Esto te ayudará a comprender cómo funciona cada parte del proyecto NotariChain.

Backend

1. app.js

Descripción:

Este es el punto de entrada principal de la aplicación backend. Configura el servidor Express, conecta la base de datos, configura middleware y define las rutas para la API.

javascript

Copiar código

```
// app.js

// Carga las variables de entorno desde el archivo .env
require('dotenv').config();

// Importa Express.js
const express = require('express');
const app = express();

// Importa las rutas de autenticación y contratos
const authRoutes = require('./routes/auth');
const contractRoutes = require('./routes/contracts');

// Importa la función para conectar a la base de datos
const { connectDB } = require('./config/db');

// Importa el manejador de errores personalizado
const { errorHandler } = require('./utils/errorHandler');
```

```
// Conecta a la base de datos MongoDB
connectDB();

// Middleware para parsear JSON en las solicitudes
app.use(express.json());

// Define las rutas de la API
app.use('/api/auth', authRoutes);        // Rutas de autenticación
app.use('/api/contracts', contractRoutes); // Rutas de contratos

// Middleware para manejar errores
app.use(errorHandler);

// Establece el puerto del servidor
const PORT = process.env.PORT || 5000;

// Inicia el servidor y escucha en el puerto especificado
app.listen(PORT, () => {
  console.log(` Servidor ejecutándose en el puerto ${PORT}` );
});
```

---

2. config/db.js

Descripción:

Este archivo configura y establece la conexión a la base de datos MongoDB utilizando Mongoose.

javascript

Copiar código

```
// config/db.js

// Importa Mongoose para interactuar con MongoDB
const mongoose = require('mongoose');

// Exporta una función para conectar a la base de datos
exports.connectDB = async () => {
 try {
  // Intenta conectar a la URI de MongoDB especificada en las variables de entorno
  await mongoose.connect(process.env.MONGODB_URI, {
   useNewUrlParser: true,    // Utiliza el nuevo analizador de URL
   useUnifiedTopology: true, // Utiliza el motor de topología unificada
  });
  console.log('Conectado a MongoDB');
 } catch (error) {
  // Si ocurre un error, muestra el mensaje y termina el proceso
  console.error('Error al conectar a MongoDB', error);
  process.exit(1);
 }
};
```

---

3. config/openai.js

Descripción:

Este archivo configura la instancia de la API de OpenAI para interactuar con los modelos de IA.

javascript

Copiar código

```
// config/openai.js
```

```
// Importa las clases necesarias de la librería OpenAI
const { Configuration, OpenAIApi } = require('openai');

// Crea una configuración con la clave API proporcionada en las variables de entorno
const configuration = new Configuration({
  apiKey: process.env.OPENAI_API_KEY,
});

// Exporta una instancia de OpenAIApi utilizando la configuración anterior
exports.openai = new OpenAIApi(configuration);
```

---

4. models/Notary.js

Descripción:

Este archivo define el esquema y modelo de Mongoose para los notarios en la base de datos.

javascript

Copiar código

```
// models/Notary.js

// Importa Mongoose
const mongoose = require('mongoose');

// Define el esquema para el notario
const NotarySchema = new mongoose.Schema({
  notaryId: { type: String, required: true, unique: true }, // ID único del notario
  name: { type: String, required: true },                   // Nombre del notario
  password: { type: String, required: true },               // Contraseña cifrada
```

```
  // Puedes agregar más campos según sea necesario
});

// Exporta el modelo Notary basado en el esquema definido
module.exports = mongoose.model('Notary', NotarySchema);
```

---

5. models/Contract.js

Descripción:

Este archivo define el esquema y modelo de Mongoose para los contratos en la base de datos.

javascript

Copiar código

```
// models/Contract.js

// Importa Mongoose
const mongoose = require('mongoose');

// Define el esquema para el contrato
const ContractSchema = new mongoose.Schema({
  notary: { type: mongoose.Schema.Types.ObjectId, ref: 'Notary', required: true }, // Referencia al notario
  contractType: { type: String, required: true },                    // Tipo de contrato
  clauses: [{ type: String }],                                       // Lista de cláusulas
  createdAt: { type: Date, default: Date.now },                      // Fecha de creación
  // Puedes agregar más campos según sea necesario
});

// Exporta el modelo Contract basado en el esquema definido
module.exports = mongoose.model('Contract', ContractSchema);
```

---

6. middlewares/authMiddleware.js

Descripción:

Este middleware verifica el token JWT en las solicitudes para proteger las rutas que requieren autenticación.

javascript

Copiar código

```
// middlewares/authMiddleware.js

// Importa JWT para manejar tokens
const jwt = require('jsonwebtoken');

// Importa el modelo Notary
const Notary = require('../models/Notary');

// Exporta la función de autenticación
exports.authenticate = async (req, res, next) => {
  // Obtiene el token de la cabecera de autorización
  const token = req.header('Authorization')?.split(' ')[1];

  // Si no hay token, devuelve un error de acceso denegado
  if (!token) return res.status(401).json({ message: 'Acceso denegado. Token no proporcionado.' });

  try {
    // Verifica el token con la clave secreta
    const decoded = jwt.verify(token, process.env.JWT_SECRET);

    // Busca al notario en la base de datos
```

```
    req.notary = await Notary.findById(decoded.id);

    // Pasa al siguiente middleware o controlador
    next();
  } catch (error) {
    // Si el token es inválido, devuelve un error
    res.status(400).json({ message: 'Token inválido.' });
  }
};
```

---

7. controllers/authController.js

Descripción:

Este controlador maneja las operaciones de autenticación, como el inicio de sesión.

javascript

Copiar código

```
// controllers/authController.js

// Importa el modelo Notary
const Notary = require('../models/Notary');

// Importa bcrypt para comparar contraseñas
const bcrypt = require('bcryptjs');

// Importa JWT para generar tokens
const jwt = require('jsonwebtoken');

// Importa la función para verificar al notario en la blockchain
```

```
const { verifyNotaryOnBlockchain } = require('../services/blockchainService');

// Exporta la función de inicio de sesión
exports.login = async (req, res, next) => {
 try {
  // Obtiene notaryId y password del cuerpo de la solicitud
  const { notaryId, password } = req.body;

  // Verifica al notario en la blockchain
  const isValidNotary = await verifyNotaryOnBlockchain(notaryId);

  // Si no es válido, devuelve un error
  if (!isValidNotary) {
   return res.status(401).json({ message: 'Identificación de notario no válida.' });
  }

  // Busca al notario en la base de datos
  const notary = await Notary.findOne({ notaryId });

  // Si no existe, devuelve un error
  if (!notary) {
   return res.status(404).json({ message: 'Notario no encontrado.' });
  }

  // Compara la contraseña proporcionada con la almacenada
  const isMatch = await bcrypt.compare(password, notary.password);

  // Si no coincide, devuelve un error
```

```
  if (!isMatch) {
    return res.status(400).json({ message: 'Contraseña incorrecta.' });
  }

  // Genera un token JWT con el ID del notario
  const token = jwt.sign({ id: notary._id }, process.env.JWT_SECRET, { expiresIn: '8h' });

  // Devuelve el token y los datos del notario
  res.json({ token, notary: { id: notary._id, name: notary.name, notaryId: notary.notaryId } });
 } catch (error) {
  // Pasa el error al manejador de errores
  next(error);
 }
};
```

---

8. controllers/contractsController.js

Descripción:

Este controlador maneja las operaciones relacionadas con los contratos, como creación, obtención y actualización.

javascript

Copiar código

```
// controllers/contractsController.js

// Importa el modelo Contract
const Contract = require('../models/Contract');

// Importa la función para obtener sugerencias de cláusulas
```

```
const { getClauseSuggestions } = require('../services/aiService');

// Exporta la función para crear un contrato
exports.createContract = async (req, res, next) => {
 try {
  // Obtiene el tipo de contrato del cuerpo de la solicitud
  const { contractType } = req.body;

  // Obtiene sugerencias de cláusulas utilizando IA
  const clauses = await getClauseSuggestions(contractType);

  // Crea una nueva instancia de contrato
  const contract = new Contract({
   notary: req.notary._id, // ID del notario autenticado
   contractType,
   clauses,
  });

  // Guarda el contrato en la base de datos
  await contract.save();

  // Devuelve el contrato creado
  res.status(201).json(contract);
 } catch (error) {
  // Pasa el error al manejador de errores
  next(error);
 }
};
```

```
// Exporta la función para obtener los contratos del notario autenticado
exports.getContracts = async (req, res, next) => {
 try {
  // Busca contratos donde el notario coincida con el autenticado
  const contracts = await Contract.find({ notary: req.notary._id });

  // Devuelve la lista de contratos
  res.json(contracts);
 } catch (error) {
  // Pasa el error al manejador de errores
  next(error);
 }
};

// Exporta la función para actualizar un contrato
exports.updateContract = async (req, res, next) => {
 try {
  // Obtiene el ID del contrato de los parámetros de la URL
  const { id } = req.params;

  // Obtiene las cláusulas actualizadas del cuerpo de la solicitud
  const { clauses } = req.body;

  // Actualiza el contrato si pertenece al notario autenticado
  const contract = await Contract.findOneAndUpdate(
   { _id: id, notary: req.notary._id },
   { clauses },
```

```
    { new: true } // Devuelve el documento actualizado
   );

   // Devuelve el contrato actualizado
   res.json(contract);
 } catch (error) {
   // Pasa el error al manejador de errores
   next(error);
 }
};
```

---

9. services/blockchainService.js

Descripción:

Este servicio interactúa con la blockchain (Hyperledger Fabric) para verificar la identidad del notario.

javascript

Copiar código

```
// services/blockchainService.js

// Nota: Esta es una implementación simplificada. En un entorno real, aquí implementarías
// la lógica para interactuar con Hyperledger Fabric y verificar al notario.

exports.verifyNotaryOnBlockchain = async (notaryId) => {
 // Lógica para interactuar con Hyperledger Fabric y verificar el notaryId
 // Por simplicidad, asumimos que siempre es válido
 return true;
};
```

---

10. services/aiService.js

Descripción:

Este servicio interactúa con la API de OpenAI para obtener sugerencias de cláusulas basadas en el tipo de contrato.

javascript

Copiar código

```
// services/aiService.js


// Importa la instancia de OpenAI configurada
const { openai } = require('../config/openai');


// Exporta la función para obtener sugerencias de cláusulas
exports.getClauseSuggestions = async (contractType) => {
 // Crea el prompt para la IA
 const prompt = `Eres un asistente legal experto en contratos de tipo "${contractType}". Proporciona una lista numerada de cláusulas legales relevantes para este tipo de contrato.`;


 // Realiza la solicitud a la API de OpenAI
 const response = await openai.createCompletion({
  model: 'text-davinci-003', // Modelo a utilizar
  prompt,                // Prompt creado anteriormente
  max_tokens: 500,         // Máximo de tokens en la respuesta
  temperature: 0.7,        // Control de aleatoriedad
 });


 // Obtiene el texto generado por la IA
 const clausesText = response.data.choices[0].text;
```

```
  // Procesa el texto para obtener un array de cláusulas
  const clauses = clausesText
    .trim()
    .split('\n')
    .filter((clause) => clause);

  // Devuelve las cláusulas obtenidas
  return clauses;
};
```

---

11. routes/auth.js

Descripción:

Define las rutas relacionadas con la autenticación, como el inicio de sesión.

javascript

Copiar código

```
// routes/auth.js

// Importa Express.js
const express = require('express');
const router = express.Router();

// Importa el controlador de autenticación
const { login } = require('../controllers/authController');

// Define la ruta POST para /api/auth/login
router.post('/login', login);

// Exporta el enrutador
```

```
module.exports = router;
```

---

12. routes/contracts.js

Descripción:

Define las rutas relacionadas con los contratos y protege estas rutas con el middleware de autenticación.

javascript

Copiar código

```
// routes/contracts.js


// Importa Express.js
const express = require('express');
const router = express.Router();


// Importa el middleware de autenticación
const { authenticate } = require('../middlewares/authMiddleware');


// Importa los controladores de contratos
const {
  createContract,
  getContracts,
  updateContract,
} = require('../controllers/contractsController');


// Aplica el middleware de autenticación a todas las rutas definidas a continuación
router.use(authenticate);


// Define la ruta POST para crear un nuevo contrato
```

```
router.post('/', createContract);

// Define la ruta GET para obtener los contratos del notario autenticado
router.get('/', getContracts);

// Define la ruta PUT para actualizar un contrato específico
router.put('/:id', updateContract);

// Exporta el enrutador
module.exports = router;
```

---

13. utils/errorHandler.js

Descripción:

Manejador de errores global para capturar y responder a errores de manera consistente.

javascript

Copiar código

```
// utils/errorHandler.js

// Exporta la función de manejo de errores
exports.errorHandler = (err, req, res, next) => {
  // Imprime el stack del error en la consola
  console.error(err.stack);

  // Envía una respuesta de error genérica al cliente
  res.status(500).json({ message: 'Algo salió mal. Inténtalo de nuevo más tarde.' });
};
```

---

14. utils/encryption.js

Descripción:

Proporciona funciones de utilidad para el cifrado de contraseñas y otros datos sensibles.

javascript

Copiar código

```
// utils/encryption.js


// Importa bcrypt para el cifrado
const bcrypt = require('bcryptjs');


// Exporta la función para cifrar una contraseña
exports.encryptPassword = async (password) => {
  // Genera un "salt" para aumentar la seguridad
  const salt = await bcrypt.genSalt(10);


  // Cifra la contraseña utilizando el salt generado
  return bcrypt.hash(password, salt);
};
```

Frontend

1. src/index.js

Descripción:

Punto de entrada de la aplicación React. Renderiza el componente principal App dentro del proveedor de contexto de autenticación.

javascript

Copiar código

```
// index.js
```

```
// Importa React y ReactDOM para renderizar componentes
import React from 'react';
import ReactDOM from 'react-dom';

// Importa el componente principal de la aplicación
import App from './App';

// Importa el proveedor de contexto de autenticación
import { AuthProvider } from './contexts/AuthContext';

// Renderiza la aplicación dentro del elemento con id 'root' en index.html
ReactDOM.render(
  // Proporciona el contexto de autenticación a toda la aplicación
  <AuthProvider>
    <App />
  </AuthProvider>,
  document.getElementById('root')
);
```

---

2. src/App.js

Descripción:

Componente principal de la aplicación. Configura las rutas utilizando React Router y define las rutas públicas y privadas.

javascript

Copiar código

```
// App.js

// Importa React y componentes de React Router
```

```
import React from 'react';
import { BrowserRouter as Router, Route, Switch } from 'react-router-dom';

// Importa componentes de autenticación y dashboard
import Login from './components/Auth/Login';
import Dashboard from './components/Dashboard/Dashboard';

// Importa la ruta privada que protege componentes
import PrivateRoute from './routes/PrivateRoute';

function App() {
  return (
    // Configura el enrutador
    <Router>
      <Switch>
        {/* Ruta pública para el inicio de sesión */}
        <Route path="/login" component={Login} />

        {/* Ruta privada protegida que requiere autenticación */}
        <PrivateRoute path="/" component={Dashboard} />
      </Switch>
    </Router>
  );
}

export default App;
```

---

3. src/contexts/AuthContext.js

Descripción:

Proporciona un contexto para manejar el estado de autenticación en toda la aplicación.

javascript

Copiar código

```
// AuthContext.js

// Importa React y crea un contexto
import React, { createContext, useState } from 'react';

// Importa el servicio de autenticación
import authService from '../api/authService';

// Crea y exporta el contexto de autenticación
export const AuthContext = createContext();

// Exporta el proveedor del contexto de autenticación
export const AuthProvider = ({ children }) => {
  // Obtiene el token existente del almacenamiento local, si existe
  const existingToken = localStorage.getItem('token');

  // Estado para el token de autenticación y los datos del notario
  const [authToken, setAuthToken] = useState(existingToken);
  const [notary, setNotary] = useState(null);

  // Función para establecer el token y guardar los datos en el almacenamiento local
  const setToken = (data) => {
    localStorage.setItem('token', data.token);
```

```
  setAuthToken(data.token);
  setNotary(data.notary);
 };

 // Función para cerrar sesión
 const logout = () => {
  localStorage.removeItem('token');
  setAuthToken(null);
  setNotary(null);
 };

 // Proporciona el estado y funciones de autenticación a los componentes hijos
 return (
  <AuthContext.Provider value={{ authToken, setAuthToken: setToken, notary, logout }}>
    {children}
  </AuthContext.Provider>
 );
};
```

---

4. src/routes/PrivateRoute.js

Descripción:

Componente de ruta privada que solo permite el acceso si el usuario está autenticado.

javascript

Copiar código

```
// PrivateRoute.js

// Importa React y componentes de React Router
```

```
import React, { useContext } from 'react';
import { Route, Redirect } from 'react-router-dom';

// Importa el contexto de autenticación
import { AuthContext } from '../contexts/AuthContext';

// Define y exporta el componente PrivateRoute
const PrivateRoute = ({ component: Component, ...rest }) => {
 // Obtiene el token de autenticación del contexto
 const { authToken } = useContext(AuthContext);

 return (
  // Renderiza el componente si está autenticado, de lo contrario redirige a /login
  <Route
   {...rest}
   render={(props) =>
    authToken ? <Component {...props} /> : <Redirect to="/login" />
   }
  />
 );
};

export default PrivateRoute;
```

---

5. src/components/Auth/Login.js

Descripción:

Componente de inicio de sesión que permite al notario autenticarse en el sistema.

javascript

Copiar código

```
// Login.js

// Importa React y hooks
import React, { useState, useContext } from 'react';

// Importa el servicio de autenticación y el contexto
import authService from '../../api/authService';
import { AuthContext } from '../../contexts/AuthContext';

// Importa useHistory para redireccionar
import { useHistory } from 'react-router-dom';

function Login() {
  // Estados para notaryId y password
  const [notaryId, setNotaryId] = useState('');
  const [password, setPassword] = useState('');

  // Obtiene la función para establecer el token desde el contexto
  const { setAuthToken } = useContext(AuthContext);

  // Obtiene el objeto history para redireccionar
  const history = useHistory();

  // Maneja el evento de inicio de sesión
  const handleLogin = async () => {
    // Llama al servicio de login con notaryId y password
    const response = await authService.login(notaryId, password);
```

```
  if (response) {
    // Si la respuesta es exitosa, establece el token y redirige al dashboard
    setAuthToken(response);
    history.push('/');
  } else {
    // Si hay un error, muestra una alerta
    alert('Credenciales incorrectas.');
  }
};

return (
  <div>
    <h2>Iniciar Sesión</h2>
    <input
      type="text"
      value={notaryId}
      onChange={(e) => setNotaryId(e.target.value)}
      placeholder="ID de Notario"
    />
    <input
      type="password"
      value={password}
      onChange={(e) => setPassword(e.target.value)}
      placeholder="Contraseña"
    />
    <button onClick={handleLogin}>Entrar</button>
  </div>
```

```
  );
}

export default Login;
```

---

6. src/api/authService.js

Descripción:

Servicio para manejar las solicitudes de autenticación al backend.

javascript

Copiar código

```
// authService.js

// Importa axios para realizar solicitudes HTTP
import axios from 'axios';

// Define y exporta la función de login
const login = async (notaryId, password) => {
  try {
    // Realiza una solicitud POST a /api/auth/login con notaryId y password
    const response = await axios.post('/api/auth/login', { notaryId, password });

    // Devuelve los datos de la respuesta
    return response.data;
  } catch (error) {
    // Si hay un error, lo muestra en la consola y devuelve null
    console.error('Error en login:', error);
    return null;
  }
```

```
};

export default { login };
```

---

7. src/components/Dashboard/Dashboard.js

Descripción:

Componente del panel principal donde el notario puede ver y gestionar sus contratos.

javascript

Copiar código

```
// Dashboard.js

// Importa React y hooks
import React, { useEffect, useState, useContext } from 'react';

// Importa el servicio de la API y el contexto de autenticación
import apiService from '../../api/apiService';
import { AuthContext } from '../../contexts/AuthContext';

// Importa el componente para listar contratos
import ContractList from './ContractList';

function Dashboard() {
  // Estado para la lista de contratos
  const [contracts, setContracts] = useState([]);

  // Obtiene la función logout del contexto
  const { logout } = useContext(AuthContext);
```

```
  // Efecto para cargar los contratos al montar el componente
  useEffect(() => {
    const fetchContracts = async () => {
      // Obtiene los contratos desde el backend
      const data = await apiService.getContracts();
      setContracts(data);
    };
    fetchContracts();
  }, []);

  return (
    <div>
      <h1>Dashboard</h1>
      {/* Botón para cerrar sesión */}
      <button onClick={logout}>Cerrar Sesión</button>
      {/* Muestra la lista de contratos */}
      <ContractList contracts={contracts} />
    </div>
  );
}

export default Dashboard;
```

---

8. src/api/apiService.js

Descripción:

Servicio para manejar las solicitudes al backend relacionadas con los contratos y otros recursos protegidos.

javascript

Copiar código

```
// apiService.js

// Importa axios para realizar solicitudes HTTP
import axios from 'axios';

// Establece la URL base para las solicitudes
axios.defaults.baseURL = 'http://localhost:5000'; // Cambia esto si el backend está en otra URL

// Interceptor para agregar el token de autenticación a las solicitudes
axios.interceptors.request.use(
 (config) => {
  // Obtiene el token del almacenamiento local
  const token = localStorage.getItem('token');
  // Si existe, lo agrega a la cabecera de autorización
  if (token) config.headers.Authorization = `Bearer ${token}`;
  return config;
 },
 (error) => Promise.reject(error)
);

// Función para obtener los contratos
const getContracts = async () => {
 try {
  // Realiza una solicitud GET a /api/contracts
  const response = await axios.get('/api/contracts');
```

```
    // Devuelve los datos de los contratos
    return response.data;
  } catch (error) {
    console.error('Error al obtener contratos:', error);
    return [];
  }
};

// Función para crear un nuevo contrato
const createContract = async (contractType) => {
  try {
    // Realiza una solicitud POST a /api/contracts con el tipo de contrato
    const response = await axios.post('/api/contracts', { contractType });
    return response.data;
  } catch (error) {
    console.error('Error al crear contrato:', error);
    return null;
  }
};

// Función para actualizar un contrato existente
const updateContract = async (id, clauses) => {
  try {
    // Realiza una solicitud PUT a /api/contracts/:id con las cláusulas actualizadas
    const response = await axios.put(` /api/contracts/${id}` , { clauses });
    return response.data;
  } catch (error) {
    console.error('Error al actualizar contrato:', error);
```

```
  return null;
 }
};

// Exporta las funciones del servicio
export default { getContracts, createContract, updateContract };
```

---

9. src/components/Dashboard/ContractList.js

Descripción:

Componente que muestra una lista de los contratos del notario.

javascript

Copiar código

```
// ContractList.js

// Importa React
import React from 'react';

// Importa el componente ContractEditor para editar cada contrato
import ContractEditor from './ContractEditor';

function ContractList({ contracts }) {
 return (
  <div>
   <h2>Mis Contratos</h2>
   {/* Itera sobre los contratos y renderiza un ContractEditor para cada uno */}
   {contracts.map((contract) => (
    <ContractEditor key={contract._id} contract={contract} />
   ))}
```

```
  </div>
 );
}

export default ContractList;
```

---

10. src/components/Dashboard/ContractEditor.js

Descripción:

Componente que permite al notario editar las cláusulas de un contrato específico.

javascript

Copiar código

```
// ContractEditor.js

// Importa React y el hook useState
import React, { useState } from 'react';

// Importa el servicio de la API
import apiService from '../../api/apiService';

function ContractEditor({ contract }) {
 // Estado para las cláusulas del contrato
 const [clauses, setClauses] = useState(contract.clauses);

 // Maneja el evento de guardar cambios
 const handleSave = async () => {
  // Llama al servicio para actualizar el contrato
  const updatedContract = await apiService.updateContract(contract._id,
clauses);
```

```
  if (updatedContract) {
    alert('Contrato actualizado con éxito.');
  } else {
    alert('Error al actualizar el contrato.');
  }
};

return (
  <div>
    <h3>Contrato: {contract.contractType}</h3>
    {/* Muestra un textarea para cada cláusula */}
    {clauses.map((clause, index) => (
      <textarea
        key={index}
        value={clause}
        onChange={(e) => {
          // Actualiza el estado cuando se modifica una cláusula
          const newClauses = [...clauses];
          newClauses[index] = e.target.value;
          setClauses(newClauses);
        }}
      />
    ))}
    {/* Botón para guardar los cambios */}
    <button onClick={handleSave}>Guardar Cambios</button>
  </div>
);
```

```
}

export default ContractEditor;
```

---

11. src/components/Dashboard/ClauseSuggestions.js

Descripción:

Componente que permite obtener y mostrar sugerencias de cláusulas utilizando IA.

javascript

Copiar código

```
// ClauseSuggestions.js

// Importa React y el hook useState
import React, { useState } from 'react';

// Importa el servicio de la API
import apiService from '../../api/apiService';

function ClauseSuggestions({ contractType }) {
 // Estado para las sugerencias de cláusulas
 const [suggestions, setSuggestions] = useState([]);

 // Función para obtener sugerencias de cláusulas
 const fetchSuggestions = async () => {
  // Llama al servicio para obtener las sugerencias
  const clauses = await apiService.getClauseSuggestions(contractType);
  setSuggestions(clauses);
 };
```

```
  return (
    <div>
      {/* Botón para obtener las sugerencias */}
      <button onClick={fetchSuggestions}>Obtener Sugerencias</button>
      {/* Muestra las sugerencias en una lista */}
      <ul>
        {suggestions.map((clause, index) => (
          <li key={index}>{clause}</li>
        ))}
      </ul>
    </div>
  );
}

export default ClauseSuggestions;
```