TEDays

3ª consegna

A cura di: (1085186) Davide Mai, Haaim Syed (1086229)

CONTENUTI

La Lambda Function per i video consigliati

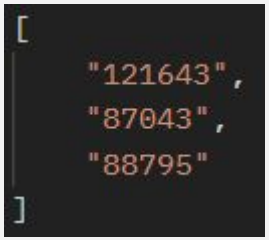
Abbiamo creato una nuova lambda function Get_Watch_Next basandoci sulla LF creata durante l'esercitazione. Questa Lambda Function permetterà all'applicazione di consigliare alcuni talk alla fine della riproduzione di un talk.
Sfruttiamo la struttura dei dati su MongoDB, in cui abbiamo salvato, per ogni talk, un array di id di talk che trattano argomenti simili. Abbiamo anche modificato lo schema dei dati.

```
const related_schema = new mongoose.Schema({
    _id: String,
    title: String,
    url: String,
    description: String,
    speakers: String,
    watch_next_id: Array,
    watch_next_title: Array,
    comprehend_analysis: mongoose.Schema.Types.Mixed
}, { collection: 'tedx_data' });

module.exports = mongoose.model('related', related_schema);
```

La Lambda Function per i video consigliati

La parte principale della nostra funzione, in cui viene cercato un video dato il suo id e viene ritornato l'array degli id dei video da consigliare.

```
try {
    await connect_to_db();
    console.log('=> get_watch_next');

    body.doc_per_page = body.doc_per_page || 10;
    body.page = body.page || 1;

    const talks = await related.find({ "_id" : body._id, "watch_next_id": {$exists: true, $ne: []}})
        .skip((body.doc_per_page * body.page) - body.doc_per_page)
        .limit(body.doc_per_page);

    console.log('Found talks:', talks);

    const watch_next_id = [].concat(...talks.map(related => related.watch_next_id));
    console.log('Related ids:', watch_next_id);

    return callback(null, {
        statusCode: 200,
        body: JSON.stringify(watch_next_id)
    });

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



Il risultato su Postman

La Lambda Function per le festività

```
const holidayName = holidayData ? holidayData.Name : 'Nome non trovato';
    const talks = await videos.find({});

    console.log('Found talks:', talks);

    const videoSimilarities = [];

    talks.forEach(element => {
        const videoTitle = element.title || '';
        const jaroSimilarity = distance.string.jaro(videoTitle, holidayName);
        videoSimilarities.push({
            videoId: element._id,
            videoTitle: videoTitle,
            holidayName: holidayName,
            fullVideoObject: element,
            similarityScore: jaroSimilarity
        });
    });

    videoSimilarities.sort((a, b) => b.similarityScore - a.similarityScore);
    const top5Videos = videoSimilarities.slice(0, 5);
    const top5VideoIds = top5Videos.map(video => video.videoId);

    let randomSelectedVideo = null;
```

Questa funzione fa interagire due lambda function. In particolare, riceve in input una data, e cerca il nome della festività utilizzando un'altra Lambda Function.

Dopo aver recuperato la lista di tutti i talk, viene usata poi una funzione di processamento del linguaggio naturale per ordinare i talk più rilevanti alla festività trovata.

I nostri dati

Cerca poi i 5 talk il cui titolo è più simile al nome della festività, e ritorna l'id di un video casuale tra quei 5.

Tutte le Lambda Function e i pacchetti utilizzati sono su GitHub.

```
    if (top5Videos.length > 0) {

        const randomIndex = Math.floor(Math.random() * top5Videos.length);

        randomSelectedVideo = top5Videos[randomIndex].fullVideoObject;
    }

    let finalResponse = {
      holidayName: holidayName,
      selectedVideo: randomSelectedVideo
};

if (randomSelectedVideo === null) {
    finalResponse.message = "Nessun video pertinente trovato o errore nella selezione.";
}

  return callback(null, {
      statusCode: 200,
      body: JSON.stringify(finalResponse)
  });
```

Wink NLP

Abbiamo utilizzato il modulo Wink NLP per poter processare il linguaggio naturale. Un modulo node.js completamente free-to-use e open source, Abbiamo in particolare usato l'algoritmo di comparazione di Jaro-Winkler, questo restituisce un valore vicino allo 0 quando le stringhe sono molto simili tra di loro, e un valore vicino a 1 quando le due stringhe non presentano molta somiglianza.

La risposta dell'API

È sufficiente passare una data all'API, e avremo come risposta tutti i dettagli del video.


```
"holidayName": "Kamehameha Day observed",
"selectedVideo": {
    "_id": "187589",
    "slug": "yue_xiu_lim_white",
    "speakers": "Yue Xiu Lim",
    "title": "\"White\"",
    "url": "https://www.ted.com/talks/yue_xiu_lim_white",
    "description": "Yue Xiu Lim has long been enamored with the peaceful sound of the Chinese guzheng, a harp-like instrument that
        dates back to ancient times. She delights the audience with her calming and elegant original piece, \"White.\"",
    "duration": "155",
    "publishedAt": "2018-09-11T19:47:11Z",
    "tags": [
        "music",
        "performance"
    ],
```

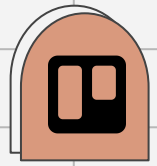
Criticità

- I titoli dei video e i nomi delle feste sono troppo corti e troppo diversi tra loro, quindi spesso i video trattano argomenti che non hanno nessuna correlazione con la festività. Questo è dato sia dalla struttura dei dati, sia dal funzionamento dei vari algoritmi di processamento del linguaggio naturale. Abbiamo utilizzato l'algoritmo che restituiva risultati migliori e con una certa varianza, in quanto gli altri algoritmi da noi testati riportavano tutti i video con la stessa similarità. Ciò portava alla visione di video scelti in maniera completamente casuale.

- Non tutti i giorni presentano una festività, è possibile che non venga visualizzato nulla in tal caso.

- I dati da processare sono tanti, la risposta dell'API non è immediata.

Grazie!



CREDITS: This presentation template was created by **Slidesgo**, and includes icons by **Flaticon** and infographics & images by **Freepik**