Essbai Wasim (1060652)
Locatelli Matteo (1059210)

01

TEDx PausaAttiva

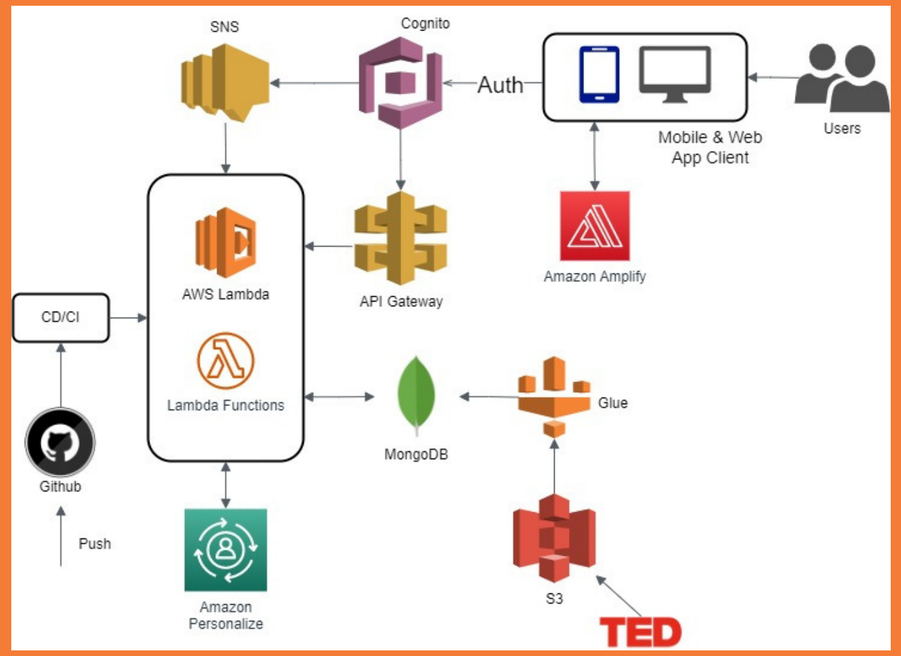
Autenticazione

Come previsto da architettura, l'autenticazione è stata implementata usando il servizio AWS **Cognito**, creando un gruppo di utenti dedicato.

Cognito fornisce una web page di default per la registrazione e login che si può trovare al seguente link.

Il servizio di autenticazione verrà usato per autorizzare l'accesso alle API esposte e per venire a conoscenza dell'utente che le chiama.

Per esporre le API è stato usato il servizio **AWS API Gateway**.



Authorizer usato per le API.

Get_Watch_Next_By_Idx

La prima lambda function realizzata serve ad esporre un API che ottiene l'ID di un talk e restituisce una lista di *watch_next*. Questa API viene usata per offrire dei suggerimenti ad un utente dopo aver finito la riproduzione di un talk.

```
console.log( => get talk by idx );
talk.find({_id: body.idx})
    .then(talk_found => {
            const watch_next_idx_list = talk_found[0].watch_next;
            console.log(watch_next_idx_list);
            console.log('=> get watch next');
            talk.find({_id: { $in: watch_next_idx_list}})
            .skip((body.doc_per_page * body.page) - body.doc_per_page)
            .limit(body.doc_per_page)
            .then( talks =>{
                console.log(talks);
                callback(null, {
                statusCode: 200,
                body: JSON.stringify(talks)
            })
        })
        .catch(err =>
          callback(null, {
            statusCode: err.statusCode || 500,
            headers: { 'Content-Type': 'text/plain' },
            body: "Get Watch Next fetch error"
          })
        );
    })
    .catch(err =>
        callback(null, {
            statusCode: err.statusCode || 500,
            headers: { 'Content-Type': 'text/plain' },
            body: 'Could not fetch the talks.'
          })
    );
```

Payload

```
{
    "idx": "3b5f6a108cac226703a717e6ff9692e3",
    "doc_per_page": 10,
    "page": 1
}
```

Risposta

```
[
    {
        "tags": Array,
        "watch_next": Array,
        "_id": String,
        "main_speaker": String,
        "title": String,
        "details": String,
        "posted": String,
        "url": String,
        "num_views": String,
        "duration": String,
        "num_likes": String
    }
]
```

Get_Talks_By_Tags_And_Duration

Questa lambda function serve per ottenere tutti i talk che matchano con determinati valori di tag e la cui durata è inferiore ad una tempistica specificata.

```
let condition_1 = {
    $add:[
            {$multiply: [{$toInt: {$substr: ["$duration", 0, 1]}}, 3600]},
            {$multiply: [{$toInt: {$substr: ["$duration", 3, 2]}}, 60]}
        ]
}
let condition_2 = {
    $add:[
            {$multiply: [{$toInt: {$substr: ["$duration", 0, 2]}}, 60]},
            {$multiply: [{$toInt: {$substr: ["$duration", 3, 2]}}, 1]}
        ]
}
let condition_3 = {
    $add:[
            {$multiply: [{$toInt: {$substr: ["$duration", 0, 1]}}, 60]},
            {$multiply: [{$toInt: {$substr: ["$duration", 2, 2]}}, 1]}
        ]
}
let addTotDuration = {
    $addFields:
    {
      "total_duration" :
      {
          $switch: {
             branches: [
                { case: { $eq: [{ $strLenCP: "$duration" }, 6] }, then: condition_1 },
                { case: { $eq: [{ $strLenCP: "$duration" }, 5] }, then: condition_2 },
                { case: { $eq: [{ $strLenCP: "$duration" }, 4] }, then: condition_3 }
             ]
          }
      }
    }
};
```

Le durate dei talk sono state salvate come stringhe, pertanto è stato necessario trasformarle in secondi. Il codice qui a sinistra mostra come fare ciò via query, usando la funzione di aggregazione di ***mongoDB***.

La parte di aggregazione e query sulla durata viene aggiunta solo se il campo durata non è vuoto.
Infine, si accoda la parte di **Skip** e **Limit**.

```
if (body.duration) {
    agg_query.push(addTotDuration);
    find_query.total_duration = {$lte: body.duration};
}

agg_query.push({$match: find_query});
agg_query.push({$skip : (body.doc_per_page * body.page) - body.doc_per_page});
agg_query.push({$limit : body.doc_per_page});
```

Get_Talks_By_Tags_And_Duration

I talk estratti vengono restituiti in ordine di popolarità, calcolata come rapporto tra numero di like e quello di visualizzazioni.

Payload

```
{
    "tags": ["design"],
    "duration": 600,
    "doc_per_page": 10,
    "page": 1
}
```

```
connect_to_db().then(() => {
    console.log("Aggregate query", agg_query);
    talk.aggregate(agg_query)
    .then(talks => {
                callback(null, {
                    statusCode: 200,
                    body: JSON.stringify(talks.sort(compareTalkPopularity))
                })
            }
        )
        .catch(err =>
            callback(null, {
                statusCode: err.statusCode || 500,
                headers: { 'Content-Type': 'text/plain' },
                body: 'Could not fetch the talks.'
            })
        );
});
```

Risposta

```
[{
    "_id": String,
    "main_speaker": String,
    "title": String,
    "details": String,
    "posted": String,
    "url": String,
    "num_views": String,
    "duration": String,
    "tags": Array,
    "watch_next": Array,
    "num_likes": String,
    "total_duration": Int
}]
```

Share_Talks

È stato sviluppato un servizio per dare la possibilità di condividere talks tra gli utenti.

```
const params = {
    UserPoolId: process.env.userPoolId,
    Filter: `username = \"${body.id_user_to_share}\"`,
}

cognito.listUsers(params, function(err, data) {
  if (err){
      callback(null, {
            statusCode: err.statusCode || 500,
            headers: { 'Content-Type': 'text/plain' },
            body: 'Could not fetch the user to share.'
        })
      return;
  }

  if(!data.Users){
      callback(null, {
                    statusCode: 500,
                    headers: { 'Content-Type': 'text/plain' },
                    body: 'User to share not found.'
        })
    return;
  }
});
```

Si controlla, tramite le API di cognito, che l'utente a cui si condivide il talk esista. Per fare ciò è stato creato un nuovo layer che contiene il package ***aws-sdk***.

SI controlla l'esistenza del talk.

```
connect_to_db().then(() => {
    talk.find({_id: body.idx}).then(talkToShare => {
                    console.log('TedxTalk to Share', talkToShare)
                    if (!talkToShare[0]) {
                      callback(null, {
                          statusCode: 500,
                          body: 'Talk to share not found'
                      })

                      return;
                    }

                    add_user_share(username, body, callback);
                })
                .catch(err =>
                  callback(null, {
                      statusCode: err.statusCode || 500,
                      headers: { 'Content-Type': 'text/plain' },
                      body: 'Could not fetch the talk to share.'
                  })
                );
});
```

Formato dati

```
  id_user: "278f0915-3aa5-4e38-af34-dab5e2d57111"
  id_user_to_share: "0faf2c0e-a3f9-4528-a0ae-a32455a98124"
  __v: 0
▸ idx_shared_talks: Array
```

```
function add_user_share(username, body, callback) {

  const query = {"id_user": username,"id_user_to_share": body.id_user_to_share};
  const update = { $push : { "idx_shared_talks" : body.idx}};
  const options = { upsert: true };

  user_shares.updateOne(query, update, options).then(updated => {
                callback(null, {
                    statusCode: 200,
                    body: "UPDATED!"
                })
            })
            .catch(err =>
                callback(null, {
                    statusCode: err.statusCode || 500,
                    headers: { 'Content-Type': 'text/plain' },
                    body: 'Could not share the talk. Try again.'
                })
            );
}
```

Si fa l'update di record, creandolo se non esiste. **upsert** fa ciò in automatico.

Payload

```
{
    "idx": "12927c6bea0aa4f763e95cadd02a1d53",
    "id_user_to_share": "0faf2c0e-a3f9-4528-a0ae-a32455a98124"
}
```

Esperienza utente

Grazie alle funzioni sviluppate, gli utenti possono esplorare nuovi talk e interagire con altri utenti che hanno interessi simili.

La piattaforma non solo offre la possibilità di esplorare i talk, ma tiene anche conto del tempo a disposizione degli utenti per navigare tra di essi.

Criticità

- Si può condividere un talk più volte allo stesso utente.
- Il formato delle durate non è omogeneo nei diversi record.
- Suggerire un video per tag potrebbe portare a talk che trattano anche di argomenti diversi.

Sviluppi futuri

- Si potrebbe tenere traccia di tutti i talk visti da ciascun utente per non suggerirli.
- Lo score di popolarità potrebbe considerare altri fattori, quali il numero di commenti, dislike e di utenti che hanno salvato il video.
- Aggiungere la possibilità di escludere tag nella ricerca per avere risultati più pertinenti.

TEDxPausaAttiva

Board GitHub

Thank you!