**🧱 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)?

**Fronend**

explicame que hace el archivo index.html "<!doctype html> <html lang="en"> <head> <meta charset="UTF-8" /> <link rel="icon" type="image/jpg" href="/public/images/logo-instituto.jpg" /> <meta name="viewport" content="width=device-width, initial-scale=1.0" /> <title>Sistema Gestion de Certificación-Instituto Académico de Idiomas</title> </head> <body> <div id="root"></div> <script type="module" src="/src/main.jsx"></script> </body> </html>s "

**ChatGPT dijo:**

Claro, te explico qué hace este archivo **index.html** paso a paso:

**Explicación del archivo index.html**

* **<!doctype html>**  
  Indica que el documento es una página HTML5.
* **<html lang="en">**  
  Define el inicio del documento HTML y establece que el idioma principal es inglés (en).
* **<head>**  
  Contiene información meta y recursos que no se ven directamente en la página, pero configuran cómo se muestra o funciona.
  + **<meta charset="UTF-8" />**  
    Define la codificación de caracteres del documento como UTF-8, que es estándar para que los caracteres se muestren correctamente.
  + **<link rel="icon" type="image/jpg" href="/public/images/logo-instituto.jpg" />**  
    Establece el ícono que aparecerá en la pestaña del navegador, usando la imagen ubicada en esa ruta.
  + **<meta name="viewport" content="width=device-width, initial-scale=1.0" />**  
    Hace que la página sea responsiva, adaptándose al tamaño de la pantalla del dispositivo donde se visualice.
  + **<title>**  
    El título que aparece en la pestaña del navegador, en este caso:  
    *Sistema Gestión de Certificación - Instituto Académico de Idiomas*.
* **<body>**  
  Es el contenido visible de la página.
  + **<div id="root"></div>**  
    Un contenedor vacío con id "root" donde se cargará dinámicamente la aplicación (por ejemplo, con React u otra librería JavaScript).
  + **<script type="module" src="/src/main.jsx"></script>**  
    Importa y ejecuta el archivo main.jsx, que probablemente contiene el código principal de la aplicación escrito en React u otra tecnología que usa módulos ES6. Este script es el que “inyecta” la interfaz y la funcionalidad dentro del <div id="root">.

**En resumen:**

Este archivo HTML es la plantilla base que carga la aplicación web. Define la configuración inicial y el contenedor donde el código JavaScript se encargará de construir y mostrar la interfaz de usuario.

¿Quieres que te explique qué podría contener el archivo main.jsx o cómo se relaciona con este index.html?

**Dijiste:**

archivo main.jsx "import { StrictMode } from 'react' import { createRoot } from 'react-dom/client' import './tailwind.css' import App from './App.jsx' createRoot(document.getElementById('root')).render( <StrictMode> <App /> </StrictMode>, ) "

**ChatGPT dijo:**

Claro, te explico qué hace el archivo **main.jsx** que compartiste:

**Explicación del archivo main.jsx**

* **Importaciones**:

jsx

CopiarEditar

import { StrictMode } from 'react'

import { createRoot } from 'react-dom/client'

import './tailwind.css'

import App from './App.jsx'

* + **StrictMode**: Es un componente de React que ayuda a identificar problemas potenciales en la aplicación durante el desarrollo, activando advertencias adicionales y comprobaciones.
  + **createRoot**: Método de React 18 para crear una raíz donde renderizar la aplicación en el DOM.
  + **'./tailwind.css'**: Importa estilos CSS generados con Tailwind CSS, una librería de utilidades CSS para estilizar componentes rápidamente.
  + **App**: El componente principal de la aplicación, importado desde App.jsx.
* **Renderizado de la aplicación**:

jsx

CopiarEditar

createRoot(document.getElementById('root')).render(

<StrictMode>

<App />

</StrictMode>,

)

* + Busca en el DOM el elemento con el id "root" (que está en el index.html).
  + Crea un “root” de React en ese elemento.
  + Renderiza el componente <App /> dentro del componente <StrictMode>, que envuelve la aplicación para ayudar a detectar problemas.

**En resumen:**

Este archivo es el punto de entrada de la aplicación React. Se encarga de:

1. Importar estilos y el componente principal.
2. Encontrar el contenedor en el HTML donde va a “inyectar” la aplicación.
3. Renderizar la aplicación React en ese contenedor, usando las mejores prácticas con StrictMode.

¿Quieres que te explique cómo podría estar estructurado el componente App.jsx o necesitas ayuda con otra cosa?

**Dijiste:**

el resumen que sea mas formal para la tesis

**ChatGPT dijo:**

Por supuesto, aquí tienes una versión más formal para tu tesis:

El archivo **main.jsx** constituye el punto de entrada principal de la aplicación desarrollada con React. Su función es importar los estilos necesarios y el componente raíz de la aplicación, denominado **App**. A continuación, localiza en el documento HTML el elemento con el identificador root y crea en él la raíz de React mediante el método createRoot. Finalmente, renderiza el componente **App** dentro del contenedor especificado, empleando el componente StrictMode para activar verificaciones adicionales que permiten detectar posibles problemas durante el desarrollo. De esta manera, se establece la base para la representación dinámica de la interfaz de usuario en el navegador.

¿Quieres que te ayude a redactar también el resumen del archivo index.html con el mismo nivel formal?

**Dijiste:**

parafracea "El archivo main.jsx constituye el punto de entrada principal de la aplicación desarrollada con React como se muestra en la figura 35, su función es importar los estilos necesarios y el componente raíz de la aplicación, denominado App."

**ChatGPT dijo:**

Claro, aquí tienes una versión parafraseada y formal:

“El archivo main.jsx representa el punto de inicio fundamental de la aplicación creada con React, como se observa en la figura 35. Su propósito es importar tanto los estilos requeridos como el componente principal de la aplicación, denominado App.”

**Dijiste:**

archivo App.jsx "import { BrowserRouter as Router, Routes, Route, Navigate } from 'react-router-dom'; import Navbar from './components/Navbar'; import Home from './pages/Home'; import Login from './pages/Login'; import Register from './pages/Register'; import IssueCertificate from './pages/IssueCertificate'; import VerifyCertificate from './pages/VerifyCertificate'; import GetCertificate from './pages/getCertificate'; import AllCertificates from './pages/AllCertificates'; import './tailwind.css'; // Helpers para proteger rutas const isAuthenticated = () => !!localStorage.getItem('token'); const getRole = () => localStorage.getItem('role'); // 'admin' o 'student' // Rutas protegidas por rol const ProtectedRoute = ({ element, allowedRoles }) => { const token = isAuthenticated(); const role = getRole(); if (!token) return <Navigate to="/login" />; if (!allowedRoles.includes(role)) return <Navigate to="/" />; return element; }; function App() { return ( <Router> <div className="min-h-screen bg-gray-100"> <Navbar /> <div className="p-6"> <Routes> <Route path="/" element={<Home />} /> <Route path="/login" element={<Login />} /> <Route path="/register" element={<Register />} /> <Route path="/verify" element={<VerifyCertificate />} /> {/\* Rutas protegidas \*/} <Route path="/issue" element={<ProtectedRoute element={<IssueCertificate />} allowedRoles={['admin']} />} /> <Route path="/all" element={<ProtectedRoute element={<AllCertificates />} allowedRoles={['admin']} />} /> <Route path="/get" element={<ProtectedRoute element={<GetCertificate />} allowedRoles={['student']} />} /> </Routes> </div> <footer className="footer text-center p-4 bg-gray-100 border-t mt-10"> <p>&copy; 2025 Sistema de Gestión de Certificados de Idiomas - UCE</p> </footer> </div> </Router> ); } export default App; "

**ChatGPT dijo:**

Claro, aquí tienes una explicación formal y detallada del archivo **App.jsx** para tu tesis:

**Explicación del archivo App.jsx**

El archivo **App.jsx** contiene el componente principal de la aplicación, encargado de gestionar la navegación entre las diferentes páginas mediante React Router.

* **Importaciones principales:**  
  Se importan componentes para la navegación (BrowserRouter, Routes, Route, Navigate), componentes visuales como la barra de navegación (Navbar) y las distintas páginas (Home, Login, Register, entre otras), así como los estilos CSS generados con Tailwind.
* **Manejo de autenticación y roles:**  
  Se definen funciones auxiliares para verificar si el usuario está autenticado (isAuthenticated) y para obtener el rol del usuario (getRole), almacenados en el almacenamiento local (localStorage). Los roles contemplados son principalmente admin y student.
* **Rutas protegidas:**  
  El componente ProtectedRoute actúa como un guardián para ciertas rutas, asegurando que solo usuarios autenticados con los roles permitidos puedan acceder a ellas. Si el usuario no está autenticado, se redirige a la página de inicio de sesión. Si su rol no está autorizado para la ruta, se lo redirige a la página principal.
* **Definición de rutas:**  
  Se configuran rutas públicas accesibles a todos los usuarios, como la página principal (/), login, registro y verificación de certificados.  
  Las rutas protegidas, accesibles solo a ciertos roles, incluyen emisión de certificados (/issue) y listado de todos los certificados (/all) para administradores, y obtención de certificados (/get) para estudiantes.
* **Estructura visual:**  
  La aplicación se organiza dentro de un contenedor principal con estilos que garantizan una altura mínima y fondo gris claro. Se incluye una barra de navegación en la parte superior y un pie de página con información institucional en la parte inferior.

**En resumen:**

El componente **App.jsx** orquesta la estructura principal y la navegación de la aplicación, implementando un sistema de protección basado en autenticación y roles de usuario, que permite controlar el acceso a las diferentes funcionalidades según el perfil del usuario.

¿Quieres que te ayude a redactar esto de forma aún más formal o técnica para la tesis?

2/2

**Dijiste:**

si

**ChatGPT dijo:**

Por supuesto, aquí tienes una versión más formal y técnica para tu tesis:

El archivo **App.jsx** constituye el componente principal de la aplicación, encargado de gestionar la navegación y el control de acceso mediante la biblioteca React Router. En este archivo se importan los componentes visuales y funcionales necesarios, así como los estilos mediante Tailwind CSS.

Para garantizar la seguridad y la correcta asignación de permisos, se implementan funciones auxiliares que verifican la autenticación del usuario y recuperan su rol almacenado en el almacenamiento local (localStorage). Los roles definidos son principalmente "admin" y "student", que determinan los niveles de acceso a las distintas rutas.

Se emplea un componente de protección de rutas, denominado ProtectedRoute, que evalúa la validez del token de autenticación y verifica que el rol del usuario se encuentre dentro de los permitidos para cada ruta específica. En caso de incumplimiento, redirige al usuario a la página de inicio de sesión o a la página principal, según corresponda.

La configuración de rutas contempla tanto rutas públicas, accesibles sin restricción, como rutas protegidas, cuyo acceso se limita a usuarios con roles específicos. Las rutas públicas incluyen el acceso a la página principal, inicio de sesión, registro y verificación de certificados. Las rutas protegidas permiten a los administradores emitir certificados y visualizar todos los registros, mientras que los estudiantes pueden acceder únicamente a la obtención de sus certificados.

Finalmente, la estructura visual del componente asegura una disposición coherente de los elementos mediante un contenedor principal con estilos para garantizar una presentación adecuada, incluyendo una barra de navegación fija y un pie de página institucional.

¿Quieres que te ayude también a redactar la explicación de algún otro archivo o componente?

**Dijiste:**

radacta "El archivo App.jsx constituye el componente principal de la aplicación, encargado de gestionar la navegación y el control de acceso mediante la biblioteca React Router, la figura 36 muestra su contenido, en este archivo se importan los componentes visuales y funcionales necesarios, así como los estilos mediante Tailwind CSS. "

**ChatGPT dijo:**

Claro, aquí tienes una redacción más formal y fluida para tu tesis:

El archivo **App.jsx** representa el componente principal de la aplicación, responsable de gestionar la navegación y el control de acceso a través de la biblioteca React Router. Tal como se muestra en la figura 36, en este archivo se importan los componentes visuales y funcionales esenciales, así como los estilos definidos mediante la herramienta Tailwind CSS.

**Dijiste:**

archivo Navbar.jsx "import { Link, useNavigate } from 'react-router-dom'; //DESPLAZAMIENTO DE LAS PAGINAS const Navbar = () => { const token = localStorage.getItem('token'); const role = localStorage.getItem('role'); // 'admin' o 'student' const navigate = useNavigate(); const handleLogout = () => { localStorage.removeItem('token'); localStorage.removeItem('role'); navigate('/login'); }; return ( <nav className="bg-indigo-600 text-white shadow-md"> <div className="max-w-7xl mx-auto px-4 py-3 flex justify-between items-center"> <h1 className="text-xl font-bold">🎓 Instituto Académico de Idiomas </h1> <div className="space-x-4 flex items-center"> <Link to="/" className="hover:underline">Inicio</Link> <Link to="/verify" className="hover:underline">Verificar</Link> {/\* Opciones visibles solo si NO está logeado \*/} {!token && ( <> <Link to="/login" className="ml-2 underline text-yellow-200 hover:text-white">Login</Link> </> )} {/\* Si es admin, puede emitir certificados y consultar todos los certificados \*/} {token && role === 'admin' && ( <Link to="/issue" className="hover:underline">Emitir</Link> )} {token && role === 'admin' && ( <Link to="/all" className="hover:underline">Certificados</Link> )} {token && role === 'admin' && ( <Link to="/register" className="hover:underline">Registrar</Link> )} {/\* Si es estudiante, puede consultar los suyos \*/} {token && role === 'student' && ( <Link to="/get" className="hover:underline">Consultar</Link> )} {/\* Botón de cerrar sesión para todos los logeados \*/} {token && ( <button onClick={handleLogout} className="ml-2 underline text-blue-300 hover:text-white" > Cerrar sesión </button> )} </div> </div> </nav> ); }; export default Navbar;"

**ChatGPT dijo:**

Aquí tienes una redacción formal y adecuada para tu tesis sobre el archivo **Navbar.jsx**:

El archivo **Navbar.jsx** define el componente de la barra de navegación principal de la aplicación, cuya funcionalidad consiste en facilitar el desplazamiento entre las diferentes vistas del sistema. Este componente se adapta dinámicamente según el estado de autenticación y el rol del usuario, utilizando la biblioteca react-router-dom para la navegación y gestión de rutas.

Como se observa en la figura correspondiente, el componente consulta el almacenamiento local del navegador (localStorage) para determinar si existe un token de autenticación y qué rol tiene el usuario (ya sea "admin" o "student"). En función de esta información, se renderizan de manera condicional los enlaces disponibles en la interfaz.

Entre las funcionalidades destacadas, se incluye la opción de cierre de sesión, que elimina el token y el rol del usuario del almacenamiento local y redirige automáticamente a la página de inicio de sesión mediante el hook useNavigate.

La barra de navegación presenta un diseño estilizado utilizando Tailwind CSS, proporcionando una experiencia visual consistente y accesible, con opciones como "Inicio", "Verificar", "Emitir", "Certificados", "Registrar", "Consultar" y "Cerrar sesión", según el perfil del usuario.

El archivo **Home.jsx** define la página principal del sistema, la cual se presenta al usuario al acceder a la ruta raíz de la aplicación. Este componente utiliza React junto con los hooks useState y useEffect para gestionar el estado del rol del usuario, el cual se obtiene desde el almacenamiento local del navegador (localStorage) al momento de la carga del componente.

La interfaz está diseñada con Tailwind CSS y se estructura en tres secciones principales:

1. **Encabezado o banner superior**:  
   Incluye un mensaje destacado que describe la funcionalidad del sistema (“Sistema de Certificados de Idiomas con Blockchain y SmartContracts”) acompañado del logotipo institucional. Esta sección está estilizada con un fondo en color rojo y texto blanco, resaltando visualmente el propósito de la plataforma.
2. **Contenido central**:  
   Se muestra una imagen alusiva a los certificados, seguida de un mensaje informativo que explica brevemente las funciones del sistema: administración, verificación y emisión de certificados. El contenido se encuentra centrado y adaptado para una visualización responsiva.
3. **Mensajes personalizados por rol**:  
   En función del rol del usuario almacenado, se despliega un mensaje contextual:
   * Si el usuario tiene rol **admin**, se muestra un aviso en verde indicando que ha iniciado sesión como administrador.
   * Si el usuario tiene rol **student**, se muestra un mensaje en azul informando que puede consultar y descargar sus certificados.
   * Si no hay sesión iniciada, se despliega un mensaje en rojo indicando que el usuario no ha iniciado sesión.

Este componente cumple una función introductoria y de orientación dentro del sistema, proporcionando al usuario información relevante desde el primer acceso, adaptada a su perfil.

El archivo **Login.jsx** define el componente encargado de gestionar el proceso de inicio de sesión dentro del sistema. Esta vista permite a los usuarios autenticarse utilizando sus credenciales institucionales (correo electrónico y contraseña), y está construida utilizando React con los hooks useState para el manejo del estado y useNavigate para la redirección de rutas.

El formulario, estilizado mediante Tailwind CSS, incluye campos para capturar el correo electrónico y la contraseña. Al enviar los datos, se realiza una solicitud HTTP de tipo **POST** al servidor, utilizando la librería Axios, con el fin de validar las credenciales del usuario.

Una vez autenticado correctamente, el sistema almacena en el navegador el token de acceso y el rol del usuario (ya sea admin o student) en el localStorage, lo que permite gestionar el acceso a funcionalidades protegidas. Según el rol obtenido, el componente redirige al usuario a la vista correspondiente: los administradores son dirigidos a la página de emisión de certificados (/issue), mientras que los estudiantes son redirigidos a la página principal (/).

Además, se incorpora una opción para que los usuarios sin cuenta puedan acceder al formulario de registro, mediante un enlace que redirige a la ruta /register. En caso de error durante el inicio de sesión, el sistema muestra un mensaje de alerta informando la causa del problema.

Este componente es esencial para controlar el acceso al sistema y asegurar que solo usuarios autorizados puedan acceder a las funcionalidades correspondientes a su perfil.

El archivo **Register.jsx** define el componente encargado del registro de nuevos usuarios dentro del sistema. Esta vista permite capturar los datos necesarios mediante un formulario que solicita el correo institucional, la contraseña, la cédula de identidad y el rol del usuario (estudiante o administrador), como se muestra en la Figura correspondiente.

El componente utiliza el hook useState para manejar el estado de los campos del formulario y realiza una validación específica de la cédula ecuatoriana a través de la función validarCedulaEcuatoriana, la cual aplica un algoritmo estándar para verificar su validez estructural.

Una vez completado el formulario, al enviarlo, se ejecuta una solicitud HTTP de tipo **POST** hacia el backend del sistema utilizando la librería Axios. Si los datos ingresados son válidos y el registro es exitoso, el formulario se limpia y se muestra un mensaje de confirmación al usuario.

Este componente es esencial para permitir el acceso controlado al sistema, garantizando que solo usuarios previamente registrados puedan autenticarse e interactuar con las funcionalidades disponibles según su perfil.

El archivo **IssueCertificate.jsx** define el componente responsable de la emisión de certificados digitales dentro del sistema, una funcionalidad exclusiva para usuarios con rol de administrador. Esta vista presenta un formulario que recopila información clave como el nombre del instituto, la facultad, los datos del estudiante (nombre y cédula), el idioma cursado y el nivel alcanzado, tal como se muestra en la Figura correspondiente.

El componente gestiona el estado del formulario utilizando el hook useState y actualiza sus valores conforme el usuario interactúa con los campos. Al momento de enviar el formulario, se realiza una solicitud HTTP de tipo **POST** a través de la biblioteca Axios, incluyendo en los encabezados el token de autenticación previamente almacenado en el navegador, lo cual garantiza que solo usuarios autorizados puedan emitir certificados.

Una vez completado el proceso de emisión, se notifica al usuario mediante un mensaje de confirmación y el formulario se restablece a su estado inicial. Además, se ofrece un acceso directo a la opción de registro de nuevos usuarios, facilitando la gestión administrativa dentro del mismo entorno.

Este componente es crucial para garantizar que los certificados emitidos sean generados de forma controlada y segura, validando previamente los datos ingresados y asociándolos con los registros almacenados en blockchain mediante el backend del sistema.

El archivo **VerifyCertificate.jsx** define el componente encargado de verificar la validez de un certificado digital emitido por el sistema, como se observa en la figura correspondiente. Esta funcionalidad está orientada a cualquier usuario que desee comprobar la autenticidad de un certificado mediante el ingreso del número de cédula del titular.

El componente gestiona internamente dos estados: el número de cédula ingresado y el resultado de la validación (true, false o null). Al hacer clic en el botón “Verificar”, se realiza una solicitud HTTP de tipo **GET** al servidor, el cual consulta en la base de datos y/o blockchain si existe un certificado correspondiente a la cédula ingresada. En función de la respuesta, se muestra un mensaje visual que indica si el certificado es válido o no.

Además, se incluye un botón de “Limpiar” que permite reiniciar el formulario, borrando la cédula ingresada y el resultado de validación, lo que mejora la experiencia del usuario y facilita nuevas consultas.

Este componente cumple un papel fundamental en la transparencia del sistema, ya que permite a terceros verificar de manera inmediata y confiable la autenticidad de los certificados emitidos, fortaleciendo así la confianza en el proceso de certificación digital basado en blockchain.

El archivo **GetCertificate.jsx** define el componente que permite a los usuarios consultar y visualizar los certificados emitidos en el sistema mediante el número de cédula del estudiante, como se muestra en la figura correspondiente. Esta funcionalidad está orientada tanto a los administradores como a los estudiantes, con el objetivo de acceder fácilmente a la información de un certificado específico.

El componente maneja dos estados principales: la **cédula** ingresada por el usuario y el objeto **cert**, que almacena los datos del certificado una vez recuperados. Al hacer clic en el botón "Consultar", se envía una solicitud **GET** al backend para obtener el certificado correspondiente. Si el certificado existe, se despliega en pantalla un resumen detallado de sus atributos: nombre del instituto, facultad, nombre del estudiante, idioma, nivel, fecha de emisión y estado de validez (válido o no válido).

Además, el componente incorpora un botón adicional que permite **descargar el certificado en formato PDF** mediante un enlace directo al servidor, lo cual mejora la accesibilidad y utilidad del sistema para fines institucionales o personales.

Este componente es esencial para la trazabilidad y disponibilidad de los certificados emitidos, permitiendo a los usuarios acceder de forma rápida, visual y confiable a la información almacenada tanto en la base de datos como en blockchain.

El archivo **AllCertificates.jsx** implementa un componente de React encargado de mostrar, buscar, descargar e invalidar certificados emitidos dentro del sistema. Esta interfaz está diseñada principalmente para **usuarios administradores** que requieren una vista global y control centralizado sobre los certificados existentes.

Al cargar el componente, se ejecuta automáticamente una llamada a la API a través del hook useEffect, recuperando todos los certificados almacenados mediante una solicitud **GET** autenticada al endpoint /api/certificates/all. El resultado es almacenado en el estado certificates, y se muestra en una tabla interactiva.

El componente incluye un campo de búsqueda que permite **filtrar los certificados por cédula** en tiempo real, lo que facilita la localización rápida de un registro específico entre muchos.

Cada fila de la tabla muestra la siguiente información: nombre del estudiante, facultad, número de cédula, idioma, nivel, fecha de emisión y acciones disponibles. Las acciones son:

* **Descargar PDF:** Utiliza una petición GET al endpoint /api/certificates/{cedula}/download, recuperando el archivo en formato blob y forzando su descarga en el navegador.
* **Invalidar certificado:** Si el certificado aún es válido, se puede invalidar con un clic. Esto se realiza mediante una petición PUT al endpoint /api/certificates/invalidate/{cedula}. El estado del certificado se actualiza dinámicamente en la interfaz tras una respuesta exitosa.

Se incluye además un sistema de control de errores y mensajes, como alertas en caso de fallas en la conexión o en la autenticación del token JWT.

Este componente cumple un rol fundamental en la **gestión administrativa de los certificados emitidos**, permitiendo mantener el control sobre su validez, accesibilidad y trazabilidad.