{EPITECH}

BOOTSTRAP_

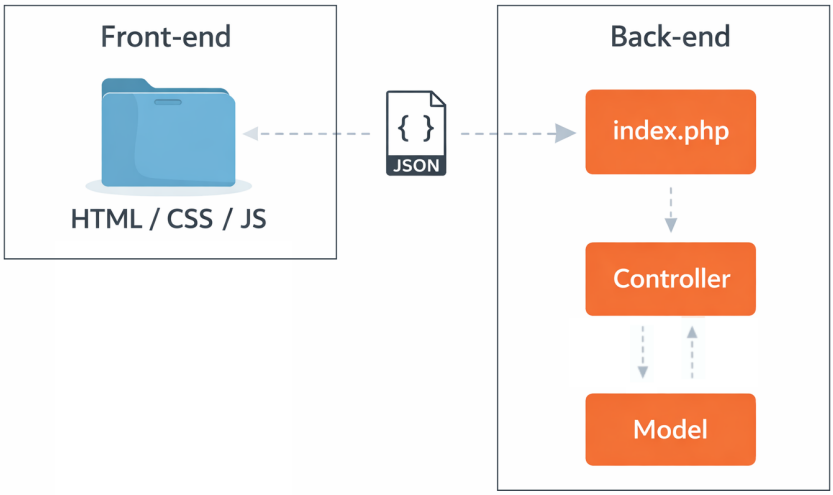
< MVC / POO / API />

BOOTSTRAP

Bootstrap - Architecture MVC et POO

Ce bootstrap vous guide pas à pas pour mettre en place l'architecture d'un projet PHP orienté **POO**, en **MVC**, avec un **backend API** consommé par un front minimal.
L'objectif est que vous puissiez démarrer rapidement, créer vos fichiers et comprendre les liens entre **Front-end / Back-end / JSON**.

Voici une représentation simplifiée de l'architecture développée dans ce bootstrap.



Objectif

À la fin de ce bootstrap, vous devriez être capable de :

- ✓ structurer un projet PHP en **MVC / POO**
- ✓ créer vos **modèles, repositories, controllers et fichier index.php**
- ✓ exposer une **API REST JSON**
- ✓ connecter un front simple

Prérequis

Avant de commencer, assurez-vous d'avoir :

- ✓ un environnement PHP fonctionnel (PHP 8 ou plus recommandé)
- ✓ des bases en HTML / CSS / JavaScript
- ✓ des bases en PHP orienté objet
- ✓ un serveur local (Apache, Nginx, serveur PHP intégré...)

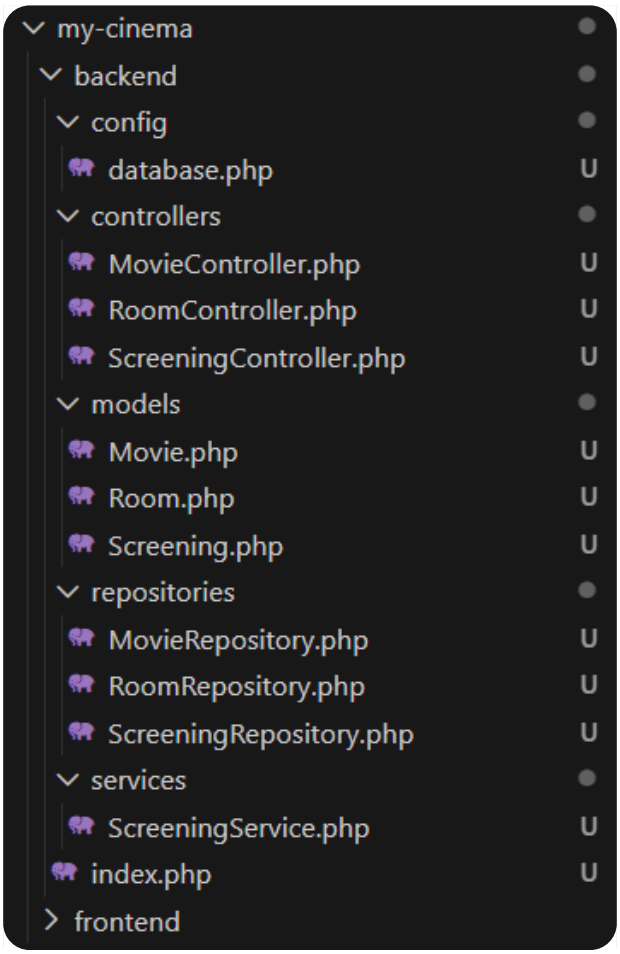
Structure minimale du projet

Il est important de bien séparer le frontend et le backend de votre projet. Commencez par créer chacun de ces deux dossiers.

Le dossier `frontend` contiendra le code **HTML, CSS, JavaScript** de votre projet, ainsi que les ressources nécessaires (images, polices éventuelles, etc).
Le dossier `backend` contiendra exclusivement le code **PHP et MySQL** du projet.

Dans ce bootstrap, nous nous concentrons sur la construction de l'architecture backend. L'architecture qui sera mise en place est une proposition seulement, il ne s'agit pas d'une réponse unique au projet. Vous pouvez la faire évoluer selon vos besoins.

Voici à quoi ressemblera votre arborescence après la réalisation des différentes étapes de ce bootstrap :



Point d'entrée de l'API

Le backend doit avoir **un seul et unique** point d'entrée. Cela signifie que tous vos appels API feront référence au même fichier PHP.

Créez un fichier **index.php** dans votre dossier backend. Il fera office de point d'entrée.

C'est lui qui :

- ✓ reçoit les requêtes HTTP
- ✓ instancie le controller correspondant (en fonction d'un paramètre `$\$$_GET` d'URL)

Par exemple, votre frontend pourra appeler l'URL `http://localhost/my-cinema/index.php?action=list_movies` pour récupérer la liste des films ou bien l'URL `http://localhost/my-cinema/index.php?action=add_room` pour créer une nouvelle salle.

Le fichier `index.php` doit donc récupérer le paramètre `action` et appeler le contrôleur correspondant, chargé de réaliser l'action en question.

Exemple simple :

```php
$request = $_GET['action'] ?? ''; // Récupération du paramètre d'URL action indiquant la
    route API
switch($request) {
    case 'list_movies':
        $controller = new MovieController();
        $controller->list(); // Appel de la méthode list du MovieController
        break;
    default:
        echo json_encode(["error" => "Action not found"]);
        break;
}
```

Controllers

Les controllers sont responsables de :

- ✓ recevoir les données
- ✓ appeler les **repositories ou services**
- ✓ renvoyer des réponses JSON

Ils peuvent être placés dans un dossier `controllers` de votre backend.

Afin d'avoir une architecture propre, il est recommandé de les créer sous forme de classe. Chaque route API aura une méthode associée dans l'un des contrôleurs.

Par exemple, pour MovieController :

```
class MovieController {
    private $repository;
    public function __construct() {
        $this->repository = new MovieRepository(); // repository créé par la suite
    }

    public function list() { // Méthode appelée par le fichier index.php
        echo json_encode($this->repository->getAll());
    }

    // Autres méthodes correspondant aux autres routes API.
}
```

À noter : toute la logique métier complexe (ex : conflits de séances) doit être dans un **service**, pas directement dans le controller.

Configuration de la base de données

Afin d'éviter la répétition du code, vous allez centraliser la connexion à la base de données pour la réutiliser partout où cela est nécessaire.

1. Créez le fichier /config/database.php dans votre backend
2. Configurez la connexion PDO, par exemple :

```
$dsn = "mysql:host=localhost;dbname=my_cinema;charset=utf8mb4";
$user = "root";
$pass = "";
$pdo = new PDO($dsn, $user, $pass, [
    PDO::ATTR_ERRMODE => PDO::ERRMODE_EXCEPTION,
    PDO::ATTR_DEFAULT_FETCH_MODE => PDO::FETCH_ASSOC
]);
```

Vous pouvez bien sûr créer une fonction ou une classe selon vos préférences.

Modèles (Models)

Chaque modèle représente une entité métier avec ses propriétés. Pour chaque table en base de données, une entité (un modèle) sera créée et reprendra les différentes colonnes de la table.

Exemple avec le fichier `Movie.php` pour les films :

```
class Movie {
    public $id;
```

```
    public $title;
    public $description;
    public $duration;
    public $release_year;
    public $genre;
    public $director;
}
```

Le Model ne contient **aucune logique SQL**. Il décrit seulement la structure des données. Vous pouvez créer un dossier `models` pour regrouper tous vos modèles.

Repositories

Les repositories contiennent **tout l'accès à la base de données** via PDO. Un repository par table sera créé, et contiendra une méthode par échange avec la base de données (création, modification, suppression, récupération).

Les repositories sont appelés par les contrôleurs comme l'exemple précédent.

Par exemple, pour les films, vous pouvez créer un fichier `MovieRepository.php` contenant entre autres :

```
class MovieRepository {
    private $pdo;
    public function __construct() {
        global $pdo;
        $this->pdo = $pdo;
    }

    public function getAll() {
        $stmt = $this->pdo->query("SELECT * FROM movies");
        return $stmt->fetchAll(PDO::FETCH_CLASS, "Movie");
    }

    public function add(Movie $movie) {
        $stmt = $this->pdo->prepare("INSERT INTO movies (title, description, duration,
            release_year, genre, director) VALUES (?, ?, ?, ?, ?, ?)");
        $stmt->execute([
            $movie->title,
            $movie->description,
            $movie->duration,
            $movie->release_year,
            $movie->genre,
            $movie->director
        ]);
    }

    // Méthodes update, delete, find, etc similaires
}
```

Regroupez tous vos repositories dans un dossier `repositories`.

Vous ne devriez avoir **aucune** requête SQL en dehors des repositories.

Services

Les services contiennent la **logique métier** indépendante de la base et seront appelés par vos contrôleurs. Vous pouvez les regrouper dans un dossier `services`.

Vous pouvez par exemple créer un fichier `ScreeningService.php` pour gérer la logique liée aux séances : vérifier qu'il n'y a pas de chevauchement entre deux séances, quelles salles sont disponibles etc.

A la différence des models et des repositories, ne créez de service que si cela est nécessaire. Vous n'aurez pas de services associés à chacune de vos entités.

Introduction au frontend

Vous créerez votre frontend HTML/CSS/JS dans votre dossier `frontend`.
A chaque fois que cela est nécessaire, par exemple pour la récupération de films, l'enregistrement d'une séance, la suppression d'une salle, etc, vous appelerez votre API avec **fetch** vers **index.php?action=xxx**.

Les données échangées avec l'API (envoyées ou reçues) seront en **json**.

Exemple JS :

```
fetch("index.php?action=list_movies")
    .then(res => res.json())
    .then(data => console.log(data));
```

C'est le **seul lien entre le front et le back**.

Bonnes pratiques

- ✓ Toujours valider les entrées côté serveur à minima
- ✓ Séparer : controller, repository, model, service
- ✓ Retourner uniquement du JSON au front
- ✓ Ne jamais exposer PDO ou SQL directement au front
- ✓ N'oubliez pas d'implémenter un soft delete (`active`) pour les séances comme vu dans le projet

v1