



# **TED GO**

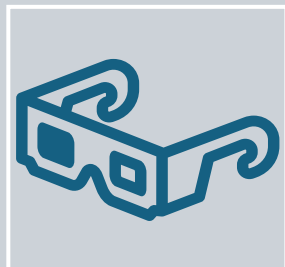
**I TUOI TEDX OVUNQUE TU SIA!**

Beccarelli Raissa mat. 1086785

Locatelli Giacomo mat. 1086262

Valceschini Marco mat. 1086356

# API SVILUPPATE



## API PER I WATCH NEXT

In questa API abbiamo deciso di mostrare per ogni video i suoi relativi **watch next**. In questo modo basta mettere **l'id** del video che si vuole visualizzare e verranno direttamente mostrati i video suggeriti con il loro titolo.



## API PER LA SUDDIVISIONE IN CANALI

In questa API abbiamo deciso di mostrare per ogni **canale tematico** (arte e design, business, scienza, educazione, intrattenimento, politica, sport e tecnologia) i suoi relativi video, con delle **informazioni utili** per ogni video: il titolo, la descrizione, lo speaker, l'url e l'orario della messa in onda.

# LAMBDA FUNCTION: GET\_WATCH\_NEXT\_BY\_ID

## Talk.js

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

const talk_schema = new mongoose.Schema({
  _id: String,
  title: String,
  url: String,
  watch_next: [Number]
}, { collection: 'tedx_data' });

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

Lo schema che abbiamo deciso di utilizzare per mostrare il Talk e i relativi watch next è formato dall'**\_id del video** (che nel nostro dataset è definito come String), dal **titolo**, dall'**url** e da un **array numerico** di watch next. Abbiamo preso questi dati dalla **collection** `tedx_data`.

Il nostro obiettivo è quello di andare a mostrare, dato l'id di un video, il titolo del talk stesso e per i suoi watch next:

- L'id del video
- Il titolo del video
- L'url del video

Questi dati abbiamo deciso di mostrarli nella API per utilizzarli nella griglia di **contenuti consigliati** che verrà implementata successivamente.

# LAMBDA FUNCTION: GET\_WATCH\_NEXT\_BY\_ID

## Handler.js

```
try {
  await connect_to_db();
  console.log('=> Fetching watch_next for ID:', body._id);
  const result = await talk.findById(body._id);

  if (!result) {
    return callback(null, {
      statusCode: 404,
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ error: 'Talk not found' })
    });
  }

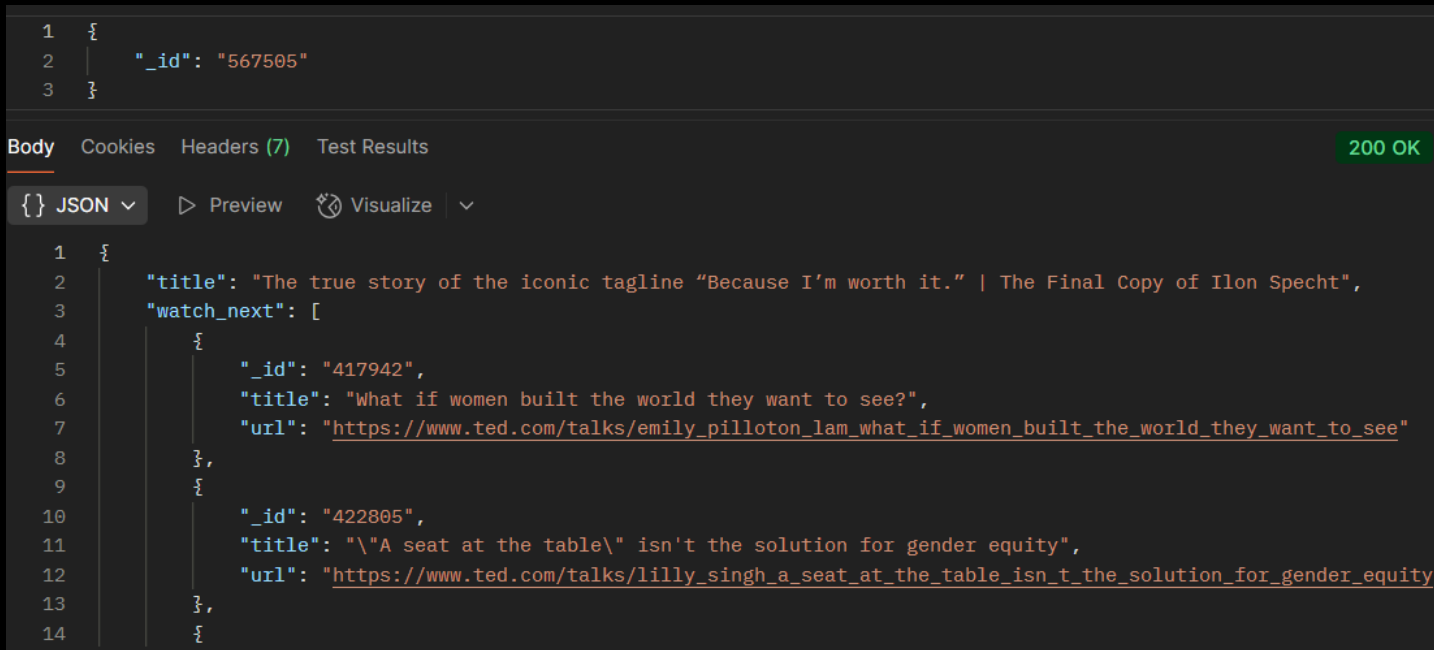
  const watchNextTalks = await talk.find({ _id: { $in: result.watch_next } });
  // Crea una lista con solo _id, title e url
  const watchNextWithTitles = watchNextTalks.map(t => ({
    _id: t._id,
    title: t.title,
    url: t.url
  }));
}
```

```
return callback(null, {
  statusCode: 200,
  headers: { 'Content-Type': 'application/json' },
  //body: JSON.stringify({watch_next: result.watch_next})
  body: JSON.stringify({
    title: result.title,
    watch_next: watchNextWithTitles
  }, null, 2)
});

} catch (err) {
  console.error('Error fetching watch next:', err);
  return callback(null, {
    statusCode: err.statusCode || 500,
    headers: { 'Content-Type': 'text/plain' },
    body: 'Could not fetch the watch next.'
  });
}
```

# LAMBDA FUNCTION: GET\_WATCH\_NEXT\_BY\_ID

## Postman



Tramite l'**handler** implementato nella slide precedente, abbiamo innanzitutto verificato che l'\_id inserito fosse all'interno del nostro dataset.

Se trova l'\_id, vengono presi tutti i suoi watch next e per ognuno di essi viene fatto un **map** che crea una lista con all'interno i dati citati prima.

Successivamente viene effettuata una **callback** per stampare i dati e visualizzare uno schema come quello in figura.

Per testare questa API incolla su **Postman** questo link:

[https://lpgx6rn6i.execute-api.us-east-1.amazonaws.com/default/Get\\_WatchNext\\_By\\_ID](https://lpgx6rn6i.execute-api.us-east-1.amazonaws.com/default/Get_WatchNext_By_ID)

# LAMBDA FUNCTION: GET\_TALK\_BY\_CHANNEL

## Talk.js

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

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

const Talk = mongoose.model('talk', talk_schema);

const channel_schema = new mongoose.Schema({
  _id: String,
  id_associati: [Number]
}, { collection: 'tedx_canali' });

const Channel = mongoose.model('channel', channel_schema);
module.exports = {Talk, Channel};
```

In questa API abbiamo deciso di utilizzare due schemi diversi, per rappresentare due collection diverse:

- «**talk\_schema**» si riferisce al dataset che abbiamo usato anche prima, e lo utilizziamo per andare a mostrare i dati di ogni talk;
- «**channel\_schema**» si riferisce al dataset tedx\_canali, creato precedentemente, dove all'interno abbiamo suddiviso i video per canale tematico.

Abbiamo quindi esportato entrambi gli schemi, richiamandoli nell'handler in questo modo:

```
const {Talk, Channel} = require('./Talk');
```

# LAMBDA FUNCTION: GET\_TALK\_BY\_CHANNEL

```
try {
  await connect_to_db();
  console.log('=> Fetching talks for channel ID:', body._id);

  const channel = await Channel.findById(body._id);

  if (!channel) {
    return callback(null, {
      statusCode: 404,
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({ error: 'channel not found' })
    });
  }

  const talks = await Talk.find({ _id: { $in: channel.id_associati.map(String) } });

  let startTime = new Date();
  const talkDetails = talks.map(t => {
    const durationInSeconds = parseInt(t.duration, 10) || 0;

    const scheduleTime = new Date(startTime);

    //con padStart formattiamo la lunghezza dell'orario
    const hours = scheduleTime.getHours().toString().padStart(2, '0');
    const minutes = scheduleTime.getMinutes().toString().padStart(2, '0');
    const day = scheduleTime.getDate().toString().padStart(2, '0');
    const month = (scheduleTime.getMonth() + 1).toString().padStart(2, '0');
    const formattedTime = `${hours}:${minutes} on ${day}/${month}`;

    startTime = new Date(startTime.getTime() + durationInSeconds * 1000);

    return {
      title: t.title,
      speakers: t.speakers,
      description: t.description,
      url: t.url,
      schedule_time: `Streaming at ${formattedTime}`
    };
  });
};
```

## Handler.js

In questa funzione abbiamo innanzitutto cercato nel dataset **tedx\_canali**, la presenza del canale tramite il suo **\_id** e fatto i controlli sulla sua esistenza.

Successivamente nel dataset **tedx\_data** abbiamo preso tutti i talk i cui **\_id** erano presenti nel dataset precedente per quel canale, e per ognuno di essi siamo andati a recuperare i dati necessari: titolo, speaker, descrizione, url e durata. Per quanto riguarda la **durata**, abbiamo deciso inizialmente di **convertirla** sotto forma di intero e poi di formattarlo in mm:ss.

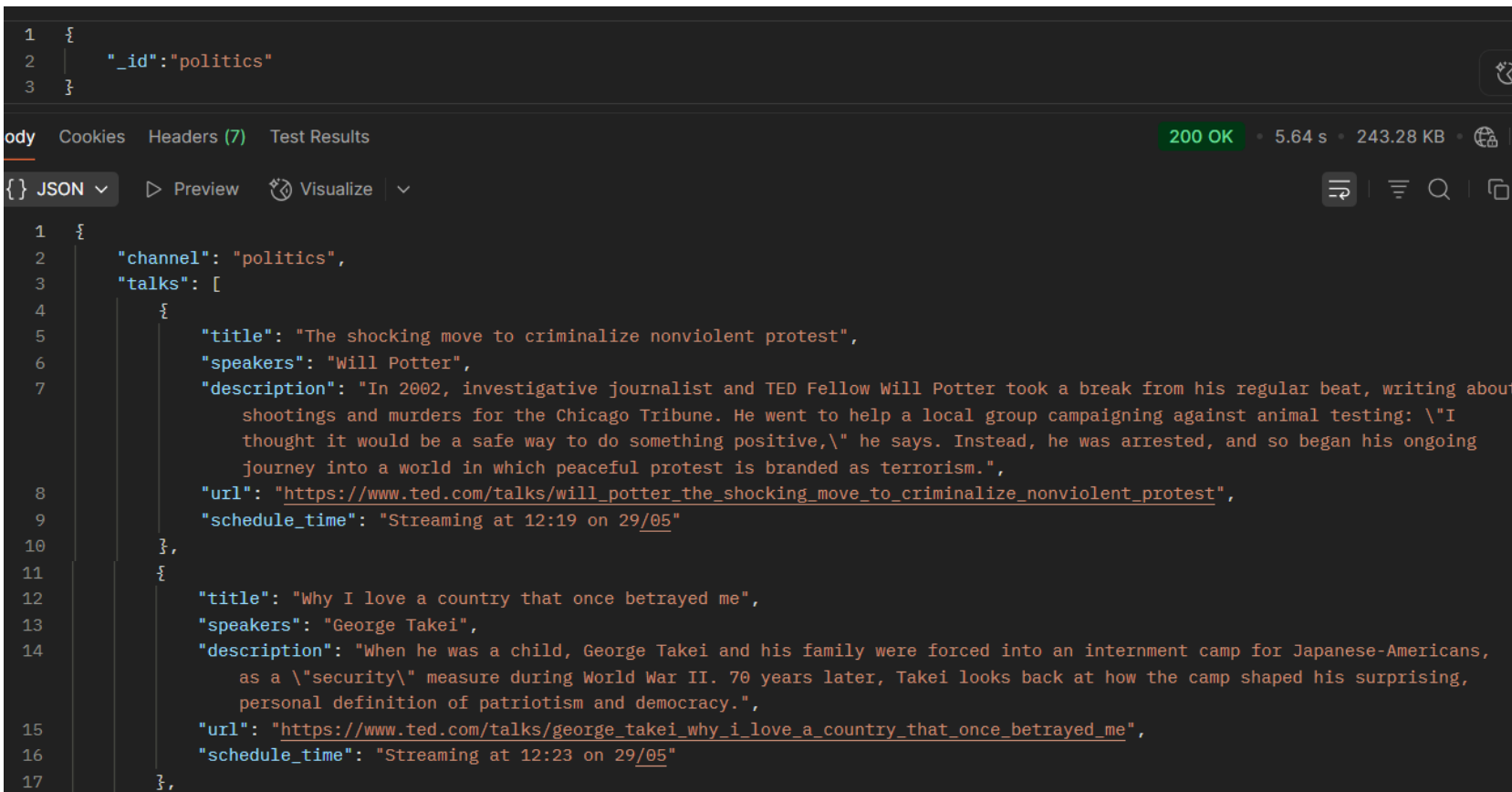
Infine abbiamo anche aggiunto un metodo per andare a calcolare la **programmazione** dei talk, di modo che a partire dalla data attuale, per ogni video venga visualizzata la sua messa in onda in forma di «Streaming at hh:mm on dd/mm».

Alla fine di questo codice, come prima, abbiamo stampato tramite la funzione di **callback**:

```
return callback(null, {
  statusCode: 200,
  headers: { 'Content-Type': 'application/json' },
  body: JSON.stringify({
    channel: body._id,
    talks: talkDetails
  }), null, 2)
});
```

# LAMBDA FUNCTION: GET\_TALK\_BY\_CHANNEL

## Postman



```
1 {
2   "_id": "politics"
3 }

body Cookies Headers (7) Test Results 200 OK • 5.64 s • 243.28 KB
{} JSON Preview Visualize

1 {
2   "channel": "politics",
3   "talks": [
4     {
5       "title": "The shocking move to criminalize nonviolent protest",
6       "speakers": "Will Potter",
7       "description": "In 2002, investigative journalist and TED Fellow Will Potter took a break from his regular beat, writing about shootings and murders for the Chicago Tribune. He went to help a local group campaigning against animal testing: \"I thought it would be a safe way to do something positive,\" he says. Instead, he was arrested, and so began his ongoing journey into a world in which peaceful protest is branded as terrorism.",
8       "url": "https://www.ted.com/talks/will_potter_the_shocking_move_to_criminalize_nonviolent_protest",
9       "schedule_time": "Streaming at 12:19 on 29/05"
10    },
11    {
12       "title": "Why I love a country that once betrayed me",
13       "speakers": "George Takei",
14       "description": "When he was a child, George Takei and his family were forced into an internment camp for Japanese-Americans, as a \"security\" measure during World War II. 70 years later, Takei looks back at how the camp shaped his surprising, personal definition of patriotism and democracy.",
15       "url": "https://www.ted.com/talks/george_takei_why_i_love_a_country_that_once_betrayed_me",
16       "schedule_time": "Streaming at 12:23 on 29/05"
17    },
18  ]
19 }
```

Per testare su questa API incolla su **Postman** questo link:  
[https://pk135uzjb1.execute-api.us-east-1.amazonaws.com/default/Get\\_Talks\\_By\\_Channel](https://pk135uzjb1.execute-api.us-east-1.amazonaws.com/default/Get_Talks_By_Channel)



# CRITICITA' E IMPLEMENTAZIONI FUTURE

**Programmazione:** con l'implementazione attuale, la programmazione dei talk che andranno in onda parte da una data che è quella del **timestamp** in cui viene eseguita la chiamata. Per il futuro andremo ad impostare una **data fissa** che non si ricalcoli ad ogni esecuzione.

**Collection:** nello sviluppo della seconda lambda function abbiamo dovuto affrontare il problema di lavorare su due collection **diverse**. Questo problema è stato risolto tramite la creazione di due **schemi** diversi che venivano poi esportati come due proprietà di un oggetto Talk.

**Tipo di dati:** il primo problema che abbiamo incontrato è stato capire con quale tipo di dati stavamo lavorando. L'errore visualizzato nel terminale indicava che non veniva trovato alcun Talk, ed era dovuto al fatto che il metodo «**findById**» non restituiva risultati. Inizialmente avevamo impostato il campo `_id` come «**Number**», ma solo dopo alcuni log di test ci siamo accorti che in realtà era definito come «**String**».

**Sottotitoli:** un'implementazione futura può essere quella di sviluppare una lambda function che sia in grado di prendere il **testo** di ogni Talk e poi tramite un API esterna sia in grado di **sincronizzarla** con lo speaker.

# I NOSTRI LINK

