
TEDxSCHOOL 3
AWS LAMBDA FUNCTIONS
Rota Leonardo 1086029
Bonomelli Pietro 1087035
1

Introduzione

Prima di procedere con la funzione Get_Watch_Next_by_Idx è necessario affrontare un problema.

Problema

I metadati standard (titolo, autore) non bastano per un consiglio didattico efficace.

Soluzione

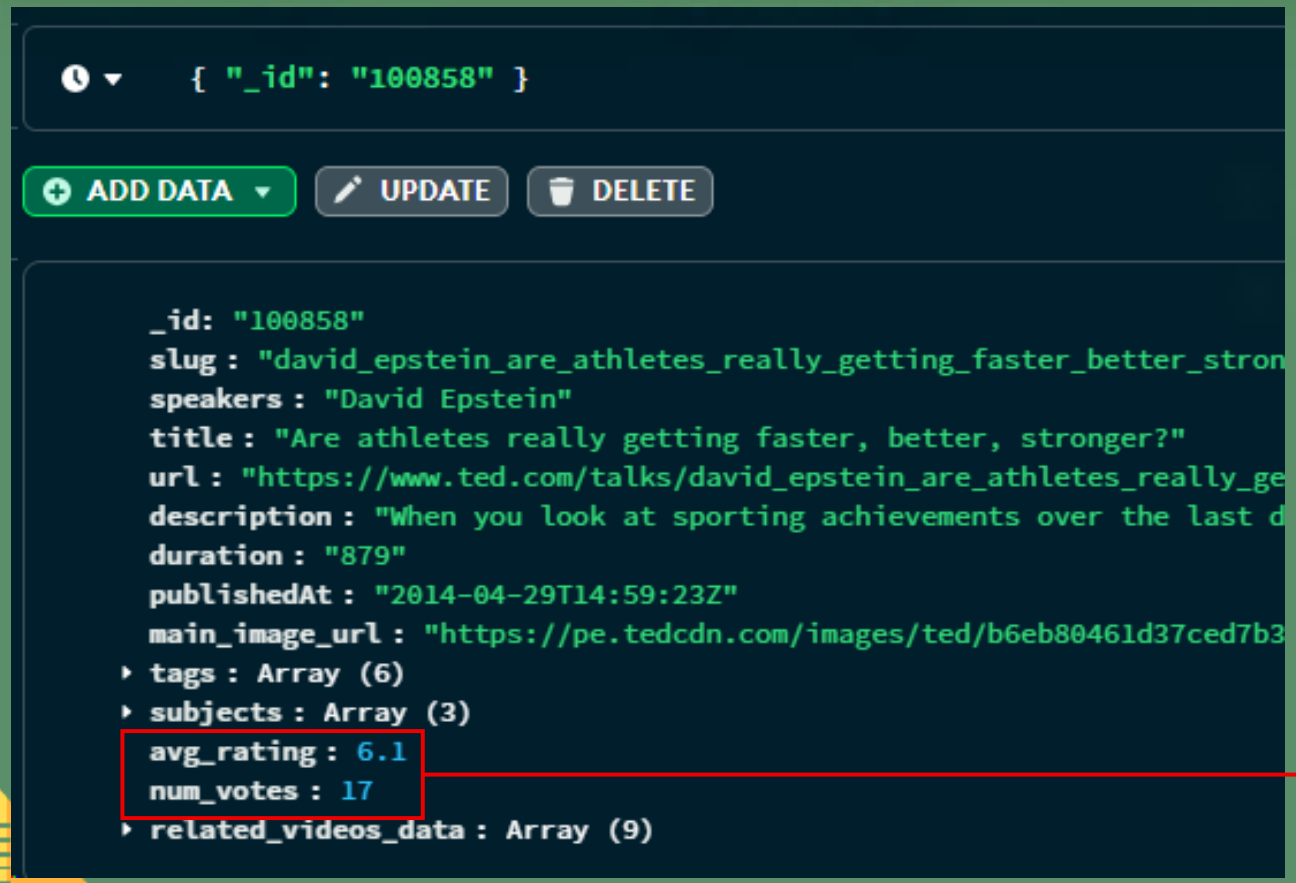
Abbiamo potenziato la collezione MongoDB e aggiunto una nuova funzionalità.

Obiettivi

- Trasformare un database statico in un sistema dinamico che apprende dai feedback dei professori.
- Permettere ai professori di votare l'efficacia didattica di un video (scala 1-10). In questo modo il ranking non è basato su algoritmi commerciali, ma sul valore educativo percepito dai professori.

(Per i fini del progetto, abbiamo generato i voti in modo casuale).

Nuova struttura dati necessaria



- **avg_rating:** Il voto medio assegnato dalla comunità dei docenti.
- **num_votes:** Il numero totale di valutazioni ricevute.

1° Lambda Function: Submit_Rating

Codice completo della LF su GitHub

```python
import json
from pymongo import MongoClient

MONGO_URI = "mongodb+srv://unibg2025:unibg2025@mycluster.mpofoqq.mongodb.net/"
client = MongoClient(MONGO_URI)
db = client.unibg_tedx_2025
collection = db.tedx_clean_data_2

def lambda_handler(event, context):
    try:
        # Parsing dei dati in ingresso (ID del video e Voto)
        body = json.loads(event.get('body', '{}'))
        talk_id = body.get('id')
        nuovo_voto = body.get('rating') # Un numero da 1 a 10

        if not talk_id or nuovo_voto is None:
            return {"statusCode": 400, "body": json.dumps("Dati mancanti (id o rating)")}

        if nuovo_voto < 1 or nuovo_voto > 10:
            return {"statusCode": 400, "body": json.dumps("Il voto deve essere compreso tra 1 e 10")}

        # Recupero dati attuali
        talk = collection.find_one({"_id": talk_id})
        if not talk:
            return {"statusCode": 404, "body": json.dumps("Talk non trovato")}

        avg_vecchia = talk.get('avg_rating', 0)
        voti_vecchi = talk.get('num_votes', 0)

        # Calcolo della Nuova Media
        nuovi_voti_totali = voti_vecchi + 1
        nuova_avg = ((avg_vecchia * voti_vecchi) + nuovo_voto) / nuovi_voti_totali
        nuova_avg = round(nuova_avg, 1) # Arrotondiamo a una cifra decimale
```

Esempio di POST API di un professore quando valuta un Talk

POST https://ctfug5w42c.execute-api.us-east-1.amazonaws.com/prod/Submit_Rating

Docs | Params | Authorization | Headers (9) | Body • | Scripts • | Settings

none | form-data | x-www-form-urlencoded | raw | binary | GraphQL | JSON

```
{
  "id": "100858",
  "rating": 10
}
```

ID del video

Voto

Body | Cookies | Headers (8) | Test Results (1/1)

{} JSON | Preview | Visualize

```
{
    "message": "Voto registrato con successo",
    "vecchia_avg": 6.1,
    "new_avg": 6.3,
    "total_votes": 18
}
```

Il *rating* viene aggiornato correttamente calcolando la nuova media, e *total_votes* viene incrementato di 1.

Modello di Analisi: Come scegliamo cosa consigliare?

Adesso che abbiamo introdotto la funzione Submit_Rating possiamo procedere con la funzione richiesta: *Get_Watch_Next_by_Idx.*

Per offrire un consiglio coerente, abbiamo stabilito una gerarchia di importanza:

1. **Affinità Didattica** (Priorità 1): Confronto tra le materie del video attuale e dei video correlati.
2. **Qualità Validata** (Priorità 2): A parità di materia, il sistema premia il video con l'avg_rating più alto.
3. **Dettaglio Tematico** (Priorità 3): Utilizzo dei tags comuni per affinare la selezione finale.

Esempio: se l'utente sta guardando un video di Matematica, il sistema cercherà prima altri video di Matematica, mettendo in cima quelli più apprezzati dagli altri professori.

2° Lambda Function: Get_Watch_Next_by_Idx

Codice completo della LF su GitHub

```python
import json
import os
from pymongo import MongoClient

MONGO_URI = "mongodb+srv://unibg2025:unibg2025@mycluster.mpofoqq.mongodb.net/"
client = MongoClient(MONGO_URI)
db = client.unibg_tedx_2025
collection = db.tedx_clean_data

def lambda_handler(event, context):
    try:
        # Recupero ID dall'URL (es: /watchnext?id=100858)
        talk_id = event.get('queryStringParameters', {}).get('id')

        if not talk_id:
            return {"statusCode": 400, "body": json.dumps("ID mancante")}

        # Recupero il talk principale
        current_talk = collection.find_one({"_id": talk_id})
        if not current_talk:
            return {"statusCode": 404, "body": json.dumps("Talk non trovato")}

        related_list = current_talk.get('related_videos_data', [])

        # Recuperiamo i voti aggiornati e le materie dei correlati
        related_ids = [v['id'] for v in related_list]
        full_related_docs = list(collection.find({"_id": {"$in": related_ids}}))

        # IL MODELLO DI ANALISI
        current_subjects = set(current_talk.get('subjects', []))
        current_tags = set(current_talk.get('tags', []))

        results = []
        for doc in full_related_docs:
            doc_subjects = set(doc.get('subjects', []))
            doc_tags = set(doc.get('tags', []))

            # Calcolo Score
            # Punteggio 1: Materie in comune
            subject_score = len(current_subjects.intersection(doc_subjects))
            # Punteggio 2: Voto dei Professori (avg_rating creato dal job Glue)
            rating_score = doc.get('avg_rating', 0)
            # Punteggio 3: Tag in comune
            tag_score = len(current_tags.intersection(doc_tags))
```

Esempio di GET API.

GET https://umghcgya0m.execute-api.us-east-1.amazonaws.com/prod/watchnext?id=100858

Docs | Params • | Authorization | Headers (7) | Body | Scripts • | Settings

Query Params

Key	Value
id	100858
Key	Value

Body | Cookies | Headers (8) | Test Results (1/1)

{} JSON | Preview | Visualize

Il primo video è quello più adatto.


```
[
  {
    "id": "4603",
    "title": "Why did I ski to the North Pole?",
    "speaker": null,
    "image": "https://pe.tedcdn.com/images/ted/172_480x360.jpg",
    "subjects": [
      "educazione fisica",
      "geopolitica",
      "scienze",
      "educazione civica",
      "tecnologia"
    ],
    "rating": 3.4,
    "scores": {
      "subject_match": 3,
      "tag_match": 1
    },
    "reason": "Scelto perché affine a tecnologia"
  },
  {
    "id": "21389",
    "title": "My 12 pairs of legs",
```


Media dei voti dei professori

Materie affini

Ragione della scelta

3° Lambda Function: Assign_Video

- Come funzionalità aggiuntiva a scelta, abbiamo deciso di implementare un modulo di Gestione Classroom.
  L'obiettivo è permettere ai docenti di creare un ponte diretto con i propri studenti, assegnando i talk TEDx come compiti o approfondimenti mirati.

- Per la struttura dati, abbiamo popolato su MongoDB Atlas una collezione chiamata classrooms. (GitHub)
  La classe è stata concepita come un'entità condivisa in cui possono operare più professori contemporaneamente. Nel documento della classe, ogni compito assegnato è un oggetto che contiene l'ID del video, la materia e l'email del docente.

- Il tutto viene gestito dalla funzione Lambda Assign_Video. Questa funzione riceve dall'applicazione i dettagli del compito e interagisce con il database tramite l'operatore atomico $push. Questo ci permette di assegnare nuovi talk all'array dei compiti garantendo integrità dei dati anche in caso di accessi simultanei da parte di professori diversi.

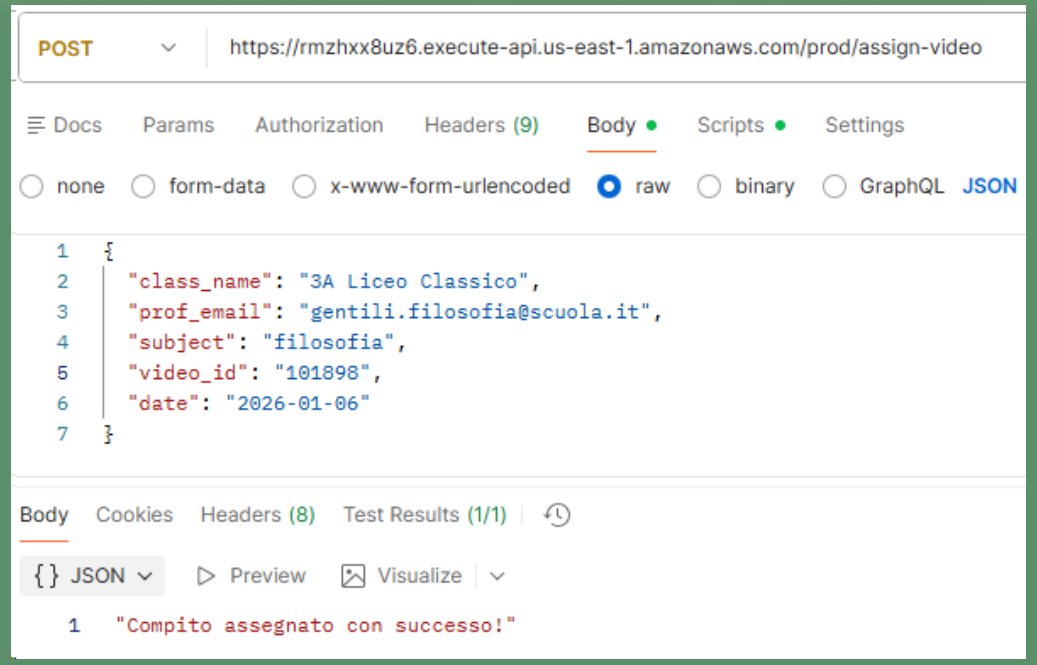
- L'interazione avviene tramite un'API POST dedicata, esposta su AWS API Gateway, di cui vedremo un esempio nella prossima slide.

3° Lambda Function: Assign_Video

Codice completo della LF su GitHub

```python
import json
from pymongo import MongoClient

MONGO_URI = "mongodb+srv://unibg2025:unibg2025@mycluster.mpofoqq.mongodb.net/"
client = MongoClient(MONGO_URI)
db = client.unibg_tedx_2025
collection = db.classrooms

def lambda_handler(event, context):
    try:
        # Recupero dati dal body della richiesta (da Flutter)
        body = json.loads(event.get('body', '{}'))
        class_name = body.get('class_name') # "3A Liceo Classico"
        prof_email = body.get('prof_email')
        subject = body.get('subject')
        video_id = body.get('video_id')
        date = body.get('date', "2026-01-06") # Default se manca

        if not all([class_name, video_id, prof_email]):
            return {"statusCode": 400, "body": json.dumps("Dati mancanti")}

        # Aggiornamento su MongoDB tramite $push
        result = collection.update_one(
            {"name": class_name},
            {
                "$push": {
                    "assignments": {
                        "prof_email": prof_email,
                        "subject": subject,
                        "video_id": video_id,
                        "date": date
                    }
                }
            }
        )
```

Esempio di POST API: il professore assegna un compito di filosofia.



Criticità tecniche

- Durante lo sviluppo si è riscontrata una criticità legata al timeout predefinito di AWS Lambda. L'esecuzione falliva dopo 3 secondi a causa della latenza necessaria per l'inizializzazione della connessione con il cluster MongoDB Atlas. Il problema è stato risolto aumentando il timeout a 10 secondi.

- Una limitazione tecnica rilevante è emersa nella gestione delle librerie non native, come PyMongo. Poiché l'ambiente di runtime di Lambda non include driver per database esterni, è stato necessario implementare dei Deployment Packages manuali.

- L'attuale algoritmo di raccomandazione potrebbe consigliare video che l'utente ha già visto.

Possibili evoluzioni

- Il sistema è predisposto per integrare funzionalità di Adaptive Learning. Incrociando i dati sulla difficoltà dei video con i feedback dei docenti, la piattaforma potrebbe suggerire percorsi di studio personalizzati.
- Notificare gli studenti quando il professore aggiunge un compito.
- Una soluzione al problema delle raccomandazioni a video che l'utente ha già visto: consiste nell'introdurre una gestione dello stato della sessione per tracciare i contenuti già visualizzati o suggeriti, garantendo una navigazione sempre varia.


GitHub
Trello
Rota Leonardo 1086029
Bonomelli Pietro 1087035
11