

**TECNOLÓGICO NACIONAL DE  
MÉXICO**



**INSTITUTO TECNOLÓGICO DEL  
VALLE DE OAXACA**

**INGENIERA EN TECNOLOGÍAS DE LA  
INFORMACIÓN Y COMUNICACIÓN**

**TALLER DE BASE DE  
DATOS**

**5.1 APLICACIÓN CON CONEXIÓN  
A BASE DE DATOS**

**Enlace de l Repositorio:**

**<https://github.com/Lupitha24/Conexion-a-Base-de-Datos.git>**

**Grupo: IT 4A**

**Nombre de la estudiante:  
Guadalupe Gómez Ibarra**

**Nombre del docente:  
Cardoso Jiménez Ambrosio**

**Num.Control:23920267**

**Fecha de elaboracion: 25 de Mayo del 2025**

**Ex Hacienda de Nazareno Santa CRuz Xoxocotlán  
Oaxaca.**





# INSTALACIÓN Y CONFIGURACIÓN DEL “BUN”

## Instalación y Configuración del Run ´Postgres “Bun”

### ¿QUÉ ES BUN?

Bun es un entorno de ejecución para JavaScript, parecido a Node.js, pero diseñado para ser más rápido y sencillo. Es un conjunto de herramientas que incluye un administrador de paquetes, un empaquetador y un ejecutor de pruebas, lo cual facilita el trabajo de los desarrolladores web.

### Comó instale el “Bun”

- Abrimos el siguiente enlace Nos dirigimos hacia la pagina <https://bun.sh/docs/installation>

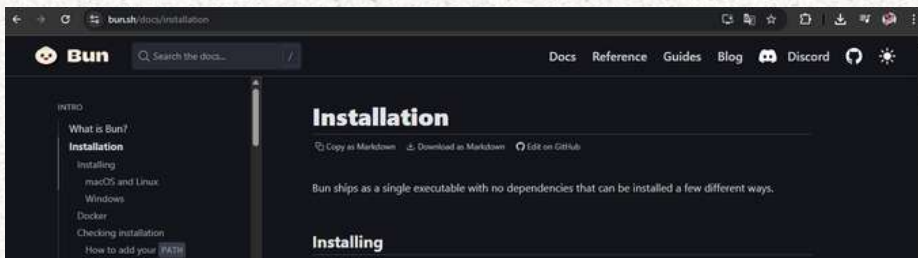


Imagen1.1

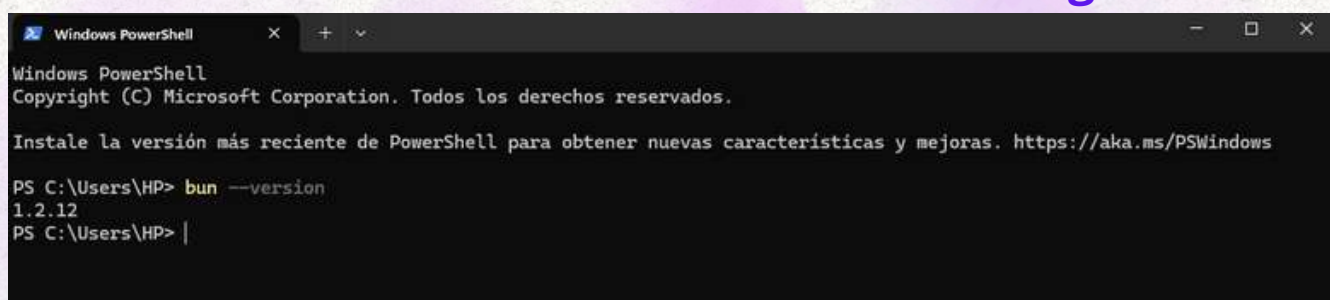
- Instalamos el de Windows porque ese es nuestro sistema operativo en seguida nos vamos a donde estan los códigos que nos proporciona para su instalación pero eso se hace desde powershell o CMD sea el caso.



Imagen1.2

- Vamos a copiar el código de PowerShell, luego abriremos una nueva ventana del mismo programa y lo pegaremos. (Al hacerlo, comenzará la descarga de “bun”; hay que esperar a que llegue al 100%). Una vez finalizada la instalación, para verificar si se instaló la versión más reciente, usaremos el siguiente comando:

Imagen1.3





# INSTALACIÓN Y CONFIGURACIÓN DEL “BUN”

## Instalación y Configuración del Run ´Postgres “Bun”

- Una vez que la descarga haya terminado, iremos a una carpeta específica para hacer la instalación. En este caso, se trata de una carpeta que contiene un proyecto ya preparado con la base de datos “DVDrental”. Nos dirigimos a la ubicación de esa carpeta, escribimos “cmd” en la barra de direcciones y se abrirá una ventana de la consola.

Imagen1.4

```
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.
```

- 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

```
C:\Windows\System32\cmd.exe x + v  
Microsoft Windows [Versión 10.0.22631.5189]  
(c) Microsoft Corporation. Todos los derechos reservados.  
C:\Users\HP\Downloads\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  
C:\Users\HP\Downloads\javascript_postgres>
```

Imagen1.5

- Después de crear el proyecto, ejecutamos el siguiente comando:
- `bun add drizzle-orm pg bun dotenv`
- Este comando sirve para instalar varias librerías que necesitaremos, incluyendo Drizzle ORM para manejar la base de datos, el conector de PostgreSQL y la herramienta para gestionar variables de entorno.

```
C:\Users\HP\Downloads\javascript_postgres>bun add drizzle-orm pg dotenv  
bun add v1.2.12 (32a47ae4)  
installed drizzle-orm@0.43.1  
installed pg@8.16.0  
installed dotenv@16.5.0  
[54.00ms] done  
C:\Users\HP\Downloads\javascript_postgres>
```

Imagen1.6

```
C:\Users\HP\Downloads\javascript_postgres>bun add drizzle orm pg bun dotenv  
bun add v1.2.12 (32a47ae4)  
installed drizzle@1.0.0  
installed orm@0.1  
installed pg@8.16.0  
installed bun@1.2.14 with binaries:  
- bun  
- bunx  
installed dotenv@16.5.0  
535 packages installed [125.84s]  
Blocked 5 postinstalls. Run 'bun pm untrusted' for details.  
C:\Users\HP\Downloads\javascript_postgres>
```

Imagen1.7



# INSTALACIÓN Y CONFIGURACIÓN DEL “BUN”

## Instalación y Configuración del Run´Postgres “Bun”

- Una vez terminado ese paso, en mi caso ya contaba con un proyecto proporcionado por mi profesor, el cual incluye un archivo llamado `server.ts` que se encarga de realizar la conexión local (`localhost`). Al intentar ejecutarlo, apareció un error, por lo que fue necesario instalar la librería `dotenv`.

```
C:\Windows\System32\cmd.exe X + -
Microsoft Windows [Versión 10.0.22631.5189]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\HP\Downloads\javascript_postgres-main>bun server.ts
3 | const { Pool } = pkg;
  | ^
error: Unexpected const
    at C:\Users\HP\Downloads\javascript_postgres-main\src\db\index.ts:3:1

Bun v1.2.12 (Windows x64)

C:\Users\HP\Downloads\javascript_postgres-main>bun server.ts
4 |
5 | const app = new Hono();
6 |
7 | app.use('*', errorHandler); // Aplica a todas las rutas
8 | app.route('/', actorRouter);
9 | app.route('/', filmRouter);
  | ^
ReferenceError: filmRouter is not defined
    at C:\Users\HP\Downloads\javascript_postgres-main\app.ts:9:26
    at loadAndEvaluateModule (2:1)

Bun v1.2.12 (Windows x64)

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

Imagen1.8

- Esta es la forma con la que se conecta hacia el `localhost`



# INSTALACIÓN Y CONFIGURACIÓN DEL “MAVEN”

- **Maven es una herramienta que ayuda a automatizar el proceso de construcción de software. Aunque se usa principalmente en proyectos Java, también puede aplicarse a otros lenguajes como C# o Scala. Su objetivo es facilitar la administración de proyectos, realizando automáticamente tareas como compilar el código, gestionar las librerías necesarias (dependencias), generar reportes, publicar versiones y organizar los lanzamientos del software.**

## Files

Maven is distributed in several formats for your convenience. Simply pick a ready-made binary distribution archive and follow the [installation instructions](#). Use a source archive if you intend to build Maven yourself.

In order to guard against corrupted downloads/installations, it is highly recommended to [verify the signature](#) of the release bundles against the public [KEYS](#) used by the Apache Maven developers.

	Link	Checksums	Signature
Binary tar.gz archive	<a href="#">apache-maven-3.9.9-bin.tar.gz</a>	<a href="#">apache-maven-3.9.9-bin.tar.gz.sha512</a>	<a href="#">apache-maven-3.9.9-bin.tar.gz.asc</a>
Binary zip archive	<a href="#">apache-maven-3.9.9-bin.zip</a>	<a href="#">apache-maven-3.9.9-bin.zip.sha512</a>	<a href="#">apache-maven-3.9.9-bin.zip.asc</a>

Imagen1.1

- **Primero vamos a entrar a la pagina de descarga en el cual descargaremos el archivo en binario en zip, Lo extraeremos en la carpeta que deseemos.**

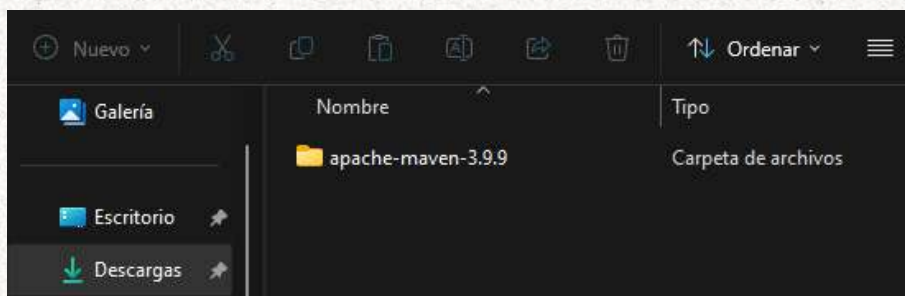


Imagen1.2

- **Lo vamos a copiar y pegar la carpeta extraida en el disco local.**

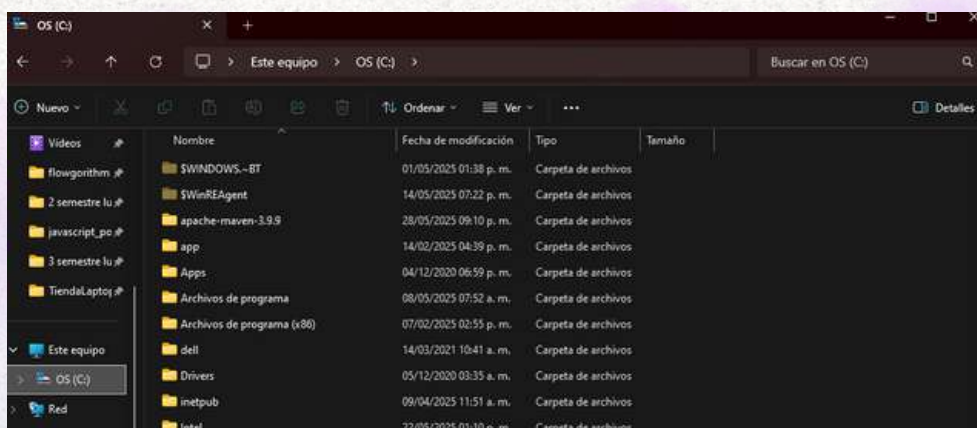


Imagen1.3



# INSTALACIÓN Y CONFIGURACIÓN DEL “MAVEN”

- Ahora vamos a trabajar con las variables de entorno. Crearemos una nueva variable a nivel del sistema y le asignaremos como valor la ruta de la carpeta que tenemos guardada en el disco local.

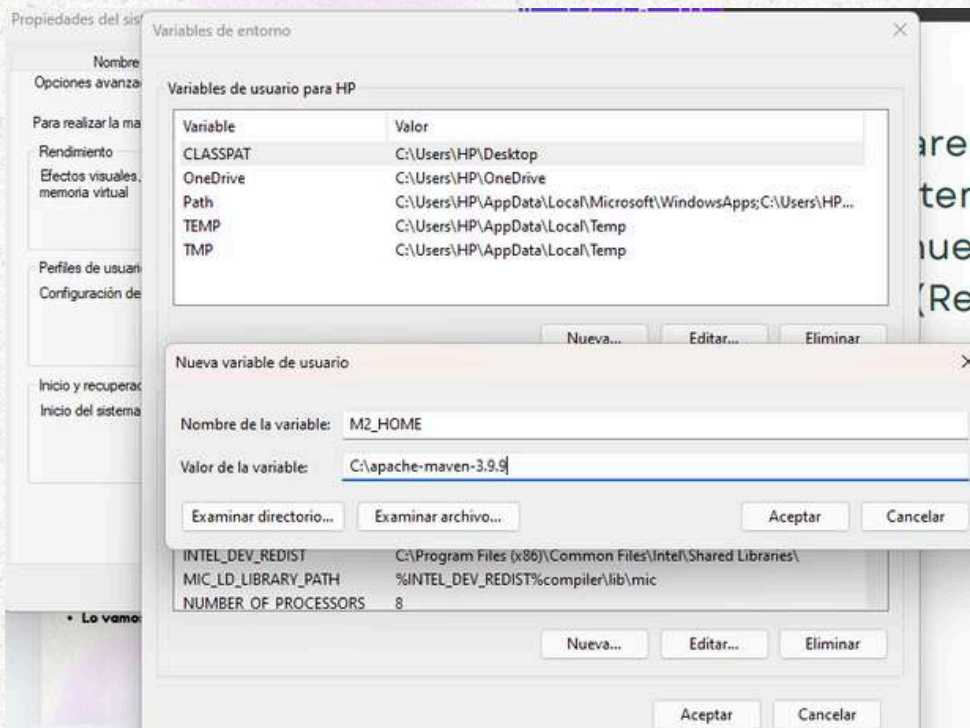


Imagen1.4

- Entraremos en PATH y crearemos una variable de entorno.

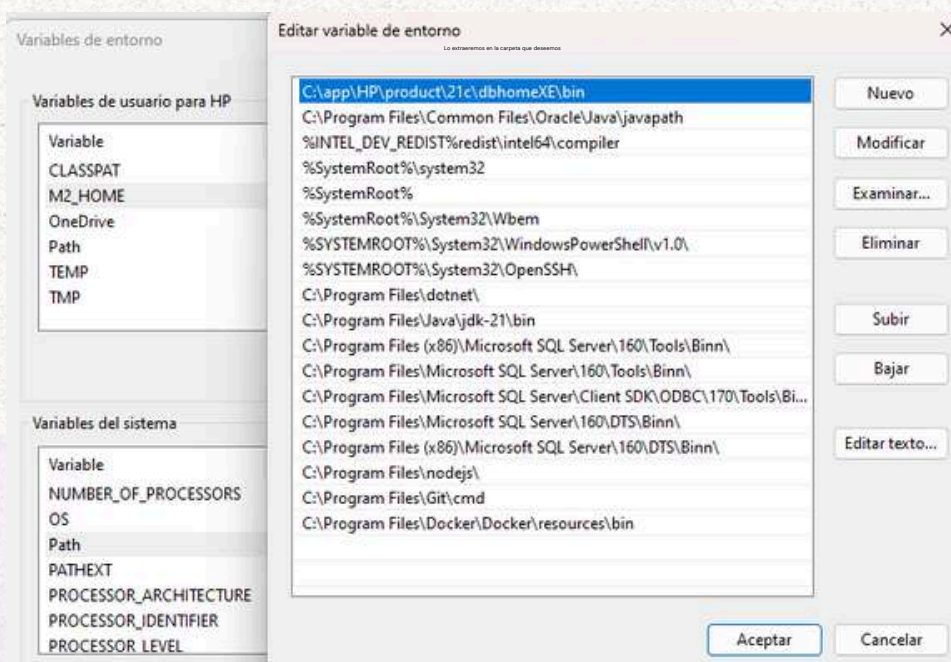


Imagen1.5



# INSTALACIÓN Y CONFIGURACIÓN DEL “MAVEN”

- **Vamos a abrir una terminal en Windows para comprobar que Maven se haya instalado correctamente. Para ello, escribimos el comando `mvn -v`, el cual nos mostrará la versión de Maven instalada y confirmará que la instalación fue exitosa.**

## CONFIGURACION CON EL IDE INTELLIJ IDEA

- **Ahora entraremos al software y realizaremos la siguientes configuraciones**

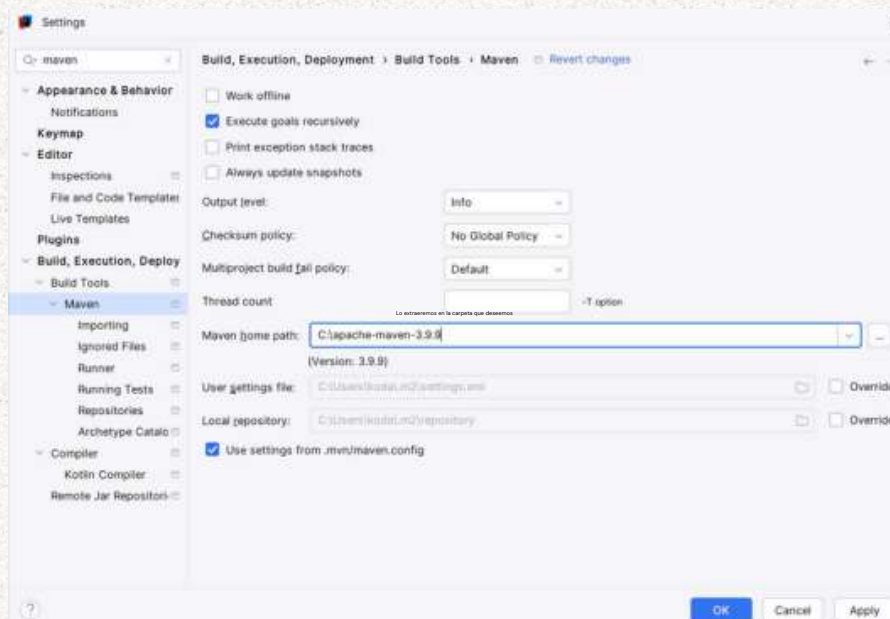


Imagen1.6



# CONEXIÓN DE BASE DE DATOS

## “JAVASCRIPT-POSTGRES”

Para la conexión con la base de datos, utilizaremos el programa IntelliJ IDEA. Este es un entorno de desarrollo integrado (IDE) desarrollado por JetBrains, diseñado especialmente para facilitar la creación de software, principalmente en los lenguajes Java y Kotlin.

Primero revisaremos la organización de las carpetas del proyecto:

- **SRC:** Es la carpeta principal que contiene todo el código fuente.
- **Controllers:** Se encarga de manejar la lógica relacionada con las peticiones y respuestas HTTP.
- **db:** Aquí se encuentra la configuración para conectar con la base de datos (como PostgreSQL).
- **Middlewares:** Son funciones que se ejecutan antes de llegar al controlador, como la validación o autenticación.
- **Repositories:** Esta capa permite interactuar directamente con la base de datos, ya sea escribiendo consultas o usando un ORM.
- **Routes:** Define los diferentes caminos (endpoints) de la API y los conecta con los controladores correspondientes.
- **Schemas:** Se utiliza para establecer cómo deben estar estructurados los datos.
- **Services:** Se encargan de procesar la información antes de almacenarla en la base de datos.
- **Utils:** Contiene funciones útiles que pueden ser usadas en distintas partes del proyecto.

**Archivos principales en la raíz del proyecto:**

- **app.ts:** Se encarga de configurar Express, incluyendo los middlewares generales y las rutas principales.
- **server.ts:** Es el archivo que arranca el servidor y lo pone a escuchar en el puerto 3000.
- **.env:** Contiene variables de entorno, como claves o configuraciones sensibles.
- **package.json:** Define las dependencias del proyecto y los comandos disponibles, por ejemplo, `npm run dev` para iniciar el entorno de desarrollo.

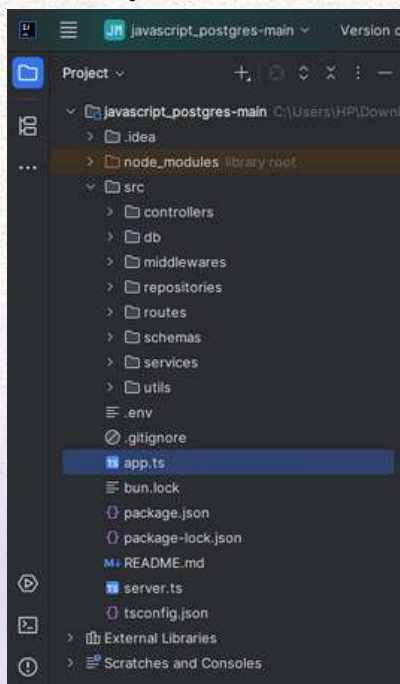


Imagen1.1



# CONEXIÓN DE BASE DE DATOS

## “JAVASCRIPT-POSTGRES”

### 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.

```
1 import { pgTable, serial, varchar, integer } from 'drizzle-orm/pg-core';
2
3 export const actors = PgTableWithColumns<({ name: 'actor', schema: ... }) => pgTable({ name: 'actor', { Show usages
4   actor_id: serial({ name: 'actor_id' }).primaryKey(),
5   first_name: varchar({ name: 'first_name', { length: 100 }},
6   last_name: varchar({ name: 'last_name', { length: 100 }},
7   });
8
9 export const films = PgTableWithColumns<({ name: 'film', schema: ... }) => pgTable({ name: 'film', { no usages
10   film_id: serial({ name: 'film_id' }).primaryKey(),
11   title: varchar({ name: 'title', { length: 255 }},
12   description: varchar({ name: 'description', { length: 500 }},
13   release_year: integer({ name: 'release_year' }),
14   language_id: integer({ name: 'language_id' }),
15   rental_duration: integer({ name: 'rental_duration' }),
16   length: integer({ name: 'length' }),
17   });
18
```

Imagen1.2

### 2. Repositories:

En esta carpeta se realizan las consultas SQL basadas en el ID que el usuario desea utilizar. En los archivos TypeScript correspondientes a actor y film, se importa el esquema previamente definido para acceder a la estructura de las tablas.

```
1 import { db } from '../db';
2 import { actors } from '../db/schema.ts';
3 import { eq } from 'drizzle-orm/sql/expressions/conditions';
4
5 export const ActorRepository = { Show usages
6   findAll: async () => Promise<({ actor_id: number; first_name: string }) => db.select().from(actors),
7   findById: async (id: number) => Promise<({ actor_id: number; first_name: string }) => {
8     const [actor] = await db
9       .select()
10      .from(actors)
11      .where(eq(actors.actor_id, id));
12     return actor;
13   },
14   add: async (data: { first_name: string; last_name: string }) => Promise<({ actor_id: number; first_name: string }) => {
15     db.insert(actors).values(data).returning(),
16   };
17
```

Imagen1.3

```
1 import { db } from '../db';
2 import { films } from '../db/schema.ts';
3 import { eq } from 'drizzle-orm/sql/expressions/conditions';
4
5 export const filmRepository = { Show usages
6   findAll: async () => Promise<({ film_id: number; title: string }) => db.select().from(films),
7   findById: async (id: number) => Promise<({ film_id: number; title: string }) => {
8     const [film] = await db
9       .select()
10      .from(films)
11      .where(eq(films.film_id, id));
12     return film;
13   },
14   add: async (data: { title: string; description: string; rental_duration: number }) => Promise<({ length: number }) => {
15     db.insert(films).values(data).returning(),
16   };
17
```

Imagen1.4



# CONEXIÓN DE BASE DE DATOS

## “JAVASCRIPT-POSTGRES”

### 3. Schemas:

En esta carpeta se establecen las validaciones para cada uno de los atributos definidos previamente. Por ejemplo, los nombres y apellidos se validan como cadenas de texto utilizando `.string`. Todo esto se organiza mediante objetos que representan la estructura de los datos.

A screenshot of a code editor showing the `actor_schema.ts` file. The code defines an `actorSchema` using `zod` for validation. It imports `z` from `'zod'` and exports `actorSchema` as a `ZodObject` with two properties: `first_name` and `last_name`, both of type `z.string().min(1)` with messages indicating they are required.

```
1 import { z } from 'zod';
2
3 export const actorSchema = ZodObject({first_name: ZodString; last_n... = z.object({ Show usages
4   first_name: z.string().min( minLength 1, message: "El nombre es obligatorio"),
5   last_name: z.string().min( minLength 1, message: "El apellido es obligatorio"),
6 });
```

Imagen1.5

A screenshot of a code editor showing the `film_schema.ts` file. The code defines a `FilmSchema` using `zod` for validation. It imports `z` from `'zod'` and exports `FilmSchema` as a `ZodObject` with two properties: `title` and `description`, both of type `z.string().min(10)` with messages indicating they are required and have a minimum length of 10.

```
1 import { z } from 'zod';
2
3 export const FilmSchema = ZodObject({title: ZodString; description... = z.object({ Show usages
4   title: z.string().min( minLength 10, message: "El título es obligatorio con la longitud mínima de 10"),
5   description: z.string().min( minLength 10, message: "La descripción es obligatorio"),
6 });
```

Imagen1.6

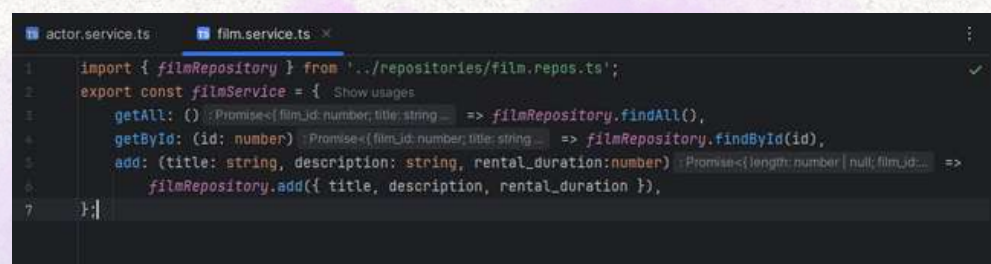
### 4. Services:

En esta carpeta se define cómo deben presentarse los resultados, por ejemplo, en formato JSON, y se ajusta la forma en que la información será mostrada al usuario final.

A screenshot of a code editor showing the `actor.service.ts` file. The code defines an `ActorService` that uses `ActorRepository` to perform database operations. It imports `ActorRepository` from `'../repositories/actor.repos.ts'` and exports `ActorService` with methods `getAll`, `getById`, and `add`.

```
1 import { ActorRepository } from '../repositories/actor.repos.ts';
2
3 export const ActorService = { Show usages
4   getAll: () => Promise<{ actor_id: number; first_name: s... => ActorRepository.findAll(),
5   getById: (id: number) => Promise<{ actor_id: number; first_name: s... => ActorRepository.findById(id),
6   add: (first_name: string, last_name: string) => Promise<{ actor_id: number; first_name: s... =>
7     ActorRepository.add({ first_name, last_name }),
8 };
9
```

Imagen1.7

A screenshot of a code editor showing the `film.service.ts` file. The code defines a `FilmService` that uses `FilmRepository` to perform database operations. It imports `FilmRepository` from `'../repositories/film.repos.ts'` and exports `FilmService` with methods `getAll`, `getById`, and `add`.

```
1 import { FilmRepository } from '../repositories/film.repos.ts';
2
3 export const filmService = { Show usages
4   getAll: () => Promise<{ film_id: number; title: string... => filmRepository.findAll(),
5   getById: (id: number) => Promise<{ film_id: number; title: string... => filmRepository.findById(id),
6   add: (title: string, description: string, rental_duration: number) => Promise<{ length: number | null; film_id... =>
7     filmRepository.add({ title, description, rental_duration }),
8 };
9
```

Imagen1.8



# CONEXIÓN DE BASE DE DATOS

## “JAVASCRIPT-POSTGRES”

### 5. Controllers:

En esta carpeta se implementan las funciones para obtener y agregar información relacionada con actores y películas. También se encarga de conectar con la base de datos para guardar los datos cuando el usuario desea añadir nuevos registros.

```
actor.controller.ts  film.controller.ts
1
2 C:\Users\HP\Downloads\javascript_postgres-main\src\controllers\actor.controller.ts
3
4
5 export const ActorController = { Show usages
6   getAll: async () : Promise<{ status: HTTP_STATUS; body: { su... } => {
7     try {
8       const actors : {actor_id: number; first_name: string} ... = await ActorService.getAll();
9       return HttpResponse.ok(actors, message: "Actores recuperados correctamente");
10    } catch (error) {
11      return HttpResponse.error( message: "Error al recuperar los actores");
12    }
13  },
14
15   getById: async (id: number) : Promise<{ status: HTTP_STATUS; body: { su... } => {
16     try {
17       const actor : {actor_id: number; first_name: string} ... = await ActorService.getById(id);
18       if (!actor) {
19         return HttpResponse.notFound( message: "Actor no encontrado");
20       }
21       return HttpResponse.ok([actor], message: "Actor encontrado");
22     } catch (error) {
23       return HttpResponse.error( message: "Error al recuperar el actor");
24     }
25   },
26
27   add: async (body: { first_name: string; last_name: string }) : Promise<{ status: number; body: { success... } => {
```

Imagen1.9

Imagen1.10

```
actor.controller.ts  film.controller.ts
1 import {HttpResponse} from "../utils/http_reponse.ts";
2 import {filmService} from "../services/film.service.ts";
3
4 export const FilmController = { Show usages
5   getAll: async () : Promise<{ status: HTTP_STATUS; body: { su... } => {
6     try {
7       const actors : {film_id: number; title: string} null, ... = await filmService.getAll();
8       return HttpResponse.ok(actors, message: "Películas recuperados correctamente");
9     } catch (error) {
10      return HttpResponse.error( message: "Error al recuperar las películas");
11    }
12  },
13
14   getById: async (id: number) : Promise<{ status: HTTP_STATUS; body: { su... } => {
15     try {
16       const actor : {film_id: number; title: string} null, ... = await filmService.getById(id);
17       if (!actor) {
18         return HttpResponse.notFound( message: "Película no encontrado");
19       }
20       return HttpResponse.ok([actor], message: "Película encontrado");
21     } catch (error) {
22       return HttpResponse.error( message: "Error al recuperar el actor");
23     }
24   },
25 }
```



# CONEXIÓN DE BASE DE DATOS

## “JAVASCRIPT-POSTGRES”

### 6. Routers:

En esta carpeta se establecen las rutas del proyecto y se vinculan con sus respectivos controladores. Además, se importan los esquemas correspondientes según el archivo TypeScript que se esté utilizando, ya sea para actores o películas.

```
actor.route.ts x film.route.ts x
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( path: '/actors', async (): Promise<Response> => {
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( path: '/actors/:id', async (c: Context<BlankEnv, '/actors/:id', BlankInput>) : Promise<Response> => {
17   const id : number = Number(c.req.param( key: 'id' ));
18   const { status, body } = await ActorController.getById(id);
19   return new Response(JSON.stringify(body), {
20     status: status,
21     headers: { 'Content-Type': 'application/json' }
22   });
23 });
24
```

Imagen1.11

```
actor.route.ts x film.route.ts x
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';
5 import { filmRepository } from '../repositories/film.repos.ts'; // este es el nuevo middleware
6
7 const filmRouter = new Hono();
8
9 filmRouter.get( path: '/films', async (): Promise<Response> => {
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( path: '/films/:id', async (c: Context<BlankEnv, '/films/:id', BlankInput>) : Promise<Response> => {
18   const id : number = Number(c.req.param( key: 'id' ));
19   const { status, body } = await FilmController.getById(id);
20   return new Response(JSON.stringify(body), {
21     status: status,
22     headers: { 'Content-Type': 'application/json' }
23   });
24 });
25
26 filmRouter.post(
27   path: '/films',
28   async (c: Context<BlankEnv, '/films', BlankInput>) : Promise<Response> => {
29     const { status, body } = await FilmController.create(filmRepository);
30     return new Response(JSON.stringify(body), {
31       status: status,
32       headers: { 'Content-Type': 'application/json' }
33     });
34   });
35
```

Imagen1.12



# CONEXIÓN DE BASE DE DATOS

## “JAVASCRIPT-POSTGRES”

### 7. App:

En esta carpeta define las rutas para su ejecutable y se pueda hacer en el navegador.

```
app.ts
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(path: '*', errorHandler); // Aplica a todas las rutas
9 app.route(path: '/', actorRouter);
10 app.route(path: '/', filmRouter);
11 export default app;
```

Imagen1.13

### 8. Ejecución:

Desde la terminal, ubicados en la carpeta principal del proyecto, se ejecuta el archivo server.ts usando el comando bun server.ts. Luego, en el navegador, se puede acceder al servidor local escribiendo localhost\ seguido de la ruta deseada, como actors o films, para visualizar los datos correspondientes.

```
C:\Windows\System32\cmd.exe
Microsoft Windows [Versión 10.0.22631.5189]
(c) Microsoft Corporation. Todos los derechos reservados.

C:\Users\HP\Downloads\javascript_postgres-main>bun server.ts
3 | const { Pool } = pkg;
  | ~~~~~
error: Unexpected const
    at C:\Users\HP\Downloads\javascript_postgres-main\src\db\index.ts:3:1

Bun v1.2.12 (Windows x64)

C:\Users\HP\Downloads\javascript_postgres-main>bun server.ts
4 |
5 | const app = new Hono();
6 |
7 | app.use(path: '*', errorHandler); // Aplica a todas las rutas
8 | app.route(path: '/', actorRouter);
9 | app.route(path: '/', filmRouter);
  | ~~~~~
ReferenceError: filmRouter is not defined
    at C:\Users\HP\Downloads\javascript_postgres-main\app.ts:9:26
    at loadAndEvaluateModule (2:1)

Bun v1.2.12 (Windows x64)

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

Imagen1.14

Imagen1.16

```
localhost:3000/actors
Dar formato al texto
{
  "success": true,
  "message": "Actores recuperados correctamente",
  "data": [
    {
      "actor_id": 1,
      "first_name": "Penelope",
      "last_name": "Guinness"
    },
    {
      "actor_id": 2,
      "first_name": "Nick",
      "last_name": "Mahlberg"
    },
    {
      "actor_id": 3,
      "first_name": "Ed",
      "last_name": "Chase"
    },
    {
      "actor_id": 4,
      "first_name": "Jennifer",
      "last_name": "Davis"
    },
    {
      "actor_id": 5,
      "first_name": "Johnny",
      "last_name": "tollibrigida"
    },
    {
      "actor_id": 6,
      "first_name": "Bette",
      "last_name": "Nicholson"
    },
    {
      "actor_id": 7,
      "first_name": "Grace",
      "last_name": "Mostel"
    }
  ]
}
```

Imagen1.15

```
localhost:3000/films
Dar formato al texto
{
  "success": true,
  "message": "Películas 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": 394,
      "title": "Grosse Wonderful",
      "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
    },
    {
      "film_id": 8,
      "title": "Airport Pollock",
      "description": "A Epic Tale of a Moose And a Girl who must Confront a Monkey in Ancient India",
      "release_year": 2006,
      "language_id": 1,
      "rental_duration": 6,
      "length": 54
    },
    {
      "film_id": 98,
      "title": "Bright Encounters",
      "description": "A Fateful Yarn of a Lumberjack And a Feminist who must Conquer a Student in A Jet Boat",
      "release_year": 2006,

```





# CONCLUSIÓN

**La experiencia de conectar la base de datos PostgreSQL (DVD Rental) desde IntelliJ IDEA ha representado una oportunidad enriquecedora para afianzar conceptos esenciales en el manejo de bases de datos dentro de un entorno de desarrollo profesional. Este ejercicio nos permitió familiarizarnos con la configuración del entorno, la utilización del controlador JDBC, la ejecución de consultas y la navegación por la estructura de la base de datos.**

**Además, enfrentarnos a posibles errores de conexión nos brindó una visión más realista del trabajo con bases de datos, destacando la importancia de una configuración adecuada y el uso responsable de recursos como las credenciales y las conexiones.**

**En suma, esta práctica no solo fortaleció nuestras habilidades técnicas, sino que también nos preparó para futuros desarrollos donde la gestión de bases de datos será un componente clave.**

