Vanilla PHP - Rest API application

The application utilises REST API written in PHP for Creating, Reading, Updating, and Deleting (CRUD) repositories of films, actors and genres saved in MySQL database. It demonstrates the practical use of Object Oriented Programming (OOP) and utilises the Model-Controller-Router paradigm.

Requirements

The app is written in PHP 8.1. It uses match expressions, so it's not complied with versions of PHP lower than 8.0
It also uses the PDO (PHP Data Objects) PHP extension to access the database. So make sure the extension is installed in your interpreter before use.

The app utilises the composer (package manager for PHP) to auto-import the project files. It doesn't use any external dependencies. However, the application automatically imports the project files if the composer is not used. A user doesn't need to install anything.

How to run the app

Docker

The most straightforward way to run the app is to use Docker (https://www.docker.com/products/docker-desktop/) from the `docker-compose.yml` file that comes with the application.

Inside `docker-compose.yml` are defined two containers:

- API: it contains the PHP interpreters and runs the development server
- DB: it runs MySQL server, and on the image runtime, it imports `db/create_tables.sql` to create MYSQL tables.

The database credentials are defined inside the `.env` file and automatically injected to the containers.

To run the application navigate to the folder and execute the command:

```
docker-compose up -d --build
```

After a couple of minutes of building containers. The application runs MySQL server with tables with the development server that contains the REST API application, and it's available on the URL:

```
http://localhost:8080
```

```
RESPONSE:
Controller is not set.
```

Sweet! The application is running.

Here (commands-docker.md) you can find some useful commands for managing the docker containers.

Local interpreter or WAMP, MAMP servers

To use your local interpreter or WAMP, MAMP. Follow belows steps:

1. You need to create a MySQL database.
2. Import the file `db/create_tables.sql` to the created database to create tables.
3. Edit app/Api/Models/DbConnection.php (app/Api/Models/DbConnection.php) and add the database credentials, like on the snippet below:

```
class DbConnection
{
    // ... begin of the class code
    protected static string $host = 'localhost';
    protected static string $name = 'movies';
    protected static string $user = 'db_user';
    protected static string $password = 'password';
    protected static int $port = 3306;
    // ... rest of the class code
}
```

To run the app by using a local interpreter of PHP and its development server, run the command:

```
php -S localhost:8080 -t /path/to/application/
```

The app should be available at the URL:

```
http://localhost:8080
```

For WAMP and MAMP servers, the app hasn't been tested, but it should be able to run. It might return 404 errors on routes for controllers (e.g., `http://localhost:8080/app/films`). If the 404 error emerges, the app supports using get parameters of the following structure `http://localhost:8080/app/?controller=films`.

Testing

To test the app, you can use the CURL (https://curl.se/) or Postman (https://www.postman.com/).
The examples below utilised the curl library and were being tested in a bash terminal.
The application utilises three controllers:

1. Films - for managing films base URL is http://localhost:8080/films or http://localhost:8080/?controller=films
2. Genres - for managing genres base URL is http://localhost:8080/genres or http://localhost:8080/?controller=genres
3. Actors - for managing actors base URL is http://localhost:8080/actors or http://localhost:8080/?controller=actors

---

Add a new entry

Method: **POST**
Header: **Content-Type: application/json**
Response codes: **202 Created** | **400 Bad Request**

Films URL: http://localhost:8080/films or http://localhost:8080/?controller=films
Films POST Params:

```
{
  "name": "string-required",
  "year": "int-required",
  "genre": "int|string-required"
}
```

Genres URL: http://localhost:8080/genres or http://localhost:8080/?controller=genres
Genres POST Params:

```
{
  "name": "string-required"
}
```

Actors URL: http://localhost:8080/actors or http://localhost:8080/?controller=actors
Actors POST Params:

```
{
  "name": "string-required",
  "surname": "string-required",
  "gender": "male|female-required"
}
```

Curl example:

```
curl -X POST --location "http://localhost:8080/films" \
    -H "Content-Type: application/json" \
    -d "{
          \"title\": \"Matrix\",
          \"year\": 1999,
          \"genre\": \"Science fiction\"
        }"
```

In this example, the genre is a string, so the new record is created in the foreign table (genres) with the given name if the integer is treated as a genre ID.

Success response:

```
{
  "message": "Successfully created",
  "data": {
    "id": 2,
    "title": "Matrix",
    "year": 1999,
    "genre": 3
  }
}
```


Bad response:


```
{
  "error": "Wrong parameters given. YEAR is required"
}
```

Display all entries

Method: **GET**
Response code: **200 OK**
Films URL: http://localhost:8080/films or http://localhost:8080/?controller=films
Genres URL: http://localhost:8080/genres or http://localhost:8080/?controller=genres
Actors URL: http://localhost:8080/actors or http://localhost:8080/?controller=actors

Curl example:

```
curl -X GET --location "http://localhost:8080/films"
```

Success response:


```
[
  {
    "id": 1,
    "title": "Omen",
    "year": 1974,
    "genre": "Horror"
  },
  {
    "id": 2,
    "title": "Matrix",
    "year": 1999,
    "genre": "Science fiction"
  },
  {
    "id": 3,
    "title": "Snatch",
    "year": 2000,
    "genre": "Comedy"
  }
]
```


Edit an entry

Method: **PUT**
Header: **Content-Type: application/json**
Response codes: **202 Accepted** | **400 Bad request** | **404 Not Found**
GET params: **{$id} int-required**

Films URL: http://localhost:8080/films/{$id} or http://localhost:8080/?controller=films&id={$id}
Films POST Params:

```
{
  "name": "string-required",
  "year": "int-required",
  "genre": "int|string-required"
}
```

Genres URL: http://localhost:8080/genres/{$id} or http://localhost:8080/?controller=genres&id={$id} \

Genres POST Params:

```
{
  "name": "string-required"
}
```

Actors URL: http://localhost:8080/actors/{$id} or http://localhost:8080/?controller=actors&id={$id}
Actors POST Params:

```
{
  "name": "string-required",
  "surname": "string-required",
  "gender": "male|female-required"
}
```

Curl example:

```
curl -X PUT --location "http://localhost:8080/films/1" \
    -H "Content-Type: application/json" \
    -d "{
          \"title\": \"The Omen\",
          \"year\": 1976,
          \"genre\": \"Horror\"
        }"
```

Success response:

```
{
  "message": "Successfully updated",
  "data": {
    "id": 1,
    "title": "The Omen",
    "year": 1976,
    "genre": 6
  }
}
```

Bad response:

```
{
  "error": "Wrong parameters given. YEAR is required."
}
```

Not found response:

```
{
  "error": "The record has not been found."
}
```

Get a single entry

Method: **GET**
Response codes: **202 Accepted** | **404 Not Found**
GET params: **{$id} int-required**
Films URL: http://localhost:8080/films/{$id} or http://localhost:8080/?controller=films&id={$id}
Genres URL: http://localhost:8080/genres/{$id} or http://localhost:8080/?controller=genres&id={$id}
Actors URL: http://localhost:8080/actors/{$id} or http://localhost:8080/?controller=actors&id={$id}

Curl example:

```
curl -X GET --location "http://localhost:8080/films/1"
```

Success response:

```
{
  "id": 1,
  "title": "The Omen",
  "year": 1976,
  "genre": "Horror"
}
```

Not found response:

```
{
  "msg": "The record has not been found."
}
```

Delete an entry

Method: **DELETE**
Response codes: **202 Accepted** | **404 Not Found**
GET params: **{$id} int-required**
Films URL: http://localhost:8080/films/{$id} or http://localhost:8080/?controller=films&id={$id}
Genres URL: http://localhost:8080/genres/{$id} or http://localhost:8080/?controller=genres&id={$id}
Actors URL: http://localhost:8080/actors/{$id} or http://localhost:8080/?controller=actors&id={$id}

Curl example:

```
curl -X DELETE --location "http://localhost:8080/films/1"
```

Success response:

```
{
  "message": "Successfully deleted"
}
```

Not found response:

```
{
  "msg": "The record has not been found."
}
```