Progetto TCM

Giuseppe Daidone - *1068102*
Benedetta Lenuzza - *1068745*
Gloria Pasinetti - *1066654*
Nicolò Zaffaroni - *1065978*

Register Race(1)

Lambda function

Un utente fa una chiamata POST all'endpoint di **`register_race`**, inserendo negli headers:

- **`race_name`** Nome della gara.
- **`race_date`** Data della gara.
- **`race_place`** Luogo della gara.
- **`email`** E-mail dell'utente che ha effettuato il caricamento.

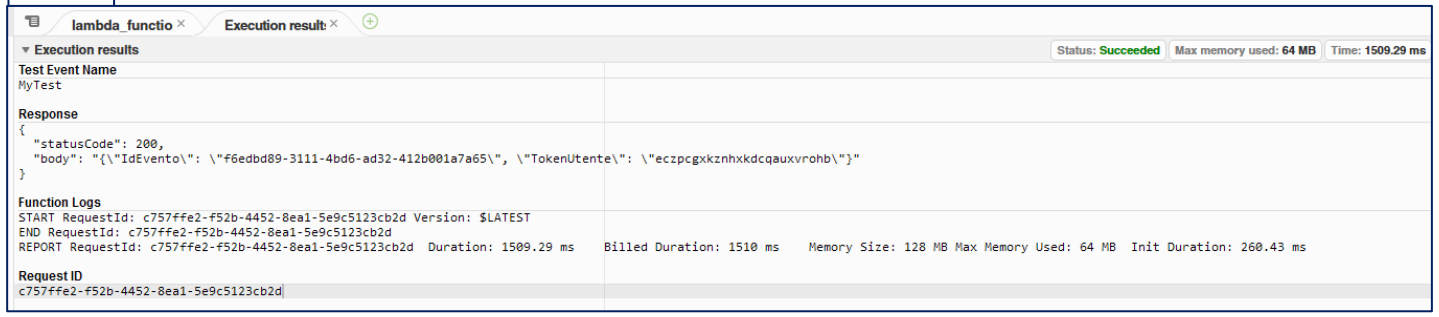
L'utente riceve un JSON di risposta, composto da:

- **`race_id`** Identificativo univoco della gara.
- **`token`** Identificativo univoco per ogni utente amministratore.

Codice

```python
import json
import xml.etree.ElementTree as ET
import boto3
import uuid
import string
import random

def lambda_handler(event, context):

    content = event["body"]
    race_name = event["headers"]["race_name"]
    race_date = event["headers"]["race_date"]
    race_place = event["headers"]["race_place"]
    email = event["headers"]["email"]
    encoded_string = content.encode("utf-8")
    assert headers_validation(race_name, race_date, race_place, email)
    assert check_if_race_exists()
    assert check_if_user_exists()

    # Genero gli identificativi di gara e utente
    i = {}
    t = {}
    r = {}
    id = str(id_gen())
    token = token_gen()

    # Upload nel DynamoDB della gara registrata
    dynamo = boto3.resource('dynamodb')
    table = dynamo.Table('GareRegistrate')
    item = {
        'Id': id,
        'RaceName': race_name,
        'RaceDate': race_date,
        'RacePlace': race_place,
        'Email': email
    }
    table.put_item(Item=item)
    r.update({"IdEvento" : id})

    # Upload nel DynamoDB del nuovo utente
    table = dynamo.Table('Amministratori')
    item = {
        'Token': token,
        'Email': email
    }
    table.put_item(Item=item)
    r.update({"TokenUtente" : token})

    # Valori di ritorno
    body = json.dumps(r)

    return {
        'statusCode': 200,
        'body': body
    }

def headers_validation(race_name, race_date, race_place, email): # Check se header non sono vuoti
    return True

def check_if_race_exists(): # Check se la gara è stata già registrata
    return True

def check_if_user_exists(): # Check se l'utente è già stato registrato
    return True

def id_gen(): # Genero id univoco
    return uuid.uuid4()

def token_gen(): # Genero un token casuale di 24 caratteri (controllare se l'utente esiste già nel DynamoDB)

    token = string.ascii_lowercase
    return ''.join(random.choice(token) for i in range(24))
```

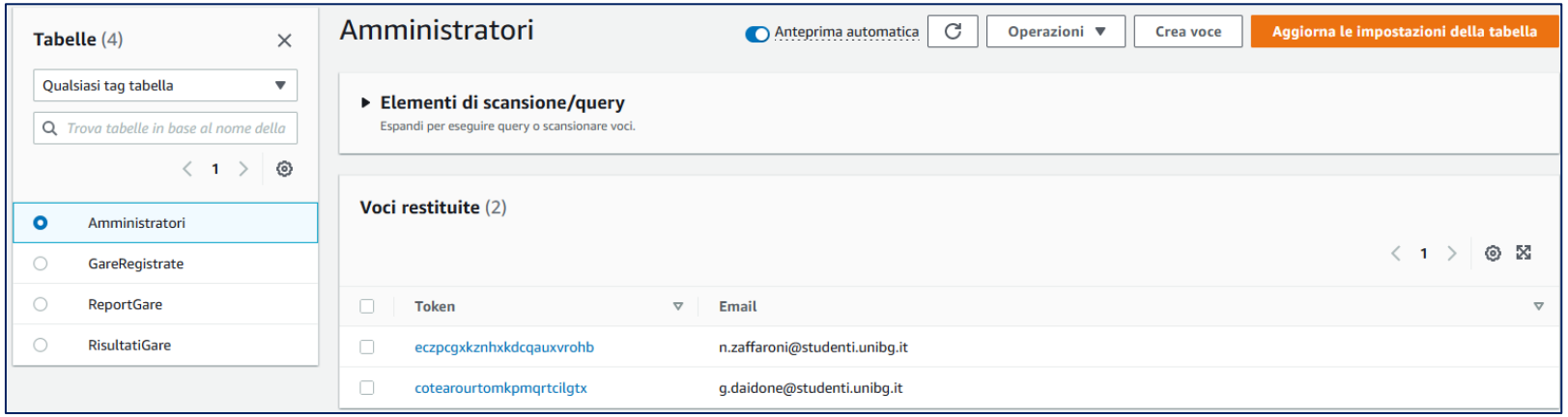
Test



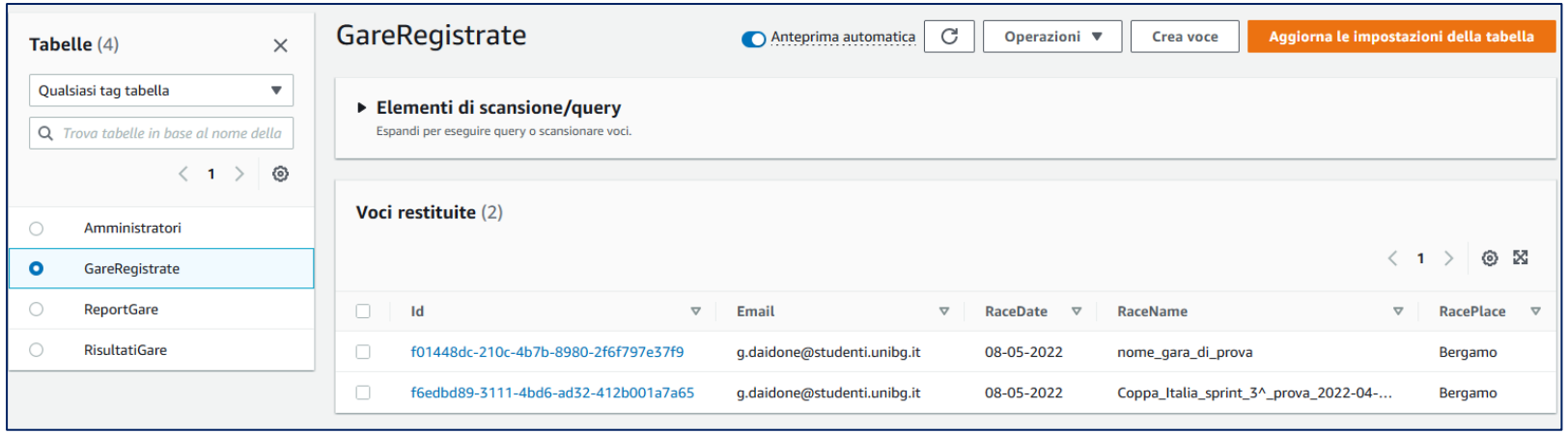
Register Race(2)

DynamoDB

- **Amministratori**:
Elenco degli amministratori registrati, con i relativi token e E-mail.



- **Gare Registrate**:
Elenco delle gare caricate, con i dati relativi (ID, data, nome, luogo) e l'E-mail dell'amministratore che ha effettuato il caricamento.



Upload XML(1)

Lambda function

Un utente fa una chiamata POST all'endpoint di **`upload_xml`**, inserendo:

- Header — Token personale da amministratore.
- Body — Codice XML della gara da caricare.

Si controlla che il formato XML rispetti lo standard. Dopo aver controllato il formato, verificato il token:

1. Carica il file XML temporaneo nel Bucket S3.
2. Carica la gara salvata in DynamoDB.
3. Carica il file XML nel bucket S3 eliminando il file temporaneo.

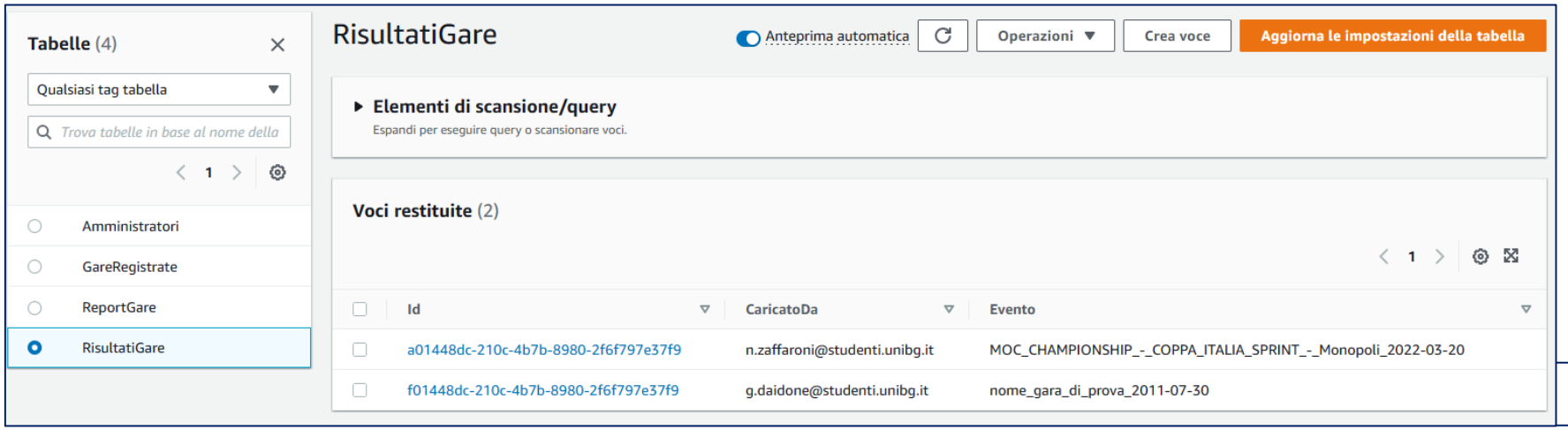
```python
import json
import xml.etree.ElementTree as ET
import boto3
import uuid

def lambda_handler(event, context):

    content = event["body"]
    id = event["headers"]["id"]
    token = event["headers"]["token"]
    encoded_string = content.encode("utf-8")
    assert check_xml(content)
    assert headers_validation(id, token)

    # Get utente dal DynamoDB tramite il suo token
    email = find_user(token)
    if(email != ""):
        # Upload del file XML nel BucketS3 con un nome temporaneo
        bucket_name = "xmlrequests"
        file_name = "tmp_gara.xml"
        s3_path = "test/" + file_name
        s3 = boto3.resource("s3")
        s3.Bucket(bucket_name).put_object(Key=s3_path, Body=encoded_string)

        # Upload nel DynamoDB della gara
        s3 = boto3.client("s3")
        s3_object = s3.get_object(Bucket=bucket_name, Key=s3_path)
        tree = ET.parse(s3_object['Body'])
        root = tree.getroot()
        name_root = root.find('Event')
        name_event = name_root.find('Name').text
        name = find_name(id, email)
        if(name == name_event):
            date_event = name_root.find('./StartTime/Date').text
            xml_name = name_event + " " + date_event
            xml_name = xml_name.replace(" ", "_")
            dynamo = boto3.resource('dynamodb')
            table = dynamo.Table('RisultatiGare')
            item = {
                'Id': id,
                'Evento': xml_name,
                'CaricatoDa': email
            }
            table.put_item(Item=item)

            # Elimina il file XML temporaneo dal BucketS3
            s3 = boto3.resource("s3")
            s3.Object(bucket_name, s3_path).delete()

            # Upload del file XML nel BucketS3 in via definitiva
            file_name = xml_name + ".xml"
            s3_path = "test/" + file_name
            s3.Bucket(bucket_name).put_object(Key=s3_path, Body=encoded_string)
            body = "Caricamento del file \'" + xml_name + ".xml\' effettuato con successo"
        else:
            # Elimina il file XML temporaneo dal BucketS3
            s3 = boto3.resource("s3")
            s3.Object(bucket_name, s3_path).delete()

            body = "La gara che si vuole caricare non è stata ancora registrata"
    else:
        body = "Utente inesistente"

    return {
        'statusCode': 200,
        'body': body
    }
```

Body

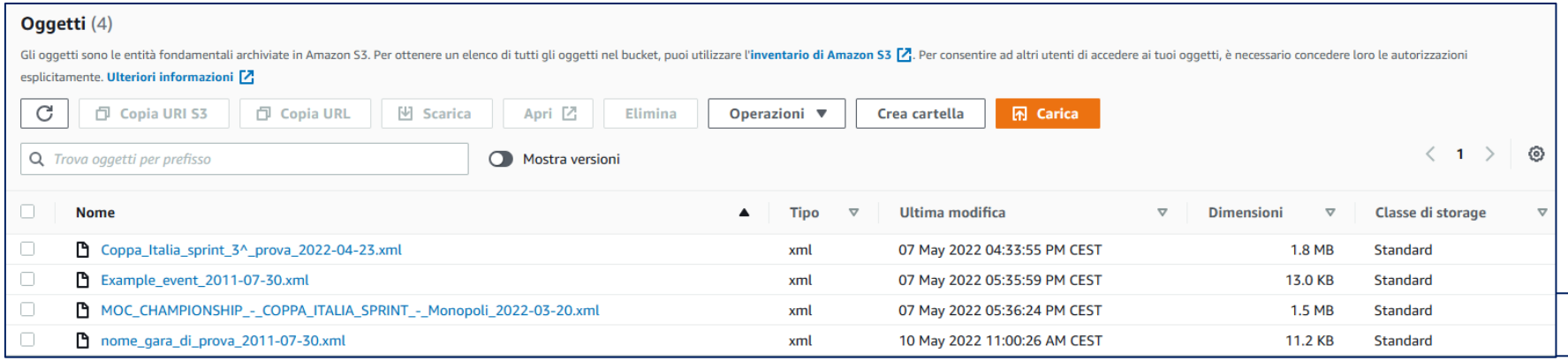
```python
def check_xml(content): # Check se rispetta l'XSD
    return True

def headers_validation(id, token): # Check se header non sono vuoti
    return True

def find_user(token): # Cerca se l'utente che sta caricando la gara è presente tra gli amministratori
    email = ""
    dynamo = boto3.resource('dynamodb')
    table = dynamo.Table('Amministratori')
    response = table.get_item(
    Key={
        'Token': token
    })
    email = response["Item"]["Email"]
    return email

def find_name(id, email): # Cerca se la gara del file XML è già stata registrata
    name = ""
    dynamo = boto3.resource('dynamodb')
    table = dynamo.Table('GareRegistrate')
    response = table.get_item(
    Key={
        'Id': id
    })
    name = response["Item"]["RaceName"]
    return name
```

Functions

Upload XML(2)



DinamoDB



Bucket S3

List Races

Lambda function

Un utente fa una chiamata GET all'endpoint di list_races.

Riceverà come risposta una lista JSON delle gare caricate fino a quel momento, con i relativi dati:

- `race_name` Nome della gara.
- `race_date` Data della gara.
- `race_place` Luogo della gara.
- `email` E-mail dell'utente che ha effettuato il caricamento.

Codice

```python
import json
import xml.etree.ElementTree as ET
import boto3
import uuid

def lambda_handler(event, context):

    content = event["body"]

    # Get nome della gara dal DynamoDB tramite il suo ID
    dynamo = boto3.resource('dynamodb')
    table = dynamo.Table('GareRegistrate')
    response = table.scan()
    resp = table.scan(ProjectionExpression="Id, RaceName, RaceDate, RacePlace")

    body = resp["Items"]

    return {
        'statusCode': 200,
        'body': body
    }
```

Test

lambda_functio × Execution result ×

Execution results — Status: Succeeded | Max memory used: 64 MB | Time: 1392.49 ms

Test Event Name

MyTest

Response


```
{
  "statusCode": 200,
  "body": [
    {
      "RaceDate": "08-05-2022",
      "RaceName": "nome_gara_di_prova",
      "Id": "f01448dc-210c-4b7b-8980-2f6f797e37f9",
      "RacePlace": "Bergamo"
    },
    {
      "RaceDate": "08-05-2022",
      "RaceName": "Coppa_Italia_sprint_3^_prova_2022-04-23.xml",
      "Id": "f6edbd89-3111-4bd6-ad32-412b001a7a65",
      "RacePlace": "Bergamo"
    }
  ]
}
```


Function Logs

```
START RequestId: b481c25d-7456-4b0d-83ac-ca30f90642e6 Version: $LATEST
END RequestId: b481c25d-7456-4b0d-83ac-ca30f90642e6
REPORT RequestId: b481c25d-7456-4b0d-83ac-ca30f90642e6  Duration: 1392.49 ms    Billed Duration: 1393 ms    Memory Size: 128 MB Max Memory Used: 64 MB  Init Duration: 272.51 ms
```

Request ID

b481c25d-7456-4b0d-83ac-ca30f90642e6

List Classes

Lambda function

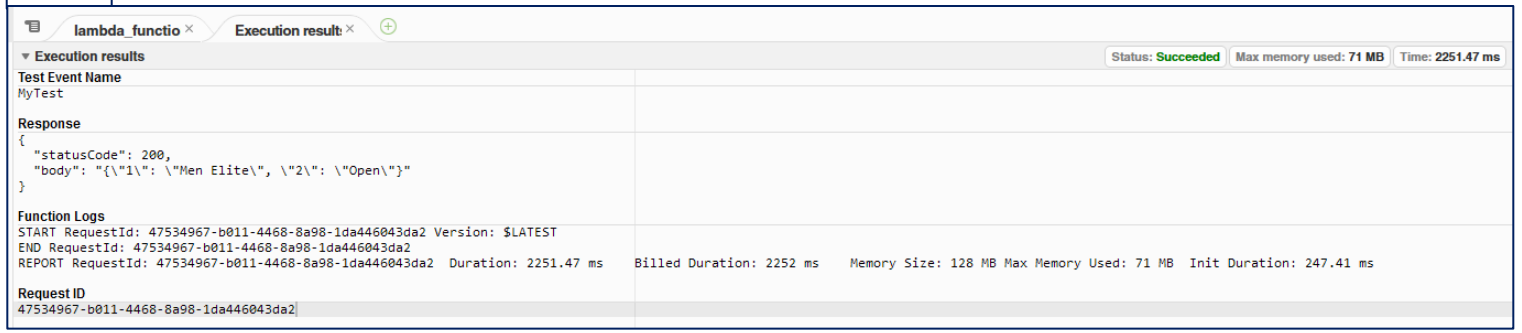
Un utente fa una chiamata GET all'endpooint di **`list_classes`**, passando come parametro l'ID relativo alla gara interessata.

Riceverà come risposta una lista JSON di tutti le categorie presenti nella gara richiesta, selezionata tramite ID unvoco.

Codice

```python
import json
import xml.etree.ElementTree as ET
import boto3
import uuid

def lambda_handler(event, context):

    content = event["body"]
    id = event['queryStringParameters']['id']

    # Get nome della gara dal DynamoDB tramite il suo ID
    dynamo = boto3.resource('dynamodb')
    table = dynamo.Table('RisultatiGare')
    response = table.get_item(
    Key={
        'Id': id
    })
    event_name = response["Item"]["Evento"]

    bucket_name = "xmlrequests"
    s3_path = "test/" + event_name + ".xml"

    # Get file XML dal BucketS3 ed estrai le categorie
    s3 = boto3.client("s3")
    s3_object = s3.get_object(Bucket=bucket_name, Key=s3_path)
    tree = ET.parse(s3_object['Body'])
    root = tree.getroot()
    categories = {}
    for child in root.iter('Class'):
        i = child.find('Id').text
        categories.update({i : child.find('Name').text})
    body = json.dumps(categories)


    return {
        'statusCode': 200,
        'body': body
    }
```

Test



Results

Lambda function

Un utente fa una chiamata GET all'endpooint di `results`, passando come parametri l'ID gara e una categoria relativa ad essa

Riceverà come risposta una lista JSON con classifica relativa alla categoria della gara specificata.

Codice

```python
import json
import xml.etree.ElementTree as ET
import boto3
import uuid

def lambda_handler(event, context):

    content = event["body"]
    id = event['queryStringParameters']['id']
    category = event['queryStringParameters']['class']

    # Get nome della gara dal DynamoDB tramite il suo ID
    dynamo = boto3.resource('dynamodb')
    table = dynamo.Table('RisultatiGare')
    response = table.get_item(
    Key={
        'Id': id
    })
    event_name = response["Item"]["Evento"]

    bucket_name = "xmlrequests"
    s3_path = "test/" + event_name + ".xml"

    # Get file XML dal BucketS3 ed estrai le categorie
    s3 = boto3.client("s3")
    s3_object = s3.get_object(Bucket=bucket_name, Key=s3_path)
    tree = ET.parse(s3_object['Body'])
    root = tree.getroot()
    c = {}
    body = "Categoria " + category + " non trovata"
    for child in root.findall("./ClassResult"):
        if(child.find("./Class/Name").text == category):
            for person in child.findall("PersonResult"):
                i = person.find('Result/Position').text
                c.update({i : person.find('Person/Name/Family').text})
            # Gli atleti sono inseriti nel file XML per ordine di arrivo
            # Se si volessero ordinare togliere il commento
            #c = dict(sorted(c.items()))
            body = json.dumps(c)

    return {
        'statusCode': 200,
        'body': body
    }
```

Test

lambda_functio × Execution result: ×

Execution results — Status: Succeeded | Max memory used: 43 MB | Time: 2983.20 ms

Test Event Name

MyTest

Response


```
{
  "statusCode": 200,
  "body": "{\"1\": \"RANCAN\", \"2\": \"EIDSMO\", \"3\": \"SUTER\", \"6\": \"NIEMI\", \"7\": \"HADORN\", \"9\": \"OKSANEN\", \"10\": \"HUBACEK\", \"12\": \"AHOLA\", \"13\": \"SOLDINI\", \"15\": \"K
}
```


Function Logs

```
START RequestId: da5c3e84-8837-40da-a874-653d4baa6788 Version: $LATEST
END RequestId: da5c3e84-8837-40da-a874-653d4baa6788
REPORT RequestId: da5c3e84-8837-40da-a874-653d4baa6788  Duration: 2983.20 ms    Billed Duration: 2984 ms    Memory Size: 128 MB Max Memory Used: 43 MB
```

Request ID

da5c3e84-8837-40da-a874-653d4baa6788

Download XML

Lambda function

Un utente fa una chiamata GET all'endpoint di **`downloadxml`**, passando come parametro l'ID gara interessata.

Riceverà come risposta il file XML della gara dal Bucket S3.

```python
import json
import xml.etree.ElementTree as ET
import boto3
import uuid

def lambda_handler(event, context):

    content = event["body"]
    id = event['queryStringParameters']['id']
    file_name = parameter_validation(id)
    file_name = file_name + ".xml"

    # Carico il BucketS3
    bucket_name = "xmlrequests"
    s3_path = "test/" + file_name
    s3_client = boto3.client("s3")

    #print(bucket_name + " " + file_name + " " + s3_path)

    # Scarico il file
    s3_client.download_file(bucket_name, s3_path, '/tmp/{}'.format(file_name))
    body = "File XML richiesto scaricato"

    return {
        'statusCode': 200,
        'body': body
    }

def parameter_validation(id): # Check se l'id corrisponde
    event = ""
    dynamo = boto3.resource('dynamodb')
    table = dynamo.Table('RisultatiGare')
    response = table.get_item(
    Key={
        'Id': id
    })
    event = response["Item"]["Evento"]
    return event
```

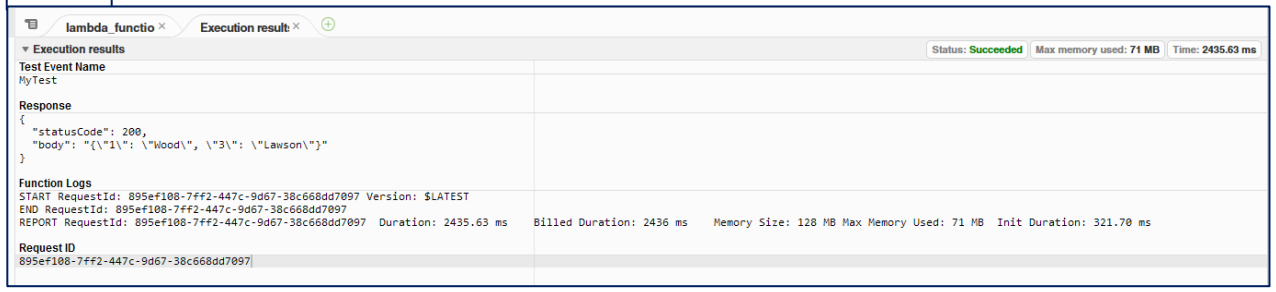
Results v2

Lambda function

Un utente fa una chiamata GET all'endpooint di **`results_v2`**, passando come parametro l'ID gara e il nome di un club che ha partecipato all'evento.

Riceverà una lista JSON degli atleti partecipanti alla gara di quel club.

Test



Codice

```python
import json
import xml.etree.ElementTree as ET
import boto3
import uuid

def lambda_handler(event, context):

    content = event["body"]
    id = event['queryStringParameters']['id']
    org = event['queryStringParameters']['organisation']

    # Get nome della gara dal DynamoDB tramite il suo ID
    dynamo = boto3.resource('dynamodb')
    table = dynamo.Table('RisultatiGare')
    response = table.get_item(
    Key={
        'Id': id
    })
    event_name = response["Item"]["Evento"]

    bucket_name = "xmlrequests"
    s3_path = "test/" + event_name + ".xml"

    # Get file XML dal BucketS3 ed estrai i club
    s3 = boto3.client("s3")
    s3_object = s3.get_object(Bucket=bucket_name, Key=s3_path)
    tree = ET.parse(s3_object['Body'])
    root = tree.getroot()
    o = {}
    body = "Il club " + org + " non è stato trovato"
    for child in root.findall("./ClassResult"):
        for person in child.findall("PersonResult"):
            if(person.find("Organisation/Name").text == org):
                i = person.find("Person/Id").text
                f = person.find("Person/Name/Family").text
                o.update({i : f})
    body = json.dumps(o)

    return {
        'statusCode': 200,
        'body': body
    }
```