UNIVERSIDAD CENTRAL DEL ECUADOR

FACULTAD DE CIENCIAS APLICADAS

INGENIERÍA EN SISTEMAS DE INFORMACIÓN

PROGRAMACIÓN DISTRIBUIDA

DOCENTE: ING. DIEGO PINTO

SEMESTRE: OCTAVO

INTEGRANTES:

- BOLAÑOS JOSUE
- GORDILLO ERICK

12 DE MAYO DEL 2023

Tema: Informe sobre el Sistema Distribuido de Gestión de Libros y Autores

Contenido

1. Objetivos

1. El sistema tiene como objetivo gestionar información sobre libros y sus autores de manera eficiente y escalable, permitiendo a los usuarios buscar registros de la base de datos Postgres de manera rápida y confiable.
2. El sistema distribuido debe ser capaz de manejar altas cargas de usuarios y peticiones concurrentes, escalando horizontalmente para adaptarse a la creciente demanda y garantizar una alta disponibilidad en todo momento.
3. La arquitectura basada en microservicios y el cumplimiento de los 12 factores de microservicios buscan facilitar la mantención del sistema, permitiendo la actualización y despliegue continuo de componentes individuales sin afectar a la totalidad del sistema.

2. Desarrollo y Explicación

El sistema distribuido de gestión de libros y autores está construido utilizando Helidon, un framework de código abierto para aplicaciones Java basadas en microservicios. La elección de Helidon se debe a su enfoque en la simplicidad y su facilidad de integración con tecnologías modernas como Kubernetes y microPerimeters de OpenID Connect.

El sistema además se conecta a una base de datos Postgres para almacenar y recuperar la información de libros y autores. Postgres se seleccionó por su capacidad para manejar grandes volúmenes de datos y soporte para operaciones complejas de consulta. Además, la elección de una base de datos relacional como Postgres es adecuada para el tipo de datos estructurados que se manejarán en el sistema.

La arquitectura del sistema sigue los principios de microservicios, dividiendo la funcionalidad en componentes autónomos y desacoplados. Cada microservicio se enfoca en una tarea específica, como la gestión de libros y autores. Esto permite una mayor flexibilidad y escalabilidad, ya que cada componente puede ser desplegado y escalado independientemente según sea necesario.

Cada microservicio se comunica con los demás a través de API RESTful. La comunicación entre los servicios se realiza de manera asíncrona y desacoplada, lo que permite una mayor tolerancia a fallos y facilita la integración con nuevos componentes en el futuro.

Los 12 Factores

El sistema distribuido sigue los 12 factores de microservicios, una serie de principios y prácticas que guían el desarrollo de aplicaciones modernas y escalables. Estos factores incluyen:

1. Código base: El sistema utiliza control de versiones para el seguimiento de cambios en el código base y asegurar que cada instancia se despliegue con el código correcto.
2. Dependencias: Las dependencias del sistema, como bibliotecas y frameworks, están claramente definidas y aisladas.
3. Configuración: La configuración del sistema se encuentra en archivos externos y es accesible mediante variables de entorno o servicios de configuración, lo que permite cambios sin necesidad de volver a compilar el código.
4. Backing Services: La base de datos Postgres se trata como un servicio externo, lo que facilita la gestión y la escalabilidad.

5. Build, Release, Run: El proceso de desarrollo, generación de artefactos, despliegue y ejecución está automatizado y separado en cada etapa.
6. Procesos: Cada microservicio se ejecuta como un proceso individual y aislado, lo que permite una mejor escalabilidad y tolerancia a fallos.
7. Port Binding: Los microservicios se enlazan a puertos individuales y son accesibles a través de una red.
8. Concurrencia: El sistema es capaz de escalar horizontalmente para manejar la concurrencia de peticiones.
9. Desacoplamiento de servicio: Los microservicios son independientes entre sí y se comunican a través de interfaces bien definidas.
10. Paridad entre desarrollo y producción: Los entornos de desarrollo y producción son similares para evitar sorpresas al desplegar el sistema.
11. Trazabilidad: El sistema registra eventos y errores para facilitar el diagnóstico y la resolución de problemas.
12. Administración de procesos: Las operaciones administrativas se pueden realizar a través de comandos y API, lo que facilita la gestión del sistema.

Archivo "build.gradle"

Es un archivo de configuración utilizado principalmente en proyectos de Java y Kotlin que se construyen con la herramienta de construcción Gradle. Gradle es una herramienta de automatización de construcción de código abierto que se utiliza para compilar, probar, empaquetar y desplegar aplicaciones.

El contenido de build.gradle puede variar según las necesidades del proyecto, pero generalmente incluye las siguientes secciones:

- Plugins: Esta sección específica los plugins de Gradle que se utilizarán en el proyecto. Los plugins proporcionan funcionalidades adicionales, como soporte para Java, Kotlin, aplicaciones web, pruebas, entre otros.
- Dependencias: Aquí se definen las dependencias externas que necesita el proyecto. Por ejemplo, bibliotecas o frameworks que se deben descargar y utilizar en el proyecto. Gradle se encargará de resolver estas dependencias y descargarlas automáticamente desde repositorios como Maven Central o JCenter.
- Tareas personalizadas: Es posible definir tareas personalizadas que se ejecutarán como parte del proceso de construcción. Estas tareas pueden estar relacionadas con la compilación, ejecución de pruebas, empaquetado, despliegue o cualquier otro proceso específico del proyecto.
- Configuraciones del proyecto: En esta sección, se pueden especificar configuraciones específicas para diferentes aspectos del proyecto, como configuraciones para compilación, pruebas, empaquetado, etc.

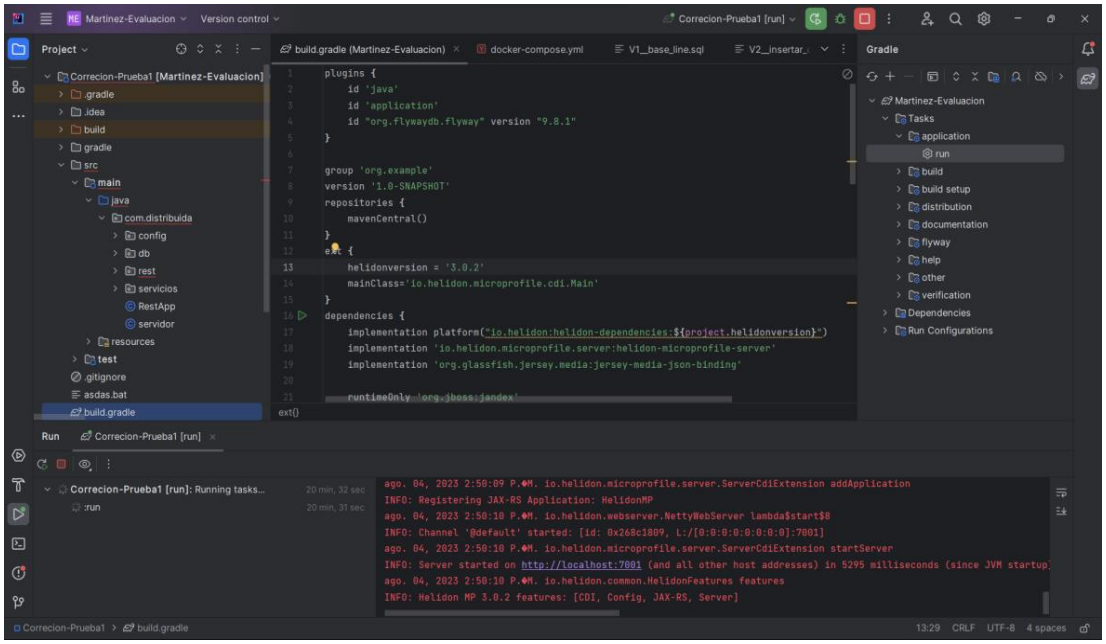
build.gradle (Martinez-Evaluacion) | docker-compose.yml | V1__base_line.sql | V2__insertar_datos.sql | microprofile-config.propert

```
plugins {
    id 'java'
    id 'application'
    id "org.flywaydb.flyway" version "9.8.1"
}

group 'org.example'
version '1.0-SNAPSHOT'
repositories {
    mavenCentral()
}
ext {
    helidonversion = '3.0.2'
    mainClass='io.helidon.microprofile.cdi.Main'
}
dependencies {
    implementation platform("io.helidon:helidon-dependencies:${project.helidonversion}")
    implementation 'io.helidon.microprofile.server:helidon-microprofile-server'
    implementation 'org.glassfish.jersey.media:jersey-media-json-binding'

    runtimeOnly 'org.jboss:jandex'
    // lombok con gradle sin plugin
    compileOnly 'org.projectlombok:lombok:1.18.24'
    annotationProcessor 'org.projectlombok:lombok:1.18.24'

    testCompileOnly 'org.projectlombok:lombok:1.18.24'
    testAnnotationProcessor 'org.projectlombok:lombok:1.18.24'
    //
    implementation group: 'org.jdbi', name: 'jdbi3-core', version: '3.36.0'
    implementation group: 'org.jdbi', name: 'jdbi3-sqlobject', version: '3.36.0'
    implementation 'com.zaxxer:HikariCP:5.0.1'
    implementation 'org.postgresql:postgresql:42.5.1'
```

build.gradle (Martinez-Evaluacion) | docker-compose.yml | V1__base_line.sql | V2__insertar_datos.sql | microprofile-config.properties

```
    implementation group: 'org.jdbi', name: 'jdbi3-sqlobject', version: '3.36.0'
    implementation 'com.zaxxer:HikariCP:5.0.1'
    implementation 'org.postgresql:postgresql:42.5.1'
    implementation group: 'org.flywaydb', name: 'flyway-core', version: '9.12.0'
}
flyway {
    url = 'jdbc:postgresql://localhost:5432/distribuida'
    user = 'postgres'
    password = 'postgres'
    schemas = ['public']
}
sourceSets{
    main{
        output.resourcesDir = file("${buildDir}/classes/java/main")
    }
}
task copyLibs(type: Copy) {
    from configurations.runtimeClasspath
    into 'build/libs/libs'
}
jar {
    archiveFileName = "${project.name}.jar"
    manifest {
        attributes ('Main-Class': "${project.mainClass}" ,
                'Class-Path': configurations.runtimeClasspath.files.collect { File it -> "libs/$it.name" }.join(' ')
        )
    }
}
application {
    mainClass = "${project.mainClass}"
}
```

2.1. App Books



La aplicación book es una API REST para gestionar libros, lo que significa que permite realizar operaciones CRUD (Crear, Leer, Actualizar y Eliminar) sobre una colección de libros. A continuación, describiré la funcionalidad básica de la API:

- Obtener todos los libros:

Método: GET

URL: http://localhost:7001/books

Descripción: Esta solicitud permite obtener una lista de todos los libros almacenados en la base de datos. Los libros se devuelven en formato JSON con detalles como el título, el autor, el género y el año de publicación.

- Obtener un libro por su ID:

Método: GET

URL: http://localhost:7001/books/{id}

Descripción: Esta solicitud permite obtener un libro específico identificado por su ID. El ID se especifica en la URL y la respuesta contiene los detalles del libro en formato JSON.

- Crear un nuevo libro:

Método: POST

URL: http://localhost:7001/books

Cuerpo de la solicitud: JSON con los detalles del libro (id, ISBN, título, autor, costo).

Descripción: Esta solicitud permite agregar un nuevo libro a la colección. Los detalles del libro se envían en el cuerpo de la solicitud y se almacenan en la base de datos. El servidor devuelve la representación JSON del libro creado con su ID asignado.

- Actualizar un libro:

Método: PUT

URL: http://localhost:7001/books/{id}

Cuerpo de la solicitud: JSON con los nuevos detalles del libro (id, ISBN, título, autor, costo).

Descripción: Esta solicitud permite actualizar los detalles de un libro existente identificado por su ID. Los nuevos detalles del libro se envían en el cuerpo de la solicitud y se actualizan en la base de datos.

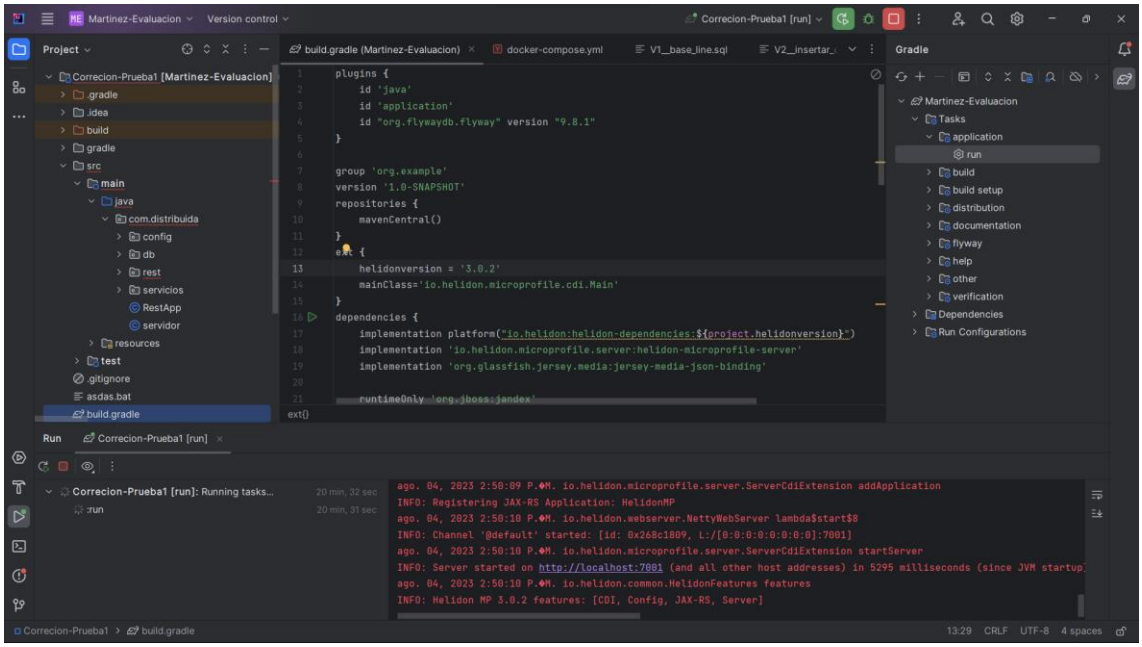
- Eliminar un libro:

Método: DELETE

URL: http://localhost:7001/books/{id}

Descripción: Esta solicitud permite eliminar un libro existente identificado por su ID de la base de datos. El libro correspondiente se elimina de la colección.

2.2. App Author



La aplicación "app-authors" es una API REST diseñada para administrar la información de los autores que están asociados con los libros en la colección gestionada por la aplicación "books". Esta API permite realizar operaciones relacionadas con los autores, como agregar nuevos autores, actualizar sus detalles y eliminarlos si es necesario. A continuación, se describe la funcionalidad básica de la API "app-authors":

- Obtener todos los autores:

Método: GET

URL: http://localhost:7002/authors

Descripción: Esta solicitud permite obtener una lista de todos los autores almacenados en la base de datos. Los autores se devuelven en formato JSON con detalles como el nombre, la nacionalidad, la fecha de nacimiento y cualquier otra información relevante.

- Obtener un autor por su ID:

Método: GET

URL: http://localhost:7002/authors/{id}

Descripción: Esta solicitud permite obtener información detallada de un autor específico identificado por su ID. El ID se proporciona en la URL y la respuesta contiene los detalles del autor en formato JSON.

- Agregar un nuevo autor:

Método: POST

URL: http://localhost:7002/authors

Cuerpo de la solicitud: JSON con los detalles del autor (id, nombre, nacionalidad, fecha de nacimiento, etc.).

Descripción: Esta solicitud permite agregar un nuevo autor a la base de datos. Los detalles del autor se envían en el cuerpo de la solicitud y se almacenan para su uso posterior. El servidor devuelve la representación JSON del autor creado con su ID asignado.

- Actualizar detalles de un autor:

Método: PUT

URL: http://localhost:7002/authors/{id}

Cuerpo de la solicitud: JSON con los nuevos detalles del autor (id, nombre, nacionalidad, fecha de nacimiento, etc.).

Descripción: Esta solicitud permite actualizar los detalles de un autor existente identificado por su ID. Los nuevos detalles del autor se envían en el cuerpo de la solicitud y se actualizan en la base de datos.

- Eliminar un autor:

Método: DELETE

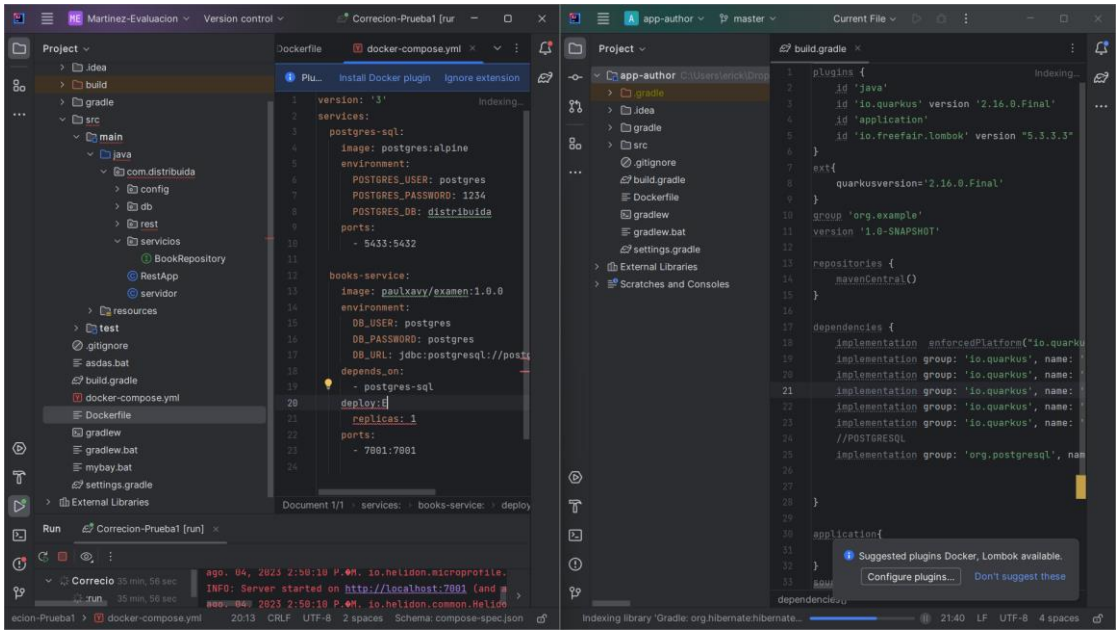
URL: http://localhost:7002/authors/{id}

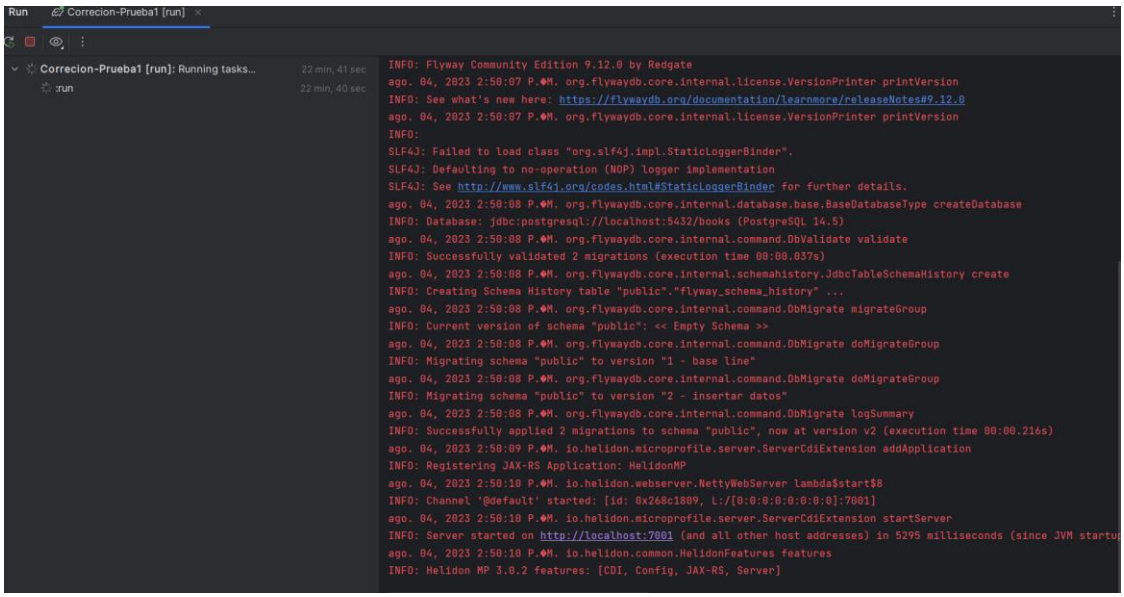
Descripción: Esta solicitud permite eliminar un autor existente identificado por su ID de la base de datos. Toda la información relacionada con el autor se elimina de manera segura.

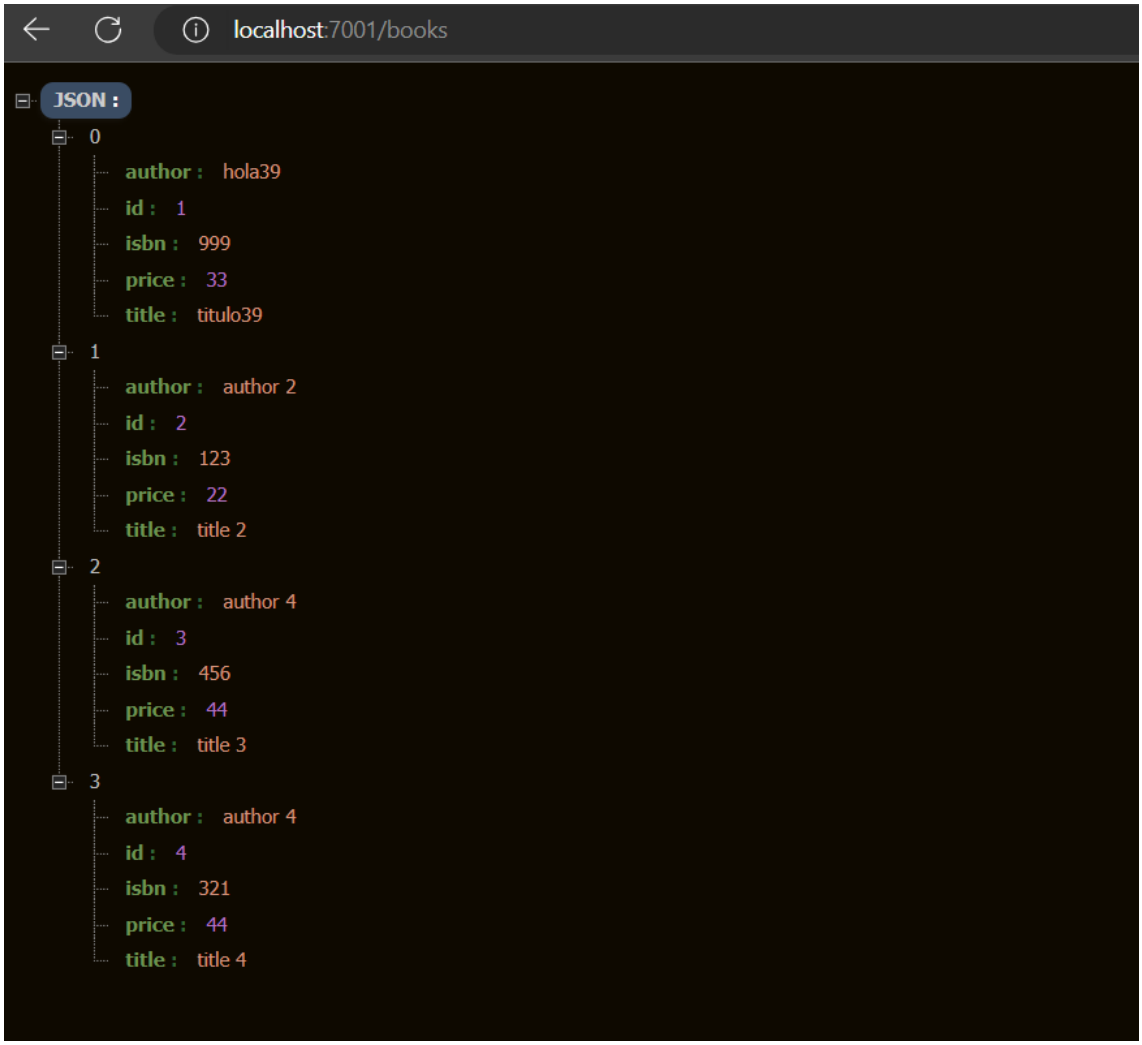
La aplicación "app-authors" está diseñada para trabajar en conjunto con la aplicación "books", permitiendo a los usuarios gestionar y relacionar autores con los libros almacenados en la colección. Esta integración ofrece una experiencia completa para administrar la información tanto de los libros como de sus respectivos autores.

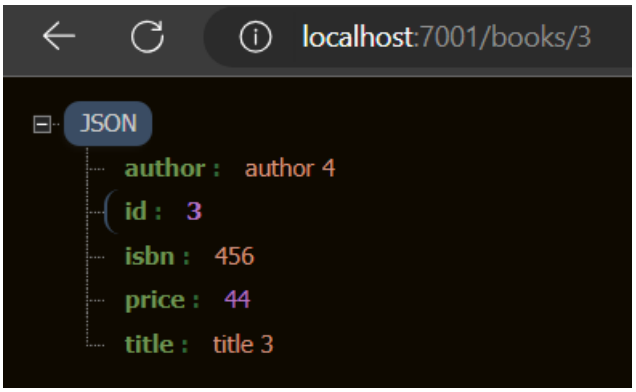
2.3. Ejecución

A continuación, se presenta el sistema en ejecución









3. Conclusiones

La arquitectura distribuida basada en microservicios permite una mayor flexibilidad, escalabilidad y despliegue continuo del sistema. Cada componente puede desarrollarse y mantenerse de manera independiente, lo que facilita la evolución del sistema a medida que cambian los requisitos y se introducen nuevas funcionalidades.

La elección de Helidon como framework de desarrollo brinda ventajas significativas, como la simplicidad y la facilidad de integración con tecnologías modernas. Esto acelera el desarrollo y permite que los equipos enfoquen sus esfuerzos en la lógica de negocio en lugar de preocuparse por la infraestructura y la comunicación entre componentes.

La utilización de una base de datos Postgres proporciona una solución sólida para el almacenamiento de datos estructurados. Al seleccionar tecnologías confiables y probadas, se asegura un alto rendimiento y la capacidad de manejar grandes volúmenes de información de libros y autores.