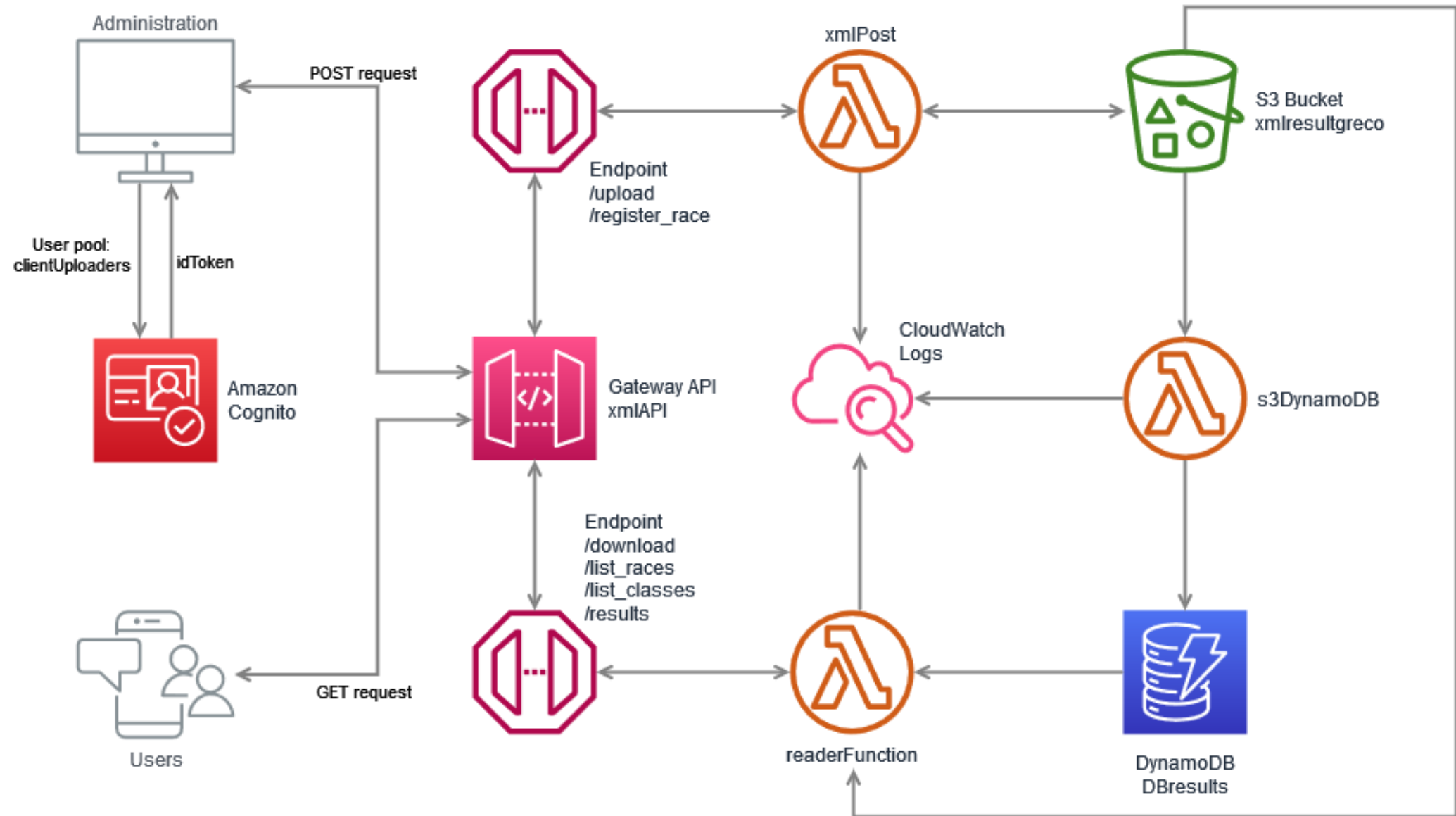


Infrastruttura cloud AWS

Componenti del gruppo:

Greco Daniele-1065570, Matias Negro-1065808

Infrastruttura generale



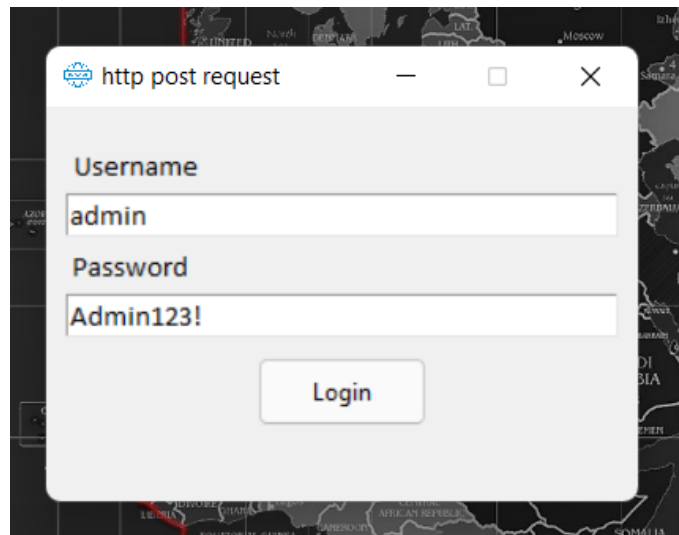
/upload
/register_race
Metadata

È stato aggiunto l'Endpoint ***/register_race*** al quale è possibile inviare una POST request contenente due parametri, nome della gara e data di inizio. All'interno dell'*header* viene inserito l'Id identificativo assegnato da Amazon Cognito che consente di caricare il file all'interno del bucket e di recuperare *username* e *email* dell'utente che sta caricando il file. Viene quindi generato il file XML contenente i dati fondamentali della gara. Nei ***Metadata*** del file caricato vengono aggiunti i campi ***username*** e ***email*** per identificare il proprietario del file. All'utente è restituito il nome (univoco) del file nel *Bucket*.

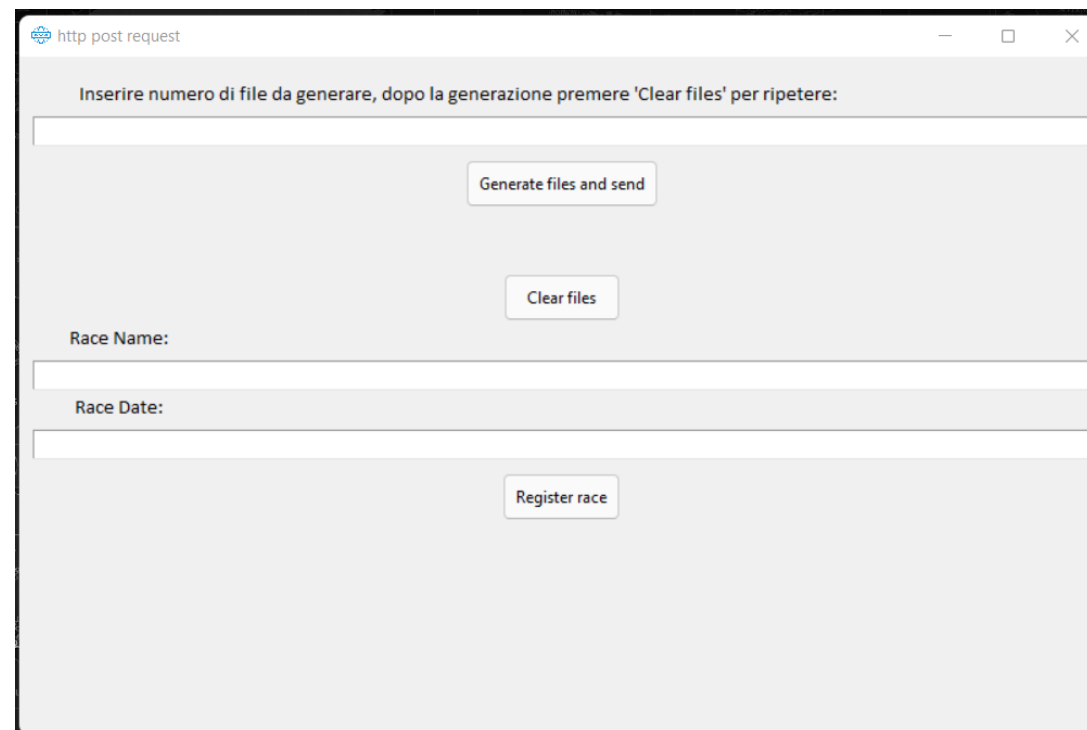
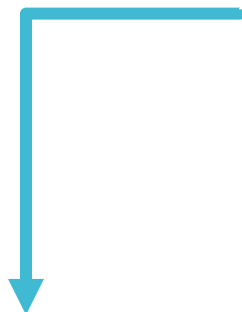
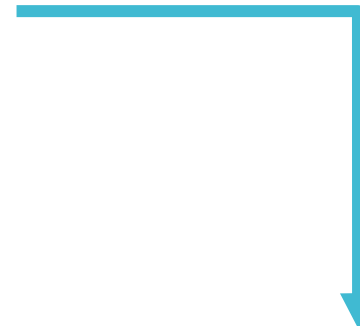
Ad ogni inserimento di file nel bucket attraverso l'Endpoint ***/upload*** viene controllato che il file esista già, se è già presente viene controllato nei ***Metadata*** chi è l'utente che ha caricato il file. Se l'utente è lo stesso, il file viene sovrascritto, previo controllo del formato.

Nel caso il file sia nuovo, viene controllato il formato; nel caso sia corretto viene caricato insieme ai Metadata del proprietario.

Login
(solo per gli utenti che
vogliono eseguire un
upload)



A screenshot of a web application window titled "http post request". It contains a login form with two input fields: "Username" with the value "admin" and "Password" with the value "Admin123!". Below the fields is a "Login" button. The window is set against a dark world map background.



A screenshot of the main application interface, titled "http post request". It features a form for generating and registering races. The form includes a text input for "Inserire numero di file da generare, dopo la generazione premere 'Clear files' per ripetere:", followed by "Generate files and send" and "Clear files" buttons. Below these are fields for "Race Name:" and "Race Date:", each with a text input. At the bottom is a "Register race" button.

/upload

http post request

Inserire numero di file da generare, dopo la generazione premere 'Clear files' per ripetere:

1

Generate files and send

QSQbODLEF2020-01-17.xml

result

The following files have been uploaded:
QSQbODLEF2020-01-17.xml

OK

Race Name:

Race Date:

Register race

Amazon S3 > Buckets > xmlresultgreco > partite/ > QSQbODLEF2020-01-17.xml

QSQbODLEF2020-01-17.xml

Copy S3 URIDownloadOpenObject actions

PropertiesPermissionsVersions

Object overview

Owner
awslabsc0w1392920t1605579974

AWS Region
US East (N. Virginia) us-east-1

Last modified
May 11, 2022, 22:39:08 (UTC+02:00)

Size
25.6 KB

Type
xml

Key
partite/QSQbODLEF2020-01-17.xml

S3 URI
s3://xmlresultgreco/partite/QSQbODLEF2020-01-17.xml

Amazon Resource Name (ARN)
arn:aws:s3:::xmlresultgreco/partite/QSQbODLEF2020-01-17.xml

Entity tag (Etag)
a0d3919eb609f8a8e05bf4d7d689824e

Object URL
https://xmlresultgreco.s3.amazonaws.com/partite/QSQbODLEF2020-01-17.xml

Metadata (3)

Metadata is optional information provided as a name-value (key-value) pair. [Learn more](#)

[Edit](#)

Type	Key	Value
System defined	Content-Type	binary/octet-stream
User defined	x-amz-meta-email	sundman1012@gmail.com
User defined	x-amz-meta-uploader	admin

/registe_race

http post request

Inserire numero di file da generare, dopo la generazione premere 'Clear files' per ripetere:

Generate files and send

Clear files

Race Name:

test

Race Date:

11-05-2022

Register race

result

Race registered succesfully!
Key = test11-05-2022.xml

OK

Amazon S3 > Buckets > xmlresultgreco > partite/ > test11-05-2022.xml

test11-05-2022.xml

Copy S3 URICopy S3 URIDownloadDownloadOpenOpenObject actionsObject actions

PropertiesPermissionsVersions

Object overview

Owner

awslabsc0w1392920t1605579974

AWS Region

US East (N. Virginia) us-east-1

Last modified

May 11, 2022, 22:47:30 (UTC+02:00)

Size

199.0 B

Type

xml

Key

partite/test11-05-2022.xml

S3 URI

s3://xmlresultgreco/partite/test11-05-2022.xml

Amazon Resource Name (ARN)

arn:aws:s3:::xmlresultgreco/partite/test11-05-2022.xml

Entity tag (Etag)

2d91198ea9449206472f59a58f5b1970

Object URL

https://xmlresultgreco.s3.amazonaws.com/partite/test11-05-2022.xml

Metadata (3)

Metadata is optional information provided as a name-value (key-value) pair. [Learn more](#)

[Edit](#)

Type	Key	Value
System defined	Content-Type	binary/octet-stream
User defined	x-amz-meta-email	sundman1012@gmail.com
User defined	x-amz-meta-uploader	admin

Snippet delle funzioni *lambda* per il controllo del formato e del proprietario

```
61
62 def name_control(file_name, username):
63     '''
64     Controllo il bucket per vedere se il nome del file è già presente, nel ca
65     '''
66     flag = True
67     for o in bucket.objects.all():
68         if (prefix+file_name) == o.key:
69             obj = s3_resource.Object(BUCKET_NAME, o.key)
70             meta = getattr(obj, 'metadata')
71             metaUser = meta["uploader"]
72             if metaUser != username:
73                 flag = False
74     return flag
75
```

```
29
30 def format_validator(root):
31     source_file = ET.tostring(root)
32     schema_file = 'schema.xsd'
33     schema_file_type = 'schema_type.xsd'
34     flag = True
35     flag_type = True
36
37     with open(schema_file) as f_schema, open(schema_file_type) as f_schema_type:
38
39         schema_doc = etree.parse(f_schema)
40         schema_type_doc = etree.parse(f_schema_type)
41         schema = etree.XMLSchema(schema_doc)
42         schema_type = etree.XMLSchema(schema_type_doc)
43         parser = etree.XMLParser(schema=schema)
44         parser_type = etree.XMLParser(schema = schema_type)
45
46         try:
47             doc = etree.fromstring(source_file, parser)
48         except etree.XMLSyntaxError as e:
49             # this exception is thrown on schema validation error
50             # print(e) è solo per eventuale debugging
51             #print(e)
52             flag = False
53
54         try:
55             doc = etree.fromstring(source_file, parser_type)
56         except etree.XMLSyntaxError as e:
57             #print(e)
58             flag_type = False
59
60     return (flag or flag_type)
61
```

/list_races

/list_classes?id=...

/results?id=...&class=...

/download?id=...

- */list_races*

- Viene costruito nella funzione lambda *readerFunction* un *Dictionary* che ha come indice un vettore di *int* e per ogni elemento vi sono gli attributi *race_name*, *race_date* e *race_id* di tutte le gare presenti nel **Database**. La risposta è in formato *json*.

- */list_classes?id=...*

- Estrae l'*id* e il *name* delle categorie contenute nell'evento corrispettivo all'*id* inserito come parametro nella *GET request*. La risposta è in formato *json*

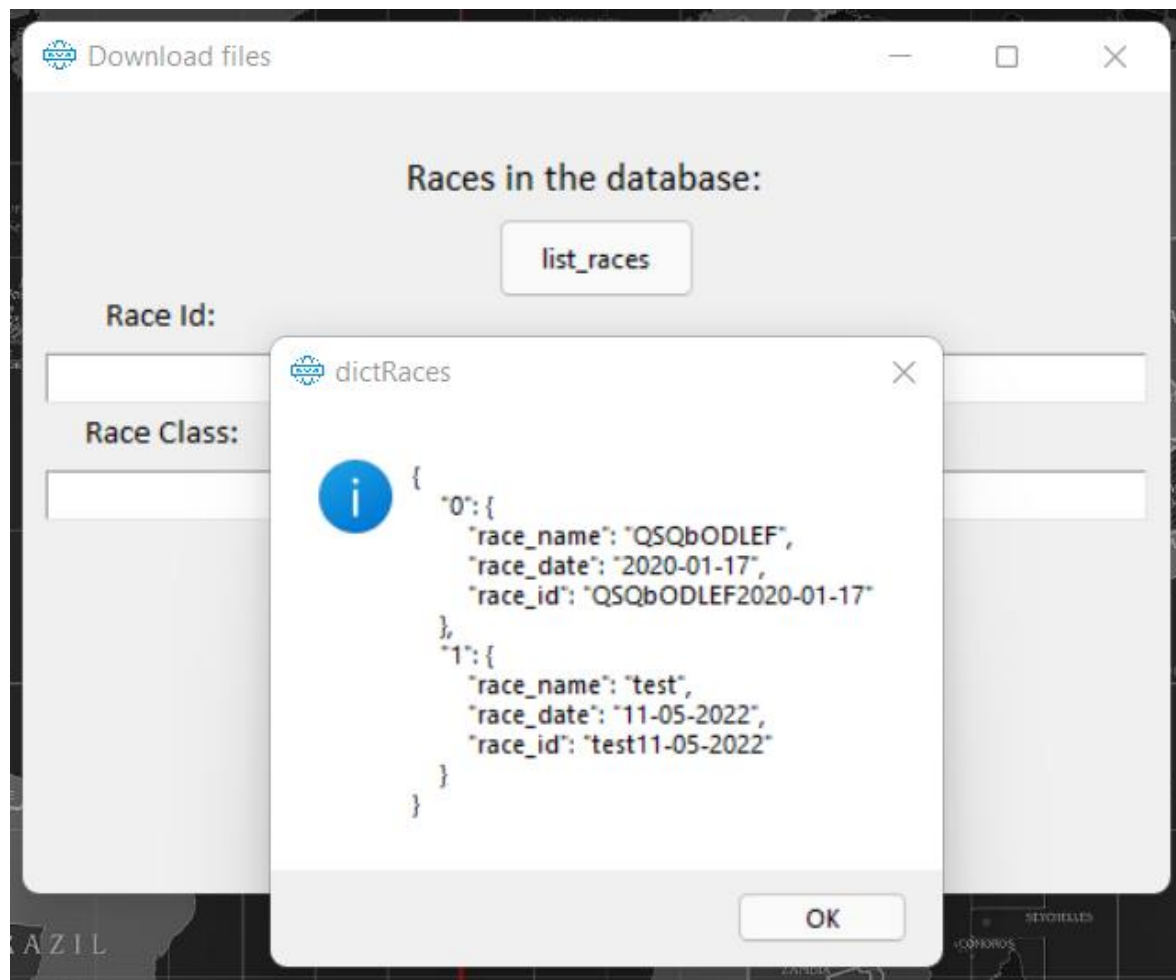
- */results?id=...&class=...*

- Preso in input *id* e *class* della gara da prendere in considerazione, restituisce un *Dictionary* che ha come indice un vettore di *int* a cui corrisponde il posizionamento in classifica e attributo l'*id* del giocatore corrispondente. Viene restituito in formato *json* in quanto l'adattamento in *flutter* del front-end permetterà un'interpretazione più agevole delle informazioni restituite dall'infrastruttura cloud.

- */Download?id=...*

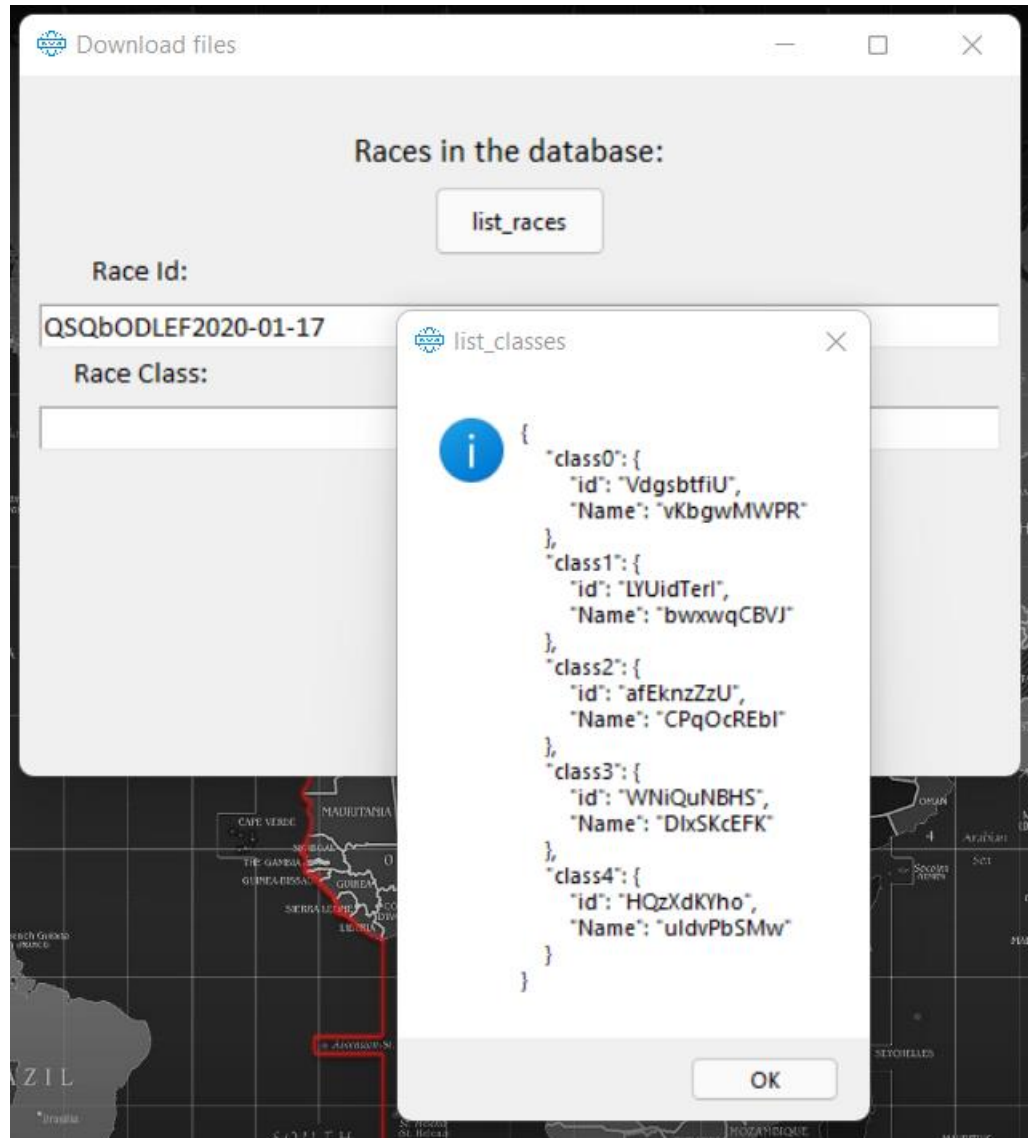
- Preso in input l'*id* della gara da scaricare, la funzione lambda preleva dal **Bucket** il contenuto del file corrispondente e lo inserisce all'interno di un file creato in locale nella cartella */downloads* dell'app di test */readerApp*.

/list_races



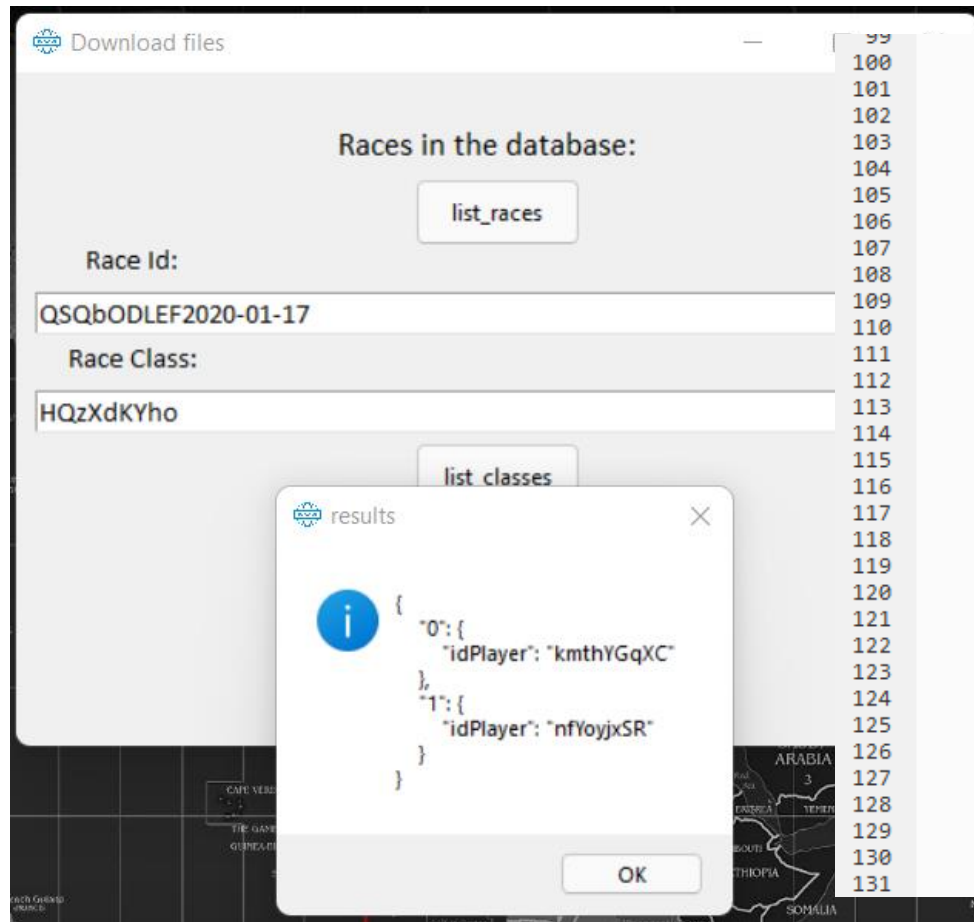
```
33
34
35 if resource == 'list_races':
36     dict = {}
37     r = ddb_client.scan(
38         TableName='DBresults',
39         AttributesToGet=[
40             'event',
41             'Event'
42         ],
43     )
44     j = 0
45     for i in r['Items']:
46         dict[j] = {
47             'race_name' : i['Event']['M']['Name']['S'],
48             'race_date' : i['Event']['M']['StartTime']['M']['Date']['S'],
49             'race_id' : i['event']['S'],
50         }
51         j += 1
52     dict_json = json.dumps(dict, indent=4)
53     dict_json = dict_json.split(', "ResponseMetadata":')
54     dict_json = dict_json[0]
55     if dict=={}:
56         response['body'] = 'Races not found'
57     else:
58         response['body'] = dict_json
```

/list_classes?id=...



```
70
71 elif resource=='list_classes':
72     dict = {}
73     id = event["queryStringParameters"]["id"]
74     r = ddb_client.scan(
75         TableName='DBresults',
76         AttributesToGet=[
77             'event',
78             'ClassResults'
79         ],
80     )
81     j = 0
82     for i in r['Items']:
83         if (i['event']['S'] == id):
84             classResults = i['ClassResults']['M']
85             i=0
86             for k in classResults:
87                 dict['class'+str(i)] = {
88                     'id' : classResults[k]['M']['Class']['M']['Id']['S'],
89                     'Name' : classResults[k]['M']['Class']['M']['Name']['S']
90                 }
91                 i += 1
92     dict_json = json.dumps(dict, indent=4)
93     dict_json = dict_json.split(', "ResponseMetadata":')
94     dict_json = dict_json[0]
95     if dict=={}:
96         response['body'] = 'Error 404: Race not found'
97     else:
98         response['body'] = dict_json
99
```

/results?id=...&class=...



```
100
101
102 elif(resource == 'results'):
103     dict = {}
104     id = event["queryStringParameters"]["id"]
105     cl = event["queryStringParameters"]["class"]
106     r = ddb_client.scan(
107         TableName='DBResults',
108         AttributesToGet=[
109             'event',
110             'ClassResults'
111         ],
112     )
113     for i in r['Items']:
114         if (i['event']['S'] == id):
115             for k in i['ClassResults']['M']:
116                 if (i['ClassResults']['M'][k]['M']['Class']['M']['Id']['S'] == cl):
117                     for p in i['ClassResults']['M'][k]['M']:
118                         if 'PersonResult' in p:
119                             idPlayer = i['ClassResults']['M'][k]['M'][p]['M']['Person']['M']['Id']['S']
120                             playerPosition = i['ClassResults']['M'][k]['M'][p]['M']['Result']['M']['Position']['S']
121                             dict[playerPosition] = {
122                                 'idPlayer': idPlayer
123                             }
124
125 dict_json = json.dumps(dict, indent=4)
126 dict_json = dict_json.split(', "ResponseMetadata: "')
127 dict_json = dict_json[0]
128 if dict=={}:
129     response['body'] = 'Error 404: Race not found'
130 else:
131     response['body'] = dict_json
```


/download?id=...

Download files

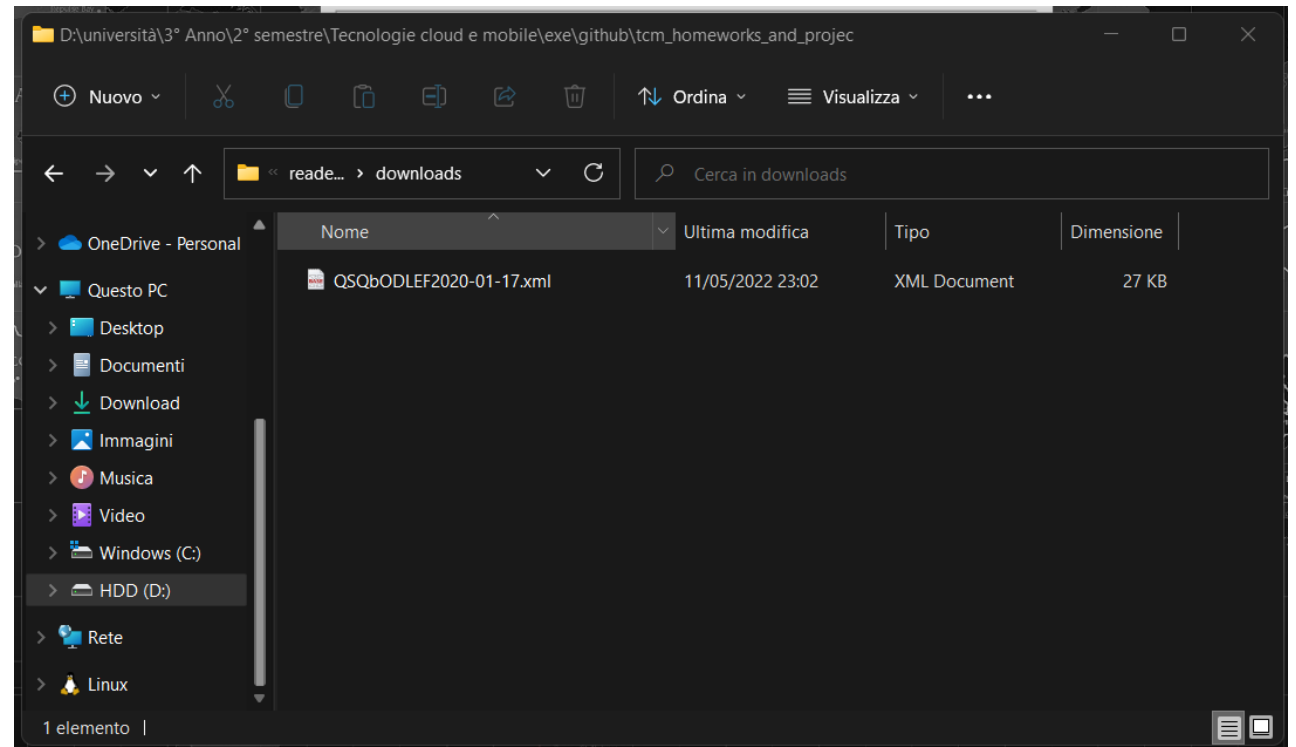
Races in the database:

Race Id:

Race Class:

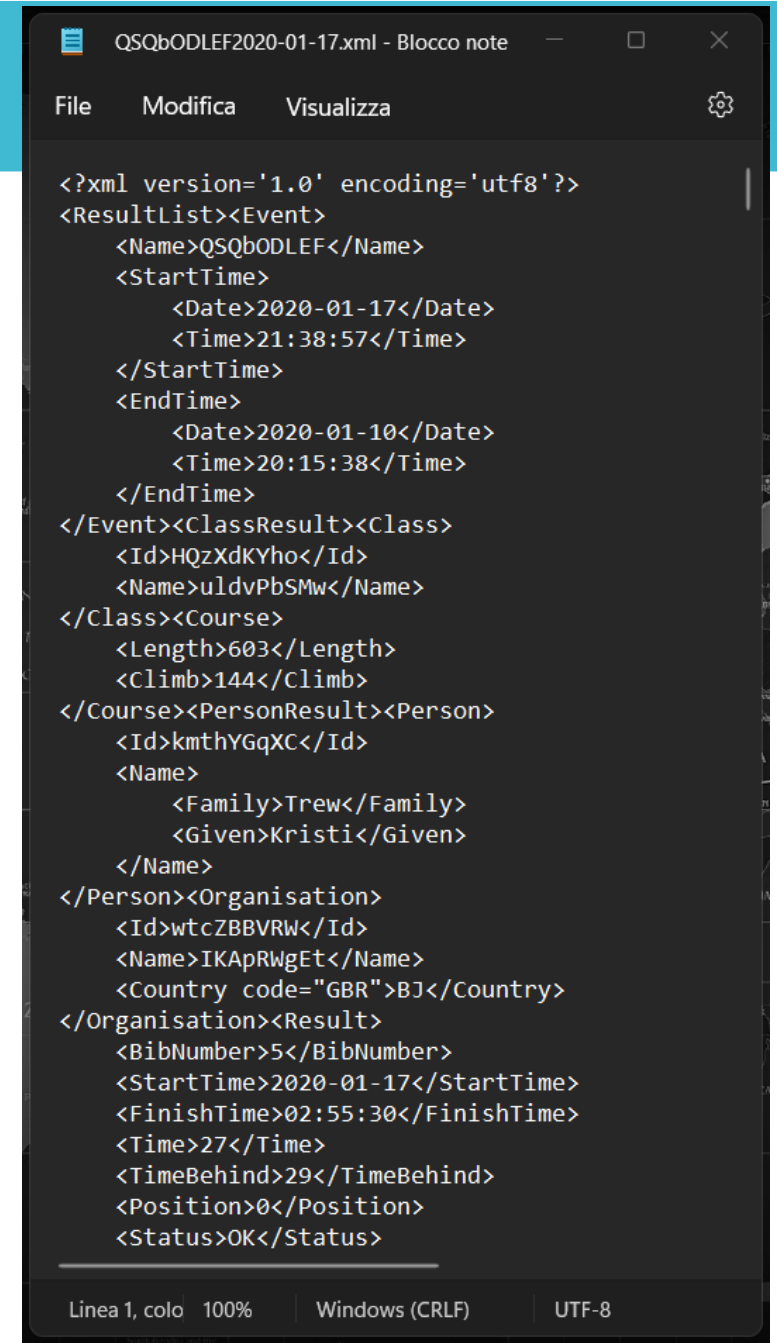
 File downloaded succesfully, check downloads folder.

```
59
60 elif resource == 'download':
61     #Getting the key
62     key = prefix + event["headers"]["filename"]
63     try:
64         race = s3_client.get_object(Bucket=BUCKET_NAME, Key=key)
65         xml = race['Body'].read().decode('utf-8')
66         response['body'] = xml
67     except:
68         response['body'] = 'error404'
69
```



/download?id=...

XML generato dall'algoritmo di simulazione usato come esempio negli screenshot.

A screenshot of a 'Blocco note' (Notepad) application window. The title bar reads 'QSQbODLEF2020-01-17.xml - Blocco note'. The menu bar includes 'File', 'Modifica', 'Visualizza', and a settings icon. The main text area contains XML data. The status bar at the bottom shows 'Linea 1, colo 100%', 'Windows (CRLF)', and 'UTF-8'.

```
<?xml version='1.0' encoding='utf8'?>
<ResultList><Event>
  <Name>QSQbODLEF</Name>
  <StartTime>
    <Date>2020-01-17</Date>
    <Time>21:38:57</Time>
  </StartTime>
  <EndTime>
    <Date>2020-01-10</Date>
    <Time>20:15:38</Time>
  </EndTime>
</Event><ClassResult><Class>
  <Id>HQzXdKYho</Id>
  <Name>uldvPbSMw</Name>
</Class><Course>
  <Length>603</Length>
  <Climb>144</Climb>
</Course><PersonResult><Person>
  <Id>kmthYGqXC</Id>
  <Name>
    <Family>Trew</Family>
    <Given>Kristi</Given>
  </Name>
</Person><Organisation>
  <Id>wtcZBBVRW</Id>
  <Name>IKApRWgEt</Name>
  <Country code="GBR">BJ</Country>
</Organisation><Result>
  <BibNumber>5</BibNumber>
  <StartTime>2020-01-17</StartTime>
  <FinishTime>02:55:30</FinishTime>
  <Time>27</Time>
  <TimeBehind>29</TimeBehind>
  <Position>0</Position>
  <Status>OK</Status>
```

GitHub

Link alla repository con le applicazioni *uploaderAppm*, *readerApp*, le funzioni *lambda* e gli **screenshot** in formato .png visibili pubblicamente:

https://github.com/MatiasNegro/tcm_homeworks_and_project