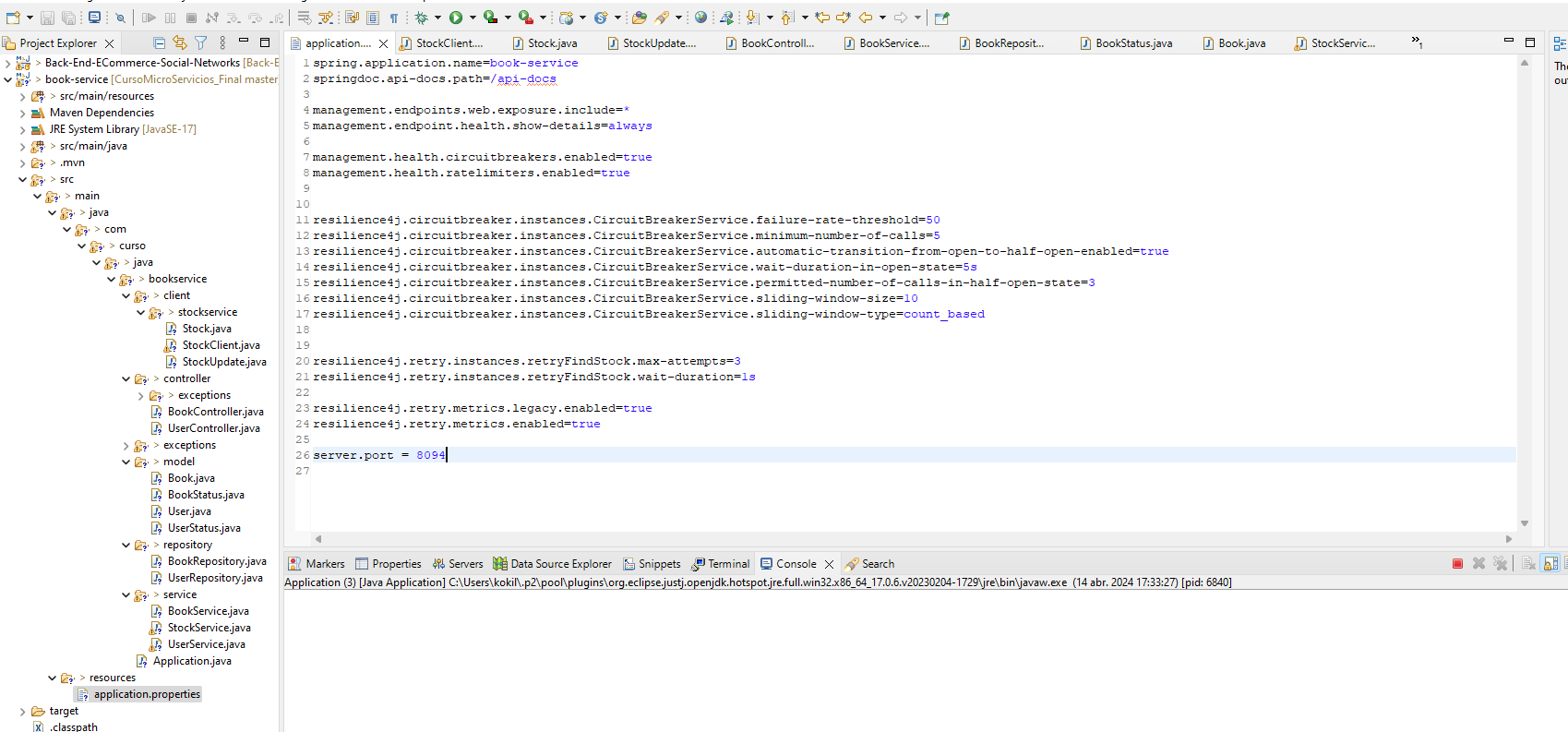
# **Prácticas Sesión 5 Arquitectura de Microservicios**

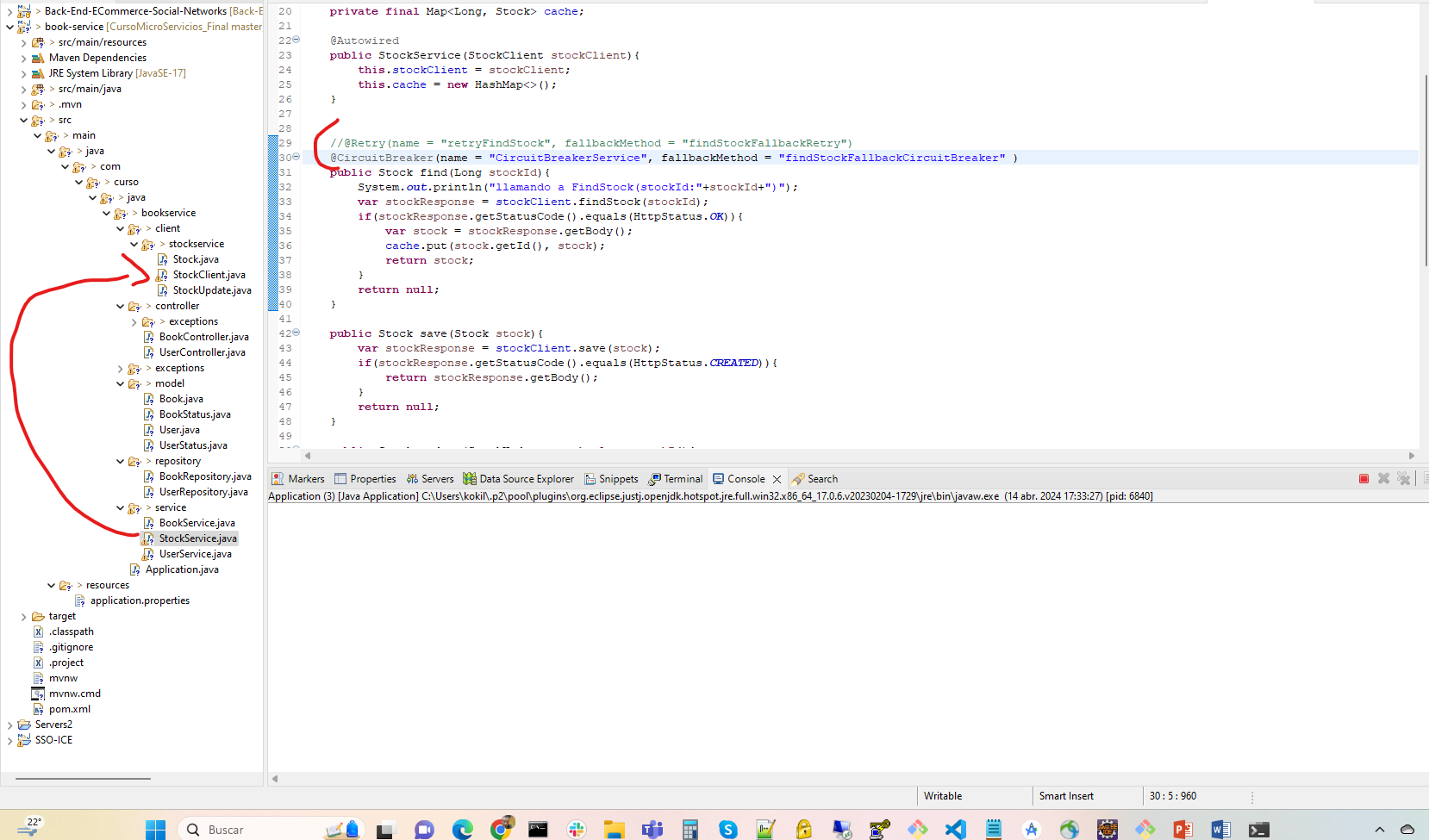
1. Descargar el proyecto para la lección. Este se compone de 2 módulos book-service y stock-service. El proyecto book-service consume el Rest API de stock-service.
2. Al ejecutar el proyecto book-service este usará en el puerto 8094. Al ejecutar el proyecto stock-service este usará el puerto 8093.
3. Implemenatr lo que falta en el proyecto para que los servicios o microservicios book-service y stock-service puedan trabajar con los patrone de diseño de retries y fallbak y de circuit breaker y fallback con consultas de cache.
4. Que responder cuando trate de crear un libro y el microservicio de existencias está abajo, que opciones podrían desarrollarse.

**Práctica Guiada Arquitectura de Microservicios.**

**Solución Item #3.**

****

Agregue en el application.properties localizado en el directorio de resources la siguiente configuración para que se puedan ejecutar tanto los retries y obtener por cache los datos de los libros y existencias además de poder de hacerlo por el patrón de diseño CircuitBreaker.



Revisar como el StockService del Book-Service llama al Stock Cliente de Feign y este puede ejecuar los retries y el fallback o el circuit breaker.



Agregar en el stock los siguientes dos métodos que ejecutan el caché por retries y el caché por circuit breaker.

**Consumir Rest API.**

1. Analice el uso de Open Feign, para construir el cliente Rest de Stocks en el proyecto book-service.
2. Utilice la colección de Postman adjunta en el laboratorio, para consumir los Rest API de book-service, observe como al crear un Book, u obtener libros, este servicio está interactuando con stock-service.

GET http://localhost:8080/api/books/1

{

"id": 1,

"title": "The Hobbit",

"summary": "Lorem ipsum",

"author": "J.R.R.Tolkien",

"year": 1937,

"status": "ACTIVE",

"stock": {

"id": 1,

"currentQuantity": 10

}

}

GET http://locahost:8091/api/stocks/1

{

"id": 1,

"currentQuantity": 10

}

Fault Tolerance

Analice el manejo de Retries automáticos en el código, haciendo uso de Resilient4J. Analice las entradas agregadas en el archivo application.properties.

Ejecutemos un request para obtener un libro.

GET http://localhost:8080/api/books/1

{

"id": 1,

"title": "The Hobbit",

"summary": "Lorem ipsum",

"author": "J.R.R.Tolkien",

"year": 1937,

"status": "ACTIVE",

"stock": {

"id": 1,

"currentQuantity": 10

}

}

Ahora detengamos el servicio stock-service, pero dejemos a book-service corriendo (sin detenerlo). Volvamos a acceder al mismo book.

GET http://localhost:8080/api/books/1

{

"id": 1,

"title": "The Hobbit",

"summary": "Lorem ipsum",

"author": "J.R.R.Tolkien",

"year": 1937,

"status": "ACTIVE",

"stock": {

"id": 1,

"currentQuantity": 10

}

}

¡Interesante!!! No se cae, a pesar que dicha operación consume a stock-service el cual no está disponible. Veamos los logs en consola, y observemos como al detectar que el método FindStock está fallando, luego de varios intentos ejecuta un fallback que retorna datos que tenemos almacenados en una cache.

Note el tiempo de respuesta (esto es porque está realizando intentos).

llamando a FindStock(1)

llamando a FindStock(1)

llamando a FindStock(1)

llamando a FindStock(1)

... Ejecutando Fallback por Retries

... Obteniendo Stock desde caché debido a Retries

Volvamos a iniciar stock-service. (Sus datos se eliminaron al detener el servicio, puesto que se almacenan en memoria). Ingresemos directamente a este servicio el stock

POST http://localhost:8091/api/stocks

{

"currentQuantity": 10

}

Ahora con ambos servicios disponibles, obtengamos el book.

GET http://localhost:8080/api/books/1

y analicemos los logs

llamando a FindStock(stockId:1)

llamando a FindStock(stockId:1)

llamando a FindStock(stockId:1)

Ya no hubo más llamados al fallBack, ni retries.

Podemos concluir que, mediante uso de estrategias de tolerancia a fallos, fuimos capaces de tener una degradación suavizada de la aplicación mientras uno de los servicios no estuvo disponible. De manera que la totalidad de la aplicación no se vió impactada durante el evento.

**Solución Item#4.**

Si el servicio de stock-service está inactivo en el momento en que intenta crear un libro en su servicio y no hay una respuesta almacenada en caché disponible, se enfrentará a una situación en la que no podrá obtener la información de existencias para el libro. En tales casos, normalmente tiene algunas opciones:

Maneje el error con elegancia: implemente mecanismos de manejo de errores en su servicio para manejar con elegancia la situación cuando el servicio de stock no esté disponible. Puede registrar el error, devolver un mensaje de error significativo al cliente y posiblemente volver a intentar la operación después de un retraso, si corresponde.

Devolver un valor de marcador de posición: si es aceptable para su aplicación, puede devolver un valor de marcador de posición o información de stock predeterminada para el libro. Este podría ser un valor predefinido que indica que la información de stocks no está disponible en este momento.

Poner en cola la solicitud: si el servicio de existencias está inactivo es un problema temporal, puede poner en cola la solicitud de creación de libros en su servicio y volver a intentarlo más tarde, cuando el servicio de existencias vuelva a estar en línea. Puede utilizar una cola de mensajes o un mecanismo similar para implementar esto.

Fallar rápido: Dependiendo de los requisitos de su aplicación, puede optar por fallar la solicitud de creación del libro inmediatamente e informar al cliente que la operación no se puede completar debido a la falta de disponibilidad del servicio de stock. Este enfoque garantiza que el cliente conozca el problema con prontitud y pueda tomar las medidas adecuadas.

La elección entre estas opciones depende de los requisitos específicos de su aplicación, el impacto de no tener la información de stock en el momento de la creación del libro y el comportamiento deseado en caso de fallas en el servicio.