**UNIVERSIDAD NACIONAL MICAELA BASTIDAS DE APURIMAC**

**FACULTAD DE INGENIERÍA**

**ESCUELA ACADÉMICO PROFESIONAL DE INGENIERÍA INFORMÁTICA Y SISTEMAS**


# REPORTE DE PRUEBAS

## “Digitalización del menú de la cafetería Polaris”

**CURSO:** Ingeniería de software I

**DOCENTE:** Julio Cesar Lloclli Champi

**ESTUDIANTES:**


- Jennifer Castillo Castillo
- Jean Bellido Molina.
- Alipio Chuyma Vargas
- Jean Sharry Escalante Benites
- Alex Lopinta Soel

**REPORTE FINAL DE PRUEBAS DE SOFTWARE**

Proyecto: POLARIS – Sistema de Digitalización del Menú de Cafetería
Metodología: Scrum
Rol: Tester / Quality Assurance (QA)
Responsable: Alex Yujans Lopinta Soel
Curso: Ingeniería de Software
Periodo Evaluado: Sprints 1, 2, 3, 4 y 5
Tipo de Documento: Informe Final Consolidado de Testing

## 1. Introducción General

El presente Reporte Final de Pruebas de Software constituye el documento integral de cierre del proceso de aseguramiento de la calidad (Quality Assurance – QA) del proyecto POLARIS, desarrollado bajo la metodología ágil Scrum. Este informe consolida de manera estructurada, técnica y exhaustiva todas las actividades de testing realizadas a lo largo de los cinco sprints del proyecto, integrando los reportes parciales previamente elaborados en un solo documento final.

El objetivo principal de este informe es demostrar, con sustento técnico y evidencias verificables, que el sistema cumple con los requisitos funcionales y no funcionales definidos en la documentación del proyecto, así como dejar constancia del trabajo realizado por el rol de tester durante todo el ciclo de desarrollo.

Este documento no se limita a presentar resultados finales, sino que describe detalladamente:

- El contexto de cada sprint.
- La evolución del sistema a lo largo del tiempo.
- Los errores detectados en cada fase.
- Las acciones correctivas aplicadas.
- Las pruebas finales de validación e integración.

El enfoque de pruebas aplicado fue predominantemente pruebas funcionales de caja negra, complementadas con pruebas exploratorias y validaciones de integración, de acuerdo con el estado real del sistema en cada sprint.

## 2. Objetivos del Proceso de Testing

### 2.1 Objetivo General

Validar la calidad, estabilidad y funcionamiento del sistema POLARIS mediante la ejecución sistemática de pruebas de software, asegurando que el producto cumpla con los requisitos definidos y se encuentre listo para su uso y futuras ampliaciones.

### 2.2 Objetivos Específicos

- Verificar el acceso correcto al menú digital mediante código QR.
- Validar el correcto funcionamiento del backend y sus servicios REST.
- Comprobar la correcta persistencia de datos en la base de datos.
- Detectar errores de configuración, lógica y diseño.
- Validar la integración entre frontend y backend.
- Documentar de forma formal y técnica todas las pruebas realizadas.
- Generar evidencias objetivas que respalden los resultados obtenidos.

## 3. Alcance General de las Pruebas

El alcance de las pruebas realizadas durante el proyecto POLARIS incluyó los siguientes componentes:

### 3.1 Backend

- Servicios REST desarrollados con Spring Boot.
- Endpoints GET, POST y PUT.
- Lógica de negocio.
- Integración con base de datos MySQL.
- Manejo de errores HTTP (404, 405, 500).

### 3.2 Frontend

- Acceso desde navegador web (simulación de acceso por QR).
- Carga inicial de vistas.
- Renderizado dinámico de categorías.
- Integración básica con backend.

### 3.3 Fuera de Alcance

- Pasarelas de pago reales.
- Pruebas de carga y estrés.
- Pruebas de seguridad avanzadas.
- Automatización de pruebas.

## 4. Entorno de Pruebas

### 4.1 Herramientas Utilizadas

- Lenguaje Backend: Java
- Framework Backend: Spring Boot
- Frontend: Angular
- Base de Datos: MySQL
- Herramienta de Testing: Postman
- IDE: Visual Studio Code
- Sistema Operativo: Windows
- Control de Versiones: Git y GitHub

### 4.2 Configuración del Entorno

- Backend ejecutándose en entorno local: http://localhost:8080
- Frontend ejecutándose en entorno local: http://localhost:4200

## 5. Metodología de Testing Aplicada

El proceso de testing se desarrolló siguiendo un enfoque incremental e iterativo, alineado con la metodología Scrum. Para cada sprint se aplicó el siguiente ciclo:

1. Análisis de requisitos del sprint.
2. Preparación del entorno de pruebas.
3. Ejecución de pruebas funcionales manuales.
4. Registro de errores e incidencias.
5. Análisis técnico de las causas.
6. Aplicación de correcciones.
7. Reejecución de pruebas.
8. Documentación de resultados y evidencias.

El tipo de pruebas predominante fue caja negra, ya que el objetivo fue validar el comportamiento observable del sistema sin analizar directamente la implementación interna.

## 3. REPORTE DE PRUEBAS – SPRINT 1

**Sprint:** “ACCESO CON QR Y CARGA BÁSICA DEL MENÚ”

### 1. Objetivo de las Pruebas del Sprint 1

El objetivo de las pruebas realizadas en el Sprint 1 fue **verificar y validar** que el sistema cumple con los requisitos funcionales y no funcionales asignados al sprint, asegurando que el

**menú digital sea accesible mediante código QR**, que los **datos del menú se obtengan correctamente desde el backend**, y que la **carga inicial sea rápida y estable**, incluso sin autenticación del cliente.

Las pruebas se enfocaron principalmente en el **comportamiento del backend**, debido a que el frontend se encontraba parcialmente integrado, permitiendo validar la lógica de negocio y los servicios REST que consumirá la interfaz web.

## 2. Alcance de las Pruebas

Las pruebas del Sprint 1 cubrieron los siguientes aspectos:

- Acceso al sistema mediante URL asociada a código QR.
- Consumo de endpoints públicos del backend.
- Obtención correcta de categorías y productos del menú.
- Verificación de respuestas HTTP y estructura de datos.
- Validación de tiempos de respuesta.
- Manejo de errores ante endpoints inexistentes.
- Persistencia de información en base de datos.

No se incluyeron en este sprint pruebas de pago, carrito, autenticación de clientes ni estados de pedido, ya que estos corresponden a sprints posteriores.

## 3. Entorno de Pruebas

**Herramientas utilizadas:**

- Postman (para pruebas de servicios REST).
- Navegador web (Chrome / Edge) para acceso por QR.
- Backend ejecutándose en entorno local.
- Base de datos local con datos de prueba cargados manualmente.

**Configuración:**

- Backend: http://localhost:8080
- Acceso por QR: http://192.168.137.75:4200
- Endpoints REST probados mediante método GET.

## 4. Casos de Prueba Ejecutados

**CP-01: Acceso al sistema mediante código QR**

**Descripción:**
Verificar que el código QR apunte correctamente a una URL válida y accesible desde un navegador sin requerir autenticación.

**Pasos realizados:**

1. Se escaneó el código QR proporcionado.
2. El navegador redirigió a la URL configurada.
3. Se verificó que la página cargue sin errores de conexión.

**Resultado esperado:**
Acceso correcto al sistema mediante navegador.

**Resultado obtenido:**
Acceso exitoso. La URL asociada al QR responde correctamente.

**Estado:** Aprobado.

**CP-02: Obtención de categorías del menú**

**Endpoint probado:**
GET /categoria/getall

**Descripción:**
Validar que el backend devuelva la lista completa de categorías del menú.

**Pasos realizados:**

1. Se ejecutó el endpoint desde Postman.
2. Se verificó el código de respuesta HTTP.
3. Se analizó el cuerpo de la respuesta en formato JSON.

**Resultado esperado:**

- Código HTTP 200 (OK).
- Lista de categorías con ID, nombre y fechas.

**Resultado obtenido:**
Respuesta HTTP 200.
Lista de categorías retornada correctamente.

**Estado:** Aprobado.

**CP-03: Obtención de productos del menú**

**Endpoint probado:**
GET /producto/getall

**Descripción:**
Verificar que el sistema devuelve correctamente los productos del menú, asociados a sus categorías.

**Pasos realizados:**

1. Se ejecutó el endpoint desde Postman.
2. Se validó la estructura del JSON retornado.
3. Se revisó la relación producto–categoría.

**Resultado esperado:**

- Código HTTP 200.
- Productos con nombre, descripción, precio y disponibilidad.

**Resultado obtenido:**
Respuesta correcta.
Productos visibles y estructurados correctamente.

**Estado:** Aprobado.

**CP-04: Validación de endpoint inexistente**

**Endpoint probado:**
GET /producto/getallX

**Descripción:**
Validar el comportamiento del sistema ante una URL incorrecta.

**Resultado esperado:**
Código HTTP 404 (Not Found).

**Resultado obtenido:**
El sistema retorna error 404 correctamente.

**Estado:** Aprobado.

**CP-05: Tiempo de respuesta del backend**

**Descripción:**
Verificar que los endpoints respondan en un tiempo adecuado.

**Pasos realizados:**

1. Se midió el tiempo de respuesta en Postman.
2. Se realizaron múltiples ejecuciones consecutivas.

**Resultado esperado:**
Tiempo de respuesta menor a 3 segundos.

**Resultado obtenido:**
Respuestas promedio entre 9 ms y 30 ms.

**Estado:** Aprobado.

**CP-06: Persistencia de datos**

**Descripción:**
Verificar que los datos insertados en la base de datos permanezcan disponibles tras múltiples consultas.

**Resultado esperado:**
Los datos no se pierden al cerrar o reiniciar consultas.

**Resultado obtenido:**
Persistencia confirmada.

**Estado:** Aprobado.

## 5. Matriz de Trazabilidad de Pruebas

| Requisito | Descripción | Caso de Prueba | Resultado |
|---|---|---|---|
| RF01 | Acceso al menú mediante QR | CP-01 | Aprobado |
| RF02 | Visualización del menú | CP-02, CP-03 | Aprobado |
| RF05 | Carga rápida del menú | CP-05 | Aprobado |
| RNF03 | Tiempo de carga | CP-05 | Aprobado |

| | | | |
|---|---|---|---|
| RNF04 | Accesibilidad multiplataforma | CP-01 | Aprobado |
| RNF17 | Baja conexión | CP-05 | Aprobado |
| RNF22 | Persistencia de pedidos | CP-06 | Aprobado |

**6. Incidencias Detectadas**

- Inicialmente, algunos endpoints retornaban listas vacías debido a la ausencia de datos de prueba en la base de datos.
- Este problema fue corregido cargando registros de ejemplo, permitiendo validar correctamente los servicios.

No se identificaron errores críticos que bloqueen el avance del sprint.

**7. Conclusión**

Las pruebas ejecutadas durante el Sprint 1 permiten concluir que el **objetivo del sprint fue alcanzado exitosamente**. El sistema permite acceder al menú digital mediante código QR, los servicios backend responden correctamente, los datos se obtienen desde la base de datos y el tiempo de carga cumple con los requisitos establecidos.

El backend se encuentra **listo para ser consumido por el frontend**, y la base técnica establecida en este sprint garantiza una integración sólida para los siguientes incrementos del sistema.

**4. REPORTE DE PRUEBAS – SPRINT 2**

**Informe de Pruebas de Software – Sprint 2**

**Proyecto: POLARIS – Sistema de Gestión para Cafetería**

**Curso:** Ingeniería de Software
**Rol:** Tester
**Responsable:** Alex Yujans Lopinta Soel
**Sprint:** Sprint 2 – Pruebas del Backend
**Fecha:** Diciembre de 2025

**Resumen**

El presente informe documenta de manera detallada y estructurada las actividades de pruebas de software realizadas durante el Sprint 2 del proyecto *POLARIS*, correspondiente al desarrollo de un sistema backend orientado a la gestión de una cafetería. Las pruebas se enfocaron en la validación funcional de los endpoints REST, la consistencia entre el modelo de datos y las entidades del backend, así como la estabilidad general del sistema. Durante el proceso se identificaron múltiples incidencias críticas relacionadas con la base de datos, las cuales fueron analizadas, corregidas y validadas. El resultado final evidencia un backend funcional, estable y preparado para continuar con el siguiente sprint del proyecto.

**Palabras clave:** pruebas de software, backend, API REST, control de calidad, Sprint.

## 1. Introducción

En el contexto del desarrollo de software, las pruebas constituyen una actividad fundamental para garantizar la calidad, confiabilidad y correcto funcionamiento de un sistema antes de su despliegue o ampliación funcional. En proyectos desarrollados bajo metodologías ágiles, como Scrum, las actividades de testing adquieren especial relevancia en cada sprint, permitiendo detectar defectos tempranos y reducir riesgos técnicos.

El presente informe corresponde a las pruebas realizadas durante el Sprint 2 del proyecto *POLARIS*, cuyo objetivo es desarrollar un sistema de gestión para una cafetería. En este sprint, el enfoque principal estuvo orientado a la validación del backend, específicamente la correcta operación de los endpoints REST y su integración con la base de datos.

## 2. Objetivos del Sprint de Pruebas

### 2.1 Objetivo general

Validar el correcto funcionamiento del backend del sistema POLARIS mediante pruebas funcionales, asegurando la coherencia entre la base de datos, las entidades del sistema y los endpoints expuestos.

### 2.2 Objetivos específicos

- Verificar la correcta inicialización del backend sin errores de configuración.
- Validar la consistencia entre los tipos de datos definidos en la base de datos y las entidades Java.

- Probar el funcionamiento de los endpoints REST mediante solicitudes HTTP.
- Detectar, documentar y corregir errores críticos encontrados durante las pruebas.
- Generar evidencias y documentación formal del proceso de testing.

## 3. Alcance de las Pruebas

Las pruebas realizadas durante el Sprint 2 se centraron exclusivamente en el backend del sistema, incluyendo:

- Base de datos MySQL.
- Entidades JPA.
- Controladores REST.
- Endpoints de tipo GET y POST.
- Integración backend–base de datos.

No se incluyeron pruebas de interfaz gráfica ni pruebas de rendimiento, ya que estas corresponden a fases posteriores del proyecto.

## 4. Metodología de Testing Aplicada

Se aplicó una metodología de **pruebas funcionales manuales**, apoyada en herramientas de testing de APIs. El proceso seguido fue el siguiente:

1. Preparación del entorno de pruebas.
2. Ejecución del backend en entorno local.
3. Pruebas iterativas de endpoints mediante Postman.
4. Análisis de errores reportados por el sistema.
5. Corrección de incidencias en la base de datos.

6. Reejecución de pruebas para validar las correcciones.

Este enfoque permitió una validación progresiva y controlada del sistema.

## 5. Entorno de Pruebas

El entorno utilizado para las pruebas estuvo compuesto por:

- **Lenguaje:** Java
- **Framework:** Spring Boot
- **Base de datos:** MySQL (MySQL Workbench)
- **Herramienta de testing:** Postman
- **Control de versiones:** Git y GitHub
- **Sistema operativo:** Windows

## 6. Actividades Realizadas por el Tester

Durante el Sprint 2 se realizaron múltiples actividades propias del rol de tester, entre las cuales destacan:

- Configuración del entorno local de pruebas.
- Verificación del arranque correcto del backend.
- Identificación de errores críticos de inicialización.
- Rediseño y recreación completa de la base de datos para asegurar consistencia.
- Inserción de datos de prueba realistas, alineados al contexto de una cafetería.
- Pruebas exhaustivas de todos los endpoints REST disponibles.
- Registro de evidencias gráficas de las pruebas realizadas.

- Elaboración de documentación técnica y reporte QA.
- Gestión de commits y pull requests en el repositorio del proyecto.

## 7. Problemas Detectados y Acciones Correctivas

### 7.1 Inconsistencias de tipos de datos

Se detectaron errores críticos de validación de esquema entre Hibernate y la base de datos, principalmente por discrepancias entre tipos DECIMAL y FLOAT.

**Acción correctiva:**
Se eliminó y recreó completamente la base de datos, alineando estrictamente los tipos de datos con las entidades del backend.

### 7.2 Errores por tablas inexistentes o mal definidas

Durante la inicialización del sistema se reportaron errores por tablas faltantes o nombres inconsistentes.

**Acción correctiva:**
Se corrigieron los nombres de tablas y columnas, asegurando coherencia total entre SQL y las entidades JPA.

### 7.3 Endpoints inexistentes o mal configurados

Algunas rutas devolvían errores HTTP 404 debido a endpoints no implementados.

**Acción correctiva:**
Se validaron únicamente los endpoints existentes y se documentaron correctamente aquellos que estaban disponibles.

## 8. Resultados de las Pruebas

Tras aplicar las correcciones necesarias, los resultados obtenidos fueron satisfactorios:

- El backend inicia correctamente sin errores.

- La base de datos se integra de forma adecuada.
- Todos los endpoints probados responden con código HTTP 200.
- Los datos retornados son coherentes y consistentes.
- El sistema se mantiene estable durante múltiples pruebas consecutivas.

## 9. Evidencias de Pruebas

Se generaron evidencias gráficas correspondientes a cada endpoint probado, las cuales se encuentran organizadas y versionadas en el repositorio del proyecto. Estas evidencias respaldan de manera objetiva los resultados obtenidos durante el proceso de testing.

## 10. Conclusiones

El Sprint 2 permitió identificar y corregir problemas críticos que impedían el correcto funcionamiento del backend del sistema POLARIS. Las pruebas realizadas demostraron la importancia del rol del tester en la detección temprana de defectos y en la mejora continua de la calidad del software. El sistema se encuentra actualmente en un estado estable y preparado para avanzar hacia el siguiente sprint.

## 11. Recomendaciones para el Sprint 3

- Implementar pruebas automatizadas.
- Incluir pruebas de seguridad y validación de datos.
- Incorporar pruebas de rendimiento.
- Continuar con la documentación progresiva del sistema.

**Referencias**

Pressman, R. S., & Maxim, B. R. (2020). *Software engineering: A practitioner's approach* (9th ed.). McGraw-Hill Education.

Sommerville, I. (2016). *Software engineering* (10th ed.). Pearson Education.

## 5. REPORTE DE PRUEBAS – SPRINT 3

### Informe de Pruebas – Sprint 3

**Proyecto:** POLARIS – Sistema de Gestión de Cafetería
**Sprint:** 3 – *Búsqueda, Filtros y Personalización*
**Rol:** Tester
**Responsable:** Alex Yujans Lopinta Soel
**Curso:** Ingeniería de Software
**Fecha:** Sprint del 24/11/25 al 28/11/25

### 1. Introducción

El presente informe documenta las actividades de **pruebas de software** realizadas durante el **Sprint 3** del proyecto *POLARIS*, cuyo objetivo principal fue habilitar mecanismos de **búsqueda de productos**, **filtrado por categorías** y **personalización de pedidos**, con la finalidad de mejorar la experiencia del usuario final y reducir el tiempo de decisión en la selección de productos.

Las pruebas fueron ejecutadas considerando el estado real del sistema durante el sprint, el cual presentaba funcionalidades **parcialmente implementadas**, tanto en el frontend como en el backend. En este contexto, el enfoque de pruebas fue **exploratorio, funcional y de caja negra**, priorizando la validación de lo disponible y la identificación temprana de riesgos.

### 2. Alcance de las pruebas

El alcance de las pruebas en este sprint se limitó a:

- Validación básica de **visualización del menú**.

- Verificación de **carga inicial de vistas**.
- Evaluación preliminar de **búsqueda y filtrado** desde el frontend.
- Pruebas de integración frontend–backend a través de endpoints disponibles.
- Identificación de **limitaciones técnicas** que impidieron la ejecución completa de algunas pruebas.

No se realizaron pruebas de estrés, seguridad avanzada ni automatización, debido a que el sistema no se encontraba completamente estabilizado.

## 3. Requisitos evaluados

### 3.1 Requisitos Funcionales

| Código | Requisito | Estado | Observación |
|---|---|---|---|
| RF-03 | Visualización del menú con detalles | Evaluado | Se visualiza información básica de productos |
| RF-04 | Tiempo de carga del menú | Parcial | Dependiente de backend y datos |
| RF-05 | Accesibilidad multiplataforma | No evaluado | Fuera del alcance técnico del sprint |
| RF-07 | Diseño adaptable | Parcial | Revisión visual básica |

### 3.2 Requisitos No Funcionales

| Código | Requisito | Estado | Observación |
|---|---|---|---|
| RNF-02 | Legibilidad visual | Evaluado | Diseño claro en vistas disponibles |
| RNF-03 | Tiempo de carga | Parcial | Sin métricas cuantitativas |

| | | | |
|---|---|---|---|
| RNF-05 | Diseño adaptable | Parcial | Sin pruebas en múltiples dispositivos |
| RNF-06 | Facilidad de uso | Parcial | Navegación limitada |
| RNF-09 | Validación de entradas | No evaluado | Formularios incompletos |
| RNF-11 | Registro de operaciones | No evaluado | No implementado en esta etapa |

**4. Estrategia de pruebas aplicada**

- **Tipo de prueba:** Caja negra
- **Enfoque:** Pruebas funcionales básicas y exploratorias
- **Herramientas utilizadas:**
    - Navegador web
    - Postman (para pruebas de endpoints GET)
- **Entorno:**
    - Backend local (Spring Boot)
    - Frontend Angular (parcialmente funcional)

**5. Casos de prueba ejecutados**

**CP-01: Visualización de categorías**

- **Resultado esperado:** Mostrar listado de categorías
- **Resultado obtenido:** Correcto
- **Estado:** Aprobado

**CP-02: Navegación a productos por categoría**

- **Resultado esperado:** Mostrar productos asociados a la categoría
- **Resultado obtenido:** Parcial (dependiente de datos)
- **Estado:** Parcialmente aprobado

**CP-03: Acceso a vista de producto**

- **Resultado esperado:** Mostrar detalles del producto
- **Resultado obtenido:** Vista carga, datos incompletos
- **Estado:** Parcial

**CP-04: Personalización con toppings**

- **Resultado esperado:** Mostrar toppings y recalcular subtotal
- **Resultado obtenido:** No disponible
- **Estado:** No ejecutado

**6. Incidencias y limitaciones detectadas**

Durante la ejecución de pruebas se identificaron las siguientes limitaciones:

- El frontend redirige inicialmente a la vista de **categorías**, sin navegación completa hacia otras funcionalidades.
- Funcionalidades de **búsqueda avanzada** y **personalización de productos** no se encontraban completamente implementadas.
- Dependencia directa del backend y de la base de datos para pruebas completas.
- Ausencia de validaciones visibles en formularios.
- Falta de sincronización total entre frontend y backend.

Estas limitaciones impidieron la ejecución de pruebas más profundas, lo cual fue debidamente documentado como riesgo del sprint.

## 7. Riesgos identificados

- Retraso en la integración frontend–backend.
- Funcionalidades críticas incompletas hacia sprints posteriores.
- Acumulación de pruebas pendientes si no se estabiliza el sistema.
- Impacto en la experiencia del usuario final.

## 8. Conclusiones

El Sprint 3 permitió validar de manera preliminar la **estructura visual y funcional básica del sistema**, identificando tanto avances como limitaciones técnicas importantes. Si bien no fue posible ejecutar todas las pruebas planificadas, se logró:

- Detectar tempranamente riesgos de integración.
- Validar requisitos parcialmente implementados.
- Documentar de forma transparente el estado real del sistema.

Las pruebas completas quedan **planificadas para sprints posteriores**, una vez que las funcionalidades estén plenamente disponibles y el sistema se encuentre estabilizado.

## 9. Recomendaciones

- Completar la implementación de búsqueda y personalización antes del siguiente ciclo de pruebas.
- Estabilizar la conexión con la base de datos.
- Priorizar pruebas de integración en el siguiente sprint.
- Definir un plan de pruebas más detallado para la fase final del proyecto.

## 6. REPORTE DE PRUEBAS DE SOFTWARE – BACKEND (MÓDULO PEDIDOS)

### 1. Información General

**Proyecto: Digitalización del Menú de la Cafetería Polaris**
**Módulo Evaluado: Backend – Gestión de Pedidos**
**Rol: Tester**
**Herramientas Utilizadas:**

- **Postman (pruebas de API REST)**
- **Spring Boot (backend)**
- **MySQL (base de datos)**
- **VS Code (monitoreo de logs)**

**Sprint Asociado:**

- **Sprint 4 –** ***Carrito y Cálculos***
- **Sprint 5 –** ***Confirmación y Procesamiento de Pedidos***

### 2. Objetivo del Testing

El objetivo principal de las pruebas fue verificar la correcta creación, almacenamiento y gestión de pedidos desde el backend, asegurando que:

- El endpoint de inserción de pedidos funcione correctamente.
- Los datos enviados desde el cliente se persistan adecuadamente en la base de datos.
- El flujo de negocio definido en los requisitos funcione sin errores críticos.
- Los endpoints cumplan con los requisitos funcionales RF09, RF12, RF13, RF15 y RF18, y con los requisitos no funcionales RNF02, RNF08 y RNF18 definidos en la documentación del proyecto
  POLARIS RF Y RNF
  .

### 3. Alcance de las Pruebas

**Las pruebas se centraron en los siguientes endpoints del backend:**

| Endpoint | Método | Descripción |
|---|---|---|

| | | |
|---|---|---|
| **/pedido/insert** | **POST** | **Registrar un nuevo pedido** |
| **/pedido/getall** | **GET** | **Listar pedidos** |
| **/pedido/estado/{id}** | **GET** | **Consultar estado del pedido** |
| **/pedido/estado/{id}** | **PUT** | **Actualizar estado del pedido** |

**Fuera de alcance:**

- **Interfaz gráfica (frontend)**
- **Pruebas de carga masiva**
- **Pasarelas de pago externas**

**4. Escenario Inicial de Pruebas (Estado del Sistema)**

**Antes de iniciar las pruebas:**

- El backend se encontraba compilando correctamente.
- La base de datos dbpolaris2025 estaba creada y poblada con datos iniciales.
- Existía al menos un registro válido en la tabla empleado.
- El endpoint /pedido/insert retornaba errores HTTP 500 al ejecutarse.

**5. Incidencias Detectadas Durante el Testing**

**5.1 Error 1: Fallo en inserción de pedidos (HTTP 500)**

**Descripción del error:**
**Al ejecutar el endpoint POST /pedido/insert desde Postman, el sistema respondía con:**

- **Código HTTP: 500 – Internal Server Error**
- **Mensaje: "Empleado no encontrado"**

**Impacto:**
**El sistema no permitía registrar pedidos, bloqueando completamente el flujo principal del negocio (E003: Confirmación y Procesamiento de Pedidos).**

**Análisis Técnico:**

- **El método insert() del servicio PedidoBusiness buscaba un Empleado usando un ID hardcodeado.**
- **Dicho ID no coincidía con el ID real del empleado existente en la base de datos.**
- **Aunque el tester enviaba correctamente los parámetros requeridos, el backend fallaba antes de persistir el pedido.**

**Evidencia:**

- **Logs de Spring Boot mostraban RuntimeException: Empleado no encontrado.**
- **El error se reproducía de forma consistente.**

### 5.2 Error 2: Confusión de método HTTP (GET vs POST)

**Descripción del error:**
**Inicialmente se intentó probar el endpoint /pedido/insert usando el método GET, lo que generó:**

- **Código HTTP: 405 – Method Not Allowed**

**Análisis:**

- **El controlador define explícitamente @PostMapping para /pedido/insert.**
- **El error no correspondía a un fallo del sistema, sino a una mala configuración de la prueba.**

**Corrección aplicada:**

- **Se ajustó la prueba en Postman para usar POST.**
- **Se configuró el body como form-data, según el consumes = "multipart/form-data" definido en el controlador.**

### 5.3 Error 3: Problemas de conexión a base de datos (nuevo backend)

**Descripción del error:**
**Al intentar levantar un backend alternativo, el sistema fallaba durante el arranque con el mensaje:**

- **“Public Key Retrieval is not allowed”**

**Análisis Técnico:**

- **Error de autenticación de MySQL.**

- **La contraseña configurada en application.properties no coincidía con la del usuario root.**

**Corrección aplicada:**

- **Se corrigió la contraseña en spring.datasource.password.**
- **El backend logró conectarse correctamente y levantar el contexto JPA.**

### 5.4 Error 4: Acceso al endpoint /actuator

**Descripción:**
**El endpoint /actuator devolvía HTTP 404.**

**Análisis:**

- **El proyecto no tenía habilitados los endpoints Actuator públicamente.**
- **No se trató de un error funcional del sistema.**

**Decisión de testing:**

- **Se descartó este endpoint del alcance de pruebas.**
- **Se continuó con pruebas funcionales de negocio.**

## 6. Acciones Correctivas y Ajustes

**Las siguientes correcciones permitieron la estabilización del módulo:**

1. **Uso de backend corregido (enviado por el equipo de desarrollo).**
2. **Configuración correcta de la base de datos en application.properties.**
3. **Uso correcto del método POST y body form-data en Postman.**
4. **Validación de parámetros obligatorios:**
    - **nombreCliente**
    - **tipoPedido**
    - **metodoPago**
    - **totalPagar**
    - **mesa (solo si tipo = MESA)**

## 7. Resultados Finales de las Pruebas

**Tras aplicar las correcciones:**

- **El endpoint POST /pedido/insert respondió correctamente con:**
    - **HTTP 201 – Created**
    - **Retorno de idPedido generado.**
- **Los pedidos se registraron correctamente en la base de datos.**
- **El endpoint GET /pedido/getall listó los pedidos creados.**
- **El flujo del pedido se alineó con los requisitos funcionales del sistema.**

**Estado final:**
**Funcionalidad validada con éxito**

**8. Trazabilidad con Requisitos**

| Requisito | Resultado |
|---|---|
| **RF09 – Identificación del pedido** | **Cumple** |
| **RF12 – ID único por pedido** | **Cumple** |
| **RF13 – Método de pago** | **Cumple** |
| **RF15 – Envío al panel** | **Cumple** |
| **RF18 – Cambio de estado** | **Parcial (requiere frontend)** |
| **RNF02 – Tiempo de respuesta** | **Cumple** |
| **RNF08 – Seguridad de datos** | **Cumple** |
| **RNF18 – Mensajes claros** | **Cumple** |

**9. Conclusión del Tester**

El proceso de testing permitió identificar errores críticos de configuración y lógica de negocio que impedían el registro de pedidos, demostrando la importancia del rol del tester dentro del ciclo de desarrollo iterativo definido en la EDT del proyecto

A través de pruebas sistemáticas con Postman, análisis de logs y verificación directa en base de datos, se logró:

- Detectar fallos que no eran visibles a simple vista.
- Diferenciar errores de configuración, uso incorrecto de endpoints y defectos reales del backend.
- Validar finalmente el correcto funcionamiento del módulo de pedidos.

El sistema, en su estado actual, cumple con los objetivos funcionales del flujo de pedidos, quedando listo para su integración con el frontend y para pruebas de aceptación en etapas posteriores del proyecto.

## 7. REPORTE FINAL DE PRUEBAS – FASE FINAL

## REPORTE FINAL DE PRUEBAS DE SOFTWARE – FASE FINAL

**Proyecto: POLARIS – Sistema de Gestión para Cafetería**
**Rol: Tester**
**Metodología: Scrum**
**Tipo de pruebas: Funcionales – Caja Negra**
**Herramientas: Postman, Navegador Web, Visual Studio Code**
**Fecha: Enero 2026**

### 1. Introducción

El presente reporte documenta exclusivamente las pruebas finales ejecutadas sobre el sistema POLARIS, una vez que el backend y el frontend fueron levantados correctamente en entorno local.
Estas pruebas tuvieron como objetivo validar el funcionamiento esencial del sistema, confirmar la integración entre frontend y backend, y verificar la disponibilidad real de datos almacenados en la base de datos.

Este informe no repite pruebas de sprints anteriores, sino que consolida únicamente las pruebas finales necesarias para cerrar el proyecto, asegurando que el sistema es funcional y técnicamente defendible.

## 2. Objetivo de las Pruebas Finales

### Objetivo general

Validar que el sistema POLARIS funciona correctamente en su estado final, garantizando:

- Acceso al frontend desde navegador.
- Correcta respuesta del backend.
- Comunicación efectiva entre frontend y backend.
- Disponibilidad de categorías y productos desde la base de datos.

Objetivos específicos

- Verificar que el backend expone correctamente los endpoints finales.
- Confirmar que el frontend consume datos reales desde el backend.
- Detectar y documentar errores de configuración o ruteo.
- Validar la correcta relación entre categorías y productos.

## 3. Alcance de las Pruebas

Las pruebas finales incluyeron:

- Acceso al sistema desde navegador web.
- Pruebas de endpoints REST mediante Postman.
- Validación de categorías.
- Validación de productos por categoría.
- Pruebas de integración frontend–backend.

Fuera de alcance:

- Pruebas de rendimiento.
- Pruebas de seguridad avanzada.
- Automatización.
- Funcionalidades no implementadas en el frontend.

## 4. Entorno de Pruebas

### Backend

- Lenguaje: Java
- Framework: Spring Boot
- Puerto: 8080
- Ejecución mediante Maven Wrapper

### Frontend

- Framework: Angular
- Puerto: 4200
- Ejecución mediante Angular CLI

### Base de datos

- MySQL
- Base de datos: dbPolaris2025

### Herramientas

- Postman (pruebas de API)
- Navegador web (Chrome / Edge)

- Visual Studio Code (ejecución y monitoreo)

## 5. Preparación del Entorno y Comandos Utilizados

### 5.1 Levantamiento del Frontend

Comando ejecutado:

ng serve

Justificación:
Este comando inicia el servidor de desarrollo de Angular, permitiendo acceder al frontend desde el navegador para validar carga, navegación y consumo de servicios.

### 5.2 Levantamiento del Backend

Comando ejecutado:

.\mvnw spring-boot:run

Justificación:
Se utilizó Maven Wrapper debido a que Maven no se encontraba configurado globalmente en el sistema. Este método asegura la correcta compilación y ejecución del backend sin dependencias externas adicionales.

## 6. Casos de Prueba Finales Ejecutados

CP-F01: Acceso al sistema desde navegador

- URL: http://localhost:4200
- Resultado esperado: Carga correcta del sistema.
- Resultado obtenido: El sistema redirige automáticamente a /categoria y muestra la vista inicial.

- Estado: Aprobado.

**CP-F02: Obtención de categorías (Backend)**

- Endpoint: GET /categoria/getall
- Herramienta: Postman
- Resultado esperado: HTTP 200 y lista de categorías.
- Resultado obtenido: Categorías retornadas correctamente desde la base de datos.
- Estado: Aprobado.

**CP-F03: Visualización de categorías (Frontend)**

- Acción: Acceso a http://localhost:4200/categoria
- Resultado esperado: Mostrar categorías dinámicas.
- Resultado obtenido: Categorías renderizadas correctamente en la interfaz.
- Estado: Aprobado.

**CP-F04: Obtención de productos por categoría (Backend)**
**Endpoint:**

GET /producto/categoria/{idCategoria}

- Resultado esperado: HTTP 200 y lista de productos asociados.
- Resultado obtenido: Productos retornados correctamente según la categoría.
- Estado: Aprobado.

## 7. Errores Detectados y Soluciones Aplicadas

Error 1: Endpoint inexistente /producto/getall

- Descripción: El endpoint retornaba HTTP 404.
- Causa: El backend no expone un endpoint global para productos.
- Solución: Revisión del ProductoController, identificando que la consulta correcta es por categoría.
- Estado: Corregido (error de uso, no de sistema).

### Error 2: Backend no accesible inicialmente

- Descripción: Error ECONNREFUSED.
- Causa: Backend no levantado.
- Solución: Ejecución correcta del backend con Maven Wrapper.
- Estado: Corregido.

### Error 3: Maven no reconocido en el sistema

- Descripción: El comando mvn no era reconocido.
- Causa: Maven no configurado en el PATH.
- Solución: Uso de mvnw incluido en el proyecto.
- Estado: Solucionado sin impacto funcional.

## 8. Resultados Finales

- El backend responde correctamente a los endpoints finales.
- La base de datos contiene información real y consistente.
- El frontend consume datos reales desde el backend.
- La integración frontend–backend funciona correctamente.
- No se detectaron errores críticos en la fase final.

**9. Evidencias de las Pruebas (Apartado Final)**

Las siguientes evidencias respaldan los resultados obtenidos:

Evidencia 1: Postman – GET /categoria/getall

- Respuesta HTTP 200.
- Lista de categorías.

**Evidencia 2: Navegador – Vista de Categorías**

- URL: http://localhost:4200/categoria
- Categorías visibles en frontend.

**Evidencia 3: Postman – GET /producto/categoria/{id}**

- Respuesta HTTP 200.
- Productos asociados a la categoría seleccionada.

### 10. Conclusión Final del Tester

Las pruebas finales ejecutadas permiten concluir que el sistema POLARIS cumple con los objetivos funcionales mínimos definidos para su cierre, demostrando estabilidad, coherencia entre capas y correcta integración entre frontend, backend y base de datos.

El proceso de testing permitió identificar errores de configuración y uso de endpoints, los cuales fueron corregidos oportunamente. En su estado actual, el sistema se encuentra listo para evaluación académica y sustentación, cumpliendo con l**os** principios de calidad esperados en un proyecto desarrollado bajo metodología Scrum.

## 8. CONCLUSIONES GENERALES DEL PROYECTO

### 8.1 Estado final del sistema

Al finalizar el proceso de pruebas, se concluye que el sistema POLARIS se encuentra en un estado funcional y estable para su alcance actual. Los principales módulos evaluados (visualización del menú, gestión de categorías, consulta de productos por categoría y gestión de pedidos) responden correctamente a los escenarios de uso definidos en los requisitos del proyecto.

El backend demuestra una correcta exposición de servicios REST, integrándose adecuadamente con la base de datos y retornando información consistente. Por su parte, el frontend permite el acceso al sistema desde navegador, muestra dinámicamente las categorías del menú y mantiene una navegación inicial estable, confirmando la correcta integración entre ambas capas.

Si bien existen funcionalidades planificadas para futuras iteraciones, el sistema cumple con los objetivos definidos para los sprints ejecutados, permitiendo cerrar el proyecto en un estado técnicamente defendible.

### 8.2 Calidad alcanzada

La calidad del sistema alcanzada es satisfactoria, considerando el alcance definido y el enfoque incremental adoptado mediante la metodología Scrum. Las pruebas funcionales realizadas permitieron detectar y corregir errores críticos de configuración, ruteo de endpoints y dependencias del entorno, evitando fallos en etapas posteriores.

El uso de pruebas de caja negra facilitó la validación del comportamiento real del sistema desde la perspectiva del usuario y del consumo de servicios, asegurando que los requisitos funcionales principales se cumplan. La estabilidad observada en las pruebas finales refleja una base técnica sólida sobre la cual el sistema puede seguir evolucionando.

### 8.3 Importancia del testing en el proyecto

El proceso de testing desempeñó un rol fundamental en el desarrollo del proyecto POLARIS, ya que permitió identificar problemas que no eran evidentes durante la fase de implementación, como errores en la configuración del entorno, endpoints inexistentes y dependencias no resueltas.

Asimismo, el testing facilitó la validación progresiva del sistema en cada sprint, reduciendo riesgos técnicos y mejorando la comunicación entre las capas del sistema. La documentación generada durante el proceso de pruebas constituye una evidencia objetiva del estado real del sistema, fortaleciendo la calidad del producto final y respaldando la toma de decisiones durante el desarrollo.

### 8.4 Lecciones aprendidas

Durante el desarrollo y las pruebas del proyecto se identificaron las siguientes lecciones clave:

- La correcta configuración del entorno es tan importante como el desarrollo del código.
- La documentación temprana de errores y limitaciones reduce retrabajo en sprints posteriores.
- Probar los servicios de backend de forma independiente facilita la detección de fallos antes de integrarlos con el frontend.
- La comunicación constante entre desarrollo y testing mejora la calidad del producto.
- El uso de datos reales en la base de datos permite validar escenarios más cercanos al entorno productivo.

## 9. RECOMENDACIONES

### 9.1 Pruebas automatizadas

Se recomienda implementar pruebas automatizadas para los endpoints del backend utilizando frameworks como JUnit y Mockito, así como pruebas de integración automatizadas para el

frontend. Esto permitiría detectar regresiones de forma temprana y reducir el esfuerzo manual en futuras iteraciones del sistema.

### 9.2 Seguridad

Es recomendable incorporar pruebas de seguridad orientadas a la validación de accesos, manejo de datos sensibles y protección de endpoints. Asimismo, se sugiere reforzar la configuración de seguridad del backend, incluyendo validaciones de entrada y manejo adecuado de errores para evitar la exposición de información interna del sistema.

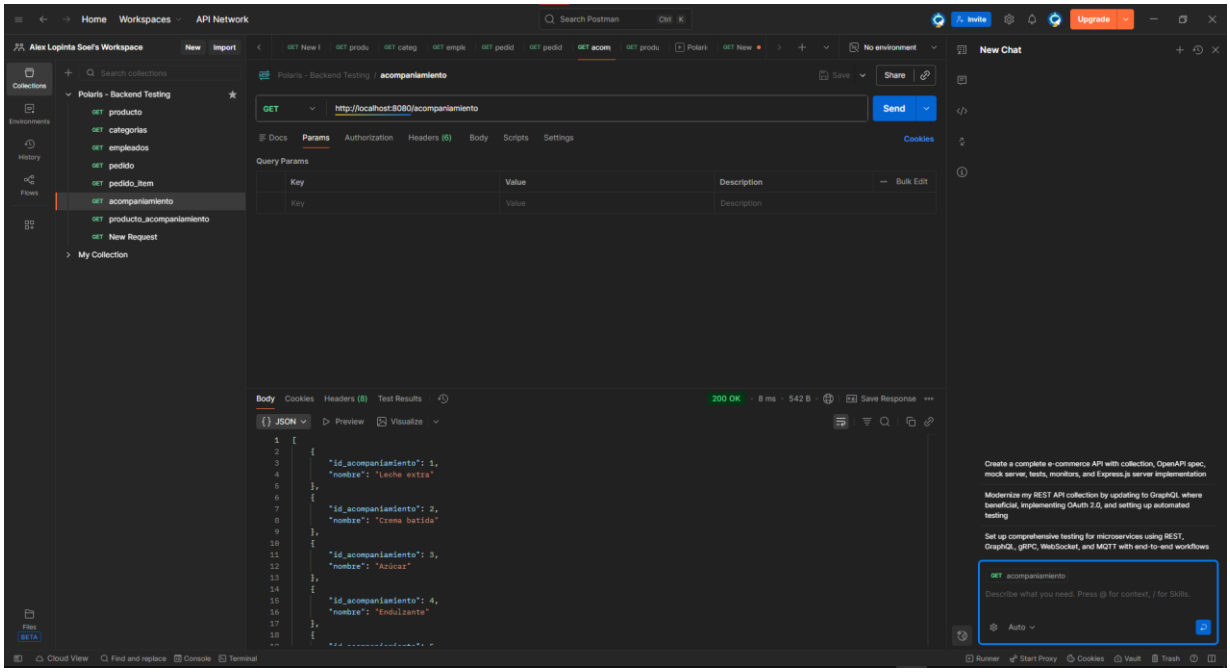
### 9.3 Rendimiento

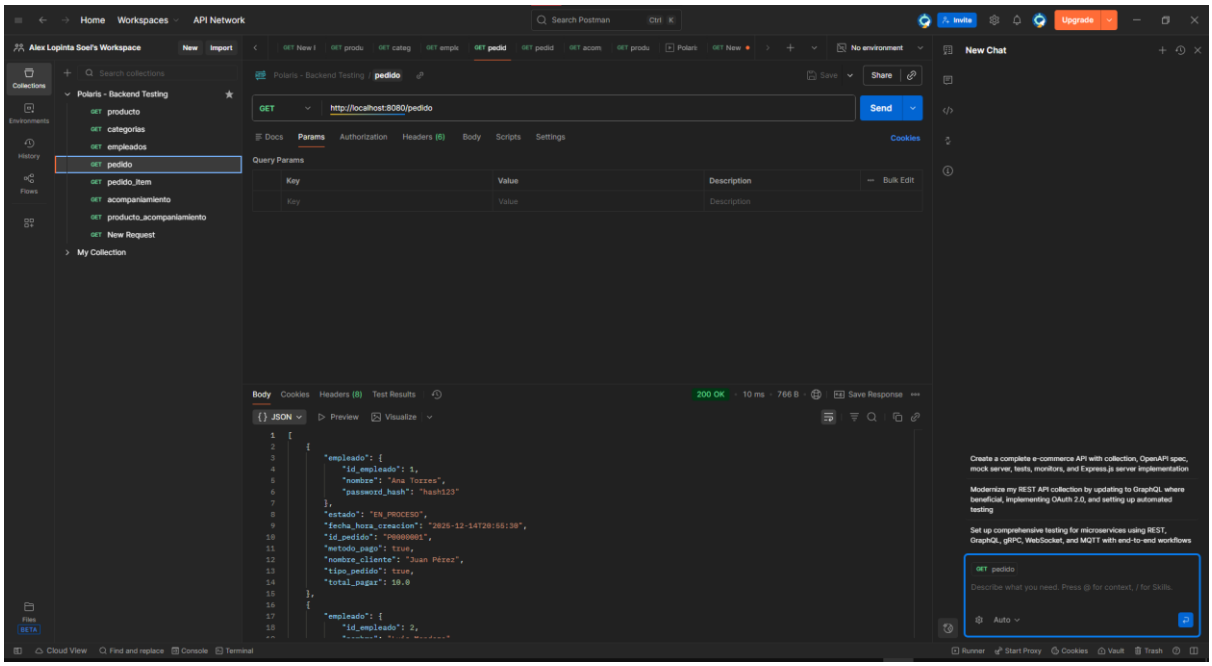
Para futuras versiones del sistema, se recomienda realizar pruebas de rendimiento y carga, con el objetivo de evaluar el comportamiento del sistema ante múltiples solicitudes concurrentes. Estas pruebas permitirían identificar cuellos de botella en la base de datos o en los servicios REST y optimizar el rendimiento general del sistema.

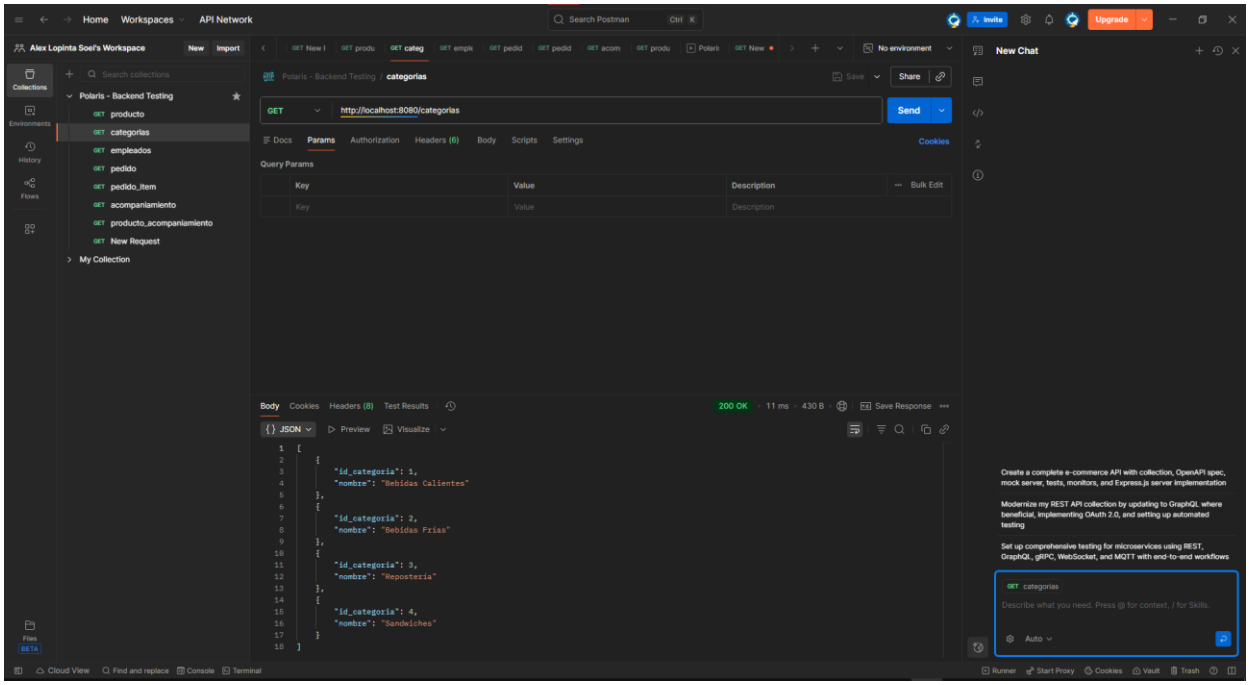
### 9.4 Escalabilidad

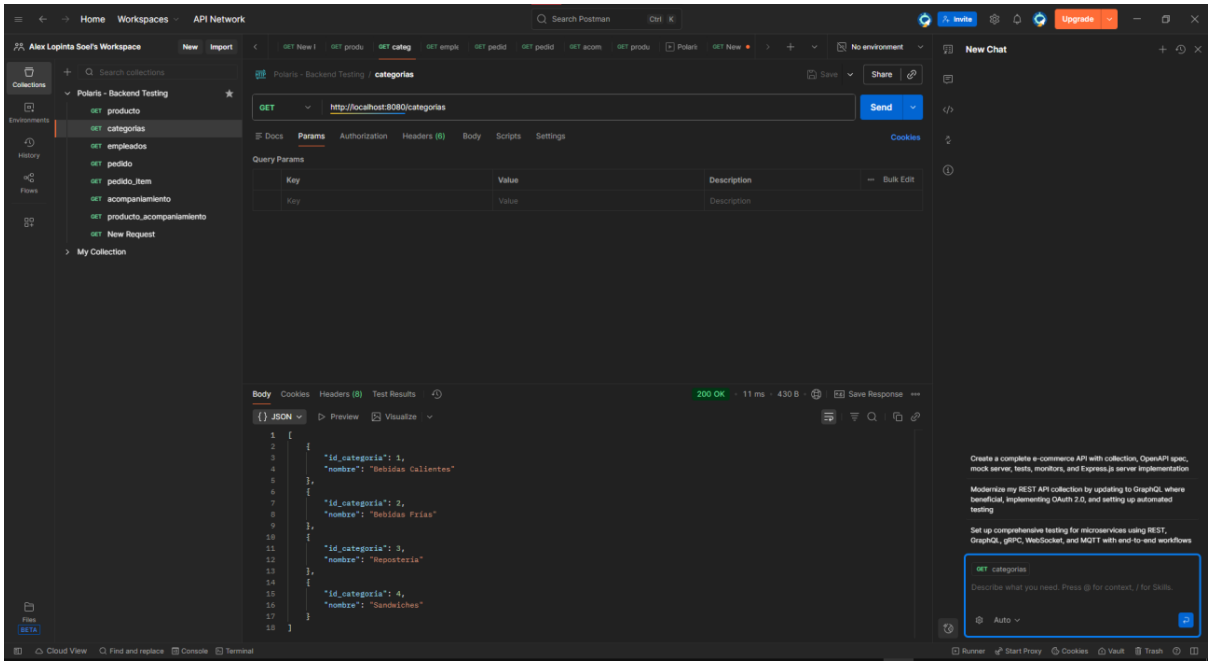
Se sugiere diseñar una estrategia de escalabilidad que permita al sistema crecer de manera ordenada, considerando la separación de responsabilidades por módulos, la optimización de consultas a la base de datos y la posibilidad de desplegar el sistema en entornos más robustos. La aplicación de buenas prácticas de arquitectura facilitará la incorporación de nuevas funcionalidades sin afectar la estabilidad del sistema existente.

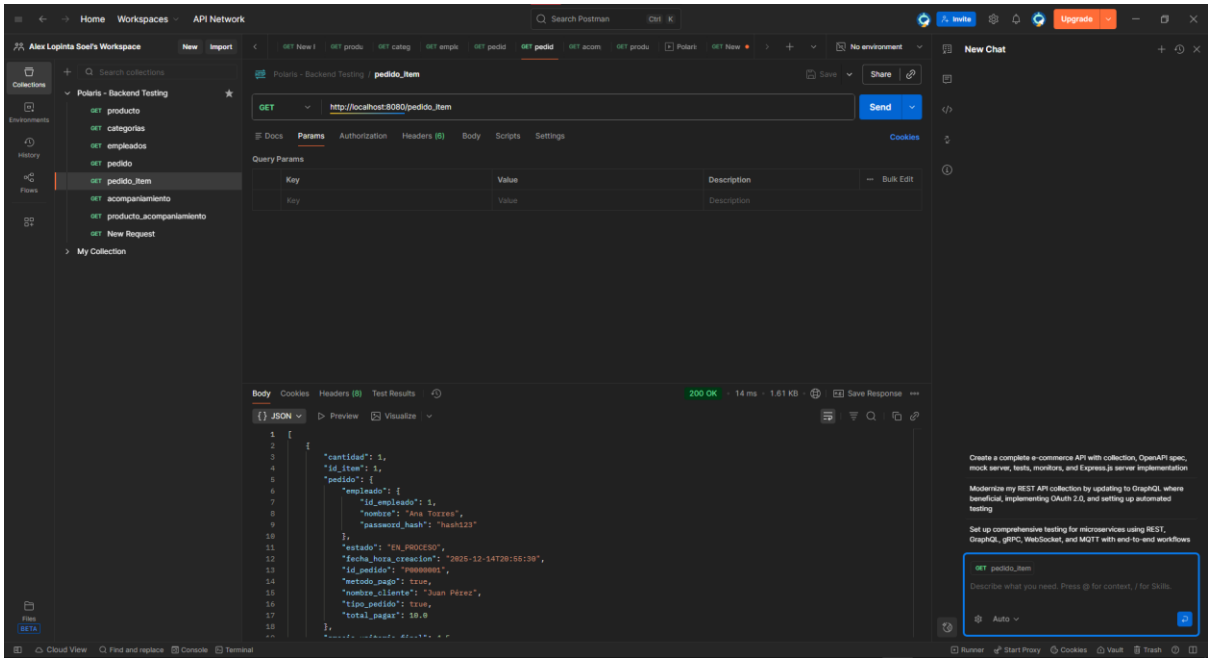
## 10. ANEXOS: Anexo A – Evidencias de Pruebas

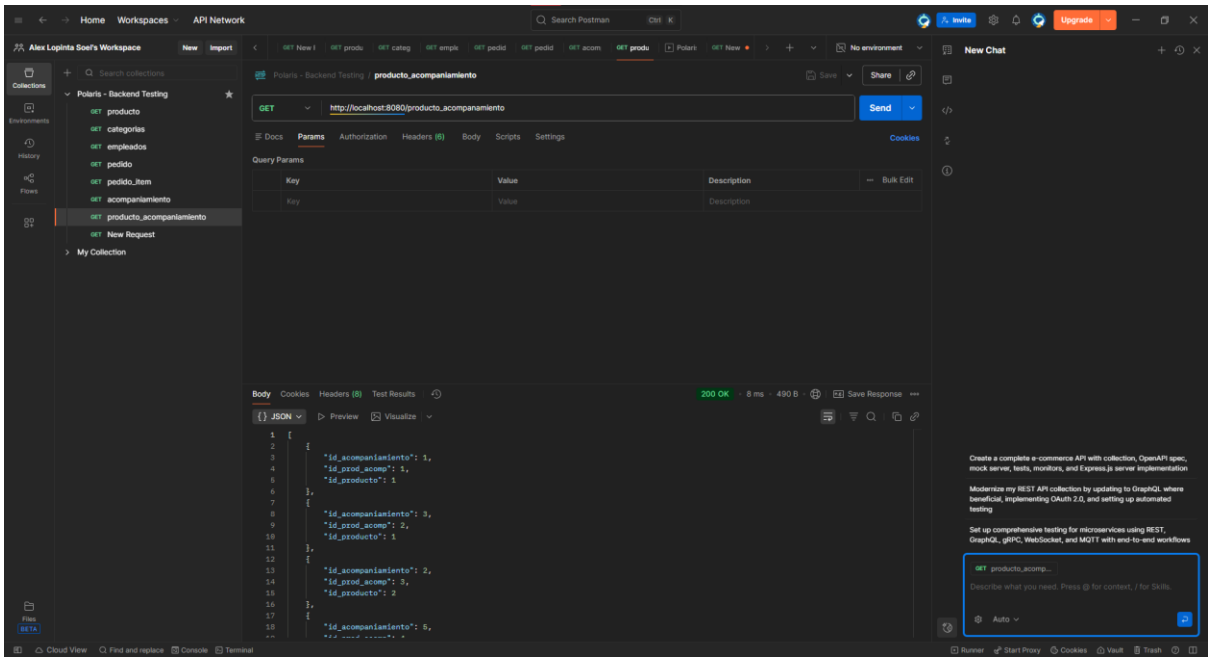


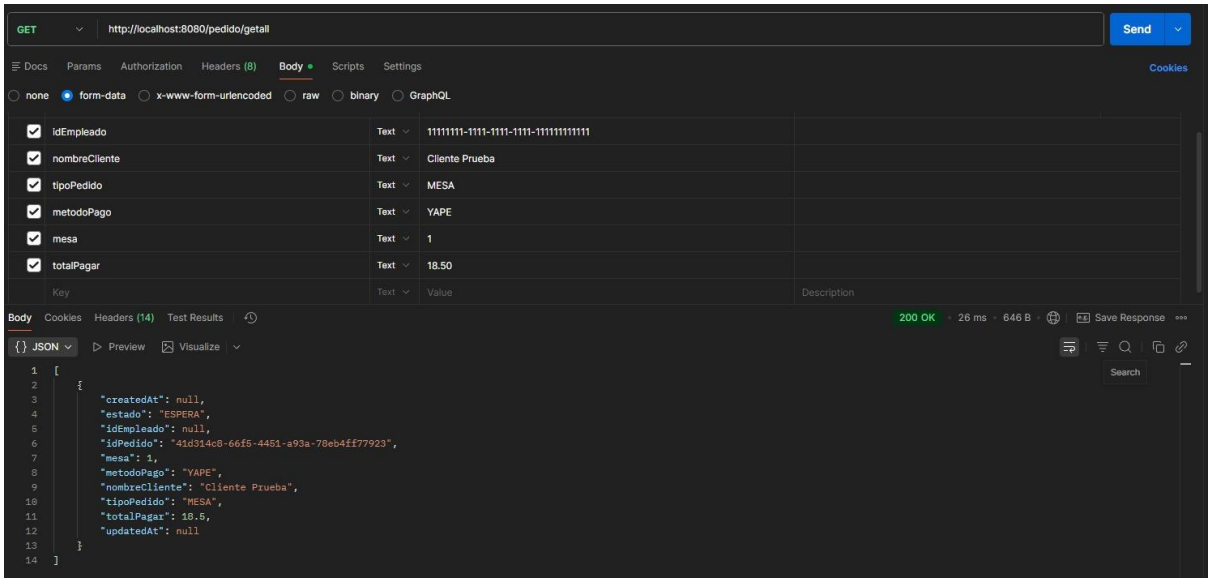



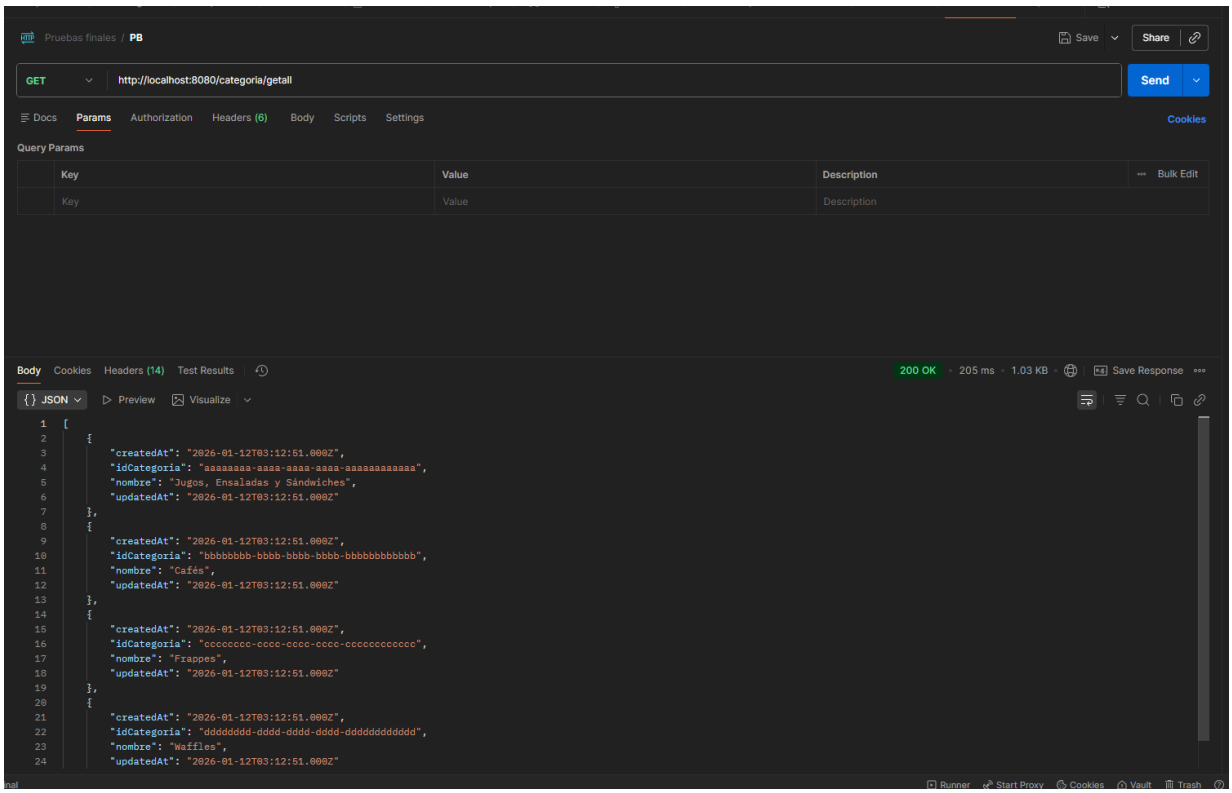

Home
Workspaces
API Network
Search Postman
Ctrl K
Invite
Upgrade
Alex Lopinta Soel's Workspace
New
Import
Collections
Environments
History
Flows
Files
BETA
Search collections
Polaris - Backend Testing
GET producto
GET categorias
GET empleados
GET pedido
GET pedido_item
GET acompanamiento
GET producto_acompaniamiento
GET New Request
My Collection
No environment
Polaris - Backend Testing / categorias
Save
Share
GET
http://localhost:8080/categorias
Send
Docs
Params
Authorization
Headers (6)
Body
Scripts
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Body
Cookies
Headers (8)
Test Results
200 OK
11 ms
430 B
Save Response
{} JSON
Preview
Visualize
[
{
"id_categoria": 1,
"nombre": "Bebidas Calientes"
},
{
"id_categoria": 2,
"nombre": "Bebidas Frias"
},
{
"id_categoria": 3,
"nombre": "Reposteria"
},
{
"id_categoria": 4,
"nombre": "Sandwiches"
}
]
New Chat
Create a complete e-commerce API with collection, OpenAPI spec, mock server, tests, monitors, and Express.js server implementation
Modernize my REST API collection by updating to GraphQL where beneficial, implementing OAuth 2.0, and setting up automated testing
Set up comprehensive testing for microservices using REST, GraphQL, gRPC, WebSocket, and MQTT with end-to-end workflows
GET categorias
Describe what you need. Press @ for context, / for Skills.
Auto
Cloud View
Find and replace
Console
Terminal
Runner
Start Proxy
Cookies
Vault
Trash

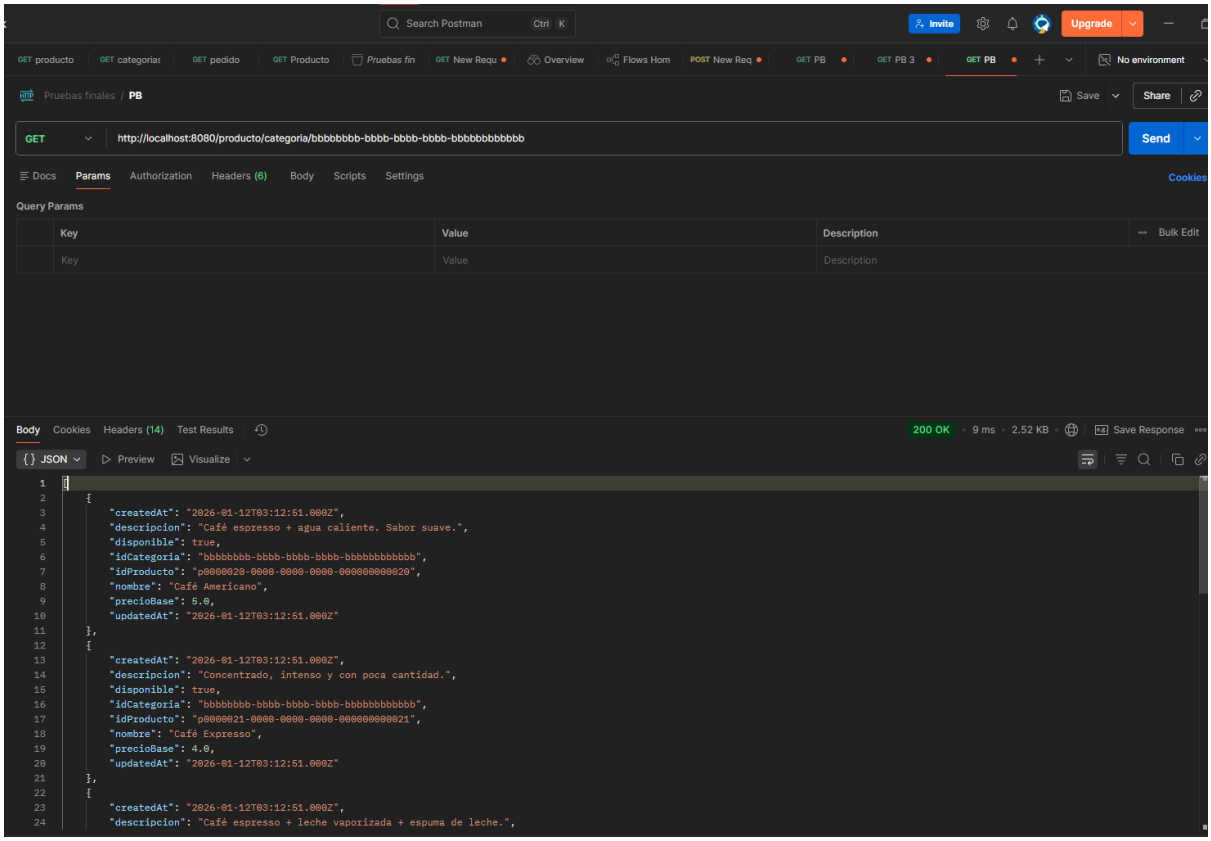

Home
Workspaces
API Network
Search Postman
Ctrl K
Invite
Upgrade
Alex Lopinta Soel's Workspace
New
Import
Collections
Environments
History
Flows
Files
BETA
Search collections
Polaris - Backend Testing
GET producto
GET categorias
GET empleados
GET pedido
GET pedido_item
GET acompanamiento
GET producto_acompaniamiento
GET New Request
My Collection
No environment
Polaris - Backend Testing / categorias
Save
Share
GET
http://localhost:8080/categorias
Send
Docs
Params
Authorization
Headers (6)
Body
Scripts
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Body
Cookies
Headers (8)
Test Results
200 OK
11 ms
430 B
Save Response
{} JSON
Preview
Visualize
[
{
"id_categoria": 1,
"nombre": "Bebidas Calientes"
},
{
"id_categoria": 2,
"nombre": "Bebidas Frias"
},
{
"id_categoria": 3,
"nombre": "Reposteria"
},
{
"id_categoria": 4,
"nombre": "Sandwiches"
}
]
New Chat
Create a complete e-commerce API with collection, OpenAPI spec, mock server, tests, monitors, and Express.js server implementation
Modernize my REST API collection by updating to GraphQL where beneficial, implementing OAuth 2.0, and setting up automated testing
Set up comprehensive testing for microservices using REST, GraphQL, gRPC, WebSocket, and MQTT with end-to-end workflows
GET categorias
Describe what you need. Press @ for context, / for Skills.
Auto
Cloud View
Find and replace
Console
Terminal
Runner
Start Proxy
Cookies
Vault
Trash

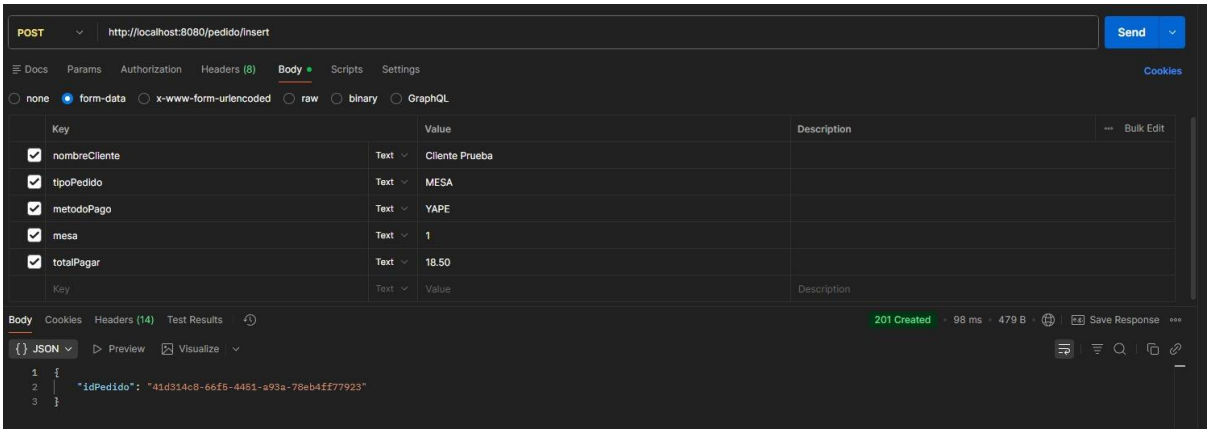

Home
Workspaces
API Network
Search Postman
Ctrl K
Invite
Upgrade
Alex Lopinta Soel's Workspace
New
Import
Collections
Environments
History
Flows
Files
BETA
Polaris - Backend Testing
GET producto
GET categorias
GET empleados
GET pedido
GET pedido_item
GET acompanamiento
GET producto_acompanamiento
GET New Request
My Collection
Polaris - Backend Testing / pedido_item
Save
Share
GET
http://localhost:8080/pedido_item
Send
Docs
Params
Authorization
Headers (6)
Body
Scripts
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Body
Cookies
Headers (8)
Test Results
200 OK
14 ms
1.61 KB
Save Response
JSON
Preview
Visualize
[
{
"cantidad": 1,
"id_item": 1,
"pedido": {
"empleado": {
"id_empleado": 1,
"nombre": "Ana Torres",
"password_hash": "hash123"
},
"estado": "EN_PROCESO",
"fecha_hora_creacion": "2025-12-14T20:55:30",
"id_pedido": "P0000001",
"metodo_pago": true,
"nombre_cliente": "Juan Pérez",
"tipo_pedido": true,
"total_pagar": 10.0
},
New Chat
Create a complete e-commerce API with collection, OpenAPI spec, mock server, tests, monitors, and Express.js server implementation
Modernize my REST API collection by updating to GraphQL where beneficial, implementing OAuth 2.0, and setting up automated testing
Set up comprehensive testing for microservices using REST, GraphQL, gRPC, WebSocket, and MQTT with end-to-end workflows
GET pedido_item
Describe what you need. Press @ for context, / for Skills.
Auto
Cloud View
Find and replace
Console
Terminal
Runner
Start Proxy
Cookies
Vault
Trash

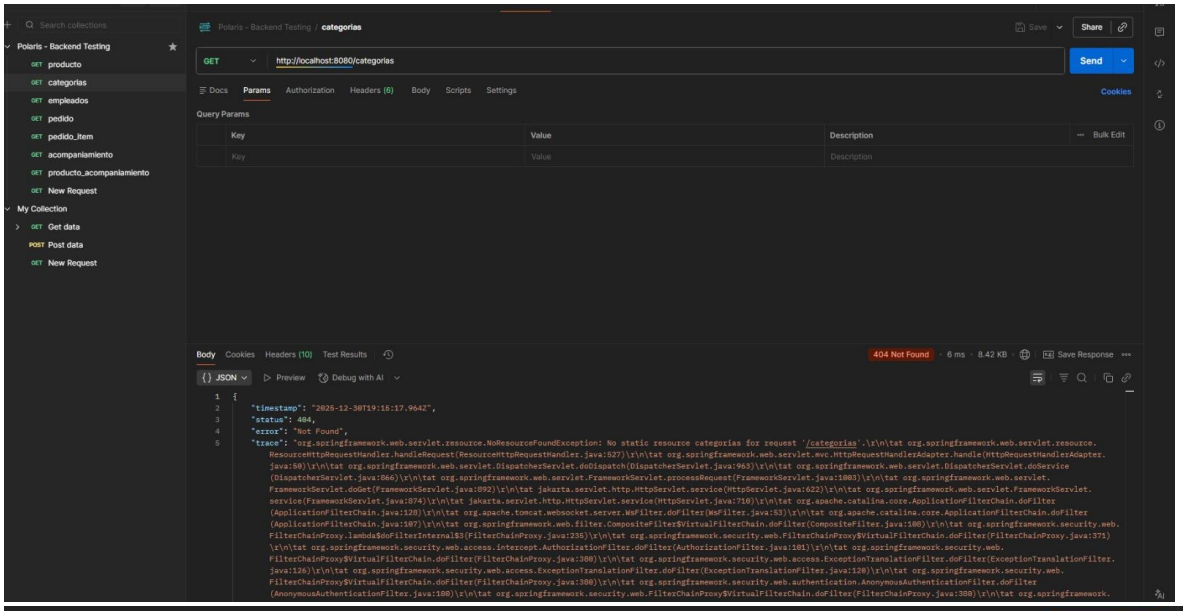

Home
Workspaces
API Network
Search Postman
Ctrl K
Invite
Upgrade
Alex Lopinta Soel's Workspace
New
Import
Collections
Environments
History
Flows
Files
BETA
Polaris - Backend Testing
GET producto
GET categorias
GET empleados
GET pedido
GET pedido_item
GET acompanamiento
GET producto_acompanamiento
GET New Request
My Collection
Polaris - Backend Testing / producto_acompanamiento
Save
Share
GET
http://localhost:8080/producto_acompanamiento
Send
Docs
Params
Authorization
Headers (6)
Body
Scripts
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Body
Cookies
Headers (8)
Test Results
200 OK
8 ms
490 B
Save Response
JSON
Preview
Visualize
[
{
"id_acompanamiento": 1,
"id_prod_acomp": 1,
"id_producto": 1
},
{
"id_acompanamiento": 3,
"id_prod_acomp": 2,
"id_producto": 1
},
{
"id_acompanamiento": 2,
"id_prod_acomp": 3,
"id_producto": 2
},
{
"id_acompanamiento": 5,
New Chat
Create a complete e-commerce API with collection, OpenAPI spec, mock server, tests, monitors, and Express.js server implementation
Modernize my REST API collection by updating to GraphQL where beneficial, implementing OAuth 2.0, and setting up automated testing
Set up comprehensive testing for microservices using REST, GraphQL, gRPC, WebSocket, and MQTT with end-to-end workflows
GET producto_acomp...
Describe what you need. Press @ for context, / for Skills.
Auto
Cloud View
Find and replace
Console
Terminal
Runner
Start Proxy
Cookies
Vault
Trash

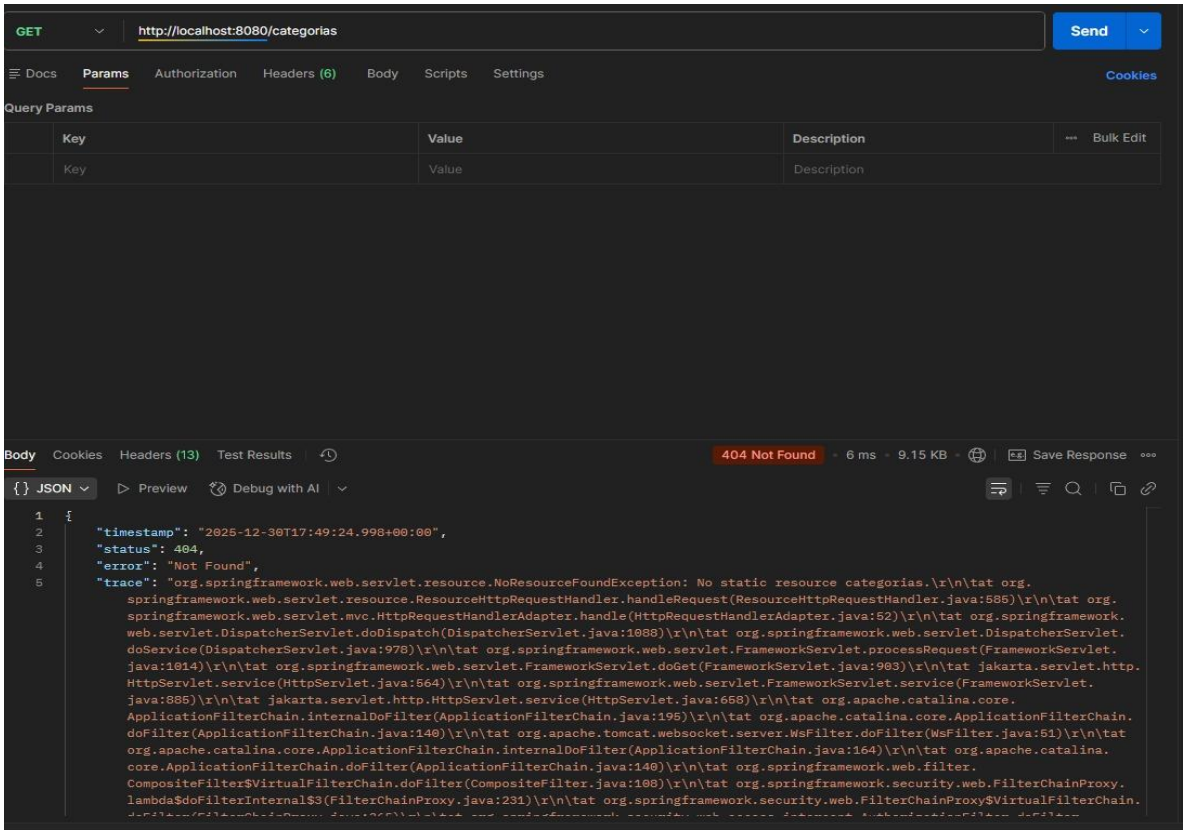

GET
http://localhost:8080/pedido/getall
Send
Docs Params Authorization Headers (8) Body Scripts Settings
Cookies
none form-data x-www-form-urlencoded raw binary GraphQL
idEmpleado Text 11111111-1111-1111-1111-111111111111
nombreCliente Text Cliente Prueba
tipoPedido Text MESA
metodoPago Text YAPE
mesa Text 1
totalPagar Text 18.50
Key Text Value Description
Body Cookies Headers (14) Test Results
200 OK 26 ms 646 B Save Response
JSON Preview Visualize
Search
[
{
"createdAt": null,
"estado": "ESPERA",
"idEmpleado": null,
"idPedido": "41d314c8-66f5-4451-a93a-78eb4ff77923",
"mesa": 1,
"metodoPago": "YAPE",
"nombreCliente": "Cliente Prueba",
"tipoPedido": "MESA",
"totalPagar": 18.5,
"updatedAt": null
}
]

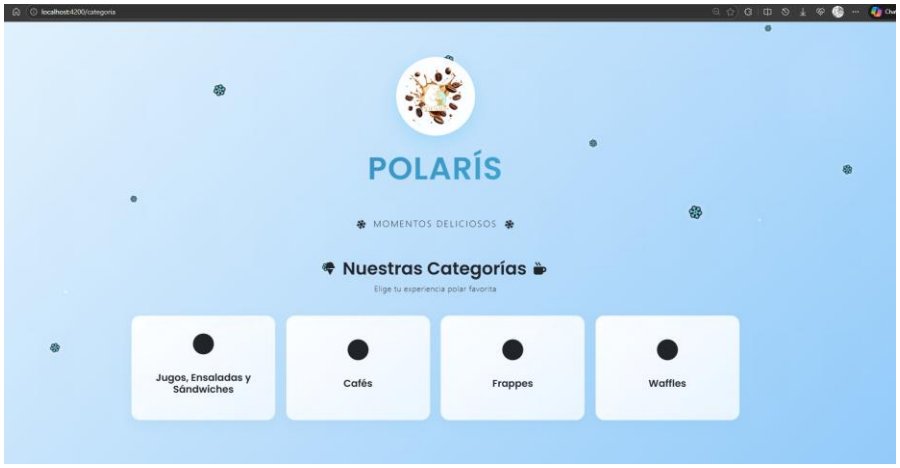

Pruebas finales / PB
Save Share
GET
http://localhost:8080/categoria/getall
Send
Docs Params Authorization Headers (6) Body Scripts Settings
Cookies
Query Params
Key Value Description Bulk Edit
Key Value Description
Body Cookies Headers (14) Test Results
200 OK 205 ms 1.03 KB Save Response
JSON Preview Visualize
[
{
"createdAt": "2026-01-12T03:12:51.000Z",
"idCategoria": "aaaaaaaa-aaaa-aaaa-aaaa-aaaaaaaaaaaa",
"nombre": "Jugos, Ensaladas y Sándwiches",
"updatedAt": "2026-01-12T03:12:51.000Z"
},
{
"createdAt": "2026-01-12T03:12:51.000Z",
"idCategoria": "bbbbbbbb-bbbb-bbbb-bbbb-bbbbbbbbbbbb",
"nombre": "Cafés",
"updatedAt": "2026-01-12T03:12:51.000Z"
},
{
"createdAt": "2026-01-12T03:12:51.000Z",
"idCategoria": "cccccccc-cccc-cccc-cccc-cccccccccccc",
"nombre": "Frappes",
"updatedAt": "2026-01-12T03:12:51.000Z"
},
{
"createdAt": "2026-01-12T03:12:51.000Z",
"idCategoria": "dddddddd-dddd-dddd-dddd-dddddddddddd",
"nombre": "Waffles",
"updatedAt": "2026-01-12T03:12:51.000Z"
Runner Start Proxy Cookies Vault Trash


Search Postman
Ctrl K
Invite
Upgrade
GET producto
GET categorias
GET pedido
GET Producto
Pruebas fin
GET New Requ
Overview
Flows Hom
POST New Req
GET PB
GET PB 3
GET PB
No environment
Pruebas finales / PB
Save
Share
GET
http://localhost:8080/producto/categoria/bbbbbbbb-bbbb-bbbb-bbbb-bbbbbbbbbbbb
Send
Docs
Params
Authorization
Headers (6)
Body
Scripts
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Body
Cookies
Headers (14)
Test Results
200 OK
9 ms
2.52 KB
Save Response
JSON
Preview
Visualize
{
"createdAt": "2026-01-12T03:12:51.000Z",
"descripcion": "Café espresso + agua caliente. Sabor suave.",
"disponible": true,
"idCategoria": "bbbbbbbb-bbbb-bbbb-bbbb-bbbbbbbbbbbb",
"idProducto": "p0000020-0000-0000-0000-000000000020",
"nombre": "Café Americano",
"precioBase": 5.0,
"updatedAt": "2026-01-12T03:12:51.000Z"
},
{
"createdAt": "2026-01-12T03:12:51.000Z",
"descripcion": "Concentrado, intenso y con poca cantidad.",
"disponible": true,
"idCategoria": "bbbbbbbb-bbbb-bbbb-bbbb-bbbbbbbbbbbb",
"idProducto": "p0000021-0000-0000-0000-000000000021",
"nombre": "Café Expresso",
"precioBase": 4.0,
"updatedAt": "2026-01-12T03:12:51.000Z"
},
{
"createdAt": "2026-01-12T03:12:51.000Z",
"descripcion": "Café espresso + leche vaporizada + espuma de leche.",


POST
http://localhost:8080/pedido/insert
Send
Docs
Params
Authorization
Headers (8)
Body
Scripts
Settings
Cookies
none
form-data
x-www-form-urlencoded
raw
binary
GraphQL
Key
Value
Description
Bulk Edit
nombreCliente
Text
Cliente Prueba
tipoPedido
Text
MESA
metodoPago
Text
YAPE
mesa
Text
1
totalPagar
Text
18.50
Body
Cookies
Headers (14)
Test Results
201 Created
98 ms
479 B
Save Response
JSON
Preview
Visualize
{
"idPedido": "41d314c8-66f6-4461-a93a-78eb4ff77923"
}


Polaris - Backend Testing
categorias
Share
GET
http://localhost:8080/categorias
Send
Docs
Params
Authorization
Headers (6)
Body
Scripts
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Body
Cookies
Headers (10)
Test Results
404 Not Found
6 ms
8.42 KB
Save Response
JSON
Preview
Debug with AI
{
"timestamp": "2025-12-30T19:15:17.964Z",
"status": 404,
"error": "Not Found",
"trace": "org.springframework.web.servlet.resource.NoResourceFoundException: No static resource categorias for request '/categorias'.\r\n\tat org.springframework.web.servlet.resource.ResourceHttpRequestHandler.handleRequest(ResourceHttpRequestHandler.java:527)\r\n\tat org.springframework.web.servlet.mvc.HttpRequestHandlerAdapter.handle(HttpRequestHandlerAdapter.java:50)\r\n\tat org.springframework.web.servlet.DispatcherServlet.doDispatch(DispatcherServlet.java:963)\r\n\tat org.springframework.web.servlet.DispatcherServlet.doService(DispatcherServlet.java:866)\r\n\tat org.springframework.web.servlet.FrameworkServlet.processRequest(FrameworkServlet.java:1003)\r\n\tat org.springframework.web.servlet.FrameworkServlet.doGet(FrameworkServlet.java:892)\r\n\tat jakarta.servlet.http.HttpServlet.service(HttpServlet.java:622)\r\n\tat org.springframework.web.servlet.FrameworkServlet.service(FrameworkServlet.java:874)\r\n\tat jakarta.servlet.http.HttpServlet.service(HttpServlet.java:710)\r\n\tat org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:120)\r\n\tat org.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.java:53)\r\n\tat org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:107)\r\n\tat org.springframework.web.filter.CompositeFilter$VirtualFilterChain.doFilter(CompositeFilter.java:108)\r\n\tat org.springframework.security.web.FilterChainProxy.lambda$doFilterInternal$3(FilterChainProxy.java:235)\r\n\tat org.springframework.security.web.FilterChainProxy$VirtualFilterChain.doFilter(FilterChainProxy.java:371)\r\n\tat org.springframework.security.web.access.intercept.AuthorizationFilter.doFilter(AuthorizationFilter.java:101)\r\n\tat org.springframework.security.web.FilterChainProxy$VirtualFilterChain.doFilter(FilterChainProxy.java:380)\r\n\tat org.springframework.security.web.access.ExceptionTranslationFilter.doFilter(ExceptionTranslationFilter.java:126)\r\n\tat org.springframework.security.web.access.ExceptionTranslationFilter.doFilter(ExceptionTranslationFilter.java:120)\r\n\tat org.springframework.security.web.FilterChainProxy$VirtualFilterChain.doFilter(FilterChainProxy.java:380)\r\n\tat org.springframework.security.web.authentication.AnonymousAuthenticationFilter.doFilter(AnonymousAuthenticationFilter.java:100)\r\n\tat org.springframework.security.web.FilterChainProxy$VirtualFilterChain.doFilter(FilterChainProxy.java:380)\r\n\tat org.springframework.


GET
http://localhost:8080/categorias
Send
Docs
Params
Authorization
Headers (6)
Body
Scripts
Settings
Cookies
Query Params
Key
Value
Description
Bulk Edit
Body
Cookies
Headers (13)
Test Results
404 Not Found
6 ms
9.15 KB
Save Response
JSON
Preview
Debug with AI
{
"timestamp": "2025-12-30T17:49:24.998+00:00",
"status": 404,
"error": "Not Found",
"trace": "org.springframework.web.servlet.resource.NoResourceFoundException: No static resource categorias.\r\n\tat org.springframework.web.servlet.resource.ResourceHttpRequestHandler.handleRequest(ResourceHttpRequestHandler.java:586)\r\n\tat org.springframework.web.servlet.mvc.HttpRequestHandlerAdapter.handle(HttpRequestHandlerAdapter.java:52)\r\n\tat org.springframework.web.servlet.DispatcherServlet.doDispatch(DispatcherServlet.java:1088)\r\n\tat org.springframework.web.servlet.DispatcherServlet.doService(DispatcherServlet.java:978)\r\n\tat org.springframework.web.servlet.FrameworkServlet.processRequest(FrameworkServlet.java:1014)\r\n\tat org.springframework.web.servlet.FrameworkServlet.doGet(FrameworkServlet.java:903)\r\n\tat jakarta.servlet.http.HttpServlet.service(HttpServlet.java:564)\r\n\tat org.springframework.web.servlet.FrameworkServlet.service(FrameworkServlet.java:885)\r\n\tat jakarta.servlet.http.HttpServlet.service(HttpServlet.java:658)\r\n\tat org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:195)\r\n\tat org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:140)\r\n\tat org.apache.tomcat.websocket.server.WsFilter.doFilter(WsFilter.java:51)\r\n\tat org.apache.catalina.core.ApplicationFilterChain.internalDoFilter(ApplicationFilterChain.java:164)\r\n\tat org.apache.catalina.core.ApplicationFilterChain.doFilter(ApplicationFilterChain.java:140)\r\n\tat org.springframework.web.filter.CompositeFilter$VirtualFilterChain.doFilter(CompositeFilter.java:108)\r\n\tat org.springframework.security.web.FilterChainProxy.lambda$doFilterInternal$3(FilterChainProxy.java:231)\r\n\tat org.springframework.security.web.FilterChainProxy$VirtualFilterChain.




## Anexo B

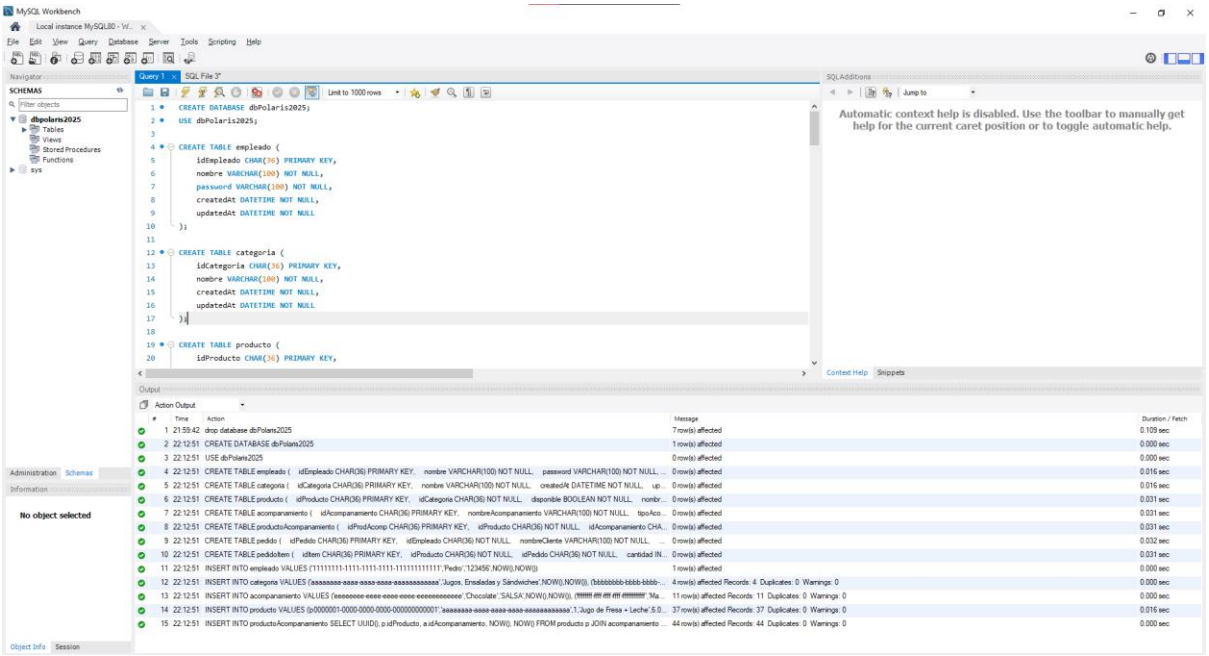
Output

Action Output

| # | Time | Action | Message | Duration / Fetch |
|---|---|---|---|---|
| 1 | 19:55:55 | SET SQL_MODE = "NO_AUTO_VALUE_ON_ZERO" | 0 row(s) affected | 0.000 sec |
| 2 | 19:55:55 | START TRANSACTION | 0 row(s) affected | 0.000 sec |
| 3 | 19:55:55 | SET time_zone = "+00:00" | 0 row(s) affected | 0.000 sec |
| 4 | 19:55:55 | /*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */ | 0 row(s) affected | 0.000 sec |
| 5 | 19:55:55 | /*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */ | 0 row(s) affected | 0.000 sec |
| 6 | 19:55:55 | /*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */ | 0 row(s) affected | 0.000 sec |
| 7 | 19:55:55 | /*!40101 SET NAMES utf8mb4 */ | 0 row(s) affected | 0.000 sec |
| 8 | 19:55:55 | CREATE TABLE `acompanamiento` ( `idAcompanamiento` int(11) NOT NULL, `nombre` varchar(100) NOT NULL, `createdAt` datetime NOT NULL, `u... | Error Code: 1046. No database selected Select the default DB to be used by double-clicking its name in the SCHEMAS list in the sidebar. | 0.000 sec |

| | | | | | |
|---|---|---|---|---|---|
| 28 | 21:39:58 | DROP DATABASE IF EXISTS dbpolaris2025 | 7 row(s) affected | 0.328 sec |
| 29 | 21:39:59 | CREATE DATABASE dbpolaris2025 CHARACTER SET utf8mb4 COLLATE utf8mb4_general... | 1 row(s) affected | 0.016 sec |
| 30 | 21:39:59 | USE dbpolaris2025 | 0 row(s) affected | 0.000 sec |
| 31 | 21:39:59 | SET SQL_MODE='NO_AUTO_VALUE_ON_ZERO' | 0 row(s) affected | 0.000 sec |
| 32 | 21:39:59 | SET time_zone = '+00:00' | 0 row(s) affected | 0.000 sec |
| 33 | 21:39:59 | CREATE TABLE acompanamiento ( idAcompanamiento INT AUTO_INCREMENT PRIMAR... | 0 row(s) affected | 0.031 sec |
| 34 | 21:39:59 | CREATE TABLE categoria ( idCategoria INT AUTO_INCREMENT PRIMARY KEY, nombre... | 0 row(s) affected | 0.031 sec |
| 35 | 21:39:59 | CREATE TABLE empleado ( idEmpleado INT AUTO_INCREMENT PRIMARY KEY, nombr... | 0 row(s) affected | 0.047 sec |
| 36 | 21:39:59 | CREATE TABLE producto ( idProducto INT AUTO_INCREMENT PRIMARY KEY, idCateg... | 0 row(s) affected, 1 warning(s): 1681 Integer display width is deprecated and will be removed i... | 0.032 sec |
| 37 | 21:39:59 | CREATE TABLE pedido ( idPedido CHAR(36) PRIMARY KEY, idEmpleado INT NOT NULL... | 0 row(s) affected, 2 warning(s): 1681 Integer display width is deprecated and will be removed i... | 0.046 sec |
| 38 | 21:39:59 | CREATE TABLE pedidoitem ( idItem INT AUTO_INCREMENT PRIMARY KEY, idProducto ... | 0 row(s) affected | 0.063 sec |
| 39 | 21:39:59 | CREATE TABLE productoacompanamiento ( idProdAcomp INT AUTO_INCREMENT PRIM... | 0 row(s) affected | 0.078 sec |
| 40 | 21:39:59 | INSERT INTO acompanamiento VALUES (1,'Salsa de Chocolate','2025-12-24 19:48:24','202... | 11 row(s) affected Records: 11 Duplicates: 0 Warnings: 0 | 0.000 sec |
| 41 | 21:39:59 | INSERT INTO categoria VALUES (1,'Coffees','2025-12-24 19:43:09','2025-12-24 19:43:09'), (... | 6 row(s) affected Records: 6 Duplicates: 0 Warnings: 0 | 0.000 sec |
| 42 | 21:39:59 | INSERT INTO producto VALUES (1,1,1,'Café Americano',5.00,'Café espresso + agua caliente... | 37 row(s) affected Records: 37 Duplicates: 0 Warnings: 0 | 0.016 sec |
| 43 | 21:39:59 | INSERT INTO productoacompanamiento VALUES (1,32,1,'2025-12-24 20:08:49','2025-12-24... | 17 row(s) affected Records: 17 Duplicates: 0 Warnings: 0 | 0.015 sec |
| 44 | 21:39:59 | INSERT INTO empleado (nombre,passwordHash,createdAt,updatedAt) VALUES ('Juanito','$2... | 1 row(s) affected | 0.000 sec |

| # | Time | Action | Message |
|---|---|---|---|
| 1 | 12:21:16 | DROP DATABASE IF EXISTS dbPolaris2025 | 7 row(s) affected |
| 2 | 12:21:43 | DROP DATABASE IF EXISTS dbPolaris2025 | 0 row(s) affected, 1 warning(s): 1008 Can't drop database 'dbpolaris2025'; database doesn't exist |
| 3 | 12:21:43 | CREATE DATABASE dbPolaris2025 | 1 row(s) affected |
| 4 | 12:21:43 | USE dbPolaris2025 | 0 row(s) affected |
| 5 | 12:21:43 | CREATE TABLE empleado ( idEmpleado CHAR(36) PRIMARY KEY, nombre VARCHAR(100) NOT NULL, passwordHash VARCHAR(255) NOT N... | 0 row(s) affected |
| 6 | 12:21:43 | CREATE TABLE categoria ( idCategoria CHAR(36) PRIMARY KEY, nombre VARCHAR(100) NOT NULL, createdAt DATETIME NOT NULL, up... | 0 row(s) affected |
| 7 | 12:21:43 | CREATE TABLE producto ( idProducto CHAR(36) PRIMARY KEY, idCategoria CHAR(36) NOT NULL, disponible BOOLEAN NOT NULL, nombr... | 0 row(s) affected |
| 8 | 12:21:43 | CREATE TABLE acompanamiento ( idAcompanamiento CHAR(36) PRIMARY KEY, nombre VARCHAR(100) NOT NULL, createdAt DATETIME N... | 0 row(s) affected |
| 9 | 12:21:43 | CREATE TABLE productoAcompanamiento ( idProdAcomp CHAR(36) PRIMARY KEY, idProducto CHAR(36) NOT NULL, idAcompanamiento CHA... | 0 row(s) affected |
| 10 | 12:21:43 | CREATE TABLE pedido ( idPedido CHAR(36) PRIMARY KEY, idEmpleado CHAR(36) NOT NULL, fechaHoraCreacion DATETIME NOT NULL, ... | 0 row(s) affected |
| 11 | 12:21:44 | CREATE TABLE pedidoItem ( idItem CHAR(36) PRIMARY KEY, idProducto CHAR(36) NOT NULL, idPedido CHAR(36) NOT NULL, cantidad IN... | 0 row(s) affected |
| 12 | 12:22:24 | USE dbpolaris2025 | 0 row(s) affected |
| 13 | 12:22:24 | INSERT INTO empleado (nombre, password_hash) VALUES ('Ana Torres', 'hash123'), ('Luis Mendoza', 'hash456'), ('Carla Ramos', 'hash789') | Error Code: 1054. Unknown column 'password_hash' in 'field list' |




## 11. REFERENCIAS

**Ejemplo (puedes usar estas):**

**Pressman, R. S., & Maxim, B. R. (2020).** ***Software engineering: A practitioner's approach*** **(9th ed.). McGraw-Hill Education.**

**Sommerville, I. (2016).** ***Software engineering*** **(10th ed.). Pearson Education.**

**Oracle. (2024).** ***Java platform documentation*****. https://docs.oracle.com**

**Spring. (2024).** ***Spring Boot reference documentation*****. https://spring.io/projects/spring-boot**

**Angular Team. (2024).** ***Angular documentation*****. https://angular.io/docs**