

## Summary & Introduction

StudyLink is an app designed to help students easily make study buddies with peers from their university classes. By offering a way to find fellow classmates and reach out, StudyLink reduces the anxiety of asking for help in large lectures and creates a more personal and approachable environment for students who want to practice, collaborate, or seek help in their courses. Using the Canvas API, StudyLink automates the process of pulling student enrollment data and course information, making it effortless for students to discover peers in their classes for collaboration and study sessions.

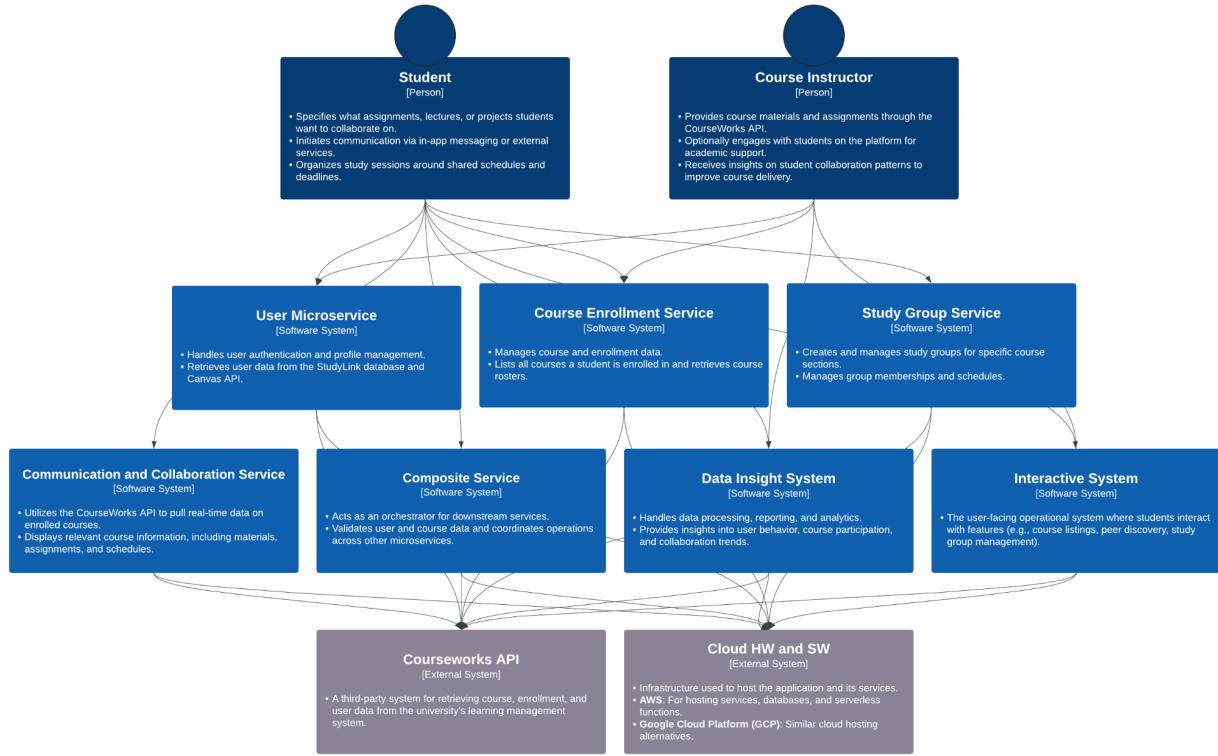
Some examples of the tasks StudyLink addresses are:

- Providing all of the courses in which a student is actively enrolled
- Students can see a list of classmates who are also using the app.
- Students can manage study groups by creating, joining, deleting, or editing details
- Facilitating communication between students via in-app messaging—between individual students or study groups
- A calendar that integrates key deadlines and events to help students manage their studying sessions.

StudyLink is a cloud-based app that delivers a significant improvement in how students connect for academic collaboration, making it easier to form study groups and engage with peers. The value to students is increased access to study support and reduced anxiety in large, impersonal lecture settings. The benefit to universities is that StudyLink fosters a more supported, connected, and collaborative student body.

# Solution Overview

## Domains



## Overall Solution

StudyLink is a cloud-based platform designed to foster academic collaboration by connecting students within their university classes. The app addresses the challenges students face in large, impersonal lecture settings by providing a seamless way to discover peers for study groups, collaborate on assignments, and exchange ideas. StudyLink integrates with the university's learning management system (Canvas API) to automate data retrieval, ensuring minimal manual effort for users. Through its modular design, StudyLink offers robust support for user management, course enrollment, study group formation, communication, and scheduling.

## Major Systems and Subsystems

### User Service

Students log in using their university credentials and the service retrieves their personal information (e.g., name, email) from the Canvas API. Students can update their profile settings directly in StudyLink.

- Handles user authentication and manages user profiles.
- Retrieves user data from Canvas API and StudyLink's database.
- Updates user-specific settings and preferences.

### Course Enrollment Service

Students can view a list of all their enrolled courses, fetched from the Canvas API. Within a selected course, the service provides a filtered list of classmates using StudyLink.

- Automates the retrieval of course and enrollment data.
- Fetches course lists for users from Canvas.
- Lists classmates (and their IDs) enrolled in specific courses.

### Study Group Service

Students can create a study group for a course if none exists or join an existing group.

- Facilitates the creation and management of study groups.
- Create study groups for courses if none exist.
- Manages group memberships and meeting details.

### Communication and Collaboration Service

Students can send messages to classmates, either one-on-one or in study groups, to discuss specific assignments, share resources, and organize group study sessions.

- Enables seamless interaction among users.
- Provide in-app messaging and group chat functionality.
- Allows notifications through email or external messaging platforms.

### Composite Service

When students create a study group, join a discussion, or send a message, the Composite Service coordinates these operations by interacting with relevant downstream services (e.g., verifying enrollment and updating group memberships).

- Acts as an orchestrator for downstream microservices.
- Validates user and course data for study group creation.
- Handles asynchronous requests to improve performance.

### Data Insight System

Students can view their participation metrics, such as study groups created or messages exchanged.

- Provides analytics and insights into user behavior and engagement.
- Analyzes student participation in study groups.
- Identifies collaboration trends to improve the app experience.

### Interactive System

Students interact with all app features (e.g., viewing courses, joining groups, messaging) through a user-friendly interface. They can sync deadlines and organize study sessions directly through the interface.

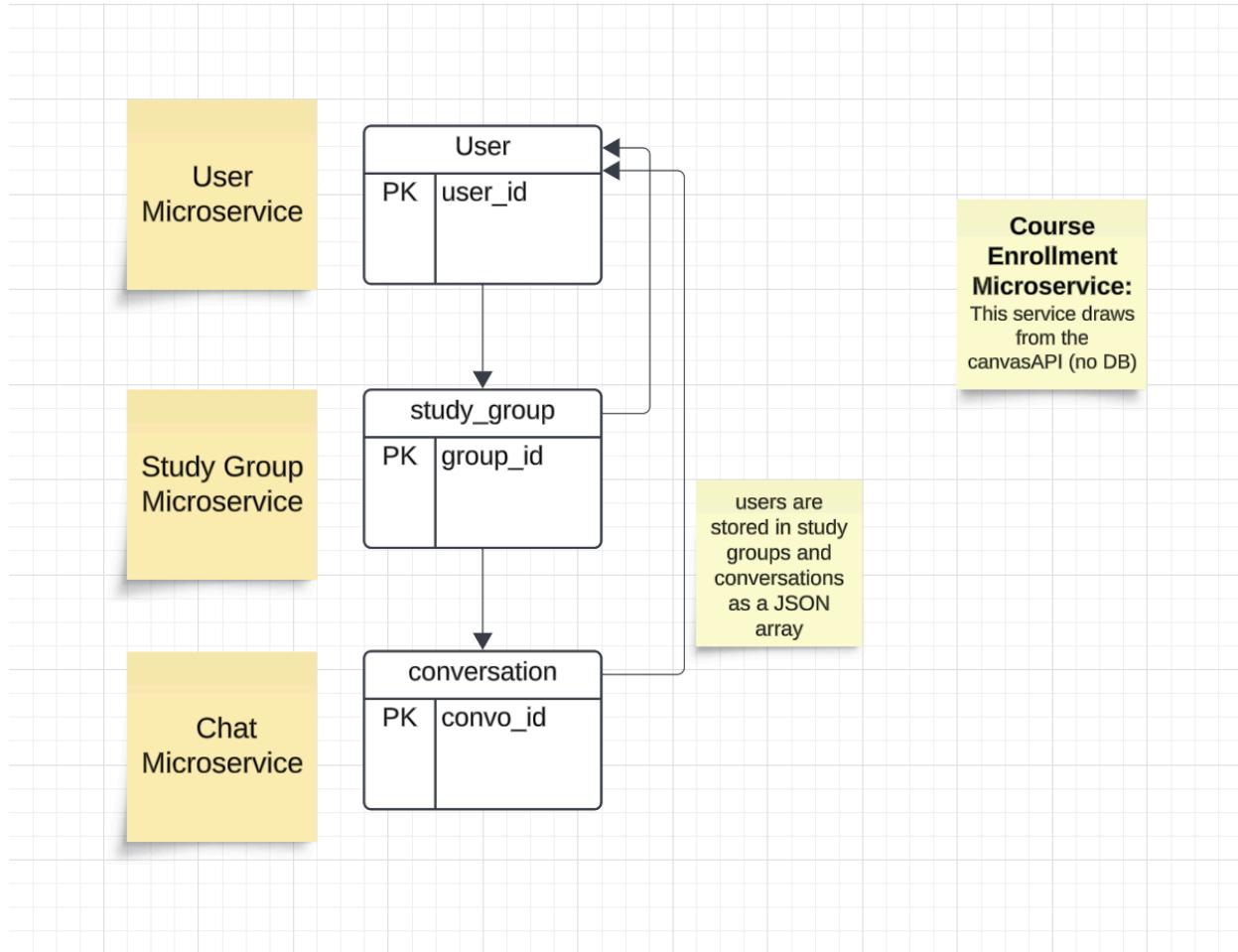
- Front-end system that provides the primary user interface.
- Enables students to view courses, find classmates, and manage study groups.
- Displays schedules, deadlines, and messaging functionality.

## External Systems

- Canvas API: Integrates with the university's learning management system to fetch course and enrollment data.
- Cloud Infrastructure: Hosts StudyLink's services, ensuring scalability and availability.

## Resource Model

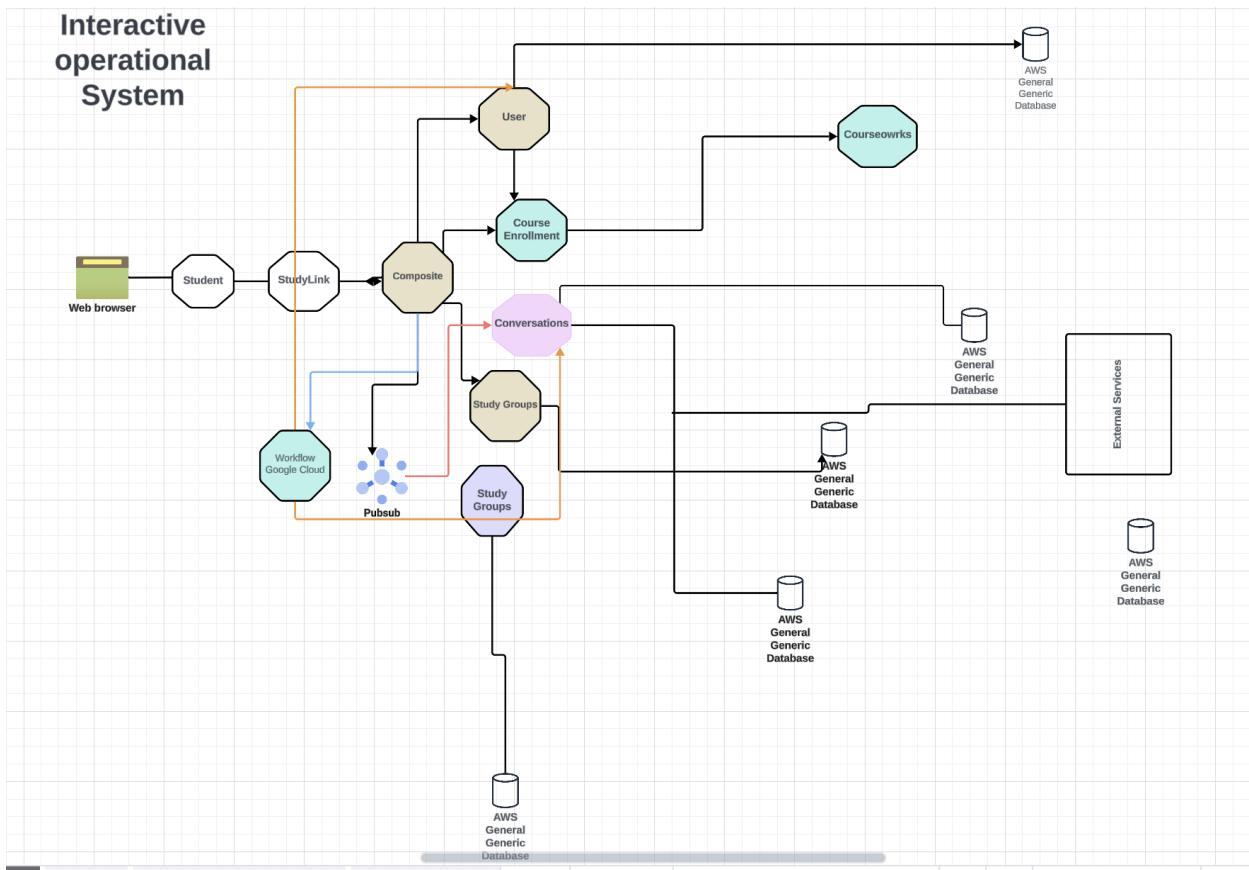
[Resource Model Link](#)



## Description

- User: CU student in courseworks that has signed up
- Study group: study groups for students in the same course
- Chat: private and group conversations

## Interactive and Operations System



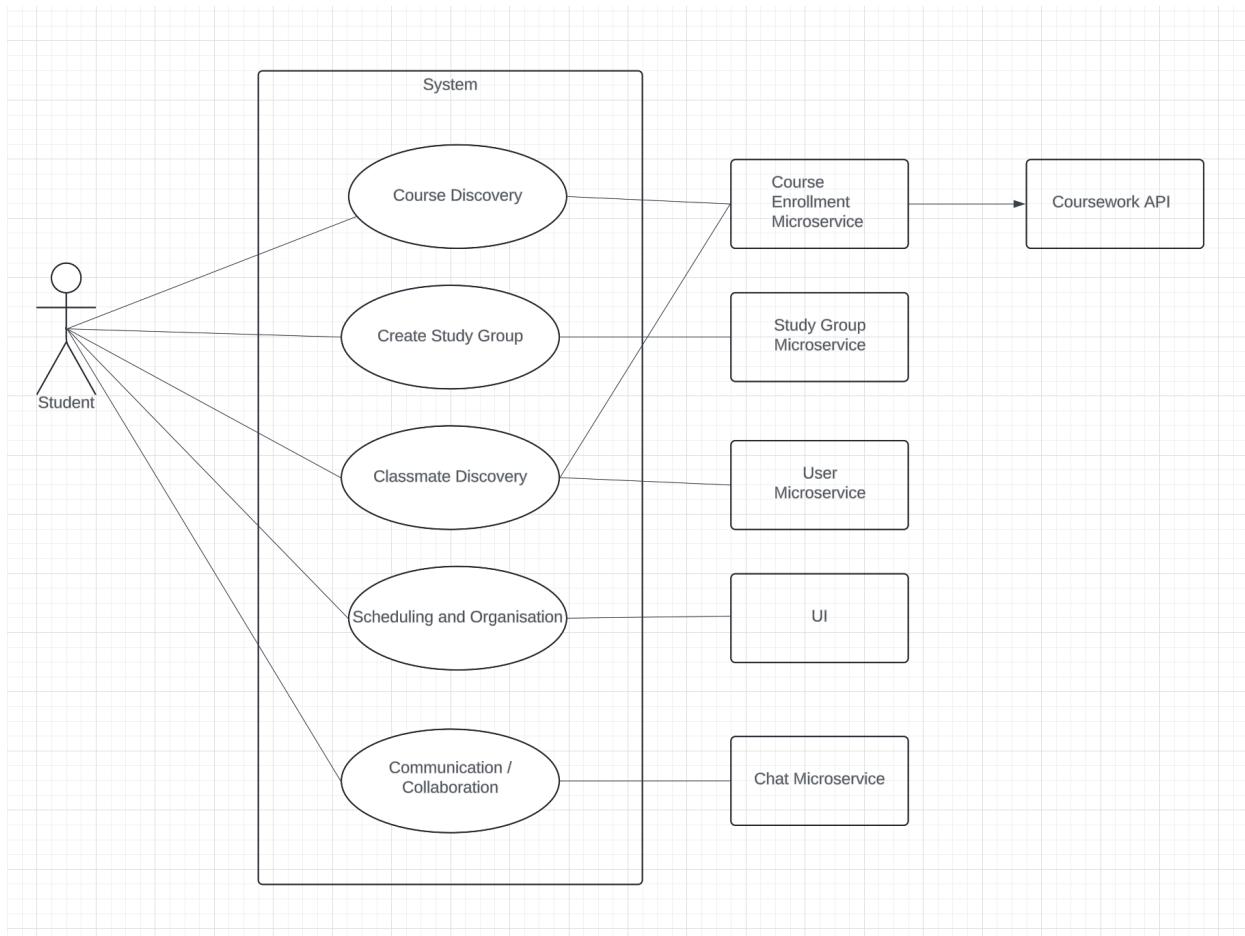
### System Overview:

Begin by logging in through the browser user interface to access our application, which offers a variety of functionalities:

- 1) Course Enrollment: Retrieve course details and enrollment data from the Courseworks API. Store the info in the application database for easy access and management.
- 2) User: Access and manage the student's personal information, including name, ID, and contact details. Obtain information from the Canvas API and the application database.
- 3) Conversations/Messaging Service: Allow students to create conversations within a group chat or direct message format.
- 4) Student Groups: Help students of the same course to create study groups and study together

## Use Case Diagram

[Diagram Link](#)



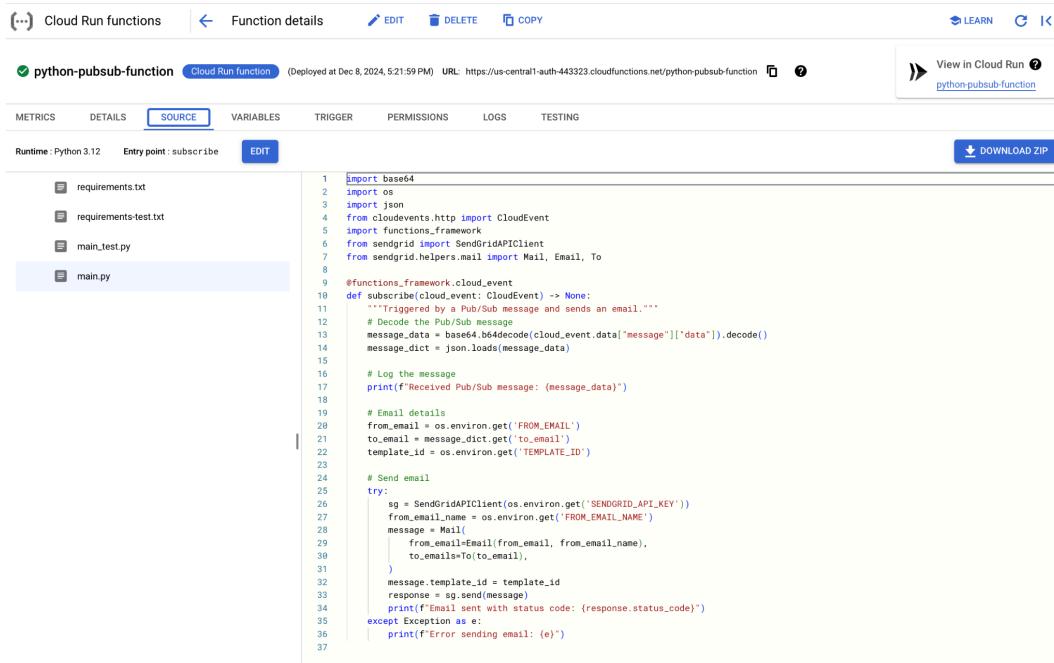
# Microservices

## User Profile Microservice

The user profile microservice is responsible for pulling user-specific information from Canvas API and the application database.

When creating an account for StudyLink, the user authenticates their account via Google Login and a Canvas token. User information—full name, email address, enrolled courses, and pronouns—are retrieved from the user's Canvas profile. As user profile information can be updated in Canvas, and active courses change every semester, the service updates this data accordingly. The User microservice contains an authentication section in the middleware that checks the Google Token and the JWT Token passed in the headers of PUT, POST and DELETE requests, as well as some GET requests. This section will be used by the Composite microservice to verify authentication for PUT, POST and DELETE requests for all microservices. It is also an additional security check, on top of what is implemented in the API Gateway.

Upon signing up to the application, the user will receive an automatic email notification confirming they have signed up. This FaaS was implemented through Google Cloud Run Functions. The email is sent via an integration with SendGrid, which sends the user a dynamic email template, which updates based on the name of the new user:



The screenshot shows the Google Cloud Run Function details page for the 'python-pubsub-function'. The function is deployed at Dec 8, 2024, 5:21:59 PM with the URL https://us-central1-auth-44323.cloudfunctions.net/python-pubsub-function. The source tab is selected, showing the following Python code:

```
1 import base64
2 import os
3 import json
4 from cloudevents.http import CloudEvent
5 import functions_framework
6 from sendgrid import SendGridAPIClient
7 from sendgrid.helpers.mail import Mail, Email, To
8
9 # functions_framework.cloud_event
10 def subscribe(cloud_event: CloudEvent) -> None:
11     """Triggered by a Pub/Sub message and sends an email."""
12     # Decode the Pub/Sub message
13     message_data = base64.b64decode(cloud.event.data["message"]["data"]).decode()
14     message_dict = json.loads(message_data)
15
16     # Log the message
17     print(f'Received Pub/Sub message: {message_data}')
18
19     # Email details
20     from_email = os.environ.get('FROM_EMAIL')
21     to_email = message_dict.get('to_email')
22     template_id = os.environ.get('TEMPLATE_ID')
23
24     # Send email
25     try:
26         sg = SendGridAPIClient(os.environ.get('SENDGRID_API_KEY'))
27         from_email_name = os.environ.get('FROM_EMAIL_NAME')
28         message = Mail(
29             from_email=Email(from_email, from_email_name),
30             to_emails=To(to_email),
31         )
32         message.template_id = template_id
33         response = sg.send(message)
34         print(f'Email sent with status code: {response.status_code}')
35     except Exception as e:
36         print(f'Error sending email: {e}')
37
```

The User Profile Microservice can create and update a user profile, retrieve a single user profile or query for a list of students by name or course, and delete a profile:

## OpenAPI Documentation

users

POST /users/{user_id}/profile	
POST	Create Or Update Profile
GET /users/{user_id}/profile	
GET	Get User Profile
DELETE /users/{user_id}/profile	
DELETE	Delete User Profile
GET /users	
GET	Get Users
GET /users/{user_id}/login	
GET	User Login

### Create or Update Profile (POST)

POST /users/{user\_id}/profile

```
curl -X POST "http://localhost:8000/users/{user_id}/profile" \
-H "Authorization: Bearer <JWT_TOKEN>" \
-H "Content-Type: application/json" \
-d '{ "token": "<PROFILE_TOKEN>" }'
```

Replace {user\_id} with the user ID, <JWT\_TOKEN> with the actual token, and <PROFILE\_TOKEN> with the profile token.

### Get User Profile (GET)

GET /users/{user\_id}/profile

```
curl -X GET "http://localhost:8000/users/{user_id}/profile" \
-H "Authorization: Bearer <JWT_TOKEN>"
```

Replace {user\_id} and <JWT\_TOKEN> with the user ID and token.

### Delete User Profile (DELETE)

DELETE /users/{user\_id}/profile

```
curl -X DELETE "http://localhost:8000/users/{user_id}/profile" \
-H "Authorization: Bearer <JWT_TOKEN>"
```

Replace {user\_id} and <JWT\_TOKEN> with the user ID and token.

### Get Users List (GET)

GET /users

```
curl -X GET
"http://localhost:8000/users?skip=0&limit=10&name=<NAME>&course=<COURSE>" \
-H "Authorization: Bearer <JWT_TOKEN>"
```

Replace <NAME> and <COURSE> with search parameters if needed. You can omit them for general listing. Replace <JWT\_TOKEN> as required.

### User Login (GET)

GET /users/{user\_id}/login

```
curl -X GET "http://localhost:8000/users/{user_id}/login" \
-H "Authorization: Bearer <JWT_TOKEN>"
```

Replace {user\_id} and <JWT\_TOKEN> with appropriate values.

## FastAPI Responses

- Uses public IP address from EC2 instance
- Demonstrates connection to Canvas and StudyLink database
- HATEOAS and links
- Correlation ID and propagation (x-trace-id)

*Example of creating a user profile (POST):*

The screenshot shows a POST request to `/users/{user_id}/profile` with the following parameters:

- `user_id` (required, string, path): azs2117
- `token` (required, string, query): 1396-UKvEfVmFL8F6vv3XnamxzfwD43QDmDbwKm8TZahuGZKX83hRueULALqrNxzAKhUm

The interface includes sections for Responses, Curl command, Server response, and Response headers, showing detailed JSON responses and headers.

## Logging and Tracing

```
@app.middleware("http")
async def log_requests(request: Request, call_next):
    trace_id = str(uuid.uuid4()) # Generate a unique trace ID for the request
    request.state.trace_id = trace_id
    logging.info(f"TRACE_ID={trace_id} - Incoming Request: {request.method} {request.url}")

    start_time = time.time()
    response = await call_next(request)
    processing_time = time.time() - start_time

    logging.info(
        f"TRACE_ID={trace_id} - Completed Request: {request.method} {request.url} "
        f"with Status {response.status_code} in {processing_time:.4f}s"
    )
    response.headers["X-Trace-Id"] = trace_id
    return response
```

### Example of EC2 Instance (AWS)

```
[ec2-user@ip-172-31-19-159 ~]$ uvicorn app.main:app --host 0.0.0.0 --port 8000
INFO:  Started server process [4142]
INFO:  Waiting for application startup.
INFO:  Application startup complete.
INFO:  Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
INFO:root:TRACE_ID=c736ad1b-f65e-44fe-b837-0b29da481dec - Incoming Request: GET http://54.227.241.75:8000/users/er2788/profile
user ('iss': 'https://accounts.google.com', 'exp': '869720524007~v780ntia4rplrefmjocnlque6bn570a.apps.googleusercontent.com', 'aud': '869720524007~v780ntia4rplrefmjocnlque6bn570a.apps.googleusercontent.com', 'sub': '118248010382975206780', 'hd': 'columbia.edu', 'email': 'er2788@columbia.edu', 'email_verified': True, 'name': 'Emanuela Romano', 'picture': 'https://lh3.googleusercontent.com/a/ACg8ocI601w1kRRybVwHVF8j0SGjNxM0NkxNzg5LCPyXQlOj93M0QMDUs019.OStMhM4YTw9_GcaPz71uj0rn9-IRYHzsDP-XIXA', 'jti': '1734305389', 'iat': 1734308989, 'exp': 1734308989, 'nbf': 1734305389, 'given_name': 'Emanuela', 'family_name': 'Romano', 'iat': 1734305389, 'exp': 1734308989, 'nbf': 1734305389, 'jti': '5828b746ab0a3d68fa3dc6f2a98c6794c484e9a')
jwt_secret_key very-secret-key
jwt_algorithm HS256
Verifying token...
Credentials scheme: 'Bearer' credentials: 'eyJhbGciOiJIUzI1NiIsInR5cIiKpXCVJ9_eyJlc2VyX2lkIjoixXyNzg4iwiwzWhnawioIjci3ODhAY29sdWliaEuZWRll1iwiZjJhnkzIjpbinwzX16cmVhCtsInVzXX162GVwZXRI1iwidXNlcjpwb3N0IividXNlcjpwdXQIKSw1XhWjoxNzM0NkxNzg5LCPyXQlOj93M0QMDUs019.OStMhM4YTw9_GcaPz71uj0rn9-IRYHzsDP-XIXA'
Google ('iss': 'https://accounts.google.com', 'exp': '869720524007~v780ntia4rplrefmjocnlque6bn570a.apps.googleusercontent.com', 'aud': '869720524007~v780ntia4rplrefmjocnlque6bn570a.apps.googleusercontent.com', 'sub': '118248010382975206780', 'hd': 'columbia.edu', 'email': 'er2788@columbia.edu', 'email_verified': True, 'name': 'Emanuela Romano', 'picture': 'https://lh3.googleusercontent.com/a/ACg8ocI601w1kRRybVwHVF8j0SGjNxM0NkxNzg5LCPyXQlOj93M0QMDUs019.OStMhM4YTw9_GcaPz71uj0rn9-IRYHzsDP-XIXA', 'jti': '1734305389', 'iat': 1734308989, 'exp': 1734308989, 'nbf': 1734305389, 'given_name': 'Emanuela', 'family_name': 'Romano', 'iat': 1734305389, 'exp': 1734308989, 'nbf': 1734305389, 'jti': '5828b746ab0a3d68fa3dc6f2a98c6794c484e9a')
Token eyJhbGciOiJIUzI1NiIsInR5cIiKpXCVJ9_eyJlc2VyX2lkIjoixXyNzg4iwiwzWhnawioIjci3ODhAY29sdWliaEuZWRll1iwiZjJhnkzIjpbinwzX16cmVhCtsInVzXX162GVwZXRI1iwidXNlcjpwb3N0IividXNlcjpwdXQIKSw1XhWjoxNzM0NkxNzg5LCPyXQlOj93M0QMDUs019.OStMhM4YTw9_GcaPz71uj0rn9-IRYHzsDP-XIXA'
Access token: eyJhbGciOiJIUzI1NiIsInR5cIiKpXCVJ9_eyJlc2VyX2lkIjoixXyNzg4iwiwzWhnawioIjci3ODhAY29sdWliaEuZWRll1iwiZjJhnkzIjpbinwzX16cmVhCtsInVzXX162GVwZXRI1iwidXNlcjpwb3N0IividXNlcjpwdXQIKSw1XhWjoxNzM0NkxNzg5LCPyXQlOj93M0QMDUs019.OStMhM4YTw9_GcaPz71uj0rn9-IRYHzsDP-XIXA', 'grants': ['user:read', 'user:delete', 'user:post', 'user:put'], 'exp': 1734301789, 'iat': 1734305389}
Comparing emails - JWT: er2788@columbia.edu, Google: er2788@columbia.edu
Checking for required grant: user:read
Verification successful, returning payload
INFO:root:TRACE_ID=c736ad1b-f65e-44fe-b837-0b29da481dec - Completed Request: GET http://54.227.241.75:8000/users/er2788/profile with Status 200 in 0.05728s
INFO:  3.25..32.76% - "GET /users/er2788/profile" 200 OK
INFO:root:TRACE_ID=f6a5a80c-d282-4dee-b97b-869c151abfa - Incoming Request: GET http://54.227.241.75:8000/users/er2788/login
user ('iss': 'https://accounts.google.com', 'exp': '869720524007~v780ntia4rplrefmjocnlque6bn570a.apps.googleusercontent.com', 'aud': '869720524007~v780ntia4rplrefmjocnlque6bn570a.apps.googleusercontent.com', 'sub': '118248010382975206780', 'hd': 'columbia.edu', 'email': 'er2788@columbia.edu', 'email_verified': True, 'name': 'Emanuela Romano', 'picture': 'https://lh3.googleusercontent.com/a/ACg8ocI601w1kRRybVwHVF8j0SGjNxM0NkxNzg5LCPyXQlOj93M0QMDUs019.OStMhM4YTw9_GcaPz71uj0rn9-IRYHzsDP-XIXA', 'jti': '1734305442', 'iat': 1734308989, 'exp': 1734308989, 'nbf': 1734305442, 'given_name': 'Emanuela', 'family_name': 'Romano', 'iat': 1734305442, 'exp': 1734308989, 'nbf': 1734305442, 'jti': '988229a7541ca2a7d6702ff8f03bc1f730052c')
INFO:root:Login in user_id=er2788
INFO:root:TRACE_ID=f6a5a80c-d282-4dee-b97b-869c151abfa - Completed Request: GET http://54.227.241.75:8000/users/er2788/login with Status 200 in 0.01478s
INFO:  3.25..39.21% - "GET /users/er2788/login" 200 OK
```

### Example of GCP logging

1,536 results							Actions
SEVERITY	TIME	SUMMARY					
1,437	> * 2024-12-15 16:41:14.144	GET	307	302 B	4 ms	Chrome 131.	
90	> * 2024-12-15 16:41:14.207	GET	200	886 B	277 ms	Chrome 13	
5	> * 2024-12-15 16:41:14.211	https://91h3sz8zwa.execute-api.us-east-1.ama...					
4	> * 2024-12-15 16:41:14.755	GET	307	302 B	3 ms	Chrome 131.	
t... 759	> * 2024-12-15 16:41:14.823	GET	200	891 B	288 ms	Chrome 13	
	> * 2024-12-15 16:41:14.828	https://91h3sz8zwa.execute-api.us-east-1.ama...					
	> * 2024-12-15 16:41:15.180	GET	307	302 B	4 ms	Chrome 131.	
440	> * 2024-12-15 16:41:15.241	GET	200	886 B	278 ms	Chrome 13	
243	> * 2024-12-15 16:41:15.245	https://91h3sz8zwa.execute-api.us-east-1.ama...					
90	> * 2024-12-15 16:41:15.794	GET	307	302 B	4 ms	Chrome 131.	
/ 4	> * 2024-12-15 16:41:15.866	GET	200	886 B	281 ms	Chrome 13	
1,536	> * 2024-12-15 16:41:15.870	https://91h3sz8zwa.execute-api.us-east-1.ama...					
	> * 2024-12-15 16:41:16.223	GET	307	302 B	4 ms	Chrome 131.	
	> * 2024-12-15 16:41:16.290 EST	GET	200	891 B	280 ms	Chrome 13	
	> * 2024-12-15 16:41:16.290	https://91h3sz8zwa.execute-api.us-eas...					

# Course Enrollment Service

- **Functionality:** This microservice pulls the enrollment data from the university's system via the Canvas API. It is responsible for listing all the courses a student is enrolled in and all the students enrolled in a course. Accepts GET requests.
  - Retrieve student-specific enrollment data when given a uni and Canvas token.
  - Retrieve roster of all students in a course when given course code and Canvas token.
  - Update and manage course data for each student.
  - Provide an interface for other services to access a student's course list.
- **How It Works:** When a student logs into StudyLink, this service fetches all the courses in which the student is enrolled in. When a student clicks to create a study group, they will see a list of their classes and subsequently their peers who are also enrolled in StudyLink (using information from the User microservice database).

Incorporates github workflow that deploys a microservice on commit:

The screenshot shows a GitHub Actions workflow named "Deploy to EC2". It lists nine deployment steps, each with a green checkmark, indicating success. The steps are: "Update README.md", "Update README.md", "middleware and logging wit", "Update main.py", "Update students.py", "Update courses.py", "Update courses.py", "Update students.py", and "Update students.py". Each step includes a link to the commit (e.g., fc2a786, e4baef32, c538e5e, 7d76138, 79057fd, a42c901, aea89de, 90c77f8), the branch (main), the time of deployment (3 days ago), and a copy icon.

Showing application running in AWS:

The screenshot shows a terminal window within the AWS CloudWatch interface. The title bar says "aws Services Search [Option+S]". The terminal displays a command-line session where a user runs "python3 main.py". The log output shows the application starting up, attempting to bind to port 8000, and failing due to an address already in use error (Errno 98). The session ends with the application shutting down. A blue banner at the top provides a keyboard shortcut: "To tab out of the terminal window and select the next button element, press the left and right Shift keys together."

```
[ec2-user@ip-172-31-88-215 app]$ python3 main.py
INFO: Started server process [28896]
INFO: Waiting for application startup.
INFO: Application startup complete.
ERROR: [Errno 98] error while attempting to bind on address ('0.0.0.0', 8000): address already in use
INFO: Waiting for application shutdown.
INFO: Application shutdown complete.
[ec2-user@ip-172-31-88-215 app]$
```

FastAPI entry page, using public IP address from EC2 instance + calls on FastAPI to show connection to Canvas and Database:

The screenshot shows the FastAPI documentation page for version 0.1.0, generated by OAS 3.1. It includes the OpenAPI JSON link. The API schema is organized into sections: **students**, **courses**, and **default**. Under **students**, there is a **GET /users/{student\_id}/courses** endpoint labeled "Get Student Courses". Under **courses**, there is a **GET /course/{course\_code}/students** endpoint labeled "Get Course Students". Under **default**, there is a **GET /** endpoint labeled "Root". Below these sections, under **Schemas**, are two error classes: **HTTPValidationError** and **ValidationError**, each with an "Expand all object" link.

Showing HATEOAS Links and Correlation ID propagation and tracking in response:

The screenshot shows the detailed response for the GET request to `/users/jam2492/courses`. The response code is 200. The response body is a JSON object containing the student's ID, a list of courses, and a self-link. The response headers include Content-Length, Content-Type, Date, Server, and X-Correlation-ID, all tracking the correlation ID `cca8d801-c3a4-45ff-ac54-92695f7676ed`.

```
curl -X 'GET' \
'http://52.91.211.126:8000/users/jam2492/courses' \
-H 'accept: application/json' \
-H 'token: cca8d801-c3a4-45ff-ac54-92695f7676ed'

Request URL
http://52.91.211.126:8000/users/jam2492/courses

Server response
Code Details
200 Response body
{
  "student_id": "jam2492",
  "courses": [
    "AMSTGU4300_001_2024_3 - Latina/o/x NY",
    "COMSN4153_001_2024_3 - Cloud Computing",
    "EDUCBC3030_001_2024_3 - Critical Pedagogies",
    "EDUCBC3043_001_2024_3 - Making Change: Activism, Social Movement",
    "PSYCBC1010_005_2024_3 - INTRO LAB EXPERIMENTAL PSYCH"
  ],
  "links": {
    "self": {
      "href": "/users/jam2492/courses"
    }
  }
}

Response headers
content-length: 336
content-type: application/json
date: Thu, 12 Dec 2024 17:40:39 GMT
server: uvicorn
x-correlation-id: cca8d801-c3a4-45ff-ac54-92695f7676ed
```

## Logging for Microservice through CloudWatch:

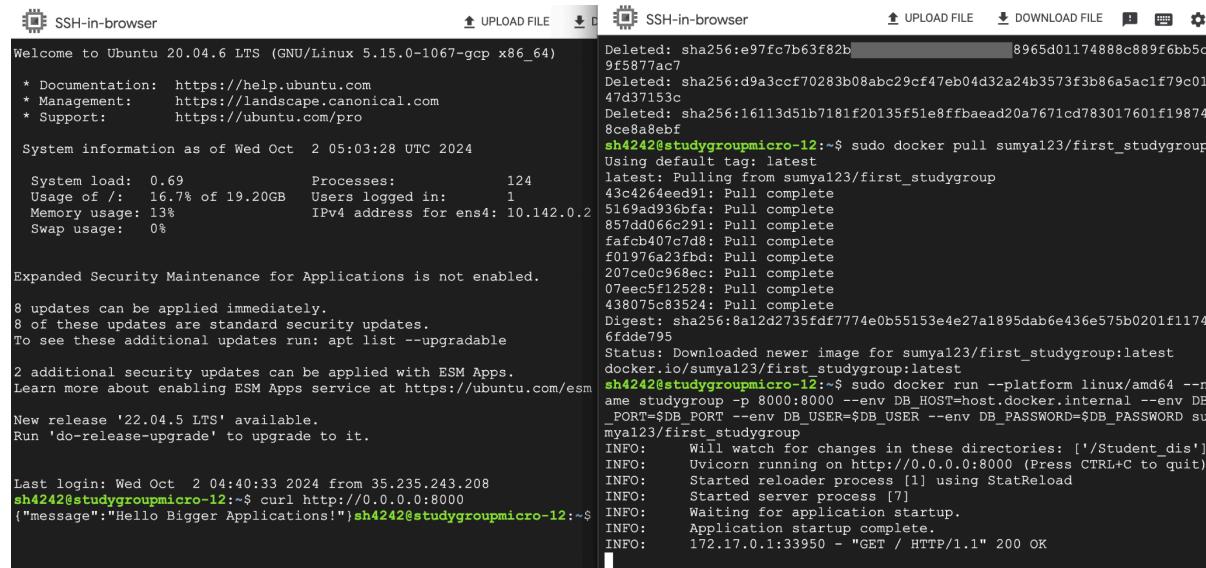
Log events		Actions ▾	Start tailing	Create metric filter		
Filter events - press enter to search		1m	1h	Local timezone ▾	Display ▾	⚙️
▶	Timestamp	Message				
There are older events to load. <a href="#">Load more</a> .						
▶	2024-12-13T20:47:54.000-05:00	2024-12-14 01:47:54,725 INFO: Request path: /docs - Correlation-ID: Not provided [in /ho...				
▶	2024-12-13T20:47:55.000-05:00	2024-12-14 01:47:55,280 INFO: Request path: /openapi.json - Correlation-ID: Not provided...				
▶	2024-12-13T20:48:24.000-05:00	2024-12-14 01:48:24,346 INFO: Request path: /docs - Correlation-ID: Not provided [in /ho...				
▶	2024-12-13T20:49:09.000-05:00	2024-12-14 01:49:09,157 INFO: Request path: /docs - Correlation-ID: Not provided [in /ho...				
▶	2024-12-13T20:49:09.000-05:00	2024-12-14 01:49:09,392 INFO: Request path: /openapi.json - Correlation-ID: Not provided...				
▶	2024-12-13T20:53:04.000-05:00	2024-12-14 01:53:04,303 INFO: Request path: / - Correlation-ID: Not provided [in /home/e...				
▶	2024-12-13T20:55:59.000-05:00	2024-12-14 01:55:59,230 INFO: Request path: / - Correlation-ID: Not provided [in /home/e...				
▶	2024-12-14T01:06:56.000-05:00	2024-12-14 06:06:56,214 INFO: Request path: /aaa9 - Correlation-ID: Not provided [in /ho...				
▶	2024-12-14T01:06:57.000-05:00	2024-12-14 06:06:57,936 INFO: Request path: /aab8 - Correlation-ID: Not provided [in /ho...				
▶	2024-12-14T01:06:59.000-05:00	2024-12-14 06:06:59,691 INFO: Request path: / - Correlation-ID: Not provided [in /home/e...				
▶	2024-12-14T02:36:01.000-05:00	2024-12-14 07:36:01,454 INFO: Request path: / - Correlation-ID: Not provided [in /home/e...				
▶	2024-12-14T02:43:10.000-05:00	2024-12-14 07:43:10,079 INFO: Request path: /favicon.ico - Correlation-ID: Not provided ...				
▶	2024-12-14T04:34:27.000-05:00	2024-12-14 09:34:27,446 INFO: Request path: / - Correlation-ID: Not provided [in /home/e...				
▶	2024-12-14T06:19:14.000-05:00	2024-12-14 11:19:14,318 INFO: Request path: / - Correlation-ID: Not provided [in /home/e...				
▶	2024-12-14T13:18:56.000-05:00	2024-12-14 18:18:56,524 INFO: Request path: / - Correlation-ID: Not provided [in /home/e...				
▼	2024-12-14T18:12:47.000-05:00	2024-12-14 23:12:47,302 INFO: Request path: / - Correlation-ID: Not provided [in /home/e...				

## Study Group Service

- The Study Group Microservice is a central component of the StudyLink application, which aims to help students bring students together, allowing them to study together.
- Through this microservice, it is possible to:
  - Create new study groups
  - Update existing groups with relevant details
  - Delete groups no longer needed
  - Retrieve information about all study groups or a specific one

Used Docker to containerize the whole application and deploy on Google Cloud (Ubuntu Platform)

### Docker Deployment



The screenshot shows two side-by-side terminal windows titled "SSH-in-browser".

**Left Terminal:**

```
Welcome to Ubuntu 20.04.6 LTS (GNU/Linux 5.15.0-1067-gcp x86_64)

 * Documentation: https://help.ubuntu.com
 * Management: https://landscape.canonical.com
 * Support: https://ubuntu.com/pro

System information as of Wed Oct 2 05:03:28 UTC 2024

System load: 0.69      Processes:          124
Usage of /: 16.7% of 19.20GB   Users logged in:    1
Memory usage: 13%           IPv4 address for ens4: 10.142.0.2
Swap usage:  0%
```

Expanded Security Maintenance for Applications is not enabled.

8 updates can be applied immediately.  
8 of these updates are standard security updates.  
To see these additional updates run: apt list --upgradable

2 additional security updates can be applied with ESM Apps.  
Learn more about enabling ESM Apps service at https://ubuntu.com/esm

New release '22.04.5 LTS' available.  
Run 'do-release-upgrade' to upgrade to it.

Last login: Wed Oct 2 04:40:33 2024 from 35.235.243.208  
**sh4242@studygroupmicro-12:**\$ curl http://0.0.0.0:8000

{"message": "Hello Bigger Applications!"}

**Right Terminal:**

```
Deleted: sha256:e97fc7b63f82b          8965d01174888c889f6bb5c
9f5877ac7
Deleted: sha256:d9a3ccf70283b08abc29cf47eb04d32a24b3573f3b86a5acf79c01
47d37153c
Deleted: sha256:16113d51b7181f20135f51e8ffbaead20a7671cd783017601f19874
8ce8a8ebf
sh4242@studygroupmicro-12:$ sudo docker pull sumya123/first_studygroup
Using default tag: latest
latest: Pulling from sumya123/first_studygroup
43c4264eed91: Pull complete
5169ad936bfa: Pull complete
857dd066c291: Pull complete
fafcb407c7d8: Pull complete
f01976a23fb: Pull complete
207ce0968ec: Pull complete
07eec5f12528: Pull complete
438075c83524: Pull complete
Digest: sha256:8a12d2735fd7774e0b55153e4e27a1895dab6e436e575b0201f1174
6fdde795
Status: Downloaded newer image for sumya123/first_studygroup:latest
docker.io/sumya123/first_studygroup:latest
sh4242@studygroupmicro-12:$ sudo docker run --platform linux/amd64 --name studygroup -p 8000:8000 --env DB_HOST=host.docker.internal --env DB_PORT=$DB_PORT --env DB_USER=$DB_USER --env DB_PASSWORD=$DB_PASSWORD sumya123/first_studygroup
INFO: Will watch for changes in these directories: ['/Student_dis']
INFO: Uvicorn running on http://0.0.0.0:8000 (Press CTRL+C to quit)
INFO: Started reloader process [1] using StatReload
INFO: Started server process [7]
INFO: Waiting for application startup.
INFO: Application startup complete.
INFO: 172.17.0.1:33950 - "GET / HTTP/1.1" 200 OK
```

## Log and trace, Correlation ID and propagation, in a cloud's logging tools

This shows the microservice running on GCP with logging and correlation ID propagation. The left shows the cloud logging, and the right shows the OpenAPI docs and most recent request (DELETE).

The screenshot displays two side-by-side interfaces. On the left is the Cloud Logging interface, showing a log entry for a DELETE request to /study-group/{group\_id}. The log includes detailed information about the study group, such as its name ('Test Study Group'), creation date ('2024-12-07T15:01:926Z'), and members ('["user1"]'). On the right is the OpenAPI Swagger UI, specifically for the DELETE /study-group/{group\_id} endpoint. It shows the 'Parameters' section with 'group\_id' set to '16'. Below it are sections for 'Responses', 'Curl', 'Request URL', 'Server response', and 'Code Details'. The 'Code Details' section shows the JSON response body: { "status": "success", "message": "Study group 16 has been deleted" }. The 'Server response' section shows the raw HTTP response: HTTP/1.1 200 OK [redacted].

Logs are also in this format:

```
logging.info(f"TRACE_ID={trace_id} - Incoming Request: {request.method} {request.url}")

start_time = time.time()
response = await call_next(request)
processing_time = time.time() - start_time

logging.info(
    f"TRACE_ID={trace_id} - Completed Request: {request.method} {request.url} "
    f"with Status {response.status_code} in {processing_time:.4f}s"
)
response.headers["X-Trace-ID"] = trace_id # Include trace ID in response headers
```

## OpenAPI documents

The screenshot shows the FastAPI - Swagger UI interface at [34.55.93.248:8002/docs](http://34.55.93.248:8002/docs). The top bar indicates the version is 0.1.0 and OAS 3.1. The main content area is titled "study\_group" and contains several API endpoints:

- GET /study-group** Get All Study Groups
- POST /study-group** Post Study Group
- GET /study-group/{group\_id}** Get Single Study Group
- DELETE /study-group/{group\_id}** Delete Study Group
- PUT /study-group/{group\_id}** Edit Study Group

Below this is a section titled "default" with a single endpoint:

- GET /** Root

At the bottom, there is a "Schemas" section containing definitions for `HTTPValidationError` and `ValidationError`.

## HATEOAS and support for links

This demonstrates a few success calls for post, put and get

POST	PUT
<pre>{   "status": "success",   "message": "Group 'Test Study Group' created successfully.",   "group_id": 17,   "data": {     "group_name": "Test Study Group",     "created_by": "test_user",     "created_at": "2025-06-30",     "course_id": "1234",     "is_recurring": true,     "meeting_date": "2025-06-30",     "recurrence_frequency": "weekly",     "recurrence_end_date": "2025-06-30",     "start_time": "09:00:00",     "end_time": "10:00:00",     "members": [       "user1",       "user2",       "test_user"     ]   },   "_links": {     "self": {       "href": "/study-group/17"     },     "list": {       "href": "/study-group"     }   } }</pre>	<pre>{   "status": "success",   "message": "Study group 18 has been updated successfully",   "updated_fields": {     "group_name": "Updated Study Group Name",     "is_recurring": false,     "meeting_date": "2025-06-30",     "recurrence_end_date": "2025-07-30",     "start_time": "09:30:00",     "end_time": "10:30:00",     "members": ["user3", "user4"]   },   "_links": {     "self": {       "href": "/study-group/18"     },     "list": {       "href": "/study-group"     },     "edit": {       "href": "/study-group/18",       "method": "PUT"     },     "delete": {       "href": "/study-group/18",       "method": "DELETE"     }   } }</pre>

<pre>         "href": "/study-group"     } } } </pre>	<pre>     } } </pre>
<p><b>GET</b></p> <pre> {   "status": "success",   "data": [     {       "group_id": 4,       "group_name": "Cloud Computing",       "created_by": "jyl2196",       "created_at": "2024-12-06T21:15:01.926Z",       "is_recurring": false,       "meeting_date": "2024-12-07",       "recurrence_frequency": "",       "start_time": "16:00",       "end_time": "16:30",       "recurrence_end_date": "",       "course_id": "COMSW4153_001_2024_3 - Cloud Computing",       "members": [         "jyl2196",         "er2788"       ],       "_links": {         "self": {           "href": "/study-group/4"         },         "edit": {           "href": "/study-group/4",           "method": "PUT"         },         "delete": {           "href": "/study-group/4",           "method": "DELETE"         }       }     },     ...<i>the rest of the groups</i> ],   "message": "Fetched 3 study group(s)" } </pre>	

## Chat Service

- **Functionality:** The chat microservice allows the creation, modification and retrieval of conversation details.
  - Create Conversations: Users can initiate new conversations, whether it's a one-on-one chat or a group discussion. The service generates unique identifiers for each conversation, ensuring easy access and management.
  - Retrieve Conversation Details: The microservice provides endpoints to fetch conversation details by convo ID / retrieve all convo details.
  - Modify Conversations: Users' conversation will be updated when new convos occur. This includes adding or removing participants, changing the conversation title, or simply having new messages .
  - Delete Conversation Details: The microservice provides endpoints to delete conversation details by convo ID.
- **How it works:** By supporting these functionalities, the chat microservice fosters dynamic interactions between users

## OpenAPI documents

The screenshot displays the OpenAPI documentation for a Chat Service, organized into sections: conversations, tasks, and default.

**conversations**

- POST** /conversations/ Create Conversation
- GET** /conversations/ Get All Conversations
- PUT** /conversations/{conversation\_id} Update Conversation
- GET** /conversations/{conversation\_id} Get Conversation
- DELETE** /conversations/{conversation\_id} Delete Conversation

**tasks**

- GET** /tasks/{task\_id} Get Task Status

**default**

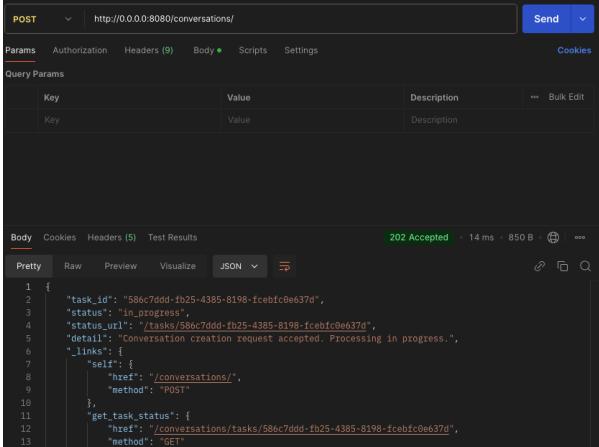
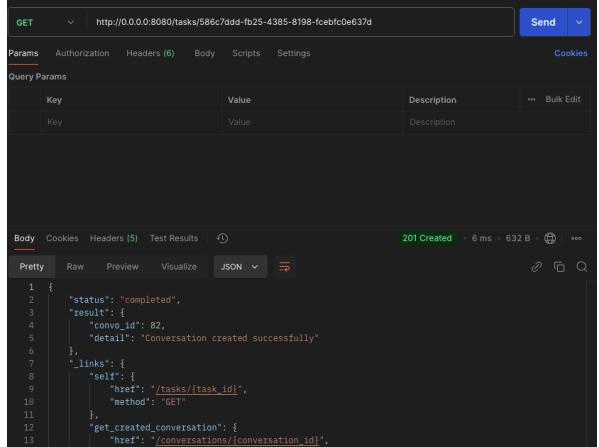
- GET** / Root

**Schemas**

- HTTPValidationError > Expand all `object`
- ValidationError > Expand all `object`

## For POST, implement 201 created with a link header

When creating a conversation successfully, return 201 status code.

<p>Status Code 202 - Create Request Accepted</p> 	<p>Status Code 201 - Conversation Created</p> 
--	--

## Asynchronous Execution Pattern

<p>Create Conversation Response</p> <pre>{   "task_id": "7107cc35-03c3-4fd0-be38-ebaba2f99dab",   "status": "in_progress",   "status_url": "/tasks/7107cc35-03c3-4fd0-be38-ebaba2f99dab",   "detail": "Conversation creation request accepted. Processing in progress.",   "_links": {     "self": {       "href": "/conversations/",       "method": "POST"     },     "get_task_status": {       "href": "/conversations/tasks/7107cc35-03c3-4fd0-be38-ebaba2f99dab",       "method": "GET"     },     "get_created_conversation": {       "href": "/conversations/{conversation_id}",       "method": "GET"     },     "delete_conversation": {       "href": "/conversations/{conversation_id}",       "method": "DELETE"     },     "list_conversations": {       "href": "/conversations/",       "method": "GET"     },     "create_conversation": {       "href": "/conversations/"     }   } }</pre>	<p>Get Create Conversation Task Status</p> <pre>{   "status": "completed",   "result": {     "convo_id": 82,     "detail": "Conversation created successfully"   },   "_links": {     "self": {       "href": "/tasks/7107cc35-03c3-4fd0-be38-ebaba2f99dab",       "method": "GET"     },     "get_created_conversation": {       "href": "/conversations/82",       "method": "GET"     },     "delete_conversation": {       "href": "/conversations/82",       "method": "DELETE"     },     "list_conversations": {       "href": "/conversations/",       "method": "GET"     },     "create_conversation": {       "href": "/conversations/",       "method": "POST"     }   } }</pre>
---	--

```

        "method": "POST"
    }
}
}

STATUS: 202 Accepted

```

STATUS: 201 Created

## Pagination for GET /conversations

GET [conversations/?page=1&limit=3](#)

```
{
  "detail": [
    {
      "convo_id": 63,
      "name": "Jessica Liang - jyl2196 - Emanuela Romano - er2788",
      "participants": "[\"er2788\", \"jyl2196\"]",
      "messages": "[{\\"text\": \"Hi Jess!\", \"sender\": \"er2788\", \"timestamp\": \"17:04:07\"}]",
      "isGroup": 0
    },
    {
      "convo_id": 64,
      "name": "Cloud Computing",
      "participants": "[\"jyl2196\", \"er2788\"]",
      "messages": "[{\\"text\": \"Hi group!\", \"sender\": \"er2788\", \"timestamp\": \"17:04:15\"}]",
      "isGroup": 1
    },
    {
      "convo_id": 65,
      "name": "Sprint 1 Review",
      "participants": "[\"jyl2196\", \"er2788\", \"azs2117\"]",
      "messages": "[ ]",
      "isGroup": 1
    }
  ],
  "total": 14,
  "page": 1,
  "limit": 3,
  "total_pages": 5
}
```

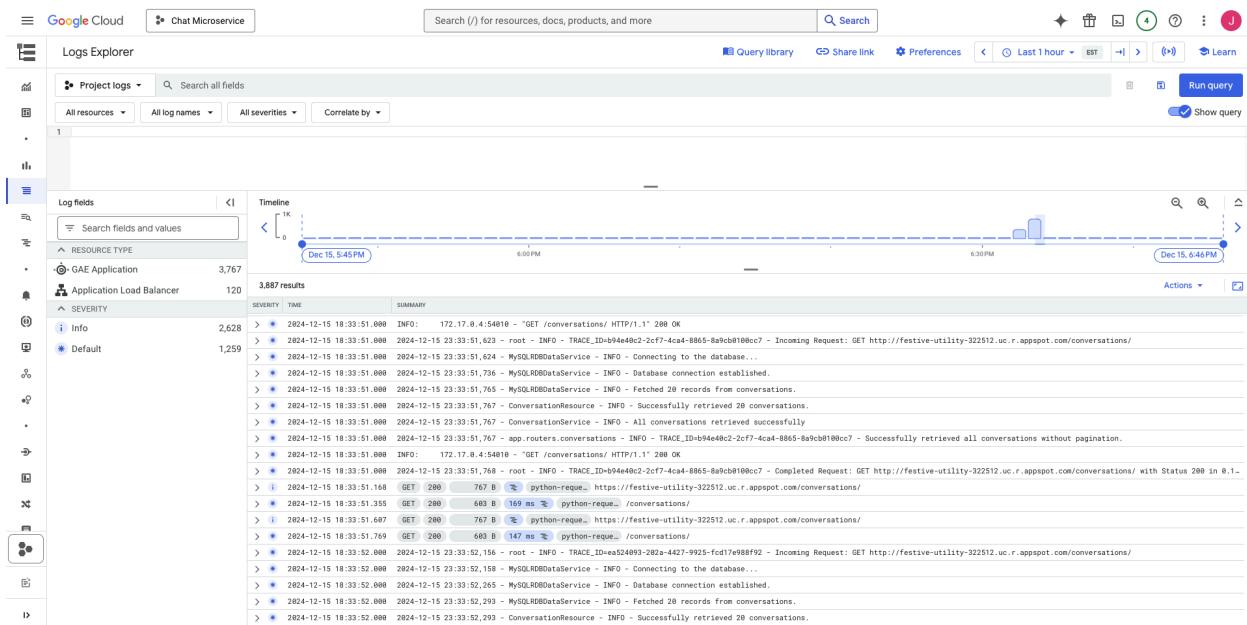
GET [conversations/?page=2&limit=2](#)

```
{
  "detail": [
    {
      "convo_id": 65,
      "name": "Sprint 1 Review",
      "participants": "[\"jyl2196\", \"er2788\", \"azs2117\"]",
      "messages": "[ ]",
      "isGroup": 1
    },
    {
      "convo_id": 66,
      "name": "Coding Workshop",
      "participants": "[\"John Doe\", \"Jane Smith\", \"Bob Johnson\", \"CurrentUser\"]",
      "messages": "[{\\"text\": \"Welcome to our project group!\", \"sender\": \"CurrentUser\", \"timestamp\": \"9:00 AM\"}]",
      "isGroup": 1
    }
  ],
  "total": 14,
  "page": 2,
  "limit": 2,
  "total_pages": 7
}
```

## Logging + Correlation ID and propagation

```
2024-12-14 08:59:18,446 - root - INFO - TRACE_ID=08f9744a-2d8d-40b3-bc43-4fd44d6088f0 - Incoming Request: GET http://0.0.0.0:8080/conversations/72
2024-12-14 08:59:18,447 - MySQLRDBDataService - INFO - Connecting to the database...
2024-12-14 08:59:18,492 - MySQLRDBDataService - INFO - Database connection established.
2024-12-14 08:59:18,500 - MySQLRDBDataService - INFO - Successfully fetched a record from conversations.
2024-12-14 08:59:18,501 - ConversationResource - INFO - Conversation with ID 72 retrieved successfully.
2024-12-14 08:59:18,501 - ConversationService - INFO - Conversation with ID 72 retrieved successfully
2024-12-14 08:59:18,501 - app.routers.conversations - INFO - TRACE_ID=08f9744a-2d8d-40b3-bc43-4fd44d6088f0 - Successfully retrieved conversation with ID 72
2024-12-14 08:59:18,501 - root - INFO - TRACE_ID=08f9744a-2d8d-40b3-bc43-4fd44d6088f0 - Completed Request: GET http://0.0.0.0:8080/conversations/72 with Status 200 in 0.0555s
INFO: 127.0.0.1:49781 - "GET /conversations/72 HTTP/1.1" 200 OK
```

## Demonstrate in a cloud's logging tools



# Composite Service

## API Lists and description

### 1. User Service

- a. Get /StudyLink/v1/users/{user\_id}/login → call the user microservice logging endpoint
- b. Get /StudyLink/v1/users/{user\_id}/profile → get request, has JWT and Google Token as a header, pass along to the user service to verify
- c. Post /StudyLink/v1/users/{user\_id}/profile → post request, has query parameters (canvas token) and Google Token
- d. Delete /StudyLink/v1/users/{user\_id}/profile → delete request, has JWT and Google Token as a header, pass along to the user service to verify
- e. Get /StudyLink/v1/users/ → retriever all users along with JWT and Google Token as the header

### 2. Study Service

- a. Get /StudyLink/v1/study-group/{group\_id} → get request to the study-group microservice
- b. Get /StudyLink/v1/study-group/" → get a request to the study-group microservice (retrieve all groups)
- c. Post /StudyLink/v1/{user\_id}/study-group/ → *asynchronous* post request to user service + study-group + perform rollback operation if the post is successful but the get request to the user service is not successful
- d. Delete /StudyLink/v1/{user\_id}/study-group/{group\_id} → synchronous call to the user service + study-group to delete a group
- e. Put /StudyLink/v1/{user\_id}/study-group/{group\_id}" → synchronous call to the user service + study-group to update a group

### 3. Course Enrollment

- a. Two get requests to retrieve a student's courses and find students of the course

### 4. Chat Service

- a. Post /StudyLink/v1/{user\_id}/conversations → post a chat; has choreography pattern to call the user service and the chat service
- b. Delete "/StudyLink/v1/{user\_id}/conversations/{conversation\_id}", → Has google workflow to call the user and chat services
- c. Put /StudyLink/v1/conversations/{conversation\_id} → Synchronous call between the user and chat service
- d. Get StudyLink/v1/{user\_id}/conversations/{conversation\_id} → Retrieves conversation
- e. Get /StudyLink/v1/conversations → Get all the conversations

## OpenAPI: [Link](#)

The screenshot shows the SwaggerHub interface for the StudyLink API version 1.0.0. The left sidebar contains navigation links for Info, Tags, Servers, and Schemas. The main content area displays the API definition in YAML format, which includes endpoints for users, courses, conversations, and study-group. The right side features a 'Read Only' section with tabs for Code (API definition), Form (request body), Preview (sample response), and Export (JSON). Below these tabs are sections for users, courses, conversations, and study-group, each listing their respective GET, POST, PUT, and DELETE methods with detailed descriptions and examples.

## Logging

```
--  
insertId: 121uq80f4op40g  
logName: projects/cloud-computing-437302/logs/user-login  
receiveTimestamp: '2024-12-15T22:15:52.995159967Z'  
resource:  
  labels:  
    module_id: default  
    project_id: cloud-computing-437302  
    version_id: 20241215t171157  
    zone: projects/90605875943/zones/us6  
  type: gae_app  
textPayload: requesting the user logging function  
timestamp: '2024-12-15T22:15:52.995159967Z'  
__  
insertId: 121uq80f4op40f  
jsonPayload:  
  logging.googleapis.com/diagnostic:  
    instrumentation_source:  
      - name: python  
      version: 3.11.3  
logName: projects/cloud-computing-437302/logs/user-login  
receiveTimestamp: '2024-12-15T22:15:52.995159967Z'  
resource:  
  labels:  
    module_id: default  
    project_id: cloud-computing-437302  
    version_id: 20241215t171157  
    zone: projects/90605875943/zones/us6  
  type: gae_app  
timestamp: '2024-12-15T22:15:52.995159967Z'  
__  
insertId: 3d0eme3f80s2w  
logName: projects/cloud-computing-437302/logs/user-login  
receiveTimestamp: '2024-12-15T22:00:34.346478387Z'  
resource:  
  labels:  
    module_id: default  
    project_id: cloud-computing-437302  
    version_id: 20241215t165646  
    zone: projects/90605875943/zones/us6  
  type: gae_app  
textPayload: requesting the user logging function  
timestamp: '2024-12-15T22:00:34.346478387Z'  
__  
insertId: 3d0eme3f80s2v  
jsonPayload:  
  _op: Sign-in
```

```

self.logging_client = logging.Client()

#this will perform the rest call
def get_user_login(self, user_id:str, google:str):
    payload = {
        "Google-Token": google
    }
    url = f"{self.user_config}/users/{user_id}/login"
    try:
        print("google",google)
        print("calling get user",google)
        log_name = "user-login"
        logger = self.logging_client.logger(log_name)
        text = "requesting the user logging function"
        logger.log_text(text)

        response = requests.get(url, headers=payload, timeout=10)
        response.raise_for_status()

        print(self.user_config, user_id)
        return (response.json(), response.status_code)
    except requests.exceptions.Timeout:
        print("The request timed out!")
        return ({'error': "The request timed out"},408)
    except HTTPError as http_err:
        print(f"HTTP error occurred: {http_err}")
        try:
            # Try to parse the response as JSON
            response_data = response.json() # This is from the external
            print(f'External service response (JSON): {response_data}')
        except json.JSONDecodeError:
            print(f'External service response (non-JSON): {response.text}')

```

## Deployment On PaaS

*app.yaml:*

```

runtime: python312
entrypoint: gunicorn -w 4 -k uvicorn.workers.UvicornWorker app.main:app
env_variables:
  USER_MICRO : "https://9lh3sz8zwa.execute-api.us-east-1.amazonaws.com/user"
  STUDY_MICRO : "https://9lh3sz8zwa.execute-api.us-east-1.amazonaws.com/studygroup"
  CHAT_MICRO : "https://9lh3sz8zwa.execute-api.us-east-1.amazonaws.com/chat"
  COURSE_MICRO : "https://9lh3sz8zwa.execute-api.us-east-1.amazonaws.com/course"
  PROJECT: "cloud-computing-437302"
  LOCATION : "us-central1"
  WORKFLOW : "myFirstWorkflow"

```

*Google deployment:*

The screenshot shows the Google Cloud Platform interface for managing App Engine versions. The top navigation bar includes 'Cloud computing' and a search bar. The main header says 'App Engine' with a dropdown arrow, followed by 'Versions', 'Refresh', 'Delete', 'Stop', 'Start', 'Migrate traffic', and 'Split traffic'. A 'Learn' button is also present.

The left sidebar has links for 'Dashboard', 'Services', 'Versions' (which is selected and highlighted in blue), 'Instances', 'Task queues', and 'Cron jobs'.

The main content area displays a table for 'Versions'. The table has columns: Version, Status, Traffic Allocation, Instances, Runtime, Runtime Lifecycle, and Environment. One row is shown, labeled '20241214t172614' with 'Serving' status, 100% traffic allocation, 0 instances, runtime 'python312', and environment 'General Availability (GA)'.

## Pub-Sub

```
import threading
from app.services.service_factory import ServiceFactory
class Publisher:
    def __init__(self):
        self.subscribers = {}

    def subscribe(self, subscriber, topic):
        print(topic, subscriber)
        if topic not in self.subscribers:
            self.subscribers[topic] = []
        self.subscribers[topic].append(subscriber)

    def publish(self, message, topic, params):
        if topic in self.subscribers:
            for subscriber in self.subscribers[topic]:
                subscriber.event.set()
                subscriber.message = message
                subscriber.params = params

class Subscriber:
    def __init__(self, name):
        self.name = name
        self.event = threading.Event()
        self.message = None
        self.params = []
        service = ServiceFactory()
        self.compo_resource = service.get_service("CompositeResource")
        self.user_error = False

    def receive(self):
        self.event.wait()
        #will sent the request to the client
        print(f"{self.name} received message: {self.message}")
        if self.message == "Get User":
            response_data_get, status_code = self.compo_resource.get_user(self.params[0], self.params[1])
            if status_code != 200:
                self.event.clear()
                self.user_error = True
                return (False, response_data_get, status_code)
            else:
                self.event.clear()
```

## Sync-Asynchronous

### Async

```
async def create_group(self, user_id:str, group_data:dict, google_user:str, jwt_payload:str):
    print("Starting the create group ops")
    response_get, response_post = await asyncio.gather(
        self.compo_resource.get_user_sync_internal(user_id, google_user, jwt_payload),
        self.compo_resource.create_group(group_data)
    )
    print(response_post)
    data, response_get_status = response_get

    response_data, response_post_status= response_post

    print(f"GET Response Status: {response_get_status}")
    print(f"POST Response Status: {response_post_status}")

    # Now handle the conditions based on the status codes
    if response_get_status == 200 and response_post_status != 200:
        raise HTTPException(status_code=response_post_status, detail=response_data)

    elif response_get_status == 200 and response_post_status == 200 or response_post_status == 201:
        return {"message": response_data}

    elif response_get_status != 200 and response_post_status == 200:
        group_id = response_data.get("group_id")
        response_delete_data, status_delete = await self.compo_resource.delete_group_async(group_id)

        if status_delete != 200:
            raise HTTPException(status_code=status_delete, detail=f"rollback failed,{response_delete_data}")

    raise HTTPException(status_code=400, detail="GET request failed, rollback succeeded, Post Suspended")

    elif response_get_status != 200 and (response_post_status != 200 and response_post_status != 201):
        print("Both GET and POST requests failed.")
        raise HTTPException(status_code=response_post_status, detail=response_data)
```

## Sync

```
def update_group(self, user_id:str, group_id:int, update_data:dict, google_user:str, jwt_payload:str):
    response_data_get, status_code = self.compo_resource.get_user(user_id,google_user, jwt_payload)

    if status_code != 200:
        raise HTTPException(status_code=status_code, detail=response_data_get)

    print("update data1", update_data)
    response_data, status_code = self.compo_resource.update_group(group_id, update_data)

    if status_code != 200:
        raise HTTPException(status_code=status_code, detail=response_data)
    else:
        return response_data
```

## Google Workflow

Workflow.yaml

```
apiVersion: v1
kind: workflow
parameters: [args]
steps:
  - callGetUserService:
      call: http.get
      args:
        url: ${args.user_url} + "/users/" + args.user_id + "/profile"
        timeout: 20
        headers:
          Google-Token: ${args.google_token}
          Authorization: ${args.authorization}
      result: getResponse

  - checkGetResponse:
      switch:
        - condition: ${getResponse.statusCode == 200}
          next: callDeleteChatService
        - condition: ${getResponse.statusCode != 200}
          next: handleError

  - callDeleteChatService:
      call: http.delete
      args:
        url: ${args.chat_url} + "/conversations/" + args.chat_id
        headers:
          Google-Token: ${args.google_token}
          Authorization: ${args.authorization}
      result: deleteResponse

  - handleDeleteResponse:
      switch:
        - condition: ${deleteResponse.statusCode == 200 or deleteResponse.statusCode == 204}
          next: returnSuccess
        - condition: ${deleteResponse.statusCode != 200 and deleteResponse.statusCode != 204}
          next: handleError

  - returnSuccess:
      return:
        message: "Workflow completed successfully."
        get_response_body: ${getResponse.body}
        delete_response_status: ${deleteResponse.statusCode}

  - handleError:
      return:
        error: "An error occurred during the workflow."
        get_response_body: ${getResponse.body}
        delete_response_status: ${deleteResponse.statusCode}
```

Google workflow client

```
workflows_client = workflows_v1.WorkflowsClient()
client = executions_v1.ExecutionsClient()
project = os.getenv("PROJECT")
location = os.getenv("LOCATION")
workflow = os.getenv("WORKFLOW")
parent = workflows_client.workflow_path(project, location, workflow)
print(parent)
service = ServiceFactory()
config = service.get_service("CompositeResourceService")
user_config = config.get_user_config()
chat_config = config.get_chat_config()
args = {

    "user_url": user_config,           # User service URL
    "user_id": user_id,               # User ID
    "chat_url": chat_config,          # Chat service URL
    "chat_id": conversation_id,       # Chat ID
    "google_token": google_user,       # Google token
    "authorization": jwt_payload     # Authorization header
}

argument_json = json.dumps(args)
execution_request = executions_v1.CreateExecutionRequest(
    parent=parent,
    execution={
        "argument": argument_json
    }
)

response = client.create_execution(request=execution_request)

print(f"Execution name: {response.name}")
print(f"Execution State: {response.state}")
execution_finished = False
backoff_delay = 1 # Start wait with delay of 1 second
print("Poll for results...")
while not execution_finished:
    execution = client.get_execution(request={"name": response.name})
    execution_finished = execution.state != executions.Execution.State.ACTIVE

    # If we haven't seen the result yet, wait a second.
    if not execution_finished:
        print("- Waiting for results...")
        time.sleep(backoff_delay)
        # Double the delay to provide exponential backoff.
        backoff_delay *= 2
else:
    print(f"Execution finished with state: {execution.state.name}")
    print(f"Execution results: {execution.result}")
    print(f"Execution error {execution.error}")
return execution
```

# UI

The React and Redux-based UI connects with the User Microservice, enabling account creation, deletion, login, and retrieval of all user profiles. It's also integrated with the Courses Microservice, which allows users to view information on their enrolled courses. The UI is deployed on Google Cloud Storage and it is connected to the composite microservice via the API Gateway link. Information about the API Gateway Link and the Google Client ID (needed for google login) are configured in a .env file.

The UI allows to do the following:

- **Access/Logout:**
  - Signup - via Google Login + Courseworks Token
  - Login - via Google Login
  - Logout
  - Delete Account
- **Study Groups:**
  - Create - only for courses that the user is taking
  - Join - any group associated with a course that the user is taking
  - Edit - only for study groups created by the logged user
  - Delete - only for study groups created by the logged user
- **Calendar** - view on a calendar when the study groups that the user has joined will meet. This section relies on the StudyGroup microservice, to show calendar events for the sturdy groups that the user has joined.
- **Chat** - with users on the application, or with groups that the user has joined. This section includes integration with pagination implemented in the chat microservice.

## API Gateway

The screenshot shows the AWS API Gateway console with the following details:

- API Name:** composite-...akn|tg040i
- Integration Type:** AWS Lambda
- HTTP Method:** POST
- Route Path:** /{proxy+}
- Lambda Function:** composite-microservice-auth (us-east-1)
- Description:** -
- Payload Format Version:** 2.0 (Untested response format)
- Example Policy Statement:** -
- Timeout:** 30000
- Request Parameter Mapping:** Not configured
- Response Parameter Mappings:** Not configured

## Lambda Auth (for API Gateway)

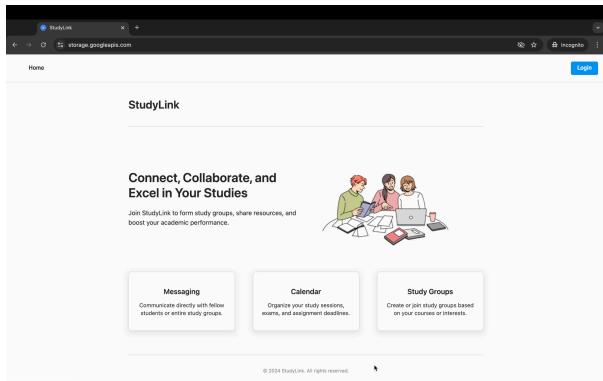
The screenshot shows the AWS Lambda function code editor. The file `lambda\_function.py` contains Python code for handling requests. It imports `jwt` and `os`, defines a `SECRET\_KEY` environment variable, and checks for its presence. It then defines a `lambda\_handler` function that extracts headers, decodes a JWT token, and generates a policy. If the token is invalid, it raises an `InvalidTokenError`. The function handles methods like GET, POST, and DELETE. It also handles Google OAuth tokens by checking for an `Authorization` header and decoding it using `google.oauth2.id\_token`. The code includes comments explaining the logic.

```
import jwt
from jwt.exceptions import InvalidTokenError
import os
SECRET_KEY = os.getenv("SECRET_KEY")
if not SECRET_KEY:
    raise ValueError("Missing SECRET_KEY environment variable")
def lambda_handler(event, context):
    # Extract headers
    headers = event.get("headers", {})
    method = event.get("method")
    auth_header = headers.get("Authorization")
    google_token = headers.get("Google-Token")
    if auth_header:
        try:
            token = auth_header.split(" ")[1]
            if " " in auth_header:
                auth_header = auth_header[1]
            decoded = jwt.decode(token, SECRET_KEY, algorithms=["HS256"])
            if decoded["method"] == "DELETE":
                return generate_policy(decoded.get("sub", "user"), "Allow", event["methodArn"])
            else:
                raise InvalidTokenError()
        except InvalidTokenError as e:
            raise InvalidTokenError(str(e))
    elif google_token:
        # Return a Deny policy if the token is invalid
        raise InvalidTokenError("User is not authenticated")
```

## UI Hosted on Google Storage

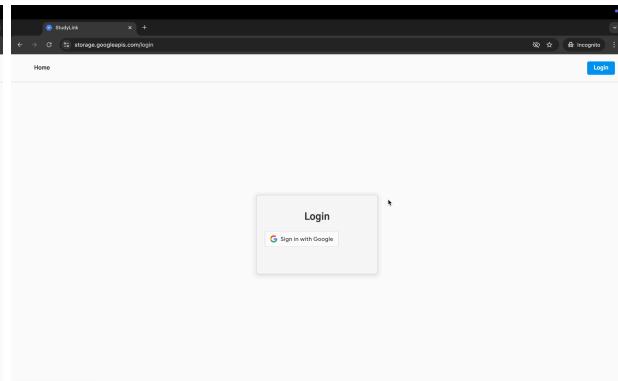
The screenshot shows the Google Cloud Storage Bucket details page for `studylink-app`. The bucket is publicly accessible. The left sidebar shows options for Overview, Buckets, Monitoring, and Settings. The main area displays bucket details like location (us-east1), storage class (Standard), and public access (Public to internet). The Objects tab lists files: `asset-manifest.json` (843 B, application/json), `index.html` (691 B, text/html), and a folder named `static/`. There are tabs for Configuration, Permissions, Protection, Lifecycle, Observability, Inventory Reports, and Operations.

## Homepage



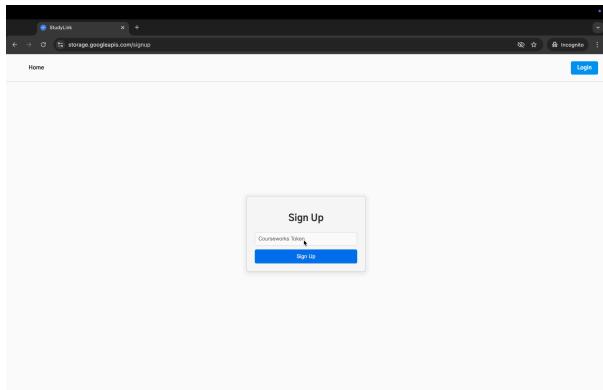
The screenshot shows the StudyLink homepage. At the top, there's a navigation bar with 'Home', 'Messages', 'Calendar', and 'Study Groups' tabs, and a 'Logout' button. Below the navigation is a main section titled 'StudyLink' with a sub-section 'Connect, Collaborate, and Excel In Your Studies'. It features a small illustration of three people studying together. Below this are three cards: 'Messaging' (communicate directly with fellow students or entire study groups), 'Calendar' (organize study sessions, exams, and assignment deadlines), and 'Study Groups' (create or join study groups based on your courses or interests). At the bottom, there's a copyright notice: '© 2024 StudyLink. All rights reserved.'

## Sign In



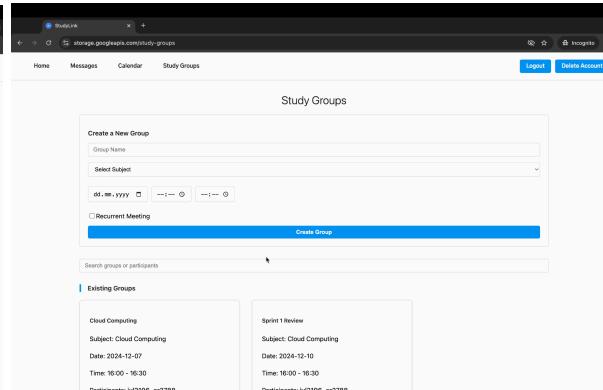
The screenshot shows the StudyLink sign-in page. The URL in the address bar is 'storage.googleapis.com/login'. The page has a 'Login' button at the top right and a 'Google Sign in with Google' button below it. There's also a 'Logout' button at the very top.

## Sign Up (adding Courseworks Token)



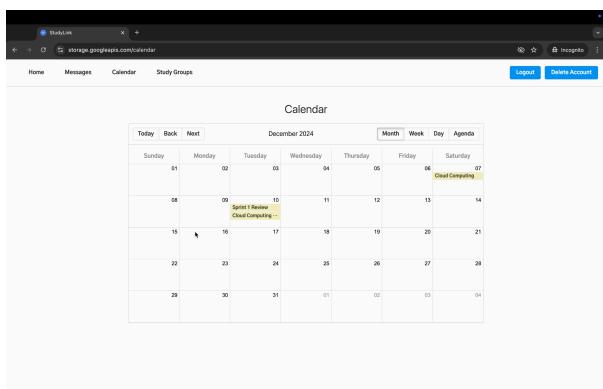
The screenshot shows the StudyLink sign-up page. The URL in the address bar is 'storage.googleapis.com/signup'. A 'Sign Up' button is prominently displayed. Below it, there's a field labeled 'Courseworks Token' with a placeholder 'Enter token here' and a 'Sign Up' button.

## Study Group Page



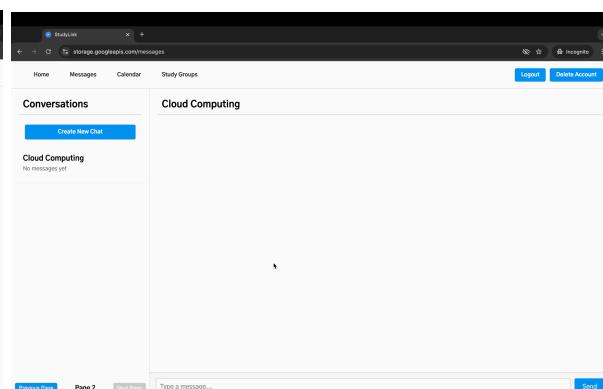
The screenshot shows the Study Group Page. The URL in the address bar is 'storage.googleapis.com/study-groups'. The page has a 'Logout' and 'Delete Account' button at the top right. It features a 'Create a New Group' form with fields for 'Group Name', 'Select Subject', 'Date - mm-yyyy', and 'Recurrent Meeting'. A 'Create Group' button is highlighted in blue. Below this, there's a 'Search groups or participants' input field. On the left, there's a sidebar for 'Existing Groups' showing two entries: 'Cloud Computing' and 'Sprint I Review'.

## Calendar



The screenshot shows the StudyLink calendar page. The URL in the address bar is 'storage.googleapis.com/calendar'. The page has a 'Logout' and 'Delete Account' button at the top right. It features a monthly calendar for December 2024. A specific event, 'Cloud Computing', is highlighted on December 10th. The calendar includes buttons for 'Today', 'Back', 'Next', 'Month', 'Week', 'Day', and 'Agenda'.

## Chat (with pagination)



The screenshot shows the StudyLink chat page. The URL in the address bar is 'storage.googleapis.com/messages'. The page has a 'Logout' and 'Delete Account' button at the top right. It features a 'Conversations' section with a 'Create New Chat' button and a 'Cloud Computing' conversation. The conversation shows a message from 'Cloud Computing' stating 'No messages yet'. At the bottom, there are pagination controls for 'Previous Page', 'Page 2', and 'Next Page', along with a 'Send' button and a 'Type a message...' input field.

## End User Notification



**Emanuela Romano**  
StudyLink Account Created  
To: Amelie Scheil

Inbox - Barnard 12:40 PM

---

Thank you for signing up to StudyLink!

[Unsubscribe](#) - [Unsubscribe Preferences](#)

# CQRS

The screenshot shows a GraphQL playground interface with the following details:

**Operation:**

```
query GetUserProfiles {
  getUserProfiles {
    user_id
    first_name
    last_name
    email
    pronouns
    courses
    created_at
    updated_at
  }
}
```

**Variables:**

**Headers:**

**Pre-Operation Script:**

**Post-Operation Script:**

**Response:**

```
{
  "data": {
    "getUserProfiles": [
      {
        "user_id": "azs2117",
        "first_name": "Amelie",
        "last_name": "Scheil",
        "email": "azs2117@barnard.edu",
        "pronouns": "",
        "courses": [
          "Advanced Programming",
          "AHISBC3960_001_2025_1 - SENIOR RESEARCH SEMINAR",
          "Art History II",
          "Behavioral Research Method Analysis",
          "Beyond El Dorado",
          "Clothing",
          "Cloud Computing",
          "Computational Linguistics",
          "Computer Science Theory",
          "Computing in Context",
          "COMSW1404_006_2022_1 - EMERGING SCHOLARS PROG SEMINAR",
          "COMSWA4170_001_2025_1 - USER INTERFACE DESIGN",
          "Data Structures",
          "Discrete Mathematics",
          "Earth Resources & Sustainable Development",
          "Elementary German"
        ]
      }
    ]
  }
}
```

200 272ms 5.8KB

# Link to Repositories, UI Demo, and Presentation

- [Presentation Slides](#)
- [UI Demo](#)
- [StudyLink Project Repo](#)
  - [Project Dashboard](#)
- [User Interface](#)
- Microservices
  - [User Microservice](#)
  - [Course Enrollment Microservice](#)
  - [Study Group Microservice](#)
  - [Chat Microservice](#)
  - [Composite Microservice](#)
- [Data Store](#)

## Contributions

Team Member	Contribution
Amelie	<ul style="list-style-type: none"><li>● User Microservice<ul style="list-style-type: none"><li>○ REST functionality</li><li>○ HATEOAS links</li><li>○ Middleware logging</li><li>○ Correlation ID</li><li>○ Communication with Courseworks through API</li></ul></li><li>● CQRS</li><li>● Domains Model</li></ul>
Emanuela	<ul style="list-style-type: none"><li>● User Interface</li><li>● API Gateway</li><li>● Elastic IPs</li><li>● Google Login</li><li>● JWT Tokens</li><li>● FaaS</li><li>● UI on Google Storage</li></ul>
Jeannie	<ul style="list-style-type: none"><li>● Created All Initial Github Repos</li><li>● Created Github Projects Dashboard</li><li>● Created Course Enrollment Microservice<ul style="list-style-type: none"><li>○ REST functionality</li><li>○ HATEOAS links</li><li>○ Middleware logging</li><li>○ Correlation ID</li></ul></li></ul>

	<ul style="list-style-type: none"> <li>○ Communication with Courseworks through API</li> <li>● Github action that deploys microservice on commit</li> <li>● New Use Case diagram</li> </ul>
Jessica	<ul style="list-style-type: none"> <li>● Study groups microservice with: <ul style="list-style-type: none"> <li>○ Logging</li> <li>○ Tracing</li> <li>○ Correlation ID throughout REST API calls</li> <li>○ Docker container on VM</li> </ul> </li> <li>● Finalized resource diagram</li> </ul>
Jonathan	<ul style="list-style-type: none"> <li>● Chat Microservice <ul style="list-style-type: none"> <li>○ Rest functionality</li> <li>○ Logging/Tracing</li> <li>○ Pagination</li> <li>○ Asynchronous API</li> <li>○ Correlation ID propagation</li> </ul> </li> <li>● Application Deployment <ul style="list-style-type: none"> <li>○ GCP Instance</li> </ul> </li> </ul>
Sumya	<ul style="list-style-type: none"> <li>● Composite Microservice <ul style="list-style-type: none"> <li>○ Rest Functionality</li> <li>○ Synchronous call</li> <li>○ Async/parallel/rollback call</li> <li>○ Choreography (coded)</li> <li>○ OrchestrationApplication Deployment</li> <li>○ Cloud Logging</li> </ul> </li> <li>● PaaS Deployment</li> </ul>