

UNIVERSIDAD CENTRAL DEL ECUADOR

FACULTAD DE CIENCIAS APLICADAS

INGENIERÍA EN SISTEMAS DE INFORMACIÓN



PROGRAMACIÓN DISTRIBUIDA

DOCENTE: ING. DIEGO PINTO

SEMESTRE: NOVENO

ESTUDIANTE:

- BRYAN CHOLANGO
- ELVIS HERRERA

FECHA: 24 AGOSTO 2023

MANUAL DE USUARIO

2023 - 2023

Contenido

1.	Introducción	3
2.	Objetivo	3
3.	Público Destinatario	3
4.	Requisitos del Sistema	3
4.1.	Requisitos de Hardware	3
4.2.	Requisitos de Software.....	4
4.3.	Herramientas Adicionales	4
4.4.	Frameworks.....	4
5.	Estructura del Manual.....	4
5.1.	Configuración de la Base de Datos.....	4
5.2.	Configuración de las Aplicaciones.....	6
5.2.1.	Configuración app-authors	8
5.2.2.	Configuración app-books	11
5.2.3.	Configuración app-books consume app-authors	16
5.3.	Ejecución de las Aplicaciones.....	18
5.3.1.	Ejecución app-authors.	18
5.3.2.	Ejecución app-books	19
5.4.	Comunicación Aplicaciones mediante Servicios REST	20
5.4.1.	Comunicación app-authors	20
5.4.2.	Comunicación app-books.....	22
5.4.3.	Comunicación app-books consume app-authors	24
5.5.	Tolerancia a Fallos de Aplicación.....	25
5.5.1.	Tolerancia app-books	25

1. Introducción

Te damos la bienvenida al Manual de Usuario para las aplicaciones app-books y app-authors. Este manual está diseñado para proporcionarte una guía detallada sobre cómo probar y comprender el funcionamiento de estos dos servicios interconectados. Tanto app-books como app-authors son aplicaciones desarrolladas en Java utilizando Gradle como sistema de construcción. app-books está construida con Helidon, mientras que app-authors utiliza el framework Quarkus. Ambas aplicaciones son sistemas distribuidos que se comunican entre sí a través de servicios REST (Representational State Transfer) y están respaldadas por una base de datos PostgreSQL. Cabe resaltar que app-books incluye implementaciones de tolerancia a fallos (fault tolerance) para garantizar la robustez del sistema.

2. Objetivo

El propósito central de este manual es brindarte un profundo entendimiento de cómo interactuar con las aplicaciones app-books y app-authors, así como llevar a cabo pruebas efectivas utilizando las capacidades de servicios REST. A lo largo de esta guía, aprenderás a configurar y ejecutar ambas aplicaciones, a enviar solicitudes REST, interpretar respuestas y aprovechar la interacción fluida entre app-aooks y app-authors.

3. Público Destinatario

Este manual está dirigido a desarrolladores, probadores y cualquier individuo interesado en adquirir conocimientos sobre aplicaciones basadas en servicios REST y en la interacción de sistemas distribuidos. Se asume que los lectores cuentan con conocimientos básicos de programación, así como una comprensión general de conceptos relacionados con servicios web y APIs REST.

4. Requisitos del Sistema

Para poder ejecutar correctamente las aplicaciones App-Books y App-Authors, asegúrate de cumplir con los siguientes requisitos en tu sistema:

4.1. Requisitos de Hardware

Procesador	Intel Core I5 o superior
Memoria RAM	4GB o más
Espacio en Disco	Al menos 200 Mb de espacio libre
Navegador Web	Google Chrome/Firefox/Edge

4.2. Requisitos de Software

Sistema Operativo	Windows 10 recomendado
Java Development Kit (JDK)	Versión 17 o 17.0.6
Base de Datos	PostgreSQL 14 instalado y configurado en el sistema.
Administrador de BD	pgAdmin 4

4.3. Herramientas Adicionales

Gradle	Versión 7.2
Software editor de código	IntelliJ IDEA 2022.3.2
Comunicación	REST
Navegador Web	Google Chrome/Firefox/Edge
Cliente API	Postman

4.4. Frameworks

Para app-authors	Quarkus 2.16.1.Final (dependencias y configuraciones necesarias)
Para app-books	Helidon 3.1.0 (dependencias y configuraciones necesarias)

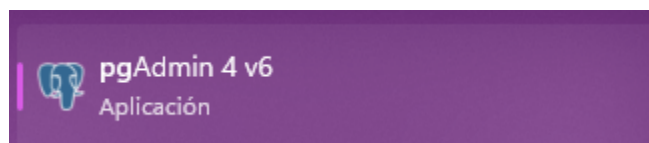
5. Estructura del Manual

5.1. Configuración de la Base de Datos

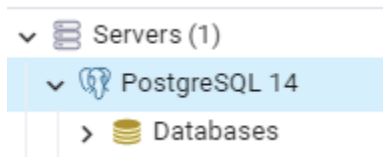
El paso inicial consiste en establecer nuestra base de datos, en caso de que aún no exista. El nombre de esta base de datos deberá ser "**distribuida**". No es necesario crear ninguna tabla, ya que la aplicación se encargará de generarlas de manera automática en caso de no encontrarlas.

Instrucciones:

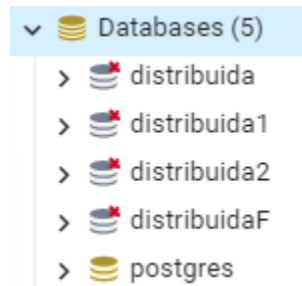
1. Accedemos a Administrador de Base de Datos:



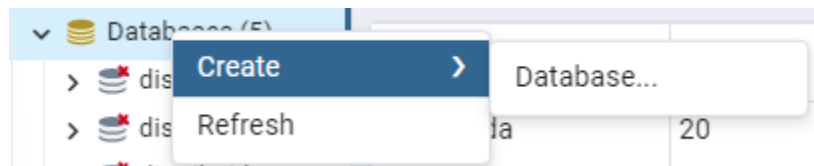
2. Una vez iniciada la aplicación nos dirigimos a Servers y después PostgreSQL 14



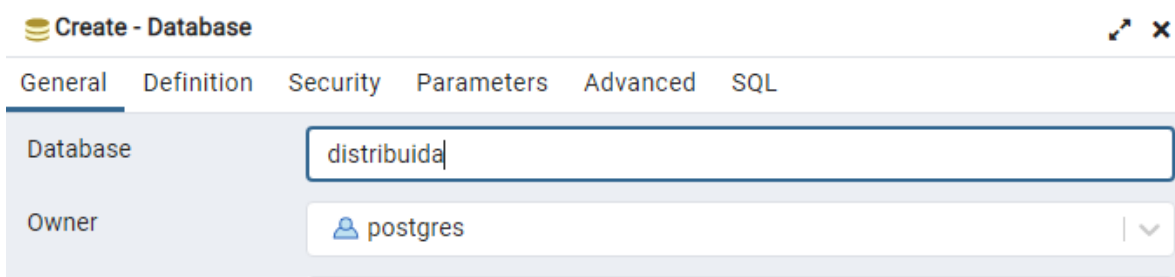
- Verificamos que en la Opción Databases existas la base de datos con el nombre **“distribuida”** caso contrario procedemos a crear la misma.



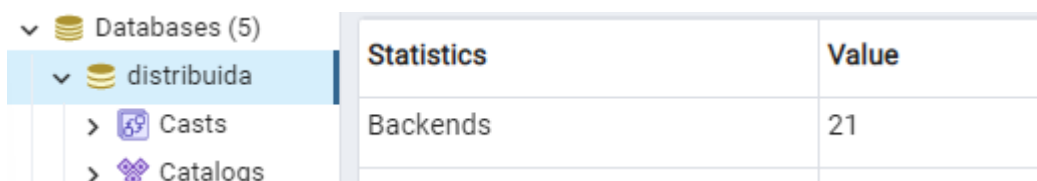
- Si no existe la base de datos **“distribuida”** vamos a crear la misma dando clic derecho encima de la opción Databases\Create\Database...



- Se nos abrirá la siguiente pantalla y en el cuadro frente a Database escribimos el nombre de nuestra base de datos la cual se llamará **“distribuida”** como se muestra a continuación:





- Una vez demos clic en el botón sabe verificamos que nuestra base ya se encuentre creada como se muestra a continuación:



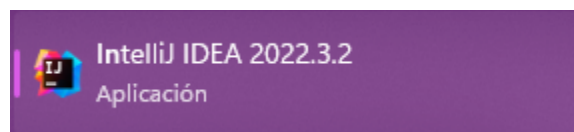
7. Ahora si ya tenemos creada nuestra base de Datos y lista para que app-books y app-authors la utilicen.

5.2. Configuración de las Aplicaciones

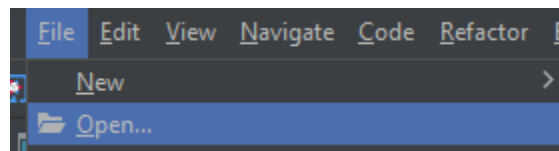
- Nuestro sistema distribuido cuenta con 2 carpetas llamadas app-books y app-authors.

 app-authors	23/8/2023 12:00	Carpeta de archivo
 app-books	23/8/2023 12:00	Carpeta de archivo

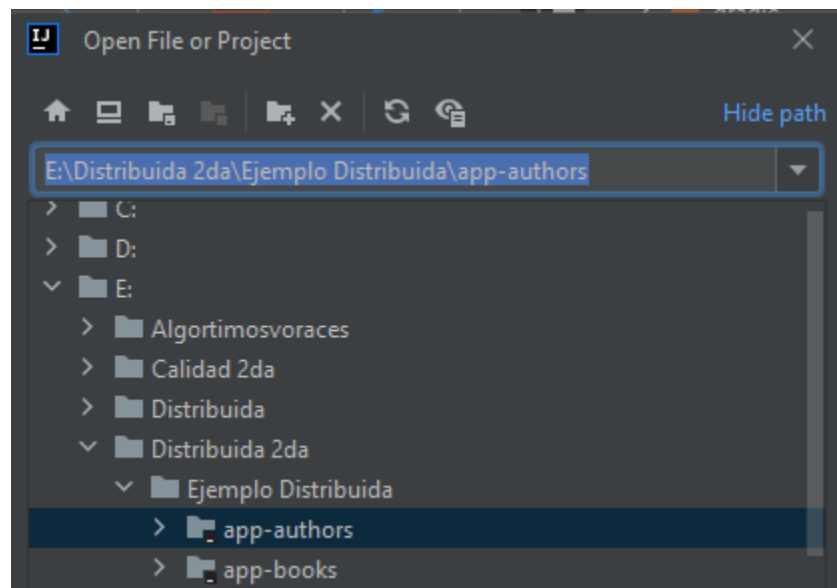
- Las cuales procederemos a abrir mediante el editor de código IntelliJ IDEA 2022.3.2



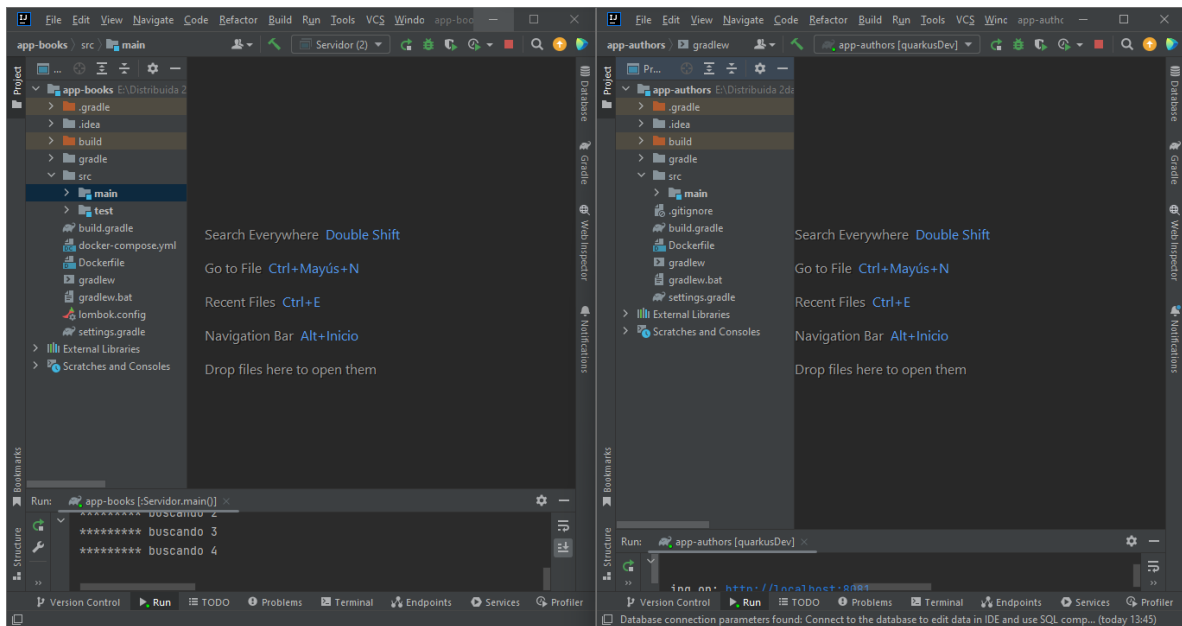
- Damos clic en **File\Open...**



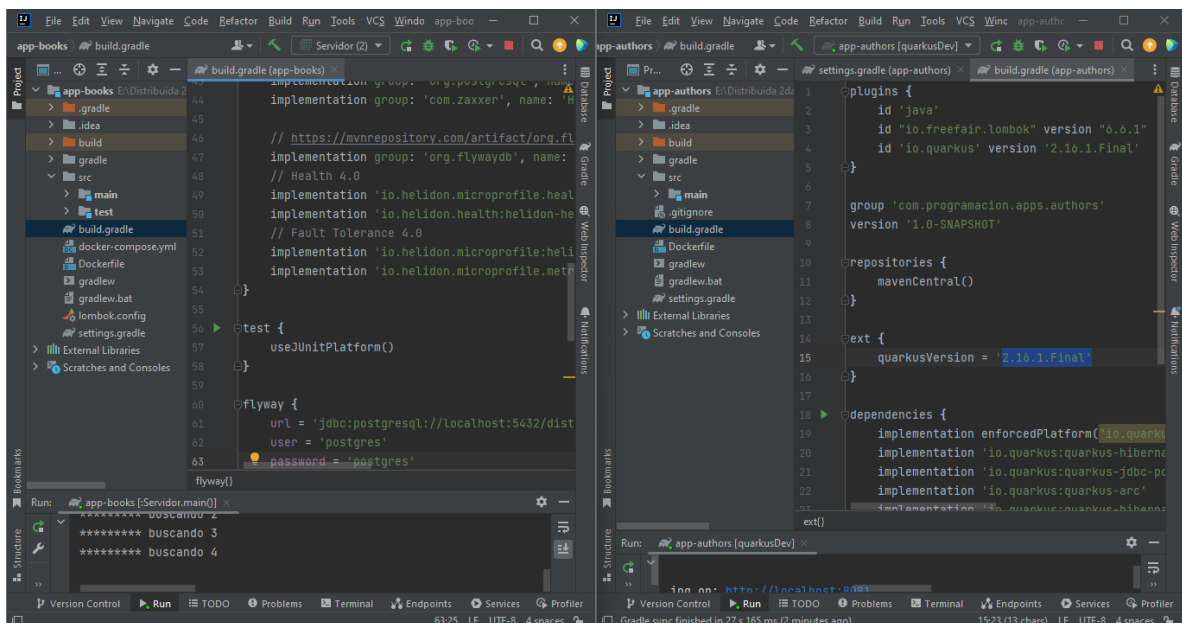
- Se presentará una ventana en la que deberemos buscar la ubicación de las dos carpetas de aplicaciones, tal como se ilustra a continuación:



- Una vez abierta nuestras dos aplicaciones tendremos estas dos interfaces:

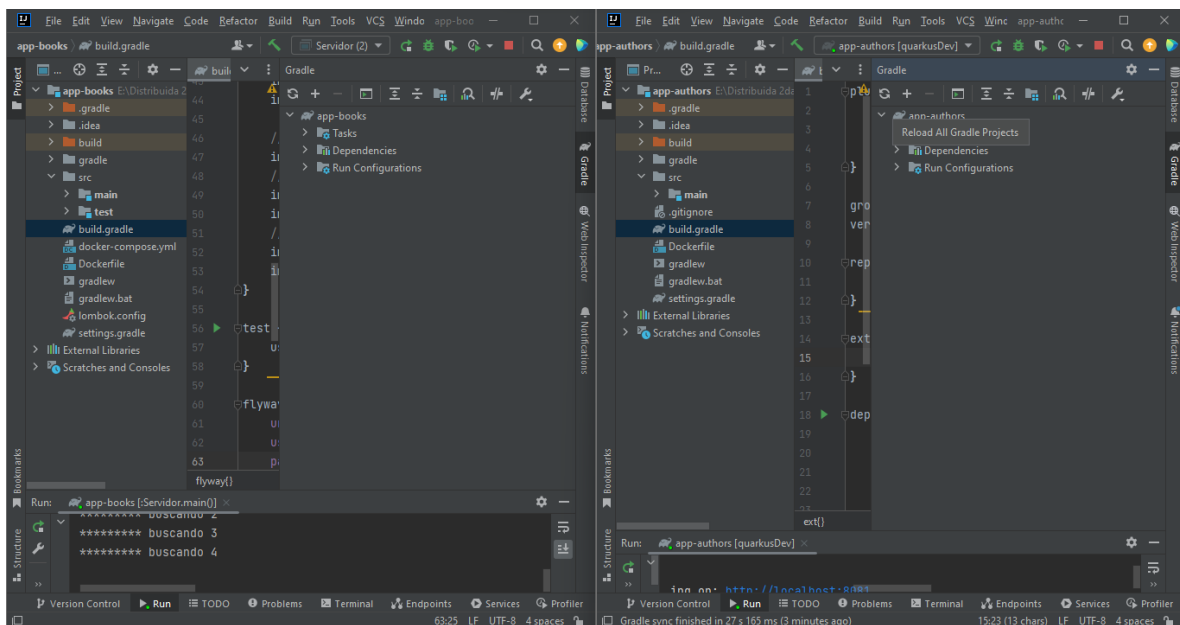


- En esta etapa, el inicio de ambas aplicaciones podría llevar un poco más de tiempo, ya que las dependencias necesarias se descargarán automáticamente, si esto no sucede debemos dirigirnos al archivo **build.grade** de la app-books y app-authors este archivo es el que contiene las dependencias.



- Ahora, es el momento de actualizar las dependencias. Primero, haz clic en el ícono del elefante ubicado en el panel derecho, el cual lleva el nombre "**Gradle**". Luego, selecciona el

ícono "Reload All Gradle Projects" para proceder a descargar manualmente las dependencias necesarias.

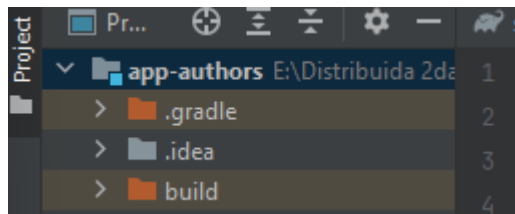


- Ahora si procederemos a configurar cada aplicación por separado.

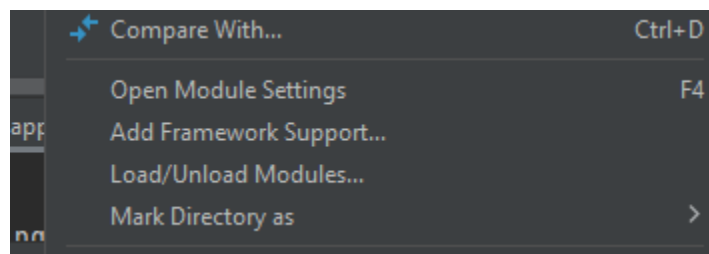
5.2.1. Configuración app-authors

5.2.1.1. Configuración SDK

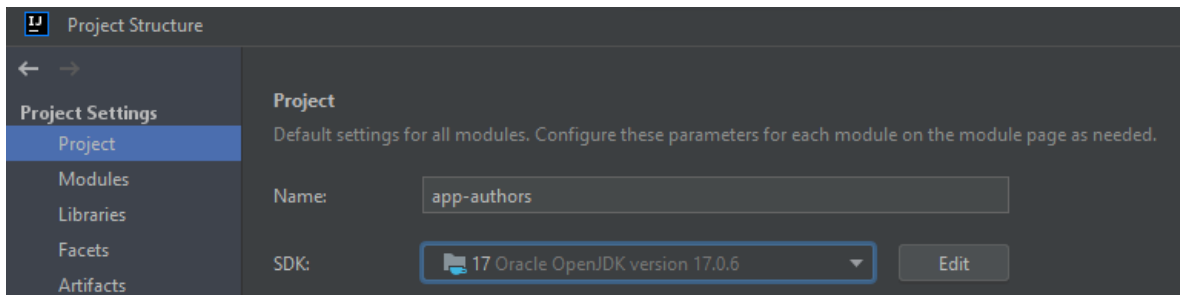
- Comencemos por configurar el JDK correcto, que debe ser la versión 17 o 17.0.6. Dirígete a la sección izquierda de la ventana de tu programa y realiza clic derecho sobre la carpeta "app-authors".



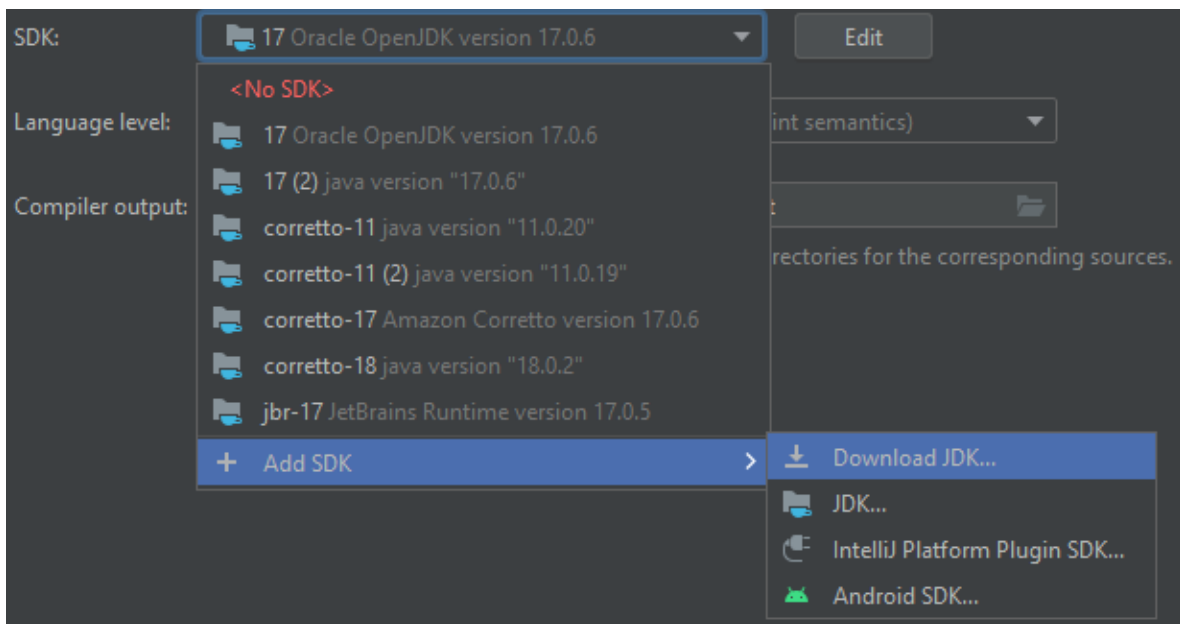
- Nos aparecerá un cuadro con varias opciones en donde escogeremos la opción "Open Module Settings".



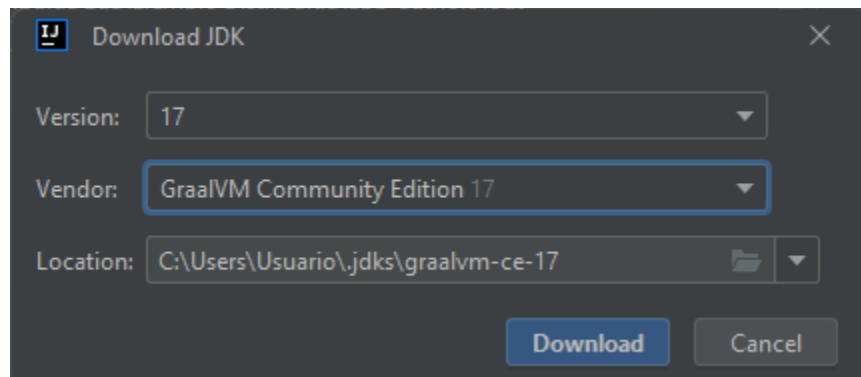
- Se nos abrirá una ventana de dialogo en donde verificaremos que nuestro SDK sea el correcto.



- Si el JDK no es el correcto, debemos agregar o descargar la versión correcta para garantizar el funcionamiento adecuado de nuestra aplicación, como se muestra a continuación:



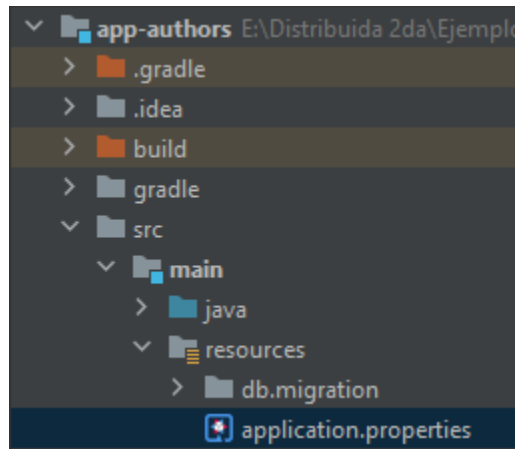
- Procedemos a la descarga del SDK correcto, lo establecemos como predeterminado y guardamos los cambios.



- Finalmente, nuestra aplicación app-authors estará configurada correctamente su SDK para su correcto funcionamiento.

5.2.1.2. Configuración Conexión a la Base de datos

- Para configurar la conexión a la base de datos debemos primero irnos al archivo ubicado en **app-authors\src\main\resources\application.properties** como se muestra a continuación:



- Una vez que hayamos abierto el archivo "**application.properties**", se presentará la siguiente pantalla en la que debemos configurar el nombre de usuario, la contraseña de PostgreSQL y el puerto en el que está activo PostgreSQL.

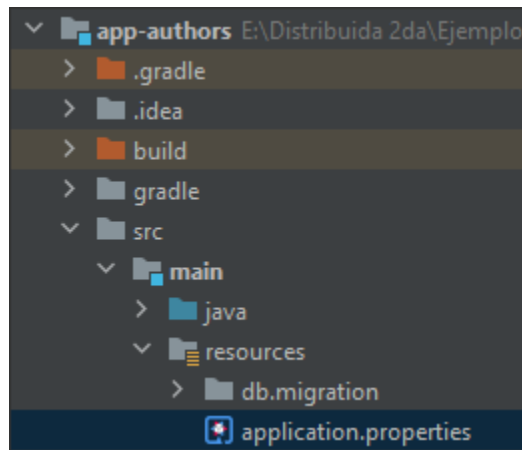
```
application.properties x
1  quarkus.hibernate-orm.database.generation.create-schemas=false
2
3  quarkus.datasource.db-kind=postgresql
4  quarkus.datasource.username=postgres
5  quarkus.datasource.password=postgres
6  quarkus.datasource.jdbc.url=jdbc:postgresql://localhost:5432/distribuida
7  quarkus.http.port=8081
```

quarkus.datasource.username	Aquí se escribe el nombre del usuario con el que se tiene acceso a PostgreSQL
quarkus.datasource.password	Aquí se escribe la contraseña del usuario con el que se tiene acceso a PostgreSQL
quarkus.datasource.jdbc.url	jdbc:postgresql://localhost:puerto_postgreSQL/nombre_BD en " puerto_postgreSQL " se ubica el puerto en el que está activo PostgreSQL y en " nombre_BD " va el nombre de la base de datos que vamos a utilizar que para nuestro caso es " distribuida "

5.2.1.3. Configuración Puerto del Servicio app-authors

La configuración del puerto en el cual se levantará el servicio se realiza en caso de que el puerto 8081 esté ocupado por otra aplicación.

- Antes de continuar, es importante verificar si otras aplicaciones están utilizando el puerto 8081. Si ese es el caso, deberemos cambiar el puerto de manera que el servicio pueda iniciarse sin conflictos.
- Para poder cambiar el puerto donde iniciara el servicio app-authors nos dirigimos al archivo ubicado en **app-authors\src\main\resources\application.properties** como se muestra a continuación:



- Una vez que hayamos abierto el archivo "**application.properties**", se presentará la siguiente pantalla en la que debemos configurar el puerto en donde queremos que nuestro servicio app-authors este activo o corriendo.

```

1  quarkus.hibernate-orm.database.generation.create-schemas=false
2
3  quarkus.datasource.db-kind=postgresql
4  quarkus.datasource.username=postgres
5  quarkus.datasource.password=postgres
6  quarkus.datasource.jdbc.url=jdbc:postgresql://localhost:5432/distribuida
7  quarkus.http.port=8081

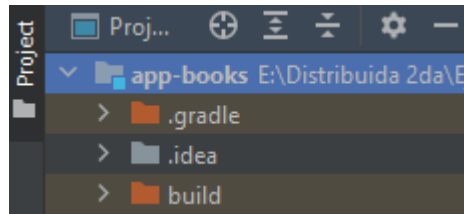
```

quarkus.http.port	Aquí escribiremos el puerto en donde queremos que nuestro servicio se encuentre activo (se debe poner un puerto el cual este libre) en este caso el puerto por defecto es el 8081 .
--------------------------	--

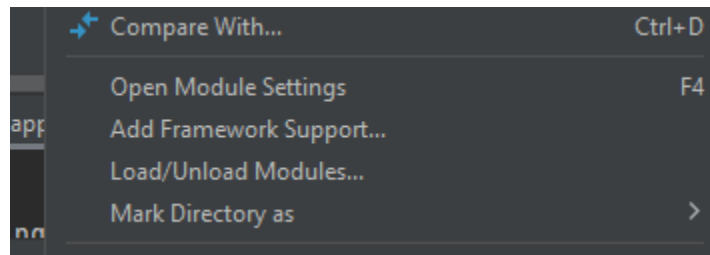
5.2.2. Configuración app-books

5.2.2.1. Configuración SDK

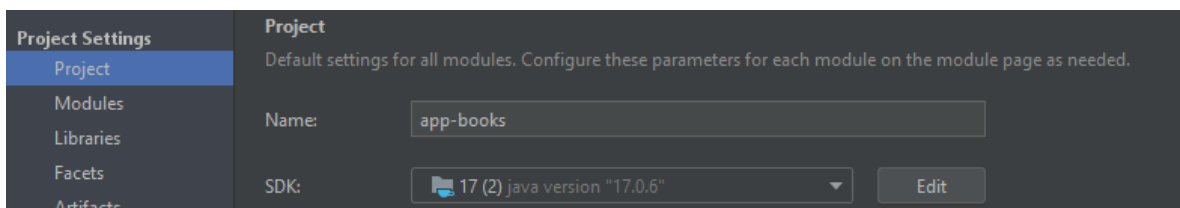
- Comencemos por configurar el JDK correcto, que debe ser la versión 17 o 17.0.6. Dirígete a la sección izquierda de la ventana de tu programa y realiza clic derecho sobre la carpeta "**app-books**".



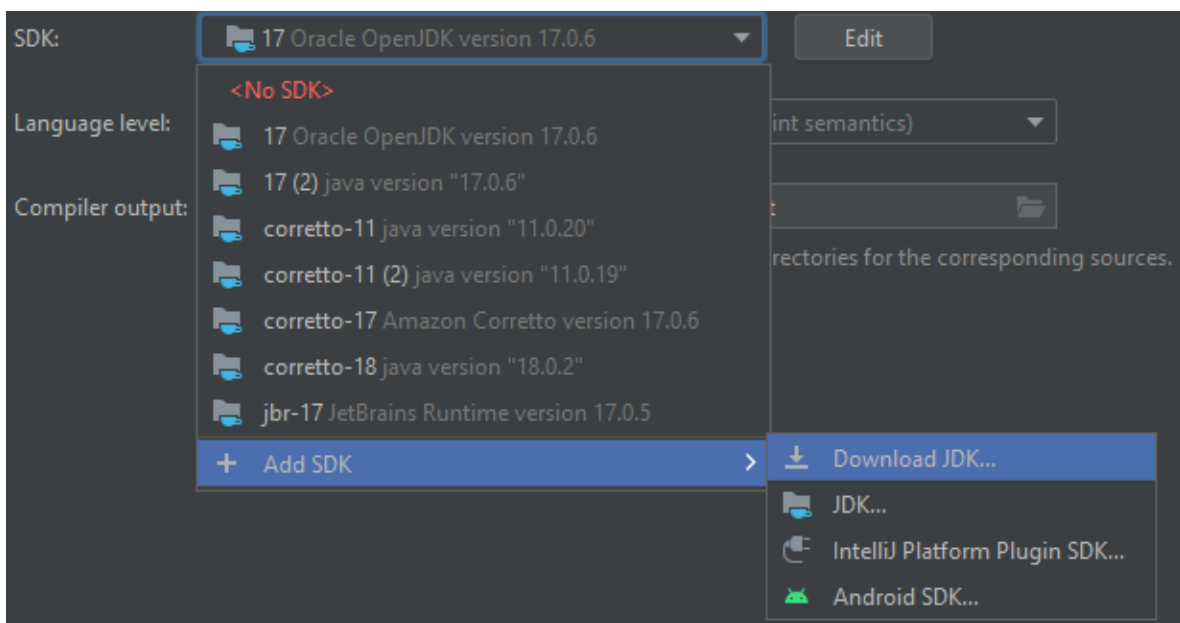
- Nos aparecerá un cuadro con varias opciones en donde escogeremos la opción **“Open Module Settings”**.



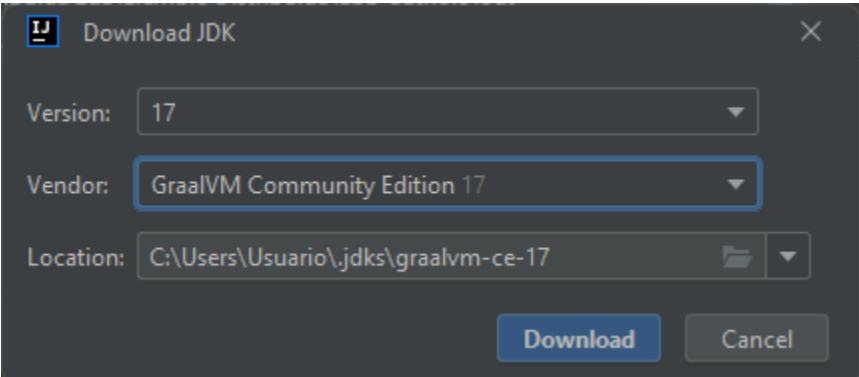
- Se nos abrirá una ventana de dialogo en donde verificaremos que nuestro SDK sea el correcto.



- Si el JDK no es el correcto, debemos agregar o descargar la versión correcta para garantizar el funcionamiento adecuado de nuestra aplicación, como se muestra a continuación:



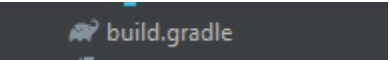
- Procedemos a la descarga del SDK correcto, lo establecemos como predeterminado y guardamos los cambios.



- Finalmente, nuestra aplicación app-books estará configurada correctamente su SDK para su correcto funcionamiento.

5.2.2.2. Configuración Conexión a la Base de Datos

- Para configurar la conexión flyway a la base de datos debemos primero irnos al archivo “**build.gradle**” en la raíz de la app-books como se muestra a continuación:

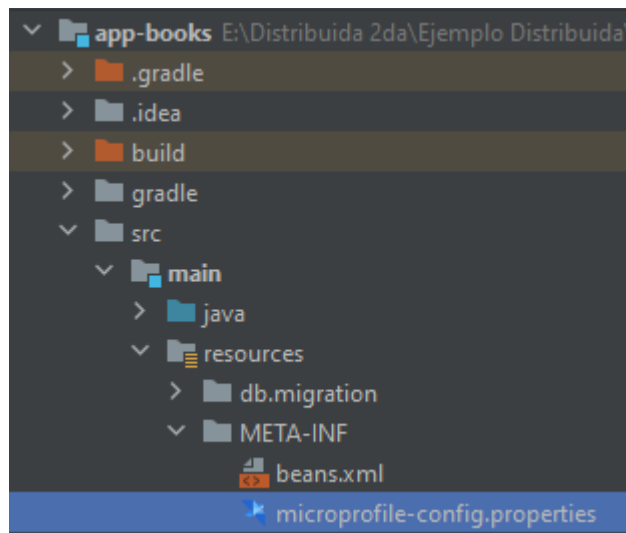


- Una vez que hayamos abierto el archivo “**build.gradle**”, se presentará la siguiente pantalla y nos dirigimos hasta flyway en la que debemos configurar el nombre de usuario, la contraseña de PostgreSQL y el puerto en el que está activo PostgreSQL.

```
60 flyway {
61     url = 'jdbc:postgresql://localhost:5432/distribuida'
62     user = 'postgres'
63     password = 'postgres'
64     schemas = ['public']
65 }
```

url	jdbc:postgresql://localhost:puerto_postgreSQL/nombre_BD en “ puerto_postgreSQL ” se ubica el puerto en el que está activo PostgreSQL y en “ nombre_BD ” va el nombre de la base de datos que vamos a utilizar que para nuestro caso es “ distribuida ”
user	Aquí se escribe el nombre del usuario con el que se tiene acceso a PostgreSQL
password	Aquí se escribe la contraseña del usuario con el que se tiene acceso a PostgreSQL
schemas	Aquí se escribe el permiso de acceso a las tablas de la base de datos “ distribuida ”.

- Para configurar la conexión a la base de datos debemos primero irnos al archivo ubicado en **app-books\src\main\resources\META-INF\microprofile-config.properties** como se muestra a continuación:



- Una vez que hayamos abierto el archivo "**microprofile-config.properties**", se presentará la siguiente pantalla en la que debemos configurar el nombre de usuario, la contraseña de PostgreSQL y el puerto en el que está activo PostgreSQL.

```

microprofile-config.properties x
1  # Configuración de la base de datos
2  db.connection.url=jdbc:postgresql://127.0.0.1:5432/distribuida
3  db.connection.username=postgres
4  db.connection.password=postgres
5  mp.config.profile=
6
7  ## servicios REST a los que se conecta
8  author.url=http://localhost:8081
9  author/mp-rest/url=${author.url}

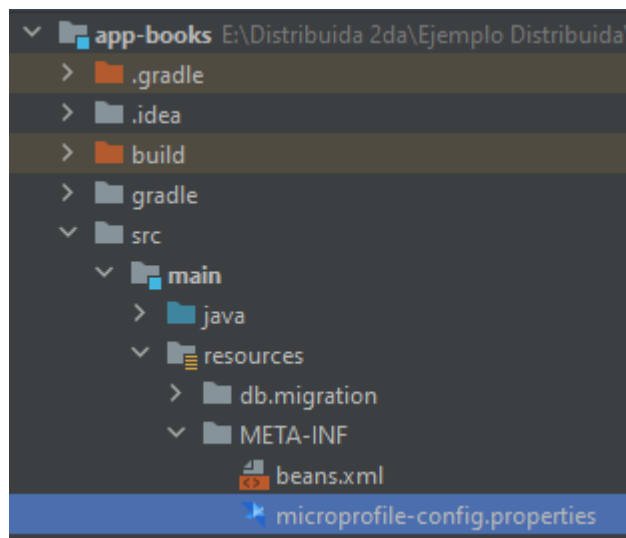
```

db.connection.url	jdbc:postgresql:// 127.0.0.1:puerto_postgreSQL/nombre_BD en " puerto_postgreSQL " se ubica el puerto en el que está activo PostgreSQL y en " nombre_BD " va el nombre de la base de datos que vamos a utilizar que para nuestro caso es " distribuida "
db.connection.username	Aquí se escribe el nombre del usuario con el que se tiene acceso a PostgreSQL
db.connection.password	Aquí se escribe la contraseña del usuario con el que se tiene acceso a PostgreSQL

5.2.2.3. Configuración Puerto del Servicio app-books

La configuración del puerto en el cual se levantará el servicio se realiza en caso de que el puerto 7001 esté ocupado por otra aplicación.

- Antes de continuar, es importante verificar si otras aplicaciones están utilizando el puerto 7001. Si ese es el caso, deberemos cambiar el puerto de manera que el servicio pueda iniciarse sin conflictos.
- Para poder cambiar el puerto donde iniciara el servicio app-books nos dirigimos al archivo ubicado en **app-books\src\main\resources\META-INF\microprofile-config.properties** como se muestra a continuación:



- Una vez que hayamos abierto el archivo " **microprofile-config.properties** ", se presentará la siguiente pantalla en la que debemos configurar el puerto en donde queremos que nuestro servicio app-authors este activo o corriendo.

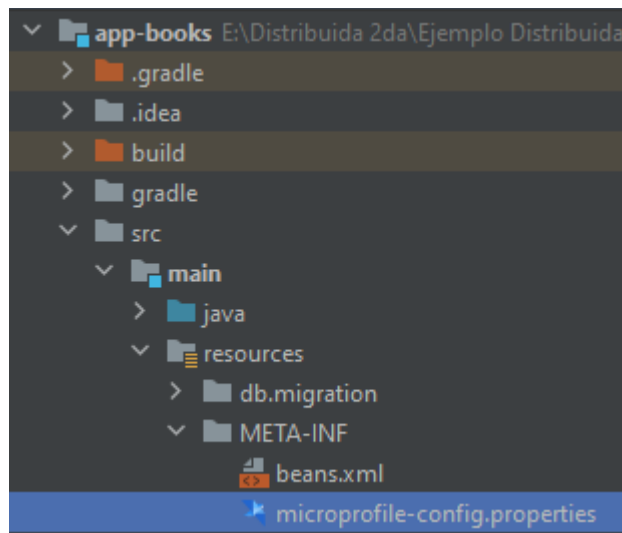
```
microprofile-config.properties x
1  # Configuración de la base de datos
2  db.connection.url=jdbc:postgresql://127.0.0.1:5432/distribuida
3  db.connection.username=postgres
4  db.connection.password=postgres
5  server.port=7001
```

server.port	Aquí escribiremos el puerto en donde queremos que nuestro servicio se encuentre activo (se debe poner un puerto el cual este libre) en este caso el puerto por defecto es el 7001 .
--------------------	--

5.2.3. Configuración app-books consume app-authors

Para configurar correctamente app-books y permitir que consuma los servicios de app-authors, es fundamental tener en cuenta que app-books realiza solicitudes a una dirección específica. Ambos son servicios independientes que se comunican a través de estas solicitudes.

- Primero debemos configurar el servicio REST a los que se va a conectar nuestra app-books, para esto nos dirigimos al siguiente archivo ubicado en **app-books\src\main\resources\META-INF\microprofile-config.properties** como se muestra a continuación:

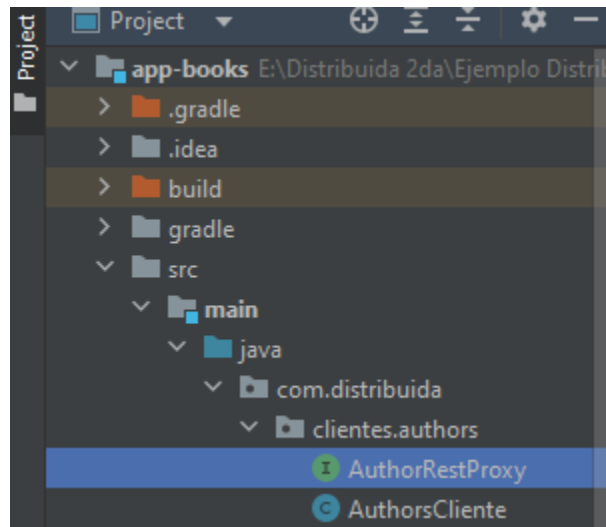


- Una vez que hayamos abierto el archivo " **microprofile-config.properties** ", se presentará la siguiente pantalla en la que debemos configurar los servicios REST a donde nuestra app-books se va a conectar.

```
7 mp.config.profile=  
8 ## servicios REST a los que se conecta  
9 author.url=http://localhost:8081  
0 author/mp-rest/url=${author.url}
```

author.url	http://localhost:puerto_servicio_app-authors en "puerto_servicio_app-authors" se ubica el puerto en el que está nuestro servicio app-authors está corriendo o está activo para nuestro caso el puerto determinado es el 8081 .
author/mp-rest/url	\${nombre_servicio.url} en "nombre_servicio" se ubica el nombre que se le quiere dar a este servicio.

- Ahora nos dirigimos al siguiente archivo ubicado en **app-books\src\main\java\com\distribuida\clientes\authors\AuthorRestProxy** como se muestra a continuación:



- Se mostrará la siguiente pantalla y procederemos a configurar la conexión a los servicios REST de app-authors.

```

9  @Path("/authors")
10  @Produces(MediaType.APPLICATION_JSON)
11  @Consumes(MediaType.APPLICATION_JSON)
12  @RegisterRestClient(baseUrl="http://localhost:8081")
13  // @RegisterRestClient(configKey = "author")

```

baseUrl	http://localhost:puerto_servicio_app-authors en “ puerto_servicio_app-authors ” se ubica el puerto en el que está nuestro servicio app-authors está corriendo o está activo para nuestro caso el puerto determinado es el 8081 .
configKey	“nombre_servicio” en “ nombre_servicio ” se ubica el nombre de referencia que se le dio a este servicio.

- Para esto existen dos maneras de consumir los servicios siempre y cuando la configuración se haya hecho de manera correcta en el archivo “**microprofile-config.properties**”.
- La primera manera es des comentar `@RegisterRestClient(baseUrl="http://localhost:8081")` y comentar `@RegisterRestClient(configKey = "author")` como se muestra a continuación:

```

12  @RegisterRestClient(baseUrl="http://localhost:8081")
13  // @RegisterRestClient(configKey = "author")

```

- Y la segunda manera es des comentar `@RegisterRestClient(configKey = "author")` y comentar `@RegisterRestClient(baseUrl="http://localhost:8081")` como se muestra a continuación:

```

12  // @RegisterRestClient(baseUrl="http://localhost:8081")
13  @RegisterRestClient(configKey = "author")

```

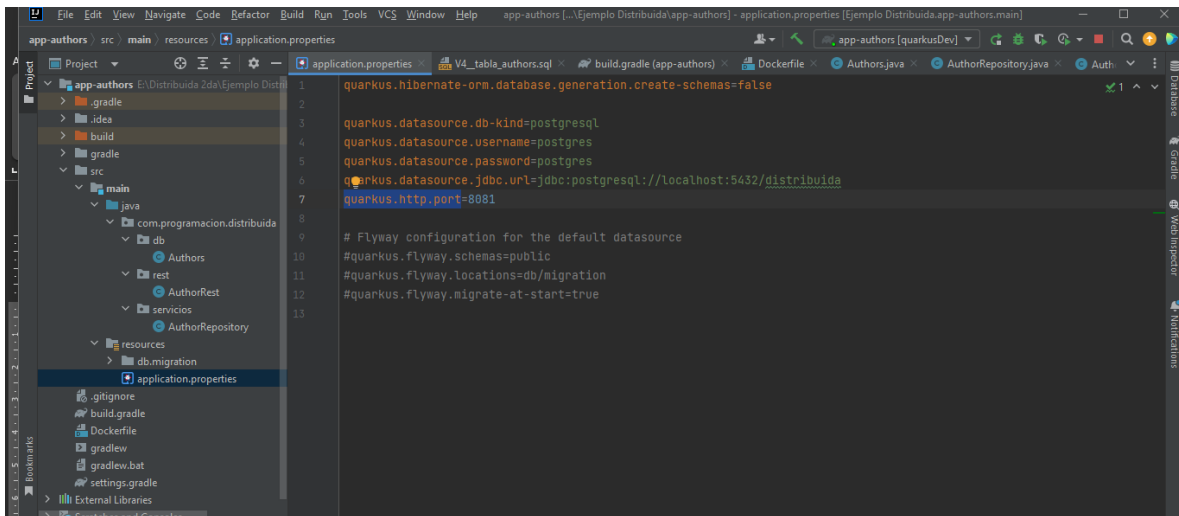
- Es importante destacar que la forma predeterminada en que consume los datos de app-authors la aplicación app-books es la primera manera.

5.3. Ejecución de las Aplicaciones

5.3.1. Ejecución app-authors.

Para ejecutar o iniciar el servicio de nuestra aplicación "**app-authors**", es imprescindible tener previamente configurado el servicio tal como se explicó anteriormente.

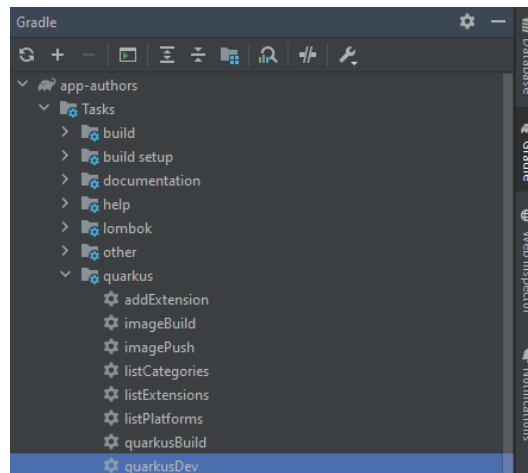
- Una vez abierto la pantalla del IntelliJ IDEA donde se encuentra nuestra aplicación app-authors como se muestra a continuación:



- Tenemos dos maneras de iniciar nuestro servicio app-authors.

5.3.1.1. Primera Manera

- Primero, hacemos clic en el ícono del elefante ubicado en el panel derecho, el cual lleva el nombre "Gradle". Luego, seleccionamos **app-authors\Tasks\Quarkus** y damos clic en **quarkusDev** como se muestra a continuación:



Una vez le demos clic la aplicación app-authors se levantará en el puerto predeterminado <http://localhost:8081>.

5.3.1.2. Segunda Manera (mediante consola)

- Nos dirigimos a la parte inferior y damos clic en terminal en donde se mostrará esta pantalla.

- Verificamos que nuestro Path este direccionado a la aplicación app-authors.
- Si el Path de nuestra aplicación es el correcto en la terminal abierta debemos ingresar el siguiente comando `./gradlew quarkusDev` sin comillas para que nuestro servicio se levante en <http://localhost:8081> así como se ve a continuación:

5.3.2. Ejecución app-books

5.3.2.1. Primera Manera

- Primero debemos dirigirnos a la clase Servidor que se encuentra en la siguiente ubicación `app-books\src\main\java\com\distribuida\Servidor` y damos clic derecho encima de este:

- Damos clic en Run '**Servidor.main()**' y la aplicación app-books se levantará de manera correcta en <http://localhost:7001> como se muestra a continuación:

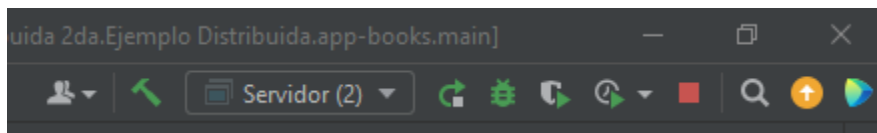
```

Run: app-books [Servidor.main()]
2023.08.24 17:57:37 INFORMATION com.zaxxer.hikari.pool.HikariPool Thread[main,5,main]: HikariPool-1 - Added connection org.postgresql.jdbc.PgConnection
2023.08.24 17:57:37 INFORMATION com.zaxxer.hikari.HikariDataSource Thread[main,5,main]: HikariPool-1 - Start completed.
2023.08.24 17:57:37 INFORMATION org.flywaydb.core.internal.database.base.BaseDatabaseType Thread[main,5,main]: Database: jdbc:postgresql://127.0.0.1:54
2023.08.24 17:57:38 INFORMATION org.flywaydb.core.internal.command.DbValidate Thread[main,5,main]: Successfully validated 4 migrations (execution time
2023.08.24 17:57:38 INFORMATION org.flywaydb.core.internal.command.DbMigrate Thread[main,5,main]: Current version of schema "public": 4
2023.08.24 17:57:38 INFORMATION org.flywaydb.core.internal.command.DbMigrate Thread[main,5,main]: Schema "public" is up to date. No migration necessary
2023.08.24 17:57:38 INFORMATION io.helidon.microprofile.server.ServerCdiExtension Thread[main,5,main]: Registering JAX-RS Application: HelidonMP
2023.08.24 17:57:38 INFORMATION io.helidon.webserver.NettyWebServer Thread[nioEventLoopGroup-2-1,10,main]: Channel '@default' started: [id: 0x28992d92,
2023.08.24 17:57:38 INFORMATION io.helidon.microprofile.server.ServerCdiExtension Thread[main,5,main]: Server started on http://localhost:7001 (and all
2023.08.24 17:57:38 INFORMATION io.helidon.common.HelidonFeatures Thread[features-thread,5,main]: Helidon MP 3.1.0 features: [CDI, Config, Db Client, F

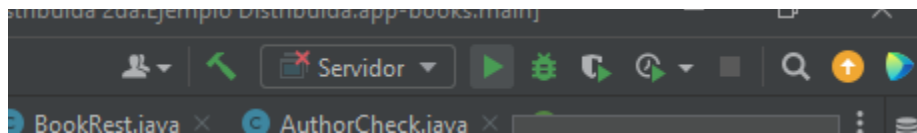
```

5.3.2.2. Segunda Manera

- Nos dirigimos a la parte superior derecha de la pantalla.



- Vemos que en el recuadro se encuentre escogido el nombre de la clase "**Servidor**" y damos clic en el botón verde de play para que nuestro servidor se levante



- Una vez demos clic nuestra aplicación app-books se levantará de manera correcta en <http://localhost:7001> como se muestra a continuación:

```

Run: app-books [Servidor.main()]
2023.08.24 18:02:24 INFORMATION com.zaxxer.hikari.pool.HikariPool Thread[main,5,main]: HikariPool-1 - Added connection org.postgresql.jdbc.PgConnection
2023.08.24 18:02:24 INFORMATION com.zaxxer.hikari.HikariDataSource Thread[main,5,main]: HikariPool-1 - Start completed.
2023.08.24 18:02:24 INFORMATION org.flywaydb.core.internal.database.base.BaseDatabaseType Thread[main,5,main]: Database: jdbc:postgresql://127.0.0.1:54
2023.08.24 18:02:24 INFORMATION org.flywaydb.core.internal.command.DbValidate Thread[main,5,main]: Successfully validated 4 migrations (execution time
2023.08.24 18:02:24 INFORMATION org.flywaydb.core.internal.command.DbMigrate Thread[main,5,main]: Current version of schema "public": 4
2023.08.24 18:02:24 INFORMATION org.flywaydb.core.internal.command.DbMigrate Thread[main,5,main]: Schema "public" is up to date. No migration necessary
2023.08.24 18:02:24 INFORMATION io.helidon.microprofile.server.ServerCdiExtension Thread[main,5,main]: Registering JAX-RS Application: HelidonMP
2023.08.24 18:02:25 INFORMATION io.helidon.webserver.NettyWebServer Thread[nioEventLoopGroup-2-1,10,main]: Channel '@default' started: [id: 0x66919c2b,
2023.08.24 18:02:25 INFORMATION io.helidon.microprofile.server.ServerCdiExtension Thread[main,5,main]: Server started on http://localhost:7001 (and all
2023.08.24 18:02:25 INFORMATION io.helidon.common.HelidonFeatures Thread[features-thread,5,main]: Helidon MP 3.1.0 features: [CDI, Config, Db Client, F

```

5.4. Comunicación Aplicaciones mediante Servicios REST

5.4.1. Comunicación app-authors

La aplicación "**app-authors**" establece comunicación con su base de datos a través de servicios REST, los cuales exploraremos en detalle a continuación:

Como dirección predeterminada (si no se ha configurado o ha hecho cambios a app-authors caso contrario se ubicará el número de puerto en donde se le configuro) tenemos:

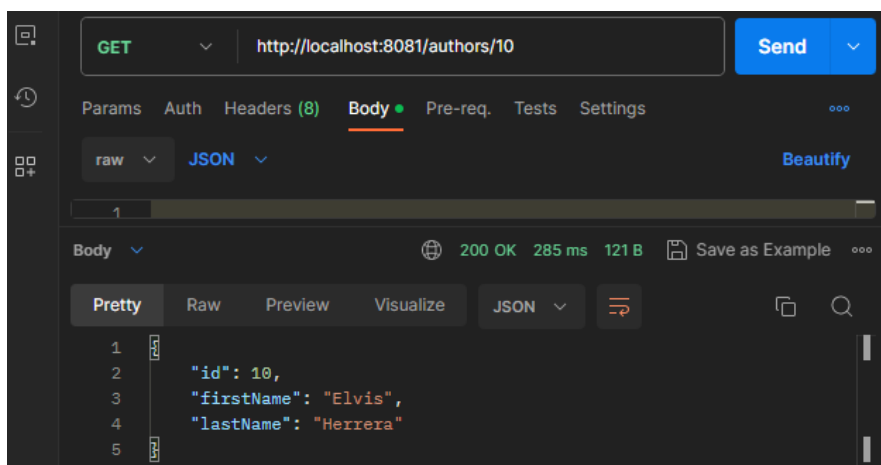
<http://localhost:8081>

Una vez que el servicio se encuentre levantado podemos acceder a sus servicios REST mediante los siguientes paths:

Método	Path	Función
GET	http://localhost:8081/authors/{id}	buscar un autor por ID
GET	http://localhost:8081/authors	listar todos los autores
POST	http://localhost:8081/authors	Insertar un autor
PUT/PATCH	http://localhost:8081/authors/{id}	Actualizar un autor
DELETE	http://localhost:8081/authors/{id}	Eliminar un autor

Servicios REST mediante aplicación POSTMAN

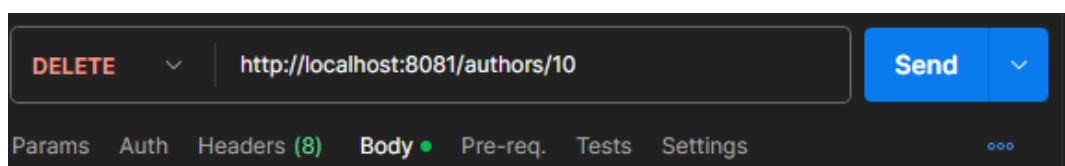
- Buscar



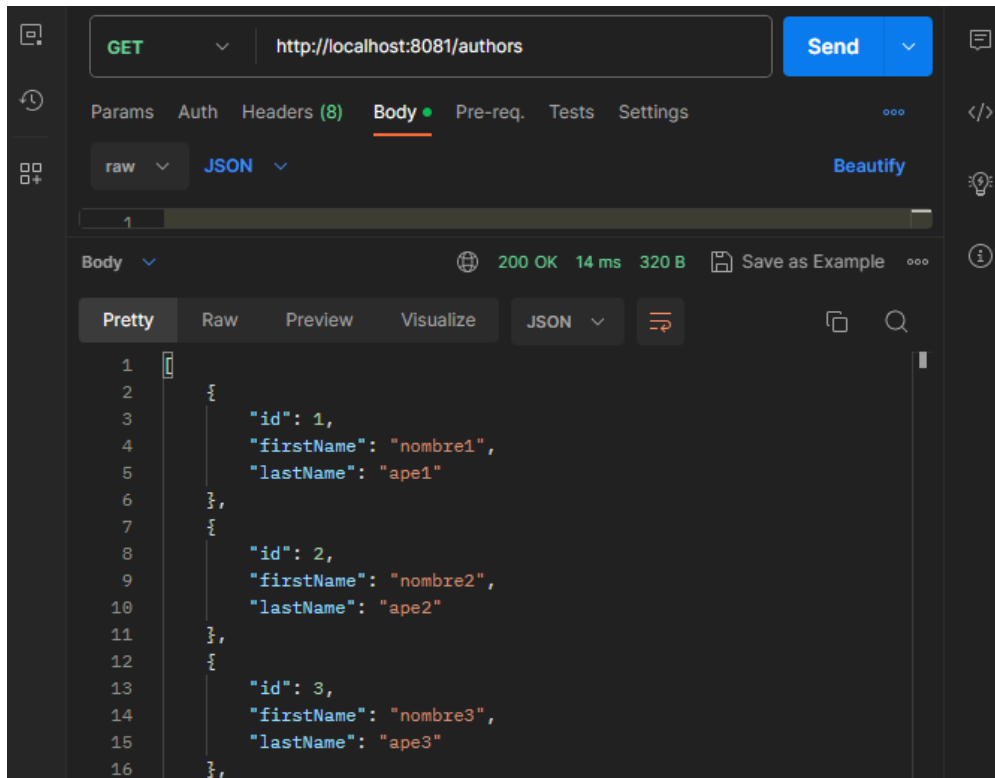
- Insertar



- Eliminar



- **Listar**



- **Actualizar**



5.4.2. Comunicación app-books

La aplicación "app-books" establece comunicación con su base de datos a través de servicios REST, los cuales exploraremos en detalle a continuación:

Como dirección predeterminada (si no se ha configurado o ha hecho cambios a app-books caso contrario se ubicará el número de puerto en donde se le configuro) tenemos:

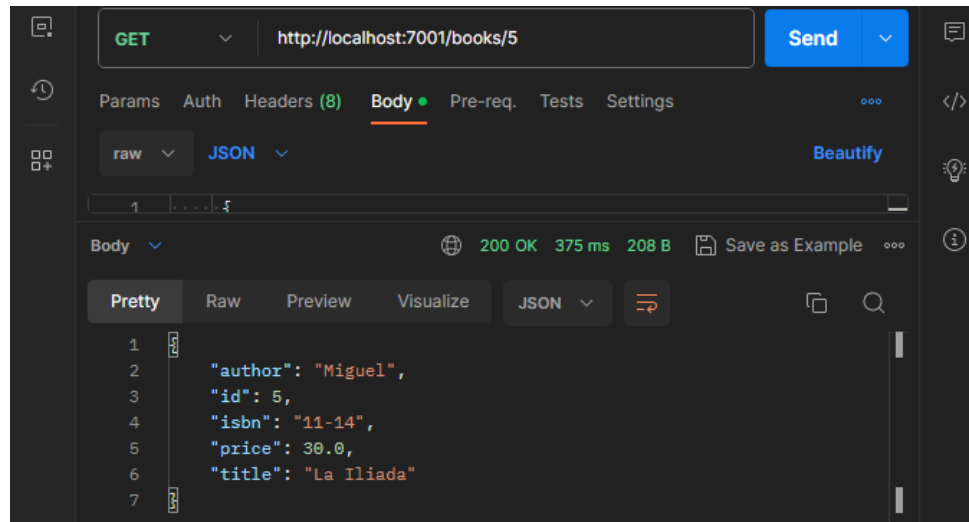
<http://localhost:7001>

Una vez que el servicio se encuentre levantado podemos acceder a sus servicios REST mediante los siguientes paths:

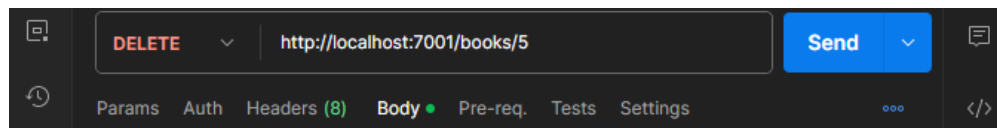
Método	Path	Función
GET	http://localhost:7001/books/{id}	buscar un libro por ID
GET	http://localhost:7001/books	listar todos los libros
POST	http://localhost:7001/books	Insertar un libro
PUT/PATCH	http://localhost:7001/books/{id}	Actualizar un libro
DELETE	http://localhost:7001/books/{id}	Eliminar un libro

Servicios REST mediante aplicación POSTMAN

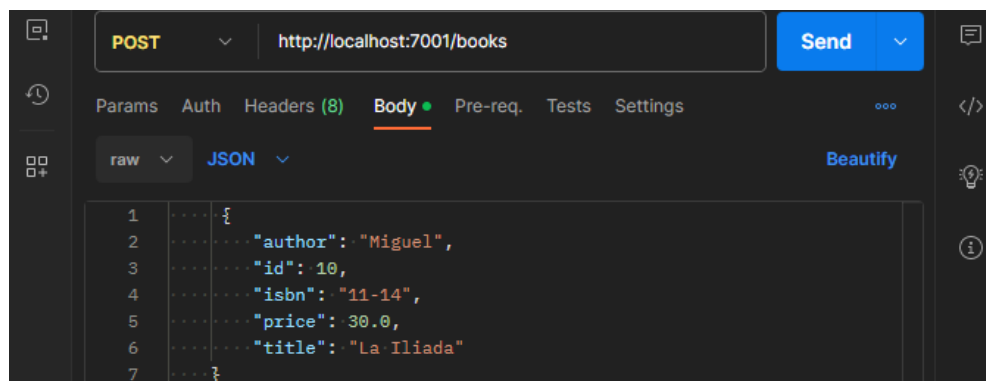
- Buscar



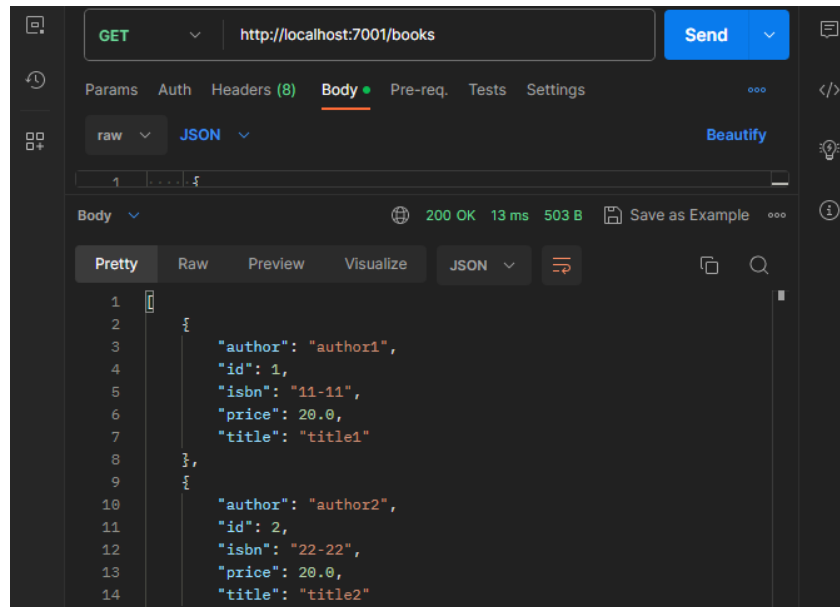
- Eliminar



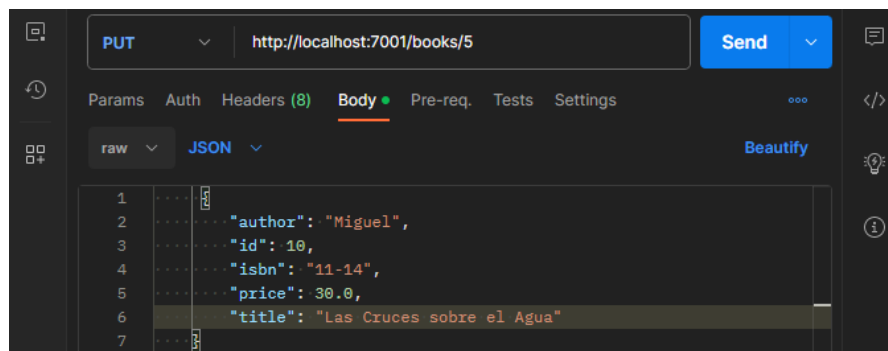
- Insertar



- Listar



- Actualizar



5.4.3. Comunicación app-books consume app-authors

La aplicación "app-books" consume a la aplicación "app-authors" través del servicio REST, el cual exploraremos en detalle a continuación:

Como dirección predeterminada (si no se ha configurado o ha hecho cambios a app-books caso contrario se ubicará el número de puerto en donde se le configuro) tenemos:

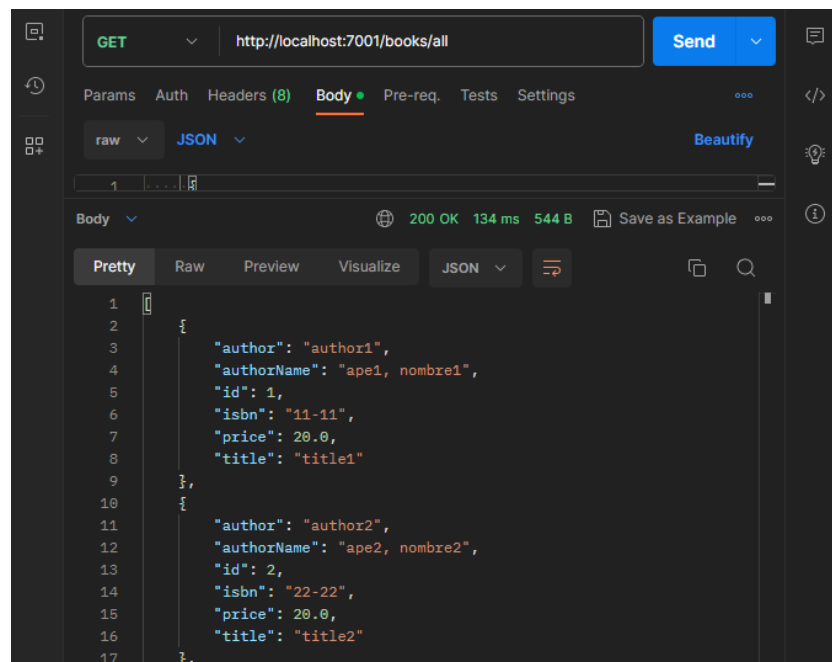
<http://localhost:7001>

Una vez que el servicio se encuentre levantado podemos acceder al servicio rest que consume a la aplicación "app-authors" mediante su siguiente Path:

Método	Path	Función
GET	http://localhost:7001/books/all	Lista todos los libros con el apellido y nombre de su respectivo autor.

Servicio REST mediante aplicación POSTMAN

- Listar Completo con nombres de Autores



5.5. Tolerancia a Fallos de Aplicación

5.5.1. Tolerancia app-books

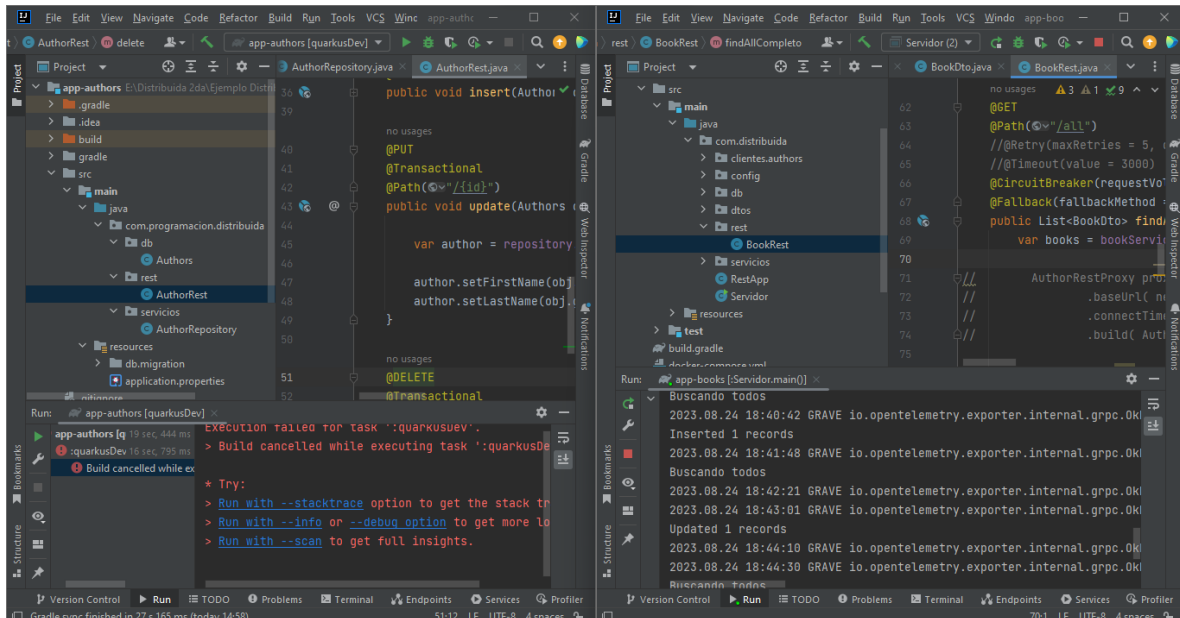
Es importante considerar que el servicio "app-books" consume el servicio "app-authors". Aunque el servicio "app-authors" pueda dejar de funcionar, la aplicación que lo consume no debería colapsar. Esto se alinea con una característica fundamental de los sistemas distribuidos, que es su capacidad de mantenerse operativos incluso si alguno de los componentes falla. A continuación, se presenta un ejemplo de cómo la aplicación "app-books" maneja esta tolerancia a fallos en caso de que "app-authors" esté fuera de servicio o experimente un fallo.

Si "app-authors" no está disponible o ha caído, se implementó 2 técnicas a la aplicación "app-books" técnicas de tolerancia a fallos para mantener su funcionamiento. Estas técnicas son:

- **Circuit Breaker:** Se implementa un circuit breaker para detectar la falta de respuesta del servicio "app-authors". Si se detectan múltiples intentos fallidos, el circuito se abre y las solicitudes posteriores a "app-authors" se bloquean temporalmente. Esto evita que la aplicación "app-books" realice intentos infructuosos y se sobrecargue.
- **Fallback:** Se establece un comportamiento de "fallback" en la aplicación "app-books". En caso de que "app-authors" no responda, la aplicación puede usar datos en caché o valores predeterminados para asegurar que continúe funcionando.

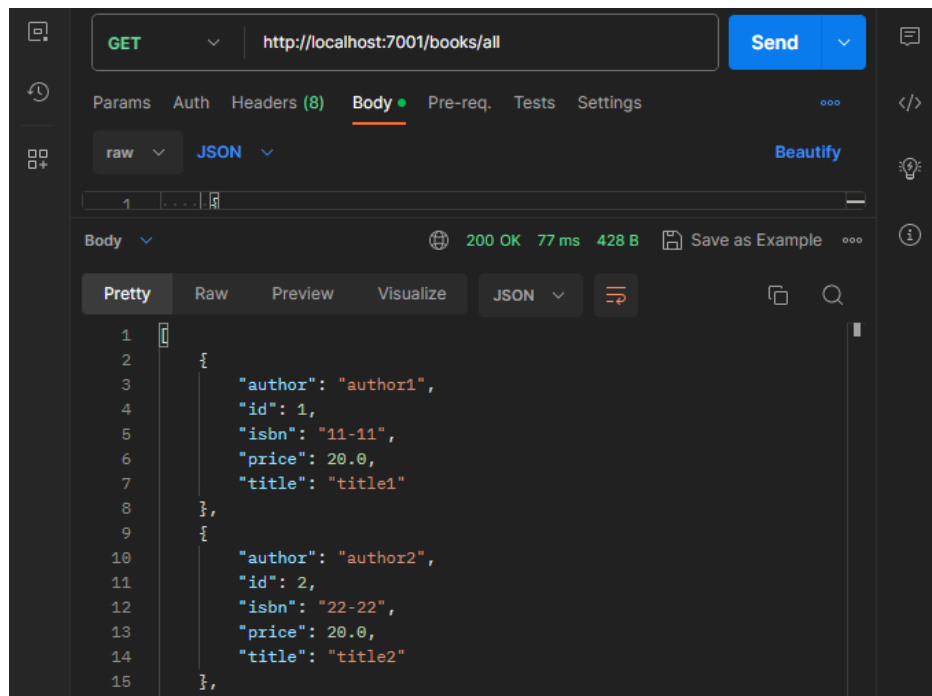
```
66 @CircuitBreaker(requestVolumeThreshold = 4, failureRatio = 0.5, delay = 2000)
67 @Fallback(fallbackMethod = "findAll")
68 public List<BookDto> findAllCompleto() throws Exception {
```

- Para poder notar que este servicio es tolerante a fallos lo que debemos hacer es parar el servicio de app-authors como se muestra a continuación:



- Una vez que se pare el servicio app-authors podemos acceder al servicio rest que consume a la aplicación “app-authors” mediante su siguiente Path:

Método	Path	Función
GET	http://localhost:7001/books/all	Lista todos los libros con el apellido y nombre de su respectivo autor.



- En la imagen anterior podemos observar como el servicio “app-books” es tolerante a fallos.