**Objetivo de la prueba: Evaluar tus habilidades en Backend**

**Ejercicio 1: Consumo de APIs**

Crea una aplicación Node.js que consuma la API de GitHub y muestre los 10 repositorios

más populares del usuario "google".

**Ejercicio 2: Nomenclatura**

Te presentamos el siguiente fragmento de código:

**function operation**(number1, number2, number3) {

**let** sumNumbers = number1 + number2;

**let** productNumbers = sumNumbers \* number3;

**let** result = Math.sin(productNumbers);

**return** result;

}

Reemplaza los nombres de las variables con nombres más descriptivos que reflejen mejor

su función.

**Ejercicio 3: Pensamiento lógico**

Escribe una función que tome un número entero como entrada y devuelva un array con

todos los números enteros impares desde 1 hasta el número de entrada. Por ejemplo, si el

número de entrada es 9, la función debería devolver [1, 3, 5, 7, 9].

**function** **arrayOddNumber**(numberEnd){

**let** arrayOdd = [];

**for** (**let** index = 1; index <= numberEnd; index++) {

**if**(index % 2 !== 0){

arrayOdd.push(index);

}

}

**return** arrayOdd;

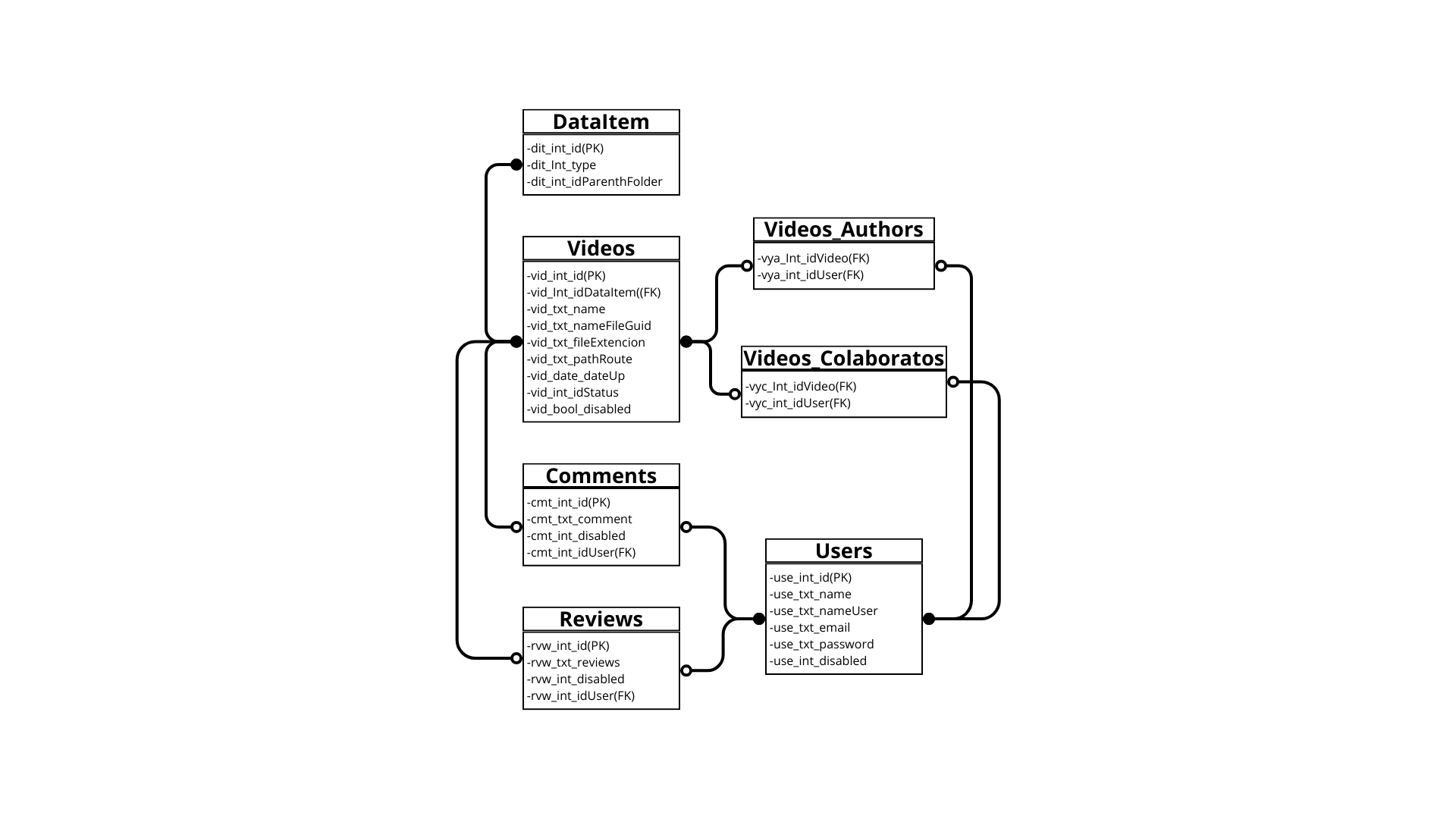
}

**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. Asegúrate de incluir las claves primarias, las claves foráneas y las restricciones

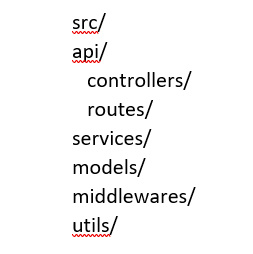
de integridad necesarias para que el sistema funcione correctamente.

**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.



Haría una estructura de arquitectura de modelos vista controlador (MVC)

con la siguiente distribución:

Usuaria en el backend NodeJs y React, donde NodeJs sería adecuado para el manejo de las peticiones concurrentes y el manejo de los eventos realizados en tiempo real, como lo son los mensajes instantáneos o la solicitud de actualización de un listado de productos.

Para la base de datos usaría motores como MySql, SQL o PostgreSQL para darle manejo a los productos, pedidos, etc. Además, usaría una base de datos no estructurada como MongoDb para darle el manejo a los datos menos estructurados como sesiones de carrito de compras, carritos y perfiles de usuarios por la cantidad de datos, esto me ofrece una alta velocidad de escritura y lectura, lo cual es muy importante para este tipo de aplicaciones.

En el tema de autorizaciones y autenticaciones de usuarios, sería conveniente usar JSON Web Tokens (JWT) para autenticar y, autorizar de manera segura y eficiente los usuarios, esto aplicando una doble encriptación de las credenciales tanto en el usuario como en su password y renovando el JWT cada hora como máximo tiempo de inactividad, almacenando los productos del carrito en la base de datos de Mongo y en su local store.

En cuanto a los servidores y almacenamiento usaría un servidor robusto y escalable para plataformas en la nube como AWS, Azure o Google Cloud que son opciones populares.

También, haría uso de herramientas como Git, Docker y Kubernetes para implementar la integración continua y el despliegue. Estas herramientas garantizan que el código se integre de manera eficiente y sin errores permitiendo el despliegue de la aplicación en un entorno controlado y escalable. Con respecto al patrón de diseño, implementaría las siguientes metodologías: Patrón Repositorio, Patrón Singleton e Inyección de Dependencias, aplicando soluciones probadas y recomendadas para problemas específicos que se han encontrado en el desarrollo de software.

**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.

1. Nombre de bases de datos

Las bases de datos deben ser en snake\_Case, con un nombre que describa claramente su propósito y se debe evitar el uso de abreviaturas no estándar; preferir la claridad sobre la brevedad.

Ejemplos: clientes, factura\_Detalles, empleado\_Datos.

Los nombres de tablas y campos deben ser en minúsculas y seguir el formato de snake\_case.

Ejemplos: nombre\_cliente, factura\_numero, id\_factura, primer\_nombre.

1. Variables, constantes y funciones

Utilizar camelCase para nombres de variables y ser descriptivo en la elección de nombres para mejorar la legibilidad del código.

Las funciones deben ser nombradas en camelCase, con un nombre que describa claramente su propósito y evitar funciones excesivamente largas para favorecer la modularidad.

Utilizar camelCase para nombres de métodos y ser coherente con la nomenclatura de funciones para mantener la uniformidad en el código.

Las constantes deben estar en mayúsculas con palabras separadas por guiones bajos (UPPER\_CASE).

Priorizar el uso de constantes en lugar de valores literales para mejorar la mantenibilidad.

1. Clases y Objetos

Utilizar PascalCase para nombres de clases siendo descriptivo y utilizar sustantivos que representen claramente la entidad que la clase representa.

Utilizar camelCase para nombres de objetos siendo consistente en la nomenclatura de objetos para facilitar la comprensión del código.

1. Git

Las ramas deben ser nombradas de manera descriptiva en minúsculas y con guiones (-), para separar las palabras, se debe prefijar nombres de ramas según el tipo de trabajo (feature/, bugfix/, hotfix/, etc.).

Escribir mensajes de confirmación claros y concisos que expliquen los cambios realizados utilizando imperativos en los mensajes de confirmación como "Añadir", "Modificar", "Corregir", etc.

Al utilizar etiquetas de versiones hacerlo con un formato semántico (ej. v1.0.0) para facilitar la gestión de versiones.

1. Genral

Los nombres de archivos deben ser en minúsculas y con guiones bajos, para separar las palabras.

Las comas deben ser seguidos de un espacio.

Los bloques de código y llaves deben ser colocados en una nueva línea.