



Instituto Politécnico Nacional
Escuela Superior de Computo
Ingeniería en Sistemas Computacionales.



Unidad de Aprendizaje:
Sistemas Distribuidos.

Nombre del Profesor:
Carreto Arellano Chadwick.

Nombre del Alumno:
Cruz Cubas Ricardo

Grupo:
7CM1.

Practica 7:
Microservicios.

Contenido

Introducción	3
Planteamiento del Problema	6
Propuesta de Solución.....	7
Materiales y Métodos Empleados.....	9
Materiales	9
Métodos Empleados	9
Desarrollo.....	11
Creación del Proyecto para el Microservicio de Usuarios (User-Service)	11
Creación del Proyecto para el Microservicio de Productos (Product-Service).....	12
Resultados	16
Conclusión	19

Introducción

Los microservicios son un enfoque arquitectónico que implica dividir una aplicación grande y compleja en una serie de servicios pequeños e independientes. Cada servicio realiza una función o tarea específica y se ejecuta de forma autónoma. Estos servicios se comunican entre sí a través de interfaces bien definidas, como APIs RESTful o mensajes.

Características principales de los microservicios:

1. Desacoplamiento:

- Cada microservicio es independiente y puede ser desarrollado, desplegado y escalado de manera aislada. No depende directamente de otros servicios para su funcionamiento, lo que facilita el mantenimiento y la evolución de la aplicación.

2. Escalabilidad:

- Puedes escalar cada microservicio de manera independiente. Si un servicio requiere más recursos debido a una mayor carga, solo necesitarás escalar ese servicio específico sin afectar a otros.

3. Desarrollo autónomo:

- Equipos pequeños y autónomos pueden trabajar en diferentes microservicios sin interferir entre sí. Esto permite un desarrollo más ágil y rápido, especialmente en equipos grandes.

4. Diversidad tecnológica:

- En una arquitectura de microservicios, cada servicio puede usar la tecnología que mejor se adapte a su propósito. Por ejemplo, un servicio puede estar escrito en Java, mientras que otro puede estar en Python o Node.js. Esta flexibilidad hace que puedas elegir las mejores herramientas para cada necesidad.

5. Resiliencia:

- Los microservicios permiten que si un servicio falla, no afecte a toda la aplicación. Como los servicios son independientes, el fallo de uno no tiene que causar el fallo de los demás.

6. Despliegue independiente:

- Cada microservicio se puede desplegar de manera independiente. Esto reduce el riesgo de fallos durante el

despliegue de la aplicación completa y permite hacer actualizaciones de manera más frecuente.

Ventajas de los Microservicios:

1. Escalabilidad y flexibilidad:

- Como cada servicio es independiente, puedes escalar solo los servicios que realmente lo necesiten. Esto mejora la eficiencia de los recursos y facilita la adaptación a cambios en la demanda.

2. Desarrollo ágil y rápido:

- Los equipos de desarrollo pueden trabajar en paralelo en diferentes microservicios, lo que acelera el proceso de desarrollo y permite una implementación más rápida de nuevas características.

3. Fácil mantenimiento y actualización:

- Los microservicios permiten realizar cambios en un servicio sin afectar al resto de la aplicación. Esto facilita la corrección de errores, la actualización de versiones o la implementación de nuevas funcionalidades sin interrumpir el servicio completo.

4. Resiliencia y tolerancia a fallos:

- Si un microservicio falla, los demás pueden seguir funcionando, lo que aumenta la fiabilidad del sistema. Además, puedes implementar patrones como circuit breaker para gestionar fallos y evitar que se propaguen.

Desventajas de los Microservicios:

1. Complejidad en la gestión:

- A medida que el número de microservicios crece, también lo hace la complejidad en términos de gestión, monitoreo, y mantenimiento. Requiere herramientas avanzadas para la gestión de contenedores, orquestación y monitoreo de servicios.

2. Comunicación entre servicios:

- Los microservicios deben comunicarse entre sí, generalmente a través de APIs RESTful o mensajería. Esto puede generar latencia y complicaciones si no se maneja correctamente.

3. Distribución de datos:

- En una arquitectura de microservicios, cada servicio tiene su propia base de datos o almacenamiento. Esto puede llevar a problemas de consistencia y dificultad en las transacciones distribuidas.

4. Desarrollo y pruebas más complejas:

- Dado que los servicios son independientes, las pruebas deben considerar la interacción entre microservicios. Esto implica más esfuerzo en la creación de pruebas y simulaciones para servicios dependientes.



Planteamiento del Problema

En la actualidad, las aplicaciones grandes y complejas requieren una arquitectura que permita una gestión eficiente, escalable y flexible. La arquitectura de microservicios es un enfoque que aborda estos desafíos, dividiendo una aplicación en servicios más pequeños e independientes que pueden ser gestionados y desplegados de manera autónoma. Esta arquitectura es especialmente útil en aplicaciones que requieren un alto grado de escalabilidad y disponibilidad.

En este contexto, se plantea el desarrollo de un sistema utilizando microservicios, con el objetivo de gestionar dos áreas fundamentales de una aplicación: usuarios y productos. Estos microservicios estarán diseñados para exponer un conjunto de APIs RESTful que permitan a los usuarios consultar los recursos de manera sencilla y eficiente.

El sistema estará compuesto por dos microservicios principales:

1. User-Service: Encargado de gestionar la información de los usuarios de la aplicación.
2. Product-Service: Encargado de gestionar la información de los productos disponibles en la aplicación.

Ambos servicios se implementarán de manera independiente, con su propio ciclo de vida, y se comunicarán a través de solicitudes HTTP. El desarrollo de los microservicios utilizará Spring Boot, una herramienta que facilita la creación de aplicaciones Java basadas en microservicios, y que permite exponer fácilmente los servicios a través de endpoints HTTP.

Objetivo Específico:

Desarrollar e implementar dos microservicios básicos que expongan los siguientes endpoints:

- User-Service: Proporcionar un endpoint para consultar la lista de usuarios de la aplicación.
 - Endpoint: GET /users
 - Respuesta esperada: "Lista de usuarios"
- Product-Service: Proporcionar un endpoint para consultar la lista de productos disponibles.
 - Endpoint: GET /products
 - Respuesta esperada: "Lista de productos"

Propuesta de Solución

Para abordar el desafío de crear un sistema escalable y mantenible basado en microservicios, se propone el siguiente enfoque utilizando la arquitectura de microservicios con Spring Boot. La solución se centrará en la creación de dos microservicios independientes que gestionarán diferentes recursos de la aplicación: usuarios y productos. Cada uno de estos servicios será responsable de una funcionalidad específica y se desplegará de forma independiente, permitiendo una gestión más eficiente y escalable.

Microservicio de Usuarios (User-Service)

El User-Service será responsable de gestionar la información de los usuarios de la aplicación. Este microservicio expondrá un endpoint HTTP que permitirá consultar la lista de usuarios. En esta etapa inicial, la respuesta será una cadena de texto estática, pero en el futuro podrá ser extendida para incluir acceso a bases de datos y lógica de negocio compleja.

- Tecnologías a utilizar:
 - Spring Boot: Para crear el microservicio de manera rápida y eficiente.
 - RESTful API: Para exponer el servicio mediante un endpoint HTTP.
 - JUnit: Para realizar pruebas unitarias y asegurar la calidad del código.
- Endpoint:
 - GET /users: Devuelve una lista de usuarios (en esta fase inicial, simplemente devuelve un mensaje de texto estático).

Microservicio de Productos (Product-Service)

El Product-Service será el microservicio encargado de gestionar la información sobre los productos de la aplicación. Al igual que el servicio de usuarios, este microservicio expondrá un endpoint que permitirá consultar los productos disponibles.

- Tecnologías a utilizar:
 - Spring Boot: Para el desarrollo del microservicio.
 - RESTful API: Para exponer el servicio mediante un endpoint HTTP.

- JUnit: Para realizar pruebas unitarias y verificar el funcionamiento del microservicio.
- Endpoint:
 - GET /products: Devuelve una lista de productos (también devolverá un mensaje de texto estático en esta fase inicial).

Despliegue Independiente de los Microservicios

Ambos microservicios serán desplegados en diferentes puertos para asegurar que operen de forma independiente. El User-Service se desplegará en el puerto 8081, mientras que el Product-Service se desplegará en el puerto 8082. Esto garantizará que no haya conflictos entre los servicios y permitirá escalarlos de forma independiente si es necesario.

Pruebas de Integración

Para probar que ambos microservicios funcionan correctamente, se utilizará Postman, una herramienta de pruebas de APIs, para enviar solicitudes HTTP a los endpoints de cada microservicio:

- Prueba del microservicio de usuarios: Enviar una solicitud GET a `http://localhost:8081/users` y verificar que la respuesta sea "Lista de usuarios".
- Prueba del microservicio de productos: Enviar una solicitud GET a `http://localhost:8082/products` y verificar que la respuesta sea "Lista de productos".

Escalabilidad y Mantenimiento

Una de las ventajas principales de la arquitectura de microservicios es la escalabilidad. Cada microservicio puede ser escalado de manera independiente según las necesidades del sistema. Por ejemplo, si el servicio de usuarios recibe una alta carga de tráfico, solo este servicio podrá ser escalado sin afectar al servicio de productos.

Además, al mantener cada servicio de forma independiente, se facilita el mantenimiento y la evolución de la aplicación. Si en el futuro se desea agregar más funcionalidades, como la integración con bases de datos o la adición de características avanzadas (autenticación, autorización, etc.), esto podrá ser realizado de manera modular, sin afectar el funcionamiento de otros servicios.

Materiales y Métodos Empleados

Materiales

1. Spring Boot:
 - Spring Boot es un framework que facilita la creación de aplicaciones basadas en microservicios. Proporciona un conjunto de herramientas y dependencias preconfiguradas para crear aplicaciones web y RESTful rápidamente.
2. Java 17:
 - La versión de Java seleccionada para este proyecto es Java 17, una de las versiones LTS (Long Term Support) que asegura estabilidad y soporte a largo plazo, lo que es adecuado para aplicaciones de producción.
3. Maven:
 - Maven es una herramienta de automatización de compilación y gestión de dependencias utilizada para administrar el ciclo de vida del proyecto y las bibliotecas necesarias.
4. Visual Studio Code:
 - Editor de código ligero que, con la ayuda de extensiones, facilita el desarrollo en Java y Spring Boot, proporcionando soporte para edición de código, depuración y pruebas.
5. Postman:
 - Herramienta de pruebas de API que permite realizar solicitudes HTTP a los microservicios y verificar que los endpoints respondan correctamente.
6. JDK (Java Development Kit) 17:
 - Kit de desarrollo que proporciona todas las herramientas necesarias para compilar y ejecutar aplicaciones Java. Se utiliza para compilar el código fuente y ejecutar los microservicios.
7. Git:
 - Sistema de control de versiones utilizado para gestionar el código fuente y mantener un historial de los cambios realizados durante el desarrollo del proyecto.

Métodos Empleados

1. Creación del Proyecto:
 - Se creó un proyecto de microservicios utilizando Spring Boot a través de Spring Initializr. Este enfoque permite generar un proyecto preconfigurado con las dependencias necesarias para trabajar con Spring Boot.
2. Desarrollo de Microservicios:
 - Se desarrollaron dos microservicios:

- User-Service: Para gestionar usuarios y exponer un endpoint /users.
 - Product-Service: Para gestionar productos y exponer un endpoint /products.
 - Ambos servicios fueron creados como aplicaciones independientes con sus propios controladores REST que devuelven mensajes simples en formato de texto.
3. Configuración de la Aplicación:
- Se configuró el archivo application.properties para establecer el nombre de la aplicación (spring.application.name) y los puertos de escucha para cada servicio (8081 para el servicio de usuarios y 8082 para el servicio de productos).
4. Despliegue y Ejecución:
- Los servicios fueron ejecutados de forma independiente en diferentes puertos, usando la herramienta Maven para compilar y ejecutar los microservicios.
 - Los microservicios fueron desplegados localmente para pruebas, usando los comandos mvn spring-boot:run para levantar cada uno de los servicios.
5. Pruebas con Postman:
- Postman fue utilizado para realizar pruebas sobre los endpoints /users y /products, enviando solicitudes HTTP GET a las direcciones locales http://localhost:8081/users y http://localhost:8082/products, respectivamente. Las respuestas de los servicios fueron verificadas para asegurar que ambos microservicios respondieran correctamente con los mensajes "Lista de usuarios" y "Lista de productos".

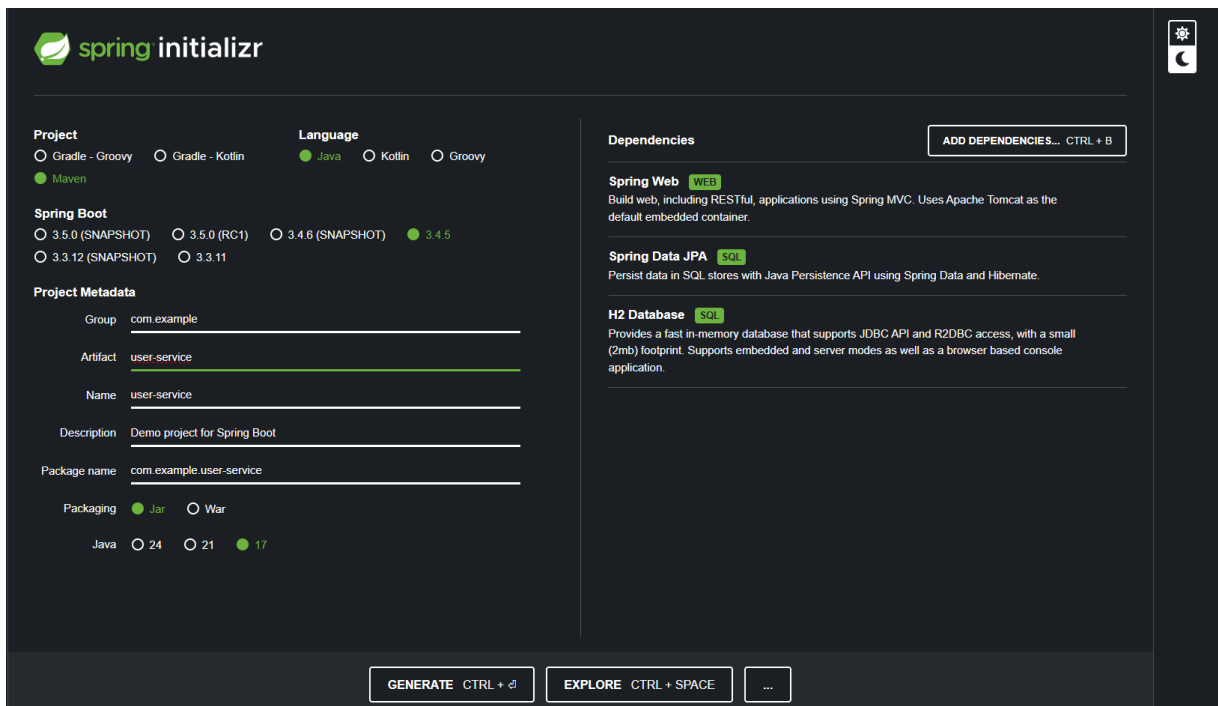
Desarrollo

El desarrollo del sistema se basó en la creación de dos microservicios independientes utilizando Spring Boot. Cada uno de estos microservicios gestiona un conjunto de funcionalidades relacionadas y expone un endpoint HTTP para interactuar con el sistema.

Creación del Proyecto para el Microservicio de Usuarios (User-Service)

Inicialización del Proyecto: Se utilizó Spring Initializr para crear un nuevo proyecto de Spring Boot con las siguientes configuraciones:

- Nombre del Proyecto: user-service
- Dependencias: Spring Web, Spring Boot DevTools (para recarga automática en desarrollo), y Spring Boot Starter.



The screenshot displays the Spring Initializr web interface. The 'Project' section shows 'Maven' selected. The 'Language' section has 'Java' selected. The 'Spring Boot' section shows version '3.4.5' selected. The 'Project Metadata' section includes fields for 'Group' (com.example), 'Artifact' (user-service), 'Name' (user-service), 'Description' (Demo project for Spring Boot), and 'Package name' (com.example.user-service). The 'Packaging' section has 'Jar' selected. The 'Dependencies' section lists 'Spring Web', 'Spring Data JPA', and 'H2 Database'. At the bottom, there are buttons for 'GENERATE', 'EXPLORE', and a menu icon.

Project

☐ Gradle - Groovy ☐ Gradle - Kotlin ☒ Java ☐ Kotlin ☐ Groovy

Spring Boot

☐ 3.5.0 (SNAPSHOT) ☐ 3.5.0 (RC1) ☐ 3.4.6 (SNAPSHOT) ☒ 3.4.5

☐ 3.3.12 (SNAPSHOT) ☐ 3.3.11

Project Metadata

Group: com.example

Artifact: user-service

Name: user-service

Description: Demo project for Spring Boot

Package name: com.example.user-service

Packaging: ☒ Jar ☐ War

Java: ☐ 24 ☐ 21 ☒ 17

Dependencies

Spring Web **WEB**
Build web, including RESTful, applications using Spring MVC. Uses Apache Tomcat as the default embedded container.

Spring Data JPA **SQL**
Persist data in SQL stores with Java Persistence API using Spring Data and Hibernate.

H2 Database **SQL**
Provides a fast in-memory database that supports JDBC API and R2DBC access, with a small (2mb) footprint. Supports embedded and server modes as well as a browser based console application.

GENERATE CTRL + G **EXPLORE** CTRL + SPACE ...

Creación del Controlador: El controlador de User-Service fue creado para exponer un endpoint HTTP que devuelva una lista de usuarios (en esta fase, la lista es estática).

```
package com.example.user_service.controller;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class UserController {

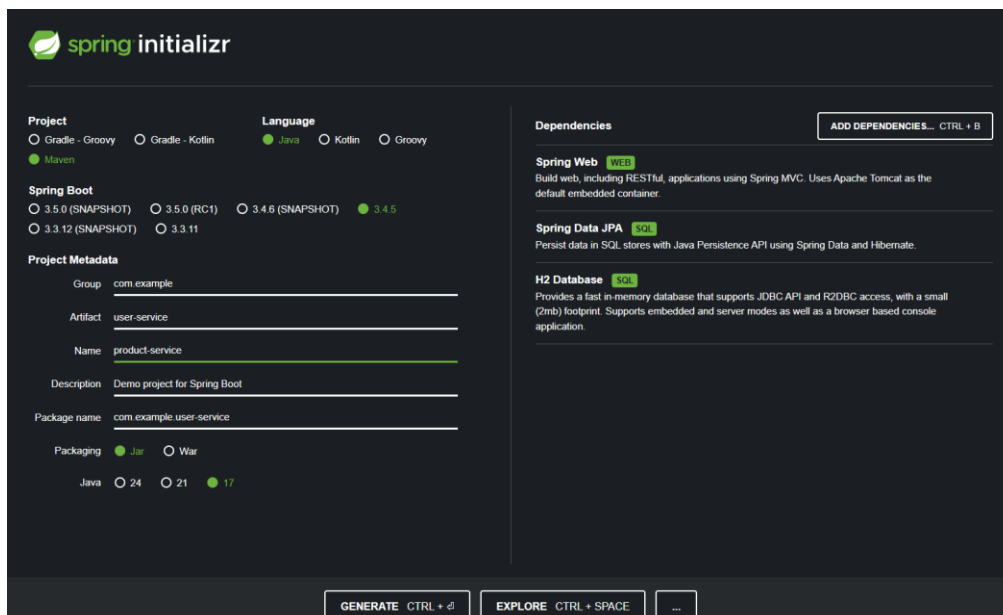
    @GetMapping("/users")
    public String getUsers() {
        return "Lista de usuarios";
    }
}
```

Este controlador define un endpoint GET /users que responde con el mensaje "Lista de usuarios".

Creación del Proyecto para el Microservicio de Productos (Product-Service)

Inicialización del Proyecto: Similar al microservicio de usuarios, se creó un nuevo proyecto de Spring Boot para el microservicio de productos con las siguientes configuraciones:

- Nombre del Proyecto: product-service
- Dependencias: Spring Web, Spring Boot DevTools, y Spring Boot Starter.



The screenshot shows the Spring Initializr web application interface. The 'Project' section has 'Maven' selected. The 'Language' section has 'Java' selected. The 'Spring Boot' section has '3.4.5' selected. The 'Project Metadata' section shows the following fields: Group (com.example), Artifact (user-service), Name (product-service), Description (Demo project for Spring Boot), and Package name (com.example.user-service). The 'Packaging' section has 'Jar' selected. The 'Java' section has '17' selected. The 'Dependencies' section shows 'Spring Web' and 'Spring Data JPA' selected. The 'H2 Database' dependency is also visible. At the bottom, there are buttons for 'GENERATE', 'EXPLORE', and a menu icon.

El controlador de Product-Service fue creado de manera similar al de User-Service.

```
package com.example.product_service.controller;

import org.springframework.web.bind.annotation.GetMapping;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class ProductController {

    @GetMapping("/products")
    public String getProducts() {
        return "Lista de productos";
    }
}
```

Este controlador define un endpoint GET /products que responde con el mensaje "Lista de productos".

Se configuraron los puertos en los que se ejecutarán los microservicios. Para esto, se modificó el archivo application.properties de cada servicio:

- user-service

```
server.port=8081
spring.application.name=user-service
```

- producto-service

```
server.port=8082
spring.application.name=product-service
```

De esta forma, los microservicios se ejecutan en puertos distintos (8081 para User-Service y 8082 para Product-Service).

En la ejecución debemos para iniciar cada microservicio, se utilizó Maven con el siguiente comando desde la terminal en la carpeta de cada microservicio:

```
C:\Users\AMD>cd C:\Users\AMD\OneDrive - Instituto Politecnico Nacional\Documents\NetBeansProjects\product-service
C:\Users\AMD\OneDrive - Instituto Politecnico Nacional\Documents\NetBeansProjects\product-service>mvn spring-boot:run
```

```
C:\Users\AMD>cd C:\Users\AMD\OneDrive - Instituto Politecnico Nacional\Documents\NetBeansProjects\user-service
C:\Users\AMD\OneDrive - Instituto Politecnico Nacional\Documents\NetBeansProjects\user-service>mvn spring-boot:run
[INFO] Scanning for projects...
```

Este comando inicia el servidor de Spring Boot en el puerto configurado (8081 para el servicio de usuarios y 8082 para el servicio de productos).

En la consola, se verificó que cada microservicio se haya iniciado correctamente. El mensaje "Tomcat started on port" indica que el servicio está corriendo correctamente en el puerto correspondiente.

- user-service

```
[INFO] Copying 0 resource from src/main/resources to target/classes
[INFO] --- compiler:3.10.0:compile (default-compile) @ product-service ---
[INFO] Recompiling the module because of added or removed source files.
[INFO] Compiling 2 source files with javac [debug parameters release 17] to target/classes
[INFO] --- resources:3.3.1:testResources (default-testResources) @ product-service ---
[INFO] skip non existing resourceDirectory C:\Users\AND\OneDrive - Instituto Politecnico Nacional\Documents\NetBeansProjects\product-service\src\test\resources
[INFO] --- compiler:3.10.0:testCompile (default-testCompile) @ product-service ---
[INFO] Recompiling the module because of changed dependency.
[INFO] Compiling 1 source file with javac [debug parameters release 17] to target\test-classes
[INFO] <<< spring-boot:3.4.5:run (default-cli) < test-compile @ product-service >>>
[INFO] --- spring-boot:3.4.5:run (default-cli) @ product-service ---
[INFO] Attaching agents: []

Spring Boot (v3.4.5)

2025-04-27T02:11:35:070-06:00 INFO 4124 --- [product-service] [main] c.e.p.ProductServiceApplication : Starting ProductServiceApplication using Java 17.0.12 with PID 4124 (C:\Users\AND\OneDrive - Instituto Politecnico Nacional\Documents\NetBeansProjects\product-service\src\test\resources)
2025-04-27T02:11:35:076-06:00 INFO 4124 --- [product-service] [main] jpaaseConfigurationJpaWebConfiguration : No active profile set, falling back to 1 default profile: "default"
2025-04-27T02:11:35:071-06:00 INFO 4124 --- [product-service] [main] s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2025-04-27T02:11:35:076-06:00 INFO 4124 --- [product-service] [main] s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 12 ms. Found 0 JPA repository interfaces.
2025-04-27T02:11:35:411-06:00 INFO 4124 --- [product-service] [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port 8082 (http)
2025-04-27T02:11:35:426-06:00 INFO 4124 --- [product-service] [main] o.apache.catalina.core.StandardEngine : Starting service [Tomcat]
2025-04-27T02:11:35:426-06:00 INFO 4124 --- [product-service] [main] o.apache.catalina.core.StandardEngine : Starting Servlet engine: [Apache Tomcat/10.1.40]
2025-04-27T02:11:35:494-06:00 INFO 4124 --- [product-service] [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2025-04-27T02:11:35:495-06:00 INFO 4124 --- [product-service] [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1350 ms
2025-04-27T02:11:35:637-06:00 INFO 4124 --- [product-service] [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2025-04-27T02:11:35:835-06:00 INFO 4124 --- [product-service] [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
2025-04-27T02:11:35:837-06:00 INFO 4124 --- [product-service] [main] o.hibernate.jpa.internal.util.LogHelper : HH000024: Processing PersistenceUnitInfo [name: default]
2025-04-27T02:11:35:846-06:00 INFO 4124 --- [product-service] [main] org.hibernate.Version : HH000012: Hibernate ORM core version 6.0.13.Final
2025-04-27T02:11:35:985-06:00 INFO 4124 --- [product-service] [main] o.h.e.internal.RegionFactoryInitiator : HH000026: Second-level cache disabled
2025-04-27T02:11:37:292-06:00 INFO 4124 --- [product-service] [main] o.s.o.j.p.SpringPersistenceUnitInfo : No LoadTimeWeaver setup; ignoring JPA class transformer
2025-04-27T02:11:37:465-06:00 INFO 4124 --- [product-service] [main] org.hibernate.orm.connections.pooling : HH000010: Database info:
Database JDBC URL [Connecting through datasource "HikariDataSource (HikariPool-1)"]
Database driver: undefined/unknown
Database version: 2.3.232
Autocommit mode: undefined/unknown
Isolation level: undefined/unknown
Minimum pool size: undefined/unknown
Maximum pool size: undefined/unknown
2025-04-27T02:11:37:751-06:00 INFO 4124 --- [product-service] [main] o.h.e.t.j.p.i.JtaPlatformInitiator : HH000089: No JTA platform available (set 'hibernate.transaction.jta.platform' to enable JTA platform integration)
2025-04-27T02:11:37:764-06:00 INFO 4124 --- [product-service] [main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2025-04-27T02:11:37:884-06:00 INFO 4124 --- [product-service] [main] jpaaseConfigurationJpaWebConfiguration : Spring JPA open-in-view is enabled by default. Therefore, database queries may be performed during view rendering.
2025-04-27T02:11:38:165-06:00 INFO 4124 --- [product-service] [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat started on port 8082 (http) with context path '/'
2025-04-27T02:11:38:175-06:00 INFO 4124 --- [product-service] [main] c.e.p.ProductServiceApplication : Started ProductServiceApplication in 3.654 seconds (process running for 4.157)
2025-04-27T02:17:56:374-06:00 INFO 4124 --- [product-service] [nio-8082-exec-2] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2025-04-27T02:17:56:374-06:00 INFO 4124 --- [product-service] [nio-8082-exec-2] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2025-04-27T02:17:56:374-06:00 INFO 4124 --- [product-service] [nio-8082-exec-2] o.s.web.servlet.DispatcherServlet : Completed initialization in 2 ms
```

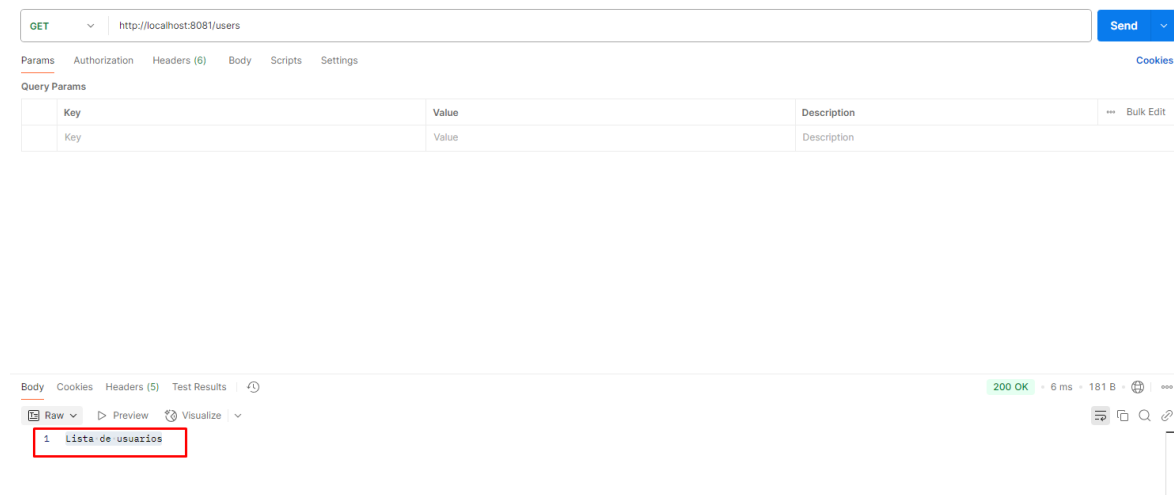
- product-service

```
Downloaded from central: https://repo.maven.apache.org/maven2/net/java/dev/jna/jna-5.13.0-jar (1.9 MB at 4.4 MB/s)
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/shared/maven-dependency-tree/3.2.1/maven-dependency-tree-3.2.1-jar
Downloaded from central: https://repo.maven.apache.org/maven2/org/eclipse/aether/aether-util/1.0.0.v20140518/aether-util-1.0.0.v20140518-jar
Downloaded from central: https://repo.maven.apache.org/maven2/org/om2/asm/asm-commons/9.5/asm-commons-9.5-jar (77 kB at 106 kB/s)
Downloaded from central: https://repo.maven.apache.org/maven2/org/eclipse/aether/aether-api/1.0.0.v20140518/aether-api-1.0.0.v20140518-jar
Downloaded from central: https://repo.maven.apache.org/maven2/org/om2/asm/asm-tree/9.5/asm-tree-9.5-jar (52 kB at 113 kB/s)
Downloaded from central: https://repo.maven.apache.org/maven2/org/comens/fo-commons-fo/2.12.42/fo-commons-fo-2.12.42-jar
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/maven/shared/maven-dependency-tree/3.2.1/maven-dependency-tree-3.2.1-jar (43 kB at 91 kB/s)
Downloaded from central: https://repo.maven.apache.org/maven2/org/valer/jdependency/2.8.0/jdependency-2.8.0-jar
Downloaded from central: https://repo.maven.apache.org/maven2/org/eclipse/aether/aether-util/1.0.0.v20140518/aether-util-1.0.0.v20140518-jar (146 kB at 305 kB/s)
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/commons/commons-collections4/4.4/commons-collections4-4.4-jar
Downloaded from central: https://repo.maven.apache.org/maven2/org/jdom/jdom2/2.0.6.1/jdom2-2.0.6.1-jar (126 kB at 262 kB/s)
Downloaded from central: https://repo.maven.apache.org/maven2/org/eclipse/aether/aether-api/1.0.0.v20140518/aether-api-1.0.0.v20140518-jar (136 kB at 270 kB/s)
Downloaded from central: https://repo.maven.apache.org/maven2/org/valer/jdependency/2.8.0/jdependency-2.8.0-jar (233 kB at 441 kB/s)
Downloaded from central: https://repo.maven.apache.org/maven2/org/apache/commons/commons-collections4/4.4/commons-collections4-4.4-jar (752 kB at 1.4 MB/s)
[INFO] Attaching agents: []

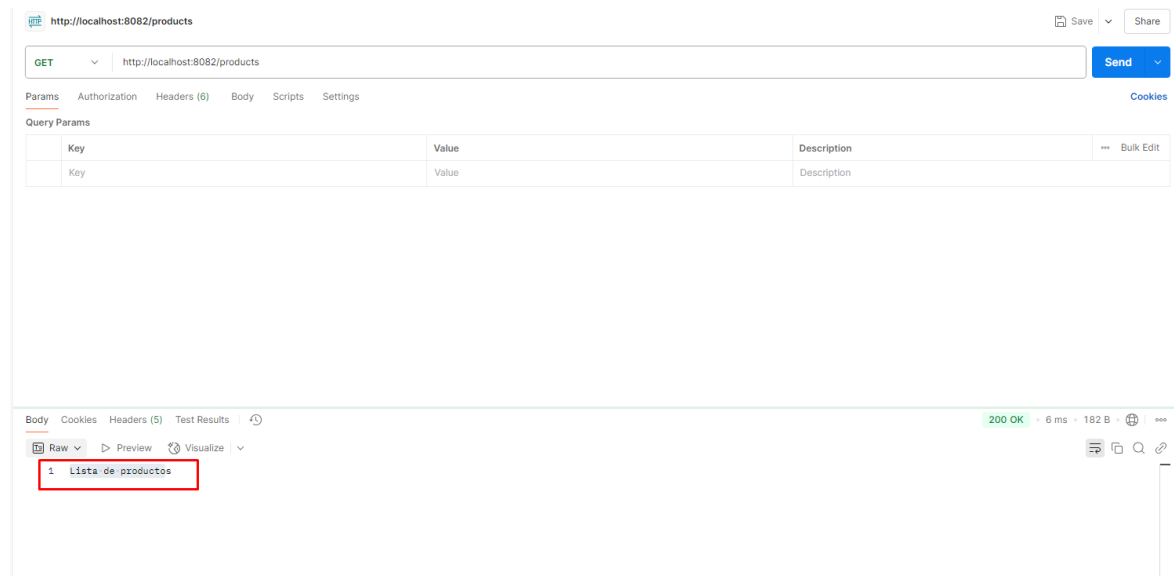
Spring Boot (v3.4.5)

2025-04-27T02:10:18:376-06:00 INFO 15748 --- [user-service] [main] c.e.user.service.UserServiceApplication : Starting UserServiceApplication using Java 17.0.12 with PID 15748 (C:\Users\AND\OneDrive - Instituto Politecnico Nacional\Documents\NetBeansProjects\user-service)
2025-04-27T02:10:18:380-06:00 INFO 15748 --- [user-service] [main] c.e.user.service.UserServiceApplication : No active profile set, falling back to 1 default profile: "default"
2025-04-27T02:10:18:387-06:00 INFO 15748 --- [user-service] [main] s.d.r.c.RepositoryConfigurationDelegate : Bootstrapping Spring Data JPA repositories in DEFAULT mode.
2025-04-27T02:10:18:401-06:00 INFO 15748 --- [user-service] [main] s.d.r.c.RepositoryConfigurationDelegate : Finished Spring Data repository scanning in 16 ms. Found 0 JPA repository interfaces.
2025-04-27T02:10:18:482-06:00 INFO 15748 --- [user-service] [main] o.s.b.w.embedded.tomcat.TomcatWebServer : Tomcat initialized with port 8081 (http)
2025-04-27T02:10:18:497-06:00 INFO 15748 --- [user-service] [main] o.apache.catalina.core.StandardEngine : Starting service [Tomcat]
2025-04-27T02:10:18:564-06:00 INFO 15748 --- [user-service] [main] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring embedded WebApplicationContext
2025-04-27T02:10:18:566-06:00 INFO 15748 --- [user-service] [main] w.s.c.ServletWebServerApplicationContext : Root WebApplicationContext: initialization completed in 1129 ms
2025-04-27T02:10:18:698-06:00 INFO 15748 --- [user-service] [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Starting...
2025-04-27T02:10:18:881-06:00 INFO 15748 --- [user-service] [main] com.zaxxer.hikari.HikariDataSource : HikariPool-1 - Start completed.
2025-04-27T02:10:18:936-06:00 INFO 15748 --- [user-service] [main] o.hibernate.jpa.internal.util.LogHelper : HH000024: Processing PersistenceUnitInfo [name: default]
2025-04-27T02:10:19:051-06:00 INFO 15748 --- [user-service] [main] org.hibernate.Version : HH000012: Hibernate ORM core version 6.0.13.Final
2025-04-27T02:10:19:021-06:00 INFO 15748 --- [user-service] [main] o.h.e.internal.RegionFactoryInitiator : HH000026: Second-level cache disabled
2025-04-27T02:10:19:123-06:00 INFO 15748 --- [user-service] [main] o.s.o.j.p.SpringPersistenceUnitInfo : No LoadTimeWeaver setup; ignoring JPA class transformer
2025-04-27T02:10:19:484-06:00 INFO 15748 --- [user-service] [main] org.hibernate.orm.connections.pooling : HH000010: Database info:
Database JDBC URL [Connecting through datasource "HikariDataSource (HikariPool-1)"]
Database driver: undefined/unknown
Database version: 2.3.232
Autocommit mode: undefined/unknown
Isolation level: undefined/unknown
Minimum pool size: undefined/unknown
Maximum pool size: undefined/unknown
2025-04-27T02:10:19:810-06:00 INFO 15748 --- [user-service] [main] o.h.e.t.j.p.i.JtaPlatformInitiator : HH000089: No JTA platform available (set 'hibernate.transaction.jta.platform' to enable JTA platform integration)
2025-04-27T02:10:19:825-06:00 INFO 15748 --- [user-service] [main] j.LocalContainerEntityManagerFactoryBean : Initialized JPA EntityManagerFactory for persistence unit 'default'
2025-04-27T02:10:19:861-06:00 INFO 15748 --- [user-service] [main] jpaaseConfigurationJpaWebConfiguration : Spring JPA open-in-view is enabled by default. Therefore, database queries may be performed during view rendering.
2025-04-27T02:10:19:907-06:00 INFO 15748 --- [user-service] [main] c.e.user.service.UserServiceApplication : Started UserServiceApplication in 3.327 seconds (process running for 3.702)
2025-04-27T02:17:28:818-06:00 INFO 15748 --- [user-service] [nio-8081-exec-2] o.a.c.c.C.[Tomcat].[localhost].[/] : Initializing Spring DispatcherServlet 'dispatcherServlet'
2025-04-27T02:17:28:818-06:00 INFO 15748 --- [user-service] [nio-8081-exec-2] o.s.web.servlet.DispatcherServlet : Initializing Servlet 'dispatcherServlet'
2025-04-27T02:17:28:818-06:00 INFO 15748 --- [user-service] [nio-8081-exec-2] o.s.web.servlet.DispatcherServlet : Completed initialization in 1 ms
```

Con el servicio de usuarios corriendo en el puerto 8081, se utilizó Postman para enviar una solicitud GET a la URL `http://localhost:8081/users`. La respuesta esperada fue:



De manera similar, se envió una solicitud GET a `http://localhost:8082/products` para verificar que el microservicio de productos respondiera correctamente. La respuesta esperada fue:



Ambas pruebas fueron exitosas, y ambos microservicios respondieron correctamente con los mensajes esperados. Esto confirma que los microservicios están funcionando de manera independiente y exponen los endpoints correctamente.

Resultados

Ejecución Correcta de los Microservicios

Al ejecutar ambos microservicios, User-Service y Product-Service, en sus respectivos puertos (8081 y 8082), se observó que ambos servicios se inicializaron correctamente y estuvieron disponibles para recibir solicitudes. Los mensajes de inicio en la consola confirmaron que tanto User-Service como Product-Service habían arrancado sin errores. A continuación, se presentan los detalles de la ejecución:

- User-Service:
 - El servicio fue accesible a través de la URL `http://localhost:8081/users`.
 - Respuesta esperada: "Lista de usuarios".
 - Confirmación: La respuesta se obtuvo sin problemas, validando que el servicio estaba funcionando como se esperaba.
- Product-Service:
 - El servicio fue accesible a través de la URL `http://localhost:8082/products`.
 - Respuesta esperada: "Lista de productos".
 - Confirmación: La respuesta se recibió correctamente, lo que indica que el servicio también estaba funcionando como se esperaba.

Ambos servicios respondieron con éxito y sin errores, lo que demuestra que la configuración de Spring Boot fue adecuada para el entorno local.

Pruebas de Funcionalidad con Postman

Al realizar las pruebas utilizando Postman, se logró verificar que ambos microservicios son capaces de manejar solicitudes GET y devolver las respuestas esperadas:

- Prueba en User-Service:
 - Al hacer una solicitud GET a `http://localhost:8081/users`, se recibió el mensaje "Lista de usuarios" como respuesta.
 - Esto indica que el microservicio de usuarios está correctamente configurado para manejar solicitudes y proporcionar una respuesta simple.

- Prueba en Product-Service:
 - Al hacer una solicitud GET a `http://localhost:8082/products`, la respuesta fue "Lista de productos".
 - Esto confirma que el microservicio de productos también responde correctamente a las solicitudes.

Ambos servicios pasaron las pruebas de manera satisfactoria, lo que demuestra que las solicitudes HTTP fueron manejadas correctamente y las respuestas fueron devueltas como se esperaba. A pesar de que las respuestas son estáticas, esta implementación puede servir como base para extender la funcionalidad hacia operaciones más complejas como el acceso a bases de datos o la integración de más lógica de negocio.

Implementación Sencilla y Escalable

Los resultados obtenidos indican que la estructura de microservicios, con dos servicios independientes funcionando en diferentes puertos, proporciona una arquitectura escalable y fácil de mantener. Los beneficios de la arquitectura de microservicios incluyen:

- Independencia de los servicios: Cada microservicio es independiente y tiene su propio ciclo de vida, lo que facilita el mantenimiento y la evolución de cada uno de ellos sin afectar al otro. Por ejemplo, si es necesario modificar el servicio de productos, se puede hacer sin impactar el servicio de usuarios.
- Escalabilidad: Cada microservicio puede ser escalado de manera independiente. Si uno de los servicios recibe más tráfico, se puede incrementar su capacidad sin necesidad de modificar o escalar el otro servicio.
- Flexibilidad para agregar nuevas funcionalidades: Si en el futuro se requiere agregar más funcionalidades, como la gestión de productos o usuarios desde una base de datos, esto se puede hacer sin mayor complejidad, ya que cada servicio está encapsulado de forma independiente.

Limitaciones y Oportunidades para Futuras Mejoras

Aunque los microservicios están funcionando correctamente en este momento, existen áreas de mejora que pueden ampliarse para hacer la solución más robusta:

- Persistencia de Datos:

- Actualmente, los servicios devuelven respuestas estáticas (listas de usuarios y productos). En una implementación real, los datos de usuarios y productos probablemente se almacenarían en una base de datos (como MySQL, PostgreSQL o MongoDB).
 - Se podría integrar un repositorio en cada microservicio que conecte con una base de datos, permitiendo que los servicios manejen y devuelvan datos dinámicos en lugar de respuestas estáticas.
- Autenticación y Autorización:
 - Los servicios actuales no implementan ningún mecanismo de seguridad. En una solución de microservicios real, se requeriría una capa de seguridad para asegurar que solo los usuarios autenticados puedan acceder a ciertos recursos.
 - Se podría integrar Spring Security para agregar autenticación mediante tokens JWT o integrarse con un sistema de autenticación basado en OAuth2.
- Comunicación entre Microservicios:
 - En la actual arquitectura, los microservicios son independientes y no se comunican entre sí. En una solución más compleja, los microservicios podrían necesitar comunicarse entre sí, por ejemplo, para compartir datos de productos con usuarios.
 - Esto se puede lograr utilizando RESTful APIs o tecnologías como Apache Kafka o RabbitMQ para la comunicación asíncrona entre servicios.
- Monitoreo y Gestión:
 - Para servicios desplegados en producción, es crucial implementar soluciones de monitoreo como Spring Boot Actuator, que permita verificar el estado y el rendimiento de los servicios.
 - Además, se podrían implementar herramientas como Prometheus o Grafana para la visualización de métricas y alertas.

La implementación básica de los microservicios User-Service y Product-Service ha sido exitosa. Ambos servicios están configurados correctamente y responden a las solicitudes HTTP de manera eficiente. Este enfoque

modular y desacoplado de los servicios ofrece una base sólida para construir aplicaciones escalables y flexibles. Sin embargo, es fundamental mejorar la persistencia de datos, la seguridad y la comunicación entre microservicios para llevar la solución a un entorno de producción real.

Este proyecto demuestra cómo los microservicios pueden mejorar la organización y escalabilidad de aplicaciones grandes y complejas, permitiendo que los diferentes componentes sean gestionados de forma independiente.

Conclusión

La implementación de microservicios en este proyecto ha demostrado ser una solución eficaz para estructurar una aplicación que gestiona diferentes servicios de forma independiente y escalable. A través de los microservicios User-Service y Product-Service, hemos logrado dividir la lógica en componentes autónomos que pueden evolucionar y escalar de manera independiente. Cada uno de estos servicios es capaz de responder correctamente a las solicitudes HTTP, como lo demostraron las pruebas realizadas con Postman.

El uso de Spring Boot como marco de desarrollo facilitó la creación de estos microservicios, proporcionándonos un entorno flexible y robusto para construir y desplegar los servicios. Además, al ejecutar ambos servicios en diferentes puertos, hemos logrado simular un entorno básico de microservicios que puede ser fácilmente ampliado con nuevas funcionalidades.

Sin embargo, esta implementación básica solo cubre las operaciones de lectura con respuestas estáticas. Para convertir este sistema en una solución más robusta y adecuada para producción, es necesario integrar bases de datos para la persistencia de datos, mejorar la seguridad con mecanismos de autenticación y autorización, y permitir la comunicación entre microservicios para crear una solución más dinámica.

Por lo que podemos decir que los microservicios ofrecen una arquitectura moderna que mejora la organización, mantenimiento y escalabilidad de las aplicaciones. Este proyecto sentó las bases para desarrollar un sistema más complejo y distribuido, preparado para enfrentarse a desafíos más grandes en un entorno real.