



# Progetto TCM

---

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

# Register Race<sup>(1)</sup>

## 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.

### Test

| Execution results |  |
|-------------------|--|
| Test Event Name   | MyTest   |
| Response          | <pre>{   "statusCode": 200,   "body": "{\"IdEvento\": \"f6edbd89-3111-4bd6-ad32-412b001a7a65\", \"TokenUtente\": \"eczpcgkxkznxkdcqauxvrohb\"}" }</pre>  |
| Function Logs     | START RequestId: c757ffe2-f52b-4452-8ea1-5e9c5123cb2d Version: \$LATEST<br>END RequestId: c757ffe2-f52b-4452-8ea1-5e9c5123cb2d<br>REPORT RequestId: c757ffe2-f52b-4452-8ea1-5e9c5123cb2d Duration: 1509.29 ms Billed Duration: 1510 ms Memory Size: 128 MB Max Memory Used: 64 MB Init Duration: 260.43 ms |
| Request ID        | c757ffe2-f52b-4452-8ea1-5e9c5123cb2d   |

### Codice

```
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('GareRegistrare')
    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))
```

# Register Race<sub>(2)</sub>

## DynamoDB

- **Amministratori:**

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

Tabelle (4)

Qualsiasi tag tabella

Q Trova tabelle in base al nome della

< 1 >

Amministratori

GareRegistrate

ReportGare

RisultatiGare

Amministratori

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"/> | Token                    | Email                         |
|--------------------------|--------------------------|-------------------------------|
| <input type="checkbox"/> | eczpcgkxznhxkdcqauxvrohb | n.zaffaroni@studenti.unibg.it |
| <input type="checkbox"/> | cotearourtomkpmqrtilgtx  | g.daidone@studenti.unibg.it   |

- **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.

Tabelle (4) ×

Qualsiasi tag tabella ▾

Q Trova tabelle in base al nome della

< 1 > ⚙

○ Amministratori

● GareRegistrate

○ ReportGare

○ RisultatiGare

GareRegistrate

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"/> | <div>Id ▾</div>                      | <div>Email ▾</div>          | <div>RaceDate ▾</div> | <div>RaceName ▾</div>                    | <div>RacePlace ▾</div> |
|--------------------------|--------------------------------------|-----------------------------|-----------------------|--|------------------------|
| <input type="checkbox"/> | f01448dc-210c-4b7b-8980-2f6f797e37f9 | g.daidone@studenti.unibg.it | 08-05-2022            | nome_gara_di_prova                       | Bergamo                |
| <input type="checkbox"/> | f6edbd89-3111-4bd6-ad32-412b001a7a65 | g.daidone@studenti.unibg.it | 08-05-2022            | Coppa_Italia_sprint_3^_prova_2022-04-... | Bergamo                |

# Upload XML<sup>(1)</sup>

## 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.

```
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('RisultatiGara')
            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

```
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<sup>(2)</sup>

Tabelle (4)

Qualsiasi tag tabella

Trova tabelle in base al nome della

< 1 >

Amministratori

GareRegistrate

ReportGare

**RisultatiGare**

RisultatiGare

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 >

|                          | Id                                   | CaricatoDa                    | Evento   |
|--------------------------|--------------------------------------|-------------------------------|--|
| <input type="checkbox"/> | a01448dc-210c-4b7b-8980-2f6f797e37f9 | n.zaffaroni@studenti.unibg.it | MOC_CHAMPIONSHIP_-_COPPA_ITALIA_SPRINT_-_Monopoli_2022-03-20 |
| <input type="checkbox"/> | f01448dc-210c-4b7b-8980-2f6f797e37f9 | g.daidone@studenti.unibg.it   | nome_gara_di_prova_2011-07-30                                |

DynamoDB

Oggetti (4)

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](#)

Carica

Operazioni

Elimina

Apri

Scarica

Copia URL

Copia URI S3

Trova oggetti per prefisso

Mostra versioni

< 1 >

|                          | Nome   | Tipo | Ultima modifica              | Dimensioni | Classe di storage |
|--------------------------|--|------|------------------------------|------------|-------------------|
| <input type="checkbox"/> | Coppa_Italia_sprint_3^_prova_2022-04-23.xml                      | xml  | 07 May 2022 04:33:55 PM CEST | 1.8 MB     | Standard          |
| <input type="checkbox"/> | Example_event_2011-07-30.xml                                     | xml  | 07 May 2022 05:35:59 PM CEST | 13.0 KB    | Standard          |
| <input type="checkbox"/> | MOC_CHAMPIONSHIP_-_COPPA_ITALIA_SPRINT_-_Monopoli_2022-03-20.xml | xml  | 07 May 2022 05:36:24 PM CEST | 1.5 MB     | Standard          |
| <input type="checkbox"/> | nome_gara_di_prova_2011-07-30.xml                                | xml  | 10 May 2022 11:00:26 AM CEST | 11.2 KB    | Standard          |

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

```
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('GareRegistrare')
    response = table.scan()
    resp = table.scan(ProjectionExpression="Id, RaceName, RaceDate, RacePlace")

    body = resp["Items"]

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

### Test

lambda\_funcio x Execution result x +

Status: **Succeeded** Max memory used: 64 MB Time: 1392.49 ms

**Execution results**

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'endpoint 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.

### Test

| Execution results |  |
|-------------------|--|
| Test Event Name   | MyTest   |
| Response          | <pre>{   "statusCode": 200,   "body": "{\"1\": \"Men Elite\", \"2\": \"Open\"}" }</pre>  |
| Function Logs     | <pre>START RequestId: 47534967-b011-4468-8a98-1da446043da2 Version: \$LATEST END RequestId: 47534967-b011-4468-8a98-1da446043da2 REPORT RequestId: 47534967-b011-4468-8a98-1da446043da2  Duration: 2251.47 ms   Billed Duration: 2252 ms   Memory Size: 128 MB Max Memory Used: 71 MB   Init Duration: 247.41 ms</pre> |
| Request ID        | 47534967-b011-4468-8a98-1da446043da2   |

### Codice

```
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
    }
```

# Results

## Lambda function

Un utente fa una chiamata GET all'endpoint 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.

### Test

| Execution results |   | Status: Succeeded | Max memory used: 43 MB | Time: 2983.20 ms |
|-------------------|---|-------------------|------------------------|------------------|
| Test Event Name   | MyTest  |                   |                        |                  |
| Response          | <pre>{   "statusCode": 200,   "body": "{\"1\": \"RANCAN\\\", \"2\": \"EIDSMO\\\", \"3\": \"SUTER\\\", \"6\": \"NIEMI\\\", \"7\": \"HADORN\\\", \"9\": \"OKSANEN\\\", \"10\": \"HUBACEK\\\", \"12\": \"AHOLA\\\", \"13\": \"SOLDINI\\\", \"15\": \"K\" }</pre>                                 |                   |                        |                  |
| Function Logs     | <pre>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</pre> |                   |                        |                  |
| Request ID        | da5c3e84-8837-40da-a874-653d4baa6788  |                   |                        |                  |

### Codice

```
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
    }
```



# 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.

```
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'endpoint 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

| Execution results |   | Status: Succeeded | Max memory used: 71 MB | Time: 2435.63 ms |
|-------------------|---|-------------------|------------------------|------------------|
| Test Event Name   | MyTest  |                   |                        |                  |
| Response          | <pre>{   "statusCode": 200,   "body": "{\\\"1\\\": \\\"Wood\\\", \\\"3\\\": \\\"Lawson\\\"}" }</pre>  |                   |                        |                  |
| Function Logs     | <pre>START RequestId: 895ef108-7ff2-447c-9d67-38c668dd7097 Version: \$LATEST END RequestId: 895ef108-7ff2-447c-9d67-38c668dd7097 REPORT RequestId: 895ef108-7ff2-447c-9d67-38c668dd7097  Duration: 2435.63 ms  Billed Duration: 2436 ms  Memory Size: 128 MB Max Memory Used: 71 MB  Init Duration: 321.70 ms</pre> |                   |                        |                  |
| Request ID        | 895ef108-7ff2-447c-9d67-38c668dd7097  |                   |                        |                  |

## Codice

```
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
    }
```