Creacion y desarrollo de la API con base de datos y frontend basico

Integrantes

Hemer Santiago Perez Nieves, Hugo Andres Forero, Miguel Angel Urrea Camacho, Miguel Angel Pinilla Baez y Yesid Alejandro Varela Alfa

Institución Educativa, Universidad Manuela Beltran Docente,

Diana Toquica

Asignatura, Ingeniería Web

Facultad, Ingeniería de Software

Bogota D.c

4 de Febrero, 2025

¿Cómo has definido los requisitos funcionales para tu API (qué operaciones necesita hacer el sistema)?

-Crear un libro (POST).

-Leer libros (GET).

-Actualizar un libro (PUT).

-Eliminar un libro (DELETE).

-Agregar libros al carrito (POST).

-Leer el carrito (GET).

-Eliminar libros del carrito (DELETE).

-Procesar la compra (POST).

-Ver historial de pedidos (GET).

-Estas son algunas de las operación que estamos manejando en el proyecto por lo que podemos concluir que no son muchas las que necesitamos para nuestro proyecto

¿Qué requerimientos no funcionales consideraste (por ejemplo, seguridad, rendimiento, escalabilidad)?:

- **Seguridad:** Implementación de autenticación y autorización con JWT.
- **Rendimiento:** Uso de paginación y optimización de consultas SQL.
- **Escalabilidad:** Uso de PostgreSQL para soportar grandes volúmenes de datos, con posible expansión a bases distribuidas.

  Al ser no funcionales no se les debe restar ninguna importancia ya que estos son los que mas pegan para el usuario final, entonces usamos los del ejemplo mismo por que los consideramos los mas importantes

---

¿Qué casos de uso principales estás cubriendo con las rutas de tu API?:

- **Usuario final:** Ver libros, agregar y eliminar libros del carrito, procesar compras.
- **Administrador:** Gestionar libros (añadir, editar, eliminar)

¿Por qué elegiste MySql como base de datos? ¿Qué ventajas ofrece frente a otras opciones (por ejemplo, SQLite)?:

- MySQL es una base de datos relacional avanzada con soporte de transacciones ACID y optimización de consultas.
- Ofrece mayor capacidad de escalabilidad y concurrencia frente a SQLite, que es más adecuado para aplicaciones pequeñas.

¿Cómo estructuraste las tablas de la base de datos? ¿Cuáles son las relaciones entre las entidades de los libros (por ejemplo, títulos, autores, precio)?:

- **Tablas principales:** libros, autores, categorias, usuarios, carrito, pedidos, **Relaciones:** Un libro tiene un autor y una categoría; un usuario puede tener múltiples libros en su carrito y en pedidos. Un pedido puede tener múltiples libros.

¿Cómo vas a validar la entrada de datos para asegurarte de que las operaciones CRUD funcionen correctamente (por ejemplo, qué campos son obligatorios)?:

- **Campos obligatorios:**
  - Creación de libros: titulo, autor_id, precio, stock, categoria_id.
  - Registro de usuario: email, contraseña, nombre.
  - Carrito: usuario_id, libro_id, cantidad.
  - Compra: usuario_id, libros (detalle del pedido).
- **Validación:** Verificación de que los campos como precio y cantidad sean numéricos y positivos; asegurar que el email no esté duplicado al registrar un usuario.

¿Cómo aseguras que las consultas SQL (como SELECT, INSERT, UPDATE, DELETE) se ejecuten correctamente con el proyecto?:

- Uso de **transacciones** para asegurar que las operaciones sean atómicas y consistentes.
- Realización de **consultas unitarias** para verificar que funcionan correctamente.
- Uso de **ORMs** como Sequelize o TypeORM para simplificar la interacción con la base de datos de manera segura.

2. Implementación de la API CRUD con MySQL

¿Qué medidas tomaste para manejar posibles errores en las rutas de la API?
Rta: Para manejar los errores utilizamos bloques try-catch en las rutas de la API. Por ejemplo en el caso de que un libro no este disponible, la API tira el código de error 404 not found y un mensaje indicando que no existe o no esta disponible. Tambien se manejan errores de conexión a la base de datos y errores de validación de datos enviados por el cliente.

¿Qué tipo de respuesta esperas de la API para cada tipo de solicitud?

- GET: Devuelve la lista de libros o un libro específico en formato JSON.
- POST: Devuelve un mensaje de éxito y los datos del libro creado.
- PUT: Devuelve un mensaje de éxito y los datos del libro actualizado.
- DELETE: Devuelve un mensaje confirmando que el libro fue eliminado.

¿Cómo gestionas la autenticación y seguridad en tu API (si aplica)?
En esta versión básica de la API aun no se ha implementado la autenticación, ya que nos hemos enfocado en CRUD. Pero en cuanto a la seguridad, se busca usar consultas parametrizadas para evitar inyecciones SQL y se validan los datos antes de insertarlos o actualizarlos en la base de datos.

3. Integración Frontend-Backend

¿Cómo haces que el frontend (HTML, CSS, JS) se comunique con el backend (Express y MySQL)?
El frontend se comunica con el backend utilizando la función fetch de JavaScript para realizar peticiones HTTP (GET, POST, PUT, DELETE) a las rutas de la API que están conectadas con la base de datos.

¿Qué métodos usas para realizar peticiones HTTP en el frontend?
Utilicé fetch porque es una función nativa de JavaScript, no necesita la instalación de librerías externas y es muy práctica para proyectos sencillos como este.

¿Cómo gestionas los errores que pueden surgir en el frontend al interactuar con la API?
Para manejar los errores en el frontend utilizo el método .catch() en las promesas de fetch. Además, muestro mensajes o alertas al usuario en caso de que ocurra algún error, por ejemplo: "Error al cargar los libros" o "No se pudo realizar la operación".

Qué tipo de pruebas realizaste para asegurarte de que la API funcione correctamente? (Por ejemplo, pruebas unitarias o de integración)
Se realizaron pruebas unitarias y pruebas funcionales en la API. Las pruebas unitarias se enfocaron en verificar que las funciones individuales de la API (como las rutas de los endpoints, la lógica de negocio y las interacciones con la base de datos) funcionen como se esperaba. Las pruebas funcionales se centraron en evaluar el comportamiento general de la API, asegurando que los endpoints funcionaran de acuerdo con los requisitos, incluyendo las respuestas correctas para las solicitudes GET, POST, PUT y DELETE. Además, se ejecutaron pruebas de pruebas de integración para validar que diferentes partes de la API interactúan correctamente entre sí y con los servicios externos.

¿Cómo probaste las funciones CRUD en Postman o Insomnia? ¿Qué resultados esperabas y

cómo validaste que fueron correctos?
Para probar las funciones CRUD (Crear, Leer, Actualizar, Eliminar) en Postman, se realizaron las siguientes acciones:

Crear (POST): Se envió una solicitud con datos válidos para crear un nuevo recurso. Esperábamos una respuesta con un código de estado 201 (creado) y los datos del recurso creado. Validamos que la respuesta tuviera los datos correctos y que el recurso fuera realmente añadido a la base de datos.

Leer (GET): Se realizaron solicitudes para obtener recursos existentes. Esperábamos una respuesta con código 200 (OK) y los datos correctos del recurso solicitado. Validamos que los datos devueltos coincidieran con los que estaban almacenados.

Actualizar (PUT/PATCH): Se enviaron solicitudes para modificar un recurso existente. Esperábamos una respuesta con código 200 (OK) y los datos modificados en el cuerpo de la respuesta. Validamos que la actualización fuera reflejada correctamente en la base de datos.

Eliminar (DELETE): Se enviaron solicitudes para eliminar un recurso. Esperábamos una respuesta con código 204 (sin contenido), y verificamos que el recurso se haya eliminado correctamente de la base de datos.

Se verificaron todos los casos de respuesta para asegurarse de que las acciones fueran exitosas y las respuestas estuvieran en el formato correcto, según el estándar de la API.

¿Cómo verificaste que el frontend interactúa correctamente con el backend?
La interacción entre el frontend y el backend se verificó mediante pruebas manuales y automáticas. Se realizó un seguimiento de las solicitudes que hacía el frontend a la API y se validaron las respuestas en el navegador utilizando las herramientas de desarrollo (como la consola de JavaScript y las redes). Además, se utilizaron herramientas como Postman para simular peticiones directamente a la API y verificar que las respuestas fueran las esperadas.

También se realizaron pruebas de integración para asegurar que el frontend recibiera y procesara correctamente los datos del backend, verificando que las funciones de frontend, como la actualización de la interfaz de usuario, funcionaran adecuadamente con los datos recibidos.

¿Qué harías si se encuentran errores de integración entre el frontend y el backend?
Si se encuentran errores de integración entre el frontend y el backend, seguiría estos pasos:

Revisión de Logs: Revisa los logs tanto del frontend como del backend para identificar posibles errores de comunicación o problemas en las respuestas de la API.

Depuración: Realizar una depuración exhaustiva para identificar posibles errores en el manejo de los datos entre ambos lados. Esto incluye revisar las rutas de las solicitudes, los parámetros enviados y las respuestas.

Verificación de formato de datos: Verificar que los datos enviados y recibidos entre el frontend y el backend sean los esperados y que no haya problemas de deserialización o formato incorrecto.

Pruebas adicionales: Utilizar herramientas como Postman para verificar que la API devuelva las

respuestas correctas en todos los casos de uso.

Revisión de configuración: Asegurarse de que la configuración de CORS y otros parámetros de red estén correctamente configurados para permitir la interacción entre el frontend y el backend.

Comunicación con el equipo: Si el error persiste, se debe comunicar con el equipo de backend para resolver problemas específicos de la API, como la corrección de rutas o lógica del servidor.

¿Cómo manejas los errores comunes que pueden ocurrir durante las pruebas (por ejemplo, registros no encontrados, datos inválidos)?
Para manejar errores comunes como registros no encontrados o datos inválidos, se toman las siguientes acciones:

Manejo de Errores en la API: Se implementan mensajes de error claros y adecuados con códigos de estado HTTP apropiados (por ejemplo, 404 para "No encontrado", 400 para "Solicitud incorrecta", etc.) para que tanto el frontend como los desarrolladores puedan comprender fácilmente el problema.

Validación de Entrada: Se valida la entrada en el servidor para asegurarse de que los datos enviados sean correctos antes de procesarlos (por ejemplo, validación de formato, tipo de dato, longitud, etc.).

Mensajes de Error Claros: Se proporcionan mensajes de error específicos en el cuerpo de la respuesta para facilitar la identificación del problema.

Pruebas de Casos Negativos: Se realizan pruebas adicionales para verificar que la API maneje correctamente casos de datos inválidos, como entradas vacías, valores fuera de rango, o datos que no existen en la base de datos.

Registros y Logs: Se mantienen registros detallados de las pruebas y de los errores encontrados, lo que facilita la identificación y resolución de problemas.

CODIFICACION Y EJECUCION DE LA BASE DE DATOS

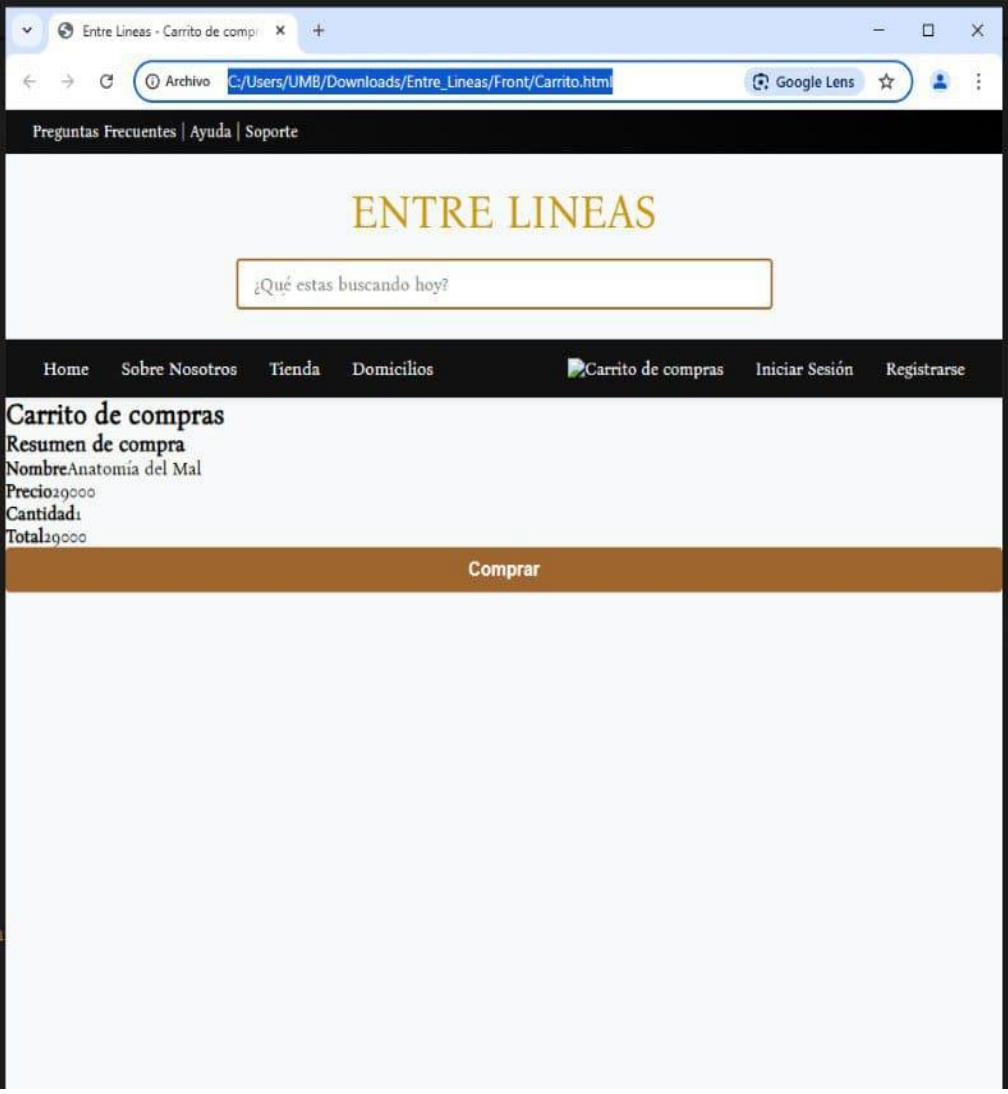
```
const resumenTotal = document.getElementById("total-price");
const resumenCantidad = document.getElementById("total-items");
const envio = document.getElementById("shipping-cost");

const carrito = {

    obtenerCarrito: function () {
        return JSON.parse(localStorage.getItem("carrito")) || [];
    },


    guardarCarrito: function (carrito) {
        localStorage.setItem("carrito", JSON.stringify(carrito));
    },


    agregarAlCarrito: function (id, nombre, precio, autor) {
        let carritoActual = this.obtenerCarrito();
        let producto = carritoActual.find(item => item.id === id);

        if (producto) {
            producto.cantidad++;
        } else {
            carritoActual.push({ id, nombre, precio, autor, cantidad: 1 });
        }

        this.guardarCarrito(carritoActual);
        this.mostrarCarrito();
        alert(`"${nombre}" añadido al carrito`);
    },


    eliminarDelCarrito: function (id) {
        let carritoActual = this.obtenerCarrito().filter(item => item.id !== id);
        this.guardarCarrito(carritoActual);
        this.mostrarCarrito();
    },


    vaciarCarrito: function () {
        localStorage.removeItem("carrito");
        this.mostrarCarrito();
    },


    mostrarCarrito: function () {
        const carritoActual = this.obtenerCarrito();
        const contenedor = document.getElementById("cart-items");
        const resumenTotal = document.getElementById("total-price");
        const resumenCantidad = document.getElementById("total-items");
        const envio = document.getElementById("shipping-cost");

        if (!contenedor) return;

        contenedor.innerHTML = "";

        if (carritoActual.length === 0) {
            contenedor.innerHTML = "<p>El carrito está vacío</p>";
            resumenTotal.textContent = "$0.00";
```

```
    mostrarCarrito: function () {
        const carritoActual = this.obtenerCarrito();
        const contenedor = document.getElementById("cart-items");
        const resumenTotal = document.getElementById("total-price");
        const resumenCantidad = document.getElementById("total-items");
        const envio = document.getElementById("shipping-cost");

        if (!contenedor) return;

        contenedor.innerHTML = "";

        if (carritoActual.length === 0) {
            contenedor.innerHTML = "<p>El carrito está vacío</p>";
            resumenTotal.textContent = "$0.00";
            resumenCantidad.textContent = "0";
            envio.textContent = "$0.00";
            return;
        }

        let total = 0;
        let cantidadTotal = 0;

        carritoActual.forEach(producto => {
            const item = document.createElement("div");
            item.className = "cart-item";
            const subtotal = producto.precio * producto.cantidad;
            total += subtotal;
            cantidadTotal += producto.cantidad;

            item.innerHTML = `
                <p><strong>${producto.nombre}</strong> x ${producto.cantidad}</p>
                <p>Autor: ${producto.autor}</p>
                <p>Precio: $${producto.precio.toLocaleString('es-CO')}</p>
                <p>Subtotal: $${subtotal.toLocaleString('es-CO')}</p>
                <button onclick="carrito.eliminarDelCarrito(${producto.id})">Eliminar</button>
            `;

            contenedor.appendChild(item);
        });


        resumenTotal.textContent = `$${total.toFixed(2)}`;
        resumenCantidad.textContent = cantidadTotal;
        envio.textContent = total > 0 ? "$5.00" : "$0.00";


        const vaciarBtn = document.createElement("button");
        vaciarBtn.textContent = "Vaciar Carrito";
        vaciarBtn.onclick = () => this.vaciarCarrito();
        contenedor.appendChild(vaciarBtn);
    }
};


document.addEventListener("DOMContentLoaded", () => carrito.mostrarCarrito());
```

carrito.js | conexion.js | login.js | Libros.txt | Carrito.html | Tienda.html

Back > carrito.js > carrito > mostrarCarrito > carritoActual.forEach() callback

```
const carrito = {
    eliminarDelCarrito: function (id) {
        this.guardarCarrito(carritoActual);
        this.mostrarCarrito();
    },


    vaciarCarrito: function () {
        localStorage.removeItem("carrito");
        this.mostrarCarrito();
    },


    mostrarCarrito: function () {
        const carritoActual = this.obtenerCarrito();
        const contenedor = document.getElementById("cart-items");
        const resumen = document.getElementById("cart-summary-details");

        if (!contenedor || !resumen) return;

        contenedor.innerHTML = "";
        resumen.innerHTML = `
            <p><strong>Nombre</strong></p>
            <p><strong>Precio</strong></p>
            <p><strong>Cantidad</strong></p>
            <p><strong>Total</strong></p>
        `;

        if (carritoActual.length === 0) {
            contenedor.innerHTML = "<p>El carrito está vacío</p>";
            return;
        }

        carritoActual.forEach(producto => {
            const subtotal = producto.precio * producto.cantidad;

            // Agregar fila en el resumen
            resumen.innerHTML += `
                <p>${producto.nombre}</p>
                <p>$${producto.precio.toLocaleString('es-CO')}</p>
                <p>${producto.cantidad}</p>
                <p>$${subtotal.toLocaleString('es-CO')}</p>
            `;

            // Mostrar también en lista del carrito si quieres
            const item = document.createElement("div");
            item.className = "cart-item";
            item.innerHTML = `
                <p>${producto.nombre} x ${producto.cantidad}</p>
                <button onclick="carrito.eliminarDelCarrito(${producto.id})">El
            `;
            contenedor.appendChild(item);
        });
    }
};


document.addEventListener("DOMContentLoaded", () => carrito.mostrarCarrito());
```



REGISTRO BASE DE DATOS