**Documentation 1PHPD**

**Candice DAMBRIN & Lucie POTIER**

**~**

Faisant partie d'Internet Movies DataBase & co, notre mission est de fournir une solution permettant aux utilisateurs d’acheter mais également de chercher des films.

Avec cette nouvelle ère technologique, l’entreprise souhaite abandonner l’ancien papier et proposer une solution complète permettant aux utilisateurs de consulter des informations en ligne. Cela doit utiliser des technologies Web telles que HTML, CSS et PHP (en utilisant une base de données SQL).

Script sql, technologies utilisées, schéma des tables, comment lancer le projet, doc api, types de requêtes, données envoyées.

Configuration du projet

Arborescence du dossier Projet :

Une image contenant texte, capture d’écran, Police, nombre

Description générée automatiquement

Docker

Tout d’abord, nous utilisons Docker, une plateforme open-source qui permet de créer, déployer et exécuter des applications dans des conteneurs légers et portables. Ces conteneurs sont des environnements logiciels autonomes qui incluent tout ce dont une application a besoin pour fonctionner, comme le code, les bibliothèques et les dépendances.

Au sein de notre conteneur « 1phpd-project-candicelucie », nous avons 4 applications :

* Phpmyadmin permettant de gérer la base de données
* Db étant la base de données mySql utilisé pour le site web
* Php-website correspondant au site web lui-même
* Php-api représentant notre API

L’API définit ici, les méthodes standardisées par lesquelles la base de données et le site web vont interagir ensemble.

Dans le dossier Api, nous trouvons 2 dossiers, dans le dossier « config » nous retrouvons le fichier de configuration par défaut de l’api « 000-default.conf :

|  |
| --- |
| <VirtualHost \*:80>  ServerAdmin webmaster@localhost  DocumentRoot /var/www/html  <Directory /var/www/html>  Options +Indexes  AllowOverride All  Require all granted  </Directory>  ErrorLog ${APACHE\_LOG\_DIR}/error.log  CustomLog ${APACHE\_LOG\_DIR}/access.log combined  </VirtualHost> |

Et le fichier de configuration Docker de l’api « Dockerfile » :

|  |
| --- |
| FROM php:8.2-apache  RUN apt-get update && apt-get upgrade -y && \  apt-get install -y zlib1g-dev libwebp-dev libpng-dev libzip-dev && \  docker-php-ext-install gd zip pdo\_mysql  RUN a2enmod rewrite && \  a2enmod autoindex  COPY 000-default.conf /etc/apache2/sites-available/000-default.conf  RUN chown -R www-data:www-data /var/www && \  chmod -R 755 /var/www  EXPOSE 80 |

Dans le dossier « website », nous avons également 2 dossiers, dans le dossier « config » nous trouvons le fichier de configuration par défaut du site web « 000-default.conf » :

|  |
| --- |
| <VirtualHost \*:80>  ServerAdmin webmaster@localhost  DocumentRoot /var/www/html  <Directory /var/www/html>  Options +Indexes  AllowOverride All  Require all granted  </Directory>  ErrorLog ${APACHE\_LOG\_DIR}/error.log  CustomLog ${APACHE\_LOG\_DIR}/access.log combined  </VirtualHost> |

Et le fichier de configuration Docker du site web « Dockerfile » :

|  |
| --- |
| FROM php:8.2-apache  RUN apt-get update && apt-get upgrade -y && \  apt-get install -y zlib1g-dev libwebp-dev libpng-dev libzip-dev && \  docker-php-ext-install gd zip pdo\_mysql  RUN a2enmod rewrite && \  a2enmod autoindex  COPY 000-default.conf /etc/apache2/sites-available/000-default.conf  RUN echo "auto\_prepend\_file='/var/www/html/init.php'" >> /usr/local/etc/php/conf.d/auto\_prepend.ini  RUN chown -R www-data:www-data /var/www && \  chmod -R 755 /var/www  EXPOSE 80 |

Les lignes suivantes :

|  |
| --- |
| RUN echo "auto\_prepend\_file='/var/www/html/init.php'" >> /usr/local/etc/php/conf.d/auto\_prepend.ini |

Permettent d’exécuter automatiquement lors du lancement du site le fichier « init.php » qui se trouve dans le dossier « src » du dossier « website » :

|  |
| --- |
| <?php  session\_start(); |

Ainsi, la session, utilisateur notamment, sera lancée directement au lancement du site.

Pour la configuration de notre conteneur, nous avons également le fichier docker-compose.yml avec ce contenu qui sera utilisé lors du lancement du conteneur :

|  |
| --- |
| version : '3'  services:  php-website:  build: website/config  container\_name: php-website  volumes:  - ./website/src:/var/www/html  - /var/log/apache2/app:/var/log/apache2/  env\_file: .env  ports:  - '80:80'  networks:  - app\_network  php-api:  build: api/config  container\_name: php-api  volumes:  - ./api/src:/var/www/html  - /var/log/apache2/app:/var/log/apache2/  env\_file: .env  ports:  - '8080:80'  networks:  - app\_network  db:  image: mysql:8.3.0  volumes:  - db\_data:/var/lib/mysql  env\_file: .env  ports:  - '3306:3306'  networks:  - app\_network  phpmyadmin:  image: phpmyadmin/phpmyadmin  env\_file: .env  ports:  - '8081:80'  networks:  - app\_network  depends\_on:  - db  networks:  app\_network:  volumes:  db\_data: |

Pour lancer et arrêter le projet, nous utilisons :

|  |
| --- |
| docker compose up -d --build  docker compose down |

Pour accéder au site, il faut saisir dans la barre url du navigateur :

|  |
| --- |
| http://localhost |

Pour accéder à l’API :

|  |
| --- |
| http://localhost:8080 |

Pour accéder à phpmyadmin :

|  |
| --- |
| http://localhost:8081 |

Pour terminer, nous utilisons un fichier d’environnement « .env » pour stocker des variables d’environnement qui vont notamment servir lors de la connexion à la base de données:

|  |
| --- |
| MYSQL\_ROOT\_PASSWORD=root  MYSQL\_DATABASE=my\_db  MYSQL\_USER=my\_user  MYSQL\_PASSWORD=my\_password  PMA\_HOST=db |

Base de données

Ensuite, dans notre projet, il est essentiel de créer une connexion entre notre api et notre base de données.   
Pour cela, dans le dossier « api » du projet, nous avons un dossier « database » dans lequel nous trouvons le fichier « database.php » :

|  |
| --- |
| <?php  **class** **Database** {  **private** **static** $instance = **null**;  **private** $conn;  **private** $host;  **private** $dbName;  **private** $username;  **private** $password;  **public** **function** **\_\_construct**() {  **try** {  $this->host = $\_ENV['PMA\_HOST'];  $this->dbName = $\_ENV['MYSQL\_DATABASE'];  $this->username = $\_ENV['MYSQL\_USER'];  $this->password = $\_ENV['MYSQL\_PASSWORD'];  $this->conn = **new** PDO("mysql:host=$this->host;dbname=$this->dbName", $this->username, $this->password);  $this->conn->setAttribute(PDO::ATTR\_ERRMODE, PDO::ERRMODE\_EXCEPTION);  $this->conn->exec("set names utf8");  } **catch**(PDOException $exception) {  **echo** "Erreur de connexion : " . $exception->getMessage();  }  }  **public** **static** **function** **getInstance**() {  **if** (!self::$instance) {  self::$instance = **new** Database();  }  **return** self::$instance->conn;  }  } |

Ce dernier contient des propriétés privées pour stocker les variables d’environnement utilisées lors de la connexion à la base de données. Nous avons également la méthode getInstance() qui permet l’accés unique à la classe Database, il n’y a qu’une seule connexion à la base de données. C’est l’API qui se connecte à la base de données.

Une image contenant texte, capture d’écran, diagramme, nombre

Description générée automatiquementVoici le schéma des tables de la base de données :

L’API

Gestion des utilisateurs (users.php)

Créer un utilisateur :

URL: /users

Méthode: POST

Description: Crée un nouvel utilisateur.

Corps de la requête :

|  |
| --- |
| {  "name": "Dambrin",  "firstname": "Candice",  "mail": "candice.dambrin@supinfo.com",  "mdp": "root"  } |

Réponse OK 200 :

|  |
| --- |
| {  "message": "User created. You can now login ! "  } |

Réponse Bad Request 400 :

|  |
| --- |
| {  "message": "Body with properties required.”  } |

Réponse Internal Server Error 500 :

|  |
| --- |
| {  "message": "Error while inserting user : [erreur]."  } |

Connexion utilisateur :

URL: /login

Méthode: POST

Description: Connexion d’un utilisateur.

Corps de la requête :

|  |
| --- |
| {  "mail": "candice.dambrin@supinfo.com",  "mdp": "root"  } |

Réponse OK 200 :

|  |
| --- |
| {  "message": "Login successful.",  "user": {  "fname": "candice",  "lname": "dambrin",  "email": "candice.dambrin@supinfo.com",  "id\_user": **2**,  "cart": []  }  } |

Réponse Bad Request 400 :

|  |
| --- |
| {  "message": "Email and password required.”  } |

Réponse Unauthorized 401 :

|  |
| --- |
| {  "message": "Invalid Credentials."  } |

Réponse Internal Server Error 500 :

|  |
| --- |
| {  "message": "Database error : [Erreur].”  } |

Gestion du panier (cart.php)

Obtenir le panier :

URL: /cart

Méthode: GET

Paramètre : ‘id\_user’ (query string)

Description: Retourne le panier d’un utilisateur connecté

Corps de la requête :

|  |
| --- |
| {  "id\_user": 78,  "cart": [  {  "title": "Godzilla x Kong : Le Nouvel Empire",  "image": "gmk2",  "price": **12**  }  ]  } |

Réponse OK 200 :

|  |
| --- |
| [  {  "title": "Godzilla x Kong : Le Nouvel Empire",  "image": "gmk2",  "price": **12**  }  ] |

Réponse Internal Server Error 500 :

|  |
| --- |
| {  "message": "Database error : [Erreur]"  } |

Ajouter au panier :

URL: /cart

Méthode: POST

Paramètre : ‘id\_user’ (query string) et ‘id\_movie’ (query string)

Description: Ajoute un film au panier utilisateur.

Corps de la requête :

|  |
| --- |
| {  "id\_user": 78,  "id\_movie": 2  } |

Réponse OK 200 :

|  |
| --- |
| [  {  "message": "Movie added to cart."  }  ] |

Réponse Bad request 400 :

|  |
| --- |
| {  "message": "The movie is already in the cart. "  } |

Réponse Internal Server Error 500 :

|  |
| --- |
| {  "message": "Failed to add movie to cart. "  } |

Supprimer du panier :

URL: /cart

Méthode: DELETE

Paramètre : ‘id\_cart’ (query string) correspondant au numéro de l’enregistrement du film dans le panier.

Description: Supprime un film du panier utilisateur.

Corps de la requête :

|  |
| --- |
| {  "id\_cart": 40  } |

Réponse OK 200 :

|  |
| --- |
| [  {  "message": "Movie removed from cart."  }  ] |

Réponse Internal Server Error 500 :

|  |
| --- |
| {  "message": "Failed to remove movie from cart. "  } |

Gestion des films de la page d’accueil (movie.php)

Obtenir les informations d’un film :

URL: /movie\_index

Méthode: GET

Paramètre : ‘id’ (query string)

Description: Retourne les détails d’un film spécifique.

Corps de la requête :

|  |
| --- |
| {  "id\_movie": 1,  } |

Réponse OK 200 :

|  |
| --- |
| {  "id\_movie": **1**,  "title": "Godzilla x Kong : Le Nouvel Empire",  "synopsis": "Le tout-puissant Kong et le redoutable Godzilla unissent leurs forces contre une terrible menace encore secrète qui risque de les anéantir et qui met en danger la survie même de l’espèce humaine. GODZILLA X KONG : LE NOUVEL EMPIRE remonte à l’origine des deux titans et aux mystères de Skull Island, tout en révélant le combat mythique qui a contribué à façonner ces deux créatures hors du commun et lié leur sort à celui de l’homme pour toujours…",  "price": **12**,  "image": "gmk2",  "video": "<iframe width=\"560\" height=\"315\" src=\"https://www.youtube.com/embed/A\_bERKuGCJM?si=YiruSyiVtU6O0GJ2\" title=\"YouTube video player\" frameborder=\"0\" allow=\"accelerometer; autoplay; clipboard-write; encrypted-media; gyroscope; picture-in-picture; web-share\" referrerpolicy=\"strict-origin-when-cross-origin\" allowfullscreen></iframe>",  "director\_id": **1**,  "director\_fname": "Adam",  "director\_lname": "Wingard",  "actors": "Rebecca Hall, Bryan Tyree Henry, Dan Stevens",  "categories": [  "Action"  ]  } |

Réponse Bad request/not found 404 :

|  |
| --- |
| {  "message": "Movie not found.”  } |

Réponse Internal Server Error 500 :

|  |
| --- |
| {  "message": "Database error : [Erreur].”  } |

Gestion de la fonction de recherche

Obtenir les résultats d’une recherche :

URL: /search

Méthode: GET

Paramètre : ‘query’ (query string) correspondant à la chaine de recherche pour les films.

Description: Obtenir les résultats d’une recherche effectuée.

Corps de la requête :

|  |
| --- |
| **GET** /php-api/**search**?query=kung+fu |

|  |
| --- |
| {  "query": "kung+fu"  } |

Réponse OK 200 :

|  |
| --- |
| [  {  "id\_movie": **2**,  "title": "Kung Fu Panda 4",  "image": "kfp4"  }  ] |

Réponse No result 404 :

|  |
| --- |
| {  "message": "No result found. "  } |

Réponse Unauthorized Request 405 :

|  |
| --- |
| {  "message": "Method not allowed. "  } |

La recherche peut se faire par titre de film ou par directeur de film.

Le fichier « api.php » est utilisé comme routeur principal pour l’application. Il reçoit les requêtes HTTP entrantes, analyse l'URL demandée et la méthode HTTP utilisée, puis dirige la requête vers le contrôleur approprié pour le traitement. C’est la méthode processRequest qui est responsable de cela.

Contenu du fichier :

|  |
| --- |
| <?php  **require\_once** 'users.php';  **require\_once** 'cart.php';  **require\_once** 'movie.php';  **class** **ApiRouter** {  **public** **function** **processRequest**() {  $uri = parse\_url($\_SERVER['REQUEST\_URI'], PHP\_URL\_PATH);  $uriSegments = explode('/', trim($uri, '/'));  $requestMethod = $\_SERVER['REQUEST\_METHOD'];  **switch** ($uriSegments[**0**]) {  **case** '':  **switch** ($requestMethod) {  **case** 'GET':  **echo** json\_encode(["message" => "Hello World !"]);  **break**;  **default**:  http\_response\_code(**405**);  **echo** json\_encode(["message" => "Method not allowed"]);  **break**;  }  **break**;  **case** 'users':  $controller = **new** UsersController();  **switch** ($requestMethod) {  **case** 'POST':  $controller->createUser();  **break**;  **default**:  http\_response\_code(**405**);  **echo** json\_encode(["message" => "Method not allowed"]);  **break**;  }  **break**;  **case** 'login':  $controller = **new** UsersController();    **switch** ($requestMethod) {  **case** 'POST':  $controller->loginUser();  **break**;  **default**:  http\_response\_code(**405**);  **echo** json\_encode(["message" => "Method not allowed"]);  **break**;  }  **break**;  **case** 'cart':  session\_start();  $controller = **new** CartController();  **switch** ($requestMethod) {  **case** 'GET':  $id\_user = $\_GET['id\_user'] ?? **null**;  $controller->getCart($id\_user);  **break**;  **case** 'POST':  $data = json\_decode(file\_get\_contents('php://input'), **true**);  $id\_user = $data['id\_user'] ?? '';  $id\_movie = $data['id\_movie'] ?? '';  $controller->addToCart($id\_user, $id\_movie);  **break**;  **case** 'DELETE':  parse\_str(file\_get\_contents('php://input'), $deleteParams);  $id\_cart = $deleteParams['id\_cart'] ?? '';  $controller->removeFromCart($id\_cart);  **break**;  **default**:  http\_response\_code(**405**);  **echo** json\_encode(["message" => "Method not allowed"]);  **break**;  }  **break**;  **case** 'movie\_index':  $controller = **new** MovieController();  **switch** ($requestMethod) {  **case** 'GET':  $controller->getMovie();  **break**;  **default**:  http\_response\_code(**405**);  **echo** json\_encode(["message" => "Method not allowed"]);  **break**;  }  **Break;**  **default**:  http\_response\_code(**404**);  **echo** json\_encode(["message" => "The requested URL was not found on this server."]);  **break**;  }  }  } |

Le site

Dans le dossier « website », nous retrouvons dans le dossier « src » l’ensemble des pages du sites :

* Index.php (page d’accueil)
* Header.php (header de la page d’accueil)
* Basicheader.php (header de toutes les autres pages)
* Connexion.php (page de connexion)
* Inscripton.php (page d’inscription)
* Footer.php (footer des pages du site)
* Movie.php (pages avec les détails de films)
* Categories.php (pages des catégories)

Dans ce même dossier, nous aurons des fichier « \_api » par exemple pour le fichier « index.php » nous trouverons un fichier « index\_api.php ». Ces fichiers envoient les requêtes à l’API avec les paramètres correspondants t récupèrent la réponse de l’API afin de la traiter notamment afficher les données sur les pages du site.