



APLICACIÓN CON CONEXIÓN A BASE DE DATOS



Alumno: Kevin Martinez irraestro

Docente: Cardoso Jimenez Ambrosio

Carrera: Ing. en Tic's

Materia: Taller de Bases de Datos

Repositorio de gitHub:

<https://github.com/kodaMtz/Conexion-a-Base-de-Datos.git>

T4A

INSTALACIÓN Y CONFIGURACIÓN DEL “BUN”

INSTALACION Y CONFIGURACION DEL RUN'POSTGRES “BUN”:

Bun es un entorno de ejecucion de JavaScript, similar al de Node.js pero con enfoque en velocidad y simplicidad, es un kit de herramientas que incluye un gestor de paquetes, un empaquetador, un ejecutor de pruebas, lo que ayuda a un desarrollador web.



Pasos para la instalacion de “Bun”:

1. Nos dirigimos hacia la pagina de doumentacion de instalacion “Bun” : <https://bun.sh/docs/installation> (Revise imagen 1).

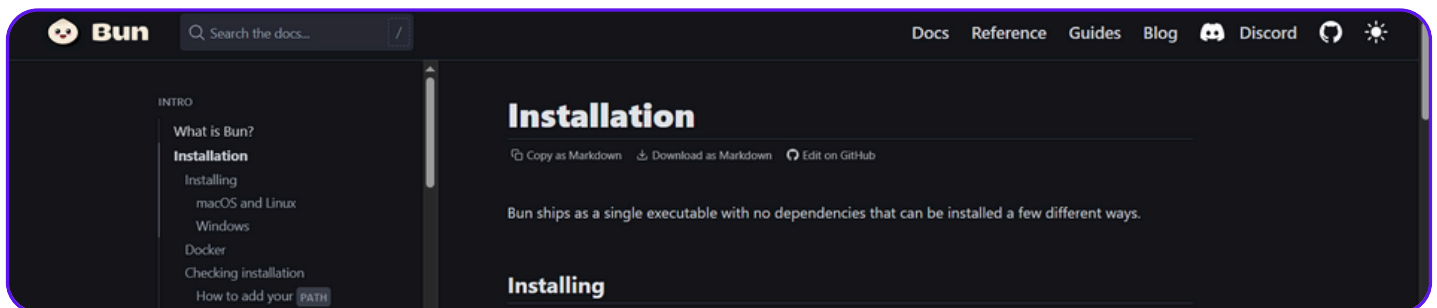


Imagen 1: “Pagina de instalacion de bun”

2. Una vez ya dentro, dependiendo del sistema operativo de la laptop, en mi caso Windows, nos dirigiremos a los codigos que nos proporciona para su instalacion desde Powershell o CMD. (Revise image 2)

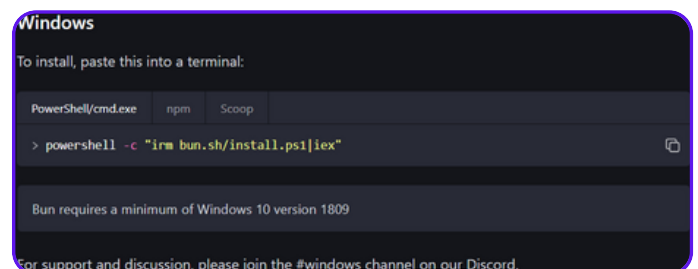


Imagen 2: “Codigo de instalacion”

3. Copiaremos el codigo de powershell abriremos una nueva ventana de este mismo programa y pegaremos el cdigo (Una vez hech, comenzara a descargar “bun” debera de esperar a que complete el 100%) una vez hecho, para revisar si se instalo la ultima version metemos el siguiente comando: (Revise imagen 3)

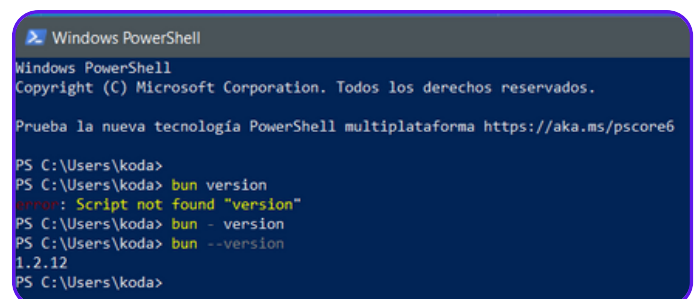


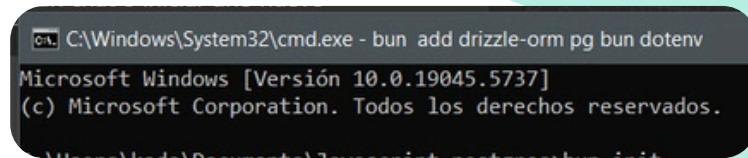
Imagen 3: “Ultima version de bun”

4. Una vez ya descargado, Nos dirigiremos a un directorio para instalarlo, en mi caso, en una carpeta donde viene un proyecto ya previsto de la base de datos “DVDrental”, nos dirigimos a su ruta de ubicacion y escribimos “cmd” nos abra una ventana en consola (Revise imagen 4)

5. Una vez ya abierto el cmd, escribiremos el comando “bun init” es un comando que se utiliza para crear un nuevo proyecto en blanco con Bun (Revise imagen 5)

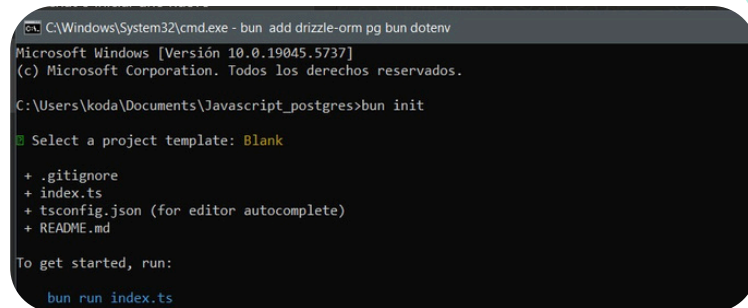
6. Una vez creado colocaremos el siguiente comando:
“bun add drizzle-orm pg bun dotenv”
para instalar varias dependencias necesarias para trabajar con Drizzle ORM, PostgreSQL y variables de entorno. (Revise imagen 6)

7. Una vez ya hecho, en mi caso ya tengo un proyecto de mi docente el cual contiene un archivo de tipo: “ts” en el cual es “server.ts” para hacer la conexion por localhost, en mi caso se mostro un error, por lo que debi de hacer la instalacion de dotenv. (Imagen 7)



```
C:\Windows\System32\cmd.exe - bun add drizzle-orm pg bun dotenv
Microsoft Windows [Versión 10.0.19045.5737]
(c) Microsoft Corporation. Todos los derechos reservados.
```

Imagen 4: “CMD en windows”



```
C:\Windows\System32\cmd.exe - bun add drizzle-orm pg bun dotenv
Microsoft Windows [Versión 10.0.19045.5737]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\koda\Documents\Javascript_postgres>bun init

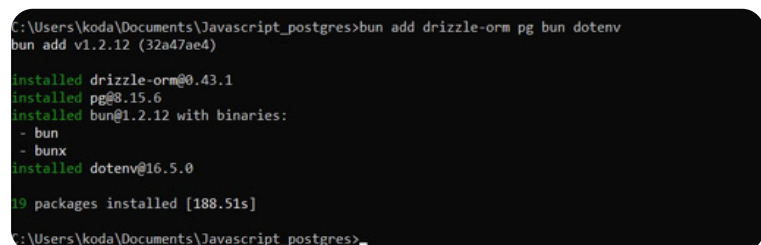
Select a project template: Blank

+ .gitignore
+ index.ts
+ tsconfig.json (for editor autocomplete)
+ README.md

To get started, run:

  bun run index.ts
```

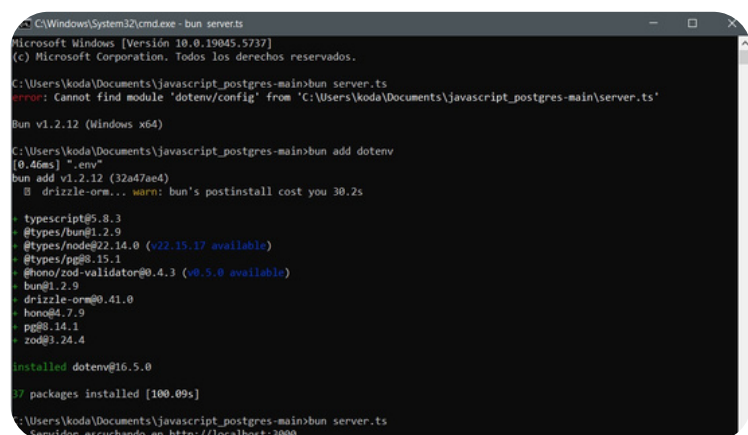
Imagen 5: “Proyecto en blanco en el directorio”



```
C:\Users\koda\Documents\Javascript_postgres>bun add drizzle-orm pg bun dotenv
bun add v1.2.12 (32a47ae4)

installed drizzle-orm@0.43.1
installed pg@8.15.6
installed bun@1.2.12 with binaries:
- bun
- bunx
installed dotenv@16.5.0
19 packages installed [188.51s]
```

Imagen 6: “Instalacion de dependencias de bun”



```
C:\Windows\System32\cmd.exe - bun server.ts
Microsoft Windows [Versión 10.0.19045.5737]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\koda\Documents\javascript_postgres-main>bun server.ts
error: Cannot find module 'dotenv/config' from 'C:\Users\koda\Documents\javascript_postgres-main\server.ts'

Bun v1.2.12 (Windows x64)

C:\Users\koda\Documents\javascript_postgres-main>bun add dotenv
bun add v1.2.12 (32a47ae4)
  drizzle-orm... warn: bun's postinstall cost you 30.2s

+ typescript@5.8.3
+ @types/bun@1.2.9
+ @types/node@22.14.0 (v22.15.17 available)
+ @types/pg@8.15.1
+ @hono/zod-validator@0.4.3 (v0.5.0 available)
+ bun@1.2.9
+ drizzle-orm@0.41.0
+ hono@4.7.9
+ pg@8.14.1
+ zod@3.24.4

installed dotenv@16.5.0
37 packages installed [100.09s]

C:\Users\koda\Documents\javascript_postgres-main>bun server.ts
Servidor escuchando en http://localhost:3000
```

Imagen 7: “Conexion con el localhost:3000”

Esta es la forma con la que se conecta hacia el localhost, mas adelante se mostrara la conexion y creacion de archivos en “IntelliJ IDEA” donde se mostrara el codigo de como se hace la ejecucion de la conexion con la base de datos y podemos acceder a las tablas por medio de nuestro navegador preferido.

CONEXION DE BASE DE DATOS

“JAVASCRIPT - POSTGRES”

Para la conexión de base de datos, estaremos utilizando el programa de IntelliJ IDEA, es un IDE (Integrated Development Environment) o entorno de desarrollo integrado, creado por JetBrains especialmente para el desarrollo de software, especialmente en Java y Kotlin.



Primero veremos la estructura de las carpetas:

- **SRC:** Carpeta principal que contiene todo el código fuente del proyecto.
- **Controllors:** Maneja la lógica de las solicitudes HTTP (req/res).
- **db:** Contiene la configuración de la base de datos (PostgreSQL).
- **Middlewares:** Funciones intermedias que procesan solicitudes antes de llegar al controlador (ej: autenticación, validación).
- **Repositories:** Capa de acceso directo a la base de datos. Aquí se escriben las consultas SQL o se usan ORMs.
- **Routes:** Define las rutas de la API y las asocia a los controladores.
- **schemas:** Define la estructura de datos.
- **services:** Procesar datos antes de guardarlos en la base de datos.
- **utils:** Funciones auxiliares reutilizables

Archivos en la raíz:

- app.ts: Configuración de Express (middlewares globales, rutas base).
- server.ts: Inicia el servidor (escucha en el puerto 3000).
- .env: Variables de entorno
- package.json: Dependencias y scripts del proyecto (ej: npm run dev para iniciar).

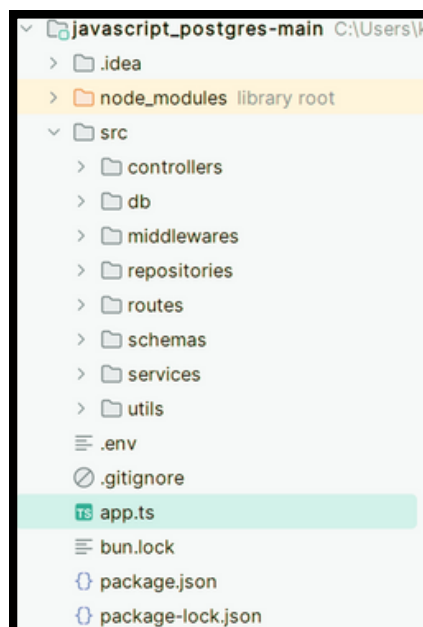


Imagen de la estructura de carpetas:

1. Db/Schema: En esta parte creamos 2 archivos de “TS” en el cual vamos a hacer la exportacion de 2 tablas a las cuales le daremos el codigo para saber el id, el titulo, entre otros, sera para actores como peliculas. (Revisa imagen 1)



Imagen 1: “Aqui se exporta lo que queremos que el usuario logre ver lo cual seria el id, actor, pelicula”

2. Repositories: En esta carpeta, se hace las consultas SQL dependiendo del ID que quiera colocar el usuario, en los archivos de TS de actor y film, se hace una importacion de schema. (Revisa imagenes 2 y 3)



Imagen 2: “Aqui se hace la consulta por medio del id para los actores”



Imagen 3: “Aqui se hace la consulta por medio del id para las peliculas”

3. Schemas: En esta carpeta, se define las validaciones de cada atributo que asignamos anteriormente, por ejemplo: para los nombres seran .string, para los apellidos seran .string, todo creados por medio de objetos. (Revisar imagen 4 y 5)



Imagen 4: “Aqui se define para actores”



Imagen 5: “Aqui se define para Peliculas”

4. Services: Aqui deben de editarse como se debe de ver los resultados, en este caso como tipo JSON y como queremos que los visualice el usuario (Revisa imagenes 6 y 7)



Imagen 6: “Para los actores se visualizara asi”



Imagen 7: “Para las peliculas se visualizara asi”

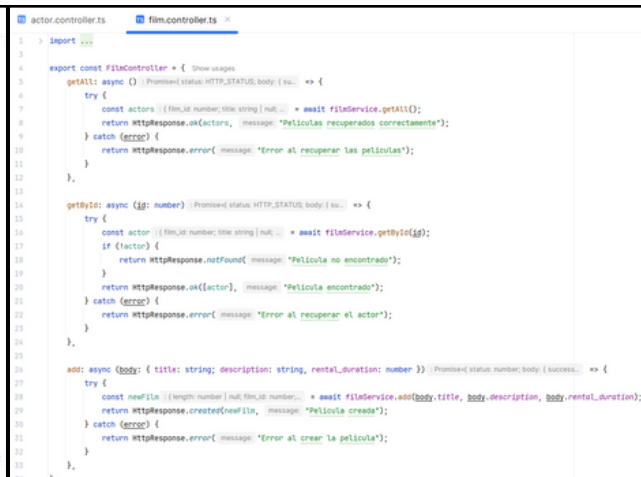
5. **Controllers:** En esta carpeta se crean los getters y los metodos de añadir para actores y para peliculas, ademas de que se añade al mismo BD cuando desea el usuario añadir peliculas o actores (Revise imagenes 8 y 9)



```

1 import { ActorService } from '../services/actor.service.ts';
2 import { HttpResponse } from '../utils/http_response.ts';
3
4 export const ActorController = {
5   getAll: async () => {
6     try {
7       const actors = await ActorService.getAll();
8       return HttpResponse.ok(actors, message: 'Actores recuperados correctamente');
9     } catch (error) {
10      return HttpResponse.error(message: 'Error al recuperar los actores');
11    }
12  },
13
14   getById: async (id: number) => {
15     try {
16       const actor = await ActorService.getById(id);
17       if (actor) {
18         return HttpResponse.ok(actor, message: 'Actor encontrado');
19       }
20       return HttpResponse.notFound(message: 'Actor no encontrado');
21     } catch (error) {
22      return HttpResponse.error(message: 'Error al recuperar el actor');
23    }
24  },
25
26   add: async (body: { first_name: string; last_name: string }) => {
27     try {
28       const newActor = await ActorService.add(body.first_name, body.last_name);
29       return HttpResponse.created(newActor, message: 'Actor creado');
30     } catch (error) {
31      return HttpResponse.error(message: 'Error al crear el actor');
32    }
33  },
34 };
  
```

Imagen 8: "Para los actores se visualizara asi"

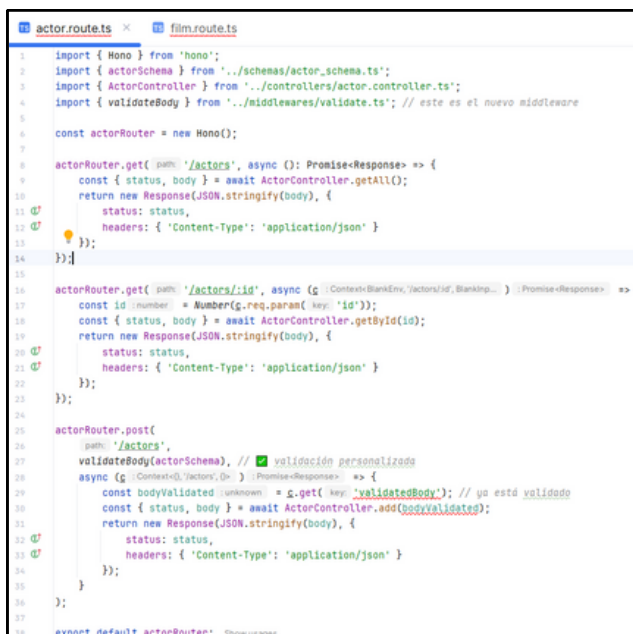


```

1 import { FilmService } from '../services/film.service.ts';
2 import { HttpResponse } from '../utils/http_response.ts';
3
4 export const FilmController = {
5   getAll: async () => {
6     try {
7       const films = await FilmService.getAll();
8       return HttpResponse.ok(films, message: 'Peliculas recuperadas correctamente');
9     } catch (error) {
10      return HttpResponse.error(message: 'Error al recuperar las peliculas');
11    }
12  },
13
14   getById: async (id: number) => {
15     try {
16       const film = await FilmService.getById(id);
17       if (film) {
18         return HttpResponse.ok(film, message: 'Pelicula encontrada');
19       }
20       return HttpResponse.notFound(message: 'Pelicula no encontrada');
21     } catch (error) {
22      return HttpResponse.error(message: 'Error al recuperar el actor');
23    }
24  },
25
26   add: async (body: { title: string; description: string; rental_duration: number }) => {
27     try {
28       const newFilm = await FilmService.add(body.title, body.description, body.rental_duration);
29       return HttpResponse.created(newFilm, message: 'Pelicula creada');
30     } catch (error) {
31      return HttpResponse.error(message: 'Error al crear la pelicula');
32    }
33  },
34 };
  
```

Imagen 9: "Para las peliculas se visualizara asi"

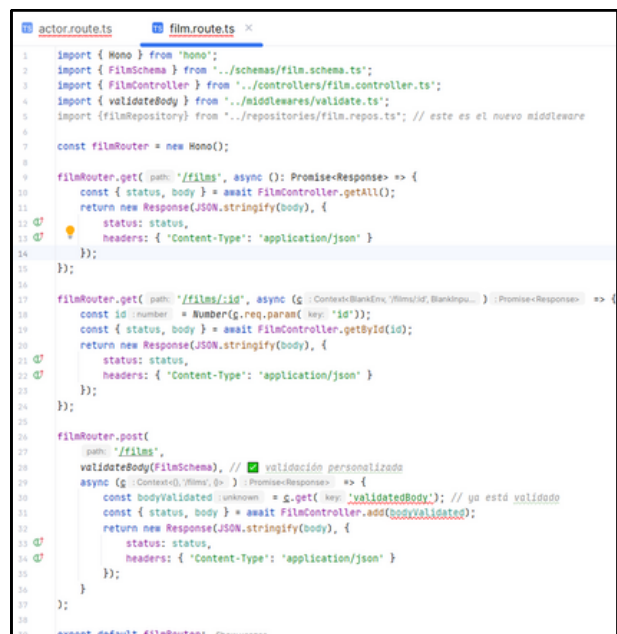
6. **Routers:** En esta carpeta define las rutas y asocia el controlador. Ademas se importan el schema dependiendo de que TS se esta escribiendo ya sea para el actor o para la pelicula (Revise imagen 10 y 11)



```

1 import { Hono } from 'hono';
2 import { actorSchema } from '../schemas/actor_schema.ts';
3 import { ActorController } from '../controllers/actor.controller.ts';
4 import { validateBody } from '../middlewares/validate.ts'; // este es el nuevo middleware
5
6 const actorRouter = new Hono();
7
8 actorRouter.get('/actors', async () => {
9   const { status, body } = await ActorController.getAll();
10   return new Response(JSON.stringify(body), {
11     status: status,
12     headers: { 'Content-Type': 'application/json' }
13   });
14 });
15
16 actorRouter.get('/actors/:id', async (c) => {
17   const { status, body } = await ActorController.getById(id);
18   return new Response(JSON.stringify(body), {
19     status: status,
20     headers: { 'Content-Type': 'application/json' }
21   });
22 });
23
24 actorRouter.post(
25   '/actors',
26   validateBody(actorSchema), // validación personalizada
27   async (c) => {
28     const { status, body } = await ActorController.add(body);
29     return new Response(JSON.stringify(body), {
30       status: status,
31       headers: { 'Content-Type': 'application/json' }
32     });
33   }
34 );
35
36 export default actorRouter;
  
```

Imagen 10: "Para los actores se visualizara asi"



```

1 import { Hono } from 'hono';
2 import { filmSchema } from '../schemas/film_schema.ts';
3 import { FilmController } from '../controllers/film.controller.ts';
4 import { validateBody } from '../middlewares/validate.ts'; // este es el nuevo middleware
5 import { filmRepository } from '../repositories/film.repo.ts'; // este es el nuevo middleware
6
7 const filmRouter = new Hono();
8
9 filmRouter.get('/films', async () => {
10   const { status, body } = await FilmController.getAll();
11   return new Response(JSON.stringify(body), {
12     status: status,
13     headers: { 'Content-Type': 'application/json' }
14   });
15 });
16
17 filmRouter.get('/films/:id', async (c) => {
18   const { status, body } = await FilmController.getById(id);
19   return new Response(JSON.stringify(body), {
20     status: status,
21     headers: { 'Content-Type': 'application/json' }
22   });
23 });
24
25 filmRouter.post(
26   '/films',
27   validateBody(filmSchema), // validación personalizada
28   async (c) => {
29     const { status, body } = await FilmController.add(body);
30     return new Response(JSON.stringify(body), {
31       status: status,
32       headers: { 'Content-Type': 'application/json' }
33     });
34   }
35 );
36
37 export default filmRouter;
  
```

Imagen 11: "Para las peliculas se visualizara asi"

7. **App:** En esta carpeta define las rutas para su ejecutable y se pueda hacer en el navegador. (Revise imagen 12)



```

1 import { Hono } from 'hono';
2 import actorRouter from './src/routes/actor.route';
3 import { errorHandler } from './src/middlewares/error_handler.ts';
4 import filmRouter from './src/routes/film.route.ts';
5
6 const app = new Hono();
7
8 app.use('*', errorHandler); // Aplica a todas las rutas
9 app.route('/', actorRouter);
10 app.route('/', filmRouter);
11 export default app;
  
```

Imagen 12: "Se aplica las rutas para actores y peliculas"

8. Ejecucion: En cmd de la carpeta donde viene todo, ejecutamos con “Bun” server.ts y en el navegador se puede ver por medio del localhost colocando el simbolo “\” y despues escribiendo ya sea “actors” o “films”

```
C:\Windows\System32\cmd.exe - bun server.ts
Microsoft Windows [Versión 10.0.19045.5854]
(c) Microsoft Corporation. Todos los derechos reservados.

.:Users\koda\Documents\javascript_postgres-main>bun server.ts
! Servidor escuchando en http://localhost:3000
```

```
localhost:3000/films
Dar formato al texto ☒

{
  "success": true,
  "message": "Peliculas recuperados correctamente",
  "data": [
    {
      "film_id": 133,
      "title": "Chamber Italian",
      "description": "A Fateful Reflection of a Moose And a Husband who must Overcome a Monkey in Nigeria",
      "release_year": 2006,
      "language_id": 1,
      "rental_duration": 7,
      "length": 117
    },
    {
      "film_id": 384,
      "title": "Grosse Monderful",
      "description": "A Epic Drama of a Cat And a Explorer who must Redeem a Moose in Australia",
      "release_year": 2006,
      "language_id": 1,
      "rental_duration": 5,
      "length": 49
    }
  ]
}
```

Imagen 13: “Visualizacion de peliculas”

```
Dar formato al texto ☒

{
  "success": true,
  "message": "Actores recuperados correctamente",
  "data": [
    {
      "actor_id": 1,
      "first_name": "Penelope",
      "last_name": "Guinness"
    },
    {
      "actor_id": 2
    }
  ]
}
```

Imagen 14: “Visualizacion de Actores”

CONCLUSION

La realización de este ejercicio de conexión a la base de datos PostgreSQL (DVD Rental) desde IntelliJ IDEA ha sido una excelente práctica que nos ha permitido consolidar conocimientos fundamentales en el manejo de bases de datos y entornos de desarrollo.

A través de este proceso, aprendimos a:

- Configurar y establecer una conexión eficiente entre IntelliJ IDEA y PostgreSQL.
- Utilizar el controlador JDBC para interactuar con la base de datos.
- Ejecutar consultas básicas y explorar la estructura de la base de datos DVD Rental.
- Gestionar posibles errores de conexión y validar la correcta configuración.

Esta práctica no solo reforzó nuestra comprensión de cómo funcionan las conexiones a bases de datos en un entorno de desarrollo integrado (IDE), sino que también nos demostró la importancia de seguir buenas prácticas, como el manejo adecuado de credenciales, el cierre correcto de conexiones y la organización del código.

En conclusión, ha sido un ejercicio muy valioso que nos prepara para proyectos más complejos, donde la interacción con bases de datos será esencial.

¡Seguiremos profundizando en estos conocimientos!