Library Management Application

Cahier des Charges

1. Introduction

1.1. Contexte du Projet

Dans le contexte d'un stage PFE à Proxym, le projet **« Library Management Application »** vise à développer une plateforme web complète qui permettra aux librairies de moderniser leurs processus internes (gestion des stocks, commandes, personnel, etc.) tout en offrant aux lecteurs un espace interactif et collaboratif.

1.2. Objectif Général

Concevoir et réaliser une application web intégrée qui propose :

- La gestion des librairies : Optimisation des processus internes (catalogue, commandes, gestion d'événements) grâce à l'automatisation via des workflows BPMN.
- Un espace communautaire pour les lecteurs (forums, avis, recommandations).
- Une gestion robuste de l'authentification et de l'autorisation via **Keycloak**.
- La possibilité de développer le backend avec **Spring Boot** (Java) ou **Node.js/Express** (JavaScript), en fonction des compétences de l'équipe et des besoins techniques.
- Un cycle de développement organisé et itératif en adoptant la méthodologie **Agile SCRUM**.

---

2. Objectifs et Périmètre du Projet

2.1. Objectifs Spécifiques

- **Pour les Librairies (Administrateurs) :**
    - Gérer le catalogue et le stock des ouvrages.
    - Suivre les commandes et la gestion des ventes.
    - Orchestrer les processus métiers à l'aide de workflows BPMN (via Flowable).

    - Administrer les comptes utilisateurs et attribuer des rôles via Keycloak.
- **Pour les Lecteurs (Utilisateurs Finaux) :**
    - Rechercher et consulter le catalogue avec des filtres avancés.
    - Participer à une communauté en ligne (forums, avis, partage de recommandations).
    - Gérer leur compte personnel dans un environnement sécurisé grâce à Keycloak.
- **Sécurité et Authentification :**
    - Assurer la sécurisation des accès et des échanges via **Keycloak**.
- **Backend Flexible :**
    - Offrir deux options de développement backend :
        - **Spring Boot** (Java)
        - **Node.js/Express** (JavaScript)
- **Gestion de Projet :**
    - Organisation selon la méthodologie **Agile SCRUM** pour favoriser la flexibilité, la communication et l'amélioration continue.

2.2. Périmètre Fonctionnel

Le projet couvrira :

- Le développement d'un frontend en **ReactJS**.
- La conception d'un backend accessible via des API REST sécurisées, avec possibilité de choix entre Spring Boot et Node.js/Express.
- L'intégration de **Flowable** pour la modélisation et l'exécution des processus métiers (BPMN).
- La conception et la gestion d'une base de données **PostgreSQL**.
- La gestion des accès et des identités avec **Keycloak**.

Les éléments non-couverts pourront être définis ultérieurement par exemple, intégration d'un système de paiement pour la finalisation des commandes, l'intégration d'agent IA (intelligence artificielle) pour les communautés de lecture.

---

3. Description Fonctionnelle

3.1. Gestion pour les Librairies (Interface Administrateur)

- **Catalogue et Stock :**
    - Ajout, modification et suppression d'ouvrages.
    - Suivi des exemplaires, gestion des réservations et disponibilité.
- **Commandes et Ventes :**
    - Suivi des commandes en ligne et en boutique.

    - Gestion des retours et échanges.
- **Workflow et Processus Métiers :**
    - Définition et exécution de workflows (validation de commandes, réapprovisionnement, etc.) via **Flowable**.
    - Interface d'administration pour visualiser et contrôler les processus métiers en cours.
- **Gestion des Utilisateurs et Rôles :**
    - Administration des comptes (administrateur, employés, membres, etc…).
    - Attribution et gestion des rôles et permissions via **Keycloak**.

3.2. Espace Communautaire pour les Lecteurs

- **Catalogue et Recherche :**
    - Moteur de recherche avancé avec filtres (auteur, catégorie, disponibilité).
    - Consultation détaillée des ouvrages (descriptions, avis, notes).
- **Interactions et Réseaux Sociaux :**
    - Forums de discussion, espaces de commentaires et système de notation.
    - Partage de recommandations et création de listes de lecture.
- **Gestion de Compte Personnel :**
    - Inscription, authentification et gestion sécurisée des comptes utilisateurs via **Keycloak**.
    - Gestion des préférences et historique des interactions.
- **Notifications et Événements :**
    - Envoi de notifications sur les nouveautés et les événements.
    - Inscription à des ateliers ou séances dédicaces.

---

4. Description Technique

4.1. Architecture Globale

- **Frontend :**
  Développement en **ReactJS** pour une interface dynamique, responsive et modulaire.
    - Utilisation de Redux ou Context API pour la gestion de l'état.
    - React Router pour la navigation.
- **Backend :**
  Deux options sont envisageables :
        - Création d'API REST sécurisées.

        - Intégration de **Flowable** pour les workflows métiers.
        - Intégration avec **Keycloak** pour l'authentification et autorisations.
- **Sécurité et Authentification :**
    - **Keycloak :**
        - Gestion centralisée des identités et des accès (authentification, autorisation, gestion des rôles).
        - Intégration avec le backend via des tokens JWT et la configuration de clients sécurisés.
- **Workflow Engine :**
    - Utilisation de **Flowable** pour la modélisation (BPMN) et l'exécution des processus métiers.
- **Base de Données :**
    - Utilisation de **PostgreSQL** pour la gestion et la persistance des données (catalogue, utilisateurs, commandes, etc.).

- **Versionning et CI/CD :**
    - Utilisation de Git pour le contrôle de version.
    - Mise en place d'un pipeline CI/CD (GitLab CI/CD) pour automatiser tests et déploiements.
- **Containerisation :**
    - Utilisation de Docker pour simplifier le déploiement et assurer la scalabilité.
- **Monitoring :**
    - Outils de monitoring (Prometheus, Grafana) et de logging ELK (Elasticsearch, Logstash, Kibana).

---

5. Contraintes et Normes

5.1. Contraintes Techniques

- **Sécurité :**
    - Gestion centralisée des accès via **Keycloak** pour garantir l'intégrité et la confidentialité.
- **Performance et Scalabilité :**
    - Optimisation du frontend (lazy loading, code splitting) et du backend (mise en cache, indexation en base de données).
    - Architecture modulable pour permettre une évolution future si nécessaire.
- **Interoperabilité :**

- Prévoir des interfaces facilitant l'intégration de services tiers (paiement, etc.).

5.2. Normes et Bonnes Pratiques

- Respect des standards de développement web (Java, REST, HTML5, CSS3, ECMAScript).
- Mise en œuvre de tests unitaires, d'intégration et fonctionnels tout au long du cycle de développement.

---

6. Organisation, Planning et Gestion de Projet

6.1. Méthodologie de Gestion de Projet

- **Méthodologie Agile SCRUM :**
  - Le projet sera découpé en sprints itératifs (typiquement de 2 à 4 semaines) permettant une livraison incrémentale et l'adaptation continue aux retours des parties prenantes.
  - Des réunions, des revues de sprint et des rétrospectives seront organisées pour assurer une communication efficace et l'amélioration continue du processus.
  - Clickup sera utilisé pour la gestion des user stories, tâches et backlog.

6.2. Phases du Projet

1. **Analyse et Conception (1-2 semaines) :**
   - Recueil des besoins et rédaction des spécifications fonctionnelles et techniques.
   - Conception de l'architecture globale incluant le choix du backend (Spring Boot ou Node.js/Express) et la modélisation des workflows BPMN.
   - Mise en place de l'environnement de développement (configuration de Keycloak, création du dépôt Git, etc.).
2. **Développement (3 à 4 mois) :**
   - Développement du frontend en ReactJS.
   - Réalisation du backend selon la technologie choisie (Spring Boot ou Node.js/Express) et intégration avec Keycloak.
   - Conception de la base de données PostgreSQL.
   - Intégration de Flowable pour la gestion des processus métiers.

6.3. Livrables

- **Architecture:**

    - Diagrammes UML : Représentation détaillée de l'architecture du système et du design des modules.
    - Documentation des API : Spécifications complètes (endpoints, paramètres, exemples d'appels) à l'aide d'outils tels que Swagger/OpenAPI.
    - Schémas de la Base de Données : Modèles relationnels détaillés (PostgreSQL) avec description des relations et des contraintes.

- **Application Fonctionnelle :**

    - Frontend développé en ReactJS offrant une interface utilisateur intuitive et responsive.
    - Backend (réalisé en Spring Boot ou Node.js/Express) fournissant des API REST sécurisées et intégrant la gestion des workflows via Flowable.
    - Intégration complète de la gestion des identités et des accès via Keycloak.
    - Base de données PostgreSQL correctement configurée et intégrée.

---

7. Critères de Validation

- **Conformité Fonctionnelle :**
    - Respect des fonctionnalités définies pour les interfaces administrateur et utilisateur.
    - Bon fonctionnement des workflows métiers et de l'interface communautaire.
- **Qualité Technique :**
    - Performance et réactivité de l'application.
    - Sécurisation des échanges et des données grâce à l'intégration de Keycloak (tokens JWT, gestion des rôles).
    - Couverture de tests suffisante.
- **Ergonomie et Accessibilité :**
    - Interface intuitive, responsive et conforme aux normes d'accessibilité.
- **Documentation et Maintenance :**
    - Documentation claire et complète.
    - Facilité d'évolution et de maintenance du code source.
