



Instituto Tecnológico de Chetumal



Carrera: Ingeniería en Sistemas Computacionales

Proyecto: Creación de API REST con token y creación de reporte

Alumnos: Andres Adrian Martin Canto

Sergio Eduardo Andrade Delgadillo

Carlos Daniel Aguilar Poot

Materia: Desarrollo Web 2

Maestro: Esquivel Pat Agustín

Fecha de entrega: 24 de mayo del 2024

Introducción

Este es un proyecto para la materia de Desarrollo Web 2 la cual tiene como objetivo crear una **API REST** la cual este construida con la estructura **MVC (Modelo, Vista, Controlador)**, en nuestro caso hicimos una API REST la cual es de un cine, la cual tiene 4 modelos, los cuales son: Usuarios, Géneros, Películas y Actores.

A estos 4 modelos leharemos las operaciones **CRUD** y la creación de reportes para cada modelo.

De igual manera consumiremos la API en JavaScript con Fetch y creación de reporteo con la librería, **fpdf** la cual usara los modelos: Actores, Películas, Usuarios y Géneros.

Creación de base de datos.

Primero empezaremos con la tabla actores la cual tiene 5 columnas que son:

- ID
- Nombre
- Nacionalidad
- Edad
- Pelicula_ID

La cuales el primer campo es de tipo int que es incrementable, luego sigue nombre la cual es varchar de 255 así como el campo de nacionalidad, luego sigue los dos últimos campos que son int que es edad y Pelicula_ID.

Esta tabla es para saber más a detalle de un actor y a cuál película pertenecen.

```
CREATE TABLE `actores` (
  `id` int(11) NOT NULL,
  `nombre` varchar(255) DEFAULT NULL,
  `nacionalidad` varchar(255) DEFAULT NULL,
  `edad` int(11) DEFAULT NULL,
  `pelicula_id` int(11) DEFAULT NULL
```

De ahí sigue la tabla géneros, la cual tiene 3 campos que son:

- ID
- Nombre
- Descripción

El cual el primer campo es tipo int que es incrementable, luego sigue el nombre que es varchar, el cual nos sirve para distinguir que genero es, luego sigue descripción la cual es tipo text que nos sirve para saber más a detalle el tipo de género, en caso de acción podría ser que vaya en descripción: películas que hay explosiones o disparos.

Esta tabla nos va a servir para saber cuáles géneros hay y asignárselo a las películas.

```
CREATE TABLE `generos` (
  `id` int(11) NOT NULL,
  `nombre` varchar(255) DEFAULT NULL,
  `descripcion` text DEFAULT NULL
```

La siguiente tabla son películas, que tienen los siguientes campos:

- ID
- Titulo
- Director
- Año estreno
- Genero ID

El primer campo es el ID la cual es un tipo int incrementable, luego sigue el título de la película que está en varchar 255, luego el director de la película que esta igual en varchar 255, luego sigue el año del estreno que esta es un int, luego sigue el genero_id el cual sabrá a que genero pertenece.

Esta tabla servirá para saber cuándo saldrá una película y se estrena.

```
CREATE TABLE `peliculas` (
  `id` int(11) NOT NULL,
  `titulo` varchar(255) DEFAULT NULL,
  `director` varchar(255) DEFAULT NULL,
  `anio_estreno` int(11) DEFAULT NULL,
  `genero_id` int(11) DEFAULT NULL
```

Luego sigue la tabla usuarios el cual tiene los siguientes campos:

- ID
- Nombre
- Correo Electronico
- Contraseña

- Token

Estos campos tienen como el primer campo de tipo int e incrementable, luego sigue el nombre del usuario que tendrá un varchar de 255, así como correo electrónico y contraseña, luego el token que tendrá el token con el cual hará para permitir hacer las peticiones.

Esta tabla nos ayudara a crear usuarios para que le dé un token de validación y encripte la contraseña del usuario.

```
CREATE TABLE `usuarios` (
  `id` int(11) NOT NULL,
  `nombre` varchar(255) NOT NULL,
  `correo_electronico` varchar(255) NOT NULL,
  `contrasenia` varchar(255) NOT NULL,
  `token` varchar(100) NOT NULL
```

Archivo .htaccess

Como primero en nuestro archivo .htaccess tenemos que habilitar el direccionamiento y para poder poner las reglas de direccionamiento, la cual se habilita con la siguiente línea de código:

```
RewriteEngine On
```

Luego sigue otra línea la cual es para no permitir la navegación de las carpetas:

```
Options All -Indexes
```

Después siguen nuestras reglas las cuales todas van a redireccionarnos al archivo index.php:

- El primero es:

```
RewriteRule ^usuarios/(login|registro) index.php?model=$1&accion=$2 [QSA]
```

El cual nos ayudara a registrar un usuario así como loguearse y obtener su token.

En esta instrucción obtenemos en el primer paréntesis el modelo lo capturamos luego puede ir login o registros y lo capturara en acción.

- Como segunda regla es:

La captura de un pdf la cual permite cualquier modelo pueda capturar un pdf basado al modelo al cual va acceder:

```
RewriteRule ^([a-z]+)/pdf$ index.php?model=$1&pdf=$2 [QSA]
```

La regla nos dice que pueden ir todo el alfabeto numérico pero que sean minúsculas por lo mínimo debe ir una letra, luego sigue la palabra pdf para poder redireccionarlo en el índice.

- Como tercera regla es:

La cual va a permitir, eliminar, actualizar, modificar y consultar todos o un valor en específico con su ID, como en la regla anterior pedimos como mínimo una letra minúscula como mínimo luego decimos que puede ir un número o es opcional.

```
RewriteRule ^([a-z]+)(/\d+)?$ index.php?model=$1&id=$3 [QSA]
```

Index.php

En este archivo lo que hacemos es definir que haremos basado a lo que nos envíen en la ruta y retornar una vista de formato **JSON** el cual nos permitirá saber el estado y el resultado de nuestra operación que le enviamos en la ruta.

Como primero importaremos las clases que nos ayudaran hacer las operaciones, primero importaremos la clase Conexiones la cual nos ayudara hacer los Querys a la base de datos.

```
require_once 'BD/ConexionBD.php';
```

De ahí seguirá la clase **ExceptionApi**, la cual nos ayudará a poder hacer una excepción cuando una petición este incorrecta o le falte el token o repetición de correos.

```
require_once 'View/ExceptionApi.php';
```

Luego sigue la clase **VistaJson** la cual nos permitirá retornar el resultado de la petición.

```
require_once 'View/VistaJson.php';
```

Importamos la clase **controllerUsuario** la cual nos permitirá llamar hacer la autenticación.

Así como el modelo.

```
require_once 'Controller/controllerUsuarios.php';
require_once 'Models/Usuario.php';
```

Estatus error

Luego siguen el estatus de error, las cuales en la siguiente tabla mostrara todos los tipos de errores.

Estatus Error	Descripción de error
403	Este error será lanzado cuando la ruta no sea válida.
405	Este error se enviará cuando el verbo http no sea válido.
505	El error se dispara cuando ocurra un error en la base de datos como la conexión o el Query esta mal escrito o no existe la tabla.
400	El error se ejecutará cuando no le envíes el token.
410	Se ejecutará cuando tu token no sea válido.

Estos son los estatus errores los cuales nos ayudaran a saber que error ocurre.

Rutas permitidas

En nuestro caso para permitir las rutas permitidas lo agregamos en un arreglo asociativo el cual tiene las rutas permitidas en la llave del arreglo asociativo,

Luego la clase de un controlador para que haga lo que corresponda:

```
$rutas = [
    'actores' => 'controllerActores',
    'generos' => 'controllerGeneros',
    'peliculas' => 'controllerPeliculas',
    'usuarios' => 'controllerUsuarios'
];
```

De ahí obtenemos el modelo para saber cuál se va a llamar:

Preguntamos si el modelo que envió existe, si no existe marco un error:

```
if (!array_key_exists($Model, $rutas)) {
    throw new ExcepcionApi([estado: ESTADO_RUTA_NO_VALIDA,
```

De lo contrario si existe, entonces importamos el controlador correspondiente:

```
// Importamos la clase dependiendo de cual se cumpla en la ruta.
require_once 'Controller/' . $rutas[$Model] . '.php';
```

Luego creamos una instancia la cual nos servirá para llamar al método correspondiente para hacer lo que me enviaron en la petición.

```
// Creación de un objeto con código de ayuda
$ objetoController = new $rutas[$Model];
// ...
```

De ahí verificamos que nos hayan enviado el ID si es null o indefinido entonces le asigna null o si esta vacío.

Creamos un array asociativo el cual nos va a permitir retorna la respuesta en un formato el cual va a dar estado o cuerpo.

Luego verificamos que me haya enviado pdf.

```
$id = $_GET['id'] ?? null;
if (empty ($id)) $id = null;
$respuesta = "";
// Retorno mi respuesta en un array
$arrayDevolver = [
    'estado' => '',
    'cuerpo' => '',
];
// #####
$pdf = $_GET['pdf'] ?? null;
```

Verificamos que pdf que no sea null, si no es entonces validamos si el verbo http es get para poder obtener el pdf del método que me hayan enviado, antes validamos el token que me hayan enviado.

```
if (!is_null($pdf)) {
    if ($metodo === 'get'){
        Usuario::autenticar();
        $objetoController->pdf();
    }else{
        throw new ExpcionApi('estado: METHOD_NOT_ALLOWED',
    }
```

Si no me enviaron el pdf entonces haremos una operación CRUD del método correspondiente.

```

} else {
    // #####
    switch ($metodo) {
        case 'get':
            // Valido que me haya enviado el token
            Usuario::autenticar();
            // Mando a llamar al método index para que haga lo siguiente.
            $respuesta = $objetoController->index($id);
            $vista->estado = 200;
            break;
        case 'post':
            $respuesta = $objetoController->store();
            // Mandamos que se ha creado correctamente.
            $vista->estado = 201;
            break;
        case 'put':
            // Valido que me haya enviado el token
            Usuario::autenticar();
            // Mando a llamar al método edit para modificar dependiendo.
            $respuesta = $objetoController->edit($id);
            $vista->estado = 200;
            break;
        case 'delete':
            // Valido que me haya enviado el token
            Usuario::autenticar();
            // Mando a llamar al método delete para eliminar dependiendo.
            $respuesta = $objetoController->delete($id);
            $vista->estado = 200;
            break;
        default:
            throw new ExpcionApi(estado: METHOD_NOT_ALLOWED, mensaje: "URL no encontrada");
            break;
    }
}

```

Como se pueden ver solo 4 verbos de http soporta nuestra API, la cual son:

- GET
- POST

- DELETE
- PUT

Las cuales cada verbo tiene asignado hacer algo como los siguiente:

VERBO HTTP GET	Descripción
http://localhost/ApiProject/actores/pdf	Esta ruta con el verbo http, es para conseguir los actores con sus películas en un pdf, está en una tabla
http://localhost/ApiProject/peliculas/pdf	La ruta nos sirve para obtener las películas con su género, en formato pdf.
http://localhost/ApiProject/usuarios/pdf	Obtiene la cantidad de usuarios, con su correo electrónico y sus contraseñas encriptadas.
http://localhost/ApiProject/generos/pdf	Obtiene en formato pdf los géneros que hay para las películas.
http://localhost/ApiProject/usuarios/8	Obtiene en formato JSON el estatus y un usuario por su ID, en este caso el ID 8.
http://localhost/ApiProject/usuarios	Trae todos los usuarios en formato JSON.
http://localhost/ApiProject/actores/1	Trae un actor por su ID en formato JSON.
http://localhost/ApiProject/actores	Trae todos los actores en formato JSON.
http://localhost/ApiProject/peliculas/1	Obtiene una película por su ID.
http://localhost/ApiProject/peliculas	Obtiene todas las películas en formato JSON.
http://localhost/ApiProject/generos/1	Obtiene un género por su ID.
http://localhost/ApiProject/generos	Obtiene todos los géneros.
VERBO POST HTTP	Descripción

http://localhost/ApiProject/usuarios/registro	Esta petición crea un usuario el cual obtiene los datos por formato JSON.
http://localhost/ApiProject/actores	Crear un actor en la base de datos.
http://localhost/ApiProject/peliculas	Crea una película en la base de datos.
http://localhost/ApiProject/generos	Crea un género en la base de datos.
Verbo Delete Http	Descripción
http://localhost/ApiProject/usuarios/1	Esta ruta elimina un usuario por su ID.
http://localhost/ApiProject/generos/1	Elimina un género por su ID.
http://localhost/ApiProject/peliculas/1	Elimina una película por su ID.
http://localhost/ApiProject/actores/1	Elimina un actor por su ID.
Verbo Put Http	Descripción
http://localhost/ApiProject/usuarios/9	Actualiza un usuario por su ID y obtiene los valores a modificar por JSON.
http://localhost/ApiProject/peliculas/2	Actualiza un usuario por su ID.
http://localhost/ApiProject/generos/3	Actualiza un género por su ID.
http://localhost/ApiProject/actores/5	Actualiza un actor por su ID.

Controlador Actor

Los controladores tienen el mismo método y hace lo mismo, en este caso el index sirve para obtener los datos por su id u obtener todos los datos:

```
public function index($id):array
{
    // Si el id no es null entonces retornara un dato.
    if (!is_null($id)){
        // Retorno el resultado.
        return Actor::getOne($id);
        // De lo contrario retornara todos.
    }else{
        // Retorno todos los datos.
        return Actor::getAll();
    }
}
```

Resultado postman:

The screenshot shows the Postman interface with a successful API call. The URL is `http://localhost/ApiProject/actores`. The response body is a JSON object containing the status code and two actor objects.

```
1 {
2     "estado": 200,
3     "cuerpo": [
4         {
5             "id": 47,
6             "nombre": "Penélope Cruz",
7             "nacionalidad": "Española",
8             "edad": 48,
9             "pelicula_id": 12
10        },
11        {
12            "id": 48,
13            "nombre": "Tom Hanks"
```

The screenshot shows the Postman interface with a successful API call. The URL is `http://localhost/ApiProject/actores/47`. The Headers tab shows an `Authorization` header with a value of `$2y$10$EYQLtBHgSwpOYF4mYy0EtOOsvsL0i..`. The Body tab displays the JSON response:

```
1 {  
2     "estado": 200,  
3     "cuerpo": {  
4         "id": 47,  
5         "nombre": "Penélope Cruz",  
6         "nacionalidad": "Española",  
7         "edad": 48,  
8         "pelicula_id": 12  
9     }  
10 }
```

Luego sigue el método `Store` el cual nos va a servir para insertar un dato a la base de datos.

```
public function store()  
{  
    $cuerpo = file_get_contents('php://input');  
    $params = json_decode($cuerpo);  
    return Actor::insertOne($params);  
}
```

Resultados postman:

The screenshot shows the Postman interface with a POST request to 'http://localhost/ApiProject/actores'. The request body is a JSON object with fields: nombre ('Miguel Moises Habilo'), nacionalidad ('Mexico'), edad (80), and pelicula_id (12). The response status is 201 Created, and the message is 'Se ha creado el actor'.

```
1 {
2   "nombre": "Miguel Moises Habilo",
3   "nacionalidad": "Mexico",
4   "edad": 80,
5   "pelicula_id": 12
6 }
```

```
1 {
2   "estado": 201,
3   "cuerpo": "Se ha creado el actor"
4 }
```

Después sigue el método edit, el cual nos permitirá modificar un dato por su id.

```
public function edit($id)
{
    // Si el id no es null entonces mandar actualizar un valor.
    if (!is_null($id)){
        $cuerpo = file_get_contents( filename: 'php://input');
        $params = json_decode($cuerpo);
        return Actor::update($params, $id);
    }else{
        throw new ExcepcionApi( estado: 400, mensaje: "Se requiere el id");
    }
}
```

Resultado postman:

The screenshot shows the Postman interface. On the left, the 'pace' sidebar lists various API endpoints under 'ApiProject'. The 'Actores' section contains a 'PUT getActor' endpoint. The main workspace shows a 'PUT' request to 'http://localhost/ApiProject/actores/47'. The 'Body' tab displays a JSON payload:

```

1 {
2   "nombre": "12",
3   "nacionalidad": "d22a",
4   "edad": 48,
5   "pelicula_id": 13
6 }

```

The response at the bottom indicates a '200 OK' status with a response time of 31 ms and a size of 324 B. The response body is:

```

1 {
2   "estado": 200,
3   "cuerpo": "Se ha modificado el actor"
4 }

```

Como penúltimo método que es el delete nos va a servir para eliminar un dato por su ID.

```

no usages

public function delete($id): string
{
    if (!is_null($id)){
        return Actor::destroy($id);
    }else{
        throw new ExcepcionApi(estado: 400, mensaje: "Se requiere el id");
    }
}

```

Resultado postman:

The screenshot shows the Postman application interface. At the top, it displays a collection named "DEL deleteActor". Below the header, the URL is set to "http://localhost/ApiProject/actores/deleteActor". The method is selected as "DELETE". A pink "Send" button is visible on the right. The "Headers" tab is active, showing a table with one row where "Authorization" is set to a hashed value starting with "\$2y\$10\$". The "Body" tab is also present, showing a JSON response with status code 200 OK and message "Se ha borrado el actor".

DEL deleteActor

HTTP ApiProject / Actores / **deleteActor**

Save Share

DELETE http://localhost/ApiProject/actores/47 Send

Params Auth Headers (7) Body Scripts Settings Cookies

Headers 6 hidden

	Key	Value	D...	...	Bulk Edit	Presets
<input checked="" type="checkbox"/>	Authorization	\$2y\$10\$EYQLtBHgSwpOYF4...				
	Key	Value		Description		

Body 200 OK 34 ms 321 B Save as example ...

Pretty Raw Preview Visualize JSON

```
1 {  
2   "estado": 200,  
3   "cuerpo": "Se ha borrado el actor"  
4 }
```

Luego sigue el método PDF el cual me obtendrá un PDF dependiendo.

```
~~~~~
public function pdf(): void
{
    // Creo la instancia
    $pdf = new PDF();
    $pdf->titulo( titulo: "Lista de actores");
    // Creo la pagina.
    $pdf->AddPage();
    $data = Actor::pdf();
    $pdf->SetFont( family: 'Arial', style: '', size: 14);
    $pdf->SetWidths(array(10, 50, 50, 20, 50));
    $pdf->SetAligns(array("C", "C", "C", "C", "C"));
    // Le asigno la cabeceras
    $pdf->Row(array("No", 'Nombre', 'Nacionalidad', "Edad", "Película"));
    // Contador para que me diga numero 1 tal y asi.
    $contador = 1;
    $pdf->SetAligns(array("C", "L", "L", "C", "L"));
    foreach ($data as $row) {
        $pdf->Row(array($contador++, $row['nombre'], $row['nacionalidad'], $row['edad'], $row['pelicula']));
    }
    // Muestro el PDF.
    $pdf->Output();
}
~~~~~
```

Postman resultado:

The screenshot shows a Postman request for the endpoint `http://localhost/ApiProject/actores/pdf`. The Headers tab is selected, showing an `Authorization` header with the value `$2y$10$EYQLtBHgSwpOYF4mYy0EtOOsvsL0i...`. The response body displays a table titled "Lista de actores" with the following data:

No	Nombre	Nacionalidad	Edad	Película
1	dada	da	48	Forrest Gump
2	Emma Stone	Estadounidense	33	La La Land
3	Javier Bardem	Español	53	No Country for Old Men
4	Scarlett Johansson	Estadounidense	38	Lost in Translation
5	Keanu Reeves	Canadiense	59	The Matrix
6	Al Pacino	Estadounidense	84	The Godfather
7	Christian Bale	Britanico	50	The Dark Knight
8	Miguel Moises Habilia	Mexico	80	Vicky Cristina Barcelona

Controlador Películas.

Método que trae una película por su id o todas las películas.

```
public function index($id){
    // Si el id no es null entonces retornara un dato.
    if (!is_null($id)){
        // Retorno el resultado.
        return Peliculas::getOne($id);
        // De lo contrario retornara todos.
    }else{
        // Retorno todos los datos.
        return Peliculas::getAll();
    }
}
```

Resultado postman:

GET getMovies

HTTP ApiProject / Peliculas / **getMovies**

Save Share

GET http://localhost/ApiProject/peliculas Send

Params Authorization Headers (7) Body Scripts Settings Cookies

Headers (6 hidden)

Key	Value	Description	...	Bulk Edit	Presets
<input checked="" type="checkbox"/> Authorization	\$2y\$10\$EYQLtBHgSwpOYF4mYy0EtOOsvsLoi..				
Key	Value	Description			

Body Cookies Headers (7) Test Results

Pretty Raw Preview Visualize JSON

Status: 200 OK Time: 10 ms Size: 1.8 KB Save as example

```
1 {
2     "estado": 200,
3     "cuerpo": [
4         {
5             "id": 12,
6             "titulo": "Vicky Cristina Barcelona",
7             "director": "Woody Allen",
8             "anio_estreno": 2008,
9             "genero_id": 4
10        },
11    ],
12    "total": 1
13 }
```

The screenshot shows the Postman interface with a successful API call. The URL is `http://localhost/ApiProject/peliculas/12`. The response body is a JSON object:

```
1 {
2     "estado": 200,
3     "cuerpo": {
4         "id": 12,
5         "titulo": "Vicky Cristina Barcelona",
6         "director": "Woody Allen",
7         "anio_estreno": 2008,
8         "genero_id": 4
9     }
10 }
```

Método que me va a servir para insertar películas.

```
/** 
 * TODO: Método que me va a servir para insertar películas.
 * @return string
 * @throws ExpcionApi
 */
public function store(): string{
    $cuerpo = file_get_contents('php://input');
    $params = json_decode($cuerpo);
    return Peliculas::insertOne($params);
}
```

Resultado Postman:

The screenshot shows the Postman interface with a successful API call. The URL is `http://localhost/ApiProject/peliculas`. The request method is `POST`. The response status is `201 Created`, and the response body is:

```
1 {  
2   "estado": 201,  
3   "cuerpo": "Se ha creado la pelicula"  
4 }
```

Método para modificar una película.

```
public function edit($id): string{  
    if (!is_null($id)){  
        $cuerpo = file_get_contents('php://input');  
        $params = json_decode($cuerpo);  
        return Peliculas::update($params, $id);  
    }else{  
        throw new ExcepcionApi(400, "Se requiere el id");  
    }  
}
```

Resultado postman:

The screenshot shows the Postman interface for a PUT request to 'registerMovies'. The URL is set to `http://localhost/ApiProject/peliculas/12`. The 'Body' tab is selected, showing a JSON payload:

```
1 {
2     "titulo": "Barbie",
3     "director": "Kevin Martinez",
4     "anio_estreno": 2007,
5     "genero_id": 4
6 }
```

Below the body, the response is shown in 'Pretty' format:

```
1 {
2     "estado": 200,
3     "cuerpo": "Se ha modificado la película"
4 }
```

Método para eliminar una película.

```
public function delete($id): string{
    if (!is_null($id)){
        return Peliculas::destroy($id);
    }else{
        throw new ExpcionApi(400, "Se requiere el id");
    }
}
```

Resultado postman:

The screenshot shows a Postman interface with a DELETE request to `http://localhost/ApiProject/peliculas/13`. The Headers tab includes an `Authorization` header with the value `$2y$10$EYQLtBhgSwpOYF4mY0EtOOsvsL0iHR5wxkjpN9fwZ0iHD2/TW6py`. The response body is a JSON object:

```

1 {
2     "estado": 200,
3     "cuerpo": "Se ha borrado la pelicula"
4 }

```

Este método crea el pdf de películas y lo muestra.

```

public function pdf(): void
{
    // Creo la instancia
    $pdf = new PDF();
    $pdf->titulo("Lista de peliculas");
    // Creo la pagina.
    $pdf->AddPage();
    $data = Peliculas::pdf();
    $pdf->SetFont('Arial', ' ', 14);
    $pdf->SetWidths(array(10, 50, 60, 30,30));
    $pdf->SetAligns(array("C", "C", "C", "C", "C"));
    $pdf->Row(array ("No", 'Titulo', 'Director',utf8_decode("Año estreno"),"Genero"));
    $contador = 1;
    $pdf->SetAligns(array("C", "C", "C", "C","C"));
    foreach ($data as $row) {
        $pdf->Row(array($contador++, utf8_decode($row['titulo']), utf8_decode($row['director']),$row[
    }
    // Muestro el PDF.
    $pdf->Output();
}

```

Controlador Usuarios.

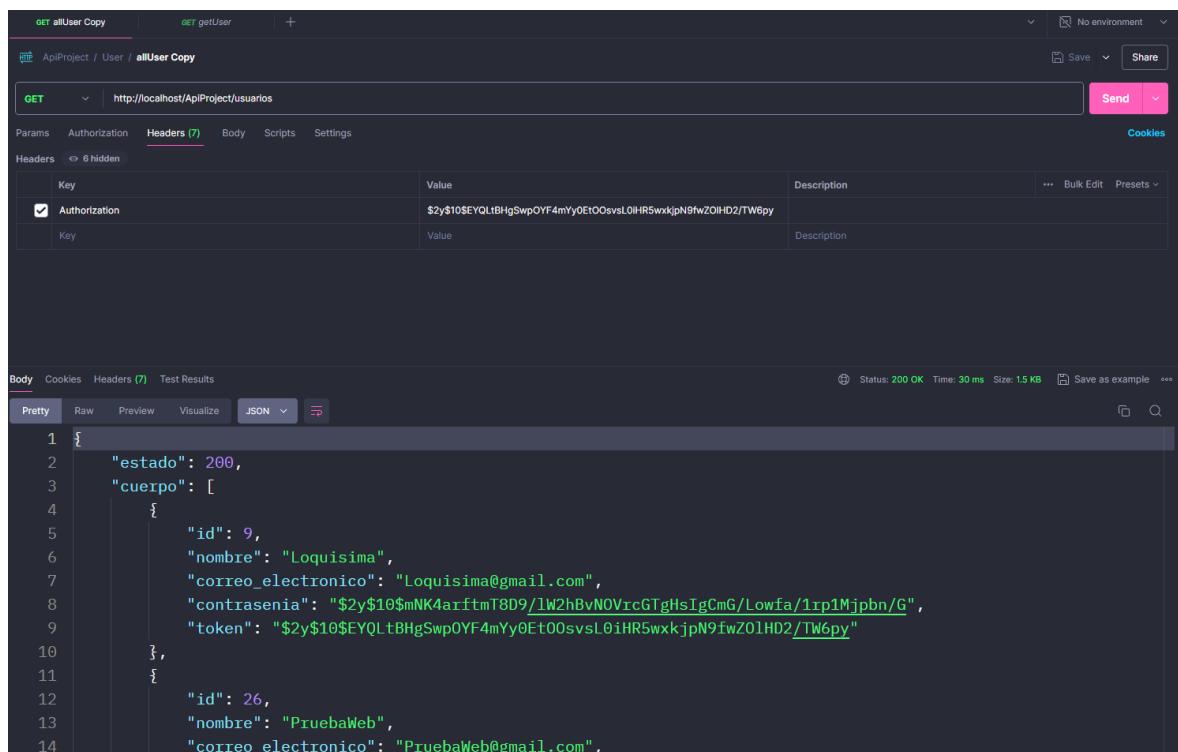
Método que trae un usuario o todos los usuarios.

```

public function index($id): array
{
    // Si el id no es null entonces retornara un dato.
    if (!is_null($id)) {
        // Retorno el resultado.
        return Usuario::getOne($id);
        // De lo contrario retornara todos.
    } else {
        // Retorno todos los datos.
        return Usuario::getAll();
    }
}

```

Resultado postman:



The screenshot shows the Postman interface with the following details:

- Request URL:** `http://localhost/ApiProject/usuarios`
- Method:** GET
- Headers:**

Key	Value	Description
Authorization	<code>\$2y\$10\$EYQLtBhgSwpOYF4mY0Et0OsVsL0iHR5wxkjN9fwZ0lHD2/TW6py</code>	
Key	Value	Description
- Body (Pretty JSON):**

```

1 {
2     "estado": 200,
3     "cuerpo": [
4         {
5             "id": 9,
6             "nombre": "Loquisima",
7             "correo_electronico": "Loquisima@gmail.com",
8             "contrasenia": "$2y$10$mNK4aiftmT8D9/lW2hBvNOVrcGTgHsIgCmG/Lowfa/1rp1Mjpbn/G",
9             "token": "$2y$10$EYQLtBhgSwpOYF4mY0Et0OsVsL0iHR5wxkjN9fwZ0lHD2/TW6py"
10        },
11        {
12            "id": 26,
13            "nombre": "PruebaWeb",
14            "correo_electronico": "PruebaWeb@gmail.com",

```
- Status:** 200 OK
- Time:** 30 ms
- Size:** 1.5 KB

Sirve para loguearse o para crear un usuario.

```
{
    $accion = $_GET['accion'] ?? "";
    if ($accion === 'login') {
        return $this->login();
    } else if ($accion === 'registro') {
        return $this->register();
    } else {
        throw new ExpcionApi(METHOD_NOT_ALLOWED, "URL no valida:");
    }
}
```

Método para loguearse

```
public function login(): string
{
    $cuerpo = file_get_contents('php://input');
    $params = json_decode($cuerpo);
    return Usuario::login($params);
}
```

Resultado postman:

The screenshot shows a Postman interface with the following details:

- Request URL:** http://localhost/ApiProject/usuarios/login
- Method:** POST
- Body:** JSON (Pretty)
- Request Body Content:**

```
1 {
2     "correo_electronico": "sergi2@gmail.com",
3     "contrasenia": "2024"
4 }
```
- Response Status:** 201 Created
- Response Time:** 60 ms
- Response Size:** 318 B
- Response Content:**

```
1 {
2     "estado": 201,
3     "cuerpo": "Se ha logueado"
4 }
```

Método para agregar usuario.

```
public function register()
{
    $cuerpo = file_get_contents('php://input');
    $params = json_decode($cuerpo);
    return Usuario::insertOne($params);
}
```

Resultado postman:

The screenshot shows a Postman interface with the following details:

- Request URL:** http://localhost/ApiProject/usuarios/registro
- Method:** POST
- Body:** JSON (Pretty)
- Request Body Content:**

```
1 {
2     "nombre": "Sergi",
3     "correo_electronico": "sergi2@gmail.com",
4     "contrasenia": "2024"
5 }
```
- Response Status:** 201 Created
- Response Time:** 118 ms
- Response Size:** 429 B
- Response Content:**

```
1 {
2     "estado": 201,
3     "cuerpo": {
4         "correo": "sergi2@gmail.com",
5         "token": "$2y$10$8W0JIqfVEzNeMuqVdHmToOoIvU3akmE/bp59jCi3ECKpYv9ef2w12"
6     }
7 }
```

Método para actualizar un usuario.

```
public function edit($id): string
{
    if (!is_null($id)) {
        $cuerpo = file_get_contents('php://input');
        $params = json_decode($cuerpo);
        return Usuario::update($params, $id);
    } else {
        throw new ExcepcionApi(400, "Se requiere el id");
    }
}
```

Resultado postman:

The screenshot shows the Postman interface with a successful API call. The request details are as follows:

- Method: PUT
- URL: <http://localhost/ApiProject/usuarios/9>
- Headers:
 - Authorization: \$2y\$10\$EYQLBHgSwpOYF4mYy0EtOOsvsL0IHR5wxkjpN9fwZOHD2/TW6py
- Body (Pretty):

```
1 {
2     "estado": 200,
3     "cuerpo": "Se ha modificado el usuario"
4 }
```

The response details are:

- Status: 200 OK
- Time: 119 ms
- Size: 325 B

Método para crear el pdf.

```
public function pdf(): void
{
    // Creo la instancia
    $pdf = new PDF();
    // Creación de titulo
    $pdf->titulo("Lista de usuarios");
    // Creo la pagina.
    $pdf->AddPage();
    // Obtengo los usuarios
    $data = Usuario::pdf();
    // Le asigno la letra, tipo y numero letra.
    $pdf->SetFont('Arial', '', 14);
    // Le asigno el tamaño que tendrá cada columna
    $pdf->SetWidths(array(10, 50, 80, 40));
    // Digo que estarán centrados
    $pdf->SetAligns(array("C", "C", "C", "C"));
    // Le asigno la cabeceras
    $pdf->Row(array("No", 'Nombre', 'Correo electrónico','Contraseña'));
    // Contador para que me diga numero 1 tal y asi.
    $contador = 1;
    // Le digo como estarán posicionados
    $pdf->SetAligns(array("C", "L", "L", "C"));

    $pdf->Row(array("No", 'Nombre', 'Correo electrónico','Contraseña'));
    // Contador para que me diga numero 1 tal y asi.
    $contador = 1;
    // Le digo como estarán posicionados
    $pdf->SetAligns(array("C", "L", "L", "C"));
    // For each para enviarle los valores al método Row y los acomode bien
    foreach ($data as $row) {
        // Agrego una fila con los valores correspondientes
        $pdf->Row(array($contador++, $row['nombre'], $row['correo_electronico'],$row['contrasenia']));
    }
    // Muestro el PDF.
    $pdf->Output();
}
```

Resultado postman:

The screenshot shows the Postman interface with the following details:

- URL:** http://localhost/ApiProject/usuarios/pdf
- Method:** GET
- Headers:** Authorization (with value \$2y\$10\$xF/vxeE5KfylLmXCDGFbiOKEn4FgcTk7tuXJmhEj9wrtDS181YY2W) and Key.
- Body:** Not present.
- Cookies:** Not present.
- Headers (Details):** Shows 6 hidden headers.
- Test Results:** Status: 200 OK, Time: 34 ms, Size: 2.82 KB.

The response body contains the following JSON data:

```
[{"id": 1, "Nombre": "loca", "Correo electrónico": "loca@gmail.com", "Contraseña": "$2y$10$05XIC/Z1tDAH19sq9uRfRO7OnpQzqOxXgoN'2ZXy6f6ceT7Laq"}, {"id": 2, "Nombre": "PruebaWeb", "Correo electrónico": "PruebaWeb@gmail.com", "Contraseña": "$2y$10$asymbyMU54FrdlXruXO8900hquDfOTOCbs8oA6gqz3N/Aa9"}, {"id": 3, "Nombre": "andres", "Correo electrónico": "andres@gmail.com", "Contraseña": "$2y$10$uuuIkVNos1LvhacrRL1V.Zl0oiBpvW8vs3q3nBqBqBcfhpv0a"}, {"id": 4, "Nombre": "Sergi", "Correo electrónico": "Sergi@gmail.com", "Contraseña": "$2y$10$V1SiP4s00VUPzUUCGgjmuYOO6n4PMvSOHn6d65pQJns1u8"}, {"id": 5, "Nombre": "Sergi", "Correo electrónico": "sergi2@gmail.com", "Contraseña": "$2y$10$upsops wSBdyd6JHjwQzQ6G 3qrlUALFnhs51Kn3Rk7Y8gv6wonZx"}]
```

Método para eliminar un usuario.

```
/*
 * TODO: Método para eliminar un usuario.
 * @throws ExpcionApi
 */
public function delete($id): string
{
    if (!is_null($id)) {
        return Usuario::destroy($id);
    } else {
        throw new ExpcionApi(400, "Se requiere el id");
    }
}
```

Resultado postman:

The screenshot shows a Postman interface with the following details:

- Method:** DELETE
- URL:** http://localhost/ApiProject/usuarios/26
- Headers:** Authorization (with value \$2y\$10\$xF/vxeE5KfyJmXCDGFbI0KEn4FgCTk7tuXJmhEjBqwtDS18YY2W)
- Body:** JSON (Pretty) - Response content:

```
1 {  
2     "estado": 200,  
3     "cuerpo": "Se ha borrado el usuario"  
4 }
```
- Status:** 200 OK
- Time:** 17 ms
- Size:** 323 B

Controlador géneros

Método que me retorna un género o todos los géneros.

```

/**
 * TODO: Método que me retorna un genero o todos los géneros.
 * @throws ExcepcionApi
 */
public function index($id):array
{
    // Si no es null el id entonces retorna un genero
    if(!is_null($id)){
        // Retorna un genero
        return Genero::getOne($id);
        // De lo contrario retorna todos los géneros.
    }else{
        // Return todos los géneros.
        return Genero::getAll();
    }
}

```

Resultado postman:

The screenshot shows the Postman interface with the following details:

- Request URL:** http://localhost/ApiProject/generos/allGeneros
- Method:** GET
- Headers:**
 - Authorization: \$2y\$10\$xFvxeE5KfyLmXCDGFbiOKEn4FgcTk7tuXJmhEJ8qwtDS181YY2W
- Body (Pretty):**

```

1 {
2     "estado": 200,
3     "cuerpo": [
4         {
5             "id": 4,
6             "nombre": "Drama",
7             "descripcion": "Pelicula donde las mujeres lloran"
8         },
9         {
10            "id": 5,
11            "nombre": "Comedia",
12            "descripcion": "Peliculas diseñadas para hacer reír al público con situaciones humorísticas."
13        }
]

```
- Status:** 200 OK
- Time:** 11 ms
- Size:** 1.73 KB

The screenshot shows a Postman interface with a GET request to `http://localhost/ApiProject/generos/4`. The Headers tab is selected, showing an `Authorization` header with the value `$2y$10$xFvxeE5KfylJmXCDGFbIOKEn4FgcTk7tuXJmhE9qwtDS18YY2W`. The Body tab shows a JSON response:

```

1 {
2     "estado": 200,
3     "cuerpo": {
4         "id": 4,
5         "nombre": "Drama",
6         "descripcion": "Pelicula donde las mujeres lloran"
7     }
8

```

Status: 200 OK Time: 10 ms Size: 408 B Save as example

Método que crear un nuevo género.

```

/***
 * TODO: Método que crear un nuevo genero.
 * @throws ExpcionApi
 */
no usages
public function store()
{
    $cuerpo = file_get_contents( filename: 'php://input' );
    $params = json_decode($cuerpo);
    return Genero::insertOne($params);
}

```

Resultado postman:

The screenshot shows a Postman interface with the following details:

- Request URL:** http://localhost/ApiProject/generos
- Method:** POST
- Body:** JSON (Pretty)
- Body Content:**

```
1 {
2   "nombre": "Feminista",
3   "descripcion": "Machistas numero dos"
4 }
```
- Response Status:** 201 Created
- Response Time:** 20 ms
- Response Size:** 328 B
- Response Body (Pretty JSON):**

```
1 {
2   "estado": 201,
3   "cuerpo": "Se ha creado el genero"
4 }
```

Método para editar un género.

```
/***
 * TODO: Método para editar un genero.
 * @throws ExpcionApi
 */
no usages
public function edit($id)
{
    if (!is_null($id)){
        $cuerpo = file_get_contents( filename: 'php://input');
        $params = json_decode($cuerpo);
        return Genero::update($params, $id);
    }else{
        throw new ExpcionApi( estado: 400, mensaje: "Se requiere el id");
    }
}
```

Resultado postman:

The screenshot shows a Postman interface for an API project. The URL is `http://localhost/ApiProject/generos/5`. The method is set to `PUT`. The `Body` tab is selected, showing the following JSON payload:

```
1 {
2     "nombre": "Drama",
3     "descripcion": "Pelicula donde las mujeres lloran"
4 }
```

Below the request, the response is displayed. The status is `200 OK`, time is `33 ms`, and size is `325 B`. The response body is:

```
1 {
2     "estado": 200,
3     "cuerpo": "Se ha modificado el genero"
4 }
```

Método para eliminar un género.

```
/** 
 * TODO: Método para eliminar un genero
 * @param $id
 * @return string
 * @throws ExpcionApi
 */
no usages
public function delete($id)
{
    if (!is_null($id)){
        return Genero::destroy($id);
    }else{
        throw new ExpcionApi(estado: 400, mensaje: "Se requiere el id");
    }
}
```

Resultado postman:

The screenshot shows a Postman interface with the following details:

- Method:** DELETE
- URL:** http://localhost/apiProject/generos/5
- Headers:** Authorization (with value \$2y\$10\$xF/vxeE5KfyLmXCDGFbI0En4FgcTk7tuXJmhEjqwtDS181YY2W)
- Body:** JSON response:


```

1 {
2   "estado": 200,
3   "cuerpo": "Se ha borrado el genero"
4 }
```
- Status:** 200 OK
- Time:** 17 ms
- Size:** 322 B

Método para crear el pdf de géneros

```

/**
 * TODO: Método para crear el pdf de géneros
 * @return void
 * @throws ExcepcionApi
 */
public function pdf(): void
{
    // Creo la instancia
    $pdf = new PDF();
    $pdf->titulo(utf8_decode(string: "Lista de géneros"));
    $pdf->AddPage();
    $data = Genero::pdf();
    // Le asigno la letra, tipo y numero letra.
    $pdf->SetFont(family: 'Arial', style: '', size: 14);
    // Le asigno el tamaño que tendrá cada columna
    $pdf->SetWidths(array(10, 80, 80));
    $pdf->SetAligns(array("C", "C", "C"));
    // Le asigno la cabeceras
    $pdf->Row(array("No", 'Nombre', utf8_decode(string: 'Descripción')));
    $contador = 1;
    $pdf->SetAligns(array("C", "L", "L"));
    foreach ($data as $row) {
        $pdf->Row(array($contador++, utf8_decode($row['nombre']), utf8_decode($row['descripcion'])));
    }
    // Muestro el PDF.
    $pdf->Output();
}

```

Resultado postman:

The screenshot shows a Postman request for a GET endpoint at `http://localhost/ApiProject/generos/pdf`. The Headers tab includes an `Authorization` header with the value `$2y$10$xF/vxeE5KfylmXCDGFbiOKEn4FgcTk7tuXJmhE@qwtDS18YY2W`. The response body displays a PDF titled "Lista de géneros" which contains the following table:

No	Nombre	Descripción
1	Drama	Película donde las mujeres lloran.
2	Drama	Película donde las mujeres lloran.
3	Ciencia Ficción	Películas que exploran temas futuristas y tecnológicos.
4	Terror	Películas que buscan asustar al público mediante el uso de suspense y elementos sobrenaturales.
5	Romance	Películas que se centran en las relaciones amorosas y los sentimientos románticos.
6	Aventura	Películas que narran viajes emocionantes y exploraciones.
7	Animación	Películas que utilizan principalmente dibujos animados, gráficos por computadora u otras técnicas de animación.
8	Feminista	Machistas numero dos

Modelo

Dentro del modelo tenemos un arreglo el cual tiene todas las columnas de la base de datos.

Modelos actores.

```
protected static $columnasTabla = [
    'nombre',
    'nacionalidad',
    'edad',
    'pelicula_id'
];
```

Luego siguen los métodos:

- GETALL: la cual obtiene todos los datos de la tabla correspondiente.

```
public static function getAll(): array
{
    try {
        // Traigo todos los datos de mi tabla
        $comando = "SELECT * FROM " . self::$table;
        // Preparar sentencia
        $sentencia = ConexionBD::obtenerInstancia()->obtenerBD()->prepare($comando);
        // Ejecutar el query.
        $sentencia->execute();
        // Retorno el resultado.
        // Si quiero traer varios datos uso fetchAll
        $respuesta = $sentencia->fetchAll(PDO::FETCH_ASSOC);
        if(!$respuesta){
            return [];
        }
        return $respuesta;
    } catch (PDOException $e) {
        throw new ExcepcionApi(estado: self::ERROR_DB, $e->getMessage());
    }
}
```

- GETONE: trae un dato por su id, dependiendo el modelo.

```
usage
public static function getOne($id): array
{
    try {
        $comando = "SELECT * FROM " . self::$table . " WHERE id =?";
        // Preparar sentencia
        $sentencia = ConexionBD::obtenerInstancia()->obtenerBD()->prepare(
            $comando);
        $sentencia->bindParam( param: 1, &var: $id, type: PDO::PARAM_INT );
        // Ejecutar el query.
        $sentencia->execute();
        // Retorno el resultado.
        $respuesta = $sentencia->fetch( mode: PDO::FETCH_ASSOC );
        // En caso de que no traiga datos.
        if (!$respuesta) {
            return [];
        }
        // Si tiene datos trae
        return $respuesta;
    } catch (PDOException $e) {
        throw new ExpcionApi( estado: self::ERROR_DB, $e->getMessage());
    }
}
```

- INSERTONE: Inserta un dato, basándose en el modelo.

```
public static function insertOne($params): string
{
    // Validación de token.
    Usuario::autenticar();
    // Método que valida que envié los parámetros correctos.
    self::validacionParams($params);
    // Hace la inserción de un nuevo actor.
    try {
        // Obtenemos el pdo para poder hacer el insert.
        $pdo = ConexionBD::obtenerInstancia()->obtenerBD();
        // Sentencia INSERT
        $comando = "INSERT INTO " . self::$table . " (" .
            self::$columnasTabla[0] . "," .
            self::$columnasTabla[1] . "," .
            self::$columnasTabla[2] . "," .
            self::$columnasTabla[3] . ")" .
            " VALUES(?, ?, ?, ?)";
        // Mandamos a validar si la sintaxis esta bien escrita.
        $sentencia = $pdo->prepare($comando);
        // Le asignamos los parametros que se enviaran a la vista.
        $sentencia->bindParam(1, &var: $params->nombre, type: PDO::PARAM_STR);
        $sentencia->bindParam(2, &var: $params->nacionalidad, type: PDO::PARAM_STR);
        $sentencia->bindParam(3, &var: $params->edad, type: PDO::PARAM_INT);
        $sentencia->bindParam(4, &var: $params->pelicula_id, type: PDO::PARAM_INT);
        // Ejecutamos el script
    }
}
```

- UPDATE: Actualiza un dato por id, dependiendo el modelo.

```
public static function update($params, $id): string
{
    // Método que valida que envié los parámetros correctos.
    self::validacionParams($params);
    // Procedimiento de actualizar un dato.
    try {
        // Obtenemos el pdo para poder hacer el insert.
        $pdo = ConexionBD::obtenerInstancia()->obtenerBD();
        $comando = "UPDATE " . self::$table . " SET " . self::$columnasTabla[1] .
            self::$columnasTabla[2] . " =? , " . self::$columnasTabla[3] . " =?
            " . self::$columnasTabla[4] . " =? , " . self::$columnasTabla[5] . " =?";
        $sentencia = $pdo->prepare($comando);
        $sentencia->bindParam(1, &var: $params->nombre, type: PDO::PARAM_STR);
        $sentencia->bindParam(2, &var: $params->nacionalidad, type: PDO::PARAM_STR);
        $sentencia->bindParam(3, &var: $params->edad, type: PDO::PARAM_INT);
        $sentencia->bindParam(4, &var: $params->pelicula_id, type: PDO::PARAM_INT);
        $sentencia->bindParam(5, &var: $id, type: PDO::PARAM_INT);
        $sentencia->execute();
        if ($sentencia->rowCount() > 0) {
            return self::ESTADO_MODIFICACO_EXITOSA;
        } else {
            return self::ESTADO_MODIFICACO_FALLIDA;
        }
    } catch (PDOException $e) {
```

- DELETE: Elimina un dato por su id

```

    public static function destroy($id): string
    {
        try{
            $pdo = ConexionBD::obtenerInstancia()->obtenerBD();
            $comando = "DELETE FROM " . self::$table . " WHERE id =?";
            $sentencia = $pdo->prepare($comando);
            $sentencia->bindParam(param: 1, &var: $id, type: PDO::PARAM_INT);
            $sentencia->execute();
            if ($sentencia->rowCount() > 0) {
                return self::ESTADO_DELETE_EXITOSA;
            }else{
                return self::ESTADO_DELETE_FALLIDA;
            }
        }catch (PDOException $e){
            throw new ExcepcionApi(estado: self::ERROR_DB, $e->getMessage());
        }
    }
}

```

- PDF: Trae los datos para hacer el reporte.

```

public static function pdf(): array
{
    try {
        // Traigo todos los datos de mi tabla nombre, Actores.nacionalidad,
        $comando = "SELECT " . self::$columnasTabla[0] . ", ". self::$columnasTabla[1] .
        " , peliculas.titulo FROM " .self::$table ." JOIN peliculas ON "
        // Preparar sentencia
        $sentencia = ConexionBD::obtenerInstancia()->obtenerBD()->prepare($comando);
        // Ejecutar el query.
        $sentencia->execute();
        // Retorno el resultado.
        // Si quiero traer varios datos uso fetchAll
        $respuesta = $sentencia->fetchAll(mode: PDO::FETCH_ASSOC);
        if(!$respuesta){
            return [];
        }
        return $respuesta;
    } catch (PDOException $e) {
        throw new ExcepcionApi(estado: self::ERROR_DB, $e->getMessage());
    }
}

```

- Validación de parámetros, dependiendo el modelo valida que le envíen todos los parámetros.

```
public static function validacionParams($params): void
{
    // Verificar si las columnasDeLaTabla las enviaron como parámetros.
    foreach ($self::$columnasTabla as $columna) {
        // Si no esta definido dentro del arreglo que me enviaron entonces ma
        if (!isset($params->$columna)) {
            // Paso el arreglo a una cadena
            $mensajeError = "Las columnas son las siguientes: " . implode( se
            throw new ExpcionApi(estado: 400, $mensajeError);
        }
    }
}
```

Modelo películas.

Este es el método que obtiene una película por su id.

```
public static function getOne($id): array
{
    try {
        // Traigo todos los datos de mi tabla
        $comando = "SELECT * FROM " . self::$table . " WHERE id =?";
        // Preparar sentencia
        $sentencia = ConexionBD::obtenerInstancia()->obtenerBD()->prepare($comando);
        $sentencia->bindParam(1, $id, PDO::PARAM_INT);
        // Ejecutar el query.
        $sentencia->execute();
        // Retorno el resultado.
        $respuesta = $sentencia->fetch(PDO::FETCH_ASSOC);
        // En caso de que no traiga datos.
        if (!$respuesta) {
            return [];
        }
        // Si tiene datos trae
        return $respuesta;
    } catch (PDOException $e) {
        throw new ExpcionApi(self::ERROR_DB, $e->getMessage());
    }
}
```

Método que trae todas las películas.

```
public static function getAll(): array
{
    try {
        // Traigo todos los datos de mi tabla
        $comando = "SELECT * FROM " . self::$table;
        // Preparar sentencia
        $sentencia = ConexionBD::obtenerInstancia()->obtenerBD()->prepare($comando);
        // Ejecutar el query.
        $sentencia->execute();
        // Retorno el resultado.
        $respuesta = $sentencia->fetchAll(PDO::FETCH_ASSOC);
        if(!$respuesta){
            return [];
        }
        return $respuesta;
    } catch (PDOException $e) {
        throw new ExpcionApi(self::ERROR_DB, $e->getMessage());
    }
}
```

Este método sirve para insertar a la base de datos una película.

```
public static function insertOne($params): string
{
    // Validación de token.
    Usuario::autenticar();
    // Validación de parámetros.
    self::validacionParams($params);
    try {
        // Obtenemos el pdo para poder hacer el insert.
        $pdo = ConexionBD::obtenerInstancia()->obtenerBD();
        $comando = "INSERT INTO " . self::$table . " (" .
            self::$columnasTabla[0] . "," .
            self::$columnasTabla[1] . "," .
            self::$columnasTabla[2] . "," .
            self::$columnasTabla[3] . ")";
        |   "VALUES (?, ?, ?, ?)";
        // Mandamos a validar si la sintaxis esta bien escrita.
        $sentencia = $pdo->prepare($comando);
        $sentencia->bindParam(1, $params->titulo, PDO::PARAM_STR);
        $sentencia->bindParam(2, $params->director, PDO::PARAM_STR);
        $sentencia->bindParam(3, $params->anio_estreno, PDO::PARAM_INT);
        $sentencia->bindParam(4, $params->genero_id, PDO::PARAM_INT);
        // Ejecutamos el script
        $resultado = $sentencia->execute();
    }
```

```

        // Si se ejecuto entonces envia un mensaje de todo correcto.
        if ($resultado) {
            return self::ESTADO_CREACION_EXITOSA;
            // De lo contrario mandara que no fue exitosa.
        } else {
            return self::ESTADO_CREACION_FALLIDA;
        }
    }catch (PDOException $e) {
        throw new ExpcionApi(self::ERROR_DB, $e->getMessage());
    }
}

```

Método para actualizar los datos de una película existente en la base de datos.

```

public static function update($params, $id): string
{
    self::validacionParams($params);
    try {
        // Obtenemos el pdo para poder hacer el insert.
        $pdo = ConexionBD::obtenerInstancia()->obtenerBD();
        $comando = "UPDATE " . self::$table . " SET " . self::$columnasTabla[0] . " =?, " . self::
            self::$columnasTabla[2] . " =?, " . self::$columnasTabla[3] . " =? WHERE id =?";
        $sentencia = $pdo->prepare($comando);
        $sentencia->bindParam(1, $params->titulo, PDO::PARAM_STR);
        $sentencia->bindParam(2, $params->director, PDO::PARAM_STR);
        $sentencia->bindParam(3, $params->anio_estreno, PDO::PARAM_INT);
        $sentencia->bindParam(4, $params->genero_id, PDO::PARAM_INT);
        $sentencia->bindParam(5, $id, PDO::PARAM_INT);
        $sentencia->execute();
        if ($sentencia->rowCount() > 0) {
            return self::ESTADO_MODIFICACO_EXITOSA;
        } else {
            return self::ESTADO_MODIFICACO_FALLIDA;
        }
    } catch (PdoException $e) {
        throw new ExpcionApi(self::ERROR_DB, $e->getMessage());
    }
}

```

Método para eliminar una película.

```
/>
public static function destroy($id): string
{
    try{
        $pdo = ConexionBD::obtenerInstancia()->obtenerBD();
        $comando = "DELETE FROM " . self::$table . " WHERE id =?";
        $sentencia = $pdo->prepare($comando);
        $sentencia->bindParam(1, $id, PDO::PARAM_INT);
        $sentencia->execute();
        if ($sentencia->rowCount() > 0) {
            return self::ESTADO_DELETE_EXITOSA;
        }else{
            return self::ESTADO_DELETE_FALLIDA;
        }
    }catch (PDOException $e){
        throw new ExcepcionApi(self::ERROR_DB, $e->getMessage());
    }
}
```

Método que valida que los parámetros sean correctos para un insert o update.

```
/* TODO: Método que valida que estén bien las columnas.
 * @param $params
 * @return void
 * @throws ExcepcionApi
 */
public static function validacionParams($params): void
{
    // Verificar si las columnasDeLaTabla las enviaron como parámetros.
    foreach (self::$columnasTabla as $columna) {
        // Si no esta definido dentro del arreglo que me enviaron entonces marco el error y lo envío.
        if (!isset($params->$columna)) {
            // Paso el arreglo a una cadena
            $mensajeError = "Las columnas son las siguientes: " . implode(', ', self::$columnasTabla);
            throw new ExcepcionApi(400, $mensajeError);
        }
    }
}
```

Método que sirve para obtener los datos de películas y el género al cual pertenece.

```
public static function pdf(): array
{
    try {
        // Traigo todos los datos de mi tabla
        $comando = "SELECT ". self::$columnasTabla[0] . " , " . self::$columnasTabla[1] . " , " .
        self::$columnasTabla[2] . ", generos.nombre FROM " . self::$table . " JOIN generos ON pelicul
        // Preparar sentencia
        $sentencia = ConexionBD::obtenerInstancia()->obtenerBD()->prepare($comando);
        // Ejecutar el query.
        $sentencia->execute();
        // Retorno el resultado.
        $respuesta = $sentencia->fetchAll(PDO::FETCH_ASSOC);
        if(!$respuesta){
            return [];
        }
        return $respuesta;
    } catch (PDOException $e) {
        throw new ExcepcionApi(self::ERROR_DB, $e->getMessage());
    }
}
```

- **Modelo Usuarios.**

Este es el método me trae un id de usuario.

```
public static function getOne($id): array
{
    try {
        $comando = "SELECT * FROM " . self::$table . " WHERE id =?";
        // Preparar sentencia
        $sentencia = ConexionBD::obtenerInstancia()->obtenerBD()->prepare($comando);
        $sentencia->bindParam(1, $id, PDO::PARAM_INT);
        // Ejecutar el query.
        $sentencia->execute();
        // Retorno el resultado.
        $respuesta = $sentencia->fetch(PDO::FETCH_ASSOC);
        // En caso de que no traiga datos.
        if (!$respuesta) {
            return [];
        }
        // Si tiene datos trae
        return $respuesta;
    } catch (PDOException $e){
        throw new ExcepcionApi(self::ERROR_DB, $e->getMessage());
    }
}
```

Método para obtener los usuarios.

```
public static function getAll(): array
{
    try{
        $comando = "SELECT * FROM " . self::$table;
        // Preparar sentencia
        $sentencia = ConexionBD::obtenerInstancia()->obtenerBD()->prepare($comando);
        // Ejecutar el query.
        $sentencia->execute();
        // Retorno el resultado.
        $respuesta = $sentencia->fetchAll(PDO::FETCH_ASSOC);
        if(!$respuesta){
            return [];
        }
        return $respuesta;
    }catch (PDOException $e){
        throw new ExcepcionApi(self::ERROR_DB, $e->getMessage());
    }
}
```

Método que actualiza un usuario.

```
public static function update($params, $id): string
{
    // Valido los parámetros
    self::validacionParams($params);
    // Encriptar contraseña.
    // Encripto la contraseña texto que me envió el usuario.
    $contrasenia = self::encryptarPassword($params->contrasenia);
    $token = self::encryptarPassword($params->contrasenia . date('Y-m-d H:i:s'));
    try {
        // Obtenemos el pdo para poder hacer el insert.
        $pdo = ConexionBD::obtenerInstancia()->obtenerBD();
        $comando = "UPDATE " . self::$table . " SET " .
        self::$columnasTable[0] . " = ?, " .
        self::$columnasTable[1] . " = ?, " .
        self::$columnasTable[2] . " = ?, " . " token = ? WHERE id = ?";
        $sentencia = $pdo->prepare($comando);
        $sentencia->bindParam(1, $params->nombre, PDO::PARAM_STR);
        $sentencia->bindParam(2, $params->correo_electronico, PDO::PARAM_STR);
        $sentencia->bindParam(3, $contrasenia, PDO::PARAM_STR);
        $sentencia->bindParam(4, $token, PDO::PARAM_STR);
        $sentencia->bindParam(5, $id, PDO::PARAM_INT);
        $sentencia->execute();
        if ($sentencia->rowCount() > 0) {
            return self::ESTADO_MODIFICADO_EXITOSA;
        }
    } catch (PDOException $e) {
        throw new ExcepcionApi(self::ERROR_DB, $e->getMessage());
    }
}
```

```
    $sentencia->execute(),
    if ($sentencia->rowCount() > 0) {
        return self::ESTADO_MODIFICADO_EXITOSA;
    } else {
        return self::ESTADO_MODIFICACION_FALLIDA;
    }
} catch (PDOException $e){
    throw new ExcepcionApi(self::ERROR_DB, $e->getMessage());
}
}
```

El método destroy sirve para eliminar un usuario.

```
public static function destroy($id): string
{
    try{
        $pdo = ConexionBD::obtenerInstancia()->obtenerBD();
        $comando = "DELETE FROM " . self::$table . " WHERE id =?";
        $sentencia = $pdo->prepare($comando);
        $sentencia->bindParam(1, $id, PDO::PARAM_INT);
        $sentencia->execute();
        if ($sentencia->rowCount() > 0) {
            return self::ESTADO_DELETE_EXITOSA;
        }else{
            return self::ESTADO_DELETE_FALLIDA;
        }
    }catch (PDOException $e){
        throw new ExcepcionApi(self::ERROR_DB, $e->getMessage());
    }
}

***
```

Método que devuelve el campo nombre, correo electrónico y contraseña para mostrar en el pdf.

```
/public static function pdf()
{
    try {
        $comando = "SELECT nombre, correo_electronico, contrasenia FROM " . self::$table ;

        // Preparar sentencia
        $sentencia = ConexionBD::obtenerInstancia()->obtenerBD()->prepare($comando);
        $sentencia->execute();
        $respuesta = $sentencia->fetchAll(PDO::FETCH_ASSOC);
        if (!$respuesta) {
            return [];
        }
        return $respuesta;
    } catch (PDOException $e) {
        throw new ExpcionApi(self::ERROR_DB, $e->getMessage());
    }
}
```

- **Modelo Genero.**

Este modelo manda a llamar id de género, de la tabla.

```
public static function getOne($id): array
{
    try {
        $comando = "SELECT * FROM " . self::$table . " WHERE id =?";
        // Preparar sentencia
        $sentencia = ConexionBD::obtenerInstancia()->obtenerBD()->prepare($comando);
        $sentencia->bindParam(1, $id, PDO::PARAM_INT);
        // Ejecutar el query.
        $sentencia->execute();
        // Retorno el resultado.
        $respuesta = $sentencia->fetch(PDO::FETCH_ASSOC);
        if (!$respuesta) {
            return [];
        }
        return $respuesta;
    } catch (PDOException $e) {
        throw new ExpcionApi(self::ERROR_DB, $e->getMessage());
    }
}
```

Este método me trae todos los géneros

```
public static function getAll(): array
{
    try {
        $comando = "SELECT * FROM " . self::$table;
        $sentencia = ConexionBD::obtenerInstancia()->obtenerBD()->prepare($comando);
        // Ejecutar el query.
        $sentencia->execute();
        // Retorno el resultado.
        $respuesta = $sentencia->fetchAll(PDO::FETCH_ASSOC);
        if (!$respuesta) {
            return [];
        }
        return $respuesta;
    } catch (PDOException $e) {
        throw new ExcepcionApi(self::ERROR_DB, $e->getMessage());
    }
}
```

Este método sirve para insertar a la base de datos género.

```
public static function insertOne($params): string
{
    // Validación de token.
    Usuario::autenticar();
    self::validacionParams($params);
    try {
        // Obtenemos el pdo para poder hacer el insert.
        $pdo = ConexionBD::obtenerInstancia()->obtenerBD();
        $comando = "INSERT INTO " . self::$table . " (" . self::$columnasTabla[0] . " , " . self::$co
        // Mandamos a validar si la sintaxis esta bien escrita.
        $sentencia = $pdo->prepare($comando);
        $sentencia->bindValue(1, $params->nombre, PDO::PARAM_STR);
        $sentencia->bindValue(2, $params->descripcion, PDO::PARAM_STR);
        $sentencia->execute();
        if ($sentencia->rowCount() > 0) {
            return self::ESTADO_CREACION_EXITOSA;
        } else {
            return self::ESTADO_CREACION_FALLIDA;
        }
    } catch (PDOException $e) {
        throw new ExpcionApi(self::ERROR_DB, $e->getMessage());
    }
}
```

Este método sirve para modificar en genero de la bace de datos.

```
public static function update($params, $id): string
{
    self::validacionParams($params);
    try {
        $pdo = ConexionBD::obtenerInstancia()->obtenerBD();
        $comando = "UPDATE " . self::$table . " SET " . self::$columnasTabla[0] . " = ? , " . self::$co
        $sentencia = $pdo->prepare($comando);
        $sentencia->bindValue(1, $params->nombre, PDO::PARAM_STR);
        $sentencia->bindValue(2, $params->descripcion, PDO::PARAM_STR);
        $sentencia->bindValue(3, $id, PDO::PARAM_INT);
        $sentencia->execute();

        if ($sentencia->rowCount() > 0) {
            return self::ESTADO_MODIFICACO_EXITOSA;
        } else {
            return self::ESTADO_MODIFICACO_FALLIDA;
        }
    } catch (PDOException $e) {
        throw new ExpcionApi(self::ERROR_DB, $e->getMessage());
    }
}
```

Este método elimina un dato de la base de datos género.

```
public static function destroy($id): string
{
    try {
        $pdo = ConexionBD::obtenerInstancia()->obtenerBD();
        $comando = "DELETE FROM " . self::$table . " WHERE id =?";  

        $sentencia = $pdo->prepare($comando);
        $sentencia->bindParam(1, $id, PDO::PARAM_INT);
        $sentencia->execute();
        if ($sentencia->rowCount() > 0) {
            return self::ESTADO_DELETE_EXITOSA;
        } else {
            return self::ESTADO_DELETE_FALLIDA;
        }
    } catch (PDOException $e) {
        throw new ExcepcionApi(self::ERROR_DB, $e->getMessage());
    }
}
```

Método que valida que los parámetros sean correctos para un insert o update.

```
public static function validacionParams($params): void
{
    // Verificar si las columnasDeLaTabla las enviaron como parámetros.
    foreach (self::$columnasTabla as $columna) {
        // Si no esta definido dentro del arreglo que me enviaron entonces marco el error y lo envío.
        if (!isset($params->$columna)) {
            // Paso el arreglo a una cadena
            $mensajeError = "Las columnas son las siguientes: " . implode(', ', self::$columnasTabla);
            throw new ExcepcionApi(400, $mensajeError);
        }
    }
}
```

Trae todos los datos de genero y lo muestra en pdf.

```
public static function pdf(): array
{
    try {
        $comando = "SELECT " . self::$columnasTabla[0] . " , " . self::$columnasTabla[1] . " FROM " .
        $sentencia = ConexionBD::obtenerInstancia()->obtenerBD()->prepare($comando);
        // Ejecutar el query.
        $sentencia->execute();
        // Retorno el resultado.
        $respuesta = $sentencia->fetchAll(PDO::FETCH_ASSOC);
        if (!$respuesta) {
            return [];
        }
        return $respuesta;
    } catch (PDOException $e) {
        throw new ExcepcionApi(self::ERROR_DB, $e->getMessage());
    }
}
```

Método que crea a un usuario.

```
public static function insertOne($params)
{
    self::validacionParams($params);
    // Si existe el correo envía la respuesta siguiente
    if (self::existeCorreо($params->correo_electronico)) {
        return "El correo ya existe";
    }
    // Encrypto la contraseña texto que me envio el usuario.
    $contrasenia = self::encryptarPassword($params->contrasenia);
    $token = self::encryptarPassword($params->contrasenia . date('Y-m-d H:i:s'));
    try {
        // Obtenemos el pdo para poder hacer el insert.
        $pdo = ConexionBD::obtenerInstancia()->obtenerBD();
        $comando = "INSERT INTO " . self::$table . " (" .
            self::$columnasTable[0] . "," .
            self::$columnasTable[1] . "," .
            self::$columnasTable[2] . "," .
            "token" . ")" .
            "VALUES(?, ?, ?, ?)";
        // Mandamos a validar si la sintaxis esta bien escrita.
        $sentencia = $pdo->prepare($comando);
        $sentencia->bindParam(1, $params->nombre, PDO::PARAM_STR);
        $sentencia->bindParam(2, $params->correo_electronico, PDO::PARAM_STR);
        $sentencia->bindParam(3, $contrasenia, PDO::PARAM_STR);
        $sentencia->bindParam(4, $token, PDO::PARAM_STR);
        // Agregar el token.
        // Ejecutamos el script
        $resultado = $sentencia->execute();
        // Si se ejecuto entonces envia un mensaje de todo correcto.
        if ($resultado) {
            // Le retorno su correo y su token.
            return ["correo" => $params->correo_electronico, "token" => $token];
            // De lo contrario manda que no fue exitosa.
        } else {
            return self::ESTADO_CREACION_FALLIDA;
        }
    } catch (PDOException $e) {
        throw new ExcepcionApi(self::ERROR_DB, $e->getMessage());
    }
}
```

Este método es para validar si ya existe este correo en la base de datos.

```
private static function existeCorreo($correoParams): bool
{
    try {
        $comando = "SELECT * FROM " . self::$table . " WHERE " . self::$columnasTable[1] . " = ?";
        // Preparar sentencia
        $sentencia = ConexionBD::obtenerInstancia()->obtenerBD()->prepare($comando);
        $sentencia->bindParam(1, $correoParams, PDO::PARAM_STR);
        $sentencia->execute();
        $respuesta = $sentencia->fetch(PDO::FETCH_ASSOC);
        if (!$respuesta) {
            return false;
        }
        return true;
    } catch (PDOException $e) {
        throw new ExcepcionApi(self::ERROR_DB, $e->getMessage());
    }
}
```

Método para validar los parámetros que sean correctos al arreglo columnas.

```
public static function validacionParams($params): void
{
    // Verificar si las columnasDeLaTabla las enviaron como parámetros.
    foreach (self::$columnasTable as $columna) {
        // Si no esta definido dentro del arreglo que me enviaron entonces marco el error y lo envío.
        if (!isset($params->$columna)) {
            // Paso el arreglo a una cadena
            $mensajeError = "Las columnas son las siguientes: " . implode(', ', self::$columnasTable);
            throw new ExcepcionApi(400, $mensajeError);
        }
    }
}
```

Método para encriptar la contraseña que me da el usuario.

```
private static function encryptarPassword($contrasenia): string
{
    return password_hash($contrasenia, PASSWORD_BCRYPT);
}
```

Este es el método de login para loguearse.

```

public static function login($params): string
{
    // Validar parámetros que son correo y contraseña.
    if (!isset($params->correo_electronico) || !isset($params->contrasenia)) {
        // Si no me los enviaron entonces envió un error.
        throw new ExcepcionApi(400, "Correo o contrasenia no valida");
    }
    // Buscar el usuario por su correo.
    if (!self::existeCorreo($params->correo_electronico)) {
        // Si no existe se lo informo.
        return "No existe su correo.";
    }
    // Obtengo el usuario.
    $usuario = self::obtenerUsuario($params->correo_electronico);
    // Si existe entonces valido, mando a desencriptar la contraseña
    if (self::descryptarPassword($params->contrasenia, $usuario['contrasenia'])) {
        return "Se ha logueado";
    }
    return "Contraseña incorrecta o correo electronico incorrecto";
}

```

Método para validar la contraseña que me dio el usuario para validar que sea la misma en la base de datos.

```

private static function descryptarPassword($contrasenia_ingresada, $hash_guardado): bool
{
    if (password_verify($contrasenia_ingresada, $hash_guardado)) {
        return true;
    }
    return false;
}

```

Método para obtener el usuario.

```

private static function obtenerUsuario($correoParams)
{
    try {
        $comando = "SELECT * FROM " . self::$table . " WHERE " . self::$columnasTable[1] . " = ?";
        // Preparar sentencia
        $sentencia = ConexionBD::obtenerInstancia()->obtenerBD()->prepare($comando);
        $sentencia->bindParam(1, $correoParams, PDO::PARAM_STR);
        $sentencia->execute();
        $respuesta = $sentencia->fetch(PDO::FETCH_ASSOC);
        if (!$respuesta) {
            return [];
        }
        return $respuesta;
    } catch (PDOException $e) {
        throw new ExcepcionApi(self::ERROR_DB, $e->getMessage());
    }
}

```

Método para autenticar el usuario.

```
public static function autenticar(): void
{
    $cabeceras = apache_request_headers();
    // Si no me enviaron el token entonces mando error.
    if (!isset($cabeceras["Authorization"])) {
        throw new ExpcionApi(
            400, "Se necesita el token.");
    }
    // Obtengo el token.
    $token = $cabeceras["Authorization"];
    // Si no hay con ese valor entonces mando un error.
    if (!self::tokenExiste($token)) {
        throw new ExpcionApi(
            410, "Clave de API no autorizada");
    }
}
```

Método que valida que haya un usuario con ese mismo token.

```
public static function tokenExiste($token): bool
{
    try{
        $comando = "SELECT COUNT( id ) FROM " . self::$table . " WHERE token = ?";
        $sentencia = ConexionBD::obtenerInstancia()->obtenerBD()->prepare($comando);
        $sentencia->bindParam(1, $token, PDO::PARAM_STR);
        $sentencia->execute();
        return $sentencia->fetchColumn(0) > 0;
    }catch (PDOException $e){
        throw new ExpcionApi(self::ERROR_DB, $e->getMessage());
    }
}
```

Creación de PDF

Pdf Usuarios

En esta parte de código validamos que nos hayan enviado pdf en la ruta, si no me lo envia valida que no este vacío si este vacío le asigna el null.

Una vez que valide si no es null entonces validamos que la petición sea get, si no es get entonces lo mandara un error el cual dira URL no valida.

Si es la petición get entonces mandara autenticar y una vez que autentique, mando a llamar al controller correspondiente, cualquiera de los 4 modelos que tengo y llamo al método pdf.

```
$pdf = $_GET['pdf'] ?? null;
if (empty ($pdf)) $pdf = null;
if (!is_null($pdf)) {
    if ($metodo === 'get'){
        Usuario::autenticar();
        $objetoController->pdf();
    }else{
        throw new ExpcionApi(estado: METHOD_NOT_ALLOWED, mensaje: "URL no valida: ");
    }
} else {
```

En este caso mostraremos el ejemplo del controllerUsuarios, el cual imprime un pdf, primero creamos la instancia de pdf, poner un titulo el cual le asignamos: Lista de usuarios, agregamos la pagina.

```
/**
 * TODO: Método que creara el pdf
 * @throws ExpcionApi
 */
public function pdf(): void
{
    // Creo la instancia
    $pdf = new PDF();
    // Creación de titulo
    $pdf->titulo(titulo: "Lista de usuarios");
    // Creo la pagina.
    $pdf->AddPage();
```

Luego llamamos al método estático pdf del modelo Usuario, el cual tiene lo siguiente.

```
// Obtengo los usuarios
$data = Usuario::pdf();
```

Este método lo que hace es obtener los campos, nombre, correo electrónico y contraseña.

La cual manda el query, lo validamos que este bien escrito y luego obtenemos el resultado.

```
/**
 * TODO: Método que devuelve el campo nombre, correo electronico y contraseña
 * @return array|mixed
 * @throws ExpcionApi
 */
public static function pdf()
{
    try {
        $comando = "SELECT nombre, correo_electronico, contrasenia FROM " . self::$table;

        // Preparar sentencia
        $sentencia = ConexionBD::obtenerInstancia()->obtenerBD()->prepare($comando);
        $sentencia->execute();
        $respuesta = $sentencia->fetchAll(mode: PDO::FETCH_ASSOC);
        if (!$respuesta) {
            return [];
        }
        return $respuesta;
    } catch (PDOException $e) {
        throw new ExpcionApi(estado: self::ERROR_DB, $e->getMessage());
    }
}
```

Regresando al método anterior el cual genera el pdf, ponemos el tipo de letra y tamaño de las cabeceras de la tabla, luego decimos que las cabeceras deben estar centrados, luego creamos las filas de los datos de la base de datos en un foreach luego lo mostramos en el navegador.

```
// Le asigno la letra, tipo y numero letra.
$pdf->SetFont('family: Arial', 'style: ', 'size: 14');
// Le asigno el tamaño que tendrá cada columna
$pdf->SetWidths(array(10, 50, 80, 40));
// Digo que estarán centrados
$pdf->SetAligns(array("C", "C", "C", "C"));
// Le asigno la cabeceras
$pdf->Row(array("No", 'Nombre', 'Correo Electrónico','Password'));
// Contador para que me diga numero 1 tal y asi.
$contador = 1;
// Le digo como estarán posicionados
$pdf->SetAligns(array("C", "L", "L", "C"));
// For each para enviarle los valores al método Row y los acomode bien
foreach ($data as $row) {
    // Agrego una fila con los valores correspondientes
    $pdf->Row(array($contador++, $row['nombre'], $row['correo_electronico'],$row['contrasenia']));
}
// Muestro el PDF.
$pdf->Output();
```

Resultado en postman:

The screenshot shows the Postman interface with the following details:

- Left Sidebar:** My Workspace, Collections, Environments, History.
- Header:** Home, Workspaces, API Network, Search Postman, Upgrade.
- Request URL:** http://localhost:4200/apiProject/users/usuarios/pdf
- Method:** GET
- Headers:** Authorization, Headers (7), Body, Scripts, Settings.
- Body:** Cookies, Headers (10), Test Results.
- Response:** Status: 200 OK, Time: 22 ms, Size: 2.7 kB, Save as example.

The response body is a table titled "Lista de usuarios" containing the following data:

Nº	Nombre	Correo electrónico	Contraseña
1	Loquisima	Loquisima@gmail.com	32c1035mNk4ar1tm7BDwWv2hBvCmsL0w9i1rp1
2	PruebaWeb	PruebaWeb@gmail.com	32c1034v13mfb1MUS4FrdnxuXO1Cbs8o4crSzg3Nz_AAA
3	andres	andres@gmail.com	32c1034v13mfb1Nos1LvhacRL1VZjooPqPw8vsWqfJyQd4fhdvA
4	Sergi	Sergi@gmail.com	32c1034v13mfb100vJPU2UicGelmpuY0G64P4RM1S2oD9P9P0PcJUNelMS

Pdf Actores

Para la creación de pdf para los actores, creamos la instancia, ponemos el título de la página, agregamos la página, luego hablamos al modelo Actor el cual habla al método estático pdf:

Este método, lo que hacemos es hacer un **JOIN** con la tabla actores y películas la cual tendrá los datos de actores y luego obtenemos el título de la película a la cual pertenece.

```

public static function pdf(): array
{
    try {
        // Traigo todos los datos de mi tabla nombre, Actores.nacionalidad, Actores.edad,
        $comando = "SELECT " . self::$columnasTabla[0] . ", " . self::$columnasTabla[1] . ", " . self::$columnasTabla[2] .
            " , peliculas.titulo FROM " . self::$table . " JOIN peliculas ON actores.pelicula_id = peliculas.id";
        // Preparan sentencia
        $sentencia = ConexionBD::obtenerInstancia()->obtenerBD()->prepare($comando);
        // Ejecutar el query.
        $sentencia->execute();
        // Retorno el resultado.
        // Si quiero traer varios datos uso fetchAll
        $respuesta = $sentencia->fetchAll(mode: PDO::FETCH_ASSOC);
        if (!$respuesta) {
            return [];
        }
        return $respuesta;
    } catch (PDOException $e) {
        throw new ExpcionApi(estado: self::ERROR_DB, $e->getMessage());
    }
}

```

Regresamos al método del controller el cual asigna el tipo de letra y tamaño, de las cabeceras de la tabla.

Luego asignamos el tamaño de cada columna, decimos que va a ir centrado, luego los nombres de las cabeceras, luego un contador el cual me va a servir para enumerar los valores, luego como va a ir en que posición, agregamos cada fila con un foreach de todos los datos y una vez que termine mandamos a mostrar el pdf.

```

$pdf->SetFont(family: 'Arial', style: '', size: 14);
$pdf->SetWidths(array(10, 50, 50, 20, 50));
$pdf->SetAligns(array("C", "C", "C", "C", "C"));
// Le asigno la cabeceras
$pdf->Row(array("No", 'Nombre', 'Nacionalidad', "Edad" , "Película"));
// Contador para que me diga numero 1 tal y asi.
$contador = 1;
$pdf->SetAligns(array("C", "L", "L", "C", "L"));
foreach ($data as $row) {
    $pdf->Row(array($contador++, $row['nombre'], $row['nacionalidad'], $row['edad'], $row['titulo']));
}
// Muestro el PDF.
$pdf->Output();

```

Resultado postman:

The screenshot shows the Postman interface with a collection named 'My Workspace' containing various API endpoints for 'User', 'Actores', and 'Generos'. A specific endpoint 'GET allActor' is selected, which performs a GET request to 'http://localhost/ApiProject/actores/pdf'. The response status is 200 OK, and the response body is a table titled 'Lista de actores' with the following data:

No	Nombre	Nacionalidad	Edad	Pelicula
1.	Prado	Mexico	60	Barbie
2.	Miguel Moses Habilo	Mexico	60	Barbie

Pdf Películas

Ahora pasaremos a la creación del pdf, que se repite lo mismo, la diferencia es que el título es otro y se llama al método pdf:

En este caso obtiene todos los valores, de la tabla película, luego el título de género.

```
public static function pdf(): array
{
    try {
        // Traigo todos los datos de mi tabla
        $comando = "SELECT ". self::$columnasTabla[0] . " , " . self::$columnasTabla[1] . " , " .
        self::$columnasTabla[2] . " , " . generos.nombre . " FROM " . self::$table . " JOIN generos ON peliculas.genero_id = generos.id";
        // Preparar sentencia
        $sentencia = ConexionBD::obtenerInstancia()->obtenerBD()->prepare($comando);
        // Ejecutar el query.
        $sentencia->execute();
        // Retorno el resultado.
        $respuesta = $sentencia->fetchAll(mode: PDO::FETCH_ASSOC);
        if(!$respuesta){
            return [];
        }
        return $respuesta;
    } catch (PDOException $e) {
        throw new ExpcionApi(estado: self::ERROR_DB, $e->getMessage());
    }
}
```

Y se repite lo mismo para obtener los datos y escribirlos en el pdf.

Resultado:

The screenshot shows the Postman interface with a collection named 'My Workspace' containing an API project. A specific endpoint 'GET allMovies' is selected. The request URL is set to 'http://localhost:ApiProject/Peliculas/allMovies'. The response body is displayed as a table:

No.	Título	Director	Año estreno	Género
1	El Padrino	Luciano Rodríguez	2023	Acción
2	Tintail Fall	Roben Roger	2024	Nóse

PDF géneros

Se crea la misma instancia, se le pone el titulo a la hoja, luego se agrega la pagina luego se llama al pdf del modelo género:

Obtiene todos los géneros del modelo y se lo retorna al controlador.

```
public static function pdf(): array
{
    try {
        $comando = "SELECT " . self::$columnasTabla[0] . " , " . self::$columnasTabla[1] . " FROM " . self::$table;
        $sentencia = ConexionBD::obtenerInstancia()->obtenerBD()->prepare($comando);
        // Ejecutar el query.
        $sentencia->execute();
        // Retorno el resultado.
        $respuesta = $sentencia->fetchAll(mode: PDO::FETCH_ASSOC);
        if (!$respuesta) {
            return [];
        }
        return $respuesta;
    } catch (PDOException $e) {
        throw new ExcepcionApi(estado: self::ERROR_DB, $e->getMessage());
    }
}
```

Luego hacemos lo mismo que anterioridad y se lo agregamos al pdf cada fila y por ultimo lo imprimimos.

The screenshot shows the Postman application interface. On the left, the 'My Workspace' sidebar lists collections: 'Api' (containing 'User', 'Actores', 'Películas', 'Generos', and 'Crud Costos, Filos'), 'ApProject' (containing 'User', 'Actores', 'Películas', 'Generos', and 'Crud Costos, Filos'), and 'Openla'. In the center, a 'GET' request is selected for the endpoint `http://localhost:3001/apiProject/generos/pdf`. The 'Headers' tab is active, showing a single header `Authorization` with the value `$2y$10$SEYQLIBgSwpOYf4mYyOE0OsvsL0HRSwkJpN9WzOHID2/TW6dy`. Below the request, the 'Test Results' section displays a table titled 'Lista de generos' with two rows:

No.	Nombre	Descripción
1	Nose	Película de nose
2	Iceto Tokyo	Película de carros

At the bottom right of the results panel, status information is shown: Status 200 OK, Time: 30 ms, Size: 2.19 kB, Save as example.