Backend

Estructura del Proyecto Backend

La estructura de carpetas y archivos del backend es la siguiente:

lua

Copiar código

```
backend/
├── app.js
├── .env
├── package.json
├── config/
│   ├── database.js
│   └── openai.js
├── controllers/
│   ├── authController.js
│   └── contractsController.js
├── middlewares/
│   └── authMiddleware.js
├── models/
│   ├── Contract.js
│   └── Notary.js
├── routes/
│   ├── auth.js
│   └── contracts.js
├── services/
│   ├── aiService.js
│   └── blockchainService.js
└── utils/
    ├── encryption.js
```

└── errorHandler.js

---

1. Archivo Principal app.js

Este es el punto de entrada de la 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;
```

---

3. Configuración de OpenAI config/openai.js

Configura el cliente de OpenAI para utilizar la API de OpenAI.

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);
```

---

4. Modelos

4.1. 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;
```

4.2. 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;
```

---

5. Controladores

5.1. 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;

  // 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 } });

  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);
 }
};
```

5.2. 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);
  }
};
```

---

6. 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.' });
  }
};
```

---

7. Rutas

7.1. 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;
```

7.2. 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;
```

---

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. Utilidades

9.1. 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);
```

};

9.2. 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.' });
};
```

---

10. Variables de Entorno .env

Crea un archivo .env en el directorio raíz del backend con el siguiente contenido:

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
```

Nota: Reemplaza tu_usuario, tu_contraseña, tu_secreto_jwt y tu_clave_api_de_openai por los valores correspondientes.

---

11. Scripts en package.json

Agrega los scripts para ejecutar el servidor:

json

Copiar código

```
// package.json

{
  "name": "notarichain-backend",
  "version": "1.0.0",
  "description": "Backend para NotariChain",
  "main": "app.js",
  "scripts": {
    "start": "node app.js",
    "dev": "nodemon app.js"
  },
  "dependencies": {
    "bcryptjs": "^2.4.3",
    "cors": "^2.8.5",
    "dotenv": "^16.0.3",
    "express": "^4.18.2",
    "jsonwebtoken": "^9.0.0",
    "openai": "^3.1.0",
    "pg": "^8.8.0",
    "pg-hstore": "^2.3.4",
    "sequelize": "^6.28.0"
  },
  "devDependencies": {
    "nodemon": "^2.0.22"
  }
}
```

---

12. Instalación de Dependencias

Ejecuta el siguiente comando en el directorio del backend para instalar todas las dependencias:

bash

Copiar código

npm install

Frontend

Estructura del Proyecto Frontend

La estructura de carpetas y archivos del frontend es la siguiente:

java

Copiar código

```
frontend/
├── public/
│   ├── favicon.ico
│   └── index.html
├── src/
│   ├── index.js
│   ├── App.js
│   ├── api/
│   │   ├── apiService.js
│   │   └── authService.js
│   ├── components/
│   │   ├── Auth/
│   │   │   ├── Login.js
│   │   │   └── Register.js
│   │   └── Dashboard/
```

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

---

1. Archivo Principal src/index.js

Actualizado para usar createRoot en React 18.

javascript

Copiar código

```
// src/index.js

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

const root = ReactDOM.createRoot(document.getElementById('root'));

root.render(
  <AuthProvider>
    <App />
  </AuthProvider>
);
```

---

2. Componente Principal src/App.js

Define las rutas de la aplicación.

javascript

Copiar código

```
// src/App.js

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

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

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

export default App;
```

---

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

Gestiona el estado de autenticación de la aplicación.

javascript

Copiar código

```
// src/contexts/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>
  );
};
```

---

4. Rutas Privadas src/routes/PrivateRoute.js

Protege las rutas que requieren autenticación.

javascript

Copiar código

```
// src/routes/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;
```

---

5. Componentes de Autenticación

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

Permite a los notarios iniciar sesión.

javascript

Copiar código

```
// src/components/Auth/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 (e) => {
    e.preventDefault();
    const response = await authService.login(notaryId, password);
    if (response) {
```

```
    setAuthToken(response);
    navigate('/');
  } else {
    alert('Credenciales incorrectas.');
  }
};

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

```
      </div>
      <button type="submit">Entrar</button>
    </form>
    <p>
      ¿No tienes una cuenta? <a href="/register">Regístrate aquí</a>.
    </p>
   </div>
  );
}

export default Login;
```

5.2. Componente de Registro src/components/Auth/Register.js

Permite a los notarios registrarse.

javascript

Copiar código

```
// src/components/Auth/Register.js

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

function Register() {
  const [notaryId, setNotaryId] = useState('');
  const [name, setName] = useState('');
  const [password, setPassword] = useState('');
  const navigate = useNavigate();

  const handleRegister = async (e) => {
```

```
  e.preventDefault();

  const response = await authService.register(notaryId, name, password);
  if (response) {
    alert('Notario registrado exitosamente. Ahora puedes iniciar sesión.');
    navigate('/login');
  } else {
    alert('Error al registrar el notario. Por favor, intenta de nuevo.');
  }
};

return (
  <div>
    <h2>Registro de Notario</h2>
    <form onSubmit={handleRegister}>
      <div>
        <label>ID de Notario:</label>
        <input
          type="text"
          value={notaryId}
          onChange={(e) => setNotaryId(e.target.value)}
          placeholder="ID de Notario"
        />
      </div>
      <div>
        <label>Nombre:</label>
        <input
          type="text"
```

```
          value={name}
          onChange={(e) => setName(e.target.value)}
          placeholder="Nombre Completo"
        />
      </div>
      <div>
        <label>Contraseña:</label>
        <input
          type="password"
          value={password}
          onChange={(e) => setPassword(e.target.value)}
          placeholder="Contraseña"
        />
      </div>
      <button type="submit">Registrar</button>
    </form>
    <p>
      ¿Ya tienes una cuenta? <a href="/login">Inicia sesión aquí</a>.
    </p>
  </div>
 );
}

export default Register;
```

---

6. Componentes del Dashboard

6.1. Componente Dashboard.js

Muestra el panel principal para el notario autenticado.

javascript

Copiar código

```
// src/components/Dashboard/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;
```

6.2. Componente ContractList.js

Lista los contratos del notario.

javascript

Copiar código

```
// src/components/Dashboard/ContractList.js

import React from 'react';
import ContractEditor from './ContractEditor';

function ContractList({ contracts }) {
  return (
    <div>
      <h2>Tus Contratos</h2>
      {contracts.map((contract) => (
        <ContractEditor key={contract.id} contract={contract} />
      ))}
    </div>
  );
}

export default ContractList;
```

6.3. Componente ContractEditor.js

Permite editar los contratos.

javascript

Copiar código

```
// src/components/Dashboard/ContractEditor.js

import React, { useState } from 'react';
import apiService from '../../api/apiService';

function ContractEditor({ contract }) {
 const [clauses, setClauses] = useState(contract.clauses || []);

 const handleSave = async () => {
  const updatedContract = await apiService.updateContract(contract.id, clauses);
  if (updatedContract) {
   alert('Contrato actualizado exitosamente.');
  } else {
   alert('Error al actualizar el contrato.');
  }
 };

 return (
  <div>
   <h3>{contract.contractType}</h3>
   {clauses.map((clause, index) => (
    <div key={index}>
     <textarea
      value={clause}
      onChange={(e) => {
       const newClauses = [...clauses];
       newClauses[index] = e.target.value;
       setClauses(newClauses);
```

```
          }}
        />
      </div>
    ))}
    <button onClick={handleSave}>Guardar Cambios</button>
  </div>
 );
}

export default ContractEditor;
```

---

7. Servicios de API

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

Maneja las solicitudes de autenticación.

javascript

Copiar código

```
// src/api/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;
 }
```

```
};

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

export default { login, register };
```

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

Maneja las solicitudes relacionadas con contratos.

javascript

Copiar código

```
// src/api/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. Configuración del Proxy en package.json

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

json

Copiar código

```
// package.json

{
  "name": "notarichain-frontend",
  "version": "1.0.0",
  "private": true,
  "dependencies": {
    "axios": "^1.4.0",
    "react": "^18.2.0",
    "react-dom": "^18.2.0",
    "react-router-dom": "^6.14.1",
    "react-scripts": "5.0.1"
  },
  "scripts": {
    "start": "react-scripts start",
```

```
  "build": "react-scripts build"
 },
 "proxy": "http://localhost:5000"
}
```

---

9. Instalación de Dependencias del Frontend

Ejecuta el siguiente comando en el directorio del frontend para instalar todas las dependencias:

bash

Copiar código

```
npm install
```

---

10. Agregar el Favicon

Para evitar el error de favicon.ico no encontrado, agrega un favicon a tu proyecto.

- Coloca un archivo favicon.ico en la carpeta public/ del frontend.
- Asegúrate de tener la siguiente línea en public/index.html dentro del <head>:

html

Copiar código

```
<link rel="icon" href="%PUBLIC_URL%/favicon.ico" />
```