R5-8 - Qualité de développement

CAHIER DES CHARGES – Projet de Gestion de Bibliothèque

Projet : Système de Gestion de Bibliothèque Universitaire

Groupe : Naharro Guerby, Bonnard Nathan, Le Bastard Théo

Date de rendu : 13/02/2026

SOMMAIRE

I. Introduction

Ce projet s'inscrit dans le cadre de la modernisation des infrastructures numériques de l'université. L'objectif est de réaliser une **application web complète (Full Stack)** permettant la gestion automatisée d'une bibliothèque universitaire.

Le projet servira de support pour évaluer :

- La maîtrise de l'architecture client-serveur (React/Django).
- La qualité du code et la gestion de base de données.
- La capacité à déployer une solution conteneurisée (Docker).

Équipe de développement :

- Naharro Guerby
- Bonnard Nathan
- Le Bastard Théo

II. Contexte du projet

A. Description du contexte

Actuellement, la bibliothèque utilise des registres papier et des fichiers Excel déconnectés. Cela entraîne des erreurs de stock, des pertes d'ouvrages et une difficulté à suivre les retards de restitution.

B. Analyse des besoins

Le système doit permettre de passer d'une gestion manuelle à une gestion informatisée centralisée. Les besoins principaux sont :

- Centraliser les données des livres et des utilisateurs.
- Automatiser le suivi des dates de retour.
- Sécuriser les accès selon le rôle de l'utilisateur (Étudiant , Enseignant, Bibliothécaire).

III. Objectifs du projet

A. Objectifs généraux

- Développer une application web robuste et maintenable.
- Respecter les standards de l'industrie (API REST, SPA, Docker).

B. Objectifs spécifiques

- **Centralisation :** Base de données unique et cohérente.
- **Ergonomie :** Interface utilisateur (UI) moderne et réactive (React/MUI).
- **Portabilité :** Fonctionnement identique sur Windows, Linux et macOS via Docker.
- **Automatisation :** Détection automatique des retards et calcul des pénalités visuelles.

IV. Fonctionnalités requises

A. Liste exhaustive des fonctionnalités

1. Module Catalogue (Front-Office)

- Affichage des livres sous forme de grille adaptative (Responsive Design).
- Recherche multicritères (Titre, Auteur, Catégorie).
- Fiche détaillée d'un ouvrage (ISBN, Éditeur, Résumé, Stock en temps réel).

2. Module Emprunts (Back-Office & User)

- Enregistrement d'un emprunt (Liaison User <-> Livre).
- Calcul automatique de la date de retour (14 jours par défaut).
- Validation du retour (Réincrémentation automatique du stock).
- Tableau "Mes Emprunts" pour l'étudiant avec indicateurs de statut.

3. Module Administration (Back-Office Bibliothécaire)

- **Dashboard :** Statistiques (Top 5 lecteurs, Livres populaires, Taux de retard, ...).
- **CRUD Livres :** Formulaire complet d'ajout/modification/suppression.

- **Gestion Utilisateurs :** Liste des inscrits, suppression de comptes, attribution de rôles.

4. Système d'Alertes

- Pop-up à l'accueil signalant les emprunts proches de l'échéance (J-3).
- Marquage visuel (rouge) des emprunts en retard dans les listes.

B. Priorisation des fonctionnalités

1. **Essentielles (MVP) :** Authentification, CRUD Livres, Emprunt simple, Retour simple.
2. **Secondaires :** Dashboard statistique, Recherche avancée, Gestion des images de couverture.
3. **Confort :** Alertes visuelles (Pop-up), Tri dynamique des colonnes.

C. Interactions entre les fonctionnalités

- Un emprunt ne peut être créé que si le stock du livre est > 0.
- Le retour d'un livre libère une place dans le stock immédiatement.
- La suppression d'un utilisateur est bloquée s'il a des emprunts "En cours".

V. Règles de gestion (Business Logic)

- Rôles (RBAC) :
    - *Étudiant :* Consultation, Emprunt standard.
    - *Enseignant :* Accès aux ouvrages "réservés", durée d'emprunt étendue et choix dans nombre de livre emprunter (exemple : 30 livres pour c'est élèves).
    - *Bibliothécaire :* Accès administrateur total (Dashboard, CRUD).
- **Limites :** Un utilisateur ne peut pas emprunter le même exemplaire deux fois simultanément.
- **Retards :** Un emprunt est considéré "En retard" si Date du jour > Date de retour prévue.

VI. Spécifications techniques

A. Stack Technologique

- **Backend :** Python 3.11, Django 5, Django REST Framework (DRF).
- **Frontend :** React 18, Vite, TypeScript, Material UI (MUI v6).
- **Base de données :** PostgreSQL 16.
- **Infrastructure :** Docker, Docker Compose.

B. Structure du projet

- **backend/** : Contient l'API Django, les modèles (ORM) et la logique métier.
- **frontend/** : Contient l'application React, les composants et les pages.
- **docker-compose.yml** : Orchestration des conteneurs (Web, API, DB).

C. Modèle de Données

- **Users :** Extension du modèle Django Auth (ajout du champ role).
- **Books :** Titre, Auteur, ISBN, Stock, ImageURL, Relation One-to-Many vers Category.
- **Loans :** Relation Many-to-One vers User et Book, DateEmprunt, DateRetour, Statut.

VII. Contraintes et limitations

A. Contraintes de temps

Le projet doit être livré fonctionnel. Le déploiement doit être instantané via Docker.

B. Contraintes techniques

- **Compatibilité OS :** Le projet doit se lancer sur Windows/Linux sans erreur de script (gestion des fins de ligne LF/CRLF via docker-compose command).
- **Code Quality :** Typage TypeScript respecté, pas de "hard-coding" des URLs API.

VIII. Tests et validation

A. Stratégie de test

- **Tests Backend :** Vérification des Endpoints API (GET /books, POST /loans).
- **Tests Frontend :** Vérification du rendu des composants (Grille, Tableaux) et de la navigation.
- **Tests d'intégration :** Scénario complet "Inscription -> Emprunt -> Retour".

B. Critères de réussite

1. L'application démarre avec une seule commande (docker-compose up).
2. Un utilisateur peut s'authentifier et voir ses emprunts.
3. L'administrateur peut ajouter un livre et voir les statistiques à jour.
4. Les retards sont correctement signalés visuellement.

IX. Livrables attendus

1. **Code source :** Dépôt Git complet et code source .ZIP.
2. Documentation :
    - README.md (Installation, Architecture, Guide Windows).
    - Ce Cahier des Charges.
3. **Plan de test :** Document listant les scénarios testés.
4. **Environnement :** Fichiers Dockerfile et docker-compose.yml fonctionnels.

X. Glossaire

- **CRUD :** Create, Read, Update, Delete (Opérations de base sur les données).
- **SPA :** Single Page Application (Application web fluide sans rechargement de page).
- **API REST :** Interface de programmation permettant au Frontend de dialoguer avec le Backend.
- **Docker :** Plateforme permettant d'exécuter l'application dans des conteneurs isolés.
- **RBAC :** Role-Based Access Control (Gestion des droits basée sur les rôles).