Project 3

Redis and DynamoDB Local

GROUP 1
Luis Alverado
Marco Gabriel
Michael Carey
Rishub Goel
Nhan Mac
CPSC 449-01 13661
Fall Semester

Table of Contents

Install and configure the AWS CLI

Context: Installing AWS CLI

```
(.venv) student@tuffix-vm:~$ sudo ./aws/install
You can now run: /usr/local/bin/aws --version
```

```
(.venv) student@tuffix-vm:~$ aws --version
aws-cli/2.13.32 Python/3.11.6 Linux/6.2.0-36-generic exe/x86_64.ubuntu.22 prompt/off
```

Context: Configuring dummy credentials for DynamoDB local

```
(.venv) student@tuffix-vm:~$ aws configure
AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
Default region name [None]: us-west-2
Default output format [None]: json
(.venv) student@tuffix-vm:~$
```

Install and configure Amazon DynamoDB local

```
(.venv) student@tuffix-vm:~$ sudo apt install --yes openjdk-19-jre-headless
Reading package lists... Done
Building dependency tree... Done
Reading state information... Done
The following packages were automatically installed and are no longer required:
  gir1.2-snapd-1 python3-debconf update-notifier-common
Use 'sudo apt autoremove' to remove them.
The following additional packages will be installed:
  ca-certificates-java java-common
```

```
(.venv) student@tuffix-vm:~/Desktop$ java -Djava.library.path=./DynamoDBLocal_lib -jar DynamoDBLocal.jar -sharedDb
Initializing DynamoDB Local with the following configuration:
Port:   8000
InMemory:       false
DbPath: null
SharedDb:       true
shouldDelayTransientStatuses:   false
CorsParams:     null
```

```
(.venv) student@tuffix-vm:~/Desktop/CPSC449-Project_3$ aws dynamodb list-tables --endpoint-url http://localhost:5500
{
    "TableNames": [
        "enrollment_class",
        "enrollment_user"
    ]
}
```

Install the AWS SDK for Python

New Database Schema For Dynamodb and Redis

Design a data model for each database

Context: There are two tables located in the DynamoDB database, the class table and the user table. For both the class and user table, the id is the partition key, with a key type of HASH. Both also have a Global Secondary Index (GSI) called index-id, that uses the id attribute.

Dynamodb Schema

Class table:

- id: int
- name: str
- course_code: str
- section_number: int
- current_enroll: int
- max_enroll: int
- department: str
- instructor_id: int
- enrolled: list (of student_ids)
- dropped: list (of student_ids)

User table:

- id: int,
- name: str,
- roles: list (of strings)

Redis

Class waitlists, with key:

- "class:{class_id}:waitlist"
- format {class_id : [(student_id, placement)]}

Student waitlists, with key:

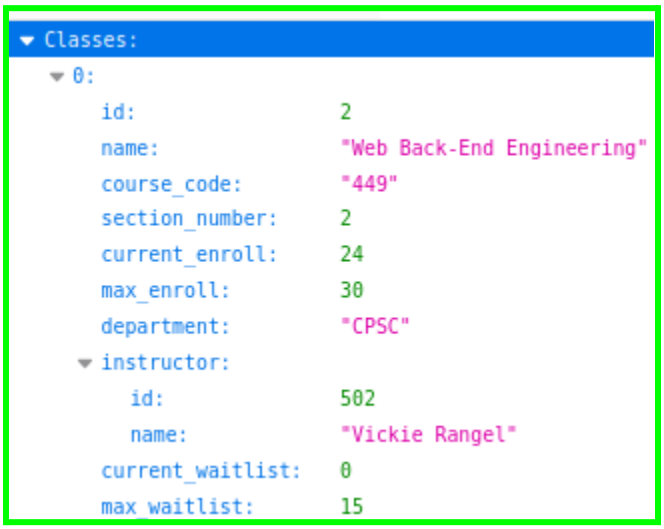
- "Student:{student_id}:waitlists"
- format {student_id: [(class_id, placement)]}

Student Endpoint

GET: Available classes

Context: We test the endpoint to see if we are able to get available classes. We use student 1 to see their classes that they are able to enroll in.

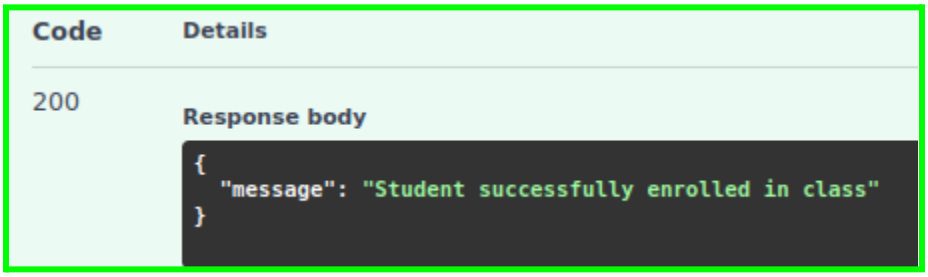
Endpoint: http://0.0.0.0:5000/students/1/classes

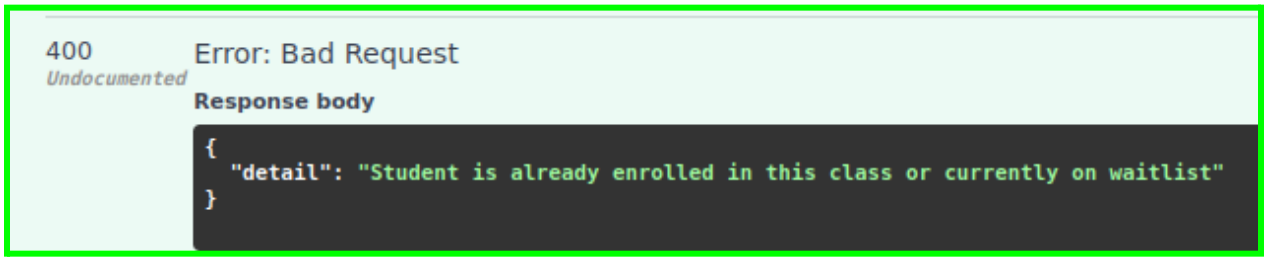


POST: Enroll into an available class

Context: We test the endpoint to check if a student is enrolled. We use student 1 to enroll them into class 2. If they are not enrolled we get a response back stating the student was enrolled successfully else we get a different response as in screenshot 2.

Endpoint: http://0.0.0.0:5000/students/1/classes/2/enroll

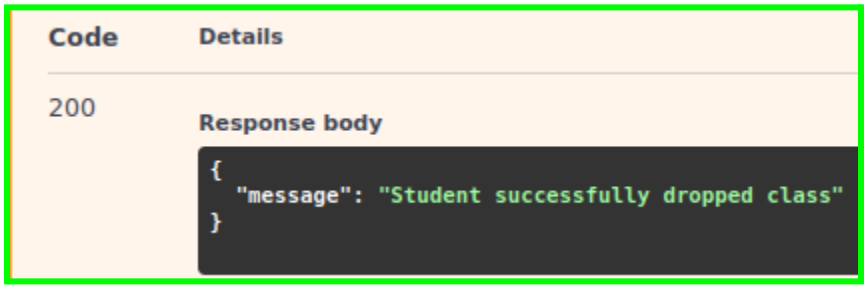


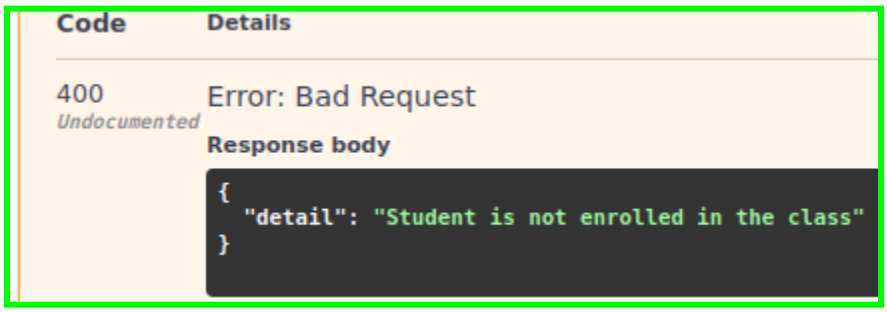


PUT: Drop a class

Context: We test the end point to check if the student is able to be dropped from the class. We use student 1 to get dropped from class 1. If the student is not enrolled in a class we get a different response such as the second screenshot.

Endpoint:http://0.0.0.0:5000/students/1/classes/2/drop/



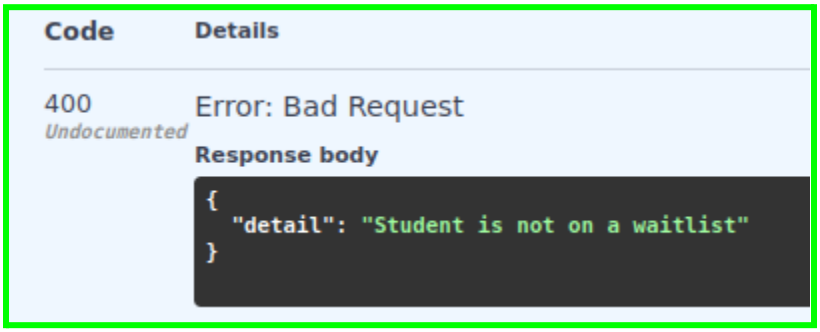


Waitlist Endpoint

GET: View waiting List from Student

Context: We test the endpoint to fetch the students' waitlist. We use student 1 to see their current waitlist. If a student is not on a waitlist we get a response that states the student is not on any waitlists.

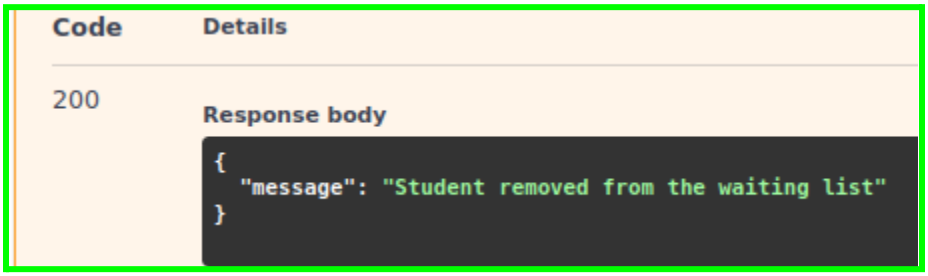
Endpoint: http://0.0.0.0:5000/waitlist/students/1

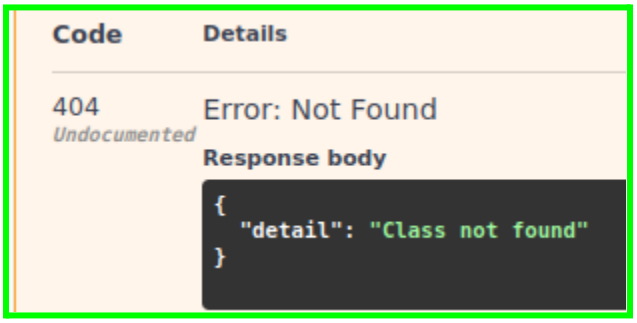


PUT: Remove from the waitlist

Context: We test this endpoint to check if we are able to remove a student from the waitlist. We use student 1 and class 8 to remove them from the waitlist. If it is successful we get a message stating that it was able to remove the student. If the student is not on any waitlist it will give us a response as the second screenshot.

Endpoint: http://0.0.0.0:5000/waitlist/students/1/classes/8/drop





GET: View current waitlist from Instructor

Context: We test this endpoint to see if we are able to get the waitlists based of an instructor. We use instructor 508 to retrieve its class 8's waitlist. If the instructor has no waitlist we get a response just like in the second screenshot.

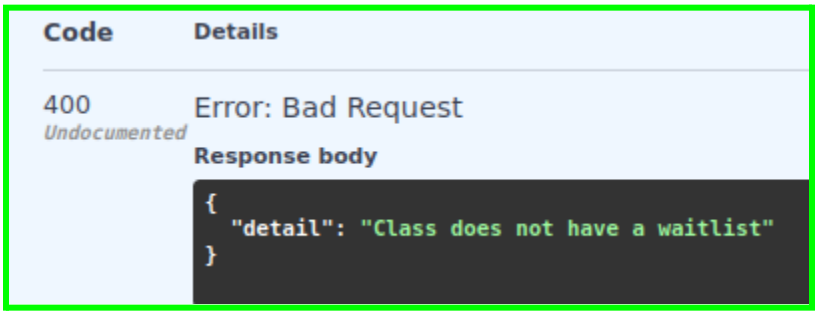
Endpoint: http://0.0.0.0:5000/waitlist/instructors/508/classes/8

Code | Details

200

Response body


```
{
  "Waitlist": [
    {
      "student": {
        "id": 1,
        "name": "James Smith"
      },
      "waitlist_position": 1
    }
  ]
}
```

Code | Details

400
Undocumented

Error: Bad Request

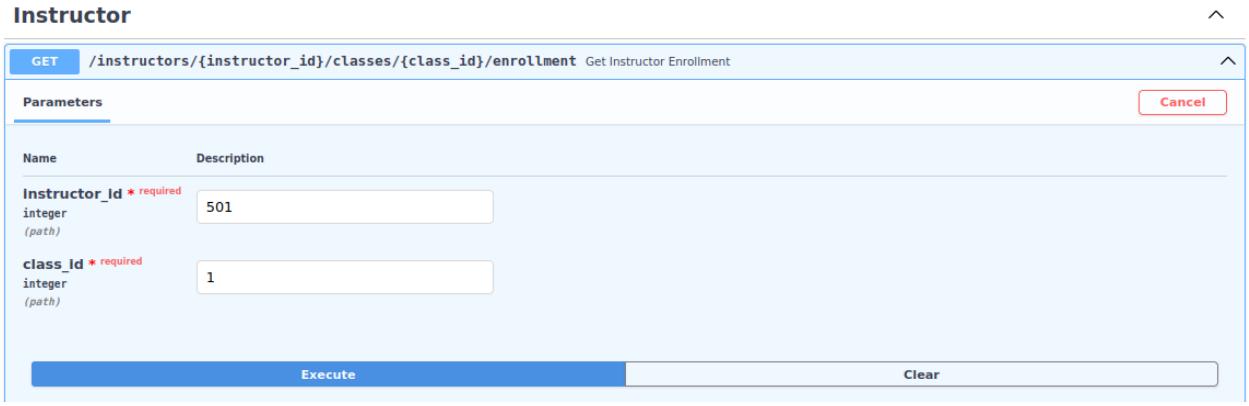
Response body


```
{
  "detail": "Class does not have a waitlist"
}
```

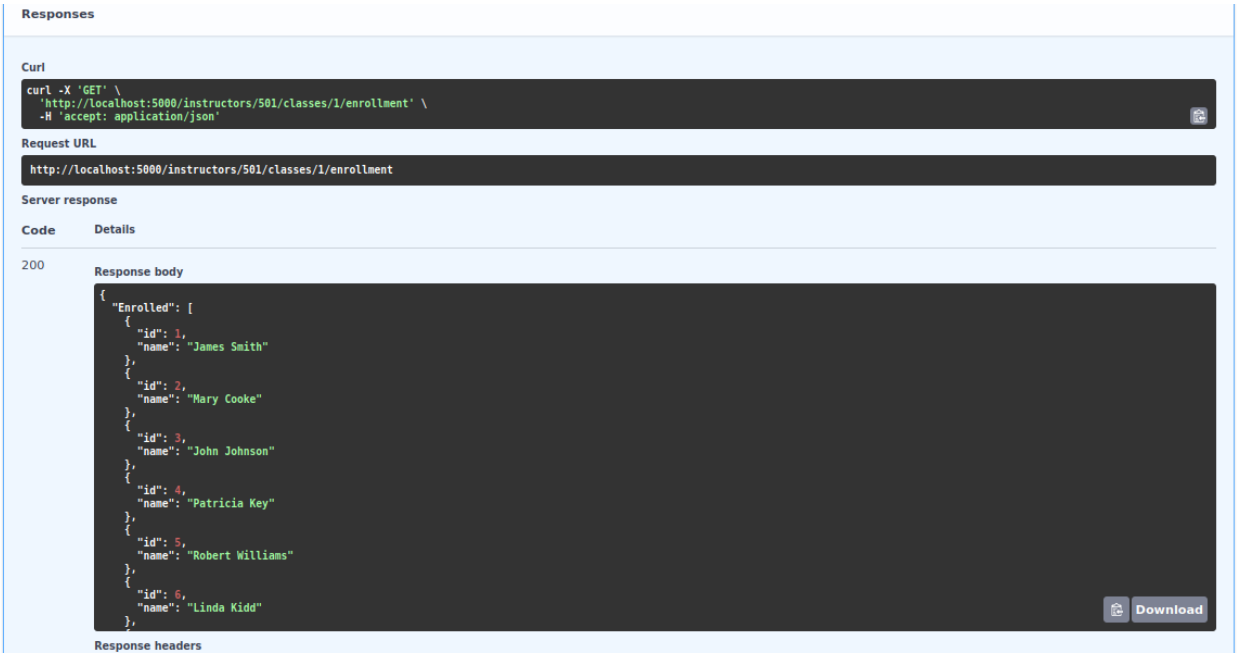
Instructor Endpoint

Get Instructor Enrollment

Context: I will try to get the enrollment of Instructor id 501 with class id 1.

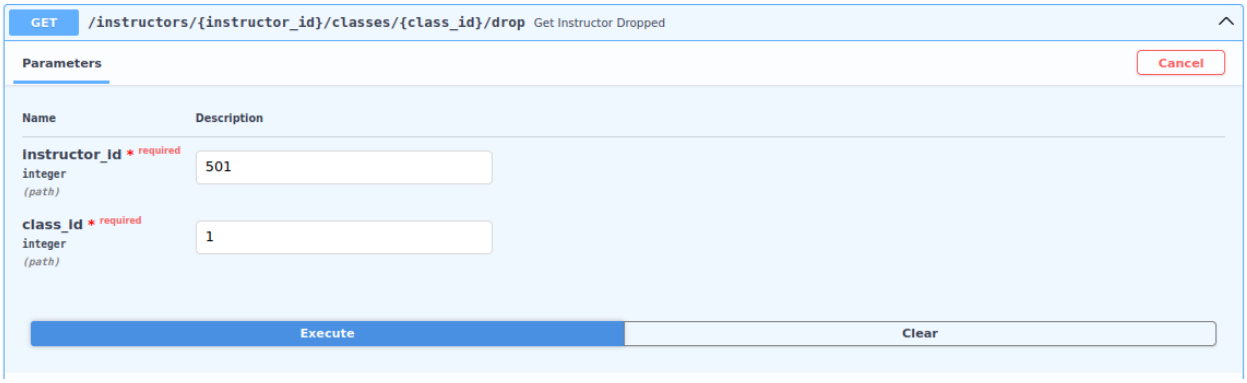


Result: We ended up getting 10 students enrolled in the class.

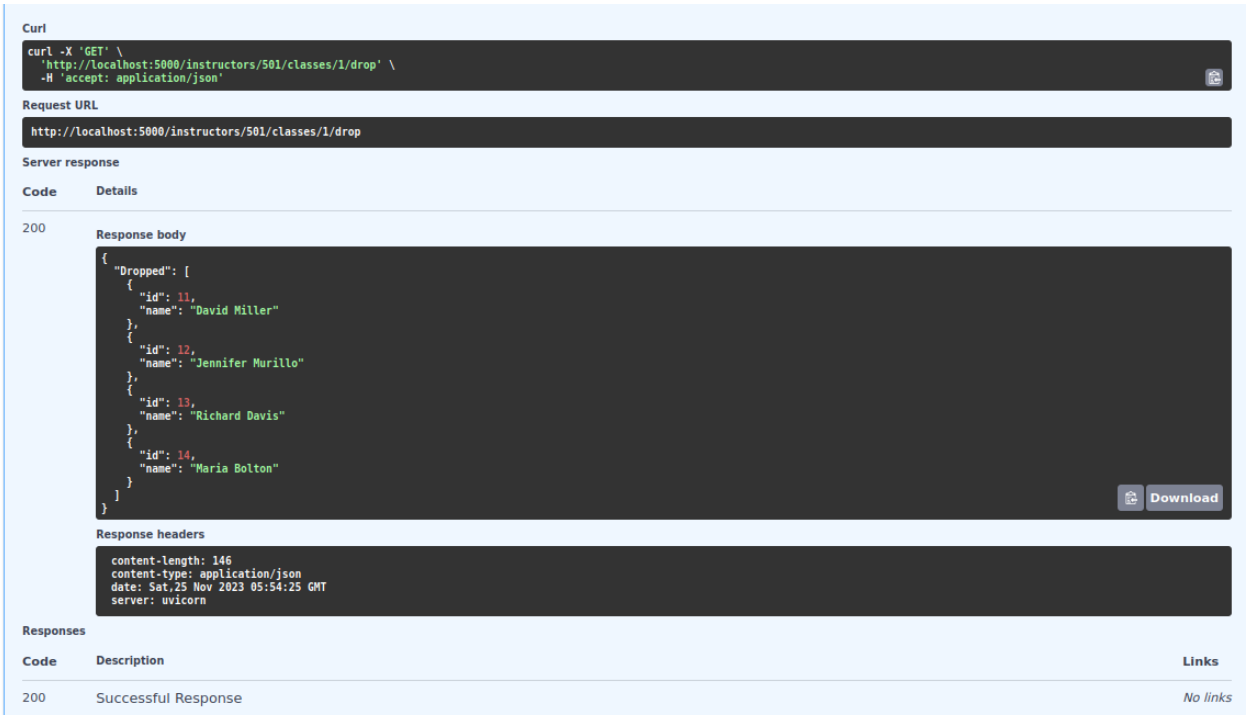


Get Instructors Dropped Students

Context: Instructor_id 501 class id 1 has 4 students present to drop.

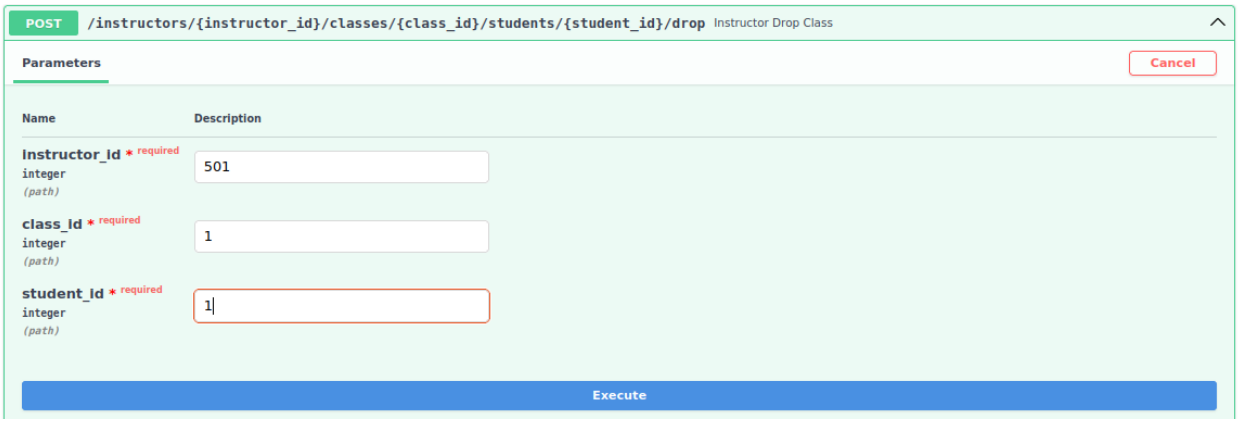


Result: When we make an API call to retrieve the list of dropped students we get student id's 11-14.



Instructor Drop Student

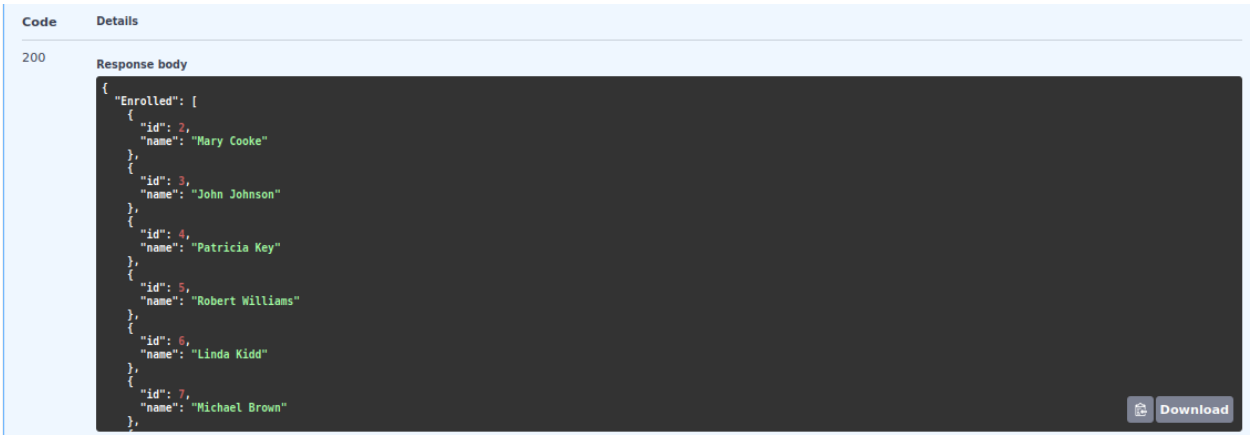
Context: Testing Endpoint using Instructor id = 1, class_id = 1, and student_id = 1 (James Smith).



Result: In our endpoint, we get a message saying that “the student was successfully dropped”

Context: Now, we will backtrack to the instructor enrollment list and dropped list to verify if the student was successfully dropped from the class and to ensure the lists get updated properly.

Test - Instructor Enrollment List Endpoint: We can see that id: 1 does not exists in the enrollment list of class_id:1 and instructor_id:501 anymore.



Code Details

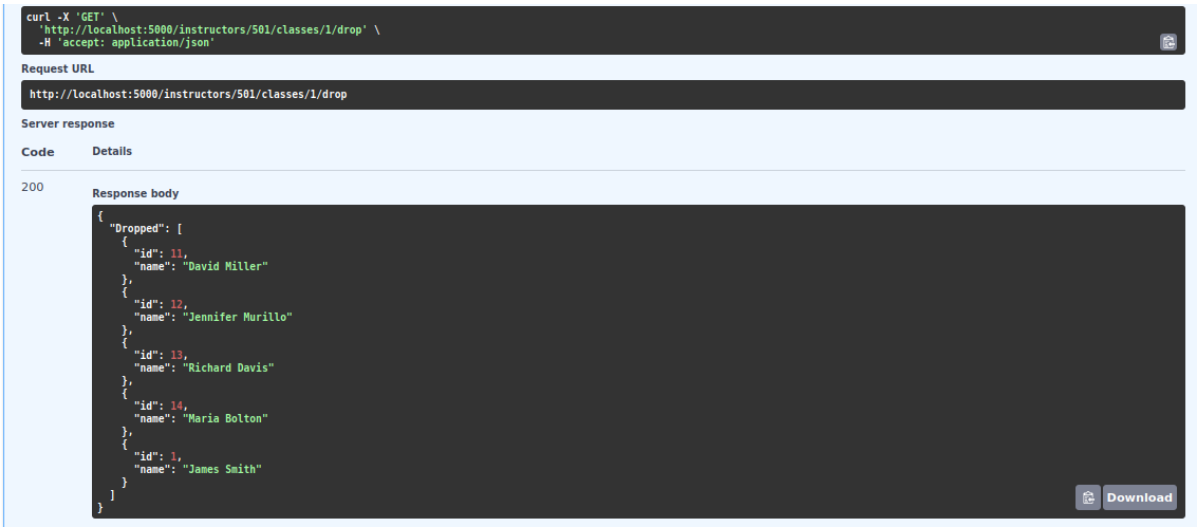
200

Response body


```
{
  "Enrolled": [
    {
      "id": 2,
      "name": "Mary Cooke"
    },
    {
      "id": 3,
      "name": "John Johnson"
    },
    {
      "id": 4,
      "name": "Patricia Key"
    },
    {
      "id": 5,
      "name": "Robert Williams"
    },
    {
      "id": 6,
      "name": "Linda Kidd"
    },
    {
      "id": 7,
      "name": "Michael Brown"
    },
```


Download

Test - Instructor Dropped List Endpoint: We will now look at the dropped list endpoint and now we can see student id 1 “James Smith” in the list.



```
curl -X 'GET' \
  'http://localhost:5000/instructors/501/classes/1/drop' \
  -H 'accept: application/json'
```

Request URL

```
http://localhost:5000/instructors/501/classes/1/drop
```

Server response

Code Details

200

Response body


```
{
  "Dropped": [
    {
      "id": 11,
      "name": "David Miller"
    },
    {
      "id": 12,
      "name": "Jennifer Murillo"
    },
    {
      "id": 13,
      "name": "Richard Davis"
    },
    {
      "id": 14,
      "name": "Maria Bolton"
    },
    {
      "id": 1,
      "name": "James Smith"
    }
  ]
}
```

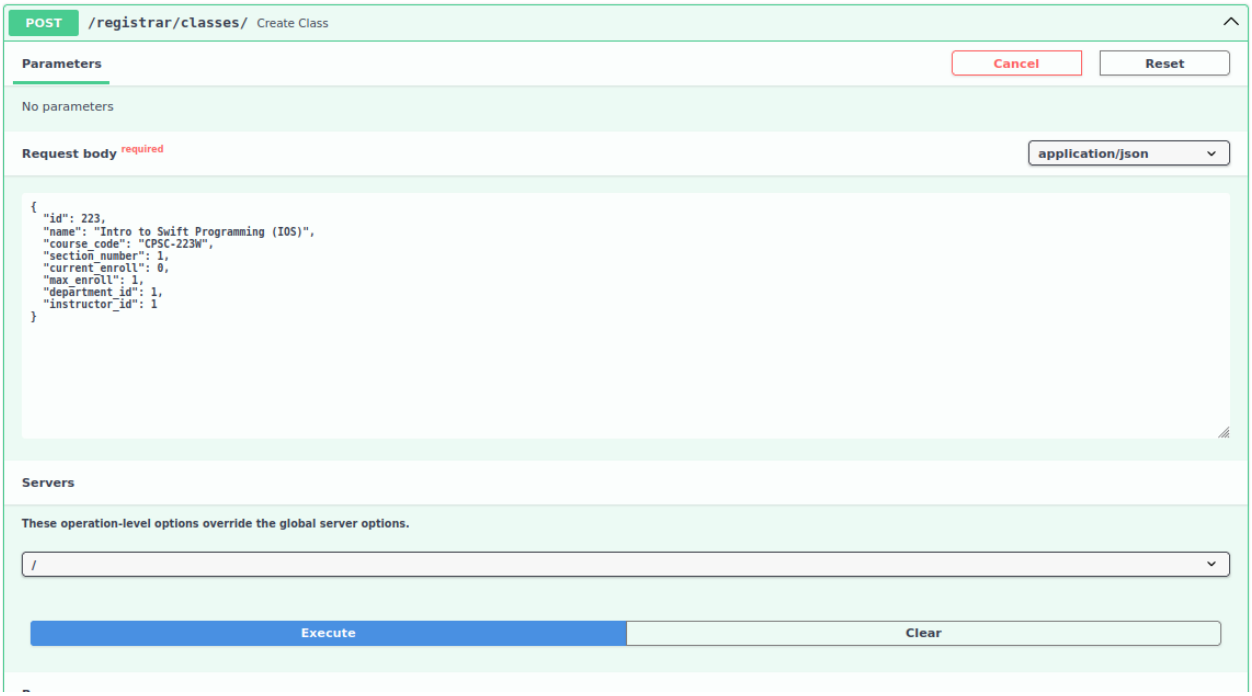

Download

Context: If the Instructor decides to remove a student again we have a constraint set up to ensure it notifies the instructor that the student no longer exists in the class.

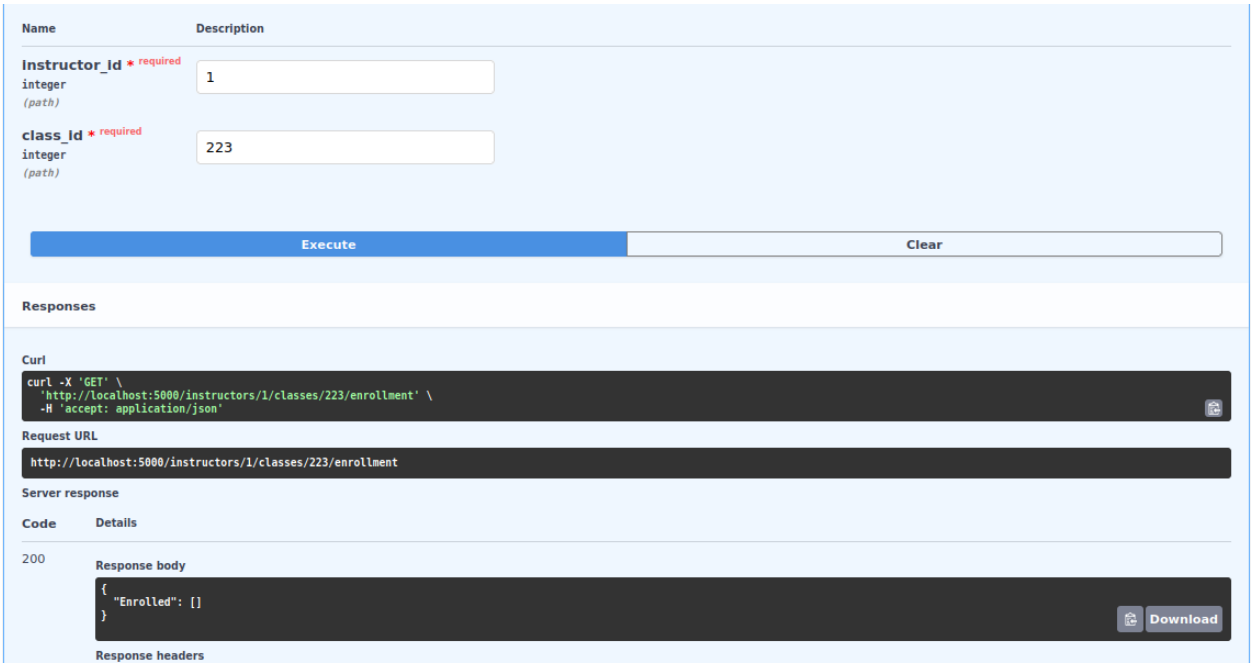
Registrar Endpoint

POST: Create Class

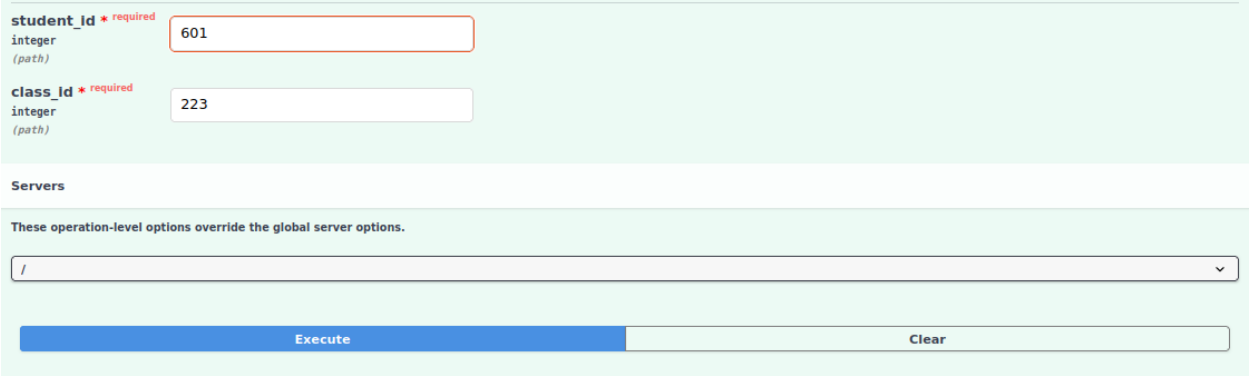
Context: Creating a swift class using the Create Class Registrar Endpoint.



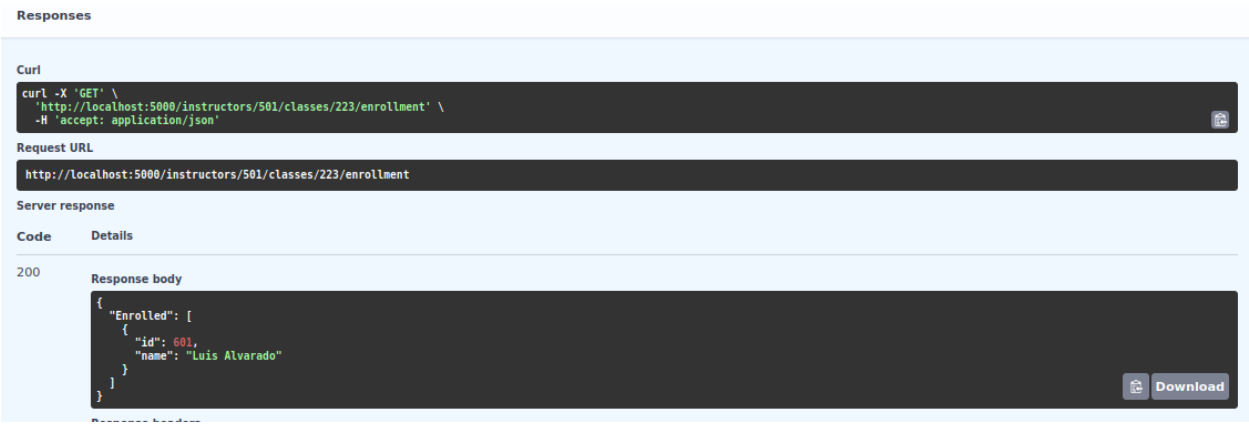
Test: Checking if class exists by using the Instructor Enrollment Endpoint. We should get a 200 status code with an empty “Enrolled” list.



Test - Add a new student to a new class.



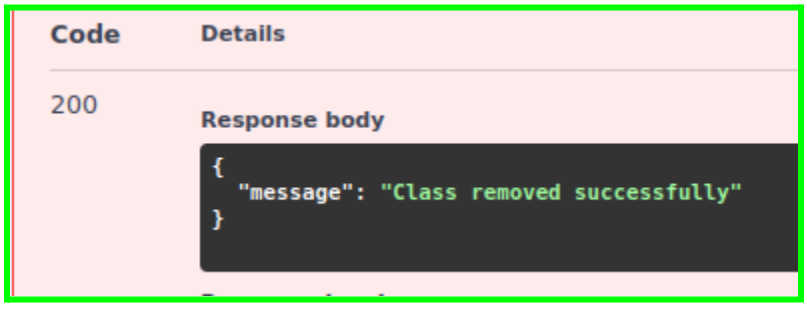
Result: Looking at the Instructor Enrolled List Endpoint now.



DELETE: Remove Class

Context: We test this endpoint if we can properly delete a class. We use class 2 to remove. If it is able to be removed we get back a response stating it was able to be removed. If the class does not exist we get a response back like in the second screenshot.

Endpoint: http://0.0.0.0:5000/registrar/classes/2

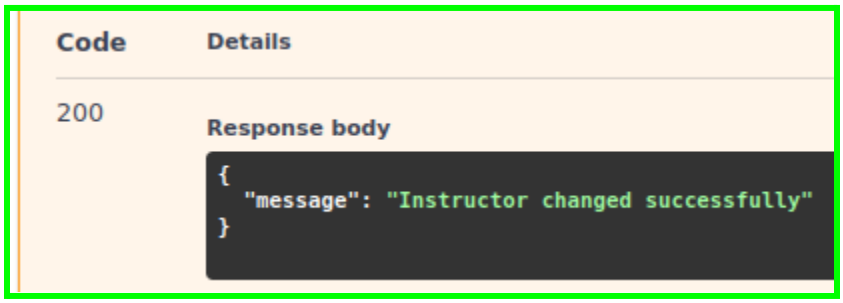


Code	Details
404 *Undocumented*	Error: Not Found **Response body** `{ "detail": "Class not found" }`

PUT: Change instructor

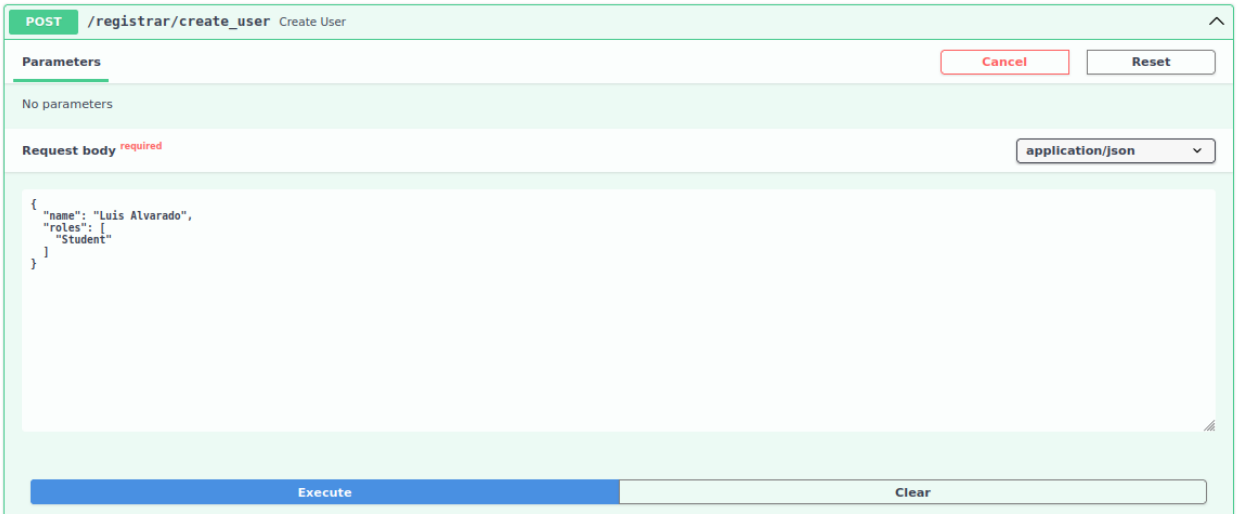
Context: We test this endpoint to make sure that we are able to change the instructor to a new class. We use class 3 and instructor 505 to be changed to the new class. If it is successful we would get a response stating that the instructor was changed successfully. If the instructor was already changed we should get a response back just like in the second screenshot.

Endpoint: http://0.0.0.0:5000/registrar/classes/3/instructors/505



POST: Create User

Context: Creating a new student named “Luis Alvarado”



Response: We successfully created a new student with incremented id.

Test - Enroll Student to an existing class.
Context: Added student 601 onto class id:1 and Instructor id 501. When calling we end up seeing student 601 in the enrolled list.