



Rapport de projet CY-Books

27 MAI

BATTAIS Nathan
DREYER Romain
LOYAU Alexis
TREPOS Pauline



Sommaire

1. Introduction

2. Description du projet

3. L'organisation

- 3.1 L'équipe
- 3.2 Les réunions

4. Présentation ULM

- 4.1 Use case
- 4.2 Diagramme de classes

5. Description des problèmes

6. L'application

7. Conclusion

1. Introduction

Notre projet consiste à développer une application de gestion de bibliothèque en Java, visant à simplifier et optimiser les tâches des bibliothécaires. Cette application, dotée d'une interface graphique conviviale et accessible via la souris et le clavier, permettra d'inscrire et de gérer les usagers, de superviser le stock de livres et de suivre les emprunts. Pour enrichir les informations sur les ouvrages, l'application intégrera l'API de la Bibliothèque Nationale de France (BNF), tandis que les données des utilisateurs, des identifiants des livres et des emprunts seront stockées localement dans une base de données.

2. Description du projet

Fonctionnalités principales

Gestion des Usagers : Enregistrer de nouveaux usagers, mettre à jour leurs informations, et effectuer des recherches avancées sur les utilisateurs.

Gestion des Livres : Rechercher des livres selon différents critères en interrogeant l'API BNF, gérer le stock de livres disponibles, et suivre leur état d'emprunt.

Gestion des Emprunts : Enregistrer et gérer les emprunts, en respectant les durées maximales autorisées et les limites de nombre de livres par usager, avec des alertes en cas de retards.

Aspects Techniques

Interface Utilisateur : Concevoir une interface graphique intuitive et facile à utiliser.

Base de Données : Utiliser une base de données locale pour le stockage des informations essentielles, avec un mécanisme pour démarrer le serveur de base de données automatiquement.

Intégration API : Connecter l'application à l'API BNF pour récupérer des informations détaillées sur les livres.

Robustesse : Assurer la stabilité de l'application avec une gestion appropriée des erreurs pour éviter les crashes et les boucles infinies.

Documentation et Versioning : Utiliser Git pour le contrôle de version du code, générer de la documentation avec JavaDoc, et stocker les fichiers sur un dépôt central accessible.

Organisation du Projet

Travail d'Équipe : Former des groupes de quatre personnes, organiser des réunions régulières avec le chargé de projet pour assurer le suivi.

Livrables : Développer d'abord une version en ligne de commande fonctionnelle avant de compléter l'interface graphique, et déposer l'ensemble du projet sur GitHub ou GitLab.

Démonstration et Évaluation : Préparer une présentation de la version finale, avec une machine dédiée aux modifications en temps réel.

3. L'organisation

3.1 L'équipe

L'équipe composé de quatre personnes s'est organisé de manière à ce que les trois grandes parties qui compose l'application soit au moins traité par deux personnes à chaque fois.

La partie qui gère le lien avec l'API de la BNF à été traité par Pauline, Romain et Alexis
Celle de la gestion de la base de données locale par Pauline, Alexis et Nathan
Enfin la partie interface utilisateur quant-a-elle a été développé par Nathan et Romain

3.2 Les réunions

Des réunions pour s'assurer du bon avancement du projet ont régulièrement été effectué, on en compte 5 plus importante qui pour 3 trois d'entre elles ont eu lieu avec le professeur encadrant.

Le 07/05, une première réunion a eu lieu avec le professeur pour procéder à une vérification des attentes de celui-ci ainsi qu'à des premiers conseils pour commencer le projet dans la bonne direction.

Le 10/05, une réunion interne a permis d'échanger sur l'ULM du projet ainsi que la répartition des différentes parties du projet.

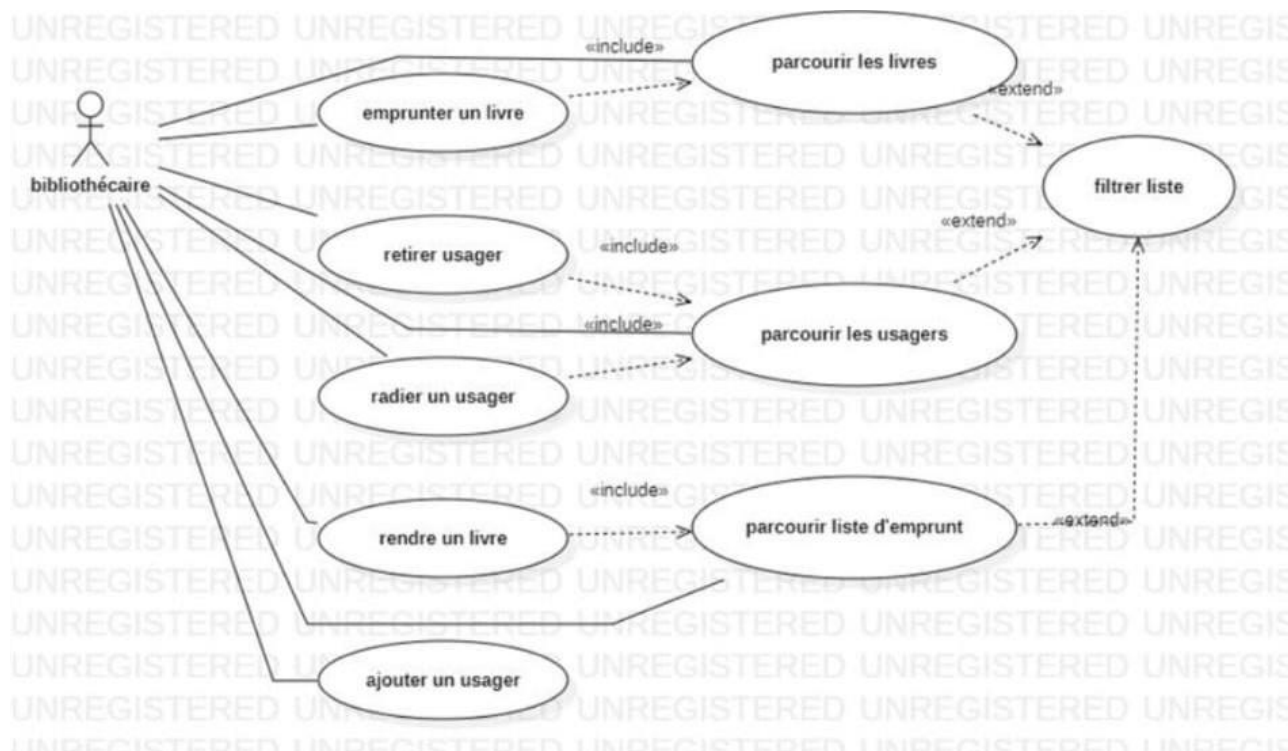
Le 18/05, une réunion de contrôle avec le professeur s'est tenue pour s'assurer que nous allions toujours dans le bon sens.

Le 23/05, a eu lieu la dernière réunion avec l'encadrant qui nous a permis de résoudre avec lui des problèmes que nous avions avec la gestion du lien avec l'API, la gestion de la base de données et de l'interface graphique.

Enfin, la dernière réunion avec une importance majeure s'est tenue le 26/05 avant de déposer notre projet sur github afin de s'assurer que tout était bon avant la dead line.

4. Présentation UML

4.1 Use case



Le diagramme use case ci-dessus montre les interactions via l'application pour le bibliothécaire :

Acteur

Bibliothécaire : Représenté par une figure humaine, c'est l'utilisateur du système qui exécute toutes les actions décrites.

Cas d'utilisation principaux

Emprunter un livre : Le bibliothécaire effectue l'emprunt d'un livre pour un utilisateur.

Rendre un livre : Le bibliothécaire enregistre le retour d'un livre emprunté.

Ajouter un usager : Le bibliothécaire ajoute un nouvel utilisateur au système.

Retirer usager : Le bibliothécaire supprime un utilisateur du système.

Radier un usager : Le bibliothécaire radie un utilisateur, probablement pour une raison administrative ou disciplinaire.

Cas d'utilisation supplémentaires (inclus)

Parcourir les livres : **Ce cas est inclus dans "Emprunter un livre", indiquant que pour emprunter un livre, le bibliothécaire doit parcourir la liste des livres disponibles.**

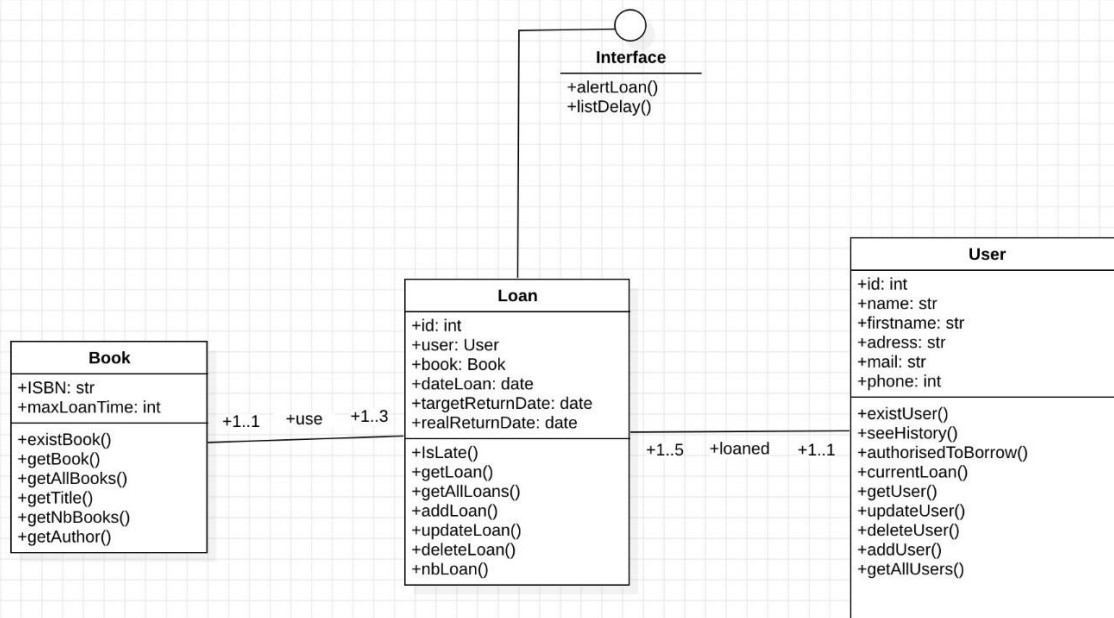
Parcourir les usagers : **Ce cas est inclus dans "Retirer usager" et "Radier un usager", indiquant que pour supprimer ou radier un utilisateur, le bibliothécaire doit d'abord parcourir la liste des usagers.**

Parcourir liste d'emprunt : **Ce cas est inclus dans "Rendre un livre", indiquant que pour enregistrer le retour d'un livre, le bibliothécaire doit parcourir la liste des emprunts.**

Extensions

Filtrer liste : **Ce cas d'utilisation peut être une extension de "Parcourir les livres" et "Parcourir liste d'emprunt". Cela signifie que lors de la recherche de livres ou d'emprunts, le bibliothécaire peut utiliser des filtres pour affiner les résultats.**

4.2 Use case



Description du diagramme de classe du système de prêt de livres

Classes

Le diagramme de classe comporte les classes principales suivantes :

Membre : Cette classe représente les membres de la bibliothèque qui peuvent emprunter des livres. Elle possède les attributs suivants :

id : identifiant unique du membre

nom : nom du membre

prénom : prénom du membre

adresse : adresse du membre

email : adresse e-mail du membre

téléphone : numéro de téléphone du membre

Livre : Cette classe représente les livres qui peuvent être empruntés par les membres. Elle possède les attributs suivants :

ISBN : numéro ISBN du livre

titre : titre du livre

auteur : auteur du livre

genre : genre du livre (par exemple, roman, science-fiction, etc.)

disponibilité : statut de disponibilité du livre (par exemple, disponible, emprunté, en réservation)

Emprunt : Cette classe représente les prêts de livres aux membres. Elle possède les attributs suivants :

id : identifiant unique du prêt

membre : objet Membre représentant le membre emprunteur

livre : objet Livre représentant le livre emprunté

dateEmprunt : date d'emprunt du livre

dateRetourPrévue : date à laquelle le livre doit être retourné

dateRetourRéal : date à laquelle le livre a été retourné (si elle est renseignée)

Bibliothécaire : Cette classe représente les bibliothécaires qui gèrent le système de prêt. Elle possède les attributs suivants :

id : identifiant unique du bibliothécaire

nom : nom du bibliothécaire

prénom : prénom du bibliothécaire

email : adresse e-mail du bibliothécaire

Relations entre les classes

Les classes du diagramme sont reliées par les relations suivantes :

Un membre peut emprunter plusieurs livres. Cette relation est représentée par l'association 1..n entre les classes Membre et Emprunt. Cela signifie qu'un membre peut avoir plusieurs prêts en cours à la fois.

Un livre peut être emprunté par un seul membre à la fois. Cette relation est représentée par l'association 0..1 entre les classes Livre et Emprunt. Cela signifie qu'un livre ne peut être emprunté que par un seul membre à la fois.

Un emprunt est associé à un membre et à un livre. Cette relation est représentée par les associations 1..1 entre les classes Emprunt, Membre et Livre. Cela signifie qu'un prêt ne peut être associé qu'à un seul membre et à un seul livre.

Un bibliothécaire peut gérer plusieurs emprunts. Cette relation est représentée par l'association *..n entre les classes Bibliothécaire et Emprunt. Cela signifie qu'un bibliothécaire peut gérer plusieurs emprunts en cours.

Fonctionnalités du système

Le système de prêt de livres présenté dans le diagramme de classe permet de gérer les prêts de livres aux membres. Il offre les fonctionnalités suivantes :

Enregistrer un nouveau membre : Un bibliothécaire peut enregistrer un nouveau membre en renseignant ses informations personnelles (nom, prénom, adresse, etc.).

Ajouter un nouveau livre : Un bibliothécaire peut ajouter un nouveau livre en renseignant ses informations bibliographiques (ISBN, titre, auteur, genre, etc.) et son statut de disponibilité.

Emprunter un livre : Un membre peut emprunter un livre en renseignant son identifiant et l'ISBN du livre. Le système vérifie la disponibilité du livre et enregistre l'emprunt.

Consulter ses prêts en cours : Un membre peut consulter la liste de ses prêts en cours.

Retourner un livre : Un membre peut retourner un livre en renseignant son identifiant et l'ISBN du livre. Le système enregistre la date de retour du livre.

Gérer les retards de retour : Le système peut envoyer des notifications aux membres lorsque leurs prêts sont en retard.

5. Description des problèmes

Problème 1 : Modules non trouvés au lancement de JavaFX

Description : Au début du projet, des erreurs indiquaient que certains modules comme `eu.hansolo.fx.countries`, `eu.hansolo.fx.heatmap`, `eu.hansolo.toolboxfx`, et `eu.hansolo.toolbox` n'étaient pas trouvés. Cela empêchait l'application de se lancer correctement.

Solution : Les modules manquants ont été résolus en modifiant la configuration du projet. Les modules ont été passés de Runtime à Compile dans les paramètres du projet (File > Project structure > Modules).

Résultat : Les erreurs de modules manquants ont été corrigées, permettant à l'application de se lancer sans problèmes liés aux modules.

Problème 2 : Erreur 'Module not found: javafx.swing'

Description : Après avoir corrigé les problèmes initiaux de modules, une nouvelle erreur est apparue, indiquant que le module `javafx.swing` n'était pas trouvé.

Solution : Le projet a été recréé sans sélectionner de modules supplémentaires. Les anciens fichiers ont été réimportés dans le nouveau projet, éliminant ainsi les erreurs liées aux modules.

Résultat : L'application a pu se lancer correctement sans l'erreur `javafx.swing`, et les modules nécessaires ont été correctement chargés.

Problème 3 : Gestion des chemins des fichiers FXML

Description : L'application ne pouvait pas charger les fichiers FXML correctement en raison de chemins incorrects spécifiés lors de la création initiale du projet.

Solution : Les chemins des fichiers FXML ont été simplifiés en supprimant les préfixes inutiles. Par exemple, au lieu de spécifier `view/leFile.fxml`, le chemin a été changé en `file.fxml`.

Résultat : Les fichiers FXML ont été chargés correctement, permettant d'afficher les interfaces utilisateur prévues.

Problème 4 : Robustesse du code pour la suppression des livres

Description : Le code initial n'était pas assez robuste, car il ne permettait de supprimer un livre que si le titre et l'auteur étaient écrits avec les mêmes majuscules et minuscules.

Solution : Pour rendre le code plus robuste, la méthode `toUpperCase()` a été utilisée lors de la lecture des entrées utilisateur, ce qui permet de comparer les titres et les auteurs de manière insensible à la casse.

Résultat : Le code est devenu plus tolérant aux variations de casse dans les entrées utilisateur, améliorant ainsi la robustesse du système de suppression de livres.

Problème 5 : Problèmes de localisation des fichiers

Description : Plusieurs erreurs liées à la localisation des fichiers ont été rencontrées, nécessitant des modifications dans `module-info.java` et d'autres configurations de chemin.

Solution : Les chemins des ressources dans `MainApp` ont été corrigés en utilisant correctement `getResource`, ce qui a permis de charger correctement les différentes pages de l'application.

Résultat : Les problèmes de localisation des fichiers ont été résolus, permettant un chargement correct des différentes pages de l'application.

Problème 6 : Initialisation de l'application principale

Description : L'application avait des difficultés à initialiser et à gérer l'objet principal MainApp, ce qui empêchait l'affichage des données et le bon fonctionnement des méthodes associées.

Solution : Une constante publique statique Lemain a été ajoutée pour que l'instance principale de l'application soit accessible à toutes les classes.

Résultat : Les fonctions ont recommencé à fonctionner correctement, mais des problèmes persistent avec l'affichage des données et la liaison des utilisateurs. Des ajustements supplémentaires sont nécessaires pour assurer que toutes les fonctionnalités soient correctement intégrées et que les données s'affichent comme prévu.

Problème 7 : Difficulté de l'utilisation de SQL avec les JAR SQLite-JDBC et SLF4J

Description : L'intégration de la base de données SQL s'est avérée complexe en raison de la nécessité de gérer les JAR sqlite-jdbc et slf4j. La charge de configuration entre un projet Java standard et un projet Maven a amplifié cette difficulté, surtout avec les nombreuses options disponibles (MySQL, SQL Server, etc.). Trouver les bons packages était crucial pour faire fonctionner les requêtes.

Solution : Nous avons décidé de structurer le projet sous Maven, ce qui a simplifié l'inclusion des dépendances. Les fichiers nécessaires ont été ajoutés au pom.xml, facilitant ainsi la gestion des bibliothèques.

Résultat : L'intégration de la base de données s'est faite plus facilement. Le projet Maven a permis une gestion plus fluide des dépendances, et l'utilisation de SQLite, un langage simple pour les bases de données, a facilité l'écriture et l'exécution des requêtes SQL.

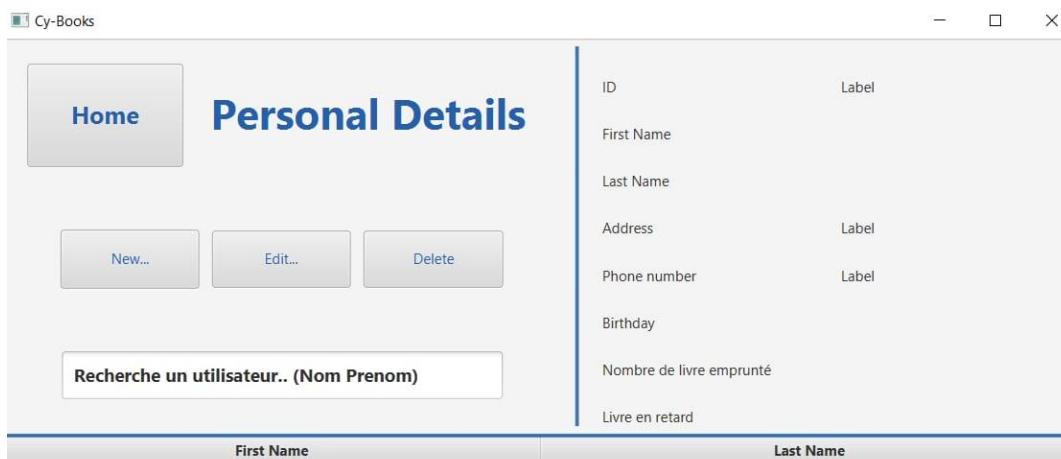
En résumé, les solutions apportées ont permis de résoudre les différents problèmes rencontrés lors du développement de l'application de gestion de bibliothèque, assurant une application plus stable et fonctionnelle. Des ajustements supplémentaires sont néanmoins nécessaires pour garantir une intégration complète et une utilisation sans faille de toutes les fonctionnalités.

6. L'application

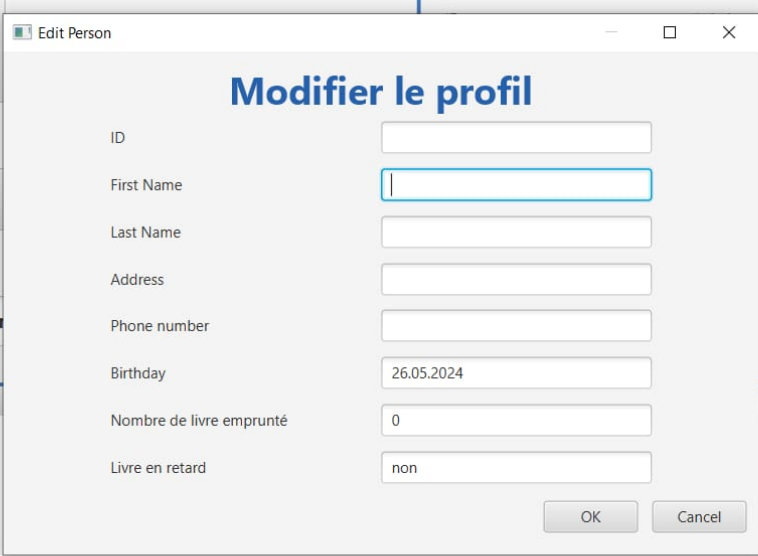
Page d'accueil lors de l'ouverture d'application :



Page de gestion d'un utilisateur :

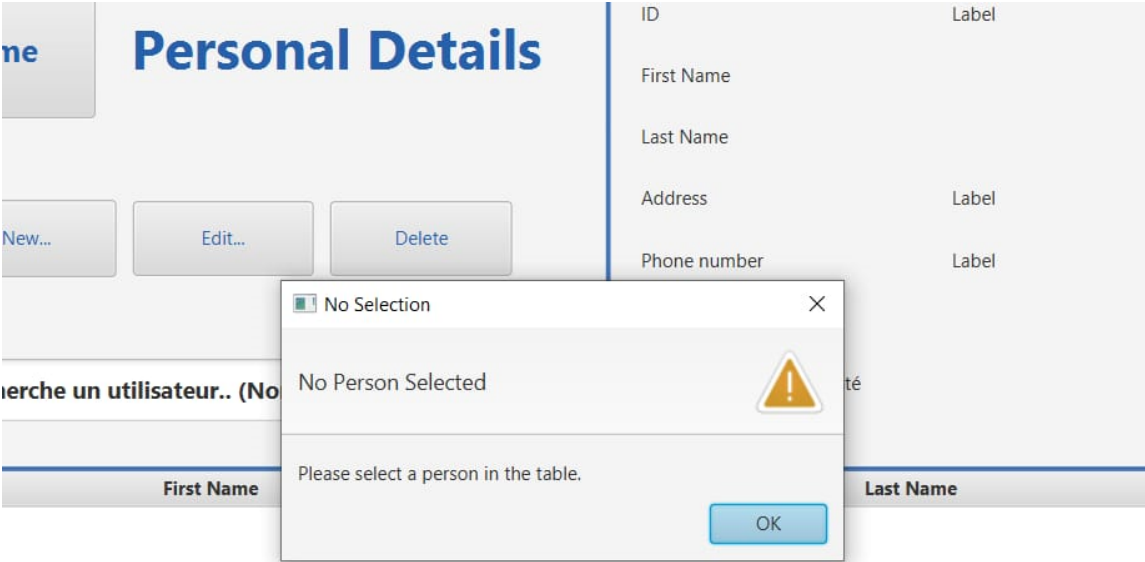


Page de création ou de modification d'un profil utilisateur :



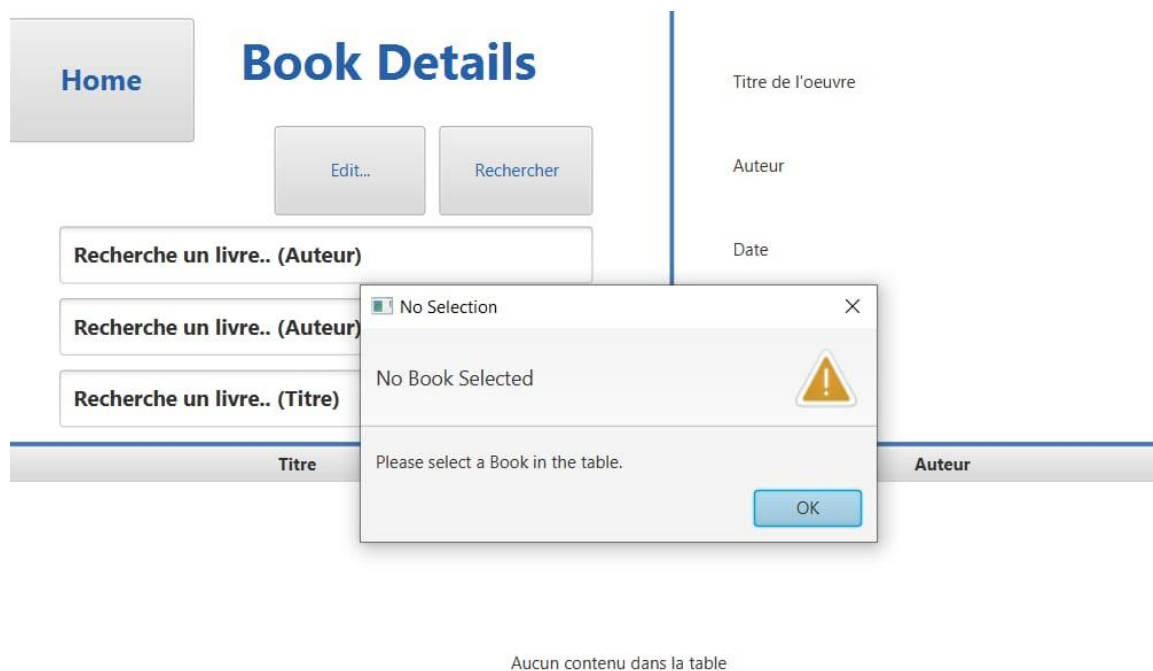
The screenshot shows a modal dialog box titled "Edit Person" with the subtitle "Modifier le profil". It contains several input fields for user information: ID, First Name, Last Name, Address, Phone number, Birthday (pre-filled with 26.05.2024), Nombre de livre emprunté (pre-filled with 0), and Livre en retard (pre-filled with non). At the bottom right are "OK" and "Cancel" buttons. Below the dialog, the text "Aucun contenu dans la table" is visible.

Page d'erreur, la modification ou la suppression d'un utilisateur est impossible s'il n'est pas sélectionné :



The screenshot shows a web page titled "Personal Details" with buttons for "New...", "Edit...", and "Delete". An error dialog box titled "No Selection" is overlaid, displaying a warning icon and the message "No Person Selected. Please select a person in the table." with an "OK" button. Below the dialog, the text "Aucun contenu dans la table" is visible.

Page d'erreur, la modification d'un livre est impossible s'il n'est pas sélectionné :



7. Conclusion

Ce projet de création d'une application de gestion de bibliothèque a été une expérience enrichissante, malgré les défis rencontrés. Nous avons réussi à surmonter des obstacles.

Cependant, quelques problèmes persistent, notamment la présence de deux main dans le projet et la difficulté à établir un lien cohérent entre le backend et le frontend. Ce projet nous a permis de mettre en pratique nos compétences en développement logiciel et en gestion de projet.