Projet Flutter

Master 1 IMIS 2022/2023

Groupe E : Fatoumata Barry, Alexandre Cailloux et Matthieu Bonbon

Application de Gestion de livres : mini-bibliothèque

Documentation technique

Enseignant : Fréderic Loulergue

Résumé du document

Ce document retrace la documentation technique de notre projet et comprends les parties suivantes:

- Scénarios d'utilisation
- Description de l'architecture
- Description courte de la mise en œuvre du design pattern
- Localisation du Widget élaboré
- Description de la campagne de tests de l'application

<< Voir l'architecture du projet dans le Readme.md >>

Scénarios d'utilisation

Notre application se présente comme suit:

Une fois l'application lancée, l'utilisateur arrive devant une page de connexion qui lui permet soit de créer un compte pour une première utilisation, soit de renseigner ces identifiants de connexion.

S'il crée un compte, il est automatiquement connecté.

Une fois connecté, il arrive sur une page "Bibliothèque" qui présente la liste des livres qui sont présents dans la base de données.

De là, il peut choisir entre :

1. **Voir son profil** : il est alors redirigé sur une page à partir de laquelle il peut voir ses informations et se déconnecter s'il le souhaite.

2. **Voir les livres présents sur sa Wishlist** : il est alors redirigé vers une nouvelle page où il pourra voir ses livres favoris (qu'il aura lui même sélectionné).

3. **Voir les livres qu'il a empruntés** : il sera à nouveau redirigé vers une page où il pourra voir la liste des livres empruntés et éventuellement **rendre** des livres s'il le souhaite.

4. Rester sur la Bibliothèque où il peut **cliquer** sur un livre pour voir ses informations en détails, l'**emprunter** s'il est libre (avec un bouton disponible pour),ou l'**ajouter** sur sa wishlist.

5. Il peut également effectuer une recherche dans la bibliothèque avec l'icône « loupe » présente sur la page de cette dernière.

S'il est **Admin** alors il pourra en plus des fonctionnalités précédentes :

6. **Créer un livre** : En renseignant les différents champs requis.

7. **Créer un auteur** : Il a accès à une liste des auteurs présents dans la base de données et peut en ajouter de nouveaux.

8 . **Créer une maison d'édition** : il a accès à la liste des maisons d'éditions présentes dans la base de données et peut en rajouter de nouvelles.

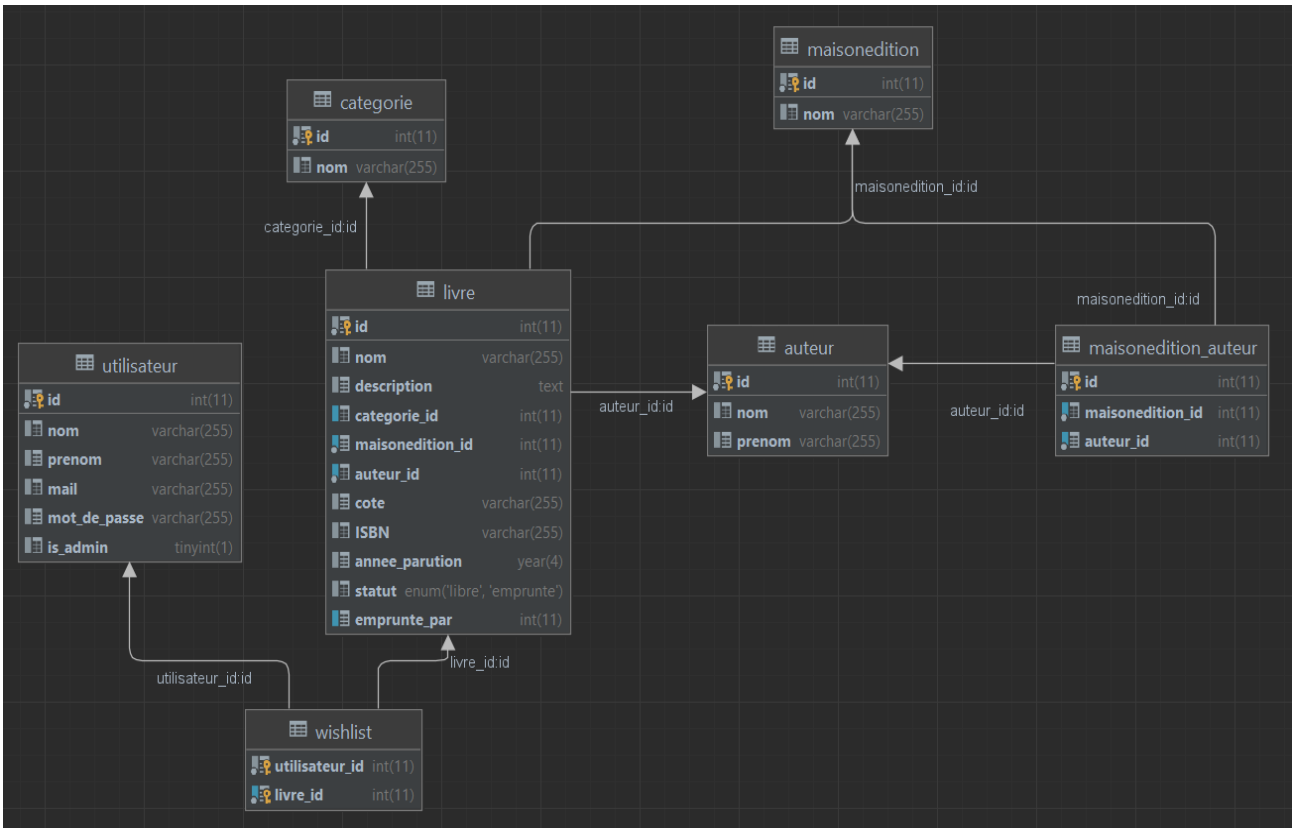
9.**Créer une catégorie** : il peut rajouter une nouvelle catégorie de livre.

Description de l'architecture

Pour mettre en place notre projet, nous avons mis au point une base de données puis implémenté un web service (api) afin de relier la base de données à l'interface de notre application.

✓ Base de données

Il s'agit d'une base de données MySQL mise en place avec WampServer et interfacée en PhpMyAdmin.
Son architecture repose sur la modélisation suivante :



✓ API

Implémentée en Rest SpringBoot avecMaven, elle renferme un dossier controller qui contient les contrôleurs des classes des différentes entités. Chaque contrôleur est associé à un Service lui même associé à un Repository afin de faire également la jonction avec la base de données.

Les endpoints sont les suivants :

Livre : /livres

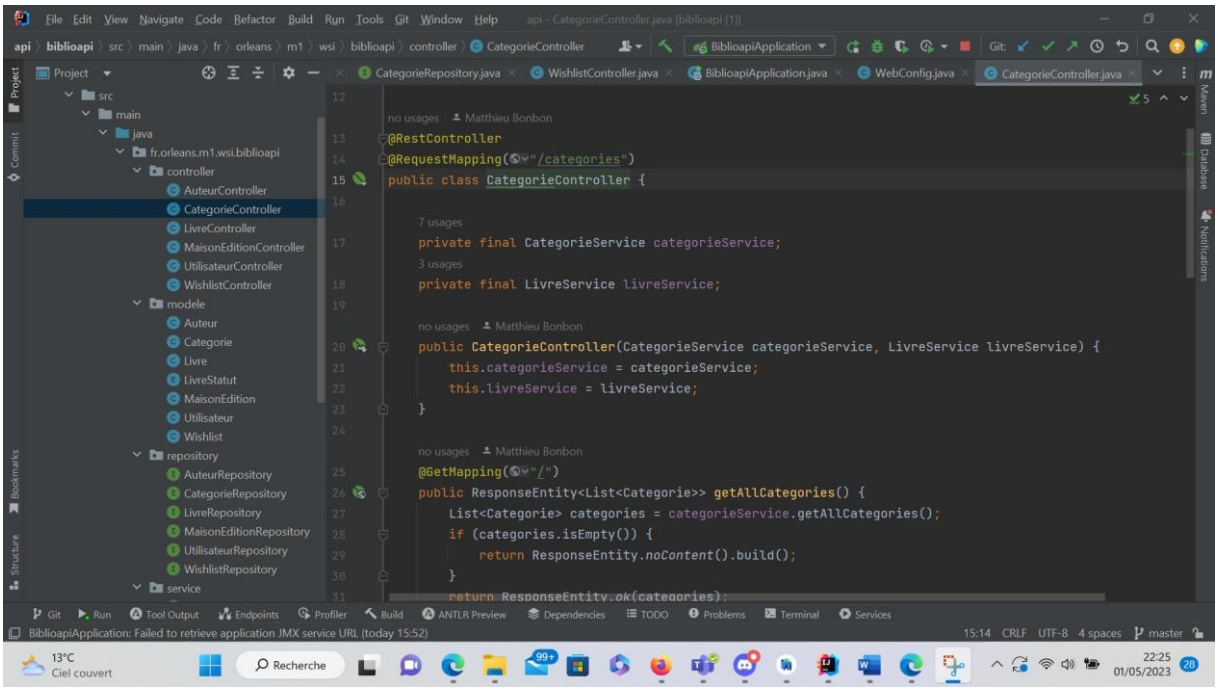
Auteur : /auteurs

Catégorie: /categories

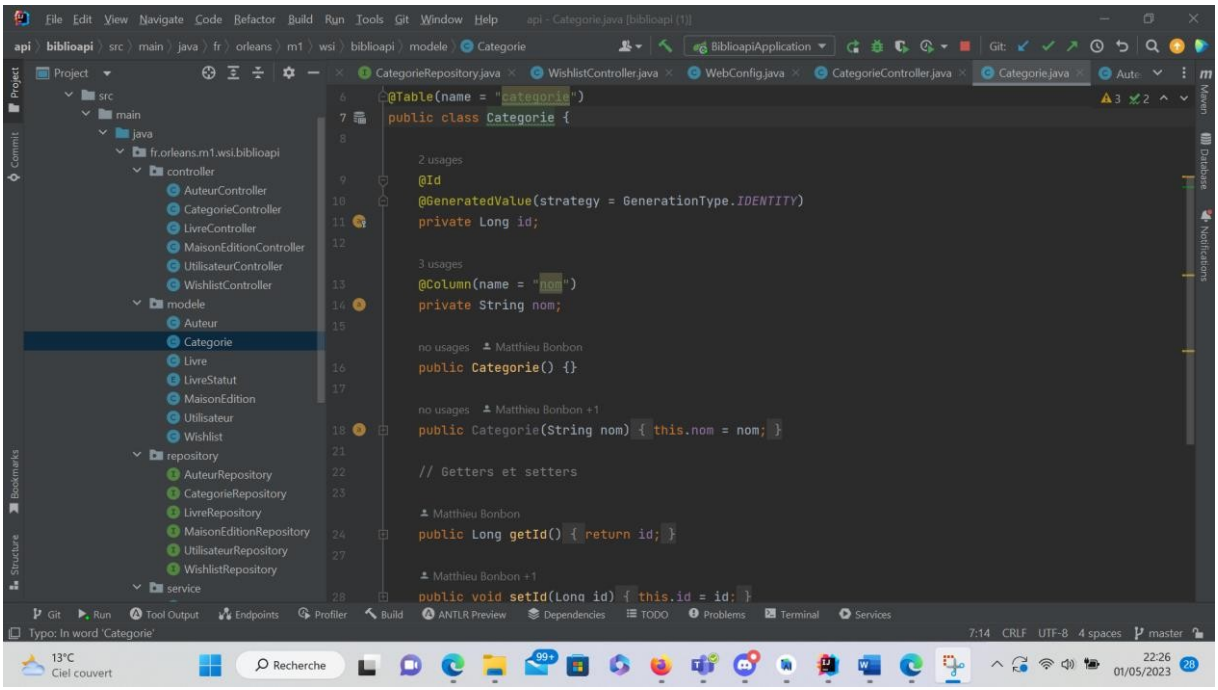
MaisonEdition: /maisoneditions

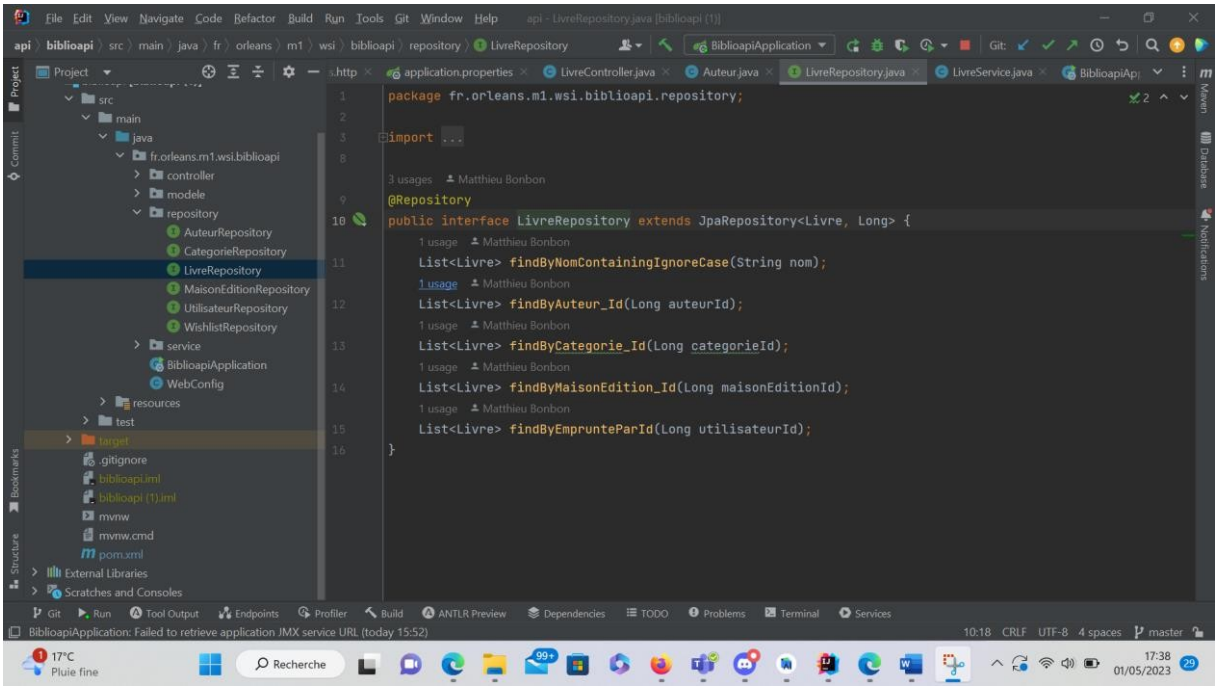
Utilisateurs: /utilisateurs

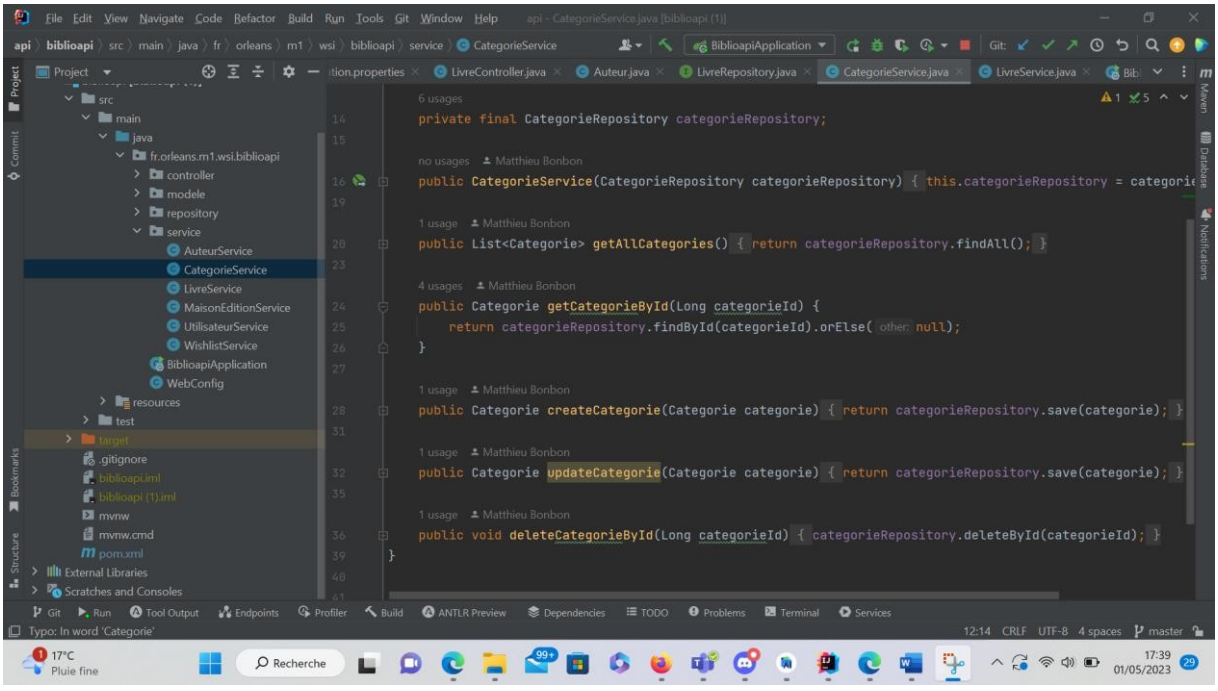
Whislist : /wishlists



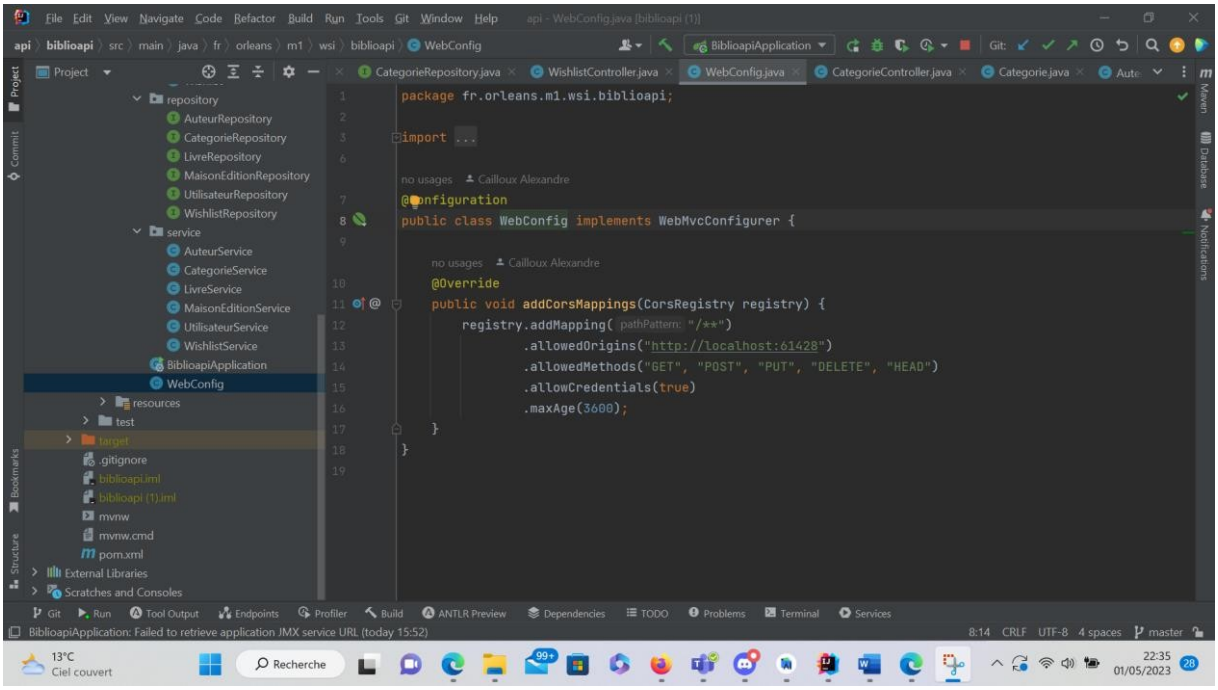
Un dossier modele qui contient les classes des différentes entités.




File Edit View Navigate Code Refactor Build Run Tools Git Window Help
api - LivreRepository.java [biblioapi (1)]
api › biblioapi › src › main › java › fr › orleans › m1 › wsi › biblioapi › repository › LivreRepository
BiblioapiApplication
Git:
Project
application.properties
LivreController.java
Auteur.java
LivreRepository.java
LivreService.java
src
main
java
fr.orleans.m1.wsi.biblioapi
controller
modele
repository
AuteurRepository
CategorieRepository
LivreRepository
MaisonEditionRepository
UtilisateurRepository
WishlistRepository
service
BiblioapiApplication
WebConfig
resources
test
target
.gitignore
biblioapi.iml
biblioapi (1).iml
mvnw
mvnw.cmd
pom.xml
External Libraries
Scratches and Consoles
package fr.orleans.m1.wsi.biblioapi.repository;
import ...
3 usages Matthieu Bonbon
@Repository
public interface LivreRepository extends JpaRepository<Livre, Long> {
1 usage Matthieu Bonbon
List<Livre> findByNomContainingIgnoreCase(String nom);
1 usage Matthieu Bonbon
List<Livre> findByAuteur_Id(Long auteurId);
1 usage Matthieu Bonbon
List<Livre> findByCategorie_Id(Long categorieId);
1 usage Matthieu Bonbon
List<Livre> findByMaisonEdition_Id(Long maisonEditionId);
1 usage Matthieu Bonbon
List<Livre> findByEmprunteParId(Long utilisateurId);
}
Git Run Tool Output Endpoints Profiler Build ANTLR Preview Dependencies TODO Problems Terminal Services
BiblioapiApplication: Failed to retrieve application JMX service URL (today 15:52)
10:18 CRLF UTF-8 4 spaces master
17°C
Pluie fine
Recherche
17:38
01/05/2023


File Edit View Navigate Code Refactor Build Run Tools Git Window Help
api - CategorieService.java [biblioapi (1)]
api › biblioapi › src › main › java › fr › orleans › m1 › wsi › biblioapi › service › CategorieService
BiblioapiApplication
Git:
Project
LivreController.java
Auteur.java
LivreRepository.java
CategorieService.java
LivreService.java
src
main
java
fr.orleans.m1.wsi.biblioapi
controller
modele
repository
service
AuteurService
CategorieService
LivreService
MaisonEditionService
UtilisateurService
WishlistService
BiblioapiApplication
WebConfig
resources
test
target
.gitignore
biblioapi.iml
biblioapi (1).iml
mvnw
mvnw.cmd
pom.xml
External Libraries
Scratches and Consoles
6 usages
private final CategorieRepository categorieRepository;
no usages Matthieu Bonbon
public CategorieService(CategorieRepository categorieRepository) { this.categorieRepository = categorie
1 usage Matthieu Bonbon
public List<Categorie> getAllCategories() { return categorieRepository.findAll(); }
4 usages Matthieu Bonbon
public Categorie getCategorieById(Long categorieId) {
return categorieRepository.findById(categorieId).orElse(other: null);
}
1 usage Matthieu Bonbon
public Categorie createCategorie(Categorie categorie) { return categorieRepository.save(categorie); }
1 usage Matthieu Bonbon
public Categorie updateCategorie(Categorie categorie) { return categorieRepository.save(categorie); }
1 usage Matthieu Bonbon
public void deleteCategorieById(Long categorieId) { categorieRepository.deleteById(categorieId); }
}
Git Run Tool Output Endpoints Profiler Build ANTLR Preview Dependencies TODO Problems Terminal Services
Typo: In word 'Categorie'
12:14 CRLF UTF-8 4 spaces master
17°C
Pluie fine
Recherche
17:39
01/05/2023

La classe WebConfig permet de surpasser la sécurité qui interdit la vision du front et du back sur le même serveur afin d'autoriser sans problèmes les requêtes venant du front.



Description courte de la mise en œuvre du design pattern

On a utilisé le Pattern Singleton. Le modèle singleton est utilisé pour garantir qu'une seule instance d'une classe est créée dans toute l'application. Il peut être utilisé pour créer des objets de données partagés, tels que des listes de livres ou l'URL du back-end, afin d'éviter les problèmes de cohérence des données et de limiter la duplication du code. Ici, nous utilisons donc le pattern Singleton afin que chaque Controller (côté front-end) utilise l'URL permettant de se connecter au back-end, celle-ci étant modifiée en fonction du type d'appareil cible.

```dart
import 'package:flutter/foundation.dart';

class PlatformService {
  static PlatformService? _instance = null;

  factory PlatformService() {
    if (_instance == null) {
      _instance = PlatformService._internal();
    }
    return _instance!;
  }

  PlatformService._internal();

  String testPlatform() {
    if(defaultTargetPlatform == TargetPlatform.android){
      print("Android :)");
      return "10.0.2.2:8080";
    }
    else {
      print("Web :(");
      return "localhost:8080";
    }
  }
}
```

Localisation du Widget élaboré

Le widget élaboré se trouve dans le dossier Pages dans la classe BookDetailsPage à partir de la ligne 277.

Ce widget retourne un Scaffold qui contient une AppBar et un SingleChildScrollView qui contient un ensemble de Texte.

Le Scaffold contient une AppBar avec un bouton pour retourner en arrière et un titre qui affiche le nom du livre en cours de lecture. Le SingleChildScrollView contient un ensemble de Text centré. Ces textes affichent des informations sur le livre, telles que le nom de l'auteur, le nom de la maison d'édition, la catégorie, la description, l'année de sortie, l'ISBN, etc. Le SingleChildScrollView permet d'ajouter une scroll bar si la page est trop petite pour afficher le contenu, évitant les erreurs liés aux différents formats d'écran.
Le SingleChildScrollView contient un FutureBuilder qui permet de récupérer des données asynchrones pour déterminer si le livre est déjà dans la wishlist ou s'il a été emprunté.
En fonction de ces données, un ensemble de boutons est affiché pour permettre à l'utilisateur de rajouter ou de retirer le livre de sa wishlist ou de le signaler comme emprunté.

Description de la campagne de tests de l'application

Dans notre cas, nous avons au début mis des données dans l'application afin de tester les différentes fonctionnalités et de pouvoir améliorer le rendu à l'écran. Une fois que l'api était fonctionnelle, nous avons réalisé une version finale de la base de données, effectué des requêtes au fur et à mesure afin d'identifier les fonctionnalités à implémenter et celles à améliorer.

Nous avons ensuite testé la version finale de l'application afin de savoir si elle fonctionnait comme nous le souhaitions et nous en avons profité pour améliorer le design général de l'application.