RAPPORT DE PROJET

CY-Books

CY Tech
Réalisé par : Quentin FILLION, Alexandre GRISEZ, Pauline MACEIRAS, Théo BELLIERE et Emilien MASSI Juin 2024

Table des matières :

I. Introduction

1. Contexte

Dans le cadre de notre première année de cycle ingénieur informatique au sein de CY-TECH, il nous a été demandé de réaliser un projet informatique. Celui-ci constitue un module à part entière : “Projet Génie Logiciel” et est nécessaire à la validation de l’année en cours. Ce projet est un travail d’équipe.

Notre équipe est constituée de cinq personnes en première année de cycle ingénieur informatique :

- Théo Belliere (ING1 GI4), qui souhaite se spécialiser dans la cybersécurité ou l’IA en dernière année.
- Emilien Massi (ING1 GI4), qui souhaite se spécialiser dans l’IA ou le développement de logiciel en dernière année.
- Pauline Maceiras (ING1 GI4), qui souhaite se spécialiser dans la cybersécurité ou l’IA en dernière année.
- Quentin Fillion (ING1 GI3), qui souhaite se spécialiser en cybersécurité en dernière année.
- Alexandre Grisez (ING1 GI3), qui souhaite se spécialiser dans la cybersécurité ou l'IA en dernière année.

2. Demande du client

“Créer une application permettant de gérer une bibliothèque, avec toutes les fonctionnalités telles que l’inscription des usagers, la gestion des livres, le stock, et les différents emprunts.”

Quelques contraintes importantes :

- L’application doit comporter une interface graphique JavaFx
- Il faut aussi une version ligne de commande pour pouvoir montrer rapidement les fonctionnalités si nécessaire.
- Les données des livres sont récupérées en passant par l’API de la Bibliothèque Nationale de France (BNF).
- Il faut pouvoir parcourir les livres avec des recherches multicritères.
- Certaines fonctionnalités sont requises comme : afficher les livres les plus populaires, afficher les résultats page par page ou afficher l’historique des emprunts d’un utilisateur précis.

Cette liste n’est pas exhaustive et l’énoncé complet peut se retrouver dans le dossier ‘docs’ de notre dépôt GitHub.

II. Parties du projet

1. Données

Pour la gestion des données, nous avons créé des classes Java ainsi que des tables SQL. Pour simplifier les échanges de données, nous avons aussi créé des classes intermédiaires qui centralisent la gestion de tous les objets et font l’interface avec le reste de l’application. Par exemple, la classe DBHandler est utilisée à chaque fois que l’on a besoin d’accéder à la base de données pour lire ou écrire des informations.

La base de données SQL est initialisée au démarrage de l'application si elle n'existe pas encore, avec quelques données déjà préremplies. Le fichier correspondant se trouve dans "resources.database". Les tables sont sauvegardées de manière à pouvoir relancer l'application sans perdre les données. La connexion à la base de données se fait avec le module JDBC. Il permet de se connecter à un SGBD, et ensuite d'effectuer des requêtes en lecture ou en écriture.

La classe Core est le noyau du modèle de données, c'est la classe qui sert d'intermédiaire entre la partie IHM (ligne de commande ou fenêtre) et la partie données (base de données, API, lien entre les deux).

Bibliothécaires :

Les bibliothécaires (désignés dans le projet par 'users' ou 'librarians') sont les utilisateurs de l'application. Ils sont représentés par des objets de la classe Librarian, et leurs informations sont stockées dans la table SQL Librarians. Ils doivent s'authentifier pour accéder à l'application, et ont donc un identifiant de connexion et un mot de passe. Un hash du mot de passe est stocké dans la base de données pour plus de sécurité. Pour chiffrer le mot de passe, nous avons utilisé le module BCrypt. Dans notre projet, il permet de chiffrer un nouveau mot de passe ou de vérifier si une chaîne brute correspondrait au mot de passe chiffré.

Adhérents :

Les adhérents (désignés dans le projet par 'members' ou 'customers') sont les visiteurs ou abonnés de la bibliothèque. Ils sont représentés par des objets de la classe Customer, et leurs informations sont stockées dans la table SQL Customers.

Livres :

Les livres sont les documents que nous récupérons via l'API de la BNF. Ils sont représentés par des objets de la classe Book. Dans la base de données, seul l'identifiant, le stock et la quantité totale est conservée. Pour les objets de la classe Book, il existe des champs qui correspondent aux champs retournés par l'API. Nous avons fait le choix de ne récupérer de la BNF que les documents dont le type est 'texte' car c'est ce qui se rapprochait le plus d'un type livre, qui lui n'existe pas dans l'API.

Nous avons laissé au bibliothécaire le choix d'indiquer combien de livres sont disponibles dans l'inventaire (stock), la valeur par défaut pour les livres ajoutés automatiquement est 5.

Au début du projet, nous nous imaginions que l'API de la BNF allait nous fournir un ISBN pour tous les documents, mais ce n'a pas été le cas. Ces documents n'étant pas tous des livres et n'ayant pas tous un ISBN, la BNF utilise un identifiant unique qui prend la forme d'un lien web vers la page du document.
Par exemple : https://gallica.bnf.fr/ark:/12148/bpt6k33646735
La difficulté a été ici d'adapter notre code déjà créé pour gérer une chaîne de caractères au lieu d'un numéro d'identification. Nous avons remarqué que le début de l'identifiant était toujours le même, alors nous avons choisi de ne stocker dans la base de données que la partie qui variait, pour faire des économies de place si de nombreux livres venaient à être ajoutés. Pour l'exemple précédent, ce n'est que "12148/bpt6k33646735" qui est stocké dans la base de données. Lorsque l'API doit être appelée pour faire une requête par identifiant, on ajoute la partie unique à la partie commune pour retrouver le lien correspondant au document sur les serveurs de la BNF.

Emprunts :

Les emprunts sont représentés par des objets de la classe Loan, et leurs informations sont stockées dans la table SQL Loans. Un emprunt a deux statuts possibles :
En cours : La date d'expiration de l'emprunt n'est pas encore passée et le membre n'a pas encore rendu le livre.
Complété : Le membre a rendu le livre qu'il avait emprunté.
Un emprunt est considéré comme en retard s'il est toujours en cours et que la date d'expiration a dépassé la date actuelle.

2. Interface graphique

L'interface graphique de l'application est réalisée avec Java FX.

Nous avons utilisé un modèle vue-contrôleur :
https://fr.wikipedia.org/wiki/Mod%C3%A8le-vue-contr%C3%B4leur

Chaque page de l'application, qui fonctionne un peu comme un site web, est gérée par une classe Java qui joue un rôle de contrôleur : à chaque interaction avec l'utilisateur, elle modifie l'affichage en fonction du bouton cliqué, des champs complétés ou des nouvelles données en jeu et gère aussi les transitions vers les autres pages.

La partie graphique se base sur certains principes de design pour offrir une expérience utilisateur optimale.

Premièrement, un menu principal en haut assure l'accessibilité aux fonctionnalités importantes de l'application, telles que la recherche d'un membre ou l'ajout d'un emprunt. Il s'agit des onglets dont un libraire pourrait avoir le plus besoin dans son utilisation. Un menu vertical, que l'on peut faire glisser, se trouve sur le côté gauche de la page. Cela permet d'accéder aux fonctionnalités annexes sans encombrer la page d'accueil.

Au niveau fonctionnel, toute l'application tourne autour de la même page, la "home-page". Cela permet d'assurer une cohérence de mise en page. Lorsque le libraire accède aux autres pages en cliquant sur les boutons, le centre de la page d'accueil est modifié à chaque fois. Voyons en détail l'utilisation et l'accès à chaque page :

Relatives au compte libraire :
La page de connexion est la première accessible lors du lancement de l'application. Elle affiche des messages d'erreurs en fonction des champs mal remplis, permet d'accéder à la page d'inscription ainsi qu'à se connecter pour accéder à la page d'accueil. Une fois sur cette page, le libraire peut accéder à une page affichant les informations de son compte en cliquant sur l'icône en haut à droite. Il peut également se déconnecter.

Relatives aux emprunts :

Les pages permettant d'accéder aux emprunts sont :
- La page de recherche d'emprunts où une recherche par état, par membre et par livre est disponible.
- La page d'affichage de tous les emprunts.
- La page des problèmes d'emprunts.
- La page d'historique d'un membre.

Dans toutes ces pages, l'état de l'emprunt peut être modifié.

Relatives aux membres :

Il s'agit des pages :

- De recherche de membres, permettant d'effectuer une recherche par nom, prénom, ID, nombre d'emprunts, etc.
- D'affichage de tous les membres.
- De profil membre.

La page de profil membre comporte ses informations et son historique. Elle est accessible dans les pages membres en cliquant sur l'icône loupe.

Relatives aux livres :

Les pages comportant des informations sur les livres sont :

- La page du top des livres les plus empruntés, permettant de sélectionner une date de début de recherche.
- La page de recherche de livres (par ID de livre, ID de membre, titre, présent dans la bibliothèque, etc.).
- La page d'information sur le livre.

La page d'information sur le livre est accessible dans la page de recherche de livre. Lorsque le livre se trouve déjà dans la base de données, une icône livre apparaît à côté de ses informations et permet d'accéder à sa page. Si le livre n'est pas dans la base de données, il suffit de cliquer sur l'icône plus présent à la place pour l'ajouter et accéder à sa page d'information.

3. Communication avec l'API

Ce projet a nécessité la communication avec l'API de la Bibliothèque Nationale de France (BNF). Nous devions l'utiliser pour obtenir les informations des livres (titre, auteur, date, etc.) et stocker ces informations en local était interdit sauf pour l'identifiant du livre.

Description de l'API :

Gallica est une API de recherche qui permet à un utilisateur de se connecter à une page web en faisant une requête de recherche de document multicritères. Parmi ces critères, on trouve notamment : titre, auteur, date de publication, éditeur, identifiant, langue, etc.

Une présentation plus détaillée de l'API et un tutoriel sont disponibles ici : https://api.bnf.fr/fr/api-gallica-de-recherche

Requête :

La requête est un paramètre GET à spécifier de l'URL :
https://gallica.bnf.fr/SRU?operation=searchRetrieve&version=1.2&query=
(Les deux autres indiquent l'opération voulue et la version utilisée.)

Cette requête doit être écrite en langage CQL, un langage de manipulation de données proche du SQL mais utilisé pour sa syntaxe plus naturelle.
Plus d'informations sur le langage CQL : https://www.loc.gov/standards/sru/cql/spec.html

Une requête de base est par exemple : dc.subject any informatique

Elle indique que l'on doit chercher "informatique" dans le thème du document. "any" indique dans le cas où on a mis plusieurs mots séparés par des espaces que seul l'un d'entre eux suffit

pour satisfaire la recherche. Si l'on veut une recherche exacte avec plusieurs mots il faut utiliser "adj" ou "all" à la place.

On peut également rajouter des paramètres optionnels à l'URL. Par exemple "maximumRecords" qui permet de choisir le nombre de résultat maximum pour une requête. La valeur par défaut est 15, la valeur maximale est 50. Dans ce projet, nous l'avons fixé à 30.

Données reçues :

Voici le résultat qu'on obtient pour la requête précédente :
https://gallica.bnf.fr/SRU?operation=searchRetrieve&version=1.2&query=dc.subject%20any%20informatique

C'est un fichier XML que nous devons parcourir. Il contient les documents trouvés par l'API. Les premières balises contiennent des informations sur la requête comme le nombre de résultats trouvés, des messages d'erreurs éventuellement, etc. Les suivantes sont des paquets contenant les informations des documents. Les sous-balises sont nombreuses, mais la majorité des informations qu'on veut obtenir se trouvent dans la balise <oai_dc:dc>.

Celle-ci contient les attributs du document au format Dublin Core (description détaillée : https://www.bnf.fr/fr/dublin-core), un format qui contient 15 champs principaux pour décrire le document.

La difficulté principale en Java a été de gérer le fait qu'il n'y a pas beaucoup d'informations sur ces attributs : on ne sait pas s'ils peuvent être présents plusieurs fois ou pas du tout dans le XML, quelles sont les valeurs possibles pour certains (dc.type par exemple, nous les avons déduites en observant les résultats sur un certain nombre de requêtes), et aussi qu'à certains moments, il peut apparaître des erreurs spontanément sur des requêtes banales, sans que des causes les justifient. Il a donc fallu mettre en place certaines sécurités et on ne sait toujours pas si notre implémentation couvre toutes les erreurs possibles (elle couvre cependant celles qu'on a pu croiser).

III. Travail en équipe

1. Organisation

Nous avons majoritairement travaillé en distanciel, chaque membre du groupe gérant le rythme de ses contributions comme il le souhaitait.
Pour faciliter la communication sur le projet et le partage des fichiers, nous avons travaillé sur un serveur Discord™ et avons pris avantage du système de salons textuels et vocaux pour mieux nous organiser.
En parallèle, nous avons déposé les avancées du code régulièrement sur un dépôt Github, comme demandé dans les consignes. Nous avons mis en place une règle qui consiste à prévenir les autres quand l'on fait un commit important sur le dépôt (plusieurs fichiers affectés, dépendances modifiées, etc.).

2. Répartition des tâches

Voici le tableau de répartition des tâches pour ce projet :

Quentin	Alexandre	Pauline	Théo	Emilien
Interface graphique (FXML), documentation, UserInput, Script Shell, pages relatives aux membres, lien avec la BDD pour la connexion/création de compte	Communication avec l'API, version ligne de commande, diagrammes, rapport	Interface graphique (FXML), contrôleurs, diagrammes, rapport	Base de données (DBHandler), Script et requêtes SQL, noyau du logiciel (Core), diagrammes, rapport	Classes

3. Réunions avec le tuteur de projet

Nous avons eu au total 3 réunions avec notre tuteur de projet : M. Haddache, enseignant en informatique à CY Tech. Cela nous a permis d'avancer sur la bonne voie tout au long du projet, surtout sur les choix de conception. En dehors de ces points d'avancement, nous avons aussi pu lui poser des questions par message Teams™ quand nous en avions besoin.

Première réunion :

Le **10/05**, nous avons organisé une première réunion pour discuter de l'énoncé et des principaux choix de conception sur le modèle de données.
Nous avons montré la première version de nos diagrammes de classes et de cas d'utilisation sur lesquels nous avons reçu des conseils : les méthodes de gestion de la base de données étaient mal placées car elles n'étaient pas directement reliées aux classes de données (Livre, Client, Emprunt), ce qui allait compliquer la gestion car il aurait été nécessaire de passer tout le temps par une classe tierce pour créer des objets ou les mettre à jour.

Deuxième réunion :

Le **17/05**, nous nous sommes réunis à nouveau pour discuter de notre avancement de la semaine.

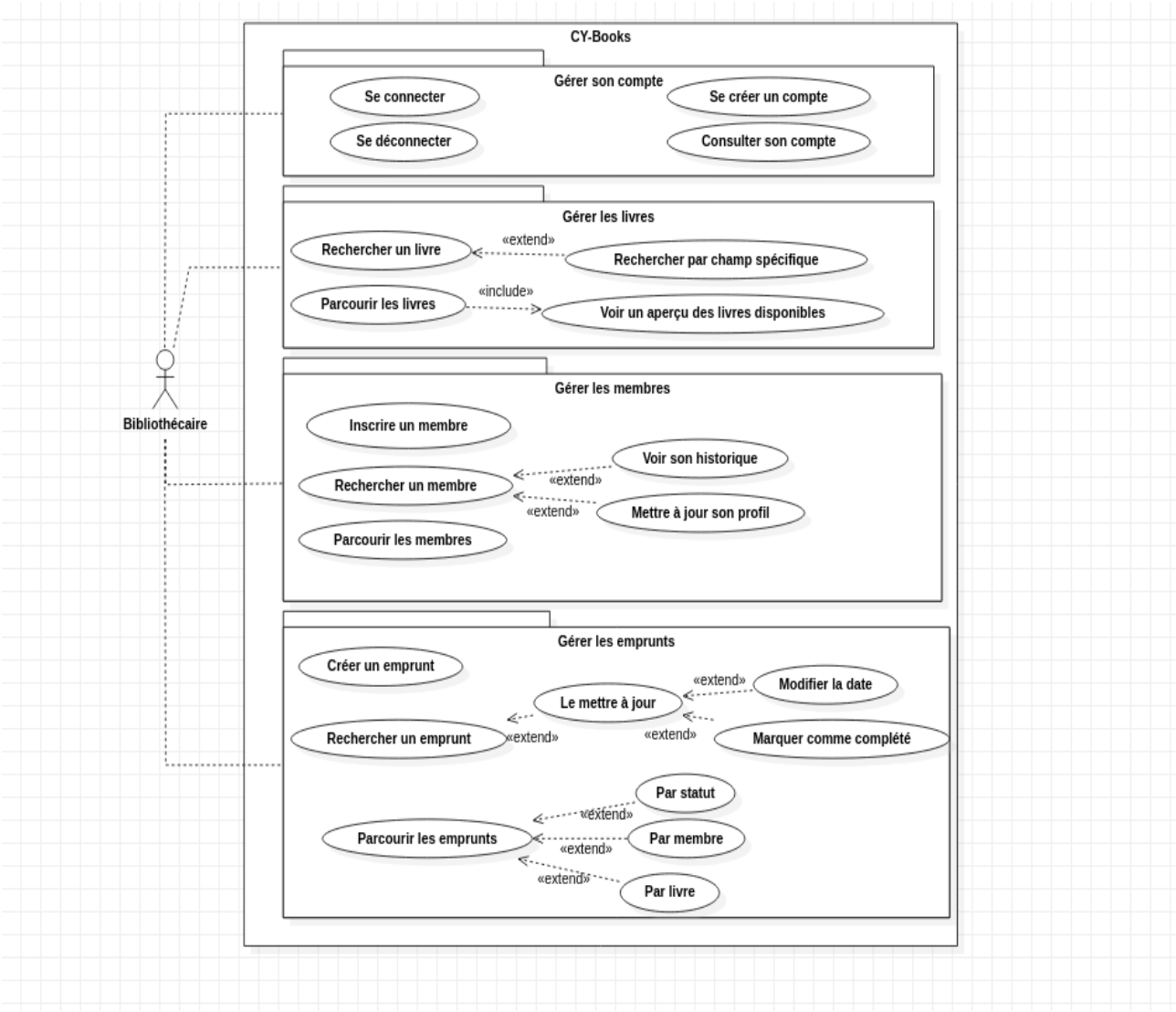
Nous avions bien avancé, la communication avec l'API ainsi qu'une grande partie des méthodes de gestion de la base de données étaient fonctionnelles. Nous avons également pu montrer un début d'interface graphique.
Il nous était apparu quelques questions supplémentaires sur les données, notamment au niveau de l'identification des retards d'emprunts. Nous avons pu éclaircir nos doutes.

Troisième réunion :

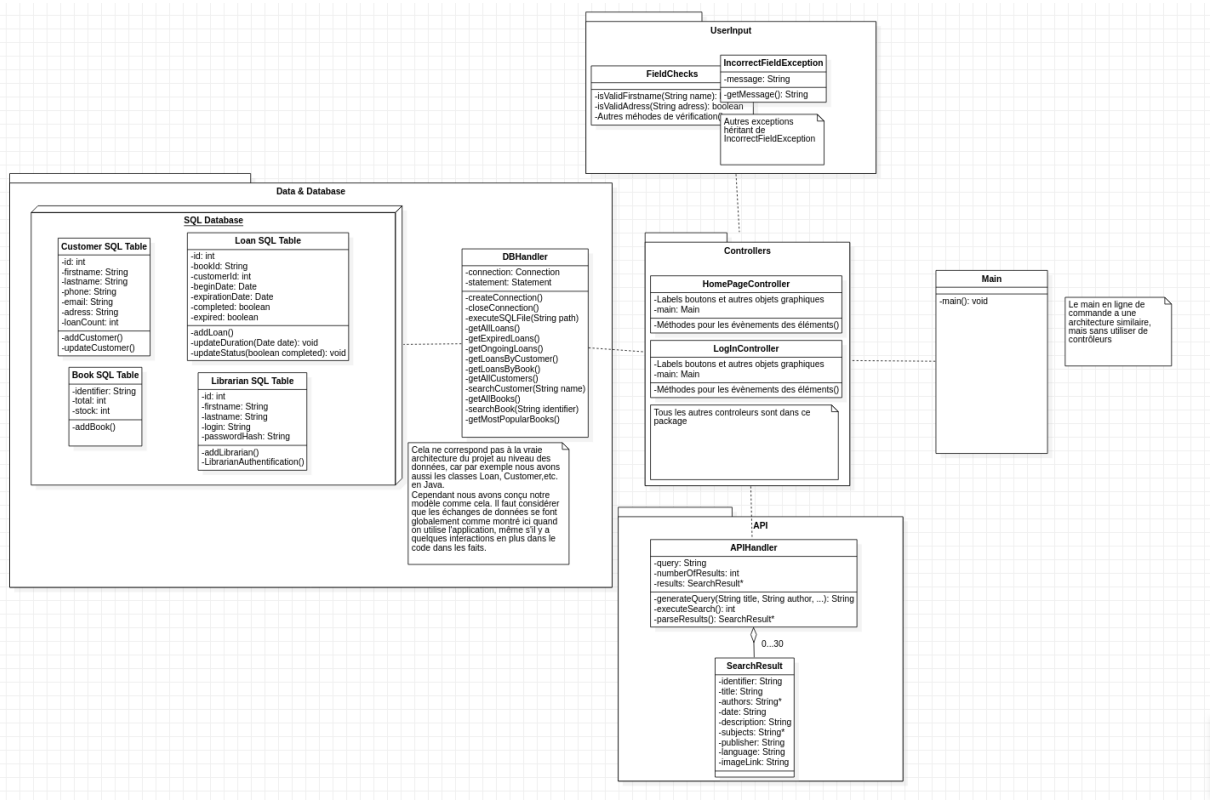
Le **22/05**, nous nous sommes rencontrés en présentiel a CY Tech pour discuter de notre avancement en vue de la dernière ligne droite.
Certains points sont encore à terminer tel que la gestion des différents filtres de recherche qui n'est pas au point ainsi que l'ajout de nouveaux livres. Il faut accélérer le développement de la partie interface graphique.

IV. Architecture de l'application

1. Diagramme des cas d'utilisation



2. Diagramme de classes



(Les diagrammes sont aussi disponibles en meilleure qualité sur le dépôt GitHub, dans le dossier ‘docs’)

V. Bibliographie

Voici quelques-unes des ressources qui nous ont aidé à réaliser ce projet :

- API de recherche Gallica : https://api.bnf.fr/fr/api-gallica-de-recherche
- MVC : https://fr.wikipedia.org/wiki/Mod%C3%A8le-vue-contr%C3%B4leur
- Langage CQL : https://www.loc.gov/standards/sru/cql/spec.html
- Format Dublin Core : https://www.bnf.fr/fr/dublin-core
- Tutoriel JavaFx : https://code.makery.ch/fr/library/javafx-tutorial/
- Stack Overflow : https://stackoverflow.com
- Palette de couleur : Palette - Coolors
- Maven : Maven – Download Apache Maven
- BCrypt : https://docs.spring.io/spring-security
- JDBC : https://docs.oracle.com/