UNIVERSITAT DE LLEIDA
EPS
Computer Engineering Degree, 4th
Distributed Computing, Computer Science

Activity 6: REST WS

Joaquim Picó Mora, Ian Palacín Aliana, Sergi Simón Balcells
Computer Science

Professors: Eloi Gabaldon, Jordi Gervas, Josep Lluís Lèrida
Date: 11th of January of 2021

Contents

1 Introduction **1**

2 UML **2**

2.1 Exam 2

2.2 User 3

2.3 Grades 3

3 Endpoint Table **4**

4 Screenshots **6**

4.1 Authentication 6

4.2 Exam 7

4.3 Grades 10

5 How To **14**

5.1 Getting Started 14

5.1.1 Prerequisites 14

5.1.2 Installing 14

5.2 Running the tests 16

6 Solution justification **16**

6.1 Web Service 16

6.1.1 Technologies 16

6.1.2 ViewSets and Generics 17

6.1.3 Decisions 18

6.2 RMI modifications 18

6.3 Time dedicated 20

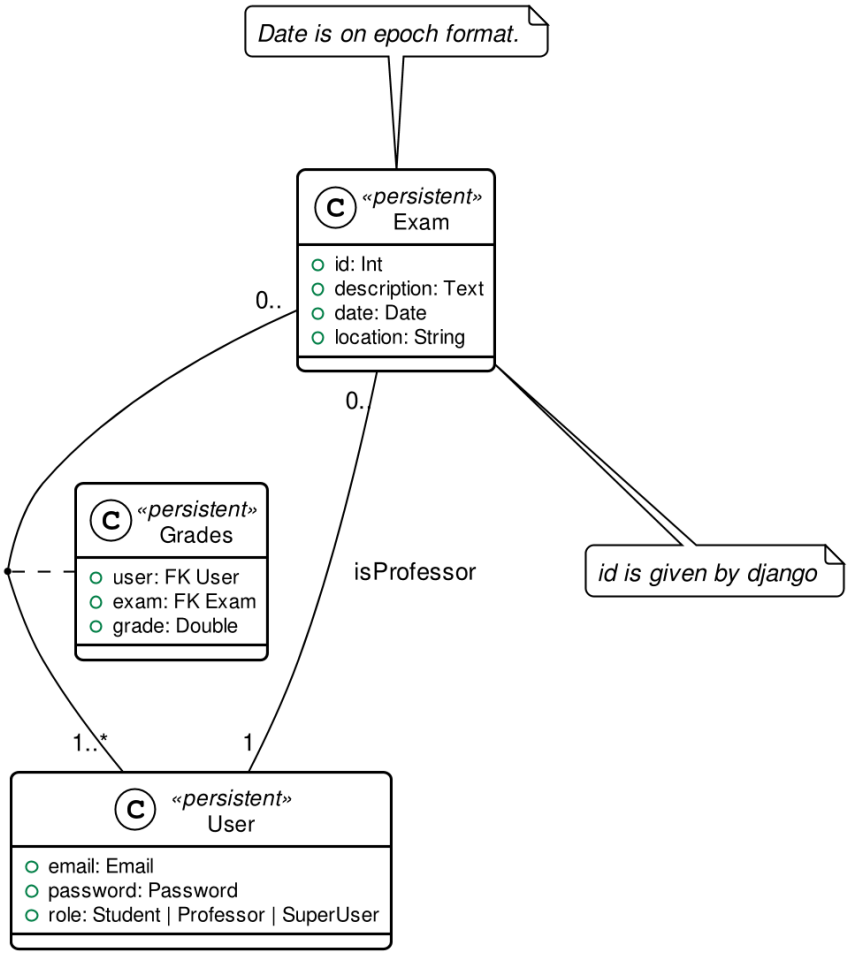
1 Introduction

In this document is it specified all the endpoints and which responses do they return, as well as a defense of which technologies we have used.

Additionally, it can be found a table containing the REST API developed, as well as a class diagram of the database model used by the API.

The integration can be found at RMI Project, at the branch `integration`. On the other hand, the Web Service can be found at WS Project.

2 UML



2.1 Exam

Exam is the class that holds all the Exam information. It stores the a description, a date and a location of an exam.

2.2 User

User is the class that stores the information of a user that is making use of our system. It's the default implementation of the Django User class.

2.3 Grades

This class it's the one that stores grades of exams made by users. It holds two foreign keys to the exam that belongs the grade, as well as the student.

3 Endpoint Table

Table 1: Methods table (Part 1).

Method	URL	What	Status code
get	exam/	List of exams	200
get	exam/{exam}/	Detail of an exam (tot)	200, 404
get	exam/search?description ={text}/	Search for a description.	200
post	exam/	Create an exam.	201, 403, 401
put	exam/{exam}/	Modify all fields of an exam	200, 403, 401
patch	exam/{exam}/	Partial update.	200, 403, 401
delete	exam/{exam}/	Deletes if user = professor and exam has no grades	204, 403, 401
post	grades/	Uploads a grade.	201, 403, 401
get	grades/{user}/user/	List of all user grades.	200
get	grades/{exam}/exam/	List of all exam grades.	200
get	grades/	List all grades.	200
get	grades/{grade-id}	Detail a grade.	200, 404
put	grades/{grade-id}	Updates a grade.	200, 403, 401
patch	grades/{grade-id}	Partially updates a grade.	200, 403, 401
delete	grades/{grade-id}	Deletes a grade.	204, 403, 401

Table 2: Methods table (Part 2).

Method	URL	What	Status code
post	auth/login/	Logins	201, 403, 401
get	auth/logout/	Logouts	200
post	auth/logout/	Logout	201, 403, 401
post	auth/password/change/	Password change.	201, 403, 401
post	auth/password/reset/	Password reset by email confirmation. Needs Email configuration	201, 403, 401
post	auth/password/reset /confirm/	Password Confirmation	201, 403, 401
post	auth/registration/	Register a new user.	201, 403, 401
post	auth/registration /verify-email	Verifies email. Needs Email configuration	201, 403, 401
get	auth/user/	Reads User. Needs authentication	200
put	auth/user/	Updates User	200, 403, 401
patch	auth/user/	Partial update.	200, 403, 401
get	user/{user}/	Gets user with pk.	200, 404

4 Screenshots

The screenshots are for the most important cases, there are endpoints that has been omitted, like user password change.

Note that due to a bug in the docs viewer, as deleting an object only returns a status code without any data, it does not correctly show that the status code is 204. Instead, only shows "undefined", even though it is properly deleted from the database.

4.1 Authentication

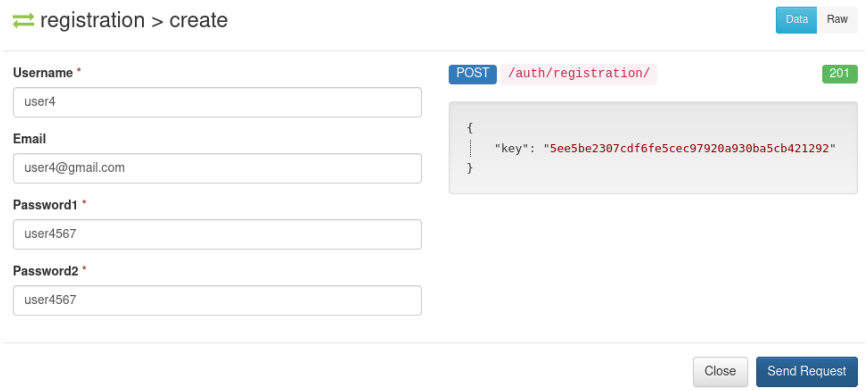


Figure 1: Register

Figure 2: Login

4.2 Exam

Figure 3: List exams

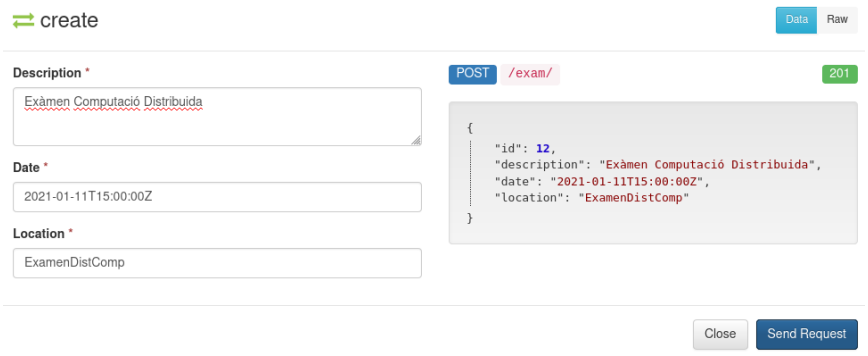


Figure 4: Create exam

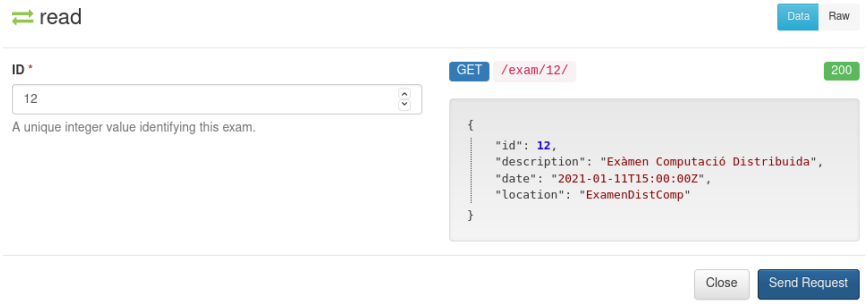


Figure 5: Read exam

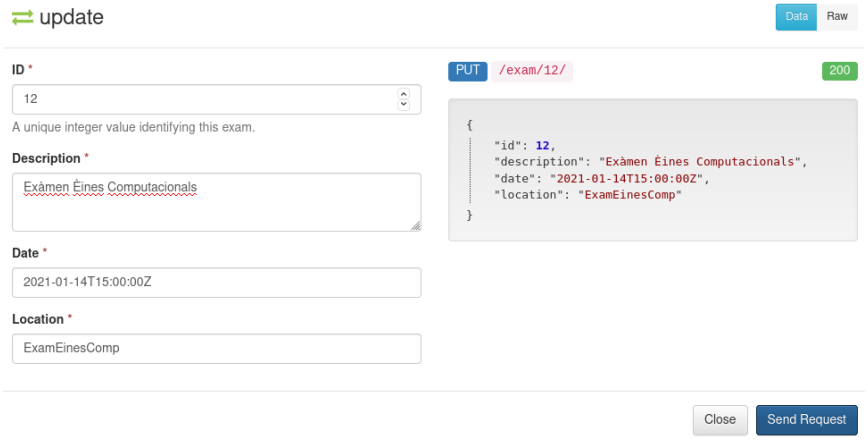


Figure 6: Update exam

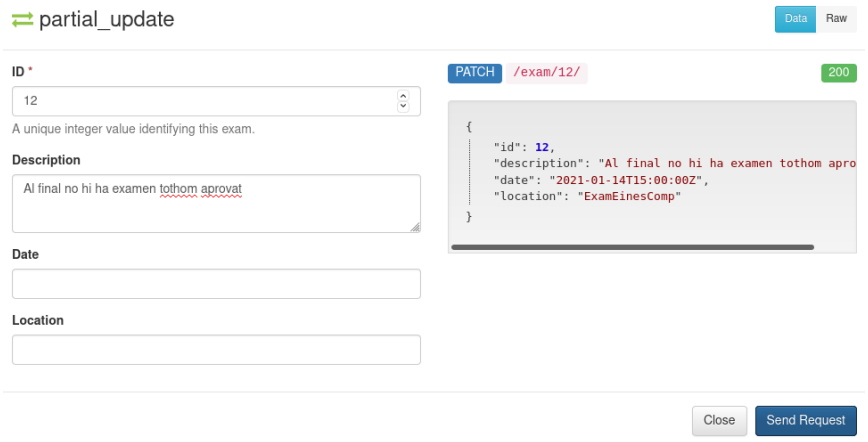


Figure 7: Patch exam

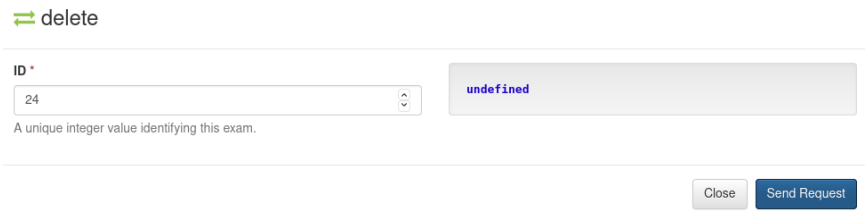


Figure 8: Delete exam

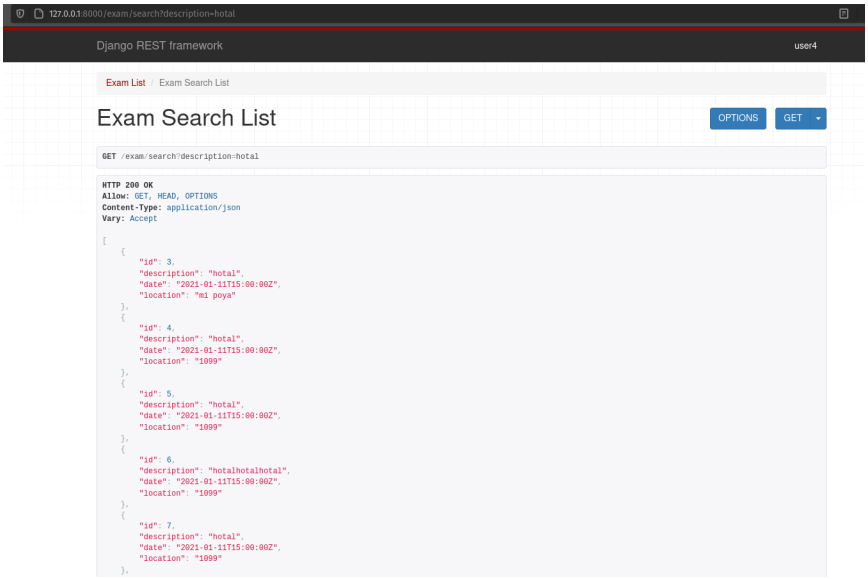


Figure 9: Search exam

4.3 Grades

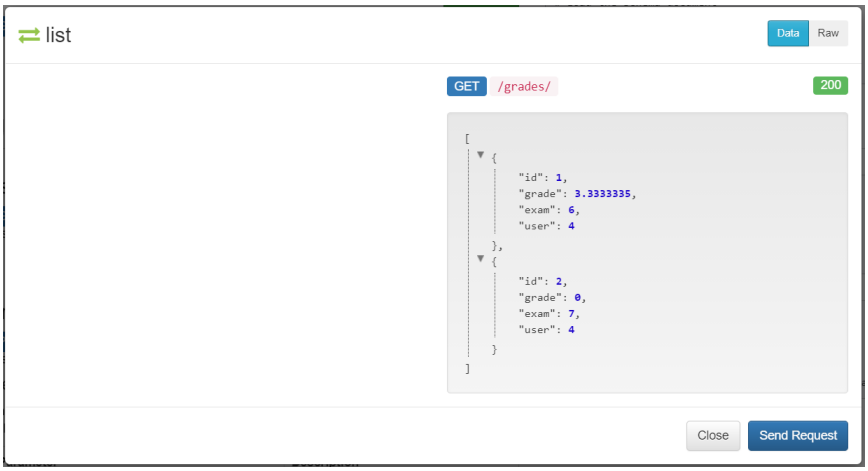


Figure 10: List grades

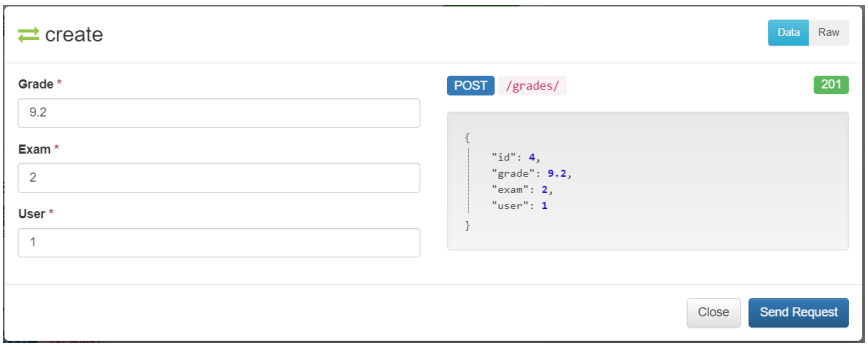


Figure 11: Create grade

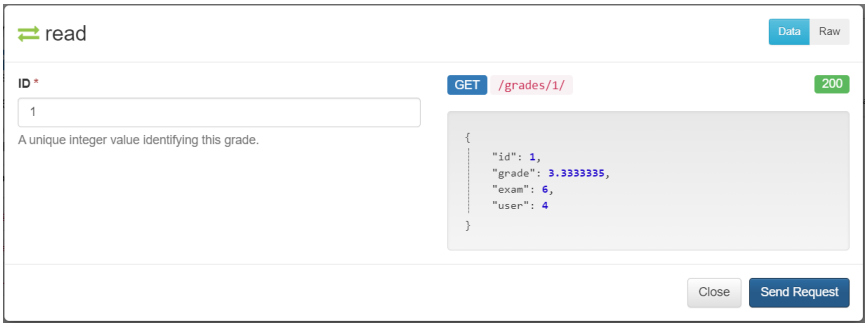


Figure 12: Read grade

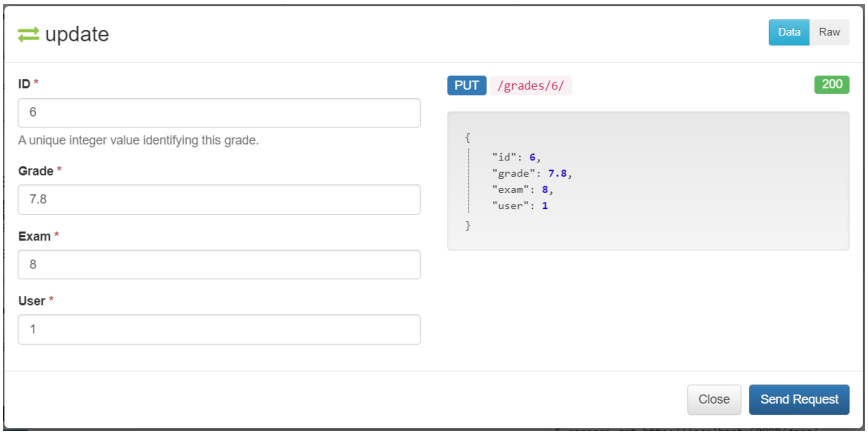


Figure 13: Update grade

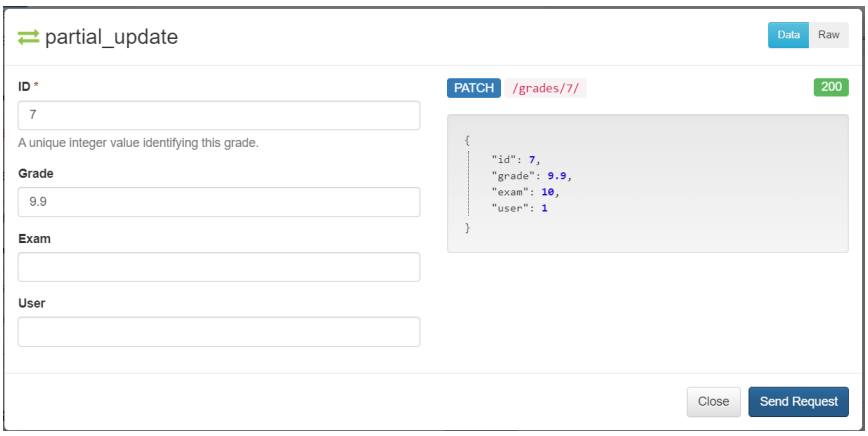


Figure 14: Patch grade

Figure 15: Delete grade

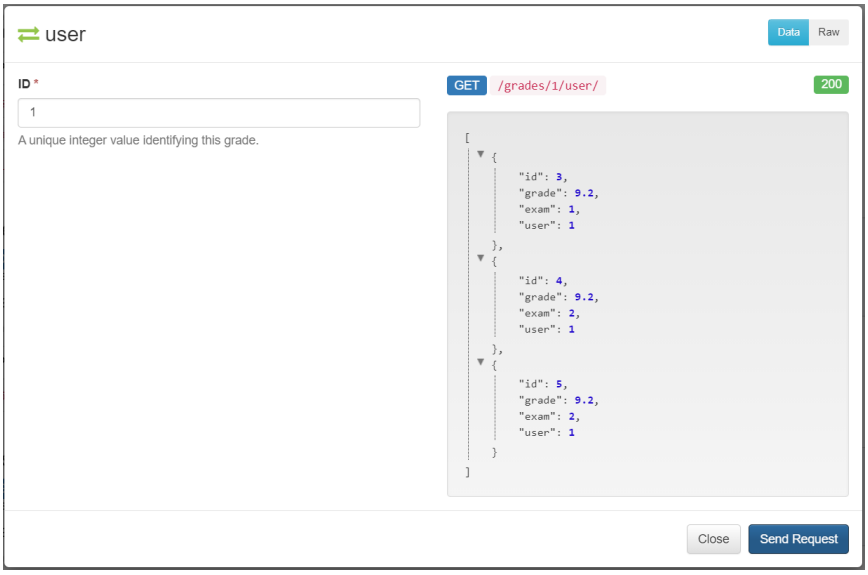


Figure 16: Search user grades

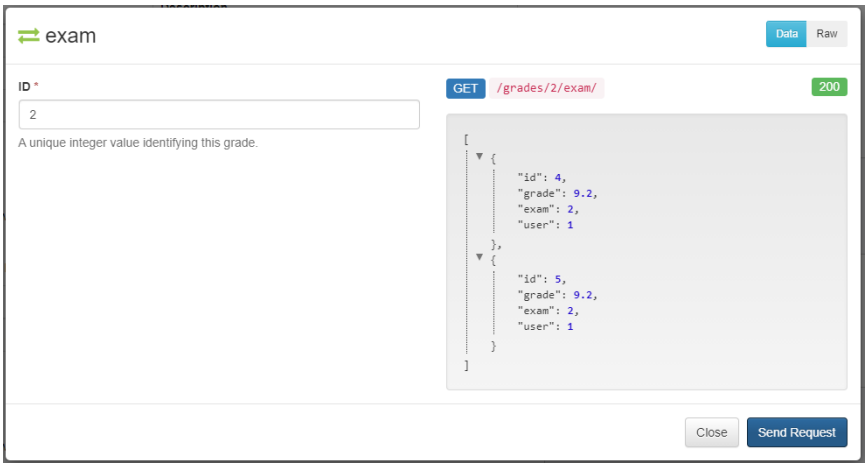


Figure 17: Search exam grades

5 How To

5.1 Getting Started

These instructions will get you a copy of the project up and running on your local machine for development and testing purposes. See deployment for notes on how to deploy the project on a live system.

5.1.1 Prerequisites

You will need to have installed docker and docker-compose. To know if this is working properly use `docker run hello-world` and `docker-compose --version`. To get them installed properly at your OS, refer to the official pages of docker and use:

```
python3 -m pip install docker-compose
```

5.1.2 Installing

Copy `example.env` to file named `.env`. Then change the variable `DJANGO_SECRET_KEY=[key]` to a value generated. For example, using this site.

So the contents of `.env` should be:

```
#Django configuration

OPEN_PORT=8000
DJANGO_PORT=8000

DJANGO_SECRET_KEY=<your secret key goes here>
DJANGO_DEBUG=1
DJANGO_ALLOWED_HOSTS=localhost 127.0.0.1 [::1] 0.0.0.0
```

```
POSTGRES_USER=postgres
POSTGRES_PASSWORD=postgres
POSTGRES_HOST=db
POSTGRES_PORT=5432
POSTGRES_NAME=postgres

DATABASE_URL="postgres://$POSTGRES_USER:
$POSTGRES_PASSWORD@$POSTGRES_HOST:
$POSTGRES_PORT/$POSTGRES_NAME"

EMAIL_OPTION=none
EMAIL_USE_TLS=True
EMAIL_HOST='smtp.gmail.com'
EMAIL_HOST_USER='mail@gmail.com'
EMAIL_HOST_PASSWORD='password1234'
EMAIL_PORT=587
```

Then apply the changes to your database using:

```
docker-compose up -d
docker-compose exec web python3 manage.py makemigrations
docker-compose exec web python3 manage.py migrate
docker-compose down
```

To create a super user, use:

```
docker-compose up -d
docker-compose exec web python3 manage.py createsuperuser
```

```
docker-compose down
```

Then use `docker-compose up -d` to get it running. Connect to `localhost:8000/admin` to see the admin login page, or `localhost:8000/docs` to see the docs.

To stop it, use `docker-compose down`

5.2 Running the tests

To execute all tests, use `docker-compose exec web python3 manage.py test`

6 Solution justification

6.1 Web Service

6.1.1 Technologies

Django We have chosen this technology because our familiarity with it and its ease to work with data models and ORM.

Django rest framework This framework is a powerful and easy-to-use tool for building web REST API's, it includes mechanisms for serialization and authentication, which we found necessary.

SQLite it is the Django default database. A PostgreSQL can be configured as a replacement for scalability and deployment purposes. It is already specified in the environment, but was left as SQLite was sufficiently for the requirements.

Docker It facilitates the encapsulation and execution of the project, as it is contained in a container.

Docker-compose Easier configuration for a docker.

6.1.2 ViewSets and Generics

Django is an opinionated framework. With this, it provides powerful abstraction if you can manage to use them. Django REST Framework, based on it, *copies* some of their abstractions and provides them for a RESTful API. For example, in Django we extend View classes and add them some information about which HTML template to use and which database model, and it will pass correctly the data.

With the REST framework, we have a similar idea. We have the concept of generics, that provides a unique endpoint to an action, as retrieving an object from the database or listing a few of them. When they did this, they saw that most of their implementations used the same parameters: where to get the objects and how to serialize them. And for this reason they build what is called `ViewSets`. They provide an abstraction to build all the `CRUD` operations of a model in the database. In conjunction with the permissions class, they can provide a quick and robust way to deploy the API. Most of our endpoints are made with this `ViewSets`, the only ones that don't use them are Filtering Views as they were made with a custom `ListAPIViews` and a custom `get_queryset` function.

A user detail is not provided by the `auth` API, but it was needed for the presentation, so we made a custom endpoint to read a specific User.

6.1.3 Decisions

Authentication We developed a simple authentication in which users once registered and logged are provided with a token. This provides a way of authentication against some endpoints in the WS, as POSTs and DELETEs. There are custom permissions to prevent forbidden actions, like a student deleting an exam, or modifying a grade. We used `dj-rest-auth`, which provides endpoints for registration, authentication, password reset, retrieve and update user details, etc.

We also used `django-all-auth`, which implements a powerful back-end to registration. It also provides with a plug-and-play of social authentication, (i.e.: login with your Google account), and email verification. Although we initially made an Email back-end, we needed to provide in the environment either a usable email or an email provider. We made a special parameter, so they are not needed, as we thought that this will cause some trouble when correcting the project rather than being a feature.

6.2 RMI modifications

HTTP We have made two adapter classes in order to encapsulate the HTTP requests made to the web service by the client and the server. To make the request we have used `OkHttp3`, as we were restricted to use a library from before Java 8 because of RMI deprecation in Java 9, but we initially intended to use HTTP of Java 11. We were unable to mock and test the API calls because `OkHttp3` Request and Response object does not implement equals, and are final.

Client flow changes Now the client has to be identified in order to enter

the exam session, so the first step is to ask for a correct username and password. Once authenticated correctly the user is given 3 options:

search <keywords> searches exams by its description and outputs the information of the matched exams.

list lists and outputs all the exams and its information.

choose <id-exam> choose the desired exam in order to connect to its session. Once an exam is chosen, the flow works as before.

Server flow changes As happens with the client, the professor has to be identified in order to create an exam session, so the first step is to ask for a correct username and password. Once authenticated correctly it will be asked to introduce the following parametters in order to create the exam:

description The description of an exam.

date Date of an exam. It needs a specific date format, as `YYYY-MM-DDThh:mm:ssZ`.

location The location of an exam (string). We decided that the location will be the bind key of the remote object that references the exact exam session.

Once the last parameter is filled, the exam will be created in the web service, as well as the session in which the students can connect to perform the exam. When a professor finishes an exam, all the grades are uploaded to the web service.

6.3 Time dedicated

It is difficult to say, but we estimate an approximation of 90 hours. We are a group of three students, and we worked in this project for 6 days, 5 hours each day.