MEMORIA

Autor: Jonatan García González

ÍNDICE

1.Descripción del problema

Como sabemos la industria del cine es una de las fuentes más grandes de entretenimiento que existen. Las personas solemos disfrutar introduciéndonos en los mundos creados en los escenarios. Con ellos se levantan ánimos en días apagados, sentimos euforia al ver una gran escena de acción o incluso pueden lograr aflorar un sinfín de sentimientos varios. La industria del cine es una industria en constante crecimiento a pesar de su longevidad. Rara es la persona que en un momento determinado no ha decidido navegar por los cientos de títulos existentes en busca de alguna película que visionar y con la que entretenerse. En ocasiones resulta complicado decidir qué película elegir debido al gran número disponible en el mercado. Además de este gran número nos encontramos, así mismo, con cuantiosos géneros y directores, conllevando una gran pérdida de tiempo decidiendo una elección.

En mi opinión esto supone un dilema a tener en cuenta, ya que hoy el tiempo es un bien muy apreciado que nadie está dispuesto a perder. Esto es lo que me ha llevado a la creación de esta aplicación con el objetivo claro de buscar con rapidez qué ver entre una gran oferta establecida de películas.

Esta aplicación te irá recomendando, según tus gustos personales, determinadas películas y servirá para realizar búsquedas generales según géneros y directores.

Estos parámetros vendrán dados por las búsquedas que hayas realizado en la aplicación, tales como: tus directores favoritos o tus géneros preferidos. Gracias a esto te podrá hacer recomendaciones muy específicas sin necesidad de buscar entre cientos de películas distintas. Como añadido permitirá ver la media de las notas que otros usuarios le han otorgado a cada película, ya que he observado que mucha gente se deja llevar por las opiniones grupales. Las recomendaciones no serán muy numerosas con el objeto de no sobrecargar en la elección y a la vez será ajustada a lo que se esté buscando en ese momento.

2.Herramientas

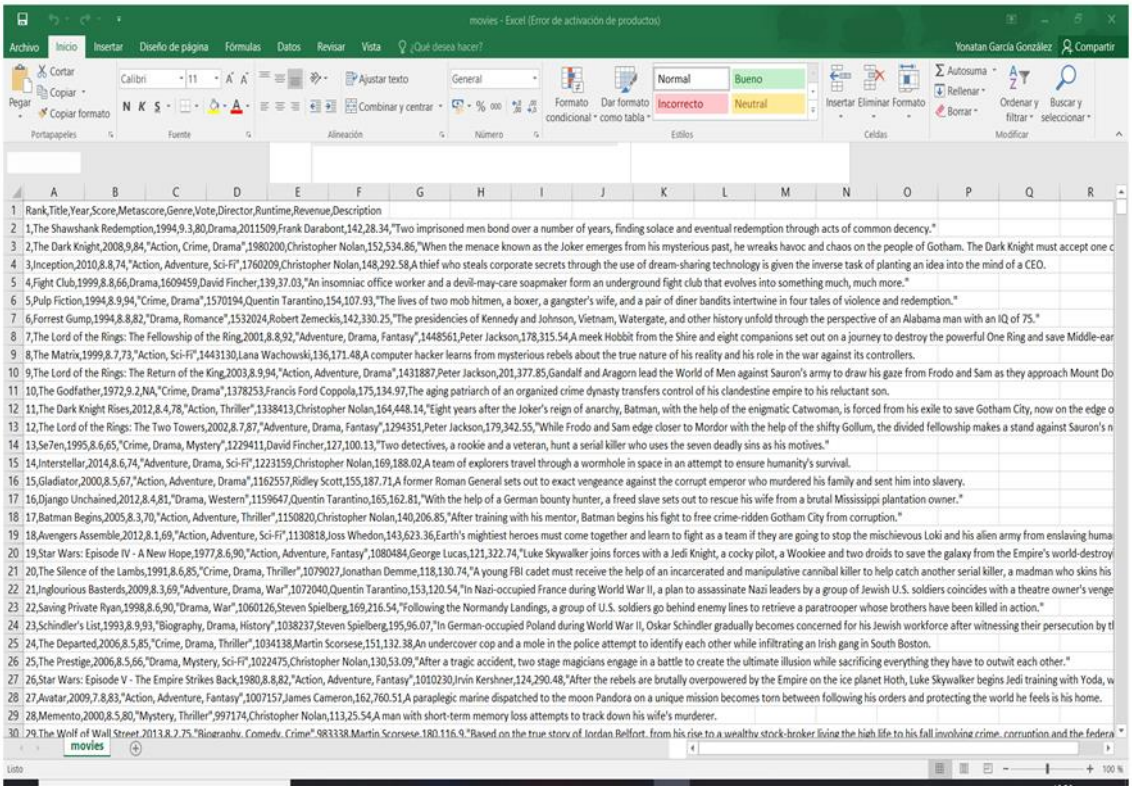
Las herramientas que he utilizado en este proyecto son:

NEO4J: es un software libre de Base de datos orientada a grafos. Gracias a este software pude conectar la base de datos escogida para este proyecto con mi aplicación.

Para ello primer tuve que aprender sobre cómo usar neo4j ya que, como es habitual en este tipo de aplicaciones, tenía su propio lenguaje: Cypher.

Para aprender este lenguaje recurrí tanto a videos de YouTube como a una guía elaborada por Andrea Fernández, una alumna que cursó esta asignatura el año anterior.

Una vez profundizado y asimilado el lenguaje Cypher busqué una base de datos acorde a lo que necesitaba. La encontré en Kaggle, una página web que almacena múltiples bases de datos. Dentro de ella elegí la base de datos de la página IMDB.



Después de esto encendí neo4j lo configuré y añadí dicha base de datos.

Posteriormente realicé el paso más complicado de esta parte, que es desarrollar el código en Cypher, el cual tuve que cambiar numerosas veces hasta encontrar la consulta que estaba buscando.

Esta es la sentencia que logre sacar:

```
LOAD CSV WITH HEADERS FROM 'file:///movies.csv' AS line
WITH line LIMIT 1000
CREATE (m:Movie { Title: line.Title, year: toInteger(line.Year), director: line.Director, score: line.Score, genre:
line.Genre })
FOREACH (n IN (CASE WHEN line.Director IS NULL THEN [] ELSE [1] END) |
MERGE (d:Director {name: line.Director})
CREATE (d)-[:DIRECTED]→(m)
)
FOREACH (n IN (CASE WHEN line.Genre IS NULL THEN [] ELSE [1] END) |
MERGE (g:Genre {name: line.Genre})
CREATE (m)-[:OF_GENRE]→(g)
)
```

Finalizado este punto comenzamos con la aplicación de tipo web, estilo de aplicación que trabajo con comodidad y que me resultó más sencilla y funcional para este caso concreto.

Este tipo de aplicaciones constan de dos partes: **Backend** y **Frontend**.

El **Backend**, es el primero que se debe de elaborar en la aplicación debido a que es la que realiza las consultas a la base de datos.

El **Backend** usa tecnologías como **node.js**, que es la encargada de la estructura **express,** necesaria para conectar backend y frontend. **El conector de neo4j,** es indispensable para conectarse a nuestra base de datos de neo4j, y por supuesto el lenguaje **Javascript,** indispensable para hacer funcionar el Backend.

El **Frontend,** es el que maneja la interfaz. Todo lo que se realice cuando se use la aplicación se conectará al backend, es decir el Frontend utilizará las consultas que necesite a la base de datos.

El **Frontend** usa tecnologías como **Vue.js,** el cual es la estructura de tu frontend y el que lo hará funcionar. Para manejar mejor la interfaz he usado también **vuetify,** el cual es una extensión de vue y permite realizar más acciones con las interfaces y que estas no sean tan simples.

Por último, para transformar a aplicación de escritorio y que sea más sencilla de usar empleé **electrón.**

3.Aplicación

En este apartado muestro una guía de cómo lanzar el programa y el uso básico de la aplicación. Una vez hecho esto explicaré parte por parte cómo funciona la propia aplicación a nivel interno.

3.1-Guía de uso y despliegue:

Para comenzar con el primer paso deberemos instalar “neo4J”, y así poder usar nuestra base de datos. Se puede descargar fácilmente desde

https://neo4j.com/download/.

Una vez instalado lo abrimos y exactamente donde pone databases pinchamos para crear una nueva.

Después le damos a CREATE A LOCAL DATABASE, pones un nombre a tu elección y una contraseña que debe ser 1236. Es importante que la versión que usemos sea la 3.5.23 ya que puede haber incompatibilidades con las demás.

Una vez hecho esto ascendemos y a la derecha, en la base de datos, pinchamos manage. Una vez aquí vamos a open folder y justo dentro damos a import. En import se introduce la base de datos que se encuentra dentro de la carpeta databases.

Realizado esto le damos a start database, luego volvemos a la pantalla de inicio y desde inicio vemos nuestra base de datos activa. Lo que haremos es pulsar directamente en open, lo cual nos llevara al browser de neo4J.

Para finalizar la parte de neo4J metemos esta sentencia en Cypher:

```
LOAD CSV WITH HEADERS FROM 'file:///movies.csv' AS line

WITH line LIMIT 1000
```

```
CREATE (m:Movie { Title: line.Title, year: toInteger(line.Year), director: line.Director, score: line.Score, genre: line.Genre })

FOREACH (n IN (CASE WHEN line.Director IS NULL THEN [] ELSE [1] END) |

MERGE (d:Director {name: line.Director})

CREATE (d)-[:DIRECTED]->(m)

)

FOREACH (n IN (CASE WHEN line.Genre IS NULL THEN [] ELSE [1] END) |

MERGE (g:Genre {name: line.Genre})

CREATE (m)-[:OF_GENRE]->(g)

)
```

Una vez hecho esto vamos a ejecutar el backend y frontend.

Para ello primero sacamos el **AppBackend** del paquete y lo ponemos en el escritorio, después ponemos “cd” en el buscador de Windows, le damos a símbolo del sistema y escribimos esto: “cd Desktop/AppBackend”.

Una vez estemos dentro descargamos e instalamos el node.js

Si no lo tuviéramos se instalaría desde https://nodejs.org/es/download/ , siempre hay que instalarlo según sus sistema operativo y versión.

Después de esto escribimos en la consola “npm install”, esperamos a que acabe y por último escribimos “node app.js”. Con esto ya debería quedar funcionando el backend.

Para el **AppFrontend** realizamos los mismos pasos que en el backend pero poniendo AppFrontend en vez de backend. Para iniciarlo en vez de poner “node app.js” pondremos: **“npm run electron:serve”.**

Con esto solo tenemos que esperar a que inicie la aplicación.

Una vez iniciada vemos en la ventana de la aplicación cuatro cajas. En la primera podrás elegir el director, la segunda te muestra el orden según opiniones de usuario, la tercera será para elegir el género y la cuarta para decidir cuántas películas quieres que muestre.

Debajo de estas cajas encontramos un hueco vacío donde podemos introducir el título de alguna película objeto de búsqueda. A la derecha se encuentran los dos botones principales: buscar y recomendaciones.

1. Buscar: realizará la búsqueda y serán mostradas todas las películas.
2. Recomendaciones: recomendará películas según tus gustos. (Recomendable realizar varias búsquedas para que el algoritmo aprenda).

3.2-Funcionamiento aplicación:

Para explicar el funcionamiento de aplicación comenzaré por partes:

1-Base de datos:

Para analizar la base de datos nos fijaremos en el código de Cypher.

```
LOAD CSV WITH HEADERS FROM 'file:///movies.csv' AS line
WITH line LIMIT 1000
CREATE (m:Movie { Title: line.Title, year: toInteger(line.Year), director: line.Director, score: line.Score, genre:
line.Genre })
FOREACH (n IN (CASE WHEN line.Director IS NULL THEN [] ELSE [1] END) |
MERGE (d:Director {name: line.Director})
CREATE (d)-[:DIRECTED]→(m)
)
FOREACH (n IN (CASE WHEN line.Genre IS NULL THEN [] ELSE [1] END) |
MERGE (g:Genre {name: line.Genre})
CREATE (m)-[:OF_GENRE]→(g)
)
```

Primero vemos que este código carga 1000 líneas. No necesito meter más porque como mucho llego a cargar 100 películas. Después creo los atributos que va a tener la base de datos: Title, director, Score y Genre.

Las siguientes líneas son las que sirven para convertir esta base de datos en un grafo, conectando sus nodos. Las relaciones que este grafo va a tener para ello son los nodos de la película con el género y los nodos de película con el director. Además de esto en esas mismas líneas se controla que no sean nulos, no vaya a ser que en la base faltase algún dato.

Todo esto da lugar a:

Una vez realizado este paso pasamos a la conexión del Backend con la base de datos.

2-Backend:

El backend será el que conectará con la base de datos para obtener los datos necesarios. Para ello dentro de esta carpeta tendremos un ejecutable llamado app.js que será el que contenga todo el código necesario.

Para la conexión tenemos estas líneas:

```
const driver = neo4j.driver(
  "bolt://localhost:7687",
  neo4j.auth.basic("neo4j", "1236")
);
const session = driver.session();
app.use(cors());
app.use(bodyParser.json());
app.use(bodyParser.urlencoded({ extended: true }));
```

Sirven únicamente para la conexión con la base de datos.

Después comenzaremos con el código que usaremos para realizar las peticiones a la base de datos:

```
app.post("/getMovies", function(req, res) {
  var director = req.body.director;
  var search = req.body.busqueda;
  var score = req.body.score;
  var limit = req.body.limit;
  var genre = req.body.genre;

 var result = [];
  if (genre != "Cualquiera") {
    query = "MATCH(m: Movie) -[* 1] - (: Genre { name: '" + genre + "' })";
  } else {
    query = "MATCH (m: Movie)";
  }
  if (director != "Cualquiera") {
    query = query.concat("WHERE m.director = '" + director + "'");
  }

  if (search != "" && director != "Cualquiera") {
    query = query.concat(" AND (m.Title =~ '(?i).*" + search + ".*') ");
  } else if (search != "") {
    query = query.concat(" WHERE (m.Title =~ '(?i).*" + search + ".*') ");
  }

    if (score) {
    query = query.concat(
      "RETURN m AS movie ORDER BY m." +
        score +
        " DESC, m.year DESC LIMIT " +
        limit
    );
}
```

Aquí se realizan consultas en función de lo que se halla marcado en la interfaz de la aplicación y con ello envía una consulta en lenguaje Cypher y extraería los datos solicitados.

Para acabar hay que ejecutar una sentencia para que se cierre la conexión con la base de datos cada vez que no se necesite.

```
  const resultPromise = session.run(query).subscribe({
    onNext: function(record) {
      var element = record.get("movie").properties;

      if (element.year == undefined) {
        element.year = { low: "--" };
        result.push(element);
      } else {
        result.push(element);
      }
    },
    onCompleted: function() {
      if (result.length < 1) result = [{ message: "Error" }];
      res.send(result);
      // session.close();
    },
    onError: function(error) {
      console.log(error);
    }
  });
});
```

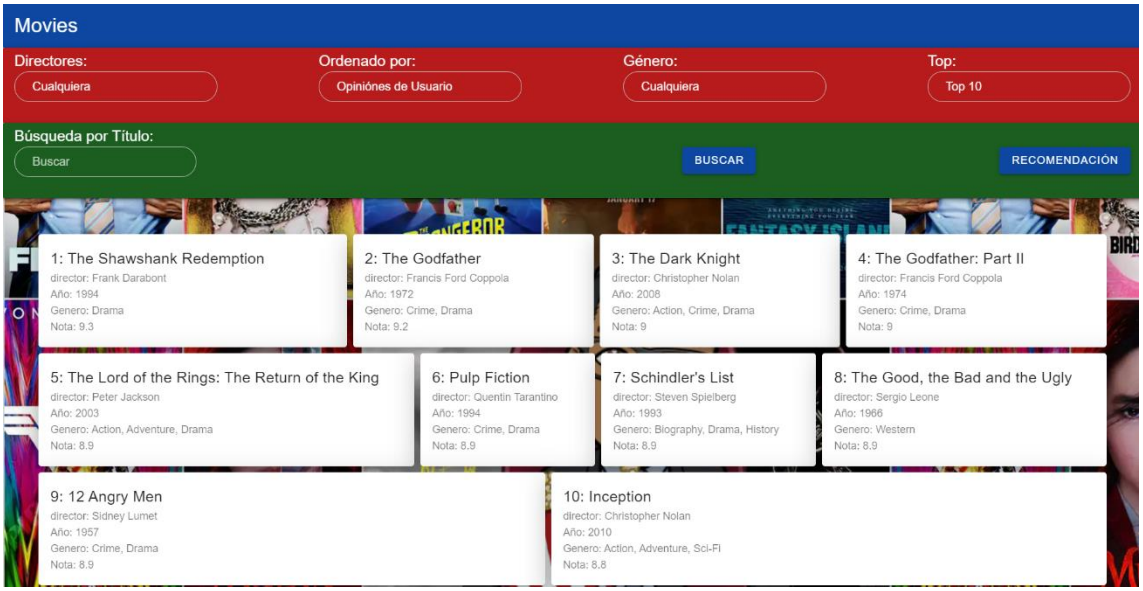
Estas líneas harán que con la función **onNext**, cada vez que el frontend necesite almacene esa petición y obtenga las propiedades de la película elegida. **OnCompleted**, una vez mandados todos los datos, realizará la petición conjunta y **OnError** manejará errores en caso de existir alguno.

Una vez funcionando esta parte iríamos con el frontend, el cual como ya hemos dicho será la interfaz y el grueso de la aplicación.

3-Frontend:

El archivo más importante del frontend será el archivo “home”, contenido en la carpeta “src” y dentro de ella en “views”. Aquí se localizará la funcionalidad, la interfaz y el contacto con el backend.

En las primeras líneas del frontend estará el diseño puro de la interfaz que el usuario podrá ver.



Así es como quedó la interfaz final de la aplicación.

Me decidí por este colorido diseño porque quería una interfaz llamativa excepto para las películas resultantes de las búsquedas. En esta última situación no apliqué color para que pudieran verse de una manera más clara a un simple golpe de vista.

En cada color hay una barra, las cuales rellené con cajas de combinación. que sirven para que puedas elegir distintas opciones, ya sea directores o géneros. Hay una excepción y esa es opiniones de usuario, ya que preferí recalcar que siempre las búsquedas estén ordenadas por notas de usuarios.

Si nos situamos más abajo nos encontramos los botones de buscar y el de recomendaciones, que es el que realizará el algoritmo de recomendaciones.

Por último, las películas aparecerán con sus características más notables: año, género, nota y título.

Para que esta interfaz se vea así realicé estas líneas.

```
<v-card color="blue darken-4" height="100px">
  <v-toolbar height="100px" color="red darken-4" dark>
    <v-toolbar-title>
      Directores:
      <v-combobox :items="directors" v-model="director" outlined dense rounded></v-combobox>
    </v-toolbar-title>
    <v-spacer></v-spacer>
    <v-toolbar-title>
      Ordenado por:
      <v-combobox :items="scores" v-model="score" outlined dense rounded></v-combobox>
    </v-toolbar-title>
    <v-spacer></v-spacer>
    <v-toolbar-title>
      Género:
      <v-combobox :items="genres" v-model="genre" outlined dense rounded></v-combobox>
    </v-toolbar-title>
    <v-spacer></v-spacer>
    <v-toolbar-title>
      Top:
      <v-combobox :items="limits" v-model="limit" outlined dense rounded></v-combobox>
    </v-toolbar-title>
  </v-toolbar>
</v-card>
```

Estas líneas crean las cajas para elegir los diferentes directores y géneros, límite de películas que aparezcan que querremos establecer, además de añadir la ordenación por nota.

Estas otras sirven para crear los recuadros que contendrán las películas con sus propiedades.

```
<v-content>
  <v-container grid-list-md text-xs-center fluid pa-12>
    <v-layout row wrap fill-height fill-width>
      <v-flex v-for="(movie, index) in movies" v-bind:key="movie.id">
        <v-card
          elevation="18"
          style="background: #3A1C71;
background: blue lighten-1;
background: blue lighten-1;"
        >
          <v-card-title>{{index + 1}}: {{movie.Title}}</v-card-title>
          <v-card-subtitle>
            director: {{movie.director}}
            <br />
            Año: {{movie.year.low}}
            <br />
            Genero: {{movie.genre}}
            <br />
            Nota: {{movie.score}}
          </v-card-subtitle>
        </v-card>
      </v-flex>
    </v-layout>
  </v-container>
</v-content>
```

En estas líneas es importante ver que se están haciendo llamadas al backend. Por ejemplo, “movies.director” está pasando la película junto con el director asociado a esa película y así con todos.

Ahora iremos con lo que va a contener esas cajas de combinaciones, ya que todas funcionan igual.

Pondré un ejemplo con los directores:

```
directors: [
  "Cualquiera",
  "David Fincher",
  "Steven Spielberg",
  "James Cameron",
  "Christopher Nolan",
  "Quentin Tarantino",
  "Robert Zemeckis",
  "Peter Jackson",
  "Frank Darabont",
  "Lana Wachowski",
  "Francis Ford Coppola",
  "Ridley Scott",
  "Joss Whedon",
  "George Lucas",
  "Jonathan Demme",
  "Martin Scorsese",
  "Sam Mendes",
  "Irvin Kershner",
  "Gore Verbinski",
  "James McTeigue",
  "Tony Kaye",
  "James Gunn",
  "Luc Besson",
  "Bryan Singer",
  "Andrew Stanton",
  "Mel Gibson"
],
```

Como podemos ver aquí están todos los directores que hay en la base de datos y el usuario podrá escoger en la caja de combinaciones de los directores.

Lo mismo se realizaría con los géneros y con la caja de “Top” que servirá para decirle a la aplicación cuántas películas queremos que muestre como máximo.

Después de esto pasaremos a los métodos que contendrán las acciones de los dos botones que tenemos: buscar y recomendar.

Lo que hace buscar es simplemente una llamada al backend para que extraiga de la base de datos todo lo que en la interfaz se ha seleccionado.

Aquí quiero recalcar la sentencia usada para dicha acción, la cual además maneja errores, es decir si tú introduces una búsqueda que no se puede realizar devolverá un mensaje de error y si es correcta publicará la respuesta.

```
this.$http.post("http://localhost:3000/getMovies", params).then(
  response => {
    if (
      response.body &&
      response.body.length &&
      response.body[0].message != "Error"
    ) {
      this.movies = response.body;
    } else {
      response.body = [
        {
          Title:
            "¡vaya, ninguna película cumple con lo que estás buscando!",
          year: "--"
        }
      ];
      this.movies = response.body;
    }
  },
  response => {
    alert("Ha habido un error en el envío: " + response.body);
  }
);
```

El método de recomendar hace algo similar, pero usando un algoritmo, el cual explicaré en el siguiente apartado.

Con esto quedaría finalizado todos los puntos del funcionamiento de mi aplicación

4.Algoritmo de recomendación

Este algoritmo es utilizado cuando el usuario pulsa el botón recomendación.

Su función es recomendarte una película utilizando para ello búsquedas previas que hayas hecho en la aplicación.

Para ello realicé una parte del algoritmo la cual irá sumando 1 cada vez que se utilice un mismo género o un mismo director, esto hará que se obtenga un valor según la cantidad de veces que selecciones la misma variable.

Ejemplo: si has seleccionado un mismo director 3 veces tu número será 3 y si se compara a otros directores que hayas buscado solo 1 vez, el ganador será 3, todo ello, por supuesto, dentro de un bucle para que se repita este proceso.

```
for (var i = 0; i < this.historial.length; i++) {
  var cDirector = 1;
  var cScore = 1;
  var cGenre = 1;

  for (var j = i; j < this.historial.length; j++) {
    if (this.historial[i].director == this.historial[j].director) {
      cDirector++;
    }
    if (mFDirector < cDirector) {
      mFDirector = cDirector;
      fDirector = this.historial[i].director;
    }

    if (this.historial[i].score == this.historial[j].score) {
      cScore++;
    }
    if (mFScore < cScore) {
      mFScore = cScore;
      fScore = this.historial[i].score;
    }

    if (this.historial[i].genre == this.historial[j].genre) {
      cGenre++;
    }
    if (mFGenre < cGenre) {
      mFGenre = cGenre;
      fGenre = this.historial[i].genre;
    }

  }
}
```

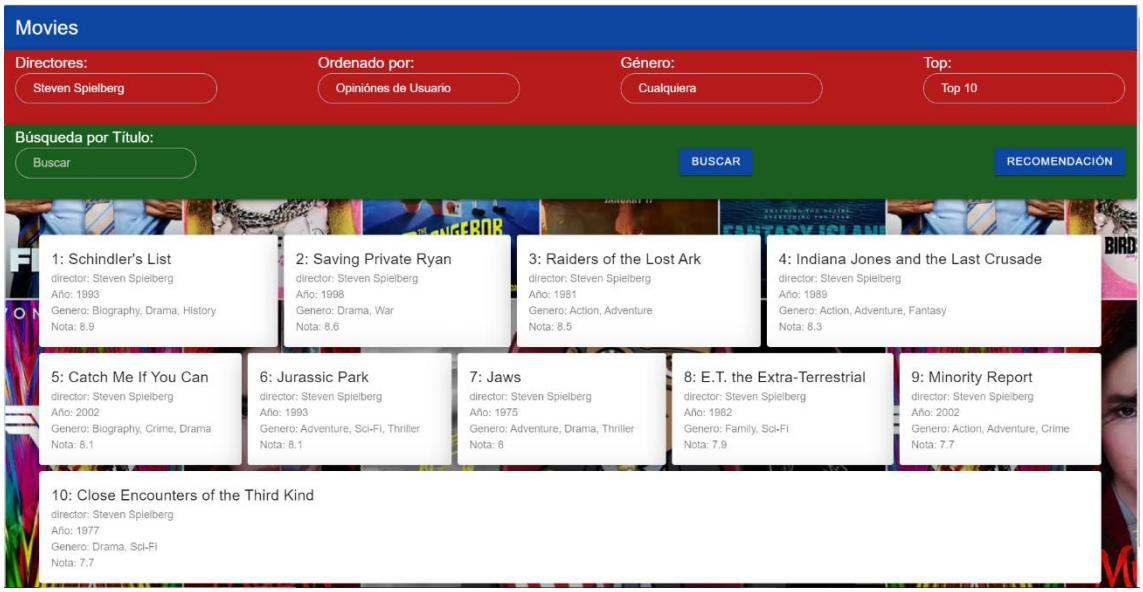
En caso de no haber realizado búsquedas saldrá una película aleatoria. Por ello es necesario realizar búsquedas para que el algoritmo detecte tus gustos.

En el caso de que hayas realizados búsquedas anteriores y estas no hayan generado resultados saldrá un mensaje de error que te dirá que realices más búsquedas para el correcto funcionamiento del algoritmo.

5.Análisis de resultados

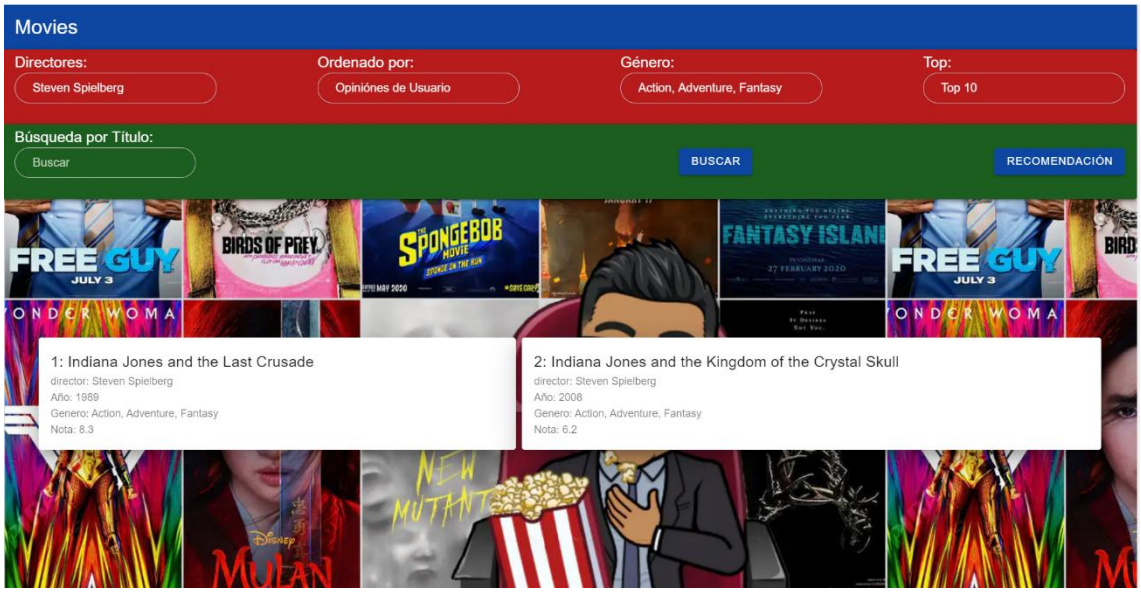
En este punto expondré varias búsquedas realizadas en el programa para observar su funcionamiento.

Primero vamos a probar a buscar las 10 mejores películas del director Steven Spielberg



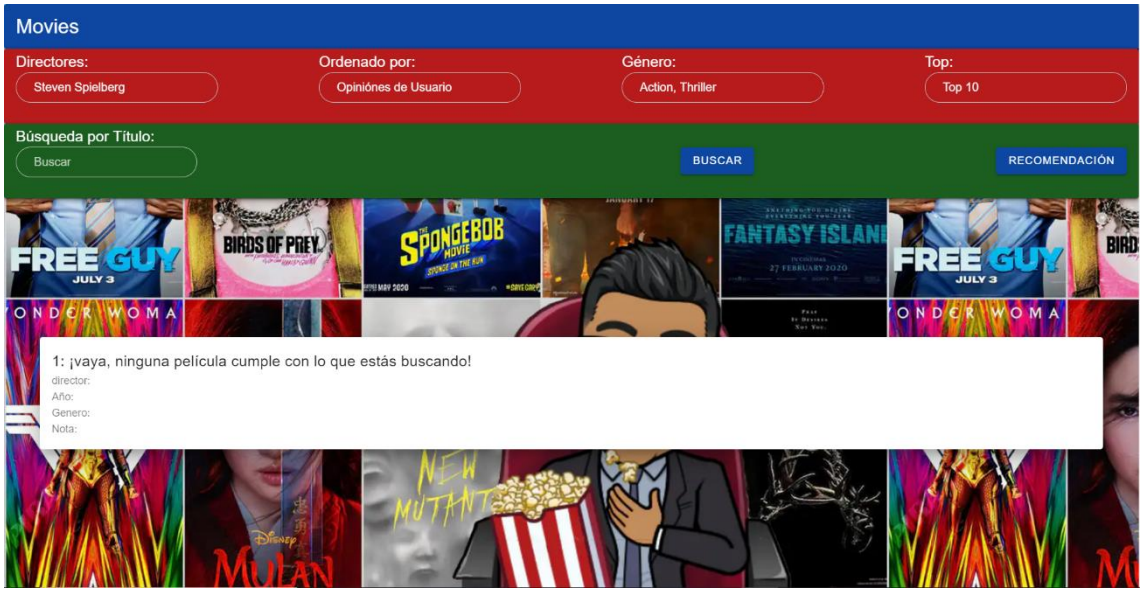
Como vemos ordena correctamente por nota las 10 mejores películas de Spielberg.

Ahora vamos a filtrarlas por género. Probamos con action, adventure, fantasy:

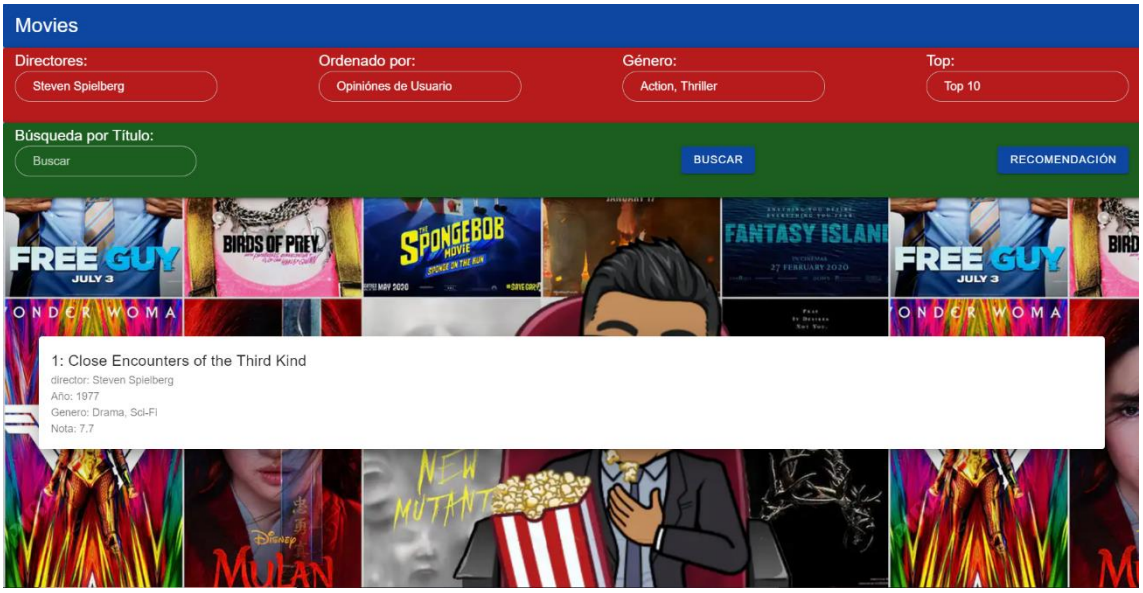


Obtenemos las dos películas de Steven Spielberg con esos géneros.

En caso de no existir una película de este director con esos géneros pasaría esto:

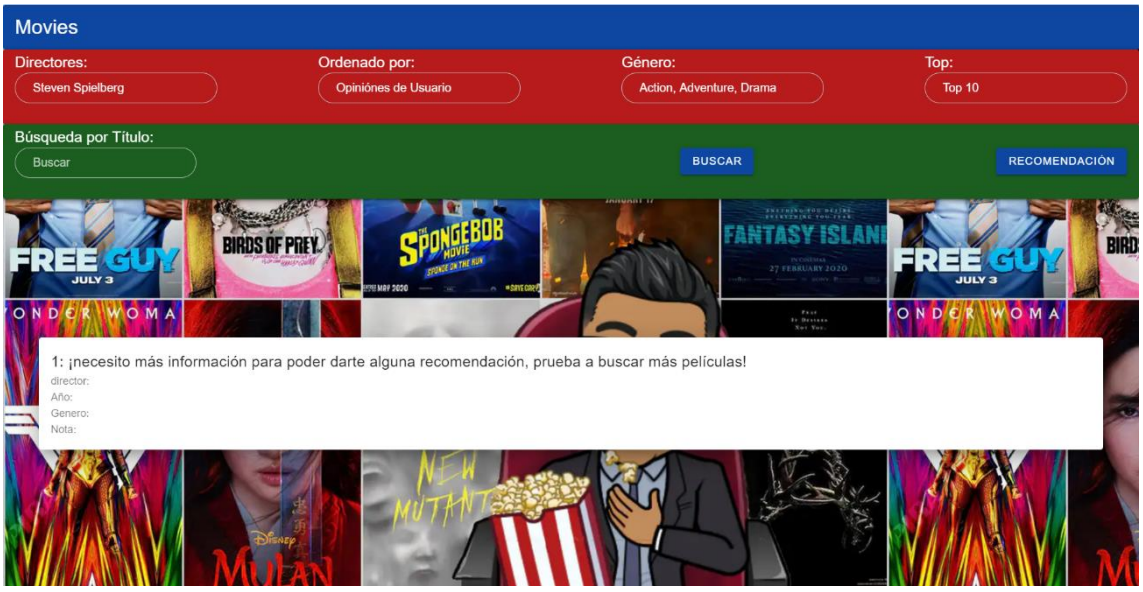


Una vez que ya hemos realizado unas búsquedas podemos probar el botón de recomendación.



Observamos que como solo hemos buscado películas de Spielberg nos recomienda una película suya.

Por último, en el caso de que hayamos realizado varias búsquedas inconclusas y después de ello usemos el botón recomendación aparecerá lo siguiente:



6.DAFO

Análisis crítico de ventajas y limitaciones del trabajo llevado a cabo.

1. **Debilidades:** Sus dos mayores debilidades serían: la imposibilidad de encontrar una base de datos en español, y por lo tanto los géneros y las películas aparecen en inglés, y que el algoritmo podría mejorarse para que las recomendaciones sean más exactas.
2. **Amenazas:** La mayor amenaza de este trabajo ha sido el problema de encontrar una base de datos adecuada para la aplicación que estaba diseñando, ya que la mayoría presentaban problemas que chocaban con ella.
3. **Fortalezas:** La mayor fortaleza de mi aplicación es su efectividad, ya que cumple perfectamente la función para la que ha sido diseñado
4. **Oportunidades:** El poder observar trabajos de otros años me ha sido de gran ayuda, ya que me inspiró y me dio muchas ideas para diseñar mi proyecto a través de la observación de sus fallos y aciertos.

7.Lineas de futuro

El cine siempre va a ser uno de los medios de entretenimiento más importantes del mundo. Está catalogado como un arte el cual lleva existiendo más de 100 años y por los números que maneja todo hace pensar que seguirá creciendo.

#	Película	Distribuidora(s)	Taquilla (Estados Unidos)	Taquilla (fuera de EE UU)	Recaudación mundial	Presupuesto	Año de estreno	Director(es)
1	*Avengers: Endgame*	Marvel Studios/Walt Disney Pictures	$858 373 000 (41,8 %)	$1 939 427 564 (58,2 %)	**$2 797 800 564**	$356 000 000	2019	Anthony y Joe Russo
2	*Avatar*	20th Century Fox	$760 507 625 (27,3 %)	$2 029 172 169 (72,7 %)	**$2 787 965 087**	$237 000 000	2009	James Cameron
3	*Titanic*	20 Century Fox / Paramount Pictures	$659 363 944 (30,1 %)	$1 528 100 000 (69,9 %)	**$2 187 463 944**	$200 000 000	1997	James Cameron
4	*Star Wars: Episodio VII - El despertar de la Fuerza*	Walt Disney Pictures	$936 662 225 (45,3 %)	$1 131 561 399 (54,7 %)	**$2 068 223 624**	$245 000 000	2015	J. J. Abrams
5	*Avengers: Infinity War*	Marvel Studios/Walt Disney Pictures	$678 815 482 (33,1 %)	$1 369 544 272 (66,9 %)	**$2 048 359 754**	$356 000 000	2018	Anthony y Joe Russo
6	*Jurassic World*	Universal Pictures	$652 270 625 (39 %)	$1 019 442 583 (61 %)	**$1 671 713 208**	$150 000 000	2015	Colin Trevorrow
7	*El rey león*	Walt Disney Pictures	$543 638 043 (32,8 %)	$1 113 305 351 (67,2 %)	**$1 656 943 394**	$260 000 000	2019	Jon Favreau
8	*The Avengers*	Marvel Studios/Walt Disney Pictures	$623 357 910 (41 %)	$895 455 078 (59 %)	**$1 518 812 988**	$220 000 000	2012	Joss Whedon
9	*Furious 7*	Universal Studios	$353 007 020 (23,3 %)	$1 163 038 891 (76,7 %)	**$1 516 045 911**	$190 000 000	2015	James Wan

Este es un ranking de las películas más taquilleras de la historia. Cómo podemos ver el top 1 es de apenas el año 2019, lo que lleva a pensar que es algo muy presente en el ocio de las personas, esto sin hablar de la cantidad millonaria de recaudación que generan. Todo parece indicar que el cine va a seguir siempre ahí y seguirá creciendo.

Respecto a las distintas aplicaciones que recomienden películas se podría pensar que al ser una industria tan longeva podríamos encontrar un amplio número, pero la realidad es que no es así, asunto que pude comprobar cuando buscaba con Google. ¿Y esto por qué?, me pregunté en su momento a lo que respondiéndome a mí mismo y a los demás explico que se debe a que la mayoría de las personas que buscan películas se dirigen a páginas como Filmaffinity o IMDB. El problema viene cuando reparas que estas páginas no recomiendan según tus gustos, si no únicamente ofrecen búsquedas según género o director, pero en ningún momento se te recomienda de manera personalizada.

Por lo tanto, yo creo que en un futuro cercano aplicaciones como la presentada en esta asignatura comenzarán a tenerse en cuenta debido a que el consumidor de cine es cada vez más exigente y valorará positivamente las recomendaciones personalizadas que mi programa ofrece.

Respecto al futuro, lo que haría sería implementar mi aplicación en páginas como IMDB. Además, añadiría la posibilidad de buscar y recomendar series, lo que la haría más completa, dado que es bastante usual que el consumidor de películas lo sea de series también.

8. Lecciones aprendidas

Este proyecto me ha enseñado varias lecciones:

La primera es a atravesar esa barrera de desconfianza que he podido llegar a sentir antes de realizar un proyecto por mí mismo. Para ello he buscado la información que he considerado, ya que hasta ahora todos los programas que he realizado han sido bajo pautas específicas a seguir, nada tan personal.

La segunda es a abandonar el miedo a usar tecnologías desconocidas que, al principio, pueden generar un poco de recelo pero que, finalmente, caes en la cuenta de que gracias a ellas puedes realizar cosas diferentes y ahondar en un mayor conocimiento en la materia. Cabe destacar la gran utilidad de las mismas a futuro. En este caso estamos hablando de NEO4J, una tecnología que es muy probable que prospere favorablemente y de la que yo a través de este trabajo dispondré de conocimiento.

La tercera y última es a trabajar de manera organizada e intensiva, ya que tuve que empezar el proyecto algo más tarde de lo que hubiera deseado por falta de tiempo y me vi obligado a realizar una rutina de trabajo intensivo diario, algo que considero muy importante a la hora de trabajar en cualquier empresa para cumplir exigentes plazos de entrega.

8.Bibliografía

1-Guía de Cypher elaborada por Andrea Fernández

2-Node.js: https://nodejs.org/es/

3-Express: https://expressjs.com/es/

4-NEO4J: https://neo4j.com

5-Driver NEO4J: https://neo4j.com/developer/language-guides/

6-Vue.js: https://vuejs.org/

7-Vuetify: https://vuetifyjs.com/en/

8-Electron: https://electronjs.org/

9-Kaggle: https://www.kaggle.com/datasets

10-Github: https://github.com/Jgarcg40/IAMovies.git

11-Wikipedia:
https://es.wikipedia.org/wiki/Anexo:Pel%C3%ADculas_con_las_mayores_recaudaciones