Fall 2023, CPSC 449, Section 1

Project 4

Edwin Peraza

Michael Carey

Melissa Huynh

Donald Novasky

Ryan Novoa

Codie Tamida

Task 1: Create an enrollment notification service

A new service was created to allow students to subscribe to updates for a course, list their current subscriptions, and to unsubscribe from course updates.

Endpoint to subscribe to a course updates:

```python
# Subscribe to notifications for a new course
@router.post("/students/{student_id}/subscribe/{class_id}", tags=["Notification"])
def subscribe_to_course(student_id: int, class_id: int, email: str = "", webhook_url:
str = ""):

   # Check if student and class exist
   # Fetch student data from db
   student_data = enrollment.get_user_item(student_id)

   # Fetch class data from db
   class_data = enrollment.get_class_item(class_id)

   # Check if exist
   if not student_data or not class_data:
       raise HTTPException(
           status_code=404, detail=f"Student_id {student_id} or class_id {class_id}
not found"
       )

   # Check if the student already has a subscription for the class
   if sub.is_student_subscribed(student_id, class_id):
       raise HTTPException(status_code=400, detail=f"Student_id {student_id} already
has a subscription for class_id {class_id}.")

   # Check if email or webhook_url is provided
   if not email and not webhook_url:
       raise HTTPException(status_code=400, detail="Provide either an email address or
a Webhook callback URL, or both.")

   # Create a subscription payload
   subscription_payload = {
       "email": email,
       "webhook_url": webhook_url,
   }

   # Store subscription in Redis
   sub.add_subscription(student_id, class_id, subscription_payload)
   sub_data = Sub_List(
       class_id=class_id,
       email=subscription_payload["email"],
       webhook_url=subscription_payload["webhook_url"]
   )

   return {"message": "Subscription successful", "subscription": sub_data}
```

Endpoint to list current subscription for an user:

```
# List all subscriptions for a student
@router.get("/students/{student_id}/subscriptions", tags=["Notification"])
def list_current_subscriptions(student_id: int):

   # check if student exists
   # Fetch student data from db
   student_data = enrollment.get_user_item(student_id)

   if not student_data:
       raise HTTPException(
           status_code=404, detail=f"Student_id {student_id} not found"
       )

   # Check if the student has any subscriptions
   if not sub.check_student_subscription(student_id):
       raise HTTPException(status_code=404, detail=f"Student_id {student_id} has no
subscriptions.")

   # Retrieve subscriptions from Redis using the correct key pattern
   subscriptions_data = sub.get_all_subscriptions(student_id)
   print(subscriptions_data)

   # Convert subscriptions to instances of Sub_List model
   subscriptions = [
       Sub_List(class_id=sub_data['class_id'], email=sub_data['email'],
webhook_url=sub_data['webhook_url'])
       for sub_data in subscriptions_data
   ]

   return {"message": "Current subscriptions", "subscriptions": subscriptions}
```

Endpoint to unsubscribe from updates from a course:

```
# Allow student to unsubscribe from a course
@router.delete("/students/{student_id}unsubscribe/{class_id}", tags=["Notification"])
def unsubscribe_from_course(student_id: int, class_id: int):

   # Check if student and class exist
   # Fetch student data from db
   student_data = enrollment.get_user_item(student_id)

   # Fetch class data from db
   class_data = enrollment.get_class_item(class_id)

   # Check if exist
   if not student_data or not class_data:
       raise HTTPException(
           status_code=404, detail=f"Student_id {student_id} or class_id {class_id}
not found"
       )

   # Check if the student has a subscription for the class
   if not sub.is_student_subscribed(student_id, class_id):
       raise HTTPException(status_code=404, detail=f"Student_id {student_id} is not
subscribed to class_id {class_id}.")
```

```
   sub.delete_subscription(student_id, class_id)

   return {"message": f"Successfully unsubscribed student_id {student_id} from
class_id {class_id}"}
```

Redis model for notifications service

Subscription information was stored in redis using the following format:

Key: "subscriptions:{student_id}"

Data: {class_id: {“email”: email string, “webhook_url”: url string}}

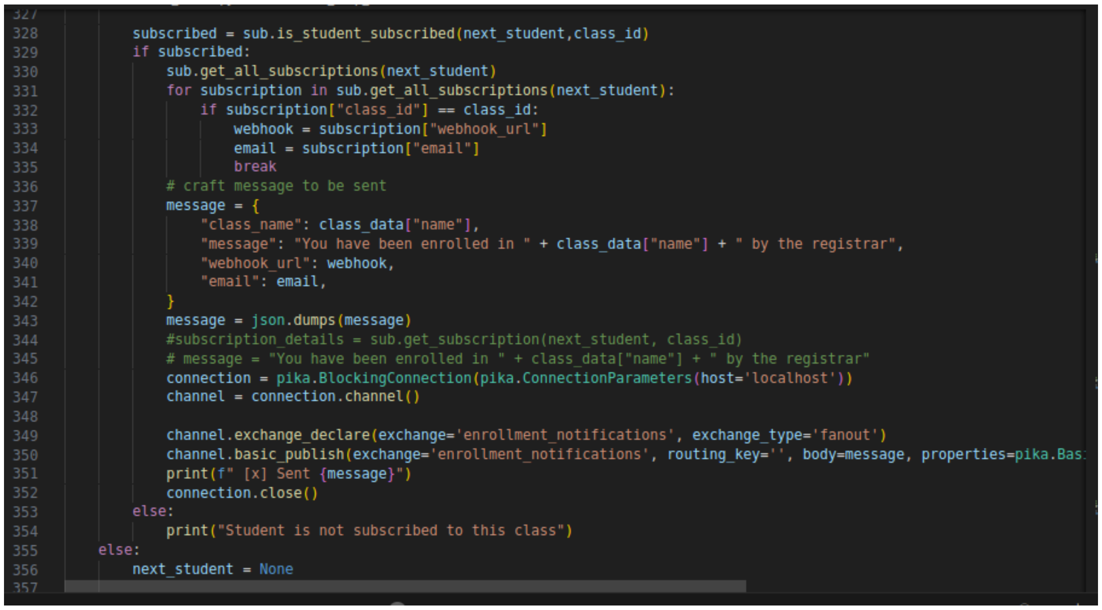
For reference the class_id and student_id are justintegers, which represents the unique id of the class and student respectively.

There was also a class created in the enrollment_redis file called “Subscription” which has several functions used to manipulate data in the redis db.

Task 2: Producing enrollment notifications

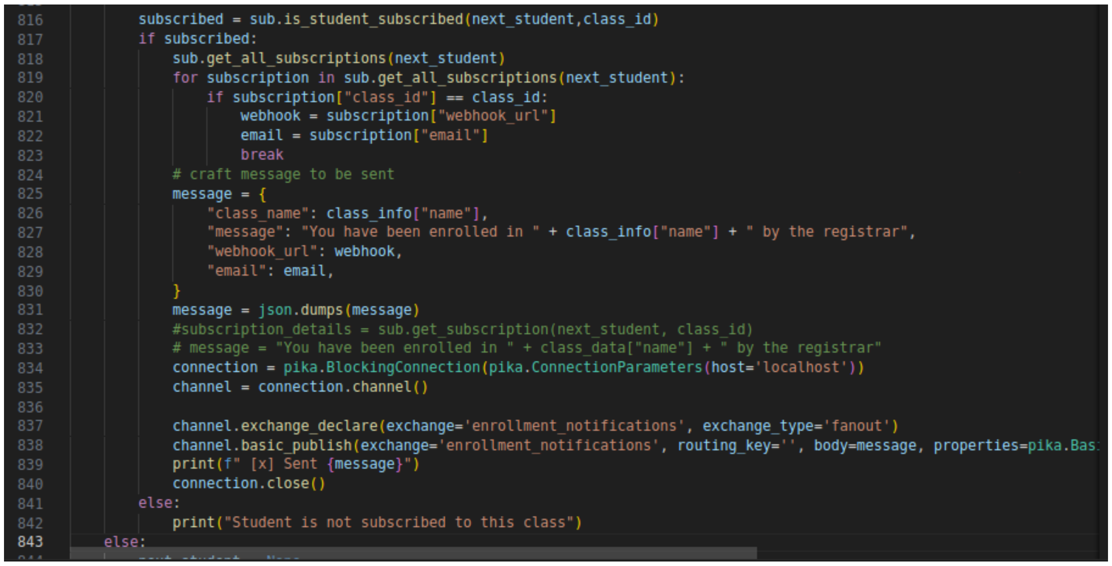
Producer process is created for when a student is added to the class from the waitlist. The producer code for 2 endpoints will send a message to the RabbitMQ exchange.

For the endpoint where a student drops from a class and student is then added from the waitlist:



```
        subscribed = sub.is_student_subscribed(next_student,class_id)
        if subscribed:
            sub.get_all_subscriptions(next_student)
            for subscription in sub.get_all_subscriptions(next_student):
                if subscription["class_id"] == class_id:
                    webhook = subscription["webhook_url"]
                    email = subscription["email"]
                    break
            # craft message to be sent
            message = {
                "class_name": class_data["name"],
                "message": "You have been enrolled in " + class_data["name"] + " by the registrar",
                "webhook_url": webhook,
                "email": email,
            }
            message = json.dumps(message)
            #subscription_details = sub.get_subscription(next_student, class_id)
            # message = "You have been enrolled in " + class_data["name"] + " by the registrar"
            connection = pika.BlockingConnection(pika.ConnectionParameters(host='localhost'))
            channel = connection.channel()

            channel.exchange_declare(exchange='enrollment_notifications', exchange_type='fanout')
            channel.basic_publish(exchange='enrollment_notifications', routing_key='', body=message, properties=pika.Bas
            print(f" [x] Sent {message}")
            connection.close()
        else:
            print("Student is not subscribed to this class")
    else:
        next_student = None
```

For the endpoint, where an instructor administratively drops a student and a student is added from the waitlist:



```
        subscribed = sub.is_student_subscribed(next_student,class_id)
        if subscribed:
            sub.get_all_subscriptions(next_student)
            for subscription in sub.get_all_subscriptions(next_student):
                if subscription["class_id"] == class_id:
                    webhook = subscription["webhook_url"]
                    email = subscription["email"]
                    break
            # craft message to be sent
            message = {
                "class_name": class_info["name"],
                "message": "You have been enrolled in " + class_info["name"] + " by the registrar",
                "webhook_url": webhook,
                "email": email,
            }
            message = json.dumps(message)
            #subscription_details = sub.get_subscription(next_student, class_id)
            # message = "You have been enrolled in " + class_data["name"] + " by the registrar"
            connection = pika.BlockingConnection(pika.ConnectionParameters(host='localhost'))
            channel = connection.channel()

            channel.exchange_declare(exchange='enrollment_notifications', exchange_type='fanout')
            channel.basic_publish(exchange='enrollment_notifications', routing_key='', body=message, properties=pika.Bas
            print(f" [x] Sent {message}")
            connection.close()
        else:
            print("Student is not subscribed to this class")
    else:
```

Task 3: Consuming enrollment notifications

Two consumer processes were created, one to send email notifications and one to send Webhook callback notifications.

Email consumer:

```
import pika
import smtplib
from email.message import EmailMessage
import json

def email_callback(ch, method, properties, body):
    print(f" [x] Received {body}")
    data = json.loads(body)
    to_address = data.get('email')
    message_text = data.get('message')

    # Create EmailMessage object
    msg = EmailMessage()
    msg.set_content(message_text)
    msg['Subject'] = f'Enrollment Notification for {data.get("class_name")}'
    msg['From'] = 'edwinperaza@csu.fullerton.edu'
    msg['To'] = to_address

    # Send email using smtplib
    server = smtplib.SMTP('localhost', 8025)
    # server = smtplib.SMTP('localhost')
    server.send_message(msg)
    server.quit()
    print(f" [x] Sent email to {to_address}")
    ch.basic_ack(delivery_tag=method.delivery_tag)

def main():
    # Set up RabbitMQ connection and channel
    connection = pika.BlockingConnection(pika.ConnectionParameters('localhost'))
    channel = connection.channel()

    # Declare a fanout exchange
    channel.exchange_declare(exchange='enrollment_notifications',
exchange_type='fanout')

    # Declare a queue and bind it to the exchange
    result = channel.queue_declare(queue='', exclusive=True, durable=True)
    queue_name = result.method.queue

    channel.queue_bind(exchange='enrollment_notifications', queue=queue_name)

    # Set up the consumer callback
    channel.basic_consume(queue=queue_name, on_message_callback=email_callback)

    # Start consuming messages
    print('Email Notification Consumer is waiting for messages. To exit press
CTRL+C')
    channel.start_consuming()


if __name__ == '__main__':
    main()
```

Webhook consumer:

```
import pika
import httpx
import json

def webhook_callback(ch, method, properties, body):
    print(f" [x] Received {body}")
    data = json.loads(body)
    webhook_url = data.get('webhook_url')
    message_text = data.get('message')

    try:
        response = httpx.post(webhook_url, json={'message': message_text})
        response.raise_for_status()
        ch.basic_ack(delivery_tag=method.delivery_tag)
        print(f" [x] Sent webhook callback to {webhook_url}")
    except httpx.HTTPError as e:
        print(f"Error sending Webhook callback: {e}")

def main():
    # Set up RabbitMQ connection and channel
    connection = pika.BlockingConnection(pika.ConnectionParameters('localhost'))
    channel = connection.channel()

    # Declare a fanout exchange
    channel.exchange_declare(exchange='enrollment_notifications',
exchange_type='fanout')

    # Declare a queue and bind it to the exchange
    result = channel.queue_declare(queue='', exclusive=True, durable=True)
    queue_name = result.method.queue

    channel.queue_bind(exchange='enrollment_notifications', queue=queue_name)

    # Set up the consumer callback
    channel.basic_consume(queue=queue_name, on_message_callback=webhook_callback)

    # Start consuming messages
    print('Webhook Callback Consumer is waiting for messages. To exit press
CTRL+C')
    channel.start_consuming()


if __name__ == '__main__':
    main()
```

Task 4: Testing

For testing, the project uses the aiosmtpd SMTP server and smee.io for webhooks. The aiosmtpd server is installed and run in debugging mode using the following commands:

```
python -m pip install aiosmtpd
python -m aiosmtpd -n -d
```

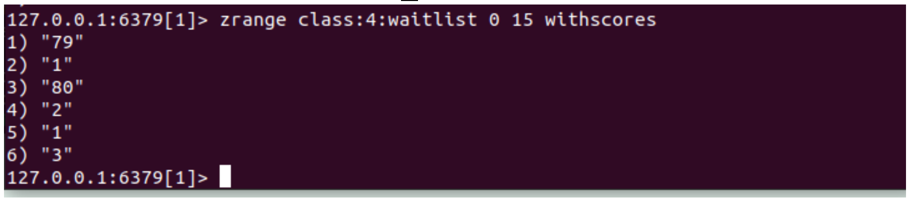
Updated Procfile:

```
enrollment: uvicorn enrollment.enrollment:app --host 0.0.0.0 --port $PORT
--reload
primary: bin/litefs mount -config etc/primary.yml
secondary: bin/litefs mount -config etc/secondary.yml
tertiary: bin/litefs mount -config etc/tertiary.yml
krakend: echo ./etc/krakend.json | entr -nrz krakend run --config
etc/krakend.json --port $PORT
dynamodb: java -Djava.library.path=./DynamoDBLocal_lib -jar DynamoDBLocal.jar
-sharedDb --port $PORT
notification: uvicorn notification.notification:app --host 0.0.0.0 --port $PORT
--reload
email-consumer: python consumer/email_consumer.py
webhook-consumer: python consumer/webhook_consumer.py
```

```
aiosmtpd: python -m aiosmtpd -n -d
```

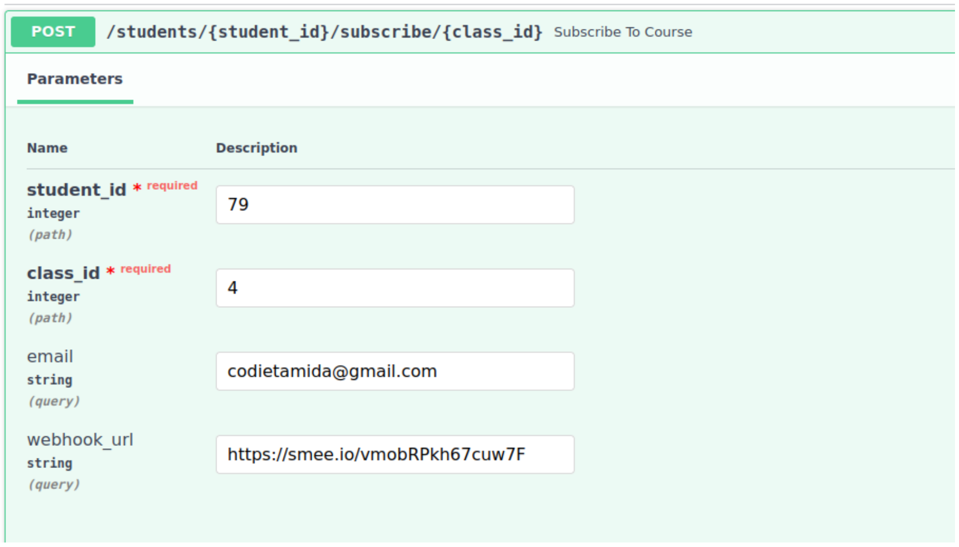
Testing when a student that is enrolled and drops from the class. The 1st student on the waitlist should be added to the class and receive a notification to the webhook URL or via e-mail or both.

Below is the waitlist for the students in class_id 4



```
127.0.0.1:6379[1]> zrange class:4:waitlist 0 15 withscores
1) "79"
2) "1"
3) "80"
4) "2"
5) "1"
6) "3"
127.0.0.1:6379[1]>
```

The user with student_id 79, is subscribing to the notification service

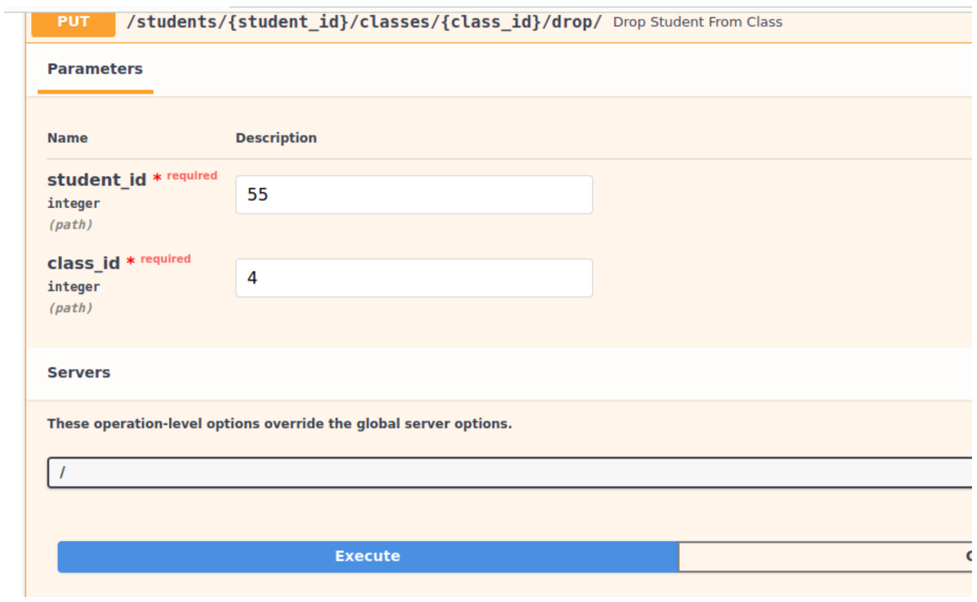


POST /students/{student_id}/subscribe/{class_id} Subscribe To Course

Parameters

Name	Description
student_id * required integer (path)	79
class_id * required integer (path)	4
email string (query)	codietamida@gmail.com
webhook_url string (query)	https://smee.io/vmobRPkh67cuw7F

Student_id 55 is dropping from class_id 4


PUT `/students/{student_id}/classes/{class_id}/drop/` Drop Student From Class

Parameters

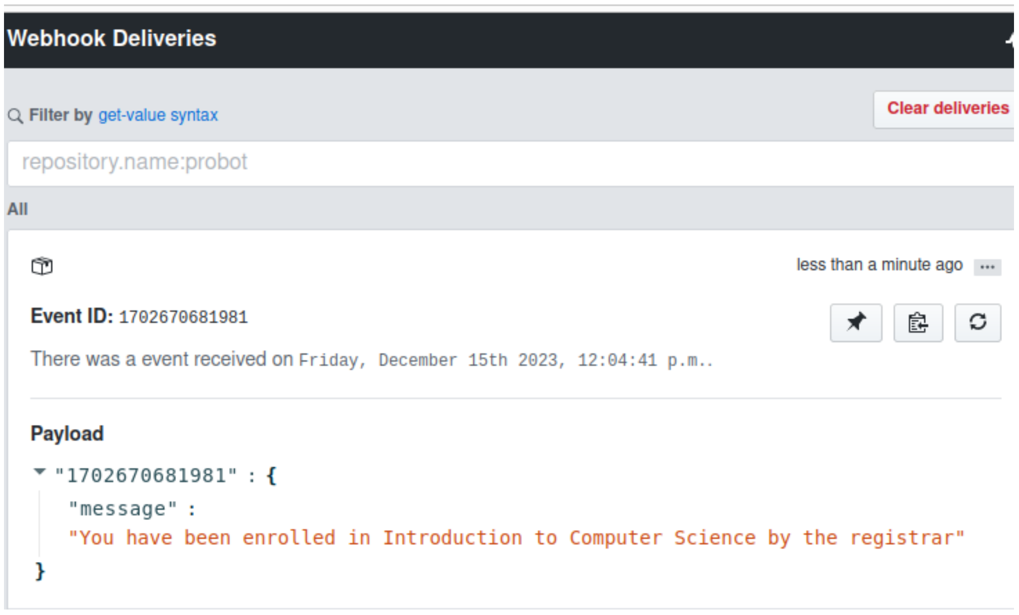
Name	Description
student_id * required integer (path)	55
class_id * required integer (path)	4

Servers

These operation-level options override the global server options.

/

Notification from the webhook URL that the student was enrolled into class_id 4


Webhook Deliveries

Filter by get-value syntax

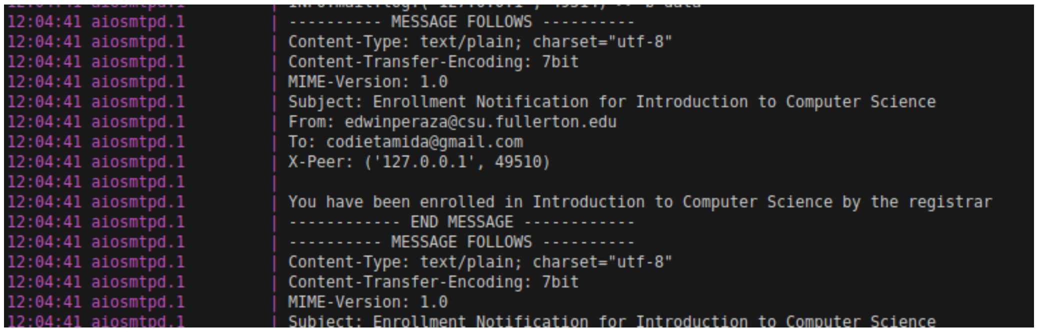
Event ID: 1702670681981

There was a event received on Friday, December 15th 2023, 12:04:41 p.m..

Payload

```
"1702670681981" : {
  "message" :
  "You have been enrolled in Introduction to Computer Science by the registrar"
}
```

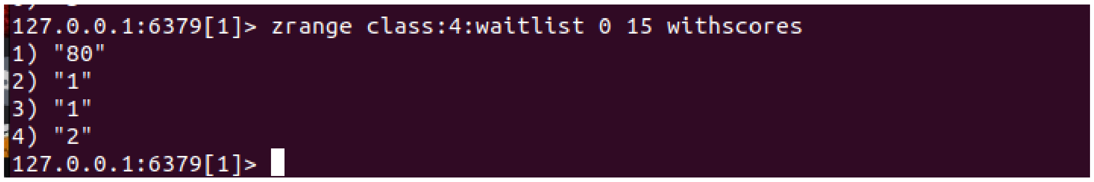
Notification through aiosmtpd that the student was registered/enrolled for class_id 4 (We have multiple e-mail and webhook consumers so there are multiple messages delivered)



```
12:04:41 aiosmtpd.1           | ---------- MESSAGE FOLLOWS ----------
12:04:41 aiosmtpd.1           | Content-Type: text/plain; charset="utf-8"
12:04:41 aiosmtpd.1           | Content-Transfer-Encoding: 7bit
12:04:41 aiosmtpd.1           | MIME-Version: 1.0
12:04:41 aiosmtpd.1           | Subject: Enrollment Notification for Introduction to Computer Science
12:04:41 aiosmtpd.1           | From: edwinperaza@csu.fullerton.edu
12:04:41 aiosmtpd.1           | To: codietamida@gmail.com
12:04:41 aiosmtpd.1           | X-Peer: ('127.0.0.1', 49510)
12:04:41 aiosmtpd.1           |
12:04:41 aiosmtpd.1           | You have been enrolled in Introduction to Computer Science by the registrar
12:04:41 aiosmtpd.1           | ------------ END MESSAGE ------------
12:04:41 aiosmtpd.1           | ---------- MESSAGE FOLLOWS ----------
12:04:41 aiosmtpd.1           | Content-Type: text/plain; charset="utf-8"
12:04:41 aiosmtpd.1           | Content-Transfer-Encoding: 7bit
12:04:41 aiosmtpd.1           | MIME-Version: 1.0
12:04:41 aiosmtpd.1           | Subject: Enrollment Notification for Introduction to Computer Science
```

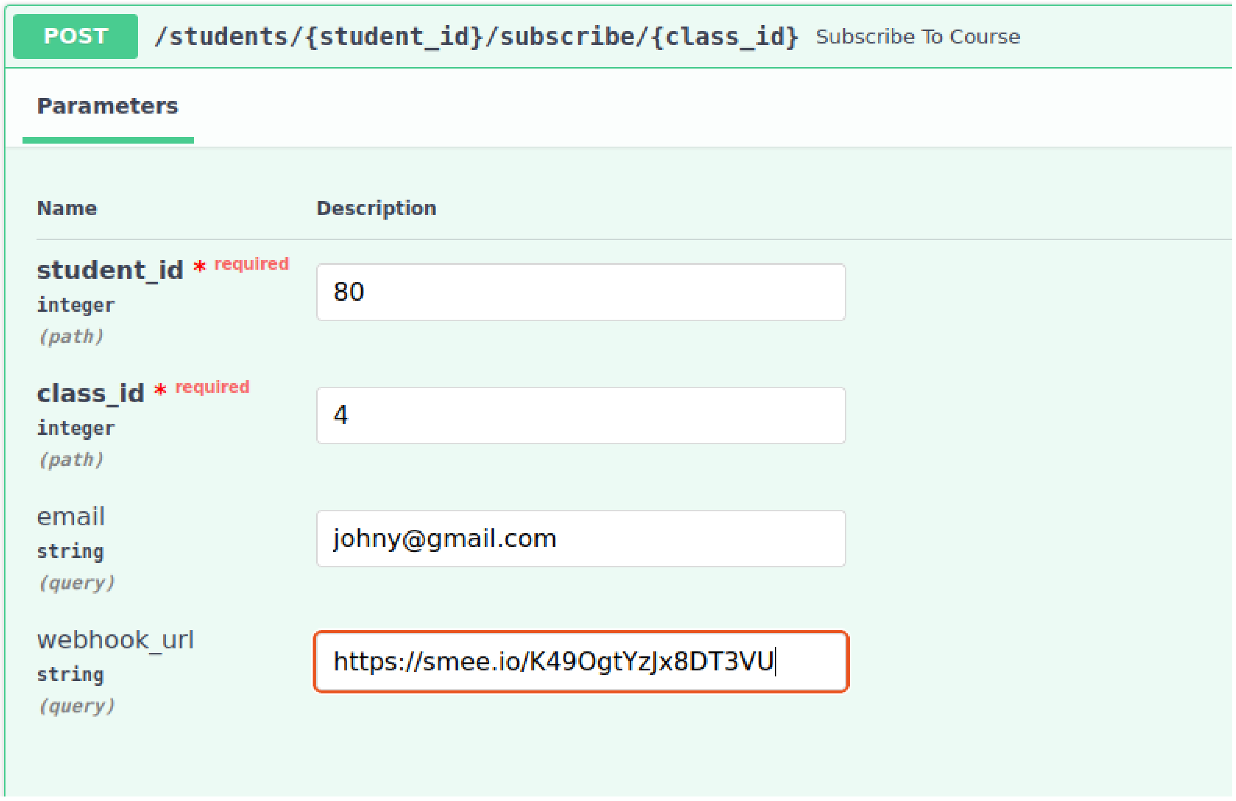
Testing when a student is dropped by the instructor, administratively, and the first student on the waitlist is enrolled into the class_id 4

The current waitlist for class_id 4



```
127.0.0.1:6379[1]> zrange class:4:waitlist 0 15 withscores
1) "80"
2) "1"
3) "1"
4) "2"
127.0.0.1:6379[1]>
```

Student_id 80 is subscribing to notifications for class_id 4 from webhook URL and e-mail

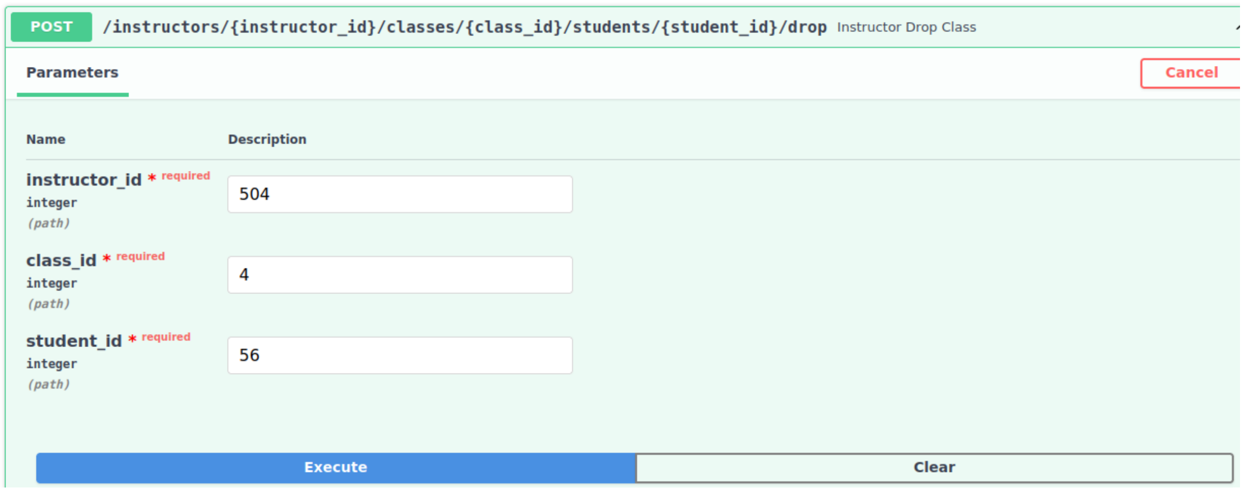


POST /students/{student_id}/subscribe/{class_id} Subscribe To Course

Parameters

Name	Description
student_id * required integer (path)	80
class_id * required integer (path)	4
email string (query)	johny@gmail.com
webhook_url string (query)	https://smee.io/K49OgtYzJx8DT3VU

The instructor of class_id 4 (instructor_id 504) dropping student_id 56 from the course

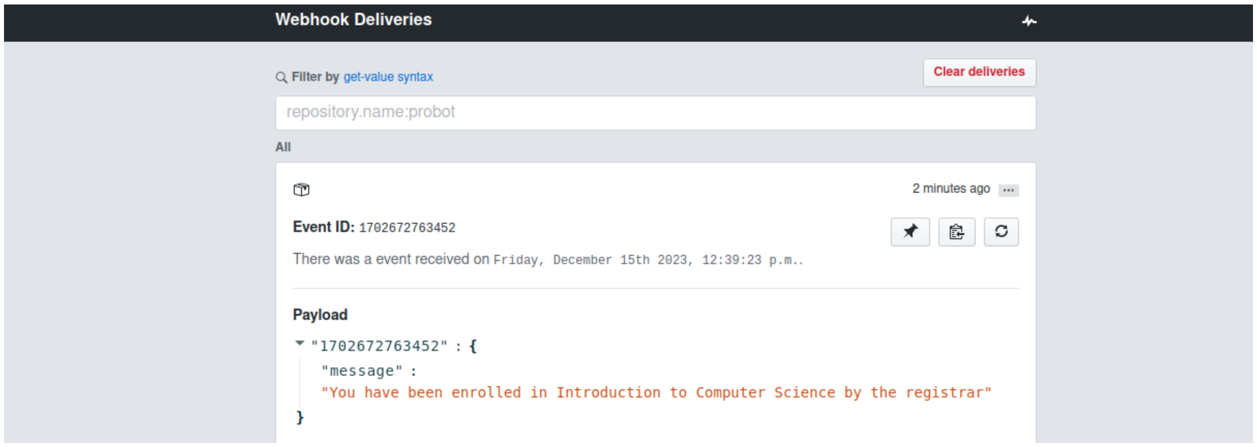


POST `/instructors/{instructor_id}/classes/{class_id}/students/{student_id}/drop` Instructor Drop Class

Parameters

Name	Description
instructor_id * required integer (path)	504
class_id * required integer (path)	4
student_id * required integer (path)	56

Webhook URL updated, notifying student that they were enrolled into the class



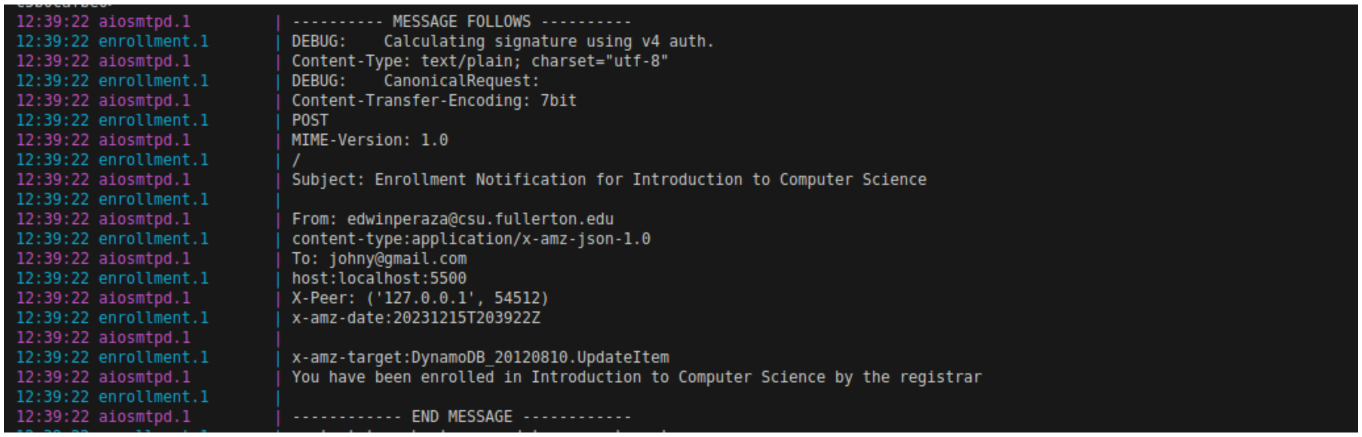
Event ID: 1702672763452

There was a event received on Friday, December 15th 2023, 12:39:23 p.m..

Payload

```
"1702672763452" : {
  "message" :
  "You have been enrolled in Introduction to Computer Science by the registrar"
}
```

Message via aiosmtpd that e-mail was sent to the the e-mail address for student_id 80 (johny@gmail.com) that they were enrolled into the class



```
12:39:22 aiosmtpd.1          | ---------- MESSAGE FOLLOWS ----------
12:39:22 enrollment.1        | DEBUG:    Calculating signature using v4 auth.
12:39:22 aiosmtpd.1          | Content-Type: text/plain; charset="utf-8"
12:39:22 enrollment.1        | DEBUG:    CanonicalRequest:
12:39:22 aiosmtpd.1          | Content-Transfer-Encoding: 7bit
12:39:22 enrollment.1        | POST
12:39:22 aiosmtpd.1          | MIME-Version: 1.0
12:39:22 enrollment.1        | /
12:39:22 aiosmtpd.1          | Subject: Enrollment Notification for Introduction to Computer Science
12:39:22 enrollment.1        |
12:39:22 aiosmtpd.1          | From: edwinperaza@csu.fullerton.edu
12:39:22 enrollment.1        | content-type:application/x-amz-json-1.0
12:39:22 aiosmtpd.1          | To: johny@gmail.com
12:39:22 enrollment.1        | host:localhost:5500
12:39:22 aiosmtpd.1          | X-Peer: ('127.0.0.1', 54512)
12:39:22 enrollment.1        | x-amz-date:20231215T203922Z
12:39:22 aiosmtpd.1          |
12:39:22 enrollment.1        | x-amz-target:DynamoDB_20120810.UpdateItem
12:39:22 aiosmtpd.1          | You have been enrolled in Introduction to Computer Science by the registrar
12:39:22 enrollment.1        |
12:39:22 aiosmtpd.1          | ------------ END MESSAGE ------------
```

Task 5: Cache waiting list position

Cache was added to the waitlist endpoint to reduce the amount of traffic since there is no notification service implemented for the times that a student is moved up the waitlist.
Etag was used to check the cache.
Etag generator:

```python
def generate_etag(data):
    data_string = json.dumps(data, sort_keys=True)
    # Create a hash of this string
    return hashlib.md5(data_string.encode()).hexdigest()
```

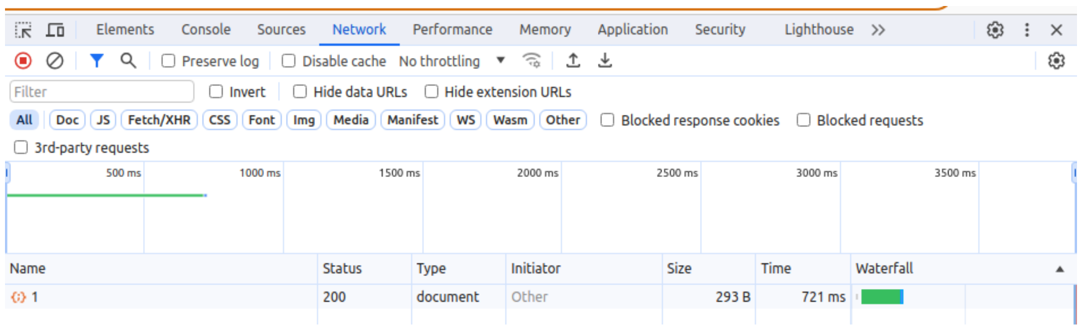
In the endpoint to check the waitlist, the following snippet of code was added:

```python
 # Generate an ETag for waitlist data
    etag = generate_etag(waitlist_data)

    # Obtain current ETag & compare against new etag,
    # return status 304 if same ETag
    if_none_match = request.headers.get("If-None-Match")
    if if_none_match and if_none_match == etag:
        raise HTTPException(
            status_code=status.HTTP_304_NOT_MODIFIED,
            detail="Waitlist for student has not been modified"
        )


# Update ETag if changed
```

```
    response.headers["ETag"] = etag
```

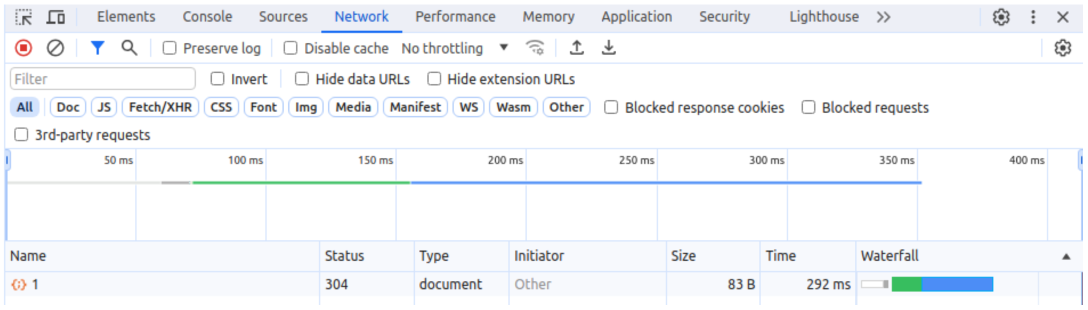
Task 6: Testing

On the first request for the waitlist for student 1 using the endpoint
http://127.0.0.1:5000/waitlist/students/1
The first attempt gives the following information in the network tab:



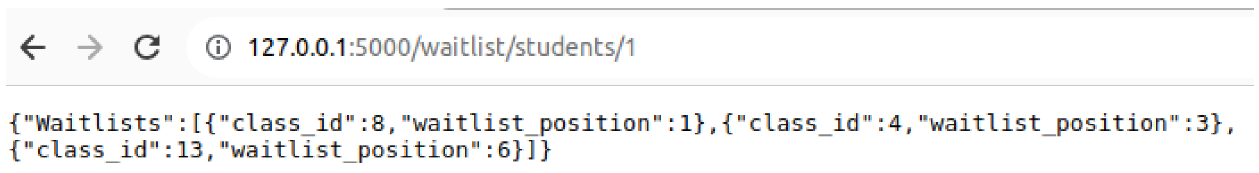
Name	Status	Type	Initiator	Size	Time	Waterfall
1	200	document	Other	293 B	721 ms	

However, if we reattempt to run the endpoint, since nothing has been modified we get a different status code:



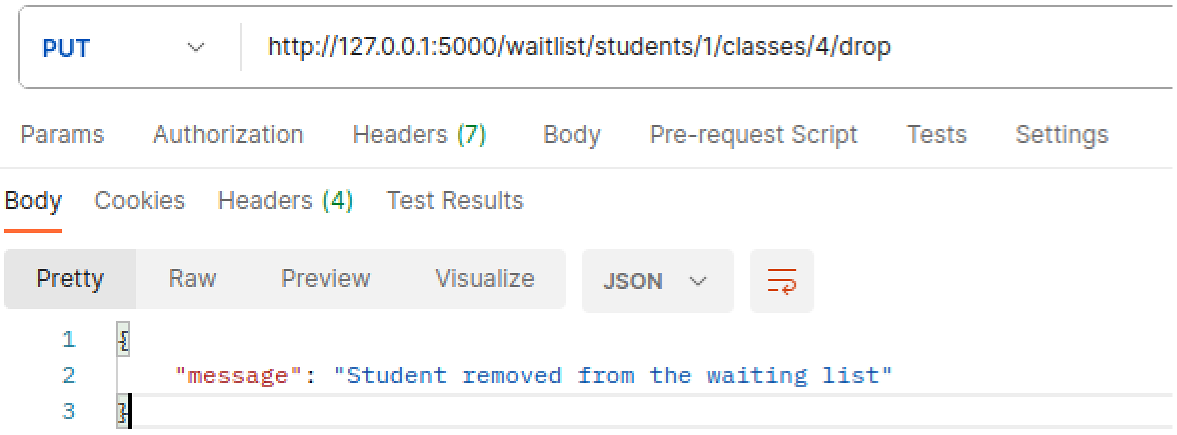
Name	Status	Type	Initiator	Size	Time	Waterfall
1	304	document	Other	83 B	292 ms	

The json response is still visible on the browser:


127.0.0.1:5000/waitlist/students/1

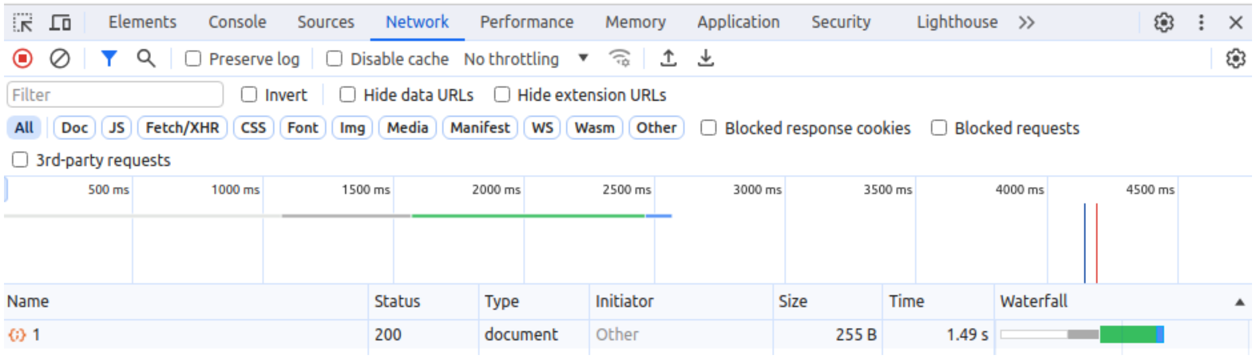
```
{"Waitlists":[{"class_id":8,"waitlist_position":1},{"class_id":4,"waitlist_position":3},
{"class_id":13,"waitlist_position":6}]}
```

Furthermore, if I drop the student from the waitlist for the class with class_id '8', the request will give once again a status code of 200.


PUT http://127.0.0.1:5000/waitlist/students/1/classes/4/drop

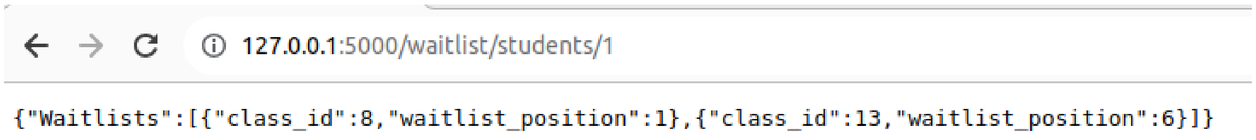
```
{
    "message": "Student removed from the waiting list"
}
```

Making a request once again to http://127.0.0.1:5000/waitlist/students/1



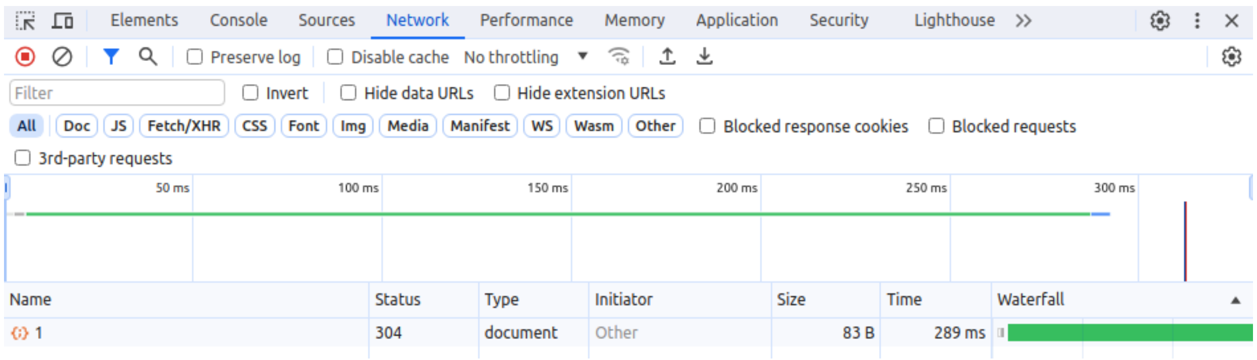
Name	Status	Type	Initiator	Size	Time	Waterfall
1	200	document	Other	255 B	1.49 s	

Now, the json response is different since the student is no longer on the waitlist for class_id ‘8’.


127.0.0.1:5000/waitlist/students/1

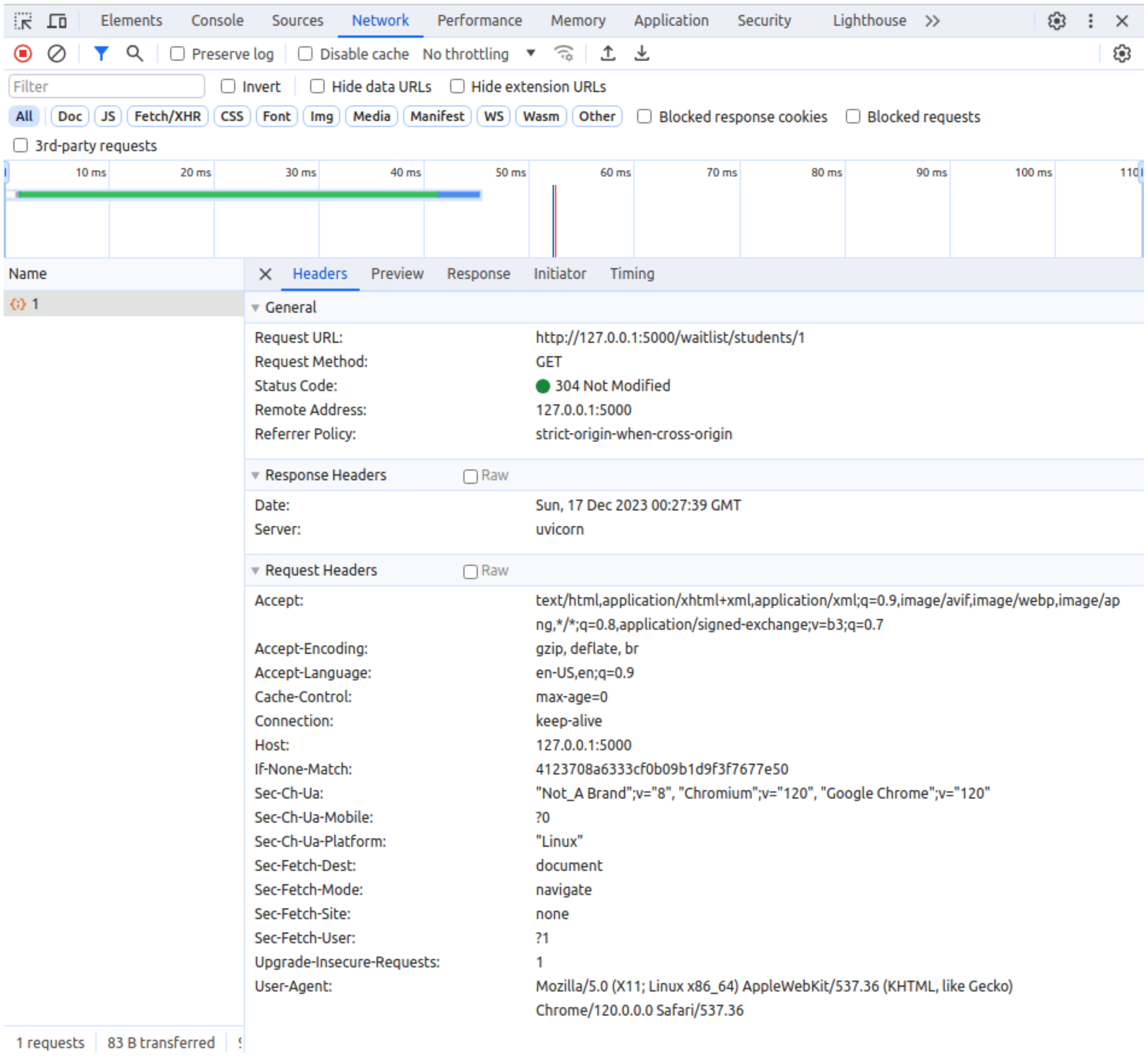
```
{"Waitlists":[{"class_id":8,"waitlist_position":1},{"class_id":13,"waitlist_position":6}]}
```

Once again, when refreshing the page, the new status code is 304.



Name	Status	Type	Initiator	Size	Time	Waterfall
1	304	document	Other	83 B	289 ms	

The If-None-Match header is receiving the etag.



1

General

```
Request URL:        http://127.0.0.1:5000/waitlist/students/1
Request Method:     GET
Status Code:        ● 304 Not Modified
Remote Address:     127.0.0.1:5000
Referrer Policy:    strict-origin-when-cross-origin
```

Response Headers Raw

```
Date:               Sun, 17 Dec 2023 00:27:39 GMT
Server:             uvicorn
```

Request Headers Raw

```
Accept:                     text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,image/ap
                            ng,*/*;q=0.8,application/signed-exchange;v=b3;q=0.7
Accept-Encoding:            gzip, deflate, br
Accept-Language:            en-US,en;q=0.9
Cache-Control:              max-age=0
Connection:                 keep-alive
Host:                       127.0.0.1:5000
If-None-Match:              4123708a6333cf0b09b1d9f3f7677e50
Sec-Ch-Ua:                  "Not_A Brand";v="8", "Chromium";v="120", "Google Chrome";v="120"
Sec-Ch-Ua-Mobile:           ?0
Sec-Ch-Ua-Platform:         "Linux"
Sec-Fetch-Dest:             document
Sec-Fetch-Mode:             navigate
Sec-Fetch-Site:             none
Sec-Fetch-User:             ?1
Upgrade-Insecure-Requests:  1
User-Agent:                 Mozilla/5.0 (X11; Linux x86_64) AppleWebKit/537.36 (KHTML, like Gecko)
                            Chrome/120.0.0.0 Safari/537.36
```

1 requests | 83 B transferred