Facultad de Ingeniería

Universidad de la República

Webir 2021

Informe Final

Grupo 11

Maika Lemes	5.392.374–2
Sebastián Pandolfi	4.850.374–5
Bruno Rienzi	2.914.248–3
Gabriel Astapenco	4.606.113–5
Gabriel Corujo	4.791.612–3

{maika.lemes, sebastian.pandolfi, brienzi,
gabriel.astapenco, gabriel.corujo}@fing.edu.uy

Índice

1. Introducción

En esta instancia se busca crear un sistema alternativo para recuperar información (en este caso videos) de YouTube[1], mediante el uso de una API pública y una herramienta conocida con motor de base de datos orientado a documentos e índices como lo es ElasticSearch[2].

Una vez recuperada dicha información, la misma se filtra y se le presenta al usuario de manera ordenada (según criterios) y amigable.

En este documento se comienza entonces realizando una descripción del problema en cuestión junto a la solución implementada. Luego, se detalla su diseño e implementación y finalmente se muestran las funcionalidades y resultados obtenidos del sistema.

Además se discuten luego posibles mejoras como trabajo futuro, para hacer el sistema más mantenible, efectivo en sus búsquedas y que además cubra un espectro más amplio de funcionalidades.

2. Problema

Normalmente cuando buscamos una canción en YouTube, estamos acostumbrados a buscar dicha canción por su título, su intérprete o una combinación de palabras entre el título y el intérprete, pues esto es lo que permite el buscador integrado en el sitio.

¿Qué sucedería si se quisiera realizar dicha búsqueda basada en otras especificaciones, como por ejemplo coincidencias con la letra de la misma? En este caso, sería de mucha ayuda contar con un buscador alternativo donde el usuario pueda ingresar en un campo de texto libre un fragmento de la letra de una canción para encontrarla. Esto podría ser muy útil para encontrar una canción para la cual no se recuerdan ni fragmentos del título o el intérprete pero en la que sí se recuerda parte de su letra.

Para ello se precisaría almacenar la información relacionada a la letra de las canciones en alguna base de datos orientada a documentos, en la cual se pueda almacenar grandes volúmenes de información (en este caso la letra de todos los videos) y a su vez crear diversos índices y estrategias para recuperar y filtrar esta información como se desee.

En este caso, a diferencia de YouTube que busca a través del nombre o la descripción del video, una alternativa para llevar a cabo lo mencionado podría ser buscar a través de los subtítulos (captions) auto generados por YouTube.

Esto presenta un desafío ya que:

- No todos los videos contienen captions auto-generados.
- No todos los idiomas son soportados por esta funcionalidad.
- La traducción automática voz-texto no es tan sencilla ya que depende de los idiomas, y a su vez en las canciones se modifica, acorta o utiliza de diferente manera el lenguaje para que se adecúe al ritmo de la canción.
- Cada video/canción contiene un volúmen significativo de texto, por lo que se necesita una cantidad considerable de almacenamiento.
- Como las búsquedas recorren grandes volúmenes de datos, estas pueden tener altos tiempos de respuesta.
- Las letras de las canciones contienen una alta cantidad de palabras no-relacionadas en común entre ellas, como por ejemplo los pronombres, los cuales deberían tratarse con cuidado para no mostrar resultados muy alejados de lo que se busca.

3. Enfoque de la solución

Para llevar todo esto a cabo, se propone como solución un sistema web de búsqueda y recuperación de información, el cual almacene la información (captions automáticos de los videos de música de YouTube) en una base de datos acorde para este fin, y luego la misma pueda ser recuperada mediante búsquedas avanzadas con índices, filtros personalizados y/o algoritmos de búsqueda con tiempos de respuesta acordes para el volúmen de datos que se consulte.

La información almacenada en la base de datos mencionada sería obtenida mediante un proceso automático separado del sistema. Este proceso utilizaría la API de YouTube y se ejecutaría en horarios de menor frecuencia de uso del sistema, recorriendo los diferentes videos de YouTube y así guardando sus captions indizados junto con sus metadatos para que puedan ser recuperados posteriormente por búsquedas hechas por los usuarios del sistema.

Estas búsquedas por parte de los usuarios se realizarían en un sitio web con un buscador, donde el usuario ingresaría parte de la letra de una canción que le guste, y el sistema le mostraría de forma amigable y ordenada los videos cuyos captions coincidan total o parcialmente con la búsqueda efectuada. En esta búsqueda el usuario también podría indicar diferentes filtros para así personalizarla más y ayudar al sistema a encontrar resultados más exactos o relacionados con lo que el usuario realmente espera.

Además de los vídeos correspondientes a la búsqueda realizada, en cada ítem del listado de vídeos mostrado se incluiría información o metadatos sobre los resultados obtenidos, para que el usuario sepa información sobre el vídeo encontrado como puede ser el autor, título de la canción, género, etc.

4. Diseño

En un principio se estudiaron diferentes tecnologías para evaluar cuales serían las mas adecuadas para implementar la solución. Luego de investigar se decidió utilizar *ElasticSearch*[2] para la base de datos, *NodeJS*[3] para el Backend y *React*[3] para el Frontend.

El diagrama de los componentes del sistema es el siguiente:

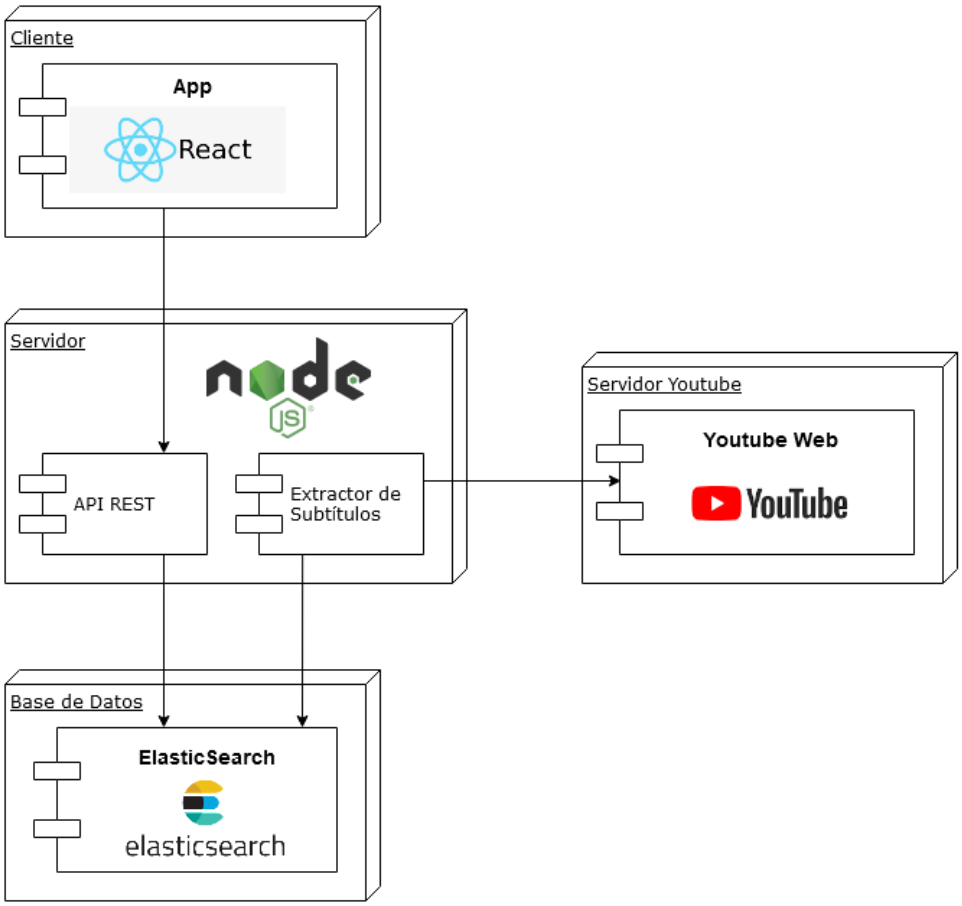

Figura 1: Arquitectura del Sistema

Para la Base de Datos se utilizó *ElasticSearch* porque al momento de investigar sobre las posibles tecnologías se comprobó que era una de las herramientas mas adecuadas para este aplicación ya que los datos que se deben almacenar son textos, y esta herramienta esta pensada para guardar, buscar y analizar grandes volúmenes de texto. Además posee un lenguaje de consulta muy completo, permitiendo buscar por campos y palabras claves, cosa que podía ser de utilidad para implementar la solución, además de devolver un puntaje al momento de hacer la búsqueda permitiendo comprobar que tan acertada fue la misma.

Para el Backend se decidió utilizar *NodeJS* por su facilidad para crear un servidor REST sobre el cual se harían las consultas, otro punto decisivo fue que algunos de los

integrantes tenían un poco de experiencia en el lenguaje por lo que no sería necesario dedicar tanto tiempo a aprender el mismo.

Para el Frontend se decidió utilizar *React* porque crear una aplicación web sencilla no es complicado en el lenguaje, en este caso también ocurrió que algunos de los integrantes del grupo tenían algo de experiencia en el lenguaje, influyendo al momento de decidir el mismo.

5. Implementación

5.1. Extracción de Subtítulos

Para extraer los subtítulos de los vídeos de Youtube se utilizaron dos técnicas diferentes, una para obtener los subtítulos manuales (los que se agregan a mano) y otra para obtener los subtítulos generados automáticamente.

Para poder obtener los **subtítulos manuales** primero se hace una consulta al siguiente endpoint de Youtube:

`https://www.youtube.com/api/timedtext?type=list&v=videoID`

De aquí se obtiene un archivo XML que contiene información de los lenguajes de los subtítulos manuales que existan para el vídeo, para obtener solo la información necesaria se utilizan expresiones regulares. En caso de comprobar que existen para el lenguaje deseado se hace la siguiente consulta:

`https://www.youtube.com/api/timedtext?lang=codigoLenguaje&v=videoID&name=nombreLenguaje`

De aquí se obtienen un archivo XML con los subtítulos, por lo que es necesario "limpiar"los datos para obtener solamente los subtítulos, para esto se utilizan expresiones regulares.

En caso de que no existan subtítulos manuales para los lenguajes deseados se obtienen los **subtítulos generados automáticamente**, para esto se utiliza un módulo de *nodeJS*, `youtube-dl`, este módulo provee muchas funcionalidades para trabajar con Youtube.

Una vez de obtienen los subtítulos estos se guardan en la base de datos.

5.2. API

El backend contiene los siguientes endpoints:

- GET `http://localhost:4000/api/v1//videos/search?text=textoBuscado`
  el cual hace la búsqueda de el texto en la base de datos de *ElasticSearch*, creando:

```
query = {
     index: ‘‘videos’’,
     body: {
          query: {
             match: ‘‘captions’’: textoBuscado
          }
     }
}
```

para luego llamar a `elasticClient.search(query)` y devolver los resultados al Frontend.

- POST `http://localhost:4000/api/v1/videos/videoID`

  con los parámetros: ID del vídeo (el cual se utiliza como KEY dentro de la base de datos) y las captions del video, de la siguiente forma:

  ```
  { videoID: videoId, captions: videoCaptions }
  ```

5.3. Dificultades

- Al momento de empezar a familiarizarse con como maneja Youtube los subtítulos se comprobó que hay vídeos en los que no se pueden obtener ningún tipo de subtítulos, debido a que el usuario que los sube puede decidir no permitir que se generen automáticamente. Se buscaron diferentes formas de intentar obtenerlos de alguna otra manera pero se comprobó que no era posible obtenerlos directamente.

- Se tuvieron que “limpiar” los subtítulos para poder obtenerlos de manera correcta (debido a que la consulta los devuelve con formato XML), también se tuvo que tener en cuenta que muchos de los subtítulos manuales contienen iconos o símbolos que si no se eliminar generan que las búsquedas sean menos acertadas, por lo que fue necesario eliminar esos elementos.

- Al final del proyecto, al momento de crear este informe, se presentó un problema con los endpoints de Youtube. Lo que ocurrió fue que Youtube dio de baja los endpoints que se usaban para obtener los subtítulos manuales (explicados en la sección Extracción de Subtítulos), por lo que ya no es posible obtenerlos de esta forma. Al mismo tiempo, el módulo de *NodeJS* que se utilizaba para obtener los subtítulos auto-generados (los cuales se obtenían en caso de que no existieran subtítulos manuales) también dejó de funcionar, sospechando que el mismo también hacia uso de alguno de los endpoints que Youtube dio de baja.

  Debido a que este problema ocurrió muy cerca de la fecha de entrega del proyecto se decidió dejar la implementación como estaba para poder mostrarla, ya que no iba a ser posible buscar otra solución para el problema en tan poco tiempo, y utilizar la base de datos con los subtítulos que ya estaban cargados antes de que ocurriera este cambio en los endpoints.

6. Funcionalidades y uso

Al abrir la aplicación se puede observar que cuenta con un buscador básico, donde se debe ingresar el texto de la canción:

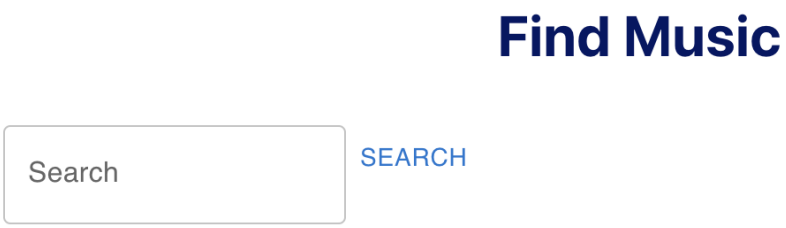

Figura 2: inicio

Al ingresar el texto deseado se realiza la búsqueda y se despliegan los resultados.

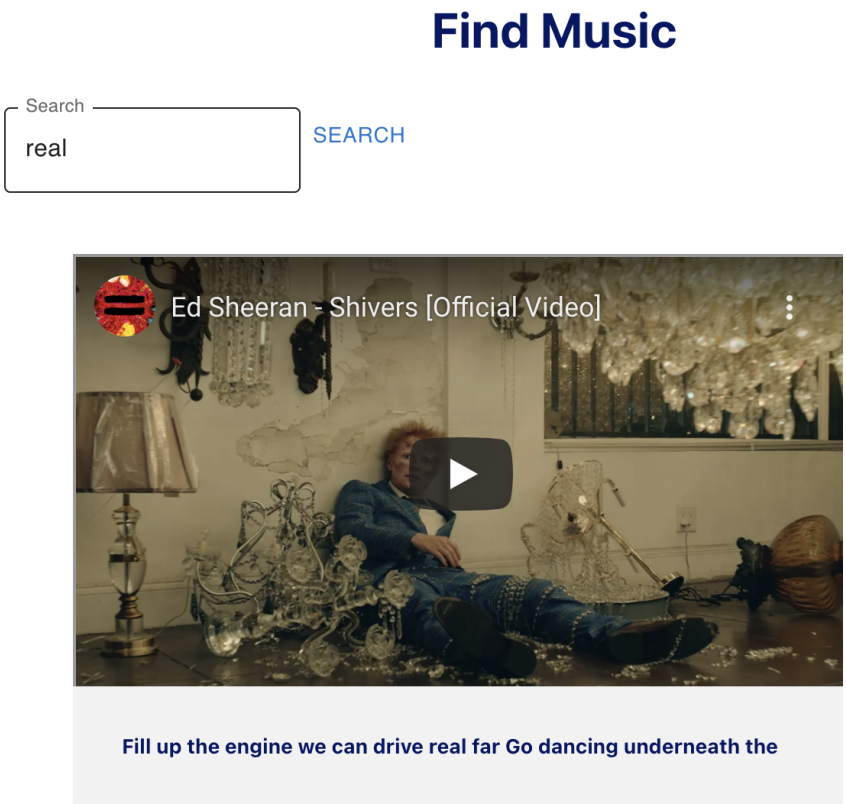

Figura 3: Búsqueda con un único resultado

En este caso, únicamente se encontró un único resultado que coincida con la búsqueda. Además del vídeo (ya listo para reproducir), se le agregó debajo la frase donde se encontró la búsqueda.
En el caso de que no se encuentre la frase explícitamente la búsqueda es realizada a

partir de cada una de las palabras contenidas en la frase.
Esto pareció útil ya que sin la necesidad de tener que reproducir el vídeo, el usuario se puede dar cuenta si dicho vídeo es o no el deseado.

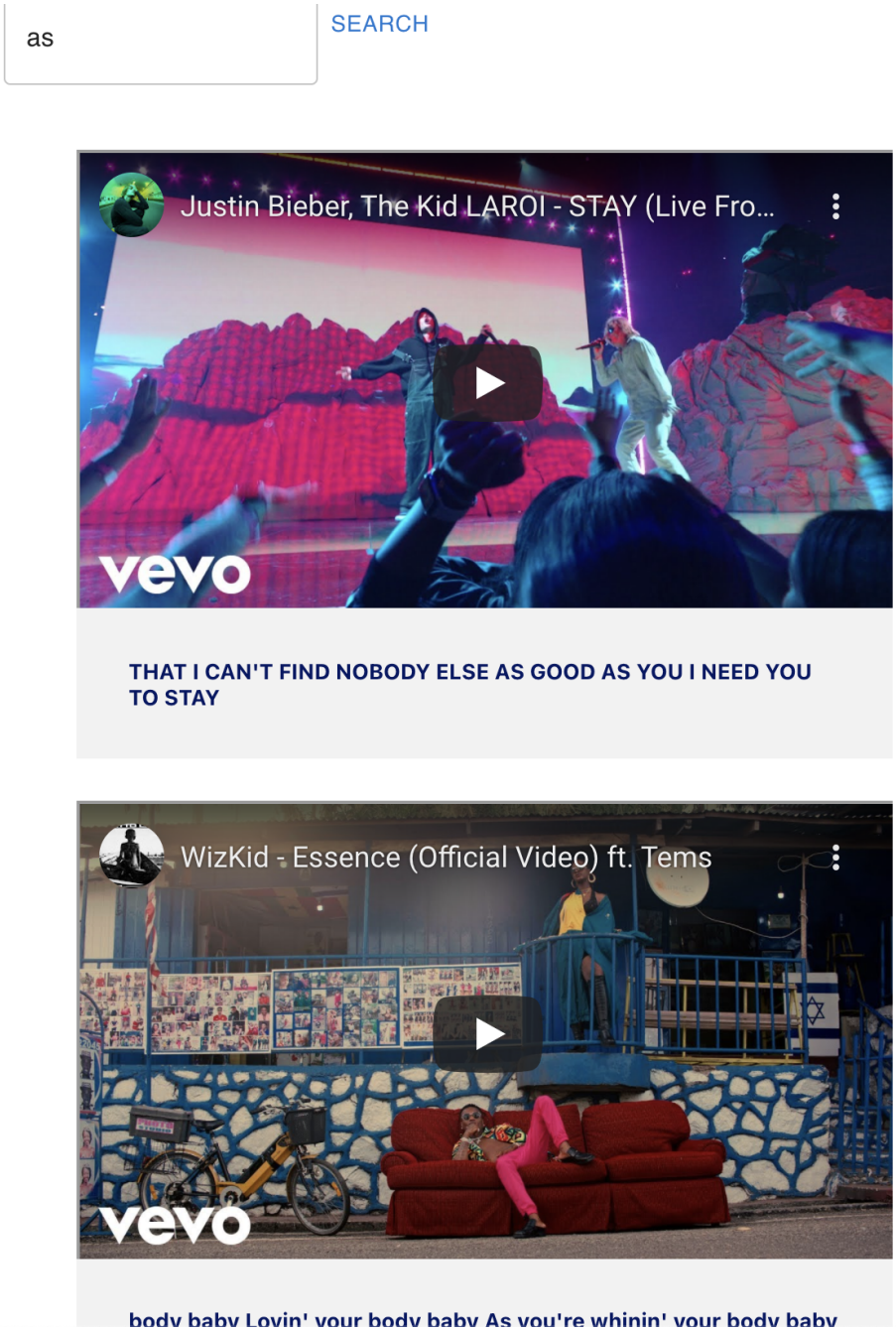

Figura 4: Búsqueda con varios resultados

Este otro caso de uso, la búsqueda tiene varias coincidencias. Por lo tanto se despliegan todos los resultados, donde son ordenados por el *Score* que les asigna *ElasticSearch*.

7. Evaluación y resultados

Se obtuvo de esta forma un sistema amigable para el usuario, capaz de extender la funcionalidad principal de YouTube de búsqueda de videos basada en coincidencias con el título o la descripción del mismo.

En este sistema, el usuario es capaz de buscar una canción sin siquiera saber su título o descripción, solamente conociendo parte de la letra de la misma. Esto es muy útil ya que sucede que en la gran mayoría de los casos recordamos el estribillo o parte de la letra de una canción pero no su título (y menos aún la descripción que podría tener el video de esta canción publicado en YouTube).

A su vez, la búsqueda en el sistema es efectiva ya que se utiliza un motor diseñado para esta clase de búsquedas, como lo es ElasticSearch, en el cual se generan índices y filtros según se vea y crea necesario para la funcionalidad que se esté implementando.

8. Conclusiones

Se logró crear una aplicación que cumple con la idea propuesta al inicio del proyecto, por lo que se cree que se hizo un buen trabajo. Cabe destacar de igual forma que se pensaron otras funcionalidades que mejorarían la aplicación y que parecen interesantes de implementar, pero debido a los tiempos no fue posible hacerlo. Se comentan algunas de ellas en la sección Trabajo a Futuro.

El proyecto también brindó la posibilidad de trabajar y aprender tecnologías nuevas para todos los participantes, principalmente *ElasticSearch*, la cual resultó ser una herramienta muy potente para trabajar con textos.

Además podemos concluir que es posible observar como mediante la recuperación de información en internet existe un gran potencial para sacar provecho de información de público acceso, logrando resultados realmente interesantes que pueden aportar a diferentes utilidades en nuestra vida cotidiana.

9. Trabajo Futuro

9.1. Endpoints de Youtube

Solucionar el problema que se presentó al final del proyecto con los endpoints de Youtube. Buscando nuevos endpoints u otra forma de obtener los subtítulos manuales, y buscar otro módulo de *NodeJs* del cual se puedan obtener los subtítulos auto-generados.

9.2. Filtros

El agregado de filtros extras permite al usuario personalizar más su búsqueda, es decir manualmente guiar mejor al sistema a que recupere la información más acercada a lo que este usuario desee.

A su vez, cuanta más información se tenga sobre lo que el usuario desee, más fácilmente se pueden ejecutar las búsquedas mediante la utilización de filtros o índices personalizados para el caso y más certeza se va a tener de que los resultados obtenidos son los deseados.

9.3. Idiomas

El sistema actual implementado tiene una cantidad acotada y usual de idiomas soportados, pero sería deseable agregar más idiomas para que más usuarios puedan acceder al mismo. El agregar más idiomas haría a su vez más compleja a la búsqueda, nuevos índices deberían ser creados y el volúmen de datos aumentaría significativamente, por lo que hay que ser prudente cuando se desee implementar.

9.4. Metadatos

En cualquier sistema de búsqueda y recuperación de información que se presenta al usuario, es necesario que el usuario obtenga datos relacionados al resultado obtenido, como puede ser el texto que coincidió con la búsqueda que él mismo realizó, la fuente de dicha información, fecha y/ autores según aplique para el resultado obtenido, etc.

Actualmente en nuestro sistema, debajo de cada video mostrado como resultado se muestra el texto completo (frase u oración) relacionado a la coincidencia que se obtuvo con respecto a la búsqueda efectuada (de esta forma el usuario puede leer la letra completa para confirmar si es efectivamente lo que estaba buscando).

Para mejoras futuras se podría brindar más información sobre los datos relacionados al resultado mostrado (metadatos), mostrando alguno de los datos mencionados anteriormente.

9.5. Presentación

La interfaz planteada al usuario al momento es sencilla y minimalista, pero para dar una buena impresión y fomentar su uso sería deseable mejorar la misma cambiando el diseño de las capas, los campos y botónes de búsqueda, la sección que brinda los metadatos para cada video, etc.

10. Referencias

[1] YouTube
https://youtube.com

[2] ElasticSearch
https://www.elastic.co/elasticsearch/

[3] NodeJS
https://nodejs.org

[4] React
https://reactjs.org/