Application Web « Blogs Films »

Cahier des charges

1. Contexte et objectif

Créer une application web communautaire dédiée aux critiques de films. Les utilisateurs peuvent :

- ajouter un film à leur collection personnelle,
- rédiger et publier une critique,
- commenter les publications d'autres utilisateurs,
- liker ou disliker des critiques pour mettre en avant les plus populaires.

L'application doit aussi permettre à un administrateur de modérer et gérer le contenu du site (films, critiques, commentaires, utilisateurs).

Technologies : **Frontend React**, **Backend Spring Boot**, **Base de données relationnelle (PostgreSQL)**.

2. Périmètre fonctionnel

2.1 Acteurs

- **Visiteur** : consulte les critiques, films, tendances.
- **Utilisateur authentifié** : ajoute des films, écrit des critiques, commente, réagit (like/dislike).
- **Administrateur** : modère les contenus, gère les utilisateurs et la base de films.

2.2 Fonctionnalités principales

2.2.1 Parcours visiteur

- Accueil avec les critiques populaires et récentes.
- Recherche d'un film par titre, genre ou année.
- Consultation d'une fiche film : affiche les critiques associées, moyenne des notes et commentaires.

2.2.2 Parcours utilisateur

- Inscription / connexion (email, pseudo, mot de passe).
- Profil utilisateur : avatar, bio, liste de films ajoutés, critiques publiées.
- Ajout d'un film à sa collection personnelle (titre, réalisateur, année, synopsis, affiche).
- Rédaction d'une critique :
    - note (1–10), texte, image ou affiche optionnelle,
    - publication visible par tous.
- Interaction avec d'autres publications :
    - commenter une critique,
    - liker / disliker une critique.
- Fil d'actualité personnalisé (critique de ses abonnés ou recommandations selon préférences, optionnel).

2.2.3 Parcours administrateur

- Interface d'administration sécurisée.
- Gestion CRUD :
    - Films (base de données principale),
    - Critiques (modération, suppression de contenu inapproprié),
    - Commentaires,
    - Utilisateurs (blocage/suppression, changement de rôle).
- Statistiques : nombre de publications, likes, activité utilisateur.

3. Exigences techniques

3.1 Architecture

- **Frontend** : React (Vite ou CRA), gestion d'état avec Redux Toolkit ou Context API.
- **Backend** : Spring Boot (REST API), Spring Data JPA, Spring Security.
- **Base de données** : PostgreSQL, migrations Flyway/Liquibase.
- **Authentification** : JWT (stateless) + hachage des mots de passe (bcrypt).
- **Stockage** : affiches et avatars sur disque ou cloud (S3 compatible).
- **Déploiement** : Docker, CI/CD via GitHub Actions.

3.2 API (exemples)

Public

- `GET /api/films` — liste des films (filtres : genre, année, popularité)
- `GET /api/films/{id}` — fiche film avec critiques associées
- `GET /api/reviews` — liste des critiques récentes ou populaires

Utilisateur (ROLE_USER)

- `POST /api/auth/register` — inscription
- `POST /api/auth/login` — connexion
- `POST /api/films` — ajouter un film à sa collection
- `POST /api/reviews` — créer une critique
- `POST /api/reviews/{id}/comments` — commenter
- `POST /api/reviews/{id}/like` — liker ou disliker
- `GET /api/users/me` — profil et collection personnelle

Admin (ROLE_ADMIN)

- `GET /api/admin/dashboard` — statistiques
- `DELETE /api/admin/reviews/{id}` — suppression d'une critique
- `DELETE /api/admin/comments/{id}` — suppression d'un commentaire
- `PUT /api/admin/users/{id}` — blocage ou changement de rôle

3.3 Schéma de données (proposition)

Tables principales :

- `users` (id, email, password_hash, pseudo, avatar_url, role, created_at)
- `films` (id, title, director, year, genre, synopsis, poster_url, created_by, created_at)
- `reviews` (id, film_id, user_id, title, content, rating, likes, dislikes, created_at)
- `comments` (id, review_id, user_id, content, created_at)
- `likes` (id, review_id, user_id, value [1/-1])

Relations :

- Un utilisateur a plusieurs critiques.
- Un film a plusieurs critiques.
- Une critique a plusieurs commentaires.
- Un utilisateur peut liker plusieurs critiques.

4. Non-fonctionnel

- **Sécurité** : validation d'entrée, contrôle d'accès, XSS/CSRF protection.
- **Performance** : pagination (reviews, comments), cache pour pages populaires.
- **Scalabilité** : architecture stateless, API REST claire, stockage externe des images.
- **Logs et monitoring** : journalisation via SLF4J + Logback.
- **Accessibilité et UX** : interface responsive, accessible mobile et tablette.

5. UI / UX (haut niveau)

- **Accueil** : critiques récentes et populaires, barre de recherche.
- **Page film** : fiche + critiques associées.
- **Page critique** : texte complet, likes/dislikes, zone de commentaires.
- **Profil utilisateur** : collection, critiques publiées, avatar.
- **Admin dashboard** : tables de gestion (films, critiques, commentaires, utilisateurs).

6. Sécurité et conformité

- RGPD : suppression du compte et des données personnelles sur demande.
- Protection des contenus (signalement possible).
- Audit trail sur actions admin.

7. Tests et recette

- **Unitaires** : JUnit, Mockito, Jest.
- **Intégration** : API (Spring Boot Test), interactions critiques (likes, commentaires).
- **E2E** : Cypress / Playwright (création critique, commentaire, like/dislike).
- **Critères d'acceptation** :
    - Un utilisateur peut créer une critique et la partager.
    - D'autres peuvent commenter et liker/disliker.
    - L'admin peut modérer les contenus et utilisateurs.

8. Déploiement

- Conteneurisation Docker.
- CI/CD : build + tests + déploiement staging/prod.
- DB hébergée (PostgreSQL) avec sauvegardes.
- Hébergement sur Render, AWS, ou serveur dédié.

9. Roadmap (suggestion)

1. **MVP (2–3 sprints)** : Auth, ajout de films, critiques, commentaires.
2. **V2** : Système de like/dislike, profils enrichis, pagination.
3. **V3** : Fil d’actualité, notifications, recherche avancée, modération fine.

10. Cas d’usage

- **UC1** : Utilisateur ajoute un film et rédige une critique.
- **UC2** : Un autre utilisateur commente cette critique.
- **UC3** : Des utilisateurs likent ou dislikent la critique.
- **UC4** : L’admin supprime une critique signalée.

11. Critères de livraison

- Code source complet (React + Spring Boot).
- Documentation API (Swagger).
- Scripts de migration DB et seed initial (films de test).
- Tests >70% sur les fonctionnalités critiques.
- Déploiement fonctionnel sur environnement de test.

Notes finales

Ce cahier des charges constitue la base du projet « Blogs Films ». Il peut être enrichi avec :

- des user stories détaillées,
- des maquettes UI,
- un modèle de données (ERD) et spécifications API Swagger pour développement.