

Documentación Técnica – API Comercio Electrónico

Versión 1-0

Elaborado por: Andrés Pérez

El presente documento es para establecer las especificaciones técnicas de la API de comercio electrónico elaborada, para ello se dividirá el documento en varias secciones para explicar cómo utilizar de manera efectiva la API.

1) **Tecnologías usadas en el proyecto**

Para el desarrollo del proyecto se usaron distintas tecnologías para poder crear una API sólida que tenga robustez y escalabilidad, entre las tecnologías usadas tenemos:

- a) **TypeScript**: Para el desarrollo de esta API, se decidió utilizar TypeScript debido a las múltiples ventajas que ofrece sobre JavaScript puro, especialmente en términos de tipado estático. Además, TypeScript nos permite definir clases, interfaces y tipos personalizados, lo que resulta fundamental para estructurar el código de manera más clara y escalable. Si bien Node.js permite trabajar con JavaScript de manera flexible, la ausencia de tipado estático puede hacer que ciertas validaciones sean más complejas y propensas a errores. Otro punto a favor de TypeScript es su total compatibilidad con JavaScript, lo que nos da la libertad de aprovechar el ecosistema y las bibliotecas de Node.js sin limitaciones.

- b) **PostgreSQL**: Para el desarrollo de este proyecto, se optó por PostgreSQL como base de datos principal debido a su robustez, rendimiento y su capacidad de manejar altos volúmenes de datos. Además, al ser un sistema de gestión de bases de datos relacional, permite estructurar la información de manera eficiente mediante tablas, relaciones y consultas SQL avanzadas. Para el funcionamiento correcto del proyecto se debe crear una base de datos denominada: Ecommerce_API, Prisma se encargará del resto.

- c) **Prisma**: se utilizó en el desarrollo del proyecto porque facilita la interacción con la base de datos de una manera más sencilla, segura y eficiente en comparación con otras herramientas o con el uso directo de SQL. Uno de sus principales beneficios es que proporciona un ORM (Object-Relational Mapping) que permite trabajar con la base de datos utilizando código en TypeScript, esto hace que las consultas sean más intuitivas y fáciles de escribir en comparación con las consultas SQL tradicionales.
- d) **Express.js**: Se optó por Express.js como framework de desarrollo, porque permite desarrollar una API más segura, escalable y fácil de mantener, sin perder la eficiencia y simplicidad que caracteriza a Express.
- e) **JWT**: es un estándar abierto (RFC 7519) que define una forma compacta y autónoma de transmitir información de manera segura entre dos partes, como un cliente y un servidor. el uso de JWT como Token es importante porque permite una autenticación segura y eficiente sin necesidad de almacenar datos del usuario en el servidor. El Token es generado en el momento del inicio de sesión y se envía al cliente.
- f) **Uuid**: Se usó la librería uuid para la generación de los ID. Esta librería permite crear identificadores únicos universales (UUID), lo cual garantiza que cada registro en la base de datos tenga un identificador único y no repetible.

Además de las tecnologías utilizadas se decidió utilizar Arquitectura Hexagonal para el desarrollo del proyecto, la Arquitectura Hexagonal, es un patrón de diseño de software que busca separar las responsabilidades del sistema y mejorar su modularidad, flexibilidad y mantenibilidad con la finalidad crear una estructura que haga que el sistema sea fácil de modificar, probar y escalar.

El proyecto se ha organizado en tres módulos principales para mantener una estructura clara y modular que facilite tanto el desarrollo como el mantenimiento del código:

- Usuario
- Producto
- Pedido

Cada uno de estos módulos sigue los principios de la Arquitectura Hexagonal y está dividido en tres capas fundamentales: Dominio, Aplicación e Infraestructura. Esta organización permite mantener una separación clara entre las distintas responsabilidades del sistema, lo que a su vez facilita la escalabilidad y la adaptabilidad a cambios futuros.

- a) **Capa de Dominio:** Contiene toda la lógica del negocio, que es la parte más importante e independiente de cualquier tecnología o detalle de implementación. En el contexto del proyecto, la capa de Dominio es responsable de gestionar las reglas de negocio y las entidades principales como Usuario, Producto y Pedido.
- b) **Capa de Aplicación:** Esta capa actúa como intermediaria entre la capa de Dominio y las interfaces externas. Aquí se gestionan los casos de uso y los servicios que orquestan las operaciones de negocio, pero sin entrar en los detalles de cómo se ejecutan esas operaciones.
- c) **Capa de Infraestructura:** esta capa es responsable de manejar todos los detalles de implementación externa que interactúan con el sistema, como bases de datos, servicios externos, APIs o cualquier otra tecnología que se utilice para hacer que el sistema funcione en un entorno real. Esta capa incluye implementaciones como el acceso a la base de datos, la comunicación con servicios externos y otros componentes técnicos.

2) Estructuración de Peticiones a la API

Primero que todo es importante destacar que, en el proyecto, se utilizan diferentes códigos HTTP para manejar las respuestas de manera adecuada.

- El código 200, se emplea cuando una solicitud se procesa correctamente y se completa con éxito.
- El código 201, se usa para indicar que un recurso ha sido creado correctamente, como cuando un nuevo usuario o pedido se genera en la base de datos.
- El código 400, que significa: "Bad Request", se devuelve cuando la solicitud contiene errores del cliente, como datos faltantes o mal formateados.
- El código 500, se usa cuando ocurre un error interno en el servidor al procesar la solicitud.

Estos códigos ayudan a comunicar de manera clara el estado de las operaciones, permitiendo una correcta gestión de errores y el flujo de información entre el cliente y el servidor.

Como se mencionó previamente el proyecto está compuesto por tres grandes módulos: Usuario, Producto y Pedido. Cada uno de estos módulos tiene su ruta principal una vez que se levanta el servidor en la ruta establecida, ahora se visualizarán cada uno de los endpoints de cada módulo especificando que valores de deben agregar para que funcione adecuadamente:

1) Usuario

Este módulo cuenta con la ruta principal:

`http://localhost:puerto/usuarios`

Esta es la ruta predeterminada de usuarios cuando se levanta la aplicación, utilizando el comando: **`npm run start`**; el puerto se puede configurar en el "archivo. Venv" que se encuentra en el proyecto.

Posterior a esto, el módulo usuario contiene 4 rutas más para realizar peticiones y cumplir con los requerimientos del proyecto. Entre las cuales tenemos:

a) Ruta para crear Usuario:

http://localhost:puerto/usuarios/registro

método: Post.

Para crear un usuario es necesario incluir en el apartado Body de la solicitud un Json con la siguiente estructura:

```
{ "nombre" : "nombre",  
  "apellido" : "apellido"  
  "correo" : "correo@correo.com"  
  "clave" : "clave" }
```

Cada uno de los campos colocados deben ser agregados en el Body de la petición al hacer la solicitud, sustituyendo los valores de la derecha. Si el usuario se creó exitosamente, la solicitud retornara el código 201. En caso contrario de algún error, se retornará el código 400 o 500 con el mensaje de error correspondiente.

b) Ruta para iniciar sesión de un Usuario:

http://localhost:puerto/usuarios/login

método: Post.

Para iniciar sesión de un usuario es necesario incluir en el apartado Body de la solicitud un Json con la siguiente estructura:

```
{ "correo" : "correo@correo.com",  
  "clave" : "clave" }
```

Con esta información la API se encargará de validar con la base de datos que el usuario con los datos proporcionados exista. Y retornará el código de error correspondiente en caso de éxito o error. Además, es importante destacar que como mecanismo de autenticación para el proyecto se usó JWT; por lo que al ser un caso de éxito la API retorna el Token JWT del usuario. Este Token es muy importante para su posterior uso en otras funciones del proyecto. Y Tiene una duración de 1 hora, esto se puede modificar en el proyecto.

C) Ruta para buscar todos los usuarios:

http://localhost:puerto/usuarios

método: Get

Este endpoint de la API no tiene valores de entrada, para usarlo solo debemos utilizar el método GET a la ruta especificada y devolverá una lista con los usuarios registrados en la base de datos.

D) Ruta para eliminar un usuario por su id:

http://localhost:puerto/usuarios/:id

método: Delete

Este endpoint de la API tiene como valor de entrada el id del usuario, a diferencia de los otros endpoints este valor se pasará por la ruta en donde se encuentra el: **:id**. Es importante destacar que, para poder eliminar un usuario, se debe realizar una autenticación con el Token del JWT. Es decir que se debe utilizar el Token generado al utilizar el endpoint Login y ese Token debe ser colocado en la sección:

Authorization -> Auth Type: Bearer Token -> Se coloca el Token

Params **Authorization** ● Headers (8) Body ● Scripts Tests Settings

Auth Type

Bearer Token ▼

Token

Token

The authorization header will be automatically generated when you send the request. [Learn more about](#)

Una vez colocado el Token la API lo validará y permitirá eliminar el usuario o generar el error correspondiente según sea el caso.

Estos son los endpoints disponibles para el módulo de Usuario en esta primera versión del proyecto.

2) Producto

Este módulo cuenta con la ruta principal:

`http://localhost:puerto/productos`

Este módulo cuenta con 5 rutas, en las cuales se pueden realizar peticiones a la API, entre las cuales tenemos:

a) Ruta para crear un producto:

`http://localhost:puerto/productos/crear`

método: Post.

Para crear un producto es necesario incluir en el apartado Body de la solicitud un Json con la siguiente estructura:

```
{ "nombre" : "nombre",  
  "descripcion" : "descripcion"  
  "precio": 20  
  "disponibilidad" : 10 }
```

En este caso se necesitan agregar 4 campos con sus respectivos valores para la creación del producto. Los campos nombre y descripción son de tipo string, por lo que deben estar entre llaves los valores. Mientras que precio es de tipo número decimal y disponibilidad de número entero. Si el producto se creó exitosamente la petición retornará el código 201 con el producto, en caso contrario retornará un código 400 o 500 con el respectivo mensaje de error.

b) Ruta para editar un producto:

http://localhost:puerto/productos/editar/:id

método: Put.

Para editar un producto será necesario primero pasar por la ruta el id del producto a editar, esto es importante ya que le permitirá a la API buscar en base de datos cual es el producto a editar. De igual forma para editarlo se deberá agregar en el apartado Body de la petición un Json con la siguiente estructura:

```
{ "nombre" : "nombre",  
  "descripcion" : "descripcion"  
  "precio": 20  
  "disponibilidad" : 10 }
```

Se deben pasar estos campos en el Body para que se pueda modificar el producto. La API tomará el ID del producto y reemplazará sus valores con los colocados en los campos correspondientes. En caso de editarse correctamente retornará un código 200; en caso contrario retornará un código 400 o 500 con su respectivo mensaje de error.

c) Ruta para eliminar un producto por su id:

http://localhost:puerto/productos/eliminar/:id

método: Delete

Este endpoint de la API tiene como valor de entrada el id del producto, a diferencia de los otros endpoints este valor se pasará por la ruta en donde se encuentra el **:id**. Y a partir de este valor la API lo validará y permitirá eliminar el usuario o generar el error correspondiente según sea el caso.

d) Ruta para buscar todos los productos:

http://localhost:puerto/productos

método: Get

Este endpoint de la API no tiene valores de entrada, para usarlo solo debemos utilizar el método GET a la ruta especificada y devolverá una lista con los productos registrados en la base de datos.

e) Ruta para buscar un producto por su id:

http://localhost:puerto/productos/:id

Este endpoint de la API tiene como valor de entrada el ID de producto y debe ser colocado al final de la ruta en donde se visualiza **:id**, con este valor la API se encargará de validar en base de datos si el producto existe y si existe lo retornará con sus detalles, en caso contrario realizará validaciones y retornará el error correspondiente.

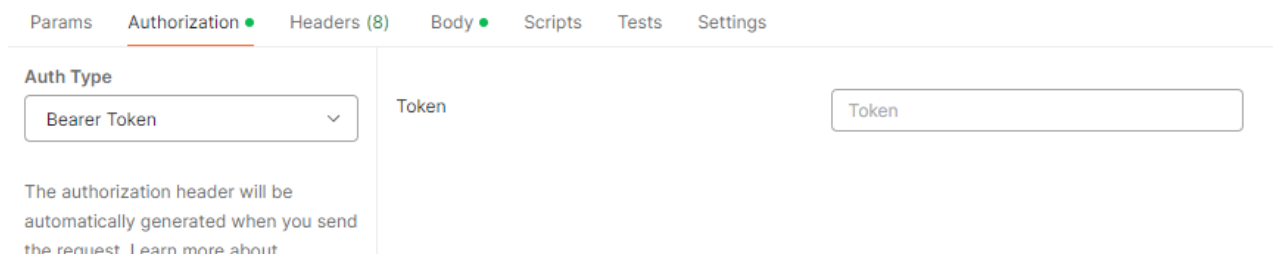
3) Pedido

Este módulo cuenta con la ruta principal:

`http://localhost:puerto/pedidos`

Este módulo cuenta con 3 rutas de endpoints disponibles para hacer peticiones. Es importante destacar que todas las rutas de este módulo requieren autenticación por Token de JWT, es decir que se debe hacer Login para poder acceder a cualquiera de las rutas. Para poder hacer Login y obtener el Token que permita acceder a las rutas del módulo Pedido; primero se debe hacer Inicio de Sesión con la ruta de inicio de sesión de Usuario, realizar la autenticación y posteriormente obtener el Token generado. Con el Token generado se debe colocar en la sección:

Authorization -> Auth Type: Bearer Token -> Se coloca el Token



The screenshot shows the 'Authorization' tab in a REST client interface. The 'Auth Type' dropdown is set to 'Bearer Token'. To the right, there is a 'Token' label and an input field. Below the dropdown, a note states: 'The authorization header will be automatically generated when you send the request. Learn more about'.

De cualquiera de las peticiones que se realicen a las rutas del módulo para poder realizar las acciones. Entre las rutas disponibles del módulo tenemos:

a) Ruta para crear un pedido:

`http://localhost:puerto/pedidos/`

método: Post.

Para crear un pedido es necesario incluir en el apartado Body de la solicitud un Json con la siguiente estructura:

```
{
  "fecha": "2025 - 01 - 31 12:12:00",
  "estado": "PROCESADO",
  "detalleProductos": [
    {
      "idProducto": "id",
      "cantidad": 2,
      "precio": 50.25
    },
    {
      "idProducto": "id",
      "cantidad": 3,
      "precio": 50.25
    }
  ]
}
```

Esto es una muestra de cómo debe ser colocado el Json del Body al hacer una petición a la ruta para crear un pedido, hay varios aspectos a tomar en cuenta:

- El ID de Usuario asociado al pedido se tomará de JWT generado, es decir el ID de Usuario equivalente para el pedido enviado en la petición será con el que se inició sesión en el endpoint de Login de Usuario.
- El campo fecha es de tipo string por lo que debe estar entre comillas. Debe estar en formato: AAAA-mm-DD hh-MM-ss; porque si no arrojará un error de formato.
- El campo estado es de tipo string por lo que debe estar entre comillas el valor. A nivel de la capa de dominio el campo Estado es de tipo Enumerado, y tiene los valores: 'PAGADO', 'ENVIADO', 'ENTREGADO', 'CANCELADO', 'PROCESADO'. Por lo que cualquier valor que no esté entre los retornará un mensaje de error.
- Luego viene el campo detalleProductos, este campo debe ser manejado como un arreglo de objetos, cada objeto del arreglo debe tener la estructura:

```
{  
  "idProducto": "id",  
  "cantidad": 3,  
  "precio": 50.25  
}
```

También es importante destacar que el ID producto debe ser un id válido de un producto guardado en la base de datos, ya que la API valida esto para poder agregarlo.

b) Ruta para buscar todos los pedidos:

http://localhost:puerto/pedidos/

método: Get

Este endpoint de la API no tiene valores de entrada, para usarlo solo debemos utilizar el método GET a la ruta especificada y devolverá una lista con todos los pedidos registrados en la base de datos. Como se mencionó previamente solo se podrá acceder a este endpoint una vez autenticado con el JWT.

c) Ruta para buscar pedidos por id de usuario:

http://localhost:puerto/pedidos/usuario/:idUsuario

Este endpoint de la API tiene como valor de entrada el ID de un Usuario y debe ser colocado al final de la ruta en donde se visualiza **:idUsuario**, con este valor la API se encargará de validar en base de datos si existen pedidos asociados al usuario y si existen los retornará con sus detalles, en caso contrario realizará validaciones y retornará el error correspondiente. Como se mencionó previamente solo se podrá acceder a este endpoint una vez autenticado con el JWT.