

ACTIVITY 6 - WS

JOAQUIM PICÓ MORA, IAN
PALACÍN ALIANA, SERGI SIMÓN
BALCELLS

Created: 2021-01-11 Mon 13:43

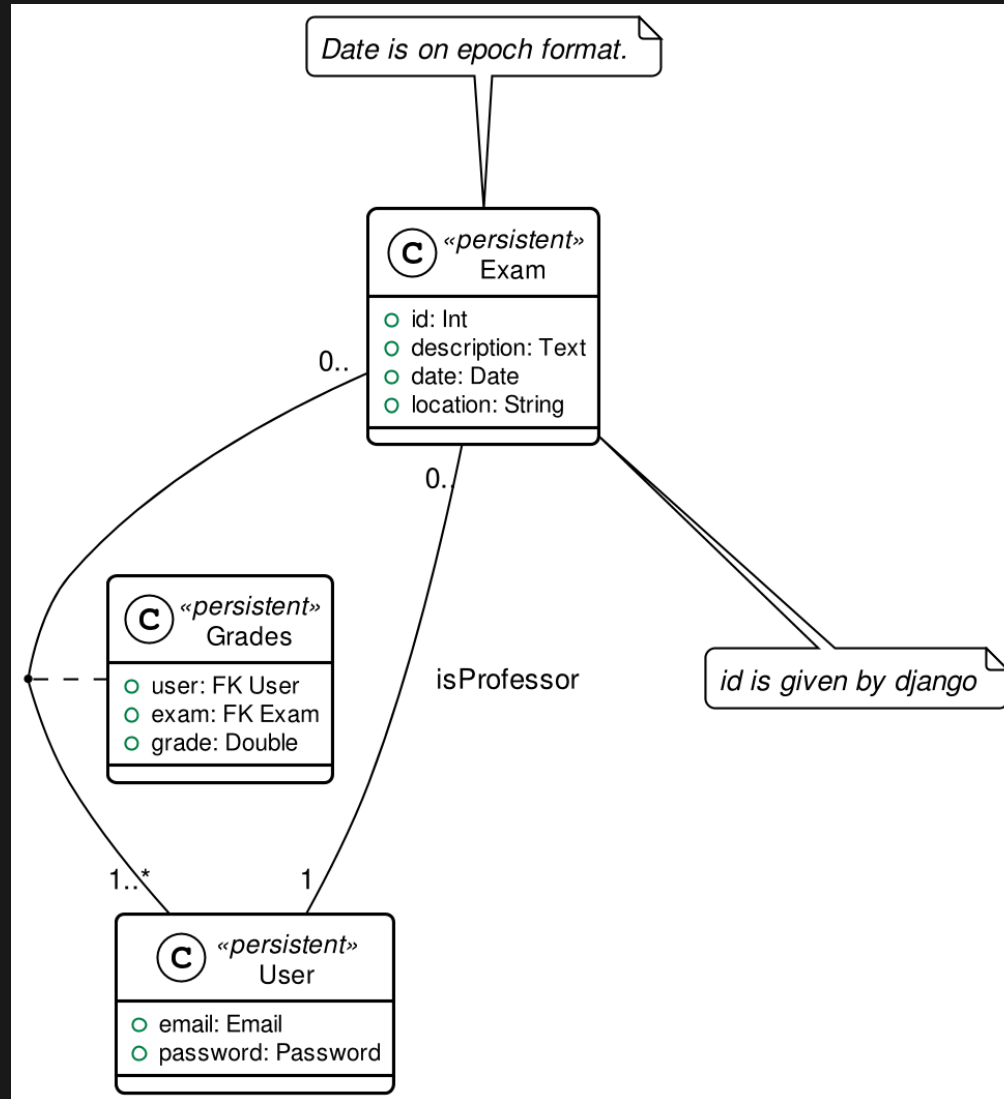
1 INTRODUCTION

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

Method	URL	What	Status code
get	exam/	List all the exams	200
get	exam/{exam}/	Detail of an exam	200, 404
get	exam/search? description={text}/	Search a description	200

post	exam/	Creates 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 is professor and has no grades	204, 403, 401
--------	--------------	--	---------------

post	grades/	Uploads an exam.	201, 403, 401
------	---------	------------------	---------------

get	grades/	List all grades.	200
-----	---------	------------------	-----

get	grades/{grade-id}	Detail a grade.	200, 404
get	grades/{user}/user/	List all grades of a user	200
get	grades/{user}/exam/	List all grades of an exam	200

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

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.

post auth/password/reset/

Password
reset by
email
confirmation.
Needs Email
configuration

post auth/password/reset/confirm/

Password
Confirmation

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

↔ registration > create Data Raw

Username *

Email

Password1 *

Password2 *

POST **/auth/registration/** **201**

```
{
  "key": "5ee5be2307cdf6fe5cec97920a930ba5cb421292"
}
```

Close Send Request

Figure 2: Register

login > create

DataRaw

Username

user4

Email

Password *

user4567


POST /auth/login/ 200

```
{
  "key": "5ee5be2307cdf6fe5cec97920a930ba5cb421292"
}
```

Close

Send Request

Figure 3: Login

 read

ID *

1

A unique integer value identifying this user.

GET

/user/1/

200

```
{
  "id": 1,
  "username": "sergi",
  "email": "sergi@sergi.com"
}
```

Close

Send Request

Figure 4: User retrieve.

4.2 EXAM

The screenshot shows a REST client interface with a 'list' tab selected. A GET request to the endpoint '/exam/' has been executed, returning a 200 status code. The response is displayed in a JSON format, showing a list of four exam items. Each item contains an id, a description, a date, and a location. The interface includes buttons for 'Data' and 'Raw' views, and a 'Send Request' button at the bottom right.

```
[
  {
    "id": 1,
    "description": "Hola calvo",
    "date": "2021-01-11T15:00:00Z",
    "location": "St"
  },
  {
    "id": 2,
    "description": "hola",
    "date": "2021-01-11T15:00:00Z",
    "location": "st"
  },
  {
    "id": 3,
    "description": "hotal",
    "date": "2021-01-11T15:00:00Z",
    "location": "mi poya"
  },
  {
    "id": 4,
    "description": "hotal",
    "date": "2021-01-11T15:00:00Z",
    "location": "1099"
  }
]
```

Figure 5: List exams

 create

Description *

Exàmen Computació Distribuida

Date *

2021-01-11T15:00:00Z

Location *

ExamenDistComp

POST

/exam/


201

```
{
  "id": 12,
  "description": "Exàmen Computació Distribuida",
  "date": "2021-01-11T15:00:00Z",
  "location": "ExamenDistComp"
}
```

Close

Send Request

Figure 6: Create exam

 read

ID *

12

^

v

A unique integer value identifying this exam.

GET

/exam/12/

200

```
{
  "id": 12,
  "description": "Exàmen Computació Distribuida",
  "date": "2021-01-11T15:00:00Z",
  "location": "ExamenDistComp"
}
```

Close

Send Request

Figure 7: Read exam

↔ update

Data

Raw

ID *

12

A unique integer value identifying this exam.

Description *

Exàmen Èines Computacionals

Date *

2021-01-14T15:00:00Z

Location *

ExamEinesComp

PUT

/exam/12/

200

```
{
  "id": 12,
  "description": "Exàmen Èines Computacionals",
  "date": "2021-01-14T15:00:00Z",
  "location": "ExamEinesComp"
}
```

Close

Send Request

Figure 8: Update exam

⇌ partial_update

Data Raw

ID *

12

A unique integer value identifying this exam.

Description

Al final no hi ha examen tothom aprovat

Date

Location

PATCH /exam/12/


200

```
{
  "id": 12,
  "description": "Al final no hi ha examen tothom apro
  "date": "2021-01-14T15:00:00Z",
  "location": "ExamEinesComp"
}
```

Close

Send Request

Figure 9: Patch exam

 delete

ID *

↑
↓

undefined

A unique integer value identifying this exam.

Close

Send Request

Figure 10: Delete exam

127.0.0.1:8000/exam/search?description=hotal

Django REST framework user4

Exam List / Exam Search List

Exam Search List

OPTIONS GET

GET /exam/search?description=hotal

HTTP 200 OK
Allow: GET, HEAD, OPTIONS
Content-Type: application/json
Vary: Accept

```
[
  {
    "id": 3,
    "description": "hotal",
    "date": "2021-01-11T15:00:00Z",
    "location": "mi poya"
  },
  {
    "id": 4,
    "description": "hotal",
    "date": "2021-01-11T15:00:00Z",
    "location": "1099"
  },
  {
    "id": 5,
    "description": "hotal",
    "date": "2021-01-11T15:00:00Z",
    "location": "1099"
  },
  {
    "id": 6,
    "description": "hotalhotalhotal",
    "date": "2021-01-11T15:00:00Z",
    "location": "1099"
  },
  {
    "id": 7,
    "description": "hotal",
    "date": "2021-01-11T15:00:00Z",
    "location": "1099"
  }
]
```

Figure 11: Search exam


4.3 GRADES

The screenshot shows a REST client interface with a 'list' tab. A GET request to the endpoint `/grades/` has been made, resulting in a 200 status code. The response is a JSON array containing two objects, each representing a grade record.

```
[
  {
    "id": 1,
    "grade": 3.3333335,
    "exam": 6,
    "user": 4
  },
  {
    "id": 2,
    "grade": 0,
    "exam": 7,
    "user": 4
  }
]
```

At the bottom right, there are 'Close' and 'Send Request' buttons.

Figure 12: List grades

 create

Grade *

9.2

Exam *

2

User *

1

POST

/grades/

201

```
{
  "id": 4,
  "grade": 9.2,
  "exam": 2,
  "user": 1
}
```

Close

Send Request

Figure 13: Create grade

 read

ID *

A unique integer value identifying this grade.

GET

/grades/1/


200

```
{
  "id": 1,
  "grade": 3.3333335,
  "exam": 6,
  "user": 4
}
```

Close

Send Request

Figure 14: Read grade

 update

ID *

A unique integer value identifying this grade.

Grade *

Exam *

User *

PUT

/grades/6/


200

```
{  
  "id": 6,  
  "grade": 7.8,  
  "exam": 8,  
  "user": 1  
}
```

Close

Send Request

Figure 15: Update grade

 partial_update

ID *

A unique integer value identifying this grade.

Grade

Exam

User

PATCH

/grades/7/


200

```
{
  "id": 7,
  "grade": 9.9,
  "exam": 10,
  "user": 1
}
```

Close

Send Request

Figure 16: Patch grade

 delete

ID *

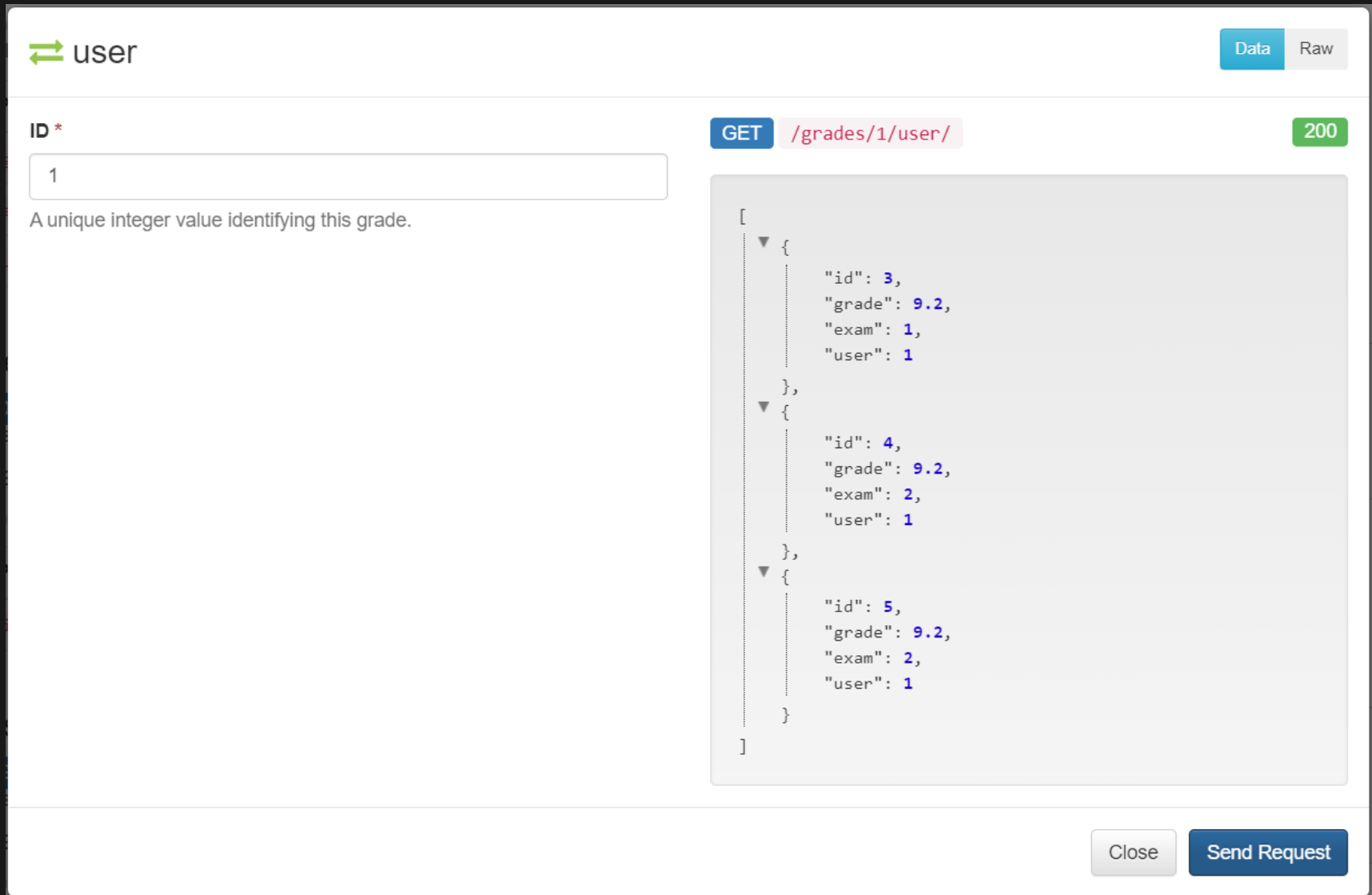
undefined

A unique integer value identifying this grade.

Close

Send Request

Figure 17: Delete grade



↔ user

DataRaw

ID *

1


A unique integer value identifying this grade.

GET /grades/1/user/ 200

```
[
  {
    "id": 3,
    "grade": 9.2,
    "exam": 1,
    "user": 1
  },
  {
    "id": 4,
    "grade": 9.2,
    "exam": 2,
    "user": 1
  },
  {
    "id": 5,
    "grade": 9.2,
    "exam": 2,
    "user": 1
  }
]
```

Close Send Request

Figure 18: Search user grades

 exam

ID *

A unique integer value identifying this grade.

GET

/grades/2/exam/

200

```
[
  {
    "id": 4,
    "grade": 9.2,
    "exam": 2,
    "user": 1
  },
  {
    "id": 5,
    "grade": 9.2,
    "exam": 2,
    "user": 1
  }
]
```

Close

Send Request

Figure 19: Search exam grades

5 SOLUTION JUSTIFICATION

5.1 WEB SERVICE

5.1.1 TECHNOLOGIES

5.1.2 VIEWSETS AND GENERICS

5.1.3 DECISIONS: AUTHENTICATION

5.2 RMI MODIFICATIONS

5.2.1 HTTP

5.2.2 CLIENT FLOW CHANGES

1. Search
2. List
3. Choose

5.2.3 SERVER FLOW CHANGES

1. Description
2. Date
3. Location

5.3 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.