


| | |
|--|---|
|  <p>1 2 9 0</p> <p>FACULDADE DE CIÊNCIAS E TECNOLOGIA UNIVERSIDADE DE COIMBRA</p> <p>UNIVERSIDADE DE COIMBRA FACULDADE DE CIÊNCIAS E TECNOLOGIA <i>Departamento de Engenharia Informática</i></p> | <p>2º Projeto / 2nd Assignment</p> <p>Integração de Sistemas Enterprise Application Integration</p> <p>2023/24 – 1st Semester MEI, MES, MEIG</p> <p>Deadline: 2023-11-10</p> |
| <p><u>Nota:</u> A fraude denota uma grave falta de ética e constitui um comportamento não admissível num estudante do ensino superior e futuro profissional. Qualquer tentativa de fraude pode levar à reprovação na disciplina tanto do facilitador como do prevaricador.</p> <p>MUITO IMPORTANTE: o código entregue pelos alunos vai ser submetido a um sistema de deteção de fraudes.</p> <p>VERY IMPORTANT: the code delivered by students will be submitted to a fraud detection system.</p> | |

Reactor Core/Web Reactive

Objectives

- ☐ Learn to Program Using a Declarative Reactive Programming Model

Reference Material

Reactor 3 Reference Guide:

<https://projectreactor.io/docs/core/release/reference/>

Reactor core reference:

<https://projectreactor.io/docs/core/release/api/index.html>

Class Flux<T>:

<https://projectreactor.io/docs/core/release/api/reactor/core/publisher/Flux.html>

Class Mono<T>:

<https://projectreactor.io/docs/core/release/api/reactor/core/publisher/Mono.html>

Web Reactive:

<https://docs.spring.io/spring-framework/docs/current/reference/html/web-reactive.html>

Start your own Spring Boot project:

<https://start.spring.io>

Software Configuration

To create the project and complete the assignment, students have, at least, two options: either installing everything directly on their operating system or using Docker and Docker compose. The latter might be heavier for the computer but makes configuration easier. Additionally, students may gain some skills that are quite useful for the industry. The professors will make a version of Docker compose available that includes:

- ☐ Java 18
- ☐ Latest PostgreSQL (version 16)
- ☐ Maven 3

Webflux Maven Dependency

To solve some of the exercises, on the client side, students should include the following dependency (with whatever version number is the newest):

```
<dependency>
  <groupId>org.springframework.boot</groupId>
  <artifactId>spring-boot-starter-webflux</artifactId>
  <version>3.1.4</version>
</dependency>
```

For the server, students should need, at least, the following dependencies, which they may add during the configuration of a new Spring Boot project (refer to the link above):

- ☐ spring-boot-starter-webflux
 - ☐ spring-boot-starter-data-r2dbc
 - ☐ lombok (may includeSlf4j)
-

Integrated Development Environment

Eclipse, **IntelliJ**, and **Visual Studio Code** are the recommended IDEs that will have some support from the professors. Students should use **Maven** instead of IDE-managed projects. **Visual Studio Code** integrates beautifully with the containerized environment; IntelliJ also includes some support for containers.

Training (not for evaluation)

Java Lambda Expressions

- 1- Sort an array in Java resorting to a Lambda function to make the individual comparisons. The Lambda expression should return a negative number if the first

one is less than the second, zero if they are equal and a positive number otherwise. This is standard behavior for a comparator function.

- 2- Assign a Lambda expression to a variable, such that you can later use this variable instead of using a direct Lambda expression as you did in the previous exercise.
- 3- Create your function to compare two integers according to another function that it receives as a parameter. Invoke that function, by passing a Lambda expression that does the comparison.

Reactor Flux

- 4- Create a simple Flux stream that generates integer numbers and includes a map, a filter, and a subscriber. Check which thread is doing the work in each step.
- 5- Create a flux stream that generates integer numbers. Change the stream type to output a string with "Number x", where x is the original number.
- 6- Create a stream of integers that computes the moving average with a window of 7.
- 7- Transform a Flux<Integer> stream into a Mono<List<Integer>>.
- 8- Create a stream that generates integer numbers and that includes an operator that works on pairs of numbers: if the first one in the pair is greater than the second, the pair can go through, otherwise, numbers are dropped. The final subscriber should work on a Flux<Integer>.
- 9- Create a stream with a subscriber that exerts back pressure on the publisher, such that the subscriber only requests one item at a time. This involves creating a subscriber with hooks for onSubscription(), onNext(), and onComplete().
 - a) First, generate a sequence with 2 integers and on subscription request 1 item. This is the only request this subscriber ever does. You should not see both integers of the sequence. Does the program terminate?
 - b) Now, in addition to the previous request, ask for an additional item upon receiving each new item. Can you now see both integers of the sequence? Does the program terminate?

Reactor Flux Over the Network

- 10- Create a web client that accesses an online service, for example, the web pages of DEI. Transform the response into a Mono<String> and subscribe to that Mono, printing the string on the subscriber. If the program finishes right away, can you see the web page you requested? And what if you add 10 seconds of sleep before finishing the program? Relate what you are observing with what you observed in the previous question.

11-Create a simple web application with Spring with the following reactive endpoints:

- a) GET /pets (Flux<Pet>)
- b) GET /pets/{id}, (Mono<Pet>)
- c) POST /pets (Mono<Pet>)
- d) PUT /pets/{id} (Mono<Pet>)
- e) DELETE /pets/{id} (Mono<Pet>)

Make sure you use the default structure for Spring Applications: Controller, Service, Repository, and Entity. For this exercise, you can use a HashMap to store data in memory.

12-Create a reactive Web Client to consume the data provided by the previous endpoints.

Project (for evaluation)

Overview

In this project, students will develop a web application that exposes web services and a client application that will consume these web services. Students should write the applications using WebFlux. Next, we detail the requirements of these applications. Students may add additional functionality under professors' guidance.

Reactive Server

Students should create a server that will expose the following data via web services:

- ☐ Pet data, including:
 - i. Identifier.
 - ii. Name.
 - iii. Species.
 - iv. Birth date.
 - v. Weight.
- ☐ Owner data, including:
 - i. Identifier.
 - ii. Name.
 - iii. Telephone Number.

The Pet and Owner entities have a one-to-many relationship, as a single owner may have many pets.

The server is a legacy application with basic functionality that students must not try to enrich, to simplify client-side queries. Server services are, therefore, limited to simple CRUD operations, according to the following list:

- ☐ Create Pet/Owner.
- ☐ Read specific Pet/Owner or all Pets/Owners.
- ☐ Update specific Pet/Owner.
- ☐ Delete specific Pet/Owner (if the owner is not connected to another pet).

The server **must not** provide a service with all the data, like, owners and all the respective pets, names, and remaining details. Nonetheless, the server can provide a service that lists the identifiers of pets given an owner identifier. This has some impact on the last queries ahead.

Students must use logging on the server side.

Client Using Reactive Code

On the client side, students should collect data from the server according to the following queries. Responses may go to different files and may run in parallel. For example, the output of query #1 may go to file resp1.txt, output of query #2 may go to file resp2.txt, etc. No interaction and no menu are necessary:

1. Names and telephones of all Owners.
2. Total number of Pets.
3. Total number of dogs.
4. Total number of animals weighting more than 10 kg. Sort this list by ascending order of animal weight.
5. Average and standard deviations of animal weights.
6. The name of the eldest Pet.
7. Average number of Pets per Owner, considering only owners with more than one animal.
8. Name of Owner and number of respective Pets, sorted by this number in descending order. In this and other exercises, students should minimize the blocking points for their applications, e.g., by means of a `block()` call.
9. The same as before but now with the names of all pets instead of simply the number. Note that most of the work should occur on the client, as mentioned before, on the limitations imposed on the server.

Time matters. Students should try to make these queries run as quickly as possible, by exploring all the data in the same query, or by taking advantage of threads. They are allowed to introduce delays on the server side to emulate network delays. Students are not allowed to copy the entire databases to the client or use some other technique that defeats the idea that the client is interacting with a legacy third-party server. For example, any change to the server requirements should be discussed with the professors.

At least one of the client-side queries should be able to tolerate network failures, by retrying up to three times to reconnect, before giving up. Students may create a special service on the server and a special query on the client, outside any time control, to emulate this case.

Techniques **not to use** in the project:

- ☐ Using standard Java Streams.
- ☐ Atomic Objects (unless there is a good reason for that).
- ☐ Abuse of `collectList()` to make the posterior use of a standard for outside the flux.
- ☐ Other approaches that circumvent the natural operation of Reactor core.

Final Delivery

- ☐ The assignment should be made by groups of two students. Do not forget to associate the colleague during the submission process.
- ☐ This assignment contains two parts: one is for training only and does not count for evaluation. Students should only deliver the other part.
- ☐ Students must submit their project in a zip file using Inforestudante. The submission contents are:

- Source code of the project ready to compile and execute. Do not include compiled code, like classes and executables.
- A small report in pdf format (6 pages maximum) about the implementation of the project. See below for details.

Suggestion of Report

The report should focus on aspects that are not trivial. Most of the server might be straightforward apart from details concerning reactivity, the data access layer, or exception generation. If any of these or other aspect is unusual, students should add them in the report. On the other hand, the client is much more complex and worth of attention. Students should outline the ideas that drive each one of the queries they do. A very interesting aspect concerning these queries is what kind of optimization is involved to make them run faster. Students should output the lessons they have learned regarding performance in the report. While this is not mandatory, students may even present the evolution of performance they managed to achieve explaining the reasons for that evolution.

Good Work!