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".

4.Pruebas

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 esperaba 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.

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.

ESPECIFICACION DE REQUISITOS:

Funcionales:

1.

Especificación de requerimientos funcionales	
Requerimiento Funcional N°: 01	**Nombre**
RQF 01	Crear Cuenta
Tipo:	**Prioridad:**
Necesario	Alta
Descripción:	
El sistema debe permitir al usuario crear una nueva cuenta para el uso dentro de la página debe almacenar el usuario en la base de datos	

2.

Especificación de requerimientos funcionales	
Requerimiento Funcional N°:02	**Nombre**
RQF 02	Iniciar Sesión
Tipo:	**Prioridad:**
Necesario	Alta
Descripción:	
El sistema permitirá ingresar al aplicativo digitando las credenciales de correo y contraseña	

3.

Especificación de requerimientos funcionales	
Requerimiento Funcional N°:03	**Nombre**
RQF 03	Editar perfil del usuario
Tipo:	**Prioridad:**
Necesario	Media
Descripción:	
El sistema permite al usuario ingresar a las configuraciones de su perfil, permitiendo modificar diferentes variables correspondientes al usuario (nombre, correo, contraseña) y variables dependientes del proceso de compra.	
Criterios de aceptación	
• Los nuevos datos deben cumplir el mismo formato que los ingresados previamente • Los datos deben ser actualizados en la base de datos del sistema • Mostrar un mensaje de confirmación de actualización de datos • El nuevo correo no debe estar vinculado con ninguna otra cuenta del sistema	

Nota: Esta tabla Especifica los criterios para actualizar datos dentro de el sistema

4.

Requerimiento funcional n°4 [Barra de búsqueda]

Especificación de requerimientos funcionales	
Requerimiento Funcional N°:04	**Nombre**
RQF 04	Barra de búsqueda
Tipo:	**Prioridad:**
Necesario	Alta
Descripción:	
El sistema debe permitir consultas de libros basado en el nombre del mismo, mostrando la descripción.	
Criterios de aceptación	
• El campo de búsqueda no puede enviarse vacío • Realizar la consulta en la base de datos • Mostrar detalles del libro, o libros similares • En dado de no encontrarlo mostrar un mensaje	

5.

Requerimiento funcional n° 5[Lista de libros (Admin)]

Especificación de requerimientos funcionales	
Requerimiento Funcional N°:05	**Nombre**
RQF 05	Lista de libros (Administrador)
Tipo:	**Prioridad:**
Necesario	Alta
Descripción:	
El administrador podrá revisar las listas de libros en inventario existente, en ese caso podrá consultarlos, modificarlos o eliminarlos en caso de ser necesarios, todo visualizado en un lista de libros.	
Criterios de aceptación	
- El sistema deberá confirmar la existencia de los libros en la base de datos. - Al actualizar información se deberá actualizar en la base de datos.	

6.

Requerimiento funcional n° 6 [Filtros]

Especificación de requerimientos funcionales	
Requerimiento Funcional N°:06	**Nombre**
RQF 06	Filtros
Tipo:	**Prioridad:**
Necesario	Media
Descripción:	
El sistema permitirá realizar consultas en la barra de búsqueda por medio de diferentes filtros generados, en ese caso se buscarán por medio de Autor, Año, Género entre otros…	
Criterios de aceptación	
• El sistema deberá generar grupos por medio de los filtros escogidos. • El sistema contrastará en la base de datos las consultas por medio de filtros.	

7.

Requerimiento funcional n° 7 [Crear Filtros]

Especificación de requerimientos funcionales	
Requerimiento Funcional N°:07	**Nombre**
RQF 07	Crear categorías/filtros (Administrador)
Tipo:	**Prioridad:**
Necesario	Media
Descripción:	
El administrador podrá generar filtros en la búsqueda de libros para el usuario, permitiendo tener mejor acceso a las búsquedas y consultas para el usuario.	
Criterios de aceptación	
- El sistema deberá revisar si no está creado el filtro en la base de datos antes de agregarlo. - El sistema avisará si es posible o no crear el filtro de búsqueda para el usuario, esto con mensajes por pantalla.	

8.

Requerimiento funcional n° 8 [Recomendaciones]

Especificación de requerimientos funcionales	
Requerimiento Funcional N°:08	**Nombre**
RQF 08	Recomendaciones
Tipo:	**Prioridad:**
Necesario	Medio
Descripción:	
El sistema identificará los libros más vendidos, los cuales mostrará de primeras a los usuarios en forma de recomendaciones de compra.	
Criterios de aceptación	
- El sistema deberá revisar los libros más comprados para poder dejarlos en recomendaciones para todos los usuarios.	

9.

Requerimiento funcional n° 9 [Información del libro]

Especificación de requerimientos funcionales	
Requerimiento Funcional N°:09	**Nombre**
RQF 09	Información del libro
Tipo:	**Prioridad:**
Necesario	Alta
Descripción:	
El sistema tendrá los libros guardados, en ese caso, aloja la información de cada uno para mostrar su sinopsis y diferentes detalles del libro para poder mostrar al usuario la descripción del libro.	
Criterios de aceptación	
- El sistema no permitirá dejar espacio en blanco para la descripción del libro. - El sistema deberá tener límite de palabras para la creación de la descripción de los libros.	

10.

Requerimiento funcional n°10 [Gestión de libros (Usuario)]

Especificación de requerimientos funcionales	
Requerimiento Funcional N°:10	**Nombre**
RQF010	Gestión de libros (Usuario)
Tipo:	**Prioridad:**
Necesario	Alta
Descripción:	
El administrador podrá gestionar el catálogo de libros visibles en la biblioteca, permitiendo modificar el catálogo que podrá ser mostrado para el usuario y disponible para su compra	
Criterios de aceptación	
- El sistema debe permitir agregar, editar y eliminar libros en el catálogo - La modificación en el catálogo se debe actualizar en la base de datos - El catálogo actualizado será mostrado a los usuarios al ingresar a la página	

11.

Requerimiento funcional n° 11 [Información del libro (Usuario)]

Especificación de requerimientos funcionales	
Requerimiento Funcional N°:11	**Nombre**
RQF011	Información del libro (Usuario)

Tipo:	**Prioridad:**
Necesario	Alta
Descripción:	
Muestra los detalles de un libro, tales como : ● Título ● Autor ● Género ● Número de ejemplares	
Criterios de aceptación	
- La página deben mostrar los detalles de cualquier libro registrado en la biblioteca	

12.

Requerimiento funcional n° 12 [Editar detalles del libro (Administrador)]

Especificación de requerimientos funcionales	
Requerimiento Funcional N°:12	**Nombre**
RQF012	Editar detalles del libro (Administrador)
Tipo:	**Prioridad:**
Necesario	Alta
Descripción:	
Permite al administrador modificar la información de un libro en el catálogo, más no eliminarlo	
Criterios de aceptación	
- Solo los administradores pueden editar los detalles de lo libros - Los detalles del libro deben ser actualizados en la base de datos	

13.

Requerimiento funcional n° 13 [Agregar libros (Administrador)]

Especificación de requerimientos funcionales	
Requerimiento Funcional N°:13	**Nombre**
RQF013	Agregar libros (Administrador)
Tipo:	**Prioridad:**
Necesario	Alta
Descripción:	
El administrador ha de poder agregar libros a la base de datos, esto incluye, la descripción del libro, su título, autor, categorías, género, imagen de portada o a mostrar de libro	

Criterios de aceptación
- El sistema ha de actualizar y subir la base de datos la información de libro a crear por el administrador

14.

Requerimiento funcional n° 14 [Eliminar libros (Administrador)]

Especificación de requerimientos funcionales	
Requerimiento Funcional N°:14	**Nombre**
RQF014	Eliminar libros (Administrador)
Tipo:	**Prioridad:**
Necesario	Alta
Descripción:	
El administrador ha de poder acceder a la base de datos de la librería y eliminar algún libro de ésta, en caso de que ya no desee venderlo o necesitarlo en la librería	
Criterios de aceptación	
- El sistema ha de borrar totalmente la información del libro deseado borrar por el administrador	

15.

Requerimiento funcional n° 15 [Cantidad a comprar]

Especificación de requerimientos funcionales	
Requerimiento Funcional N°:15	**Nombre**
RQF015	Cantidad a comprar
Tipo:	**Prioridad:**
Necesario	Alta
Descripción:	

El usuario al momento de comprar un libro, ha de poder seleccionar la cantidad a comprar antes de agregarla al carrito de compras
Criterios de aceptación
- El sistema ha de actualizar el carrito con la cantidad indicada del usuario

16.

Especificación de requerimientos funcionales	
Requerimiento Funcional N°:16	**Nombre**
RQF016	Carrito de compras
Tipo:	**Prioridad:**
Necesario	Alta
Descripción:	
El usuario ha de poder visualizar el contenido dentro de su carrito de compras, al igual que la posibilidad de editar el contenido de esta misma	
Criterios de aceptación	
- El sistema ha de actualizar el carrito de compras del usuario - El sistema ha de guardar la información del carrito de compras para luego ser procesado en la compra	

17.

Requerimiento funcional n° 17 [Entrega]

Especificación de requerimientos funcionales	
Requerimiento Funcional N°:17	**Nombre**
RQF017	Entrega
Tipo:	**Prioridad:**
Necesario	Alta

Descripción:
El usuario ha de poder ver el estado de su pedido, si este anda en el almacén, en camino, fue regresado, etc.
Criterios de aceptación
- El administrador ha de poder cambiar el estado de entrega del pedido - El sistema ha de actualizar la información la cual únicamente va a ser visible para el usuario

18.

Requerimiento funcional n° 18 [Historial de compra]

Especificación de requerimientos funcionales	
Requerimiento Funcional N°:18	**Nombre**
RQF018	Historial de compra
Tipo:	**Prioridad:**
Necesario	Alta
Descripción:	
El usuario ha de poder ver su historial de compras que ha tenido en la página	
Criterios de aceptación	
- El sistema ha de mostrar todas las compras que ha tenido el usuario	

19.

Requerimiento funcional n° 19 [Buscar en el registro de compra (Administrador)]

Especificación de requerimientos funcionales	
Requerimiento Funcional N°:19	**Nombre**
RQF019	Buscar en el registro de compra (Administrador)
Tipo:	**Prioridad:**
Necesario	Alta

Descripción:
El administrador ha de poder consultar los registros de las compras de los usuarios en orden de más reciente a menos reciente, en un lapso de 1 mes y/o buscar por ID del recibo
Criterios de aceptación
- El sistema ha de cargar los últimos registros de compras - El sistema ha de permitir buscar por un registro en específico con su ID

20.

Requerimiento funcional n° 20 [Reseñas]

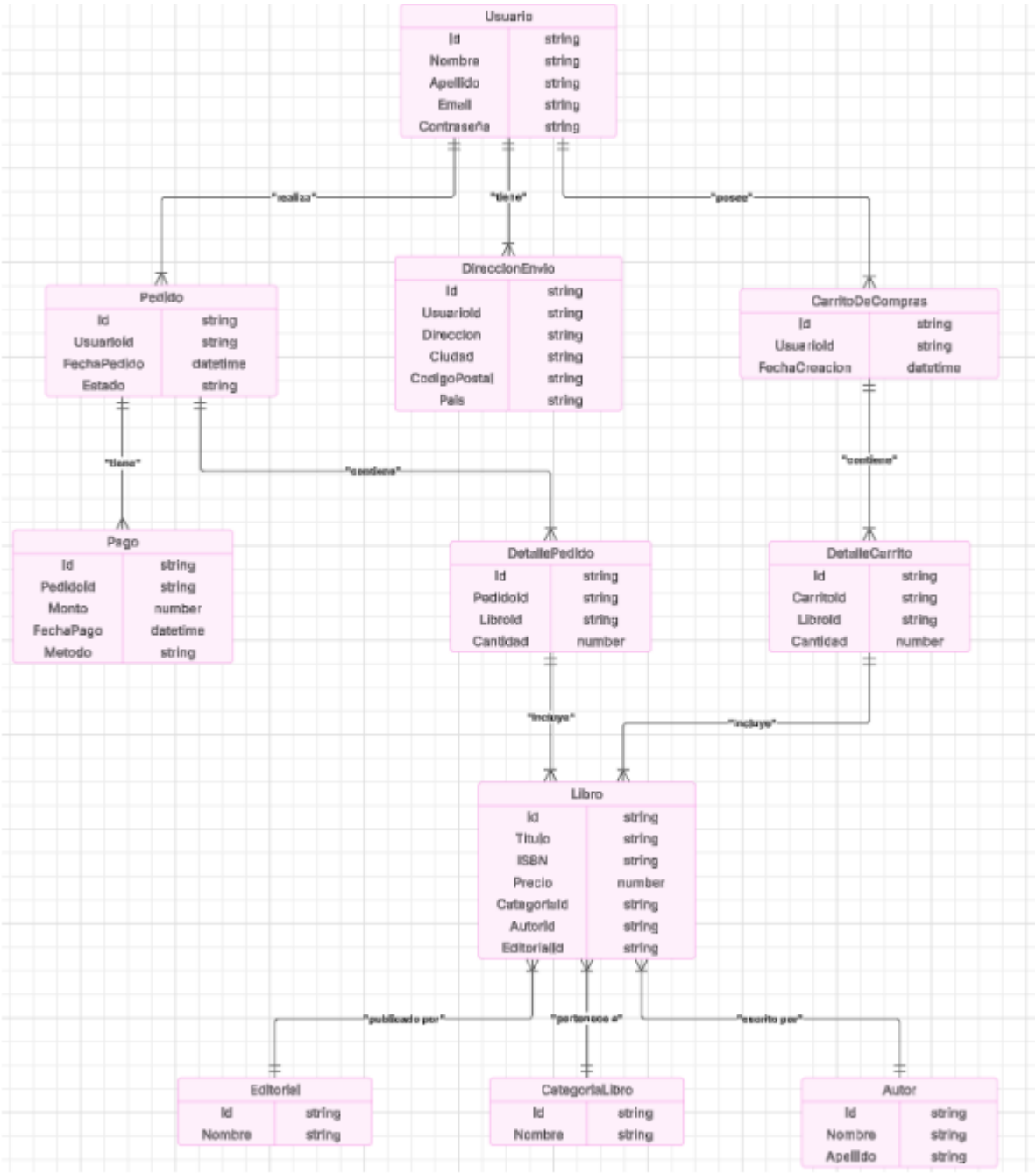
Especificación de requerimientos funcionales	
Requerimiento Funcional N°:20	**Nombre**
RQF020	Reseñas
Tipo:	**Prioridad:**
Necesario	Alta
Descripción:	
El usuario ha de poder dejar una reseña en el libro deseado o que esté mirando, solo si este tiene una cuenta dentro de la página y su sesión está activa en el momento de crear la reseña	
Criterios de aceptación	
- La reseña digitada no debe ser vacía - La reseña del usuario se ha de guardar correctamente en la base de datos - La reseña del usuario se ha de mostrar a demás usuarios inclusive a él mismo	

21.

Requerimiento funcional n° 21 [Procesamiento de pagos]

Especificación de requerimientos funcionales	
Requerimiento Funcional N°:21	**Nombre**
RQF021	Procesamiento de pagos
Tipo:	**Prioridad:**
Necesario	Alta
Descripción:	
El sistema ha de procesar los pagos y verificar la validez del método de pago seleccionado por el usuario	
Criterios de aceptación	
• El pago debe ser válido • En caso de que el pago sea negado se le ha de informar al usuario que su pago no fue procesado	

Diagrama de base de datos



El modelo Entidad Relación anteriormente presentado está diseñado para una tienda virtual de libros, la cual se observan las relaciones entre las distintas tablas y actores. El modelo Entidad-Relación descrito corresponde a una tienda virtual de libros, donde se detallan las relaciones entre las diferentes tablas y actores involucrados en el sistema. Comenzamos con la tabla Usuario, que es una entidad fuerte y establece tres relaciones con las tablas Pedido, DirecciónEnvio y CarritoDeCompras. En cuanto a Pedido, existe una relación de uno a muchos, ya que un usuario puede realizar varios pedidos, pero cada pedido está asociado a un único usuario. La tabla DirecciónEnvio también tiene una relación de uno a muchos con Usuario, ya que un usuario puede tener múltiples direcciones de envío, pero cada dirección está vinculada a un solo usuario. Por último, CarritoDeCompras mantiene una relación de uno a muchos con Usuario, ya que un usuario puede tener varios carritos de compra, pero cada carrito corresponde a un solo usuario. La tabla Pedido se conecta con otras dos tablas: Pago y DetallePedido. Pago, al ser una entidad débil, depende de Pedido y mantiene una relación de uno a muchos, ya que un pedido puede tener varias

formas de pago, pero cada pago está asociado a un único pedido. Por su parte, DetallePedido es otra entidad débil, creada para resolver la relación muchos a muchos entre Pedido y Libro. En este caso, la relación entre DetallePedido y Pedido es de uno a muchos, ya que un pedido puede contener varios detalles de productos (libros), pero cada detalle corresponde a un único pedido. Se observa la relación de las tablas CarritoDeCompras y DetalleCarrito formando una entidad débil para romper la relación de muchos a muchos de las tablas libros y CarritoDeCompras obteniendo como resultado uno a muchos para ambas tablas. Detallando las relaciones entre las distintas tablas y actores involucrados en el sistema. La tabla principal es Libro, que representa los libros disponibles en la tienda y contiene atributos como Id, Título, ISBN, Precio, CategoriaId, AutorId y EditorialId. Se establecen diversas relaciones con otras entidades para estructurar correctamente la información: un libro es escrito por un único autor, lo que representa una relación de uno a muchos, ya que un autor puede escribir varios libros, pero cada libro tiene un solo autor; un libro es publicado por una editorial, en una relación de uno a muchos, ya que una editorial puede publicar varios libros, pero cada libro pertenece a una única editorial; y un libro pertenece a una categoría específica, en una relación de uno a muchos, dado que una categoría puede agrupar varios libros, pero cada libro pertenece a una sola categoría. Cada entidad relacionada con la tabla Libro tiene sus propios atributos: Autor contiene Id, Nombre y Apellido, permitiendo identificar a los escritores; Editorial contiene Id y Nombre, representando las casas editoriales de los libros; y CategoriaLibro incluye Id y Nombre, clasificando los libros en diferentes categorías. Este modelo facilita la gestión de una tienda virtual de libros al organizar eficientemente la información sobre autores, editoriales y categorías, garantizando integridad y coherencia en los datos.

Descripcion de las rutas de api

Get http://localhost:3300/carrito: Consulta los elementos actuales del carrito desde la base de datos.

Post http://localhost:3300/carrito: Inserta un nuevo libro al carrito en la base de datos.

Delete http://localhost:3300/carrito: Elimina uno o varios libros del carrito.

Put http://localhost:3300/carrito: Edita los elementos del carrito (por ejemplo, cambiar la cantidad de un libro).

Post http://localhost:3300/login: Recibe los datos del formulario de inicio de sesión, verifica credenciales y permite el acceso.

Get http://localhost:3300/login: Muestra la página de inicio de sesión.

Post http://localhost:3300/registrar: Recibe los datos del formulario de registro y guarda el nuevo usuario en la base de datos.

Post http://localhost:3300/registrar: Muestra la página de registro.

Asegúrate de tener Node.js instalado en tu computadora, así como un servidor de base de datos MySQL en funcionamiento.

Crea una base de datos en MySQL con el nombre "libreriabd".

Dentro de esa base de datos, crea dos tablas:

Una tabla llamada "Usuario" que debe tener los siguientes campos: ID como entero autoincremental, Nombre como texto, Apellido como texto, Email como texto único, Contrasena como texto y Telefono como texto.

Una tabla llamada "Carrito" que debe tener los siguientes campos: ID como entero autoincremental, name como texto, author como texto y price como número decimal.

Descarga o copia el archivo "conexion.js", que contiene el código del servidor backend hecho con Node.js y Express.

Abre una terminal, ubícate en la carpeta donde se encuentra el archivo "conexion.js" y ejecuta el comando "node conexion.js" para iniciar el servidor.

Una vez iniciado, el servidor quedará disponible en la dirección "http://localhost:3300".

Para registrar un nuevo usuario, se debe hacer una solicitud de tipo POST a la ruta "/registrar". El contenido de la solicitud debe incluir nombre, apellidos, correo electrónico, contraseña y teléfono. Todos los campos son obligatorios.

Para iniciar sesión, se debe hacer una solicitud de tipo POST a la ruta "/login", enviando el correo y la contraseña del usuario registrado. Si los datos coinciden, el servidor devolverá un mensaje de éxito.

Para ver los libros que hay en el carrito, se puede hacer una solicitud de tipo GET a la ruta "/carrito". Esto devolverá todos los libros almacenados en la tabla "Carrito".

Para agregar un libro al carrito, se debe hacer una solicitud de tipo POST a la ruta "/carrito", enviando el nombre del libro, el autor y el precio.

Para actualizar la información de un libro en el carrito, se debe hacer una solicitud de tipo PUT a la ruta "/carrito" seguida del ID del libro que se desea actualizar. Es obligatorio enviar el nuevo nombre, autor y precio del libro.

Para eliminar un libro del carrito, se debe hacer una solicitud de tipo DELETE a la ruta "/carrito" seguida del ID del libro a eliminar. Si el ID existe, el libro será eliminado correctamente.

El backend tiene activado CORS, por lo que puede ser consumido desde un frontend que se ejecute en la dirección "http://localhost:3000" sin problemas de conexión entre puertos.

CODIFCACION Y EJECUCION DE LA BASE DE DATOS

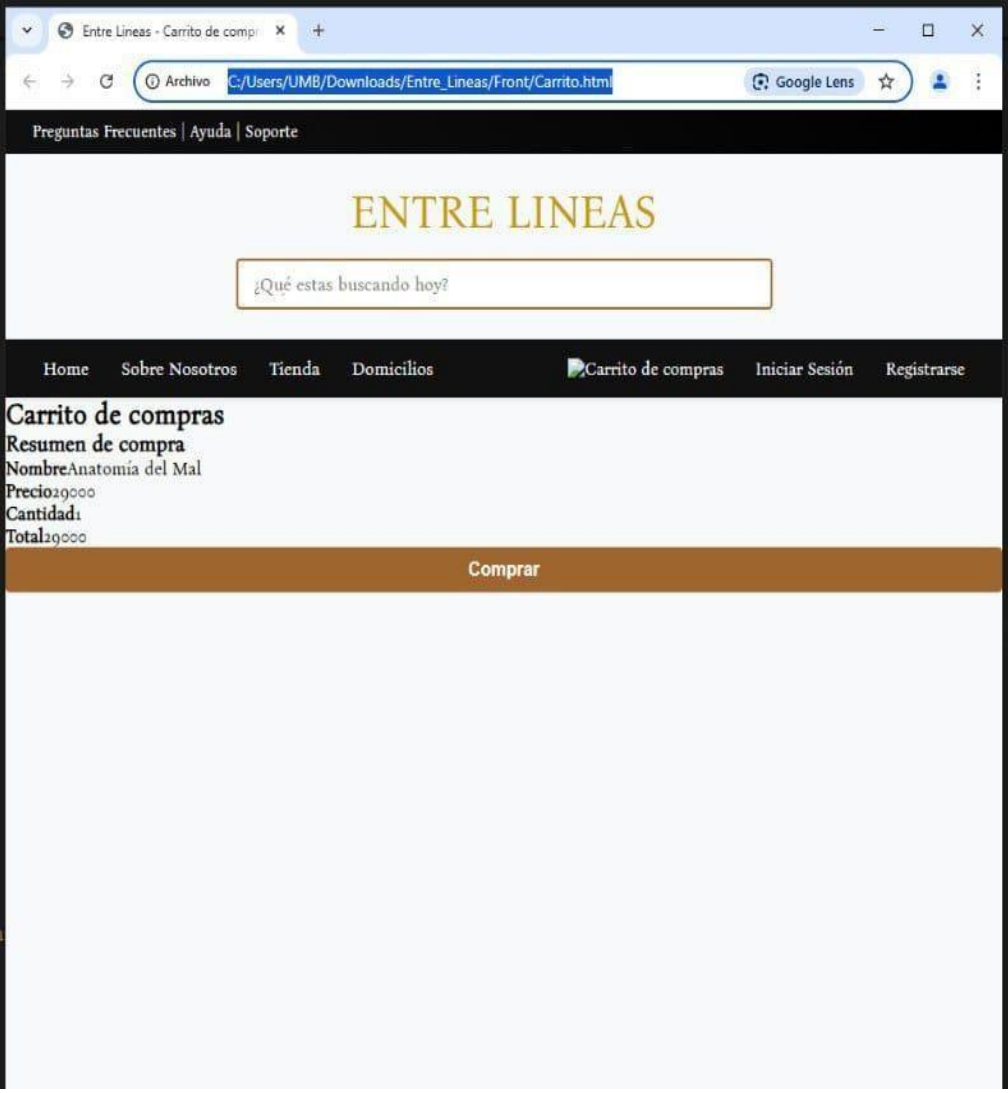
```
const resumenTotal = document.getElementById("total-price");
const resumenCantidad = document.getElementById("total-items");
const envio = document.getElementById("shipping-cost");

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