1 2 9 0 FACULDADE DE CIÊNCIAS E TECNOLOGIA UNIVERSIDADE D COIMBRA *Departamento de Engenharia Informática*	**2º Projeto / 2nd Assignment** **Integração de Sistemas** **Enterprise Application Integration** **2024/25 – 1st Semester** MEI, MES, MEIG **Deadline: 2024-11-08**

Nota: 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.

MUITO IMPORTANTE: o código entregue pelos alunos vai ser submetido a um sistema de deteção de fraudes.

VERY IMPORTANT: the code delivered by students will be submitted to a fraud detection system.

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

Spring WebFlux:
https://docs.spring.io/spring-framework/reference/web/webflux.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 21
- 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>3.3.4</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 include Slf4j)

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 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 makes. 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 /media (Flux<Media>)
b) GET /media/{id}, (Mono<Media>)
c) POST /media (Mono<Media>)
d) PUT /media/{id} (Mono<Media>)
e) DELETE /media/{id} (Mono<Media>)

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 those web services. The web application manages media content and user interactions, providing functionality to catalog and rate movies and TV shows. Students should write both applications using Spring Web Flux. 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:

- Media data, including:
    a. Identifier.
    b. Title.
    c. Release date.
    d. Average rating (between 0 and 10).
    e. Type (Movie or TV Show).
- User data, including:
    a. Identifier.
    b. Name.
    c. Age.
    d. Gender.

The media and user entities have a many-to-many relationship, thus requiring a third table in the database.

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 media/user.
- Create relationship.
- Read all media/user.
- Read specific media/user.
- Update specific media/user.
- Delete specific media/user (if they are not connected to another media/user).
- Delete relationship.
- Create relationship.
- Read relationship. This service can only return the identifiers of some media/user, not the entire media data, i.e., students should not create a service that immediately provides, say, a user with all data of all the user's media.

Students must use logging on the server side.

Client Using Reactive Code

On the client side, students should collect data from the server and produce the following reports to one or more files whose names are program parameters:

1. Titles and release dates of all media items.
2. Total count of media items.
3. Total count of media items that are really good (i.e., that have more than 8 on average rating).
4. Total count of media items that are subscribed.
5. Data of media items that are from the 80's (i.e., whose release dates are between 01-01-1980 and 31-12-1989). This list should be sorted so that media items with greater average ratings come first. This data does not need to include the users.
6. Average and standard deviations of all media items ratings.
7. The name of the oldest media item.
8. The average number of users per media item. Note that some media items may not have users and vice versa.
9. Name and number of users per media item, sorted by user age in descending order.
10. Complete data of all users, by adding the names of subscribed media items.

These results should appear in sequence in the program, no menu is necessary. 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, e.g., .java files. Do not include compiled code, like .class files.
    - 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!