Clase 19: Integración de Node.js con MongoDB y React

Sitio:	Centro de E-Learning - UTN.BA	Imprimido por:	Nilo Crespi
Curso:	Curso de Backend Developer - Turno Noche	Día:	Friday, 23 de January de 2026, 10:35
Libro:	Clase 19: Integración de Node.js con MongoDB y React		

Descripción

Objetivos

- Integrar un backend en Node.js con MongoDB mediante Mongoose.
- Crear un frontend en React que consuma los datos del backend.
- Entender el flujo completo desde el manejo de los datos en la base de datos hasta la presentación en el frontend.

Tabla de contenidos

1. Introducción

El siguiente texto está orientado a apoyar la integración de un backend en Node.js con MongoDB y un frontend desarrollado en React. Veremos cómo crear una API REST en Node.js utilizando Express y MongoDB, y cómo React puede consumir dicha API para interactuar con los datos.

2. Configuración del Backend: Node.js + MongoDB

1 . Configuración del Backend: Node.js + MongoDB

Primero, crearemos un **servidor Node.js** que utilice **Express** para manejar rutas y **Mongoose** para interactuar con MongoDB.

1.1 Instalación de dependencias

Para comenzar, instala las dependencias necesarias en el proyecto de Node.js:

```
npm init -y
npm install express mongoose dotenv cors
```

- **Express** : Framework para crear el servidor web.
- **Mongoose** : ORM para interactuar con MongoDB.
- **Dotenv** : Para gestionar variables de entorno (como la conexión a MongoDB).
- **CORS** : Habilita la comunicación entre el frontend (React) y el backend (Node.js) en diferentes dominios.

1.2 Configurar el Servidor

Crea un archivo server.js que configures el servidor con las rutas necesarias para manejar las solicitudes.

```
 const express = require('express');
const mongoose = require('mongoose');
const cors = require('cors');  const
dotenv = require('dotenv');  const
app = express();  dotenv.config();

 // Middleware  app.use(express.json()); // Para parsear JSON en
las solicitudes  app.use(cors()); // Para permitir acceso desde
el frontend
 // Conexión a la base de datos
 mongoose.connect(process.env.MONGODB_URI)
   .then(() => console.log('Conectado a MongoDB'))
   .catch(err => console.error('Error al conectar a MongoDB:', err));

 // Definir rutas  app.use('/api/hechizos',
require('./routes/hechizoRoutes'));

 // Iniciar servidor  const PORT = process.env.PORT ||
5000;  app.listen(PORT, () => {  console.log(`Servidor
corriendo en el puerto ${PORT}`);
}) ;
```

1.3 Crear el Modelo con Mongoose

Definimos el modelo **Hechizo** que será almacenado en MongoDB. Crea un archivo models/ HechizoModel.js con el siguiente código:

```
 const mongoose = require('mongoose');

 const hechizoSchema = new mongoose.Schema({
    nombre: { type: String, required: true },
    descripcion: { type: String, required: true },
    nivel: { type: Number, required: true }
 });  module.exports = mongoose.model('Hechizo',

hechizoSchema);
```

1.4 Rutas y Controladores

Crear Rutas

Crea el archivo routes/hechizoRoutes.js para definir las rutas que manejarán las operaciones CRUD sobre los hechizos.

```
 const express = require('express');  const router = express.Router();
const hechizoController = require('../controllers/hechizoController');

 router.get('/',      hechizoController.obtenerHechizos);

router.post('/',         hechizoController.crearHechizo);

router.put('/:id',  hechizoController.actualizarHechizo);

router.delete('/:id', hechizoController.eliminarHechizo);

module.exports = router;
```

Crear Controladores

Crea el archivo controllers/hechizoController.js para manejar la lógica de cada una de las operaciones CRUD.

```
 const Hechizo = require('../models/HechizoModel');

 // Obtener todos los hechizos
exports.obtenerHechizos = async (req, res) => {
try {  const hechizos = await Hechizo.find();
res.json(hechizos);
    }  catch (error) {  res.status(500).json({ message: "Error al obtener
        los hechizos" });
    }
} ;

 // Crear un nuevo hechizo
exports.crearHechizo = async (req, res) => {
const nuevoHechizo = new Hechizo(req.body);
try {  await nuevoHechizo.save();
res.status(201).json(nuevoHechizo);
    }  catch (error) {  res.status(500).json({ message: "Error al
        crear el hechizo" });
     }
} ;
```

```
 // Actualizar un hechizo  exports.actualizarHechizo = async (req, res) => {  try {  const
hechizoActualizado  =  await  Hechizo.findByIdAndUpdate(req.params.id,  req.body,
{  new: true });
        res.json(hechizoActualizado);
    }  catch (error) {  res.status(500).json({ message: "Error al
        actualizar el hechizo" });
     }
 };

 // Eliminar un hechizo  exports.eliminarHechizo =
async (req, res) => {  try {  await
Hechizo.findByIdAndDelete(req.params.id);  res.json({
message: "Hechizo eliminado" });
     } catch (error) {  res.status(500).json({ message: "Error al
        eliminar el hechizo" });
     }
 };
```

3. Configuración del Frontend: React

2. Configuración del Frontend: React

2.1 Crear proyecto React

Crear el proyecto React, dirigirnos a la carpeta e iniciarlo.

```
 npm create vite@latest
frontend  cd frontend  npm
start
```

2.2 Consumir la API con Axios

Instala **Axios** para hacer las solicitudes HTTP al backend desde React:

```
 npm install axios
```

2.3 Crear componente para Listar Hechizos

En el directorio src , crea una carpeta components y dentro un archivo HechizoList.js para mostrar los hechizos obtenidos desde la API.

```
import React, { useState, useEffect } from 'react';
import axios from 'axios';

 const HechizoList = () => {  const [hechizos,
    setHechizos] = useState([]);

     useEffect(() => {
        axios.get('http://localhost:5000/api/hechizos')
             .then(response => {
                setHechizos(response.data);
            })
             .catch(error => {  console.error('Error fetching
    hechizos:', error); }) ;  }, []);
```

```
        return (
            <div>
                <h1>Lista de Hechizos</h1>
                <ul>
                    {hechizos.map(hechizo => (
                        <li key={hechizo._id}>
                            <strong>{hechizo.nombre}</strong>:
{hechizo.descripcion}  (Nivel: {hechizo.nivel})
                        </li>
                    ))}
                </ul>
            </div>
        );
    };

    export default HechizoList;
```

2.4 Mostrar los Hechizos en la página principal Edita src/App.js para mostrar el componente de hechizos:

```
import React from 'react';
import HechizoList from './components/HechizoList';

function App() {
  return (
    < div className="App">
      < HechizoList />
     </div>
  ) ;
}

export default App;
```

4. Habilitar CORS en el Backend

3. Habilitar CORS en el Backend

Es importante habilitar CORS en el servidor de Node.js para permitir que el frontend (React) pueda comunicarse con el backend.

En el archivo server.js , asegúrate de que el middleware de CORS esté habilitado:

```
const cors = require('cors');
app.use(cors());
```

5. Ejercicio práctico

Ejercicio práctico

Objetivo

El objetivo del ejercicio es implementar la funcionalidad de **crear** y **eliminar un elemento** desde el frontend, usando formularios en React para enviar datos al backend mediante Axios.

Requerimientos:

1. **Crear un formulario** en React para agregar un nuevo elemento.
2. **Enviar la solicitud POST** al backend con los datos del nuevo elemento.

3 . **Implementar botones** en cada elemento de la lista para eliminarlo. La acción de eliminar debe enviar una solicitud **DELETE** al backend.

6. Anexo

Anexo

Comparativa entre Fetch API y Axios: ¿Cuál elegir?

Las opciones más populares para manejar solicitudes HTTP en JavaScript son la **Fetch API** y **Axios** . Mientras que Fetch es una solución nativa ligera que no requiere instalaciones adicionales, Axios es una biblioteca poderosa con características avanzadas que simplifican el manejo de errores y respuestas. A continuación exploramos las diferencias clave entre ambas herramientas y brindamos recomendaciones sobre cuándo utilizar cada una dependendiendo el las necesidades del proyecto.

Comparación entre Fetch y Axios

Fetch API:

- Introducida en **2015** , es nativa de JavaScript y compatible con la mayoría de los navegadores modernos.
- Ideal para proyectos simples donde no se requieren características avanzadas. Es ligera y fácil de implementar sin dependencias externas, pero el manejo de errores y respuestas debe hacerse manualmente.

Axios:

- Creada en **2014** , es una biblioteca de terceros que se debe instalar mediante un gestor de paquetes como npm.
- Perfecta para aplicaciones más complejas que requieren interceptores, manejo robusto de errores y funcionalidades adicionales como la cancelación de solicitudes. Ofrece una API más sencilla para trabajar con datos.

Conclusión

Ambas opciones tienen sus ventajas y desventajas. En general:

- Usa **Fetch** para proyectos simples que no necesitan funcionalidades avanzadas.
- Usa **Axios** para aplicaciones que requieren un manejo más robusto y fácil de las solicitudes HTTP.