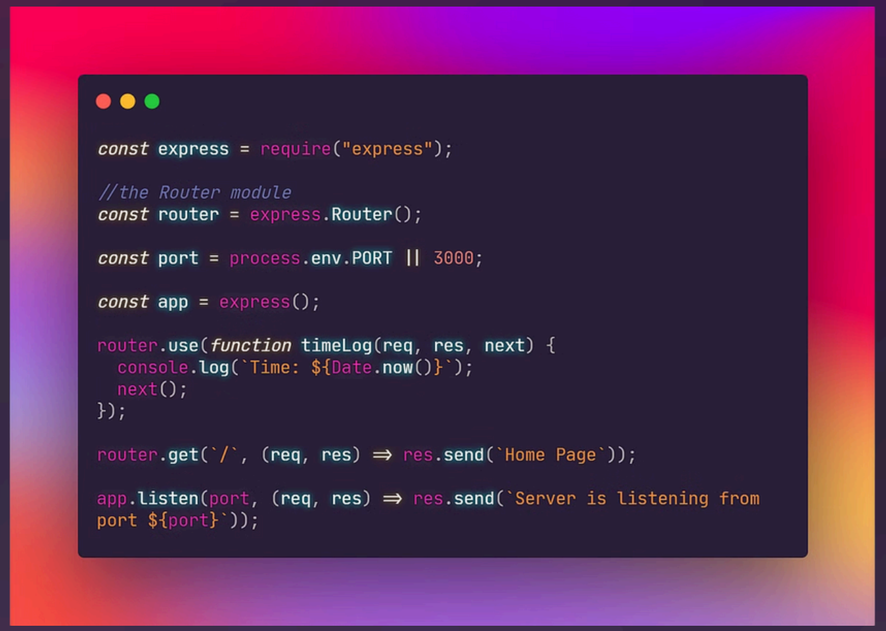
Express.js nos permite definir rutas y manejar los diferentes verbos HTTP (GET, POST, PUT, DELETE) de manera sencilla.

2 Middleware de Enrutamiento

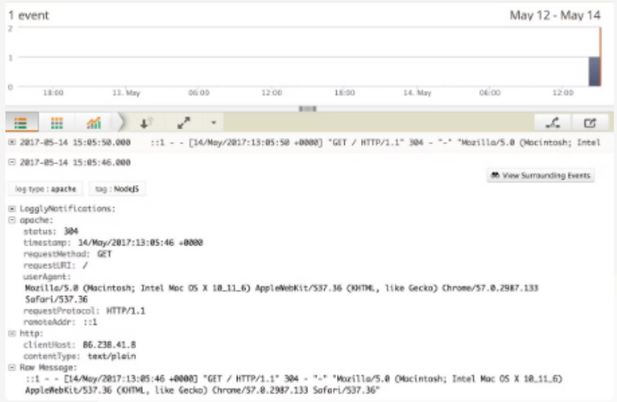
Express.js proporciona un sistema de middleware que nos permite agregar lógica personalizada a nuestras rutas, como autenticación o validación de datos.

3 Respuestas Personalizadas

Podemos enviar respuestas HTTP con el formato y contenido adecuado, como JSON, HTML o archivos estáticos.



```
const express = require("express");

//the Router module
const router = express.Router();

const port = process.env.PORT || 3000;

const app = express();

router.use(function timeLog(req, res, next) {
  console.log(`Time: ${Date.now()}`);
  next();
});

router.get(`/`, (req, res) => res.send(`Home Page`));

app.listen(port, (req, res) => res.send(`Server is listening from
port ${port}`));
```



Registro de actividad con Morgan

Registro de Solicitudes

Morgan es un middleware de registro de solicitudes HTTP que nos permite monitorear y analizar la actividad en nuestra aplicación.

Análisis de Rendimiento

Morgan nos proporciona información valiosa sobre los tiempos de respuesta y el rendimiento de nuestra aplicación.

Personalización Flexible

Podemos personalizar el formato y la información que se registra, lo que facilita el monitoreo y la depuración.

Base de datos en memoria con Turso

Almacenamiento Rápido	Turso es una base de datos en memoria que ofrece un acceso ultra rápido a los datos, lo que es ideal para un chat en tiempo real.
Persistencia Opcional	Turso también permite la persistencia de datos en el disco, lo que garantiza la seguridad de los mensajes del chat.
API Sencilla	Turso proporciona una API sencilla y bien documentada para interactuar con la base de datos desde Node.js.

Envío y recepción de mensajes en tiempo real

1

Cliente Envía Mensaje

El usuario escribe un mensaje en el chat y lo envía a través de Socket.io.

2

Servidor Recibe Mensaje

El servidor recibe el mensaje a través de Socket.io y lo almacena en la base de datos Turso.

3

Servidor Transmite Mensaje

El servidor utiliza Socket.io para transmitir el mensaje a todos los clientes conectados en tiempo real.

Autenticación y autorización de usuarios

Autenticación de Usuario

Implementaremos un sistema de autenticación seguro que permita a los usuarios iniciar sesión y utilizar el chat.

Autorización de Acceso

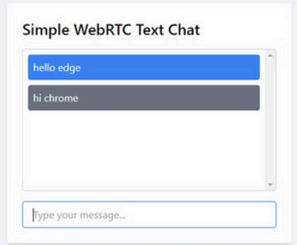
Restringiremos el acceso a ciertos recursos o funcionalidades del chat según el nivel de autorización del usuario.

Protección de Datos

Aseguraremos que los datos de los usuarios, como contraseñas y mensajes, se mantengan seguros y confidenciales.

Conclusión y próximos pasos

En esta presentación hemos explorado los fundamentos de Node.js y cómo utilizarlos para construir un chat en tiempo real. Ahora tienes las herramientas y conocimientos necesarios para continuar desarrollando y mejorando tu aplicación de chat.



Comandos para ejecutar una aplicación Node.js

Para ejecutar una aplicación Node.js, necesitarás tener Node.js instalado en tu sistema. Una vez instalado, puedes ejecutar tu aplicación con el siguiente comando en la línea de comandos:

```
node index.js
```

Donde index.js es el nombre del archivo que contiene el código fuente de tu aplicación. Si tu archivo principal tiene un nombre diferente, asegúrate de cambiarlo en el comando.

Instalación de dependencias

Node.js utiliza un administrador de paquetes llamado npm para instalar las dependencias de tus proyectos. Para instalar una dependencia, utiliza el siguiente comando:

```
npm install [nombre-de-la-dependencia]
```

Por ejemplo, para instalar la dependencia Express.js, ejecutarías:

```
npm install express
```

Para instalar Morgan, Socket.io y Turso puedes usar los siguientes comandos:

```
npm install morgan socket.io turso
```

Las dependencias instaladas se almacenarán en un archivo llamado package.json en la raíz de tu proyecto.