Technical Report - **Product specification**

eDuca

Course:	IES - Introdução à Engenharia de Software
Date:	Aveiro, 26/10/2023
Students:	107647:Diogo Silva 107660:Miguel Cruzeiro 107476:Rafael Vilaça
Project abstract:	The Multi-School Student Management Platform is a system designed to enhance the management of multiple schools' student-related activities. It provides a centralized hub for students, teachers, and school administrators to efficiently manage and access student information, academic records, communication, and more. The key concept revolves around improving the educational experience for all stakeholders while ensuring data security, scalability, and user-friendliness.

Table of contents:

1 Introduction

This report presents the concept and vision for a web-based solution designed to revolutionize the way schools handle academic data.

The objective of this project is to create a platform that facilitates the management of grades, notes,, and related information for both teachers and students in a school environment.

Additionally, it will promote smooth communication between teachers and students.

2 Product concept

Vision statement

This system will be used for:

- Allowing teachers to efficiently input and publish grades, notes, and send alerts.
- Enabling students to access their academic information.
- Simplifying the communication process between teachers and students regarding academic matters.

Personas and Scenarios

Persona 1:

Name: Sara

Age:17

Profession:Student

Sara is an interested student and always wants to be up to date with messages that teachers can send.

She likes to plan her studies and for that she needs some way of knowing when there are tests or meetings.

Worried about her average to enter university, she likes to keep her grades from all her subjects.

Persona 2:

Name:João

Age:49

Profession:Math Teacher

The teacher wants to publish the grades for his subject.

He also wants to warn students about the content that will be covered in next week's test.

If there is some error on the correction of a test, he wants to be able to edit the grade of the test.

Persona 3:

Name:Ana

Age:47x

Profession.School Administrator

As the school administrator, when new students/teachers arrive, she adds them to the system or when they leave she removes them from the system..

She has access to the grades and of students from the respective school.

Product requirements (User stories)

User Registration and Profiles:

This epic focuses on user registration and profile management.

User Stories:

- As a student/teacher, I want to edit my profile.
- As a school administrator, I want to manage and verify user profiles within my school.

Student Enrollment and Records:

This epic deals with student enrollment, record management, and academic progress tracking.

User Stories:

- As a school administrator, I want to be able to edit everything (grades,students,teachers), enroll students in my school and input their basic information.
- As a teacher, I want to update student records with grades.

Communication and Notifications:

This epic focuses on communication between the administrator, the teachers and the students.

User Stories:

- As a teacher, I want to send notifications and updates to students.
- As a student, I want to receive notifications about my academic performance and school events.

Grades:

This epic involves checking grades.

User Stories:

- As a teacher, I want to see all the grades I gave and at which subject and edit if needed.
- As an administrator I want to receive notifications when the average of all students in a specific subject in a specific class is negative.
- As a student, I want to check my grades and see my average.

3 Architecture notebook

Key requirements and constraints

The system should be able to generate and consume data automatically, simulating what would happen in a functioning school.-

The system cannot allow third-party access to confidential data, requiring an authentication system.-

It should be ensured that the data generated by the simulator is not mixed within the message broker with the received data.-

The users should be able to have access to the WebApp at any moment.-

Due to the interactions users can perform, the Python script needs to consume the updated new data to ensure the congruence of the generated data.-

For example, when the student leaves a class it doesn't make sense to keep generating data to the student.-

The school administrator should be the only one able to manage all accounts (create,update and delete (teacher and student accounts)).-

The school administrator parameters should be name,password.-

The student account parameters should be nmec, name, email, school, Grades/Classes and password.

The teacher account parameters should be name, email, nmec, school, Subject(s) Taught, grades given, subjects given to each class and password.

The student account has a read only general use (the students can check their information and can only update some of their personal information (password is allowed, name etc isn't).

The teacher account has the same limitations as the student (in personal data update and information check) but has some write privileges, he can send and edit grades sended from him.

The school admin account must have access to the information of every student and teacher.

Architectural view

Web Application:

To develop our web application, we opted for Angular mainly because of its adeptness in creating dynamic applications. Our familiarity with this technology, alongside its component reusability and effortless maintenance of consistently refreshed data, influenced our decision.

eDuca: In our Spring Boot-based service, we've established a comprehensive architecture. Our models, including Student, Teacher, Grades, Notifications, School Administrator, Subject and Teaching Assignments, define entities crucial for system functionality.

The Teaching Assignment model represents the attribution of a teacher to a class for teaching a specific subject. Each model has a corresponding repository, leveraging Spring Data JPA for seamless CRUD operations and efficient database interaction. Our services orchestrate operations on the data, interfacing with the repositories to ensure business logic is executed effectively.

Controllers define REST API endpoints, providing a clear and accessible interface for our application. To enhance security, we've implemented JWT Authentication. Upon user login, the system generates an encrypted token containing user information like email and role. This token is then utilized to manage access, allowing us to control which pages are accessible to different user types.

This approach enhances security by tightly regulating user permissions throughout the application. We've also incorporated Cross-Origin Resource Sharing (CORS) to facilitate secure communication between our Spring Boot service and the web application. CORS ensures that our API endpoints can be accessed by authorized domains, enhancing the overall security and compatibility of our system.

Simulation of Data

All of the default data shown in the application will be generated by a python script this python script will generate on execution user,teachers etc (some of the attributes are completely randomized) when the initial data generation is finished the script starts waiting for a message from the web-app the message can be (init or periodic) upon receiving the init one the script starts processing the data that it generated and sending it to the respective channel in the broker, the script then keeps going and consuming the messages that the springboot is sending when the message received is periodic the script will check the data in the application using its endpoints directly it first checks if the insertion is already concluded in the respective controller (since the initial message has a lot of data to be inserted without this progress control to see when the data is completely inserted it would result in problems using incomplete data) then it uses the data from the springboot (this way the grades are always generated with real time data) to generate grades randomly (random subject) with certain limits (max 20 grades-subject-student) all the endpoints consulting are made with admin privileges (done using the autentication endpoint and adding the token to all the requests)

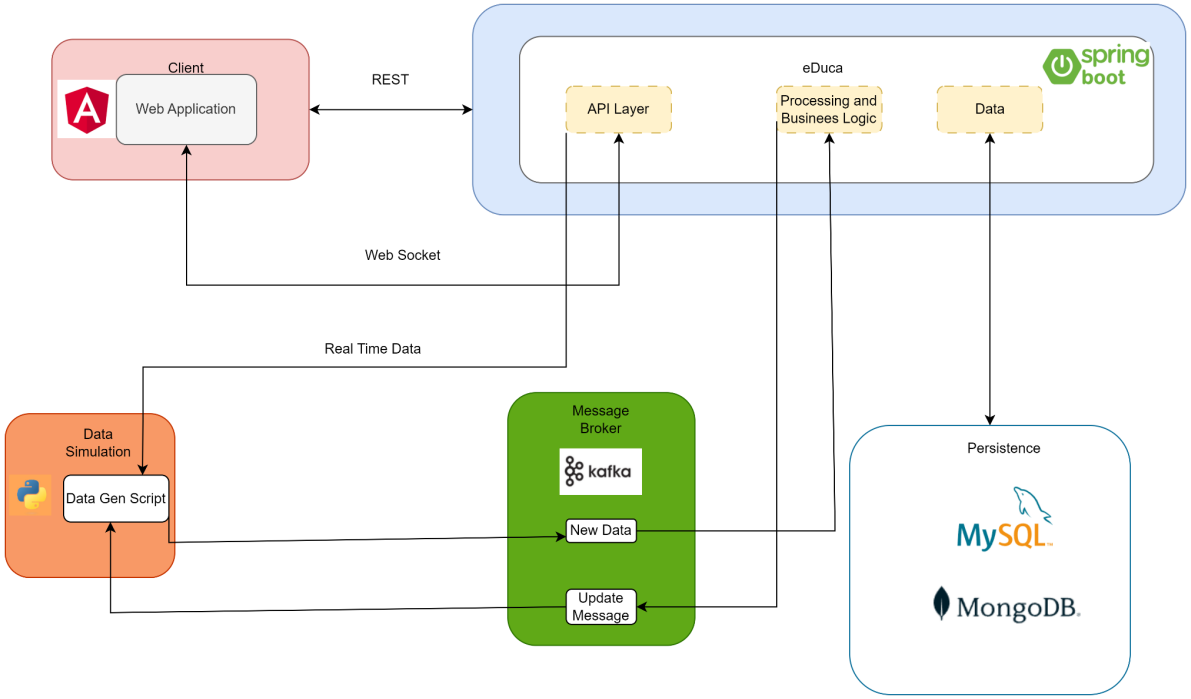
Message Broker

We chose Kafka as the message broker because it was the technology that best fit our use case, it is fault tolerant, scalable and has high performance .

The webapp initially sends a message (init to the message broker (channel-1) when it fully starts) the python script upon receiving that message sends back (channel-2) the system initial data this includes teachers,subjects,classes,students and teaching_assignments the web app consumes and inserts the given data in its database, after the init message the webapp starts sending a periodic message (channel-1) the python script upon receiving the message process it and sends back the grades (channel-2) using different channels assures that the data doesn't get mixed.

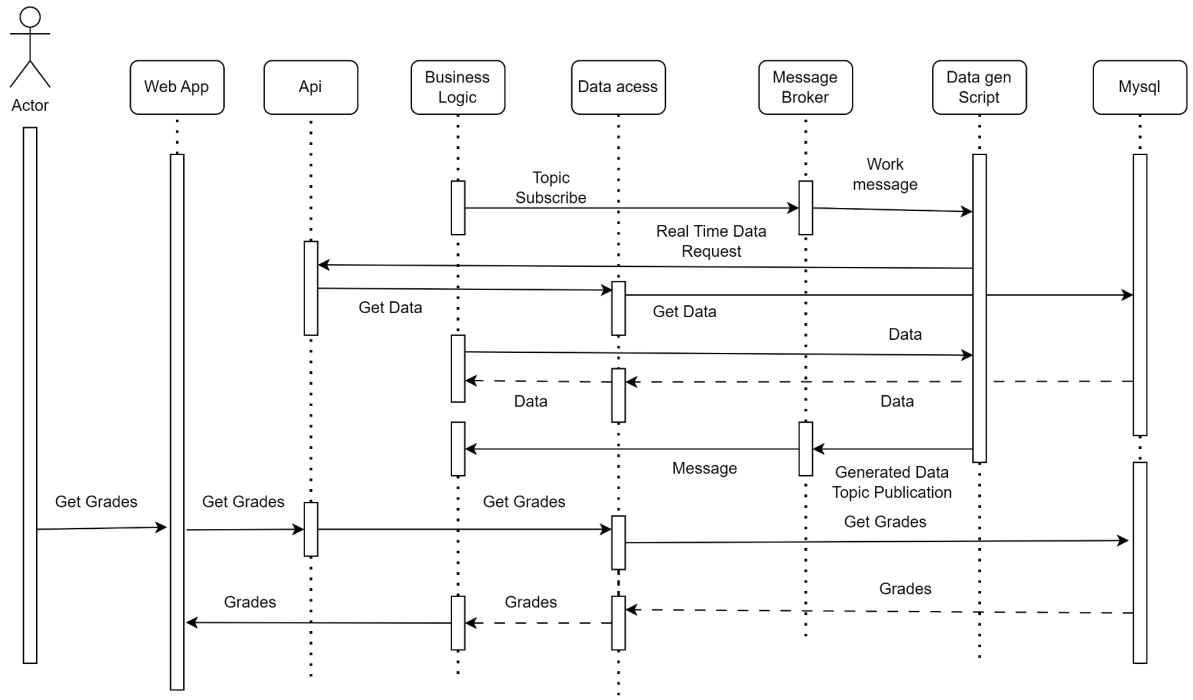
Databases

We will use MySQL to store the most important data of the system, the main reason for this is that the entities are deeply related and we already had the knowledge of the practical classes. Additionally, we will utilize MongoDB to store information such as notifications that students receive the main reason for such is that this way it is easier to manage the notifications this way we can have multiple notification types and only one model besides that mongo offers better performance and if it was needed to change the notification it was easier to make adjustments.



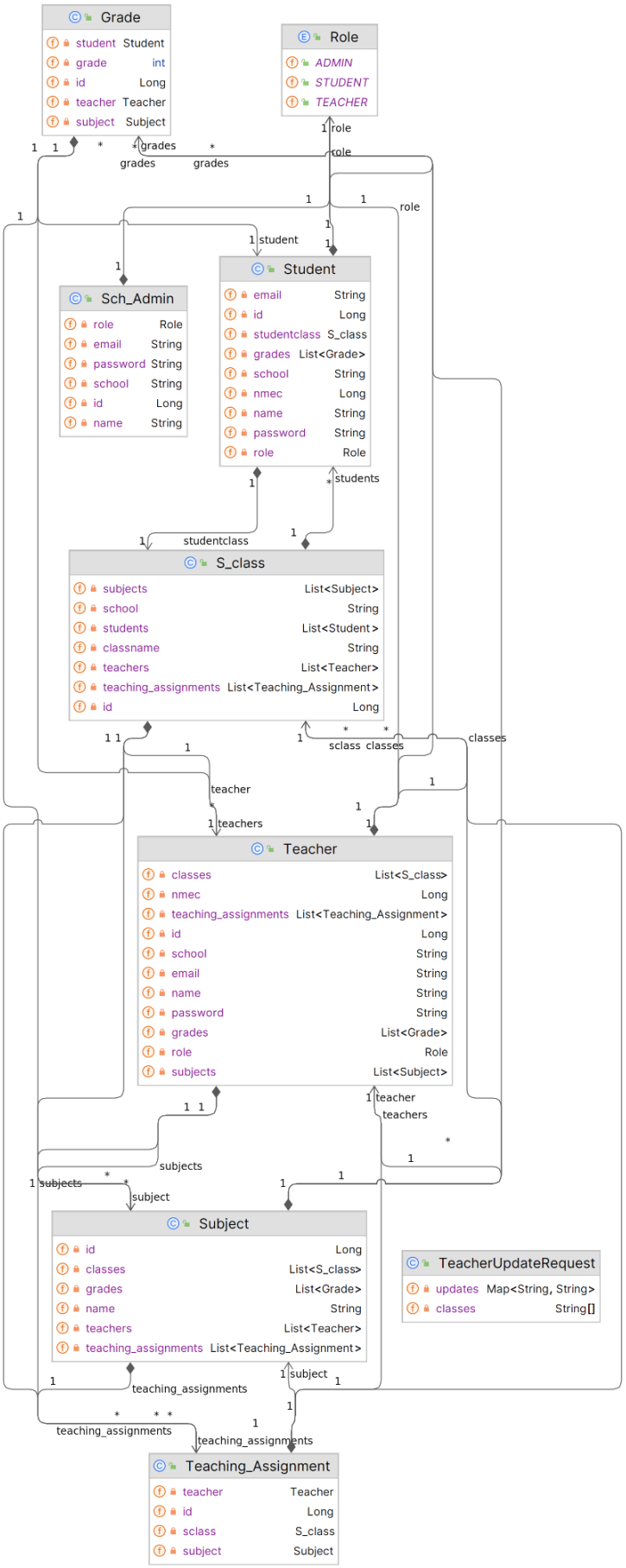
Module interactions

- All of the data presented initially in the application is generated by the Data gen script when the business logic subscribes to a topic, it sends the init message to the broker and the script consumes it.
- After the initial data is inserted the script starts consuming data in real time (this is done accessing the respectives endpoints of the api) then it uses the data to produce grades and sends them back to the respective topic in the message broker, the message broker sends it to the business logic and the data is inserted in the system.
- There are websockets doing work in the background, through them various types of notifications are sent to the respectives users.
- When a request is made by a Student (or someone else) to see grades, a call is made to the api, which will trigger the retrieval of the data that is stored in the database finally the data will be show in the application



4 Information perspetive

We used MySQL to handle the data in our system.The are our entities and their relations:


Grade
student Student
grade int
id Long
teacher Teacher
subject Subject
Role
ADMIN
STUDENT
TEACHER
Sch_Admin
role Role
email String
password String
school String
id Long
name String
Student
email String
id Long
studentclass S_class
grades List<Grade>
school String
nmec Long
name String
password String
role Role
S_class
subjects List<Subject>
school String
students List<Student>
classname String
teachers List<Teacher>
teaching_assignments List<Teaching_Assignment>
id Long
Teacher
classes List<S_class>
nmec Long
teaching_assignments List<Teaching_Assignment>
id Long
school String
email String
name String
password String
grades List<Grade>
role Role
subjects List<Subject>
Subject
id Long
classes List<S_class>
grades List<Grade>
name String
teachers List<Teacher>
teaching_assignments List<Teaching_Assignment>
TeacherUpdateRequest
updates Map<String, String>
classes String[]
Teaching_Assignment
teacher Teacher
id Long
sclass S_class
subject Subject
grades
role
student
students
studentclass
sclass
classes
teacher
teachers
subjects
subject
teaching_assignments

The system comprises three main types of users: **Student**, **Teacher**, and **System Admin**.

Every entity in our domain has as Primary Key the id attribute.

The Role class functions as an enumeration class, helping in distinguishing between different types of users within the system.

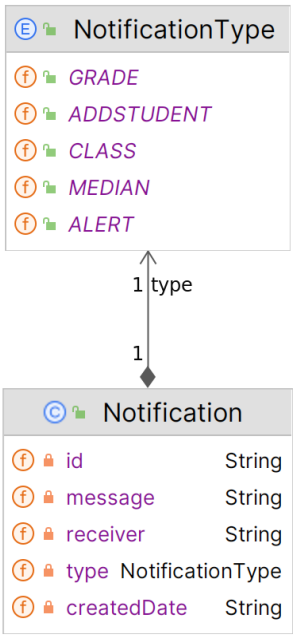
In the **Student** entity, the primary key is the id and email and nmec are unique attributes. Notably, each **Studen**t is associated with the **Grades** entity through the grades attribute. The **Grades** entity, with id as its primary key, includes attributes such as student (referencing the **Studen**t entity), subject (indicating the relevant academic discipline and referencing the **Subject** entity), teacher (identifying those responsible for lecturing the subject and referencing the **Teacher** entity), and grade (numeric value of the grade between -2 and 20).

Each **S_Class** entity is characterized by a unique classname and establishes a one-to-many relationship (1:N) with the **Student** entity through its 'students' attribute, representing all the students belonging to that class. Additionally, the **S_Class** entity features an attribute named 'subjects,' reflecting a one-to-many relationship (1:N) that encapsulates all the subjects associated with a specific **S_Class**. Furthermore, the **S_Class** entity includes the 'teachers' attribute, encompassing all the teachers affiliated with that particular class.Finally, it also has a teaching_assignment attribute associated with the **Teaching_Assignment** entity that represents the attribution of a subject and a class to a teacher and that in the case of the **S_Class** contains all the teaching assignments of **Teachers** for that **S_Class**.

A **Teacher** entity possesses unique attributes, namely *email* and nmec. The classes attribute signifies that a teacher can instruct multiple classes, and the subjects attribute encapsulates all the subjects the teacher is qualified to teach. Additionally, the entity incorporates a teaching assignment attribute, containing information about the teaching assignments for the teacher across various **S_Classes** and **Subjects.**

The School Admin entity possesses the unique attribute email and manages all the entities in the system.

In our system, the notification feature has been implemented using MongoDB:



For efficient notification functionality, we've designed a **Notification** entity comprising a primary key id for unique identification. The entity includes a message attribute containing the notification content, a receiver attribute specifying the intended recipient, and a createdDate attribute denoting the timestamp of notification creation. Additionally, a type attribute is linked to the **NotificationType** enumeration class, aiding in categorizing the notification type.

5 References and resources

All the api endpoints are documented with swagger in the path
localhost:8080/swagger-ui/index.html ->link
Use this link in the bar -> /api-docs

SpringBoot -> https://spring.io/

Spring Initializr -> https://start.spring.io/

Maven Repository -> https://mvnrepository.com/

Angular Documentation -> https://angular.io/

Material fornecido pelos docentes