



UNIVERSIDAD DE LAS FUERZAS ARMADAS

APLICACIONES DISTRIBUIDAS

DEPARTAMENTO DE CIENCIAS DE LA COMPUTACIÓN

Actividad en clase de API'S

Estudiantes:

Erick Mijail Andrade Pichoasamin

Docente:

Ing. Dario Morales

Nrc: 2553

Informe del Proyecto: Gestión de Pedidos con Spring Boot y MySQL	2
Introducción	2
Objetivos	2
Objetivo General	3
Objetivos Específicos	3
Alcance del Proyecto	3
Especificaciones Técnicas	4
Tecnologías Utilizadas	4
Arquitectura del Sistema	4
Pruebas y Validación	4
Arquitectura del Sistema	5
Componentes Principales	5
1. Controlador (Controller)	5
2. Servicio (Service)	5
3. Repositorio (Repository)	5
4. Modelo de Datos (Model)	6
5. Base de Datos	6
Flujo de Datos en el Sistema	6
Tecnologías Soportadas por la Arquitectura	7
Implementación del Sistema	7
1. Configuración del Entorno de Desarrollo	7
2. Desarrollo del Microservicio	7
2.1. Diseño de la Entidad Pedido	7
2.2. Implementación de la Lógica de Negocio	8
2.3. Exposición de Endpoints REST	8
3. Pruebas del Sistema	9
3.1. Pruebas Funcionales con Postman	9
3.2. Pruebas de Base de Datos con MySQL Workbench	11
Despliegue	12
Proceso de Contenedorización	12
Ejecución del Sistema	12
Resultados y Conclusiones	13
Posibles Mejoras	13
Anexos	13
Bibliografía	13

Informe del Proyecto: Gestión de Pedidos con Spring Boot y MySQL

Introducción

El desarrollo de aplicaciones modernas basadas en microservicios se ha convertido en una práctica esencial para garantizar escalabilidad, modularidad y mantenibilidad en los sistemas de software. En este contexto, el proyecto titulado "Gestión de Pedidos utilizando Spring Boot, MySQL y Docker" se centra en la creación de un microservicio robusto y eficiente que permite gestionar las operaciones básicas relacionadas con la entidad "Pedido". Este sistema, diseñado con principios de arquitectura limpia y buenas prácticas de desarrollo, está orientado a facilitar la administración de pedidos en un entorno empresarial.

El objetivo principal del proyecto es implementar una solución que aborde las necesidades comunes de un sistema de gestión de pedidos, incluyendo la creación, consulta, actualización y eliminación de registros. Para ello, se utilizan tecnologías de vanguardia como Spring Boot, un marco que simplifica el desarrollo de aplicaciones Java empresariales, y MySQL, una base de datos relacional ampliamente utilizada en sistemas de producción. Además, el proyecto incorpora la tecnología de contenedores con Docker, lo que garantiza portabilidad, facilidad de despliegue y un entorno controlado para la ejecución del sistema.

El proyecto se estructura alrededor de una API RESTful diseñada para interactuar con la entidad "Pedido", que incluye atributos como el identificador único, el número de pedido, el cliente, la fecha del pedido y el total asociado. Estas propiedades están cuidadosamente validadas a través de anotaciones en el modelo de datos, asegurando la integridad y consistencia de la información almacenada en la base de datos. Por ejemplo, el número de pedido sigue un formato específico, y los valores numéricos como el total están restringidos a rangos lógicos para evitar errores.

El desarrollo del sistema incluye diversas capas que trabajan de manera coordinada para ofrecer un flujo de datos confiable y eficiente. La capa de control gestiona las solicitudes entrantes y delega la lógica de negocio a servicios especializados, que a su vez interactúan con el repositorio de datos para realizar las operaciones necesarias. Este enfoque modular asegura que el código sea fácil de mantener y escalar en el futuro.

La infraestructura del proyecto se configura mediante un archivo Docker Compose, que automatiza la creación de contenedores tanto para la base de datos MySQL como para la aplicación Spring Boot. Esto permite a los desarrolladores desplegar el sistema completo en cualquier entorno con soporte para Docker, eliminando problemas de configuración y compatibilidad.

Objetivos

Objetivo General

Desarrollar un microservicio eficiente y escalable para la gestión de pedidos utilizando Spring Boot, MySQL y Docker, que permita realizar operaciones CRUD (Crear, Leer, Actualizar y Eliminar) sobre la entidad "Pedido" en un entorno controlado y confiable.

Objetivos Específicos

1. **Diseñar e implementar una API RESTful** que facilite la interacción con el sistema, garantizando una arquitectura modular y desacoplada.
2. **Configurar y utilizar Docker** para la contenedorización y despliegue de los servicios, asegurando portabilidad y facilidad de integración en cualquier entorno.
3. **Implementar una base de datos relacional en MySQL**, asegurando la persistencia y consistencia de los datos relacionados con los pedidos.
4. **Realizar validaciones estrictas de datos** en el modelo "Pedido", asegurando la calidad y fiabilidad de la información almacenada.
5. **Realizar pruebas funcionales con Postman**, verificando el correcto funcionamiento de las operaciones de la API.
6. **Utilizar IntelliJ IDEA como entorno de desarrollo**, aprovechando sus herramientas avanzadas para depuración, generación de código y productividad.
7. **Monitorear y gestionar la base de datos con MySQL Workbench**, asegurando integridad y optimización de las consultas SQL.

Alcance del Proyecto

El proyecto tiene un alcance bien definido y está limitado a los siguientes aspectos:

- **Gestión de la entidad "Pedido"**: Implementación de un sistema que permita registrar, consultar, actualizar y eliminar pedidos en una base de datos.
- **Validaciones específicas**: Los datos ingresados a través de la API deben cumplir con reglas de negocio, como formatos predefinidos (por ejemplo, el número de pedido debe seguir el patrón PED-XXX).
- **Pruebas funcionales**: Uso de Postman para probar las operaciones de la API y validar el manejo de errores, asegurando la robustez del sistema.
- **Despliegue con Docker**: Automatización del despliegue de la base de datos MySQL y el microservicio en contenedores para simplificar la ejecución del proyecto en diferentes entornos.
- **Persistencia de datos**: Implementación de una base de datos relacional que permita almacenar y gestionar información de pedidos de forma segura y eficiente.

Limitaciones del alcance:

- El proyecto no incluye integración con sistemas externos ni módulos avanzados como reportes o analítica.
- No se implementan medidas de seguridad avanzadas, como autenticación o autorización, ya que el enfoque principal está en la funcionalidad básica del sistema.

Especificaciones Técnicas

Tecnologías Utilizadas

- **Lenguaje de programación:** Java 17.
- **Framework principal:** Spring Boot.
- **Base de datos:** MySQL 8.0, gestionada con MySQL Workbench para consultas y administración.
- **Contenedorización:** Docker y Docker Compose para gestionar la base de datos y la aplicación.
- **Entorno de desarrollo:** IntelliJ IDEA para escribir y depurar el código.
- **Pruebas:** Postman para probar las rutas de la API y validar el correcto funcionamiento de las operaciones.

Arquitectura del Sistema

- **Modelo de datos:**
 - Entidad **Pedido**: incluye los campos **id**, **numeroPedido**, **cliente**, **fechaPedido** y **total**.
 - Validaciones con anotaciones como **@NotNull**, **@NotBlank**, y expresiones regulares para formatos específicos.
- **Capas del sistema:**
 - **Controlador**: Maneja las solicitudes HTTP entrantes y delega las operaciones al servicio correspondiente.
 - **Servicio**: Implementa la lógica de negocio y orquesta las operaciones con la base de datos.
 - **Repositorio**: Interactúa directamente con la base de datos mediante el uso de Spring Data JPA.
- **Configuración de la base de datos:**
 - Conexión mediante JDBC: **jdbc:mysql://localhost:3306/sp**.
 - Credenciales: Usuario **root**, contraseña **123456789**.
 - Configuración en el archivo **application.properties**.

Pruebas y Validación

- **Postman:**
 - Se realizaron pruebas en los endpoints:
 - **POST /api/pedidos** para crear nuevos pedidos.
 - **GET /api/pedidos** y **GET /api/pedidos/{id}** para consultar pedidos.
 - **PUT /api/pedidos/{id}** para actualizar registros.
 - **DELETE /api/pedidos/{id}** para eliminar pedidos.
 - Validación de respuestas y manejo de errores (códigos de estado como 400, 404 y 201).
- **MySQL Workbench:**

- Consulta y verificación de datos almacenados en la base de datos para confirmar la integridad y consistencia tras las operaciones.
- **Docker:**
 - Se ejecutaron contenedores para la base de datos y la aplicación, asegurando que ambos servicios interactuaran correctamente.

Arquitectura del Sistema

El sistema se diseñó siguiendo un enfoque de arquitectura multicapa, asegurando la separación de responsabilidades y facilitando el mantenimiento y la escalabilidad. A continuación, se describen las principales capas y componentes:

Componentes Principales

1. Controlador (Controller)

- **Ubicación:** `src/main/java/com/pedido/controllers`
- **Responsabilidad:**
 - Manejar las solicitudes HTTP entrantes (endpoints RESTful).
 - Validar las solicitudes utilizando las anotaciones de validación proporcionadas por Jakarta Validation.
 - Delegar la lógica de negocio al servicio correspondiente.
 - Formatear las respuestas HTTP (códigos de estado y datos).
- **Ejemplo de endpoints implementados:**
 - `POST /api/pedidos`: Crear un nuevo pedido.
 - `GET /api/pedidos`: Listar todos los pedidos.
 - `GET /api/pedidos/{id}`: Buscar un pedido por ID.
 - `PUT /api/pedidos/{id}`: Actualizar un pedido existente.
 - `DELETE /api/pedidos/{id}`: Eliminar un pedido por ID.

2. Servicio (Service)

- **Ubicación:** `src/main/java/com/pedido/services`
- **Responsabilidad:**
 - Implementar la lógica de negocio del sistema.
 - Interactuar con el repositorio para realizar operaciones de persistencia.
 - Garantizar la correcta manipulación de los datos antes de pasarlos a la capa de repositorio.
- **Clase principal:**
 - `PedidoServiceImpl`: Implementa la interfaz `PedidoService` con métodos como `findAll()`, `findById(Long id)`, `save(Pedido pedido)` y `deleteById(Long id)`.

3. Repositorio (Repository)

- **Ubicación:** `src/main/java/com/pedido/repositories`

- **Responsabilidad:**
 - Manejar las operaciones de persistencia mediante Spring Data JPA.
 - Interactuar directamente con la base de datos MySQL.
- **Clase principal:**
 - `PedidoRepository`: Extiende `CrudRepository<Pedido, Long>`, proporcionando métodos CRUD básicos sin necesidad de código adicional.

4. Modelo de Datos (Model)

- **Ubicación:** `src/main/java/com/pedido/models/entities`
- **Responsabilidad:**
 - Representar las entidades que serán gestionadas en la base de datos.
 - Incluir validaciones a nivel de modelo para garantizar la consistencia de los datos.
- **Clase principal:**
 - `Pedido`: Define los atributos del pedido (`id`, `numeroPedido`, `cliente`, `fechaPedido`, `total`) y contiene validaciones con anotaciones como `@NotNull`, `@NotBlank`, `@Pattern`, entre otras.

5. Base de Datos

- **Tecnología utilizada:** MySQL 8.0.
- **Gestión de conexión:**
 - Configurada en el archivo `application.properties` con parámetros como URL, usuario, contraseña y controlador JDBC.
 - La base de datos se inicializa y gestiona automáticamente gracias a Hibernate (`spring.jpa.hibernate.ddl-auto=update`).
- **Persistencia:** La entidad `Pedido` está mapeada a la tabla `pedido` en la base de datos, con las siguientes columnas:
 - `id`: Clave primaria generada automáticamente.
 - `numero_pedido`: Único, obligatorio, con formato validado (e.g., `PED-123`).
 - `cliente`: Obligatorio, admite solo letras y espacios (hasta 50 caracteres).
 - `fecha_pedido`: Obligatorio, almacena la fecha de creación del pedido.
 - `total`: Obligatorio, con valores entre 1 y 999,999,999.

Flujo de Datos en el Sistema

1. **Solicitud Entrante:**
 - Un cliente realiza una solicitud HTTP (por ejemplo, un `POST` para crear un pedido) hacia un endpoint proporcionado por el controlador.
2. **Validación y Delegación:**
 - El controlador valida la solicitud usando las anotaciones en el modelo `Pedido`.
 - Si los datos son válidos, delega la operación al servicio.
3. **Lógica de Negocio:**
 - El servicio implementa la lógica de negocio correspondiente.

- Realiza llamadas al repositorio para interactuar con la base de datos.
- 4. **Persistencia de Datos:**
 - El repositorio utiliza JPA para ejecutar las operaciones de CRUD sobre la base de datos MySQL.
- 5. **Respuesta:**
 - El controlador devuelve una respuesta HTTP adecuada (por ejemplo, código **201 Created** con el objeto creado).

Tecnologías Soportadas por la Arquitectura

- **Spring Boot:** Framework principal para el desarrollo del microservicio.
- **Spring Data JPA:** Abstracción para interactuar con la base de datos utilizando repositorios.
- **MySQL:** Motor de base de datos relacional para la persistencia.
- **Docker:** Plataforma para la contenedorización y despliegue del sistema.
- **Postman:** Herramienta para realizar pruebas funcionales de los endpoints.
- **Hibernate:** Marco de trabajo ORM para gestionar la persistencia de objetos.

Implementación del Sistema

La implementación del sistema se llevó a cabo utilizando un enfoque estructurado, incorporando las mejores prácticas en el desarrollo de microservicios. A continuación, se describen los pasos clave y las herramientas utilizadas:

1. Configuración del Entorno de Desarrollo

- **IDE Utilizado:** IntelliJ IDEA, elegido por su integración nativa con proyectos Spring Boot y herramientas como Docker y Git.
- **Gestión de Dependencias:**
 - **Herramienta:** Maven, configurada en el archivo **pom.xml**.
 - **Dependencias principales:**
 - Spring Boot Starter Web.
 - Spring Boot Starter Data JPA.
 - Driver JDBC para MySQL.
 - Jakarta Validation.
- **Base de Datos:**
 - **Motor:** MySQL 8.0.
 - **Cliente Utilizado:** MySQL Workbench para administrar y verificar las tablas y datos almacenados.

2. Desarrollo del Microservicio

2.1. Diseño de la Entidad **Pedido**

- Se creó la clase **Pedido** en el paquete **models.entities** para representar los datos de un pedido.

- La clase está mapeada a la tabla `pedido` en la base de datos mediante anotaciones de JPA:
 - `@Entity` y `@Table` definen la estructura de la tabla.
 - Validaciones con anotaciones como `@NotBlank`, `@Pattern`, `@Min`, y `@Max` garantizan la integridad de los datos antes de almacenarlos.

2.2. Implementación de la Lógica de Negocio

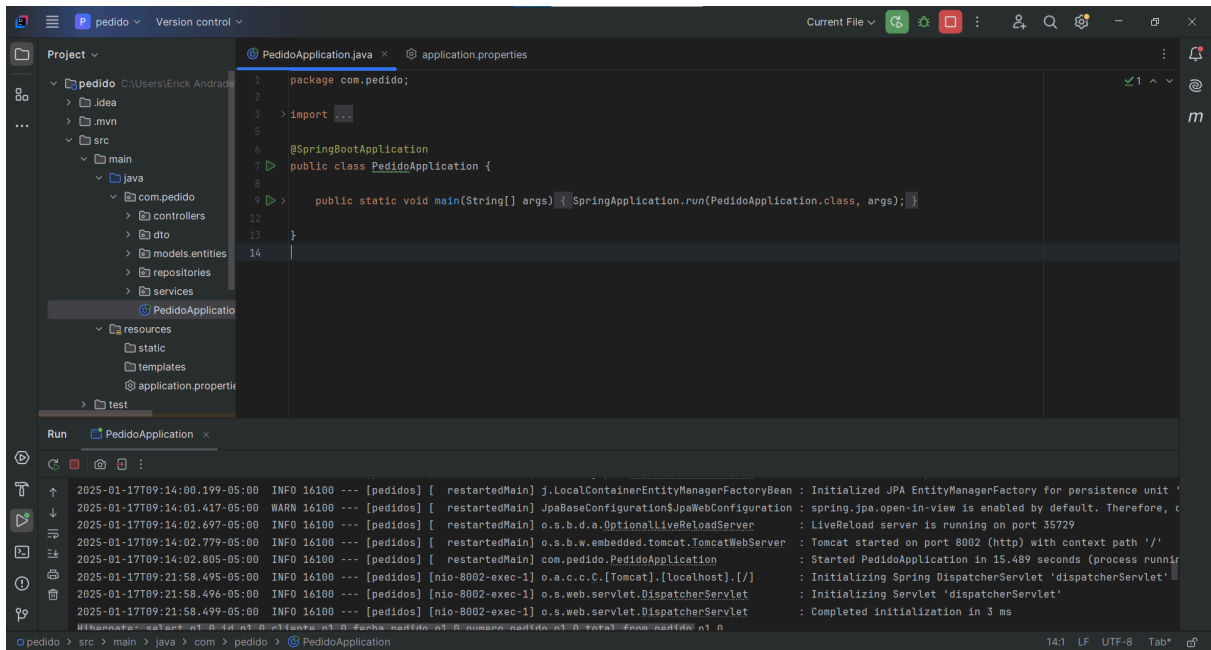
- La lógica de negocio reside en la capa de servicio (`PedidoServiceImpl`).
- Se implementaron métodos clave como:
 - `findAll()`: Retorna todos los pedidos almacenados.
 - `findById(Long id)`: Busca un pedido por su identificador único.
 - `save(Pedido pedido)`: Persiste un nuevo pedido o actualiza uno existente.
 - `deleteById(Long id)`: Elimina un pedido por su ID.

2.3. Exposición de Endpoints REST

- La clase `PedidoController` maneja las solicitudes entrantes a través de endpoints RESTful.
- Cada endpoint está asociado a una operación específica (CRUD).
- Validaciones de las solicitudes con `@Valid` y manejo de errores mediante la clase `ErrorResponse`.
- Ejemplo de endpoint para crear un pedido:

`@PostMapping`

```
public ResponseEntity<?> crear(@Valid @RequestBody Pedido pedido,
BindingResult result) {
    if (result.hasErrors()) {
        return validar(result);
    }
    return
    ResponseEntity.status(HttpStatus.CREATED).body(service.save(pedido))
;
}
```

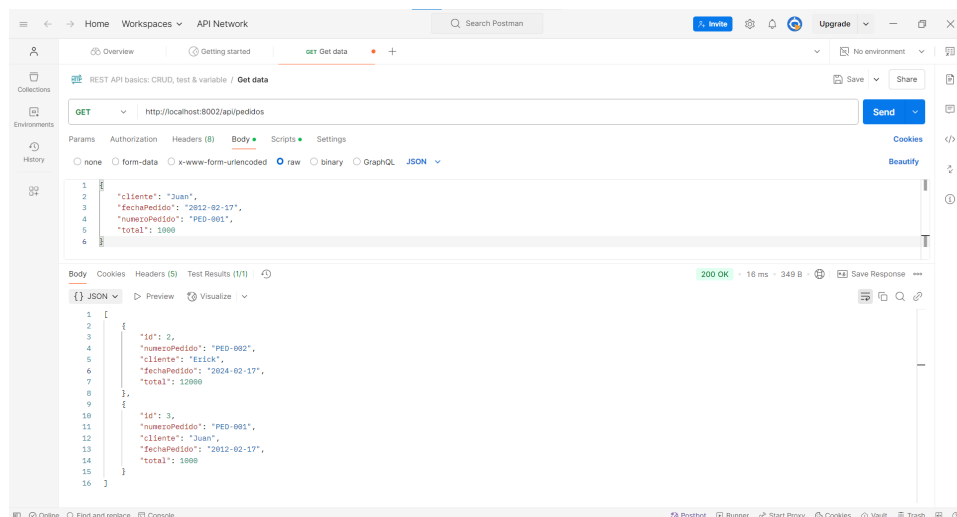


3. Pruebas del Sistema

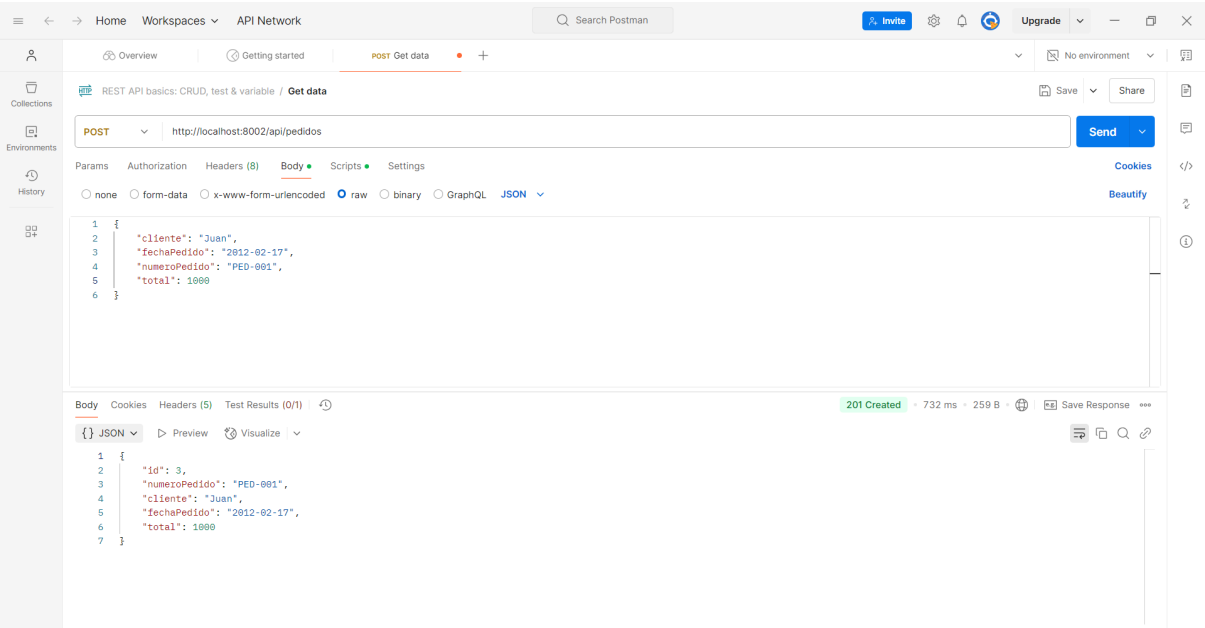
3.1. Pruebas Funcionales con Postman

- Se diseñaron y ejecutaron pruebas para validar el correcto funcionamiento de los endpoints.
- Las pruebas incluyeron:
 - Creación de pedidos (POST `/api/pedidos`).
 - Listado de todos los pedidos (GET `/api/pedidos`).
 - Búsqueda de un pedido por ID (GET `/api/pedidos/{id}`).
 - Actualización de un pedido (PUT `/api/pedidos/{id}`).
 - Eliminación de un pedido (DELETE `/api/pedidos/{id}`).
- Ejemplo de solicitud POST en Postman:

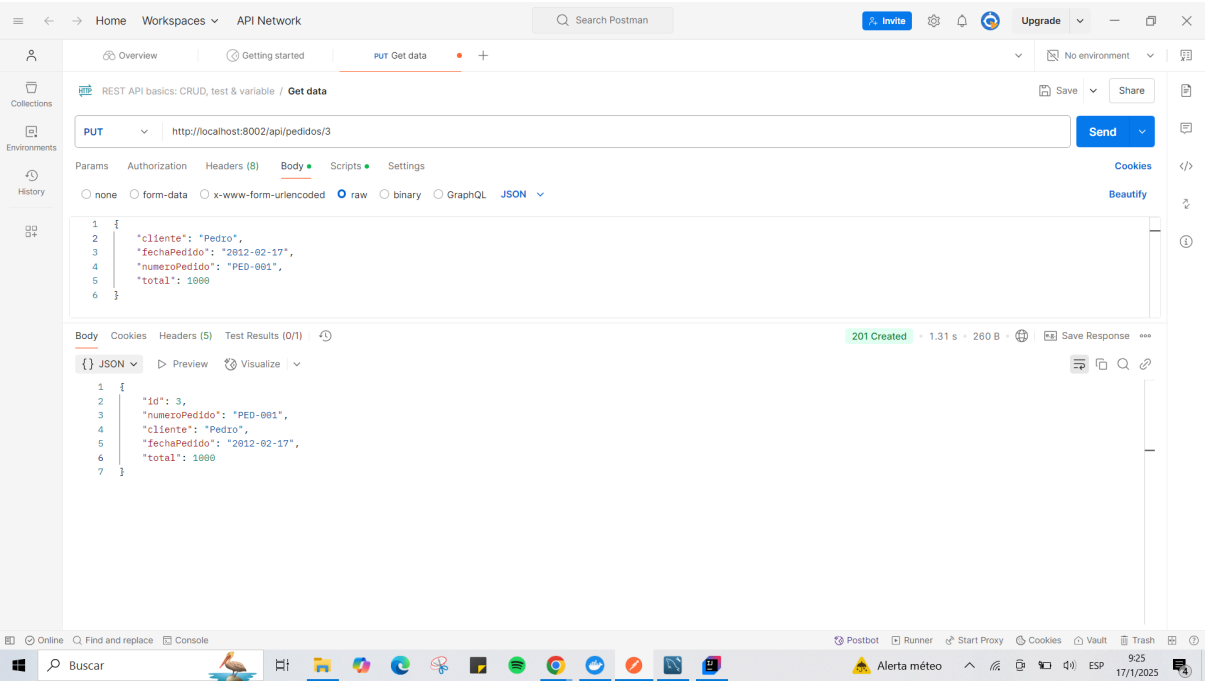
Método Get



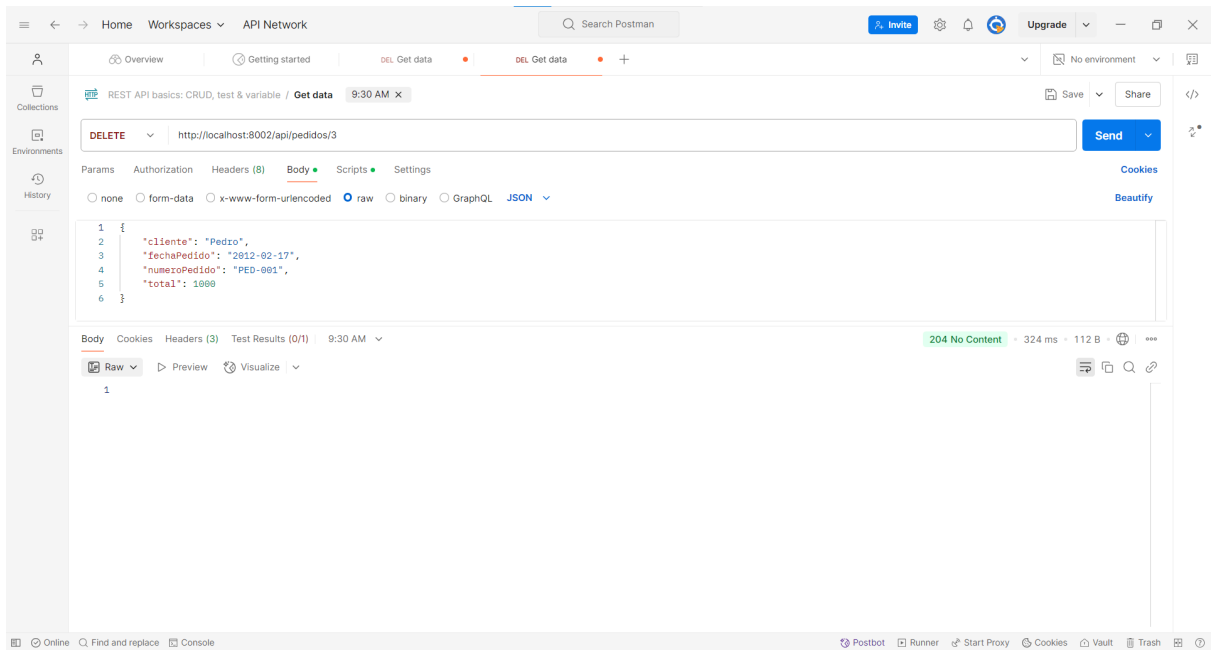
Método Post



Método Put

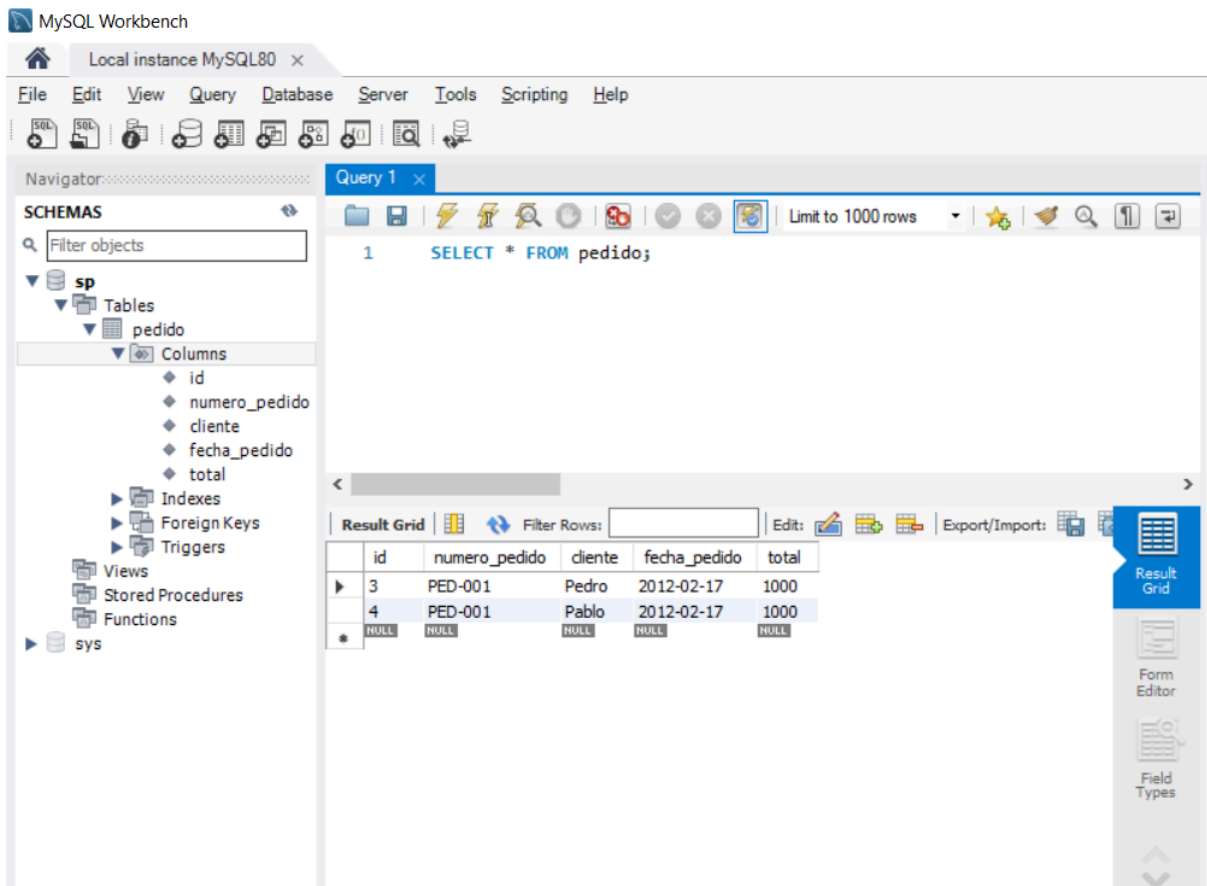


Método Delete



3.2. Pruebas de Base de Datos con MySQL Workbench

- Se verificaron los datos almacenados en la tabla `pedido` tras cada operación.
- Se comprobaron los esquemas, restricciones y relaciones de la base de datos.



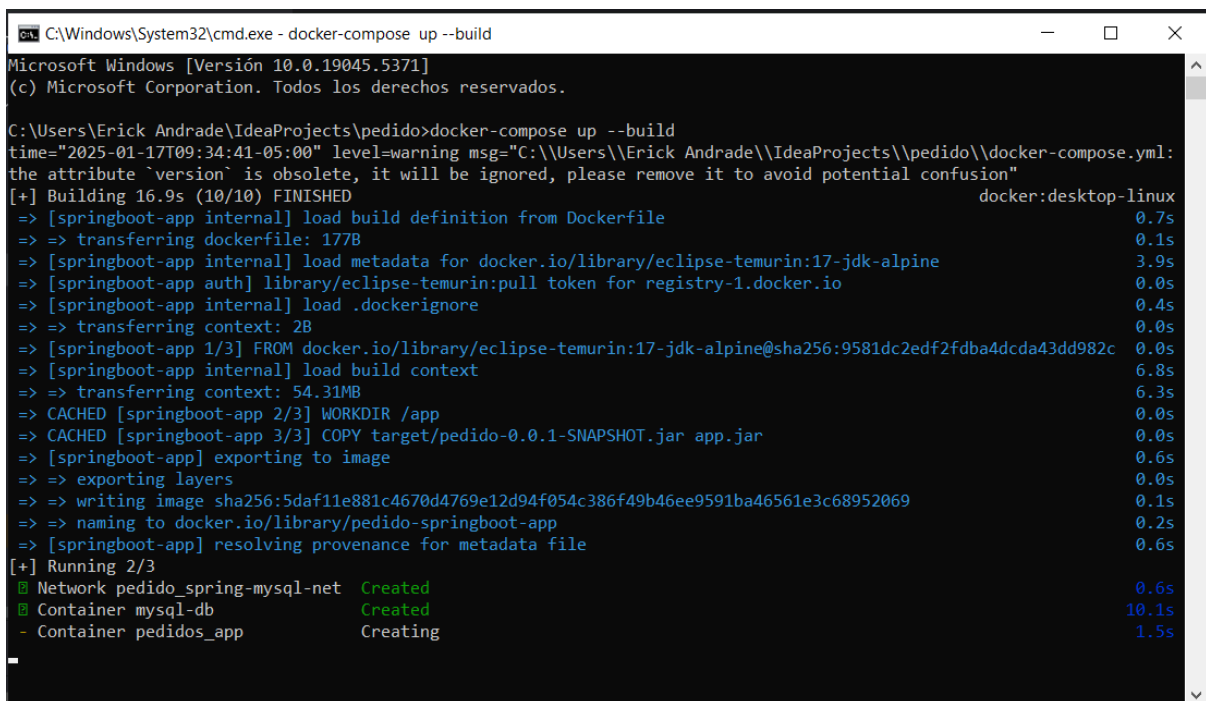
Despliegue

Proceso de Contenedorización

1. **MySQL:** Configurado en el archivo `docker-compose.yml` para ejecutarse en el puerto `3307` del host.
2. **Spring Boot:** La aplicación fue contenedorizada mediante un Dockerfile que utiliza una imagen base de Java 17.
3. **Red de Contenedores:** Los servicios se comunican mediante una red bridge personalizada llamada `spring-mysql-net`.

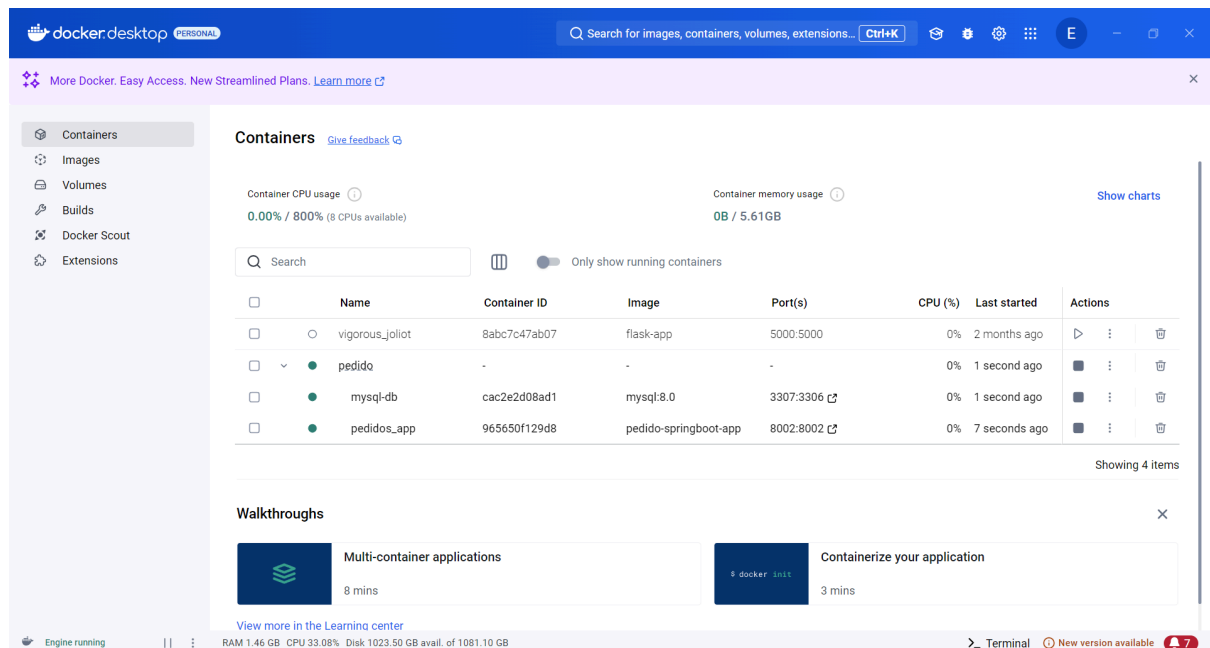
Ejecución del Sistema

1. Construir la imagen de Spring Boot:
`docker build -t pedido_app .`
2. Levantar los servicios:
`docker-compose up`
3. Verificar el estado:
`docker ps`



```
C:\Windows\System32\cmd.exe - docker-compose up --build
Microsoft Windows [Versión 10.0.19045.5371]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\Erick Andrade\IdeaProjects\pedido>docker-compose up --build
time="2025-01-17T09:34:41-05:00" level=warning msg="C:\\Users\\Erick Andrade\\IdeaProjects\\pedido\\docker-compose.yml:
the attribute `version` is obsolete, it will be ignored, please remove it to avoid potential confusion"
[+] Building 16.9s (10/10) FINISHED
=> [springboot-app internal] load build definition from Dockerfile                                docker:desktop-linux 0.7s
=> => transferring dockerfile: 177B                                                            0.1s
=> [springboot-app internal] load metadata for docker.io/library/eclipse-temurin:17-jdk-alpine    3.9s
=> [springboot-app auth] library/eclipse-temurin:pull token for registry-1.docker.io             0.0s
=> [springboot-app internal] load .dockerignore                                                0.4s
=> => transferring context: 2B                                                                0.0s
=> [springboot-app 1/3] FROM docker.io/library/eclipse-temurin:17-jdk-alpine@sha256:9581dc2edf2fdb4dcda43dd982c 0.0s
=> [springboot-app internal] load build context                                              6.8s
=> => transferring context: 54.31MB                                                            6.3s
=> CACHED [springboot-app 2/3] WORKDIR /app                                                  0.0s
=> CACHED [springboot-app 3/3] COPY target/pedido-0.0.1-SNAPSHOT.jar app.jar                 0.0s
=> [springboot-app] exporting to image                                                       0.6s
=> => exporting layers                                                                        0.0s
=> => writing image sha256:5daf11e881c4670d4769e12d94f054c386f49b46ee9591ba46561e3c68952069    0.1s
=> => naming to docker.io/library/pedido-springboot-app                                     0.2s
=> [springboot-app] resolving provenance for metadata file                                   0.6s
[+] Running 2/3
  Network pedido_spring-mysql-net    Created                                0.6s
  Container mysql-db                 Created                             10.1s
  Container pedidos_app              Creating                            1.5s
```



Resultados y Conclusiones

El proyecto demostró ser una solución efectiva para la gestión de pedidos, proporcionando una API robusta y funcional. La integración de Docker permite un despliegue sencillo y eficiente, mientras que el uso de Spring Boot y JPA asegura un rendimiento óptimo.

Posibles Mejoras

1. Implementar autenticación y autorización mediante Spring Security.
2. Integrar un sistema de cacheo para mejorar el rendimiento.
3. Ampliar el sistema con métricas de monitoreo usando herramientas como Prometheus y Grafana.

Anexos

<https://github.com/ericktibu07/Distribuidas.git>

Bibliografía

1. Oracle Corporation. (2025). *MySQL 8.0 Reference Manual*. Recuperado de <https://dev.mysql.com/doc/>
2. Docker, Inc. (2025). *Docker Documentation*. Recuperado de <https://docs.docker.com/>
3. Eclipse Foundation. (2025). *Eclipse Temurin Documentation*. Recuperado de <https://adoptium.net/docs/>
4. Postman. (2025). *Postman API Platform Documentation*. Recuperado de <https://learning.postman.com/>
5. JetBrains. (2025). *IntelliJ IDEA Documentation*. Recuperado de <https://www.jetbrains.com/idea/documentation/>

6. Pivotal Software, Inc. (2025). *Spring Framework Documentation*. Recuperado de <https://spring.io/projects/spring-boot>
7. Jakarta EE. (2025). *Jakarta Validation Specification 3.0*. Recuperado de <https://jakarta.ee/specifications/validation/>
8. International Organization for Standardization. (2020). *ISO/IEC 19510: BPMN 2.0*. Ginebra: ISO.
9. OpenAPI Initiative. (2025). *OpenAPI Specification 3.1.0*. Recuperado de <https://swagger.io/specification/>
10. Asociación Americana de Psicología. (2020). *Normas de publicación de la APA: Manual de Publicación de la American Psychological Association* (7.^a ed.). Washington, D.C.: Autor.