Project 4

Asynchronous messaging and Conditional GET

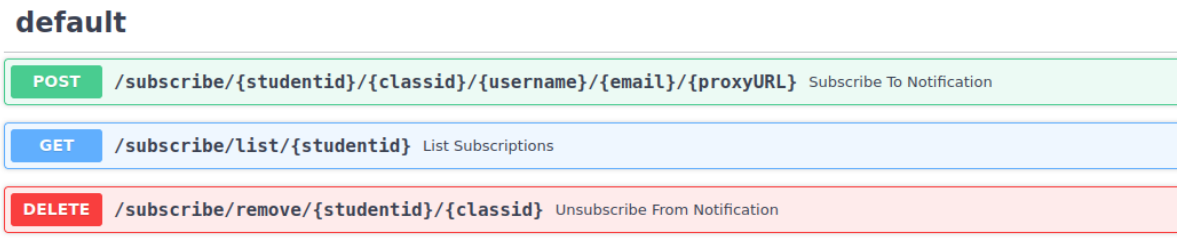
Group 9
Liam Hernandez
David Harboyan
Abhinav Singh
Viditi Vartak
Rishub Goel

CPSC 449-01
Fall 23 Semester

Enrollment Notification Service

Notification Endpoints

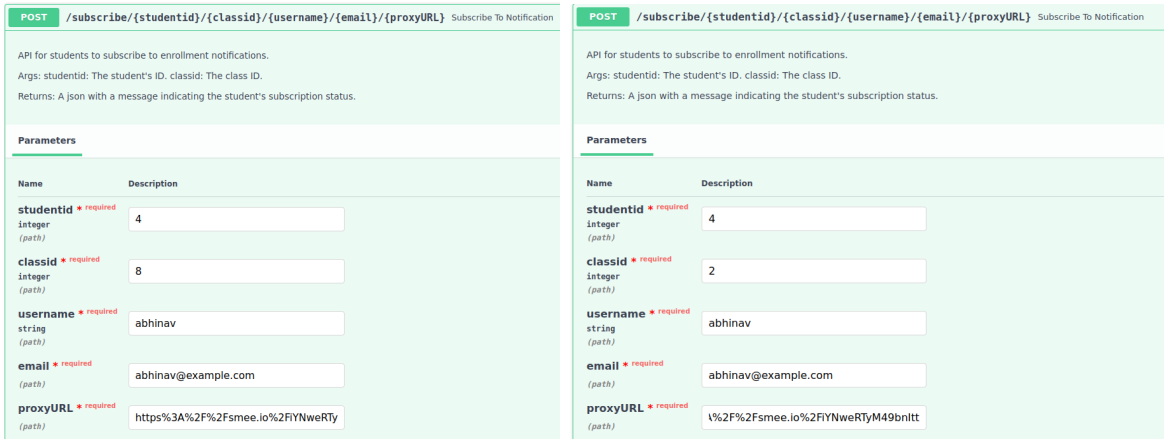
Context: Created three endpoints for managing enrollment notifications - subscribe to class enrollment notifications, list all notifications, and one to unsubscribe from class enrollment notifications.



Note - The data for these subscriptions are being stored in Redis and the key is "subscription:<student-id>". This key has a json with different <class-id> as the keys which student is subscribed to. Further these <class-id>'s has another dictionary with two keys "email" and "proxy" to store the emails and proxyURL for webhook forwarding. To add a subscription simply a new <class-id> is added in the already existing json.

Testing this service :

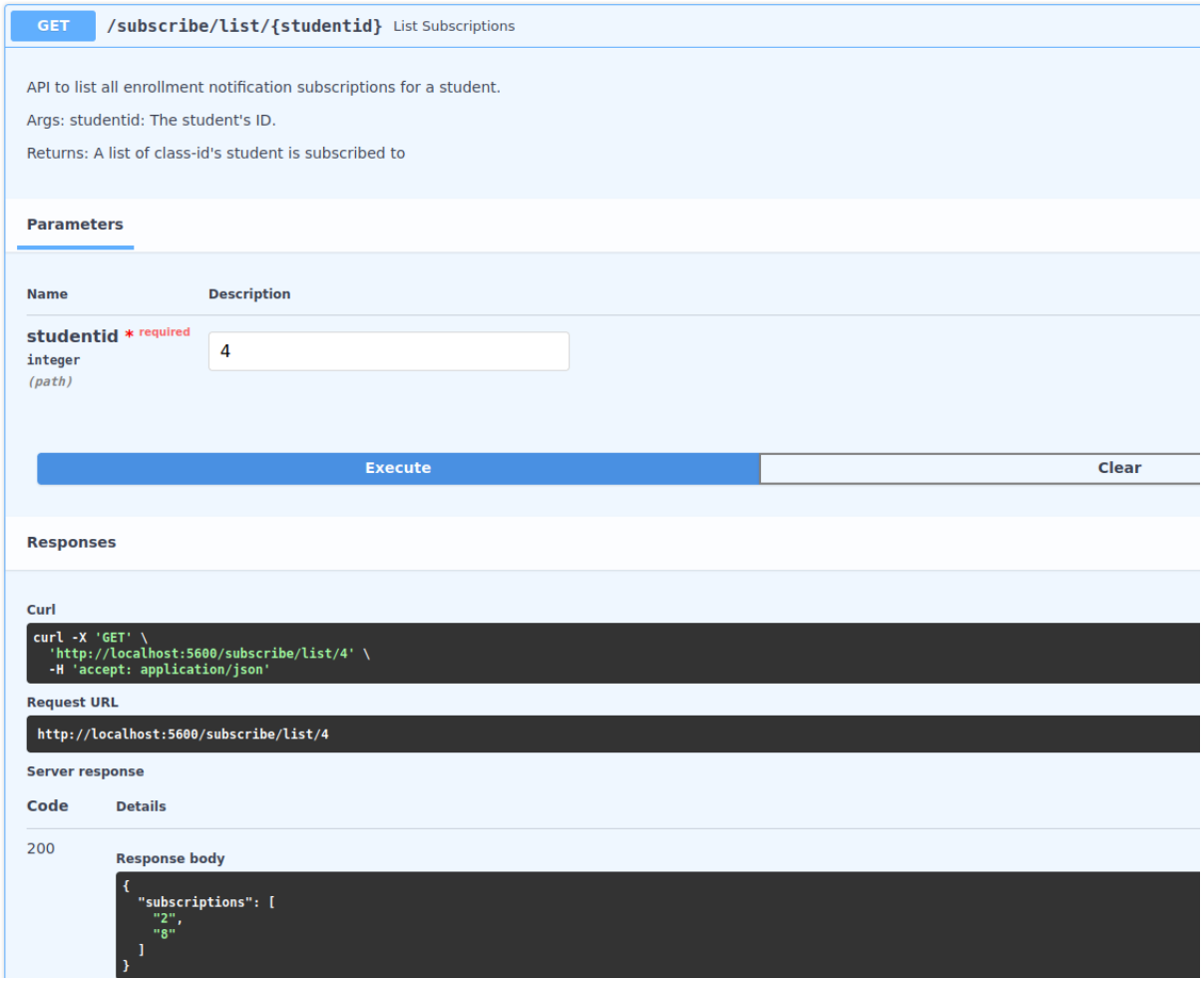
Subscribing to receive enrollment notification for class id - 2 & 8 for same student



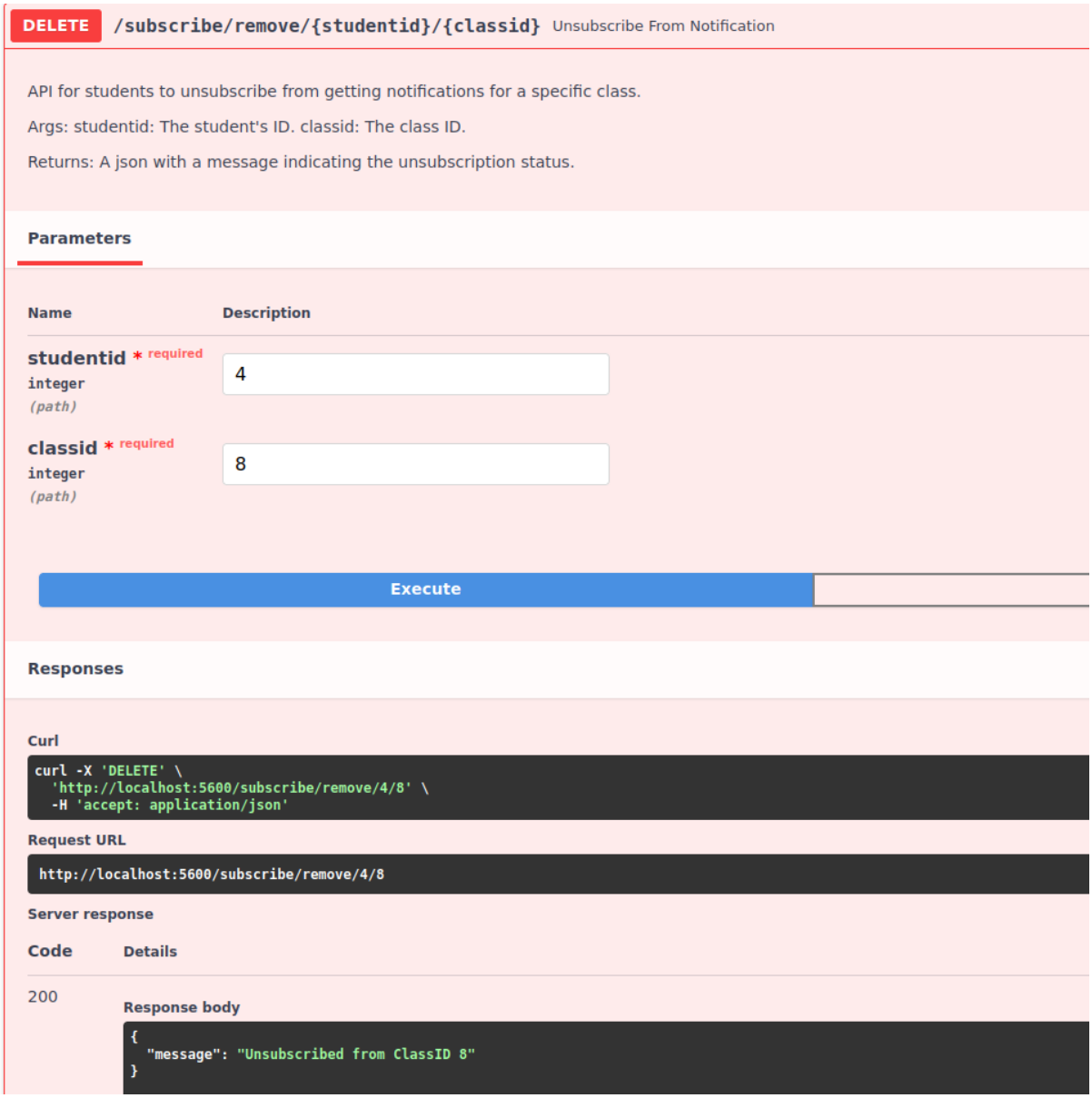
Data in Redis

```
127.0.0.1:6379> get subscription:4
"{\"2\": {\"email\": \"abhinav@example.com\", \"proxy\": \"https%3A%2F%2Fsmee.io%2FiYNweRTyM49bnItt\"}, \"8\": {\"email\": \"abhinav@example
.com\", \"proxy\": \"https%3A%2F%2Fsmee.io%2FiYNweRTyM49bnItt\"}}"
127.0.0.1:6379>
```

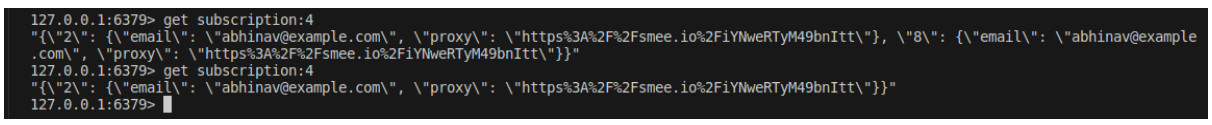
Listing classes to which student is subscribed to: we can see class id's 2 & 8 that we added before



Deleting a class subscription



Checked Redis id data really got removed: Yes, it did.



Enrollment notifications via email and webhook

Context: Added logic to send notification in the drop classes API where automatic enrollment is happening. It sends the message, student Email/proxyURL to RabbitMQ publisher along with the class id that the student is enrolled to.
There are two consumers at play here listening to any new messages and then sending email/webhook accordingly from their callbacks. For sending email, we have used **smtplib**, and the mail server is **aiosmtpd**. And for webhooks, we have used smee.io to generate proxyURLs and HTTPX to POST to the callback URL.

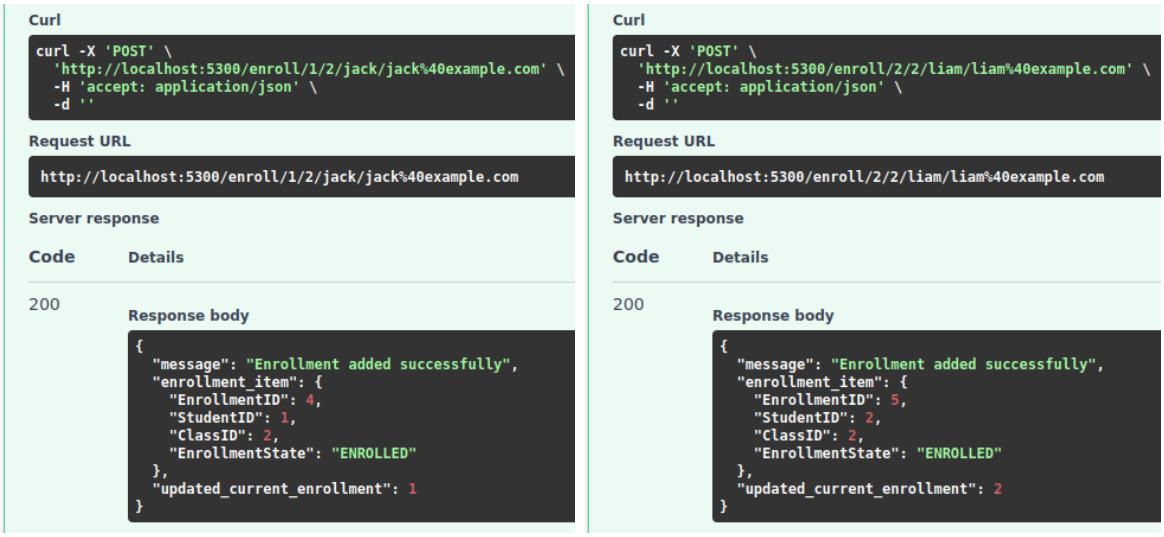
```
aiosmtpd: python -m aiosmtpd -n -d
email_consumer: python -m notification.email_consumer
webhook_consumer: python -m notification.webhook_consumer
```

```
21:24:54 aiosmtpd.1           | started with pid 26932
21:24:54 email_consumer.1     | started with pid 26934
21:24:54 webhook_consumer.1   | started with pid 26935
```

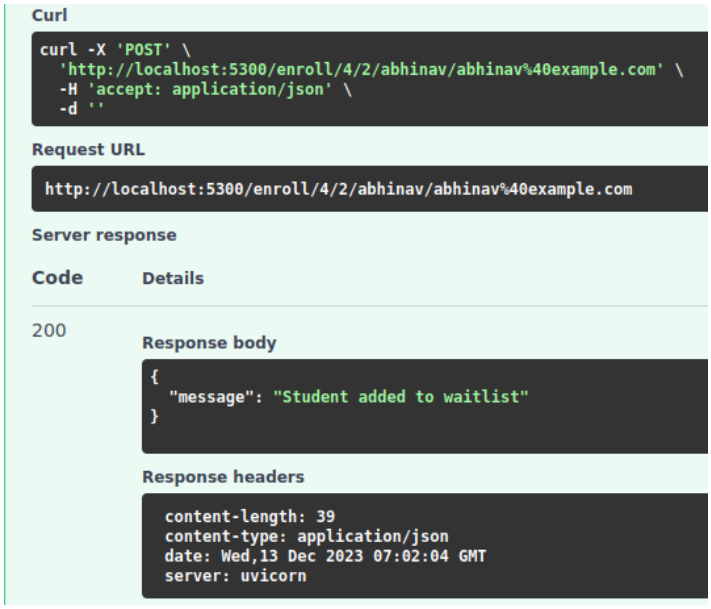
These processes for aiosmtpd mail server and the consumers for emails and webhooks starts with the fastapi server and is active throughout.

Testing email and webhook notifications:

Enrolled two students to a class (id - 2) whose max enrollment is set to 2, so if we add more than two students, they will go on the waitlist.

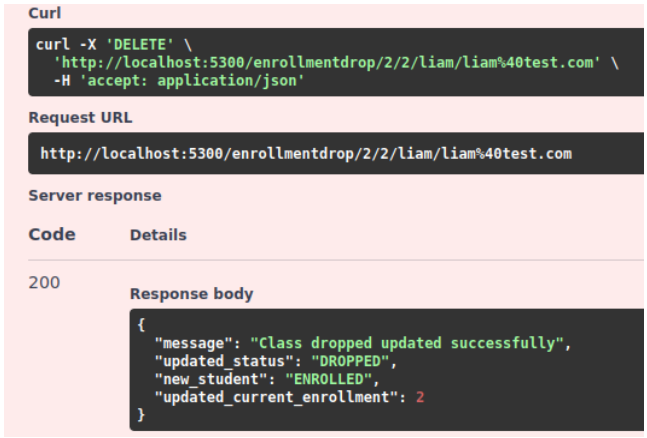


Adding another student:



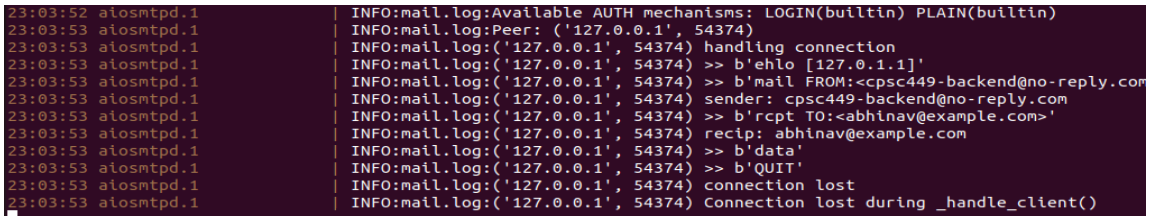
Now dropping the student with id 1 who is enrolled in class (id - 2), this will lead to the automatic enrollment of student with id 4, and since he is subscribed to receive email and webhook notification for class 2, he should receive both notifications.

Dropped student with id 2, so now student with id 4 got enrolled.

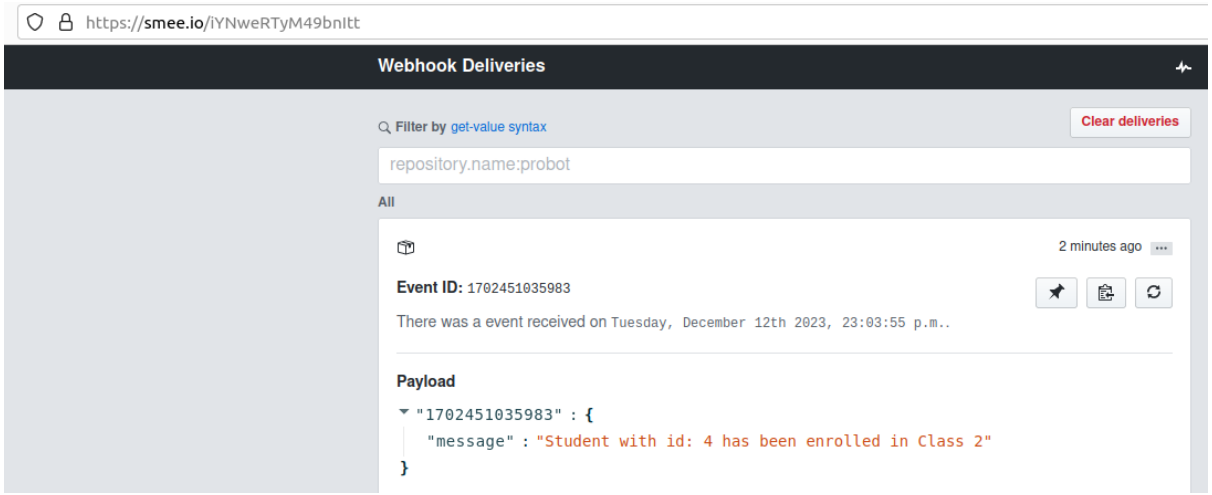


As soon as the student is enrolled from the waitlist, the student email and the class id is sent to the RabbitMQ publisher and later consumed by the different consumers. In the callback of these consumers logic is written to send email and webhook respectively.

Can see that the email got sent and is received by aiosmtpd in the terminal logs



Webhook is also forwarded :



Cache waiting list positions / Conditional GET

Logic for implementing HTTP Conditional Requests

Context: In order to reduce the amount of traffic to the "view current position" endpoint, we needed to implement HTTP Conditional Requests. To do so, we needed to keep track in Redis of the Last-Modified time whenever we are adding or removing from the waitlist. When we GET from the waitlist, we can check Redis if the list has been added to or removed from since the last GET request, and if not we can set the Last-Modified from the GET request and continue as before returning with status 200. From the GET, if the request's If-Modified-Since is greater than or equal to the Last-Modified in the Redis we can respond with status 304 and avoid looking up the user within the waitlist.

Additions to the "view current position" endpoint:

```
last_modified = r.get(f"last-modified:{classid}")
if last_modified:
    last_modified = last_modified.decode()
    response.headers["Last-Modified"] = last_modified
    if 'If-Modified-Since'.lower() in request.headers:
        if_modified_since = request.headers['If-Modified-Since']
        print(f"{if_modified_since} >= {last_modified}")
        if if_modified_since >= last_modified:
            response.headers["Last-Modified"] = last_modified
            response.status_code = status.HTTP_304_NOT_MODIFIED
            return
else:
    now_gmt = time.gmtime()
    r.set(f"last-modified:{classid}", time.strftime('%a, %d %b %Y %H:%M:%S GMT', now_gmt))
    response.headers["Last-Modified"] = time.strftime('%a, %d %b %Y %H:%M:%S GMT', now_gmt)
```

Kept track of the Last-Modified time in the endpoints that added or removed from the waitlist:

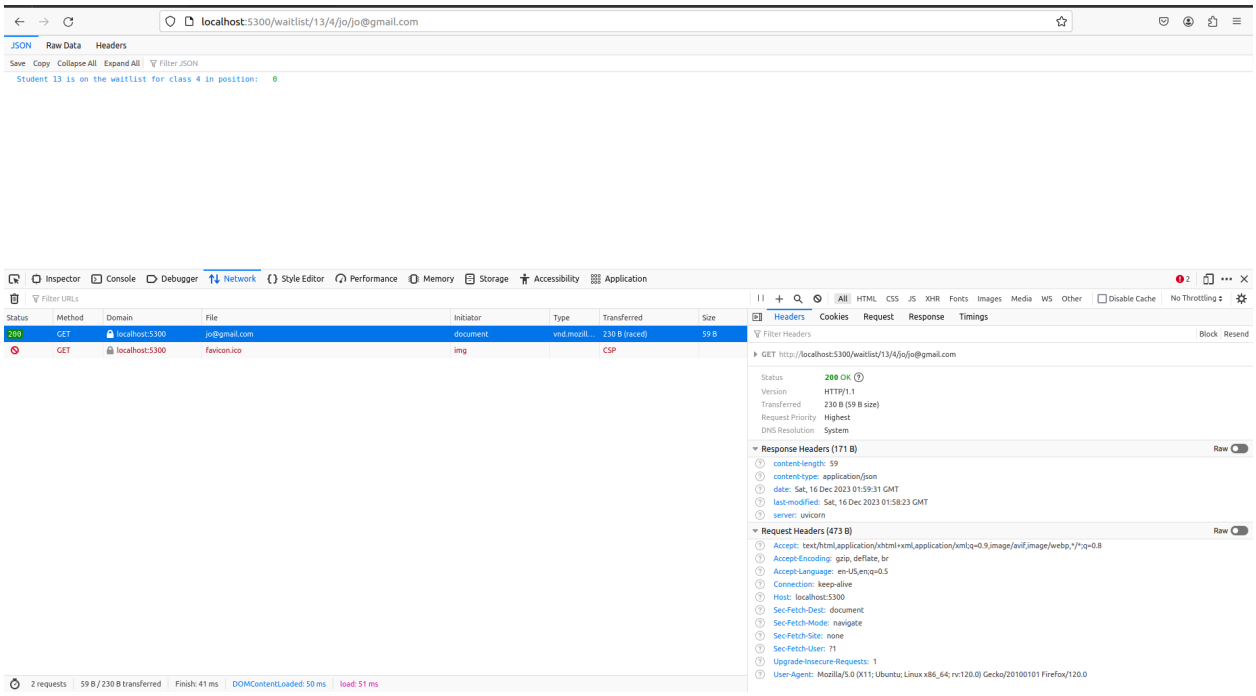
```
r.set(f"last-modified:{class_id}", time.strftime('%a, %d %b %Y %H:%M:%S GMT', time.gmtime()))
```

Note: We chose to use the "Last-Modified: / If-Modified-Since:" headers, but could have also implemented HTTP Conditional Requests with the "ETag: / If-None-Match:" headers.

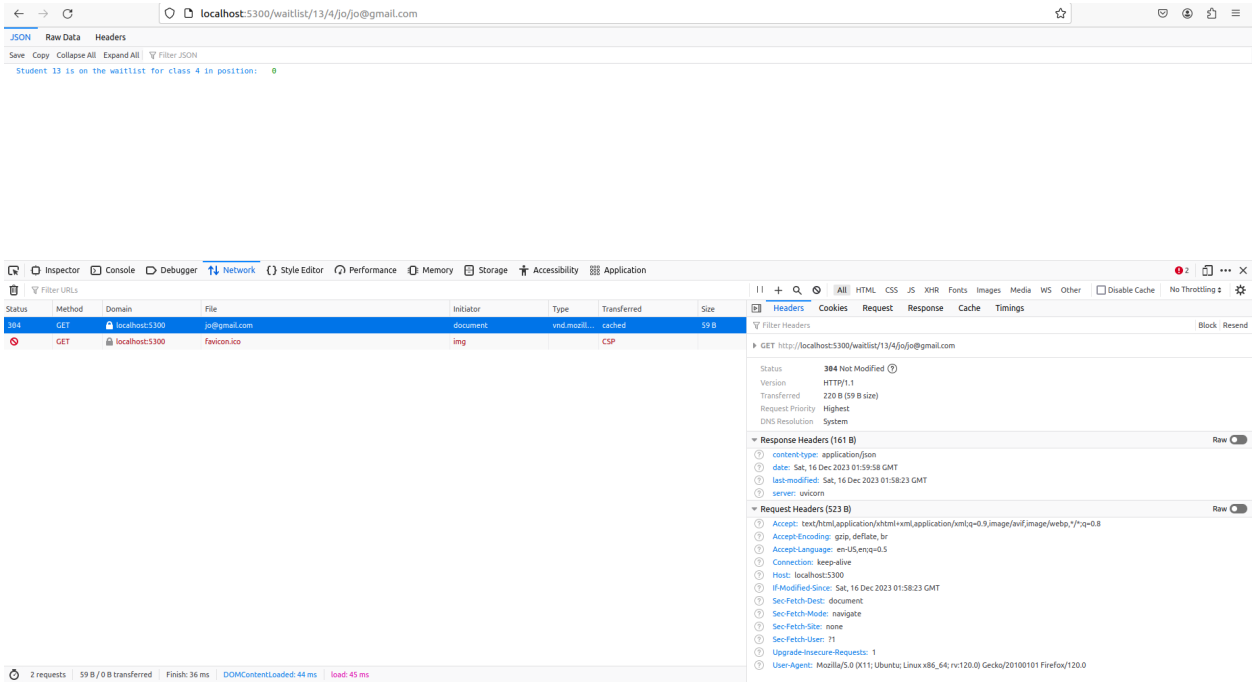
Testing that conditional requests function correctly:

The setup for testing was registering two students: David and Jo. Then log in as David and enroll in the class with ID of 4 (which was temporarily modified to have a class size of 1). After that, log in as Jo and try to enroll in the same class, putting Jo on the waitlist for the class with ID of 4.

Heading to our browser, we opened the Network tab and hit the “view current position” as Jo. The url was http://localhost:5300/waitlist/13/4/jo/jo@gmail.com. We notice that Jo’s position was returned and in the Network tab, we can see that we got a status code of 200 OK which we expected.



After clicking the refresh button, we can see that the returned value is the same and in the network tab, the status code is 304 Not Modified. By taking a closer look at the headers, If-Modified-Since >= Last-Modified which was the condition we were checking for.



Based on the results from testing, we can conclude that the conditional requests function correctly.