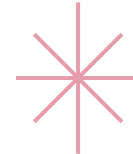
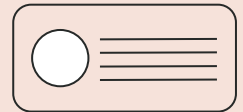
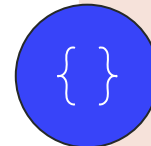




CommuniTEDx Applicazione Mobile

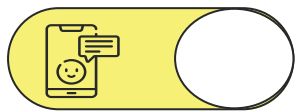
Progetto per il corso di
tecnologie Cloud e Mobile

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



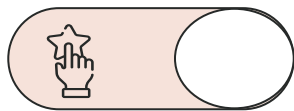
Descrizione e obiettivo di CommuniTEDx

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



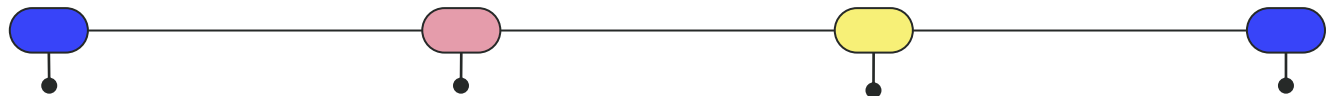
Intrattenimento

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

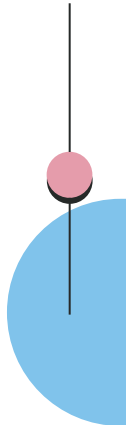
L'applicazione suggerisce determinati video in base alle proprie preferenze e consente di creare collegamenti con altri utenti

Salvataggio dei video TEDx

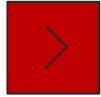
L'utente può creare, modificare ed eliminare playlist in cui è possibile salvare video TEDx a cui un utente è particolarmente interessato

Chat e condivisione

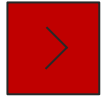
Possibilità di condividere video con i propri collegamenti con l'ulteriore possibilità di iniziare una conversazione



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

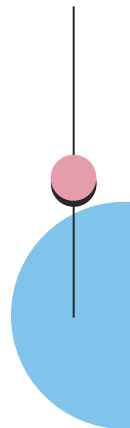





A chi è rivolto CommuniTEDx?

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.

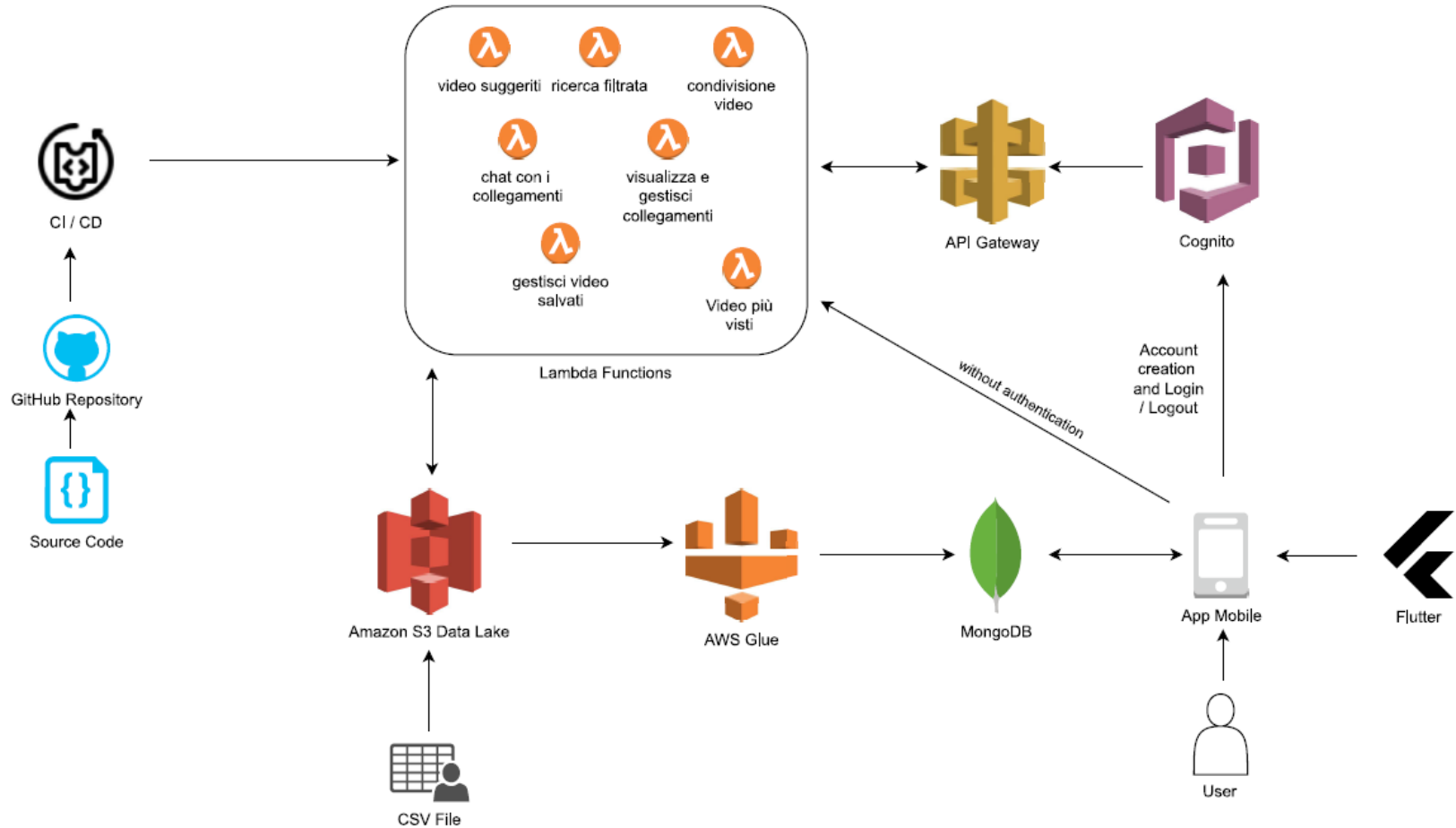
Gli utenti dell'app possono accedere tramite il pool di utenti o accedere in modalità federata, in particolare Amazon Cognito implementa lo standard *OAuth 2* che permette l'autenticazione tramite un gestore dell'identità digitale (IdP) di terze parti come Google o Facebook.

Queste funzionalità di Amazon Cognito sono utili perché permettono di:

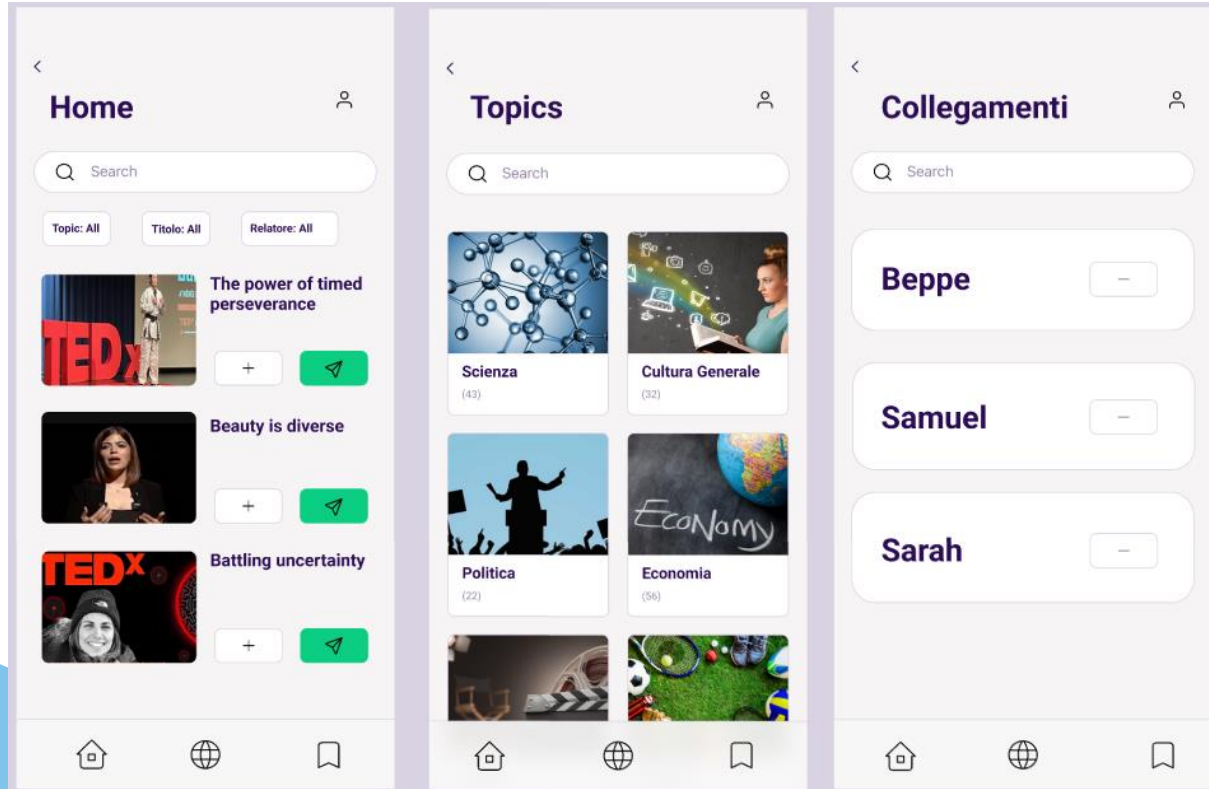
- Progettare pagine web di iscrizione e accesso per l'app tramite un pool di utenza.
- Generare credenziali AWS temporanee per utenti non autenticati tramite un pool di identità.

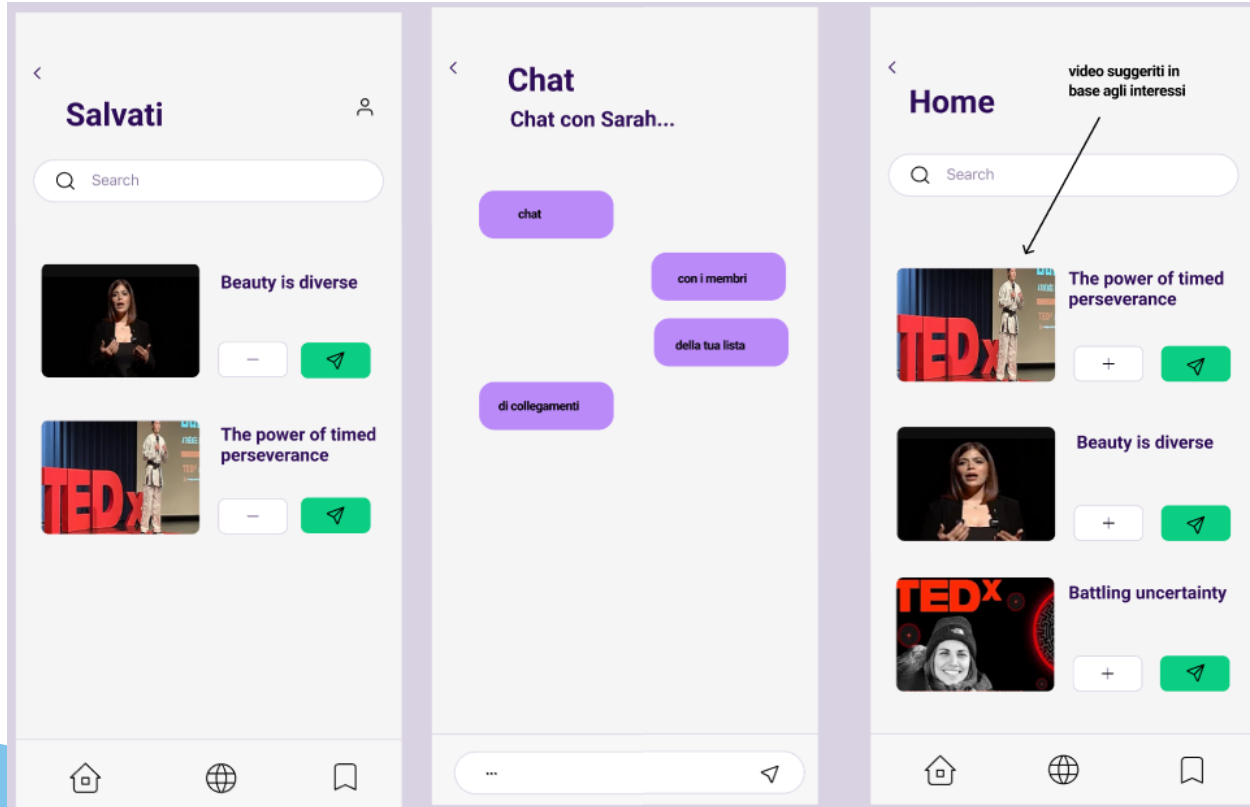
• •
• •
• •
• •
• •
• •
• •
• •

Architettura



Presentazione dell'interfaccia grafica





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

Trello Board

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

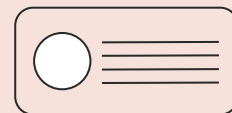
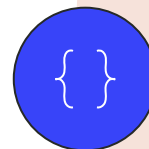
The screenshot shows a Trello board with four columns. The first column, labeled 'Fatto', lists completed tasks. The subsequent columns, labeled '01', '02', and '03', show tasks in progress. The '04' column is partially visible on the right.

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()
```

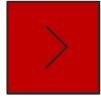
Vengono filtrati solamente i dati che interessano, controllando che tra i tag dei relativi video ci siano : ' science ' oppure ' technology '

Documento MongoDB

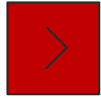
Dal dataset iniziale vengono esclusi i dati che non interessano al progetto.
Un documento si presenta in questo modo.

```
_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

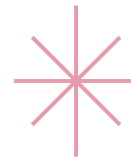
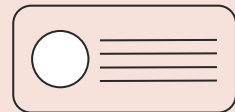
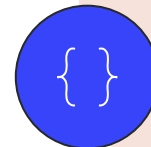
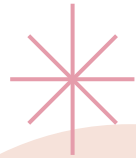
In un documento sono state inserite appositamente molte informazioni e dati, anche più di quelle che effettivamente servono. Questo perché si è pensato alle implementazioni future del progetto e alle eventuali evoluzioni che esso può avere.





PARTE 3

Sviluppo delle Lambda Functions



Introduzione alla parte 3

La parte 3 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.

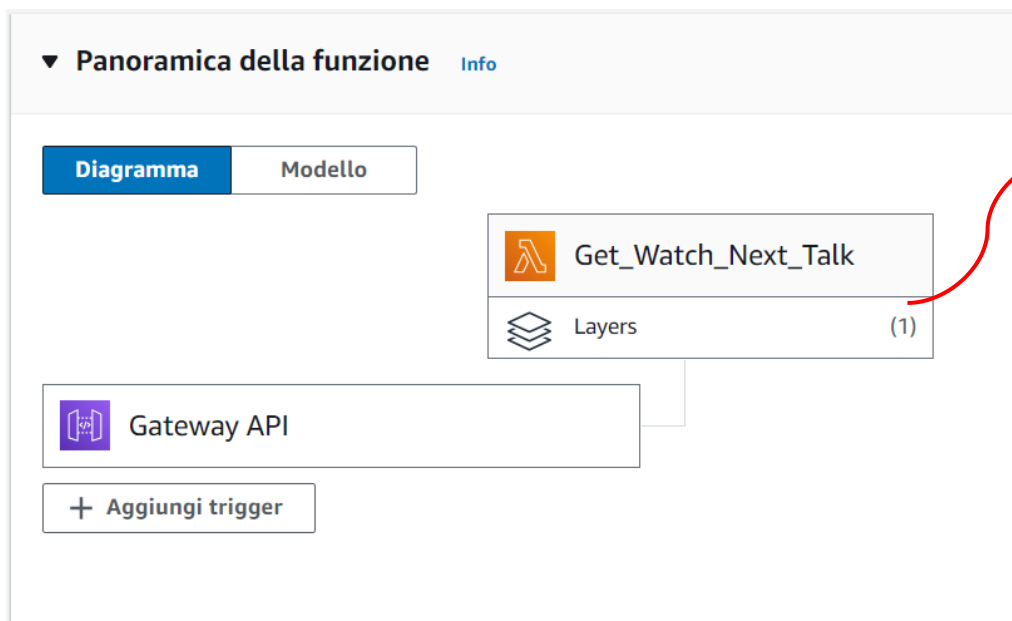
La seconda, seleziona i Talk che rispettano i tags indicati dall'utente e ne fornisce i dettagli.



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. Vengono selezionati, quindi, solo gli attributi 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**

Prima di tutto viene svolto un controllo sull' url in modo che nella richiesta venga specificato, altrimenti viene sollevato un errore

```
// 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 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

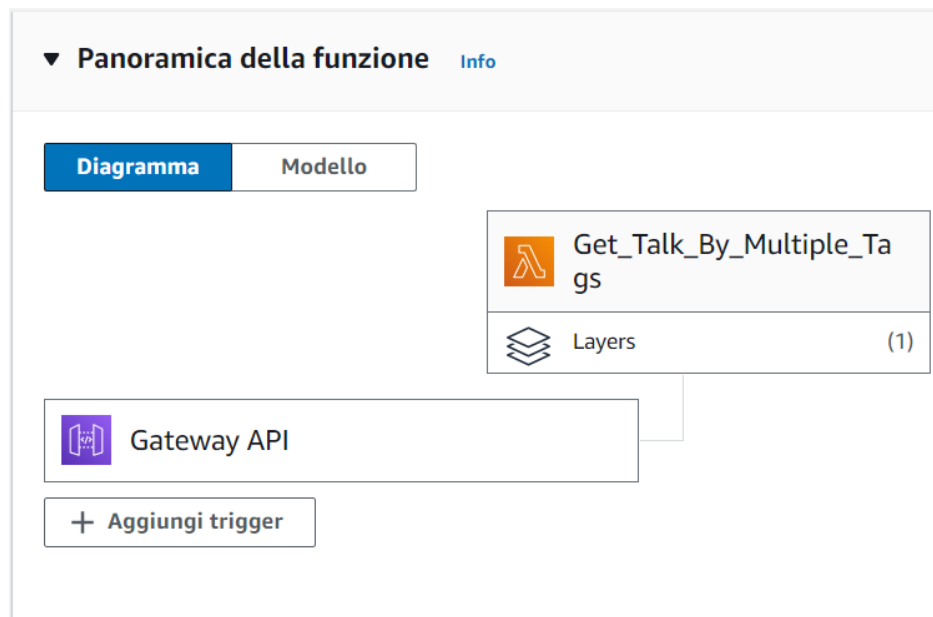
```
connect_to_db().then(() => {
  console.log('=> get_all talks');
  talk.find({url:body.url},{ _id : 0, url : 1, id_next : 1, title_next : 1 })
    .skip((body.doc_per_page * body.page) - body.doc_per_page)
    .limit(body.doc_per_page)
    .then(talks => {
      callback(null, {
        statusCode: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

In questo modo 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' });

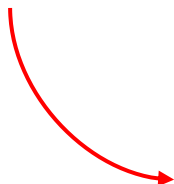
module.exports = mongoose.model('talk', talk_schema);
```




- **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.'
  })
}
```



Importante, in questo caso fare un controllo su tag e tag1. Se almeno uno dei due non è specificato, viene sollevato 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

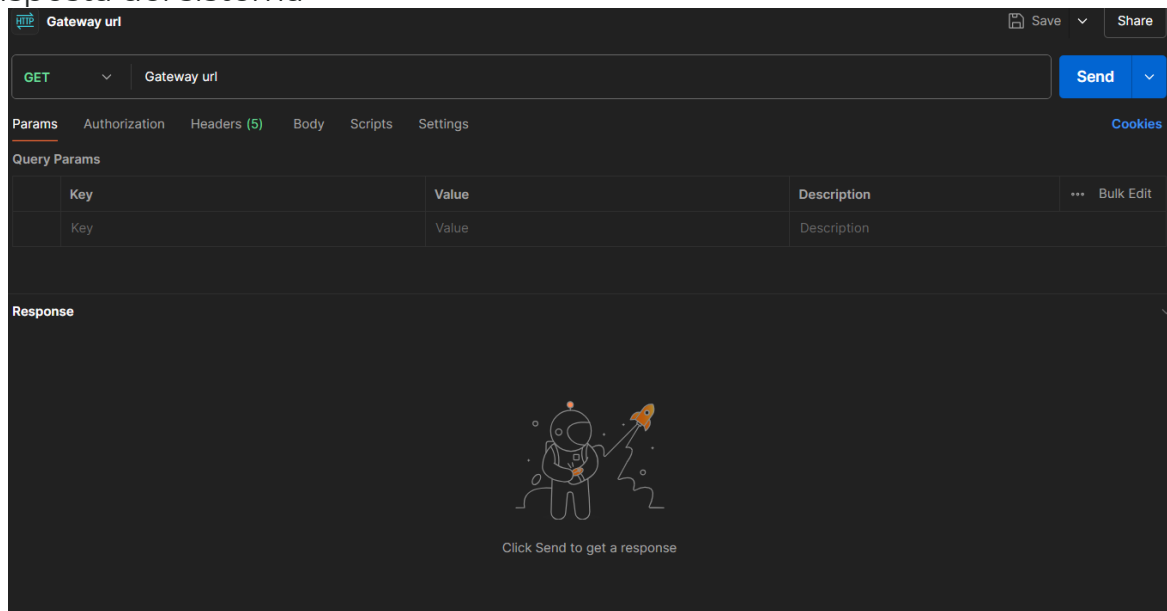
```
connect_to_db().then(() => {
  console.log('=> get_all talks');
  talk.find({$and: [{tags: body.tag},{tags: body.tag1}]},{ _id : 0, id_next : 0, title_next : 0 })
    .skip((body.doc_per_page * body.page) - body.doc_per_page)
    .limit(body.doc_per_page)
    .then(talks => {
      callback(null, {
        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.'
    })
  );
});
```

In questo caso si è scelto di non visualizzare i dati relativi ai video suggeriti e l'id del video stesso

Test con Postman

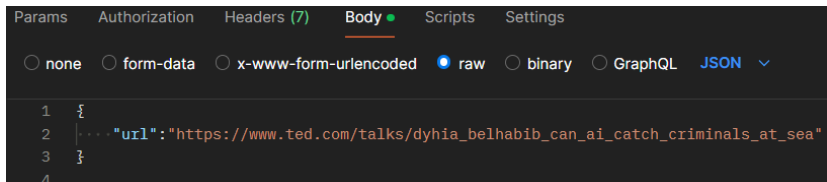
Per concludere questa parte di progetto sono stati eseguiti una serie di test per verificare il corretto funzionamento delle Lambda Function precedentemente descritte.

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

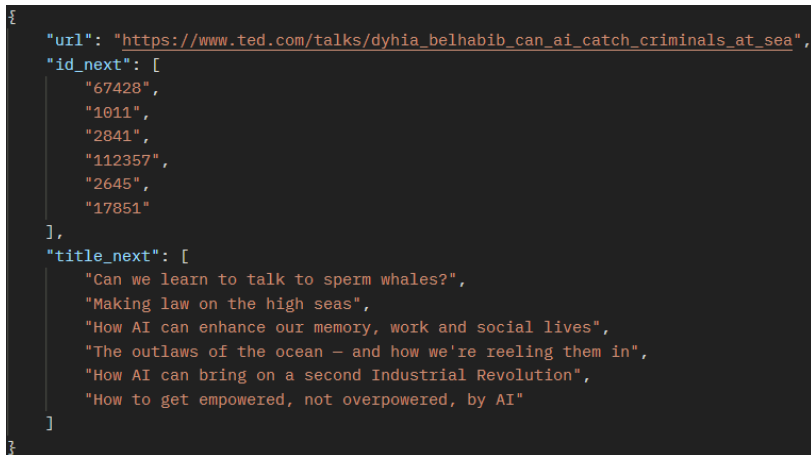


- 1) video suggeriti

Si specifica url del video

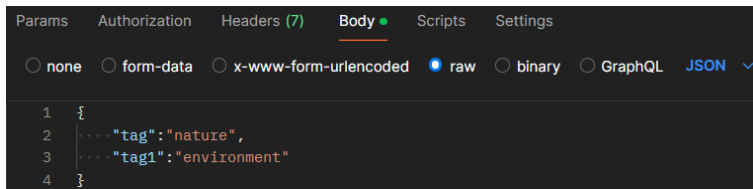


Viene correttamente restituito il JSON con i parametri richiesti



- **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.

