



HOMEWORK2

Giudici Mattia 1065452

Trapletti Andrea 1066781

Verdi Michele 1067606



Tutte la parti di codice del progetto
si possono trovare nella repository
di Git

<https://github.com/Mattia-Giudici/TCM-TraplettiVerdiGiudici>

Lo scheletro dell'architettura cloud
è stato caricato su GitHub.

Primo punto

Abbiamo modificato l'api aggiungendo la risorsa register_race e il metodo POST collegato alla funzione lambda: "register_race.py" (codice su Git).

▼ /

▼ /lambda_tgv

POST

▼ /gare_tgv

GET

▼ /{proxy+}

GET

▼ /{proxy+}

GET

▼ /register_race

POST

▼ /uploadxml

POST

Fornisci informazioni sulle impostazioni di autorizzazione di questo metodo e i parametri che può ricevere.

Impostazioni

Autorizzazione

NESSUNO

Validatore richiesta

Convalida parametri di stringa query e intestazioni

Chiave API necessaria

false

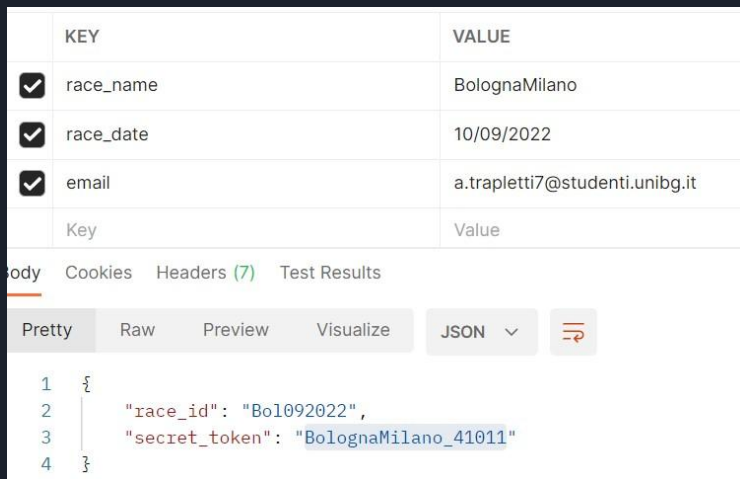
▼ Parametri della stringa di query URL

Nome	Campo obbligatorio
email	<input checked="" type="checkbox"/>
race_date	<input checked="" type="checkbox"/>
race_name	<input checked="" type="checkbox"/>

+

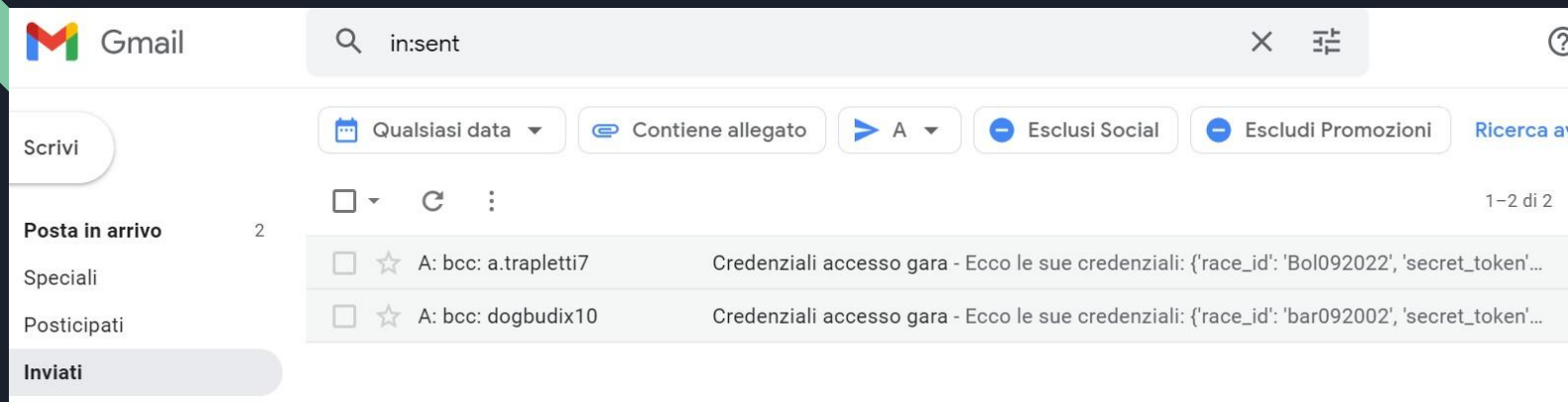
[Aggiungi stringa di query](#)

Postman:



L'utente A (adibito al caricamento della gara BolognaMilano), dovrà tenere il race_id (composto dalle prime 3 lettere della gara e la data della gara stessa) e il secret_token generato dalla funzione lambda.

La funzione lambda, inoltre, manda una email al richiedente con id_gara e secret_token.



Questa è la casella di “posta inviata” di una email creata ad hoc per mandare i parametri agli utenti:

orienteering.infopoint@gmail.com

La funzione manda i parametri alla mail inserita dall'utente che inserisce durante la richiesta di POST.



Il token della gara viene tenuto in un apposito DB (DynamoDB, “Token_Table”).

<input type="checkbox"/>	race_id ▾	email ▾	raceDate ▾	raceName ▾	SecretToken
<input type="checkbox"/>	Ber092022	a.trapletti7...	['12', '09', '2...	BergamoBr...	BergamoBrescia_59809
<input type="checkbox"/>	Bol092022	a.trapletti7...	['10', '09', '2...	BolognaMil...	BolognaMilano_60866

Secondo punto

Sottoscriviamo un nuovo endpoint uploadxml con metodo POST:

The screenshot shows an API management console interface. On the left, a tree view lists endpoints: /gare/gv (GET), /{{proxy+}} (GET), /{{proxy+}} (GET), /register_race (POST), and /uploadxml (POST). The /uploadxml endpoint is selected and highlighted in blue. The main panel displays the configuration for this endpoint:

- Autorizzazione:** api-authorizer (with edit and info icons)
- Validatore richiesta:** Convalida parametri di stringa query e intestazioni (with edit and info icons)
- Chiave API necessaria:** false (with edit icon)
- Parametri della stringa di query URL:** (with add icon)

Nome	Campo obbligatorio
race_id	<input checked="" type="checkbox"/>


L'endpoint uploadxml è quindi collegata alla funzione lambda "Uploadxml.py". L'endpoint, inoltre, richiede l' "Autorizzazione": "api-authorizer", ovvero l'autorizzazione creata appositamente da noi durante il primo homework. L'autorizzazione della nostra API è collegata alla funzione lambda "APIauthorizer.py" (il codice di tale lambda ha subito delle leggere modifiche rispetto al primo homework, si può trovare la versione aggiornata nella repository git sotto homework2).


L'utente A per caricare i file della gara nel bucket s3 "garetyv" e nel DB GareOrienteering, dovrà fare una richiesta POST aggiungendo nelle Headers (di Postman per esempio), nel campo key: "jwt_token" e come Value l'id della gara seguito da un "-" e dal token di autorizzazione concesso in precedenza.

<input checked="" type="checkbox"/>	Connection	keep-alive
<input checked="" type="checkbox"/>	jwt_token	Ber092022-BergamoBrescia_27983

Body Cookies Headers (7) Test Results

La lambda function controlla, utilizzando il DB, la chiave di inserimento (race + token). Se la richiesta viene accettata, il file xml viene diviso per class result e inserito nel bucket "garetyv".

<input type="checkbox"/>	 Ber092022Men Elite.xml	xml
<input type="checkbox"/>	 Bol092022Open.xml	xml
<input type="checkbox"/>	 Bol092Men Elite.xml	xml



La gara viene aggiunta sia nel bucket che nel DB “GareOrienteering”. La funzione lambda garantisce che il file viene aggiunto nel bucket quando arriva la prima chiamata (al tempo X) dell’utente A.

Se l’utente A vuole inserire un aggiornamento della gara ad un tempo $X + Y$, il file inserito precedentemente viene sostituito nel bucket con l’ultima versione aggiornata.

TERZO PUNTO: il terzo punto è già stato spiegato in queste slides, infatti è stato mostrato un utente per la gestione della gara BolognaMilano e uno per la gara BergamoBrescia.

Gestione dei metodi GET (punti: 5, 6, 7, 9)

Creiamo un nuovo endpoint con metodo GET (impostazioni → proxy), questo endpoint, chiamato “list_races”, manda in output la lista delle gare registrate.

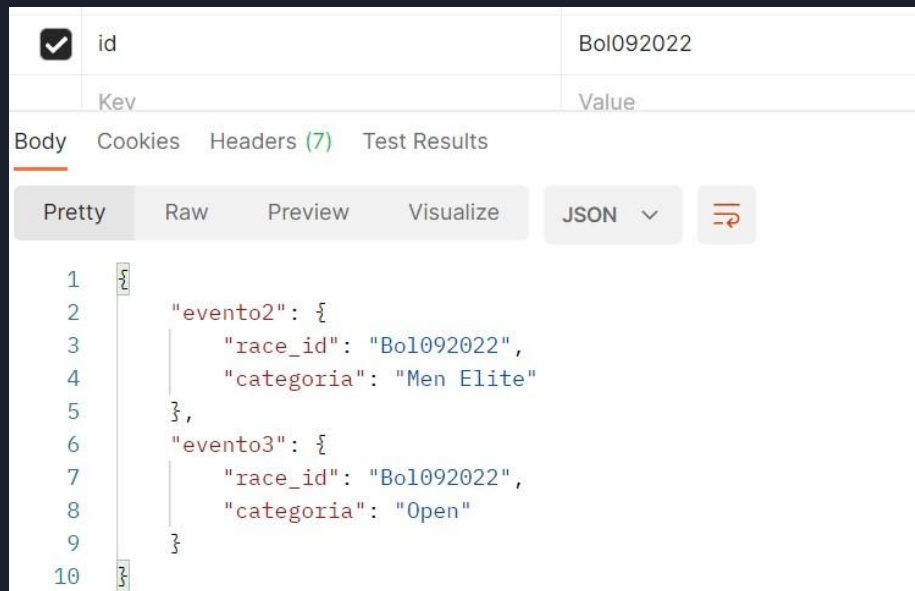
```
1  {}
2      "evento1": {
3          "race_name": "BergamoBrescia",
4          "race_date": "['12', '09', '2022']",
5          "race_id": "Ber092022"
6      },
7      "evento2": {
8          "race_name": "BolognaMilano",
9          "race_date": "['10', '09', '2022']",
10         "race_id": "Bol092022"
11     }
12 }
```

In GETfunction.py si trova il codice per gestire le richieste GET.

Creiamo endpoint per la gestione di list_classes: metodo GET (impostazioni → proxy).

Impostazioni di “Esecuzione metodo”:

Output postman:



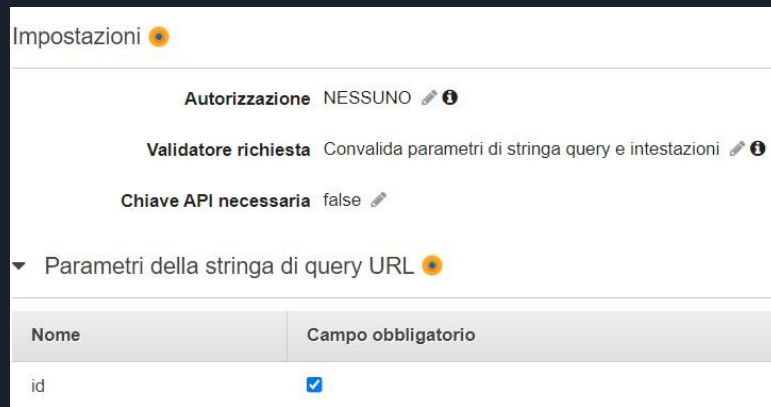
A screenshot of the Postman interface showing the results of a GET request. The 'id' parameter is set to 'Bo1092022'. The response body is displayed in JSON format, showing two event objects: 'evento2' and 'evento3'. 'evento2' has 'race_id' 'Bo1092022' and 'categoria' 'Men Elite'. 'evento3' has 'race_id' 'Bo1092022' and 'categoria' 'Open'.

id	Value
Bo1092022	

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON

```
1 {
2   "evento2": {
3     "race_id": "Bo1092022",
4     "categoria": "Men Elite"
5   },
6   "evento3": {
7     "race_id": "Bo1092022",
8     "categoria": "Open"
9   }
10 }
```



A screenshot of the Postman 'Impostazioni' (Settings) dialog for a GET request. The 'Autorizzazione' (Authorization) is set to 'NESSUNO' (None). The 'Validatore richiesta' (Request Validator) is set to 'Convalida parametri di stringa query e intestazioni' (Validate query string parameters and headers). The 'Chiave API necessaria' (API Key required) is set to 'false'. The 'Parametri della stringa di query URL' (URL query string parameters) section is expanded, showing a table with one parameter: 'id' with a checked 'Campo obbligatorio' (Required) checkbox.

Impostazioni

Autorizzazione NESSUNO


Validatore richiesta Convalida parametri di stringa query e intestazioni



Chiave API necessaria false



Parametri della stringa di query URL


Nome	Campo obbligatorio
id	<input checked="" type="checkbox"/>


L'endpoint "results" è utilizzato sia per il punto 7 che per il 9. Metodo GET (impostazioni → proxy).

Impostazioni 

Autorizzazione NESSUNO  


Validatore richiesta Convalida parametri di stringa query e intestazioni  

Chiave API necessaria false 

▼ Parametri della stringa di query URL 

Nome	Campo obbligatorio
Categoria	<input checked="" type="checkbox"/>
id	<input checked="" type="checkbox"/>
organisation	<input type="checkbox"/>

Si può notare che "organization" è reso non obbligatorio, quindi se dalla richiesta postman arrivano 2 parametri query allora la funzione lambda gestisce la funzionalità del punto 7, altrimenti il punto 9.



Punto 7. Output postman: la classifica è stilata in ordine crescente di tempi.

```
1  {
2    "atleta1": {
3      "nome": "Allison",
4      "cognome": "Jenkins",
5      "tempo": 1025,
6      "N_controllo": 3
7    },
8    "atleta2": {
9      "nome": "Jim",
10     "cognome": "Hughes",
11     "tempo": 1149,
12     "N_controllo": 3
13   },
14   "atleta3": {
15     "nome": "Penny",
16     "cognome": "Jenkins",
17     "tempo": 1025,
18     "N_controllo": 3
19   }
20 }
```

Punto 9. Output Postman

Params ☒ Authorization Headers (5) Body Pre-request Script

<input checked="" type="checkbox"/> id	Ber092022
<input checked="" type="checkbox"/> Categoria	Open
<input checked="" type="checkbox"/> organization	Forest Vagabonds

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON

```
1 [
2   "Adams Brenda",
3   "Collins Anne",
4   "Fisher Penny",
5   "Hall Brent",
6   "Jenkins Allison",
7   "Reid Samuel"
8 ]
```

<input checked="" type="checkbox"/> id	Ber092022
<input checked="" type="checkbox"/> Categoria	Open
<input checked="" type="checkbox"/> organization	Super Fast Orienteers

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON

```
1 [
2   "Hughes Jim",
3   "Parker Jeffrey"
4 ]
```

Ottavo punto

Siccome si chiede come risposta dalla richiesta GET un file in download, abbiamo utilizzato un endpoint che supportasse un metodo GET ma senza le impostazione di proxy. Vista quest'ultima differenza abbiamo deciso di utilizzare come collegamento con l'endpoint una nuova funzione lambda: "download_xml.py". Le impostazioni dell'endpoint:

. Esecuzione metodo:

Impostazioni

Autorizzazione NESSUNO

Validatore richiesta Convalida parametri di stringa query e intestazioni

Chiave API necessaria false

Parametri della stringa di query URL

Nome	Campo obbligatorio
id	<input checked="" type="checkbox"/>

. Richiesta di integrazione → modelli di mappatura.

▼ Modelli di mappatura

Richiesta corpo passthrough

☐ Quando nessun modello corrisponde all'intestazione Content-Type della richiesta

☒ Quando non ci sono modelli definiti (consigliato) ⓘ

☐ Mai ⓘ

Content-Type
application/json

+ Aggiungi modello di mappatura

application/json

Genera modello: ▼

```
1 ## See http://docs.aws.amazon.com/apigateway/latest/developerguide/api-gateway
  -mapping-template-reference.html
2 ## This template will pass through all parameters including path, querystring,
  header, stage variables, and context through to the integration endpoint via the
  body/payload
3 #set($allParams = $input.params())
4 {
5   "body-json" : $input.json('$'),
6   "params" : {
7     #foreach($type in $allParams.keySet())
8       #set($params = $allParams.get($type))
9       "$type" : {
10        #foreach($paramName in $params.keySet())
11          "$paramName" : "$util.escapeJavaScript($params.get($paramName))"
12        #if($foreach.hasNext),#end
13      }#end
14    }#end
15 }
```


. Risposta metodo

Stato HTTP

▼ 200

Intestazioni di risposta per 200

Nome
Content-Disposition

+

[Aggiungi intestazione](#)

Corpo risposta per 200

Tipo di contenuto	Modelli
application/xml	Empty

+

[Aggiungi modello risposta](#)

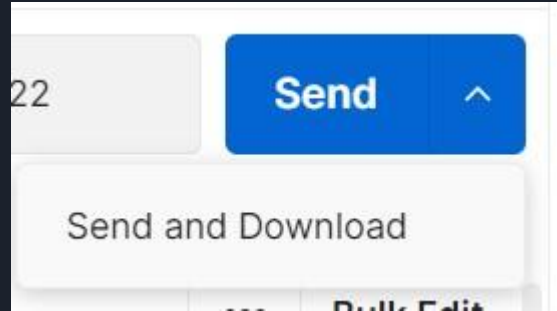
In “Nome” aggiungere Content-Disposition, in “Tipo di contenuto” rimuovere application/json e aggiungere application/xml.

. Risposta di integrazione → mappatura di integrazione

▼ Mappature intestazione	
Intestazione della risposta	Valore mappatura ⓘ
Content-Disposition	'attachment; filename="Result.xml"'

Determina con che nome verrà salvato l'xml in download (in questo caso "Result.xml").

In postman se viene mandata la richiesta downloadxml, bisogna cambiare “send” con “send and download”.



Se la richiesta viene mandata tramite url direttamente dal browser allora il file viene scaricato in locale immediatamente.