Sujet de projet — Développement Web :
Base de données de films

Enseignant : Behuet Corentin
Module : Développement Web

Contexte et objectif

L'objectif du projet est de concevoir et développer un site web complet (back-end + front-end) s'appuyant sur une base de données de films. L'application devra au minimum permettre :

- de stocker des informations sur des films (titre, année, genre, durée, réalisateur, etc.) ;
- de consulter la liste des films (avec recherche / tri simples) ;
- d'afficher la fiche détaillée d'un film ;
- d'ajouter, modifier et supprimer des films (fonctionnalités de type CRUD).

Le projet doit aboutir à :

- un dépôt GitHub contenant tout le code source du projet ;
- une documentation minimale sous la forme d'un `README.md` expliquant comment installer, configurer et lancer le projet ;
- un rapport écrit (4-8 pages, structure libre) présentant les choix techniques, l'architecture générale, les principales fonctionnalités, diagramme UML, diagramme relationnel et les limites éventuelles ;
- une présentation orale (démo) du projet de max 5mn et min 3mn.

Le but est de montrer votre capacité à :

- concevoir une application web simple mais complète (de la BD jusqu'à l'interface utilisateur) ;
- implémenter proprement les fonctionnalités de base ;
- documenter votre travail de façon suffisamment claire pour qu'une autre personne puisse installer et comprendre votre projet.

Vous pouvez réaliser ce projet **seul** ou en **binôme**.

1 Contraintes techniques (au choix)

Vous devez choisir l'une des approches suivantes pour la partie serveur :

- **Symfony** : utilisation du framework Symfony (version récente), avec une structure de projet standard (contrôleurs, entités, templates, routes, etc.) ;
- **PHP « pur »** : développement sans framework lourd, en organisant vous-même votre architecture (fichiers PHP, organisation en modèles/vues/contrôleurs si souhaité, etc.).

Pour la base de données, **MariaDB** est privilégié, mais vous pouvez utiliser un autre SGBD (MySQL, PostgreSQL, etc.) si vous le souhaitez.

Pour l'accès à la base de données, vous avez le choix entre :

- un **ORM** (p.ex. Doctrine si vous êtes en Symfony) ;

— **PDO** (requêtes SQL préparées, gestion des erreurs, etc.).

Le projet doit respecter les contraintes suivantes :

— être hébergé sur un dépôt **GitHub public** (un dépôt par projet / binôme) ;

— être **déployable localement** facilement (p.ex. via `php -S`, Docker (celui utiliser en TP ou autre), ou un serveur local), avec des instructions claires dans le `README` ;

— contenir au minimum :

  — un script SQL ou une procédure documentée pour créer la base de données et les tables ;

  — un petit jeu de données d'exemple (quelques films déjà présents) ;

  — une page d'accueil, une page "liste des films" et une page "fiche d'un film" ;

  — des formulaires pour ajouter / modifier / supprimer des films avec une validation minimale.

2 Fonctionnalités obligatoires

1. **Base de données de films** : table(s) contenant au minimum titre, année, durée, synopsis, genres, prix de location par défaut, et éventuelles images/affiches.
2. **Affichage catalogue** : page listant les films avec pagination / filtres (genre, année) et fiche détaillée par film.
3. **Tarification dynamique** : possibilité de définir des prix de location différents selon le jour de la semaine (p.ex. tarif réduit le mardi). Le calcul du prix doit être visible sur la fiche ou le panier. Il faudra des effets dynamiques sur la page.
4. **Authentification / comptes** : inscription, connexion, récupération de mot de passe (faire simple).
5. **Profil utilisateur** : on "retient" la personne connectée (p.ex. sessions, cookie etc.) ; page profil avec historique des locations.
6. **Favoris** : un utilisateur peut marquer des films en favoris.
7. **Enregistrement de films** : un utilisateur (pas besoin de faire une gestion fine des droits) peut ajouter/modifier/supprimer un film via une interface (ou API).
8. **Workflow de location** : ajout au panier, confirmation de location, enregistrement de la transaction (de manière fictive et simplifié). Cela peut inclure une partie dynamique (JS).

3 Fonctionnalités optionnelles (bonus)

— **Recherche avancée :** Système de recherche plus évolué (full-text, filtres combinés, suggestions automatiques, corrections orthographiques).

— **Modélisation et documentation technique :** Fournir un schéma UML (diagramme de classes, cas d'utilisation, p.ex. `plantuml`), un schéma relationnel décrivant la base de données et une documentation bien complété (`phpdoc`, `doctum`).

— **Tests automatisés :** Mise en place de tests unitaires, fonctionnels et potentiellement de tests d'intégration.

— **Patrons de conception :** Utilisation appropriée de design patterns pertinents (Repository, Service Layer, MVC clairement structuré, Factory, Adapter si nécessaire).

— **Algorithme de recommandation :** Intégration d'un module de recommandation afin de proposer des films personnalisés à l'utilisateur. Vous êtes encouragés à utiliser une méthode probabiliste ou basée sur le principe exploration/exploitation (par exemple *Upper Confidence Bound (UCB)*, ϵ-*greedy*, un modèle de bandits manchots, filtrage collaboratif simple, etc.).

— **Dynamisme et interactivité du site :** Mise en place d'une interface moderne et réactive comprenant, par exemple :
  — filtrage et recherche instantanés (AJAX, requêtes asynchrones, fetch API) ;
  — animations légères et transitions (carrousels, effets d'apparition, micro-interactions) ;
— **Export / Import de données :** Export des listes de favoris, de l'historique ou des données administratives (CSV, JSON, XML).

4 Contraintes de rendu (livrables)

1. Dépôt GitHub et instructions d'installation **claires**(README).
2. Rapport écrit (PDF) : 4 à 8 pages — architecture, choix techniques, diagramme des tables, captures d'écran, tests effectués, difficultés rencontrées etc.
3. Dossier "Fiche IA" (voir section dédiée) rempli si l'IA a été utilisée pour produire du code ou des textes significatifs.
4. Préparation d'un oral de 5 minutes.

Bonus et pénalités

Pénalités	
Projet inutilisable (trop d'erreurs, pages manquantes, crash systématique)	-7
Pas de dépôt GitHub accessible	-2
Pas de README / pas d'instructions d'installation	-2
Pas de base de données (stockage uniquement en mémoire)	-4
Code non structuré, absence totale de séparation logique (p.ex. un unique fichier PHP illisible)	-2
Bonus	
Documentation technique (diagrammes, explications, architecture, phpdoc)	+2
Présence de tests (unitaires / fonctionnels)	+1
Utilisation pertinente de design patterns (Repository, Service, MVC, etc.)	+2
Utilisation d'un chargement dynamique (fetch sur fichier JSON ou API interne)	+1
Fiche IA remplie sérieusement (voir section dédiée)	+1
Interface utilisateur propre et ergonomique (UX soignée)	+1 à +2
Utilisation d'un algorithme de recommandation avancé (UCB, ϵ-greedy, filtrage collaboratif, etc.)	+2
Filtre dynamique du catalogue en JS	+1 à +2
Petites touches supplémentaires (animation, micro-interactions, UI moderne)	+1

Remarque : la note finale est la somme du total brut plus/minus les pénalités, bonus et des fonctionnalités implémentés. La notation peut cependant être amener à changer.

5 Consignes sur l'utilisation de l'IA

L'usage d'outils d'IA (ChatGPT, Copilot, etc.) est autorisé. **Toute utilisation significative doit être documentée** via la fiche IA jointe. Le but est la transparence : savoir où l'IA a aidé, vérifier la compréhension et évaluer la capacité à corriger/adapter le code généré.