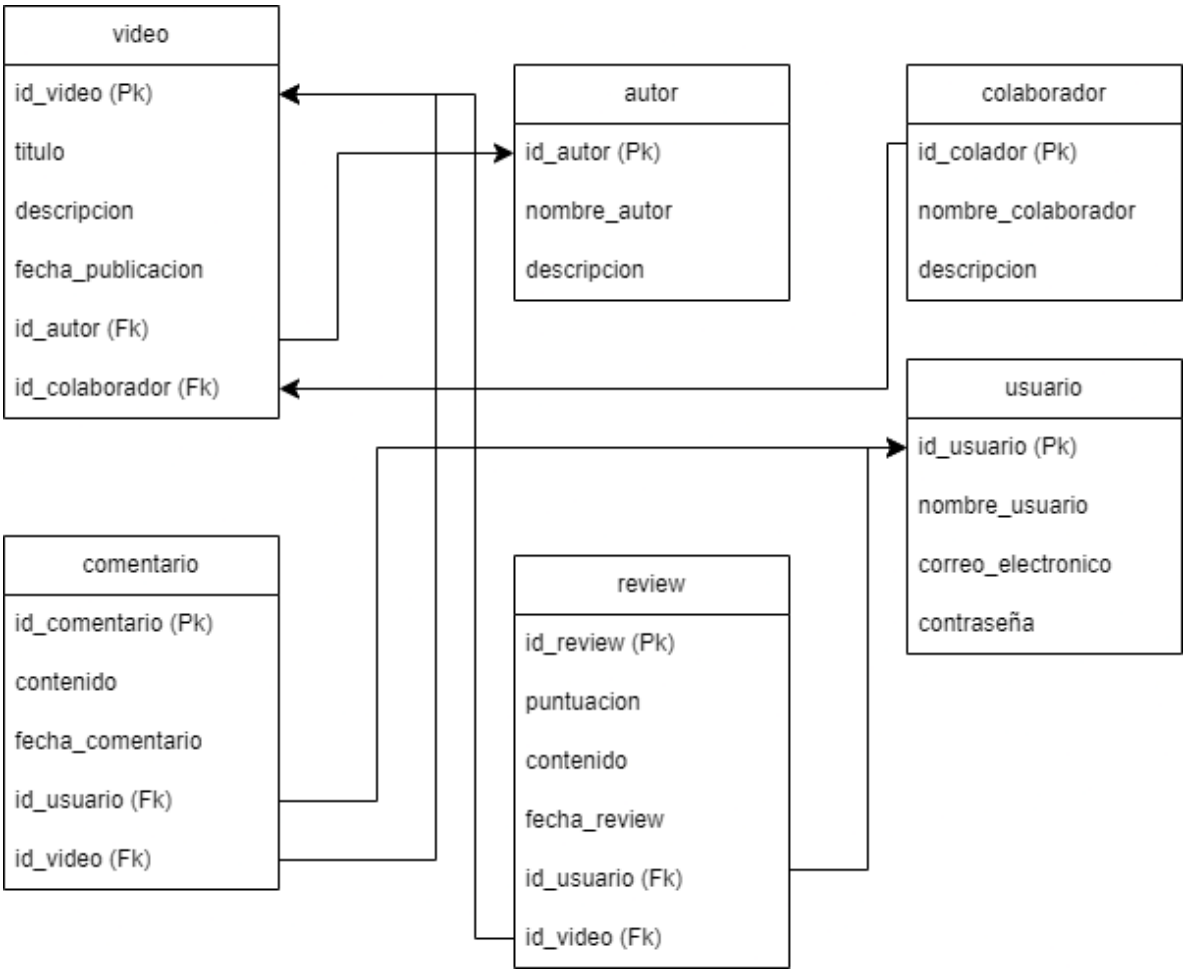
Ejercicio 4: Modelado de bases de datos

Imagina que estás construyendo un sistema de gestión de vídeos. Diseña un modelo de base de datos que incluya tablas para vídeos, autores, colaboradores, comentarios, reviews y usuarios. Asegúrate de incluir las claves primarias, las claves foráneas y las restricciones de integridad necesarias para que el sistema funcione correctamente.



Ejercicio 5:

Arquitectura del backend Describe cómo estructurarías el backend de una aplicación de comercio electrónico. Habla sobre las tecnologías que utilizarías, la organización de los archivos, el uso de patrones de diseño, etc.

1. Arquitectura: Una opción común para el backend de una aplicación de comercio electrónico es adoptar una arquitectura de tipo cliente-servidor. En este enfoque, el servidor actúa como el punto central que procesa las solicitudes y gestiona la lógica del negocio, mientras que el cliente (por ejemplo, una aplicación web o móvil) se comunica con el servidor para realizar consultas y recibir respuestas.

2.
3. Tecnologías: Las tecnologías utilizadas pueden variar según las preferencias y requisitos específicos, pero algunos elementos clave pueden incluir:

    a. Lenguaje de programación: Por ejemplo, Java, Python, Node.js, Ruby, entre otros, son opciones populares para el desarrollo del backend.
    b. Frameworks: Utilizar un framework puede agilizar el desarrollo y proporcionar funcionalidades comunes. Algunos ejemplos populares son Spring (Java), Django (Python), Express (Node.js), Ruby on Rails (Ruby), etc.
    c. Base de datos: Para almacenar información de productos, usuarios, pedidos, etc., se pueden utilizar bases de datos relacionales como MySQL, PostgreSQL o bases de datos NoSQL como MongoDB o Cassandra, dependiendo de los requisitos y la escalabilidad necesaria.
    d. APIs: Es probable que se integren servicios externos, como pasarelas de pago o servicios de envío, a través de APIs. El uso de estándares como RESTful puede facilitar la comunicación y el intercambio de datos entre el backend y estos servicios externos.
    e. Seguridad: La seguridad es crucial en una aplicación de comercio electrónico. Se pueden utilizar tecnologías como JSON Web Tokens (JWT) para la autenticación y autorización, así como medidas para proteger contra ataques como inyección SQL o ataques de denegación de servicio (DDoS).

4. Organización de archivos: Una buena práctica es estructurar los archivos en función de la modularidad y la separación de responsabilidades. Puedes organizar el código en módulos o componentes, donde cada uno se ocupe de una funcionalidad específica, como autenticación, gestión de productos, pedidos, etc. Dentro de cada módulo, se pueden tener archivos para manejar rutas (endpoints), controladores para procesar las solicitudes, modelos para interactuar con la base de datos y servicios para encapsular la lógica del negocio.

5. Patrones de diseño: Los patrones de diseño son técnicas probadas para resolver problemas comunes en el desarrollo de software. Algunos patrones que podrían ser útiles en una aplicación de comercio electrónico incluyen:
    f. Patrón MVC (Modelo-Vista-Controlador): Ayuda a separar la lógica de negocio, la presentación y la interacción con la base de datos.
    g. Patrón Repositorio: Proporciona una capa de abstracción para acceder y manipular los datos almacenados en la base de datos.
    h. Patrón Singleton: Puede ser utilizado para gestionar la conexión a la base de datos u otros componentes que deben ser compartidos globalmente.
    i. Patrón Factory: Puede ser útil para la creación de objetos

En resumen, para estructurar el backend de una aplicación de comercio electrónico, se pueden seguir los siguientes puntos:

1. Adoptar una arquitectura cliente-servidor.

2. Utilizar tecnologías como lenguajes de programación, frameworks, bases de datos y APIs.
3. Organizar los archivos de manera modular, separando las responsabilidades de cada componente.
4. Aplicar patrones de diseño como MVC, Repositorio, Singleton y Factory para resolver problemas comunes.
5. Priorizar la seguridad implementando medidas como autenticación, autorización y protección contra ataques.
6. Considerar la escalabilidad y el rendimiento al diseñar la arquitectura y seleccionar las tecnologías.
7. Documentar el código y mantener buenas prácticas de desarrollo para facilitar la colaboración y el mantenimiento a largo plazo.

Estos son solo aspectos generales, y la estructura específica del backend puede variar según los requisitos y las preferencias del proyecto.

Ejercicio 6:

Nomenclatura Crea un documento de políticas de nomenclatura para el equipo de desarrollo de una compañía, la política debe incluir nomenclatura de:bases de datos, variables, funciones, clases, git, etc.

Política de Nomenclatura - Equipo de Desarrollo

1. Bases de datos:
    a. Nombres de bases de datos: Utilizar nombres descriptivos y significativos que reflejen el propósito y la función de la base de datos.
    b. Nombres de tablas: Utilizar nombres en singular, evitando plurales o nombres ambiguos. Utilizar nombres descriptivos y concisos que reflejen el contenido de la tabla.
    c. Nombres de columnas: Utilizar nombres descriptivos que indiquen claramente el tipo de datos y la información que almacena la columna.
2. Variables:
    d. Nombres de variables: Utilizar nombres descriptivos que reflejen el propósito y la función de la variable. Utilizar camelCase para nombres de variables compuestas por múltiples palabras, comenzando con minúscula.
    e. Evitar abreviaturas confusas o demasiado cortas. Priorizar la claridad y la legibilidad del código.
3. Funciones:
    f. Nombres de funciones: Utilizar nombres descriptivos que indiquen claramente la acción o el propósito de la función. Utilizar camelCase para nombres de funciones compuestas por múltiples palabras, comenzando con minúscula.
4. Clases:
    g. Nombres de clases: Utilizar nombres descriptivos y sustantivos que reflejen claramente la responsabilidad y el propósito de la clase. Utilizar PascalCase para nombres de clases, comenzando con mayúscula.
5. Git:

    h. Nombres de ramas (branches): Utilizar nombres descriptivos y significativos para las ramas. Utilizar kebab-case para nombres de ramas compuestas por múltiples palabras, todo en minúscula.
    i. Commits: Escribir mensajes de commit claros y concisos que describan los cambios realizados. Utilizar un formato consistente y utilizar verbos en imperativo para indicar la acción realizada.
6. General:
    j. Evitar nombres genéricos o ambiguos que no transmitan claramente su función o propósito.
    k. Utilizar nombres en inglés para garantizar la consistencia y facilitar la colaboración con otros equipos o desarrolladores.
    l. Mantener una convención de nomenclatura coherente en todo el código y la documentación del proyecto.
    m. Comentar el código de manera adecuada para explicar la funcionalidad y la intención del mismo.

Esta política de nomenclatura tiene como objetivo mejorar la legibilidad, la mantenibilidad y la colaboración dentro del equipo de desarrollo. Al seguir estas pautas, esperamos lograr un código más claro, comprensible y consistente en todos los proyectos de la compañía.