Web Application 2 2020-2021

Exercise 4: Warehouse Service: Reactive and Blocking performance comparison

The Warehouse represents the physical place, in the e-commerce infrastructure, where various products, of different categories and quantities, are stored. New products can be added to the Warehouse, and their quantity can be updated. In the same way, products can be taken from the warehouse, if they are available.

A Product is characterized by:

- ID
- Name
- Category. There can be different types of categories, for example Food, Electronic, Home, Sport…
- Price
- Quantity

Step 1

Create a new project, configuring it to use the Spring Reactive Stack and R2DBC to interact, in a reactive way, with the MariaDB relational database. Include, in the dependencies, Spring Reactive Web, Spring Data R2DBC and MariaDB driver.

The end-points to expose are:

- /warehouse/products [POST]: Add a product, with a given initial quantity, to the warehouse
- /warehouse/products/{productID} [PATCH]: Update the quantity of the product available in the warehouse. Use a negative number to take a given quantity of the product out from the Warehouse. If the resulting quantity is negative, the operation will fail, leaving the product unmodified, otherwise the updated product is returned.
- /warehouse/products/{productID} [GET]: Get the product by id
- /warehouse/products [GET]: Get all products
- /warehouse/productsByCategory?category=<category> [GET]: Get all products by a given category

Use Kotlin Coroutines to handle, in the non-blocking way, the warehouse service. The Repository should extends the *CoroutineCrudRepository*. Custom methods added to the repository interface must be labelled with the @Query annotation, enclosing the SQL statement to be executed. If such a statement modifies the database, the method should also be labelled with the annotation @Modifying.

Implement the necessary checks, to make sure the whole system remains consistent.

Step 2

Create another project. Implement again the Warehouse Service, but this time in the blocking way, using Spring Web MVC, Spring Data JPA and MariaDB driver.

Step 3

Done this, we can compare the performances between Reactive and Blocking stack for two services that expose the same end-points and store the same data.

For this purpose, we'll use *ab*, *Apache HTTP server benchmarking tool*, used to measure the performances of HTTP web servers. Apache Bench is part of the package *apache2-utils.* If you don't have ab available on your system, you'll likely need to install the apache2-utils package via your package manager. If you're on

Windows, just download the Apache Binaries .zip file, extract it, and copy the ab.exe file wherever you need it to be.

With ab, we'll test the two implementations. Since the most expensive APIs are those of data retrieval, we'll focus on the end-point to retrieve all the products from warehouse, and on the end-point to get all the products by a given category.

The command to run is:

ab –c 10 –n 100 http://localhost:8080/warehouse/products

- –c stays for *concurrency*: the number of multiple requests to perform at time, aka the number of concurrent threads
- *-n* stays for *request:* the number of requests to perform for the benchmarking session

So, the *ab –c 10 –n 100 URL* simulates 100 connections over 10 concurrent threads.

The command outputs the benchmark, with some metrics, including *Requests per seconds*, that represents the measure of how the system scales horizontally.

Run this command both for the blocking and for the non-blocking services. Write the results in an excel file.

Re-run the same command for different combination of number of threads and number of requests, writing the observed results in the file.

Plots the two curves: how do the two systems respond? What solution scales better?