UNIVERSIDAD CENTRAL DEL ECUADOR

Omnium potentior est sapientia

Ejemplo de sistema distribuido de un CRUD de libros y de autores.

Agosto

2023

Este documento contiene un ejemplo de la implementación de un CRUD para la gestión de libros y autores con la herramienta de IntelliJ IDEA.

Ingeniería Informática

- Dany Simbaña
- Jairo Mena

.

Programación Distribuida

FACULTAD DE INGENIERÍA Y CIENCIAS APLICADAS

TRABAJO

Contenido

TÍTULO:	Ejemplo de sistema distribuido de un CRUD de libros y de autores.
Resumen:	El proyecto de implementación de un sistema distribuido de gestión de libros y de autores busca optimizar las operaciones de una biblioteca, a través de la gestión efectiva, mediante el uso de la herramienta de IntelliJ IDEA.
Autores:	Dany Simbaña Jairo Mena
Responsabilidad del desarrollo y ejecución:	Estudiantes de la Facultad de Ingeniería y Ciencias Aplicadas de la carrera de Ingeniería Informática.
Responsabilidad de la revisión:	Ing. Diego Pinto
Distribución:	A los estudiantes de la Universidad Central del Ecuador

1. Introducción

En el contexto de las bibliotecas modernas, la gestión de libros y autores es una tarea crítica que puede beneficiarse enormemente de la implementación de un sistema distribuido. La distribución de los datos y las operaciones en diferentes nodos permitirá mejorar la escalabilidad y la tolerancia a fallos del sistema, garantizando un servicio confiable y eficiente.

El presente trabajo tiene como objetivo diseñar e implementar un sistema distribuido de gestión de libros y autores, que permita realizar operaciones de CRUD (Crear, Leer, Actualizar y Borrar) de manera efectiva y escalable. Mediante el uso de la herramienta de desarrollo IntelliJ IDEA, se busca proporcionar un entorno de desarrollo robusto y amigable para la construcción y administración del sistema.

2. Objetivo

2.1 Objetivo General

Diseñar e implementar un sistema distribuido de gestión de libros y autores para bibliotecas.

2.2 Objetivos específicos

- Diseñar una arquitectura de microservicios para la gestión de libros y autores, donde cada microservicio se dedique a una funcionalidad específica del CRUD.
- Utilizar IntelliJ IDEA como la herramienta de desarrollo principal para la construcción del sistema distribuido, aprovechando sus capacidades de desarrollo y depuración.
- Garantizar la tolerancia a fallos y la coherencia de los datos en todo el sistema mediante el uso de tecnologías y protocolos de comunicación adecuados.

3. Requisitos mínimos

- Repositorio GitHub
- Base de datos PostgreSQL
- IntelliJ IDEA
- Java 17

4. Repositorio GitHub

1. Repositorio de GitHub, descargamos los dos servicios: **app-authors y app-books.**

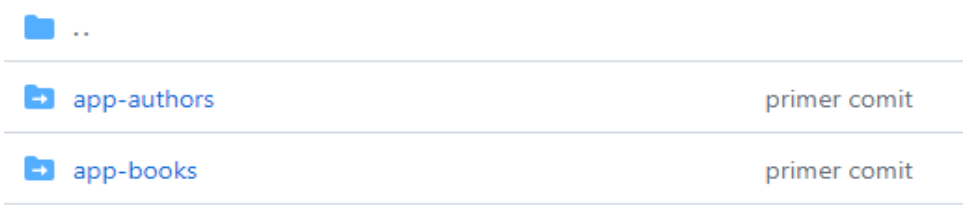

Figura 1. Servicio repositorio de GitHub.

5. Base de datos

1. Modelo entidad relación de base de datos, el motor de base de datos es PostgreSQL

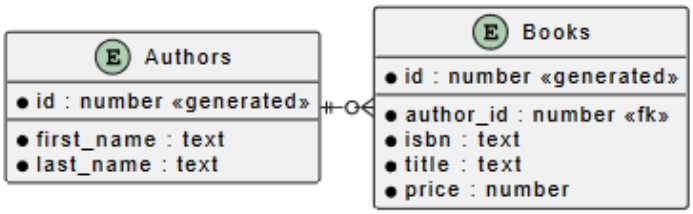

Figura 2. El servicio libros esta consume el servicio de autores, base de datos PostgreSQL.

2.. Donde iniciamos creando una base de datos en PostgreSQL.

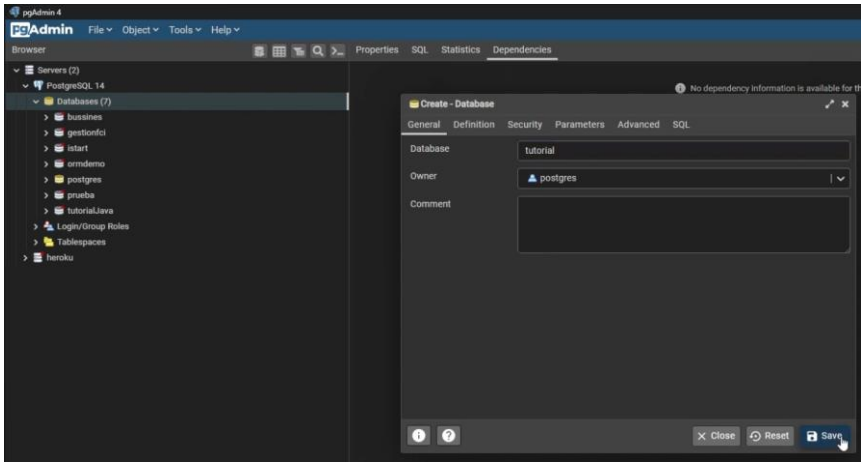

Figura 3. Creación de la base de datos PostgreSQL.

3. Exportamos la base datos. Para este ejemplo se le a nombrado como “distribuida.sql”.

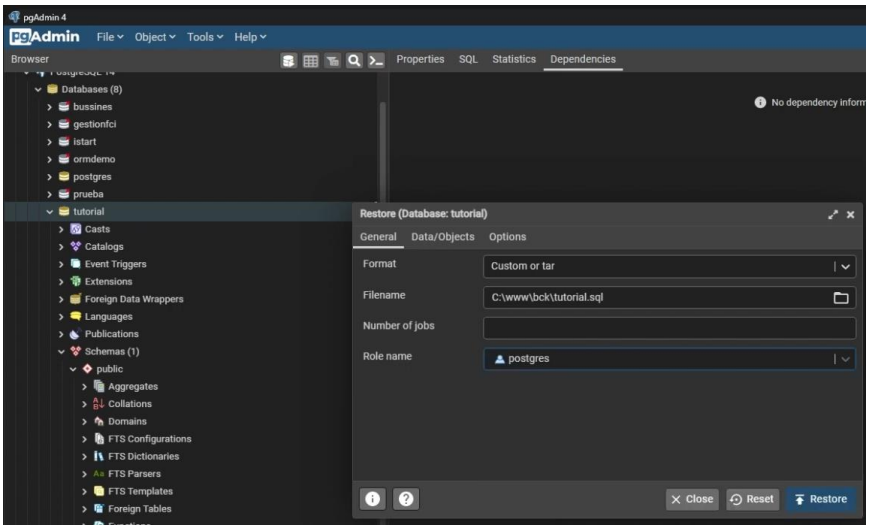

Figura 4. Exportamos la base de datos

4. Verificamos que la exportación de la BD se encuentre todas las tablas.

Figura 5. BD “distribuida” y con las respectivas tablas y campos de cada tabla.

6. Diseño de la arquitectura del sistema

El despliegue debe contener:
app-books
app-authors
Base de datos (postgresql)
Gateway (traefik)
Fault Tolerance

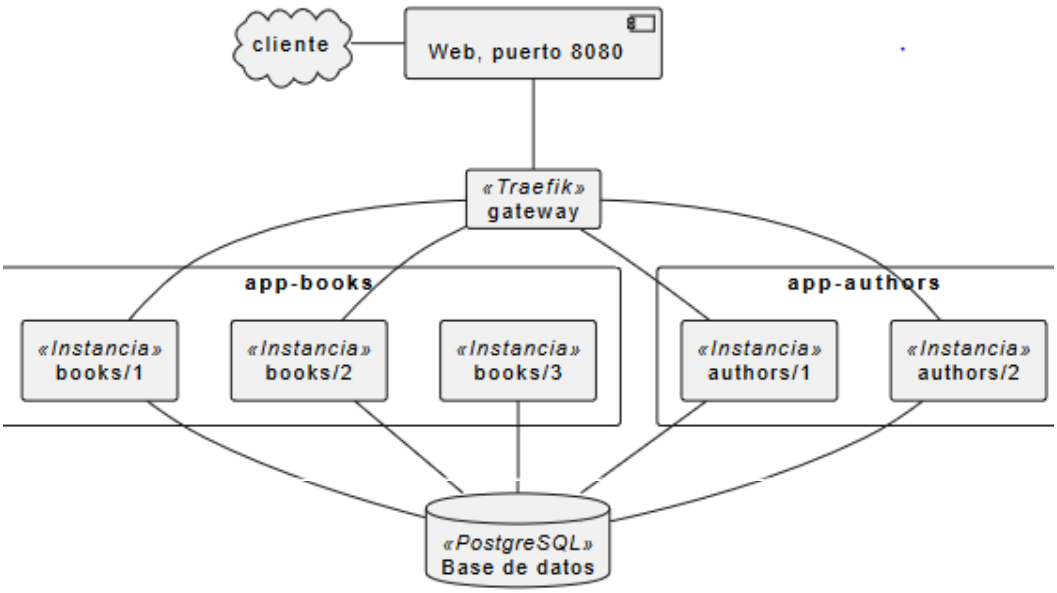

Figura 6. Se muestra el despliegue de la aplicación.

7. Descripción de los módulos

app-books

Este módulo permite realizar operaciones CRUD sobre la tabla Books. La tecnología a utilizar en este módulo corresponde a:

- Helidon 3.1
- Acceso a la base de datos a través de DBClient

app-authors

Este módulo permite realizar operaciones CRUD sobre la tabla Authors. La tecnología a utilizar en este módulo corresponde a:

- Qurkus 2.16
- Acceso a la base de datos a través de Panache-Quarkus Data Repository

8. Implementación de los servicios

En el desarrollo de un sistema distribuido de gestión de libros y autores, es común utilizar una serie de dependencias o bibliotecas para facilitar la implementación de ciertas funcionalidades y tareas específicas.

1. Iniciamos los servicios en IntelliJ IDEA.

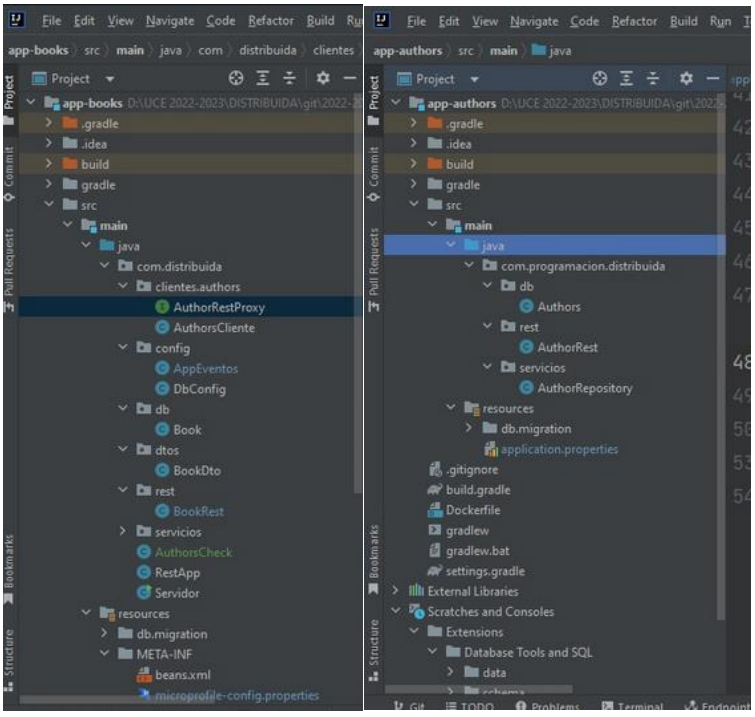

Figura 7. Se muestra los servicios de libros y autores.

2. Empezamos a descargar las dependencias, esto descargara los componentes que se utilizan tanto en el servicio de autores como libros, vamos a build.gradle y damos clic en build.

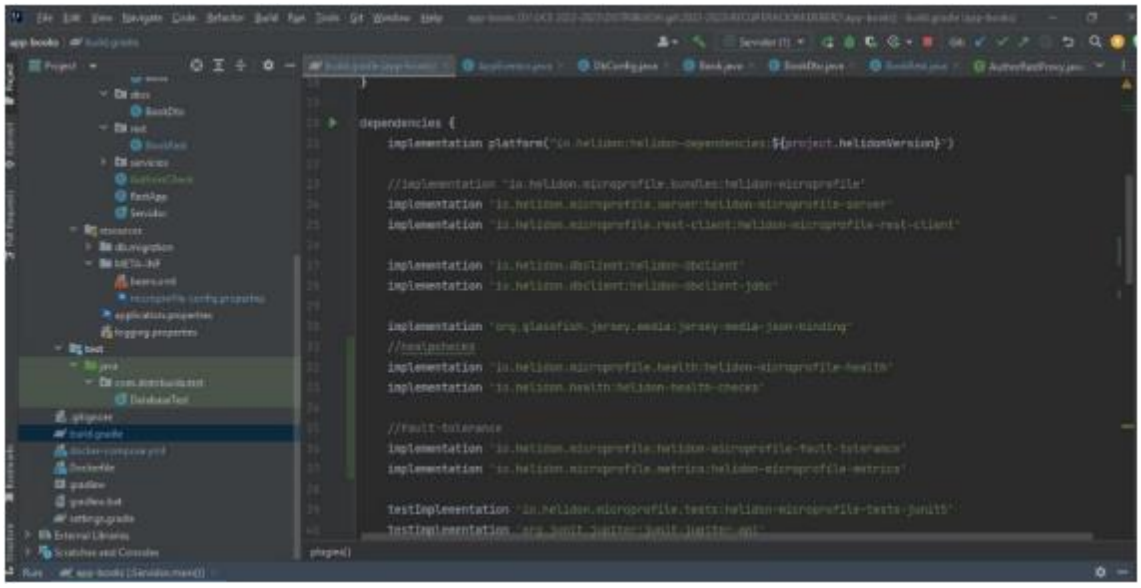

Figura 8. Descarga de dependencias.

3. Para la conexión a la base de datos debemos tener configurado con el usuario y password personal, esto lo configuramos en el archivo de configuración application.properties.

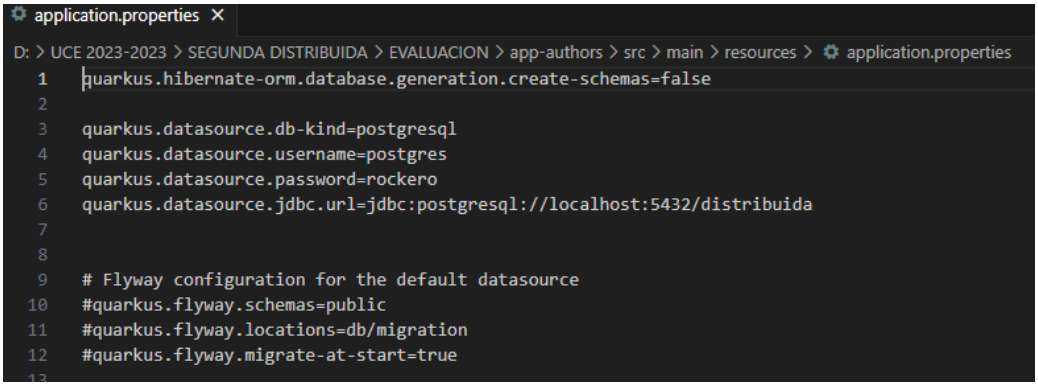

Figura 9. Conexión base de datos PostgreSQL.

4. La tolerancia a fallos es una propiedad importante en sistemas distribuidos que garantiza que el sistema pueda continuar funcionando correctamente, incluso si uno o más componentes fallan o se vuelven inaccesibles. En un sistema distribuido, debido a la naturaleza de su arquitectura, es más probable que ocurran fallos en diferentes partes del sistema, como nodos, comunicaciones o recursos compartidos. La tolerancia a fallos busca mitigar los efectos adversos de estos fallos y asegurar la disponibilidad y fiabilidad del sistema en todo momento.

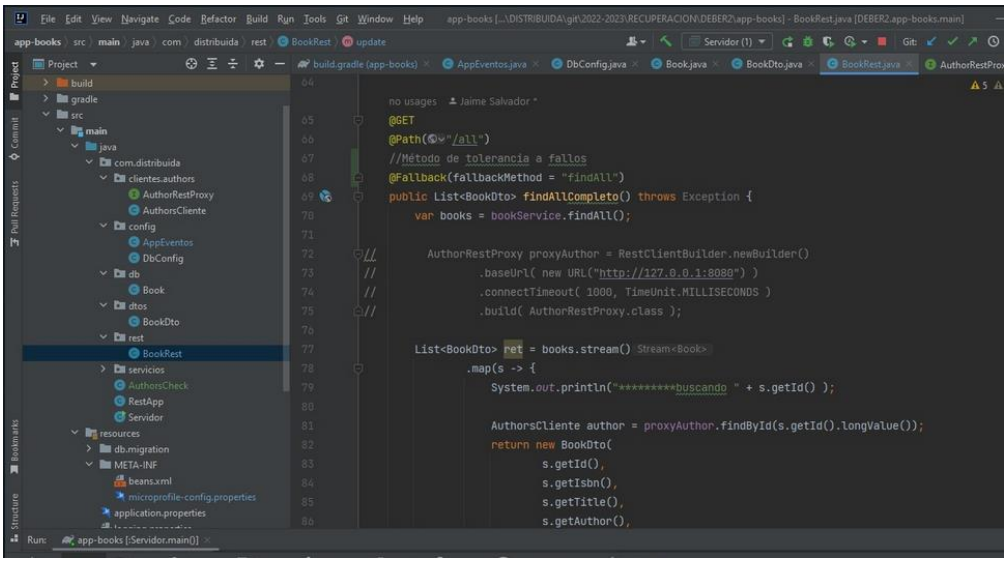

Figura 10. Se muestra el método de la tolerancia a fallos

9. Ejecución del programa

1. Damos clic en el archivo Servidor y lo mandamos a Ejecutar

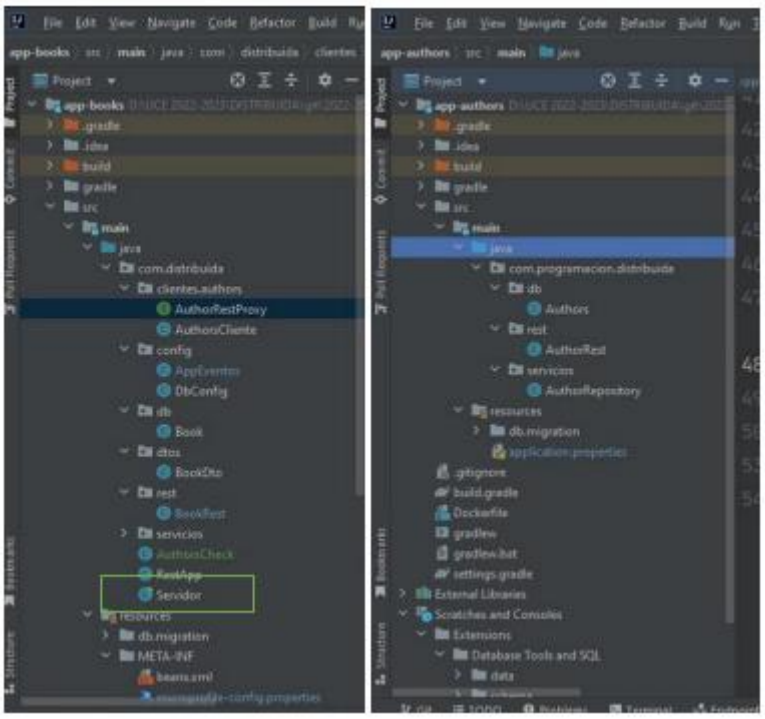

Figura 11. Iniciamos la ejecución del programa

2. Una vez este corriendo las dos apps se mostrará en pantalla los datos que tenemos en la base de datos.

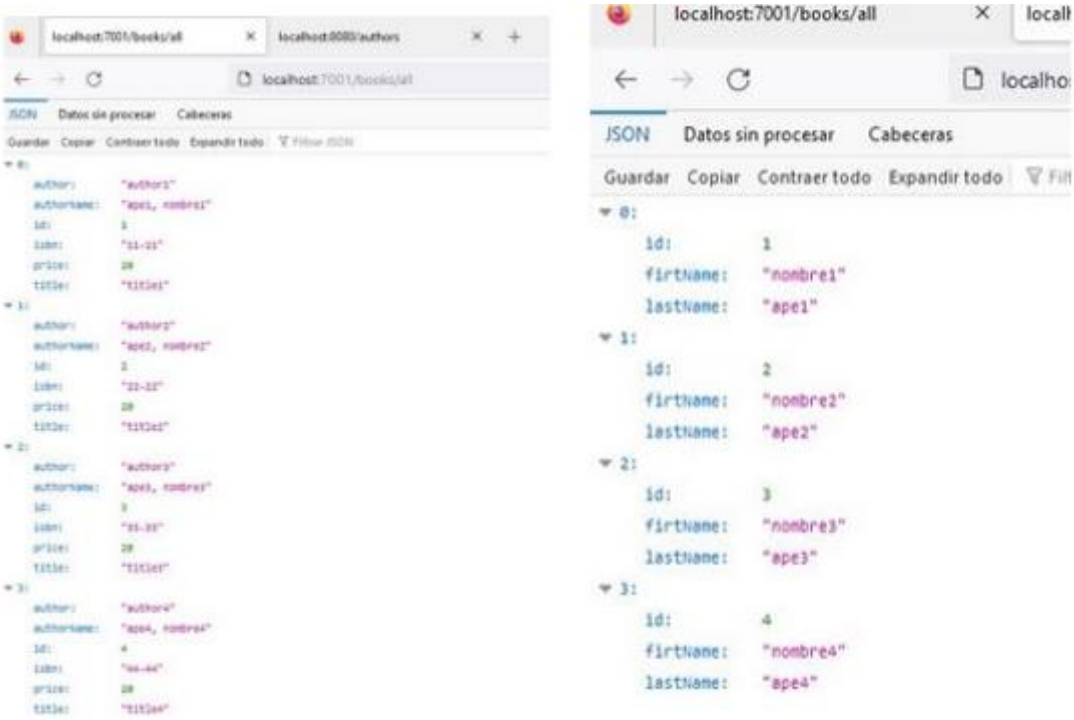

Figura 12. Ejecución del programa, books, authors.

10. Conclusiones

- Tolerancia a Fallos y Alta Disponibilidad: La incorporación de técnicas de tolerancia a fallos ha fortalecido la fiabilidad del sistema distribuido. La redundancia de datos y la detección temprana de fallos han asegurado que el sistema pueda mantenerse operativo incluso en situaciones adversas, garantizando una alta disponibilidad y evitando la pérdida de datos críticos.
- Facilidad de Mantenimiento y Escalado: La arquitectura de microservicios utilizada en el sistema ha facilitado el mantenimiento y la escalabilidad del proyecto. El modularidad de los microservicios ha permitido realizar actualizaciones y cambios de manera independiente, sin afectar al resto del sistema. Esto ha facilitado el mantenimiento y ha acelerado el desarrollo de nuevas funcionalidades o la incorporación de más nodos para atender un crecimiento futuro de la biblioteca.