TED[Together]

Share, learn, discuss.

<Irene Bravo [1071820]> <Paolo Leidi [1059156]> <Matteo Mariani [1057026]> <Michele Zenoni [1059950]>

> Fase di preparazione

TED[Together]

```
# CREATE THE AGGREGATE MODEL, ADD WATCH_NEXT TO TEDX_DATASET
watch_next_dataset_agg = watch_next_dataset.groupBy(col("idx").alias("idx_ref")) \
    .agg(collect_list(struct("watch_next_idx","main_speaker","title","details","posted","url","tags"))\
    .alias("watch_next_videos"))

tedx_dataset_final = tedx_dataset_agg.join(watch_next_dataset_agg, tedx_dataset_agg._id ==  \
    watch_next_dataset_agg.idx_ref, "left") \
    .drop("idx_ref") \
    .select(col("_id"), col("*"))
```

Abbiamo modificato il job PySpark ***CreateDataLake*** per aggiungere alla collection ***tedx_video_data*** la lista *"watch_next_videos"* dei relativi video correlati, in modo da avere maggiore accessibilità e visibilità dei dati.

> Lambda: Get_Watch_Next_by_Idx

TED[Together]

L'handler (`handler.js`) è stato modificato in maniera tale da poter restituire l'elenco dei video *watch_next* dato l'id di un talk.

```
connect_to_db().then(() => {
        console.log('=> get_all talks');
        talk.find({_id: body.idx}, {_id : 0, watch_next_videos : 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.'
                })
            );
    });
```

> Lambda: Get_Watch_Next_by_Idx

TED[Together]

Il codice di `talk.js` è stato adattato per restituire la lista dei video *watch_next* in accordo con quanto specificato nella funzione dell'handler.

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

const talk_schema = new mongoose.Schema({
    _id: String,
    watch_next_videos: Array
}, { collection: 'tedx_video_data' });

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

> Lambda: Get_Talks_by_Multiple_Tags

TED[Together]

La lambda function:
`Get_Talks_by_Multiple_Tags`
permette, tramite una richiesta GET, di inviare più tag e di trovare video che fanno il match di tutti i tag inseriti.

Di default, se non vengono trovati almeno 5 video che soddisfano tutti i tag, vengono aggiunti i video che fanno il match dei primi 3 tag inseriti.

```
connect_to_db().then(() => {
    console.log('=> get_all talks');
    talk.find({tags: {$all: body._tags}})
        .skip((body.doc_per_page * body.page) - body.doc_per_page)
        .limit(body.doc_per_page)
        .then(talks => {
                if(talks.length < body.doc_per_page && body.must_match < body.doc_per_page) {
                    let slicedTags = body._tags.slice(0, body.must_match)
                    talk.find({tags: {$all: slicedTags}})
                        .limit(body.doc_per_page - talks.length)
                        .then(adjustedTalks => {
                            talks = talks.concat(adjustedTalks)
                            callback(null, {
                                statusCode: 200,
                                body: JSON.stringify(talks)
                            })
                        })
                } else {
                    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.'
            })
        );
});
```

> Lambda: Get_Talks_by_Multiple_Tags

TED[Together]

In particolare è stato aggiunto il campo “*must_match*” alla richiesta che indica il numero minimo di tag tra quelli specificati in “*_tags*” che devono essere presenti in ogni video restituito.

Qui troviamo un esempio della struttura della risposta sulla base dei tag sopra riportati.

```
{
    "_tags": ["AI", "society", "science", "TED", "choice", "future", "humanity"],
    "must_match": 3,
    "doc_per_page": 5,
    "page": 1
}
```

```
[
    {
        "video_url": "...",
        "main_speaker": "...",
        "title": "...",
        "details": "...",
        "posted": "...",
        "tags": ["AI", "society", "science", "TED", "choice", "future", "humanity", "..."],
        "watch_next_videos": [
            {
                "watch_next_idx": "...",
                "main_speaker": "...",
                "title": "...",
                "details": "...",
                "posted": "...",
                "url": "...",
                "tags": ["..."]
            },
            {...}
        ]
    },
    {...}
]
```

> Esperienza utente

TED[Together]

L’implementazione di un sistema per gestire le preferenze degli utenti nella stessa stanza permette di avere un’esperienza di **apprendimento multidisciplinare**, di scoprire come gli interessi dei partecipanti possano confluire in un unico oggetto di studio che possa arricchire tutti i presenti e di conoscere nuovi ambiti di interesse.

Permette, inoltre, di creare un momento di **apprendimento condiviso** con amici, persone interessate allo stesso campo o studenti di una stessa classe o corso.

TED[Together]

> Criticità e possibili sviluppi

TED[Together]

- Computazionalmente può essere oneroso estrarre i dati necessari da un database molto corposo
  - si potrebbe fare un calcolo anticipato dei gruppi di tag più richiesti per rendere immediata la disponibilità dei video

- in caso di calcolo anticipato, si pone il problema di mantenere consistenti i dati dopo l'aggiornamento del database

- rimane il problema di alcuni set di tag che non trovano una corrispondenza efficace per un gruppo di utenti con interessi particolarmente diversificati