

# *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.

### default

POST	/subscribe/{studentid}/{classid}/{username}/{email}/{proxyURL}	Subscribe To Notification
GET	/subscribe/list/{studentid}	List Subscriptions
DELETE	/subscribe/remove/{studentid}/{classid}	Unsubscribe From Notification

**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

<p><b>POST</b> /subscribe/{studentid}/{classid}/{username}/{email}/{proxyURL} Subscribe To Notification</p> <p>API for students to subscribe to enrollment notifications.</p> <p>Args: studentid: The student's ID. classid: The class ID.</p> <p>Returns: A json with a message indicating the student's subscription status.</p> <p><b>Parameters</b></p> <table><thead><tr><th>Name</th><th>Description</th></tr></thead><tbody><tr><td>studentid * required integer (path)</td><td>4</td></tr><tr><td>classid * required integer (path)</td><td>8</td></tr><tr><td>username * required string (path)</td><td>abhinav</td></tr><tr><td>email * required (path)</td><td>abhinav@example.com</td></tr><tr><td>proxyURL * required (path)</td><td>https%3A%2F%2Fsmee.io%2FYNwerty</td></tr></tbody></table>	Name	Description	studentid * required integer (path)	4	classid * required integer (path)	8	username * required string (path)	abhinav	email * required (path)	abhinav@example.com	proxyURL * required (path)	https%3A%2F%2Fsmee.io%2FYNwerty	<p><b>POST</b> /subscribe/{studentid}/{classid}/{username}/{email}/{proxyURL} Subscribe To Notification</p> <p>API for students to subscribe to enrollment notifications.</p> <p>Args: studentid: The student's ID. classid: The class ID.</p> <p>Returns: A json with a message indicating the student's subscription status.</p> <p><b>Parameters</b></p> <table><thead><tr><th>Name</th><th>Description</th></tr></thead><tbody><tr><td>studentid * required integer (path)</td><td>4</td></tr><tr><td>classid * required integer (path)</td><td>2</td></tr><tr><td>username * required string (path)</td><td>abhinav</td></tr><tr><td>email * required (path)</td><td>abhinav@example.com</td></tr><tr><td>proxyURL * required (path)</td><td>%2F%2Fsmee.io%2FYNwertyM49bnltt</td></tr></tbody></table>	Name	Description	studentid * required integer (path)	4	classid * required integer (path)	2	username * required string (path)	abhinav	email * required (path)	abhinav@example.com	proxyURL * required (path)	%2F%2Fsmee.io%2FYNwertyM49bnltt
Name	Description																								
studentid * required integer (path)	4																								
classid * required integer (path)	8																								
username * required string (path)	abhinav																								
email * required (path)	abhinav@example.com																								
proxyURL * required (path)	https%3A%2F%2Fsmee.io%2FYNwerty																								
Name	Description																								
studentid * required integer (path)	4																								
classid * required integer (path)	2																								
username * required string (path)	abhinav																								
email * required (path)	abhinav@example.com																								
proxyURL * required (path)	%2F%2Fsmee.io%2FYNwertyM49bnltt																								

## 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

**GET** /subscribe/list/{studentid} List Subscriptions

API to list all enrollment notification subscriptions for a student.

Args: studentid: The student's ID.

Returns: A list of class-id's student is subscribed to

**Parameters**

Name	Description
<b>studentid</b> * required integer (path)	<input type="text" value="4"/>

Execute

Clear

**Responses**

**Curl**

```
curl -X 'GET' \
'http://localhost:5600/subscribe/list/4' \
-H 'accept: application/json'
```

**Request URL**

```
http://localhost:5600/subscribe/list/4
```

**Server response**

Code	Details
200	<div><b>Response body</b><pre>{   "subscriptions": [     "2",     "8"   ] }</pre></div>

## Deleting a class subscription

**DELETE** /subscribe/remove/{studentid}/{classid} Unsubscribe From Notification

API for students to unsubscribe from getting notifications for a specific class.

Args: studentid: The student's ID. classid: The class ID.

Returns: A json with a message indicating the unsubscription status.

### Parameters

Name	Description
<b>studentid</b> * required integer (path)	<input type="text" value="4"/>
<b>classid</b> * required integer (path)	<input type="text" value="8"/>

Execute

### Responses

Curl

```
curl -X 'DELETE' \
'http://localhost:5600/subscribe/remove/4/8' \
-H 'accept: application/json'
```

Request URL

```
http://localhost:5600/subscribe/remove/4/8
```

Server response

Code	Details
200	<p>Response body</p> <pre>{   "message": "Unsubscribed from ClassID 8" }</pre>

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

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

## 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.

Curl		Curl	
<pre>curl -X 'POST' \   'http://localhost:5300/enroll/1/2/jack/jack%40example.com' \   -H 'accept: application/json' \   -d ''</pre>		<pre>curl -X 'POST' \   'http://localhost:5300/enroll/2/2/liam/liam%40example.com' \   -H 'accept: application/json' \   -d ''</pre>	
Request URL		Request URL	
<code>http://localhost:5300/enroll/1/2/jack/jack%40example.com</code>		<code>http://localhost:5300/enroll/2/2/liam/liam%40example.com</code>	
Server response		Server response	
Code	Details	Code	Details
200	<pre>Response body {   "message": "Enrollment added successfully",   "enrollment_item": {     "EnrollmentID": 4,     "StudentID": 1,     "ClassID": 2,     "EnrollmentState": "ENROLLED"   },   "updated_current_enrollment": 1 }</pre>	200	<pre>Response body {   "message": "Enrollment added successfully",   "enrollment_item": {     "EnrollmentID": 5,     "StudentID": 2,     "ClassID": 2,     "EnrollmentState": "ENROLLED"   },   "updated_current_enrollment": 2 }</pre>

Adding another student:

**Curl**

```
curl -X 'POST' \
  'http://localhost:5300/enroll/4/2/abhinav/abhinav%40example.com' \
  -H 'accept: application/json' \
  -d ''
```

**Request URL**

```
http://localhost:5300/enroll/4/2/abhinav/abhinav%40example.com
```

**Server response**

Code	Details
200	<p><b>Response body</b></p> <pre>{   "message": "Student added to waitlist" }</pre> <p><b>Response headers</b></p> <pre>content-length: 39 content-type: application/json date: Wed, 13 Dec 2023 07:02:04 GMT server: uvicorn</pre>

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.

**Curl**

```
curl -X 'DELETE' \
  'http://localhost:5300/enrollmentdrop/2/2/liam/liam%40test.com' \
  -H 'accept: application/json'
```

**Request URL**

```
http://localhost:5300/enrollmentdrop/2/2/liam/liam%40test.com
```

**Server response**

Code	Details
200	<p><b>Response body</b></p> <pre>{   "message": "Class dropped updated successfully",   "updated_status": "DROPPED",   "new_student": "ENROLLED",   "updated_current_enrollment": 2 }</pre>

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

```
23:03:52 aiosmtpd.1 | INFO:mail.log:Available AUTH mechanisms: LOGIN(builtin) PLAIN(builtin)
23:03:53 aiosmtpd.1 | INFO:mail.log:Peer: ('127.0.0.1', 54374)
23:03:53 aiosmtpd.1 | INFO:mail.log:('127.0.0.1', 54374) handling connection
23:03:53 aiosmtpd.1 | INFO:mail.log:('127.0.0.1', 54374) >> b'ehlo [127.0.1.1]'
23:03:53 aiosmtpd.1 | INFO:mail.log:('127.0.0.1', 54374) >> b'mail FROM:<cpsc449-backend@no-reply.com>'
23:03:53 aiosmtpd.1 | INFO:mail.log:('127.0.0.1', 54374) sender: cpsc449-backend@no-reply.com
23:03:53 aiosmtpd.1 | INFO:mail.log:('127.0.0.1', 54374) >> b'rcpt TO:<abhinav@example.com>'
23:03:53 aiosmtpd.1 | INFO:mail.log:('127.0.0.1', 54374) recip: abhinav@example.com
23:03:53 aiosmtpd.1 | INFO:mail.log:('127.0.0.1', 54374) >> b'data'
23:03:53 aiosmtpd.1 | INFO:mail.log:('127.0.0.1', 54374) >> b'QUIT'
23:03:53 aiosmtpd.1 | INFO:mail.log:('127.0.0.1', 54374) connection lost
23:03:53 aiosmtpd.1 | INFO:mail.log:('127.0.0.1', 54374) Connection lost during _handle_client()
```

Webhook is also forwarded :

https://smee.io/iYNweRTyM49bnltt

### Webhook Deliveries

Filter by [get-value syntax](#) [Clear deliveries](#)

repository.name:probot

All

2 minutes ago

**Event ID:** 1702451035983

There was a event received on Tuesday, December 12th 2023, 23:03:55 p.m..

**Payload**

```
{
  "1702451035983" : {
    "message" : "Student with id: 4 has been enrolled in Class 2"
  }
}
```

## 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()))
```

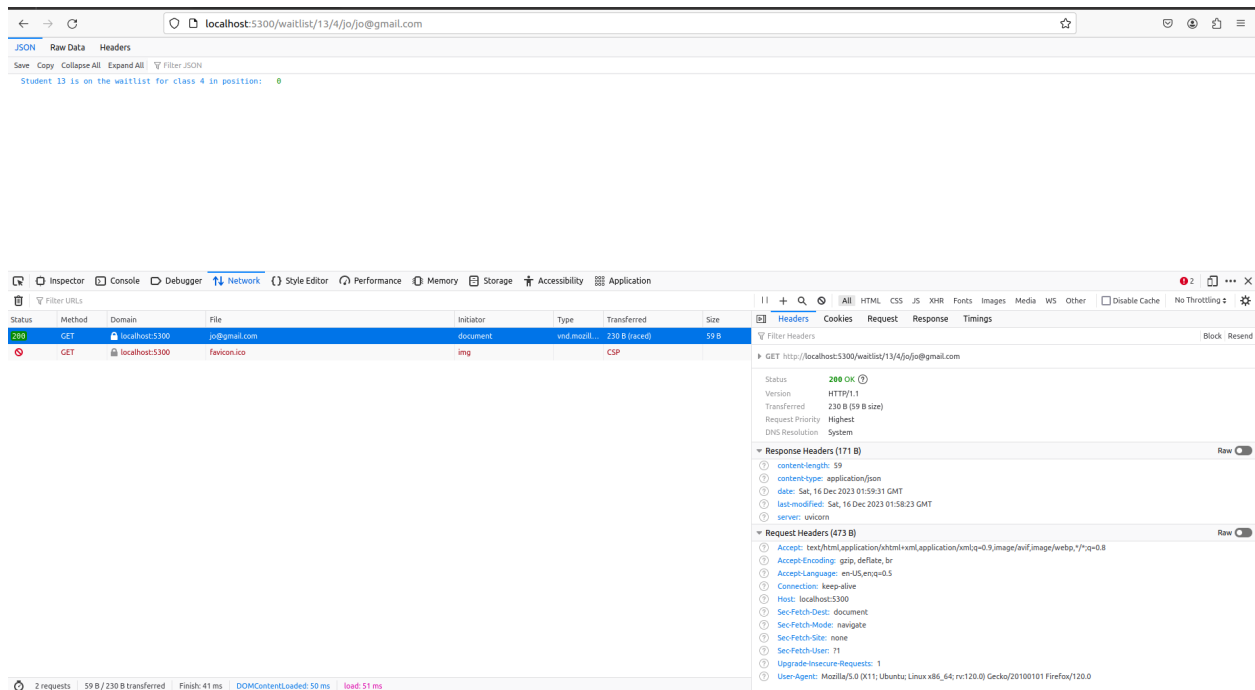
**Note:** We chose to use the “Last-Modified: / If-Modified-Since:” headers, but could have also implemented HTTP Conditional Requests with the “ETag: / If-None-Match:” headers.



## 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.

The screenshot shows a web browser window with the address bar displaying `localhost:5300/waitlist/13/4/jo@jo@gmail.com`. The browser's developer tools are open, showing the Network tab. A table of network requests is visible, with the first request highlighted:

Status	Method	Domain	File	Initiator	Type	Transferred	Size
304	GET	localhost:5300	jo@gmail.com	document	vnd.mozilla:cached	59 B	59 B

The right-hand pane of the Network tab shows the details for the selected request, including the status (304 Not Modified), version (HTTP/1.1), and various headers. The Request Headers section is expanded, showing the following details:

- Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/avif,image/webp,\*/\*;q=0.8
- Accept-Encoding: gzip, deflate, br
- Accept-Language: en-US,en;q=0.5
- Connection: keep-alive
- Host: localhost:5300
- If-Modified-Since: Sat, 16 Dec 2023 01:58:23 GMT
- Sec-Fetch-Dest: document
- Sec-Fetch-Mode: navigate
- Sec-Fetch-Site: none
- Sec-Fetch-User: 11
- Upgrade-Insecure-Requests: 1
- User-Agent: Mozilla/5.0 (X11; Ubuntu; Linux x86\_64; rv:120.0) Gecko/20100101 Firefox/120.0

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