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: Autores, Géneros, Películas y Actores.

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

De igual manera consumiremos la API en JavaScript con Fetch.

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 y 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=$2 [QSA]
  ```
  El cual nos ayudara a registrar un usuario asi como loguearse y obtener su token.
  En esta instrucción obtenemos en el primer paréntesis el modelo lo capturamos luego puede ir login o registros y lo capturara en acción.
- Como segunda regla es:
  La captura de un pdf la cual permite cualquier modelo pueda capturar un pdf basado al modelo al cual va acceder:
  ```
  RewriteRule ^([a-z]+)/(pdf)$ index.php?model=$1&pdf=$2 [QSA]
  ```
  La regla nos dice que pueden ir todo el alfabeto numérico pero que sean minúsculas por lo mínimo debe ir una letra, luego sigue la palabra pdf para poder redireccionarlo en el índex.
- Como tercera regla es:
  La cual va a permitir, eliminar, actualizar, modificar y consultar todos o un valor en especifico con su ID, como en la regla anterior pedimos como mínimo una letra minúscula como mínimo luego decimos que puede ir un numero o es opconal.
  ```
  RewriteRule ^([a-z]+)(/(\d+))?$ 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 exista 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.

```
$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 vacio.

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

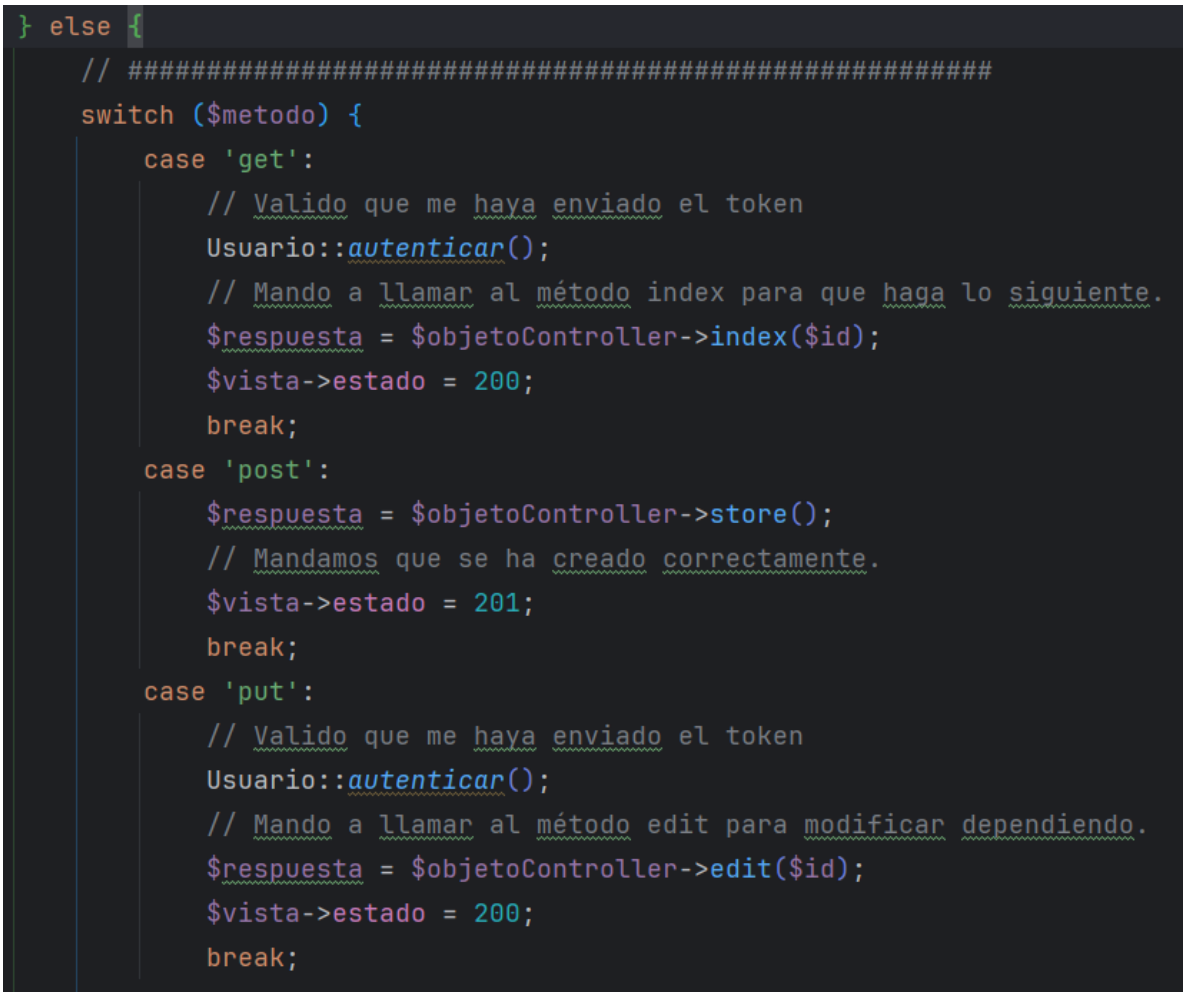
Luego verificamos que me haya enviado pdf.

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

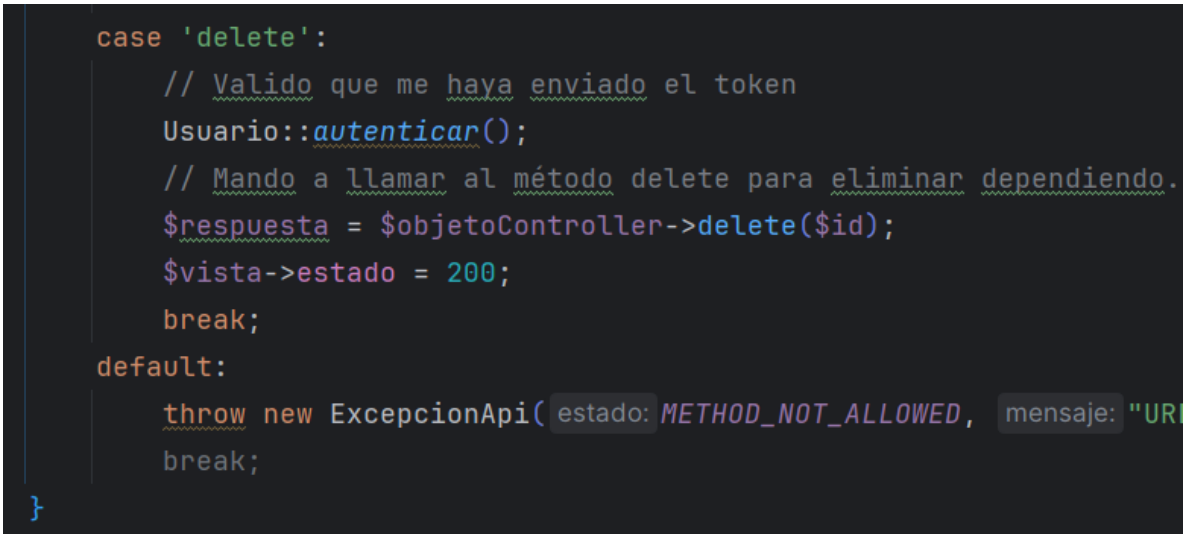
Verificamos que pdf que no sea null, si no es entonces validamos si el verbo http es get para poder obtener el pdf del método que me hayan enviado, antes validamos el token que me hayan enviado.

```php
if (!is_null($pdf)) {
    if ($metodo === 'get'){
        Usuario::autenticar();
        $objetoController->pdf();
    }else{
        throw new ExcepcionApi( 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 ExcepcionApi( estado: METHOD_NOT_ALLOWED, mensaje: "URI
            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 crear 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 genero 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.

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

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

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

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

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

```php
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");
    }
}
```

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

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

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

```php
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" ,"Pelicula"));
    // 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
    }
    // Muestro el PDF.
    $pdf->Output();
}
```

Modelo

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

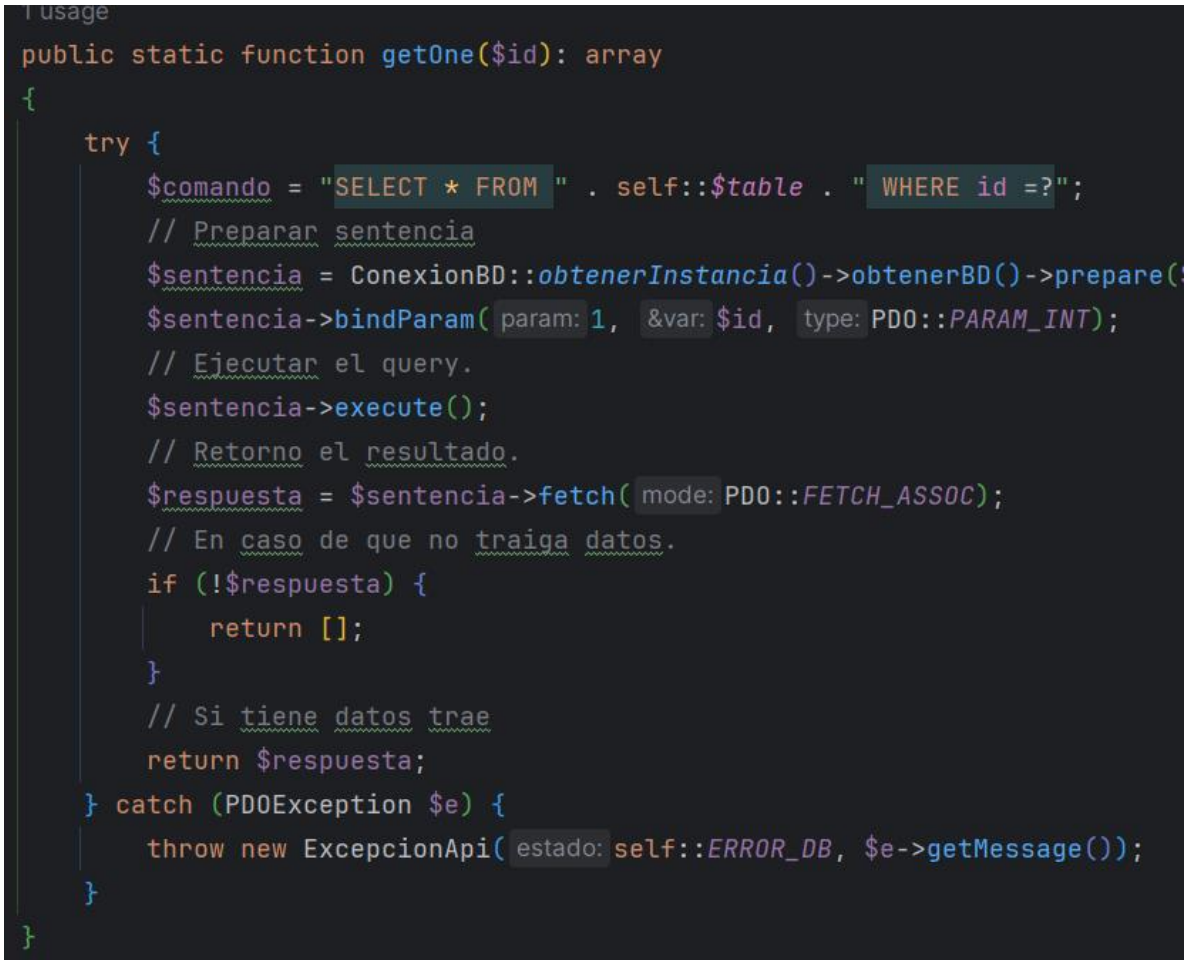
```php
protected static $columnasTabla = [
    'nombre',
    'nacionalidad',
    'edad',
    'pelicula_id'
];
```

Luego siguen los métodos:

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

```php
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(
        // 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());
    }
}
```

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

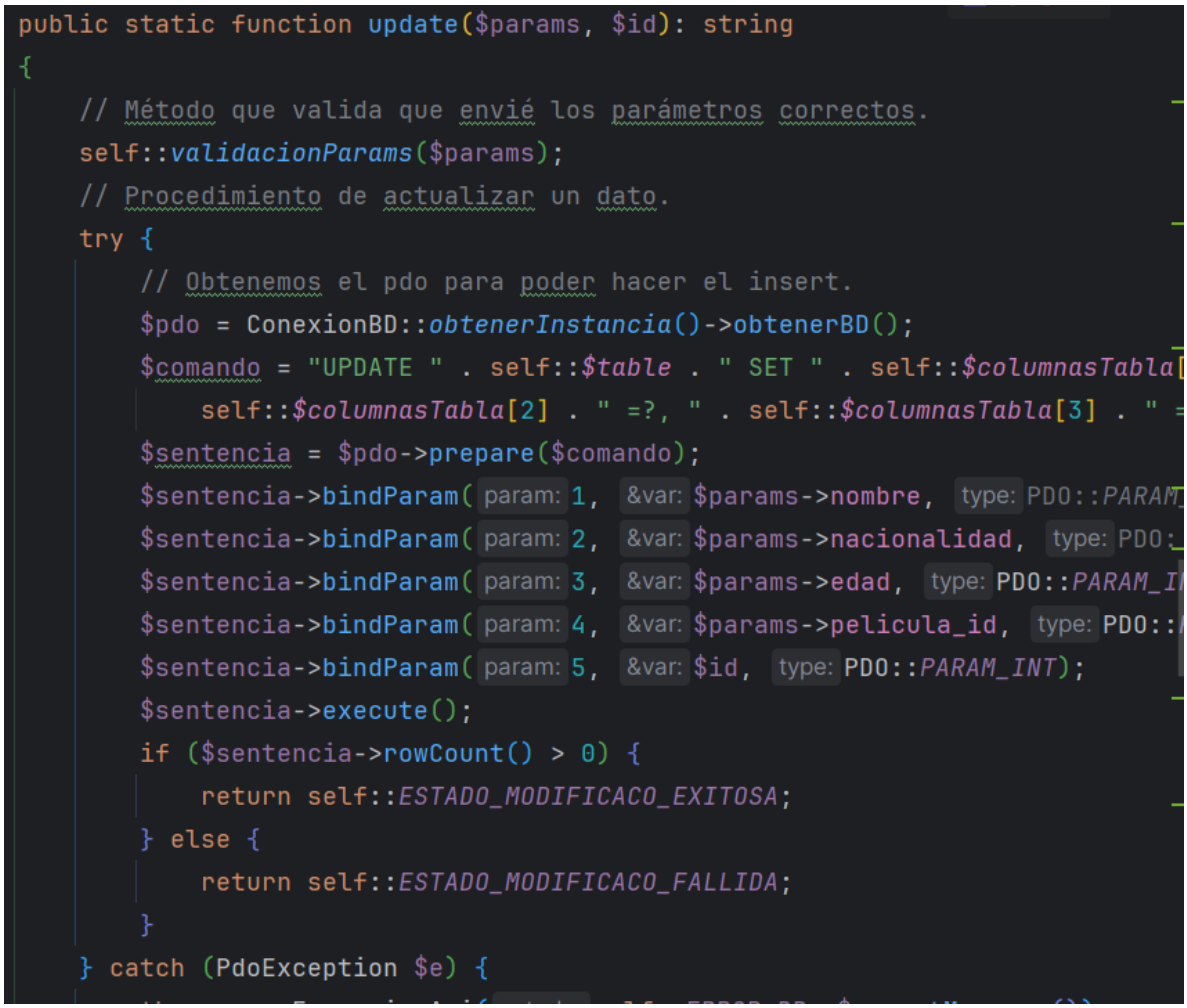


```
1 usage
public static function getOne($id): array
{
    try {
        $comando = "SELECT * FROM " . self::$table . " WHERE id =?";
        // Preparar sentencia
        $sentencia = ConexionBD::obtenerInstancia()->obtenerBD()->prepare(
        $sentencia->bindParam( param: 1, &var: $id, type: PDO::PARAM_INT);
        // 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 ExcepcionApi( estado: self::ERROR_DB, $e->getMessage());
    }
}
```

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

```php
public static function insertOne($params): string
{
    // Validación de token.
    Usuario::autenticar();
    // Método que valida que envié los parámetros correctos.
    self::validacionParams($params);
    // Hace la insercion 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( param: 1,  &var: $params->nombre,  type: PDO::PARA
        $sentencia->bindParam( param: 2,  &var: $params->nacionalidad,  type: PDO
        $sentencia->bindParam( param: 3,  &var: $params->edad,  type: PDO::PARAM_
        $sentencia->bindParam( param: 4,  &var: $params->pelicula_id,  type: PDO:
        // Ejecutamos el script
```

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



```php
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::$columnasTabla[
            self::$columnasTabla[2] . " =?, " . self::$columnasTabla[3] . " =
        $sentencia = $pdo->prepare($comando);
        $sentencia->bindParam( param: 1, &var: $params->nombre, type: PDO::PARAM_
        $sentencia->bindParam( param: 2, &var: $params->nacionalidad, type: PDO::
        $sentencia->bindParam( param: 3, &var: $params->edad, type: PDO::PARAM_I
        $sentencia->bindParam( param: 4, &var: $params->pelicula_id, type: PDO::
        $sentencia->bindParam( param: 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) {
```

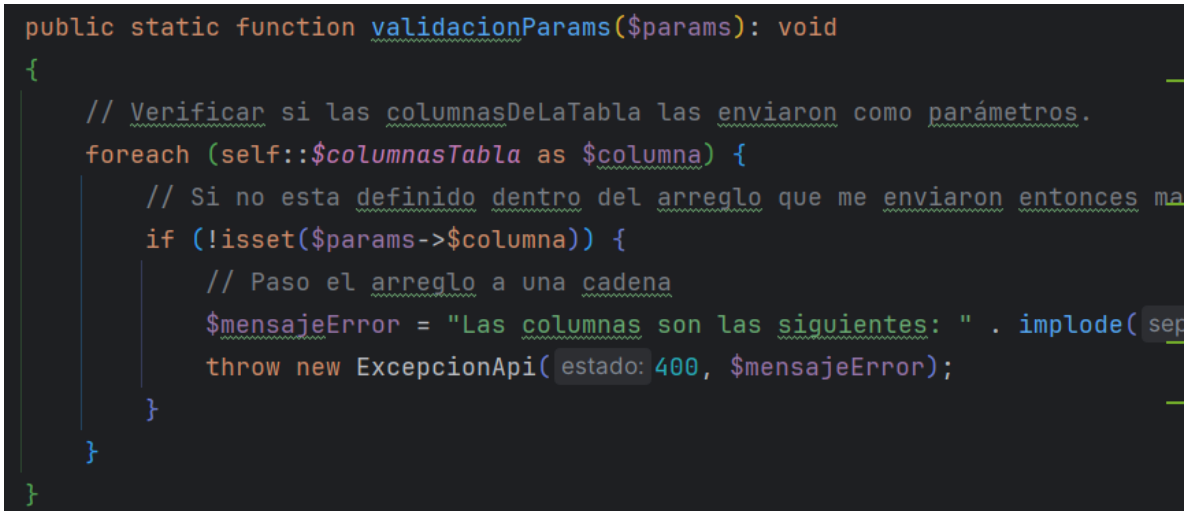
- DELETE: Elimina un dato por su id

```php
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.

```php
public static function pdf(): array
{
    try {
        // Traigo todos los datos de mi tabla nombre, Actores.nacionalidad,
        $comando = "SELECT  " . self::$columnasTabla[0] . ", ". self::$colu
            " , peliculas.titulo FROM " .self::$table ." JOIN  peliculas ON
        // Preparar sentencia
        $sentencia = ConexionBD::obtenerInstancia()->obtenerBD()->prepare($
        // 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.



```php
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( sep
            throw new ExcepcionApi( estado: 400, $mensajeError);
        }
    }
}
```