

# Base de Datos II (IC4302) – Semestre 2, 2023

---

## Proyecto 2

Jesús Andrés Cortés Álvarez – 2021579439

Aaron Ortiz Jimenez – 2022437529

David Suárez Acosta – 2020038304

Justin Acuña Barrantes - 2018093451

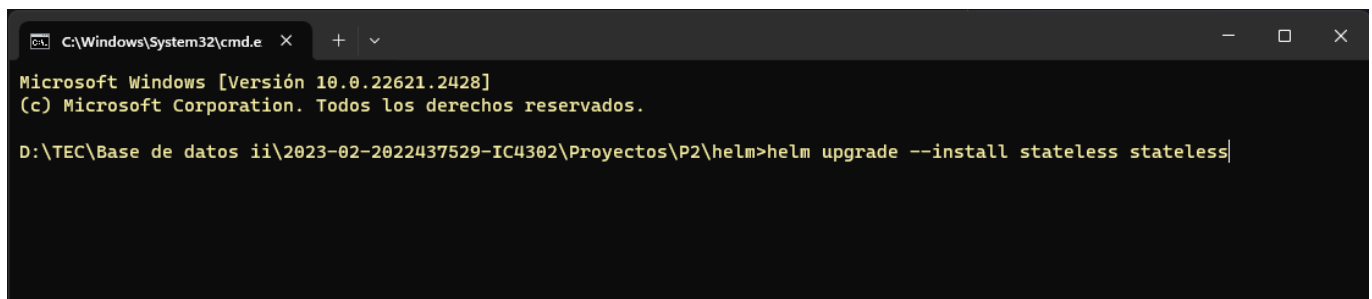
Joctan Porras Esquivel - 2021069671

---

## Ejecución del proyecto

En el siguiente proyecto se realizó un buscador de información sobre películas, para eso se emplearon dos bases de datos como los son MongoDB Atlas y Neo4j, la información empleada son dataset de pruebas que provee ambos servicios. Además, se empleó firebase como herramienta en el manejo de autenticación de los usuarios.

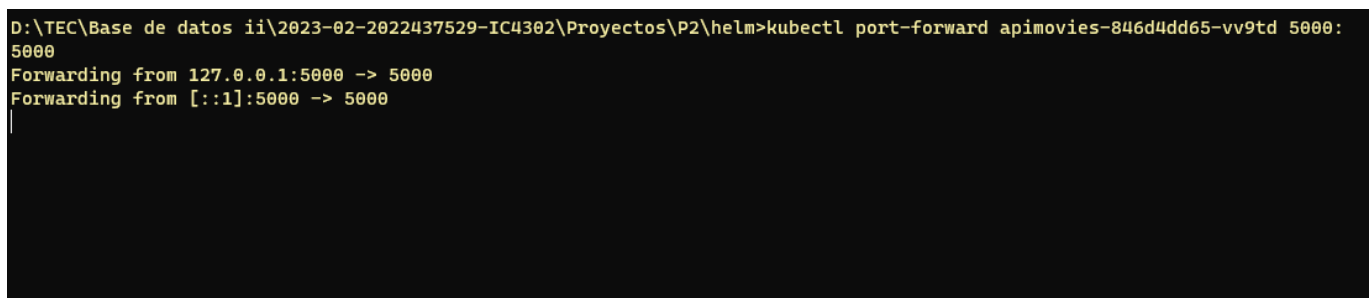
Como primer paso para ejecutar el proyecto se debe instalar el POD del API en kubernetes, para eso se abre una consola en la carpeta helm del proyecto y se ejecuta el comando presente en la siguiente imagen:



```
C:\Windows\System32\cmd.e X + v
Microsoft Windows [Versión 10.0.22621.2428]
(c) Microsoft Corporation. Todos los derechos reservados.

D:\TEC\Base de datos ii\2023-02-2022437529-IC4302\Proyectos\P2\helm>helm upgrade --install stateless stateless|
```

Luego, necesitamos exponer el puerto del pod con la máquina, para eso se ejecuta el siguiente comando en consola expuesta en la siguiente imagen, cabe aclarar de que no se debe cerrar la siguiente ventana para seguir exponiendo el puerto.

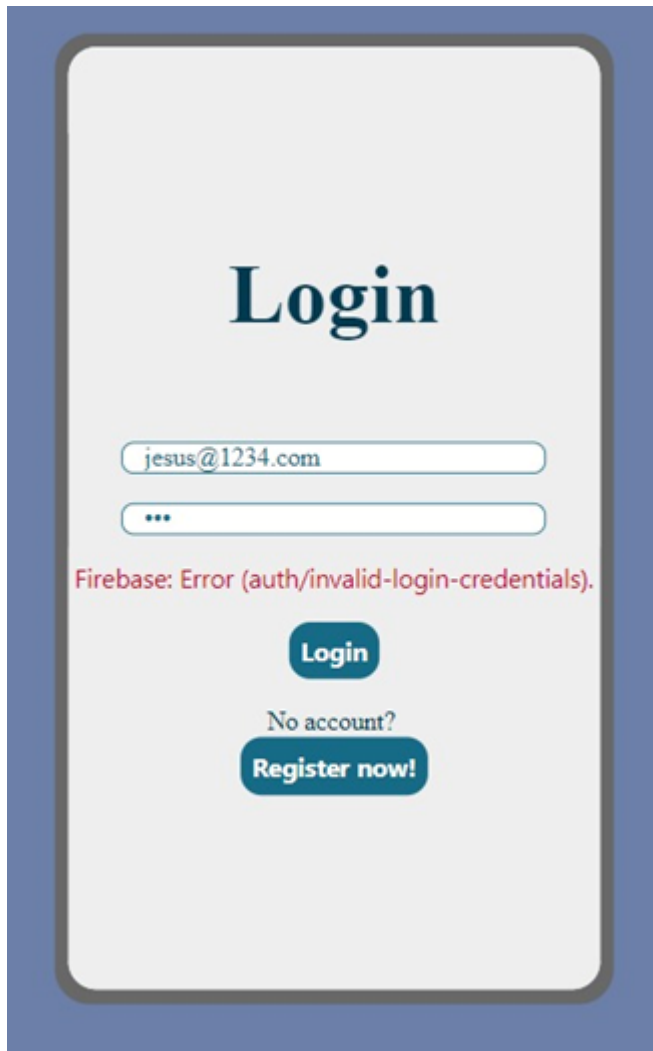


```
D:\TEC\Base de datos ii\2023-02-2022437529-IC4302\Proyectos\P2\helm>kubectl port-forward apimovies-846d4dd65-vv9td 5000:5000
Forwarding from 127.0.0.1:5000 -> 5000
Forwarding from [::1]:5000 -> 5000
```

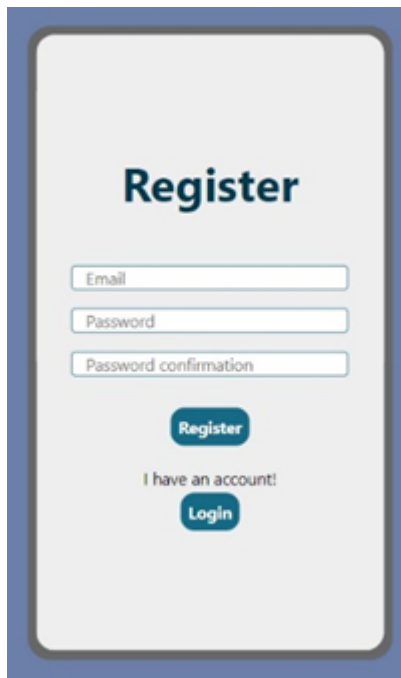
Para exponer el puerto a internet utilizaremos la aplicación ngrok, para tal efecto ejecutaremos el siguiente comando en consola desde la carpeta donde se encuentre el ejecutable ngrok.exe exponiendo el puerto del anterior paso. De igual forma se debe mantener la ventana abierta.

```
Microsoft Windows [Versión 10.0.22621.2428]  
(c) Microsoft Corporation. Todos los derechos reservados.  
  
C:\ngrok>ngrok http --domain=musical-kite-probably.ngrok-free.app 5000|
```

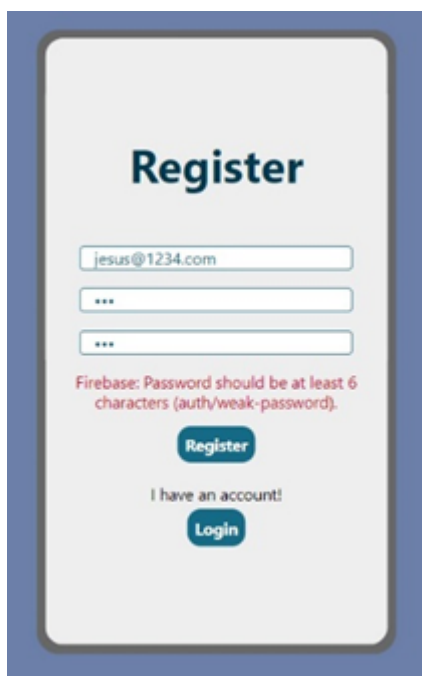
La interfaz fue creada con la plataforma Thunkable, para eso se crearon varias páginas para implementar las funcionalidades pedidas. Primero accedemos a la cuenta donde se creó el proyecto, cuando accedemos al sitio tenemos la página de Login donde se introduce el correo y la contraseña que este registrada. En caso de que el correo o contraseña no estén registradas, o no cumpla con el largo de 6 palabras en la contraseña o deje los espacios en blanco, mostrara un mensaje de error. Si los datos están correctos y presiona login, pasa a la siguiente página. Por otro lado, si se presiona: "Register now!" pasamos a la página de registro.



En la página de registro tenemos que introducir un correo no registrado, contraseña y re-digitar la contraseña para confirmar, al presionar el botón "Register" creamos un nuevo usuario con el que podamos acceder a la app si los datos son correctos, en el caso contrario de que no cumpla con el largo de 6 palabras en la contraseña o se deje un espacio en blanco, mostrara un mensaje de error.



A screenshot of a mobile application's registration screen. The screen has a light gray background with a dark blue border. At the top, the word "Register" is displayed in a bold, dark blue font. Below the title, there are three input fields: "Email", "Password", and "Password confirmation". Each field has a light gray border and a small blue icon on the right. Below the input fields, there is a dark blue button with the word "Register" in white. Underneath the button, the text "I have an account!" is displayed in a smaller, dark blue font. At the bottom, there is another dark blue button with the word "Login" in white.

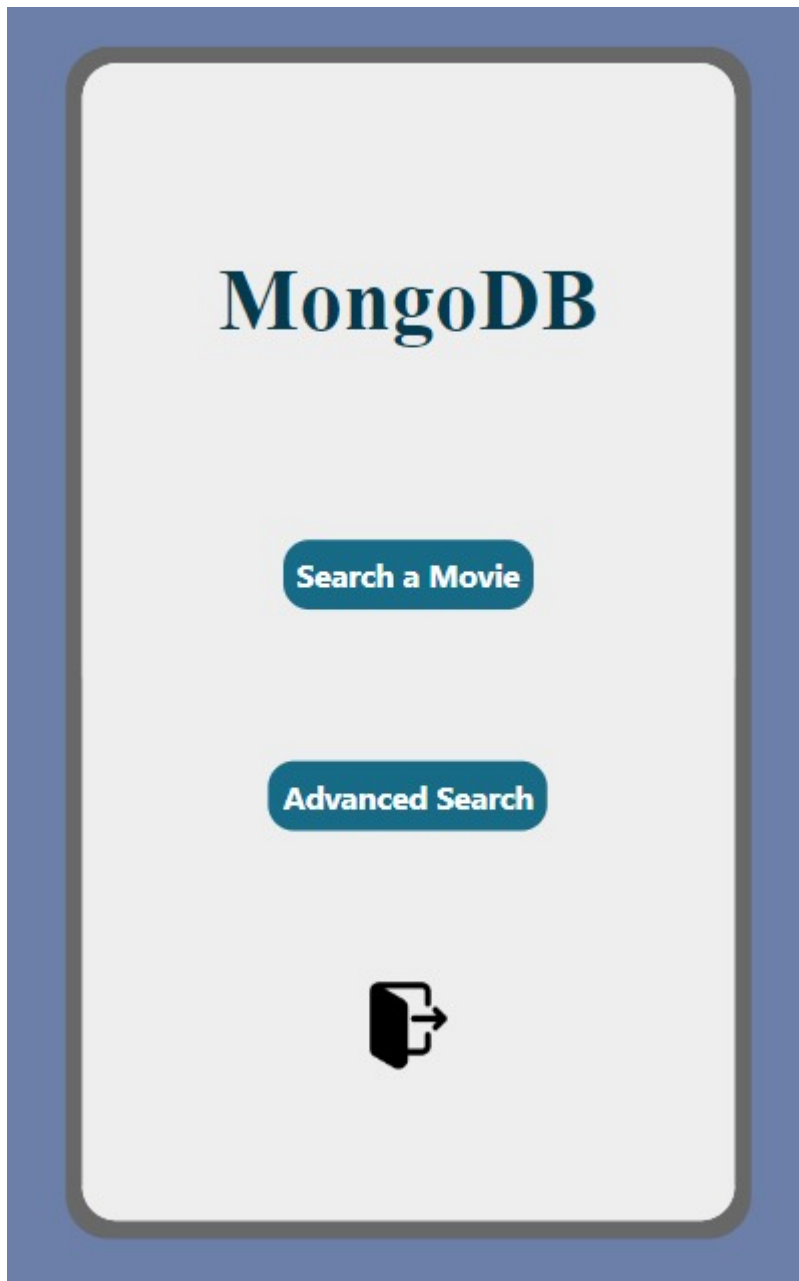


A screenshot of the same mobile application's registration screen, but with an error message. The "Email" field now contains the text "jesus@1234.com". The "Password" and "Password confirmation" fields are filled with three asterisks. Below the input fields, a red error message is displayed: "Firebase: Password should be at least 6 characters (auth/weak-password)." Below the error message, there is a dark blue button with the word "Register" in white. Underneath the button, the text "I have an account!" is displayed in a smaller, dark blue font. At the bottom, there is another dark blue button with the word "Login" in white.

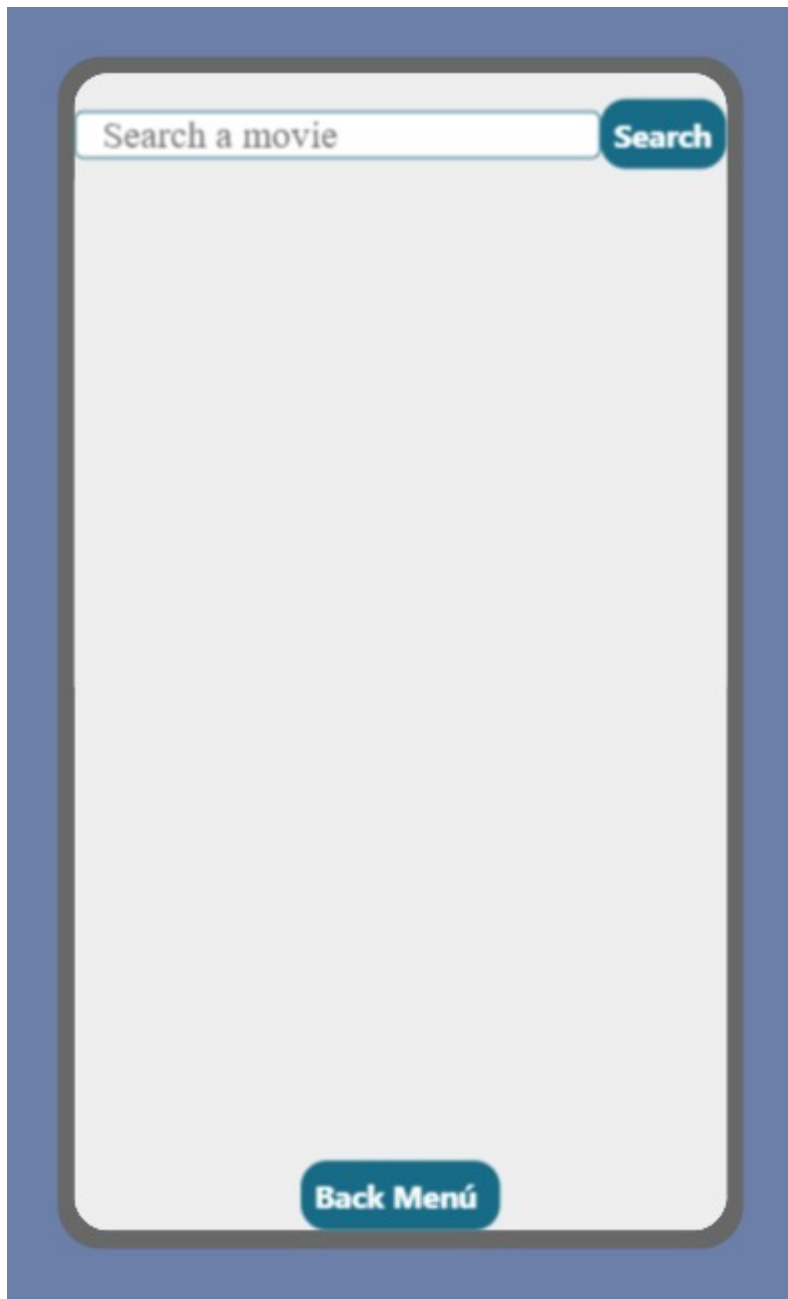
Luego de acceder al sistema, se muestran las dos bases de datos de las cuales puede seleccionar una para realizar la búsqueda, o cerrar sesión con el botón que se encuentra al final de la página.



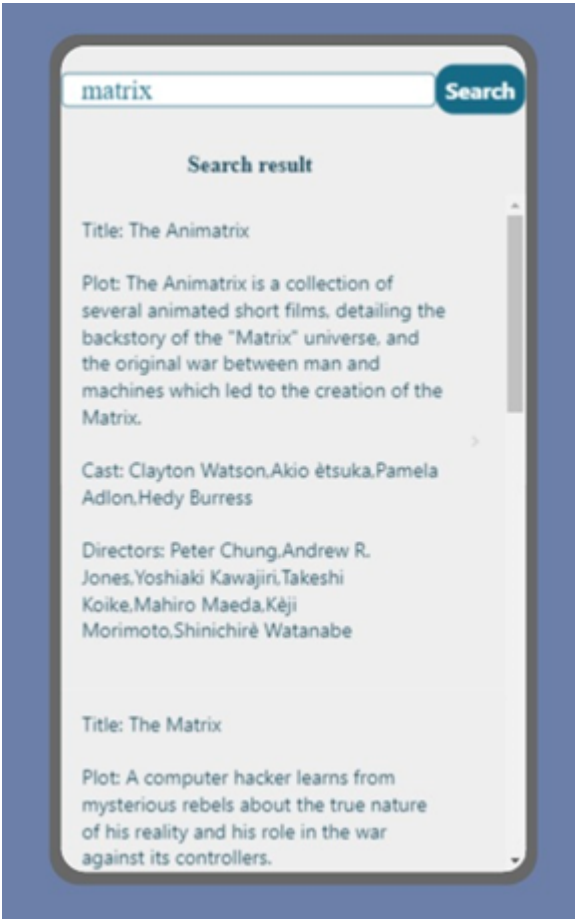
Luego de seleccionar una base de datos, se mostrará un menú en donde se podrá seleccionar búsqueda general al presionar el botón "Search a Movie", por otro lado, al presionar "Advanced Search" se mostrará la página para búsqueda avanzada.



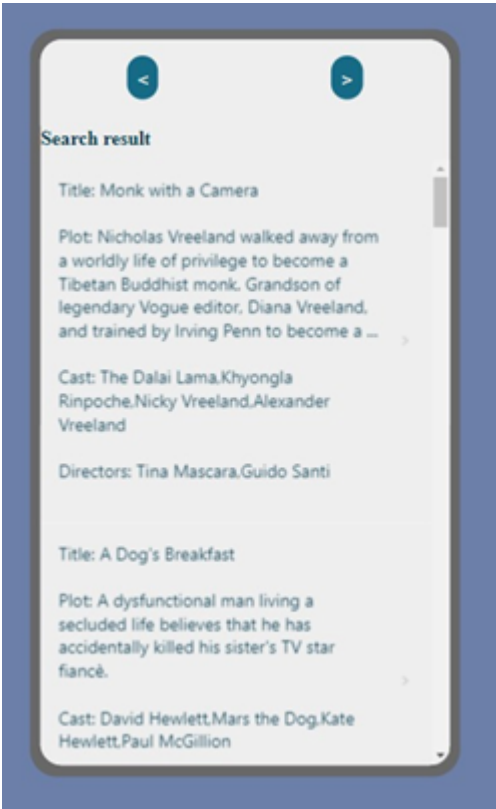
Al presionar "Search a Movie" muestra la siguiente pantalla donde tenemos un textbox donde podremos agregar un valor de búsqueda general: ya sea el título, nombre de un actor, director o plot. Si queremos volver a la página anterior se presiona "Back Menú".



Al introducir un valor de búsqueda y presionar el botón "search" se desplegarán los documentos con sus datos obtenidos de la búsqueda con el motor de base de datos empleado, al presionar el campo del documento se puede acceder a todos los datos.



En el caso de que la búsqueda devuelva muchos resultados, se habilitaran las flechas para mostrar los demás documentos.



Cuando se accede a un documento, se redirige a la página que se aprecia en la siguiente imagen, donde muestra todos los datos que contiene el documento.

## The Matrix

[Back](#)

Plot: A computer hacker learns from mysterious rebels about the true nature of his reality and his role in the war against its controllers.

Genres: Action,Sci-Fi

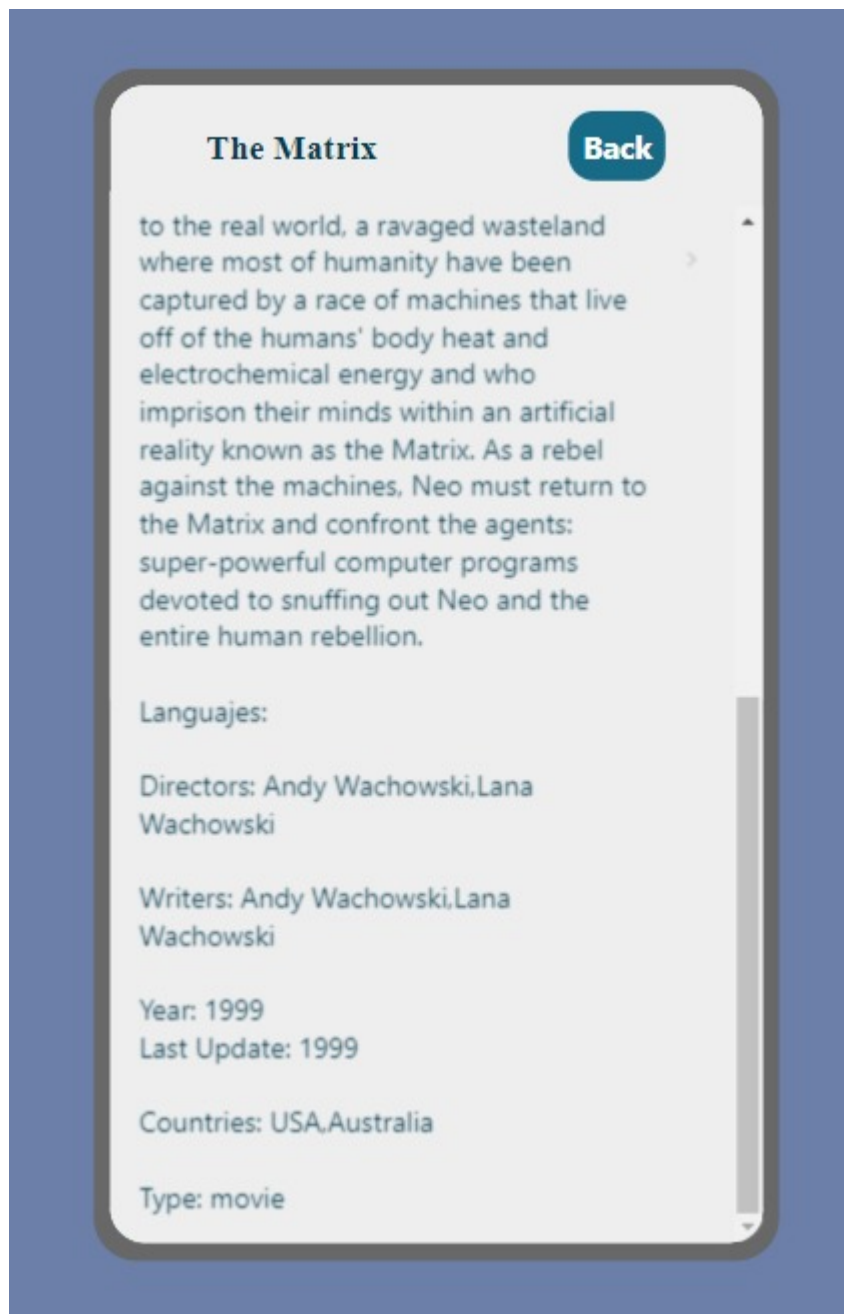
Runtime: 136

Cast: Keanu Reeves,Laurence Fishburne,Carrie-Anne Moss,Hugo Weaving

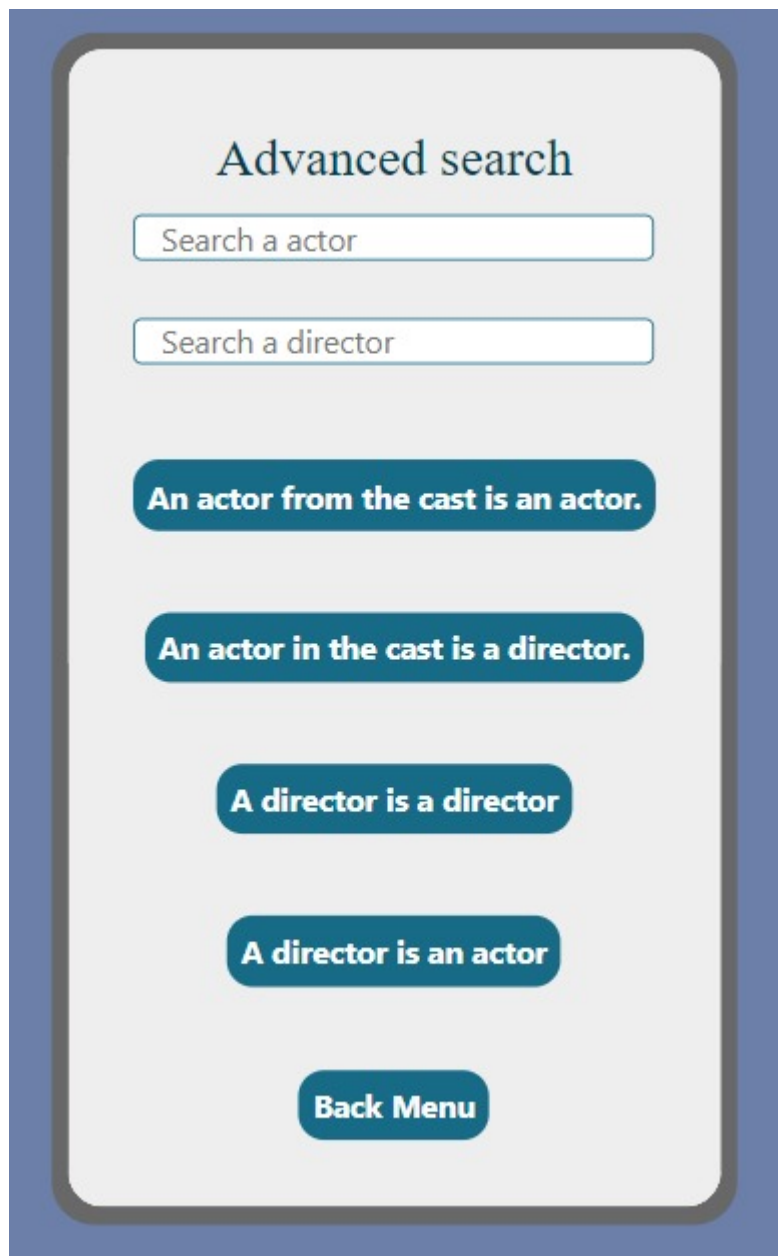
Full plot: Thomas A. Anderson is a man living two lives. By day he is an average computer programmer and by night a hacker known as Neo. Neo has always questioned his reality, but the truth is far beyond his imagination. Neo finds himself targeted by the police when he is contacted by Morpheus, a legendary computer hacker branded a terrorist by the government. Morpheus awakens Neo to the real world, a ravaged wasteland where most of humanity have been captured by a race of machines that live







Si se presiona el botón "Advanced Search" entramos a la página de advanced search donde tenemos dos campos de entrada uno para buscar por actor y otro para director, además se tienen las opciones para realizar la búsqueda de las películas donde el actor o director actúa o dirigen.



The image shows a mobile application interface for an 'Advanced search' menu. The menu is contained within a light gray rounded rectangle with a dark gray border, set against a blue background. At the top, the title 'Advanced search' is displayed in a dark blue serif font. Below the title are two search input fields: 'Search a actor' and 'Search a director', both with light blue borders. Following these are five dark blue buttons with white text, stacked vertically: 'An actor from the cast is an actor.', 'An actor in the cast is a director.', 'A director is a director', 'A director is an actor', and 'Back Menu'.

Advanced search

Search a actor

Search a director

An actor from the cast is an actor.

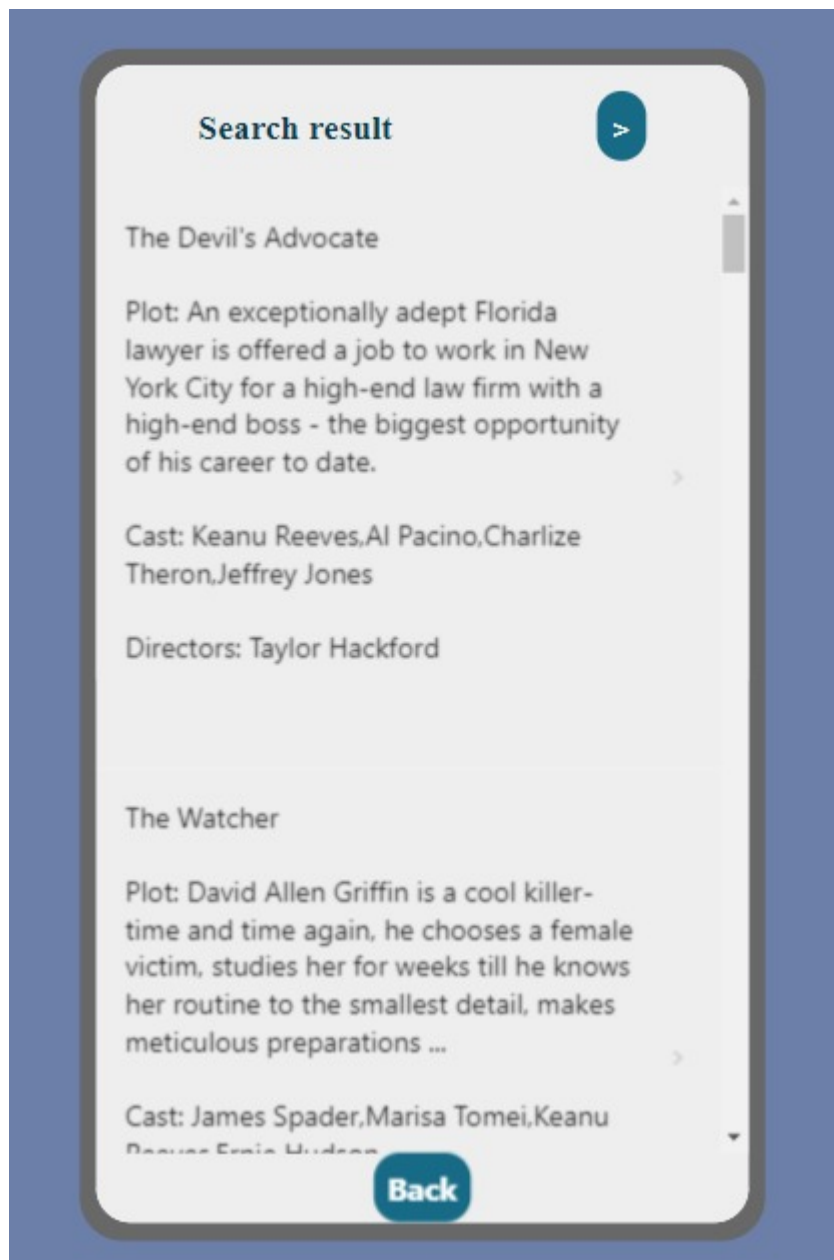
An actor in the cast is a director.

A director is a director

A director is an actor

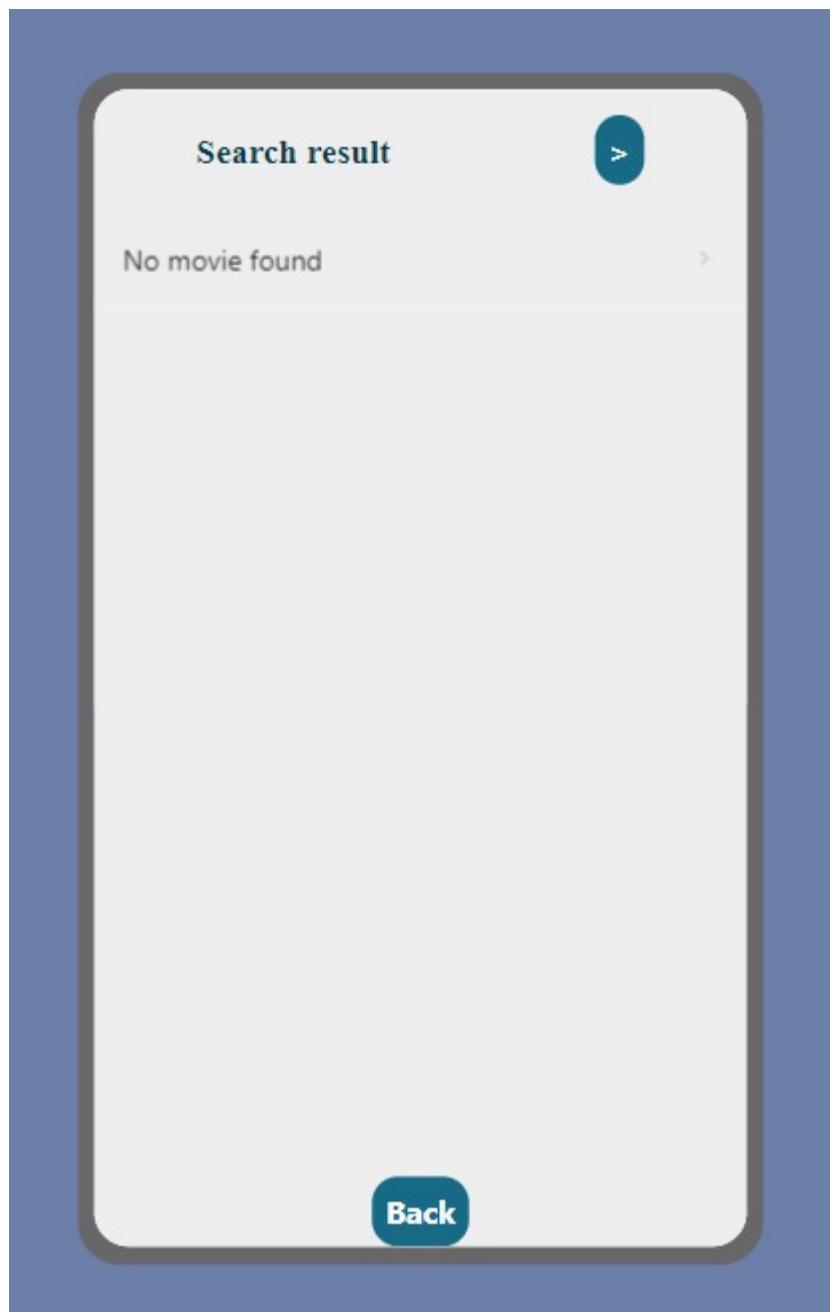
Back Menu

Si seleccionamos alguna de las opciones del menú de Advanced Search se nos mostrarán las películas resultantes con el criterio definido.



Si se realiza una búsqueda de un valor que no coincide con los datos almacenados en la base de datos, se mostrara un mensaje de error.

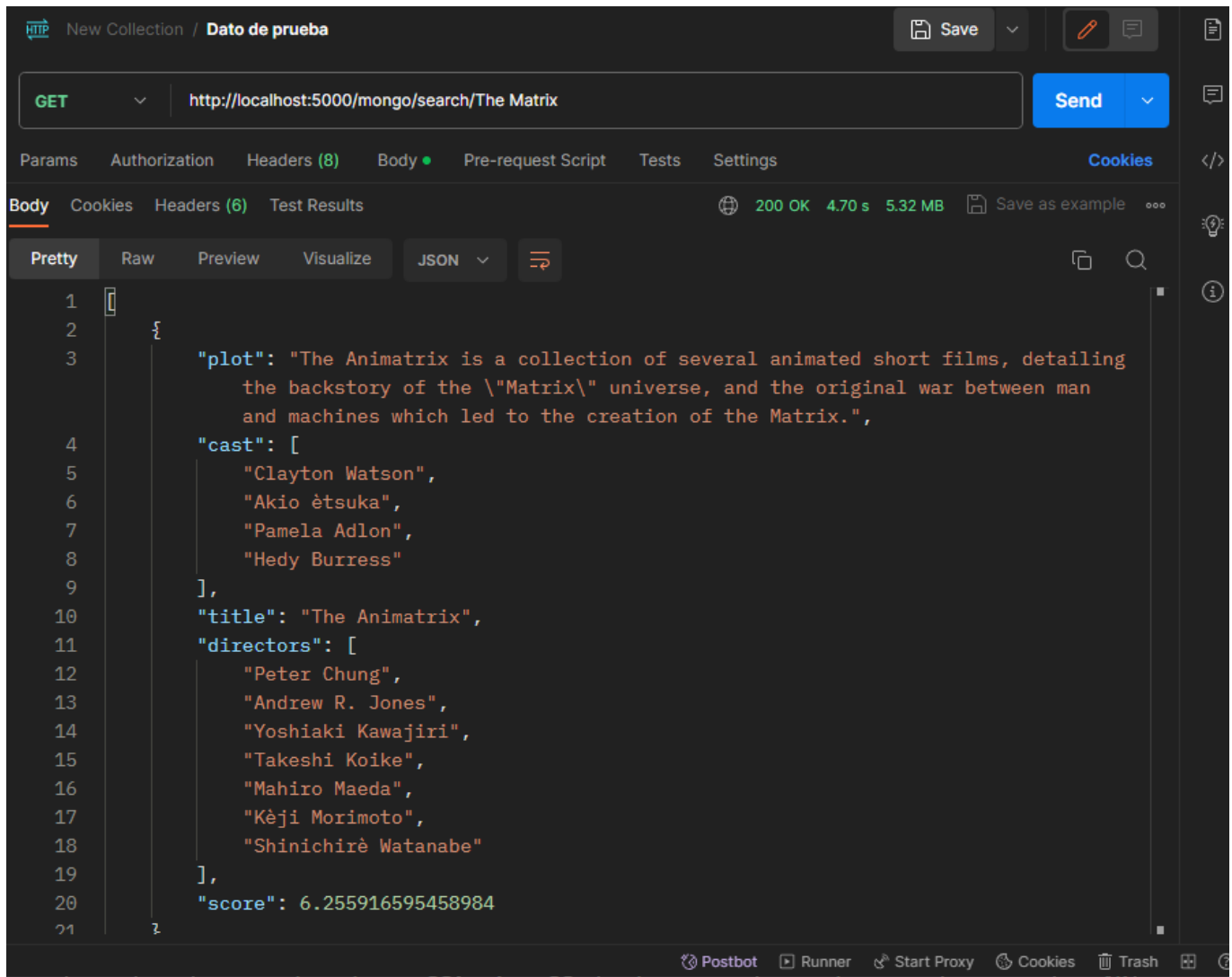




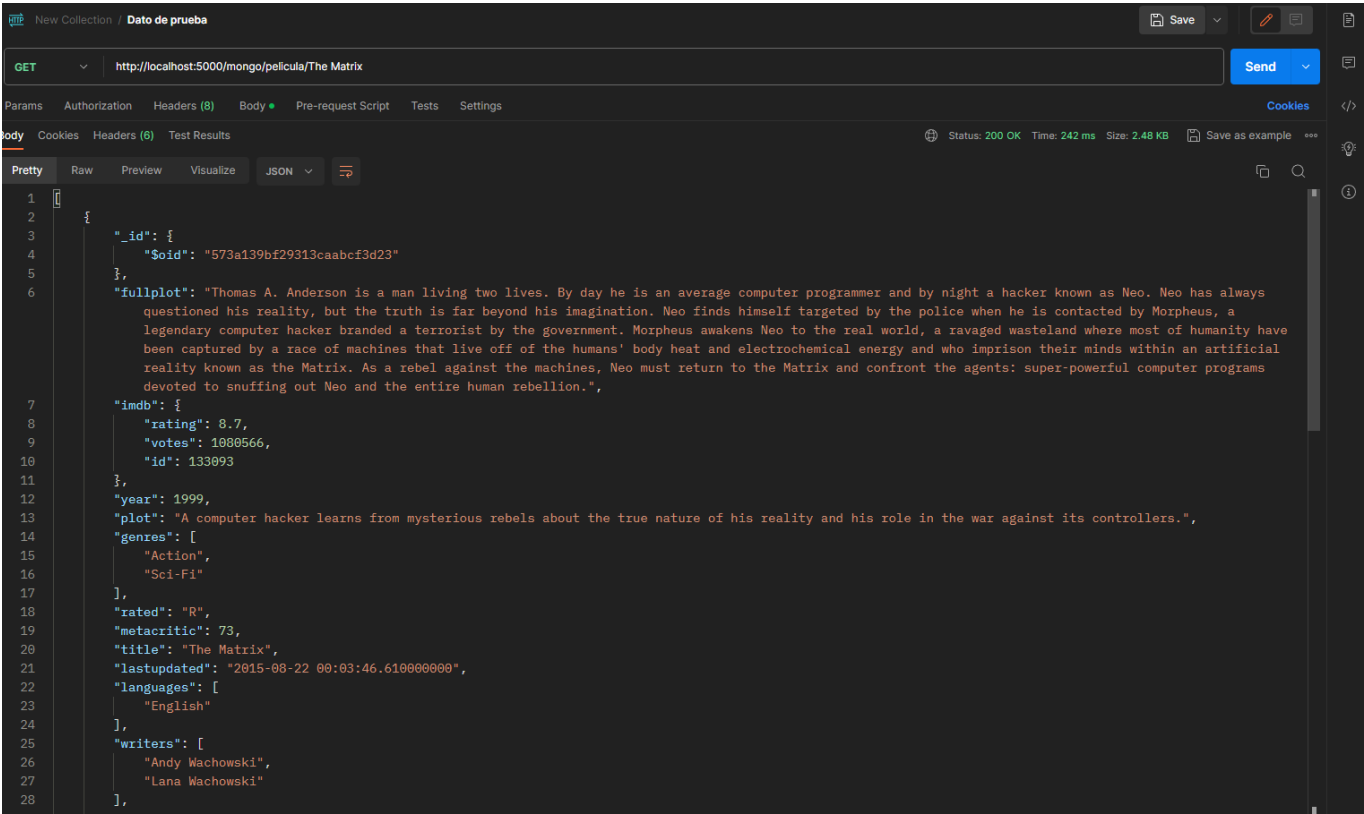
## Pruebas Realizadas

Para probar las diferentes rutas de la api se empleo la herramienta Postman, empleada en experiencias pasadas. Para ejecutar el programa, necesitamos realizar los pasos de la sección pasada, en especial exponer el pod en un puerto de la máquina. Luego de realizar eso nada más agregamos la ruta en la entrada, seleccionamos el metodo que definimos, agregamos el valor de busqueda y le damos al boton **Send**. A continuación se muestran las diferentes pruebas en el caso de Mongo.

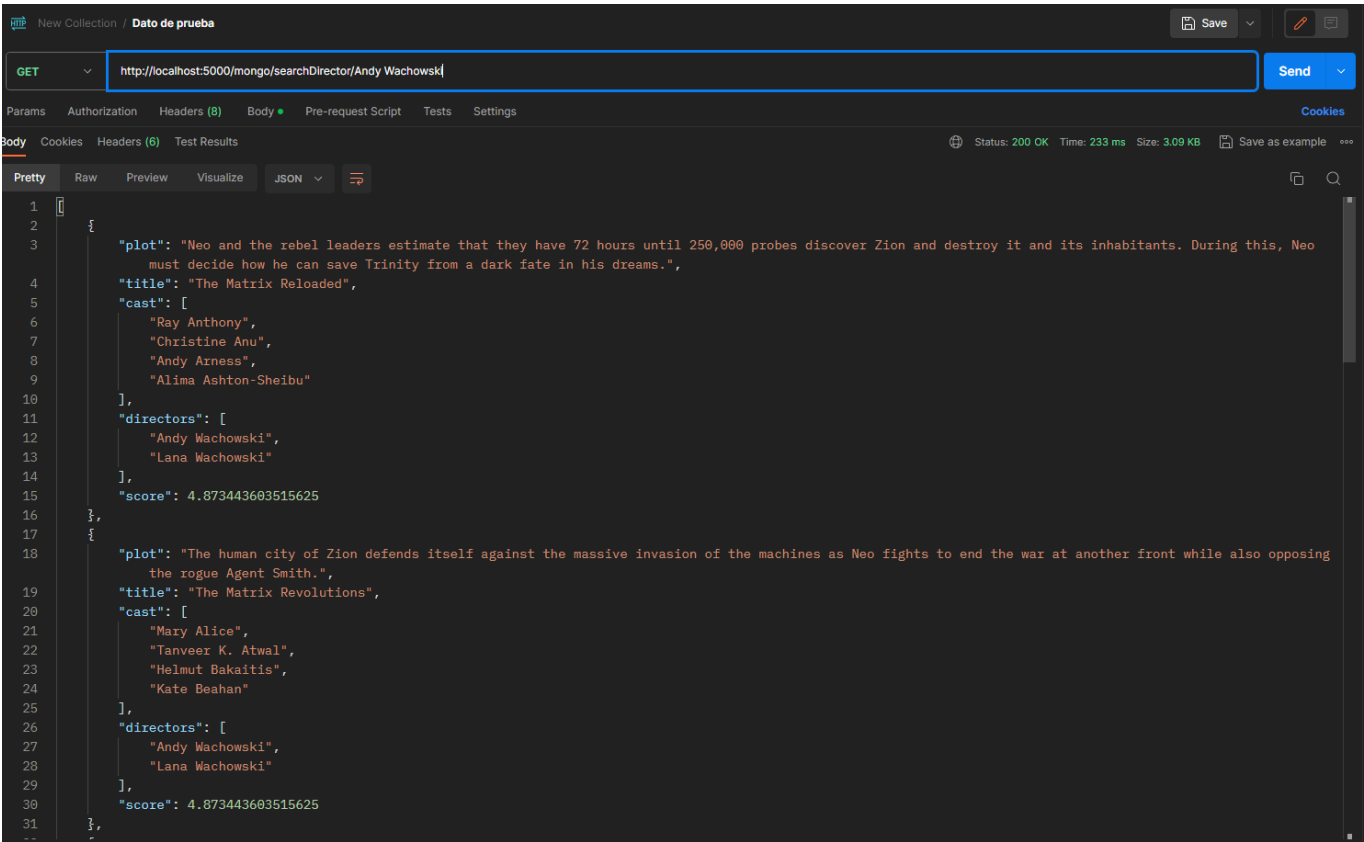
<https://musical-kite-probably.ngrok-free.app/mongo/search/<valor>>: Búsqueda de películas general en los documentos.



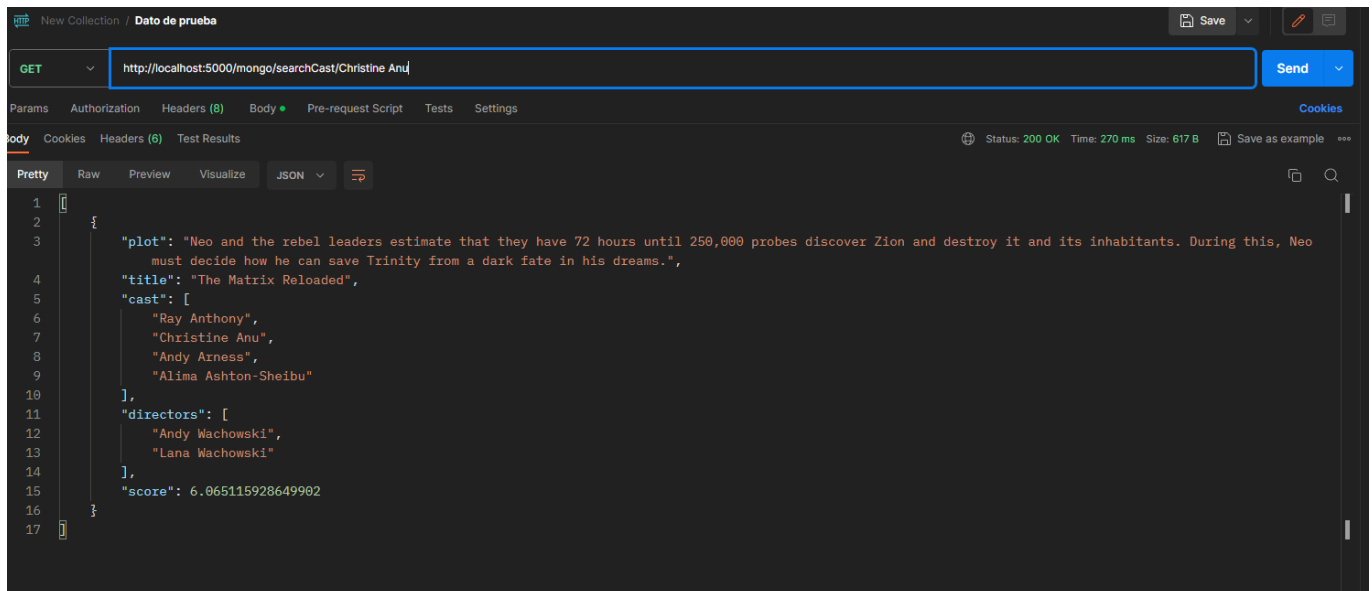
`https://musical-kite-probably.ngrok-free.app/mongo/pelicula/<valor>`: Búsqueda de películas donde devuelve todos el contenido del documento.



https://musical-kite-probably.ngrok-free.app/mongo/searchDirector/<valor>: Búsqueda de películas donde la persona dirige.

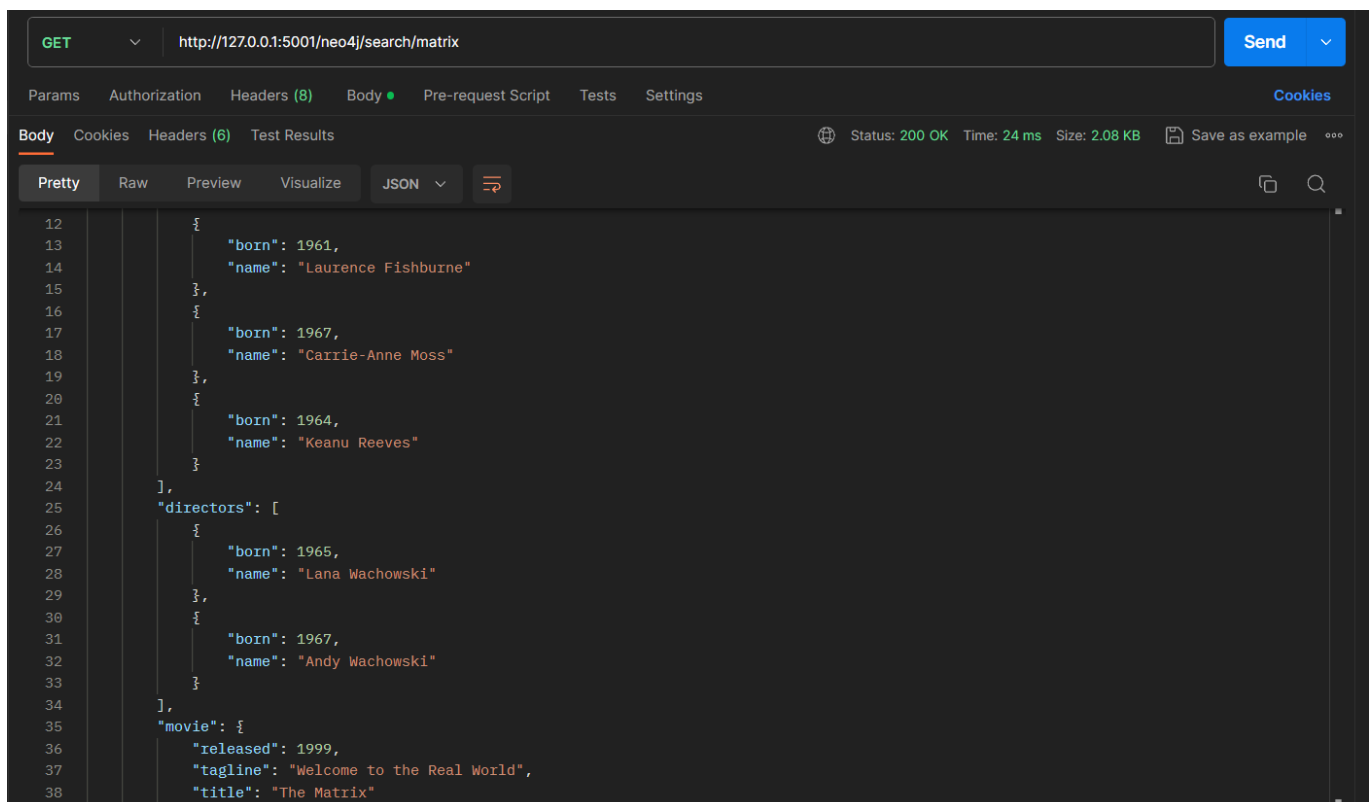


https://musical-kite-probably.ngrok-free.app/mongo/searchCast/<valor>: Búsqueda de películas donde la persona actúa.



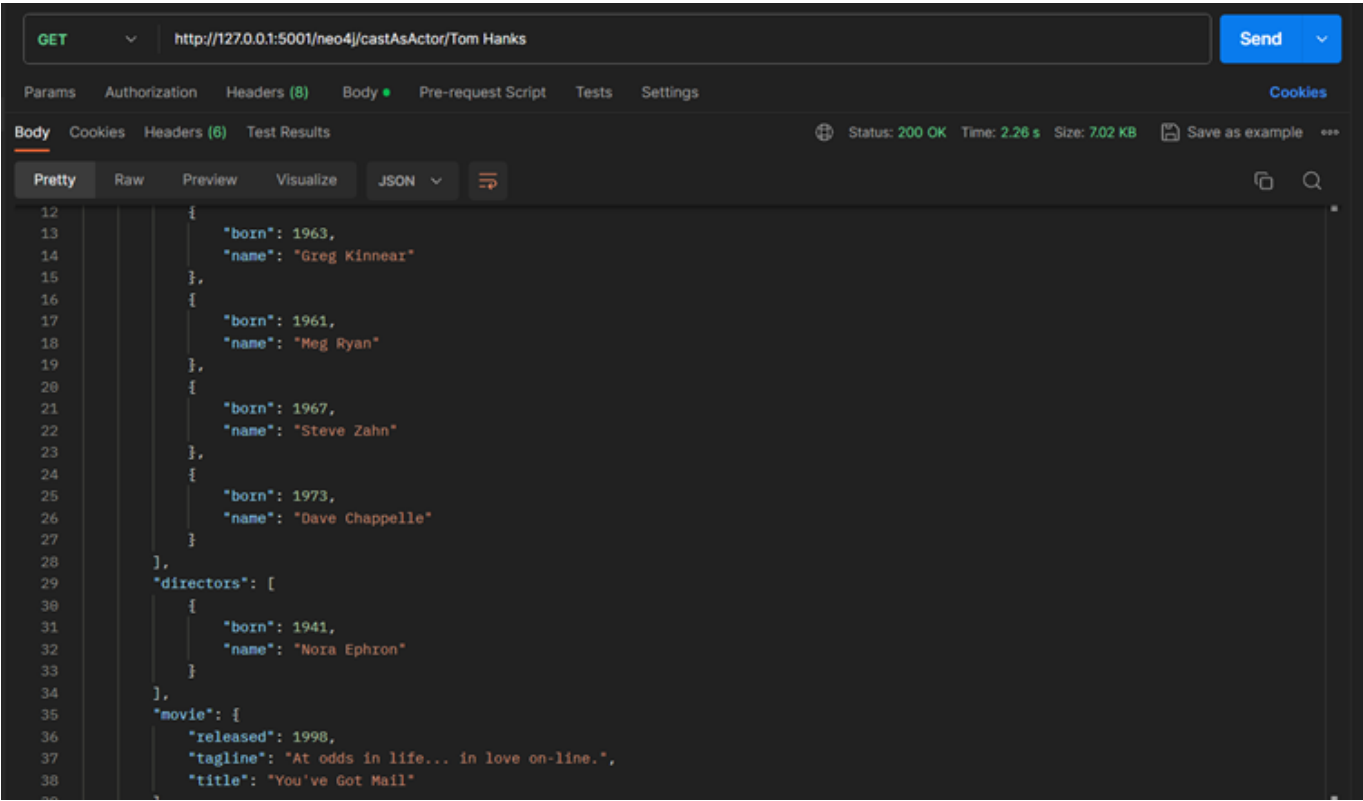
Para Neo4j tenemos las siguientes pruebas desde Postman, donde se muestran los resultados con sus diferentes rutas y con una pequeña descripción de su función.

<https://musical-kite-probably.ngrok-free.app/neo4j/search<valor>>: Búsqueda general en los documentos.

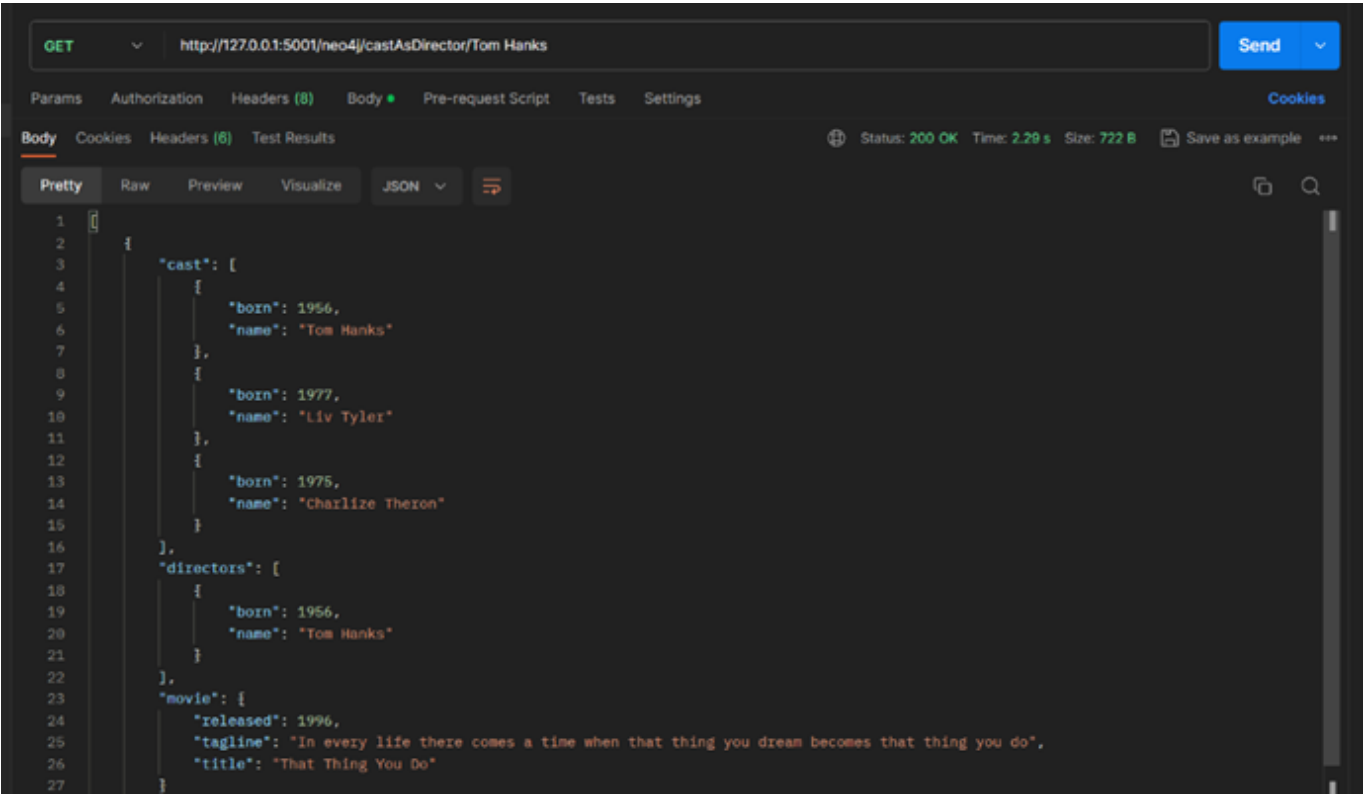


<https://musical-kite-probably.ngrok-free.app/neo4j/castAsActor/<valor>>: Búsqueda de películas donde el actor actúa.

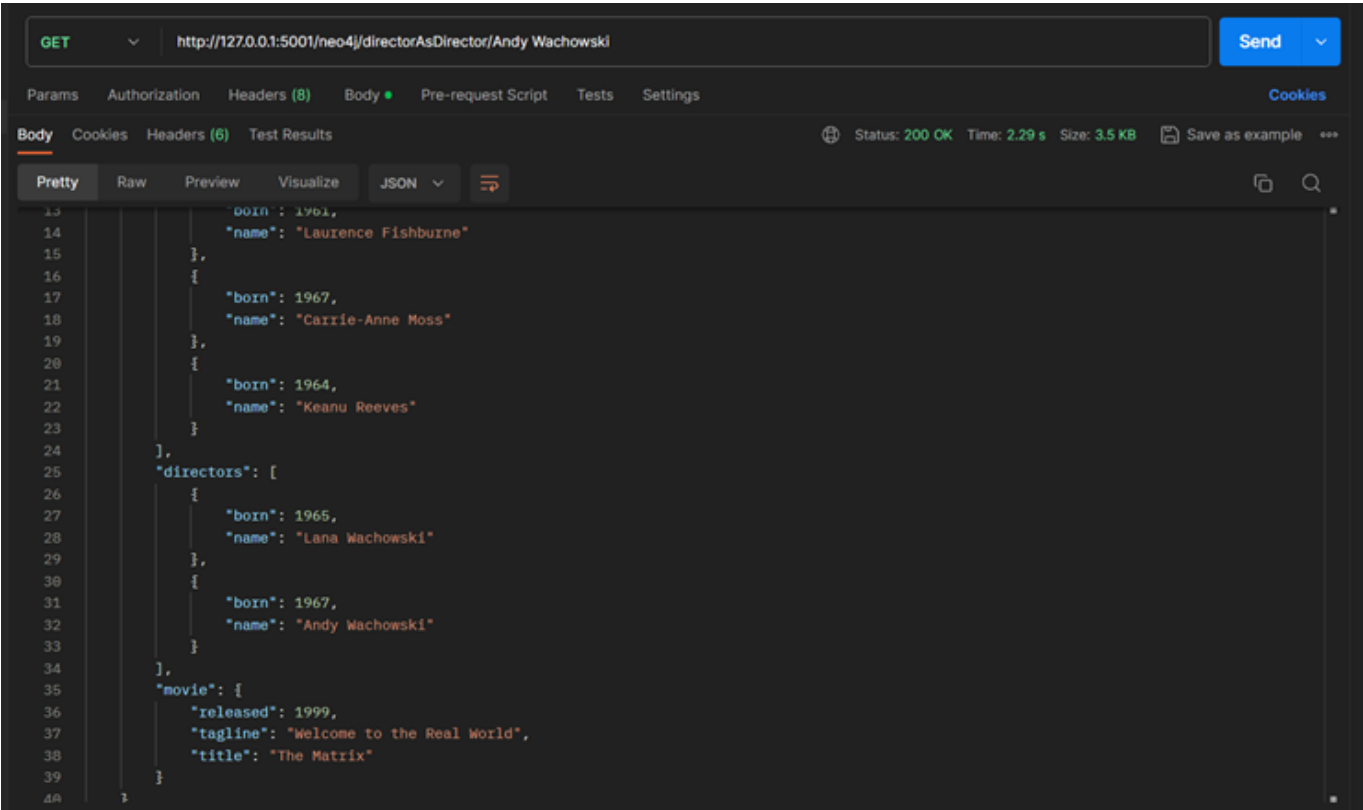




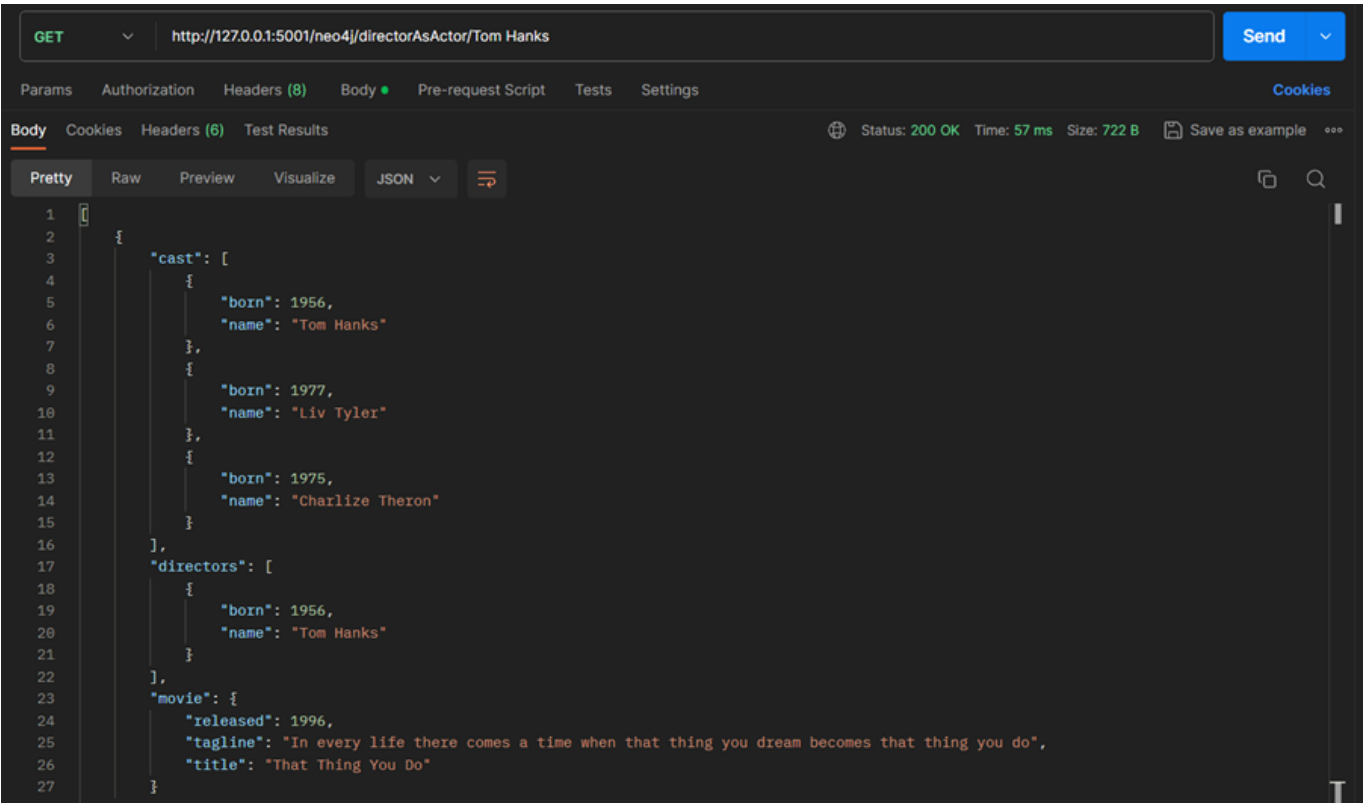
`https://musical-kite-probably.ngrok-free.app/neo4j/castAsDirector/<valor>`: Búsqueda de película donde el actor dirige.



`https://musical-kite-probably.ngrok-free.app/neo4j/directorAsDirector/<valor>`: Búsqueda de películas donde el director dirige.



https://musical-kite-probably.ngrok-free.app/neo4j/directorAsActor/<valor>: Búsqueda de películas donde el director actúa.



## Mapping

Para emplear un índice de búsqueda en Mongo Atlas se realizó el siguiente mapeo, donde se definieron los campos *cast*, *directors*, *plot* y *title*. Para mejorar las búsquedas por cada palabra se definió cada campo como

string para emplear el analizador de textos, también porque son los campos donde se realizan más búsquedas y que sean más precisas. En la siguiente imagen vemos la estructuras del mapping.

```
{
  "analyzer": "lucene.simple",
  "searchAnalyzer": "lucene.simple",
  "mappings": {
    "dynamic": false,
    "fields": {
      "cast": [
        {
          "type": "string"
        }
      ],
      "directors": [
        {
          "type": "string"
        }
      ]
    }
  },
  "plot": [
    {
      "type": "string"
    }
  ],
  "title": [
    {
      "type": "string"
    }
  ]
}
```

## Pruebas unitarias

En las siguientes imagenes se presentan el codigo realizado en **test.py** donde se realizan las pruebas con las APIs. En las primeras pruebas son realizadas con la sección de MongoDB Atlas, comprobando que las diferentes rutas devuelvan código de estado igual a 200, que significa que el request fue exitoso, además se verifica que el contenido retornado por el request no sea cero ante los casos de prueba programados. El codigo podemos apreciarlo en la siguiente imagen.

```

import unittest
import json
from app import app

class TestGeneral(unittest.TestCase):
    def __init__(self, methodName: str = "runTest") -> None:
        self.app = app.test_client(self)
        super().__init__(methodName)

    def test1(self):
        response = self.app.get('/mongo/search/Keanu Reeves')
        self.assertEqual(200, response.status_code, msg=f"{response.status_code}")
        self.assertNotEqual(0, len(response.json[0]), msg=f"Test fail: Len of the document equal to cero {response.json[0]}")
        print("Test 1 Completed")

    def test2(self):
        response = self.app.get('/mongo/searchCast/Keanu Reeves')
        self.assertEqual(200, response.status_code, msg=f"{response.status_code}")
        self.assertNotEqual(0, len(response.json[0]), msg=f"Test fail {len(response.json)}")
        print("Test 2 complete")

    def testAd(self):
        response = self.app.get('/mongo/searchDirector/Andy Wachowski')
        self.assertEqual(200, response.status_code, msg=f"{response.status_code}")
        self.assertNotEqual(0, len(response.json[0]), msg=f"Test fail, datos: {response.json[0]}")
        print("Test 3 complete")

    def test4(self):
        response = self.app.get('/mongo/pelicula/The Matrix')
        self.assertEqual(200, response.status_code, msg=f"{response.status_code}")
        self.assertNotEqual(0, len(response.json[0]), msg=f"Test fail, datos: {response.json[0]}")
        print("Test 4 complete")

if __name__ == "__main__":
    unittest.main()

```

El resultado lo podemos observar en la imagen de abajo:

```

PS C:\Users\Aaron> python d:/rec/base de datos_11/2023-02-23/...
Test 1 Completed
Test 2 complete
Test 4 complete
Test 3 complete
.
-----
Ran 4 tests in 2.224s

OK
PS C:\Users\Aaron>

```

En el caso de Neo4j, se realizan pruebas entre las diferentes rutas comprobando que cada request devuelva de código de estado igual a 200. El código es el siguiente:

```
1  import unittest
2  import requests
3  import os
4  import app
5  from app import *
6  import json
7
8  URL = 'https://18tkx655-5001.euw.devtunnels.ms/'
9
10 class test_api(unittest.TestCase):
11     def test_login(self):
12         headers = {
13             'Content-Type': 'application/json',
14         }
15
16         credentials = json.dumps({
17             "email": "unittest@estudiantec.cr",
18             "password": "unittest"
19         })
20
21         # Proporcionar los encabezados como un parámetro adicional en la solici
22         response = requests.post(URL+"login", data=credentials, headers=headers
23         self.assertEqual(response.status_code, 200)
24
25     def test_getMovies(self):
26         response = requests.get(URL+"neo4j/search/Matrix")
27         self.assertEqual(response.status_code, 200)
28
29     def test_castAsActor(self):
30         response = requests.get(URL+"neo4j/castAsActor/Tom Hanks")
31         self.assertEqual(response.status_code, 200)
32
33     def test_castAsDirector(self):
34         response = requests.get(URL+"neo4j/castAsDirector/Tom Hanks")
35         self.assertEqual(response.status_code, 200)
36
37     def test_directorAsDirector(self):
38         response = requests.get(URL+"neo4j/directorAsDirector/Tom Hanks")
39         self.assertEqual(response.status_code, 200)
40
41     def test_directorAsActor(self):
42         response = requests.get(URL+"neo4j/directorAsActor/Tom Hanks")
43         self.assertEqual(response.status_code, 200)
```

El resultado lo apreciamos en la siguiente imagen.

```
.....
-----
Ran 6 tests in 15.153s

OK
```

## Conclusiones y Recomendaciones

- Gracias a experiencias previas empleando el motor de base de datos MongoDB Atlas, se encontró otros operadores que proveen un mejor resultado a las búsquedas, además de poder emplear otras opciones que permiten los drivers de Python.
- Aun con lo implementado, la documentación posee una gran variedad de información para mejorar los resultados de búsqueda, recomendamos realizar pruebas con dataset ya cargados por el proveedor y realizar una compresión y emplear las herramientas que aporta Mongo Atlas para comprobar cual provee un mejor rendimiento.
- El manejo de emplear Thunkable para diseñar la interfaz de usuarios tiene sus puntos interesantes, pero el manejo de bloques se dificultó por un manejo menos libre de las tecnologías empleadas anteriormente.
- Para manejar Thunkable es necesario tener un orden al trabajar las secciones de la aplicación y hay que tener especial cuidado cuando se están empleando ciclos.
- El uso de bases de datos fue muy interesante debido a que las consultas tienen una estructura similar a las SQL, pero debido a que las relaciones entre nodos son más explícitas y significativas podemos comprender como se conectan, también fue de mucha utilidad la documentación que tiene Neo4j, la cual contiene muchos ejemplos que fueron de utilidad para realizar las consultas.
- Para futuros desarrollos de proyectos relacionados con neo4j se recomienda el uso de una base de datos local, debido a que en ocasiones la BD demo de películas puede no estar disponible, en este caso se utilizó Ne4j Desktop el cual brinda una forma sencilla de crear BD de grafos de manera local.
- En el desarrollo de los proyectos nos hemos dado cuenta de que Firebase es una base de datos con la que manejar usuarios se convierte en una tarea bastante sencilla de realizar, tiene una velocidad de respuesta bastante buena y tiene una disponibilidad prácticamente constante, lo que la hace una excelente opción para estas tareas.
- Si en algún momento se tiene la necesidad de hacer un manejo de usuarios para alguna plataforma web o móvil, Firebase es una solución de fácil instalación y muy rápida de poner en marcha, este semestre algunos la hemos usado tanto en proyectos del curso de Bases de Datos como en el curso de Requerimientos por la buena experiencia que tuvimos en el primer proyecto de Bases.
- Para el manejo de observabilidad trabajando on-premise tiene sus dificultades ya que tenemos que configurar todos los elementos que deseamos orquestar, donde nos encontramos muchas trabas.
- Es necesario investigar sobre como lograr orquestar correctamente servicios como lo fue MongoDB Atlas y la aplicación en Python, o buscar soluciones alternativas que sean más eficientes para realizar su implementación como es Managed Services que provee mayor seguridad, pero con la implicación del costo monetario.