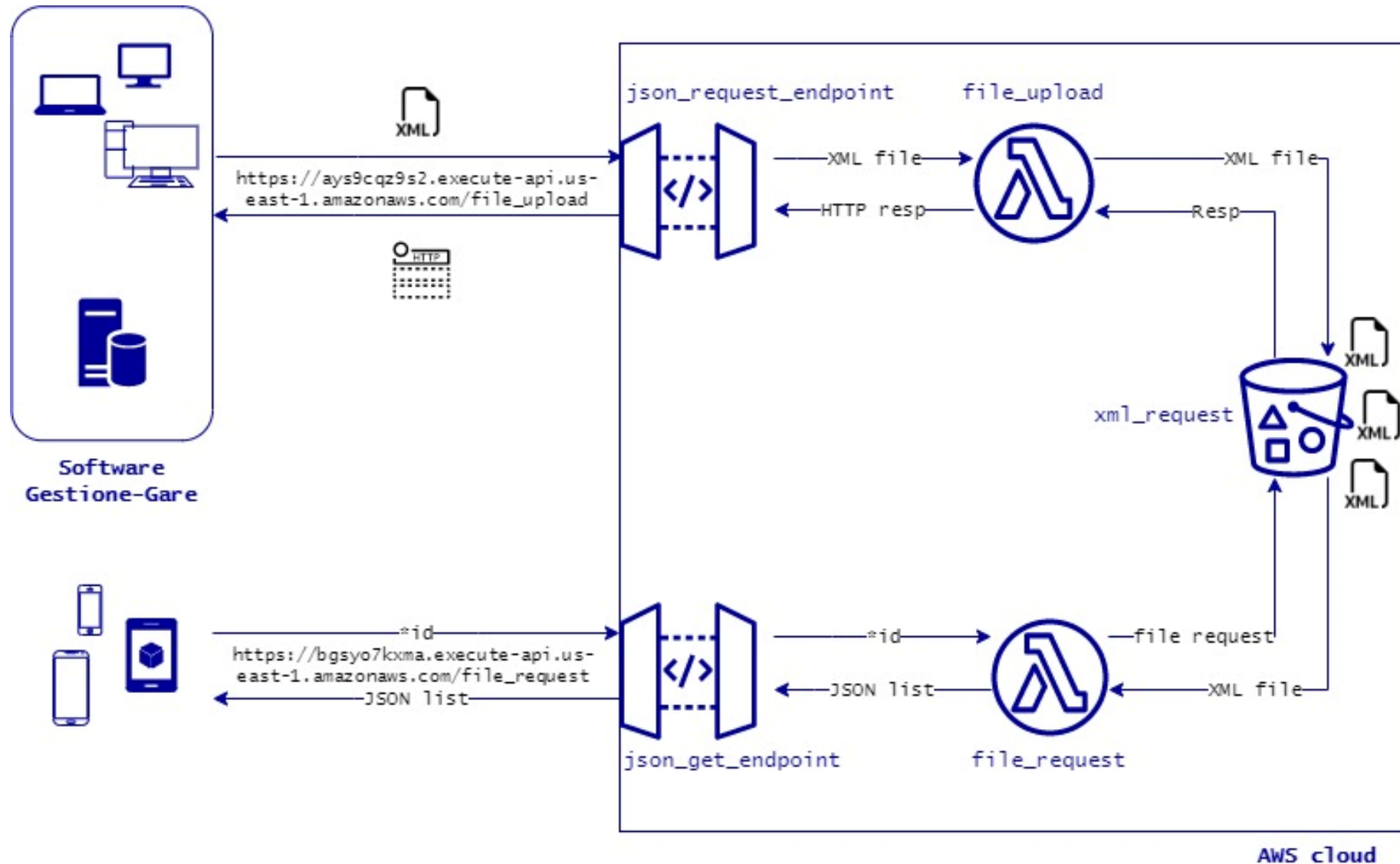




Progetto TCM

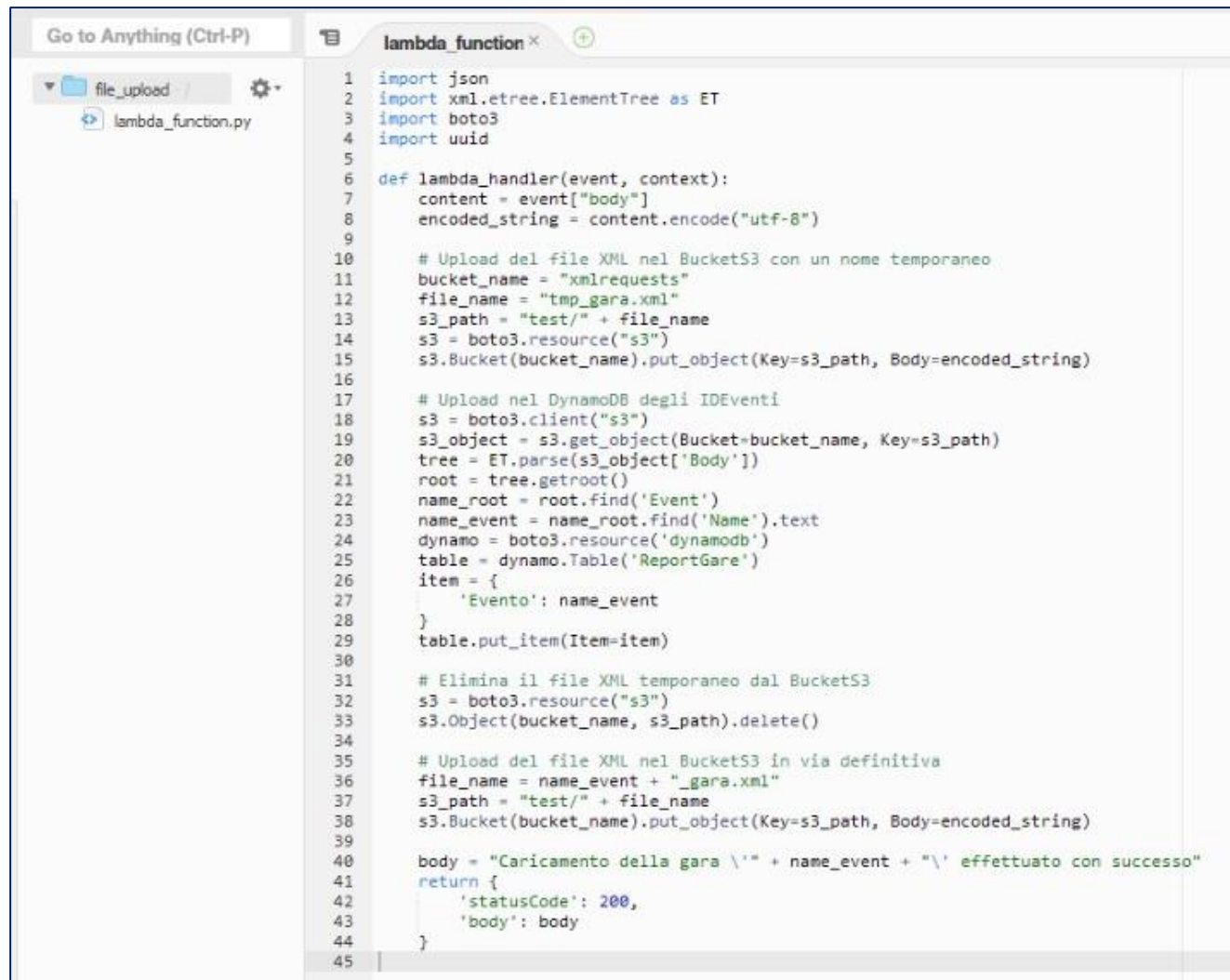
<i>Giuseppe Daidone</i>	- 1068102
<i>Benedetta Lenuzza</i>	- 1068745
<i>Gloria Pasinetti</i>	- 1066654
<i>Nicolò Zaffaroni</i>	- 1065978

Architettura cloud



File Upload (1)

Lambda function



The screenshot shows a code editor with a file explorer on the left and a code editor on the right. The file explorer shows a folder named 'file_upload' containing a file named 'lambda_function.py'. The code editor shows the following Python code:

```
1 import json
2 import xml.etree.ElementTree as ET
3 import boto3
4 import uuid
5
6 def lambda_handler(event, context):
7     content = event["body"]
8     encoded_string = content.encode("utf-8")
9
10    # Upload del file XML nel BucketS3 con un nome temporaneo
11    bucket_name = "xmlrequests"
12    file_name = "tmp_gara.xml"
13    s3_path = "test/" + file_name
14    s3 = boto3.resource("s3")
15    s3.Bucket(bucket_name).put_object(Key=s3_path, Body=encoded_string)
16
17    # Upload nel DynamoDB degli IDEventi
18    s3 = boto3.client("s3")
19    s3_object = s3.get_object(Bucket=bucket_name, Key=s3_path)
20    tree = ET.parse(s3_object['Body'])
21    root = tree.getroot()
22    name_root = root.find('Event')
23    name_event = name_root.find('Name').text
24    dynamo = boto3.resource('dynamodb')
25    table = dynamo.Table('ReportGara')
26    item = {
27        'Evento': name_event
28    }
29    table.put_item(Item=item)
30
31    # Elimina il file XML temporaneo dal BucketS3
32    s3 = boto3.resource("s3")
33    s3.Object(bucket_name, s3_path).delete()
34
35    # Upload del file XML nel BucketS3 in via definitiva
36    file_name = name_event + "_gara.xml"
37    s3_path = "test/" + file_name
38    s3.Bucket(bucket_name).put_object(Key=s3_path, Body=encoded_string)
39
40    body = "Caricamento della gara \" + name_event + "\" effettuato con successo"
41    return {
42        'statusCode': 200,
43        'body': body
44    }
45
```

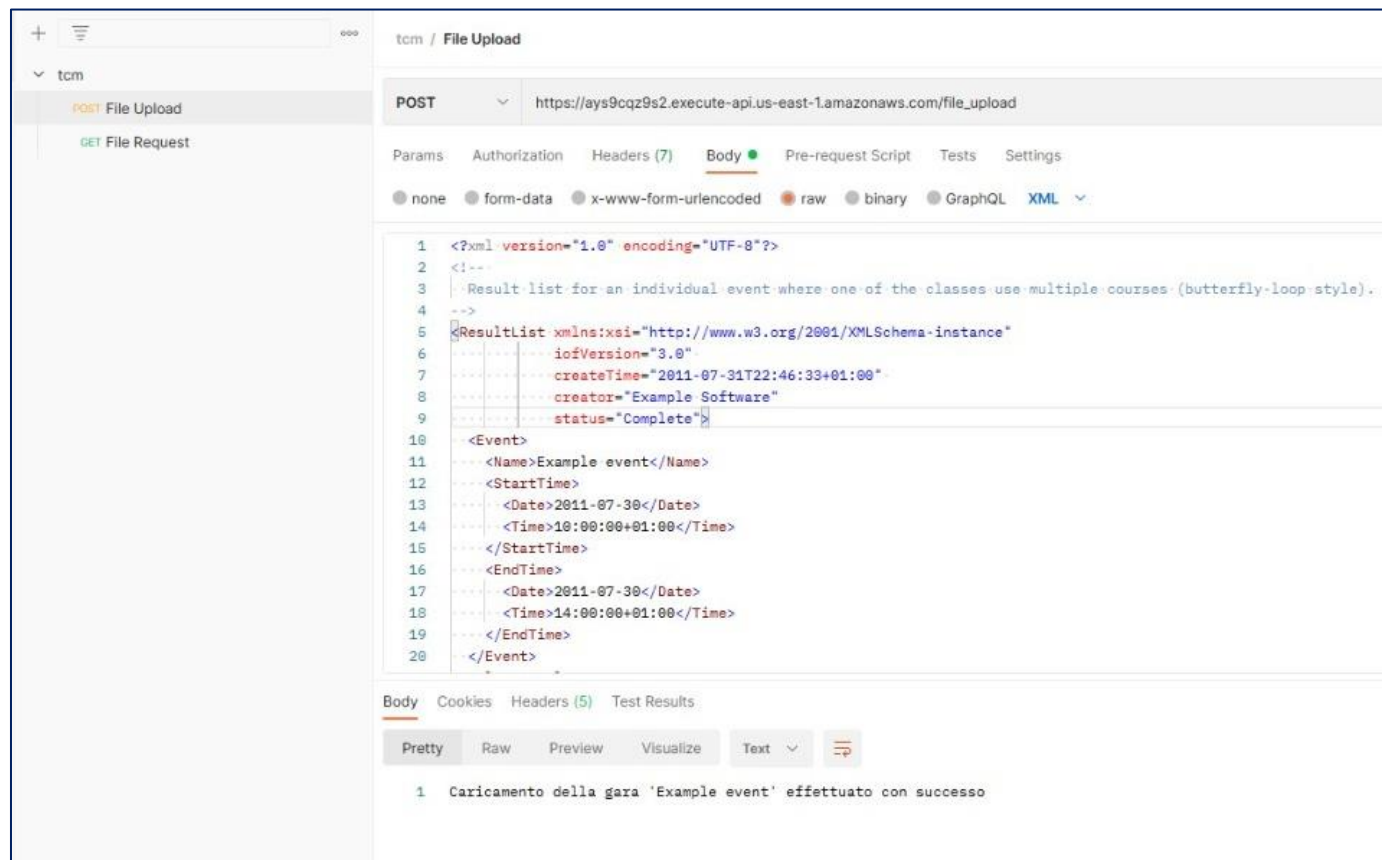
File Upload (2)

Postman

Per caricare un file all'interno del S3 Bucket dobbiamo interfacciarci tramite un Gateway ad una Lambda function, ovvero lambda file_upload.

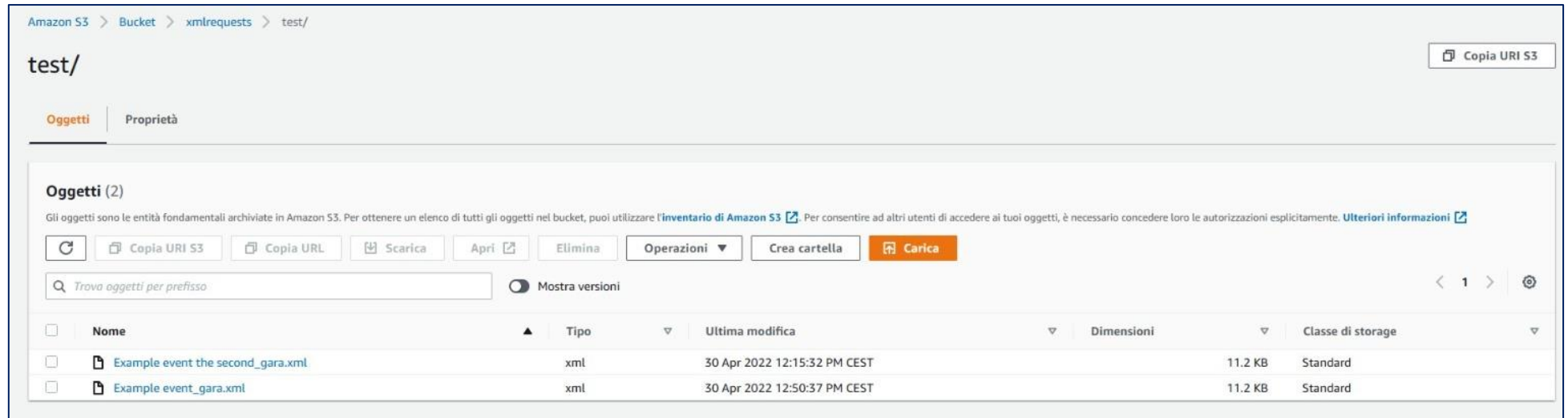
Usiamo Postman per fare una richiesta HTTP all'endpoint API (ovvero Gateway), usando il metodo Post e aggiungendo al body del pacchetto HTTP il file xml.

Se l'operazione avviene con successo viene rimandata una risposta di feedback «Caricamento della gara **nome gara** effettuato con successo».



Caricamento file

1) Bucket



Amazon S3 > Bucket > xmlrequests > test/

test/ Copia URI S3

Oggetti Proprietà

Oggetti (2)

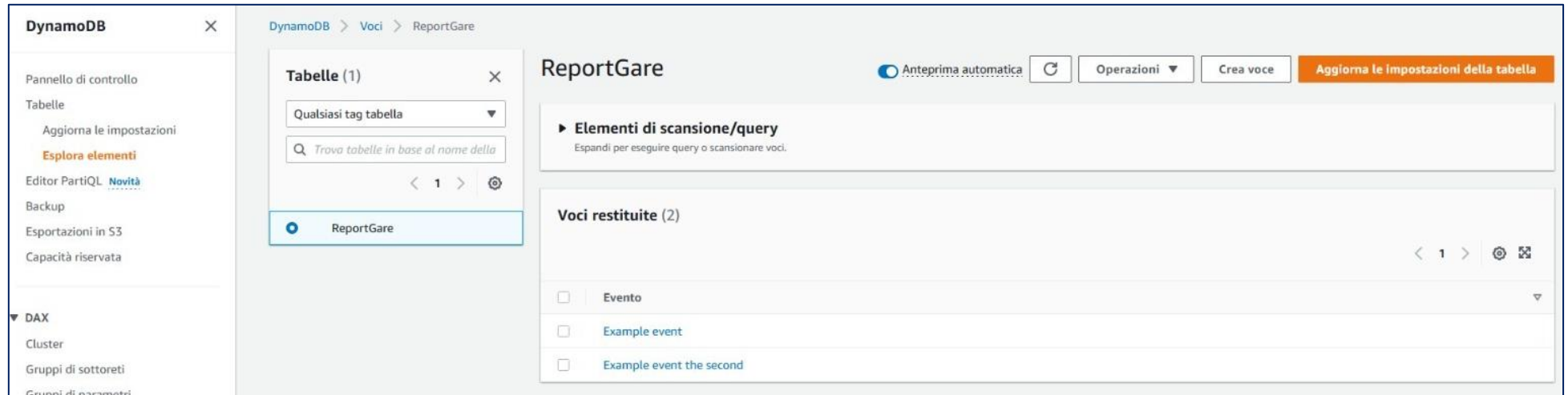
Gli oggetti sono le entità fondamentali archiviate in Amazon S3. Per ottenere un elenco di tutti gli oggetti nel bucket, puoi utilizzare l'[inventario di Amazon S3](#). Per consentire ad altri utenti di accedere ai tuoi oggetti, è necessario concedere loro le autorizzazioni esplicitamente. [Ulteriori informazioni](#)

🔄 Copia URI S3 Copia URL 📄 Scarica 🔗 Apri Elimina Operazioni ▼ Crea cartella Carica

Mostra versioni < 1 > ⚙️

<input type="checkbox"/>	Nome	Tipo	Ultima modifica	Dimensioni	Classe di storage
<input type="checkbox"/>	Example event the second_gara.xml	xml	30 Apr 2022 12:15:32 PM CEST	11.2 KB	Standard
<input type="checkbox"/>	Example event_gara.xml	xml	30 Apr 2022 12:50:37 PM CEST	11.2 KB	Standard

2) DinamoDB



DynamoDB

Pannello di controllo
Tabelle
Aggiorna le impostazioni
[Esplora elementi](#)
Editor PartiQL [Novità](#)
Backup
Esportazioni in S3
Capacità riservata

▼ DAX
Cluster
Gruppi di sottoreti
Gruppi di parametri

DynamoDB > Voci > ReportGare

Tabelle (1)

Qualsiasi tag tabella

< 1 > ⚙️

ReportGare

ReportGare Anteprima automatica 🔄 Operazioni ▼ Crea voce Aggiorna le impostazioni della tabella

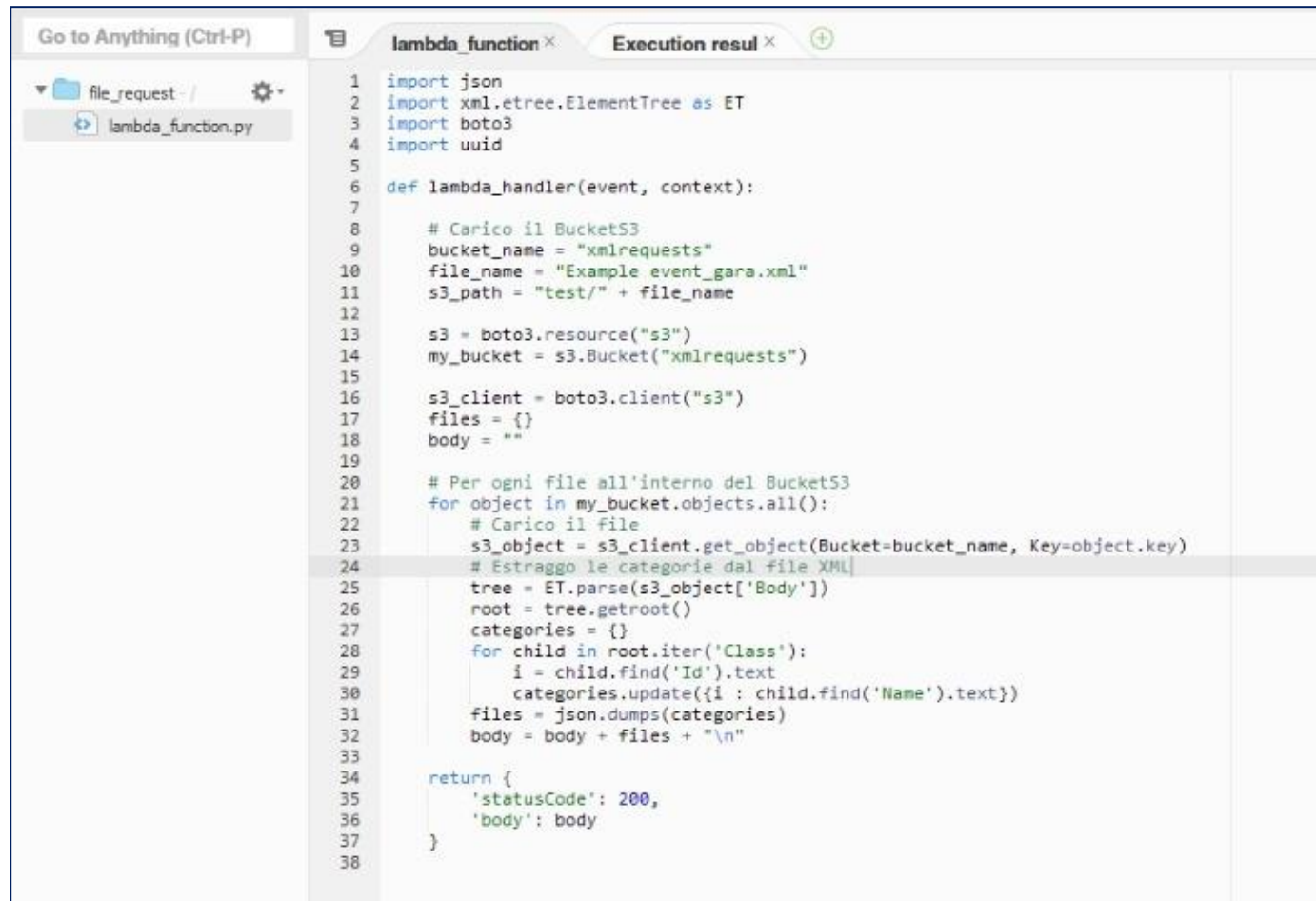
► **Elementi di scansione/query**
Espandi per eseguire query o scansionare voci.

Voci restituite (2) < 1 > ⚙️ 🔗

<input type="checkbox"/>	Evento
<input type="checkbox"/>	Example event
<input type="checkbox"/>	Example event the second

File Request ⁽¹⁾

Lambda function



```
Go to Anything (Ctrl-P)  lambda_function x  Execution result x +
file_request /
lambda_function.py

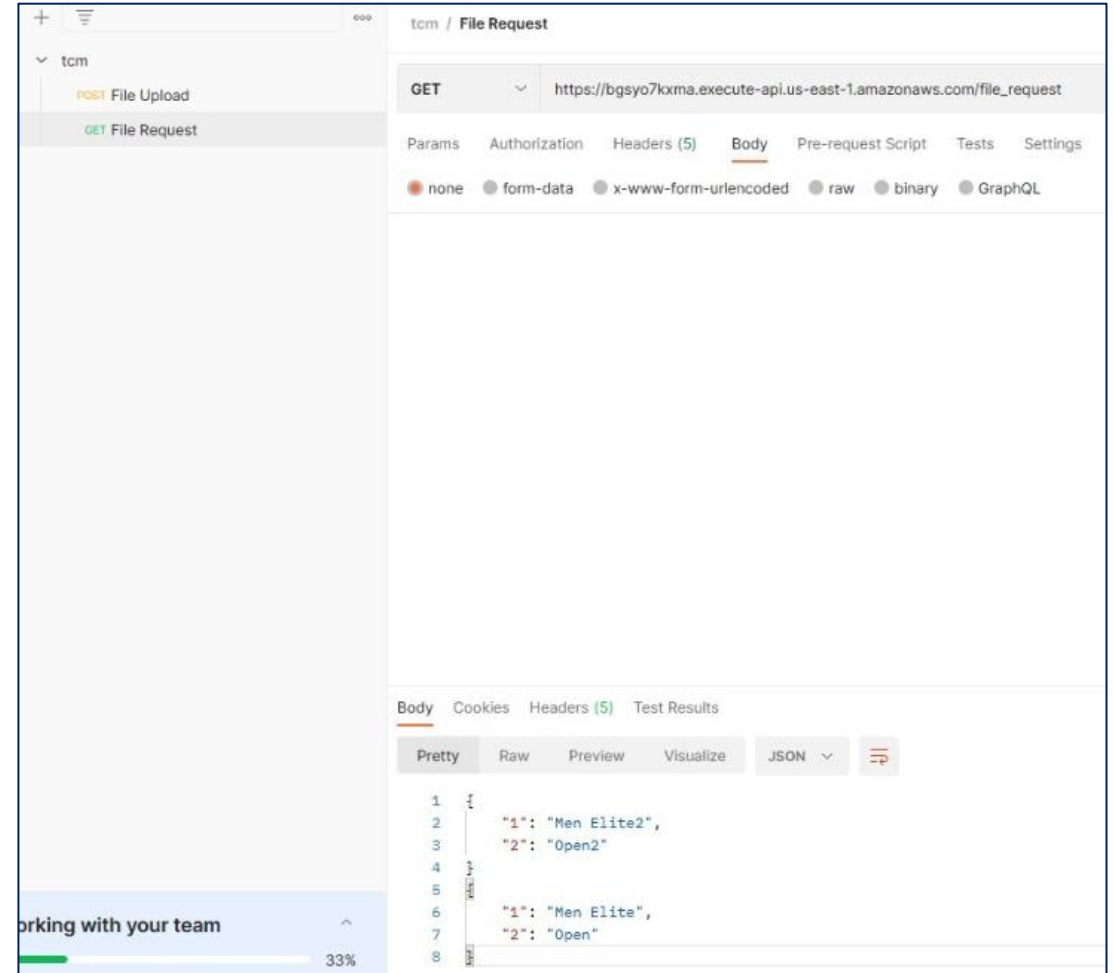
1 import json
2 import xml.etree.ElementTree as ET
3 import boto3
4 import uuid
5
6 def lambda_handler(event, context):
7
8     # Carico il BucketS3
9     bucket_name = "xmlrequests"
10    file_name = "Example_event_gara.xml"
11    s3_path = "test/" + file_name
12
13    s3 = boto3.resource("s3")
14    my_bucket = s3.Bucket("xmlrequests")
15
16    s3_client = boto3.client("s3")
17    files = {}
18    body = ""
19
20    # Per ogni file all'interno del BucketS3
21    for object in my_bucket.objects.all():
22        # Carico il file
23        s3_object = s3_client.get_object(Bucket=bucket_name, Key=object.key)
24        # Estraggo le categorie dal file XML
25        tree = ET.parse(s3_object['Body'])
26        root = tree.getroot()
27        categories = {}
28        for child in root.iter('Class'):
29            i = child.find('Id').text
30            categories.update({i : child.find('Name').text})
31        files = json.dumps(categories)
32        body = body + files + "\n"
33
34    return {
35        'statusCode': 200,
36        'body': body
37    }
38
```

File Request (2)

Postman

Per acquisire le categorie di tutte le gare (ovvero classi), salvate all'interno del S3 Bucket, utilizziamo Postman. Esso ci permette di fare una richiesta HTTP all'endpoint API (ovvero Gateway) della Lambda function, in questo caso lambda file_request, usando il metodo Get.

Otteniamo una risposta contenente un file JSON di tutte le classi caricate fino a quel momento nel S3 Bucket.



Gestione dell'accesso di utenti non autorizzati

Ipotizziamo di essere in un contesto reale, in cui le richieste HTTP vengono effettuate da un'applicazione web. Essa è fruibile da chiunque, in particolare viene utilizzata dai commissari di gara per il caricamento dei report delle gare, mentre dagli utenti per controllare i risultati.

Conseguentemente, la gestione degli accessi diviene fondamentale, in modo tale da permettere il caricamento dei risultati solo ai commissari e non ad un utente qualsiasi.

Per farlo, definiremo e gestiremo due tipi differenti di profili all'interno dell'applicazione:

- **UTENTE** La sua unica autorizzazione è quella di poter leggere i risultati. È un profilo creabile in qualsiasi momento, da qualsiasi persona, richiedendo solo pochi dati personali necessari e sufficienti.
- **AMMINISTRATORE** Possiede sia i privilegi di un profilo **UTENTE**, sia l'autorizzazione di poter caricare e rimuovere report di gare. Ogni profilo di questo tipo è dichiarato a priori dal sistema centrale di gestione dell'applicazione web, il quale gestisce tutti i profili. In questo modo non è possibile che utenti non autorizzati modifichino i risultati caricati senza autorizzazione.

Per ogni sessione, l'applicazione web richiede il login attraverso username e password in modo da riconoscere univocamente la tipologia di profilo, permettendo eventualmente la creazione di un nuovo profilo **UTENTE** o concedendo autorizzazioni ad un profilo **AMMINISTRATORE**.

Inoltre, abbiamo ipotizzato di poter filtrare le gare, dichiarandone il nome in quanto univoco, presentando al sistema richieste differenti, come indicato da **id* nell'architettura.

