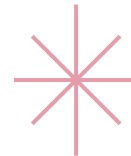
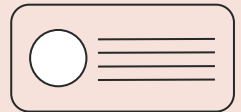
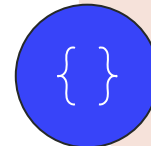
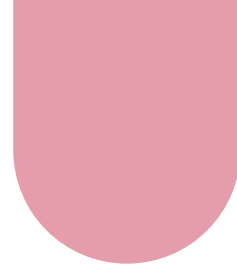




CommuniTEDx Applicazione Mobile

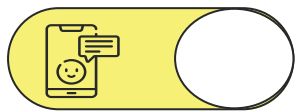
Progetto per il corso di
tecnologie Cloud e Mobile

Samuel Locatelli, mat: 1054674
Giorgio Tentori, mat: 1053248



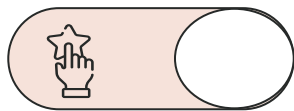
Descrizione e obiettivo di CommuniTEDx

CommuniTedx è un'applicazione di intrattenimento che ha come obiettivo principale quello di fornire agli utenti i contenuti a cui sono più interessati e creare una connessione con altri utenti con interessi comuni.



Community

Consente di conoscere e interagire con altri utenti della piattaforma tramite chat e collegamenti



Divulgazione

Favorisce la divulgazione scientifica e culturale attraverso la condivisione di video TEDx



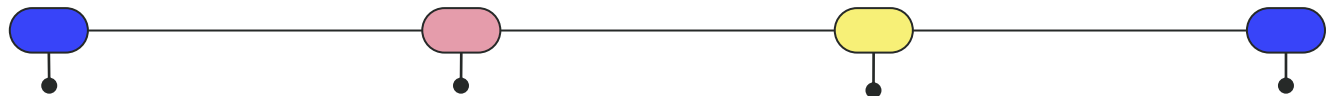
Intrattenimento

Permette un accrescimento delle conoscenze senza tralasciare il divertimento





Funzioni principali



Ricerca e filtraggio

Si possono cercare video, resi disponibili dalla piattaforma, in base al titolo, ad un particolare topic o al relatore

Suggerimenti e amicizie

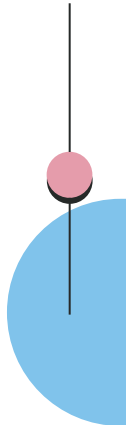
L'applicazione suggerisce determinati video in base alle proprie preferenze e consente di creare collegamenti con altri utenti

Salvataggio dei video TEDx

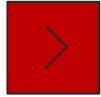
L'utente può creare, modificare ed eliminare playlist in cui è possibile salvare video TEDx a cui un utente è particolarmente interessato

Chat e condivisione

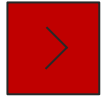
Possibilità di condividere video con i propri collegamenti con l'ulteriore possibilità di iniziare una conversazione



Criticità



Possibilità di conversazione con tutti gli utenti o solo con i collegamenti?



Quali sono i servizi a cui può accedere un utente non autenticato?



Possibilità di conversazione solamente con i propri collegamenti



Accesso solamente a video più visti in quel periodo per gli utenti non autenticati

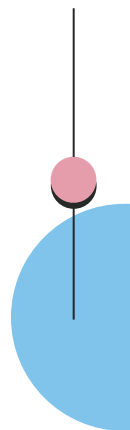





A chi è rivolto CommuniTEDx?

Questa applicazione è focalizzata su un pubblico con un forte interesse per la **scienza** e per la **tecnologia**, in particolare:

- Studenti delle scuole medie e superiori
- Studenti universitari
- Professori, docenti e insegnanti
- Ricercatori e scienziati
- Appassionati





Note sull'app e sull'interfaccia grafica


L'interfaccia grafica dell'applicazione mobile viene realizzata tramite l'ausilio del framework flutter.

L'obiettivo è quello di rendere l'intera piattaforma *user-friendly*, in modo da permetterne l'utilizzo anche agli utenti meno esperti.

L'applicazione è costruita utilizzando tecnologie cloud, come i *tools* di Amazon AWS.

• •
• •
• •
• •
• •
• •
• •
• •





Approfondimento sul servizio di autenticazione alla piattaforma: Amazon Cognito

Per la funzione di autenticazione viene utilizzato il servizio Amazon Cognito. Esso si basa su due concetti principali:

- pool di utenti
- pool di identità

I pool di utenti servono per l'autenticazione (verifica dell'identità) mentre i pool di identità servono per l'autorizzazione (controllo degli accessi).

• •
• •
• •
• •
• •
• •
• •
• •



Puoi utilizzare i pool di identità per creare identità univoche per gli utenti e consentire loro l'accesso ad altri servizi AWS.

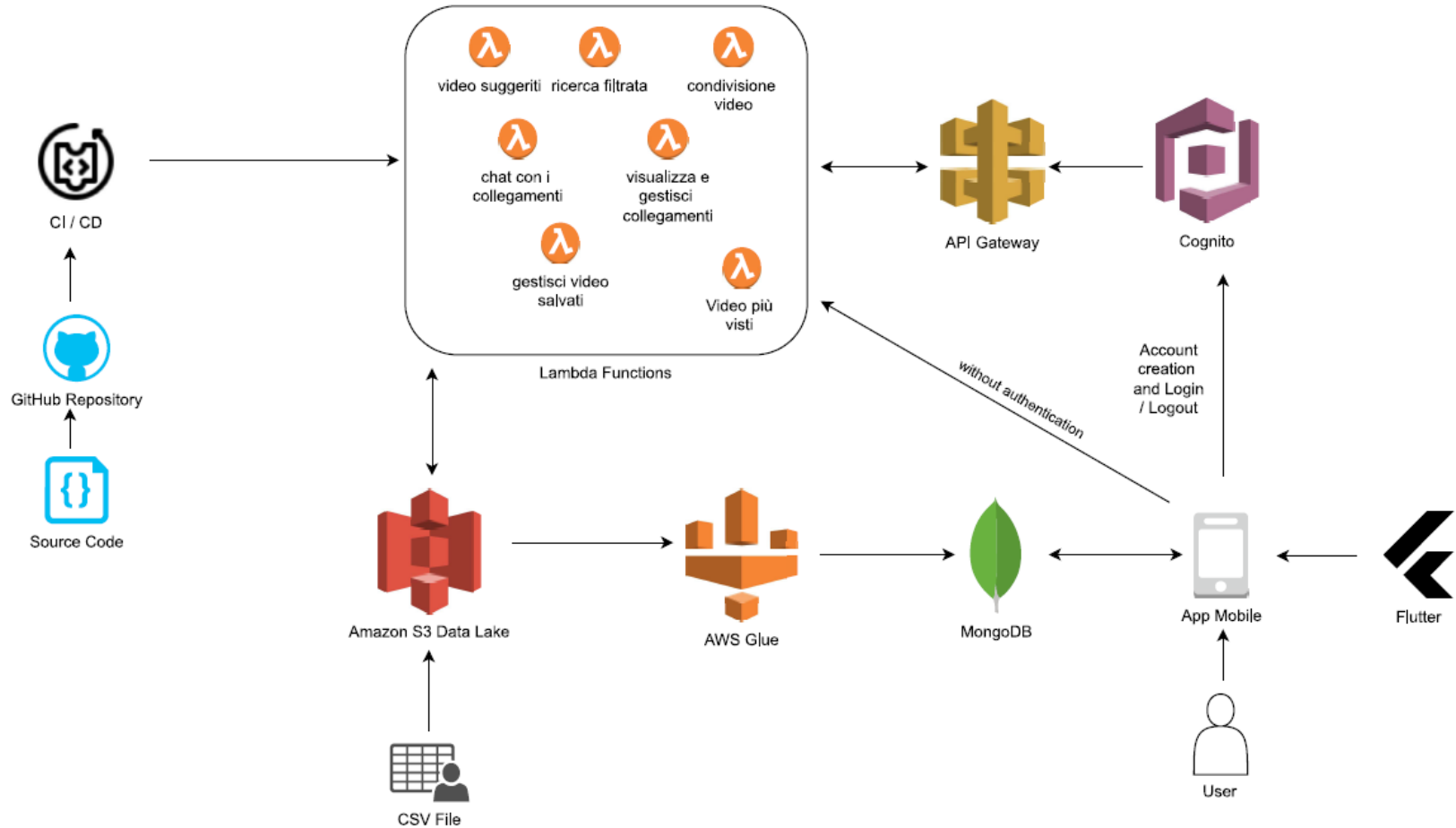
Gli utenti dell'app possono accedere tramite il pool di utenti o accedere in modalità federata, in particolare Amazon Cognito implementa lo standard *OAuth 2* che permette l'autenticazione tramite un gestore dell'identità digitale (IdP) di terze parti come Google o Facebook.

Queste funzionalità di Amazon Cognito sono utili perché permettono di:

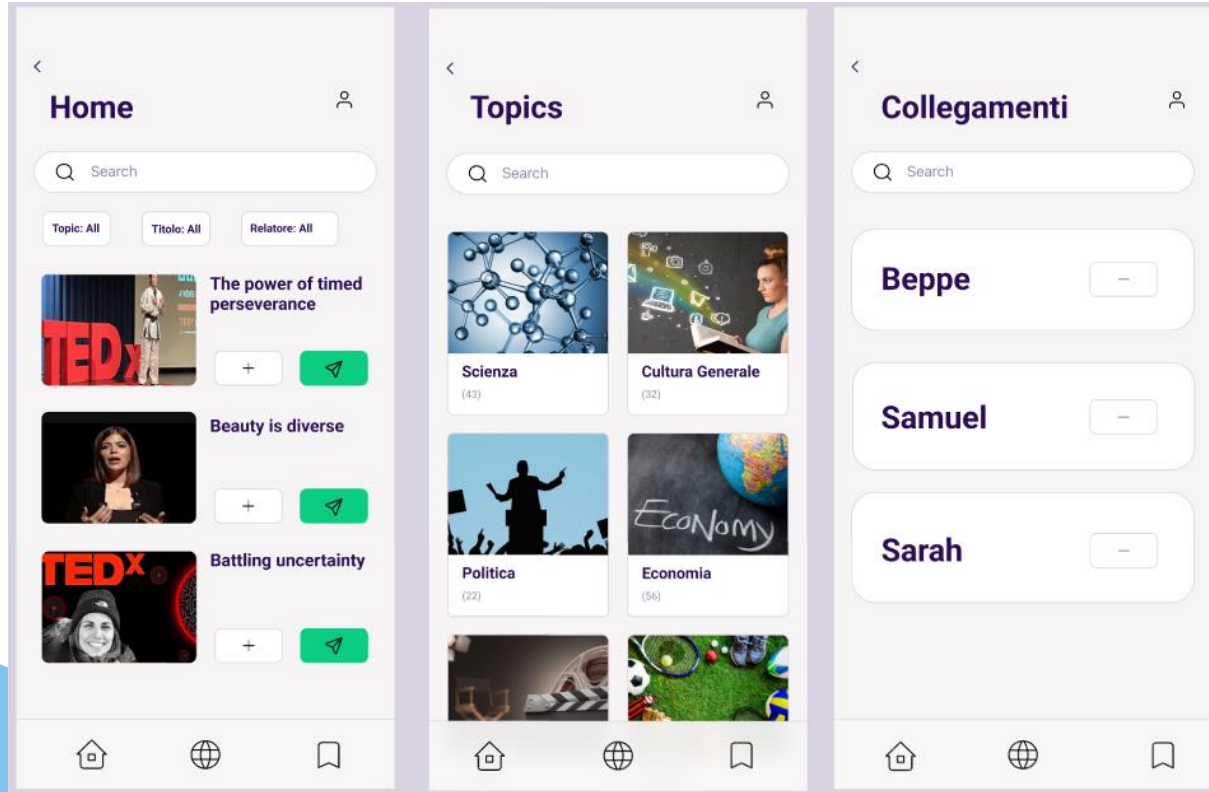
- Progettare pagine web di iscrizione e accesso per l'app tramite un pool di utenza.
- Generare credenziali AWS temporanee per utenti non autenticati tramite un pool di identità.

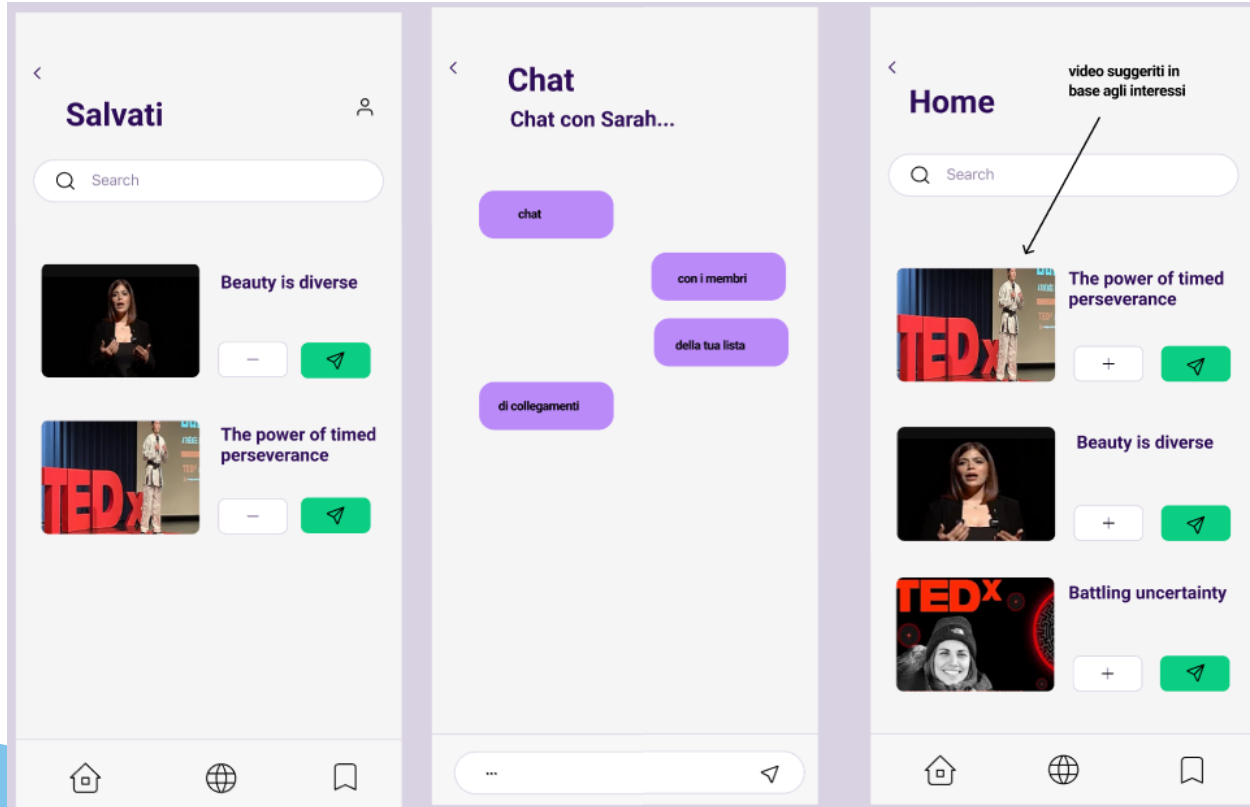
• •
• •
• •
• •
• •
• •
• •
• •

Architettura



Presentazione dell'interfaccia grafica





<https://www.figma.com/>

Trello Board

<https://trello.com/invite/b/0XzzqBXs/ATTI3aacf7431c519460e792bc42dcb6309d0EA2C9CA/communitedx>

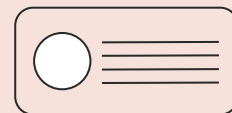
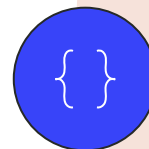
The screenshot shows a Trello board with four columns. The first column, 'Fatto', lists completed tasks. The subsequent columns, '01', '02', '03', and '04', list tasks in progress or planned. The task 'Disegnare l'interfaccia grafica' in the '01' column is highlighted with a blue border.

Fatto	01	02	03	04
Riunione iniziale	Tecnologie da utilizzare	Definire il modello dei dati su MongoDB	Implementare la funzione cerca by video	Definire piano di test
Descrizione del servizio	Disegnare l'interfaccia grafica	Importare o caricare i dati su MongoDB	Design Lambda Functions	Design App - Flutter
Definire l'architettura	ridefinire gli obiettivi dell'app	Controllo qualità	API per ricercare i TED talk in base al topic	Release App
Funzionalità del servizio	approfondire i servizi utilizzati	Caricare i dati su S3	Implementare la funzione cerca by tag (API)	Implementare interfaccia Flutter
Presentazione in Power Point	definire i dati che servono in aggiunta a Tedx	+ Aggiungi una scheda	Login e logout	+ Aggiungi una scheda
Configurazione Git e GitHub con relativi comandi			Implementare la funzione playlist	
Punti critici del progetto			+ Aggiungi una scheda	
Valore del servizio				
Fabbisogni e necessità del cliente				
Identificare i potenziali clienti ed eventuali stakeholder				
+ Aggiungi una scheda				



PARTE 2

- 1) Aggiunta dei watch next videos
- 2) Gestione del dataset per rendere i dati conformi e coerenti con l'obiettivo dell'applicazione



Video suggeriti e Selezione dati

Nella seconda parte del progetto è stata implementata, in primo luogo, la possibilità di visualizzare i video associati ad un determinato Ted (watch next videos) e in secondo luogo la selezione dei dati presenti nel dataset in base ai criteri descritti nella prima parte del progetto, ovvero video Tedx relativi a **scienza** e **tecnologia**, scremando dall'intero insieme di dati solo quelli che servono per lo sviluppo di una parte del progetto.

A questo proposito viene inizializzato e implementato un job PySpark descritto in seguito.

Aggiunta del Related_videos Dataset

Il seguente blocco di codice aggiunge al dataset iniziale i video correlati ad un determinato talk.

```
## READ WATCH_NEXT DATASET
watch_next_dataset_path = "s3://communitedx-2024-data/related_videos.csv"
watch_next_dataset = spark.read.option("header","true").csv(watch_next_dataset_path)

# ADD WATCH_NEXT TO TEDX_DATASET
watch_next_dataset = watch_next_dataset.dropDuplicates()
watch_next_dataset_agg = watch_next_dataset.groupBy(col("id").alias("id_ref")).agg(collect_list("related_id").alias("id_next"),collect_list("title").alias("title_next"))
watch_next_dataset_agg.printSchema()

# AND JOIN WITH THE AGG TABLE
tedx_dataset_agg = tedx_dataset_agg.join(watch_next_dataset_agg, tedx_dataset_agg.id == watch_next_dataset_agg.id_ref, "left") \
    .drop("id_ref") \
    .select(col("id").alias("_id"), col("*")) \
    .drop("id") \

tedx_dataset_agg.printSchema()
```

Per ogni video Tedx si possono avere più talk correlati e si è deciso di indicare per ogni documento in MongoDB due vettori: l'id dei video correlati e il relativo titolo.

Descrizione dello script

Lo script relativo al job PySpark presenta inizialmente la lettura del set di dati iniziale e il conteggio dei record che contengono un id non nullo.

```
#### READ INPUT FILES TO CREATE AN INPUT DATASET
tedx_dataset = spark.read \
    .option("header", "true") \
    .option("quote", "\"") \
    .option("escape", "\\") \
    .csv(tedx_dataset_path)

tedx_dataset.printSchema()

#### FILTER ITEMS WITH NULL POSTING KEY
count_items = tedx_dataset.count()
count_items_null = tedx_dataset.filter("id is not null").count()

print(f"Number of items from RAW DATA {count_items}")
print(f"Number of items from RAW DATA with NOT NULL KEY {count_items_null}")
```


Aggiunta Dettagli

Viene letto un altro dataset contenente i dettagli di ogni video come: descrizione e durata.

```
## READ THE DETAILS
details_dataset_path = "s3://communitedx-2024-data/details.csv"
details_dataset = spark.read \
    .option("header", "true") \
    .option("quote", "\"") \
    .option("escape", "\\") \
    .csv(details_dataset_path)

details_dataset = details_dataset.select(col("id").alias("id_ref"),
                                         col("description"),
                                         col("duration"),
                                         col("publishedAt"))
```

Il precedente dataset viene unito al precedente tramite 'Join '

```
# AND JOIN WITH THE MAIN TABLE
tedx_dataset_main = tedx_dataset.join(details_dataset, tedx_dataset.id == details_dataset.id_ref, "left") \
    .drop("id_ref")
```

Aggiunta Immagini

Si è deciso, come per i dettagli, di aggiungere i link delle immagini relativi ad un video Ted presi da un ulteriore dataset. Viene, in seguito aggiornato il dataset iniziale con quello delle immagini

```
## READ THE IMAGES
images_dataset_path = "s3://communitedx-2024-data/images.csv"
images_dataset = spark.read \
    .option("header","true") \
    .option("quote", "\"") \
    .option("escape", "\\") \
    .csv(images_dataset_path)

images_dataset = images_dataset.select(col("id").alias("id_ref"),
                                       col("url").alias("image_url"))

# AND JOIN WITH THE MAIN TABLE
tedx_dataset_main = tedx_dataset_main.join(images_dataset, tedx_dataset_main.id == images_dataset.id_ref, "left") \
    .drop("id_ref") \

tedx_dataset_main.printSchema()
```

Aggiunta Immagini

Viene, infine, aggiornato il dataset finale con l'aggiunta di un vettore di tag identificativi relativi ad un video Tedx

```
## READ TAGS DATASET
tags_dataset_path = "s3://communitedx-2024-data/tags.csv"
tags_dataset = spark.read \
    .option("header", "true") \
    .csv(tags_dataset_path)

# CREATE THE AGGREGATE MODEL, ADD TAGS TO TEDX_DATASET
tags_dataset_agg = tags_dataset.groupBy(col("id").alias("id_ref")).agg(collect_list("tag").alias("tags"))
tags_dataset_agg.printSchema()

# AND JOIN WITH THE MAIN TABLE
tedx_dataset_agg = tedx_dataset_main.join(tags_dataset_agg, tedx_dataset_main.id == tags_dataset_agg.id_ref, "left") \
    .drop("id_ref") \

tedx_dataset_agg.printSchema()
```

Gestione dei dati e filtraggio



Il dataset finale, a questo punto, deve essere aggiornato in modo da avere solamente i dati che sono interessanti al fine dell'obiettivo del progetto e quindi devono essere isolati e selezionati solo i dati relativi a **scienza** e **tecnologia**

Il seguente blocco di codice si occupa proprio di questo:

```
# FILTER TAG FROM THE DATASET
tedx_dataset_agg = tedx_dataset_agg.filter(array_contains(col("tags"),"technology") | array_contains(col("tags"),"science"))

tedx_dataset_agg.printSchema()
```

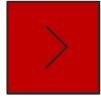
Vengono filtrati solamente i dati che interessano, controllando che tra i tag dei relativi video ci siano : ' science ' oppure ' technology '

Documento MongoDB

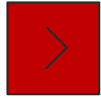
Dal dataset iniziale vengono esclusi i dati che non interessano al progetto.
Un documento si presenta in questo modo.

```
_id: "526880"  
slug: "george_zaidan_how_do_gas_masks_actually_work"  
speakers: "George Zaidan"  
title: "How do gas masks actually work?"  
url: "https://www.ted.com/talks/george_zaidan_how_do_gas_masks_actually_work"  
description: "You might think of gas masks as clunky military-looking devices. But i..."  
duration: "254"  
publishedAt: "2024-04-30T15:14:51Z"  
image_url: "https://talkstar-assets.s3.amazonaws.com/production/talks/talk_128547/..."  
▸ tags: Array (8)  
▸ id_next: Array (3)  
▸ title_next: Array (3)
```

Criticità



Nonostante i molteplici vantaggi dell'ambiente cloud, ci si aspettava tempistiche di esecuzione leggermente più veloci



Iniziale difficoltà ad apprendere i concetti legati alla programmazione PySpark



Difficoltà sulla parte di refresh dei dati in MongoDB e aumento dei costi di utilizzo per la parte di AWS (pay-per-use)



Sviluppi futuri

In un documento sono state inserite appositamente molte informazioni e dati, anche più di quelle che effettivamente servono. Questo perché si è pensato alle implementazioni future del progetto e alle eventuali evoluzioni che esso può avere.

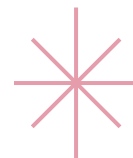
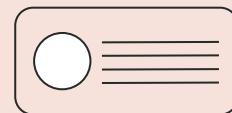
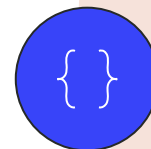




PARTE 3



Sviluppo delle Lambda
Functions e utilizzo API
Gateway in AWS



Introduzione alla parte 3

La terza parte del progetto consiste nello sviluppo di Lambda Functions previste nell'architettura del sistema .

Si è deciso di implementare due delle Lambda Functions :

- Video suggeriti (watch next videos)
- Ricerca filtrata per tags

La prima Lambda Function fornisce l'id dei video correlati ad un determinato Talk selezionato tramite url.

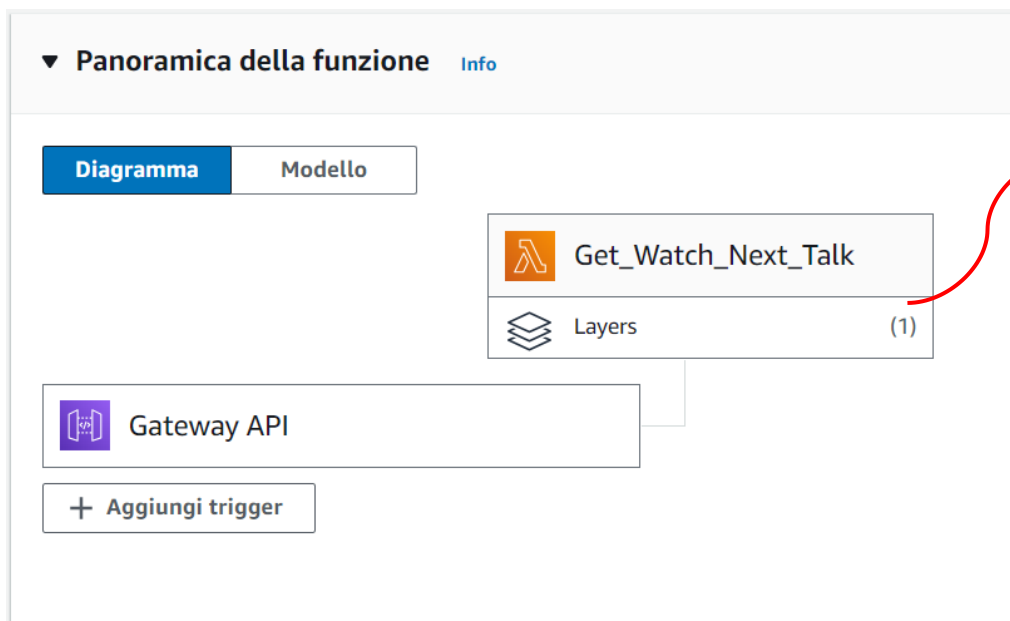
La seconda, seleziona i Talk che rispettano i tags indicati dall'utente e ne fornisce i dettagli.



Video suggeriti: Get_Watch_Next_Talk

La prima Lambda Function (Get_Watch_Next_Talk) permette, una volta cercato un video tramite url, di ottenere i video correlati allo stesso.

La funzione fornisce id e titolo dei watch next videos.



Si è utilizzato un layer (MongoDB_layer) impostando come runtime compatibili NodeJS20.x e NodeJS18.x

Descrizione del codice

- **Connessione al Database (file db.js)**

Il seguente blocco di codice implementa la connessione al database MongoDB

```
const mongoose = require('mongoose');
mongoose.Promise = global.Promise;
let isConnected;

require('dotenv').config({ path: './variables.env' });

module.exports = connect_to_db = () => {
  if (isConnected) {
    console.log('=> using existing database connection');
    return Promise.resolve();
  }

  console.log('=> using new database connection');
  return mongoose.connect(process.env.DB, { dbName: 'tedx_2024', useNewUrlParser: true, useUnifiedTopology: true }).then(db => {
    isConnected = db.connections[0].readyState;
  });
};
```

Nel file ./variables.env è contenuta la stringa di connessione relativa al database

- **Modello (file Talk.js)**

Il documento MongoDB viene mappato su attributi specifici e vengono selezionati solo quelli utili ai fini dello sviluppo della relativa funzione.

```
const mongoose = require('mongoose');

const talk_schema = new mongoose.Schema({
  _id: String,
  slug: String,
  speakers: String,
  title: String,
  url: String,
  description: String,
  duration: String,
  id_next: Array,
  title_next: Array
}, { collection: 'tedx_data' });


module.exports = mongoose.model('talk', talk_schema);
```



- **Elaborazione dati (file Handler.js) – controllo sull' url**

Inizialmente viene svolto un controllo sull'url in modo che nella richiesta venga specificato, altrimenti viene sollevato un errore gestito dall'Handler

```
// set default
if(!body.url) {
  callback(null, {
    statusCode: 500,
    headers: { 'Content-Type': 'text/plain' },
    body: 'Could not fetch the talks. Url is null.'
  })
}
```





- **Elaborazione dati (file Handler.js)**

Infine nel corpo dell'Handler vengono elaborati i dati presenti su MongoDB per ottenere da un determinato url (relativo ad un video Tedx) la serie di identificativi (id) e il titolo dei video correlati

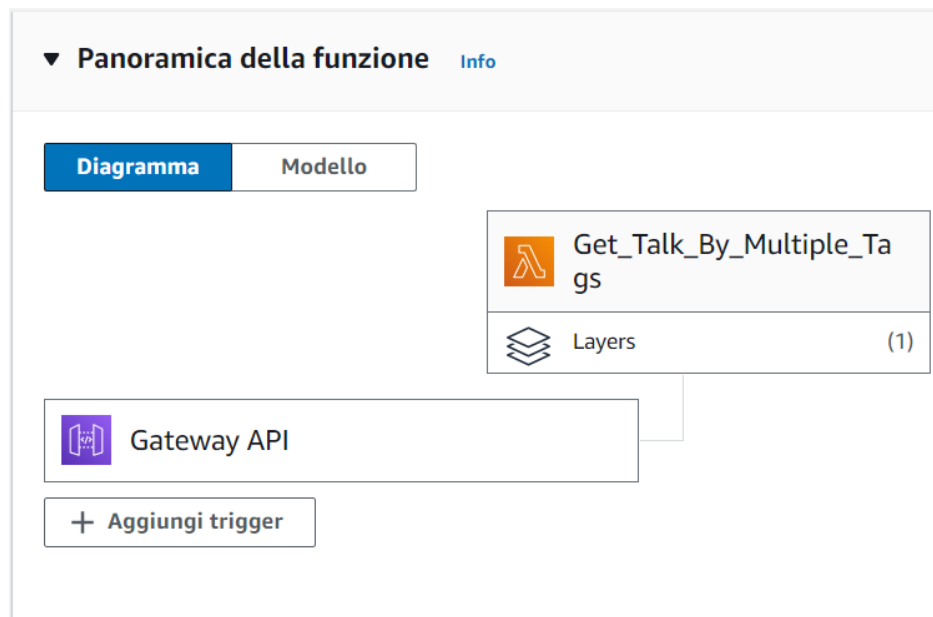
```
connect_to_db().then(() => {
  console.log('=> get_all talks');
  talk.find({url:body.url},{ _id : 0, url : 1, id_next : 1, title_next : 1 })
    .skip((body.doc_per_page * body.page) - body.doc_per_page)
    .limit(body.doc_per_page)
    .then(talks => {
      callback(null, {
        statusCode: 200,
        body: JSON.stringify(talks)
      })
    })
    .catch(err =>
      callback(null, {
        statusCode: err.statusCode || 500,
        headers: { 'Content-Type': 'text/plain' },
        body: 'Could not fetch the talks.'
      })
    );
});
```

Con questa tecnica vengono specificati gli attributi che vogliono essere considerati nella risposta della funzione:

- (0) : non considerato
- (1) : considerato

Filtro per Tags: Get_Talk_By_Multiple_Tags

La seconda Funzione Lambda che si è scelto di implementare (Get_Talk_By_Multiple_Tags) permette di selezionare Talk Tedx filtrandoli per tags. Questo significa che specificando una serie di tags la funzione seleziona solo i video che rispettano tutti i tags specificati.



Descrizione del codice

- **Connessione al Database (file db.js)**

La connessione al database è la stessa descritta per la prima Lambda Function

- **Modello (file Talk.js)**

Il documento MongoDB viene mappato su attributi specifici, vengono selezionati, quindi, solo gli attributi utili ai fini del progetto

```
const mongoose = require('mongoose');

const talk_schema = new mongoose.Schema({
  _id: String,
  slug: String,
  speakers: String,
  title: String,
  url: String,
  description: String,
  duration: String,
  tags: Array
}, { collection: 'tedx_data' });

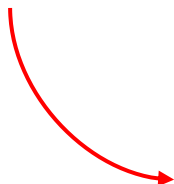
module.exports = mongoose.model('talk', talk_schema);
```




- **Elaborazione dati (file Handler.js) – controllo sui tags**

Prima di tutto viene svolto un controllo sui tags inseriti nella ricerca per verificare che tutti i parametri necessari alla ricerca siano specificati

```
// set default
if((!body.tag) || (!body.tag1)) {
  callback(null, {
    statusCode: 500,
    headers: { 'Content-Type': 'text/plain' },
    body: 'Could not fetch the talks. Tag or tag1 is null.'
  })
}
```



In questo caso è importante fare un controllo su tag e tag1. Se almeno uno dei due non è specificato, viene sollevato e gestito un errore





- **Elaborazione dati (file Handler.js)**

Infine vengono elaborati i dati presenti su MongoDB per ottenere i dettagli relativi ai video che corrispondono alla ricerca, in questo caso, basata su due tags

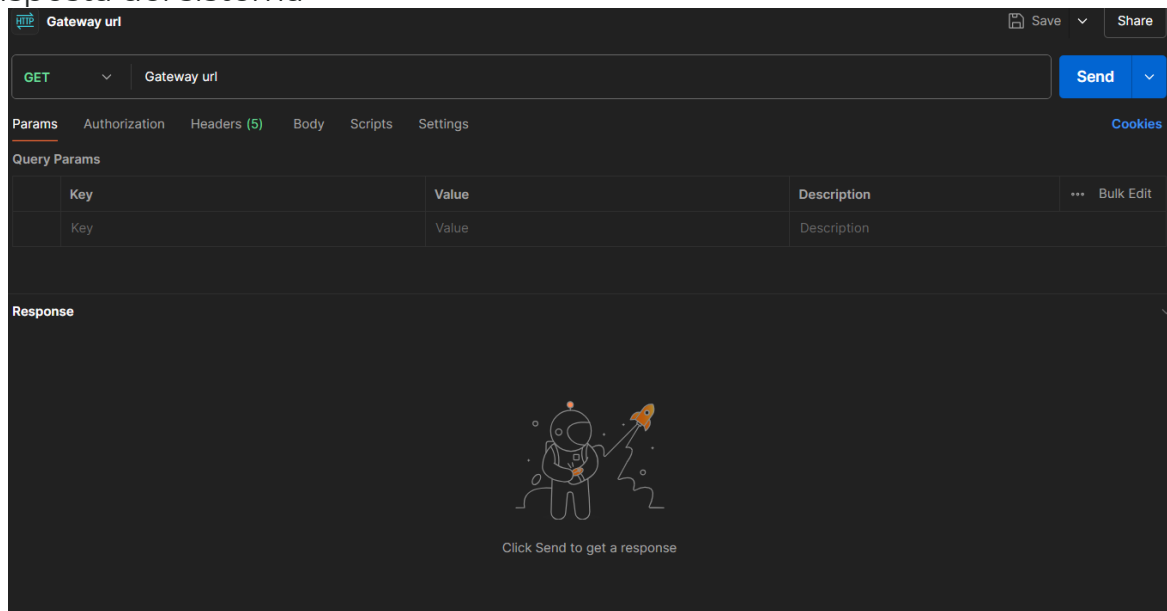
```
connect_to_db().then(() => {
  console.log('=> get_all talks');
  talk.find({$and: [{tags: body.tag},{tags: body.tag1}]},{ _id : 0, id_next : 0, title_next : 0 })
    .skip((body.doc_per_page * body.page) - body.doc_per_page)
    .limit(body.doc_per_page)
    .then(talks => {
      callback(null, {
        statusCode: 200,
        body: JSON.stringify(talks)
      })
    })
  )
  .catch(err =>
    callback(null, {
      statusCode: err.statusCode || 500,
      headers: { 'Content-Type': 'text/plain' },
      body: 'Could not fetch the talks.'
    })
  );
});
```

Si è scelto di non visualizzare i dati relativi ai video suggeriti e nemmeno l'id del video stesso

Test con Postman

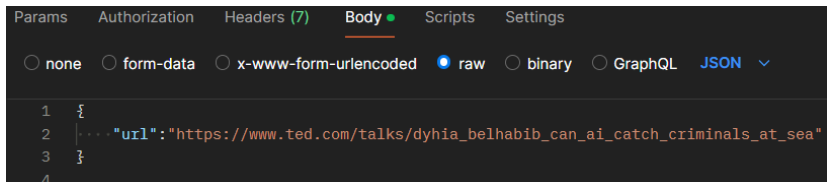
Per concludere questa parte di progetto sono stati eseguiti una serie di test per verificare il corretto funzionamento delle Lambda Function precedentemente descritte.

Viene creata una nuova richiesta, specificando l'indirizzo Gateway (del servizio di Amazon Api Gateway) della relativa funzione lambda, in formato JSON e si verifica la corretta risposta del sistema



- 1) video suggeriti

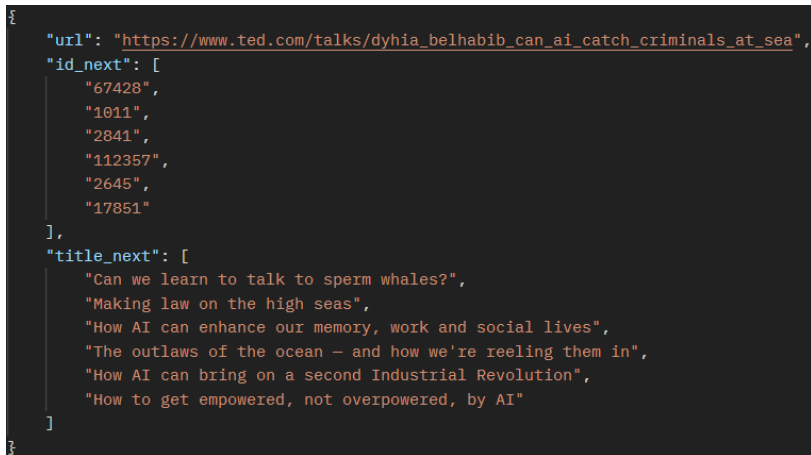
Si specifica url del video



The screenshot shows a REST client interface with tabs for Params, Authorization, Headers (7), Body, Scripts, and Settings. The 'Body' tab is selected, and the 'raw' radio button is chosen. The JSON body is as follows:

```
1 {  
2   "url": "https://www.ted.com/talks/dyhia_belhabib_can_ai_catch_criminals_at_sea"  
3 }  
4
```

Viene correttamente restituito il JSON con i parametri richiesti

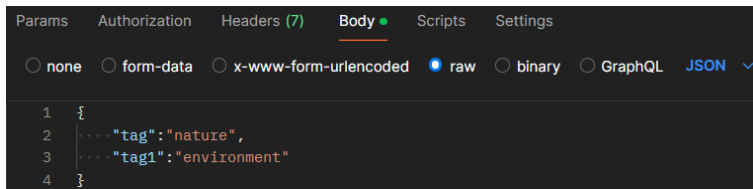


The screenshot shows the response body of the REST client. The JSON response is as follows:

```
{  
  "url": "https://www.ted.com/talks/dyhia_belhabib_can_ai_catch_criminals_at_sea",  
  "id_next": [  
    "67428",  
    "1011",  
    "2841",  
    "112357",  
    "2645",  
    "17851"  
  ],  
  "title_next": [  
    "Can we learn to talk to sperm whales?",  
    "Making law on the high seas",  
    "How AI can enhance our memory, work and social lives",  
    "The outlaws of the ocean – and how we're reeling them in",  
    "How AI can bring on a second Industrial Revolution",  
    "How to get empowered, not overpowered, by AI"  
  ]  
}
```

- **2) ricerca per tags**

Si specificano i tags nel corpo del file JSON



Viene correttamente restituito il JSON con i dettagli richiesti

```
{  
  "slug": "christy_l_haynes_can_nanoparticles_help_fight_hunger",  
  "speakers": "Christy L. Haynes",  
  "title": "Can nanoparticles help fight hunger?",  
  "url": "https://www.ted.com/talks/christy_l_haynes_can_nanoparticles_help_fight_hunger",  
  "description": "A game-changing solution to the global food crisis could come from something so tiny you can't see it with the naked  
    eye. Nanomaterials chemist Christy Haynes describes her team's work designing nanoparticles that could protect plants from  
    disease and crop loss, helping farmers reap abundant harvests and grow food that will make its way to markets and dinner tables.  
    ",  
  "duration": "683",  
  "publishedAt": "2024-02-22T15:47:30Z",  
  "image_url": "https://talkstar-photos.s3.amazonaws.com/uploads/e00c21aa-ef02-4309-bfb2-529aaa98a71f/ChristyHaynes_2022X-embed.jpg",  
  "tags": [  
    "environment",  
    "science",  
    "sustainability",  
    "food",  
    "nature",  
    "biology",  
    "chemistry",  
  ]  
}
```

Criticità



Iniziale difficoltà nella comprensione di NodeJS



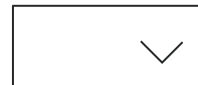
Eseguiti molti test per valutare la correttezza delle funzioni



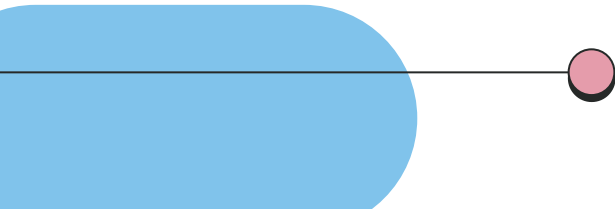
Ampia scelta nella possibilità di sviluppo delle funzioni (le alternative sono descritte brevemente in seguito)



Sviluppi futuri



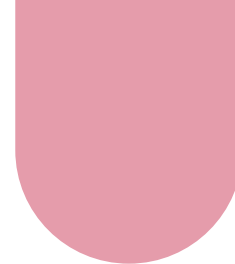
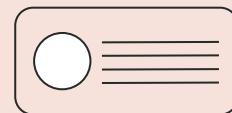
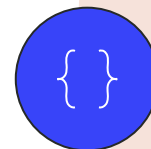
Una possibilità di migliorare le funzioni implementate, per quanto riguarda la funzione di ricerca per tags, potrebbe essere quella di utilizzare un vettore di tags e inserire tutte le parole chiave che si vogliono cercare in un array di stringhe, così da non costringere l'utente a dover scegliere quali e quante parole utilizzare nella fase di ricerca. In alternativa è possibile sviluppare funzioni che permettano la ricerca con un singolo tag o tag esclusivi. In aggiunta si potrebbe sviluppare il filtraggio non solo con i tags ma anche con altri parametri, ad esempio con l'autore del Talk.





PARTE 4

Utilizzo delle API in Flutter



Introduzione alla parte 4

La quarta parte del progetto consiste nel collegare le API (sviluppate nelle parti precedenti) all'applicazione mobile realizzata in Flutter.

Le funzionalità che si è deciso di riportare sono:

- Ricerca per un solo tag
- Ricerca filtrata per due tags
- Ricerca per URL

La prima restituisce i vari Talk contenenti il tag fornito come input

La seconda seleziona i Talk che rispettano i tags indicati dall'utente e ne fornisce i dettagli.

La terza prende in ingresso l'url di un qualsiasi video TEDx e ne restituisce il titolo e la lista dei titoli correlati

Il modello dei dati: talk.dart

Il file *talk.dart* contiene il modello dei dati che stiamo utilizzando.

All'interno della classe *Talk*, si utilizza il metodo *fromJSON* per gestire una mappa JSON contenente vari attributi (*title*, *details*, ...) ed istanziare un oggetto *Talk*.

```
lib > models > talk.dart > Talk
1  class Talk {
2    final String title;
3    final String details;
4    final String mainSpeaker;
5    final String url;
6    // title_next
7    final List<String> watchNext;
8
9    Talk.fromJSON(Map<String, dynamic> jsonMap) :
10      title = jsonMap['title'],
11      details = jsonMap['description'],
12      mainSpeaker = (jsonMap['speakers'] ?? ""),
13      url = (jsonMap['url'] ?? ""),
14      // title_next
15      watchNext = List<String>.from(jsonMap['title_next'] ?? "There are no related videos");
16  }
```

Il controller per richiamare le API: talk_repository.dart

Il file *talk_repository.dart* ha il compito di richiamare le API, applicare il modello dati e presentare i dati all'applicazione. In altre parole, questo attore contiene la logica di chiamata all'API e quella per mappare il risultato sul modello.

Di seguito, la funzione `getTalksByTag`:

```
// getTalksByTag
Future<List<Talk>> getTalksByTag(String tag, int page) async {
  var url = Uri.parse('https://b3zz9i7ati.execute-api.us-east-1.amazonaws.com/default/Get_Talks_By_Tag');

  final http.Response response = await http.post(url,
    headers: <String, String>{
      'Content-Type': 'application/json; charset=UTF-8',
    },
    body: jsonEncode(<String, Object>{
      'tag': tag,
      'page': page,
      'doc_per_page': 6
    })),
  );
  if (response.statusCode == 200) {
    Iterable list = json.decode(response.body);
    var talks = list.map((model) => Talk.fromJSON(model)).toList();
    return talks;
  } else {
    throw Exception('Failed to load talks');
  }
}
```

La funzione getTalksBy2Tags

Sempre all'interno del file *talk_repository.dart*, vi è la funzione *getTalksBy2Tags* che permette di ottenere le informazioni principali di un video TEDx, dati due tags di appartenenza.

Nel corpo della funzione asincrona, il primo step che viene svolto è la chiamata API attraverso un HTTP Post. Una volta eseguita la chiamata, vengono passati quattro parametri come payload: i due tag, la pagina e il numero di talk per pagina. Se la risposta è positiva (*statusCode* pari a 200), il corpo della risposta viene formattato in JSON e per ogni elemento viene applicato il metodo *Talk.fromJSON*.

```
// getTalksBy2Tags
Future<List<Talk>> getTalksBy2Tags(String tag1, String tag2, int page) async {
  var url = Uri.parse('https://qga57u0cqf.execute-api.us-east-1.amazonaws.com/default/Get_Talks_by_2_Tags_And_Logic');

  final http.Response response = await http.post(url,
    headers: <String, String>{
      'Content-Type': 'application/json; charset=UTF-8',
    },
    body: jsonEncode(<String, Object>{
      'tag1': tag1,
      'tag2': tag2,
      'page': page,
      'doc_per_page': 6
    })),
  );
  if (response.statusCode == 200) {
    Iterable list = json.decode(response.body);
    var talks = list.map((model) => Talk.fromJSON(model)).toList();
    return talks;
  } else {
    throw Exception('Failed to load talks');
  }
}
```

La funzione getTalksByUrl

Sempre all'interno del file *talk_repository.dart*, vi è un'ultima funzione *getTalksByUrl* che permette di ottenere il titolo di un video TEDx e quelli dei watch next dato l'url di un qualsiasi Talk.

```
// getTalksByUrl
Future<List<Talk>> getTalksByUrl(String urlTalk, int page) async {
  var url = Uri.parse('https://9jpk8yo2j.execute-api.us-east-1.amazonaws.com/default/Get_Talk_by_url');

  final http.Response response = await http.post(url,
    headers: <String, String>{
      'Content-Type': 'application/json; charset=UTF-8',
    },
    body: jsonEncode(<String, Object>{
      'url': urlTalk,
      'page': page,
      'doc_per_page': 6
    })),
  );
  if (response.statusCode == 200) {
    Iterable list = json.decode(response.body);
    var talks = list.map((model) => Talk.fromJSON(model)).toList();
    return talks;
  } else {
    throw Exception('Failed to load talks');
  }
}
```

Nella seguente immagine, si può verificare l'aggiunta del Trigger alla LF *Get_Talk_by_url* in AWS.

[Lambda](#) > [Funzioni](#) > [Get_Talk_by_url](#)

Get_Talk_by_url

Limitatore

Copia ARN

Operazioni ▼

✓ Il trigger Get_Talk_by_url-API è stato aggiunto alla funzione Get_Talk_by_url. La funzione sta ora ricevendo eventi dal trigger.



▼ **Panoramica della funzione** [Info](#)

Esporta nello Strumento per la creazione di applicazioni

Scarica ▼

Diagramma

Modello



Get_Talk_by_url



Layers

(1)



Gateway API

+ Aggiungi trigger

+ Aggiungi destinazione

Descrizione

-

Ultima modifica

2 minuti fa

ARN della funzione

arn:aws:lambda:us-east-1:533266973932:function:Get_Talk_by_url

URL della funzione [Info](#)

-

Il main e la parte di visualizzazione dell'applicazione: main.dart

In `_MyHomePageState` si dichiara:

- i controller che permettono di inserire il testo per le varie ricerche (tag, filtri e url)
- `_talks` (lista) che contiene i talks da ritornare all'utente dopo ogni ricerca
- `sceltaopzione`, molto simile ad una variabile flag che tiene traccia delle scelte che l'utente sta facendo

```
class _MyHomePageState extends State<MyHomePage> {  
  // controller per getTalksByTag  
  final TextEditingController _controller = TextEditingController();  
  // controller1 e controller2 per getTalksBy2Tags  
  final TextEditingController _controller1 = TextEditingController();  
  final TextEditingController _controller2 = TextEditingController();  
  // controller3 per getTalkByUrl  
  final TextEditingController _controller3 = TextEditingController();  
  // lista contenenti i talks da ritornare all'utente  
  late Future<List<Talk>> _talks;  
  // inizializzata prima pagina  
  int page = 1;  
  bool init = true;  
  // per capire che scelta sto facendo  
  int sceltaopzione = 0;  
}
```

La funzione getTalksByUrl

Sempre all'interno del file *talk_repository.dart*, vi è un'ultima funzione *getTalksByUrl* che permette di ottenere il titolo di un video TEDx e quelli dei watch next dato l'url di un qualsiasi Talk.

```
// getTalksByUrl
Future<List<Talk>> getTalksByUrl(String urlTalk, int page) async {
  var url = Uri.parse('https://9jpk8yo2j.execute-api.us-east-1.amazonaws.com/default/Get_Talk_by_url');

  final http.Response response = await http.post(url,
    headers: <String, String>{
      'Content-Type': 'application/json; charset=UTF-8',
    },
    body: jsonEncode(<String, Object>{
      'url': urlTalk,
      'page': page,
      'doc_per_page': 6
    })),
  );
  if (response.statusCode == 200) {
    Iterable list = json.decode(response.body);
    var talks = list.map((model) => Talk.fromJSON(model)).toList();
    return talks;
  } else {
    throw Exception('Failed to load talks');
  }
}
```


La funzione trimRight()

Durante le prime fasi di test dell'applicazione, è emerso un problema legato allo spazio inserito automaticamente dalla tastiera del telefono quando viene suggerito il tag da ricercare. Una volta che viene eseguita la chiamata all'API, vengono passati sia il tag sia lo spazio aggiunto involontariamente, ritornando così un messaggio di errore. Per ovviare a questo inconveniente, si è deciso di impiegare la funzione *trimRight()* che elimina tutti gli spazi presenti alla fine della parola.

```
// metodo getTalksByTag
void _getTalksByTag() async {
  setState(() {
    init = false;
    // set scelta
    sceltaopzione = 1;
    _talks = getTalksByTag(_controller.text.trimRight(), page);
  });
}

// metodo getTalksBy2Tags
void _getTalksBy2Tags() async {
  setState(() {
    init = false;
    // set scelta
    sceltaopzione = 2;
    _talks = getTalksBy2Tags(_controller1.text.trimRight(), _controller2.text.trimRight(), page);
  });
}
```

Parte grafica della funzionalità getTalksBy2Tags

Nella parte grafica, sono stati inseriti due *TextField*, uno per ogni tag da ricercare e un *ElevatedButton* per la ricerca dei Talk desiderati.

```
// TextField per il primo tag (getTalksBy2Tags)
TextField(
  controller: _controller1,
  decoration:
    | | const InputDecoration(hintText: 'Enter your favorite first talk'),
), // TextField

// TextField per il secondo tag (getTalksBy2Tags)
TextField(
  controller: _controller2,
  decoration:
    | | const InputDecoration(hintText: 'Enter your favorite second talk'),
), // TextField

// ElevatedButton per due tags (and logica)
ElevatedButton(
  child: const Text('Search by 2 tags'),
  onPressed: () {
    page = 1;
    _getTalksBy2Tags();
  },
), // ElevatedButton
```

Parte grafica della funzionalità getTalkByUrl

Nella parte grafica, sono stati inseriti un *TextField*, per l'url del video da ricercare e un *ElevatedButton* per dare inizio alla ricerca.

```
// TextField per url (getTalkByUrl)
TextField(
  controller: _controller3,
  decoration:
    | | const InputDecoration(hintText: 'Enter your favorite url talk'),
), // TextField

// ElevatedButton per url (getTalkByUrl)
ElevatedButton(
  child: const Text('Search by url'),
  onPressed: () {
    | page = 1;
    | _getTalkByUrl();
  },
), // ElevatedButton
```

Gestione dell'AppBar

Ogni volta che viene caricata l'AppBar, la variabile *sceltaopzione* decide quale scritta è più opportuno visualizzare. Nel primo caso (1), viene cercato un solo tag e nell'AppBar viene visualizzato il nome del tag cercato, seguito da un cancelletto; nel secondo caso (2), vengono visualizzati entrambi i tag cercati; mentre nel terzo e ultimo caso (3), viene visualizzato un messaggio statico uguale per tutte le ricerche mediante url.

La strategia adottata è stata quella di scrivere l'IF con la tecnica inline.

```
appBar: AppBar(  
  title: Text(  
    // if inline per il testo dell'AppBar  
    sceltaopzione == 1 ? "#${_controller.text}" :  
    ( (sceltaopzione == 2) ? ("#${_controller1.text}, #${_controller2.text}") :  
    ("Info about the url searched") ) ,  
  ), // Text  
) , // AppBar
```

getTalksByTag

My TEDx App

#art

The underground cities of the Byzantine Empire
Veronica Kalas

The weird and wonderful art of Niceaunties
Niceaunties

A futuristic vision for Latin America, rooted in ancient design
Catalina Lotero

Why you should disappoint your parents
Desiree Akhavan

This person isn't actually screaming
Noah Charney

How to find creativity and purpose in the face of adversity
Suleika Jaouad



getTalksBy2Tags

My TEDx App

#science , #future

How AI will step off the screen and into the real world
Daniela Rus

How AI is unlocking the secrets of nature and the universe
Demis Hassabis

The transformative potential of AGI and when it might arrive
Shane Legg and Chris Anderson

TED Explores: A New Climate Vision
TED Countdown

Artificial skin? We made it here's why
Anna Maria Coclite

How to keep AI under control
Max Tegmark



getTalkByUrl

My TEDx App

Info about the url searched

How do gas masks actually work?
[Whatever happened to the hole in the ozone layer?, Why plague doctors wore beaked masks, What's in the air you breathe?]



Criticità



Difficoltà nella programmazione con il linguaggio Dart



Difficoltà nel familiarizzare con il pattern architetturale Model View Controller



A causa delle precedenti criticità, c'è molto margine di miglioramento sulla parte grafica



Sviluppi futuri

Una modifica importante e allo stesso tempo necessaria nella parte dei risultati delle varie ricerche è quella di aggiungere un'ulteriore freccia per ritornare alla pagina precedente durante lo scorrimento dei risultati. Infine, con più tempo a disposizione andrebbe rivoluzionata l'interfaccia grafica dell'applicazione mobile.

