

# **DOSSIER PROJET**

TITRE RNCP VI,

DÉVELOPPEUR CONCEPTEUR D'APPLICATION

Romane BOIREAU



# Tables des matières

Tables des matières	3
Introduction	5
Introduction (English version)	6
Description	7
Liste de compétences du référentiel	7
Cahier des charges	8
1. Présentation	8
2. Besoins	9
3. Spécifications fonctionnelles et techniques	11
4. Contraintes	13
5. Stack technologique	14
Gestion du projet	14
1. Équipe	14
2. Planning et suivi	15
Conception visuelle	16
1. Recherche	17
2. Premières maquettes	18
3. Charte graphique	19
4. Maquettes finales	21
Environnement de travail	24
Organisation du Projet	25
1. Front	26
A. Technologies et outils	26
B. Arborescence	28
2. Back	30
A. Technologies et outils	30
B. Arborescence	32
3. Base de données	34
4. Endpoints API	35
Modélisation des données et application	39
1. MCD (modèle conceptuel de données)	39
2. MLD (modèle logique de données)	40
3. MPD (modèle physique de données)	41
4. Prisma ORM	41
A. Introduction	41
B. Schéma	42
C. Exemple de requête	43
Réalisation	45
1. Authentification	45

A. Incription	46
Front	46
Back	47
B. Connexion	48
Front	48
Back	49
C. Rafraîchissement des tokens	50
Front	50
Back	51
D. Mot de passe oublié	51
Front	52
Back	52
E. Réinitialisation du mot de passe	53
Front	53
Back	54
<b>Objectifs de qualité</b>	<b>55</b>
1. Commentaires	55
2. Documentation	55
3. Standards et bonnes pratiques	56
Difficultés techniques et humaines	57
1. Authentification des utilisateurs	57
2. Deuxième version	58
<b>Veille</b>	<b>59</b>
<b>Futures réalisations et évolutions possibles</b>	<b>60</b>
1. Futur proche	60
A. Tests	60
B. Intégration Continue	60
C. Déploiement	61
2. Moyen-long terme	61
<b>Conclusion</b>	<b>62</b>

# Introduction

J'ai eu la chance de grandir au sein d'un entourage très porté sur la littérature et qui m'y a introduit très tôt. J'ai passé énormément de temps à lire pendant mon enfance, au point que la punition par défaut était de me priver de lecture. J'ai longtemps gardé cette passion puis petit à petit, avec le collège, j'ai lentement perdu le rythme. Cette phase a duré de nombreuses années, jusqu'à ma deuxième année de licence, durant laquelle je lisais péniblement un livre en entier. Une amie qui lisait beaucoup avait créé un compte Instagram dédié à ses lectures et c'est en la suivant et en interagissant avec d'autres membres de cette communauté que ma curiosité s'est ranimée. J'ai sérieusement repris la lecture et ai même fini par, moi aussi, créer un compte dédié à ce sujet. J'ai beaucoup apprécié le temps que j'ai passé à créer ces posts et à interagir avec d'autres personnes.

Cependant, Instagram, comme la plupart des réseaux sociaux, se base sur un algorithme qui met en avant les comptes publient souvent en suivant les tendances actuelles. La lecture est une activité qui ne se fait pas dans l'urgence, elle se heurte donc à ce besoin de poster presque quotidiennement pour atteindre ou garder une certaine popularité. En tant que lectrice, je n'ai pas envie de me sentir sous pression de finir un livre pour pouvoir rapidement le transformer en post qui s'alignera sur ce qui est populaire à ce moment. Et ce, dans le simple but de rester dans la boucle et éviter d'être pénalisée par une période d'inactivité jugée trop grande. De plus, un post sur Instagram doit obligatoirement avoir un média de couverture, il faut donc également penser à travailler longuement ce visuel pour espérer plaire au plus grand nombre. Tout le monde n'a pas les moyens d'investir dans une caméra de qualité ou simplement de fibre artistique permettant la création d'un média attrayant. Ces contraintes sont souvent pointées du doigt, source de frustration et de découragement pour les personnes plus attirées par le partage que par la satisfaction de l'algorithme.

C'est en observant les sites tels que Babelio et Goodreads, deux plateformes dédiées au partage de critiques littéraires, ainsi qu'en discutant de ce sujet avec des amies, que j'ai commencé à envisager Biome. Les défauts de ces deux plateformes sont la quantité d'informations présentée à un nouvel arrivant, les sites ont un visuel très chargé qui est plutôt intimidant et peu accueillant pour un débutant. Je souhaitais créer quelque chose qui allie l'interface d'Instagram tout en gardant le contenu mis en avant sur ces autres plateformes. Biome est actuellement plus proche d'un carnet de lecture numérique que d'un réseau social, il se concentre sur le plaisir simple de lire, de réfléchir et de partager.

# Introduction (English version)

I was lucky enough to grow up in a literary family who introduced me to literature at a very early age. I spent an enormous amount of time reading as a child, to the point where the default punishment was to deprive myself of reading. I kept up this passion for a long time, then gradually, as I entered secondary school, I slowly lost the rhythm. This phase lasted for many years, right up to my second year of undergraduate studies, during which I could barely read an entire book. A friend who read a lot had created an Instagram account dedicated to her reading and it was by following her and interacting with other members of this community that my curiosity was rekindled. I seriously got back into reading and even ended up creating a dedicated account myself. I really enjoyed the time I spent creating these posts and interacting with other people.

However, Instagram, like most social networks, is based on an algorithm that highlights accounts that often publish following current trends. Reading is an activity that isn't done in a hurry, so it comes up against this need to post almost daily to achieve or maintain popularity. As a reader, I don't want to feel under pressure to finish a book so that I can quickly turn it into a post that aligns with what's popular at the moment, simply to stay in the loop and avoid being penalized by a period of inactivity deemed too long. What's more, a post on Instagram is bound to have a media cover, so you also need to think long and hard about your visuals if you hope to appeal to as many people as possible. Not everyone has the means to invest in a quality camera, or the artistic fiber to create attractive media. These constraints are often pointed out as a source of frustration and discouragement for people who are more interested in sharing than in satisfying the algorithm.

It was while observing sites such as Babelio and Goodreads, two platforms dedicated to sharing literary reviews, and discussing the subject with friends, that I began to consider Biome. The shortcomings of both platforms are the amount of information presented to a newcomer, and the sites have busy visuals that are often intimidating and unwelcoming to a beginner. I wanted to create something that combined the interface of Instagram while retaining the content highlighted on these other platforms. Biome is currently closer to a digital notebook than a social network, focusing on the simple pleasure of reading, reflecting and sharing.

# Description

Biome est une application web permettant à ses utilisateurs de créer une bibliothèque en ligne. Chaque personne peut y ajouter les livres lus en renseignant l'ISBN (International Standard Book Number) qui le numéro d'identification unique d'un livre ainsi qu'un avis (optionnel). L'objectif est de proposer un espace numérique permettant de garder une trace de ses lectures et de les partager sans pression sociale ou d'intensité de posts.

L'application permet notamment :

- L'ajout de livres grâce à leur ISBN, les informations comme l'auteur et le titre sont automatiquement récupérés à partir de celui-ci via les API Google Books et Open Library
- Gestion d'un espace personnel (bibliothèque personnelle)
- Rédaction et consultation d'avis

Le projet a été développé avec React et TypeScript pour le frontend, Node et Express pour le backend. Les données sont stockées au sein d'une base de données PostgreSQL via l'ORM Prisma. L'authentification repose sur un système de JSON Web Tokens (JWT) sécurisé via des cookies httpOnly et un Context React d'authentification. Deux API externes permettent de récupérer automatiquement les informations d'un livre à partir de son ISBN : Open Library et Google Books.

Le design a été pensé pour être simple, direct et accessible. Biome s'adresse à toute personne lisant régulièrement ou occasionnellement, qui souhaite garder une trace de ses lectures à son rythme, sans contrainte de publication et partager son avis avec d'autres membres de la plateforme.

# Liste de compétences du référentiel

N° fiche	Activités types	N° fiche	Compétences professionnelles
1	Développer une application sécurisée	1	Installer et configurer son environnement de travail en fonction du projet
		2	Développer des interfaces utilisateur
		3	Développer des composants métier
		4	Contribuer à la gestion d'un projet informatique
2	Concevoir et développer une application sécurisée organisée en couches	5	Analyser les besoins et maquetter une application
		6	Définir l'architecture logicielle d'une application
		7	Concevoir et mettre en place une base de données relationnelle
		8	Développer des composants d'accès aux données SQL
3	Préparer le déploiement d'une application sécurisée	9	Préparer et exécuter les plans de tests d'une application
		10	Préparer et documenter le déploiement d'une application
		11	Contribuer à la mise en production dans une démarche DevOps

# Cahier des charges

## 1. Présentation

Biome est une application web destinée aux lecteurs et aux lectrices souhaitant gérer une bibliothèque personnelle en ligne. Elle permet d'ajouter des livres, de rédiger des avis, de consulter les livres ajoutés par soi-même et par les autres utilisateurs, de gérer ses ajouts. Le projet s'inscrit dans le cadre de la validation du titre de Concepteur Développeur d'Applications, et s'inspire d'outils existants comme Goodreads, Babelio et Instagram. Il cherche à s'affranchir des contraintes algorithmiques imposées par les réseaux sociaux et garantir une interface accessible aux nouveaux utilisateurs.

## 2. Besoins

Les plateformes comme Goodreads ou Babelio offrent une large base de données et de nombreuses options pour leurs utilisateurs, mais peuvent paraître complexes, peu intuitives due à la quantité d'informations présente, et impersonnelles pour certains lecteurs et lectrices. À l'inverse, Instagram offre un espace d'expression plus visuel, mais qui crée une pression de publication et une dépendance aux algorithmes. Deux éléments peu compatibles avec la nature lente et non précipitée de la lecture.

Biome doit donc proposer une alternative plus personnelle et sans contraintes, pensée avant pour permettre à l'utilisateur de créer sa bibliothèque à son propre rythme et de partager son avis.

### L'utilisateur a besoin de pouvoir :

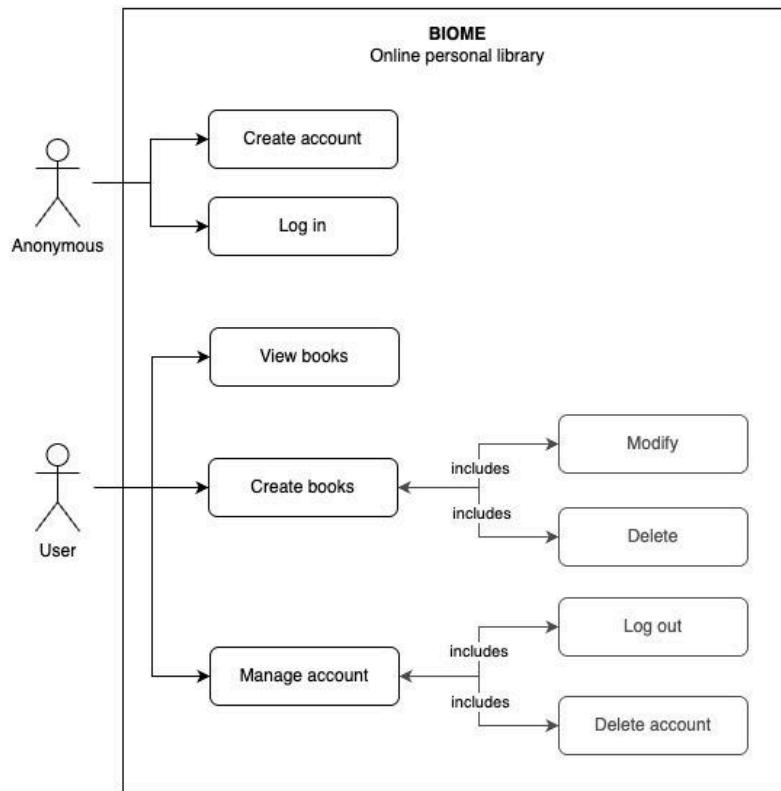
- Ajouter un livre et s'il le souhaite un avis
- Supprimer l'avis et éventuellement le livre
- Consulter ses livres ajoutés et ceux des autres
- Ajouter un livre grâce à son ISBN
- Accéder à ses informations personnelles et pouvoir les modifier
- Se connecter et se déconnecter de son compte

- Supprimer son compte et les informations liées

Une personne non connectée doit pouvoir :

- Créer un compte et s'y connecter de manière sécurisée

Voici un schéma illustrant les besoins évoqués plus tôt :



Au niveau des besoins non fonctionnels :

- L'interface doit être simple, fluide et accessible
- La navigation doit être rapide sans temps de chargement excessif
- L'affichage des erreurs doit être clair et compréhensible
- Stockage sécurisé des données (authentification, routes sécurisées...)
- L'interface doit prendre en compte les règles d'éco-conception

Grâce à l'établissement de ces besoins, j'ai pu ensuite définir les spécifications fonctionnelles de l'application.

### 3. Spécifications fonctionnelles et techniques

Les spécifications fonctionnelles décrivent ce que l'application doit faire du point de vue d'un utilisateur. Les spécifications techniques sont destinées aux personnes qui vont la développer, elles décrivent comment l'application va fonctionner. Elles sont très importantes pour cadrer le projet et déterminer comment arriver d'un point A à un point B en théorie.

#### Ajout d'un livre

##### Spécifications fonctionnelles :

- L'utilisateur doit pouvoir ajouter un livre en rentrant l'ISBN du livre
- Lorsque l'ISBN est saisi, les champs du titre et de l'auteur sont automatiquement remplis
- L'utilisateur doit pouvoir laisser son avis sur le livre via un champ de texte s'il le souhaite
- L'utilisateur doit pouvoir modifier cet avis après ajout dans sa bibliothèque, et le supprimer s'il le décide

##### Spécifications techniques :

- Utilisation des API Open Library et Google Books pour récupérer les informations du livre, telles que le titre et l'auteur grâce à son ISBN
- Stocker les livres dans une base de données PostgreSQL
- Utiliser Prisma ORM pour interagir avec la base de données
- Créer des routes API avec Express.js (POST, DELETE...) pour ajouter et supprimer un livre ou un avis de la base de données

#### Authentification et gestion des utilisateurs

##### Spécifications fonctionnelles :

- L'utilisateur doit pouvoir créer un compte avec un nom d'utilisateur, un email et un mot de passe
- L'utilisateur doit pouvoir se connecter avec l'email et le mot de passe utilisé pour la création du compte
- L'utilisateur doit pouvoir demander la réinitialisation de son mot de passe avec l'email utilisé pour la création de compte
- L'utilisateur doit pouvoir se déconnecter et supprimer son compte s'il le souhaite à tout moment
- Si un ou les identifiants de connexion sont incorrects, un message d'erreur en informe l'utilisateur
- L'utilisateur connecté doit rester connecté même après avoir rafraîchi la page

#### Spécifications techniques :

- Utiliser les JSON Web Tokens (JWT) pour gérer l'authentification.
- Utiliser un contexte React pour partager le statut de connexion à l'ensemble de l'application
- Hacher les mots de passe avec bcrypt
- Mettre en place un middleware Express pour protéger les routes sensibles telles que celles d'ajout de livre ou du compte utilisateur

## 4. Contraintes

### **Temps**

Ce projet doit être réalisé sur une période de 12 mois en parallèle de mon alternance, il me faut donc prévoir une organisation adéquate.

### **Budget**

Mes ressources financières à investir dans ce projet sont limitées, je dois prendre en compte cet aspect lors du choix de mes technologies et me diriger principalement vers celles sans frais d'utilisation. Le principal coût sera l'hébergement de l'application.

## **Utilisateurs**

Les données des utilisateurs doivent être sécurisées (hachage des mots de passe, tokens sécurisés...). L'interface doit être fluide et agréable à utiliser (API rapide et légère, pas ou peu de temps de chargement...) mais également accessible pour se conformer aux normes RGAA.

## **Maintenabilité**

Mettre en place une structure organisée et claire afin d'en garantir l'évolution et la maintenabilité (variables, CSS, composants React, documentation...)

## 5. Stack technologique

Ci-dessous, un tableau illustrant les technologies et les outils utilisés pour ce projet. Ceux-ci seront détaillés plus amplement dans les sections dédiées dans le but d'éviter de créer une redondance d'explications.

Frontend	React (TypeScript)
Backend	Node.js + Express.js (TypeScript)
Base de données	PostgreSQL + Prisma ORM
Interface de base de données	Pg admin 4
Authentification	JWT + HttpOnly cookies + bcrypt
API externes	OpenLibrary + Google Books
IDE	Visual Studio Code
Versionning	Git + GitHub

# Gestion du projet

## 1. Équipe



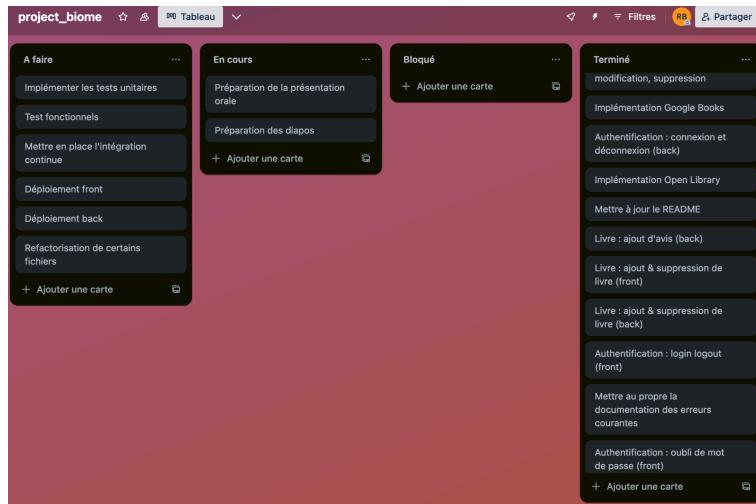
Romane BOIREAU

Développeuse frontend en alternance à AB Tasty

<https://github.com/aanatema>

## 2. Planning et suivi

Étant seule en charge du développement de ce projet, j'ai choisi d'appliquer la méthodologie **Agile** via sa variante **Kanban**. J'ai donc rapidement mis en place un Trello avec quatre grandes catégories afin de structurer mon travail, gagner en efficacité et avoir une vision claire. J'ai commencé par recenser l'ensemble des tâches principales à accomplir, sans me soucier dans un premier temps de leur ordre ou de leur priorité. Les catégories sont "à faire", "en cours", "bloqué" et "terminé". J'ai ajouté la catégorie "bloqué" pour gagner en précision. Elle me permet de ranger les tâches qui ne peuvent être complétées, soit parce qu'elles dépendent d'une autre tâche en cours, soit, car je rencontre des difficultés à l'implémenter. Je connais ainsi le statut de toutes mes tâches en un coup d'œil. Cela m'a permis d'avoir une vision globale du projet et de mieux cerner les différentes fonctionnalités à implémenter.



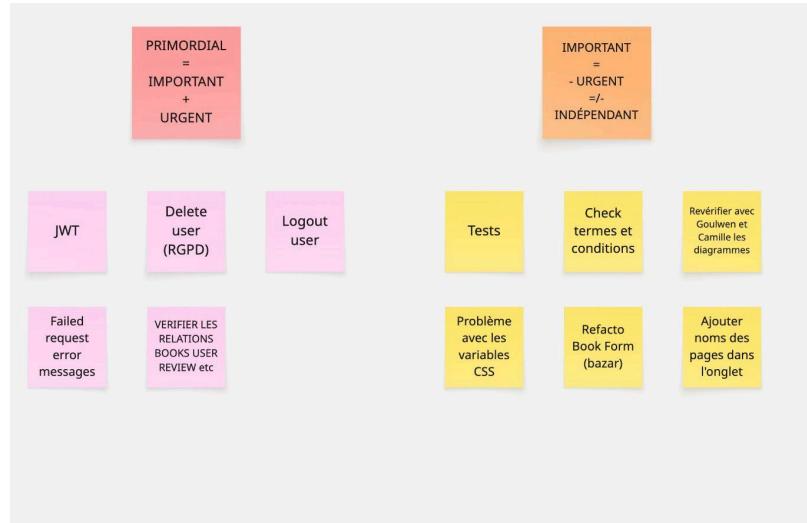
Je me suis assez vite rendu compte que cette organisation n'était pas suffisante. Elle fonctionne très bien pour lister toutes les tâches principales, mais ne me permet pas vraiment de scinder en sous-tâches et d'établir les priorités de manière claire. J'ai donc créé un autre système : j'ai sélectionné des tâches puis je les ai réparties en trois catégories. Ces catégories sont plus ou moins subjectives, car elles reposent essentiellement sur mon ressenti et mon interprétation de la difficulté que représentent les tâches.

### **Primordial**

Dans cette catégorie, j'ai rangé les éléments que j'estimais impératifs pour ce projet et/ou qui me prendrait un temps non négligeable à réaliser. Que ce soit par leur complexité, par ma non-connaissance du sujet ou encore par leur importance vis-à-vis du référentiel de compétences.

### **Important**

Ici, sont réparties les tâches qui sont importantes pour le bon fonctionnement du projet, mais que j'estime ne pas être urgentes ni bloquantes si elles sont réalisées plus tard dans la chronologie. La plupart peuvent également être effectuées de manière indépendante, ce qui me permet de m'y pencher quand je le souhaite.

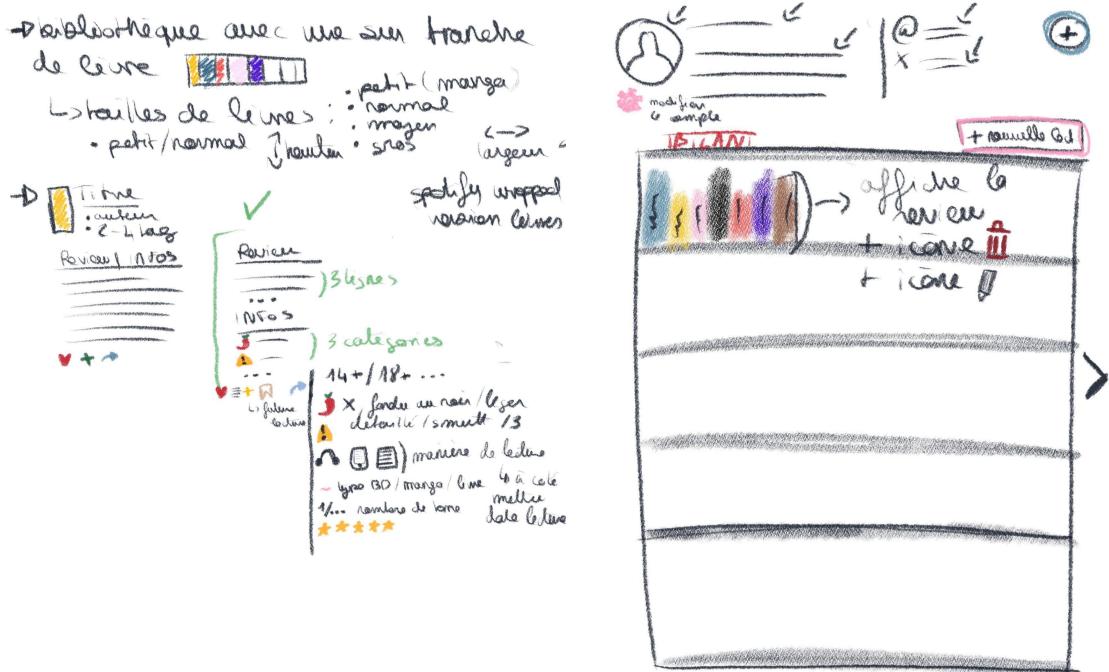


## Conception visuelle

Sketch initial	Procreate
Maquettes	Figma
Couleurs	Inspirations palettes Pinterest + Coolors

### 1. Recherche

J'ai, dans un premier temps, réfléchi de mon côté aux éléments me paraissant importants pour ce projet. J'ai pris des notes puis j'ai rapidement échangé avec une amie grande lectrice afin d'avoir son retour pour améliorer l'idée de base. Nous avons discuté via visioconférence et j'ai fait quelques croquis des différentes idées évoquées.



Premières esquisses d'idées pour Biome

Après ce premier échange, j'ai exposé mon projet dans un des Slack de mon école, Ada Tech School. J'ai ensuite proposé aux lecteurs et aux lectrices qui seraient intéressés par celui-ci d'échanger afin d'avoir l'avis de profils divers.

Il a été très intéressant d'échanger des différentes possibilités et des fonctionnalités que Biome pourrait avoir en fonction des préférences personnelle de chaque lecteur et lectrice. Cela m'a également permis de me rendre très rapidement compte du besoin impératif d'avoir une première base simple, mais efficace, qui satisferait le plus grand nombre avant d'envisager les améliorations suggérées. Le but de mon projet est d'avoir quelque chose de fonctionnel qui répond à un besoin primaire et non d'anticiper tous les besoins spécifiques. Je me suis donc obligée à me limiter à quelque chose de très simple, car j'ai tendance à partir dans plusieurs directions à la fois tout en voulant la meilleure version possible.

Je suis alors restée sur le principe d'une bibliothèque personnelle, où l'utilisateur peut ajouter sa dernière lecture ainsi que son avis s'il le souhaite. Il peut lire les avis des autres utilisateurs sur les différents livres ayant été ajoutés.

## 2. Premières maquettes

J'ai développé mes premières maquettes avec Figma par rapport aux éléments relevés lors des discussions que j'évoque plus haut. J'ai essayé de faire une première version très complète dans l'objectif de ne pas avoir à y revenir souvent.

The image shows two side-by-side wireframes from Figma. On the left is a 'New reading' form for adding a book. It includes fields for Title (with placeholder 'The priory of the orange tree'), Author name (Samantha Shannon), Illustrator name (None), Genre (Fantasy), Sub-genre (High-Fantasy), Themes / Tropes, Rating (5 stars), Recommended Age (Teenager (16+)), Spicy Level (None), Trigger Warning (None), Tome n° (1), Format (Physical), Start date (12/31/2024), and Finish date (01/07/2025). A review text area says 'The characters are really well written, I enjoyed the parallel between...'. A 'Save' button is at the bottom. On the right is a 'Your stats' interface showing a grid of book spines with the word 'Title' repeated across them. Below the grid is a large empty area labeled 'next shelves'.

Maquettes du formulaire d'ajout de livre et de la bibliothèque des lectures ajoutées.

Cette intention, bien que compréhensible, n'était pas très réaliste et c'est pourquoi un certain nombre de problèmes sont apparus avec un peu de recul. Penchons-nous sur ces éléments :

### Adaptabilité

Ces maquettes n'ont pas été créées en ayant en tête le principe d'adaptabilité à différents supports (mobile, tablette...). Le design est donc assez complexe à transformer et à adapter pour des écrans plus petits.

### Accessibilité

Il est difficile de lire le titre, car il est affiché en assez petit sur la tranche du livre. Les livres sont collés les uns aux autres, les rendant difficilement sélectionnables, ce qui n'est pas adapté pour des personnes avec des déficiences motrices.

## **Complexité visuelle**

Le formulaire d'ajout de livre présentait également un problème majeur. En voulant permettre autant de précision que possible, j'ai créé un formulaire imposant et lourd, autant à regarder qu'à remplir, ce qui aurait eu tendance à décourager les utilisateurs face à la quantité d'informations demandées.

Cette première conception n'était donc pas viable, trop peu accessible, trop chargée et visuellement, le design ne me satisfaisait pas complètement. Cette dernière problématique m'a poussé à passer par l'élaboration d'une charte graphique avant de créer les maquettes finales.

## 3. Charte graphique

Pour me permettre d'avoir une direction visuelle clairement établie à laquelle je pourrais me référer, j'ai établi une charte graphique. Afin de rester sobre, efficace et accessible dans mon design, j'ai pris le parti d'utiliser les polices d'écriture par défaut du système qui affichera Biome. L'utilisateur n'aura donc pas besoin de télécharger une police spécifique.

Ainsi, pour Windows, la police qui s'affiche sera Segoe UI, pour Mac OS, ce sera San Francisco et sur Linux, Ubuntu.

### **CHARTE GRAPHIQUE**

POLICES D'ÉCRITURE

#### **SEGOE UI**

WINDOWS

ABCDEFGHIJKLMNOPQRSTUVWXYZ

abcdefghijklmnopqrstuvwxyz

0123456789

#### **SAN FRANCISCO**

MAC OS

ABCDEFGHIJKLMNOPQRSTUVWXYZ

abcdefghijklmnopqrstuvwxyz

#### **UBUNTU**

LINUX

ABCDEFGHIJKLMNOPQRSTUVWXYZ

abcdefghijklmnopqrstuvwxyz

Pour les couleurs, j'ai choisi de prendre une base de couleurs primaires pour créer des variations modernes et harmonieuses.

La couleur principale de l'application est un blanc cassé (White #FFF5F5), il est plus doux qu'un blanc pur ce qui diminue la fatigue visuelle.

Le jaune (Yellow #FBC454) est utilisé pour accentuer certains éléments, notamment au survol des boutons de la barre de navigation.

Le rouge (Red #DB3E1D) attire facilement le regard, il est utilisé pour les éléments ayant trait à la suppression.

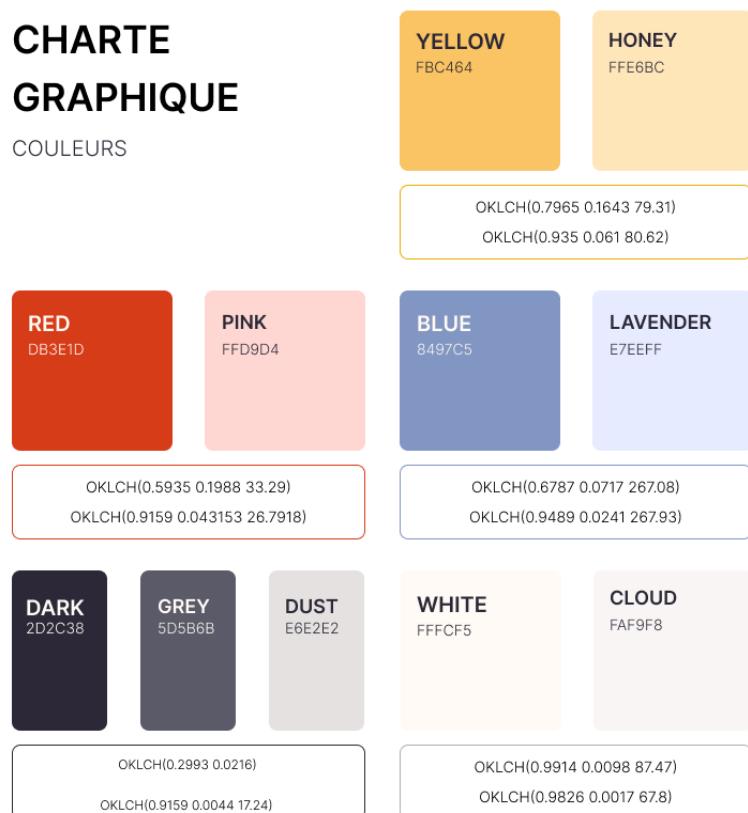
Le bleu (Blue #8497C5) n'est pas présent dans les maquettes qui seront ensuite présentées, car il sera utilisé pour une fonctionnalité future de marque-page, permettant de sauvegarder des avis ou livres afin de les consulter plus tard.

Le noir (dark #2D2C38) est utilisé sur les éléments principaux comme les boutons de confirmation.

J'ai décidé de décliner toutes ces couleurs dans le but d'utiliser leurs variantes pour accentuer des éléments secondaires, voire tertiaires.

## CHARTE GRAPHIQUE

### COULEURS



Tous les éléments interactifs ont une zone interactive d'un minimum de 44 px de hauteur, la taille minimum de la police est de 14 px. La durée d'apparition des toasts est d'un minimum de quatre secondes pour permettre une lecture lente.

J'ai utilisé l'extension web **Site Improve Accessibility Checker** pour vérifier le contraste de mes éléments dans la phase de maquettage, puis lors du développement, afin de trouver les éléments devant être améliorés.

## 4. Maquettes finales

Une fois la charte graphique établie, je me suis penchée sur la résolution des problèmes découverts précédemment.

## **Pallier le manque d'accessibilité.**

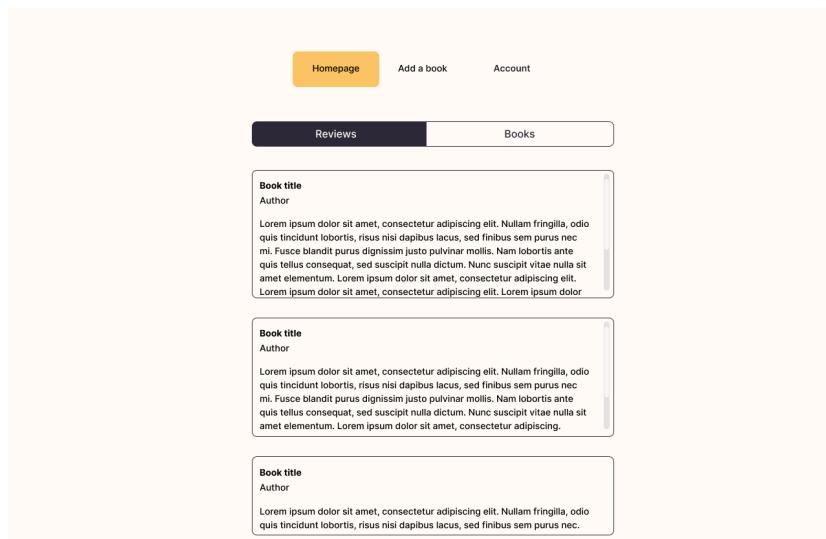
Il faut que les livres soient facilement sélectionnables et leur titre lisible peu importe la taille de l'écran

### **Éviter le trop-plein d'informations.**

Réduire la quantité d'informations demandées à l'utilisateur lors d'ajout d'un nouveau livre afin d'éviter un sentiment de charge. Seules les informations considérées comme essentielles seront demandées d'entrée.

## Avoir un style plus épuré.

Créer un style plus neutre, plus simple qui met en valeur les éléments importants, rend la navigation agréable et ôte les éléments superflus.



*Maquette de la page d'accueil une fois connecté.*

Homepage Add a book Account

**New book**

Add the ISBN of your book and we'll fetch the infos for you!

ISBN\*  
ISBN

Title\*  
Book title

Author\*  
Author

Review  
Share your thoughts here

Fields with a star ( \* ) are mandatory

Add book Clear form

Maquette de la page d'ajout de livre.

Homepage Add a book Account

**Reviews about this book**

**Username**  
Book Title - Author name

Lore ipsum dolor sit amet, consectetur adipiscing elit. Nullam fringilla, odio quis tincidunt lobortis, risus nisi dapibus lacus, sed finibus sem purus nec mi. Fusce blandit purus dignissim justo pulvinar mollis. Nam lobortis ante quis tellus consequat, sed suscipit nulla dictum. Nunc suscipit vitae nulla sit amet elementum. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Lore ipsum dolor sit amet, consectetur adipiscing elit. Lore ipsum dolor

**Username**  
Book Title - Author name

Lore ipsum dolor sit amet, consectetur adipiscing elit. Nullam fringilla, odio quis tincidunt lobortis, risus nisi dapibus lacus, sed finibus sem purus nec mi. Fusce blandit purus dignissim justo pulvinar mollis. Nam lobortis ante quis tellus consequat, sed suscipit nulla dictum. Nunc suscipit vitae nulla sit amet elementum. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Lore ipsum dolor sit amet, consectetur adipiscing elit. Lore ipsum dolor

**Username**  
Book Title - Author name

Lore ipsum dolor sit amet, consectetur adipiscing elit. Nullam fringilla, odio quis tincidunt lobortis, risus nisi dapibus lacus, sed finibus sem purus nec mi. Fusce blandit purus dignissim justo pulvinar mollis. Nam lobortis ante quis tellus consequat, sed suscipit nulla dictum. Nunc suscipit vitae nulla sit amet elementum. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Lore ipsum dolor sit amet, consectetur adipiscing elit. Lore ipsum dolor

**Username**  
Book Title - Author name

Lore ipsum dolor sit amet, consectetur adipiscing elit. Nullam fringilla, odio quis tincidunt lobortis, risus nisi dapibus lacus, sed finibus sem purus nec mi. Fusce blandit purus dignissim justo pulvinar mollis. Nam lobortis ante quis tellus consequat, sed suscipit nulla dictum. Nunc suscipit vitae nulla sit amet elementum. Lorem ipsum dolor sit amet, consectetur adipiscing elit. Lore ipsum dolor sit amet, consectetur adipiscing elit. Lore ipsum dolor

Maquette des avis sur un livre donné

Homepage Add a book Account

**Your books**

Book title Author					
Book title Author	Book title Author				

Add book Profile

Maquette présentant le profil utilisateur avec des livres ajoutés

*Maquette de la page de modification du compte utilisateur*

Les livres sont maintenant présentés sous forme de cartes. Cela permet une meilleure lisibilité et facilite grandement leur sélection, comparé avec la précédente version. Le titre ainsi que le nom du ou des auteurs sont également beaucoup plus lisibles dans cette version. Ce nouveau design est facilement adaptable sur mobile et tablette, les tailles des polices sont assez grandes pour assurer une bonne lisibilité.

Le formulaire d'ajout de livre a été simplifié pour éviter d'intimider les utilisateurs avec une trop grande quantité d'informations. Désormais, seul l'ISBN, le titre et l'auteur apparaissent lorsqu'un nouveau livre souhaite être ajouté. Le processus est davantage simplifié grâce à l'utilisation de deux API, Open Library et Google Books, permettant de récupérer un grand nombre d'informations sur un livre grâce à son ISBN. Les champs auteur et titre seront donc complétés automatiquement une fois l'ISBN ajouté.

## Environnement de travail

OS	macOS (séquoia)
Éditeur	Visual Studio Code
BDD	PostgreSQL avec Pg admin 4
Gestionnaire de paquet	Yarn v1.22.22
Node	20.9.0
Contrôle de version	Git + GitHub

J'ai changé d'ordinateur en cours d'année, passant ainsi d'un système d'exploitation Windows 11 à **macOS Séquoia**. J'ai dû me familiariser avec un environnement que je ne connaissais pas du tout, cela m'a initialement ralenti, mais le gouffre a fini par se combler. J'ai fini par trouver les raccourcis et la logique de macOS plus intuitifs que ceux de Windows.

L'IDE choisi est **Visual Studio Code**, car c'est l'éditeur de code avec lequel j'ai toujours travaillé lors de mon parcours de formation initial et de mon alternance. J'en ai amélioré la configuration initiale afin de correspondre à mes préférences de travail, ce qui m'a rendu efficace et précise dans mes manipulations.

Pour avoir une interface visuelle de ma base de données, j'ai utilisé le système **Pg admin 4**. L'interface est assez intimidante au début, mais une fois les actions principales repérées, il est facile d'y naviguer.

**Yarn** est largement reconnu pour sa rapidité d'installation et son cache hors-ligne, ce qui permet d'éviter de re-télécharger les paquets déjà installés. Lors de la comparaison avec NPM, j'ai également trouvé ses messages d'erreurs plus clairs, et en pratique, ils m'ont facilité la résolution d'erreurs liées aux dépendances.

J'ai choisi l'environnement d'exécution **Node.js** car le backend est réalisé sur une base de JavaScript, en TypeScript comme le frontend. Il possède un écosystème de bibliothèques très vaste, une large communauté active et donc quantité de ressources. C'est aussi l'environnement recommandé lorsque l'on travaille avec Prisma et Express.

J'ai utilisé **Git** et **GitHub** pour le contrôle de version et la gestion du code source. J'ai pu faire évoluer le projet sur diverses branches me permettant ainsi de travailler sur des tâches différentes en parallèle avant de fusionner celles-ci une fois les tâches terminées et fonctionnelles. Travaillant seule, je n'ai pas eu à effectuer de code review, mais si le projet doit se développer et intégrer d'autres personnes, les pull-requests et la code review seront essentiels pour une collaboration en équipe.

## Organisation du Projet

Biome est construit en trois parties.

### Frontend

Il gère l'affichage de l'interface utilisateur, ici avec les composants React, ainsi que les interactions utilisateurs. Il transmet ces actions par l'intermédiaire de requêtes vers l'API backend puis affiche les réponses obtenues de manière dynamique.

### Backend

Le backend agit comme intermédiaire entre le front et la base de données. Il reçoit et interprète les requêtes du front, vérifie les autorisations du demandeur puis

communique avec la base de données pour réaliser les actions correspondantes. Il envoie ensuite la réponse au front qui se chargera de l'afficher.

## Base de données

Elle stocke de manière permanente toutes les données importantes de l'application et les organise de manière précise. Elle est interrogée par le backend afin d'accéder ou de modifier ses données. Pour ce projet, c'est PostgreSQL qui est utilisé.

J'ai choisi de ne pas séparer les logiques front et back en deux projets distincts, car utilisant le même langage (TypeScript) dans les deux, une séparation nette ne me semblait pas nécessaire. D'un point de vue personnel, je me sens également plus à l'aise pour travailler lorsque que tout est situé au même endroit. Cependant, j'ai conscience que si le projet continue de se développer, il me faudra probablement envisager de migrer un des deux dossiers afin d'éviter une complexification trop intense de la structure.

# 1. Front

## A. Technologies et outils

Voici un tableau introductif aux technologies et outils que j'ai utilisés pour créer le frontend.

Framework	React
Styling	Tailwind CSS
Composants accessibles	shadcn/ui
Build + Dev server	Vite
Langage	TypeScript
Authentification	Context API

Après avoir bien défini mon projet, je me suis rapidement tournée vers **React** comme framework et **TypeScript** pour le langage. J'ai été initialement introduite à ces deux éléments au cours de ma formation et j'ai pu apercevoir le potentiel qu'ils renferment. J'ai eu la chance de pouvoir largement approfondir mes connaissances et ma compréhension sur le sujet au cours de mon alternance, ce qui a renforcé mon choix au cours de ce projet.

React est un des frameworks JavaScript les plus utilisés en ce moment, ce qui garantit une communauté active, des ressources nombreuses, une documentation à jour et une certaine pérennité sur laquelle se reposer.

Les avantages que React m'a apportés sont notamment :

### **Composants réutilisables**

Cela m'a permis de découper mon code en composants fonctionnels, réutilisables et qui améliorent la clarté du code. Chaque composant a un nom évocateur en plus des commentaires qui aident à la compréhension générale.

### **Mise à jour dynamique de l'interface**

À chaque changement d'état, React se met à jour dynamiquement, ce qui rend l'interface très dynamique.

### **Écosystème riche**

React m'a permis d'ajouter des bibliothèques qui m'ont été très utiles lors de mon projet, comme [React Hook Form](#) pour les formulaires, [react-router](#) pour l'organisation routes, mais aussi [shadcn/ui](#) pour des composants accessibles.

J'ai choisi d'utiliser TypeScript, car il ajoute une couche de stabilité et de clarté au code grâce à son typage fort. Il permet ainsi un meilleur maintien et évolution du projet. Voici quelques éléments techniques qui m'ont conduit à faire ce choix :

### **Prévention des erreurs**

TypeScript permet d'annoncer le type d'information que nous sommes supposés recevoir et détecte les erreurs avant même l'exécution du code si le type ne correspond pas. Un champ email d'un formulaire qui recevrait un numéro, par exemple.

### **Cohérence front / back**

TypeScript est également utilisé du côté backend, cela me permet une meilleure cohérence globale et une communication facilitée pour la gestion des modèles avec Prisma.

### **Auto-complétion et documentation**

Mon expérience a été améliorée grâce à la documentation, qui a permis de m'éclairer sur le fonctionnement interne de certains éléments, ainsi qu'à l'inférence des types.

Pour le reste, j'ai fait mon choix en prenant en compte la courbe d'apprentissage, la facilité d'utilisation et la compatibilité avec TypeScript et React, car je tenais à garder ces outils avant tout.

J'ai choisi **Vite** comme outil de bundling, parce qu'il fonctionne très bien avec React et TypeScript. Son démarrage est rapide et l'apprentissage très accessible. Il

permet de regrouper les fichiers, de compiler ceux qui doivent l'être et d'optimiser l'ensemble pour qu'il soit lisible pour le navigateur.

**Shadcn** est une bibliothèque de composants qui propose d'importer uniquement les éléments dont on a besoin et non pas la bibliothèque complète, au contraire de Material UI par exemple, ce qui allège considérablement le projet. C'est grâce à une discussion avec un collègue que j'ai découvert cette bibliothèque. Après quelques recherches supplémentaires et comparaisons, le principe d'importer un composant et d'avoir accès à son code pour le modifier à loisir m'a séduite. De plus, ses composants sont basés sur les primitives de **radix-ui** ce qui assure des éléments accessibles de qualité.

L'utilisation de shadcn m'a naturellement orientée vers **Tailwind** puisque c'est avec que ses composants sont stylisés. C'est aussi dans une volonté d'éco-conception que j'ai choisi d'appliquer cette technique au reste de mes éléments. Tailwind réduit le nombre de mes fichiers, car il me permet de styliser les composants directement à l'endroit où je les utilise au lieu d'avoir à créer une feuille de style CSS à chaque fois.

Pour gérer le partage d'information à travers l'application, comme l'état de connexion de l'utilisateur par exemple, j'ai utilisé **Context API** proposé par React. Je peux ainsi centraliser certains états ou fonctions afin de les rendre facilement accessibles lorsque j'en ai besoin, sans avoir à passer systématiquement par des props. Son utilisation et sa mise en place sont assez simples et particulièrement adaptées comparé à Redux qui serait trop lourd pour un projet de cette taille et donc inadéquat.

## B. Arborescence

Le frontend correspond à l'application React qui sert d'interface à l'utilisateur.

### **src/**

Contient toute la logique métier et l'UI.

### **api/**

Rassemble les appels aux API Google Books et Open Library.

### **components/**

Comprend tous les composants UI et est organisé en sous-dossiers pour une meilleure lisibilité : **bookComponents** pour les éléments liés aux livres tels que les cartes livres et avis, **userComponents** pour le profil utilisateur, etc.

### **context/**

Gère les contextes React comme l'**AuthContext** pour la connexion utilisateur.

### **form/**

J'ai trouvé qu'il était plus simple pour moi de travailler en ayant tous mes formulaires au même endroit tels que `LoginForm`, `ModifyUserForm`, etc. d'où la création de ce dossier.

### hooks/

En prévision de l'évolution du projet, j'ai créé ce dossier dédié aux custom hooks, c'est ici qu'est stocké `useAuth`.

### libraries/

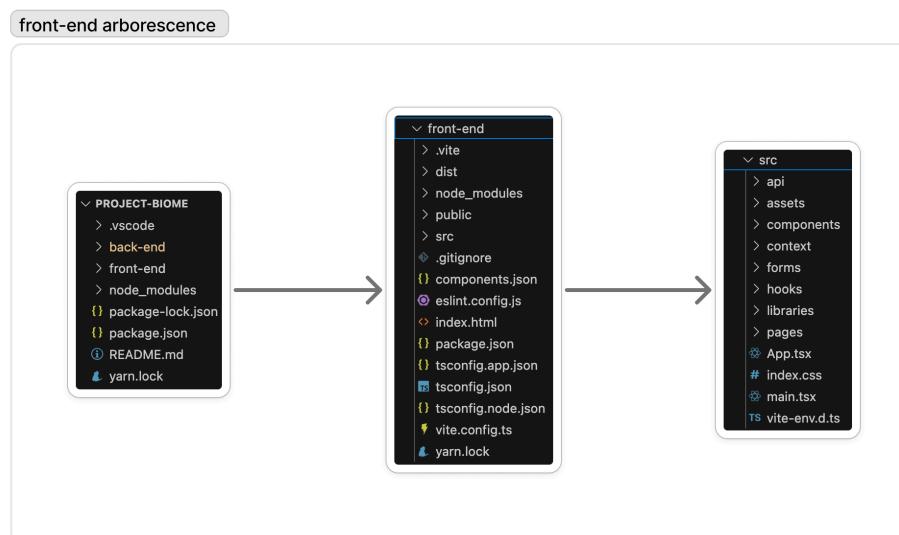
Rassemble les fichiers de configurations ou de librairies externes comme Axios et tailwind.

### pages/

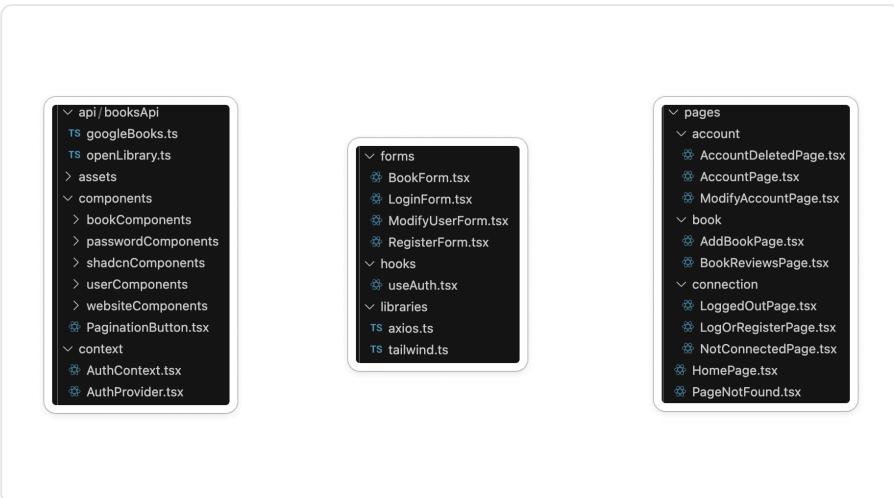
Contient toutes les pages liées au React Router, elles sont organisées en sous-sections pour garantir une organisation optimale : account, book, connection etc

Puis, viennent naturellement les fichiers globaux comme App, main, etc.

Ci-dessous, deux captures d'écran illustrant l'arborescence précédemment évoquée :



front-end arborescence



## 2. Back

### A. Technologies et outils

Ce tableau présente les éléments principaux sur lequel repose mon back.

Environnement d'exécution	Node.js
Framework HTTP	Express.js
ORM + BDD	Prisma + PostgreSQL
Authentification	JWT + cookies HttpOnly
Sécurité	CORS, cookie-parser, .env, bcrypt
Langage	TypeScript

Comme indiqué plus haut, j'ai utilisé **TypeScript** pour le front et le back. Ce choix a été motivé par plusieurs éléments :

Mon alternance s'est uniquement portée sur le front et elle a renforcé ma préférence pour ce domaine en particulier. Mes expériences avec des langages backend, tels que PHP et Java, n'ont pas été particulièrement plaisantes et je n'ai pas eu l'occasion de changer ma vision des choses. Je suis seule à porter ce projet, je dois impérativement optimiser mon temps, mon organisation et mes ressources. Faire le choix d'apprendre ou de me réintroduire à un langage à son lot de difficultés, il ne m'a

pas semblé pertinent de prendre ce risque au vu du temps que j'allais pouvoir allouer au développement de ce projet.

Par conséquent, il fallait que je trouve un langage palliant ces complications. C'est en discutant avec des personnes de l'entreprise dans laquelle j'ai effectué mon alternance que la solution m'a été suggérée. TypeScript répond parfaitement à ces problématiques. C'est un langage que je connais déjà, pour l'avoir utilisé au cours de ma formation puis quotidiennement au sein de mon alternance, je suis donc en terrain connu. Je ne reviendrai pas ici sur les avantages, car ils sont très similaires à ceux évoqués précédemment dans la partie front.

**Express.js** est un framework web minimaliste pour Node.js, il est très populaire au sein de l'écosystème JavaScript grâce à sa légèreté, sa flexibilité et sa simplicité d'utilisation. Son association avec des outils comme Prisma et les JWT m'a permis de construire un backend robuste. Voici quelques points clés

### **Structure claire**

J'ai pu mettre en place une structure claire et organisée avec Express grâce à l'utilisation de routes et de contrôleurs. Cela m'a permis d'isoler chaque fonctionnalité primaire (authentification, gestion des utilisateurs...) dans un fichier dédié, facilitant ainsi la lecture et la maintenance du code dans le temps.

### **Middleware**

Le middleware intégré m'a simplifié la gestion des erreurs et la configuration de CORS, ce qui m'a évité d'avoir du code dupliqué et répétitif pour des comportements globaux.

### **Compatibilité**

Sa compatibilité avec Prisma et la bibliothèque CORS m'a rapidement permis de construire l'API.

Prisma m'a été recommandé pour sa simplicité d'utilisation et sa configuration faite pour un projet en TypeScript, ce qui m'apporte une couche supplémentaire de sécurité typée.

### **Schéma centralisé**

Le fichier schema.prisma me permet d'avoir une vue claire et précise sur la structure de ma base de données, ce qui améliore grandement la compréhension du modèle de données.

### **Génération du client**

Prisma génère automatiquement un client en TypeScript, fortement typé, ce qui me donne accès à de la complétion et de la vérification des types lors de l'envoi de requête vers la base de données.

## Système de migration

Son système de migration versionné me donne accès à l'évolution de la base de données me permettant de suivre facilement les modifications que j'y ai apportées. Les requêtes SQL sont générées automatiquement, ce qui réduit la possibilité d'erreurs dans le code.

L'authentification des utilisateurs est un aspect essentiel de l'application, j'ai donc mis en place un système basé sur les **JSON Web Tokens** et des cookies sécurisés. Ils me permettent de vérifier que l'utilisateur connecté est bien celui qu'il prétend être et pouvoir lui donner ainsi accès, ou non, aux éléments auxquels il est autorisé.

### JWT

La librairie jsonwebtoken est stateless, cela permet une authentification sans stockage serveur. Le backend génère deux tokens de connexion, l'`access token` et le `refresh token`, lors de la création de compte. On va vérifier régulièrement que l'access token est toujours valide, si celui-ci ne l'est plus, le refresh token en génère un nouveau.

### Cookies HttpOnly

J'ai choisi de stocker les tokens dans des cookies HttpOnly. Ils ne sont donc pas accessibles via JavaScript, ce qui limite fortement les risques d'attaques de type XSS qui injectent du code malveillant pour contourner les contrôles d'accès et usurper l'identité des utilisateurs.

### Routes protégées

Certaines routes sont accessibles uniquement si un token valide est envoyé avec la requête. Le backend vérifie la validité du JWT à chaque appel afin de sécuriser les données de l'utilisateur.

Pour sécuriser les mots de passe, j'ai choisi **bcrypt**, un algorithme permettant de hacher les mots de passe des utilisateurs. Il est très largement utilisé de nos jours, car il repose sur une variante de l'algorithme de chiffrement Blowfish et y introduit un facteur coût (nombre d'itérations de hachage, ici 10) qui rend le décodage très long. Les ordinateurs devenant toujours plus rapides, ce facteur peut être augmenté afin de contrer cette vitesse, permettant ainsi de garder une sécurité optimale. Il résiste également aux attaques par force brute et est compatible avec les environnements node.js, ce qui le rend idéal pour ce projet.

## B. Arborescence

Dans le dossier backend se trouve toute la logique serveur avec Node.js et Express.

### **dist/**

Forme compilée en JavaScript des fichiers TypeScript

### **prisma/**

Contient le `schema.prisma` ainsi que toutes les migrations qui ont été effectuées.

### **src/**

Dossier principal contenant le code du serveur.

### **auth/**

Rassemble tous les fichiers liés à l'authentification tels que `auth.controllers`, `auth.cookies`, `auth.tokens` etc.

### **controllers/**

Comprend la logique métiers des entités, ce qu'elles doivent faire. ex : `bookControllers`, `userControllers...`

### **routes/**

Ce dossier rassemble les routes express comme `authRoutes`, `bookRoutes` permettant ainsi une organisation claire et allégeant les fichiers controllers.

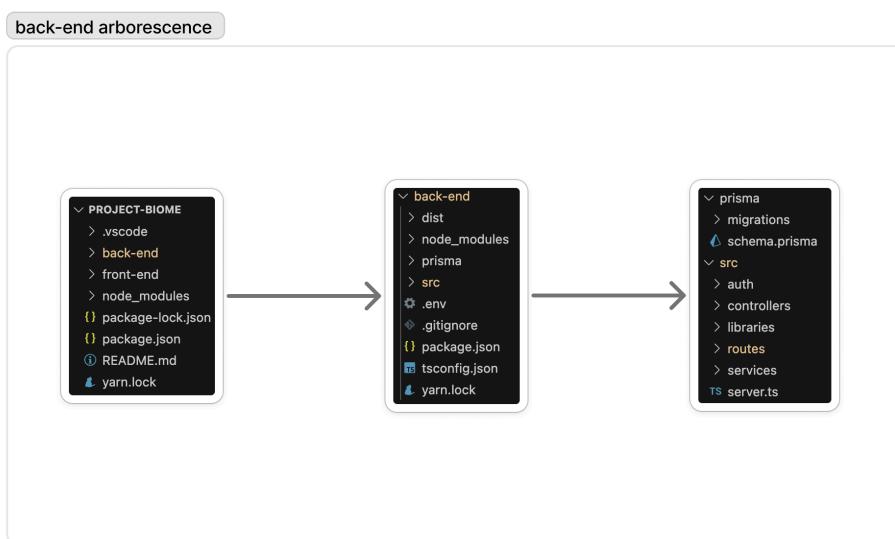
### **services/**

Contient les fonctions extérieures comme l'envoi de mail, pour la réinitialisation du mot de passe avec mailgun

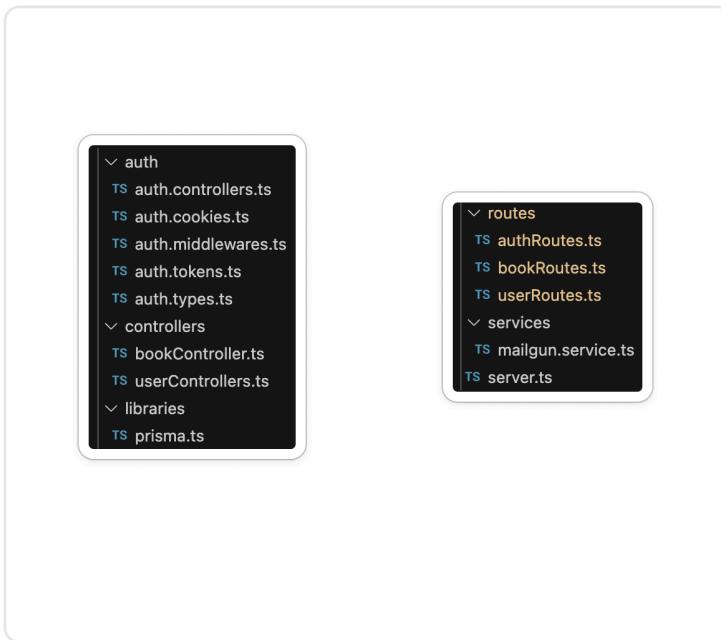
### **librairies/**

C'est ici qu'est stocké le fichier `prisma.ts` qui configure le client Prisma.

Finalement, le fichier `server.ts` qui est le point d'entrée du serveur Express.



#### back-end arborescence



### 3. Base de données

Dès le début de ce projet, j'ai choisi d'envisager les deux extrêmes vers lesquels ce projet pourrait s'orienter afin de choisir la base de données qui correspondra le mieux à mes besoins. J'ai donc pris en compte deux versions de Biome.

- Une version minimalisté basée sur le principe du journal de lecture, essentiellement dirigée sur la construction d'une bibliothèque visant à garder une trace de ses lectures.
- Une version qui tendrait plus vers le réseau social, avec le partage et l'échange entre les utilisateurs mis en avant.

Avoir une faible à moyenne quantité de données à gérer et à traiter n'a pas beaucoup de points de tensions auxquels faire attention. Cependant, si je continue à développer ce projet et que les gens sont intéressés, le volume peut augmenter rapidement. Je me suis donc renseignée sur les bases de données utilisées par des réseaux sociaux comme Instagram, Facebook, Reddit, etc. Cela m'a donné une idée globale de ce qui est populaire et stable en ce moment. Une fois cette première recherche faîte, je me suis penchée plus en détail sur les différents types de bases de données, leurs inconvénients et avantages afin de pouvoir faire un choix éclairé.

Puis j'ai récapitulé mes besoins dans le but de les comparer par rapport aux services que proposaient les différences BDD.

### **Une documentation et une communauté actives et à jour.**

Le temps pour mener à bien ce projet étant limité, je n'avais pas envie d'en perdre inutilement. Une communauté active signifie quantité de ressources et de solutions aux problèmes courants.

### **Un faible coût financier.**

Mes ressources et ma volonté d'investir financièrement dans un prototype étant limitées, il me fallait une base de données, idéalement gratuite ou payante seulement à partir d'un certain volume de données.

### **Une base stable.**

Tout comme le besoin de documentation, je souhaitais pouvoir me reposer et avoir confiance en la base que je choisirai. Ma recherche initiale sur ce qu'utilisent les grandes institutions m'a donné certaines pistes.

### **Capacité à évoluer**

Dans le cas où Biome serait amené à gérer beaucoup de données, il me fallait une base me permettant de m'adapter rapidement et simplement.

C'est ainsi que mon choix s'est porté sur **PostgreSQL**. PostgreSQL est un système de gestion de base de données relationnelles open-source largement utilisé, il existe donc beaucoup de documentations, d'informations et de réponses sur son fonctionnement. Il n'y a pas de frais pour son utilisation, même pour des logiciels commerciaux. Postgres existe depuis plus de 35 ans et est utilisé par de grandes compagnies comme Apple, Instagram, Reddit ou encore Spotify, ce qui atteste de sa stabilité et de sa capacité à gérer de très grands volumes de données. Mes recherches m'ont appris que la courbe d'apprentissage de PostgreSQL peut être abrupte lorsque l'on débute. Le fonctionnement n'est pas vraiment intuitif et diffère parfois beaucoup d'autres systèmes de gestion de base de données. Sur ce point, mon avantage résidait dans le fait que j'avais déjà utilisé des SGBD, MySQL et SQLite par exemple, mais jamais sur de longs projets. Je n'avais donc aucune habitude à déconstruire tout ayant une idée de ce qui m'attendait.

Un autre point négatif qui a souvent été mentionné est la lenteur de PostgreSQL par rapport à d'autres SGBD. Ce projet étant principalement créé pour être présenté dans la cadre du passage du diplôme RNCP, il m'a semblé assez improbable que je parvienne à avoir une quantité de données suffisante pour rencontrer ce problème. De plus, après avoir comparé avec plusieurs autres SGBD, les points positifs contrebalançaient, à mes yeux, grandement ce désavantage, quand bien même, j'en arriverai à ce scénario.

## 4. Endpoints API

Dans le cadre de ce projet, l'API backend joue un rôle élémentaire puisqu'elle assure la communication entre la base de données et les services tiers comme Google Books et Open Library pour le frontend. Les endpoints permettent de gérer l'authentification, les utilisateurs, les livres ainsi que les avis sur ceux-là. Afin d'établir un code clair, maintenable dans le temps et une séparation claire des responsabilités, l'API repose sur plusieurs éléments. Les routes ont été organisées en fonction de leur domaine fonctionnel.

### **/api/auth**

Regroupe toutes les routes ayant trait à l'authentification, telles que l'inscription, la connexion, le mot de passe...

### **/users**

Regroupe toutes les routes liées à l'utilisateur, comme la modification de son compte par exemple.

### **/books**

Regroupe la gestion des livres, des avis et la récupération des données, notamment celles liées aux API externes comme Google Books ou Open Library

L'API se base également sur les principes REST, les ressources sont au pluriel (ex : books, reviews...), les méthodes HTTP sont adaptées à ce qui est demandé (ex : POST pour la création, GET pour la lecture...). Les structures URL sont hiérarchiques (ex : [`/books/:bookId/reviews`](#) permet d'accéder directement aux avis d'un livre donné).

Ci-dessous, vous retrouverez plusieurs tableaux illustrant la manière dont les routes sont structurées.

### Utilisateur et authentification

Comme évoqué plus haut, l'authentification repose sur l'utilisation de JWT (JSON Web Tokens) stockés dans un cookie HttpOnly sécurisé. Pour rappel, lorsqu'un utilisateur se connecte à son compte, un access token est créé tandis qu'un refresh token est stocké sous forme de cookie permettant un rafraîchissement automatique sans avoir à se reconnecter.

Ainsi, les routes sensibles telles que l'ajout d'un livre, d'un avis, la modification ou la suppression de compte sont protégées par le middleware `verifyToken` qui s'assure de la validité du token avant d'autoriser l'accès à la ressource.

Fonction	Méthode	URL
Créer un nouvel utilisateur	POST	/api/auth/new_user
Connexion	POST	/api/auth/login_user
Déconnexion	POST	/api/auth/logout_user
Demande de réinitialisation de mot de passe	POST	/api/auth/forgotten_password
Réinitialisation de mot de passe	POST	/api/auth/reset_password
Modification profil utilisateur	PUT	/api/auth/modify_user
Infos utilisateur connecté	GET	/api/auth/current_user
Suppression du compte utilisateur	DELETE	/api/auth/delete_user

### Livres et avis

Fonction	Méthode	URL
Ajouter une livre et un avis simultanément	POST	/books/add_book_and_review
Récupérer tous les livres	GET	/books/books
Récupérer tous les avis	GET	/books/reviews
Récupérer les avis d'un livre	GET	/books/:bookId/reviews
Récupérer les livres de l'utilisateur	GET	/books/user_books
Supprimer un avis	DELETE	/books/reviews/:id

### API externes

Pour faciliter l'ajout de livres, j'ai décidé de simplifier le processus au maximum. Il suffit d'entrer l'ISBN du livre dans le champ dédié du formulaire et de mon côté, je fais appel à Google Books et Open Library pour récupérer le titre et l'auteur afin de compléter automatiquement ces champs. Plus tard, des informations comme la date de parution, la maison d'édition ou bien le nombre de pages pourront être facilement ajoutées à la carte du livre grâce à ces services.

J'ai d'abord choisi Open Library pour son utilisation sans frais et sa base de données conséquente, mais j'ai réalisé qu'elle était plus orientée publications académiques, scientifiques et d'archives. J'ai décidé d'ajouter Google Books pour combler les lacunes d'Open Library, c'est-à-dire, les livres de littérature contemporaine, fiction ou non fiction, parutions récentes, etc.

Source	URL	Description
Google Books	<a href="http://localhost:3000/books/search-google?q=ISBN:{ISBN} (proxy local)">http://localhost:3000/books/search-google?q=ISBN:{ISBN} (proxy local)</a>	Recherche via ISBN pour récupérer l'auteur et le titre
Open Library	<a href="https://openlibrary.org/api/books?bibkeys=ISBN:{ISBN}&amp;format=json&amp;jscmd=data">https://openlibrary.org/api/books?bibkeys=ISBN:{ISBN}&amp;format=json&amp;jscmd=data</a>	Recherche via ISBN pour récupérer l'auteur et le titre

### Service Email - Mailgun

Lorsqu'un utilisateur demande une réinitialisation de mot de passe, un e-mail est envoyé via le service Mailgun. Le lien de réinitialisation contient un token sécurisé d'une durée de vie d'une heure et utilisable une seule fois. Cette fonctionnalité renforce la sécurité du système de gestion des utilisateurs.

Fonction	Service	Description
Envoi d'email de réinitialisation	Mailgun	Envoi d'un email contenant un lien sécurisé pour réinitialiser le mot de passe

Ci-contre, un extrait du code illustrant l'utilisation de mailgun

```
export async function sendResetPasswordEmail(email: string, resetLink: string) {
  const messageData = {
    from: `No Reply <no-reply@${process.env.MAILGUN_DOMAIN}>`,
    to: email,
    subject: "Reset your password",
    text: `Hello!\n\nClick the link below to reset your password:\n\n${resetLink}\n\nIf you didn't request this, ignore this message.`,
  };

  return mg.messages.create(
    process.env.MAILGUN_DOMAIN as string,
    messageData
  );
}
```

# Modélisation des données et application

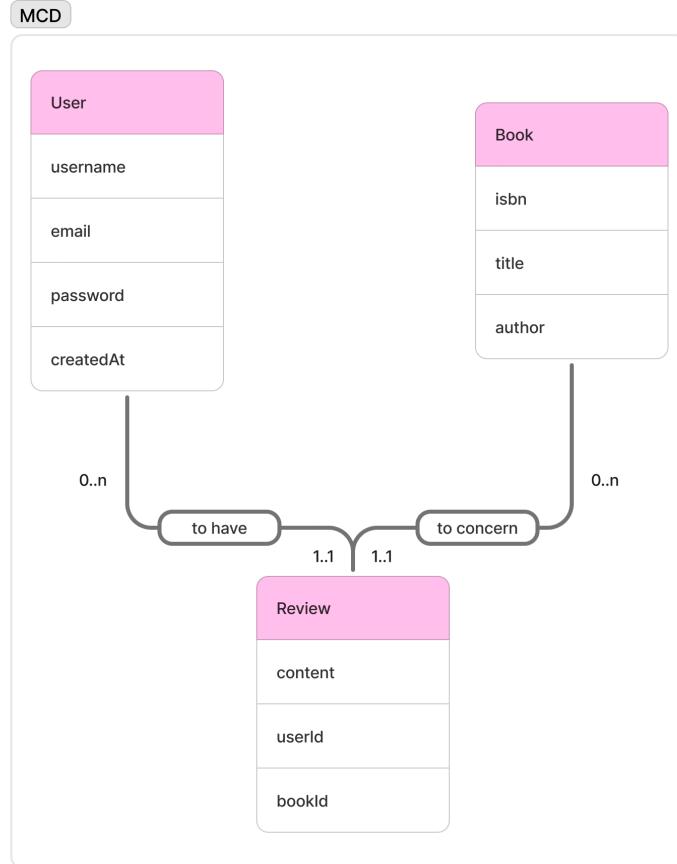
Une fois mes besoins clairement établis et ma base de données choisie, j'ai commencé à travailler sur la modélisation de mes données. J'avais initialement prévu d'utiliser un diagramme UML (Unified Modeling Language) afin de représenter les entités et leurs relations, car cette méthode est largement utilisée et réputée adaptée pour la modélisation orientée objet. Elle permet une vision complète du système avec différents types de diagrammes (cas d'utilisation, classes, séquences, etc.).

Cependant, j'ai rapidement été confrontée à la complexité de cette méthode. Cette méthode nécessite une bonne maîtrise de ses conventions et une compréhension approfondie de la modélisation orientée objet. Je me suis donc renseignée, ai lu de la documentation, suivi des cours et tutoriels, mais je me suis rendue compte d'un problème épique : le manque de convention universelle. Chaque exemple que j'ai pu trouver a été différent des précédents et des suivants. J'ai fini par demander l'avis de trois de mes collègues développeurs, profitant de mon alternance, et le problème s'est définitivement cristallisé puisqu'aucun consensus n'a pu être trouvé au cours de cette discussion. J'ai commencé à me sentir considérablement confuse et frustrée, il paraissait évident que m'obstiner dans cette voie me préparait à une perte de temps non négligeable.

Je me suis donc tournée vers la méthode **Merise**, plus axée sur la modélisation des données et particulièrement adaptée aux bases de données relationnelles. J'ai ainsi pu structurer mon projet grâce à trois diagrammes :

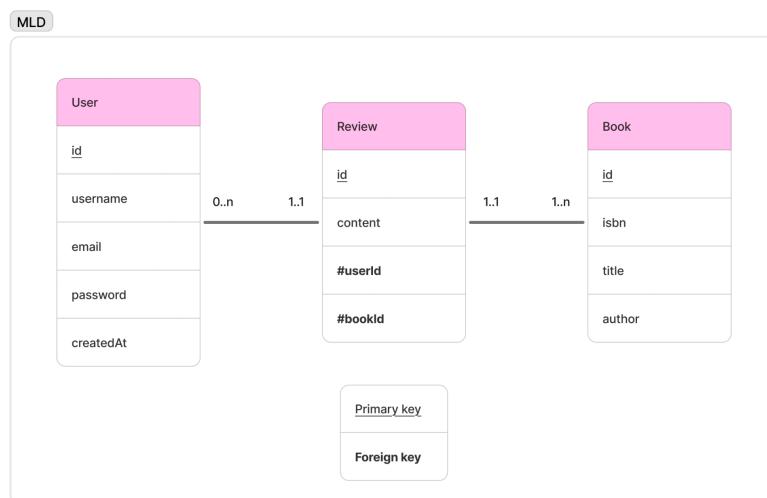
## 1. MCD (modèle conceptuel de données)

Ce diagramme me permet d'identifier les entités, leurs attributs et les relations qu'elles entretiennent entre elles.



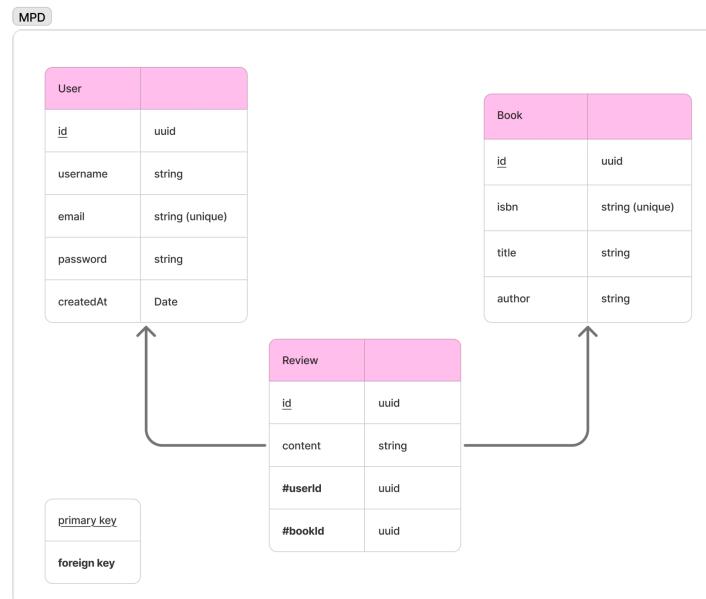
## 2. MLD (modèle logique de données)

Il permet de traduire les éléments du diagramme MCD sous une forme compatible avec un SGBD relationnel.



### 3. MPD (modèle physique de données)

Ce diagramme représente la mise en œuvre concrète du MLD dans la base de données PostgreSQL.



Ce changement de méthode m'a été bénéfique pour plusieurs raisons : j'ai eu une introduction au fonctionnement du diagramme UML ce qui m'a permis d'avoir une vision plus globale lorsque je me suis penchée sur la méthode Merise. Ces deux méthodes n'étant pas réellement en opposition, mais plutôt complémentaires, il a été très intéressant de comprendre leurs différences. J'ai pu prendre du recul et repartir sur de bonnes bases en abandonnant un sujet qui me créait beaucoup de frustration et bloquait l'avancée du projet. J'ai abordé la méthode Merise sans appréhension, de façon posée et volontaire, ce qui m'a permis d'apprendre et de l'utiliser dans de bonnes conditions. En comparaison avec le diagramme UML, la méthode Merise a été beaucoup plus intuitive, claire et facile à appliquer dans mon cas. Elle m'a également permis d'avoir une vue d'ensemble progressive et cohérente du système d'information, facilitant ainsi la mise en œuvre de ma base de données.

### 4. Prisma ORM

#### A. Introduction

Un ORM (object relational-mapping) est un outil de programmation utilisé pour faire correspondre les structures de données orientées objets avec une base de

données relationnelles. Il permet de manipuler les données comme si elles étaient des objets au sein du code, sans avoir à écrire de requêtes SQL complexes. Les opérations CRUD (Create, Read, Update, Delete) sont plus simples à gérer, car elles sont transformées en simples appels de méthodes. Les ORM ajoutent par ailleurs une couche de sécurité puisque la plupart intègrent une protection contre les injections SQL.

Pour ce projet, j'ai choisi Prisma. J'ai déjà évoqué plus haut certains de ces aspects, je ne reviendrai donc pas sur ceux-là, mais plutôt mettre en lumière quelques autres. Prisma est simple à prendre en main et simple d'utilisation, que l'on travaille sur une base de données déjà existante ou que l'on souhaite en créer une nouvelle. Particularité qui m'a bien été utile, car j'ai dû réinstaller ma base de données suite à la réinitialisation de mon ordinateur. Cette manipulation s'est faite sans accroc grâce à la commande `db pull` et j'ai rapidement pu me remettre à travailler. Son approche est centrée sur le `schema.prisma` qui donne une représentation claire de la structure de base. Les modèles sont faciles à faire évoluer, il suffit de changer le schéma puis de générer le typage avec `npx prisma generate` et de créer la nouvelle migration avec `npx prisma migrate dev`.

La modélisation Prisma me permet de retrouver les mêmes entités que lors de la modélisation Merise, mais de manière exploitable directement en code. Chaque modèle correspond à une table dans PostgreSQL, avec des relations correctement définies.

## B. Schéma

Voici le schéma de ma base de données PostgreSQL établi grâce à la modélisation Merise évoquée plus tôt.

```
model User {
    id      String  @id @default(uuid())
    username String
    email   String  @unique
    password String
    createdAt DateTime @default(now())
    Review  Review[]
}

model Book {
    id      String  @id @default(uuid())
    isbn   String  @unique
    title  String
    author String
    Review  Review[]
}

model Review {
    id      String  @id @default(uuid())
    content String
    createdAt DateTime @default(now())
    author   User    @relation(fields: [authorId], references: [id])
    authorId String
    book    Book    @relation(fields: [bookId], references: [id])
    bookId  String
}
```

## User

Chaque utilisateur a un ID unique généré automatiquement (uuid()). L'email est unique pour éviter les doublons et n'avoir qu'un compte par adresse mail. Le champ `Review[]` représente une relation 0-n : un utilisateur peut avoir zéro ou plusieurs avis (reviews).

## Book

Chaque livre a un ID unique, et son ISBN l'est également. Le tableau `Review[]` indique qu'un livre peut être lié à plusieurs critiques.

## Review

Chaque critique est liée à un utilisateur (`author`) et un livre (`book`), grâce à des relations définies avec `@relation`. Les champs `authorId` (l'utilisateur qui a posté l'avis) et `bookId` (le livre dont il traite) agissent comme des clés étrangères.

Ce modèle illustre exactement ce qui avait été modélisé grâce au MCD : une table de jointure `Review` qui relie `User` et `Book`.

## C. Exemple de requête

Ci-dessous, nous avons la fonction `createBookAndReview`, nous la prendrons comme exemple, car elle illustre plusieurs aspects de l'application :

- Authentification,
- Communication avec la base de données PostgreSQL via Prisma
- Création de relation entre entités
- Logique métier d'évitement des doublons de livres.

```

export async function createBookAndReview(req: ExpressRequest, res: Response) {
  if (!req.user) {
    return res.status(401).json({ error: "user is not authenticated" });
  }
  const { isbn, title, author, content } = req.body;

  try {
    let book = await prisma.book.findUnique({
      where: { isbn },
    });
    if (!book) {
      book = await prisma.book.create({
        data: {
          isbn,
          title,
          author,
        },
      });
    }
    const review = await prisma.review.create({
      data: {
        content,
        authorId: req.user.id,
        bookId: book.id,
      },
    });
    res.status(201).json({ book, review });
    console.log("New book and review created:", { book, review });
  } catch (error) {
    console.error(
      "Error during the creation of a new book and review",
      error
    );
    return;
  }
}

```

Cette fonction est un endpoint clé de l'application. Elle permet à un utilisateur authentifié d'ajouter un avis à un livre en fournissant son ISBN, son titre, son auteur, ainsi que le contenu de cet avis. Elle est exposée via la route POST `/books/add_book_and_review`.

Cette fonction permet l'authentification de l'utilisateur grâce au middleware `verifyToken`, qui injecte les informations de l'utilisateur dans `req.user`.

```
router.post("/add_book_and_review", verifyToken, createBookAndReview);
```

Ici, je m'assure que seul un utilisateur connecté peut ajouter un livre et un avis, si ce n'est pas le cas, le message d'erreur apparaîtra dans l'onglet network puis réponse de la console DevTools. Côté front, si l'utilisateur n'est pas connecté, il sera redirigé vers la page `<NotConnectedPage>`.

```

if (!req.user) {
  return res.status(401).json({ error: "user is not authenticated" });
}

```

You must be logged in to access this page.

[Login or Register](#)

Puis, je vérifie si le livre existe dans la base de données. Cela évite la création de doublons, si un livre avec le même ISBN est déjà présent, il est tout simplement réutilisé.

```
try {
  let book = await prisma.book.findUnique({
    where: { isbn },
  });
  if (!book) {
    book = await prisma.book.create({
      data: {
        isbn,
        title,
        author,
      },
    });
  }
}
```

Ensuite, une relation est créée et l'avis est connecté à l'utilisateur (`authorId`) qui l'a écrite et au livre (`bookId`) dont il parle.

```
const review = await prisma.review.create({
  data: {
    content,
    authorId: req.user.id,
    bookId: book.id,
  },
});
```

## Réalisation

### 1. Authentification

L'authentification est l'étape permettant à un utilisateur de prouver son identité, souvent avec un mot de passe, elle vient après l'identification qui établit l'identité de l'utilisateur grâce à son identifiant (ex : adresse mail). C'est une étape clé d'une application à partir du moment où l'utilisateur a la possibilité de créer un compte et de s'y connecter. C'est l'association de ces deux éléments qui permet à un utilisateur de se connecter de manière sécurisée.

Pour ce projet, mon authentification repose sur la logique d'inscription et de connexion, l'utilisation de JWT (JSON Web Tokens), de cookies sécurisés (HttpOnly), un système de refresh automatique, des routes protégées ainsi qu'un système de réinitialisation de mot de passe.

## A. Inscription

Front

L'utilisateur remplit les trois champs du formulaire avec un nom d'utilisateur, un email et un mot de passe. Grâce à `react-hook-form`, je mets une condition de longueur au mot de passe, il doit être d'un minimum de huit caractères. Si cette condition n'est pas respectée, un message d'erreur apparaîtra.

```
<Input
  id='password'
  type='password'
  {...register("password", {
    required: "Incorrect password",
    minLength: {
      value: 8,
      message:
        "Password must be at least 8 characters",
    },
  })}
/>
{errors.password && <p>{errors.password.message}</p>}
```

Une fois les diverses conditions remplies, les données sont envoyées à la route backend `/new_user` via Axios.

```
const onRegisterSubmit: SubmitHandler<UserProps> = async (data) => {
  const newUserData = {
    username: data.username,
    email: data.email,
    password: data.password,
  };

  try {
    await userApi.post("/new_user", newUserData);
    // ...
  } catch (error) {
    // ...
  }
}
```

Si l'inscription réussit, l'utilisateur est automatiquement connecté via la fonction `login` utilisant `useAuth` qui déclenche la route `/login_user`. L'utilisateur est finalement redirigé sur la page d'accueil (homepage) grâce au hook `useNavigate`.

```
await login(data.email, data.password);
navigate("/homepage", { replace: true });
```

En cas d'erreurs, des messages spécifiques s'affichent pour en informer l'utilisateur. Erreur 400 pour des données incorrectes et 409 si l'email est déjà utilisé. Un message d'erreur générique est affiché pour les erreurs réseaux.

```
if (status === 400) {
  toast.error("Invalid user data");
} else if (status === 409) {
  toast.error("User already exists");
```

Back

L'inscription repose donc sur un nom d'utilisateur (username), une adresse mail et un mot de passe via le formulaire d'inscription. La route liée est `/new_user`. Le serveur va vérifier si l'adresse mail n'est pas déjà utilisée, si elle l'est, un message d'erreur sera envoyé.

```
export async function createUser(req: Request, res: Response) {
  const { username, email, password } = req.body;

  try {
    let newUser = await prisma.user.findUnique({
      where: { email },
    });
    if (newUser)
      return res
        .status(409)
        .json({ error: "This email is already taken" });
  }
```

Si l'adresse mail est disponible, le mot de passe est ensuite haché avec **bcrypt** sur une base de dix itérations. Un sel aléatoire sera ajouté à chaque mot de passe, garantissant que deux mots de passes identiques ne le soit visuellement pas après le hachage. L'utilisateur et toutes ses informations sont ensuite insérés dans la table `User` via Prisma, au sein de la base de données.

```
const hashedPassword = await bcrypt.hash(password, 10);
const user = await prisma.user.create({
  data: {
    username,
    email,
    password: hashedPassword,
  },
});
```

L'objet user est ensuite nettoyé, le mot de passe est enlevé afin d'éviter de l'envoyer par erreur dans la réponse ou de permettre sa manipulation. `...userWithoutPassword` contient donc toutes les autres informations de son parent sauf le mot de passe.

```
const { password: _password, ...userWithoutPassword } = user;
```

Ensuite vient la génération des JSON web tokens, j'utilise deux fonctions me permettant de générer l'access token et le refresh token. Leurs clés secrètes sont stockées dans le `.env` et sont une longue suite de lettres et de chiffres aléatoires créées avec crypto. L'access token a une durée de vie de quinze minutes et le refresh token de sept jours.

```
const refreshTokenSecret = process.env.REFRESH_TOKEN_SECRET;
if (!refreshTokenSecret) throw new Error("Access token secret undefined");
export const generateRefreshToken = (user: JwtPayload): string => {
  return sign({ id: user.id }, refreshTokenSecret, { expiresIn: "7d" });
};

const accessToken = generateAccessToken(userWithoutPassword);
const refreshToken = generateRefreshToken(userWithoutPassword);
```

Puis, le refresh token est envoyé dans un cookie `HttpOnly`, ainsi, il est inaccessible en JavaScript, ce qui le protège des attaques XSS. Finalement, l'access token est envoyé dans la réponse JSON. Le frontend pourra ainsi le stocker jusqu'à son expiration.

```
const refreshTokenOptions = {
  httpOnly: true,
  secure: false, //false for dev, true for prod
  sameSite: "lax" as const, // strict for prod
  maxAge: 1000 * 60 * 60 * 24 * 7, // 7 days
};
export function setRefreshTokenCookie(res: Response, token: string) {
  res.cookie("refreshToken", token, refreshTokenOptions);

setRefreshTokenCookie(res, refreshToken);
res.status(201).json({
  user: userWithoutPassword,
  accessToken,
});
```

## B. Connexion

### Front

Le formulaire de connexion comporte deux champs, un pour l'adresse mail et l'autre pour le mot de passe. En cas d'oubli de mot de passe, l'utilisateur pourra cliquer sur "Forgot your password?". Le formulaire fait appel à la fonction `login()` du contexte React `AuthContext`. `login()` envoie une requête POST via Axios au backend avec l'email et le mot de passe saisis par l'utilisateur.

```
const success = await login(data.email, data.password);

const login = async (email: string, password: string) => {
  setLoading(true);
  try {
    const response = await userApi.post("/login_user", {
      email,
      password,
    });
  }
}

export const userApi = axios.create({
  baseURL: "http://localhost:3000/users",
  withCredentials: true,
});
```

Si la requête est validée par le backend, `AuthContext` est mis à jour avec les informations de l'utilisateur. Le statut connecté de l'utilisateur est maintenant partagé à l'application via ce contexte, permettant l'accès aux routes protégées telles que l'ajout de livre ou le profil utilisateur. Sinon un message d'erreur est renvoyé à l'utilisateur. L'access token lui est stocké dans le local storage. Avec Axios, j'ajoute le token à chaque requête HTTP sortante.

```
setUser(response.data.user);
localStorage.setItem("accessToken", response.data.accessToken);
```

```

const addAuthInterceptor = (apiInstance: AxiosInstance) => {
  apiInstance.interceptors.request.use((config) => {
    const token = localStorage.getItem("accessToken");
    if (token) {
      config.headers.Authorization = `Bearer ${token}`;
    }
    return config;
  });
};

addAuthInterceptor(bookApi);
addAuthInterceptor(authApi);
addAuthInterceptor(userApi);

if (!success) {
  toast.error("Login failed, please check your credentials", {
    duration: 4000,
  });
  return;
}

```

Une fois la connexion validée et établie, un toast apparaît pour indiquer que l'utilisateur est bien connecté puis il est automatiquement dirigé vers la page d'accueil (homepage).

```

navigate("/homepage", { replace: true });
reset();
toast.success("Logged in!");

```

Back

La connexion s'établit par la vérification de l'adresse email de l'utilisateur et du mot de passe. La route liée est `/login_user`. Le backend cherche un utilisateur correspondant à l'email, s'il n'est pas trouvé dans la base de donnée, un message d'erreur est envoyé.

```

const { email, password } = req.body;

try {
  const user = await prisma.user.findUnique({
    where: { email: email },
  });
  if (!user) return res.status(401).json({ error: "Unknown user" });
}

```

Si l'adresse mail est retrouvée, on compare le mot de passe saisi avec celui présent dans la base de données. Le mot de passe va passer par les mêmes étapes (nombre d'itérations, sel...) que le mot de passe stocké. Si la version hachée de ces deux mots de passe est identique, cela veut dire que le mot de passe saisi est bien le bon pour cet utilisateur. S'il ne correspond pas, un message d'erreur est retourné.

```
const validPassword = await bcrypt.compare(password, user.password);
if (!validPassword)
    return res.status(401).json({ error: "Invalid credentials" });
```

Si tout est validé, on repasse par l'étape de nettoyage de l'objet user et la génération des access et refresh tokens déjà évoqués dans la phase d'inscription. La réponse est ensuite transmise au client.

```
const { password: _password, ...userWithoutPassword } = user;
const accessToken = generateAccessToken(userWithoutPassword);
const refreshToken = generateRefreshToken(userWithoutPassword);

setRefreshTokenCookie(res, refreshToken);
res.status(200).json({ user: userWithoutPassword, accessToken });
```

## C. Rafraîchissement des tokens

Front

Lorsque le token d'accès expire, le frontend va détecter une erreur 401 Unauthorized. Cela va déclencher un appel à la route `/refresh` de la part du `AuthProvider` via le hook React `useEffect`. Cette route va générer un nouvel access token à partir du refresh token présent dans les cookies. Si cette requête réussie, un appel à `/current_user` est effectué afin de récupérer les infos utilisateurs et mettre le `AuthContext` à jour.

```
useEffect(() => {
    const fetchUserWithRefresh = async () => {
        try {
            await authApi.post("/refresh");
            const response = await userApi("/current_user");

            setUser(response.data);
        } catch (err) {
            console.error("Failed to refresh token or fetch user:", err);
            setUser(null);
        } finally {
            setLoading(false);
        }
    };

    fetchUserWithRefresh();
}, [ ]);
```

Back

Le refresh token est stocké au sein d'un cookie sécurisé HttpOnly, envoyé automatiquement par le navigateur. On vérifie que celui-ci est bien valide (signature, expiration...) et si c'est le cas, on retrouve l'utilisateur correspondant en base de données.

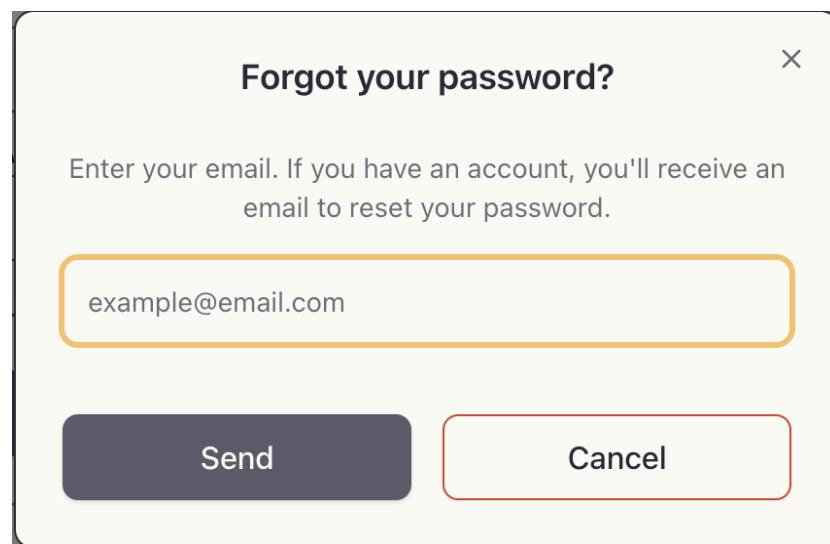
```
const token = req.cookies.refreshToken;  
  
const payload = verify(token, refreshToken) as { id: string };  
const user = await prisma.user.findUnique({  
    where: {  
        id: payload.id,  
    },  
});
```

Finalement, on génère un nouvel access token qui est ensuite envoyé dans un cookie. Ainsi, l'utilisateur n'a pas à se reconnecter à chaque expiration d'access token, garantissant une expérience utilisateur fluidifiée.

```
const newAccessToken = generateAccessToken(userWithoutPassword);  
  
return res.status(200).json({  
    accessToken: newAccessToken,  
    user: userWithoutPassword,  
});
```

## D. Mot de passe oublié

Il est fréquent qu'un utilisateur oublie son mot de passe, c'est pourquoi j'ai mis en place un système de réinitialisation de mot de passe. L'utilisateur entre son email, si celui-ci est présent dans la base de données de Biome, un mail avec un lien unique lui est envoyé et lui permet de mettre à jour son mot de passe.



## Front

L'utilisateur saisit son mot de passe, en cliquant sur "send" une requête est envoyée à la route `/forgotten_password`. Un toast confirme que le mail a bien été envoyé même si l'utilisateur n'existe pas afin de ne pas révéler quels utilisateurs sont inscrits.

```
await userApi.post("/forgotten_password", { email });
toast.success("Mail sent, check your inbox!");
```

## Back

L'utilisateur est recherché en base de données grâce à son adresse email, même s'il n'existe pas une réponse 200 ok est envoyé pour éviter les fuites d'informations sur les utilisateurs.

```
const user = await prisma.user.findUnique({ where: { email } });
if (!user) {
  return res.status(200).json({
    message: "A reset link has been sent.",
  });
}
```

Un token de réinitialisation avec une durée de vie d'une heure est créé.

```
const resetToken = jwt.sign(
  { id: user.id, email: user.email },
  process.env.JWT_RESET_SECRET as string,
  { expiresIn: "1h" }
);
```

Un lien sécurisé contenant ce token sécurisé est construit puis envoyé avec Mailgun. L'email contient un lien vers une page du frontend où l'utilisateur pourra saisir un nouveau mot de passe.

```
const resetLink = `${process.env.FRONTEND_URL}/reset-password?token=${resetToken}`;
await sendResetPasswordEmail(email, resetLink);
```

```

export async function sendResetPasswordEmail(email: string, resetLink: string) {
  const messageData = {
    from: `No Reply <no-reply@${process.env.MAILGUN_DOMAIN}>`,
    to: email,
    subject: "Reset your password",
    text: `Hello!\n\nClick the link below to reset your password:\n\n${resetLink}\n\nIf you didn't request this, ignore this message.`,
  };

  return mg.messages.create(
    process.env.MAILGUN_DOMAIN as string,
    messageData
  );
}

Hello!

Click the link below to reset your password:

http://localhost:5173/reset-password?token=eyJhbGciOiJIUzI1NiIsInR5cCl6lkpXVCJ9eyJpZC16IjYwZTExY2IxLWYxN2MlNGQwOS1iY2VhLWMyZjUxZmY4YWlzZSlslmVtYWlsjoicm9tYW5ILmJvaXJYXVAZ21haWwuY29tliwiaWF0ljoxNzUxNDk3NzkyLCJleHAiOjE3NTExMDEzOTJ9.jlfJinWRrDV0HTADv0qveGqL2R6ptfeel3wyD3sl\_A

```

## E. Réinitialisation du mot de passe

Front

Lorsque l'utilisateur clique sur le lien de réinitialisation envoyé à son adresse mail, il est dirigé vers une page lui permettant de saisir le nouveau mot de passe. Comme pour le formulaire d'inscription, ce mot de passe doit comporter au moins huit caractères et un message d'erreur s'affiche si cette condition n'est pas remplie. Le formulaire est ensuite envoyé et génère une requête POST au backend via la route `/reset_password`. Cette requête contient le token unique du lien ainsi que le nouveau mot de passe (`newPassword`).

```

await userApi.post("/reset_password", {
  token,
  newPassword: password,
});

```

Une fois la réponse positive du back reçue, un toast de succès apparaît indiquant la complétion de la réinitialisation du mot de passe. L'utilisateur pourra ensuite se connecter avec son nouveau mot de passe et accéder à son compte.

## Back

Le token est vérifié avec jwt.verify() afin d'être certain qu'il est valide.

```
const decoded = jwt.verify(  
  token,  
  process.env.JWT_RESET_SECRET as string  
) as {  
  id: string;  
};
```

Le nouveau mot de passe est ensuite haché avec bcrypt avant d'être inséré dans la base de données, la mettant ainsi à jour.

```
const hashedPassword = await bcrypt.hash(newPassword, 10);  
  
await prisma.user.update({  
  where: { id: decoded.id },  
  data: { password: hashedPassword },  
});
```

# Objectifs de qualité

## 1. Commentaires

Tout au long du projet, j'ai pris le soin de laisser des commentaires expliquant certaines fonctions, mais aussi des références à de la documentation afin de faciliter mon travail. Ces commentaires m'ont aidé à me repérer plus facilement, comprendre dans quel état j'avais laissé les choses me permettant ainsi de me replonger dans le code sans difficultés même après plusieurs semaines passées sur d'autres aspects. Ces commentaires ont évolué au fur et à mesure de l'avancée du projet, reflétant le développement de ma compréhension des techniques et des méthodes mises en place.

Voici quelques exemples :

```
// only the user can delete its own review
const canDelete = user && user.id === author.id;

// check if the signature has not been modified
const decoded = verify(token, accessTokenSecret) as { id: string };

// extract authorization header from HTTP request
const authHeader = req.headers.authorization;
const cookieToken = req.cookies?.accessToken;

// extract only the token from the header
const token = authHeader?.split(" ")[1] || cookieToken;
```

## 2. Documentation

J'ai rapidement mis en place un document dans lequel j'ai répertorié une grande partie des problèmes sur lesquels je suis tombée au cours du projet. Cela m'a beaucoup servi, notamment au début, lorsque je prenais encore mes marques. Pour chaque problème rencontré, j'ai pris une capture d'écran du problème et/ou du message d'erreur, si besoin une phrase détaillant l'impact. Une fois la solution trouvée, je l'ajoute au document en expliquant le raisonnement.

Cette technique m'a permis de gagner en temps précieux en me référant à cette check-list d'erreurs courantes et des pistes concrètes pour les corriger. Elle m'a également permis de mieux comprendre les mécanismes liés à ces erreurs, développer mes réflexes de débogage, améliorer la mise en place de mes messages d'erreurs ainsi que d'anticiper certains pièges récurrents.

Problème :

```
Type 'UseFormReturn<FieldValues, any, undefined>' has no signatures for which the
type argument list is applicable
```

Contexte :

Ce message peut apparaître lorsqu'on essaye d'associer un type générique à un hook (comme `useForm`) en le plaçant de manière incorrecte dans le code :

Solution :

Le type doit être placé avant les parenthèses du hook

### mauvaise syntaxe

```
Complexity is 8 It's time to do something...
export function BookForm() { }
| const form = useForm(<BookFormProps>);
```

Le type est placé après les parenthèses du hook, ce qui est incorrect.

### bonne syntaxe

```
Complexity is 8 It's time to do something...
export function BookForm() { }
| const form = useForm<BookFormProps>();
```

Le type `BookFormProps` est passé à l'intérieur des chevrons avant les parenthèses d'appel du hook.

## 3. Standards et bonnes pratiques

Dans le cadre de ce projet, j'ai mis en place un certain nombre de standards et de bonnes pratiques de développement afin de maintenir la cohérence, la lisibilité et la

maintenabilité du code. J'ai ainsi pu travailler sur une base saine, facilitant les additions et évolutions ainsi que la détection des erreurs.

Pour garantir un code propre avec une structure homogène sur l'ensemble du projet, j'ai utilisé trois extensions en particulier :

### **ESLint**

Permet de détecter les erreurs de syntaxes, les mauvaises pratiques et incohérences dans mes fichiers JavaScript et TypeScript.

### **Prettier**

Formate le code pour standardiser ses indentations, ses sauts de lignes, espaces...

### **Code Spell Checker**

Vérifie l'orthographe des mots en anglais sur l'ensemble du code et propose des alternatives correctes.

Pour les noms de composants, je me suis basée sur deux méthodes largement répandues :

#### **camelCase**

Je l'ai utilisé pour toutes les variables, fonctions et noms de constantes TypeScript suivant les conventions de nommage en vigueur au sein de l'écosystème React.

#### **PascalCase**

J'ai utilisé le PascalCase pour les noms des composants React, tels que (BookForm, LoginPage...) toujours dans le but de respecter les conventions React.

Dans le souci de garantir la maintenabilité, l'accessibilité, la cohérence visuelle de mon projet, j'ai structuré mon interface à l'aide de variables CSS pour la gestion des couleurs, et de la bibliothèque shadcn/ui pour les composants UI réutilisables et accessibles.

#### **CSS variables**

Utilisées principalement pour définir les couleurs, elles me permettent d'appliquer un changement global en les modifiant plutôt qu'en changeant chaque fichier dans lequel elles sont appelées.

#### **shadcn/ui**

Comme évoqué plus tôt, shadcn/ui est une bibliothèque de composants UI utilisée pour garantir une interface cohérente, esthétique et accessible. Elle repose sur Radix UI et Tailwind CSS et me permet d'avoir des composants accessibles et modernes sur l'ensemble de mon projet.

# Difficultés techniques et humaines

## 1. Authentification des utilisateurs

L'authentification est un point majeur de n'importe quel projet permettant la création de compte. C'est un sujet sensible puisqu'on manipule des données utilisateurs. Il est donc impératif de les stocker, de les utiliser de manière sécurisée et de respecter les règles sur la protection des données comme celles de la RGPD.

Ce sujet m'a beaucoup mise en difficulté technique pour plusieurs raisons. La plus mineure a été de comprendre la différence entre l'authentification et l'autorisation. Ces deux mots sont souvent utilisés de manière interchangeable, ce qui a rendu la compréhension plus complexe. À force de persévérance, j'ai fini par intégrer que l'authentification est le processus qui permet de prouver qu'un utilisateur est bien celui qu'il prétend grâce à ses informations de connexion, par exemple. L'autorisation, quant à elle, est le principe d'autoriser un utilisateur à avoir accès à certaines ressources.

Ensuite, il m'a fallu mettre en place les JSON web tokens. J'ai assez rapidement compris la différence entre l'access et le refresh token et leur mise en place au sein du back m'a paru plutôt facile de premier abord. Mais c'est au moment de lier cette logique au frontend et de la tester que je me suis rendue compte que le sujet était plus complexe que ce que j'imaginais. D'autant plus que, pour le frontend, c'était la première fois que je devais mettre en place un contexte React. J'ai donc passé plusieurs semaines à lire de la documentation, à regarder des tutoriels et à tester par moi-même pour résoudre ce problème. Cette étape a été une grosse source de frustration et m'a beaucoup ralenti, car, à force, j'ai fini par y aller à reculons. La solution a tout simplement été de faire une pause, de travailler sur des éléments de l'interface utilisateur ou de travailler, en parallèle, sur mon dossier projet afin de changer d'atmosphère et me reposer sur un travail avec lequel j'étais beaucoup plus à l'aise.

## 2. Deuxième version

Ma deuxième grosse difficulté sur ce projet a tout simplement été la motivation. Je me suis rapidement lancée dans le projet, j'avais beaucoup d'idées et peu, voire pas d'organisation. Pendant les quatre premiers mois, j'ai beaucoup travaillé sans me mettre de limite, j'ai parfois fini très tard le soir, j'ai refait plusieurs versions de certains éléments et une légère fatigue mentale a commencé à s'installer, mais l'excitation de créer quelque chose était encore forte. À l'image d'un marathonien qui s'élance à toute vitesse en début de course, je commençais à être essoufflée au tiers du parcours. Puis, plusieurs événements dans ma vie personnelle sont arrivés. Pour diverses raisons, ils

ont grandement affecté ma motivation, mon environnement de travail n'était plus optimal et la situation évoluant, j'ai dû changer l'ordre de mes priorités. Le projet est donc passé au second rang jusqu'à ce que je sois dans de meilleures conditions. Une fois ma situation stabilisée, j'ai tenté de reprendre le projet, mais je n'arrivais pas à avancer au milieu des divers sujets entamés puis laissés pour plus tard. Cette manière de travailler me convient en début de projet et lorsque j'y travaille de manière régulière, mais après une longue période, il est très complexe de s'y replonger.

Après une réflexion posée et diverses discussions avec des collègues pendant mon alternance ainsi qu'avec des proches, j'ai pris la décision de repartir de zéro au niveau du code. Je garde ainsi tout le travail de recherche et de design, mais je pars sur une base claire et propre. J'ai ainsi pu mettre en place une meilleure organisation, diviser d'une manière plus optimisée les tâches me rendant ainsi plus performante. Cependant, j'ai quand même subi de nombreuses phases de doutes et de stress, puisqu'il me fallait rattraper plusieurs mois de retard, tout en maintenant une performance à la hauteur des attentes dans l'entreprise où j'effectuais mon alternance. Je voulais mettre le maximum de chances de mon côté pour recevoir une proposition de CDI, je ne voulais donc pas être distraite par manque de sommeil ou en ayant l'esprit trop préoccupé par tout ce qu'il me restait à faire.

Je n'ai pas trouvé de solution à ce problème puisqu'il m'a suivi jusqu'à la rédaction de ce rapport, mais je pense que le recul que j'ai pu prendre me permet de ressentir une certaine satisfaction malgré ces difficultés.

## Veille

En parallèle de la réalisation de ce projet, j'ai également tenu à me tenir au courant des événements et des actualités du monde de la tech. Mes sources d'informations ont été diverses et voici quelques exemples :

➤ YouTube

### **ThePrimeTime**

Pour ses discussions sur des sujets d'actualités et en particulier celles autour de l'intelligence artificielle, ses implications sur la société et son impact sur le métier de développeur, notamment pour les profils juniors. Ces vidéos m'ont permis de prendre du recul, mais aussi de comprendre l'impact d'autres secteurs sur notre métier.

### **Fireship**

Pour ses vidéos condensées sur des sujets tech, ses présentations rapides et efficaces d'outils ou de frameworks, mais aussi ses vidéos explicatives d'une technologie en particulier qui permet d'avoir une compréhension générale du sujet en quelque minutes. Elles m'ont permis de rester à jour sans avoir à y dédier beaucoup de temps.

### **WebDevSimplify et Cosden**

Pour leurs tutoriels qui m'ont beaucoup aidé à concrétiser ce que j'avais pu lire dans la documentation. Leurs vidéos m'ont été très utiles pour la mise en œuvre de manière concrète de certains outils pour ce projet.

➤ Podcast

### **Syntax FM**

Un podcast animé par deux développeurs expérimentés, abordant des sujets variés. Les échanges sont riches en conseils et recommandations, le ton est bienveillant et rend les discussions plutôt accessibles même sans connaissances au préalable.

### **The changelog**

Podcast orienté sur les coulisses du développement de projets open source et de nouvelles technologies. J'ai pu ainsi élargir mes connaissances et ma vision des projets communautaires ainsi que leurs enjeux actuels.

C'est avec l'appui de cette veille et aux différents formats, j'ai pu maintenir ma motivation et ma curiosité pour ce projet. Cela m'a également aidé à prendre du recul et à relativiser sur certains problèmes que je rencontrais. Le partage d'expérience des ces personnes beaucoup plus expérimentées que moi, et qui pourtant rencontrent, elles aussi, des difficultés lors de l'apprentissage d'un nouveau sujet, m'a beaucoup rassuré et donné du courage.

## Futures réalisations et évolutions possibles

### 1. Futur proche

Il y a un certain nombre de choses que je n'ai pas eu le temps de mettre en place sur ce projet par manque de temps. Il y a certains éléments qui devront être traités en priorité avant le développement de nouvelles fonctionnalités.

#### A. Tests

J'ai uniquement effectué des tests manuels au cours de ce projet, ce qui m'a beaucoup aidé à repérer les problèmes puisqu'à chaque nouvelle fonctionnalité, je testais tous ces composants. Cependant, cette méthode a ses limites, elle prend du temps et plus le projet grandit, plus le temps alloué à cette tâche devient conséquent. Elle n'est pas infaillible, l'erreur est humaine et il est très probable que je manque des erreurs, des incohérences ou des bugs en testant purement par moi-même.

Ma priorité sera donc de mettre en place des tests, d'abord unitaires pour tester les fonctionnalités isolées (vérification du format de l'ISBN, vérification de l'affichage des infos utilisateur...) puis des tests d'intégrations au niveau du backend pour vérifier le fonctionnement de plusieurs éléments ensemble (processus d'ajout de livre et d'avis...).

Pour les tests unitaires et fonctionnels, ce sera **Vitest** car très adapté au build vite et pour les tests E2E **Playwright**.

## B. Intégration Continue

Dans la continuité des tests, j'aimerais privilégier la mise en place d'une intégration continue via les **Github Actions**. Cela me permettra d'automatiser le lancement des tests et vérifier qu'aucune régression n'a été commise au sein de pull requests, s'assurer de la qualité du code et le respect des standards avec ESLint et Prettier par exemple. Je pourrai également automatiser le déploiement dans le futur une fois que toutes ces vérifications de sécurité seront validées.

Pour mettre en place l'intégration continue, il me faudra créer un fichier `.yml` dans le dossier `github/workflow` de mon projet et le configurer avec les actions que je souhaite lancer à chaque pull request.

## C. Déploiement

Pour le déploiement, j'envisage de séparer le back et le front pour mieux isoler les responsabilités. Pour le front, ce serait **Vercel** qui est très simple à configurer avec un projet React/Vite comme le mien et je pourrais assurer un déploiement continu depuis Github.

Côté back, je pense utiliser **Render** car l'hébergement est gratuit pour les projets de petite taille, il supporte nativement Node.js, Express et il peut également déployer ma base de données PostgreSQL

# 2. Moyen-long terme

Grâce au travail de recherche et les discussions que j'ai pu avoir au début de ce projet, un grand nombre d'idées sont apparues, mais ne pouvaient pas être intégrées au projet original par manque de temps. En travaillant sur ce projet, d'autres fonctionnalités me sont également apparues. Voici donc une liste des idées qui m'intéressent le plus et pourraient voir le jour dans le futur.

- Création de différents thèmes visuels permettant aux utilisateurs de personnaliser leur avis

- Ajout d'un mode sombre
- Création d'une carte récapitulative des lectures du mois (inspiration Spotify Wrapped)
- Possibilité de modifier un avis posté
- Ajout de catégories et filtres (genre, thèmes, âge recommandé...)
- Ajout de la possibilité d'aimer, commenter et partager les avis
- Ajout d'une page de statistique (nombre de pages/livres lus...)
- Changer le visuel de la carte (couleurs, formes, couvertures...)

## Conclusion

Ce projet a été une expérience intéressante et révélatrice sur bien des sujets au-delà de l'aspect technique.

L'un des éléments majeurs que je retiens, c'est la nécessité de rapidement mettre en place un visuel clair, à travers une charte graphique. Mon expérience sur ce projet m'a appris que j'ai tendance à toujours vouloir améliorer ce que je fais et cela m'a amené à repenser les maquettes de nombreuses fois, changeant de style, de couleurs, etc. La mise en place de la charte graphique m'a obligée à l'utiliser comme référence et à respecter les standards qu'elle illustre (boutons, couleurs...). Le processus de création des maquettes finales a été beaucoup plus simple grâce à cette charte et je pense avoir réduit de moitié le temps passé sur leur élaboration comparé aux versions précédentes.

J'ai également pris conscience que l'organisation du temps est un facteur clé dans la réussite d'un projet aussi long que celui-ci. Me projeter sur une période de douze mois s'est révélé très abstrait et difficilement gérable. Mon organisation accès tâche plutôt que temporalité m'a aidée dans un premier temps puis s'est révélée trop abstraite. Je ne savais plus vraiment quelles tâches prioriser. C'est en recommençant mon projet à zéro et en me retrouvant avec une période de six mois seulement pour le mener à bien qui a paradoxalement été un avantage : ce cadre temporel plus court et plus intense m'a permis de me projeter de manière plus concrète de mieux répartir les tâches et d'établir un planning mensuel beaucoup plus tangible. J'ai ainsi adopté un rythme de travail plus fluide, avec des objectifs clairs à court terme.

Le manque de temps a évidemment eu un impact sur la quantité de tâches à réaliser et j'ai dû en prioriser certaines au détriment d'autres. Mon objectif sur ce projet était de terminer les fonctionnalités principales, puis à la fin, de mettre en place des

tests, l'intégration continue, créer le Dockerfile... Plus le projet avançait, plus la difficulté d'implémenter un de ces sujets me paraissait importante et effrayante, et ayant d'autres tâches sur lesquelles travailler, je n'ai fait que les repousser. J'ai fini par ne plus avoir le temps et avec le recul, je pense qu'il serait important pour moi d'alterner les types de tâches. Lors de mes futurs projets, je prendrai le temps de mettre en place des tests assez rapidement pour pouvoir me familiariser avec leur implémentation lorsque le projet n'est pas encore trop conséquent. Cela me facilitera le développement puisque j'aurais une sécurité additionnelle me prévenant immédiatement si un changement impacte le fonctionnement d'autres éléments, évitant des erreurs cachés ou à la formulation trop abstraite pour savoir rapidement de quoi il s'agit.