**ARQUITECTURA DEL BACKEND PARA PARA UNA APLICACIÓN DE COMERCIO ELECTRÓNICO**

El backend de comercio electrónico estará diseñado para una alta escalabilidad, ya que siempre esta a cargo de generar, mantener y entregar la información a los usuarios, donde habrá miles o cientos de solicitudes para llevar a cabo las operaciones realizadas.

El backend de comercio electrónico consta de 2 partes: la base de datos (donde se almacena la información) y un servidor (que conecta los datos con el navegador), por lo tanto, la arquitectura del backend de acuerdo a una investigación exhaustiva constara de lo siguiente:

Se emplearía una base de datos NoSQL, específicamente **MongoDB** , que destaca por ofrecer un eficiente almacenamiento de datos y brindando la opción de alojamiento en la nube a través de MongoDB Atlas. . En paralelo, se gestionaría la lógica del backend mediante **Node.js**, asegurando así un funcionamiento continuo del servidor en sincronía con la base de datos.

Adicionalmente, se emplearía **NestJS** como el marco web para el backend. Este framework escalable y orientado a empresas que simplifica la creación de aplicaciones web, proporcionando una interfaz más accesible. Facilita la adopción de la aplicación con una arquitectura de microservicios, donde diversas funcionalidades se dividen en conjuntos de servicios más pequeños integrados a través de **APIs**. Este enfoque no solo favorece la potenciación de servicios como pagos y productos, sino que también posibilita actualizaciones y escalabilidad independientes. Cabe destacar que las solicitudes provenientes del cliente (front end), representadas como requests, así como las respuestas, denominadas responses, se gestionan a través del protocolo HTTP.

En cuanto a la seguridad, se emplearía **JSON Web Tokens (JWT)** para la autenticación y autorización. Al recibir una solicitud, se verificaría la validez del token JWT adjunto para autenticar al usuario y autorizar el acceso a recursos específicos a través de **middleware**, permitiendo la autorización basada en roles. Además, se destacaría que se emplearía cifrado bidireccional o unidireccional, según la necesidad, utilizando bibliotecas especializadas como **crypto o bcrypt** para garantizar la confidencialidad e integridad de la información sensible. Para agregar un servicio de **procesamiento de pagos**, se podría integrar una solución de pago reconocida, utilizando sus APIs para gestionar transacciones de manera segura y eficiente.

En consonancia con lo mencionado previamente, se implementaría NestJS, aprovechando su entorno de pruebas integrado basado en **Jest**. Al crear servicios, interceptores o controladores, la CLI generaría automáticamente archivos de especificaciones para pruebas unitarias. Este enfoque automatizado eliminaría la necesidad de que los desarrolladores escriban código adicional para las pruebas unitarias, optimizando el proceso de desarrollo y asegurando la calidad del código. Asimismo, se incorporarán excepciones, como bloques try-catch, para gestionar errores de manera eficiente según las circunstancias.

Adheridos al principio SOLID en el desarrollo de software, se adoptaría una organización estructurada de archivos por **capas o módulos específicos**, simplificando la búsqueda del código necesario. En este contexto, por ejemplo, al contar con un módulo de autenticación, se crearía una carpeta denominada "auth" que albergaría todos los archivos pertinentes, tales como controladores, servicios, guardias, entre otros. Esta práctica garantiza que, cuando alguien deba realizar modificaciones en el módulo de autenticación, pueda ubicar con precisión los elementos correspondientes.

Al mismo tiempo, se seguiría el **patrón de diseño de inyección de dependencias**, en línea con el framework NestJS. Este enfoque permite una gestión más efectiva de las dependencias en la aplicación, mejorando la flexibilidad y facilitando el mantenimiento del código. Integrar este patrón en la arquitectura general refuerza la coherencia y robustez del sistema, alineándose con las mejores prácticas de desarrollo de software y proporcionando una estructura clara y mantenible.

Por último, se contemplaría la posibilidad de configurar el proyecto con **PM2**, para garantizar que un script determinado se ejecute continuamente. En paralelo, se adoptaría un enfoque de informática en la nube denominado Infraestructura como Servicio (**IaaS**). En este modelo, un proveedor externo ofrece servicios de infraestructura, como almacenamiento y virtualización, cuando son necesarios, a través de la nube o Internet. En este contexto, se podría considerar la utilización del servicio Amazon Elastic Compute Cloud (Amazon EC2) de Amazon Web Services, aprovechando así la flexibilidad y escalabilidad que ofrece esta plataforma en la nube.