LIVE

TEXPRESSO

Maffeis Riccardo 1085706 Zanotti Matteo 1085443

Get_watchnext_by_ID

La funzione, ricevuto l'ID di un talk, va a cercare nel database tutte le informazioni collegate.

Funzionamento:

- Riceve una richiesta (ID)
- Controlla che sia un JSON valido
- Verifica che l'ID esista
- Consulta il database: cerca il talk corrispondente e ne legge i dettagli.
- Restituisce i tag associati al talk e una lista di video correlati, pronti da mostrare all'utente.

Lo scopo finale è preparare la lista di "video consigliati" o "simili" da proporre a chi sta navigando.

```
module.exports.get_related_ids = async (event, context, callback) => {
  context.callbackWaitsForEmptyEventLoop = false;
  console.log('Received event:', JSON.stringify(event, null, 2));

  let body = {};
  if (event.body) {
    try {
      body = JSON.parse(event.body);
    } catch (err) {
      return callback(null, {
        statusCode: 400,
        headers: { 'Content-Type': 'text/plain' },
        body: 'Invalid JSON input.'
      });
    }
  }

  const talkId = body.talk_id || body._id;
  if (!talkId) {
    return callback(null, {
      statusCode: 400,
      headers: { 'Content-Type': 'text/plain' },
      body: 'talk_id is required.'
    });
  }

  try {
    await connect_to_db();
    console.log('=> Fetching talk:', talkId);

    const foundTalk = await Talk.findById(talkId).lean();
    if (!foundTalk) {
      return callback(null, {
        statusCode: 404,
        headers: { 'Content-Type': 'text/plain' },
        body: 'Talk not found.'
      });
    }

    console.log('foundTalk.tags =', foundTalk.tags);

    return callback(null, {
      statusCode: 200,
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify({
        tags: foundTalk.tags || [],
        related_videos_details: foundTalk.related_videos_details || []
      })
    });
  } catch (err) {
    console.error('Error fetching related_ids:', err);
    return callback(null, {
      statusCode: 500,
      headers: { 'Content-Type': 'text/plain' },
      body: 'Could not fetch related_ids.'
    });
  }
};
```

Lambda Functions

Get_news_by_tag

Get_Talk_Random

Get_Talk_by_Tag

Get_newsapi_by_tag

Get_Talk_Random

Questa funzione pesca un talk random dal Database e ne restituisce l'ID, quando viene chiamata:

- Si collega al database
  Apre una connessione a MongoDB per accedere alla collezione dei talk.

- Pesca un talk casuale
  Con un'operazione di "sampling" prende esattamente un documento a caso dalla raccolta.

- Eventualmente gestisce gli errori

```
module.exports.get_random_talk = async (event, context, callback) => {
  context.callbackWaitsForEmptyEventLoop = false;
  console.log('Picking a random talk');

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

    const [randomTalk] = await Talk.aggregate([
      { $sample: { size: 1 } }
    ]);

    if (!randomTalk) {
      return callback(null, {
        statusCode: 404,
        headers: { 'Content-Type': 'text/plain' },
        body: 'Nessun talk disponibile.'
      });
    }

    console.log('=> got randomTalk:', randomTalk._id);
    return callback(null, {
      statusCode: 200,
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify(randomTalk)
    });

  } catch (err) {
    console.error('Error fetching random talk:', err);
    return callback(null, {
      statusCode: 500,
      headers: { 'Content-Type': 'text/plain' },
      body: 'Errore durante l\'estrazione del talk casuale.'
    });
  }
};
```

Get_Talk_by_Tag

Questa funzione permette di ottenere tutti i talk associati a un certo argomento e arricchirli con l'analisi delle frasi chiave:

- Ricezione e validazione
  Si aspetta un JSON con «tag» obbligatorio e due parametri facoltativi «doc_per_page» e «page» per la paginazione.

- Lettura dal database
  Si connette a MongoDB e cerca tutti i talk che contengono quel tag.

- Risposta
  In caso di successo, risponde 200 con un array di talk "arricchiti" in JSON.

```
module.exports.get_by_tag = async (event, context, callback) => {
    context.callbackWaitsForEmptyEventLoop = false;
    console.log('Received event:', JSON.stringify(event, null, 2));

    let body = {};
    if (event.body) {
        body = JSON.parse(event.body);
    }

    if (!body.tag) {
        return callback(null, {
            statusCode: 500,
            headers: { 'Content-Type': 'text/plain' },
            body: 'Could not fetch the talks. Tag is null.'
        });
    }

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

    try {
        await connect_to_db();
        console.log('=> get_all talks');

        const talks = await talk.find({ tags: body.tag })
            .skip((body.doc_per_page * body.page) - body.doc_per_page)
            .limit(body.doc_per_page);

        const enrichedTalks = await Promise.all(talks.map(async t => {
            if (!t.comprehend_analysis && t.description) {
                try {
                    const analysis = await analyzeKeyPhrases(t.description);
                    t.comprehend_analysis = analysis;
                    await t.save();
                } catch (err) {
                    console.error(`Comprehend error on talk ${t._id}:`, err);
                }
            }
            return t;
        }));

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

    } 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.'
        });
    }
};
```

Get_news_by_tag

Questa Lambda function prende in ingresso un set di tag o keyword, le normalizza sempre in un array, eliminando spazi vuoti e voci nulle.

A quel punto costruisce dinamicamente l'URL verso l'endpoint di NewsData.io, passando API key, nazione, lingua e le parole-chiave, e restituisce così la risposta grezza delle news filtrate per quei termini.

```
exports.handler = async (event) => {
  try {
    let keywords = [];

    if (event.queryStringParameters && event.queryStringParameters.tag) {
      keywords = event.queryStringParameters.tag
        .split(',')
        .map(t => t.trim())
        .filter(t => t.length > 0);

    } else if (event.body) {
      let bodyObj;
      try {
        bodyObj = JSON.parse(event.body);
      } catch (_) {
        bodyObj = {};
      }
      if (Array.isArray(bodyObj.tags)) {
        keywords = bodyObj.tags
          .map(t => (typeof t === 'string' ? t.trim() : ''))
          .filter(t => t.length > 0);
      } else if (typeof bodyObj.tags === 'string' && bodyObj.tags.trim() !== '') {
        keywords = bodyObj.tags
          .split(',')
          .map(t => t.trim())
          .filter(t => t.length > 0);
      }
    }

    const url = new URL('https://newsdata.io/api/1/latest');
    url.searchParams.append('apikey', NEWS_API_KEY);
    url.searchParams.append('country', 'it');
    url.searchParams.append('language', 'en');
```

Get_newsapi_by_tag

Questa funzione si aspetta due parametri:

- query (obbligatoria): la parola-chiave da cercare
- pages (facoltativa): quante pagine di risultati leggere

Dopo aver verificato che query, lancia ripetute chiamate a fetchNewsPage per ciascuna pagina, fermandosi se non ci sono più articoli.

Raccoglie tutti gli articoli, quindi per ognuno costruisce un oggetto semplificato con titolo, descrizione, URL, data di pubblicazione e – soprattutto – estrae fino a 5 tag chiave dal testo.
Alla fine restituisce un JSON con l'array di articoli "taggati".


```
exports.handler = async (event) => {
  try {
    let { query, pages } = event.queryStringParameters || {};
    if (!query && event.body) {
      const body = JSON.parse(event.body);
      query = body.query;
      pages = body.pages;
    }
    if (!query) {
      return {
        statusCode: 400,
        body: JSON.stringify({ message: 'Missing "query" parameter' })
      };
    }
    const totalPages = parseInt(pages, 10) || 1;

    let allArticles = [];
    for (let page = 1; page <= totalPages; page++) {
      const arts = await fetchNewsPage(query, NEWSAPI_KEY, page);
      if (!arts.length) break;
      allArticles.push(...arts);
    }

    const tagged = allArticles.map(art => {
      const text = [art.title, art.description, art.content]
        .filter(Boolean)
        .join(' ');
      const tags = text
        ? extractTags(text).slice(0, 5)
        : [];
      return {
        source:      art.source?.name,
        author:      art.author,
        title:       art.title,
        description: art.description,
        url:         art.url,
        publishedAt: art.publishedAt,
        content:     art.content,
        tags
      };
    });

    return {
      statusCode: 200,
      headers: { 'Content-Type': 'application/json' },
      body: JSON.stringify(tagged)
    };

  } catch (err) {
    console.error('Errore interno:', err);
    return {
      statusCode: 500,
      body: JSON.stringify({ message: 'Internal error extracting tags' })
    };
  }
};
```

Esperienza Utente

Grazie a queste Lambda function l'utente potrà accedere alle notizie e ai talk di TEDx secondo i propri interessi, in particolare verranno mostrati dei talk, in base ai tag, che saranno salvati in secondo piano nel suo account, e allo stesso modo avrà delle notizie collegate agli interessi.

Nel caso egli voglia diversificare potrà recarsi nella sezione cerca e cercare un tag da lui richiesto oppure vedere dei talk casuali e indipendenti dai tag salvati.

Criticità

Numero di richieste Lamda limitate

API key necessarie

Limiti di aggiornamento basati sul tempo

Limiti di connessione in caso di tanti dati

NEWS

Possibili Evoluzioni

01 Aggiunta di un trigger

02 Integrare API a pagamento

03 Sistemazione parametri per aumentare la velocità del fetch

LIVE

TEXPRESSO