ESTRUCTURA DE UN PROYECTO WEB

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

Definir la arquitectura:

Framework:

Node.js: Permite un backend eficiente y escalable. Su modelo no bloqueante y basado en eventos optimiza el manejo de múltiples solicitudes simultáneas. En nuestra página de libros, gestionará la base de datos, autenticación de usuarios y peticiones HTTP, garantizando buen rendimiento y facilidad de integración con otras herramientas.

La principal razón para usar node.js es que este nos permite actualizar en la medida en que se deba la base de datos de la mejor manera posible y su adaptabilidad con otras herramientas nos facilita la implementación y uso de estas mismas.

Herramientas de testeo:

Postman: Postman es fundamental para probar las APIs antes de integrarlas en el frontend. Nos permite enviar solicitudes HTTP a nuestro backend y recibir las respuestas para verificar que los endpoints estén funcionando como se espera. Podemos probar rutas de libros, autenticación de usuarios, y validación de datos de manera controlada y precisa.

Cabe recalcar que también se pueden corregir los errores que muestra el Postman también con ayuda del botón inspeccionar de la web ya para errores muy específicos que no logre detectar.

RestClient: Similar a Postman, pero se integra directamente en Visual Studio Code, RestClient permite realizar pruebas de APIs sin salir de nuestro entorno de desarrollo. Esto nos permite probar rápidamente nuestros endpoints durante el proceso de desarrollo sin necesidad de cambiar de aplicación o entorno.

Como ya se comento, al no hacernos salir de nuestro entorno de desarrollo RestClient optimiza nuestro tiempo de trabajo y aporta a la agilidad de la creación del proyecto.

Base de datos:

PostgreSQL: PostgreSQL es un sistema de gestión de bases de datos relacional (RDBMS) que proporciona una gran robustez, flexibilidad y escalabilidad. Es ideal para manejar datos complejos, como los libros, autores y registros de usuarios. Ofrece potentes capacidades de consulta y permite trabajar con tipos de datos como JSON o arrays, lo que es útil para almacenar metadatos o características adicionales de los libros. También se destaca por ser una base de datos altamente compatible con transacciones y por su alta disponibilidad, ideal para aplicaciones que manejan grandes volúmenes de información.

Al ser la herramienta que mas se nos ha facilitado usar, decidimos usar postgre por ser la que mas libertades no ha brindado al momento de realizar la base de datos, además de libertades nos brinda también comodidades como una interfaz más amigable e intuitiva y un manejo mucho mas dinámico para nosotros.

Mysql: MySQL es un RDBMS de alto rendimiento, fácil de usar y ampliamente adoptado en aplicaciones web y empresariales. Es eficiente en consultas y transacciones para datos estructurados como libros y usuarios. Aunque admite JSON, es menos flexible que PostgreSQL. Se destaca por su rapidez en lectura y escalabilidad para grandes volúmenes de tráfico.

Agregamos Mysql como una segunda opción a la creación de la base de datos debido a ciertos problemas internos que hubo con el desarrollo y conexión de la base ya hecha en PostgreSQL, no esta definida como la herramienta principal para el desarrollo de la base de datos, pero la contemplamos como una herramienta de emergencia.

Lenguaje de programación:

JavaScript: JavaScript es un lenguaje que se utiliza tanto en el frontend como en el backend (mediante Node.js). Esta versatilidad simplifica la estructura del proyecto, permitiendo un desarrollo más ágil y menos propenso a errores. En el frontend, JavaScript permite interactuar con los usuarios, mostrando información sobre libros de forma dinámica y atractiva. En el backend, con Node.js, maneja la lógica del servidor, gestionando bases de datos, autenticación de usuarios y las rutas del API.

Este lenguaje ha sido el que nos han estado enseñando desde el principio de nuestra carrera entonces es bastante adecuado usar el lenguaje con el que estemos mas capacitados y el que se considera mas optimo al momento de crear una base de datos.

Implementación y autenticación:

JWT (JSON Web Token):

JWT es ideal para la autenticación sin estado, lo que significa que no necesitas almacenar información de sesión en el servidor. Cada vez que el usuario inicia sesión, se le entrega un token JWT que contiene información de autenticación y se incluye en cada solicitud posterior. Esto mejora la escalabilidad y la seguridad de nuestra aplicación web de libros, ya que no se requiere gestionar sesiones en el servidor y la validación del token se realiza de manera rápida y segura.

Como explicamos anteriormente, la generación de los token JWT al evitarnos el almacenar la información en el servidor agiliza bastante el proceso de autenticación para cada uno de los usuarios por lo que usarla era prácticamente obligatorio desde el principio, con su ayuda las solicitudes de los usuarios serán procesadas con mayor facilidad.

bcrypt: bcrypt es una librería utilizada para cifrar las contraseñas de los usuarios antes de almacenarlas en la base de datos. Este proceso convierte las contraseñas en un formato irreversible (hash), lo que hace que incluso si los datos de la base de datos se ven comprometidos, las contraseñas no puedan ser recuperadas ni utilizadas maliciosamente. Este enfoque mejora la seguridad de la página web de libros.

Este fue elegido para proporcionar la mayor seguridad a los usuarios posible para que incluso si llegaramos a sufrir un ataque de algún malware o de algún

sujeto hackeando nuestro sitio las contraseñas no podrán ser decifradas y las cuentas de nuestros usuarios se mantengan seguras.

Express Validator: Express Validator se encarga de validar y sanitizar los datos que el servidor recibe. Por ejemplo, verifica que los correos electrónicos sean válidos, que las contraseñas cumplan con los requisitos de longitud y complejidad, o que los datos enviados por los usuarios estén libres de caracteres especiales o maliciosos que podrían comprometer la seguridad de la aplicación.

Express Validator es técnicamente un requerimiento obligatorio ya que es el encargado de la verificaciones de los datos de nuestros usuarios, es la herramienta de verificación mas optima que pudimos encontrar y es una bastante completa que nos proporciona todas las facilidades que podríamos pedir de un validador de credenciales.

Dotenv: Dotenv es útil para cargar y manejar variables de entorno como claves de API, configuraciones de base de datos, o credenciales en diferentes entornos (desarrollo, producción). Utilizando un archivo .env, podemos mantener estas configuraciones fuera del código fuente y asegurar que no se expongan, mejorando así la seguridad.

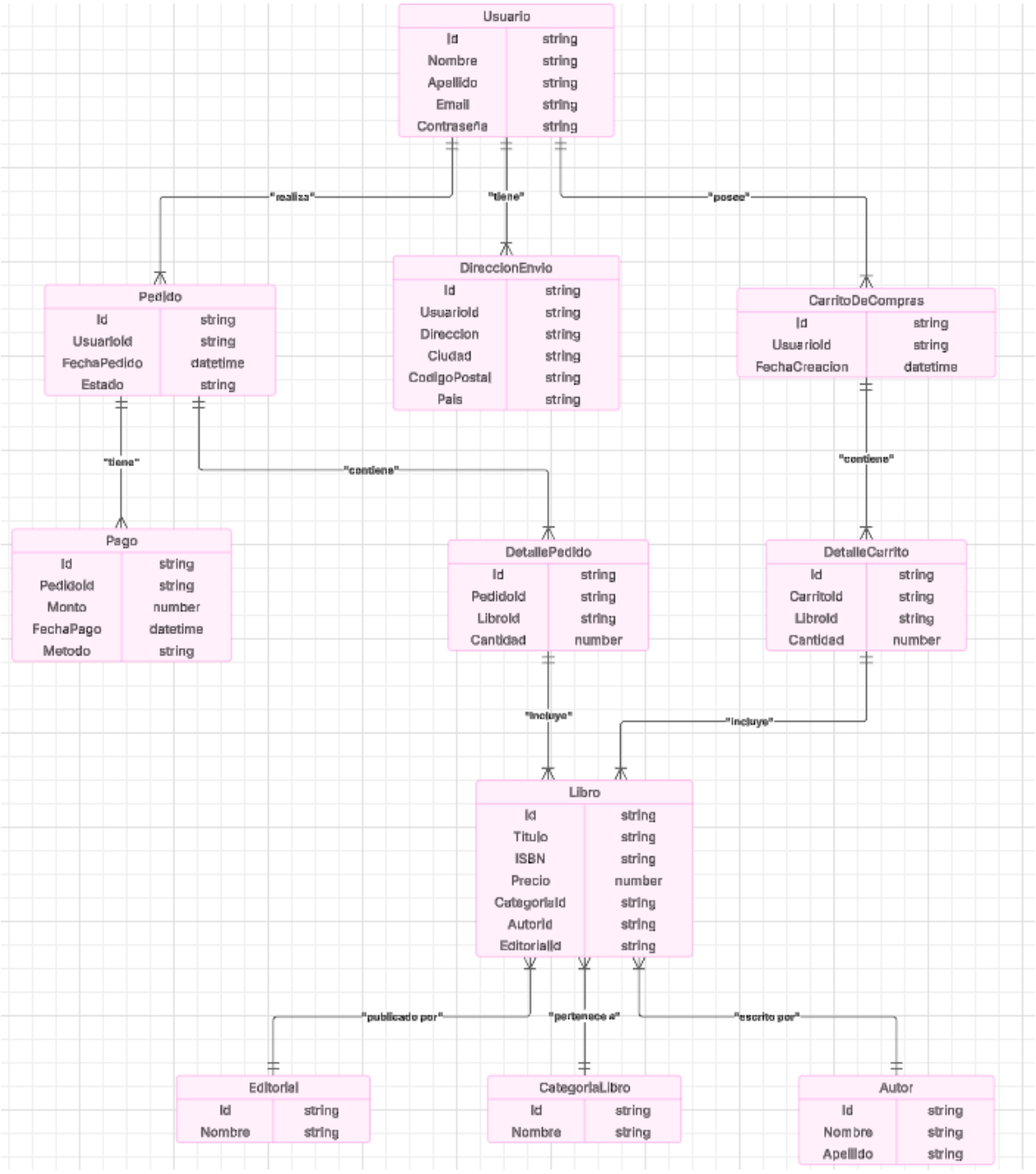
A diferencia del Express Validator, Dotenv se encarga de administrar todo tipo de variables sin importar el entorno del que vengan, y al poder ser manejas fuera del código fuente evitamos hacer algún cambio que dañe nuestro código optimizando el desarrollo de la base de datos sin perjudicarla en ningún momento.

Optimización de consultas:

Prisma: Prisma es un ORM (Object-Relational Mapping) moderno que facilita la interacción con bases de datos relacionales. A través de su cliente, puedes realizar operaciones CRUD de manera sencilla y eficiente sin escribir SQL manualmente. Prisma también nos permite manejar relaciones entre tablas (por ejemplo, entre libros y autores) de manera más intuitiva, mejorando la productividad y la legibilidad del código. Al ser un ORM altamente optimizado, Prisma ayuda a evitar consultas innecesarias o ineficientes y mejora el rendimiento en las operaciones de lectura y escritura en la base de datos.

Se eligió usar prisma porque como se comento anteriormente, al estar altamente optimizado el manejo de las tablas y relaciones dentro del proyecto se ve bastante beneficiado, ayudando a un ágil desarrollo y modificación de estas mismas. Así mismo usando prisma evitamos cualquier tipo de redundancia que pueda haber en las tablas optimizando así la creación de estas y evitando hacer alguna que no sea necesaria

Bases de Datos



El modelo Entidad Relación anteriormente presentado esta 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.