Systems Integration
Assignment 2 - Reactor

João Dionísio
uc2019217030@student.uc.pt
no. 2019217030

Sofia Alves
uc2019227240@student.uc.pt
no. 2019227240

Class PL1

November 2022

Contents

1 Introduction

The aim of this project is to build a web application using a Declarative Reactive Programming Model. The technologies used in this project were Reactor Flux, Spring WebFlux, Spring Boot R2DBC and Maven.

2 Software Architecture

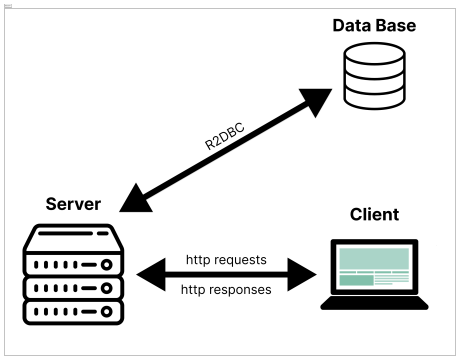

Figure 1: Software Architecture

The web application is composed of two main parts, the server and the client.

- Server - is responsible for responding to HTTP requests of the client, performing the following CRUD operations:

  - Create student/professor;
  - Create relationship;
  - Read all students/professors;
  - Read specific student/professor;
  - Update specific student/professor;
  - Delete specific student/professor;
  - Delete relationship;
  - Create relationship;
  - Read relationship.

  The requests to the database are made with the use of Spring Boot R2DBC, which simplified the process of creating queries.

- Client - On the client side it is possible to request the following informations:
  - Names and birthdates of all students;
  - Total number of students;
  - Total number of students that are active;
  - Total number of courses completed for all students;
  - Data of students that are in the last year of their graduation;
  - Average and standard deviations of all student grades;
  - Average and standard deviations of students who have finished their graduation;
  - Average number of professors per student;
  - Name and number of students per professor, sorted in descending order;
  - Complete data of all students, by adding the names of their professors.

3 Entity Relationship Diagram

The Entity-Relationship diagram has three entities: the student, the professor, and their relationship, each with its attributes. The third table refers to the relationship between the professor and the student (student_professor). Instead of using a composite key, which would impose some constraints, it uses a unique key and stores the ID of the student and the professor involved in the relationship as foreign keys. This way, we can access the members of the relationship by checking if the desired identity is present in each relationship by searching for its ID in the table.

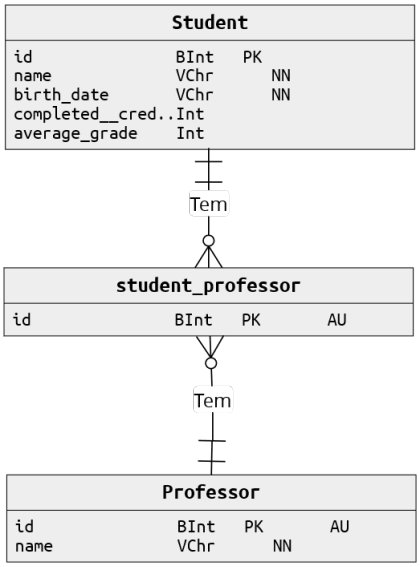

Figure 2: Entity-Relationship diagram

4 Client Functionalities

In this section we will discuss the implementation of each client's operation.

1. Names and birthdates of all students

The client requests all students from the server. The server sends the data of all students and the client inserts it into a flux and outputs the respective names and birth dates of all students. This operation was used to showcase the tolerance to network failures using the function retryWhen(), which attempts to reestablish connection three times with a five second interval between each attempt.

2. Total number of students

Similar to the previous operation, but the authors used the count() function to get the total number of students and printed the result.

3. Total number of students that are active

First, the authors used the filter() function to filter out students who have less than 180 completed credits, and then used the count() function to count those students. After getting the total number of active students, the result is printed.

4. Total number of courses completed for all students

The completed credits per student are divided by six using the map() function and then summed over the previous values using the reduce() function.

5. Data of students that are in the last year of their graduation

First, the filter() function is applied to the student objects in the flux to filter out only those students whose completed credits are at least 120 and less than 180. Then the students are sorted using the sort() function. Finally, we print the student data.

6. Average and standard deviations of all student grades

To find the average of all students' grades, a map() function was used to get the average grade of the students. Then a reduce() function was perform to get the sum of the grades, and finally the authors used a map() function to divide the sum by the total number of students. For the standard deviation, a map() and a reduce() function were used to reproduce the formula, and then the result was printed using a doOnNext() function.

7. Average and standard deviations of students who have finished their graduation

This operation is similar to the previous one, but first we filter the students by the completed credits, working only with those who have less than or equal to 180 credits.

8. The name of the eldest student

For this operation, the authors created the auxiliary function getEldest(). This function compares the birth dates of two students and returns the one whose birth date is older. This function is used in a reduce() function and returns the oldest student at the end.

9. Average number of professors per student

In this operation, the client not only queries the data of the students, but also the relationships between the students and the professors. For this purpose, a map() function was used to obtain the ID of the students, which is then used in a filter() function that queries only the relationships of the current student. Using these relationships, we can use a count function to count the number of professors of a given student. Then we use a reduce function to add the total number of professors per student. Finally, we use a map function to divide the sum by the total number of students and print that value.

10. Name and number of students per professor, sorted in descending order

First, the client retrieves all professors from the server and sorts them by the number of students they have. This step is done by a retrieval that determines the number of relations with this professor identifier. This way we can use this query as a comparison value for the sort function. After all the professors have been sorted as intended, an print is created that queries the name of the professor and the total number of students it has.

11. Complete data of all students, by adding the names of their professors

In this operation, the program first outputs the current student data and then determines the names of their professors by filtering the objects in the student_professor table whose student ID matches that of the current student. Then check the ID of the professors, query them and output their names.

5 Conclusion

Through this project, the authors were able to learn more about reactive programming, how flux streams work, the roles of a publisher and a subscriber in a flux, different operators of flux, and lambda expressions. The authors believe that the use of blocks to synchronize the prints of the results may have an impact on the performance of the program in terms of runtime, since it blocks the original thread, so a different approach could have been taken to work around this aspect. Apart from this, the authors could see the principle advantages of reactive programming, as it allows operations to be executed asynchronously, making the code more readable and allowing the consumer to signal when the emission rate is too high.