

# GitHub



<https://github.com/Alexander-Krikun-IU/movie-reviews>



# Web-Anwendung zur Organisation einer Datenbank mit **FILMKRITIKEN.**

Luca Halbritter, Alexander Krikun

```
<p> Programmiert mittels PHP,HTML  
& CSS </p>
```

# INHALTSVERZEICHNIS



01

Vorstellung des  
Projekts und  
Überblick über die  
Hauptkomponenten



02

Grundlegenden  
Systemkomponente



03

Überblick über die  
Webseiten Architektur



04

Abschluss:  
Erweiterungen, Fazit,  
Fragerunde



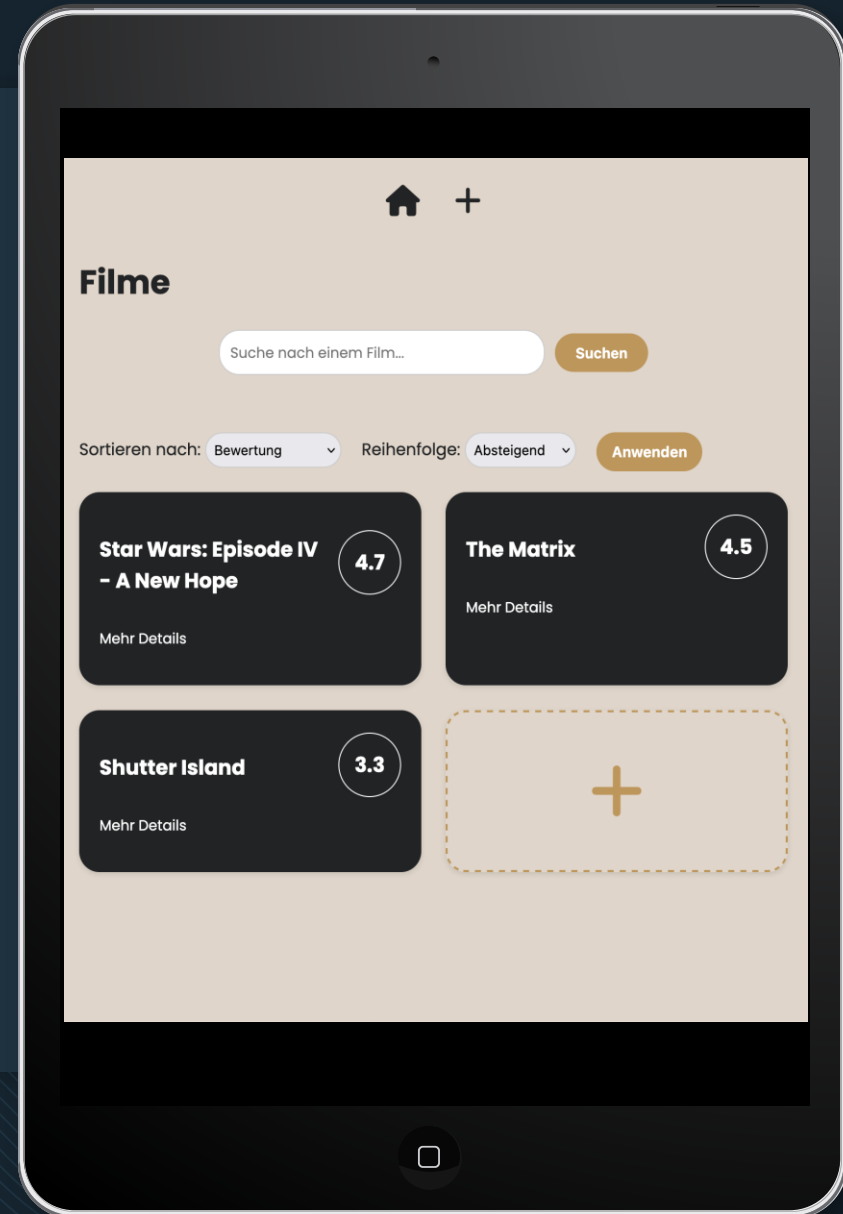


# 01 VORSTELLUNG DES PROJEKTS

`<p>` Eine kleine *Vorstellung* des Projekts, veranschaulichung  
des Klassendiagramms und die Aufteilung `</p>`

# WAS MACHT UNSERE ANWENDUNG?

<p> Wir haben eine Anwendung nach dem MVC **Entwurfsmuster** entwickelt, mit der Benutzer Filme hinzufügen, durchsuchen, bewerten und sortieren können. </p>



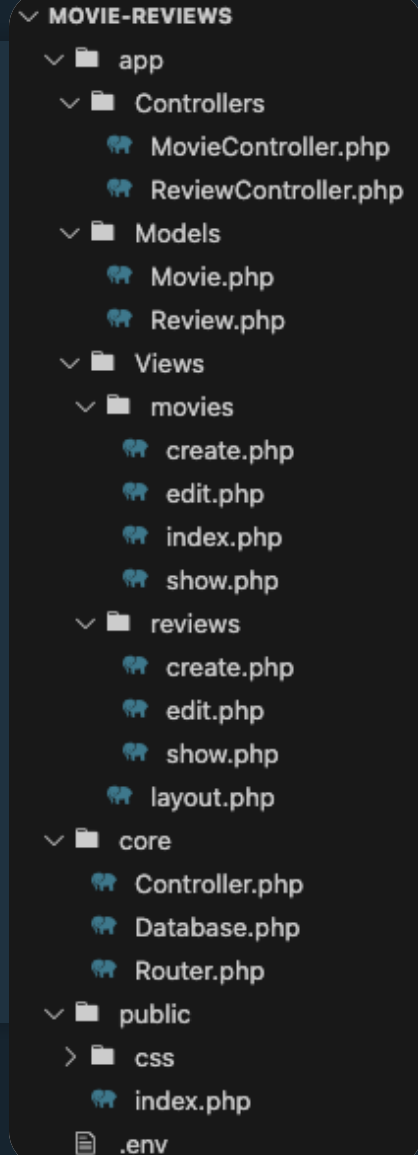
# AUFTEILUNG DER APPLIKATION

<p> Backend-Architektur </p>

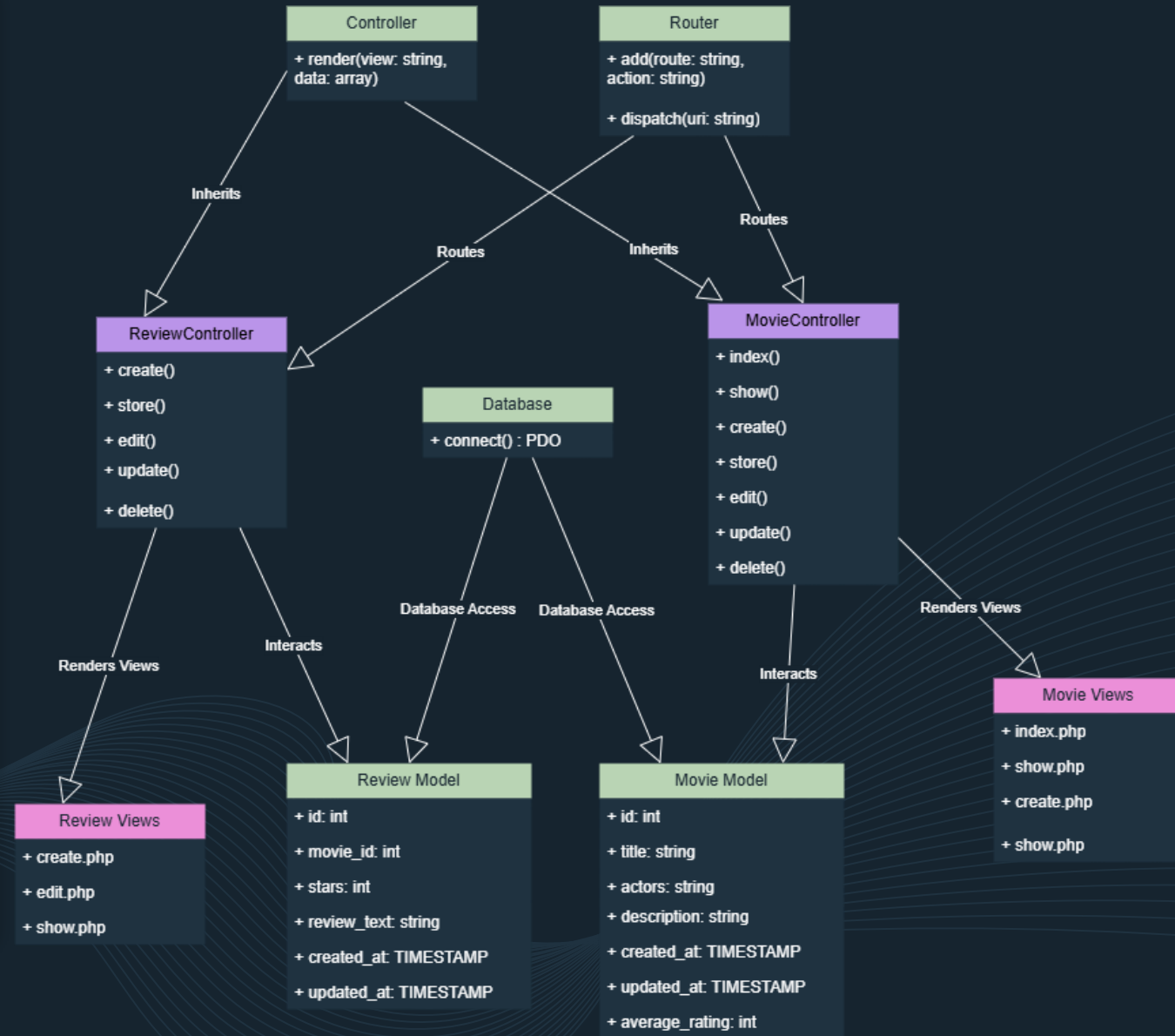
- Datenbank
- Models
- Controller
- Router

<p> Frontend-Logik </p>

- Views
- Benutzerinteraktion
- Responsivität

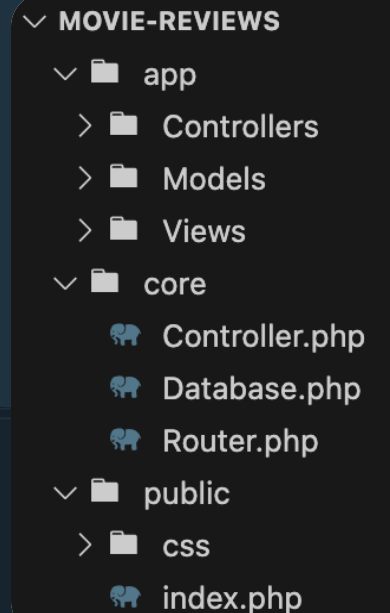


# ÜBERBLICK ÜBER DIE HAUPT- KOMPONENTE



# 02 GRUNDLEGENDE SYSTEMKOMPONENTE

<p> Datenbank, Controller, Router , Index </p>





# <h1> PostgreSQL Datenbank </h1>

```
--Tabelle für Filme erstellen
CREATE TABLE movies (
  --Eindeutige ID für jeden Film
  id SERIAL PRIMARY KEY,
  --Titel des Films
  title VARCHAR(255) NOT NULL,
  --Liste der Darsteller
  actors TEXT NOT NULL,
  --Beschreibung des Films
  description TEXT,
  --Erstellungszeitpunkt
  created_at TIMESTAMP DEFAULT
CURRENT_TIMESTAMP,
  --Letzte Aktualisierung
  updated_at TIMESTAMP DEFAULT
CURRENT_TIMESTAMP
);
```

```
-- Tabelle für Bewertungen erstellen
CREATE TABLE reviews (
  --Eindeutige ID für jede Bewertung
  id SERIAL PRIMARY KEY,
  --Fremdschlüssel zu movies.id
  movie_id INT NOT NULL,
  --Bewertung in Sternen (1-5)
  stars INT CHECK (stars BETWEEN 1 AND 5),
  --Text der Bewertung
  review_text TEXT,
  --Erstellungszeitpunkt
  created_at TIMESTAMP
  DEFAULT CURRENT_TIMESTAMP,
  --Letzte Aktualisierung
  updated_at TIMESTAMP DEFAULT
CURRENT_TIMESTAMP,
  --Beziehungen
  FOREIGN KEY (movie_id) REFERENCES
  movies(id) ON DELETE CASCADE
);
```

# <h1> core/Database.php </h1>

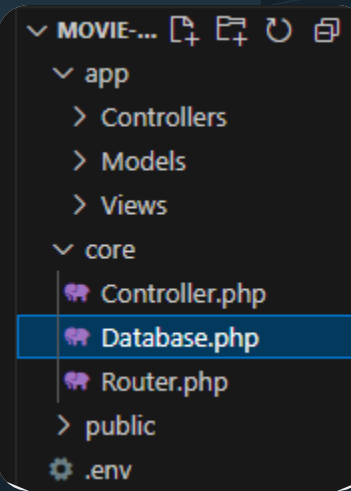
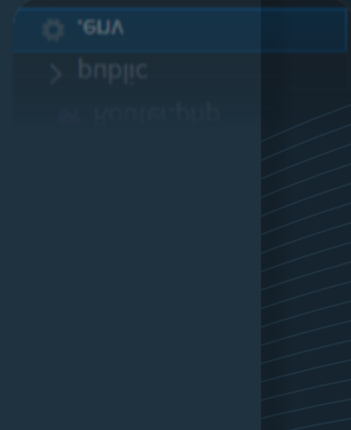
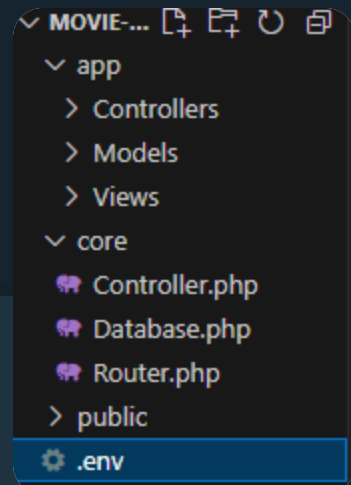
Stellt eine Verbindung zur Datenbank her

```
<?php
class Database {
    // Statische Eigenschaft für die PDO-Verbindung
    private static $pdo;
    public static function connect() {
        // Prüfen, ob die Verbindung bereits besteht
        if (!self::$pdo) {
            // Umgebungsvariablen aus einer .env-Datei laden
            $env = parse_ini_file(__DIR__ . '/../.env');

            // Datenbank-DSN (Data Source Name) erstellen
            $dsn = "pgsql:host={$env['DB_HOST']};dbname={$env['DB_NAME']}";
            try {
                // Neue PDO-Instanz erstellen und mit den Umgebungsvariablen verbinden
                self::$pdo = new PDO($dsn, $env['DB_USER'], $env['DB_PASS']);

                // PDO-Fehlermodus auf Exceptions setzen
                self::$pdo->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);
            } catch (PDOException $e) {
                // Fehler beim Verbindungsaufbau behandeln
                die("Datenbankverbindung fehlgeschlagen: " . $e->getMessage());
            }
        }
        // Verbindung zurückgeben
        return self::$pdo;
    }
}
```

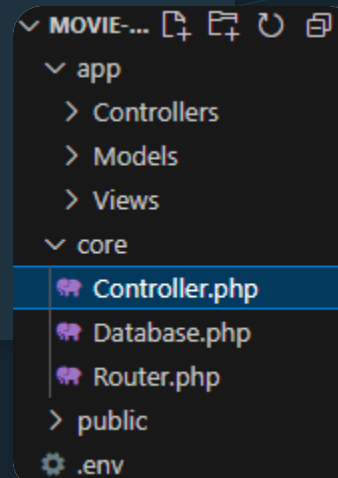
```
DB_HOST=localhost
DB_NAME=finaltest
DB_USER=postgres
DB_PASS=admin
```



# <h1> core/Controller.php </h1>

<?php

```
class Controller {  
    public function render($view, $data = []) {  
        extract($data); // Wandelt das assoziative Array $data in einzelne Variablen um  
        require __DIR__ . '/../app/views/' . $view . '.php'; // Bindet die View-Datei ein  
    }  
}
```



# <h1> core/Router.php </h1>

```
class Router {
    // Array zur Speicherung der Routen und ihrer zugehörigen Aktionen
    private $routes = [];
    public function add($route, $action) {
        $this->routes[$route] = $action;
    }

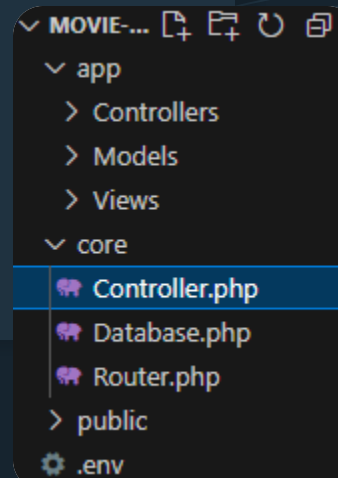
    public function dispatch($uri) {
        foreach ($this->routes as $route => $action) {
            // Platzhalter (z. B. {id}) durch reguläre Ausdrücke ersetzen
            $routePattern = preg_replace('/\{[a-z_]+\}/', '(\d+)', $route);
            $routePattern = '#^' . $routePattern . '$#';

            // Prüfen, ob die URI mit der aktuellen Route übereinstimmt
            if (preg_match($routePattern, $uri, $matches)) {
                array_shift($matches); // Das erste Element ($matches[0]) entfernen, da es die gesamte Übereinstimmung enthält
                [$controller, $method] = explode('@', $action); // Controller und Methode aus der Aktion extrahieren

                // Controller-Datei einbinden
                require_once __DIR__ . '/../app/controllers/' . $controller . '.php';

                // Controller-Instanz erstellen und Methode aufrufen
                $controllerInstance = new $controller();
                return call_user_func_array([$controllerInstance, $method], $matches); // Argumente an die Methode übergeben
            }
        }

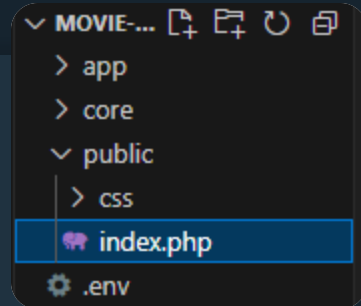
        // Wenn keine Route übereinstimmt, 404-Fehler ausgeben
        http_response_code(404);
        echo "404 - Page Not Found";
    }
}
```



# <h1> public/index.php </h1>

```
<?php
```

```
require_once __DIR__ . '/../core/Router.php'; // Router-Klasse einbinden
$router = new Router(); // Neue Router-Instanz erstellen
// Definieren der Routen und der zugehörigen Aktionen
$router->add('/', 'MovieController@index'); // Startseite
$router->add('/movies/create', 'MovieController@create'); // Seite zum Erstellen eines neuen Films
$router->add('/movies/store', 'MovieController@store'); // Route zum Speichern eines neuen Films
$router->add('/movies/edit/{id}', 'MovieController@edit'); // Seite zum Bearbeiten eines Films
$router->add('/movies/update/{id}', 'MovieController@update'); // Route zum Aktualisieren eines Films
$router->add('/movies/delete/{id}', 'MovieController@delete'); // Route zum Löschen eines Films
$router->add('/movies/show/{id}', 'MovieController@show'); // Seite mit den Details eines Films
$router->add('/reviews/create/{movie_id}', 'ReviewController@create'); // Route zum Erstellen einer Bewertung
$router->add('/reviews/edit/{id}', 'ReviewController@edit'); // Seite zum Bearbeiten einer Bewertung
$router->add('/reviews/update/{id}', 'ReviewController@update'); // Route zum Aktualisieren einer Bewertung
$router->add('/reviews/delete/{id}', 'ReviewController@delete'); // Route zum Löschen einer Bewertung
// Aktuelle URI aus dem Query-Parameter 'route' lesen, Standard ist die Startseite ('/')
$uri = isset($_GET['route']) ? $_GET['route'] : '/';
// Dispatcher aufrufen, um die Anfrage an die passende Route weiterzuleiten
$router->dispatch($uri);
```



GUU  
rugerbub



# 03 Webseiten Architektur

`<p> Models, Controller, Views </p>`

# <h1> review/layout.php </h1>

```
<!DOCTYPE html>
<html lang="en">
<head>
<meta charset="UTF-8"> <!-- Zeichencodierung setzen -->
<meta name="viewport" content="width=device-width, initial-scale=1.0"> <!-- Responsives
    Design ermöglichen -->
<title>Film Bewertungen</title> <!-- Titel der Seite -->
<!-- Google Fonts einbinden -->
<link
    href="https://fonts.googleapis.com/css2?family=Poppins:wght@400;600;700&display=swap" rel="stylesheet">
<!-- Font Awesome Icons einbinden -->
<link href="https://cdnjs.cloudflare.com/ajax/libs/font-awesome/6.0.0/css/all.min.css"
    rel="stylesheet">
<!-- Eigene CSS-Datei einbinden -->
<link rel="stylesheet" href="/movie-reviews/public/css/styles.css">
</head>
```

```
<body class="layout mobile-wrapper"> <!-- Hauptlayout-Klasse für die Seite -->
<header>
<nav>
<!-- Navigation mit Links -->
<ul>
<!-- Link zur Startseite mit Haus-Icon -->
<li><a href="/movie-reviews/public/index.php?route="><i class="fa-solid fa-house"></i></a></li>
<!-- Link zum Erstellen eines neuen Films mit Plus-Icon -->
<li><a href="/movie-reviews/public/index.php?route=/movies/create"><i class="fas fa-plus"></i></a></li>
</ul>
</nav>
</header>
<main>
<!-- Dynamischer Inhalt der Seite -->
<?= $content ?? " ">
</main>
</body>
</html>
```

▼ **MOVIE-REVIEWS**

- ▼ **app**
  - > **Controllers**
  - > **Models**
  - ▼ **Views**
    - > **movies**
    - > **reviews**
    - layout.php**
- > **core**
- > **public**



# 01 Filmliste

`<p> Movie.php,MovieController.php,index.php </p>`



# <h1> Models/Movie.php(all) </h1>

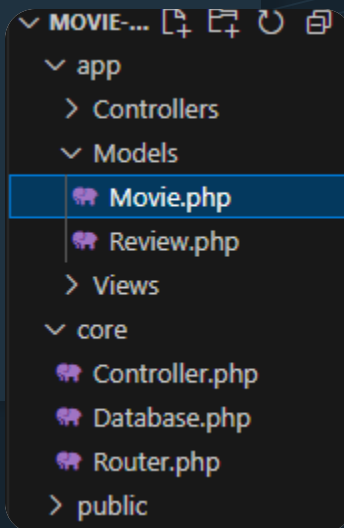
```
// Methode zum Abrufen aller Filme, optional sortiert
public static function all($sortBy = 'average_rating', $order = 'DESC') {
    $db = Database::connect(); // Verbindung zur Datenbank herstellen

    // Erlaubte Sortier- und Reihenfolge-Werte
    $allowedSortBy = ['average_rating', 'title', 'created_at'];
    $allowedOrder = ['ASC', 'DESC'];

    // Überprüfen, ob Sortier- und Reihenfolge-Werte gültig sind
    $sortBy = in_array($sortBy, $allowedSortBy) ? $sortBy : 'average_rating';
    $order = in_array($order, $allowedOrder) ? $order : 'DESC';

    // SQL-Abfrage: Filme und durchschnittliche Bewertung abrufen, sortiert nach den angegebenen Kriterien
    $sql = "
        SELECT movies.*,
               COALESCE(AVG(reviews.stars), 0) AS average_rating
        FROM movies
        LEFT JOIN reviews ON movies.id = reviews.movie_id
        GROUP BY movies.id
        ORDER BY $sortBy $order
    ";

    $stmt = $db->query($sql); // Abfrage ausführen
    return $stmt->fetchAll(PDO::FETCH_ASSOC); // Ergebnisse als assoziatives Array zurückgeben
}
```



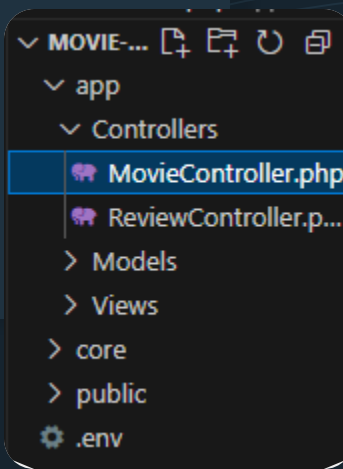
# <h1> MovieController.php(index) </h1>

```
// Methode zum Anzeigen der Filmübersicht
public function index() {
    // Suchanfrage aus URL holen, Standard ist leer
    $searchQuery = $_GET['search'] ?? '';
    // Sortierkriterium aus URL holen, Standard ist 'avg_rating'
    $sortBy = $_GET['sort_by'] ?? 'avg_rating';
    // Sortierreihenfolge aus URL holen, Standard ist 'DESC'
    $order = $_GET['order'] ?? 'DESC';

    // Alle Filme aus der Datenbank holen, sortiert nach $sortBy und $order
    $movies = Movie::all($sortBy, $order);

    // Wenn eine Suchanfrage vorliegt, Filme filtern
    if (!empty($searchQuery)) {
        $movies = array_filter($movies, function ($movie) use ($searchQuery) {
            // Titel durchsuchen (Case-Insensitive)
            return isset($movie['title']) && is_string($movie['title']) && stripos($movie['title'], $searchQuery) !== false;
        });
    }

    // View rendern und gefilterte Filme übergeben
    return $this->render('movies/index', [
        'movies' => $movies,
        'searchQuery' => $searchQuery,
        'sortBy' => $sortBy,
        'order' => $order,
    ]);
}
```



# <h1> movies/index.php </h1>

## Anzeige der Suchleiste

```
<!-- Suchleiste -->
<div class="search-bar">
  <form method="GET" action="/movie-reviews/public/index.php">
    <!-- Suchfeld für Filme -->
    <input type="text" name="search" class="search-input" placeholder="Suche nach
      einem Film..."
      value="<?= htmlspecialchars($_GET['search'] ?? '') ?>">
    <button type="submit" class="btn btn-search">Suchen</button>
  </form>
</div>
```

## Anzeige der Sortierung

```
<div class="sorting-options">
  <form method="GET" action="index.php" class="sort-form">
    <!-- Sortieren nach Feld -->
    <div class="form-group me-3">
      <label for="sort_by" class="form-label">Sortieren nach:</label>
      <select name="sort_by" id="sort_by" class="form-select rounded"
        onchange="document.getElementById('sortForm').submit()">
        <option value="average_rating" <?= ($sortBy === 'average_rating') ? 'selected' : ''
          ?>>Bewertung</option>
        <option value="title" <?= ($sortBy === 'title') ? 'selected' : '' ?>>Titel</option>
        <option value="created_at" <?= ($sortBy === 'created_at') ? 'selected' : '' ?>>Hinzugefügt am</option>
      </select>
    </div>
    <!-- Reihenfolge auswählen -->
    <div class="form-group me-3">
      <label for="order" class="form-label">Reihenfolge:</label>
      <select name="order" id="order" class="form-select rounded"
        onchange="document.getElementById('sortForm').submit()">
        <option value="DESC" <?= ($order === 'DESC') ? 'selected' : '' ?>>Absteigend</option>
        <option value="ASC" <?= ($order === 'ASC') ? 'selected' : '' ?>>Aufsteigend</option>
      </select>
    </div>
    <button type="submit" class="btn btn-primary rounded">Anwenden</button>
  </form>
</div>
```

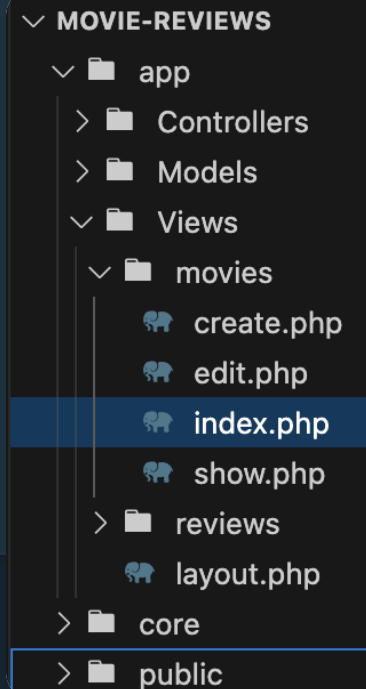
- MOVIE-REVIEWS
  - app
    - Controllers
    - Models
    - Views
      - movies
        - create.php
        - edit.php
        - index.php
        - show.php
      - reviews
      - layout.php
    - core
    - public

# <h1> movies/index.php </h1>

## Darstellung der Filme

```
<div class="movie-grid">
<?php if (!empty($movies) && is_array($movies)): // Überprüfen, ob Filme vorhanden sind ?>
<?php foreach ($movies as $movie): // Durch die Liste der Filme iterieren ?>
<div class="movie-card">
<!-- Link zu den Filmdetails -->
<a href="/movie-reviews/public/index.php?route=/movies/show/<?= $movie['id'] ?>" class="btn-card">
<div class="card-header">
<h3><?= htmlspecialchars($movie['title']) ?></h3> <!-- Filmmname anzeigen -->
<div class="rating-circle">
<?php
// Durchschnittliche Bewertung anzeigen oder "N/A", wenn keine Bewertung existiert
if (!empty($movie['average_rating'])) {
echo number_format($movie['average_rating'], 1);
} else {
echo 'N/A';
}
?>
</div>
</div>
<p>Mehr Details</p>
</a>
</div>
<?php endforeach; ?>
<?php else: // Wenn keine Filme gefunden wurden ?>
<p>Kein Film gefunden der deinen Suchanforderungen entspricht</p>
<?php endif; ?>

<!-- Karte zum Hinzufügen eines neuen Films -->
<div class="movie-card add-movie-card">
<a href="/movie-reviews/public/index.php?route=/movies/create">
<i class="fas fa-plus"></i>
</a>
</div>
```



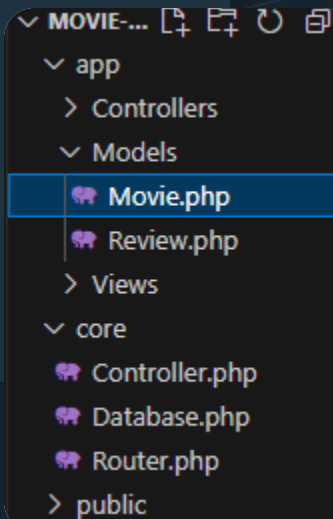


# 02 Film hinzufügen

`<p> Movie.php,MovieController.php,movies/create.php </p>`

# <h1> Movie.php(create) </h1>

```
// Methode zum Erstellen eines neuen Films
public static function create($data) {
    $db = Database::connect(); // Verbindung zur Datenbank herstellen
    $sql = "INSERT INTO " . static::$table . " (title, actors, description,
created_at)
        VALUES (:title, :actors, :description, :created_at)"; // SQL-Befehl zum
Einfügen eines neuen Films
    $stmt = $db->prepare($sql); // SQL-Anweisung vorbereiten
    $stmt->execute([
        ':title' => $data['title'], // Titel binden
        ':actors' => $data['actors'], // Schauspieler binden
        ':description' => $data['description'], // Beschreibung binden
        ':created_at' => date('Y-m-d H:i:s'), // Aktuellen Zeitstempel binden
    ]);
}
```



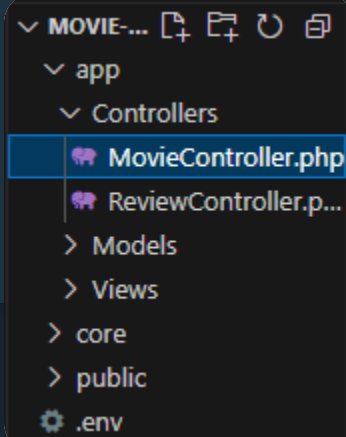
# <h1> MovieController.php(create) </h1>

```
// Methode zum Anzeigen des Formulars für einen neuen Film
public function create() {
    $this->render('movies/create'); // View rendern
}

// Methode zum Speichern eines neuen Films
public function store() {
    if ($_SERVER['REQUEST_METHOD'] === 'POST') { // Überprüfen, ob die Anfrage eine POST-Anfrage ist
        $title = $_POST['title'] ?? null; // Titel aus Anfrage holen
        $actors = $_POST['actors'] ?? null; // Schauspieler aus Anfrage holen
        $description = $_POST['description'] ?? null; // Beschreibung aus Anfrage holen

        if ($title && $actors) { // Prüfen, ob Titel und Schauspieler vorhanden sind
            // Neuen Film in der Datenbank speichern
            Movie::create([
                'title' => $title,
                'actors' => $actors,
                'description' => $description,
            ]);

            // Zur Hauptseite weiterleiten
            header('Location: /movie-reviews/public/index.php?route=');
            exit();
        } else {
            echo "Title and Actors are required!"; // Fehlermeldung bei fehlenden Feldern
        }
    } else {
        echo "Invalid request method."; // Fehlermeldung bei ungültiger Anfrage
    }
}
```



# <h1> movies/create.php

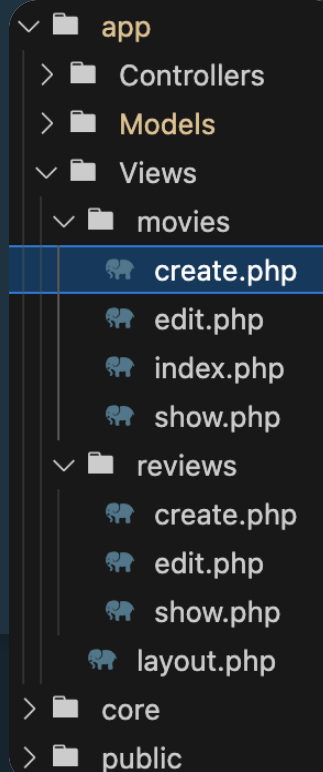
## Anlegen der Filme

```
<?php ob_start(); // Output-Buffering starten, um den Inhalt in einer Variablen zu speichern ?>
<div class="movie-create-box">
<h1>Füge einen neuen Film hinzu</h1>
<form method="POST" action="/movie-reviews/public/index.php?route=/movies/store" class="movie-form">
<!-- Eingabefeld für den Titel des Films -->
<label for="title">Titel:</label>
<input type="text" id="title" name="title" class="form-input" required>

<!-- Eingabefeld für die Schauspieler des Films -->
<label for="actors">Schauspieler:</label>
<textarea id="actors" name="actors" class="form-input" required></textarea>

<!-- Eingabefeld für die Beschreibung des Films -->
<label for="description">Beschreibung:</label>
<textarea id="description" name="description" class="form-input"></textarea>

<!-- Button zum Absenden des Formulars -->
<button type="submit" class="btn btn-submit">Speichern</button>
</form>
</div>
<?php
$content = ob_get_clean(); // Puffer leeren und Inhalt in der Variable $content speichern
?>
<?php include __DIR__ . '/../layout.php'; // Hauptlayout einbinden, das $content verwendet ?>
```







# 03 Film Details

`<p> Movie.php(find), MovieController.php(show), show.php </p>`

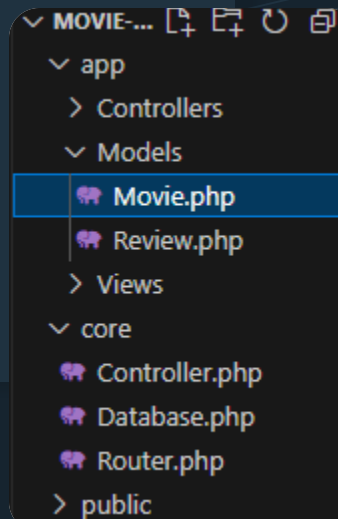
# <h1> Movie.php(find) </h1>

```
// Methode zum Finden eines Films anhand der ID
public static function find($id) {
    $db = Database::connect(); // Verbindung zur Datenbank herstellen
    $stmt = $db->prepare('SELECT * FROM movies WHERE id = :id'); // SQL-Abfrage vorbereiten
    $stmt->bindParam(':id', $id, PDO::PARAM_INT); // ID binden
    $stmt->execute(); // Abfrage ausführen
    $movieData = $stmt->fetch(PDO::FETCH_ASSOC); // Ergebnis abrufen

    if (!$movieData) {
        return null; // Wenn kein Film gefunden wurde, null zurückgeben
    }

    // Neues Movie-Objekt erstellen und Felder setzen
    $movie = new self();
    $movie->id = $movieData['id'];
    $movie->title = $movieData['title'];
    $movie->actors = $movieData['actors'];
    $movie->description = $movieData['description'];
    $movie->created_at = $movieData['created_at'];
    $movie->updated_at = $movieData['updated_at'];

    return $movie; // Movie-Objekt zurückgeben
}
```

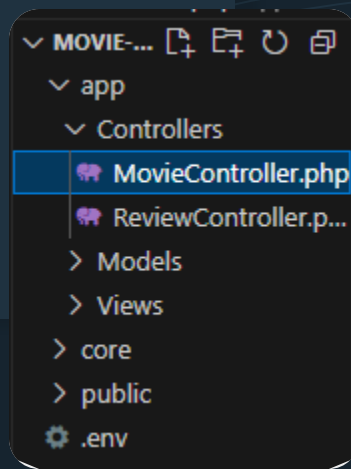


# <h1> MovieController.php(show) </h1>

```
// Methode zum Anzeigen der Details eines Films
public function show($id) {
    $movie = Movie::find($id); // Film anhand der ID suchen
    if (!$movie) {
        throw new Exception("Movie not found"); // Fehler werfen, wenn Film nicht gefunden wird
    }

    // Alle zugehörigen Rezensionen abrufen
    $reviews = Review::allByMovie($id);

    // Detail-View rendern und Film sowie Rezensionen übergeben
    $this->render('movies/show', [
        'movie' => $movie,
        'reviews' => $reviews,
    ]);
}
```



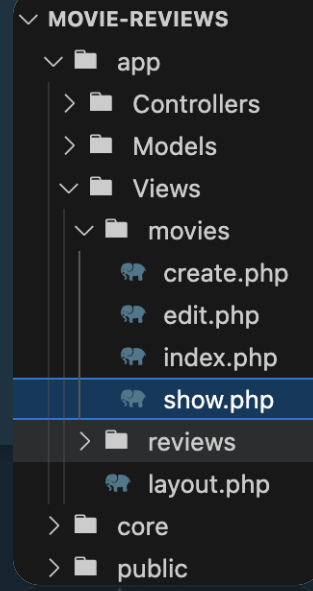
# <h1> movies/show.php </h1>

## Anzeige der Film-Details

```
<h1><?= htmlspecialchars($movie->title) ?></h1>
<div class="movie-details">
<!-- Darsteller anzeigen -->
<p><strong>Darsteller:</strong> <?= htmlspecialchars($movie->actors) ?></p>
<!-- Beschreibung des Films anzeigen -->
<p><strong>Beschreibung:</strong> <?= htmlspecialchars($movie->description)
?></p>
<!-- Erstellungsdatum des Films anzeigen -->
<p><strong>Erstellt am:</strong> <?= (new DateTime($movie->created_at))-
>format('Y-m-d H:i:s') ?></p>
<?php
// Prüfen, ob der Film bearbeitet wurde, und das Bearbeitungsdatum anzeigen
$createdAt = (new DateTime($movie->created_at))->format('Y-m-d H:i:s');
$updatedAt = (new DateTime($movie->updated_at))->format('Y-m-d H:i:s');
if ($updatedAt !== $createdAt):
?>
<p><strong>Bearbeitet am:</strong> <?= (new DateTime($movie->updated_at))-
>format('Y-m-d H:i:s') ?></p>
<?php endif; ?>
</div>
```

## Aktionen für den Film und Rezensionen

```
<!-- Aktionen für den Film -->
<div class="movie-actions">
<!-- Bearbeiten-Button -->
<a href="/movie-reviews/public/index.php?route=/movies/edit/<?= $movie->id
?>" class="btn">Bearbeiten</a>
<!-- Löschen-Button mit Bestätigungsdialog -->
<a href="/movie-reviews/public/index.php?route=/movies/delete/<?= $movie-
>id ?>"
class="btn btn-danger"
onclick="return confirm('Sind Sie sicher, dass Sie diesen Film löschen
möchten?');">Löschen</a>
</div>
</div>
```





# 04 Film bearbeiten

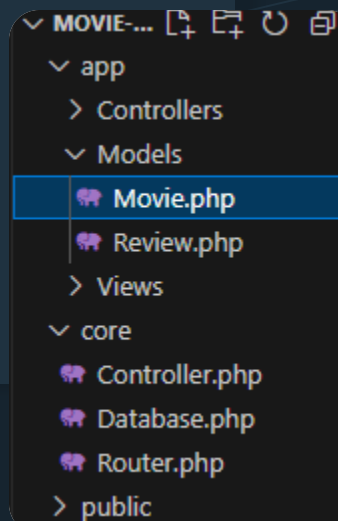
`<p> Movie.php,MovieController.php,movies/edit.php </p>`

# <h1> Movie.php(update) </h1>

```
// Methode zum Aktualisieren eines Films
public function update($data) {
    $db = Database::connect(); // Verbindung zur Datenbank herstellen
    $sql = 'UPDATE movies
        SET title = :title,
            actors = :actors,
            description = :description,
            updated_at = :updated_at
        WHERE id = :id'; // SQL-Befehl zum Aktualisieren eines Films
    $stmt = $db->prepare($sql); // SQL-Anweisung vorbereiten
    $stmt->bindParam(':title', $data['title'], PDO::PARAM_STR); // Titel binden
    $stmt->bindParam(':actors', $data['actors'], PDO::PARAM_STR); // Schauspieler binden
    $stmt->bindParam(':description', $data['description'], PDO::PARAM_STR); // Beschreibung binden
    $stmt->bindParam(':updated_at', $updatedAt, PDO::PARAM_STR); // Aktualisierungszeit binden
    $stmt->bindParam(':id', $this->id, PDO::PARAM_INT); // ID binden

    // Setze den aktuellen Zeitstempel
    $updatedAt = date('Y-m-d H:i:s');

    return $stmt->execute(); // SQL-Befehl ausführen
}
```



# <h1> MovieController.php(edit) </h1>

```
// Methode zum Bearbeiten eines Films
public function edit($id) {
    $movie = Movie::find($id); // Film anhand der ID suchen
    if (!$movie) {
        throw new Exception("Movie not found"); // Fehler werfen, wenn Film nicht gefunden wird
    }

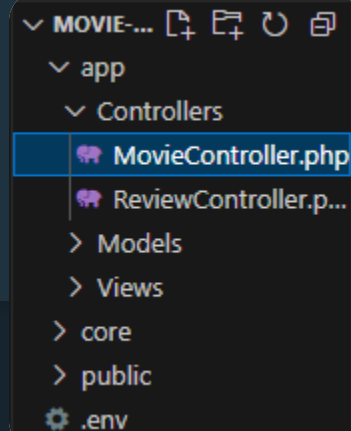
    // Bearbeitungs-View rendern und Film übergeben
    $this->render('movies/edit', ['movie' => $movie]);
}
```

```
// Methode zum Aktualisieren eines Films
public function update($id) {
    $movie = Movie::find($id); // Film anhand der ID suchen

    if (!$movie) { // Prüfen, ob der Film existiert
        echo "Movie not found.";
        exit;
    }

    // Aktualisierte Daten aus Anfrage holen
    $data = [
        'title' => $_POST['title'] ?? $movie->title,
        'actors' => $_POST['actors'] ?? $movie->actors,
        'description' => $_POST['description'] ?? $movie->description,
    ];

    // Film aktualisieren und zur Detailansicht weiterleiten
    if ($movie->update($data)) {
        header("Location: /movie-reviews/public/index.php?route=/movies/show/$id");
        exit;
    } else {
        echo "Failed to update the movie."; // Fehlermeldung bei fehlgeschlagenem Update
    }
}
```



# <h1> movies/edit.php </h1>

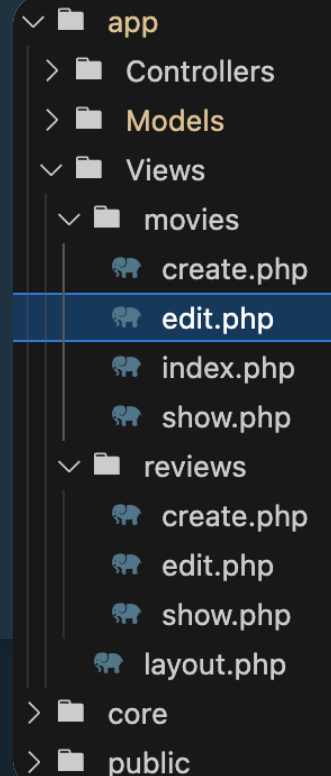
## Bearbeiten der Filme

```
<?php ob_start(); // Output-Buffering starten, um den Inhalt in einer Variablen zu speichern ?>
<h1>Bearbeite einen Film</h1>
<?php if ($movie): // Überprüfen, ob der Film existiert ?>
<form action="/movie-reviews/public/index.php?route=/movies/update/<?= $movie->id ?>" method="POST" class="movie-form">
<!-- Eingabefeld für den Titel des Films -->
<label for="title">Titel:</label>
<input type="text" id="title" name="title" class="form-input"
value="<?= htmlspecialchars($movie->title ?? '') ?>" required>

<!-- Eingabefeld für die Darsteller des Films -->
<label for="actors">Darsteller:</label>
<input type="text" id="actors" name="actors" class="form-input"
value="<?= htmlspecialchars($movie->actors ?? '') ?>" required>

<!-- Eingabefeld für die Beschreibung des Films -->
<label for="description">Beschreibung:</label>
<textarea id="description" name="description" class="form-input"><?= htmlspecialchars($movie->description ?? '') ?></textarea>

<!-- Button zum Speichern der Änderungen -->
<button type="submit" class="btn btn-submit">Änderungen speichern</button>
</form>
<?php else: // Wenn der Film nicht gefunden wurde ?>
<p>Film nicht gefunden.</p>
<?php endif; ?>
<!-- Link zurück zur Filmliste -->
<a href="/movie-reviews/public/index.php" class="btn btn-back">Zurück zur Filmliste</a>
<?php
$content = ob_get_clean(); // Puffer leeren und Inhalt in der Variable $content speichern
?>
<?php include __DIR__ . '/../layout.php'; // Hauptlayout einbinden, das $content verwendet ?>
```





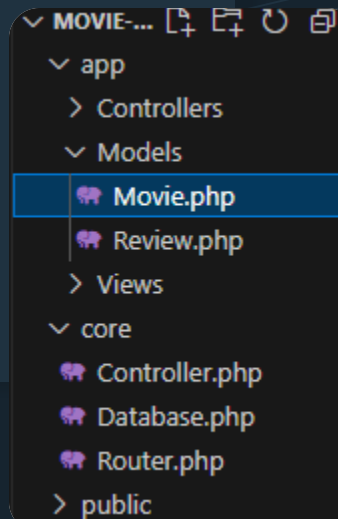


# 05 Film löschen

`<p> Movie.php,MovieController.php,movies/show.php </p>`

# <h1> Movie.php(delete) </h1>

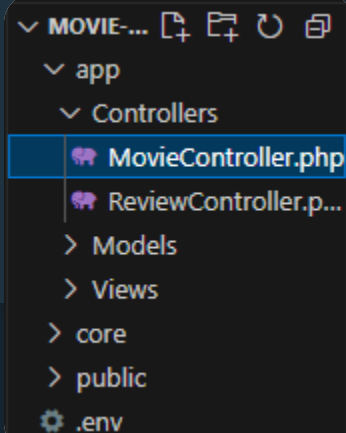
```
// Methode zum Löschen eines Films
public function delete() {
    $db = Database::connect(); // Verbindung zur Datenbank
    herstellen
    $stmt = $db->prepare('DELETE FROM movies WHERE id = :id');
    // SQL-Befehl zum Löschen eines Films
    $stmt->bindParam(':id', $this->id, PDO::PARAM_INT); // ID
    binden
    return $stmt->execute(); // SQL-Befehl ausführen
}
```



# <h1> MovieController.php(delete) </h1>

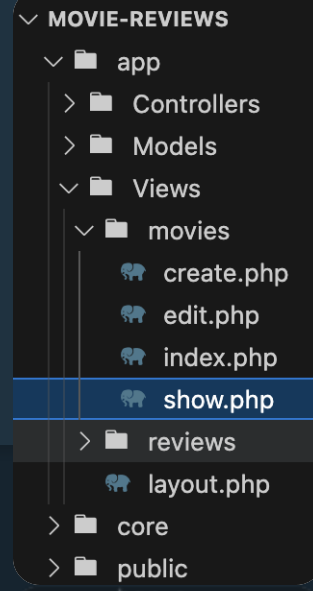
```
// Methode zum Löschen eines Films
public function delete($id) {
    $movie = Movie::find($id); // Film anhand der ID suchen
    if (!$movie) {
        throw new Exception("Movie not found"); // Fehler werfen, wenn Film nicht gefunden wird
    }

    // Film aus der Datenbank löschen
    $movie->delete();
    header('Location: /movie-reviews/public/index.php'); // Zurück zur Hauptseite leiten
    exit;
}
```



# <h1> movies/show.php </h1>

```
<!-- Löschen-Button mit Bestätigungsdialog -->
<a href="/movie-reviews/public/index.php?route=/movies/delete/<?= $movie-
>id ?>"
class="btn btn-danger"
onclick="return confirm('Sind Sie sicher, dass Sie diesen Film löschen
möchten?');">Löschen</a>
</div>
</div>
```





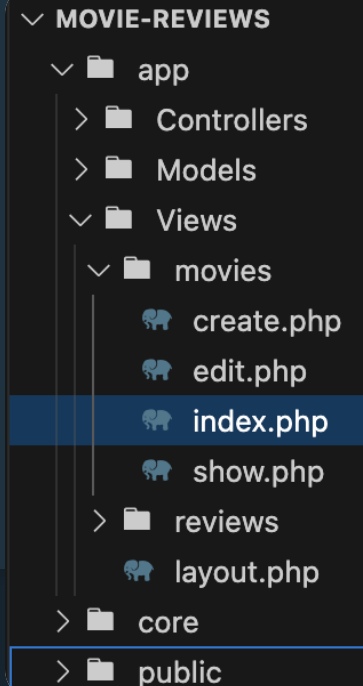
# 05 Review erstellen, anzeigen, bearbeiten

`<p> Review.php, ReviewController.php, review/*.php </p>`

# <h1> IMPLEMENTIERUNG </h1>

Einfügen von Review Komponente in die Movie Index.php

```
<!-- Rezensionen anzeigen -->
<?php include __DIR__ . '/../Views/reviews/show.php'; ?>
<!-- Formular zum Erstellen einer neuen Rezension -->
<?php include __DIR__ . '/../Views/reviews/create.php'; ?>
<!-- Link zurück zur Filmliste -->
```



# <h1> RESPONSIVE DESIGN </h1>

## Grid System

```
.movie-grid {  
  display: grid;  
  grid-template-columns: repeat  
    (auto-fill, minmax(320px, 3fr));  
  gap: 24px;  
}
```

## Breakpoints

```
@media (max-width: 1250px){  
  
  .mobile-wrapper {  
    max-width: 700px;  
  }  
  
}  
  
@media (max-width: 800px){  
  
  .mobile-wrapper {  
    max-width: 400px;  
  }  
  
}
```

### ▼ MOVIE-REVIEWS

> 📁 app

> 📁 core

▼ 📁 public

▼ 📁 css

📄 styles.css

🐘 index.php



# 04 ABSCHLUSS

<p> Mögliche Erweiterungen, Fazit, Fragerunde </p>



# MÖGLICHE ERWEITERUNGEN.



Benutzer-  
registrierung  
und Login-  
Funktionalität.



Erweiterung des  
Bewertungssystems

JS

Echtzeit-  
Interaktivität  
durch JavaScript

# Fazit


Das Projekt hat uns gezeigt, wie man eine klare Architektur erstellt und die verschiedenen Schichten eines Webprojekts miteinander verbindet.

Eine besondere Herausforderung war es, die Zusammenarbeit zwischen dem Backend, das die Daten verarbeitet und bereitstellt, und dem Frontend, das diese Daten für den Benutzer sichtbar macht, reibungslos zu gestalten.



# DANKE EUCH!

Habt Ihr noch Fragen?



Wie findet ihr unsere  
Applikation?

# CREDITS.

Presentation Template: [SlidesMania](#)

Sample Images: [Unsplash](#)

w3schools: [PHP Tutorial](#), [HTML Tutorial](#), [CSS Tutorial](#)

YouTube: [PHP Full Course for non-haters](#)

Fonts used in this presentation: Roboto Mono and **Roboto Bold**