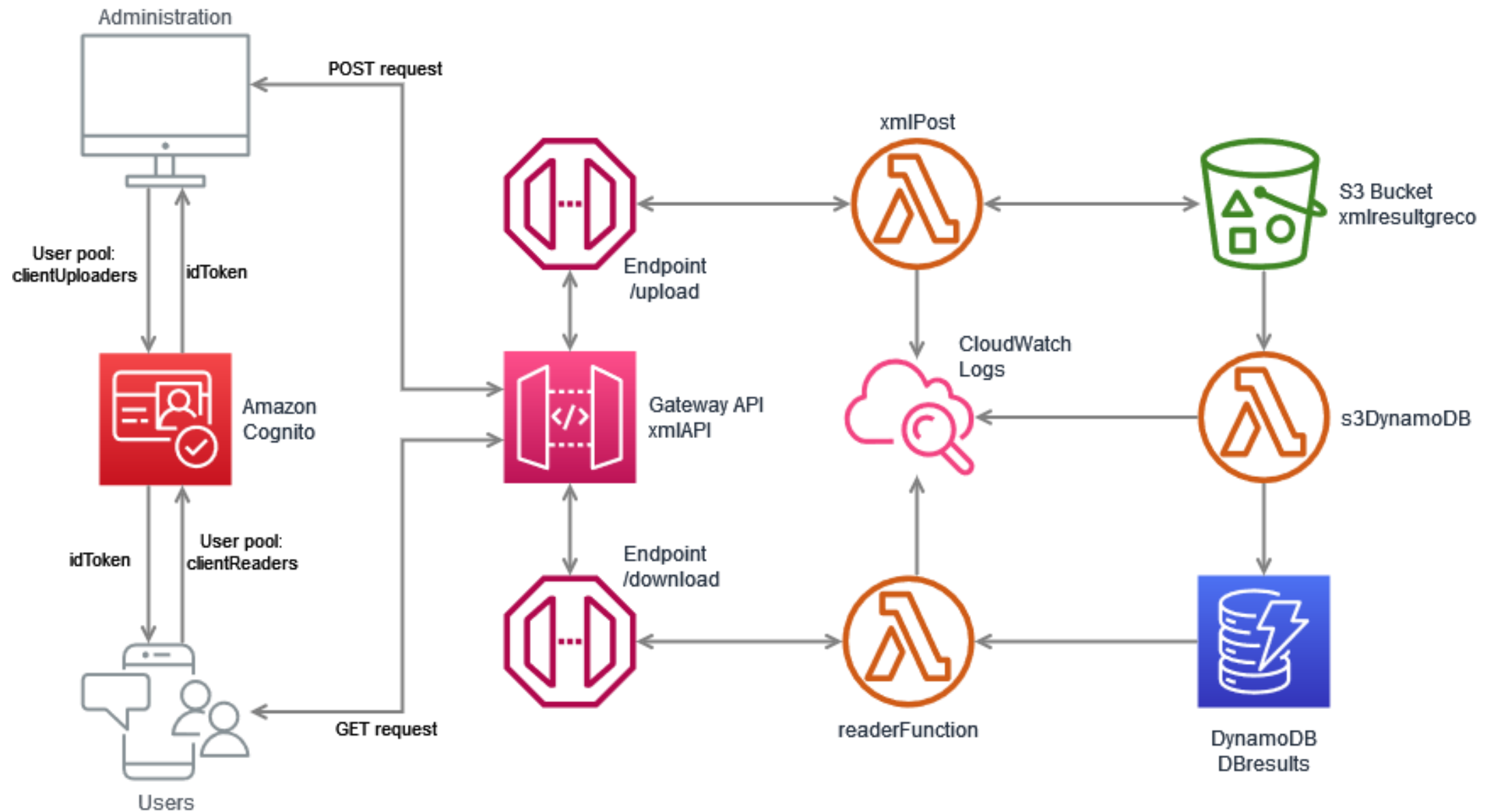


# Infrastruttura cloud AWS

Componenti del gruppo:

Greco Daniele-1065570, Matias Negro-1065808

# Infrastruttura generale



## Utenti, permessi ed Interfaccia con il cloud

Per la gestione dei permessi abbiamo creato due *user pool* utilizzando il servizio Amazon Cognito: *Administrator* con permessi di lettura e scrittura e *User* solo con permessi di lettura.

L'*Administrator* può mandare una richiesta POST all'endpoint «*/upload*» della gateway API per caricare molteplici file.

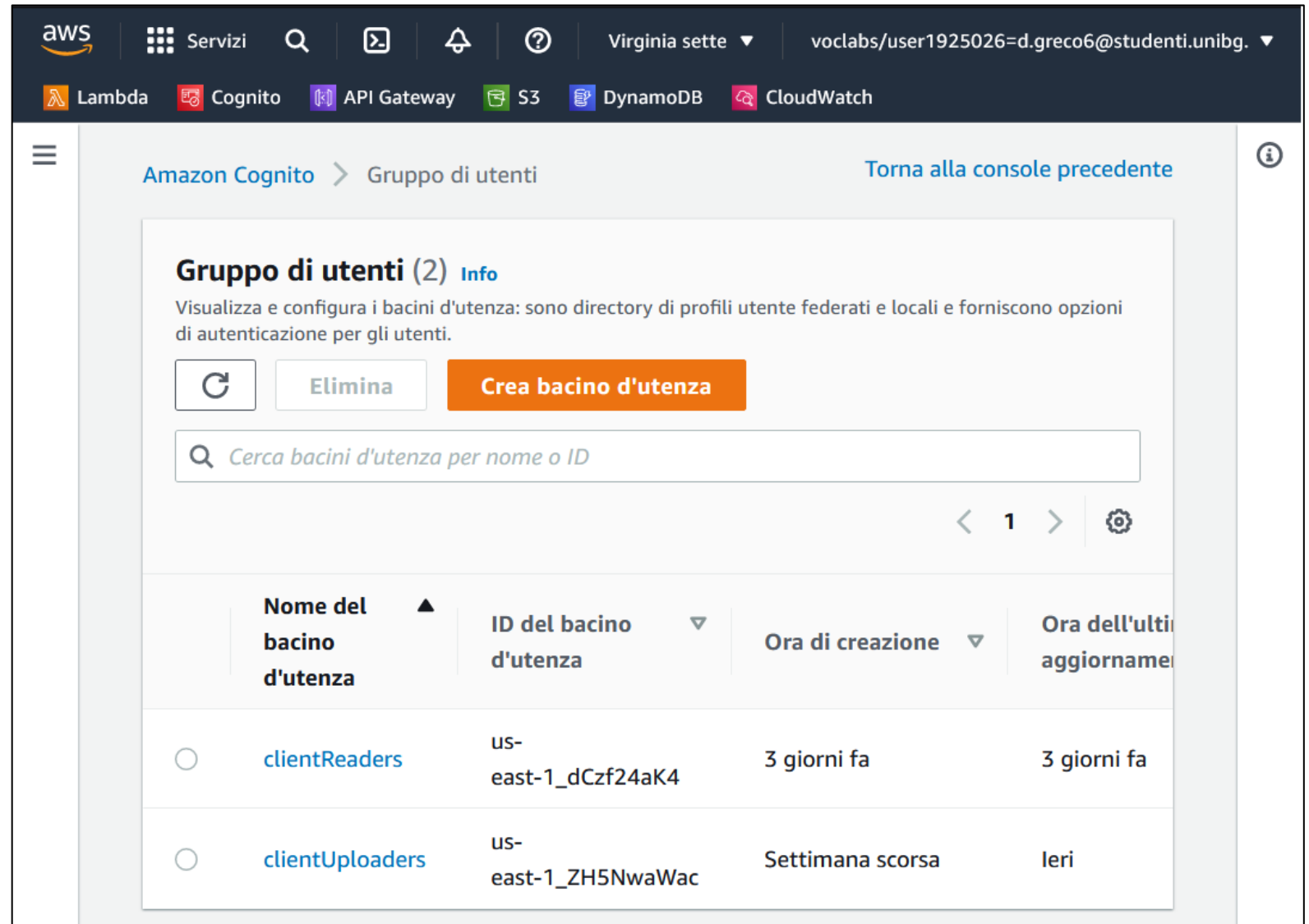
Tale richiesta viene strutturata nel modo seguente:

- *Headers*:
  - *idToken*: token di autenticazione di Amazon Cognito;
  - Nomi dei file da caricare nel Bucket.
- *Body*:
  - Contenuto dei file separato da una stringa di caratteri inserita dal client e riconosciuta dalla funzione lambda.

L'*User* può solo mandare una richiesta GET all'endpoint «*/download*» della gateway API per ottenere i nomi degli eventi presenti sul database e scaricarne il contenuto.

Si è assunto che chiunque possa registrarsi come *User* scegliendo nome utente e password, invece non è consentito registrarsi come *Administrator* se non direttamente dalla console Cognito.

# Utenti, permessi ed Interfaccia con il cloud



The screenshot shows the Amazon Cognito console interface. At the top, there's a navigation bar with the AWS logo, a 'Servizi' menu, and a search bar. Below this, a secondary bar lists services: Lambda, Cognito, API Gateway, S3, DynamoDB, and CloudWatch. The main header area shows 'Amazon Cognito > Gruppo di utenti' and a link to 'Torna alla console precedente'.

The main content area is titled 'Gruppo di utenti (2) Info'. It includes a description: 'Visualizza e configura i bacini d'utenza: sono directory di profili utente federati e locali e forniscono opzioni di autenticazione per gli utenti.' Below this are three buttons: a refresh button, an 'Elimina' button, and a 'Crea bacino d'utenza' button. A search bar is also present with the placeholder text 'Cerca bacini d'utenza per nome o ID'.

Below the search bar is a table with the following columns: 'Nome del bacino d'utenza', 'ID del bacino d'utenza', 'Ora di creazione', and 'Ora dell'ultima aggiornamento'. The table contains two rows of data:

	Nome del bacino d'utenza	ID del bacino d'utenza	Ora di creazione	Ora dell'ultima aggiornamento
<input type="radio"/>	clientReaders	us-east-1_dCzf24aK4	3 giorni fa	3 giorni fa
<input type="radio"/>	clientUploaders	us-east-1_ZH5NwaWac	Settimana scorsa	Ieri

## Gateway API e API endpoint

È stata utilizzata una REST API dal servizio Gateway API. Quando viene inviata una richiesta all'API, questa cerca un «*header*» denominato «*Authorization*» contenente l'«*idToken*» di Amazon Cognito. Verifica poi che questo id corrisponda all'id assegnato all'utente da Cognito nella corretta *user pool*.

I due endpoint dell'API si interfacciano rispettivamente a due funzioni Lambda distinte:

# Gateway API e API endpoint

aws Servizi Virginia s

Lambda Cognito API Gateway S3 DynamoDB

Amazon API Gateway API > xmlAPI (x4d1kgdj83)

API

Nomi dominio

Collegamenti VPC

API: **xmlAPI**

Risorse

- Fasi
- Autorizzazioni
- Risposte di Gateway
- Modelli
- Policy delle risorse
- Documentazione
- Pannello di controllo
- Impostazioni
- Piani di utilizzo
- Chiavi API
- Certificati del client
- Impostazioni

Risorse

- /
- /download
  - GET
  - OPTIONS
- /upload
  - OPTIONS
  - POST

Operazioni

aws Servizi Virginia settentri voclabs/user1925026=d.greco6@studenti.unibg.it @ 865

Lambda Cognito API Gateway S3 DynamoDB CloudWatch

Amazon API Gateway API > xmlAPI (x4d1kgdj83) > Risorse > /download (ria5ow)

API

Nomi dominio

Collegamenti VPC

API: **xmlAPI**

Risorse

- Fasi
- Autorizzazioni
- Risposte di Gateway
- Modelli
- Policy delle risorse
- Documentazione
- Pannello di controllo
- Impostazioni
- Piani di utilizzo
- Chiavi API
- Certificati del client
- Impostazioni

Risorse

- /
- /download
  - GET
  - OPTIONS
- /upload
  - OPTIONS
  - POST

Operazioni

/download Metodi

GET

Autorizzazione GRUPPO DI UTENTI DI COGNITO

Chiave API Non obbligatorio

OPTIONS

Endpoint fittizio

Autorizzazione NESSUNO

Chiave API Non obbligatorio

aws Servizi Virginia settentri voclabs/user1925026=d.greco6@studenti.unibg.it @ 865

Lambda Cognito API Gateway S3 DynamoDB CloudWatch

Amazon API Gateway API > xmlAPI (x4d1kgdj83) > Risorse > /upload (ktaid9)

API

Nomi dominio

Collegamenti VPC

API: **xmlAPI**

Risorse

- Fasi
- Autorizzazioni
- Risposte di Gateway
- Modelli
- Policy delle risorse
- Documentazione
- Pannello di controllo
- Impostazioni
- Piani di utilizzo
- Chiavi API
- Certificati del client
- Impostazioni

Risorse

- /
- /download
  - GET
  - OPTIONS
- /upload
  - OPTIONS
  - POST

Operazioni

/upload Metodi

OPTIONS

Endpoint fittizio

Autorizzazione NESSUNO

Chiave API Non obbligatorio

POST

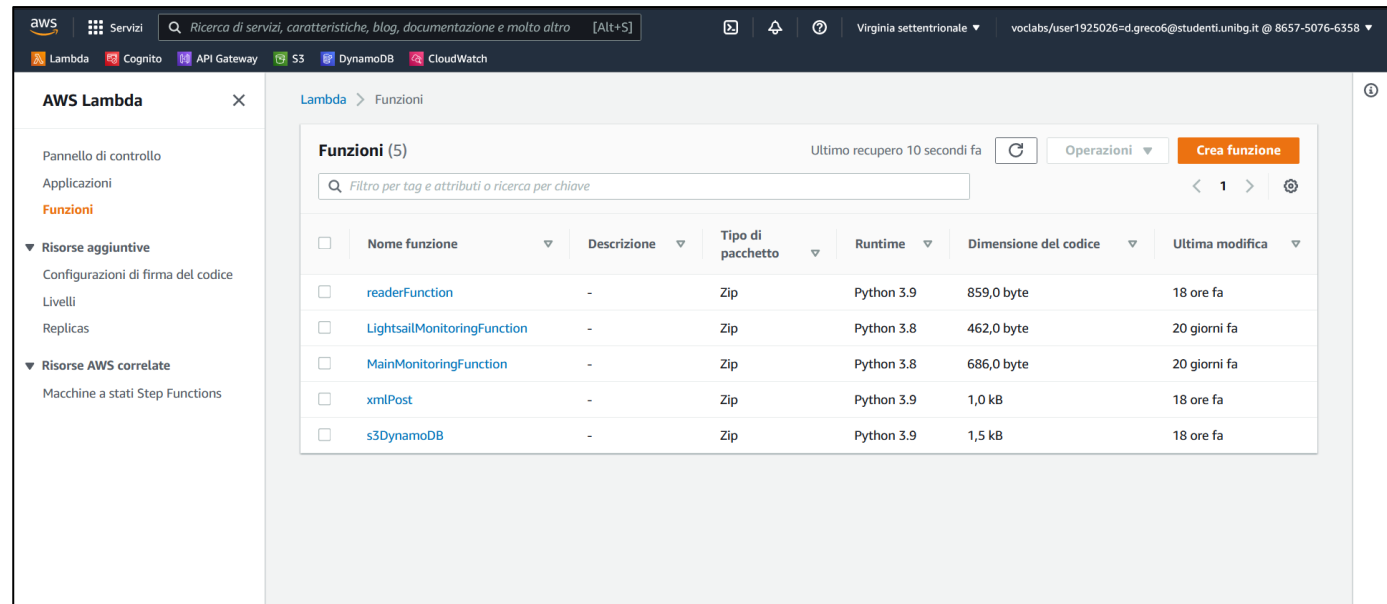
Autorizzazione GRUPPO DI UTENTI DI COGNITO

Chiave API Non obbligatorio

## Funzioni Lambda

- ***xmlPost*** è dedicata agli utenti *Administrator*. Rielabora il *body* della richiesta separandolo in più file e caricandoli uno per uno all'interno del Bucket S3. Vengono caricati anche se il nome del file è già presente nel bucket, aggiungendo il prefisso «*new\_*» al nome scelto dall'utente.
- ***readerFunction*** è dedicata agli utenti *Users*. Consente di prelevare tutti i nomi degli eventi o di scaricare il contenuto direttamente dal database. Deve essere presente nell'«*header*» della richiesta un campo *mod* settato a «*read*» o «*download*», nel primo caso la funzione Lambda restituisce i nomi degli eventi presenti nel database (si suppone che ogni volta avviata l'applicazione client vengano mostrati all'utente gli eventi presenti sul database), nel secondo caso va specificato un ulteriore «*header*» contenente il nome dell'evento da scaricare e la funzione preleva l'item corrispondente, in formato json, dal database e lo inserisce all'interno del «*body*» della risposta.

# Funzioni Lambda



The screenshot shows the AWS Lambda console interface. On the left, there's a sidebar with navigation options: Pannello di controllo, Applicazioni, Funzioni (selected), Risorse aggiuntive, and Risorse AWS correlate. The main area displays a list of functions under the heading 'Funzioni (5)'. The list includes columns for selection, function name, description, package type, runtime, code size, and last modification time.

	Nome funzione	Descrizione	Tipo di pacchetto	Runtime	Dimensione del codice	Ultima modifica
<input type="checkbox"/>	readerFunction	-	Zip	Python 3.9	859,0 byte	18 ore fa
<input type="checkbox"/>	LightsailMonitoringFunction	-	Zip	Python 3.8	462,0 byte	20 giorni fa
<input type="checkbox"/>	MainMonitoringFunction	-	Zip	Python 3.8	686,0 byte	20 giorni fa
<input type="checkbox"/>	xmlPost	-	Zip	Python 3.9	1,0 kB	18 ore fa
<input type="checkbox"/>	s3DynamoDB	-	Zip	Python 3.9	1,5 kB	18 ore fa

```
1 import boto3
2 import json
3 import xml_parser as xp
4 from xml.etree.ElementTree import fromstring, ElementTree, XMLParser
5 from boto3.dynamodb.types import TypeSerializer
6
7 s3_client = boto3.client('s3')
8 dynamodb = boto3.resource('dynamodb')
9
10 def lambda_handler(event, context):
11
12     #Getting file from bucketS3
13     bucket = event['Records'][0]['s3']['bucket']['name']
14     file_name = event['Records'][0]['s3']['object']['key']
15
16     #The file is an xml, we need the json
17     xml_obj = s3_client.get_object(Bucket = bucket, Key = file_name)['Body'].read().decode('utf-8')
18     xml_str = xml_obj[(108):len(xml_obj) - 38]
19     tree = ElementTree(fromstring(xml_str))
20
21     root = tree.getroot()
22     xmldict = xp.XmlDictConfig(root)
23
24     #Json from xml
25     to_db = json.dumps(xmldict)
26     to_db_dict = json.loads(to_db)
27
28     #Creation of the key as Name of the event + Date of the event
29     name = to_db_dict["Event"]["Name"] + to_db_dict["Event"]["StartTime"]["Date"]
30     table = dynamodb.Table('DBresults')
31
32     #Serialization of the data
33     serializer = TypeSerializer()
34     to_db_dict_serialized = serializer.serialize(to_db_dict)
35
36     #Insert of the Item inside the db
37     response = table.put_item(Item = {"event": name, **to_db_dict})
38
39     print(response)
```



# Funzioni Lambda

```
1 import logging
2 import base64
3 import boto3
4 import os
5 import xml.etree.ElementTree as ET
6 import re
7
8 logger = logging.getLogger()
9 logger.setLevel(logging.INFO)
10
11 s3_client = boto3.client('s3')
12 s3_resource = boto3.resource('s3')
13
14 response = {
15     'statusCode': 200,
16     'headers': {
17         'Access-Control-Allow-Origin': '*',
18         'Access-Control-Allow-Credentials': 'true'
19     },
20     'body': ''
21 }
22 j = 0
23
24 def lambda_handler(event, context):
25
26     BUCKET_NAME = 'xmlresultgreco'
27     bucket = s3_resource.Bucket(BUCKET_NAME)
28
29     prefix = 'partite/'
30
31     def name_control(file_name):
32         for o in bucket.objects.all():
33             if (prefix+file_name) == o.key:
34                 file_name = 'new_' + file_name
35                 file_name = name_control(file_name)
36         return file_name
37
38     files_name = []
39     files_number = event['headers']['fileNumber']
40     for i in range(int(files_number)):
41         files_name.append(event['headers']['filename-'+str(i)])
42
43     file_content = base64.b64decode(event['body'])
44
45     content_decoded = file_content.decode("utf-8")
46     content_split = content_decoded.split('FLAGSEPARATORCODE')
47     content_split.pop(len(content_split)-1)
48
49     data = ""
50     j = 0
51     response['body'] = 'The following files have been uploaded:'
52
53     for file in content_split:
54
55         file_name = name_control(files_name[j])
56         j += 1
57
58         data = file.split('\r\n\r\n')
59         data.pop(0)
60         content = data[0]
61
62         root = ET.fromstring(content)
63         root.attrib.clear()
64         xmlstr = ET.tostring(root, encoding='utf8', method='xml')
65         xmlstr = xmlstr.decode("utf8")
66         content = re.sub(' xmlns:ns0="[^"]+"', '', xmlstr, count=1)
67         content = content.replace('ns0:', '')
68
69         try:
70             s3_response = s3_client.put_object(Bucket=BUCKET_NAME, Key=(prefix+file_name), Body=content)
71             logger.info('S3 Response: {}'.format(s3_response))
72             response['body'] += "\n" + file_name
73
74         except Exception as e:
75             raise IOError(e)
76
77     return response
```

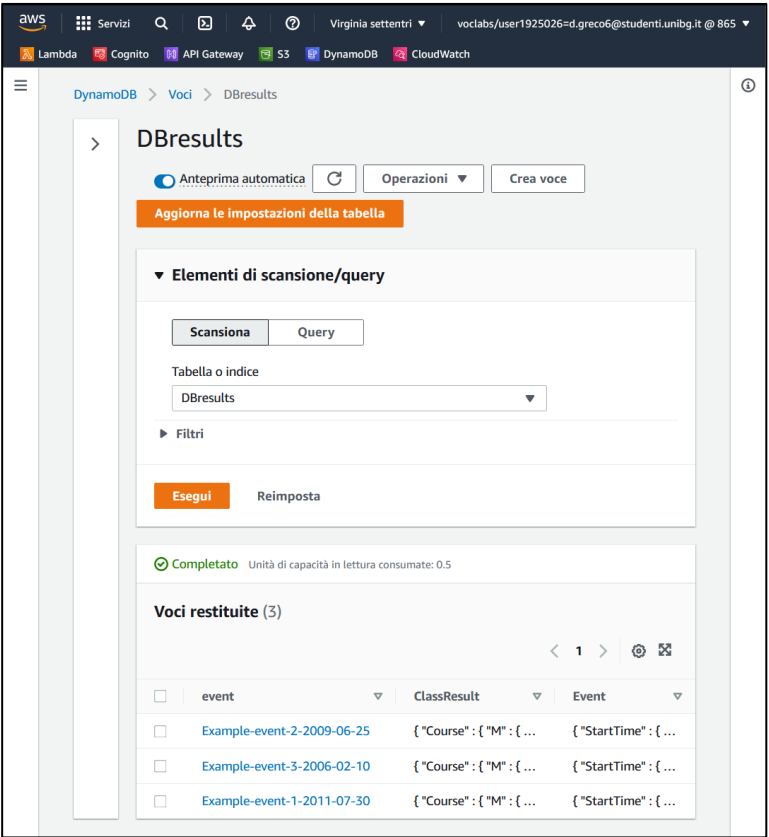
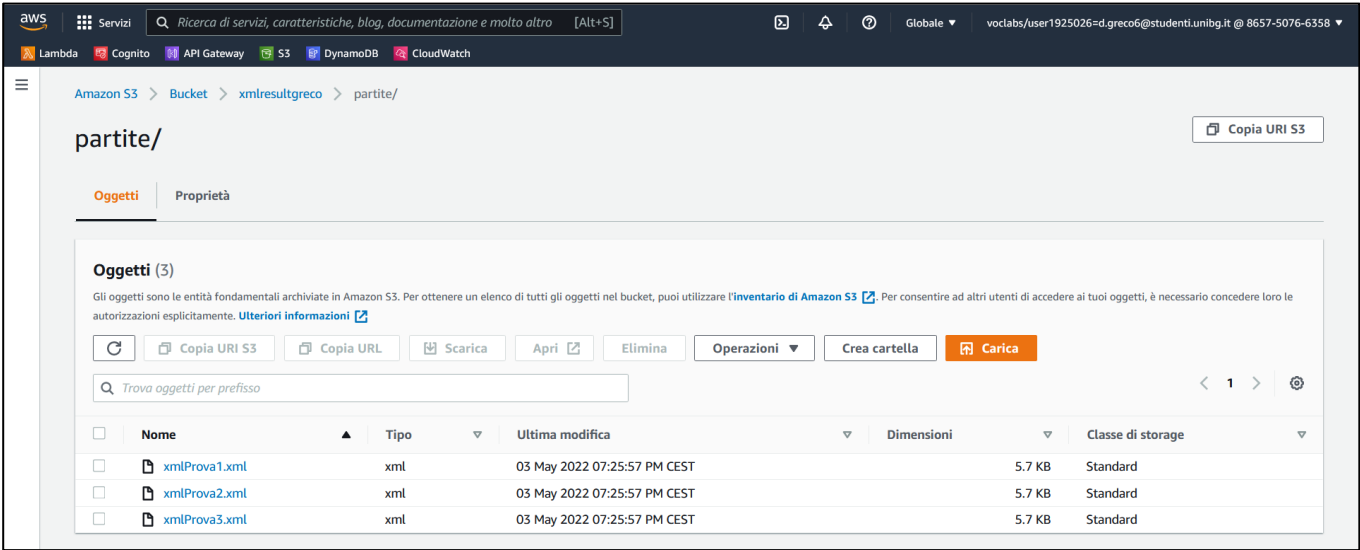
```
1 import logging
2 import json
3 import boto3
4 from boto3.dynamodb.conditions import Key
5
6 logger = logging.getLogger()
7 logger.setLevel(logging.INFO)
8
9 dynamodb = boto3.resource('dynamodb')
10 ddb_client = boto3.client('dynamodb')
11
12 response = {
13     'statusCode': 200,
14     'headers': {
15         'Access-Control-Allow-Origin': '*',
16         'Access-Control-Allow-Credentials': 'true'
17     },
18     'body': ''
19 }
20
21 def lambda_handler(event, context):
22
23     mod = event['headers']['mod']
24     response['body'] = ''
25
26     try:
27         table = dynamodb.Table('DBResults')
28
29         if(mod == 'read'):
30
31             primary_keys= []
32             count = 0
33             r = ddb_client.scan(
34                 TableName='DBResults',
35                 AttributesToGet=[
36                     'event',
37                 ],
38             )
39             count += r['Count']
40             for i in r['Items']:
41                 primary_keys.append(i['event']['S'])
42             for key in primary_keys:
43                 response['body'] += key + '\n'
44
45             elif(mod == 'download'):
46                 #Table definition
47                 #Getting the key
48                 key = event["headers"]["filename"]
49                 #Query
50                 item = table.get_item(Key={'event': key})
51                 #Converting the item <dict> to json <string>
52                 item_json = json.dumps(item)
53                 item_json = item_json.split(' ', "ResponseMetadata:")
54                 item_json = item_json[0] + "}"
55                 #Item return when the query has been possible (statusCode:200)
56                 response['body'] = item_json
57
58             else:
59                 response['body'] = 'Error 404: "mod" header not found'
60
61         except Exception as e:
62             raise IOError(e)
63
64     return response
```

## S3 Bucket e Dynamo DB

All'aggiunta di un file nel bucket viene avviata una funzione Lambda che prende il contenuto del file in formato «xml». Questo viene convertito ed inserito in formato «json» all'interno di una tabella nel database costruito utilizzando il servizio Dynamo DB. Ogni Gara è riconosciuta univocamente da una chiave composta dal nome e dalla data dell'evento.

Quando un utente *User* effettua una richiesta GET, viene preso l'«item» relativo al nome dell'evento desiderato dall'utente, il quale viene inserito nell'«*header*» della richiesta e restituito nel «*body*» della risposta.

# S3 Bucket e Dynamo DB



## Simulazione degli eventi

La simulazione degli eventi avviene attraverso l'esecuzione di uno script python, il quale, attraverso la libreria «xml.etree», genera un documento pseudo-casuale secondo lo standard «IOF-DataStandard 3.0».

La creazione del/dei file viene effettuata a livello basso, componendo dei sotto-alberi che vengono solo alla fine ricomposti; Nel caso fosse necessario generare più file, questi vengono immagazzinati in una variabile di classe «list», la quale viene iterata per effettuare il salvataggio dei singoli elementi.

Il nome dei file viene generato in base al nome dell'evento più la sua «StartDate».

Tali file vengono salvati unicamente in una cartella «Result\_Of\_Simulation», la quale viene ripulita a seguito del loro invio tramite «POST-request» seguendo le dinamiche descritte.

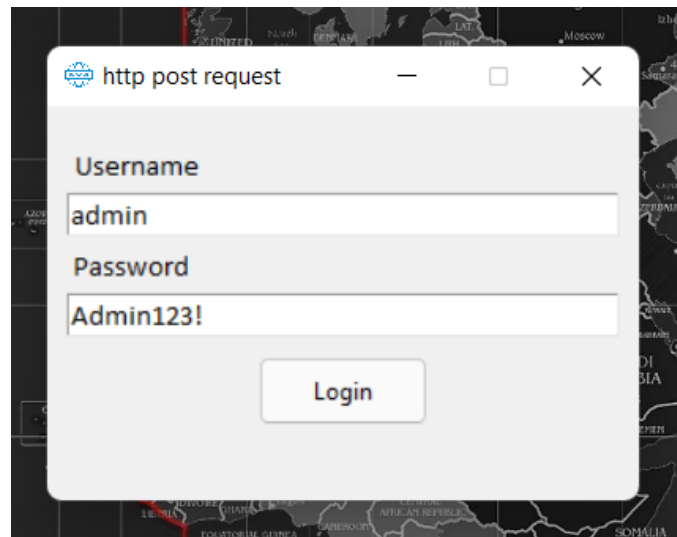
# Testing

Per eseguire i vari test per accedere al bucket e al database sono state scritte due applicazioni diverse in python, una per gli utenti *Administrator* e una per gli utenti *Users*.

L'applicazione «*uploaderApp*»: contiene un modulo per il login, i dati vengono passati ad Amazon Cognito che autentica l'utente e restituisce un token di accesso che verrà inserito nell' «header» di ogni richiesta POST effettuata. Dopo l'accesso, l'applicazione consente di generare più file sul file system locale ed inviarli in un'unica *POST* «*request*» all'endpoint con percorso «*/update*». L'API verifica il token di accesso con Amazon Cognito e inoltra la richiesta alla funzione lambda.

L'applicazione «*readerApp*»: dopo il login viene inviata automaticamente una richiesta *GET* che ha come intestazione un parametro «*mod*» impostato a «*read*». La funzione lambda risponde quindi con la lista dei nomi degli eventi presenti sul database. All'avvio si presenta quindi all'utente tale lista, ed egli si limita ad inserire nella casella di testo il nome dell'evento da scaricare ed invia una richiesta *GET* con parametro «*mod*» impostato a «*download*». La funzione lambda cerca quindi il file richiesto nel bucket e lo restituisce nel *body* della risposta. L'applicazione scrive quindi il contenuto in un file json in locale nella cartella «*downloads*» presente nella directory del progetto.

# Testing

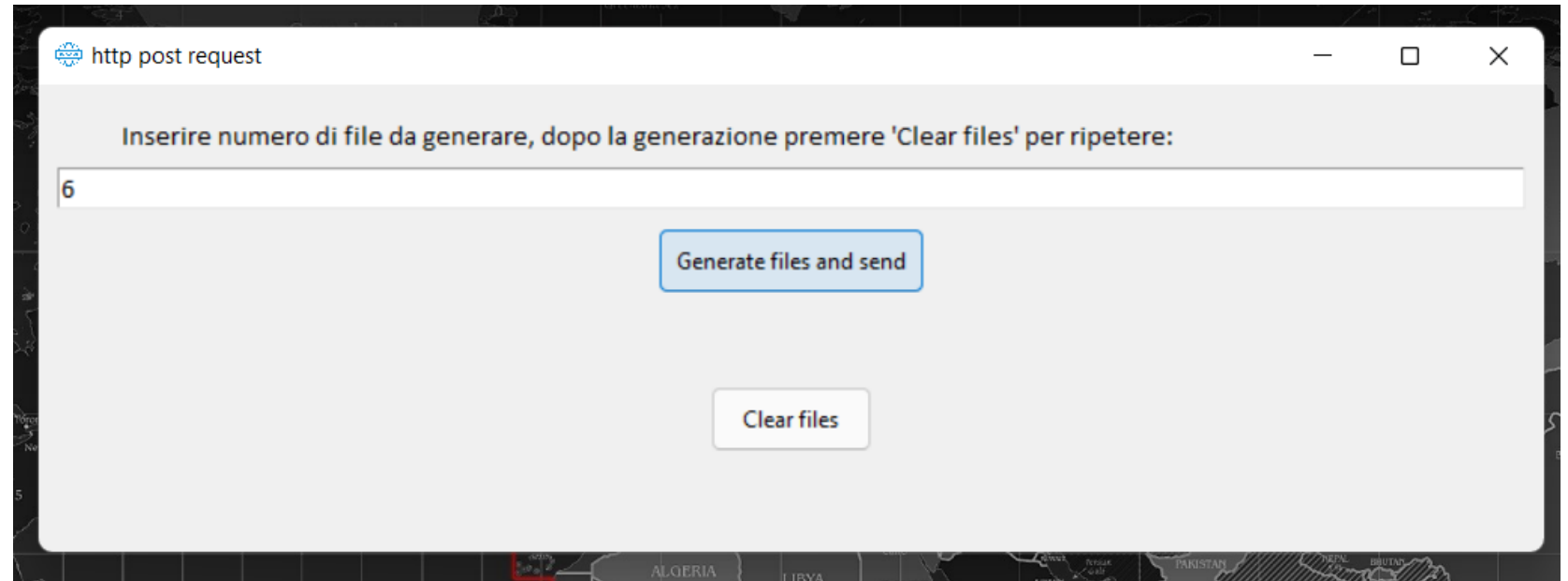


http post request

Username  
admin

Password  
Admin123!

Login



http post request

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

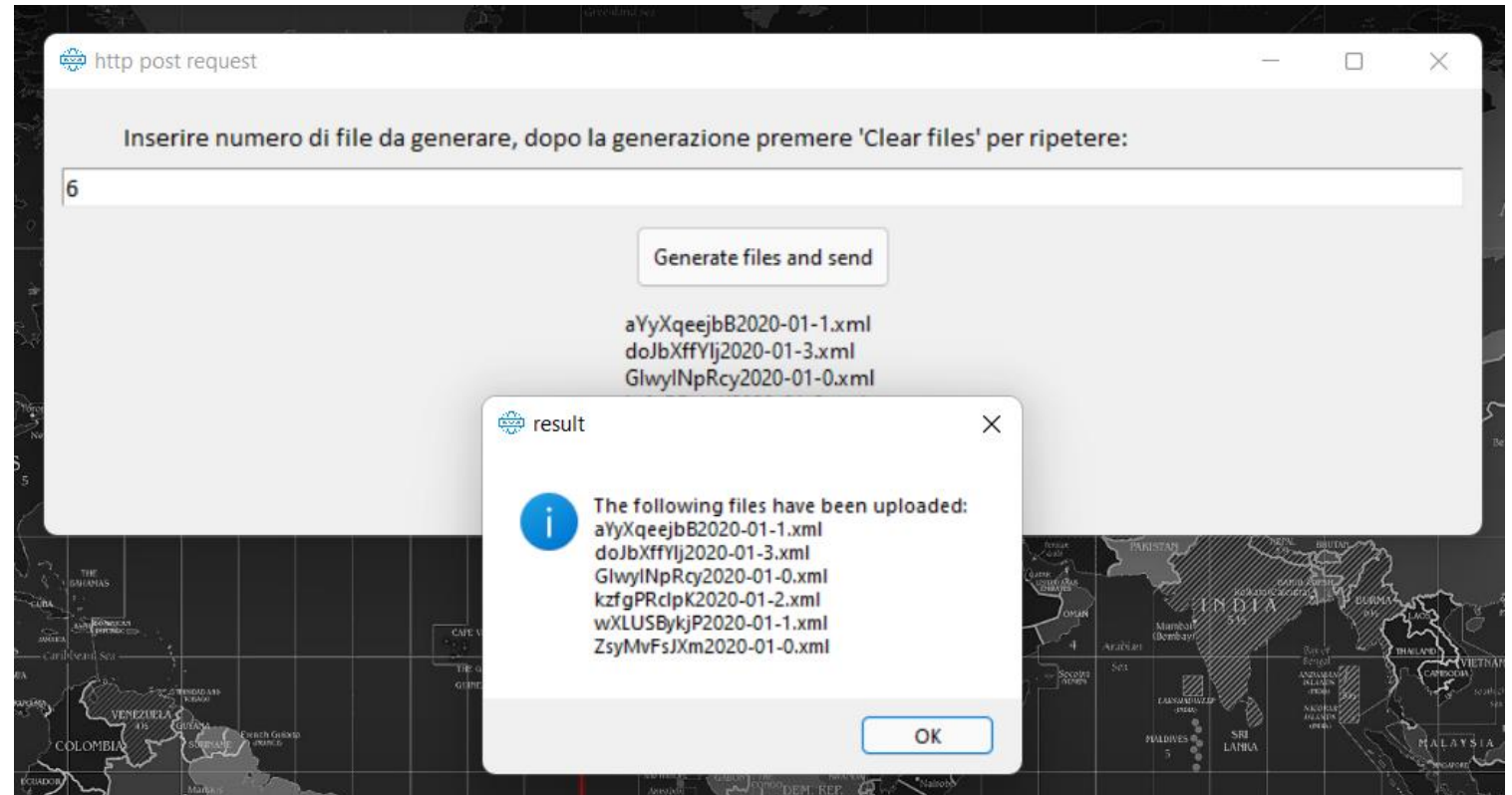
6

Generate files and send

Clear files



# Testing



# Testing result

**DynamoDB** > Voci > DResults

> **DResults**

Anteprima automatica | Operazioni ▼ | Crea voce | Aggiorna le impostazioni della tabella

▼ Elementi di scansione/query

Scansione Query

Tabella o indice  
DResults ▼

Filtri

Esegui Reimposta

Completato Unità di capacità in lettura consumate: 3

Voci restituite (6)

event	ClassResult	Event
doJbXfYq2020-01-3	{ "Course": { "Length": { "S": "1353"}, "Climb": { "S": "228"} }, "PersonResult": { "M": { "Organisation": ...	{ "StartTime": { "M": { "Date": { "S": "2020-01-3"}, "Time": { "S": "02:56:31"} }, "EndTime": { "M": { "Date": { "S": "2020-01-2"}, "Time": { "S": "05:45:42"} } }, "Name": { "...
wXLUSySkp2020-01-1	{ "Course": { "Length": { "S": "451"}, "Climb": { "S": "179"} }, "PersonResult": { "M": { "Organisation": ...	{ "StartTime": { "M": { "Date": { "S": "2020-01-1"}, "Time": { "S": "09:58:16"} }, "EndTime": { "M": { "Date": { "S": "2020-01-2"}, "Time": { "S": "19:24:27"} } }, "Name": { "...
aYyXqeBz2020-01-1	{ "Course": { "Length": { "S": "1546"}, "Climb": { "S": "853"} }, "PersonResult": { "M": { "Organisation": ...	{ "StartTime": { "M": { "Date": { "S": "2020-01-1"}, "Time": { "S": "19:37:29"} }, "EndTime": { "M": { "Date": { "S": "2020-01-0"}, "Time": { "S": "23:04:39"} } }, "Name": { "...
ZyMvMsxm2020-01-0	{ "Course": { "Length": { "S": "709"}, "Climb": { "S": "57"} }, "PersonResult": { "M": { "Organisation": { ...	{ "StartTime": { "M": { "Date": { "S": "2020-01-0"}, "Time": { "S": "08:36:12"} }, "EndTime": { "M": { "Date": { "S": "2020-01-0"}, "Time": { "S": "02:20:57"} } }, "Name": { "...
GhwYlRpRoz2020-01-0	{ "Course": { "Length": { "S": "1419"}, "Climb": { "S": "924"} }, "PersonResult": { "M": { "Organisation": ...	{ "StartTime": { "M": { "Date": { "S": "2020-01-0"}, "Time": { "S": "14:20:06"} }, "EndTime": { "M": { "Date": { "S": "2020-01-2"}, "Time": { "S": "00:02:03"} } }, "Name": { "...
kzfjgRkcp2020-01-2	{ "Course": { "Length": { "S": "798"}, "Climb": { "S": "173"} }, "PersonResult": { "M": { "Organisation": ...	{ "StartTime": { "M": { "Date": { "S": "2020-01-2"}, "Time": { "S": "13:31:13"} }, "EndTime": { "M": { "Date": { "S": "2020-01-1"}, "Time": { "S": "08:31:01"} } }, "Name": { "...

zi, caratteristiche, blog, documentazione e molto altro

[Alt+S]

S3

DynamoDB

CloudWatch

Amazon S3

>

Bucket

>

xmlresultgreco

>

partite/

partite/

Copia URI S3

Oggetti

Proprietà

Oggetti (6)

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

🔍 Trova oggetti per prefisso

< 1 > ⚙️

<input type="checkbox"/>	Nome	Tipo	Ultima modifica	Dimensioni	Classe di storage
<input type="checkbox"/>	<a href="#">aYyXqeejbB2020-01-1.xml</a>	xml	03 May 2022 09:56:11 PM CEST	3.8 KB	Standard
<input type="checkbox"/>	<a href="#">doJbXffYlj2020-01-3.xml</a>	xml	03 May 2022 09:56:11 PM CEST	5.4 KB	Standard
<input type="checkbox"/>	<a href="#">GIwylNpRcy2020-01-0.xml</a>	xml	03 May 2022 09:56:11 PM CEST	5.4 KB	Standard
<input type="checkbox"/>	<a href="#">kzfgPRclpK2020-01-2.xml</a>	xml	03 May 2022 09:56:11 PM CEST	7.1 KB	Standard
<input type="checkbox"/>	<a href="#">wXLUSBykjP2020-01-1.xml</a>	xml	03 May 2022 09:56:12 PM CEST	5.4 KB	Standard
<input type="checkbox"/>	<a href="#">ZsyMvFsJXm2020-01-0.xml</a>	xml	03 May 2022 09:56:12 PM CEST	8.8 KB	Standard





# GitHub

Link alla repository con le applicazioni *uploaderApp* e *readerApp* visibile pubblicamente:

[https://github.com/MatiasNegro/tcm\\_homeworks\\_and\\_project](https://github.com/MatiasNegro/tcm_homeworks_and_project)