**🧱 Configuración del Proyecto**

**npm init -y**

Este comando inicializa un nuevo proyecto Node.js generando automáticamente un archivo package.json con valores predeterminados. Es el punto de partida para instalar dependencias con npm.

**npm install**

Instala las dependencias listadas en package.json. Si es la primera vez, este comando no hace nada relevante hasta que agregues paquetes con otros comandos.

**npm install dotenv @nomicfoundation/hardhat-toolbox**

Instala dos dependencias:

* dotenv: Permite cargar variables de entorno desde un archivo .env (por ejemplo, claves privadas o URLs de nodos).
* @nomicfoundation/hardhat-toolbox: Es un conjunto de herramientas para Hardhat que incluye plugins como ethers.js, waffle, chai, y soporte para pruebas, despliegues, etc.

**npm install --save-dev hardhat**

Instala **Hardhat** como dependencia de desarrollo. Hardhat es un entorno de desarrollo para compilar, desplegar, probar e interactuar con contratos inteligentes de Ethereum.

**npm install ethers**

Este paquete permite interactuar con contratos inteligentes desde JavaScript. Es ampliamente usado junto a Hardhat.

**⚙️ Inicialización y ejecución de Hardhat**

**npx hardhat**

Ejecuta Hardhat y muestra un asistente interactivo para inicializar el proyecto. Si es la primera vez, puedes seleccionar "crear un proyecto básico" y Hardhat generará una estructura de carpetas (contracts/, scripts/, test/, etc.).

**npx hardhat node**

Ejecuta un nodo local de Ethereum simulado. Es útil para pruebas, ya que crea cuentas con fondos ficticios y permite desplegar contratos sin necesidad de una red pública.

**🚀 Compilación y Despliegue**

**npx hardhat compile**

Compila los contratos inteligentes ubicados en la carpeta contracts/. Genera los artefactos (ABI y bytecode) que luego se usan para desplegarlos o interactuar con ellos.

**npx hardhat run scripts/deploy.js --network localhost**

Ejecuta el script deploy.js para desplegar el contrato en la red especificada (localhost, en este caso, el nodo local que corre con npx hardhat node).  
El script suele contener código como:

javascript

const main = async () => {

const Contract = await ethers.getContractFactory("MiContrato");

const contract = await Contract.deploy();

await contract.deployed();

console.log(`Contrato desplegado en: ${contract.address}`);

};

**✅ Resumen del flujo**

1. **Inicializas el proyecto** con npm init.
2. **Instalas las herramientas necesarias**: Hardhat, dotenv, ethers, toolbox.
3. **Ejecutas el asistente de Hardhat** con npx hardhat.
4. **Compilas los contratos** con npx hardhat compile.
5. **Levantás un nodo local** con npx hardhat node.
6. **Desplegas el contrato** con npx hardhat run scripts/deploy.js --network localhost.

Si necesitas, puedo ayudarte a escribir el contenido del archivo deploy.js o el archivo .env con claves privadas y configuraciones seguras. ¿Te gustaría eso?

<https://chatgpt.com/share/68388f91-1ce0-800f-909e-1ce0f8332db5>

// Emitir certificado exports.issueCertificate = async (req, res) => { try { const { instituteName,facultyName, studentName, cedula, language, level } = req.body; //FORMA LA ESTRUCTURA const tx = await contract.issueCertificate(instituteName,facultyName, studentName, cedula, language, level); await tx.wait(); // HACE USO DEL CONTRATO, TAL CUAL DEL CONTRATO const newCert = new Certificate({ instituteName, facultyName, studentName, cedula, language, level, issueDate: new Date(), valid: true, }); await newCert.save(); //GUARDA DENTRO DE RED BLOCKCHAIN res.status(201).json({ message: 'Certificado emitido' }); } catch (err) { res.status(500).json({ message: 'Error al emitir certificado', error: err.message }); } };

**ChatGPT dijo:**

Claro, te explico esta función paso a paso, y luego te doy una **versión parafraseada y con mejor redacción** de lo que está ocurriendo:

**📌 Código: exports.issueCertificate**

Esta función es parte de un backend (posiblemente en Node.js con Express) que permite **emitir certificados** y **registrarlos tanto en blockchain como en una base de datos (MongoDB, por ejemplo)**.

**🧩 Paso a paso:**

1. **Lectura de datos desde el body de la solicitud HTTP**:

js

CopiarEditar

const { instituteName, facultyName, studentName, cedula, language, level } = req.body;

Se extraen los datos del certificado enviados desde el frontend o cliente.

1. **Interacción con el contrato inteligente**:

js

CopiarEditar

const tx = await contract.issueCertificate(...);

await tx.wait();

* + Llama a la función issueCertificate del contrato inteligente desplegado en la blockchain.
  + tx.wait() espera a que la transacción se confirme en un bloque.

1. **Creación del objeto en la base de datos**:

js

CopiarEditar

const newCert = new Certificate({ ... });

await newCert.save();

Se guarda una copia local del certificado en una base de datos tradicional, útil para consultas rápidas sin depender de la blockchain.

1. **Respuesta HTTP al cliente**:

js

CopiarEditar

res.status(201).json({ message: 'Certificado emitido' });

Informa al usuario que el certificado fue emitido exitosamente.

1. **Manejo de errores**:  
   Si algo falla, se captura el error y se responde con un estado 500.

**📝 Versión parafraseada y mejor redactada:**

Esta función permite emitir un nuevo certificado de idioma a través de una solicitud HTTP. Primero, extrae los datos proporcionados por el usuario (como nombre del estudiante, cédula, idioma, nivel, entre otros). Luego, realiza una transacción con el contrato inteligente desplegado en la red blockchain para registrar el certificado de forma descentralizada. Una vez confirmada la transacción, guarda una copia de ese certificado en la base de datos local para facilitar futuras consultas. Finalmente, envía una respuesta al cliente confirmando la emisión exitosa del certificado, o informa si ocurrió un error durante el proceso.

¿Querés que lo adapte al formato de tesis o documento académico, para incluirlo como explicación en un capítulo técnico?

**Dijiste:**

// Verificar validez exports.verifyCertificate = async (req, res) => { try { const { cedula } = req.params; const cert = await Certificate.findOne({ cedula }); if (!cert) return res.status(404).json({ message: 'No encontrado' }); res.json({ valid: cert.valid }); } catch (err) { res.status(500).json({ message: 'Error al verificar', error: err.message }); } };

**ChatGPT dijo:**

Aquí tienes la explicación detallada de la función verifyCertificate, seguida de una versión parafraseada y redactada con estilo académico:

**📌 Código: exports.verifyCertificate**

Esta función del backend permite **verificar si un certificado es válido**, usando como referencia la **cédula del estudiante**.

**🧩 Explicación paso a paso:**

1. **Obtiene la cédula desde los parámetros de la URL**:

js

CopiarEditar

const { cedula } = req.params;

Por ejemplo, si se hace una petición a /verify/12345678, la cédula será "12345678".

1. **Busca el certificado en la base de datos**:

js

CopiarEditar

const cert = await Certificate.findOne({ cedula });

Se consulta la base de datos para encontrar el certificado asociado a esa cédula.

1. **Valida si el certificado existe**:

js

CopiarEditar

if (!cert) return res.status(404).json({ message: 'No encontrado' });

Si no se encuentra, responde con un mensaje de error y estado HTTP 404.

1. **Devuelve el estado de validez**:

js

CopiarEditar

res.json({ valid: cert.valid });

Si se encuentra el certificado, devuelve un objeto con el campo valid, que será true o false.

1. **Manejo de errores**:  
   Si ocurre un error inesperado (por ejemplo, problemas de conexión), se responde con código 500.

**📝 Versión parafraseada y redactada con estilo académico:**

Esta función se encarga de verificar la validez de un certificado previamente emitido. Para ello, recibe la cédula del estudiante como parámetro desde la URL de la solicitud. A continuación, busca en la base de datos un certificado que coincida con dicha cédula. Si no se encuentra ningún resultado, responde con un mensaje indicando que no existe registro correspondiente. En caso contrario, devuelve un valor booleano (true o false) que indica si el certificado está actualmente válido. Este procedimiento resulta útil para validar la autenticidad de certificados desde aplicaciones externas o sistemas administrativos.

¿Te gustaría que haga lo mismo con el resto de las funciones? ¿O que lo pase a formato de tesis (capítulo, sección, o tabla)?

**Dijiste:**

// Obtener un certificado exports.getCertificate = async (req, res) => { try { const { cedula } = req.params; const cert = await Certificate.findOne({ cedula }); if (!cert) return res.status(404).json({ message: 'No encontrado' }); res.json(cert); } catch (err) { res.status(500).json({ message: 'Error al obtener certificado', error: err.message }); } };

**ChatGPT dijo:**

Aquí tienes la explicación paso a paso de la función getCertificate, seguida de una **versión parafraseada y redactada con estilo académico**:

**📌 Código: exports.getCertificate**

Esta función permite **obtener todos los datos de un certificado** mediante una solicitud HTTP, usando la cédula del estudiante como parámetro.

**🧩 Explicación detallada:**

1. **Obtiene la cédula desde la URL**:

js

CopiarEditar

const { cedula } = req.params;

Por ejemplo, si la solicitud es a /certificates/12345678, la cédula será "12345678".

1. **Consulta en la base de datos**:

js

CopiarEditar

const cert = await Certificate.findOne({ cedula });

Busca el certificado que tenga como clave la cédula ingresada.

1. **Valida si existe el certificado**:

js

CopiarEditar

if (!cert) return res.status(404).json({ message: 'No encontrado' });

Si no se encuentra ningún certificado con esa cédula, responde con error 404.

1. **Devuelve el certificado completo en formato JSON**:

js

CopiarEditar

res.json(cert);

Si se encuentra, se envía toda la información del certificado (nombre del estudiante, idioma, nivel, validez, etc.).

1. **Manejo de errores**:  
   Si ocurre algún problema durante la búsqueda, responde con un error 500 y el mensaje correspondiente.

**📝 Versión parafraseada y redactada con estilo académico:**

Esta función tiene como finalidad recuperar toda la información asociada a un certificado de idioma emitido. Recibe como parámetro la cédula del estudiante desde la URL de la solicitud HTTP y consulta la base de datos para encontrar un certificado que coincida con dicha clave. Si no se encuentra un registro correspondiente, se devuelve un mensaje de error con estado 404. En caso de éxito, se retorna toda la estructura del certificado en formato JSON, lo cual permite consultar sus atributos (como nombre, idioma, nivel, validez y fecha de emisión) desde sistemas externos o interfaces web.

¿Querés que lo pase a formato de tabla junto con las demás funciones, o que lo integre como parte de un capítulo técnico de tu tesis?

**Dijiste:**

// Obtener todos los certificados (admin) exports.getAllCertificates = async (req, res) => { try { if (req.user.role !== 'admin') { return res.status(403).json({ message: 'Acceso denegado. Solo los administradores pueden ver todos los certificados.' }); } const certificates = await Certificate.find(); res.status(200).json(certificates); } catch (err) { res.status(500).json({ message: 'Error del servidor', error: err.message }); } };

**ChatGPT dijo:**

Aquí tienes la explicación de la función getAllCertificates, junto con una **versión parafraseada y redactada en estilo académico**:

**📌 Código: exports.getAllCertificates**

Esta función permite **obtener la lista completa de certificados emitidos**, pero **solo está disponible para usuarios con rol de administrador**.

**🧩 Explicación paso a paso:**

1. **Verifica el rol del usuario**:

js

CopiarEditar

if (req.user.role !== 'admin') {

Antes de ejecutar la consulta, verifica si el usuario que hace la solicitud tiene permisos de administrador.

1. **Deniega el acceso si no es administrador**:

js

CopiarEditar

return res.status(403).json({ message: 'Acceso denegado...' });

Si el usuario no tiene rol de administrador, devuelve un mensaje de error y estado 403 (prohibido).

1. **Consulta todos los certificados en la base de datos**:

js

CopiarEditar

const certificates = await Certificate.find();

Si el usuario está autorizado, busca y recupera todos los certificados almacenados.

1. **Responde con la lista de certificados**:

js

CopiarEditar

res.status(200).json(certificates);

Devuelve un arreglo de objetos JSON que representan cada certificado emitido.

1. **Manejo de errores del servidor**:  
   Si ocurre un error inesperado, devuelve estado 500 y el mensaje de error correspondiente.

**📝 Versión parafraseada y con redacción académica:**

Esta función está diseñada para permitir que los administradores del sistema puedan consultar todos los certificados de idioma emitidos. Antes de realizar la operación, verifica que el usuario autenticado tenga el rol de "admin", ya que esta funcionalidad está restringida por razones de seguridad y control de acceso. En caso de no contar con los permisos necesarios, se devuelve un mensaje de acceso denegado con código de estado 403. Si la autorización es válida, la función recupera todos los certificados desde la base de datos y los retorna en formato JSON. Esta funcionalidad es especialmente útil para tareas de auditoría, reportes institucionales o supervisión administrativa.

¿Querés que compilemos todas estas funciones explicadas en una **sección de tu tesis**, por ejemplo como “Descripción de funciones del backend” o en una tabla resumen?

**Dijiste:**

exports.invalidateCertificate = async (req, res) => { try { const { cedula } = req.params; // Paso 1: Llamar al contrato inteligente const tx = await contract.invalidateCertificate(cedula); await tx.wait(); // Paso 2: Actualizar la base de datos MongoDB const cert = await Certificate.findOneAndUpdate( { cedula }, { valid: false }, { new: true } ); if (!cert) { return res.status(404).json({ message: 'Certificado no encontrado' }); } res.status(200).json({ message: 'Certificado invalidado correctamente', certificado: cert }); } catch (err) { console.error('❌ Error al invalidar certificado:', err); res.status(500).json({ message: 'Error al invalidar certificado', error: err.message }); } };

**ChatGPT dijo:**

Aquí tienes la explicación completa de la función invalidateCertificate, junto con su **versión parafraseada y redactada con estilo académico**:

**📌 Código: exports.invalidateCertificate**

Esta función permite **invalidar un certificado de idioma previamente emitido**, tanto en la **blockchain** como en la **base de datos MongoDB**, usando la **cédula del estudiante** como identificador.

**🧩 Explicación paso a paso:**

1. **Obtiene la cédula desde los parámetros de la URL**:

js

CopiarEditar

const { cedula } = req.params;

1. **Paso 1: Llama al contrato inteligente**:

js

CopiarEditar

const tx = await contract.invalidateCertificate(cedula);

await tx.wait();

Se ejecuta la función invalidateCertificate del contrato inteligente en Solidity, marcando el certificado como no válido. await tx.wait() espera a que la transacción sea confirmada en la blockchain.

1. **Paso 2: Actualiza el estado del certificado en MongoDB**:

js

CopiarEditar

const cert = await Certificate.findOneAndUpdate(

{ cedula },

{ valid: false },

{ new: true }

);

Se busca el certificado correspondiente en la base de datos y se actualiza su campo valid a false.

1. **Verifica si se encontró el certificado**:  
   Si no se encuentra un registro con esa cédula, se responde con error 404.
2. **Responde con éxito**:

js

CopiarEditar

res.status(200).json({ message: 'Certificado invalidado correctamente', certificado: cert });

Se confirma que el certificado fue invalidado correctamente tanto en la blockchain como en la base de datos.

1. **Manejo de errores**:  
   Si ocurre algún fallo en el proceso (ya sea con la blockchain o MongoDB), se responde con error 500.

**📝 Versión parafraseada y redactada con estilo académico:**

Esta función tiene como objetivo invalidar un certificado previamente emitido, asegurando que ya no pueda considerarse válido. Para ello, recibe como parámetro la cédula del estudiante y realiza dos acciones principales: primero, llama a la función invalidateCertificate del contrato inteligente desplegado en la blockchain, lo cual garantiza que el cambio sea registrado de manera inmutable; segundo, actualiza el campo valid del certificado correspondiente en la base de datos MongoDB, estableciéndolo en false. De esta manera, se mantiene la coherencia entre el registro on-chain y la base de datos off-chain. En caso de no encontrar el certificado, se devuelve un mensaje de error; si la operación se realiza con éxito, se confirma con una respuesta que incluye los datos actualizados del certificado.

¿Te gustaría que compilemos todas estas funciones explicadas en un **apartado de la tesis** o en una **tabla resumen comparativa**? Puedo ayudarte a dejarlo listo para tu documento.

**Dijiste:**

const express = require('express'); const router = express.Router(); const jwt = require('jsonwebtoken'); const User = require('../models/User'); // Registro de usuarios router.post('/register', async (req, res) => { try { const { email, password, cedula, role } = req.body; const user = new User({ email, password, cedula, role }); await user.save(); res.status(201).json({ message: 'Usuario registrado' }); } catch (err) { res.status(500).json({ message: 'Error al registrar', error: err.message }); } }); // Login router.post('/login', async (req, res) => { try { const { email, password } = req.body; const user = await User.findOne({ email }); if (!user || !(await user.comparePassword(password))) { return res.status(400).json({ message: 'Credenciales inválidas' }); } const token = jwt.sign({ id: user.\_id, role: user.role }, process.env.JWT\_SECRET); res.json({ token, role: user.role }); } catch (err) { res.status(500).json({ message: 'Error al iniciar sesión' }); } }); module.exports = router;

**ChatGPT dijo:**

Aquí tienes la explicación completa del código que gestiona el **registro e inicio de sesión de usuarios** en una API Express, junto con una **versión parafraseada y redactada en estilo académico**, ideal para incluir en tu tesis:

**📌 Código de autenticación (routes/auth.js)**

Este módulo define las rutas de autenticación de usuarios para la aplicación. Utiliza Express, JWT (JSON Web Tokens) y un modelo de usuario de MongoDB para manejar el **registro** y el **inicio de sesión**.

**🔐 Registro de usuarios**

js

CopiarEditar

router.post('/register', async (req, res) => {

try {

const { email, password, cedula, role } = req.body;

const user = new User({ email, password, cedula, role });

await user.save();

res.status(201).json({ message: 'Usuario registrado' });

} catch (err) {

res.status(500).json({ message: 'Error al registrar', error: err.message });

}

});

**✅ ¿Qué hace?**

* Recibe desde el frontend los datos: email, password, cedula y role.
* Crea una nueva instancia del modelo User.
* Guarda al usuario en la base de datos.
* Si todo sale bien, responde con un mensaje de éxito.
* En caso de error (por ejemplo, duplicación de email), retorna un error 500.

**🔑 Inicio de sesión**

js

CopiarEditar

router.post('/login', async (req, res) => {

try {

const { email, password } = req.body;

const user = await User.findOne({ email });

if (!user || !(await user.comparePassword(password))) {

return res.status(400).json({ message: 'Credenciales inválidas' });

}

const token = jwt.sign({ id: user.\_id, role: user.role }, process.env.JWT\_SECRET);

res.json({ token, role: user.role });

} catch (err) {

res.status(500).json({ message: 'Error al iniciar sesión' });

}

});

**✅ ¿Qué hace?**

* Busca al usuario por su correo electrónico.
* Verifica que la contraseña sea correcta con comparePassword() (debe estar definido en el modelo User).
* Si las credenciales son válidas, genera un **JWT token** con el ID y rol del usuario.
* Retorna el token y el rol para que el frontend lo utilice en autenticación.
* Si las credenciales no son válidas, retorna un error 400.

**📝 Versión parafraseada con redacción académica:**

El siguiente módulo permite gestionar el sistema de autenticación de usuarios. Para ello, se implementan dos rutas principales: una para el **registro** y otra para el **inicio de sesión**. En el proceso de registro (/register), se reciben desde el cliente los datos del nuevo usuario, los cuales se almacenan en la base de datos mediante el modelo User. Por otro lado, en el inicio de sesión (/login), se valida la existencia del usuario y la correspondencia de la contraseña. Si las credenciales son correctas, se genera un token JWT que contiene el identificador del usuario y su rol, permitiendo así implementar control de acceso y mantener la sesión del usuario en el cliente. Este mecanismo de autenticación es fundamental para proteger las rutas privadas y asegurar que ciertas operaciones solo puedan ser realizadas por usuarios autorizados.

¿Querés que también parafraseemos y expliquemos el modelo User que usás con Mongoose? Puedo ayudarte con eso también.

**Dijiste:**

const express = require('express'); const router = express.Router(); const authMiddleware = require('../middleware/authMiddleware'); const { issueCertificate, verifyCertificate, getCertificate, getAllCertificates, downloadCertificate, invalidateCertificate } = require('../controllers/certificateController'); router.get('/all', authMiddleware('admin'), getAllCertificates);//Rutas para obtener todos los certificados router.post('/issue', authMiddleware('admin'), issueCertificate);//ruta para emitir un certificado router.get('/verify/:cedula', verifyCertificate);//ruta para verificar un certificado por cedula router.get('/:cedula', getCertificate);//ruta para obtener un certificado por cedula router.get('/:cedula/download', downloadCertificate);//ruta para descargar un certificado por cedula router.put('/invalidate/:cedula',authMiddleware('admin'), invalidateCertificate);//ruta para invalidar un certificado module.exports = router;

**ChatGPT dijo:**

Aquí tienes una **explicación completa y parafraseada** del archivo de rutas para certificados en una API con Express. Esto es ideal para documentar en tu tesis:

**📌 Archivo de rutas de certificados (routes/certificateRoutes.js)**

Este archivo define todas las rutas HTTP relacionadas con la gestión de certificados dentro de la aplicación. Utiliza el enrutador de Express y controla el acceso a ciertas operaciones mediante un middleware de autenticación (authMiddleware), el cual restringe algunas rutas solo a usuarios con rol de **administrador**.

**🔧 Funcionalidad general**

Cada ruta está asociada a una función específica del controlador (certificateController) que realiza la lógica correspondiente, como emitir, verificar, obtener, descargar o invalidar certificados.

**📋 Descripción de las rutas**

| **Ruta** | **Método** | **Middleware** | **Descripción** |
| --- | --- | --- | --- |
| /all | GET | admin | Devuelve todos los certificados registrados. Solo accesible por administradores. |
| /issue | POST | admin | Permite emitir un nuevo certificado. Requiere privilegios de administrador. |
| /verify/:cedula | GET | — | Verifica si un certificado con la cédula dada es válido o no. |
| /:cedula | GET | — | Obtiene los datos completos de un certificado según la cédula proporcionada. |
| /:cedula/download | GET | — | Permite descargar el certificado en formato archivo (PDF, etc.). |
| /invalidate/:cedula | PUT | admin | Invalida un certificado específico, marcándolo como no válido. Solo para administradores. |

**📝 Redacción para tesis (versión parafraseada):**

Para organizar y exponer las funcionalidades del sistema de gestión de certificados, se creó un módulo de rutas utilizando Express. Este módulo centraliza las solicitudes HTTP dirigidas al sistema de certificados y las vincula con las funciones correspondientes definidas en el controlador. Además, se implementa un **middleware de autenticación basado en roles**, que permite restringir el acceso a ciertas rutas sensibles únicamente a usuarios con rol de administrador.

Entre las rutas principales se encuentra la emisión de certificados, la verificación de su validez, la consulta individual por número de cédula, la descarga y la invalidación. También se proporciona una ruta exclusiva para administradores que permite obtener un listado completo de todos los certificados emitidos en el sistema. Esta estructura modular facilita el mantenimiento, la escalabilidad del sistema y asegura el cumplimiento de políticas de acceso controlado.

¿Querés que también te ayude a parafrasear el middleware authMiddleware o el certificateController completo?

**Dijiste:**

const express = require('express'); //mediador para leer contrato const dotenv = require('dotenv'); // leer archovo .env const connectDB = require('./config/db'); const cors = require('cors'); dotenv.config(); connectDB(); const app = express(); app.use(cors({ origin: 'http://localhost:5173', credentials: true })); app.use(express.json()); app.use('/api/auth', require('./routes/authRoutes')); //contola la autentificacion app.use('/api/certificates', require('./routes/certificateRoutes')); //controla los certificados post,delite const PORT = process.env.PORT || 5000; app.listen(PORT, () => console.log(Servidor ejecutándose en puerto ${PORT}));

**ChatGPT dijo:**

Aquí tienes una **explicación clara, parafraseada y mejor redactada** del código proporcionado para que la puedas incluir en tu tesis:

**🧩 Archivo principal del servidor (server.js o index.js)**

Este archivo actúa como punto de entrada principal para la aplicación backend. Su propósito es inicializar el servidor, cargar la configuración del entorno, establecer conexión con la base de datos y definir las rutas disponibles para autenticación y gestión de certificados.

**🔍 Explicación detallada del código**

js

CopiarEditar

const express = require('express'); // Framework que facilita la creación del servidor y rutas

const dotenv = require('dotenv'); // Permite leer variables de entorno desde un archivo .env

const connectDB = require('./config/db'); // Función personalizada para conectar con la base de datos

const cors = require('cors'); // Middleware para permitir solicitudes desde el frontend (CORS)

1. **Configuración del entorno**:

js

CopiarEditar

dotenv.config();

connectDB();

Se carga la configuración desde un archivo .env y se establece la conexión con la base de datos MongoDB.

1. **Inicialización del servidor**:

js

CopiarEditar

const app = express();

Se crea una instancia de una aplicación Express.

1. **Middlewares**:

js

CopiarEditar

app.use(cors({ origin: 'http://localhost:5173', credentials: true }));

app.use(express.json());

* + Se permite el acceso desde la URL del frontend (usualmente Vite en el puerto 5173).
  + Se activa el parseo de solicitudes JSON para recibir datos en el cuerpo de las peticiones.

1. **Definición de rutas**:

js

CopiarEditar

app.use('/api/auth', require('./routes/authRoutes'));

app.use('/api/certificates', require('./routes/certificateRoutes'));

* + /api/auth: gestiona el registro e inicio de sesión de usuarios.
  + /api/certificates: maneja la emisión, verificación, consulta, invalidación y descarga de certificados.

1. **Inicio del servidor**:

js

CopiarEditar

const PORT = process.env.PORT || 5000;

app.listen(PORT, () => console.log(`Servidor ejecutándose en puerto ${PORT}`));

El servidor se ejecuta en el puerto definido en el archivo .env o, en su defecto, en el puerto 5000.

**📝 Redacción para tesis (parafraseada):**

El archivo principal del servidor tiene como función inicializar y configurar toda la aplicación backend. Para ello, se importa el framework Express junto a otras librerías como dotenv para la gestión de variables de entorno, cors para permitir solicitudes desde el frontend y una función personalizada que establece la conexión con la base de datos.

Se configura el middleware para permitir el intercambio de datos entre el backend y el frontend, especialmente útil durante el desarrollo local. Posteriormente, se definen dos grupos de rutas: una para el módulo de autenticación y otra para el manejo de certificados. Finalmente, el servidor se pone en marcha en un puerto especificado, quedando listo para recibir solicitudes desde el cliente.

¿Te gustaría que prepare también la explicación de cómo funciona el archivo .env o la conexión con MongoDB (./config/db.js)?