



universidad  
de león



# **Escuela de Ingenierías**

## **Industrial, Informática y Aeroespacial**

### **GRADO EN INGENIERÍA INFORMÁTICA**

Sistemas de Información de  
Gestión y Business Intelligence

Guía de instalación iRate

Autor: D. Álvaro González Jiménez

Tutor: D. Enrique López González

(Diciembre, 2020)

## ÍNDICE

1. Pasos comunes .....	3
2. Base de datos .....	4
3. Backend .....	9
4. Frontend.....	11

## 1. PASOS COMUNES

El primer paso que es totalmente obligatorio realizar para la correcta instalación del proyecto es la descarga del mismo.

Este proceso se puede realizar de dos formas diferentes, ambas igual de válidas.

1. En el caso de tener Git instalado en el dispositivo, basta con abrir la terminal en la ubicación en la que queramos instalar el proyecto y ejecutar el comando:

`git clone https://github.com/agonzj02/SIBI.git`

2. Si por el contrario se prefiere utilizar la interfaz gráfica, tan solo hay que acceder a <https://github.com/agonzj02/SIBI.git> y descargar el código manualmente pulsando en el botón verde y, posteriormente, en Download zip.

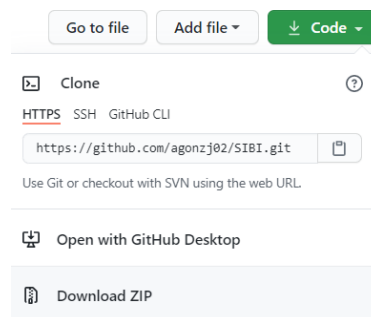


Figura 1. Descarga gráfica del proyecto

Tras haber descargado el proyecto, nos encontraremos una carpeta como la siguiente:

.vscode	08/12/2020 16:38	Carpeta de archivos	
backend	08/12/2020 16:38	Carpeta de archivos	
BBDD	08/12/2020 16:38	Carpeta de archivos	
frontend	08/12/2020 16:38	Carpeta de archivos	
iRate.pptx	08/12/2020 16:38	Presentación de M...	17.171 KB
Memoria.docx	08/12/2020 16:38	Documento de Mi...	25.584 KB
Memoria.pdf	08/12/2020 16:38	Adobe Acrobat D...	3.217 KB
Presentación.mp4	08/12/2020 16:38	Archivo MP4	16.100 KB
README.md	08/12/2020 16:38	Archivo MD	1 KB

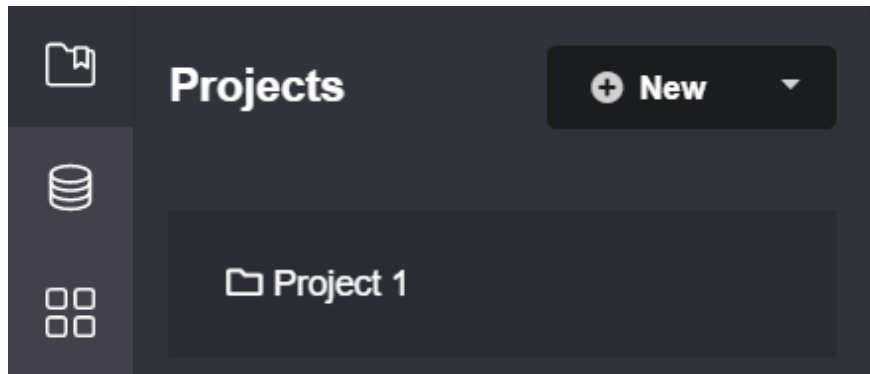
Figura 2.- Contenido proyecto

A continuación, podemos ya pasar a instalar todos los componentes del proyecto por separado.

## 2. BASE DE DATOS

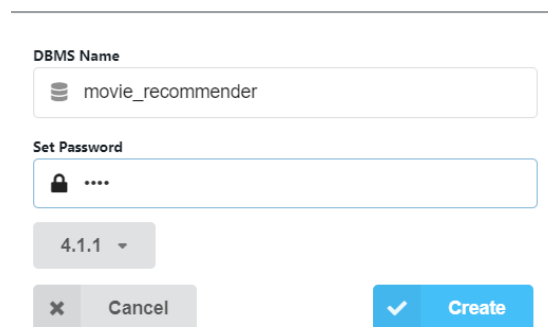
Como la base de datos utilizada es Neo4j, es estrictamente necesario descargar el programa Neo4j Desktop, que se puede encontrar en el siguiente enlace: <https://neo4j.com/download/>.

Una vez descargado el programa, lo ejecutamos y creamos un nuevo proyecto pulsando en el botón New.



*Figura 3.- Creación nuevo proyecto*

Dentro del proyecto, creamos una base de datos local.

The image shows a 'Create Database' dialog box. It has a 'DBMS Name' field with the value 'movie\_recommender'. Below it is a 'Set Password' field with a lock icon and four dots. At the bottom, there's a version selector showing '4.1.1' with a dropdown arrow. There are two buttons at the bottom: 'Cancel' with a red 'X' icon and 'Create' with a green checkmark icon.

*Figura 4.- Creacion Base datos*

Aunque el nombre y la contraseña no son relevantes, es totalmente imprescindible que la versión de la misma sea la 4.1.1.

Acto seguido, es necesario acceder al panel de configuración de la base de datos que acabamos de crear e instalar el plugin llamado APOC.

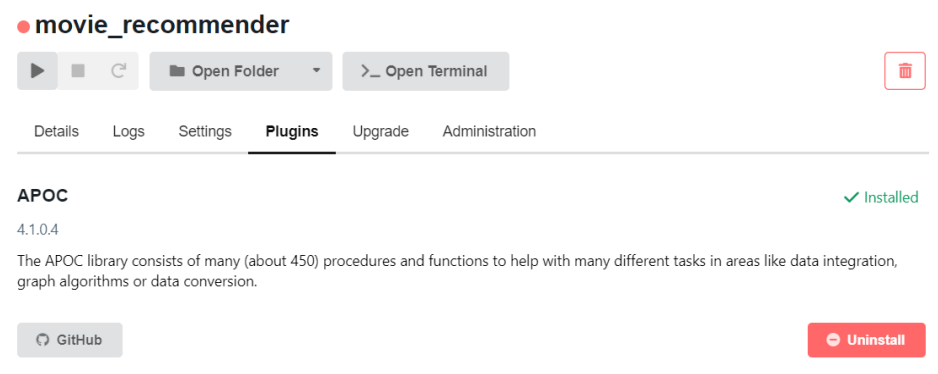


Figura 5.- Instalación plugin APOC

Posteriormente, en la pestaña Settings, es necesario añadir las siguientes 2 líneas y aplicar los cambios.

`apoc.export.file.enabled=true`

`apoc.import.file.enabled=true`

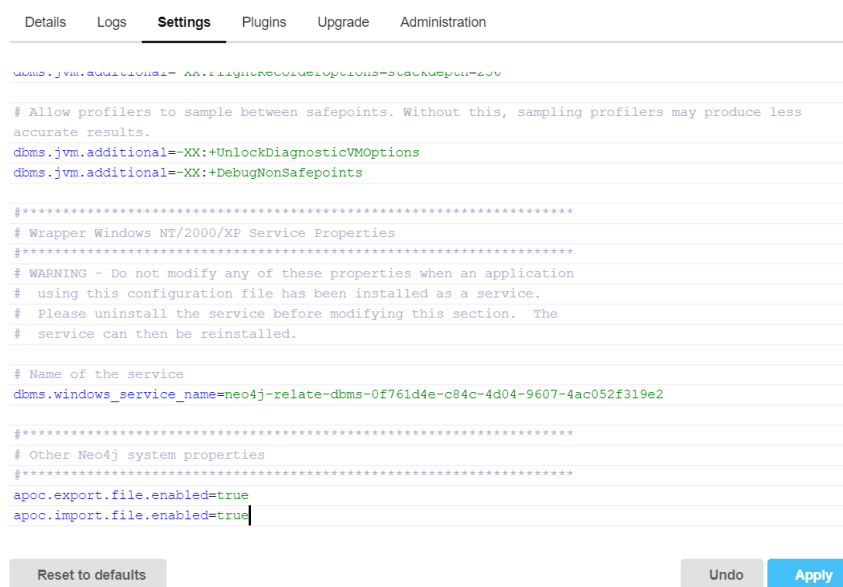


Figura 6.- Adición propiedades al archivo de configuración

Aunque yo no he tenido ningún problema al realizar la carga, en el caso de que existiera algún problema durante la misma relativo al tamaño de datos introducidos, sería necesario aumentar el tamaño del HEAP, lo cual se puede realizar modificando las siguientes líneas del mismo archivo.

```
# Java Heap Size: by default the Java heap size is dynamically calculated based
# on available system resources. Uncomment these lines to set specific initial
# and maximum heap size.
dbms.memory.heap.initial_size=512m
dbms.memory.heap.max_size=1G
```

*Figura 7.- Modificación tamaño HEAP*

Bastaría con poner 2G en la primera línea y 4G en la segunda y aplicar los cambios.

En el caso de que vayamos a tener más de una base de datos dentro del sistema gestor (que es bastante probable en el caso de hacer un uso normal del mismo), es necesario crear una base de datos a la que añadir toda la información. Si no hiciéramos esto, todo se añadiría siempre en la misma base de datos y daría lugar a inconsistencias.

Para crear dicha base de datos tan solo hay que abrir el Neo4j Browser habiendo arrancado previamente la base de datos y ejecutar la consulta

```
CREATE DATABASE movierec
```

El nombre puede ser el que se quiera, no es algo relevante. Una vez creada la base de datos, paramos la instancia y volvemos a editar el fichero de configuración. Aquí, descomentamos la siguiente línea y modificamos el nombre de la base de datos por defecto (neo4j en este caso) por la base de datos que acabamos de crear, movierec.

```
# The name of the default database.
dbms.default_database=movierec
```

*Figura 8.- Asignación BBDD por defecto*

Con esto, hacemos que cada vez que se arranque la instancia de la base de datos trabajemos por defecto con la base de datos movierec y no con neo4j.

En este punto, ya tenemos que arrancar la base de datos pulsando en Start.

Por último, ya podemos proceder a importar los datos. Para ello, primero es necesario abrir la carpeta en la que deben estar los ficheros de importación, lo cual se hace dentro de Manage, pulsando en la flecha que se encuentra al lado de Open Folder y pulsando en el botón Import.

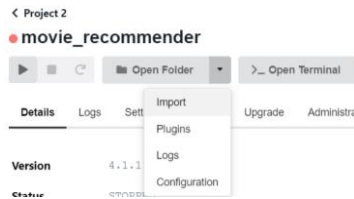


Figura 9.- Abrir carpeta import

Acto seguido, es necesario copiar el archivo denominado `export.cypher`, que puede encontrarse dentro del proyecto en la carpeta BBDD y pegarlo en la carpeta import.

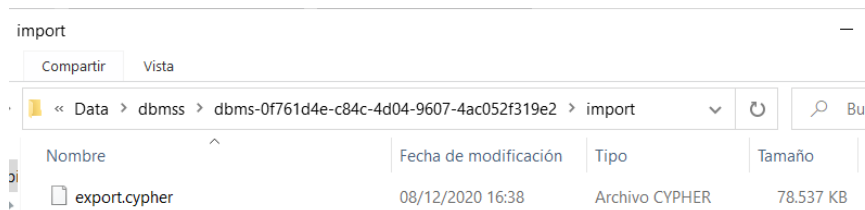


Figura 10.- Contenido carpeta import

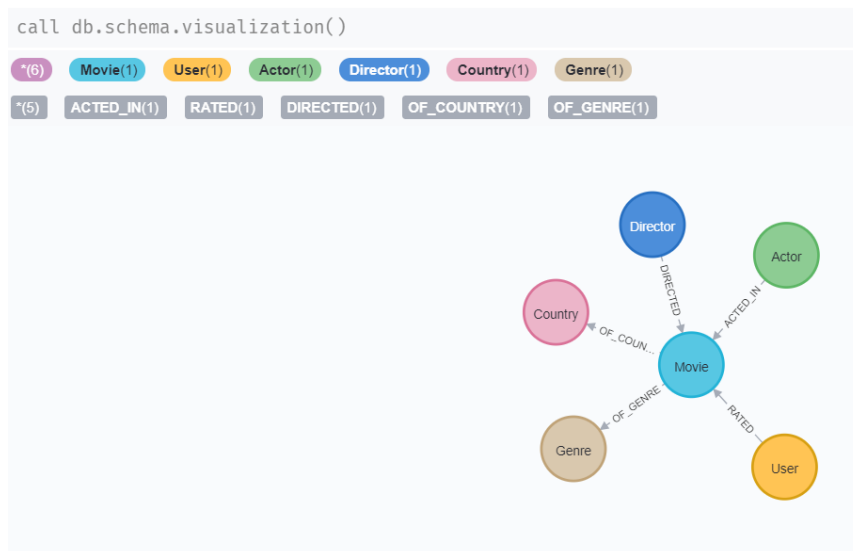
Por último, abrimos una terminal en dicha carpeta y ejecutamos el siguiente comando, que puede variar dependiendo del sistema operativo. **IMPORTANTE** tener la base de datos arrancada.

- Windows = `type export.cypher | ..\bin\cypher-shell.bat -u neo4j -p 1234`
- UNIX => `cat export.cypher | ..\bin\cypher-shell -u neo4j -p 1234`

El usuario y la contraseña deben ser `neo4j` y `1234` respectivamente. En el caso de que sean diferentes, deberá cambiarse en el comando y posteriormente en el servidor de proyecto.

Al ejecutar el comando, comenzará la carga de la base de datos. Como la base consta de 110.000 nodos y 1.000.000 de relaciones, el proceso de carga se demora un poco. En mi caso, tarda unos 20 minutos en efectuar la carga completa, pero este tiempo depende totalmente de la capacidad de procesamiento del dispositivo en el que se realice la instalación.

Cuando se termine la carga de los datos, podremos ya abrir Neo4j Browser y observar que, en efecto, se ha realizado correctamente.



*Figura 11. Comprobación importación correcta*

Ejecutando la siguiente consulta Cypher podemos comprobar si la carga se ha efectuado de manera correcta o no.



### 3. BACKEND

En este paso, nos movemos a la carpeta backend que puede encontrarse en el proyecto. Para poder ejecutar sin problemas el servidor de aplicación (backend) es necesario partir de una serie de premisas.

1. La base de datos está activa. En caso de que no se esté ejecutando obtendremos un error dado que fallará la conexión con la misma.
2. El usuario y contraseña de acceso a la base de datos son neo4j y 1234 respectivamente. En el caso de no ser así, será necesario o bien modificar dichas credenciales de acceso o bien modificar el archivo .env ubicado en la carpeta backend, que tiene el siguiente aspecto.

```
SIBI > backend > .env
1  MOVIE_DATABASE_USERNAME="neo4j"
2  MOVIE_DATABASE_PASSWORD="1234"
3  MOVIE_DATABASE_URL="bolt://localhost:7687"
```

*Figura 12. Contenido archivo .env*

En este archivo deben encontrarse las credenciales de acceso a la base de datos correctas. De no ser así, se producirá un error al intentar crear una conexión.

3. El sistema tiene instalado Python3. De no ser así, se puede descargar en el siguiente enlace: <https://www.python.org/downloads/>. Es totalmente imprescindible que la versión sea la 3, la versión 2 no es compatible.
4. El sistema tiene instalado el sistema gestor de paquetes pip. De no ser así, se puede descargar en el siguiente enlace: <https://pip.pypa.io/en/stable/installing/>

En el momento en el que se cumplan estas 4 premisas, ya se puede proceder a la instalación y ejecución del servidor.

El primer paso a realizar consiste en instalar todos los paquetes necesarios para el correcto funcionamiento del servidor. Para ello, es necesario abrir una terminal dentro de la carpeta backend del proyecto y ejecutar el siguiente comando:

```
pip install -r requirements.txt
```

Por ultimo, basta con ejecutar el siguiente comando para iniciar el servidor:

python app.py

```
PS C:\Users\alvar\Downloads\SIBI-main (1)\SIBI-main\backend> python app.py
* Serving Flask app "app" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: on
* Restarting with stat
* Debugger is active!
* Debugger PIN: 650-278-403
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
```

*Figura 13.- Ejecución servidor*

Como se puede comprobar, el servidor se ejecuta a la perfección.

## 4. FRONTEND

Para poder ejecutar el frontend, lo primero que hacemos es movernos a la carpeta frontend del proyecto.

Para poder ejecutarlo sin problemas, es totalmente necesario tener instalado nodejs. En el caso de no tenerlo instalado, se puede encontrar en el siguiente enlace: <https://nodejs.org/en/download/>.

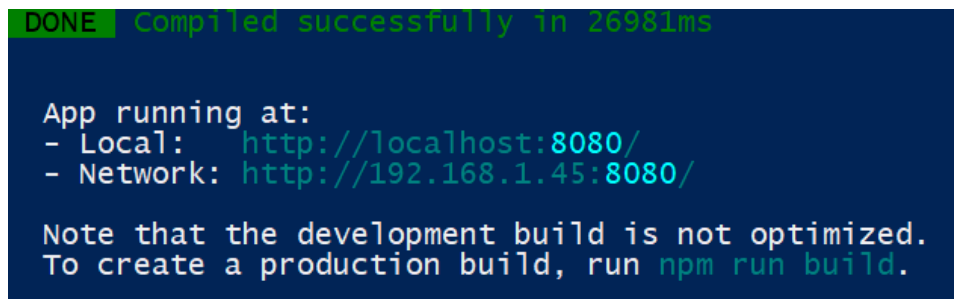
Habiendo ya instalado nodejs, y por tanto su gestor de paquetes npm, lo primero que tenemos que hacer es abrir una terminal en la carpeta frontend del proyecto y ejecutar el siguiente comando:

```
npm install
```

A continuación, tan solo habrá que ejecutar el siguiente comando para arrancar el servidor:

```
npm run serve
```

Tras esto, nos aparecerá en la terminal un mensaje similar al siguiente:

A terminal window with a dark blue background and green text. The first line says "DONE Compiled successfully in 26981ms". The second line says "App running at:". The third line shows "- Local: http://localhost:8080/". The fourth line shows "- Network: http://192.168.1.45:8080/". The fifth line says "Note that the development build is not optimized." and the sixth line says "To create a production build, run npm run build.".

```
DONE Compiled successfully in 26981ms

App running at:
- Local: http://localhost:8080/
- Network: http://192.168.1.45:8080/

Note that the development build is not optimized.
To create a production build, run npm run build.
```

*Figura 14.- Mensaje correcta compilación*

Para poder acceder a la aplicación, tan solo hay que acceder a una de las rutas indicadas en la pantalla, <http://localhost:8080/> por ejemplo, obteniendo lo siguiente.

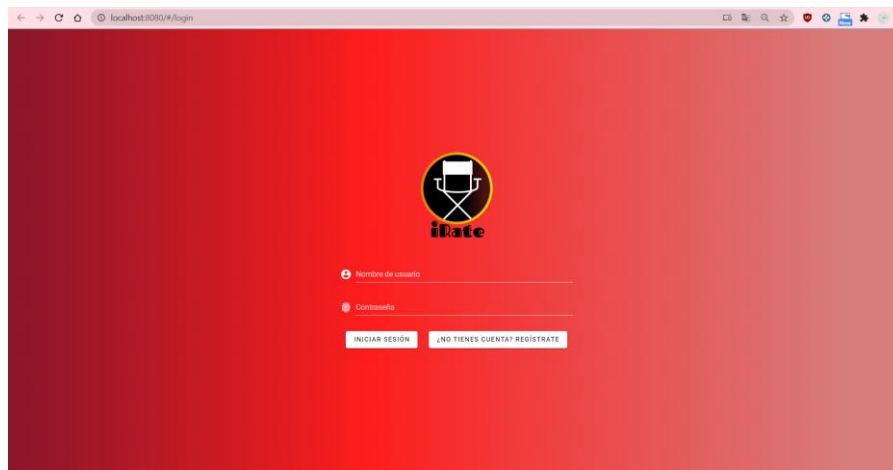


Figura 15. Acceso al frontend de la aplicación.

Si ahora probamos a registrarnos, vemos que en el backend de la aplicación se genera tráfico.

```
PS C:\Users\alvar\Downloads\SIBI-main (1)\SIBI-main\backend> python app.py
* Serving Flask app "app" (lazy loading)
* Environment: production
  WARNING: This is a development server. Do not use it in a production deployment.
  Use a production WSGI server instead.
* Debug mode: on
* Restarting with stat
* Debugger is active!
* Debugger PIN: 650-278-403
* Running on http://127.0.0.1:5000/ (Press CTRL+C to quit)
127.0.0.1 - - [09/Dec/2020 13:55:10] "[37mOPTIONS /register HTTP/1.1-[0m" 200 -
127.0.0.1 - - [09/Dec/2020 13:55:10] "[37mPOST /register HTTP/1.1-[0m" 201 -
127.0.0.1 - - [09/Dec/2020 13:55:22] "[37mOPTIONS /login HTTP/1.1-[0m" 200 -
127.0.0.1 - - [09/Dec/2020 13:55:22] "[37mPOST /login HTTP/1.1-[0m" 200 -
127.0.0.1 - - [09/Dec/2020 13:55:24] "[37mGET /top HTTP/1.1-[0m" 200 -
```

Figura 16.- Tráfico generado en el backend

Como vemos se ha generado tráfico relativo al registro, login y obtención de las películas más valoradas. La aplicación estaría por tanto funcionando a pleno rendimiento.