Backend

1. Archivo Principal app.js

Este es el punto de entrada de tu aplicación backend. Configura Express, las rutas, y sincroniza los modelos con la base de datos PostgreSQL.

javascript

Copiar código

```
// app.js


require('dotenv').config();
const express = require('express');
const cors = require('cors');
const app = express();
const { errorHandler } = require('./utils/errorHandler');
const authRoutes = require('./routes/auth');
const contractRoutes = require('./routes/contracts');
const sequelize = require('./config/database');


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


// Habilitar CORS
app.use(
  cors({
    origin: 'http://localhost:3000', // Cambia esto si tu frontend está en otro dominio
    methods: ['GET', 'POST', 'PUT', 'DELETE'],
    credentials: true,
  })
);
```

```
// Rutas
app.use('/api/auth', authRoutes);
app.use('/api/contracts', contractRoutes);

// Manejador de errores
app.use(errorHandler);

// Sincronizar modelos y iniciar el servidor
const PORT = process.env.PORT || 5000;

sequelize.sync({ alter: true })
  .then(() => {
    console.log('Modelos sincronizados con la base de datos.');
    app.listen(PORT, () => {
      console.log(` Servidor ejecutándose en el puerto ${PORT}` );
    });
  })
  .catch((error) => {
    console.error('Error al sincronizar los modelos con la base de datos:', error);
  });
```

---

2. Configuración de la Base de Datos config/database.js

Configura Sequelize para conectarse a tu base de datos PostgreSQL utilizando variables de entorno.

javascript

Copiar código

```
// config/database.js
```

```
const { Sequelize } = require('sequelize');

const {
  DB_HOST,
  DB_USER,
  DB_PASSWORD,
  DB_NAME,
  DB_PORT,
} = process.env;

const sequelize = new Sequelize(DB_NAME, DB_USER, DB_PASSWORD, {
  host: DB_HOST,
  port: DB_PORT || 5432,
  dialect: 'postgres',
  logging: false, // Cambia a true para ver las consultas SQL en la consola
});

module.exports = sequelize;
```

Archivo .env

Asegúrate de tener las variables de entorno configuradas:

makefile

Copiar código

```
DB_HOST=localhost
DB_USER=tu_usuario
DB_PASSWORD=tu_contraseña
DB_NAME=notarichain
DB_PORT=5432
```

JWT_SECRET=tu_secreto_jwt

OPENAI_API_KEY=tu_clave_api_de_openai

---

3. Modelos

3.1. Modelo Notary

Define el modelo de Notario con Sequelize.

javascript

Copiar código

```
// models/Notary.js

const { DataTypes } = require('sequelize');
const sequelize = require('../config/database');

// Define el modelo Notary
const Notary = sequelize.define('Notary', {
 notaryId: {
  type: DataTypes.STRING,
  allowNull: false,
  unique: true,
 },
 name: {
  type: DataTypes.STRING,
  allowNull: false,
 },
 password: {
  type: DataTypes.STRING,
  allowNull: false,
```

```
  },
});

module.exports = Notary;
```

3.2. Modelo Contract

Define el modelo de Contrato y establece la relación con Notary.

javascript

Copiar código

```
// models/Contract.js

const { DataTypes } = require('sequelize');
const sequelize = require('../config/database');
const Notary = require('./Notary');

// Define el modelo Contract
const Contract = sequelize.define('Contract', {
  contractType: {
    type: DataTypes.STRING,
    allowNull: false,
  },
  clauses: {
    type: DataTypes.ARRAY(DataTypes.TEXT),
    allowNull: true,
  },
});

// Establece las relaciones
Notary.hasMany(Contract, {
```

```
  foreignKey: 'notaryId',
  as: 'contracts',
});

Contract.belongsTo(Notary, {
  foreignKey: 'notaryId',
  as: 'notary',
});

module.exports = Contract;
```

---

4. Controladores

4.1. Controlador de Autenticación controllers/authController.js

Maneja el registro e inicio de sesión de notarios.

javascript

Copiar código

```
// controllers/authController.js

const Notary = require('../models/Notary');
const bcrypt = require('bcryptjs');
const jwt = require('jsonwebtoken');
const { verifyNotaryOnBlockchain } = require('../services/blockchainService');

// Registrar un nuevo notario
exports.register = async (req, res, next) => {
  try {
    const { notaryId, name, password } = req.body;
```

```
    // Verificar si el notario ya existe
    let notary = await Notary.findOne({ where: { notaryId } });
    if (notary) {
      return res.status(400).json({ message: 'El notario ya está registrado.' });
    }

    // Cifrar la contraseña
    const hashedPassword = await bcrypt.hash(password, 10);

    // Crear nuevo notario
    notary = await Notary.create({
      notaryId,
      name,
      password: hashedPassword,
    });

    res.status(201).json({ message: 'Notario registrado exitosamente.' });
  } catch (error) {
    next(error);
  }
};

// Iniciar sesión
exports.login = async (req, res, next) => {
  try {
    const { notaryId, password } = req.body;

    console.log('Intentando iniciar sesión con notaryId:', notaryId);
```

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

  if (!isValidNotary) {
    return res.status(401).json({ message: 'Identificación de notario no válida.' });
  }

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

  console.log('Resultado de la búsqueda del notario:', notary);

  if (!notary) {
    return res.status(404).json({ message: 'Notario no encontrado.' });
  }

  // Comparar contraseñas
  const isMatch = await bcrypt.compare(password, notary.password);

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

  // Generar token JWT
  const token = jwt.sign({ id: notary.id }, process.env.JWT_SECRET, { expiresIn: '8h'
});
```

```
  res.json({ token, notary: { id: notary.id, name: notary.name, notaryId:
notary.notaryId } });
 } catch (error) {
  next(error);
 }
};
```

4.2. Controlador de Contratos controllers/contractsController.js

Maneja la creación, obtención y actualización de contratos.

javascript

Copiar código

```
// controllers/contractsController.js

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

// Crear un nuevo contrato
exports.createContract = async (req, res, next) => {
 try {
  const { contractType } = req.body;

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

  // Crear el contrato
  const contract = await Contract.create({
   notaryId: req.notary.id,
   contractType,
   clauses,
```

```
    });

    res.status(201).json(contract);
  } catch (error) {
    next(error);
  }
};

// Obtener los contratos del notario autenticado
exports.getContracts = async (req, res, next) => {
  try {
    const contracts = await Contract.findAll({
      where: { notaryId: req.notary.id },
    });

    res.json(contracts);
  } catch (error) {
    next(error);
  }
};

// Actualizar un contrato
exports.updateContract = async (req, res, next) => {
  try {
    const { id } = req.params;
    const { clauses } = req.body;

    // Buscar el contrato y actualizarlo si pertenece al notario
```

```
    const contract = await Contract.findOne({
      where: { id, notaryId: req.notary.id },
    });

    if (!contract) {
      return res.status(404).json({ message: 'Contrato no encontrado.' });
    }

    contract.clauses = clauses;
    await contract.save();

    res.json(contract);
  } catch (error) {
    next(error);
  }
};
```

---

5. Middleware de Autenticación middlewares/authMiddleware.js

Protege las rutas que requieren autenticación.

javascript

Copiar código

```
// middlewares/authMiddleware.js

const jwt = require('jsonwebtoken');
const Notary = require('../models/Notary');

exports.authenticate = async (req, res, next) => {
  const token = req.header('Authorization')?.split(' ')[1];
```

```
  if (!token) return res.status(401).json({ message: 'Acceso denegado. Token no proporcionado.' });

  try {
    const decoded = jwt.verify(token, process.env.JWT_SECRET);
    const notary = await Notary.findByPk(decoded.id);

    if (!notary) {
      return res.status(401).json({ message: 'Notario no encontrado.' });
    }

    req.notary = notary;
    next();
  } catch (error) {
    res.status(400).json({ message: 'Token inválido.' });
  }
};
```

---

6. Rutas

6.1. Rutas de Autenticación routes/auth.js

javascript

Copiar código

```
// routes/auth.js

const express = require('express');
const router = express.Router();
const { login, register } = require('../controllers/authController');
```

```
router.post('/login', login);
router.post('/register', register);

module.exports = router;
```

6.2. Rutas de Contratos routes/contracts.js

javascript

Copiar código

```
// routes/contracts.js

const express = require('express');
const router = express.Router();
const { authenticate } = require('../middlewares/authMiddleware');
const {
  createContract,
  getContracts,
  updateContract,
} = require('../controllers/contractsController');

router.use(authenticate);

router.post('/', createContract);
router.get('/', getContracts);
router.put('/:id', updateContract);

module.exports = router;
```

---

7. Utilidades

7.1. Cifrado utils/encryption.js

Funciones para cifrar y comparar contraseñas.

javascript

Copiar código

```
// utils/encryption.js

const bcrypt = require('bcryptjs');

exports.encryptPassword = async (password) => {
 const salt = await bcrypt.genSalt(10);
 return bcrypt.hash(password, salt);
};

exports.comparePassword = async (password, hashedPassword) => {
 return bcrypt.compare(password, hashedPassword);
};
```

7.2. Manejador de Errores utils/errorHandler.js

javascript

Copiar código

```
// utils/errorHandler.js

exports.errorHandler = (err, req, res, next) => {
 console.error(err.stack);
 res.status(500).json({ message: 'Algo salió mal. Inténtalo de nuevo más tarde.' });
};
```

---

8. Servicios

8.1. Servicio de IA services/aiService.js

Mantiene la integración con OpenAI.

javascript

Copiar código

```
// services/aiService.js

const { openai } = require('../config/openai');

exports.getClauseSuggestions = async (contractType) => {
 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.`;

 const response = await openai.createCompletion({
  model: 'text-davinci-003',
  prompt,
  max_tokens: 500,
  temperature: 0.7,
 });

 const clausesText = response.data.choices[0].text;
 const clauses = clausesText
  .trim()
  .split('\n')
  .filter((clause) => clause);

 return clauses;
};
```

8.2. Servicio de Blockchain services/blockchainService.js

javascript

Copiar código

```
// services/blockchainService.js

// Implementación simplificada

exports.verifyNotaryOnBlockchain = async (notaryId) => {
  // Aquí implementarías la lógica para interactuar con Hyperledger Fabric
  // Por simplicidad, asumimos que siempre es válido
  return true;
};
```

---

9. Configuración de OpenAI config/openai.js

javascript

Copiar código

```
// config/openai.js

const { Configuration, OpenAIApi } = require('openai');

const configuration = new Configuration({
  apiKey: process.env.OPENAI_API_KEY,
});

exports.openai = new OpenAIApi(configuration);
```

---

10. Instalación de Dependencias

Asegúrate de instalar todas las dependencias necesarias:

bash

Copiar código

npm install express sequelize pg pg-hstore bcryptjs jsonwebtoken cors dotenv openai

---

11. Scripts en package.json

Agrega los scripts para ejecutar el servidor:

json

Copiar código

```
{
  "scripts": {
    "start": "node app.js",
    "dev": "nodemon app.js"
  }
}
```

Frontend

1. Estructura del Proyecto

Asegúrate de que tu proyecto React tenga la siguiente estructura:

css

Copiar código

```
frontend/
├── src/
│   ├── index.js
│   ├── App.js
│   ├── api/
│   │   ├── apiService.js
│   │   └── authService.js
│   ├── components/
│   │   ├── Auth/
```

```
│   │   └── Login.js
│   └── Dashboard/
│       ├── Dashboard.js
│       ├── ContractList.js
│       └── ContractEditor.js
├── contexts/
│   └── AuthContext.js
└── routes/
    └── PrivateRoute.js
├── package.json
└── ...
```

---

2. Archivo Principal src/index.js

javascript

Copiar código

```
// index.js

import React from 'react';
import ReactDOM from 'react-dom';
import App from './App';
import { AuthProvider } from './contexts/AuthContext';

ReactDOM.render(
  <AuthProvider>
    <App />
  </AuthProvider>,
  document.getElementById('root')
);
```

---

3. Componente Principal src/App.js

Actualizado para ser compatible con react-router-dom v6.

javascript

Copiar código

```
// App.js

import React from 'react';
import { BrowserRouter as Router, Routes, Route } from 'react-router-dom';
import Login from './components/Auth/Login';
import Dashboard from './components/Dashboard/Dashboard';
import PrivateRoute from './routes/PrivateRoute';

function App() {
 return (
  <Router>
   <Routes>
    {/* Ruta pública */}
    <Route path="/login" element={<Login />} />
    {/* Ruta privada */}
    <Route
     path="/"
     element={
      <PrivateRoute>
       <Dashboard />
      </PrivateRoute>
     }
    />
```

```
    </Routes>
   </Router>
  );
}

export default App;
```

---

4. Contexto de Autenticación src/contexts/AuthContext.js

javascript

Copiar código

```
// AuthContext.js

import React, { createContext, useState } from 'react';

export const AuthContext = createContext();

export const AuthProvider = ({ children }) => {
  const existingToken = localStorage.getItem('token');
  const [authToken, setAuthToken] = useState(existingToken);
  const [notary, setNotary] = useState(null);

  const setToken = (data) => {
    localStorage.setItem('token', data.token);
    setAuthToken(data.token);
    setNotary(data.notary);
  };

  const logout = () => {
```

```
  localStorage.removeItem('token');
  setAuthToken(null);
  setNotary(null);
 };

 return (
  <AuthContext.Provider value={{ authToken, setAuthToken: setToken, notary, logout }}>
   {children}
  </AuthContext.Provider>
 );
};
```

---

5. Rutas Privadas src/routes/PrivateRoute.js

Actualizado para react-router-dom v6.

javascript

Copiar código

```
// PrivateRoute.js

import React, { useContext } from 'react';
import { Navigate } from 'react-router-dom';
import { AuthContext } from '../contexts/AuthContext';

const PrivateRoute = ({ children }) => {
 const { authToken } = useContext(AuthContext);

 return authToken ? children : <Navigate to="/login" />;
};
```

export default PrivateRoute;

---

6. Componentes

6.1. Componente de Inicio de Sesión src/components/Auth/Login.js

Actualizado para usar useNavigate.

javascript

Copiar código

```
// Login.js

import React, { useState, useContext } from 'react';
import authService from '../../api/authService';
import { AuthContext } from '../../contexts/AuthContext';
import { useNavigate } from 'react-router-dom';

function Login() {
  const [notaryId, setNotaryId] = useState('');
  const [password, setPassword] = useState('');
  const { setAuthToken } = useContext(AuthContext);
  const navigate = useNavigate();

  const handleLogin = async () => {
    const response = await authService.login(notaryId, password);
    if (response) {
      setAuthToken(response);
      navigate('/');
    } else {
      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;
```

6.2. Componente del Dashboard src/components/Dashboard/Dashboard.js

javascript

Copiar código

```
// Dashboard.js
```

```
import React, { useEffect, useState, useContext } from 'react';
import apiService from '../../api/apiService';
import ContractList from './ContractList';
import { AuthContext } from '../../contexts/AuthContext';

function Dashboard() {
  const [contracts, setContracts] = useState([]);
  const { logout } = useContext(AuthContext);

  useEffect(() => {
    const fetchContracts = async () => {
      const data = await apiService.getContracts();
      setContracts(data);
    };
    fetchContracts();
  }, []);

  return (
    <div>
      <h1>Dashboard</h1>
      <button onClick={logout}>Cerrar Sesión</button>
      <ContractList contracts={contracts} />
    </div>
  );
}

export default Dashboard;
```

6.3. Componentes Adicionales

- **ContractList.js**
- **ContractEditor.js**

Mantén estos componentes como los tienes, asegurándote de que las importaciones y el uso de hooks son correctos.

---

7. Servicios de API

7.1. Servicio de Autenticación src/api/authService.js

javascript

Copiar código

```
// authService.js

import axios from 'axios';

const login = async (notaryId, password) => {
 try {
  const response = await axios.post('/api/auth/login', { notaryId, password });
  return response.data;
 } catch (error) {
  console.error('Error en login:', error.response?.data || error.message);
  return null;
 }
};

export default { login };
```

7.2. Servicio de API src/api/apiService.js

javascript

Copiar código

```
// apiService.js

import axios from 'axios';

axios.defaults.baseURL = 'http://localhost:5000'; // Cambia esto si el backend está en otra URL

axios.interceptors.request.use(
  (config) => {
    const token = localStorage.getItem('token');
    if (token) config.headers.Authorization = `Bearer ${token}`;
    return config;
  },
  (error) => Promise.reject(error)
);

const getContracts = async () => {
  try {
    const response = await axios.get('/api/contracts');
    return response.data;
  } catch (error) {
    console.error('Error al obtener contratos:', error);
    return [];
  }
};

const createContract = async (contractType) => {
  try {
```

```
    const response = await axios.post('/api/contracts', { contractType });
    return response.data;
  } catch (error) {
    console.error('Error al crear contrato:', error);
    return null;
  }
};

const updateContract = async (id, clauses) => {
  try {
    const response = await axios.put(`/api/contracts/${id}`, { clauses });
    return response.data;
  } catch (error) {
    console.error('Error al actualizar contrato:', error);
    return null;
  }
};

export default { getContracts, createContract, updateContract };
```

---

8. Instalación de Dependencias del Frontend

Asegúrate de instalar las dependencias necesarias en tu proyecto React:

bash

Copiar código

npm install react-router-dom axios

Si estás usando react-router-dom versión 6, como en este caso, no necesitas instalar versiones anteriores.

---

9. Configuración del Proxy en package.json

En el package.json del frontend, agrega la configuración del proxy:

json

Copiar código

```
{
  // ...
  "proxy": "http://localhost:5000"
}
```

Esto permite que las solicitudes al backend sean redirigidas correctamente.

Aprendizajes y Correcciones Realizadas

Durante el proceso de desarrollo y solución de errores, hemos aprendido y aplicado lo siguiente:

1. **Migración de MongoDB a PostgreSQL con Sequelize**:
    - Configuramos Sequelize como ORM para interactuar con PostgreSQL.
    - Definimos modelos y relaciones utilizando Sequelize.
    - Sincronizamos los modelos con la base de datos y manejamos migraciones.
2. **Actualización a react-router-dom v6**:
    - Reemplazamos componentes y hooks obsoletos:
        - Switch por Routes
        - Redirect por Navigate
        - useHistory por useNavigate
    - Actualizamos la estructura de las rutas en App.js y los componentes relacionados.
3. **Manejo de Errores de Importación y Sensibilidad a Mayúsculas**:
    - Aseguramos que los nombres de archivos y las importaciones coincidan exactamente, respetando mayúsculas y minúsculas.

- Corregimos rutas de importación en los componentes para evitar errores de compilación.

4. **Configuración del Proxy en el Frontend**:
    - Agregamos el proxy en package.json para manejar las solicitudes al backend sin problemas de CORS.
5. **Solución de Advertencias de Babel**:
    - Instalamos dependencias faltantes para eliminar advertencias relacionadas con Babel y asegurar la estabilidad del proyecto.
6. **Buenas Prácticas de Desarrollo**:
    - Agregamos comentarios y explicaciones en el código para mejorar la legibilidad y mantenimiento.
    - Mantenemos una estructura de archivos organizada y coherente.
    - Nos aseguramos de que las variables de entorno y datos sensibles no estén expuestos en el código fuente.