Manual técnico

Configuración inicial

El código comienza con la configuración inicial, donde se importan las bibliotecas necesarias y se crea una instancia de la aplicación Express.

```
const express = require('express');
const jwt = require('jsonwebtoken');
const axios = require('axios');

const app = express();
```

Middleware

Se utilizan middlewares de Express para analizar las solicitudes entrantes. En este caso, se utiliza `express.json()` para analizar el cuerpo de las solicitudes con formato JSON y `express.static()` para servir archivos estáticos, como por ejemplo, archivos HTML, imágenes, etc.

```
app.use(express.json());
app.use(express.static('public'));
```

Definición de usuarios y clave secreta

Se define un conjunto de usuarios con nombres de usuario y contraseñas, así como una clave secreta que se utilizará para firmar los tokens JWT.

```
const users = [
	{ username: 'admin', password: '1234' },
	{ username: 'user', password: 'abcd' }
];

const secret = 'secret';
```

Rutas

El código define varias rutas utilizando el enrutador de Express. Por ejemplo, la ruta `/login` maneja las solicitudes de inicio de sesión, la ruta `/search` maneja las solicitudes de búsqueda de usuarios en la API de GitHub, y la ruta `/contacto` redirige a los usuarios a una página de contacto.

```
// Ruta para el inicio de sesión
app.post('/login', (req, res) => {
	// ... lógica de inicio de sesión
});

// Ruta para la búsqueda de usuarios
app.get('/search', (req, res) => {
	// ... lógica de búsqueda de usuarios
});

// Ruta para redirigir a la página de contacto
app.get('/contacto', (req, res) => {
	res.redirect('http://localhost:3001/');
});
```

Inicio del servidor

El servidor se inicia y se pone a la escucha en el puerto 3000.

```
app.listen(3000, () => {
	console.log('Servidor escuchando en el puerto 3000');
```

```
});
```

Página HTML, idex.html

El codigo HTML incluye un formulario de inicio de sesión y una interfaz para buscar usuarios en la API de GitHub. Este código utiliza la biblioteca Vue.js para la manipulación del DOM y la realización de solicitudes HTTP.

Página de contacto

Configurando el proyecto

1. Se configura Mongo Atlas.
2. Se crea un nuevo directorio para el proyecto y se navega hasta él en la terminal.
3. Se Inicializa un nuevo proyecto Node.js ejecutando npm inity siguiendo las indicaciones.
4. Se instalan las dependencias requeridas ejecutando npm install express body-parser mongoose dotenv.
5. Crea un nuevo archivo llamado .env en el directorio raíz del proyecto y se establece el URI para su base de datos MongoDB.
6. Se crea un nuevo archivo llamado index.js

```
require('dotenv').config();
const express = require('express');
const bodyParser = require('body-parser');
const mongoose = require('mongoose');
const app = express();

app.use(bodyParser.urlencoded({ extended: true }));
app.use(express.static('public'));
```

```
mongoose.connect(process.env.MONGO_URI);

const PORT = process.env.PORT || 3001;
app.listen(PORT, () => {
  console.log(`Server running on port ${PORT}`);
});
```

Este código importa las dependencias requeridas, configura la aplicación Express, se conecta a la base de datos MongoDB e inicia el servidor.

Creando el formulario de contacto

1. Se crea un nuevo directorio llamado public en el directorio raíz del proyecto.
2. Se crea un nuevo archivo llamado index.html en public, con lo siguiente:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>Contact Form</title>
</head>
<body>
  <form action="/submit-form" method="POST">
    <label for="email">Email:</label>
    <input type="email" id="email" name="email" required>

    <label for="number">Number:</label>
    <input type="number" id="number" name="number" required>

    <label for="date">Date:</label>
    <input type="date" id="date" name="date" required>

    <label for="phone">Phone:</label>
    <input type="tel" id="phone" name="phone" required>

    <label for="message">Message:</label>
    <textarea id="message" name="message" required></textarea>
```

```
    <button type="submit">Submit</button>
  </form>
</body>
</html>
```

Esto crea un formulario HTML básico con campos para el correo electrónico, número, fecha, teléfono y mensaje del usuario.

Guardar la información de contacto en la base de datos.

```
const contactSchema = new mongoose.Schema({
  email: String,
  number: Number,
  date: Date,
  phone: String,
  message: String
});

const Contact = mongoose.model('Contact', contactSchema);

app.post('/submit-form', (req, res) => {
  const newContact = new Contact({
    email: req.body.email,
    number: req.body.number,
    date: req.body.date,
    phone: req.body.phone,
    message: req.body.message
  });

  newContact.save().then(() => console.log('Contact saved'));

  res.send('Data received and saved successfully.');
});
```

Esto define un esquema Mongoose para la información de contacto y crea un nuevo modelo llamado Contact. También se agrega una nueva ruta a la aplicación Express que maneja solicitudes POST para /submit-form. Cuando un usuario envía el formulario, el

código crea un nuevo Contactobjeto con los datos enviados y los guarda en la base de datos utilizando el save()método. Finalmente, el código envía una respuesta al usuario indicando que los datos fueron recibidos y guardados exitosamente.

Conclusiones

Esta es una aplicación web simple que demuestra cómo utilizar Node.js y Express para crear un servidor web, junto con las bibliotecas JWT y Axios para la autenticación de usuarios y la realización de solicitudes HTTP a una API externa, en este caso, la API de GitHub. Este tipo de aplicación es común en el desarrollo web moderno y proporciona una base solida para la creación de aplicaciones más complejas en el futuro.

Se mostró cómo establecer una conexión con una base de datos MongoDB, definir un esquema y un modelo de datos, y guardar la información del formulario en la base de datos. Además, se ha ilustrado el uso del paquete dotenv para la gestión de variables de entorno