**Documentation du Projet Logiciel : Gestion de Bibliothèque**

**Description du Projet**

L'application de gestion de bibliothèque est conçue pour faciliter la gestion des livres, des membres et des emprunts dans une bibliothèque. Elle permet aux bibliothécaires de suivre les livres disponibles, les livres empruntés et les membres inscrits. Les fonctionnalités principales incluent l'ajout et la suppression de livres, l'inscription de nouveaux membres, la gestion des emprunts et des retours, et la génération de rapports.

**Fonctionnalités Principales**

1. **Gestion des Livres**
   * Ajout de nouveaux livres dans la base de données.
   * Modification des informations des livres existants.
   * Suppression de livres.
   * Recherche de livres par titre, auteur, ISBN.
2. **Gestion des Membres**
   * Inscription de nouveaux membres.
   * Modification des informations des membres.
   * Suppression de membres.
   * Recherche de membres par nom, numéro de membre.
3. **Gestion des Emprunts**
   * Enregistrement des emprunts de livres par les membres.
   * Suivi des dates de retour prévues.
   * Enregistrement des retours de livres.
   * Gestion des pénalités pour retards.
4. **Rapports et Statistiques**
   * Génération de rapports sur les livres empruntés.
   * Statistiques sur les emprunts par période.
   * Rapports sur les membres actifs.

**Technologies Utilisées**

* **Backend** : Java, Spring Boot
* **Frontend** : HTML, CSS, JavaScript
* **Base de Données** : MySQL
* **Tests** : JUnit, Mockito, Selenium, Postman, Cucumber

**Cycle de Vie du Développement**

1. **Conception**
   * Analyse des besoins.
   * Conception des diagrammes UML (cas d'utilisation, classes, séquences).
   * Définition des spécifications fonctionnelles et techniques.
2. **Développement**
   * Implémentation des fonctionnalités backend et frontend.
   * Création de l'API REST pour la communication entre le frontend et le backend.
   * Intégration avec la base de données MySQL.
3. **Tests**
   * Écriture des tests unitaires pour le code backend (JUnit, Mockito).
   * Réalisation des tests d'intégration pour vérifier les interactions entre les composants.
   * Tests de système pour valider le fonctionnement global de l'application (Selenium).
   * Tests d'acceptation pour valider les exigences des utilisateurs finaux (Cucumber).
4. **Déploiement**
   * Déploiement de l'application sur un serveur de production.
   * Configuration des environnements de staging et de production.
   * Planification et réalisation des déploiements continus (CI/CD).
5. **Maintenance**
   * Surveillance des performances et de la disponibilité de l'application.
   * Correction des bugs et des problèmes signalés par les utilisateurs.
   * Mise à jour et ajout de nouvelles fonctionnalités.

**Exemples de Phases de Tests**

1. **Tests Unitaires (Développement)**
   * Exemple : Tester la méthode d'ajout de livre pour s'assurer qu'elle ajoute correctement un livre dans la base de données.

@Test

public void testAddBook() {

Book book = new Book("123456789", "Effective Java", "Joshua Bloch");

when(bookRepository.save(book)).thenReturn(book);

Book addedBook = bookService.addBook(book);

assertEquals("123456789", addedBook.getIsbn());

assertEquals("Effective Java", addedBook.getTitle());

assertEquals("Joshua Bloch", addedBook.getAuthor());

}

1. **Tests d'Intégration (Tests)**
   * Exemple : Tester l'intégration entre le service de gestion des emprunts et la base de données pour vérifier que les emprunts sont correctement enregistrés.

@SpringBootTest

public class BorrowServiceIntegrationTest {

@Autowired

private BorrowService borrowService;

@Autowired

private BookRepository bookRepository;

@Autowired

private MemberRepository memberRepository;

@Test

public void testBorrowBook() {

Member member = memberRepository.save(new Member("John Doe"));

Book book = bookRepository.save(new Book("123456789", "Effective Java", "Joshua Bloch"));

borrowService.borrowBook(member.getId(), book.getId());

// Assertions to verify the borrow record is correctly saved

}

}

1. **Tests de Système (Tests)**
   * Exemple : Utiliser Selenium pour automatiser un scénario de test complet où un utilisateur emprunte un livre et le retourne.

@Test

public void testBorrowAndReturnBook() {

WebDriver driver = new ChromeDriver();

driver.get("http://localhost:8080");

// Steps to log in

driver.findElement(By.id("username")).sendKeys("admin");

driver.findElement(By.id("password")).sendKeys("password");

driver.findElement(By.id("login")).click();

// Steps to borrow a book

driver.findElement(By.id("borrowBook")).click();

driver.findElement(By.id("memberId")).sendKeys("1");

driver.findElement(By.id("bookId")).sendKeys("123456789");

driver.findElement(By.id("submit")).click();

// Assertions to verify the book is borrowed

// Steps to return the book

driver.findElement(By.id("returnBook")).click();

driver.findElement(By.id("memberId")).sendKeys("1");

driver.findElement(By.id("bookId")).sendKeys("123456789");

driver.findElement(By.id("submit")).click();

// Assertions to verify the book is returned

driver.quit();

}

1. **Tests d'Acceptation (Déploiement)**
   * Exemple : Utiliser Cucumber pour définir des scénarios de test d'acceptation en langage naturel.

Feature: Borrow and return books

Scenario: Borrow a book

Given I am logged in as a librarian

When I borrow a book with ISBN "123456789" for member "John Doe"

Then the book should be marked as borrowed

Scenario: Return a book

Given I am logged in as a librarian

When I return a book with ISBN "123456789" for member "John Doe"

Then the book should be marked as available