**Proyecto VentasAPI**

Jamilton Jhony Rueda Legarda

Jonier Esneider Rendon Chamorro

**Tecnología En Desarrollo De Software**

Programación Orientada A Objetos

Tercer Semestre

**Fecha De Entrega: 26-Abril-2025**

# Tabla de contenido

Contenido

[Tabla de contenido 2](#_Toc195712362)

[Introducción 4](#_Toc195712363)

[Descripción Del Proyecto 6](#_Toc195712364)

[Dominio Del Problema 7](#_Toc195712365)

[Objetivos 8](#_Toc195712366)

[Objetivo General 8](#_Toc195712367)

[Objetivo Especifico 8](#_Toc195712368)

[Tecnologías Utilizadas 9](#_Toc195712369)

[Justificación De Las Tecnologías Elegidas 10](#_Toc195712370)

[Diseño Del Sistema 11](#_Toc195712371)

[Diagrama De Clases (UML) 12](#_Toc195712372)

[Descripción De Las Clases 13](#_Toc195712373)

[Aplicación De Los Principios De POO 15](#_Toc195712374)

[Encapsulamiento 15](#_Toc195712375)

[Herencia 15](#_Toc195712376)

[Polimorfismo 15](#_Toc195712377)

[Abstracción 15](#_Toc195712378)

[Estructura Del Proyecto Y Organización Del Código 16](#_Toc195712379)

[Endpoints RESTful Implementados 18](#_Toc195712380)

[Endpoints De Cliente 18](#_Toc195712381)

[Endpoints De Producto 18](#_Toc195712382)

[Endpoints De Factura 18](#_Toc195712383)

[Endpoints De Usuario 19](#_Toc195712384)

[Endpoints De Categoría 19](#_Toc195712385)

[Endpoints De Pedido 19](#_Toc195712386)

[Pruebas De La API 20](#_Toc195712387)

[Herramienta Utilizada (Postman) 20](#_Toc195712388)

[Evidencias De Pruebas 20](#_Toc195712389)

[Repositorio En GitHub 24](#_Toc195712390)

[Manual Técnico De Instalación Y Uso 24](#_Toc195712391)

[Conclusiones 27](#_Toc195712392)

# Introducción

Este informe presenta el desarrollo de un proyecto de software basado en los principios fundamentales de la Programación Orientada a Objetos (POO), utilizando el lenguaje Java (versión 8 o superior) junto al microframework SparkJava para la creación de una API RESTful. El propósito principal de este proyecto es demostrar la capacidad de aplicar conceptos avanzados de POO como encapsulamiento, herencia, polimorfismo y abstracción, en un entorno de desarrollo real, implementando una solución que simula la gestión administrativa de un sistema de tienda o negocio.

Para llevar a cabo este proyecto, se eligió como entorno de desarrollo el IDE IntelliJ IDEA, el cual ofreció un conjunto completo de herramientas para facilitar la organización, edición, compilación y ejecución del código fuente. Se utilizó Maven como gestor de dependencias, permitiendo integrar fácilmente SparkJava y otras librerías necesarias. El proyecto se ejecuta completamente en memoria, es decir, sin necesidad de una base de datos externa, lo que lo hace ligero y portable, ideal para propósitos académicos o de demostración.

La estructura del sistema está organizada en tres capas bien diferenciadas:

* **Modelo**: representa las entidades del dominio (Cliente, Producto, Factura, Usuario, Pedido y Categoría). Cada clase posee atributos privados y métodos públicos de acceso (getters y setters), asegurando un correcto encapsulamiento.
* **Controlador**: contiene los endpoints REST correspondientes a cada entidad, agrupados en clases separadas como ClienteController, ProductoController, etc. Aquí se implementa el CRUD completo para cada modelo.
* **Vista**: donde se encuentran la clase principal MainApi, que arranca el servidor y configura los endpoints, y la clase MenuConsola, que permite interactuar con la lógica desde consola local.

Durante el desarrollo, se integraron herramientas como Postman para realizar pruebas a los endpoints REST y generar evidencias de los resultados (éxito o fallo), los cuales también se documentan en este informe. Asimismo, el proyecto fue gestionado mediante control de versiones en GitHub, con commits descriptivos y bien organizados, permitiendo un seguimiento transparente del progreso y facilitando el trabajo colaborativo en caso de requerirse.

Este documento incluye la descripción del dominio elegido, los diagramas de clases, una explicación detallada de la arquitectura, los endpoints implementados, resultados de pruebas, y un análisis final que destaca aprendizajes, retos enfrentados y posibles mejoras a futuro.

# Descripción Del Proyecto

El presente proyecto consiste en el diseño y desarrollo de una API RESTful que permite gestionar la información clave de una tienda o negocio. Este sistema ha sido creado como parte del trabajo de la materia de Programación Orientada a Objetos, con el propósito de aplicar los conocimientos teóricos adquiridos a lo largo del semestre en un entorno práctico que simula un caso del mundo real.

La API proporciona funcionalidades que permiten registrar, consultar, modificar y eliminar datos relacionados con diversas entidades del dominio, tales como clientes, productos, facturas, pedidos, categorías y usuarios. Cada una de estas entidades representa un aspecto fundamental en el proceso de gestión de ventas, permitiendo mantener un control estructurado de los movimientos dentro del negocio.

El proyecto está basado en una arquitectura sencilla, compuesta por clases modelo que representan las entidades del sistema, controladores que exponen los servicios web mediante endpoints RESTful, y estructuras de almacenamiento en memoria que simulan la persistencia de datos. Esta estructura modular permite organizar el código de manera clara, separando responsabilidades y facilitando futuras extensiones o modificaciones del sistema.

La API está pensada para ser consumida por cualquier cliente externo, como una aplicación web, una app móvil o una interfaz gráfica, facilitando la interacción con los datos del negocio desde diferentes plataformas. A través de su conjunto de endpoints, el sistema responde a las necesidades básicas de un negocio en cuanto a la gestión de sus operaciones diarias.

Este proyecto representa un ejercicio completo de análisis, diseño e implementación de un sistema backend funcional, orientado a objetos, y preparado para ser conectado fácilmente con interfaces de usuario u otros sistemas.

# Dominio Del Problema

En el contexto de pequeños y medianos negocios, uno de los mayores desafíos es llevar un control ordenado y eficiente de sus operaciones básicas, como la gestión de productos, el registro de clientes, la emisión de facturas, el seguimiento de pedidos y la organización del inventario. Muchas veces, estos procesos se realizan de manera manual, utilizando hojas de cálculo, cuadernos o sistemas poco estructurados, lo cual genera errores, pérdida de información y una falta de trazabilidad en las actividades comerciales.

El problema radica en la ausencia de una solución tecnológica sencilla y accesible que permita automatizar estas tareas sin necesidad de una infraestructura compleja o conocimientos técnicos avanzados. Además, en muchos casos, no se cuenta con herramientas que permitan integrar estos datos con otros sistemas o acceder a ellos desde diferentes dispositivos o ubicaciones.

Este proyecto aborda precisamente ese problema: desarrollar una solución backend que funcione como base para un sistema administrativo de tienda, ofreciendo una API RESTful modular y extensible. A través de ella, se puede gestionar de forma centralizada:

El registro y control de clientes, incluyendo sus datos personales y contacto.

La administración de productos, con su respectiva cantidad, precio y categoría.

La generación de facturas, que resumen las transacciones realizadas.

La gestión de pedidos, como paso previo o alternativo al proceso de facturación.

La clasificación por categorías, facilitando la organización del catálogo de productos.

El registro de usuarios, quienes podrían tener acceso al sistema con diferentes permisos.

Al ofrecer un sistema estructurado, basado en Programación Orientada a Objetos y accesible mediante peticiones HTTP, se proporciona una solución escalable, reutilizable y adaptable a distintos contextos empresariales, contribuyendo a reducir los errores humanos, mejorar la eficiencia operativa y sentar las bases para un ecosistema digital más robusto.

# Objetivos

## Objetivo General

Desarrollar una API RESTful en Java que permita gestionar de forma eficiente los elementos clave de una tienda, aplicando los principios de la Programación Orientada a Objetos (POO)

## Objetivo Especifico

 Implementar el CRUD completo para al menos seis entidades del sistema.

 Aplicar correctamente los pilares de la POO: encapsulamiento, herencia, polimorfismo y abstracción.

 Organizar el proyecto en capas separadas para mejorar la claridad y el mantenimiento del código.

 Exponer los servicios mediante endpoints REST utilizando SparkJava.

 Validar el funcionamiento de la API utilizando Postman o Insomnia.

 Gestionar el proyecto con control de versiones mediante Git y alojarlo en GitHub

# Tecnologías Utilizadas

A lo largo del desarrollo del proyecto se utilizaron diversas herramientas y tecnologías que facilitaron la implementación, organización y prueba de la API. A continuación se describen brevemente:

* **Java (versión 8 o superior):** Lenguaje principal del proyecto, orientado a objetos, robusto y ampliamente utilizado en el desarrollo de aplicaciones backend.
* **SparkJava:** Microframework web ligero para Java que permite construir APIs RESTful de forma rápida, utilizando un enfoque minimalista y flexible.
* **Maven:** Herramienta de gestión de proyectos y dependencias que facilitó la configuración del entorno y la inclusión de librerías como Spark y Gson.
* **Gson:** Librería utilizada para la conversión entre objetos Java y JSON, esencial para el manejo de datos en los endpoints REST.
* **IntelliJ IDEA:** Entorno de desarrollo integrado (IDE) utilizado para escribir, organizar y ejecutar el código de manera eficiente.
* **Postman:** Herramientas de pruebas HTTP utilizadas para validar el correcto funcionamiento de los endpoints implementados.
* **Git y GitHub:** Sistemas de control de versiones y alojamiento del repositorio, que permitieron documentar el progreso del proyecto y mantener un historial de cambios ordenado.

# Justificación De Las Tecnologías Elegidas

Las tecnologías seleccionadas para este proyecto fueron escogidas con base en criterios de simplicidad, eficiencia, facilidad de integración y cumplimiento de los requisitos académicos del curso.

* **Java** fue elegido como lenguaje base por su enfoque orientado a objetos, su sintaxis clara y su madurez como lenguaje de backend. Además, es el lenguaje trabajado a lo largo del semestre, lo cual permite aplicar directamente los conceptos aprendidos en clase.
* **SparkJava** fue seleccionado como framework principal para construir la API RESTful debido a su ligereza, bajo consumo de recursos y facilidad de implementación. A diferencia de otros frameworks como Spring Boot, SparkJava no requiere una configuración compleja y es ideal para proyectos pequeños y medianos, como este.
* **Maven** se utilizó para gestionar dependencias y automatizar la construcción del proyecto. Su integración con SparkJava y otras bibliotecas como Gson facilita el mantenimiento del entorno de desarrollo.
* **Gson** fue la herramienta elegida para serializar y deserializar objetos Java a formato JSON, lo que resulta esencial para la comunicación entre el servidor y los clientes que consumen la API.
* **IntelliJ IDEA** fue usado como entorno de desarrollo por su compatibilidad con Maven, su facilidad para navegar entre clases y su integración con Git.
* **Postman** permitio probar los endpoints de forma clara y ordenada, simulando peticiones HTTP reales y generando evidencia del correcto funcionamiento de la API.
* **Git y GitHub** fueron claves para llevar el control de versiones del proyecto, permitiendo registrar cada avance, mantener copias de seguridad y facilitar la revisión del código.

En conjunto, estas tecnologías proporcionaron una base sólida para cumplir con los objetivos del proyecto, sin requerir una infraestructura compleja y asegurando un desarrollo fluido y organizado.

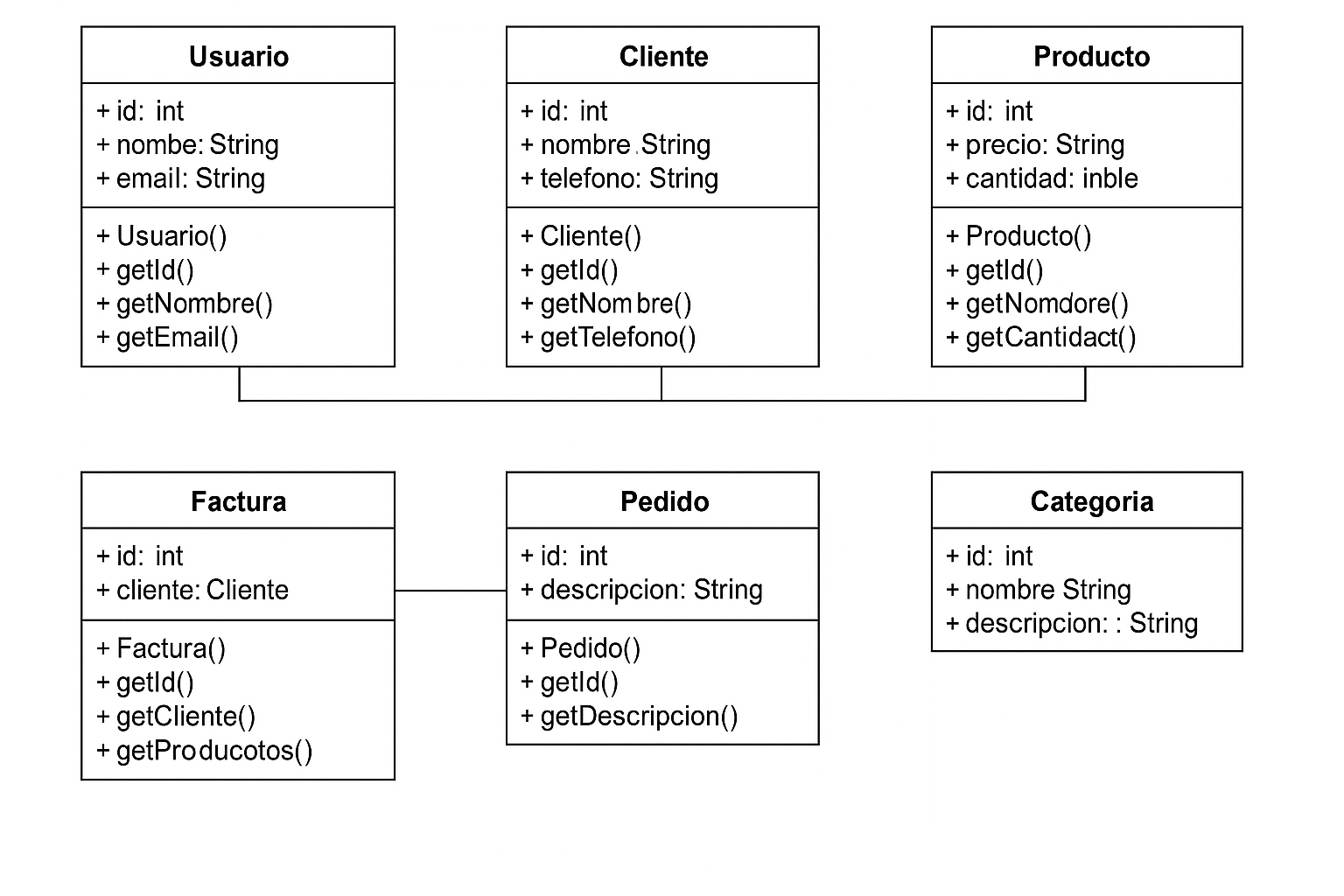
# Diseño Del Sistema

El diseño del sistema se realizó siguiendo una estructura modular y organizada, aplicando el enfoque de la Programación Orientada a Objetos (POO) y la arquitectura en capas. El objetivo principal fue separar las responsabilidades del código, facilitar su mantenimiento y asegurar la escalabilidad del proyecto.

El sistema está dividido en tres capas principales:

1. **Capa Modelo (Modelo/)**  
   Contiene todas las clases que representan las entidades del dominio, como Cliente, Producto, Factura, Usuario, Pedido y Categoría. Cada clase define sus atributos privados, métodos públicos (getters y setters) y constructores, garantizando encapsulamiento y claridad.
2. **Capa Controlador (Controlador/)**  
   Alberga las clases encargadas de gestionar las rutas HTTP expuestas por la API. Cada entidad cuenta con su propio controlador (por ejemplo, ClienteController.java, ProductoController.java) que implementa su respectivo CRUD completo mediante endpoints REST.
3. **Capa Vista / Ejecución (Vista/ y MainApi)**  
   Incluye la clase MainApi.java, que actúa como punto de entrada del sistema y configura todas las rutas de la API. También se incluye una clase MenuConsola.java que permite probar el sistema desde consola de manera local.

## Diagrama De Clases (UML)



El diagrama UML de clases presentado refleja la estructura básica del sistema desarrollado para el proyecto de API RESTful en Java. Este diseño sigue los principios de la Programación Orientada a Objetos (POO), organizando el código en clases bien definidas que interactúan entre sí

## Descripción De Las Clases

A continuación, se describen las clases principales del sistema, agrupadas por paquetes conforme a su rol dentro del diseño de la aplicación.

**POO.Proyecto.Modelo**

Estas clases representan las entidades del dominio, es decir, los objetos que manejan los datos del sistema:

* **Usuario**
  + Representa a un usuario del sistema.
  + Atributos: id (int), nombre (String), email (String).
  + Utilizado para registrar usuarios por consola y enviarlos a la API.
* **Cliente**
  + Modelo para los clientes registrados.
  + Atributos: id (int), nombre (String), telefono (String).
  + Similar a Usuario, pero con un enfoque comercial.
* **Producto**
  + Representa un producto que puede ser vendido o facturado.
  + Atributos: id (int), nombre (String), precio (double), cantidad (int).
* **Factura**
  + Clase que simula una venta. Asocia un cliente con una lista de productos.
  + Atributos: id (int), cliente (Cliente), productos (List<Producto>).
* **Pedido**
  + Modelo para representar solicitudes o pedidos personalizados.
  + Atributos: id (int), descripcion (String).
* **Categoria**
  + Agrupa productos bajo una misma categoría.
  + Atributos: id (int), nombre (String), descripcion (String).
* **BaseDatosMemoria**
  + Clase simulada que actúa como base de datos en memoria para almacenar las listas de entidades mientras se ejecuta la aplicación.

**POO.Proyecto.Vista**

Contiene la interfaz de usuario basada en consola:

* **MenuConsola**
  + Clase principal para interactuar con el usuario.
  + Presenta un menú con opciones para registrar cada tipo de entidad.
  + Realiza peticiones POST a la API REST utilizando HttpURLConnection.
* **MainApi**
  + Encargada de levantar el servidor con Spark Java.
  + Configura los endpoints para manejar las rutas de cada entidad.

**POO.Proyecto.Controlador**

Encapsula la lógica de los endpoints RESTful:

* **UsuarioController, ClienteController, ProductoController, FacturaController, PedidoController, CategoriaController**
  + Cada clase maneja un grupo de rutas (GET, POST, PUT, DELETE) específicas para su respectiva entidad.
  + Utilizan la clase BaseDatosMemoria para acceder y manipular los datos en memoria.

# Aplicación De Los Principios De POO

El desarrollo del proyecto se enfocó en aplicar los cuatro pilares fundamentales de la Programación Orientada a Objetos (POO): Encapsulamiento, Abstracción, Herencia y Polimorfismo

## Encapsulamiento

Todas las clases del sistema declaran sus atributos como private y utilizan métodos públicos (getters y setters) para acceder o modificar los datos. Esto protege la información interna de cada objeto y evita que sea alterada directamente desde otras clases.  
Ejemplo: en la clase Producto, los atributos como precio y cantidad son privados, y se accede a ellos a través de getPrecio() o setCantidad().

## Herencia

Si bien en la versión actual no se implementó una jerarquía directa con herencia entre clases (por ejemplo, una clase padre Persona), el diseño del sistema lo permite fácilmente. Tanto Usuario como Cliente comparten atributos comunes como id y nombre, por lo que se podría crear una clase base abstracta en futuras versiones para reducir la redundancia y reutilizar código.

## Polimorfismo

El sistema puede beneficiarse del polimorfismo mediante el uso de interfaces o clases abstractas en futuras mejoras, por ejemplo, para manejar diferentes tipos de usuarios o productos especiales con comportamientos distintos. Actualmente, se aplica un polimorfismo básico al utilizar listas genéricas (List<Producto>) que permiten trabajar con cualquier objeto Producto sin importar su instancia específica.

## Abstracción

Cada clase representa una entidad real del dominio del sistema, ocultando los detalles internos y mostrando solo lo necesario para su uso. Las clases como Factura, Cliente y Pedido están diseñadas para cumplir un rol específico y no mezclan responsabilidades, lo cual facilita el mantenimiento y la comprensión del código.

# Estructura Del Proyecto Y Organización Del Código

El proyecto ha sido organizado en una estructura de paquetes que separa claramente las responsabilidades de cada componente, lo que facilita la lectura, mantenimiento y escalabilidad del sistema. La organización se realizó siguiendo buenas prácticas de diseño de software, agrupando las clases según su función en la arquitectura del sistema.

POO.Proyecto/

│

├── Modelo/ ← Contiene las clases que representan el modelo de datos.

│ ├── Cliente.java

│ ├── Usuario.java

│ ├── Producto.java

│ ├── Factura.java

│ ├── Pedido.java

│ ├── Categoria.java

│ └── BaseDatosMemoria.java

│

├── Vista/ ← Incluye las clases relacionadas con la interacción del usuario.

│ ├── MenuConsola.java

│ └── MainApi.java

│

├── Controlador/ ← Contiene los controladores de la API (endpoints REST).

│ ├── ClienteController.java

│ ├── UsuarioController.java

│ ├── ProductoController.java

│ ├── FacturaController.java

│ ├── PedidoController.java

│ └── CategoriaController.java

# Endpoints RESTful Implementados

La API desarrollada expone múltiples endpoints RESTful para interactuar con las diferentes entidades del sistema. Cada conjunto de endpoints permite realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) sobre una entidad específica.

## Endpoints De Cliente

GET /clientes → Obtiene todos los clientes registrados.

GET /clientes/:id → Obtiene un cliente específico por ID.

POST /clientes → Registra un nuevo cliente.

PUT /clientes/:id → Actualiza los datos de un cliente existente.

DELETE /clientes/:id → Elimina un cliente por ID.

## Endpoints De Producto

GET /productos → Lista todos los productos disponibles.

GET /productos/:id → Busca un producto por ID.

POST /productos → Registra un nuevo producto.

PUT /productos/:id → Modifica los datos de un producto.

DELETE /productos/:id → Elimina un producto del sistema.

## Endpoints De Factura

GET /facturas → Devuelve todas las facturas registradas.

GET /facturas/:id → (Opcional si se implementa)

POST /facturas → Crea una nueva factura con cliente y productos.

(PUT y DELETE pueden ser opcionales si no se implementa modificación de facturas)

## Endpoints De Usuario

GET /usuarios → Muestra todos los usuarios registrados.

GET /usuarios/:id → Muestra un usuario específico por ID.

POST /usuarios → Crea un nuevo usuario.

PUT /usuarios/:id → Actualiza nombre o correo de un usuario.

DELETE /usuarios/:id → Elimina un usuario.

## Endpoints De Categoría

GET /categorias → Obtiene todas las categorías creadas.

POST /categorias → Registra una nueva categoría.

(PUT y DELETE pueden ser implementados si se requiere administración completa)

## Endpoints De Pedido

GET /pedidos → Lista todos los pedidos.

POST /pedidos → Registra un nuevo pedido.

(PUT y DELETE pueden agregarse si se desea editar/eliminar un pedido)

# Pruebas De La API

Para garantizar el correcto funcionamiento de los endpoints RESTful implementados, se realizaron pruebas utilizando la herramienta Postman. Las pruebas se enfocaron en validar las operaciones CRUD de cada entidad, asegurando que la API responda adecuadamente ante solicitudes válidas e inválidas.

## Herramienta Utilizada (Postman)

Postman fue empleada para enviar solicitudes HTTP a la API en ejecución local (http://localhost:4567).

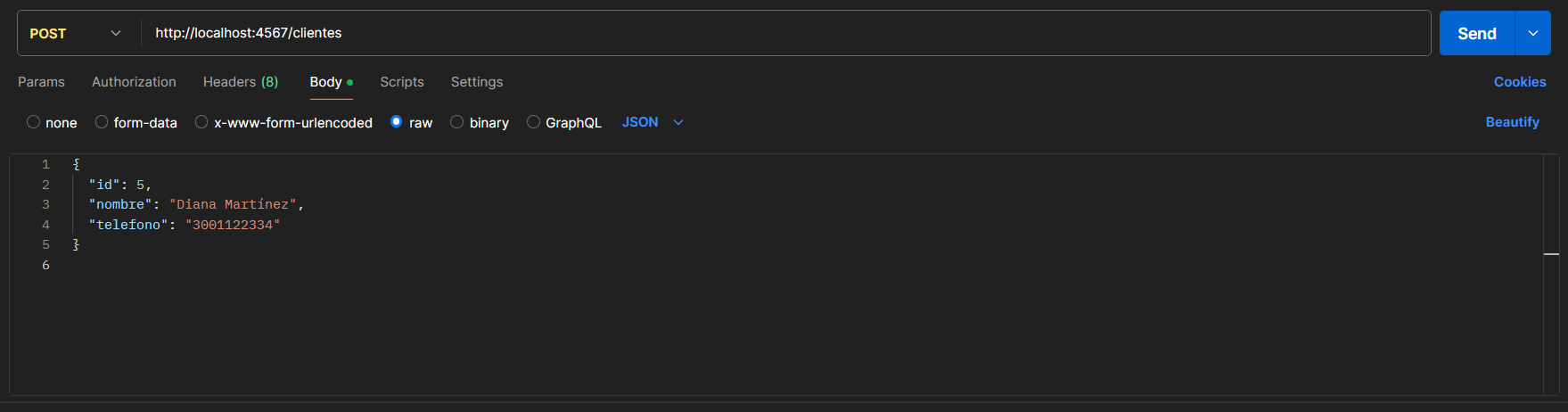
Se configuraron y probaron peticiones GET, POST, PUT y DELETE según el endpoint correspondiente.

Los datos enviados y recibidos fueron en formato JSON, de acuerdo con los estándares REST.

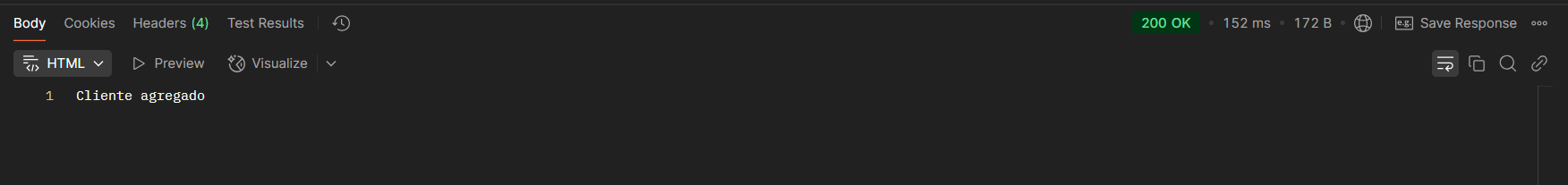
## Evidencias De Pruebas

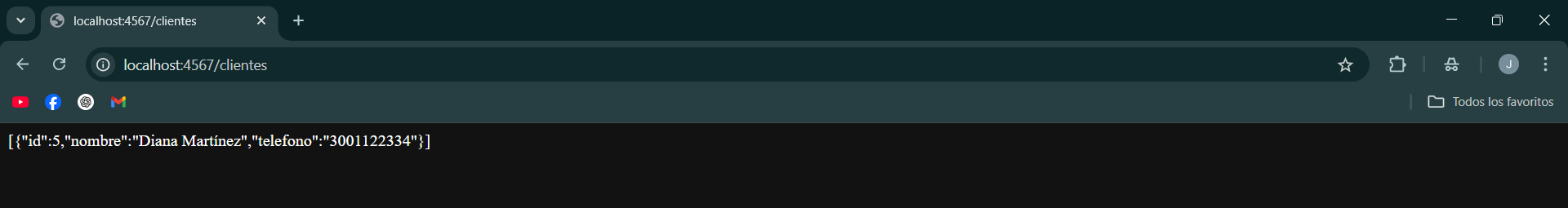
En esta sección se presentan capturas de pantalla y ejemplos reales de las pruebas realizadas a los diferentes endpoints de la API, usando la herramienta Postman. Cada evidencia incluye la solicitud enviada (request), la respuesta obtenida (response) y el código de estado HTTP devuelto por el servidor.

Prueba Exitosa – Crear Cliente (POST /clientes)

Solicitud (JSON):

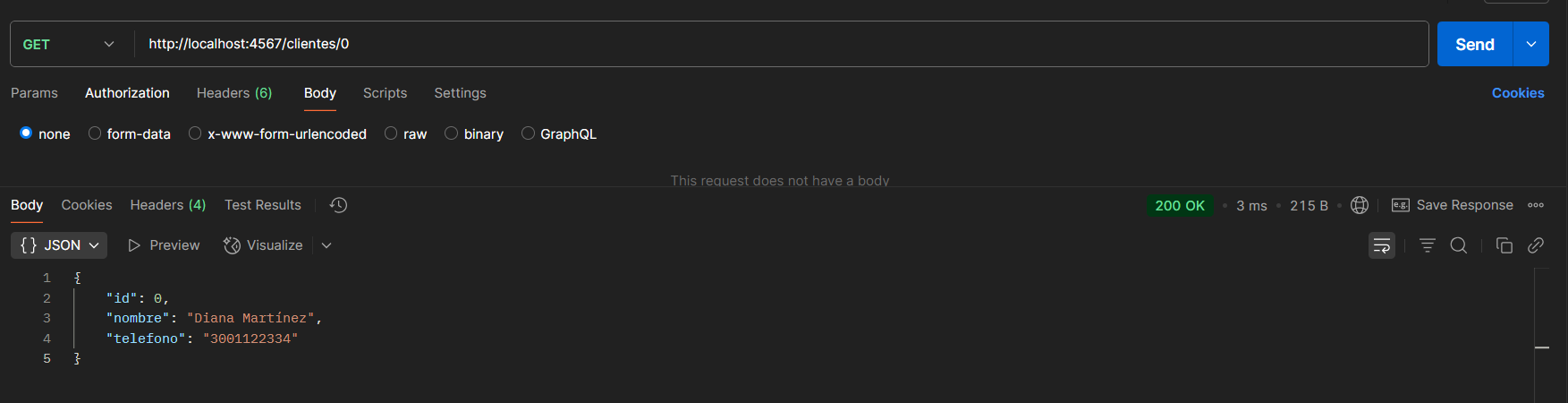
Respuesta esperada:





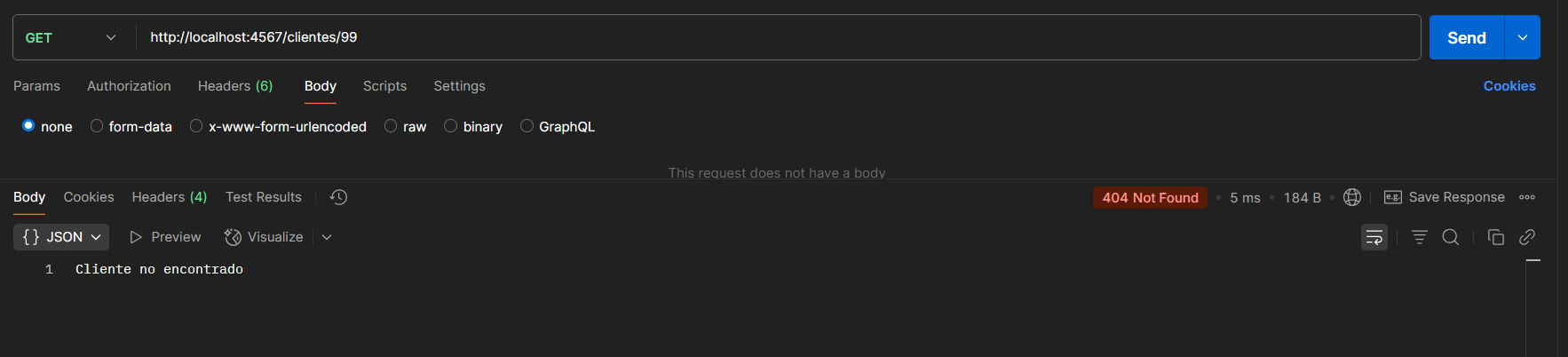
Prueba Exitosa – Obtener Cliente por ID (GET /clientes/5)

Respuesta esperada:



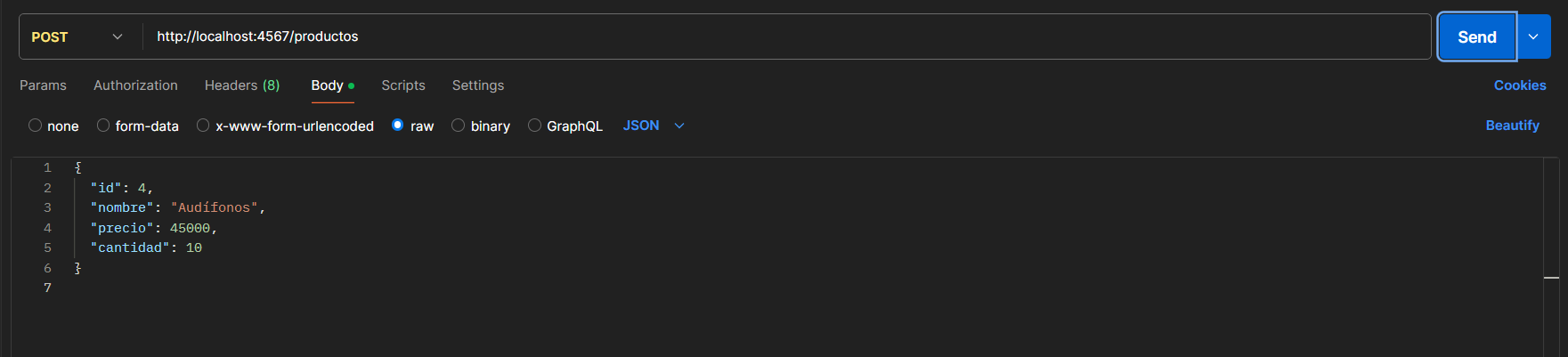
Prueba Fallida – Buscar Cliente Inexistente (GET /clientes/99)

Respuesta esperada:

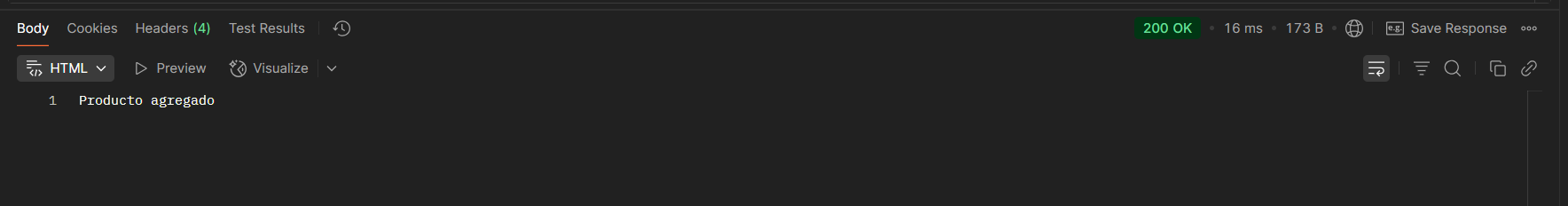


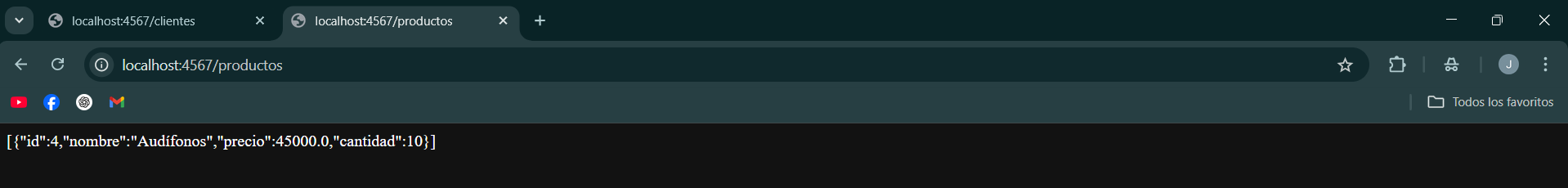
Prueba Exitosa – Crear Producto (POST /productos)

Solicitud (JSON):



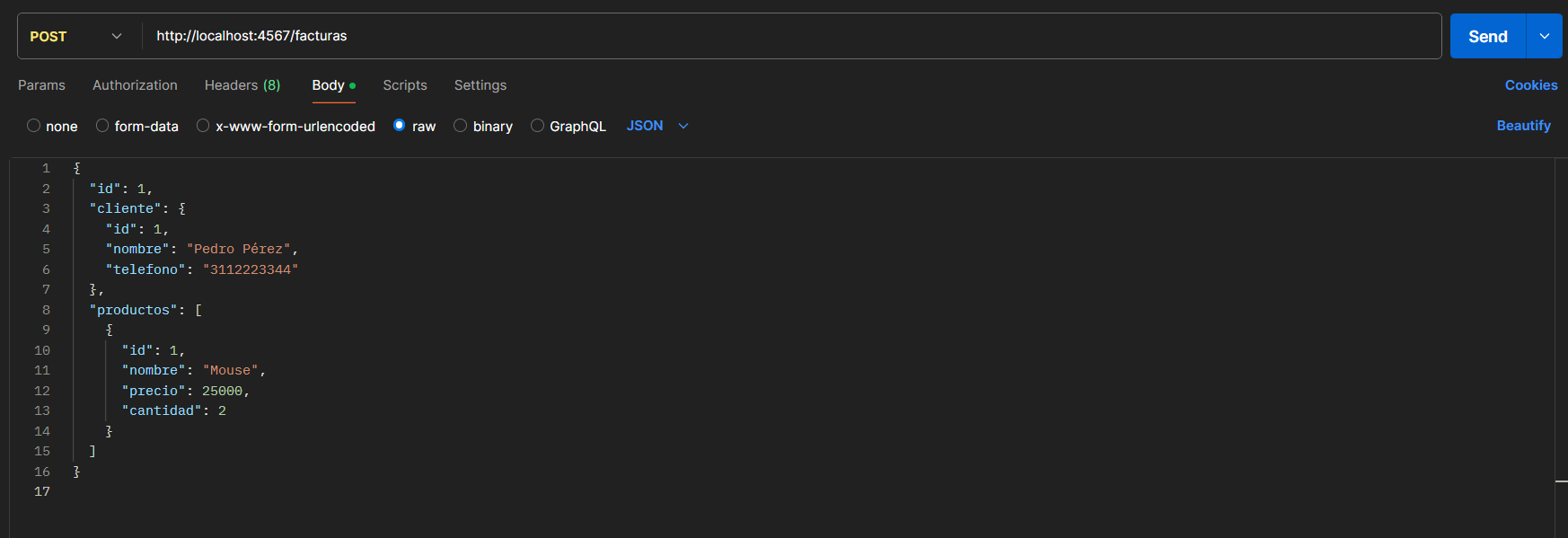
Respuesta esperada:



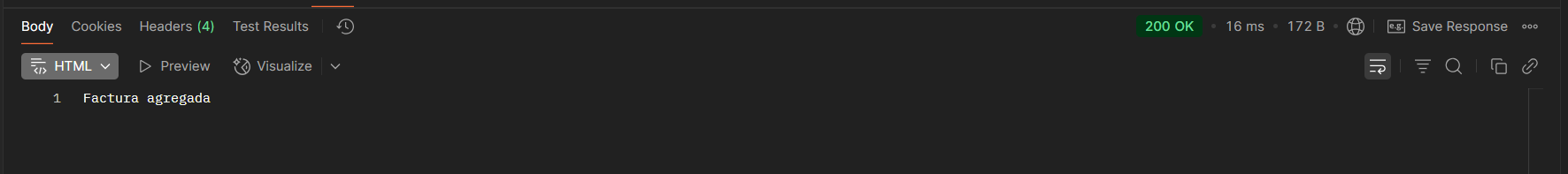


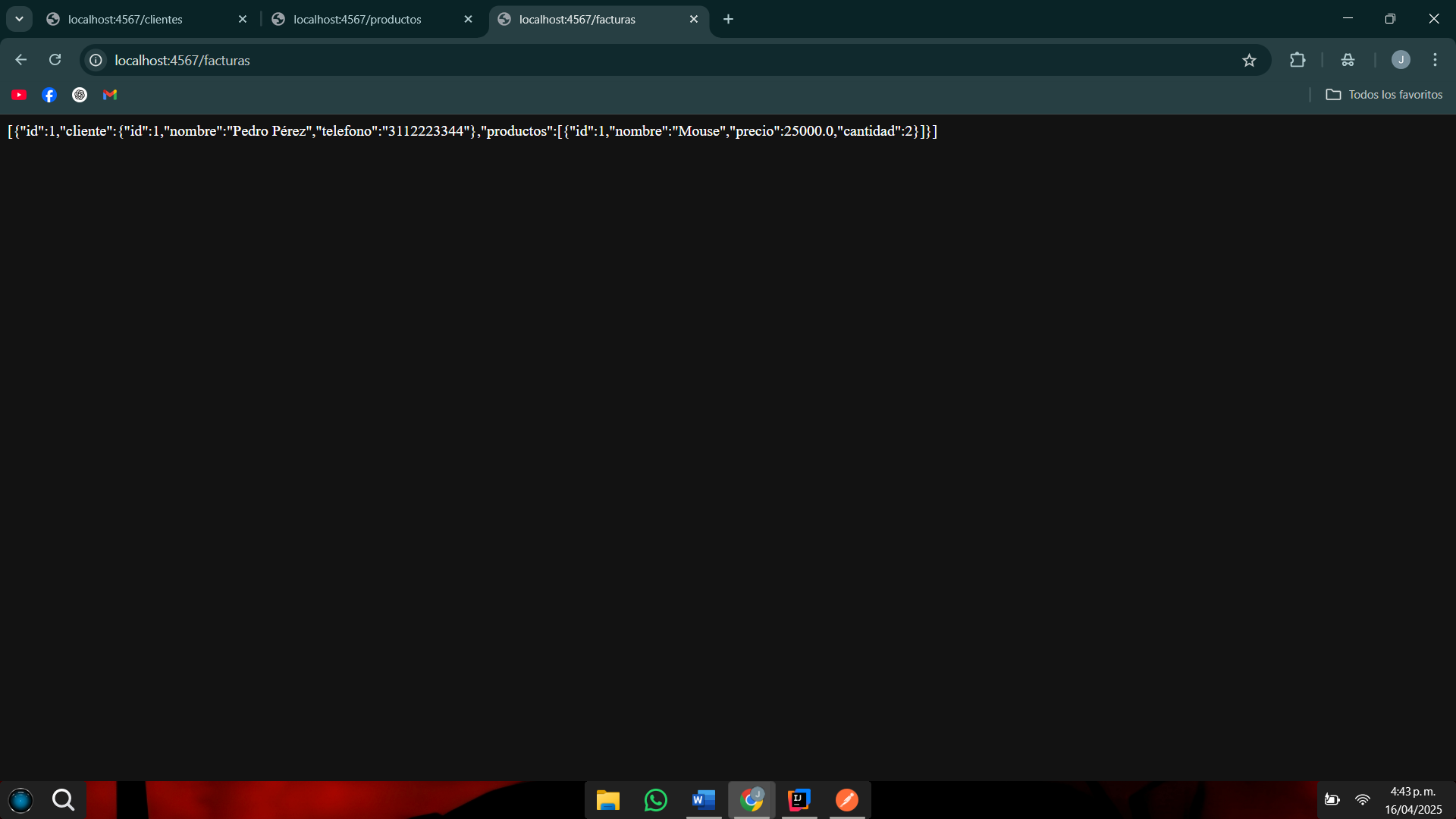
Prueba Exitosa – Crear Factura (POST /facturas)

Solicitud (simulada):



Respuesta esperada:





# Repositorio En GitHub

https://github.com/JonierRD/ProyectoPOO

# Manual Técnico De Instalación Y Uso

**Entorno de desarrollo utilizado**

- IDE: IntelliJ IDEA (versión 2022 o superior)  
- Lenguaje: Java 17  
- Framework: SparkJava (gestión de rutas y servidor embebido)  
- Gestor de dependencias: Maven  
- Serialización: Gson  
- Control de versiones: Git + GitHub  
- Pruebas de API: Postman

**Clonar y abrir el proyecto**

1. Clonar el repositorio desde GitHub:  
git clone https://github.com/tu\_usuario/tu\_repositorio.git  
  
2. Abrir IntelliJ IDEA → Open → Seleccionar la carpeta del proyecto.  
  
3. IntelliJ detectará automáticamente el proyecto como un proyecto Maven. En caso de que no lo haga:  
- Ir a: File > Project Structure > Project SDK → Seleccionar JDK 17  
- En el panel lateral derecho (Maven), hacer clic en el ícono de Refresh

**Estructura general del proyecto**

- Modelo/: Contiene las entidades del dominio (Cliente, Producto, Factura, etc.)  
- Controlador/: Define los endpoints de la API REST para cada entidad.  
- Vista/:   
 - MainApi.java: Punto de entrada de la API (levanta el servidor en Spark).  
 - MenuConsola.java: Interfaz local por consola para pruebas sin usar Postman.

**Ejecución de la API**

1. Ejecutar MainApi.java desde IntelliJ IDEA (botón verde).  
2. El servidor se inicia en el puerto 4567.  
3. La API estará disponible en: http://localhost:4567  
  
Ejemplo de endpoint activo:  
GET http://localhost:4567/clientes

**Pruebas de funcionamiento (Postman)**

- Se usó Postman para probar los endpoints REST de las entidades.  
- Los datos se envían y reciben en formato JSON.  
- Para POST o PUT, el body debe estar en formato raw > application/json.  
  
Ejemplo para registrar un cliente:  
{  
 "id": 0,  
 "nombre": "Juan Pérez",  
 "telefono": "3112233445"  
}  
  
Endpoints principales (Cliente):  
- GET /clientes  
- POST /clientes  
- PUT /clientes/:id  
- DELETE /clientes/:id  
  
Nota: El endpoint GET /clientes/:id aún no está implementado en esta versión.

**Consideraciones técnicas**

- Todos los datos se almacenan en memoria durante la ejecución.  
- No se utiliza base de datos, lo cual permite portabilidad y rapidez para pruebas locales.  
- Al reiniciar el servidor, los datos se reinician.

# Conclusiones

El desarrollo de este proyecto permitió aplicar de manera práctica y efectiva los principios fundamentales de la Programación Orientada a Objetos (POO) en un entorno realista, a través de la implementación de una API RESTful modular, funcional y organizada.

A lo largo del proceso se logró:

* Diseñar un sistema backend funcional, capaz de gestionar múltiples entidades (clientes, productos, usuarios, pedidos, categorías y facturas) mediante operaciones CRUD completas.
* Implementar una arquitectura limpia y escalable, separando el código en capas de modelo, controlador y vista, lo cual facilita su mantenimiento y comprensión.
* Utilizar herramientas modernas y profesionales, como IntelliJ IDEA, SparkJava, Gson, Maven y Postman, que son ampliamente utilizadas en el desarrollo de software en la industria.
* Comprender y aplicar las ventajas del almacenamiento en memoria, útil en etapas tempranas de desarrollo o cuando no se requiere persistencia a largo plazo.
* Documentar el proyecto correctamente, incluyendo pruebas funcionales y evidencias que demuestran el correcto funcionamiento de la API.

Además, se identificaron posibles mejoras para versiones futuras del sistema, como la integración con una base de datos, implementación de autenticación y documentación automática de la API con Swagger.

En conclusión, el proyecto cumplió satisfactoriamente con los objetivos propuestos, demostrando la utilidad de la POO en el desarrollo de soluciones reales, y proporcionando una base sólida para continuar evolucionando hacia sistemas más complejos y robustos.