Fall 2023, CPSC 449, Section 1

Project 3

Angel Santoyo

Cesar Gutierrez

Daniel Truong

Joel Anil John

Melissa Huynh

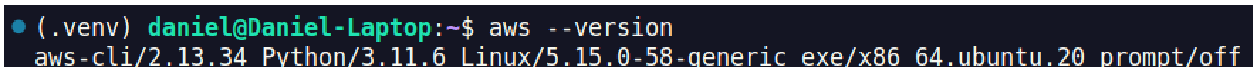
Task 1: Install and configure the AWS CLI

Objective:

The objective of this task is to install/ update AWS CLI on Linux

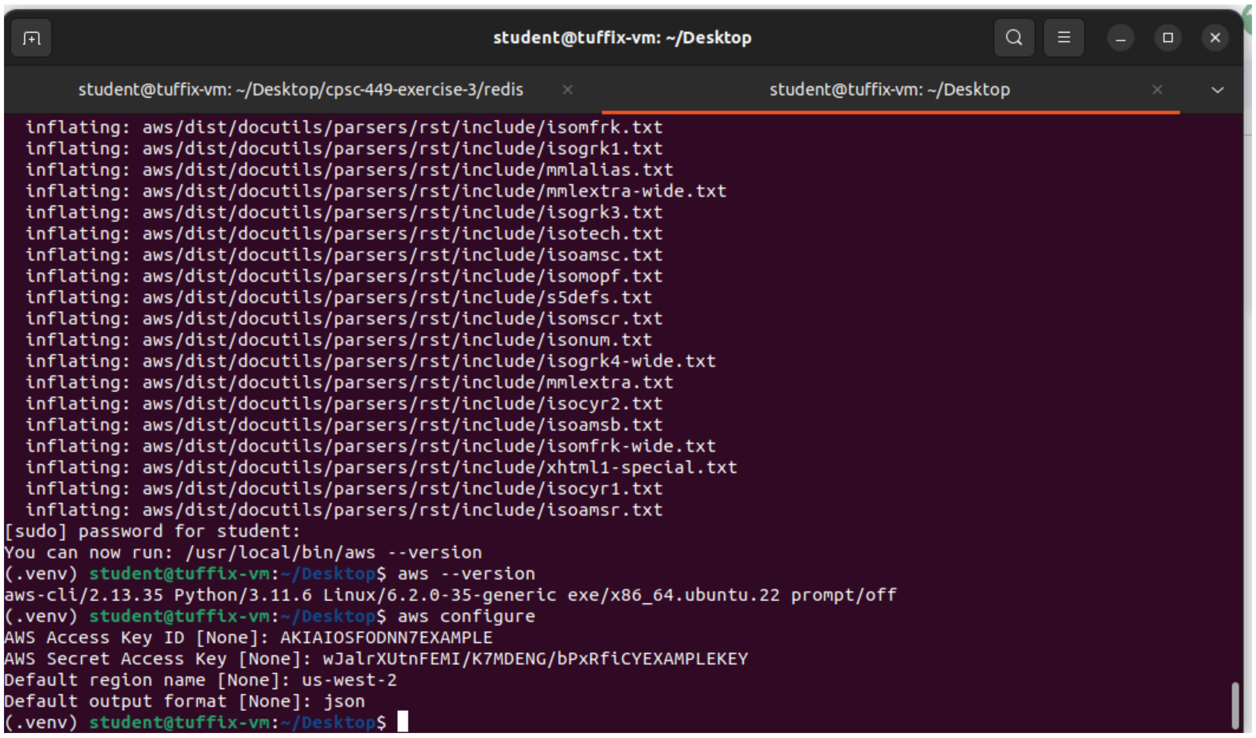
```
curl "https://awscli.amazonaws.com/awscli-exe-linux-x86_64.zip"
-o "awscliv2.zip"

unzip awscliv2.zip

sudo ./aws/install
```

Verify that AWS CLI is installed



```
(.venv) daniel@Daniel-Laptop:~$ aws --version
aws-cli/2.13.34 Python/3.11.6 Linux/5.15.0-58-generic exe/x86_64.ubuntu.20 prompt/off
```

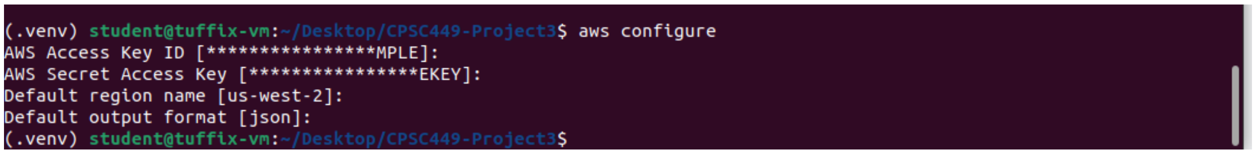
Configure AWS credentials using AWS CLI:



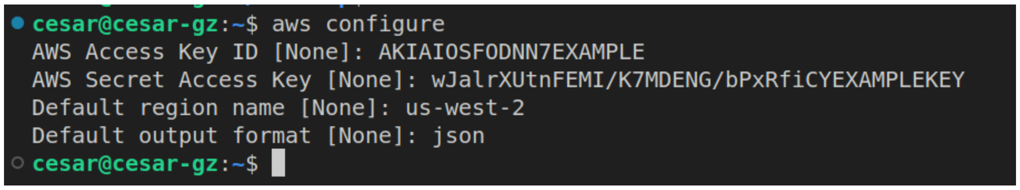
student@tuffix-vm: ~/Desktop
student@tuffix-vm: ~/Desktop/cpsc-449-exercise-3/redis
student@tuffix-vm: ~/Desktop

```
  inflating: aws/dist/docutils/parsers/rst/include/isomfrk.txt
  inflating: aws/dist/docutils/parsers/rst/include/isogrk1.txt
  inflating: aws/dist/docutils/parsers/rst/include/mmlalias.txt
  inflating: aws/dist/docutils/parsers/rst/include/mmlextra-wide.txt
  inflating: aws/dist/docutils/parsers/rst/include/isogrk3.txt
  inflating: aws/dist/docutils/parsers/rst/include/isotech.txt
  inflating: aws/dist/docutils/parsers/rst/include/isoamsc.txt
  inflating: aws/dist/docutils/parsers/rst/include/isomopf.txt
  inflating: aws/dist/docutils/parsers/rst/include/s5defs.txt
  inflating: aws/dist/docutils/parsers/rst/include/isomscr.txt
  inflating: aws/dist/docutils/parsers/rst/include/isonum.txt
  inflating: aws/dist/docutils/parsers/rst/include/isogrk4-wide.txt
  inflating: aws/dist/docutils/parsers/rst/include/mmlextra.txt
  inflating: aws/dist/docutils/parsers/rst/include/isocyr2.txt
  inflating: aws/dist/docutils/parsers/rst/include/isoamsb.txt
  inflating: aws/dist/docutils/parsers/rst/include/isomfrk-wide.txt
  inflating: aws/dist/docutils/parsers/rst/include/xhtml1-special.txt
  inflating: aws/dist/docutils/parsers/rst/include/isocyr1.txt
  inflating: aws/dist/docutils/parsers/rst/include/isoamsr.txt
[sudo] password for student:
You can now run: /usr/local/bin/aws --version
(.venv) student@tuffix-vm:~/Desktop$ aws --version
aws-cli/2.13.35 Python/3.11.6 Linux/6.2.0-35-generic exe/x86_64.ubuntu.22 prompt/off
(.venv) student@tuffix-vm:~/Desktop$ 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:~/Desktop$
```

Configure AWS credentials using AWS CLI:



```
(.venv) student@tuffix-vm:~/Desktop/CPSC449-Project3$ aws configure
AWS Access Key ID [****************MPLE]:
AWS Secret Access Key [****************EKEY]:
Default region name [us-west-2]:
Default output format [json]:
(.venv) student@tuffix-vm:~/Desktop/CPSC449-Project3$
```



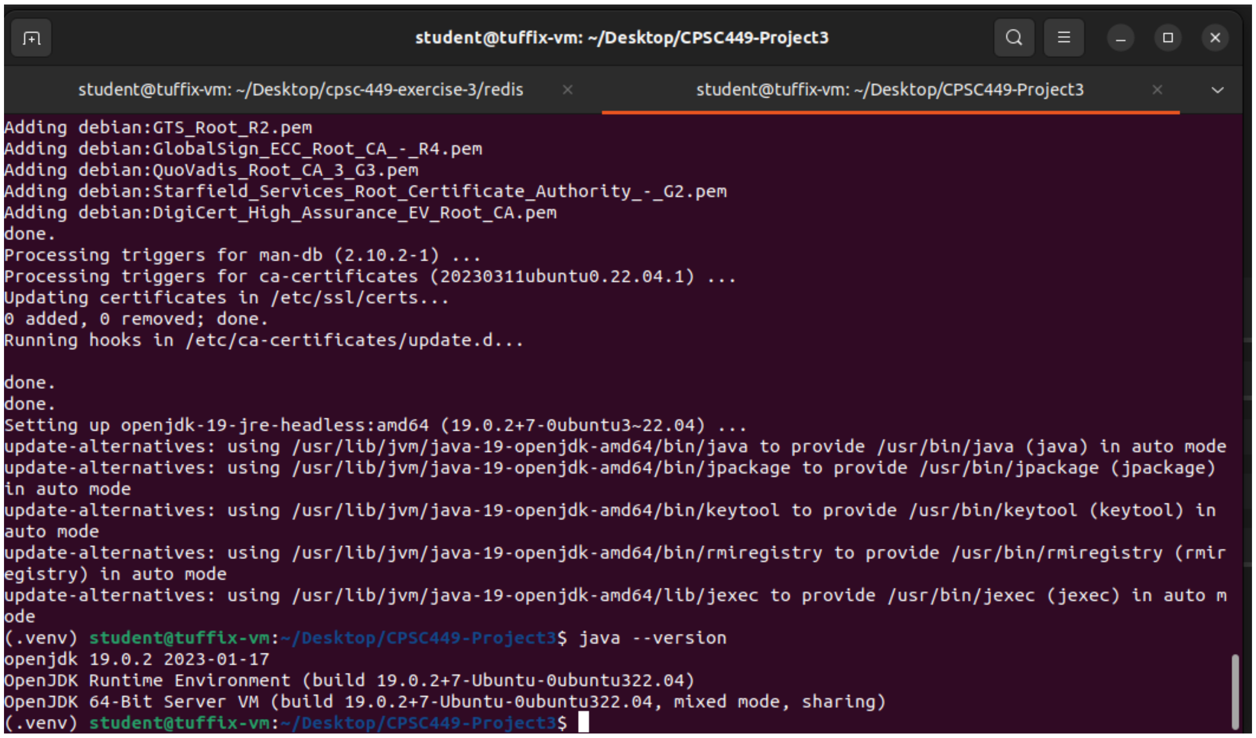
```
cesar@cesar-gz:~$ aws configure
AWS Access Key ID [None]: AKIAIOSFODNN7EXAMPLE
AWS Secret Access Key [None]: wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY
Default region name [None]: us-west-2
Default output format [None]: json
cesar@cesar-gz:~$
```

Task 2: Install and Configure Amazon DynamoDB local

Objective:

The objective of this task is to install/configure Amazon DynamoDB local and add it to Procfile so we can start DynamoDB local all at once along with our service

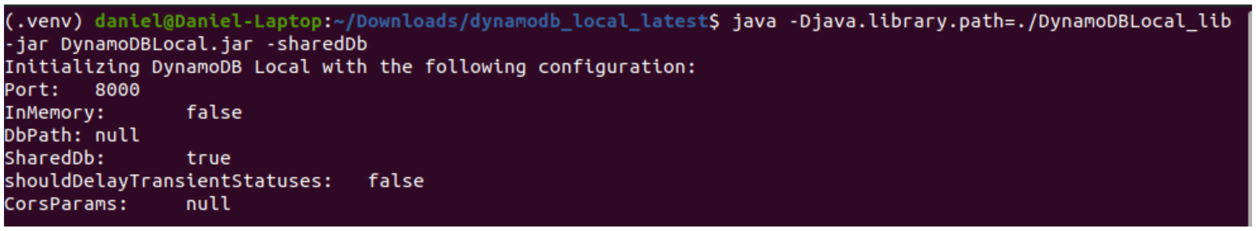
Java Runtime Environment Installation



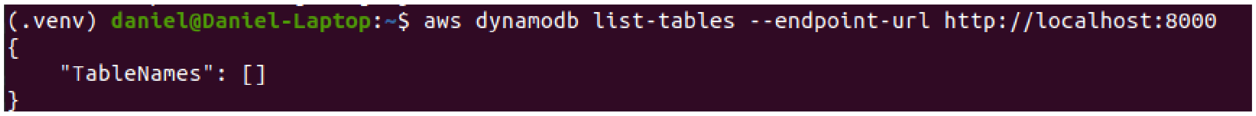
student@tuffix-vm: ~/Desktop/CPSC449-Project3
student@tuffix-vm: ~/Desktop/cpsc-449-exercise-3/redis
student@tuffix-vm: ~/Desktop/CPSC449-Project3

```
Adding debian:GTS_Root_R2.pem
Adding debian:GlobalSign_ECC_Root_CA_-_R4.pem
Adding debian:QuoVadis_Root_CA_3_G3.pem
Adding debian:Starfield_Services_Root_Certificate_Authority_-_G2.pem
Adding debian:DigiCert_High_Assurance_EV_Root_CA.pem
done.
Processing triggers for man-db (2.10.2-1) ...
Processing triggers for ca-certificates (20230311ubuntu0.22.04.1) ...
Updating certificates in /etc/ssl/certs...
0 added, 0 removed; done.
Running hooks in /etc/ca-certificates/update.d...

done.
done.
Setting up openjdk-19-jre-headless:amd64 (19.0.2+7-0ubuntu3~22.04) ...
update-alternatives: using /usr/lib/jvm/java-19-openjdk-amd64/bin/java to provide /usr/bin/java (java) in auto mode
update-alternatives: using /usr/lib/jvm/java-19-openjdk-amd64/bin/jpackage to provide /usr/bin/jpackage (jpackage)
in auto mode
update-alternatives: using /usr/lib/jvm/java-19-openjdk-amd64/bin/keytool to provide /usr/bin/keytool (keytool) in
auto mode
update-alternatives: using /usr/lib/jvm/java-19-openjdk-amd64/bin/rmiregistry to provide /usr/bin/rmiregistry (rmir
egistry) in auto mode
update-alternatives: using /usr/lib/jvm/java-19-openjdk-amd64/lib/jexec to provide /usr/bin/jexec (jexec) in auto m
ode
(.venv) student@tuffix-vm:~/Desktop/CPSC449-Project3$ java --version
openjdk 19.0.2 2023-01-17
OpenJDK Runtime Environment (build 19.0.2+7-Ubuntu-0ubuntu322.04)
OpenJDK 64-Bit Server VM (build 19.0.2+7-Ubuntu-0ubuntu322.04, mixed mode, sharing)
(.venv) student@tuffix-vm:~/Desktop/CPSC449-Project3$
```

Download DynamoDB local:



```
(.venv) daniel@Daniel-Laptop:~/Downloads/dynamodb_local_latest$ 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) daniel@Daniel-Laptop:~$ aws dynamodb list-tables --endpoint-url http://localhost:8000
{
    "TableNames": []
}
```

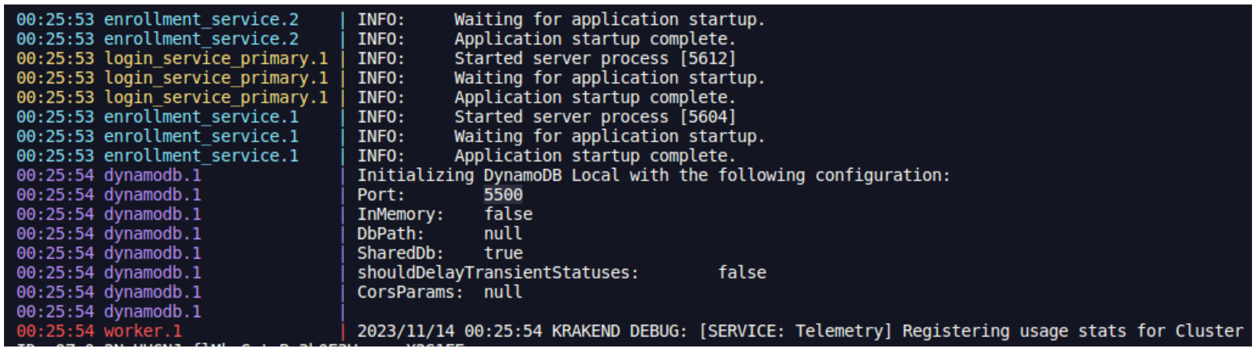
Added new process type to Procfile to start DynamoDB:

```
enrollment_service: uvicorn enrollment_service.enrollment_service:app
--port $PORT --reload

login_service_primary: ./bin/litefs mount -config etc/primary.yml

login_secondary: ./bin/litefs mount -config etc/secondary.yml

login_tertiary: ./bin/litefs mount -config etc/tertiary.yml

worker: echo ./etc/krakend.json | krakend run --config etc/krakend.json
--port $PORT

dynamodb: java
-Djava.library.path=./dynamodb_local_latest/DynamoDBLocal_lib -jar
./dynamodb_local_latest/DynamoDBLocal.jar -sharedDb --port $PORT
```

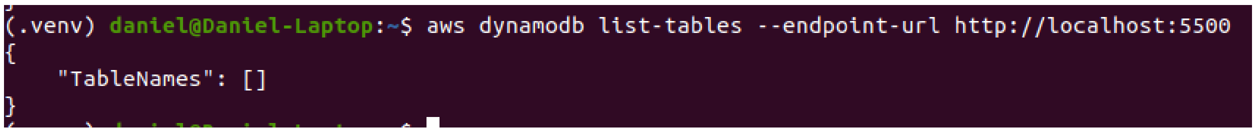
Add dynamodb to run.sh

```
foreman start -m
enrollment_service=3,login_service_primary=1,login_secondary=1,login_terti
ary=1,worker=1,dynamodb=1
```

Then, after running run.sh, we'll see all service endpoints run fine and also DynamoDB local running on port 5500



```
00:25:53 enrollment_service.2    | INFO:     Waiting for application startup.
00:25:53 enrollment_service.2    | INFO:     Application startup complete.
00:25:53 login_service_primary.1 | INFO:     Started server process [5612]
00:25:53 login_service_primary.1 | INFO:     Waiting for application startup.
00:25:53 login_service_primary.1 | INFO:     Application startup complete.
00:25:53 enrollment_service.1    | INFO:     Started server process [5604]
00:25:53 enrollment_service.1    | INFO:     Waiting for application startup.
00:25:53 enrollment_service.1    | INFO:     Application startup complete.
00:25:54 dynamodb.1              | Initializing DynamoDB Local with the following configuration:
00:25:54 dynamodb.1              | Port:        5500
00:25:54 dynamodb.1              | InMemory:    false
00:25:54 dynamodb.1              | DbPath:      null
00:25:54 dynamodb.1              | SharedDb:    true
00:25:54 dynamodb.1              | shouldDelayTransientStatuses:        false
00:25:54 dynamodb.1              | CorsParams:  null
00:25:54 dynamodb.1              |
00:25:54 worker.1                | 2023/11/14 00:25:54 KRAKEND DEBUG: [SERVICE: Telemetry] Registering usage stats for Cluster
```



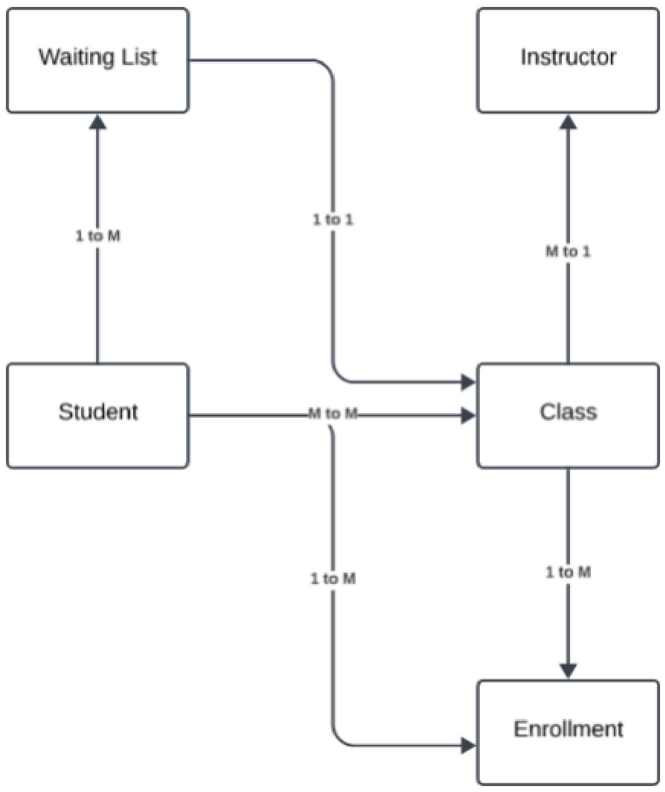
```
(.venv) daniel@Daniel-Laptop:~$ aws dynamodb list-tables --endpoint-url http://localhost:5500
{
    "TableNames": []
}
```

Task 3: Data Modeling for Enrollment Service Amazon DynamoDB & Redis

Objective:

The objective of this task is to design DynamoDB table to store information for the enrollment service and create the table using DynamoDB NoSQL Workbench, and use Redis to store class waitlist

ER Diagram for Enrollment Database

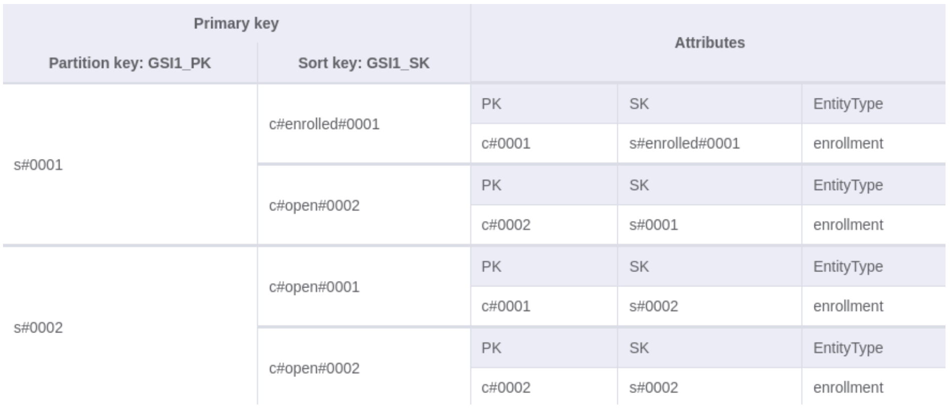


Visualize DynamoDB local using NoSQL Workbench

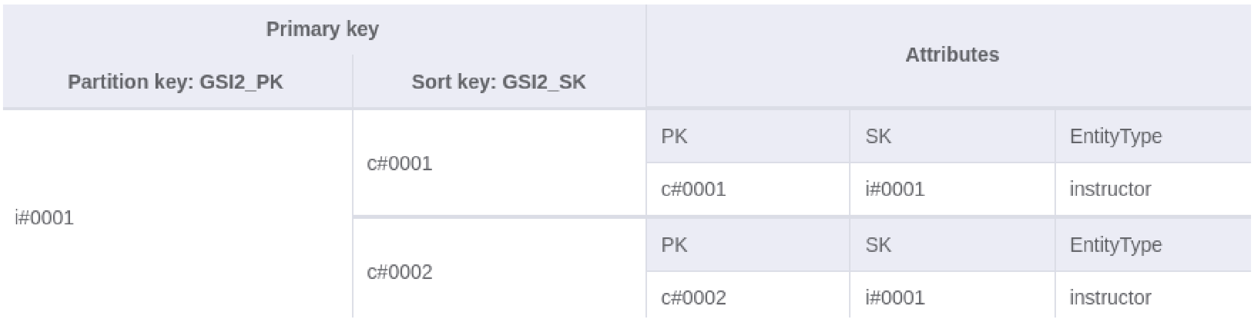
In order to visualize DynamoDB local table for data modeling, I’ve installed NoSQL Workbench for DynamoDB then export a JSON model for other people on the team to use it as the seeding database (TitanOnline.json file)

Primary key		Attributes						
Partition key: PK	Sort key: SK							
s#0001	s#0001	EntityType	Name	Email				
		student	Daniel Truong	danieltruong@fullerton.edu				
i#0001	i#0001	EntityType	Name	Email				
		instructor	Dr ABC	drabc@fullerton.edu				
c#0001	c#0001	EntityType	Detail	currentEnroll	maxEnroll	Frozen	GSI3_PK	GSI3_SK
		class	{"Department":{"S":"Computer Science"},"CourseCode":{"S":"CPSC 386"},"SectionNumber":{"S":"1"},"Name":{"S":"Game Programming"}}	10	10	false	c#0001	i#0001
	i#0001	EntityType	GSI2_PK	GSI2_SK				
		instructor	i#0001	c#0001				
	s#0002	EntityType	Detail	GSI1_PK	GSI1_SK			
		enrollment	{"Department":{"S":"Computer Science"},"CourseCode":{"S":"CPSC 386"},"SectionNumber":{"S":"1"},"Name":{"S":"Game Programming"}}	s#0002	c#open#0001			
	s#0003	EntityType	Detail	GSI1_PK	GSI1_SK			
		enrollment	{"Department":{"S":"Computer Science"},"CourseCode":{"S":"CPSC 386"},"SectionNumber":{"S":"1"},"Name":{"S":"Game Programming"}}	s#0003	c#open#0001			
	s#enrolled#0001	EntityType	Detail	GSI1_PK	GSI1_SK			
		enrollment	{"Department":{"S":"Computer Science"},"CourseCode":{"S":"CPSC 386"},"SectionNumber":{"S":"1"},"Name":{"S":"Game Programming"}}	s#0001	c#enrolled#0001			
c#0002	c#0002	EntityType	Detail	currentEnroll	maxEnroll	Frozen	GSI3_PK	GSI3_SK
		class	{"Department":{"S":"Computer Science"},"CourseCode":{"S":"CPSC 449"},"SectionNumber":{"S":"1"},"Name":{"S":"Backeng Engineering"}}	9	10	false	c#0002	i#0001
	i#0001	EntityType	GSI2_PK	GSI2_SK				
		instructor	i#0001	c#0002				
	s#0001	EntityType	Detail	GSI1_PK	GSI1_SK			
		enrollment	{"Department":{"S":"Computer Science"},"CourseCode":{"S":"CPSC 449"},"SectionNumber":{"S":"1"},"Name":{"S":"Backeng Engineering"}}	s#0001	c#open#0002			
	s#0002	EntityType	Detail	GSI1_PK	GSI1_SK			
		enrollment	{"Department":{"S":"Computer Science"},"CourseCode":{"S":"CPSC 449"},"SectionNumber":{"S":"1"},"Name":{"S":"Backeng Engineering"}}	s#0002	c#open#0002			
	s#0003	EntityType	Detail	GSI1_PK	GSI1_SK			
		enrollment	{"Department":{"S":"Computer Science"},"CourseCode":{"S":"CPSC 449"},"SectionNumber":{"S":"1"},"Name":{"S":"Backeng Engineering"}}	s#0003	c#open#0002			
s#0002	s#0002	EntityType	Name	Email				
		student	Titan Tuffy	tuffytitan@fullerton.edu				
s#0003	s#0003	EntityType	Name	Email				
		student	Student Three	stu3@fullerton.edu				
i#0002	i#0002	EntityType	Name	Email				
		instructor	Dr Tuffy	drtuffy@fullerton.edu				

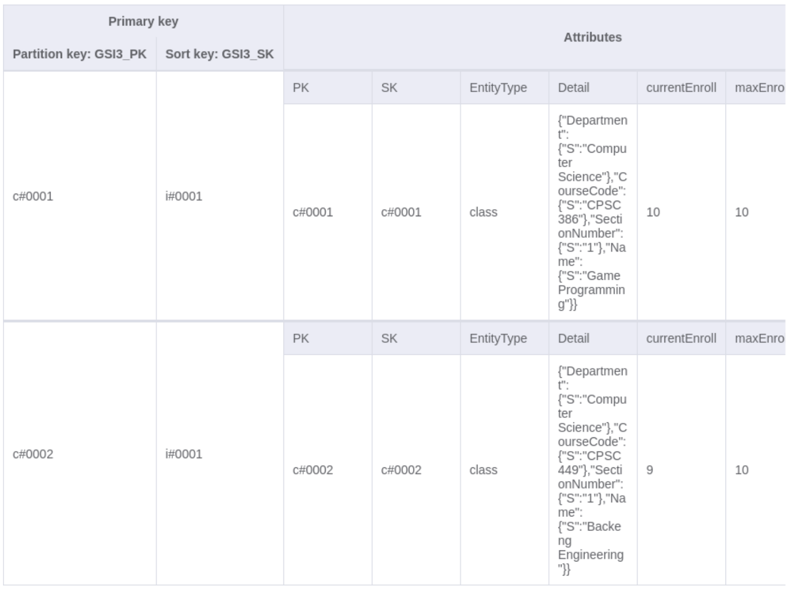
GSI1, GSI2, GSI3 for index based on studentId, instructorId, classId



Primary key		Attributes		
Partition key: GSI1_PK	Sort key: GSI1_SK			
s#0001	c#enrolled#0001	PK	SK	EntityType
		c#0001	s#enrolled#0001	enrollment
	c#open#0002	PK	SK	EntityType
		c#0002	s#0001	enrollment
s#0002	c#open#0001	PK	SK	EntityType
		c#0001	s#0002	enrollment
	c#open#0002	PK	SK	EntityType
		c#0002	s#0002	enrollment



Primary key		Attributes		
Partition key: GSI2_PK	Sort key: GSI2_SK			
i#0001	c#0001	PK	SK	EntityType
		c#0001	i#0001	instructor
	c#0002	PK	SK	EntityType
		c#0002	i#0001	instructor



Primary key		Attributes					
Partition key: GSI3_PK	Sort key: GSI3_SK						
c#0001	i#0001	PK	SK	EntityType	Detail	currentEnroll	maxEnro
		c#0001	c#0001	class	{"Department":{"S":"Computer Science"},"CourseCode":{"S":"CPSC 386"},"SectionNumber":{"S":"1"},"Name":{"S":"Game Programming"}}	10	10
c#0002	i#0001	PK	SK	EntityType	Detail	currentEnroll	maxEnro
		c#0002	c#0002	class	{"Department":{"S":"Computer Science"},"CourseCode":{"S":"CPSC 449"},"SectionNumber":{"S":"1"},"Name":{"S":"Backeng Engineering"}}	9	10

Access Pattern Design for EnrollmentService Table

Below is some access pattern example that I've used to design partition key, sort key, global secondary index (GSI)

Access Pattern	Table/ GSI/LSI	Key Condition	Example
Get student for a given studentId	Table	PK=studentId and SK=studentId	PK="s#0001" and SK ="s#0001"
Get available classes for a given studentId	GSI1	PK=studentId and SK begins_with c#open	PK="s#0001" and SK begins with c#open
Get instructor for a given instructorId	Table	PK=instructorId and SK=instructorId	PK="i#0001" and SK ="i#0001"
Get class for a given classId	Table	PK=classId and SK=classId	PK="c#0001" and SK ="c#0001"
Get enrollments given classId	Table	PK=classId and SK begins_with s#enrolled#	PK="c#0001" and SK begins with s#enrolled#
Get enrollments given studentId	GSI1	PK=studentId and SK begins_with c#enrolled	PK="s#0001" and SK begins with c#enrolled
Get students who have dropped the class given classId	Table	PK=classId and SK begins_with s#dropped	PK="c#0001" and SK begins with s#dropped
Get classes for a given instructorId	GSI2	PK=instructorId and SK begins_with c#	PK="i#0001" and SK begins_with c#
Get instructorId for a given classId	GSI3	PK=classId and SK begins_with i#	PK="c#0001" and SK begins_with i#

Redis Design for Class Waitlist

Key: waitlist:{class_id}

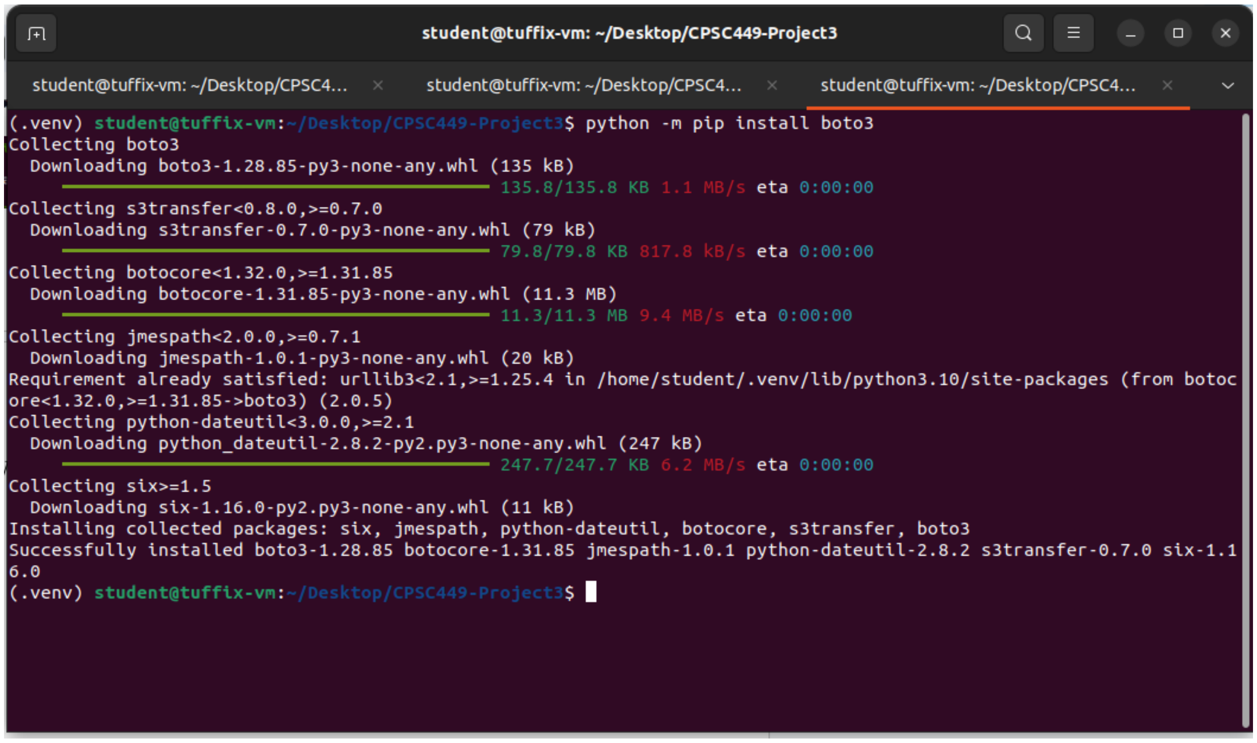
Value: s#{student_id}

Example:

Key: waitlist:0001

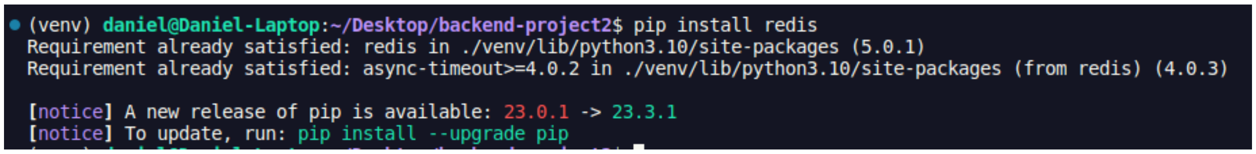
Value: [s#0001, s#0002, s#0003]

Install AWS SDK (boto3) for Python



```
(.venv) student@tuffix-vm:~/Desktop/CPSC449-Project3$ python -m pip install boto3
Collecting boto3
  Downloading boto3-1.28.85-py3-none-any.whl (135 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 135.8/135.8 KB 1.1 MB/s eta 0:00:00
Collecting s3transfer<0.8.0,>=0.7.0
  Downloading s3transfer-0.7.0-py3-none-any.whl (79 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 79.8/79.8 KB 817.8 kB/s eta 0:00:00
Collecting botocore<1.32.0,>=1.31.85
  Downloading botocore-1.31.85-py3-none-any.whl (11.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 11.3/11.3 MB 9.4 MB/s eta 0:00:00
Collecting jmespath<2.0.0,>=0.7.1
  Downloading jmespath-1.0.1-py3-none-any.whl (20 kB)
Requirement already satisfied: urllib3<2.1,>=1.25.4 in /home/student/.venv/lib/python3.10/site-packages (from botoc
ore<1.32.0,>=1.31.85->boto3) (2.0.5)
Collecting python-dateutil<3.0.0,>=2.1
  Downloading python_dateutil-2.8.2-py2.py3-none-any.whl (247 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 247.7/247.7 KB 6.2 MB/s eta 0:00:00
Collecting six>=1.5
  Downloading six-1.16.0-py2.py3-none-any.whl (11 kB)
Installing collected packages: six, jmespath, python-dateutil, botocore, s3transfer, boto3
Successfully installed boto3-1.28.85 botocore-1.31.85 jmespath-1.0.1 python-dateutil-2.8.2 s3transfer-0.7.0 six-1.1
6.0
(.venv) student@tuffix-vm:~/Desktop/CPSC449-Project3$
```

Install Redis for Python



```
(venv) daniel@Daniel-Laptop:~/Desktop/backend-project2$ pip install redis
Requirement already satisfied: redis in ./venv/lib/python3.10/site-packages (5.0.1)
Requirement already satisfied: async-timeout>=4.0.2 in ./venv/lib/python3.10/site-packages (from redis) (4.0.3)

[notice] A new release of pip is available: 23.0.1 -> 23.3.1
[notice] To update, run: pip install --upgrade pip
```

Task 4: Implemented and tested all enrollment service endpoint

Objective:

The objective of this task is to use boto3 for DynamoDB and Redis to rec-implement existing enrollment service endpoints and get rid of SQLite.

Import boto3, initialize DynamoDB Client, Redis

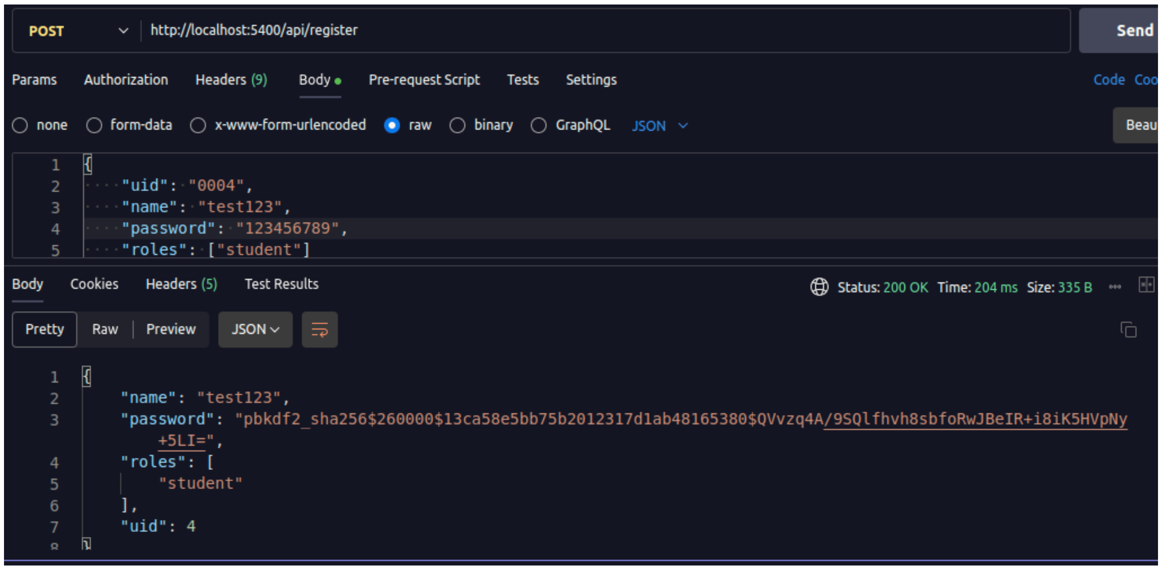
```
import redis

import boto3

dynamodb_client = boto3.client('dynamodb',
endpoint_url='http://localhost:5500')

table_name = 'TitanOnlineEnrollment'
```

```python
r = redis.Redis()
```

REST API Endpoints Testing after removing SQLite and adding DynamoDB, Redis

All of the DynamoDB query is in the query_helper.py file, I imported it to the routes.py to keep the code cleaner

Note: JWT still work perfectly fine as well as user service

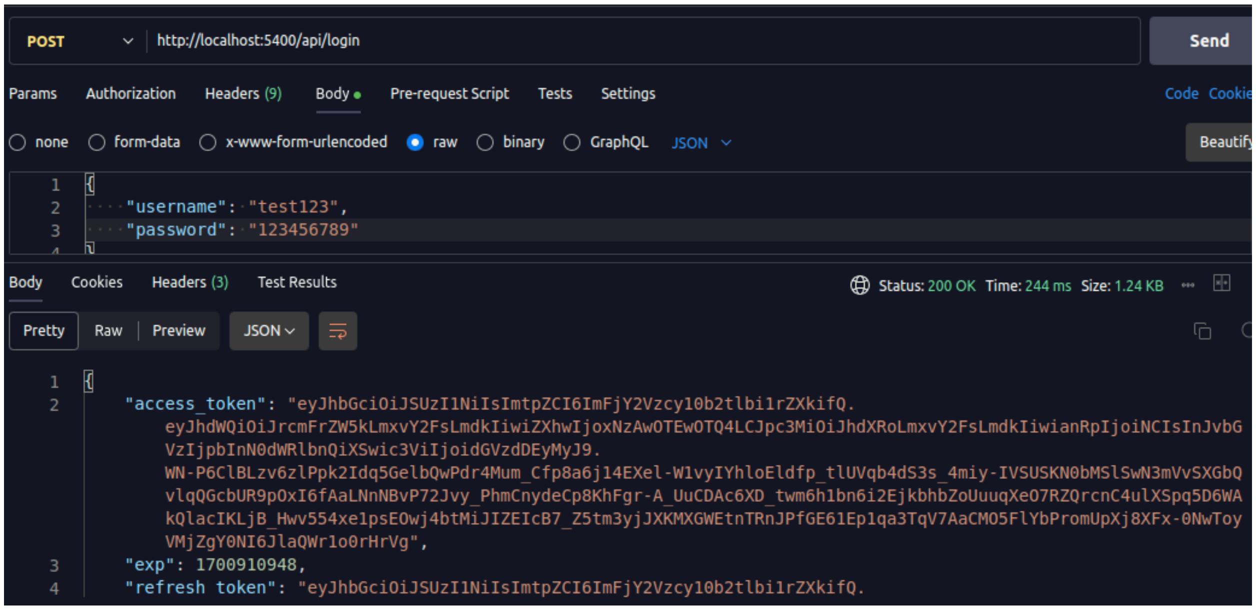


http://localhost:5400/api/register

```
{
    "uid": "0004",
    "name": "test123",
    "password": "123456789",
    "roles": ["student"]
```

Status: 200 OK Time: 204 ms Size: 335 B

```
{
    "name": "test123",
    "password": "pbkdf2_sha256$260000$13ca58e5bb75b2012317d1ab48165380$QVvzq4A/9SQlfhvh8sbfoRwJBeIR+i8iK5HVpNy
      +5LI=",
    "roles": [
        "student"
    ],
    "uid": 4
}
```



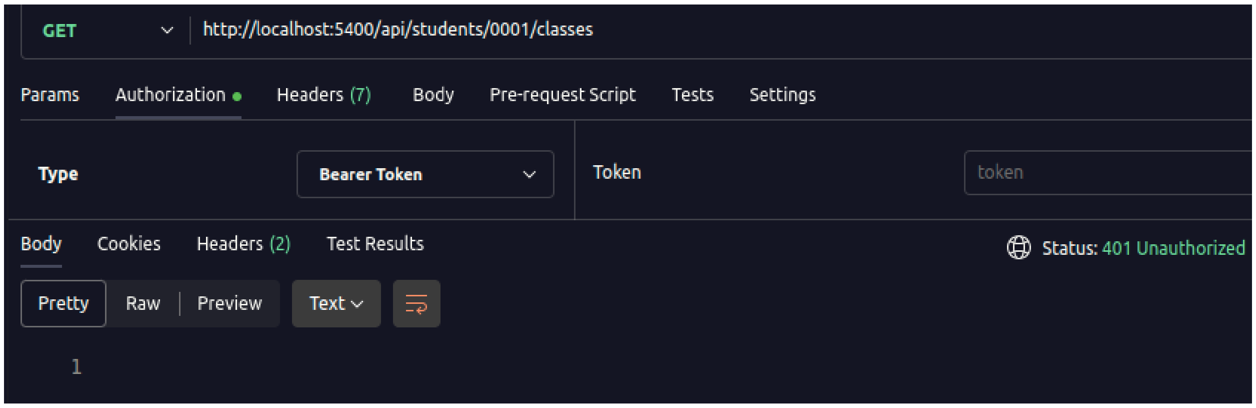
http://localhost:5400/api/login

```
{
    "username": "test123",
    "password": "123456789"
}
```

Status: 200 OK Time: 244 ms Size: 1.24 KB

```
{
    "access_token": "eyJhbGciOiJSUzI1NiIsImtpZCI6ImFjY2Vzcy10b2tlbi1rZXkifQ.
        eyJhdWQiOiJrcmFrZW5kLmxvY2FsLmdkIiwiZXhwIjoxNzAwOTEwOTQ4LCJpc3MiOiJhdXRoLmxvY2FsLmdkIiwianRpIjoiNCIsInJvbG
        VzIjpbInN0dWRlbnQiXSwic3ViIjoidGVzdDEyMyJ9.
        WN-P6ClBLzv6zlPpk2Idq5GelbQwPdr4Mum_Cfp8a6j14EXel-W1vyIYhloEldfp_tlUVqb4dS3s_4miy-IVSUSKN0bMSlSwN3mVvSXGbQ
        vlqQGcbUR9pOxI6fAaLNnNBvP72Jvy_PhmCnydeCp8KhFgr-A_UuCDAc6XD_twm6h1bn6i2EjkbhbZoUuuqXeO7RZQrcnC4ulXSpq5D6WA
        kQlacIKLjB_Hwv554xe1psEOwj4btMiJIZEIcB7_Z5tm3yjJXKMXGWEtnTRnJPfGE61Ep1qa3TqV7AaCMO5FlYbPromUpXj8XFx-0NwToy
        VMjZgY0NI6JlaQWr1o0rHrVg",
    "exp": 1700910948,
    "refresh token": "eyJhbGciOiJSUzI1NiIsImtpZCI6ImFjY2Vzcy10b2tlbi1rZXkifQ.
```

Before sign in


GET http://localhost:5400/api/students/0001/classes

Params | Authorization ● | Headers (7) | Body | Pre-request Script | Tests | Settings

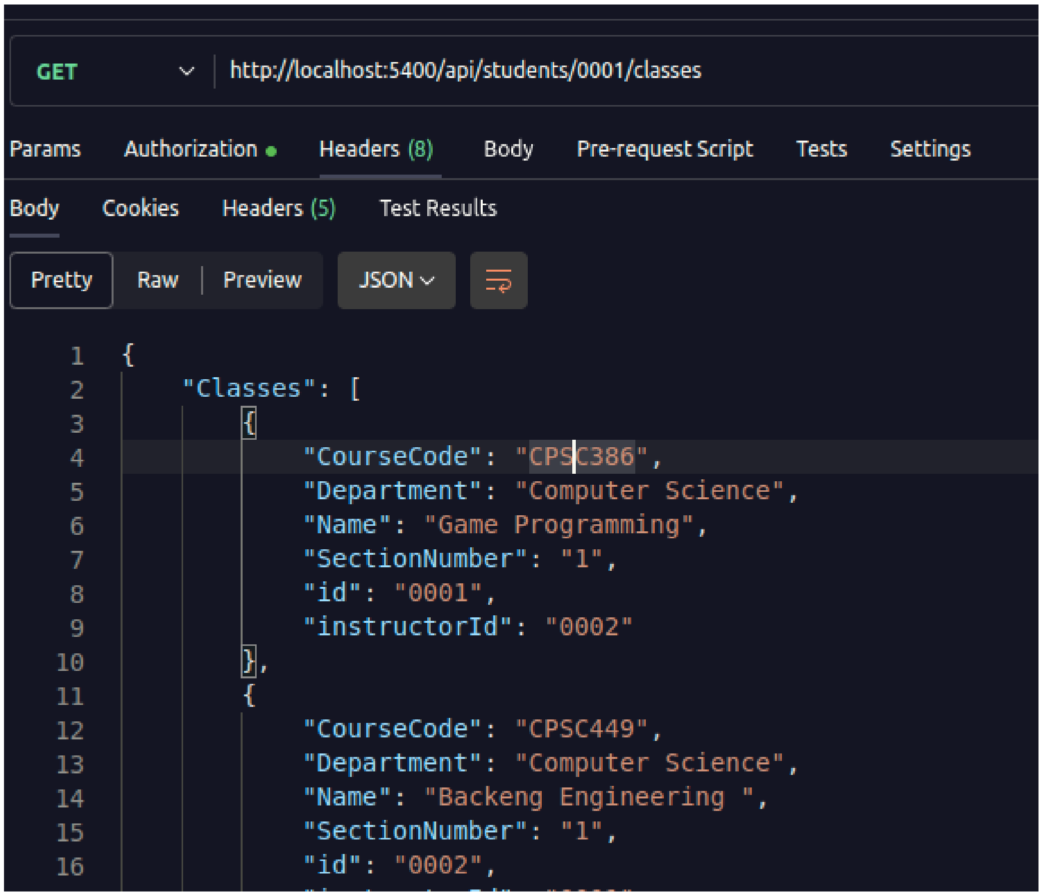
Type: Bearer Token | Token: token

Body | Cookies | Headers (2) | Test Results — Status: 401 Unauthorized

Pretty | Raw | Preview | Text

```
1
```

After signed in


GET http://localhost:5400/api/students/0001/classes

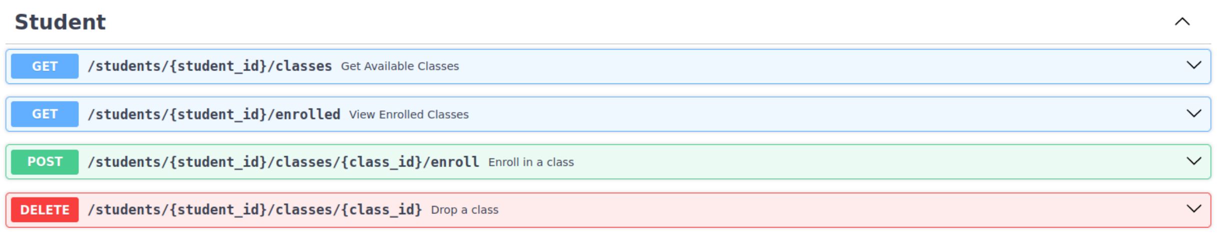
Params | Authorization ● | Headers (8) | Body | Pre-request Script | Tests | Settings

Body | Cookies | Headers (5) | Test Results

Pretty | Raw | Preview | JSON

```
{
    "Classes": [
        {
            "CourseCode": "CPSC386",
            "Department": "Computer Science",
            "Name": "Game Programming",
            "SectionNumber": "1",
            "id": "0001",
            "instructorId": "0002"
        },
        {
            "CourseCode": "CPSC449",
            "Department": "Computer Science",
            "Name": "Backeng Engineering ",
            "SectionNumber": "1",
            "id": "0002",
```

Enrollment Service Endpoints


Student

GET /students/{student_id}/classes Get Available Classes

GET /students/{student_id}/enrolled View Enrolled Classes

POST /students/{student_id}/classes/{class_id}/enroll Enroll in a class

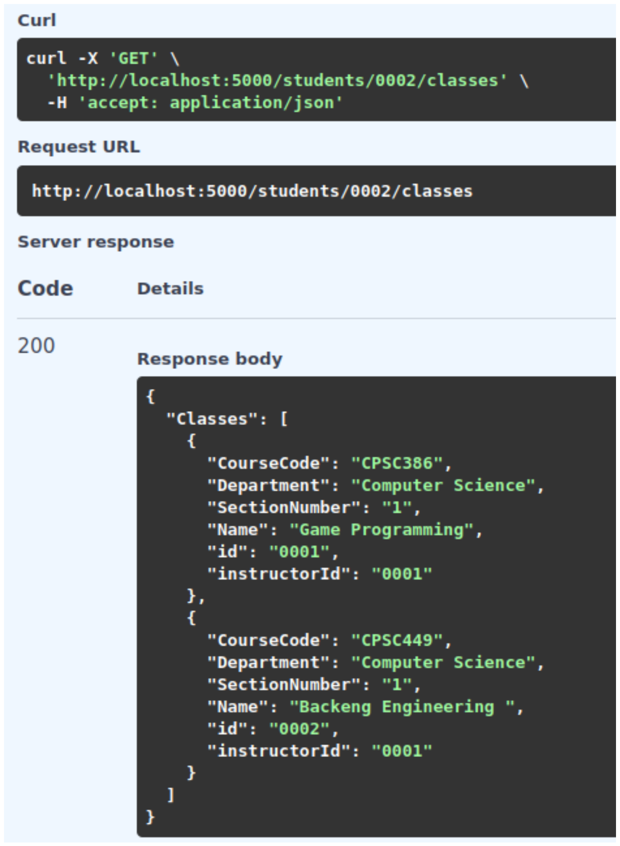
DELETE /students/{student_id}/classes/{class_id} Drop a class

GET /students/{student_id}/classes

This endpoint shows available classes given studentId, if student already enrolled/waitlisted in a class or the waitlist for the class is full then it won't show that class

```
gets available classes for a student

@router.get("/students/{student_id}/classes", tags=['Student'])

def get_available_classes(student_id: str):

   # Check if student exists in the database

   student_data = qh.query_student(dynamodb_client, student_id)

   if not student_data:

       raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="No student found")

   class_data = qh.query_available_classes(dynamodb_client, student_id)

   if not class_data:

       raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="No classes found")

   # If watlist full, don't show full classes with open waitlists

   filtered_class_data = []

   for item in class_data:

       waitlist_key = f'waitlist:{item["id"]}'

       waitlist_length = r.llen(waitlist_key)
```

```
        # Add the item to filtered_data only if the waitlist is not full

        if waitlist_length < MAX_WAITLIST or r.exists(waitlist_key) == 0:

            filtered_class_data.append(item)

    return {"Classes" : filtered_class_data}
```



Curl

```
curl -X 'GET' \
  'http://localhost:5000/students/0002/classes' \
  -H 'accept: application/json'
```

Request URL

```
http://localhost:5000/students/0002/classes
```

Server response

Code Details

200

Response body

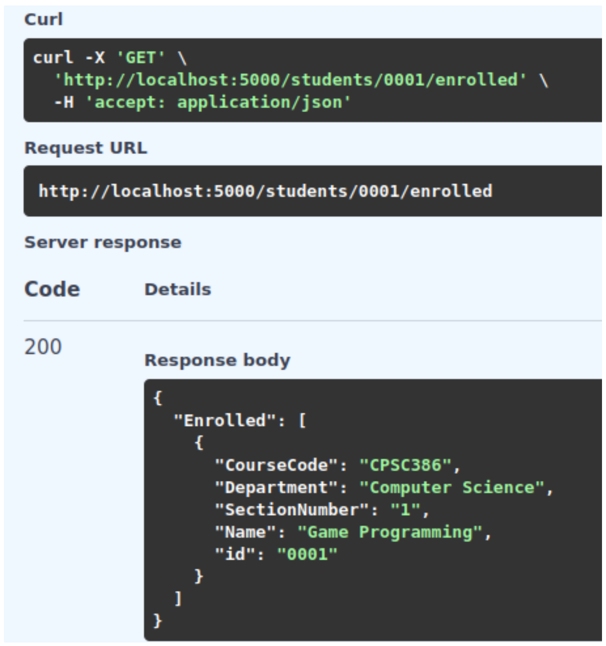
```
{
  "Classes": [
    {
      "CourseCode": "CPSC386",
      "Department": "Computer Science",
      "SectionNumber": "1",
      "Name": "Game Programming",
      "id": "0001",
      "instructorId": "0001"
    },
    {
      "CourseCode": "CPSC449",
      "Department": "Computer Science",
      "SectionNumber": "1",
      "Name": "Backeng Engineering ",
      "id": "0002",
      "instructorId": "0001"
    }
  ]
}
```

GET /students/{student_id}/enrolled

This endpoints shows enrolled classes given studentId, it also covers edge case when student/class doesn't exist

```
#gets currently enrolled classes for a student

@router.get("/students/{student_id}/enrolled", tags=['Student'])

def view_enrolled_classes(student_id: str):

    # Check if student exists in the database
```

```
   student_data = qh.query_student(dynamodb_client, student_id)

   if not student_data:

       raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="No student found")

   class_data = qh.query_enrolled_classes(dynamodb_client, student_id)

   if not class_data:

       raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="No classes found")



   return {"Enrolled": class_data}
```


Curl

```
curl -X 'GET' \
  'http://localhost:5000/students/0001/enrolled' \
  -H 'accept: application/json'
```

Request URL

```
http://localhost:5000/students/0001/enrolled
```

Server response

Code Details

200

Response body

```
{
  "Enrolled": [
    {
      "CourseCode": "CPSC386",
      "Department": "Computer Science",
      "SectionNumber": "1",
      "Name": "Game Programming",
      "id": "0001"
    }
  ]
}
```

POST /students/{student_id}/classes/{class_id}/enroll

This endpoint shows enrolled classes given studentId, if student already enrolled in a class, it’ll return error, if the class is full then student will automatically pushed to waitlist using Redis, if the class is frozen then student won't be able to enroll at all

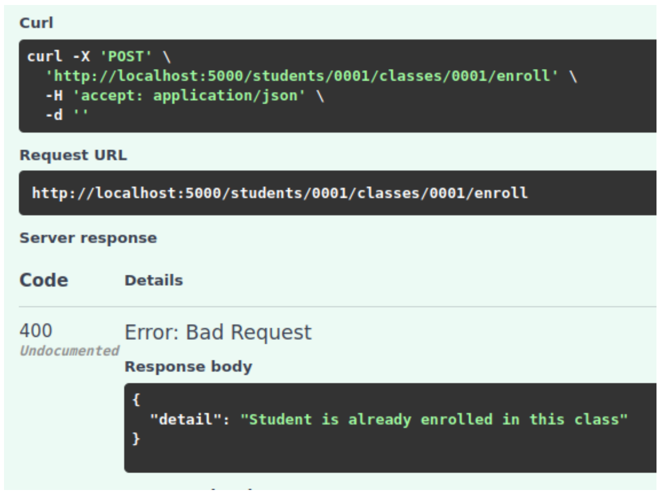
```python
# Enrolls a student into an available class,
# or will automatically put the student on an open waitlist for a full
class
@router.post("/students/{student_id}/classes/{class_id}/enroll",
tags=['Student'], summary="Enroll in a class")
def enroll_student_in_class(student_id: str, class_id: str):
   # Check if the student exists in the database
   student_data = qh.query_student(dynamodb_client, student_id)
   if not student_data:
       raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="No student found")

   # Check if the class exists in the database
   class_data = qh.query_class(dynamodb_client, class_id)
   if not class_data:
       raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="No class found")

   # Check if student is already enrolled in the class
   enrolled_class = qh.query_enrolled_classes(dynamodb_client, student_id)
   if enrolled_class:
       class_ids = [item['id'] for item in enrolled_class]
       if class_id in class_ids:
           raise HTTPException(status_code=status.HTTP_400_BAD_REQUEST,
detail="Student is already enrolled in this class")


   # Check if class is frozen
   if class_data['Frozen']:
       raise HTTPException(status_code=status.HTTP_400_BAD_REQUEST,
detail="Enrollment is frozen")


   # Check if the class is full
   if class_data['currentEnroll'] >= class_data['maxEnroll']:
       print("Class is full")
       # Waitlist handling
       waitlist_key = f"waitlist:{class_id}"
       # check if waitlist exists, add to wailist Redis with key
waitlist:class_id, value s#student_id
       if not r.exists(waitlist_key):
           r.rpush(waitlist_key, f"s#{student_id}")
           return {"message": "Student added to waitlist"}
       else:
           # check if student is already on waitlist
           id = f"s#{student_id}".encode('utf-8')
           if id in r.lrange(waitlist_key, 0, -1):
               raise
HTTPException(status_code=status.HTTP_400_BAD_REQUEST, detail="Student is
already on waitlist")
```

```
           # check if adding student to waitlist will exceed max waitlist
           if r.llen(waitlist_key) < MAX_WAITLIST:
               r.rpush(waitlist_key, f"s#{student_id}")
               return {"message": "Student added to waitlist"}
           else:
               raise
HTTPException(status_code=status.HTTP_400_BAD_REQUEST, detail="Unable to
add student to waitlist due to already having max number of waitlists")


   # Add student to enrolled class in the database


   enrolled_class = qh.update_enrolled_class(dynamodb_client, student_id,
class_id)
   if not enrolled_class:
       raise
HTTPException(status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
detail="Unable to enroll student in class")


   # Increment enrollment number in the database
   new_enrollment = class_data['currentEnroll'] + 1
   update_finished = qh.update_current_enroll(dynamodb_client, class_id,
new_enrollment)
   if not update_finished:
       raise
HTTPException(status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
detail="Unable to update class enrollment")
```

```
# Fetch the updated class data from the database

updated_class_data = qh.query_class(dynamodb_client, class_id)



return updated_class_data["Detail"]
```

Here’re different cases I’ve tested:

Student already enrolled in the class



Curl

```
curl -X 'POST' \
  'http://localhost:5000/students/0001/classes/0001/enroll' \
  -H 'accept: application/json' \
  -d ''
```

Request URL

```
http://localhost:5000/students/0001/classes/0001/enroll
```

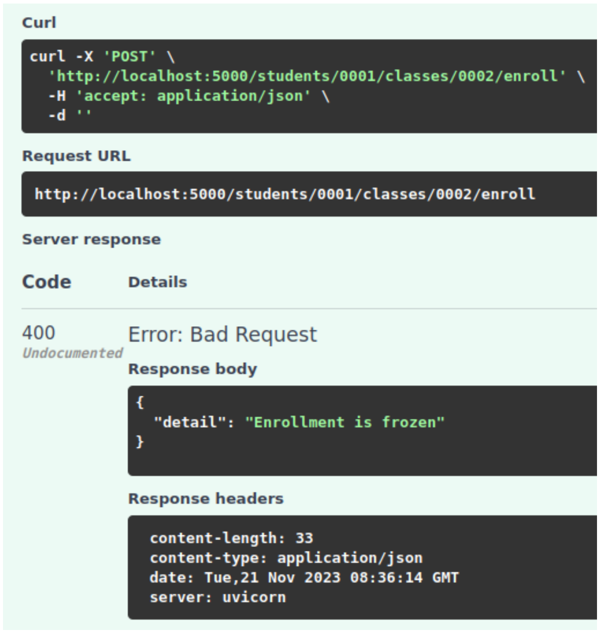
Server response

Code	Details
400 *Undocumented*	Error: Bad Request

Response body

```
{
  "detail": "Student is already enrolled in this class"
}
```

Student trying to enroll in frozen class



Curl

```
curl -X 'POST' \
  'http://localhost:5000/students/0001/classes/0002/enroll' \
  -H 'accept: application/json' \
  -d ''
```

Request URL

```
http://localhost:5000/students/0001/classes/0002/enroll
```

Server response

Code Details

400 *Undocumented* Error: Bad Request

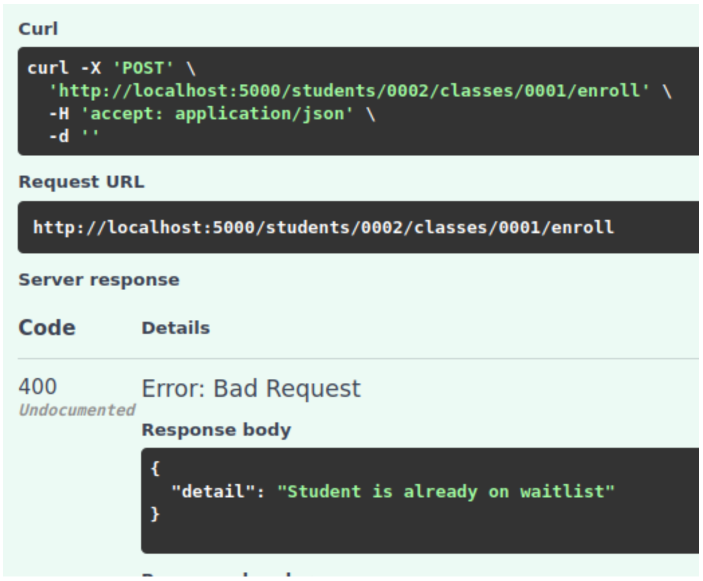
Response body

```
{
  "detail": "Enrollment is frozen"
}
```

Response headers

```
content-length: 33
content-type: application/json
date: Tue,21 Nov 2023 08:36:14 GMT
server: uvicorn
```

If student already on waitlist



Curl

```
curl -X 'POST' \
  'http://localhost:5000/students/0002/classes/0001/enroll' \
  -H 'accept: application/json' \
  -d ''
```

Request URL

```
http://localhost:5000/students/0002/classes/0001/enroll
```

Server response

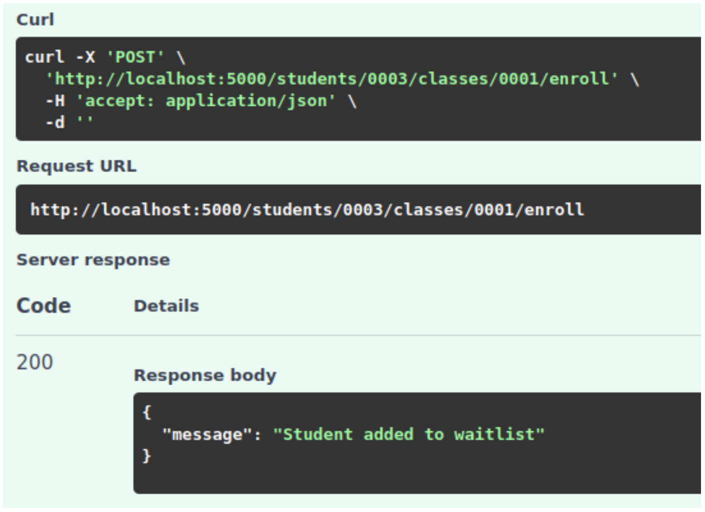
Code Details

400 *Undocumented* Error: Bad Request

Response body

```
{
  "detail": "Student is already on waitlist"
}
```

Class is full, add student to waitlist using Redis



Curl

```
curl -X 'POST' \
  'http://localhost:5000/students/0003/classes/0001/enroll' \
  -H 'accept: application/json' \
  -d ''
```

Request URL

```
http://localhost:5000/students/0003/classes/0001/enroll
```

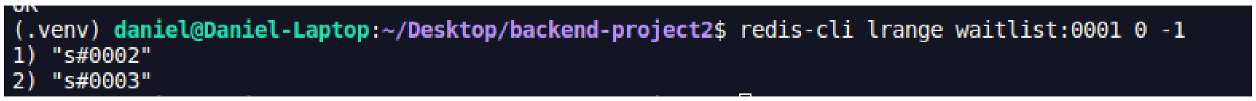
Server response

Code Details

200

Response body

```
{
  "message": "Student added to waitlist"
}
```



```
(.venv) daniel@Daniel-Laptop:~/Desktop/backend-project2$ redis-cli lrange waitlist:0001 0 -1
1) "s#0002"
2) "s#0003"
```

DELETE /students/{student_id}/classes/{class_id}

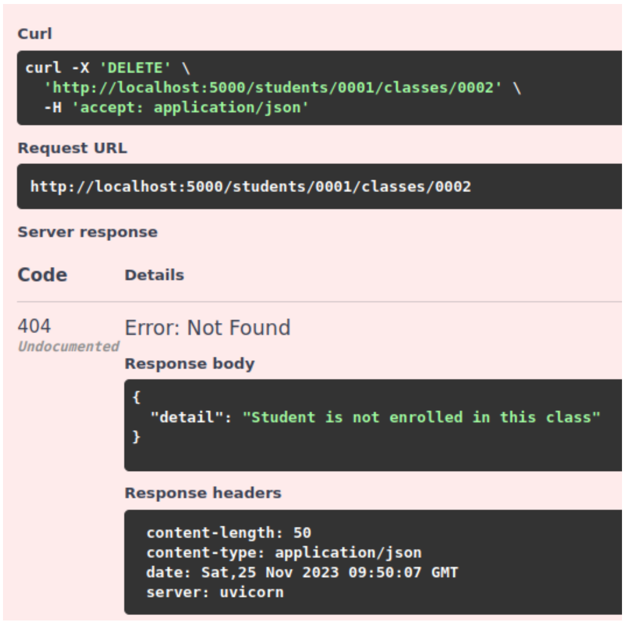
This endpoint shows how a student drops a class, student should only able to drop class if they're enrolled, and if the class has waitlist, the first person in the waitlist will be enrolled in the class unless automatic enrollment is frozen.

```
@router.delete("/students/{student_id}/classes/{class_id}",
tags=['Student'], summary="Drop a class")

def drop_student_from_class(student_id: str, class_id: str):

   # Check if the student exists in the database

   student_data = qh.query_student(dynamodb_client, student_id)

   if not student_data:

       raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="No student found")

   # Check if the class exists in the database

   class_data = qh.query_class(dynamodb_client, class_id)

   if not class_data:
       raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="No class found")

   # Check if student is enrolled in the class

   enrolled_class = qh.query_enrolled_classes(dynamodb_client, student_id)

   if not enrolled_class:

       raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="Student is not enrolled in this class")

   class_ids = [item['id'] for item in enrolled_class]

   if class_id not in class_ids:

       raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="Student is not enrolled in this class")

   # Drop student from class

   drop_finished = qh.drop_student_from_class(dynamodb_client, student_id,
class_id)

   if not drop_finished:

       raise
HTTPException(status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
detail="Unable to drop student from class")

   # Decrement enrollment number in the database

   new_enrollment = class_data['currentEnroll'] - 1

   update_finished = qh.update_current_enroll(dynamodb_client, class_id,
new_enrollment)

   if not update_finished:

       raise
HTTPException(status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
detail="Unable to update class enrollment")

   # check if waitlist exists for class
```

```
    # check if freeze is on

    if not class_data['Frozen']:

        if r.exists(f"waitlist:{class_id}"):

            # first student on waitlist is automatically enrolled

            waitlist_data = r.lrange(f"waitlist:{class_id}", 0, 0)

            waitlist_data = [item.decode('utf-8')[2:] for item in
waitlist_data]

            # Enroll student in class

            enrolled_class = qh.update_enrolled_class(dynamodb_client,
waitlist_data[0], class_id)

            if not enrolled_class:

                raise
HTTPException(status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
detail="Unable to enroll student in class")

            # Increment enrollment number in the database

            new_enrollment = class_data['currentEnroll'] + 1

            update_finished = qh.update_current_enroll(dynamodb_client,
class_id, new_enrollment)

            if not update_finished:

                raise
HTTPException(status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
detail="Unable to update class enrollment")

            # Remove student from waitlist

            r.lrem(f"waitlist:{class_id}", 0, f"s#{waitlist_data[0]}")

            # Fetch the updated class data from the database

            updated_class_data = qh.query_class(dynamodb_client, class_id)
```

```
          return {"message": "Student dropped from class and first
student on waitlist enrolled", "Class": updated_class_data["Detail"]}

   return {"message": "Student dropped from class"}
```

Here what happens when student try to drop a class they're not enrolled



Curl

```
curl -X 'DELETE' \
  'http://localhost:5000/students/0001/classes/0002' \
  -H 'accept: application/json'
```

Request URL

```
http://localhost:5000/students/0001/classes/0002
```

Server response

Code	Details
404 *Undocumented*	Error: Not Found

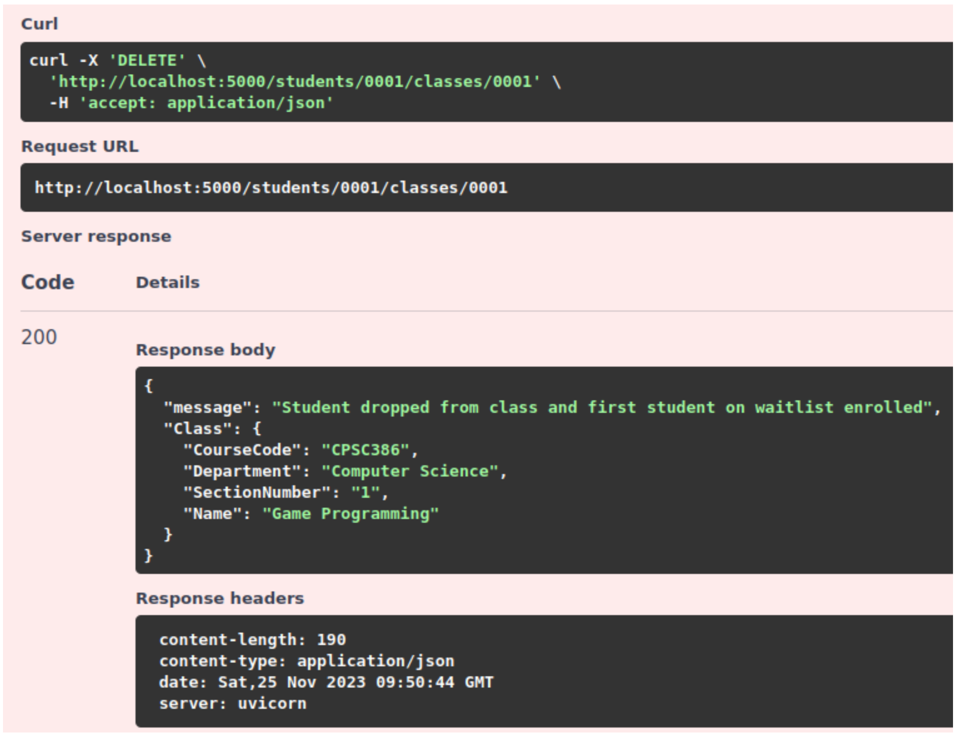
Response body

```
{
  "detail": "Student is not enrolled in this class"
}
```

Response headers

```
 content-length: 50
 content-type: application/json
 date: Sat,25 Nov 2023 09:50:07 GMT
 server: uvicorn
```

Student able to drop the class and the first student from waitlist (student 0002) automatic enrolled


Curl

```
curl -X 'DELETE' \
  'http://localhost:5000/students/0001/classes/0001' \
  -H 'accept: application/json'
```

Request URL

```
http://localhost:5000/students/0001/classes/0001
```

Server response

Code Details

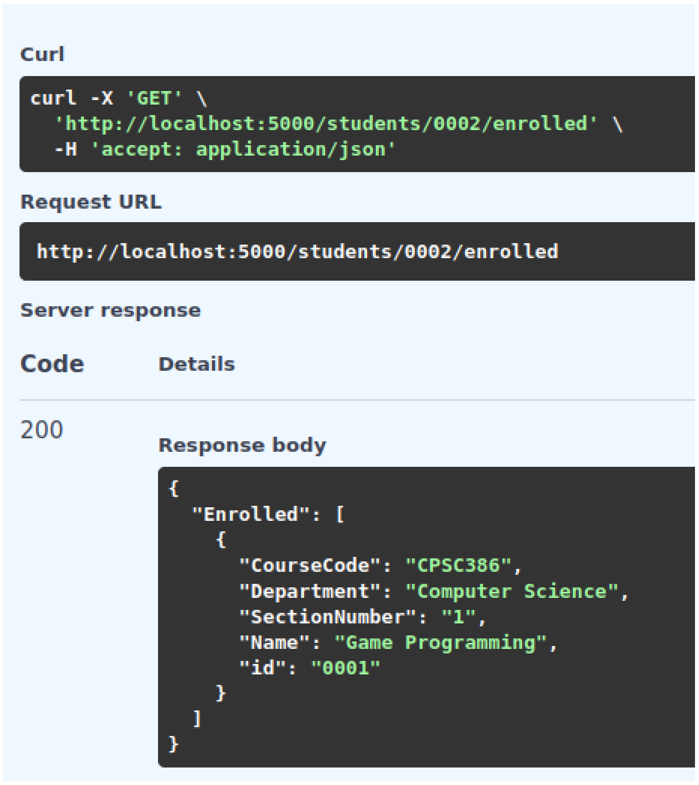
200

Response body

```
{
  "message": "Student dropped from class and first student on waitlist enrolled",
  "Class": {
    "CourseCode": "CPSC386",
    "Department": "Computer Science",
    "SectionNumber": "1",
    "Name": "Game Programming"
  }
}
```

Response headers

```
 content-length: 190
 content-type: application/json
 date: Sat,25 Nov 2023 09:50:44 GMT
 server: uvicorn
```


Curl

```
curl -X 'GET' \
  'http://localhost:5000/students/0002/enrolled' \
  -H 'accept: application/json'
```

Request URL

```
http://localhost:5000/students/0002/enrolled
```

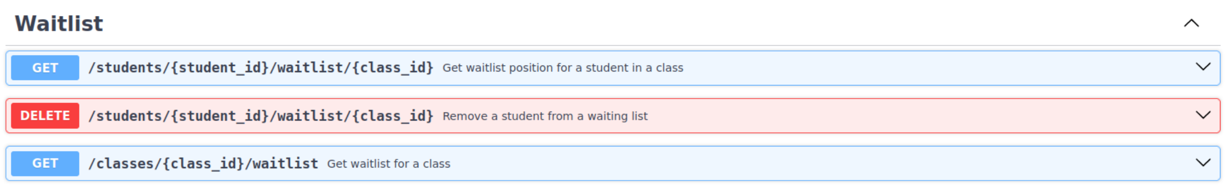
Server response

Code Details

200

Response body

```
{
  "Enrolled": [
    {
      "CourseCode": "CPSC386",
      "Department": "Computer Science",
      "SectionNumber": "1",
      "Name": "Game Programming",
      "id": "0001"
    }
  ]
}
```


Waitlist

GET /students/{student_id}/waitlist/{class_id} Get waitlist position for a student in a class

DELETE /students/{student_id}/waitlist/{class_id} Remove a student from a waiting list

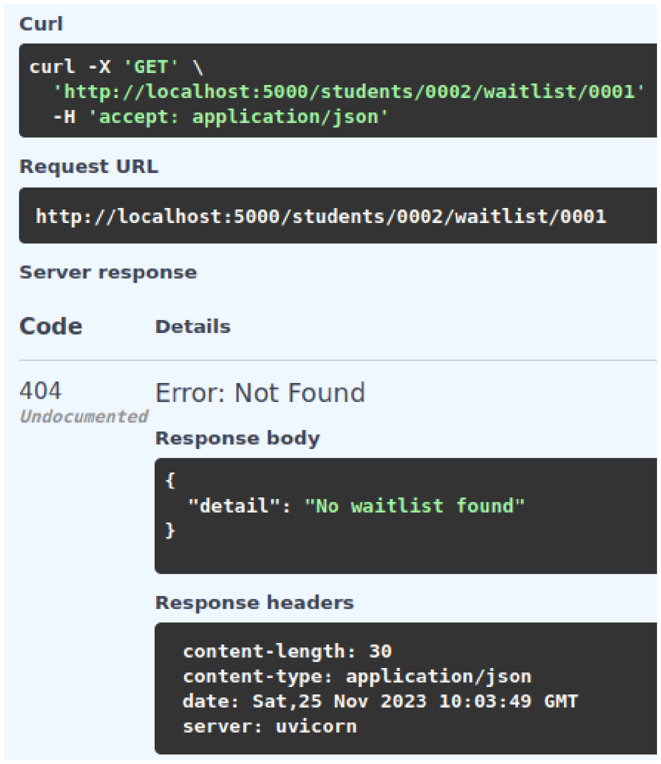
GET /classes/{class_id}/waitlist Get waitlist for a class

GET /students/{student_id}/waitlist/{class_id}

This endpoint allows student to view their waitlist position in a class, the waitlist is managed using Redis

```
@router.get("/students/{student_id}/waitlist/{class_id}",
tags=['Waitlist'], summary="Get waitlist position for a student in a
class")

def view_waiting_list(student_id: str, class_id: str):

   # check if student exists in the database

   student_data = qh.query_student(dynamodb_client, student_id)

   if not student_data:

       raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="No student found")

   # check if class exists in the database

   class_data = qh.query_class(dynamodb_client, class_id)

   if not class_data:

       raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="No class found")

   waitlist_key = f"waitlist:{class_id}"

   if not r.exists(waitlist_key):

       raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="No waitlist found")

   waitlist_data = r.lrange(waitlist_key, 0, -1)

   if not waitlist_data:
       raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="No waitlist found")

   # Check if student is on waitlist

   id = f"s#{student_id}".encode('utf-8')

   if id not in waitlist_data:

       raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="Student is not on waitlist")

   # Get student's position on waitlist

   position = waitlist_data.index(id) + 1

   return {"Waitlist Position": position}
```

Here's what happen when student not in waitlist for the class:



Curl

```
curl -X 'GET' \
  'http://localhost:5000/students/0002/waitlist/0001'
  -H 'accept: application/json'
```

Request URL

```
http://localhost:5000/students/0002/waitlist/0001
```

Server response

Code Details

404 *Undocumented* Error: Not Found

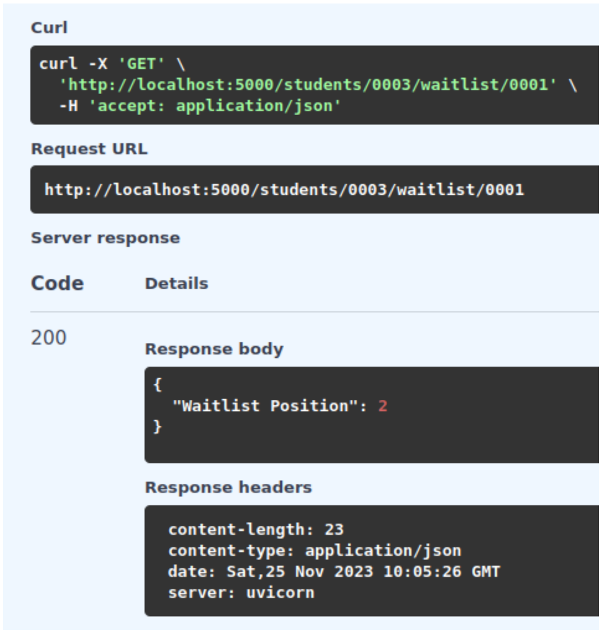
Response body

```
{
  "detail": "No waitlist found"
}
```

Response headers

```
 content-length: 30
 content-type: application/json
 date: Sat,25 Nov 2023 10:03:49 GMT
 server: uvicorn
```

Here's the result showing student 0003 waitlist position after 0001 got added to waitlist and 0003 also added to waitlist



Curl

```
curl -X 'GET' \
  'http://localhost:5000/students/0003/waitlist/0001' \
  -H 'accept: application/json'
```

Request URL

```
http://localhost:5000/students/0003/waitlist/0001
```

Server response

Code Details

200

Response body

```
{
  "Waitlist Position": 2
}
```

Response headers

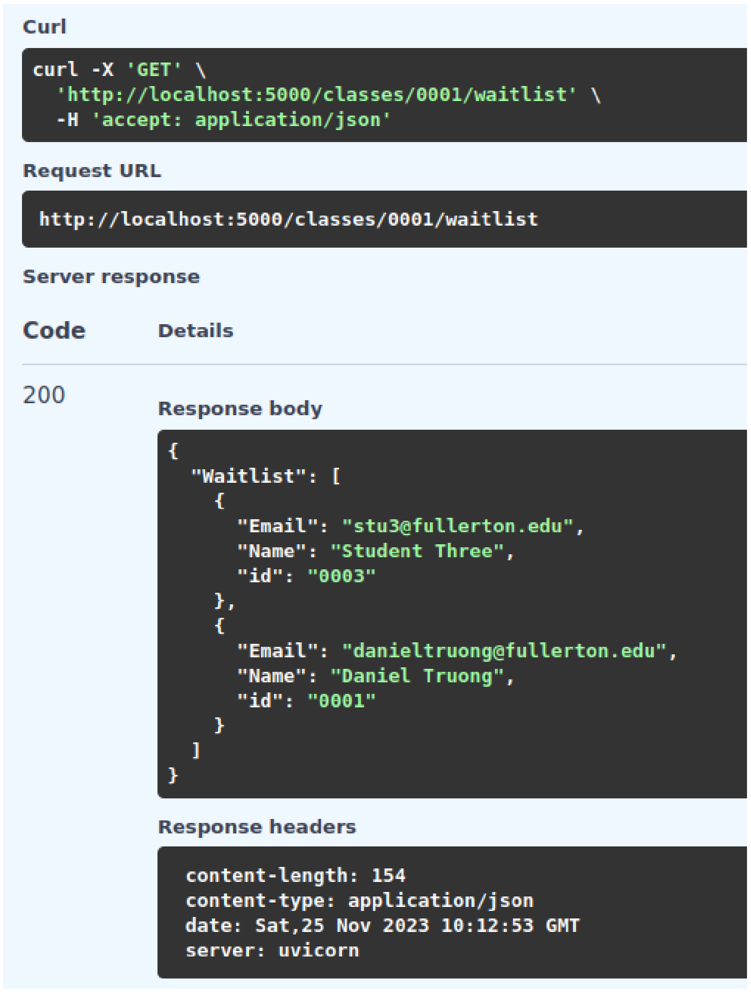
```
 content-length: 23
 content-type: application/json
 date: Sat,25 Nov 2023 10:05:26 GMT
 server: uvicorn
```

GET /classes/{class_id}/waitlist

This endpoint display student currently in the waitlist for a class

```
@router.get("/classes/{class_id}/waitlist",tags=['Waitlist'], summary="Get
waitlist for a class")

def view_current_waitlist(class_id: str):

   # Check if class exist

   class_data = qh.query_class(dynamodb_client, class_id)

   if not class_data:

       raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="No class found")

   waitlist_key = f"waitlist:{class_id}"

   if not r.exists(waitlist_key):

       raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="No waitlist found")

   waitlist_data = r.lrange(waitlist_key, 0, -1)
```

```python
    if not waitlist_data:

        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="No waitlist found")

    waitlist_data = [item.decode('utf-8') for item in waitlist_data]

    # Get student info from waitlist_data

    waitlist_data = qh.batch_query_student(dynamodb_client, waitlist_data)

    return {"Waitlist": waitlist_data}
```



Curl

```
curl -X 'GET' \
  'http://localhost:5000/classes/0001/waitlist' \
  -H 'accept: application/json'
```

Request URL

```
http://localhost:5000/classes/0001/waitlist
```

Server response

Code Details

200

Response body

```
{
  "Waitlist": [
    {
      "Email": "stu3@fullerton.edu",
      "Name": "Student Three",
      "id": "0003"
    },
    {
      "Email": "danieltruong@fullerton.edu",
      "Name": "Daniel Truong",
      "id": "0001"
    }
  ]
}
```

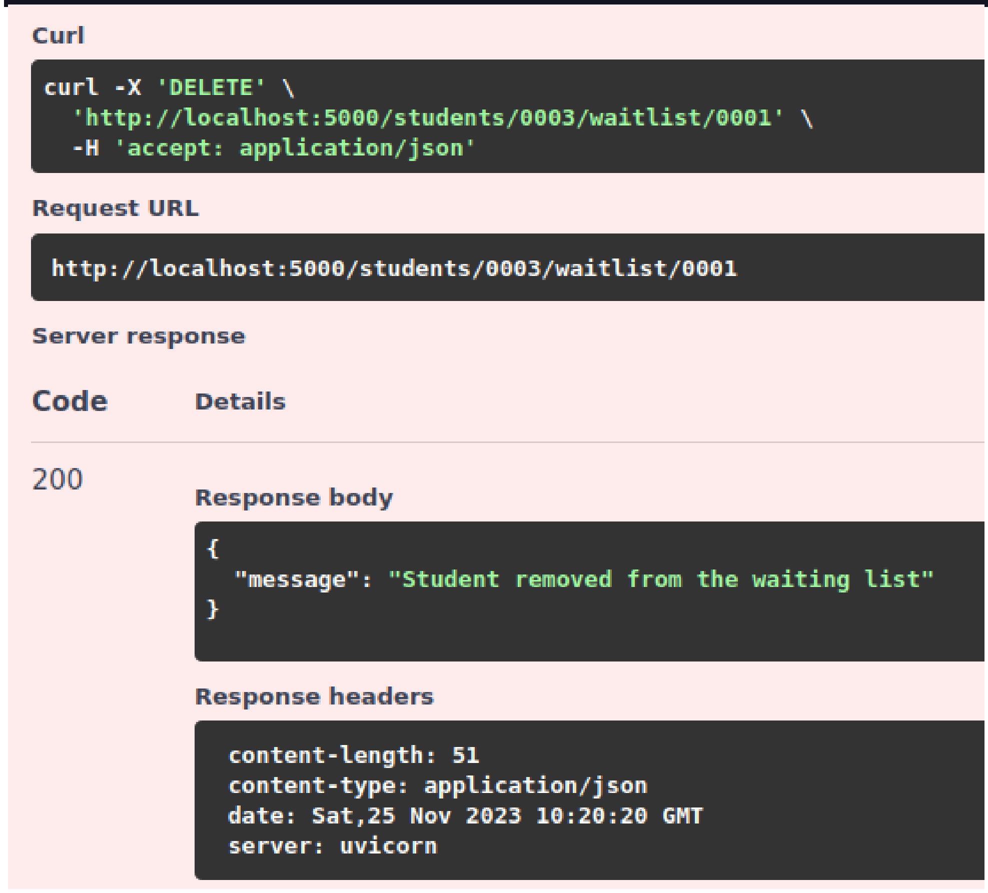
Response headers

```
 content-length: 154
 content-type: application/json
 date: Sat,25 Nov 2023 10:12:53 GMT
 server: uvicorn
```

DELETE /students/{student_id}/waitlist/{class_id}

This endpoint remove student from the waitlist, they should able to enroll in the class again and get waitlisted again

```
@router.delete("/students/{student_id}/waitlist/{class_id}",
tags=['Waitlist'], summary="Remove a student from a waiting list")

def remove_from_waitlist(student_id: str, class_id: str):

    # check if student exists in the database

    student_data = qh.query_student(dynamodb_client, student_id)

    if not student_data:

        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="No student found")

    # check if class exists in the database

    class_data = qh.query_class(dynamodb_client, class_id)

    if not class_data:

        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="No class found")

    waitlist_key = f"waitlist:{class_id}"

    if not r.exists(waitlist_key):

        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="No waitlist found")

    waitlist_data = r.lrange(waitlist_key, 0, -1)

    if not waitlist_data:

        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="No waitlist found")

    # Check if student is on waitlist

    id = f"s#{student_id}".encode('utf-8')

    if id not in waitlist_data:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="Student is not on waitlist")

    # Remove student from waitlist

    r.lrem(waitlist_key, 0, id)

    return {"message": "Student removed from the waiting list"}
```


Curl

```
curl -X 'DELETE' \
  'http://localhost:5000/students/0003/waitlist/0001' \
  -H 'accept: application/json'
```

Request URL

```
http://localhost:5000/students/0003/waitlist/0001
```

Server response

Code Details

200

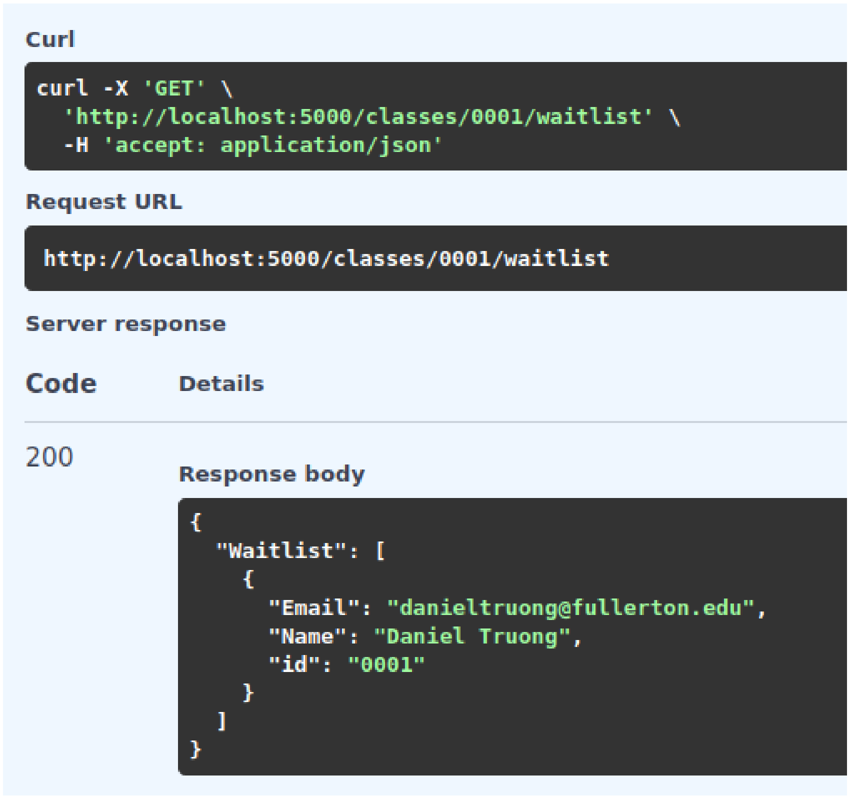
Response body

```
{
  "message": "Student removed from the waiting list"
}
```

Response headers

```
 content-length: 51
 content-type: application/json
 date: Sat,25 Nov 2023 10:20:20 GMT
 server: uvicorn
```

Now there’s only 1 student left in the waitlist for class 0001


Curl

```
curl -X 'GET' \
  'http://localhost:5000/classes/0001/waitlist' \
  -H 'accept: application/json'
```

Request URL

```
http://localhost:5000/classes/0001/waitlist
```

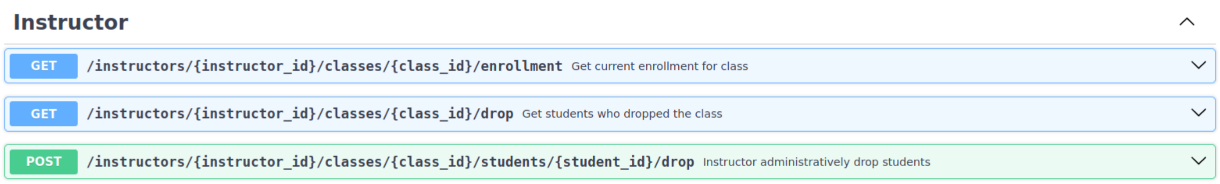
Server response

Code Details

200

Response body

```
{
  "Waitlist": [
    {
      "Email": "danieltruong@fullerton.edu",
      "Name": "Daniel Truong",
      "id": "0001"
    }
  ]
}
```


Instructor

GET /instructors/{instructor_id}/classes/{class_id}/enrollment Get current enrollment for class

GET /instructors/{instructor_id}/classes/{class_id}/drop Get students who dropped the class

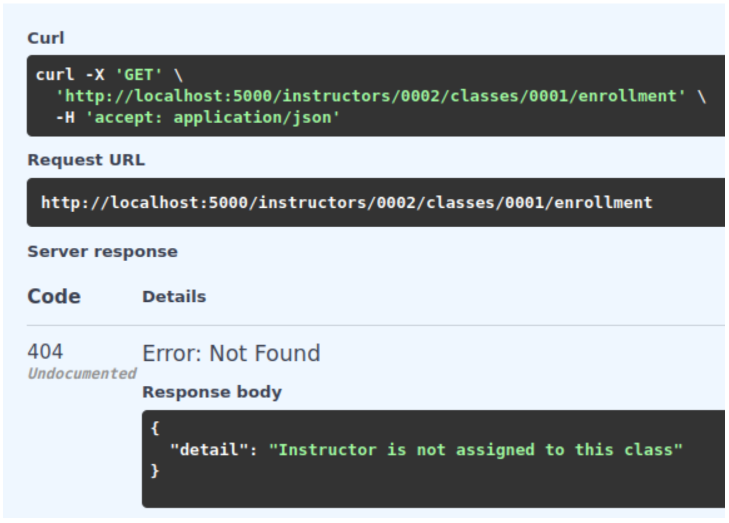
POST /instructors/{instructor_id}/classes/{class_id}/students/{student_id}/drop Instructor administratively drop students

GET /instructors/{instructor_id}/classes/{class_id}/enrollment

This endpoint get students enrolled in a class taught by instructor, instructor only able to view their assigned class

```
@router.get("/instructors/{instructor_id}/classes/{class_id}/enrollment",
tags=['Instructor'], summary="Get current enrollment for class")

def get_instructor_enrollment(instructor_id: str, class_id: str):

    # check if instructor exists in the database

    instructor_data = qh.query_instructor(dynamodb_client, instructor_id)

    if not instructor_data:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="No instructor found")

    # check if class exists in the database

    class_data = qh.query_class(dynamodb_client, class_id)

    if not class_data:

        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="No class found")

    # check if instructor is assigned to class

    class_instructor = qh.query_class_instructor(dynamodb_client,
instructor_id, class_id)

    if not class_instructor:

        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="Instructor is not assigned to this class")

    # get enrollment data

    enrollment_data = qh.query_enrolled_students(dynamodb_client, class_id)

    if not enrollment_data:

        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="No enrollment found")

    return {"Enrollment": enrollment_data}
```

Instructor 0002 trying to view enrollment, but the class is assigned to instructor 0001



Curl

```
curl -X 'GET' \
  'http://localhost:5000/instructors/0002/classes/0001/enrollment' \
  -H 'accept: application/json'
```

Request URL

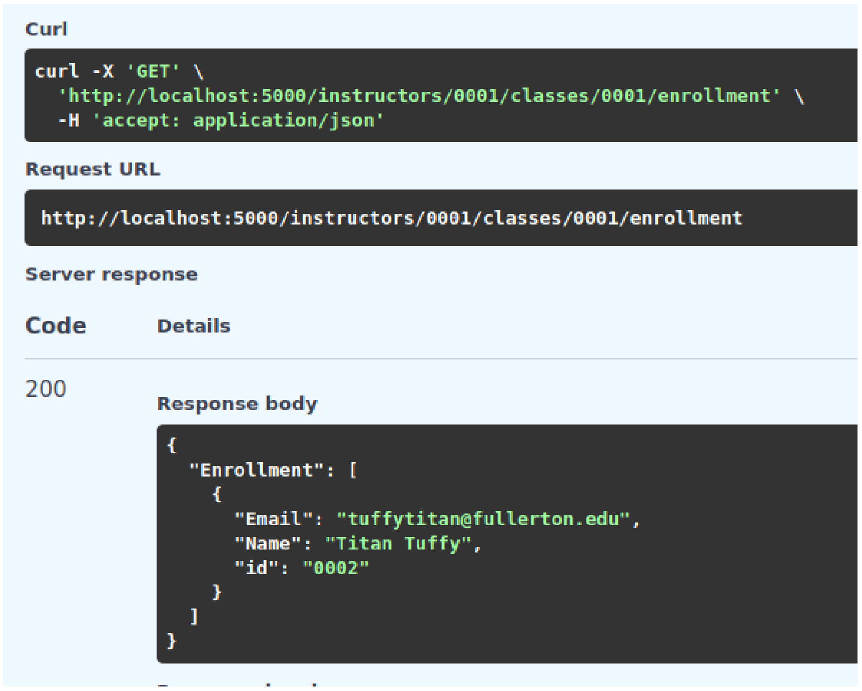
```
http://localhost:5000/instructors/0002/classes/0001/enrollment
```

Server response

Code	Details
404 *Undocumented*	Error: Not Found

Response body

```
{
  "detail": "Instructor is not assigned to this class"
}
```



Curl

```
curl -X 'GET' \
  'http://localhost:5000/instructors/0001/classes/0001/enrollment' \
  -H 'accept: application/json'
```

Request URL

```
http://localhost:5000/instructors/0001/classes/0001/enrollment
```

Server response

Code	Details
200	

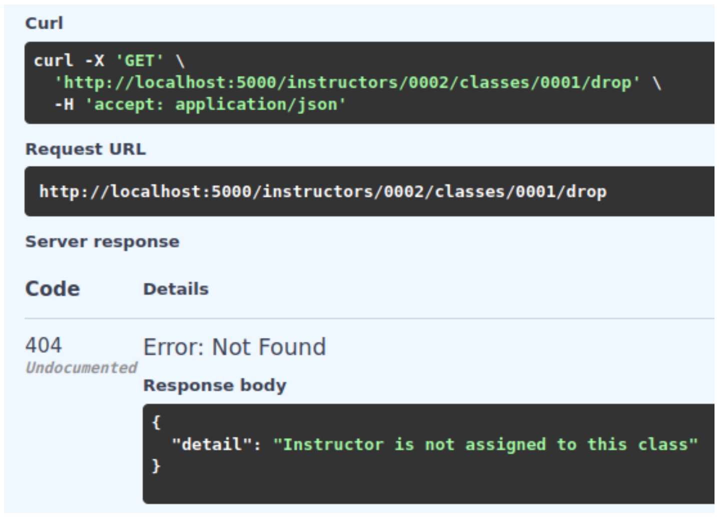
Response body

```
{
  "Enrollment": [
    {
      "Email": "tuffytitan@fullerton.edu",
      "Name": "Titan Tuffy",
      "id": "0002"
    }
  ]
}
```

GET /instructors/{instructor_id}/classes/{class_id}/drop

This endpoint allows instructor to view student who dropped their class, also cover case when instructor not assigned to the class

```
@router.get("/instructors/{instructor_id}/classes/{class_id}/drop",
tags=['Instructor'], summary="Get students who dropped the class")

def get_instructor_dropped(instructor_id: str, class_id: str):
   # check if instructor exists in the database

   instructor_data = qh.query_instructor(dynamodb_client, instructor_id)

   if not instructor_data:

       raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="No instructor found")

   # check if class exists in the database

   class_data = qh.query_class(dynamodb_client, class_id)

   if not class_data:

       raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="No class found")

   # check if instructor is assigned to class

   class_instructor = qh.query_class_instructor(dynamodb_client,
instructor_id, class_id)

   if not class_instructor:

       raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="Instructor is not assigned to this class")

   # get dropped data

   dropped_data = qh.query_dropped_students(dynamodb_client, class_id)

   if not dropped_data:

       raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="No dropped found")



   return {"Dropped": dropped_data}
```


Curl

```
curl -X 'GET' \
  'http://localhost:5000/instructors/0002/classes/0001/drop' \
  -H 'accept: application/json'
```

Request URL

```
http://localhost:5000/instructors/0002/classes/0001/drop
```

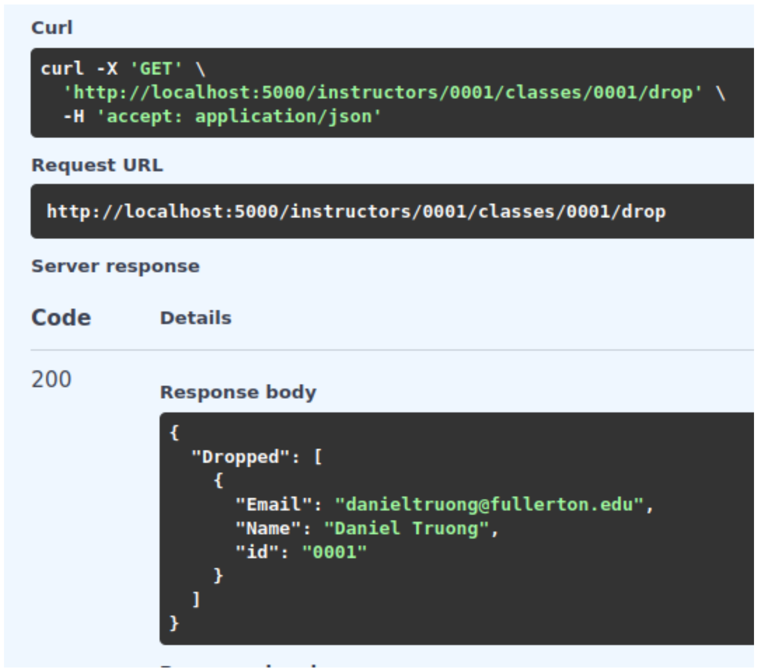
Server response

Code Details

404 *Undocumented* Error: Not Found

Response body

```
{
  "detail": "Instructor is not assigned to this class"
}
```


Curl

```
curl -X 'GET' \
  'http://localhost:5000/instructors/0001/classes/0001/drop' \
  -H 'accept: application/json'
```

Request URL

```
http://localhost:5000/instructors/0001/classes/0001/drop
```

Server response

Code Details

200

Response body

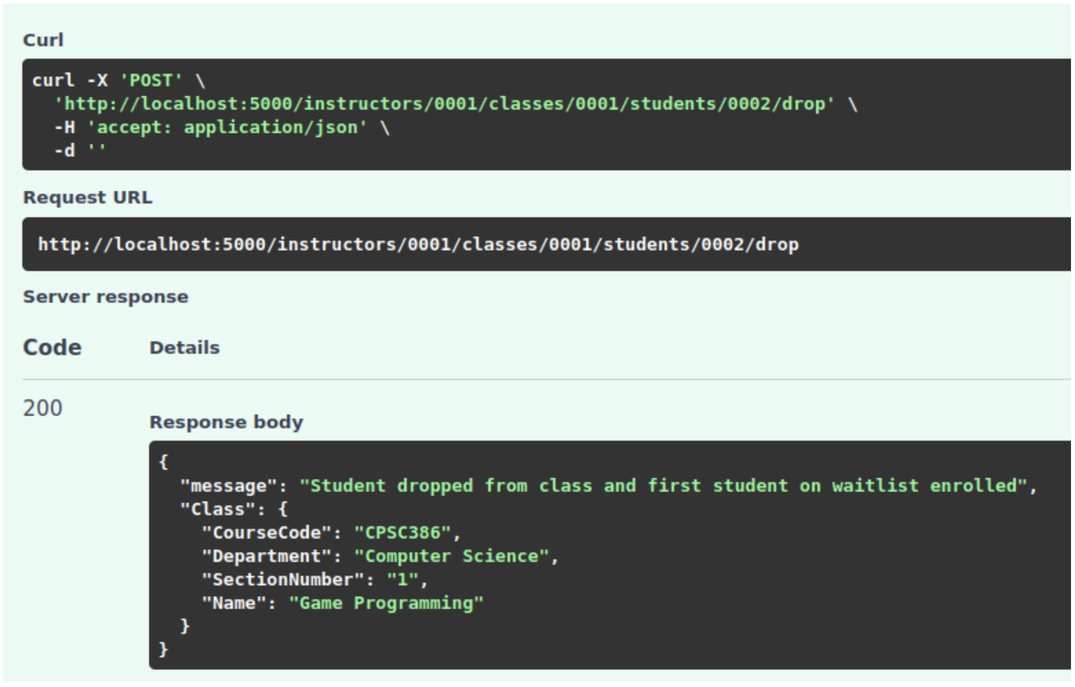
```
{
  "Dropped": [
    {
      "Email": "danieltruong@fullerton.edu",
      "Name": "Daniel Truong",
      "id": "0001"
    }
  ]
}
```

POST /instructors/{instructor_id}/classes/{class_id}/students/{student_id}/drop

This endpoint allows instructor to drop student from the class

```
@router.post("/instructors/{instructor_id}/classes/{class_id}/students/{st
udent_id}/drop", tags=['Instructor'], summary="Instructor administratively
drop students")
```

```python
def instructor_drop_class(instructor_id: str, class_id: str, student_id:
str):

    # check if instructor exists in the database

    instructor_data = qh.query_instructor(dynamodb_client, instructor_id)

    if not instructor_data:

        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="No instructor found")

    # check if class exists in the database

    class_data = qh.query_class(dynamodb_client, class_id)

    if not class_data:

        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="No class found")

    # check if instructor is assigned to class

    class_instructor = qh.query_class_instructor(dynamodb_client,
instructor_id, class_id)

    if not class_instructor:

        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="Instructor is not assigned to this class")

    # check if student exists in the database

    student_data = qh.query_student(dynamodb_client, student_id)

    if not student_data:

        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="No student found")

    # check if student is enrolled in the class

    enrolled_class = qh.query_enrolled_classes(dynamodb_client, student_id)

    if not enrolled_class:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="Student is not enrolled in this class")

    class_ids = [item['id'] for item in enrolled_class]

    if class_id not in class_ids:

        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="Student is not enrolled in this class")

    # Drop student from class

    drop_finished = qh.drop_student_from_class(dynamodb_client, student_id,
class_id)

    if not drop_finished:

        raise
HTTPException(status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
detail="Unable to drop student from class")

    # Decrement enrollment number in the database

    new_enrollment = class_data['currentEnroll'] - 1

    update_finished = qh.update_current_enroll(dynamodb_client, class_id,
new_enrollment)

    if not update_finished:

        raise
HTTPException(status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
detail="Unable to update class enrollment")

    # check if waitlist exists for class

    # check if freeze is on

    if not class_data['Frozen']:

        if r.exists(f"waitlist:{class_id}"):

            # first student on waitlist is automatically enrolled

            waitlist_data = r.lrange(f"waitlist:{class_id}", 0, 0)
            waitlist_data = [item.decode('utf-8')[2:] for item in
waitlist_data]

            # Enroll student in class

            enrolled_class = qh.update_enrolled_class(dynamodb_client,
waitlist_data[0], class_id)

            if not enrolled_class:

                raise
HTTPException(status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
detail="Unable to enroll student in class")

            # Increment enrollment number in the database

            new_enrollment = class_data['currentEnroll'] + 1

            update_finished = qh.update_current_enroll(dynamodb_client,
class_id, new_enrollment)

            if not update_finished:

                raise
HTTPException(status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
detail="Unable to update class enrollment")

            # Remove student from waitlist

            r.lrem(f"waitlist:{class_id}", 0, f"s#{waitlist_data[0]}")

            # Fetch the updated class data from the databas

            updated_class_data = qh.query_class(dynamodb_client, class_id)

            return {"message": "Student dropped from class and first
student on waitlist enrolled", "Class": updated_class_data["Detail"]}

    return {"message": "Student dropped from class"}
```



Curl

```
curl -X 'POST' \
  'http://localhost:5000/instructors/0001/classes/0001/students/0002/drop' \
  -H 'accept: application/json' \
  -d ''
```

Request URL

```
http://localhost:5000/instructors/0001/classes/0001/students/0002/drop
```

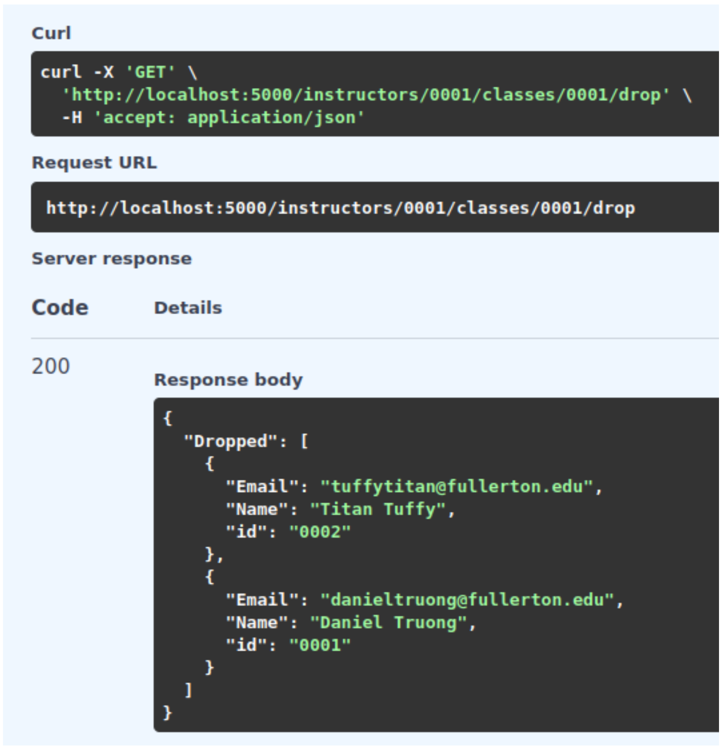
Server response

Code	Details
200	Response body

```
{
  "message": "Student dropped from class and first student on waitlist enrolled",
  "Class": {
    "CourseCode": "CPSC386",
    "Department": "Computer Science",
    "SectionNumber": "1",
    "Name": "Game Programming"
  }
}
```

Now there should be 2 students showing in the

GET /instructors/{instructor_id}/classes/{class_id}/drop



Curl

```
curl -X 'GET' \
  'http://localhost:5000/instructors/0001/classes/0001/drop' \
  -H 'accept: application/json'
```

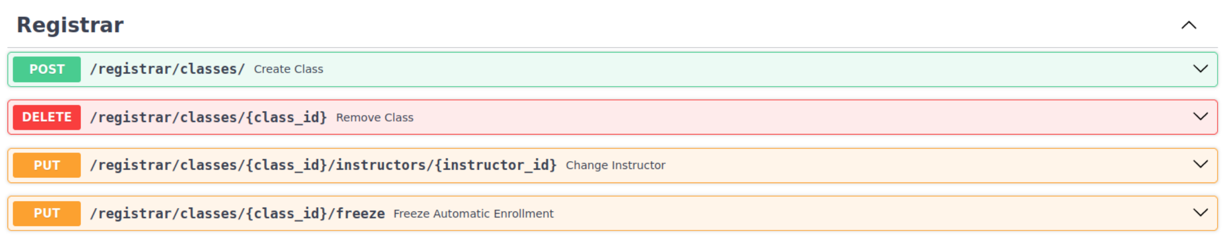
Request URL

```
http://localhost:5000/instructors/0001/classes/0001/drop
```

Server response

Code	Details
200	Response body

```
{
  "Dropped": [
    {
      "Email": "tuffytitan@fullerton.edu",
      "Name": "Titan Tuffy",
      "id": "0002"
    },
    {
      "Email": "danieltruong@fullerton.edu",
      "Name": "Daniel Truong",
      "id": "0001"
    }
  ]
}
```


Registrar

POST /registrar/classes/ Create Class

DELETE /registrar/classes/{class_id} Remove Class

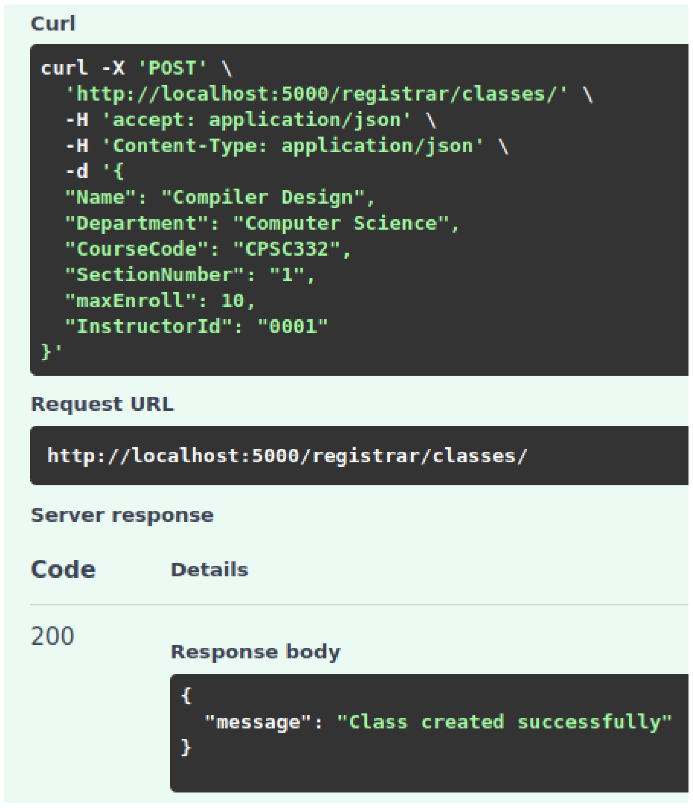
PUT /registrar/classes/{class_id}/instructors/{instructor_id} Change Instructor

PUT /registrar/classes/{class_id}/freeze Freeze Automatic Enrollment

POST /registrar/classes/

This endpoint allows registrar to create a class that students should be able to view the newly added class using the view available classes endpoint

```
@router.post("/registrar/classes/", tags=['Registrar'])

def create_class(class_data: Class):

   # Check instructor exists in the database

   instructor_data = qh.query_instructor(dynamodb_client,
class_data.InstructorId)

   if not instructor_data:

       raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="No instructor found")

   class_created = qh.create_class(dynamodb_client, class_data)

   if not class_created:

       raise
HTTPException(status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
detail="Unable to create class")

   return {"message": "Class created successfully"}
```


Curl

```
curl -X 'POST' \
  'http://localhost:5000/registrar/classes/' \
  -H 'accept: application/json' \
  -H 'Content-Type: application/json' \
  -d '{
  "Name": "Compiler Design",
  "Department": "Computer Science",
  "CourseCode": "CPSC332",
  "SectionNumber": "1",
  "maxEnroll": 10,
  "InstructorId": "0001"
}'
```

Request URL

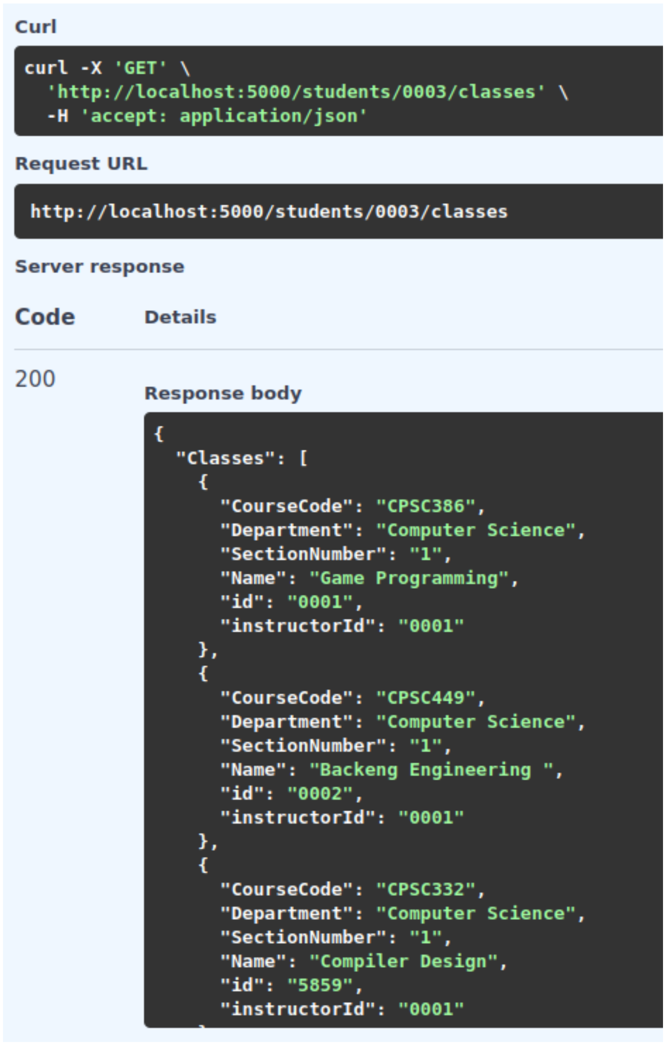
```
http://localhost:5000/registrar/classes/
```

Server response

Code	Details
200	Response body

```
{
  "message": "Class created successfully"
}
```

View the newly added class (id: 5859)


Curl

```
curl -X 'GET' \
  'http://localhost:5000/students/0003/classes' \
  -H 'accept: application/json'
```

Request URL

```
http://localhost:5000/students/0003/classes
```

Server response

Code Details

200

Response body

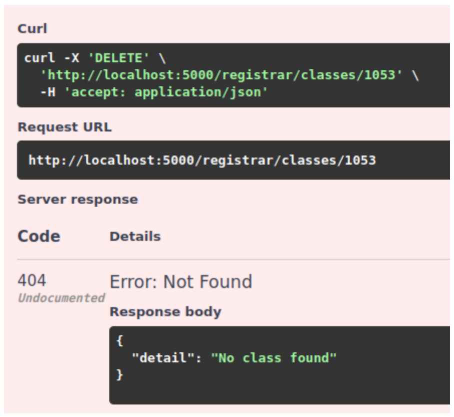
```
{
  "Classes": [
    {
      "CourseCode": "CPSC386",
      "Department": "Computer Science",
      "SectionNumber": "1",
      "Name": "Game Programming",
      "id": "0001",
      "instructorId": "0001"
    },
    {
      "CourseCode": "CPSC449",
      "Department": "Computer Science",
      "SectionNumber": "1",
      "Name": "Backeng Engineering ",
      "id": "0002",
      "instructorId": "0001"
    },
    {
      "CourseCode": "CPSC332",
      "Department": "Computer Science",
      "SectionNumber": "1",
      "Name": "Compiler Design",
      "id": "5859",
      "instructorId": "0001"
```

DELETE /registrar/classes/{class_id}

This endpoint allows registrar to remove a class, student who enrolled in that class also no longer enrolled, as well as instructor assigned to the class

```
@router.delete("/registrar/classes/{class_id}", tags=['Registrar'])

def remove_class(class_id: str):

    # Check if class exists in the database

    class_data = qh.query_class(dynamodb_client, class_id)

    if not class_data:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="No class found")

    removed_class = qh.delete_class(dynamodb_client, class_id)

    if not removed_class:

        raise
HTTPException(status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
detail="Unable to remove class")

    return {"message": "Class removed successfully"}
```

Class not found



Curl

```
curl -X 'DELETE' \
  'http://localhost:5000/registrar/classes/1053' \
  -H 'accept: application/json'
```

Request URL

```
http://localhost:5000/registrar/classes/1053
```

Server response

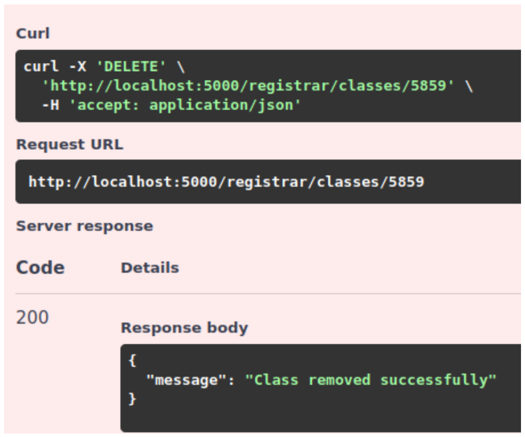
Code Details

404 *Undocumented* Error: Not Found

Response body

```
{
  "detail": "No class found"
}
```

Class removed:



Curl

```
curl -X 'DELETE' \
  'http://localhost:5000/registrar/classes/5859' \
  -H 'accept: application/json'
```

Request URL

```
http://localhost:5000/registrar/classes/5859
```

Server response

Code Details

200

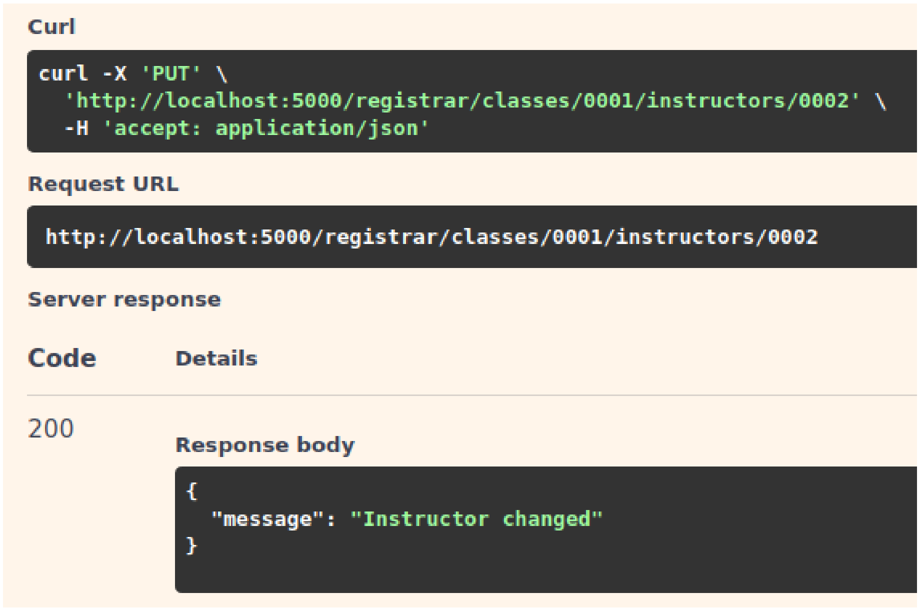
Response body

```
{
  "message": "Class removed successfully"
}
```

PUT /registrar/classes/{class_id}/instructors/{instructor_id}

This endpoint allows registrar to change instructor for a class

```
# DONE: Change the assigned instructor for a class

@router.put("/registrar/classes/{class_id}/instructors/{instructor_id}",
tags=['Registrar'])

def change_instructor(class_id: str, instructor_id: str):

    # Check if class exists in the database

    class_data = qh.query_class(dynamodb_client, class_id)

    if not class_data:

        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="No class found")

    # Check if instructor exists in the database

    instructor_data = qh.query_instructor(dynamodb_client, instructor_id)

    if not instructor_data:

        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND,
detail="No instructor found")

    # Change the assigned instructor for the class

    instructor_changed = qh.change_instructor(dynamodb_client, class_id,
instructor_id)

    if not instructor_changed:

        raise
HTTPException(status_code=status.HTTP_500_INTERNAL_SERVER_ERROR,
detail="Unable to change instructor")

    # return success message

    return {"message": "Instructor changed"}
```


Curl

```
curl -X 'PUT' \
  'http://localhost:5000/registrar/classes/0001/instructors/0002' \
  -H 'accept: application/json'
```

Request URL

```
http://localhost:5000/registrar/classes/0001/instructors/0002
```

Server response

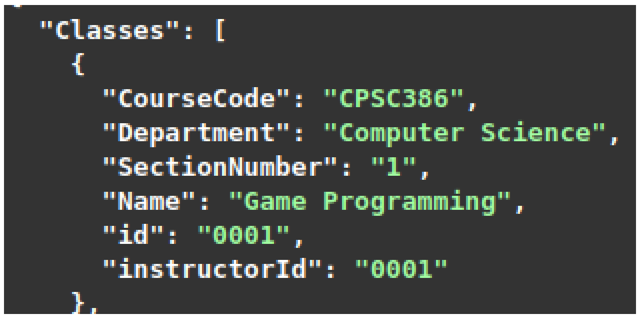
Code Details

200

Response body

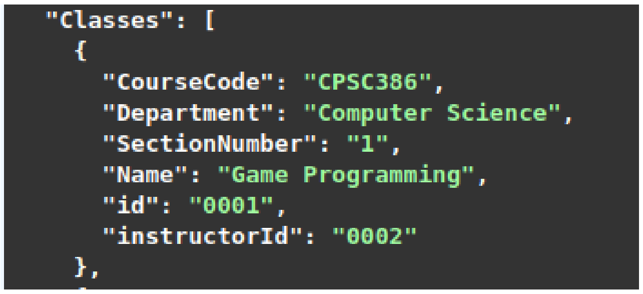
```
{
  "message": "Instructor changed"
}
```

Before:



```
  "Classes": [
    {
      "CourseCode": "CPSC386",
      "Department": "Computer Science",
      "SectionNumber": "1",
      "Name": "Game Programming",
      "id": "0001",
      "instructorId": "0001"
    },
```

After:



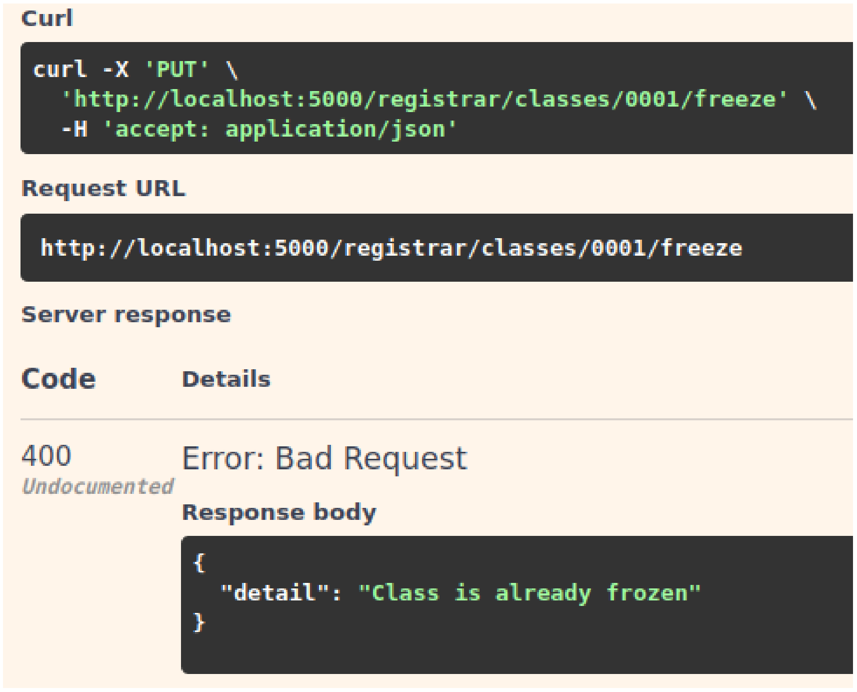
```
  "Classes": [
    {
      "CourseCode": "CPSC386",
      "Department": "Computer Science",
      "SectionNumber": "1",
      "Name": "Game Programming",
      "id": "0001",
      "instructorId": "0002"
    },
```

PUT /registrar/classes/{class_id}/freeze

This endpoint allows registrar to freeze auto enrollment of a class, so when this is on, student in the waitlist won't be able to auto enroll in the class even if a student in a class dropped, student also not able to enroll in the frozen class

```python
@router.put("/registrar/classes/{class_id}/freeze", tags=['Registrar'])
def freeze_automatic_enrollment(class_id: str):
    # Check if class exists in the database
    class_data = qh.query_class(dynamodb_client, class_id)
    if not class_data:
        raise HTTPException(status_code=status.HTTP_404_NOT_FOUND, detail="No class found")
    # Check if class is already frozen
    if class_data['Frozen']:
        raise HTTPException(status_code=status.HTTP_400_BAD_REQUEST, detail="Class is already frozen")
    # Freeze the class
    freeze_finished = qh.freeze_enrollment(dynamodb_client, class_id)
    if not freeze_finished:
        raise HTTPException(status_code=status.HTTP_500_INTERNAL_SERVER_ERROR, detail="Unable to freeze enrollment")
    # return success message
    return {"message": "Enrollment frozen"}
```

If the class already frozen


Curl

```
curl -X 'PUT' \
  'http://localhost:5000/registrar/classes/0001/freeze' \
  -H 'accept: application/json'
```

Request URL

```
http://localhost:5000/registrar/classes/0001/freeze
```

Server response

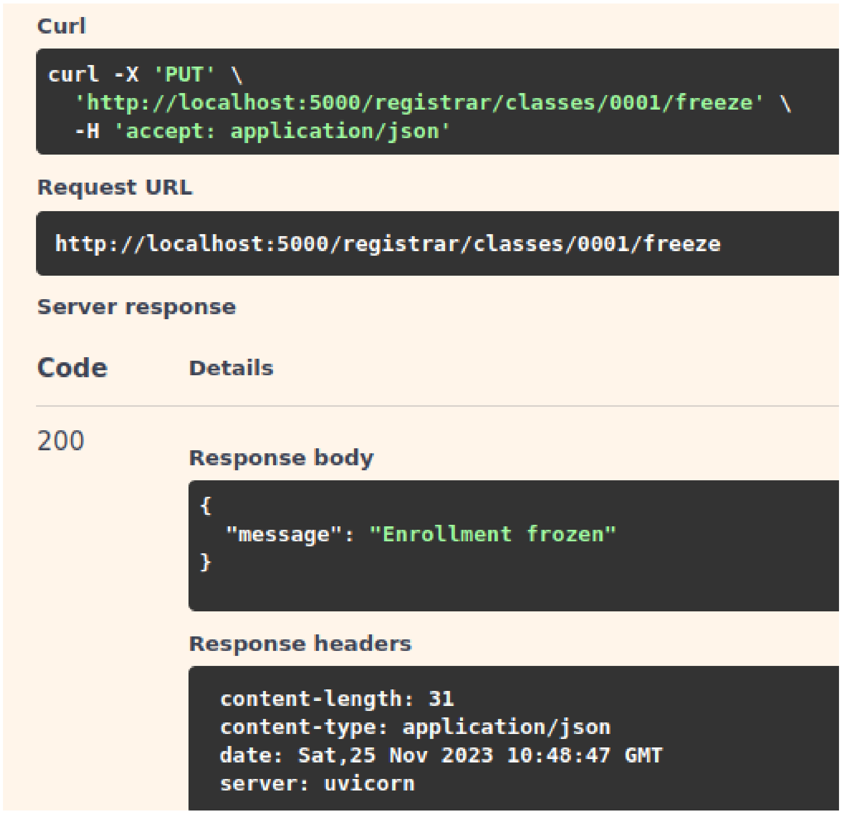
Code Details

400 *Undocumented* Error: Bad Request

Response body

```
{
  "detail": "Class is already frozen"
}
```

Class frozen successfully


Curl

```
curl -X 'PUT' \
  'http://localhost:5000/registrar/classes/0001/freeze' \
  -H 'accept: application/json'
```

Request URL

```
http://localhost:5000/registrar/classes/0001/freeze
```

Server response

Code Details

200

Response body

```
{
  "message": "Enrollment frozen"
}
```

Response headers

```
 content-length: 31
 content-type: application/json
 date: Sat,25 Nov 2023 10:48:47 GMT
 server: uvicorn
```