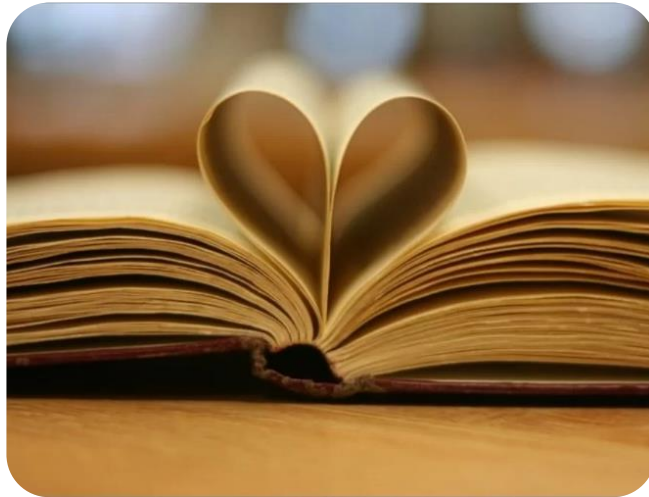


# P\_Web294 – Passion lecture

---



Fabre Antoine, Segalen Alban, Thode Mateo– MID2B  
ETML - Vennes  
32 périodes  
Charmier Gregory

# Table des matières

<b>1</b>	<b>SPÉCIFICATIONS</b>	<b>3</b>
1.1	TITRE	3
1.2	DESCRIPTION	3
1.3	MATÉRIEL ET LOGICIELS À DISPOSITION	3
1.4	PRÉREQUIS	3
1.5	CAHIER DES CHARGES	3
1.6	LES POINTS SUIVANTS SERONT ÉVALUÉS	3
1.7	VALIDATION ET CONDITIONS DE RÉUSSITE	3
<b>2</b>	<b>PLANIFICATION INITIALE</b>	<b>3</b>
<b>3</b>	<b>ANALYSE</b>	<b>4</b>
3.1	DOCUMENT D'ANALYSE ET CONCEPTION	4
<b>4</b>	<b>RÉALISATION</b>	<b>8</b>
4.1	MODE DE FONCTIONNEMENT	8
4.2	CONVENTIONS	8
4.2.1	Commits GitHub	8
4.3	DOSSIER DE RÉALISATION	9
4.3.1	Base de données	9
4.3.2	Structure du code	10
4.3.3	Routes	10
4.3.4	Sécurité	10
4.3.5	Gestion des erreurs	11
4.4	ECO-CONCEPTION WEB	11
4.5	TESTS	11
<b>5</b>	<b>CONCLUSION</b>	<b>12</b>
5.1	CRITIQUE SUR LA PLANIFICATION	12
5.2	BILAN DES FONCTIONNALITÉS DEMANDÉES	12
5.2.1	Page d'accueil	12
5.2.2	Filtrage des ouvrages par catégorie	12
5.2.3	Ajout d'un ouvrage	12
5.2.4	Modification d'un ouvrage	13
5.2.5	Suppression d'un ouvrage	13
5.2.6	Vue détail d'un livre	13
5.2.7	Droits administrateur	13
5.3	BILAN PERSONNEL	13
5.3.1	Alban	13
5.3.2	Antoine	13
5.3.3	Mateo	13
<b>6</b>	<b>ANNEXES</b>	<b>13</b>
<b>7</b>	<b>WEBOGRAPHIE</b>	<b>13</b>

# 1 SPÉCIFICATIONS

## 1.1 Titre

Passion lecture

## 1.2 Description

Réaliser le frontend d'une application permettant de partager sa passion pour la lecture.

## 1.3 Matériel et logiciels à disposition

- Microsoft Windows
- Vscode
- Serveur local
- Navigateurs Web
- Accès à Internet

## 1.4 Prérequis

Modules 106, 162, 164, 231, 293, 319, 320, 322 et 295

## 1.5 Cahier des charges

Le cahier des charges a été lu, compris et signé par tous les membres du groupe. Il est disponible [ici](#).

## 1.6 Les points suivants seront évalués

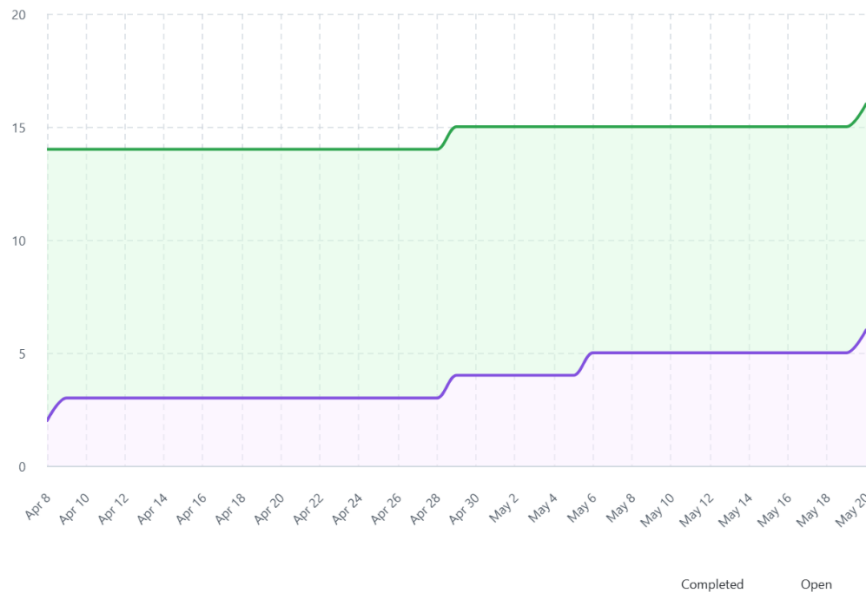
- Le rapport
- Les planifications (initiale et détaillée)
- Le journal de travail
- Le code et les commentaires
- Les documentations de mise en œuvre et d'utilisation

## 1.7 Validation et conditions de réussite

- Compréhension du travail
- Possibilité de transmettre le travail à une personne extérieure pour le terminer, le corriger ou le compléter
- Etat de fonctionnement du produit livré

# 2 PLANIFICATION INITIALE

La planification a été faite avec [GitHub project](#). Nous avons commencé par mettre toutes les tâches connues dans le Backlog (colonne de gauche) avant de commencer le développement. Il y a ensuite eu des tâches que l'on a ajoutées, mais celles-ci sont peu nombreuses.



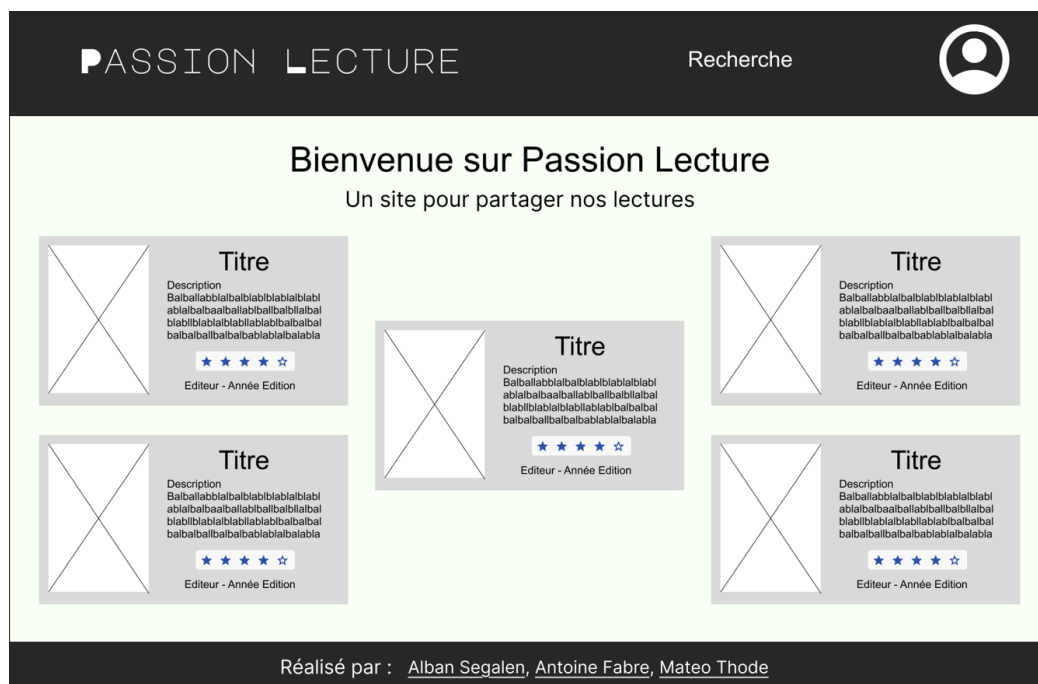
En vert les tâches planifiées et en violet celles qui sont terminées.

### 3 ANALYSE

### 3.1 Document d'analyse et conception

Au début du projet, nous avons fait la maquette des différentes pages du site web avec Figma. Nous avons également utilisé les premières maquettes réalisées lors de la première partie du projet (backend).

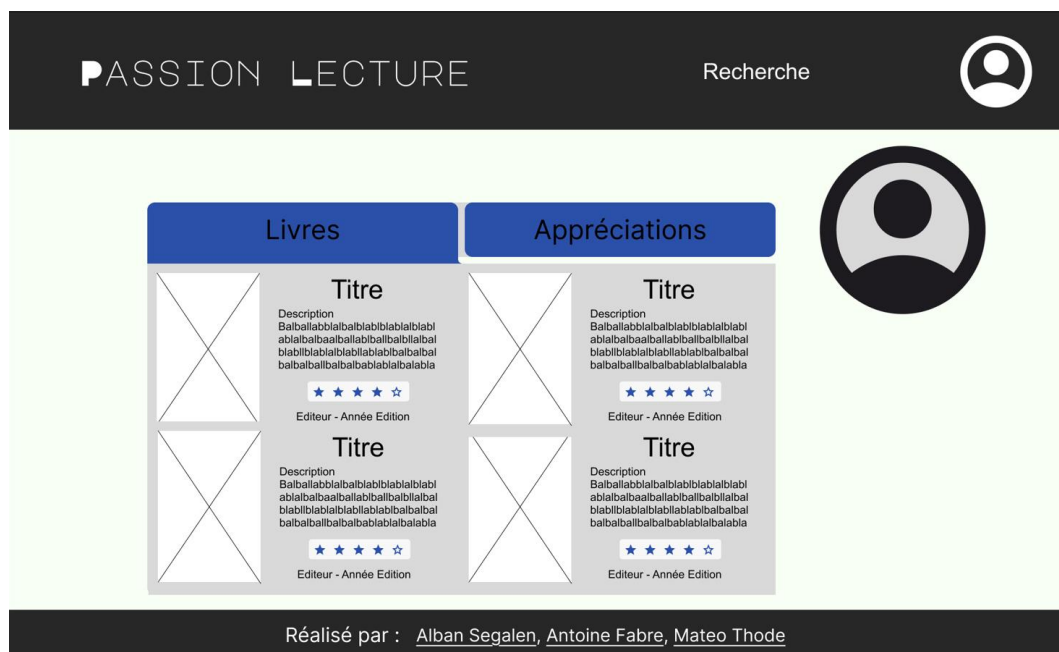
Voici la maquette de la page d'accueil :



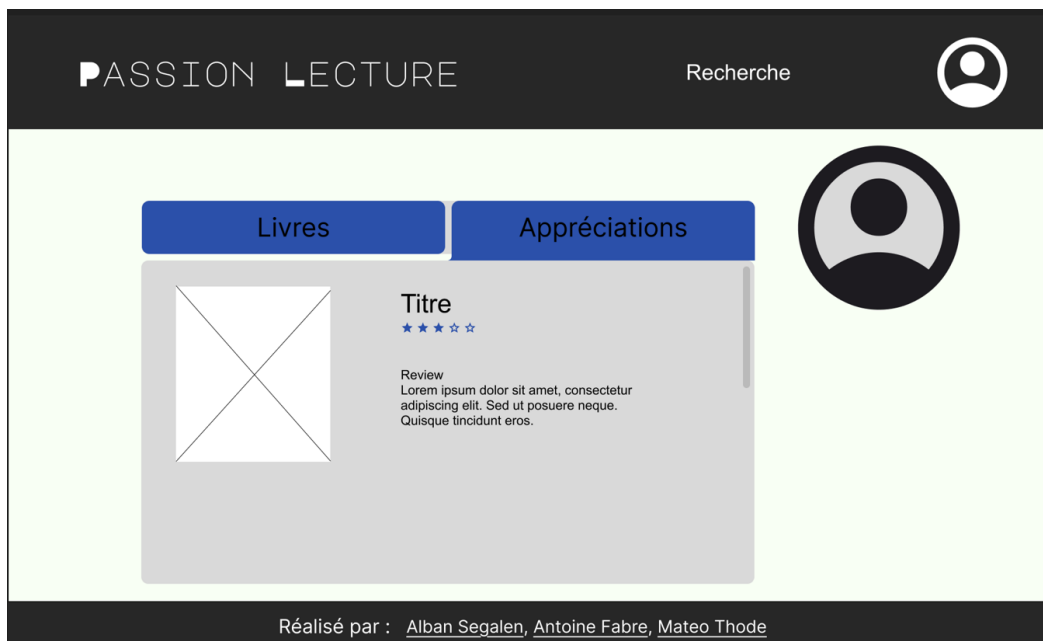
La page de détail d'un livre :



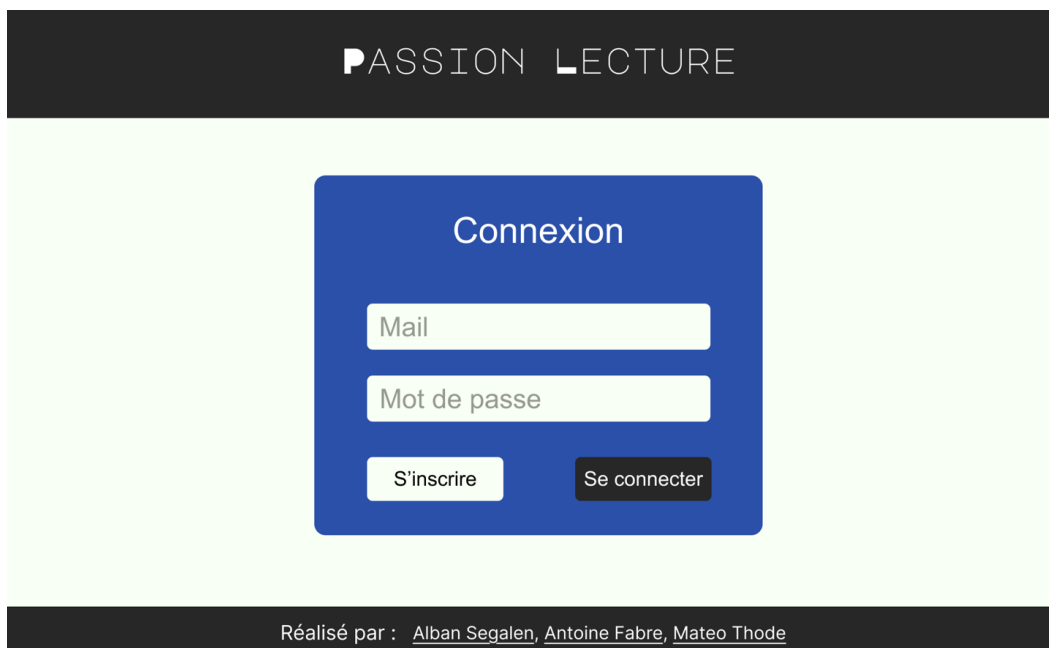
La page de profil d'un utilisateur (« Livres » sélectionné):  
C'est là qu'on voit la liste de tous les livres qu'il a postés.



La page de profil d'un utilisateur (« Appréciations » sélectionné):  
C'est là qu'on voit la liste de toutes les appréciations qu'il a postées.



La page de connexion :



La page d'inscription :


A dark-themed login and registration form titled "Inscription". The form is centered on a light gray background. It contains two input fields: "Mail" and "Mot de passe". Below the "Mot de passe" field, there are two buttons: "Se connecter" and "S'inscrire". The "S'inscrire" button is highlighted with a red border.

La page de recherche :

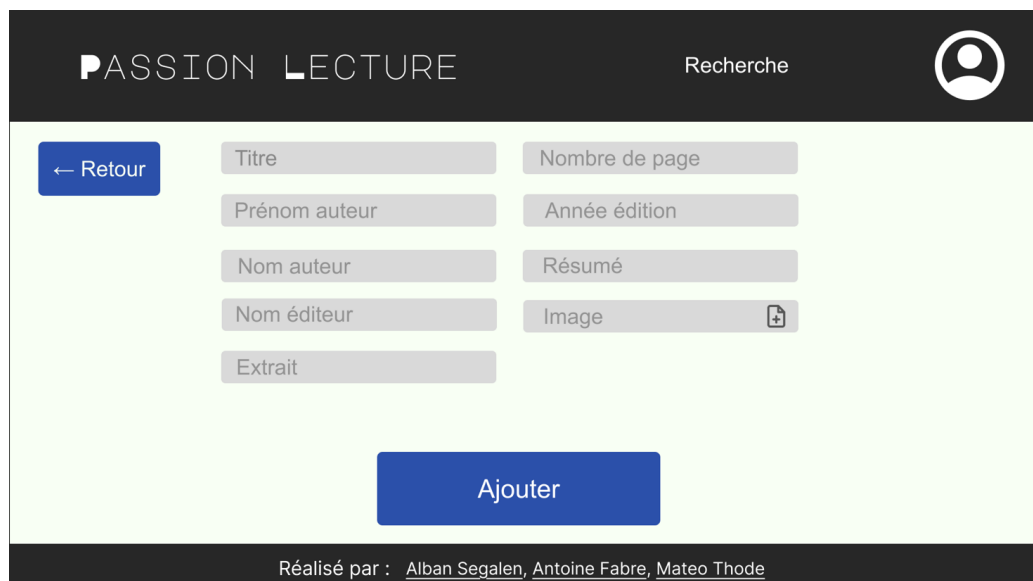
PASSION

LECTURE

Recherche



La page d'ajout d'un livre :



## 4 RÉALISATION

### 4.1 Mode de fonctionnement

Pour le développement de ce projet, nous avons utilisé la méthodologie agile ainsi que des éléments de la méthode « Scrum ». Nous prôtons la communication plutôt que les processus en ayant beaucoup d'interactions pendant le travail. Nous n'hésitons pas à nous demander de l'aide et à discuter de nos doutes quant à la réalisation.

Chaque séquence de travail est débutée par un « meeting » de 5-10 minutes (daily scrum) dans lequel nous résumons ce qui s'est passé précédemment et nous répartissons le travail pour la session. Quand la séquence est longue, nous prenons un second moment pour une réunion du même type que la première.

Notre atmosphère de travail est décontractée et amicale, afin de rendre les moments de travail agréables et pour conserver notre motivation à avancer ensemble.

Nous utilisons [GitHub](#) pour la gestion des versions de l'application et [GitHub Project](#) pour la planification et répartition des tâches.

### 4.2 Conventions

Notre équipe de développement, utilisant les méthodologies agiles, est dynamique. Cela se traduit par les éléments mentionnés au [point précédent](#) et ceux qui suivent.

#### 4.2.1 Commits GitHub

Afin de rendre la lecture et la compréhension des commits GitHub, nous nous sommes mis d'accord pour que le message de chaque commit contienne au moins un emoji. Ceci a également pour but de mettre en valeur notre dynamisme tout en restant sérieux et professionnel.

L'outil d'aide utilisé est le site [gitmoji.dev](#)

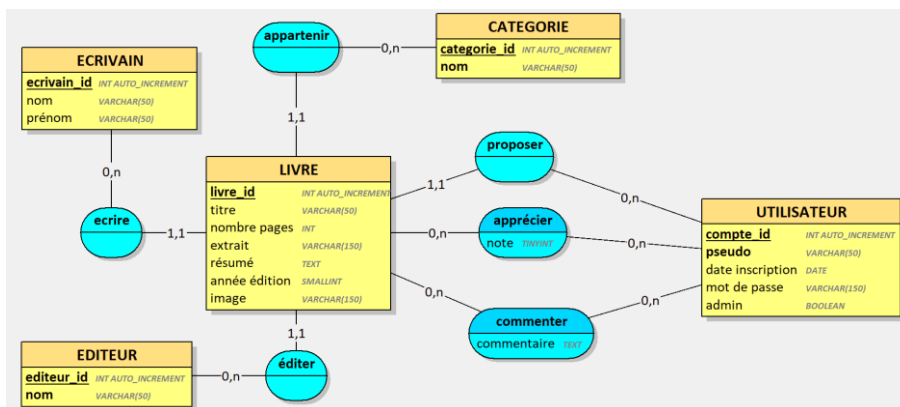


## 4.3 Dossier de Réalisation

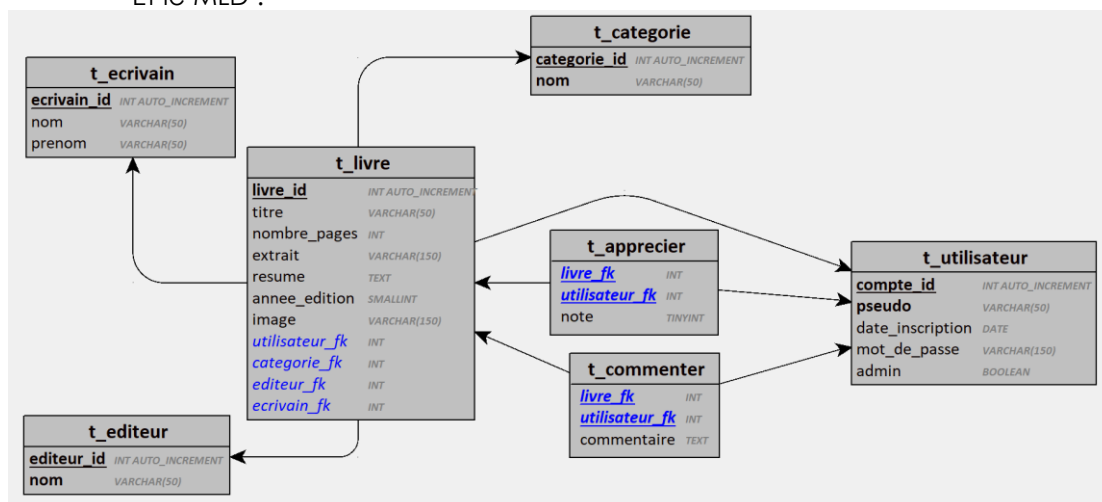
### 4.3.1 Base de données

Bien que ceci soit le rapport pour la partie frontend du projet, voici un rafraîchissement de la structure de la base de données, pour une meilleure compréhension globale.

Voici le MCD :



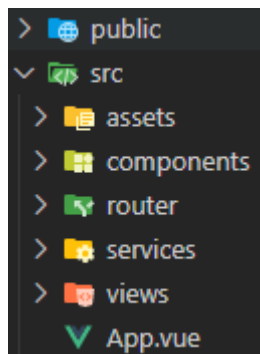
Et le MLD :



Nous avons utilisé le script SQL de création de la base de données au début de la phase de développement mais nous avons arrêté de l'utiliser quand nous avons créé les modèles. Car Sequelize crée les tables automatiquement.

#### 4.3.2 Structure du code

Voici l'organisation des sous-dossiers dans le dossier « src » :



Notre projet est donc séparé en de nombreux fichiers qui sont eux-mêmes classés selon leur fonction.

#### 4.3.3 Routes

Nous avons configuré Vue Router pour gérer les pages de manière dynamique et fluide dans le frontend.

#### 4.3.4 Sécurité

Dans notre application, nous avons veillé à la sécurité à plusieurs niveaux afin de garantir protection des données des utilisateurs ainsi que du serveur.

L'authentification repose sur l'utilisation de jetons JWT (JSON Web Token), qui permet de sécuriser l'accès à certaines routes. Chaque requête nécessitant une authentification est filtrée par un middleware qui vérifie la validité du token et l'autorisation de l'utilisateur. Un jeton expiré ou invalide causera un refus d'accès (erreur 403).

Nous avons également mis en place une gestion des permissions en différenciant les rôles des différents comptes, s'ils sont admins ou pas. Les actions critiques, comme la modification des données utilisateur ou la suppression de ressources, sont restreintes aux administrateurs. Ceci empêche un utilisateur standard d'effectuer des opérations non autorisées.

La validation des entrées utilisateur a été intégrée afin de prévenir les injections SQL. L'ORM Sequelize permet d'échapper automatiquement les valeurs insérées dans la base de données. De plus, les mots de passe sont hachés avec bcrypt avant d'être stockés, garantissant qu'ils ne puissent pas être (trop) compromis en cas de fuite.

Bien que l'API REST en backend ne gère pas d'interface utilisateur, nous avons restreint les origines autorisées et nous nous assurons que seules les requêtes légitimes peuvent accéder aux données.

Dans le frontend également nous validons les données entrées par les utilisateurs, pour les risques mentionnés ci-dessus.

Dans le frontend, nous affichons les erreurs 404 et 500 avec des messages génériques, pour ne pas communiquer des informations sensibles qui pourraient être utilisées pour prendre connaissance du fonctionnement de l'application et potentiellement mettre à bien une attaque malveillante (voir [Gestion des erreurs](#)).

Nous estimons donc que la sécurité implémentée dans notre application est suffisante et adéquate.

#### 4.3.5 Gestion des erreurs

Les erreurs sont gérées et affichées dans le frontend et affichent un message générique, pour les raisons citées au [point précédent \(sécurité\)](#).

Voici le code de la route pour afficher la page 404 dans le fichier `router/index.js` :

```
{
  //404
  path: '/*:pathMatch(.*)*',
  name: 'not-found',
  component: NotFound,
},
```

Nous affichons cette page lorsque la ressource demandée par l'utilisateur n'existe pas.

Et la route pour la page 500 :

```
{
  //500
  path: '/erreur-serveur',
  name: 'erreur-serveur',
  component: ErreurServeur,
},
```

Nous affichons cette page lorsqu'il y a une erreur côté serveur et qu'on ne peut pas afficher ce que l'utilisateur demande, à cause de l'erreur serveur.

### 4.4 Eco-conception Web

Dans notre application, nous avons, dès le début de la conception, fait attention à ne pas utiliser plus de ressources que nécessaire. Dans la mesure de nos connaissances, le code est propre et sans boucle inutile par exemple.

Nous n'avons pas de chargement d'images ou d'informations qui ne sont pas utilisés pour le frontend.

Nous ne faisons pas de rafraîchissement inutile de la page non plus.

### 4.5 Tests

Nous n'avons malheureusement pas eu le temps de faire des tests unitaires dans la partie frontend. Nous aurions dû mais le groupe a pris un certain retard à cause de l'absence d'un des membres pendant 19 périodes.

S'ils avaient été réalisés, les tests fonctionneraient comme ceci : chaque composant aurait été isolé dans des fichiers de test dédiés. On aurait vérifié les interactions utilisateur (clics, envois de formulaire), les variables ainsi que le bon rendu des données dynamiques.

Par exemple, un test unitaire pour le composant d'ajout d'ouvrage aurait vérifié que tous les champs obligatoires sont correctement contrôlés avant soumission. Un autre test aurait simulé un utilisateur ajoutant un commentaire à un ouvrage, avec vérification de la mise à jour du DOM.

Et idéalement, pour aller plus loin, les tests pourraient être intégrés avec GitHub Actions pour être exécutés automatiquement à chaque push.

## 5 CONCLUSION

Ce projet nous a permis de mettre en pratique toutes les compétences acquises en théorie et dans le module précédent. En reprenant la base déjà construite dans le backend, nous avons pu bâtir une interface utilisateur complète, fonctionnelle et agréable à utiliser. Le fait de devoir collaborer de manière autonome, avec une bonne gestion de version et des outils professionnels nous apprend toujours beaucoup. Malgré le léger stress à la fin, dû au retard que nous avons engendré, le produit final est abouti et répond aux attentes du cahier des charges (voir [point suivant](#)).

Chacun des membres a apporté ses compétences et a aidé les autres ce sur quoi ils avaient plus de peine. Ceci a permis un bon équilibre dans la réalisation du projet. Le travail en équipe s'est déroulé dans une ambiance agréable et productive.

### 5.1 Critique sur la planification

La planification initiale était correcte, mais n'a pas suffisamment pris en compte les imprévus, nous aurions également dû plus préparer l'organisation de la méthodologie de fonctionnement comme nous savions qu'un des membres allait être absent deux semaines. Le découpage des tâches était globalement bon, mais certaines tâches complexes ont été sous-estimées. Pour un projet similaire à l'avenir, il serait judicieux de prévoir des marges de sécurité.

### 5.2 Bilan des fonctionnalités demandées

#### 5.2.1 Page d'accueil

La route racine ("/") interroge l'API avec le paramètre `?limit=5&order=desc` pour récupérer les cinq livres les plus récents. L'explication de l'utilité du site est un composant statique réutilisable (`AppIntro.vue`). L'ensemble est accessible sans authentification.

#### 5.2.2 Filtrage des ouvrages par catégorie

La route `/livres/:cat?` accepte un paramètre optionnel. Lorsque `:cat` est absent, tous les ouvrages sont affichés ; sinon, un filtre SQL est appliqué côté API (`WHERE category = :cat`).

#### 5.2.3 Ajout d'un ouvrage

Cette vue est protégée et nécessite un jeton JWT pour être accédée. Le formulaire emploie v-model pour la validation (titre non vide, nombre de pages  $> 0$ , fichier extrait  $\leq 2\text{Mo}$ , etc.). La soumission déclenche un POST `/books` puis une redirection vers la page de profil.

#### 5.2.4 Modification d'un ouvrage

Il y a également une protection et la nécessité d'un jeton JWT pour accéder à la page. Le formulaire est pré-rempli via un GET /books/:id et déclenche un PUT à la sauvegarde.

#### 5.2.5 Suppression d'un ouvrage

C'est sur la page de compte que l'on peut supprimer un livre. Cette route est donc également protégée par un accès JWT. Quand le bouton est appuyé, un pop-up apparaît et demande confirmation.

#### 5.2.6 Vue détail d'un livre

Récupère l'ouvrage et ses informations, ainsi que tous ses commentaires. Nous enregistrons les appréciations de chaque livre via POST /ratings, elles vont de 0-5. Le commentaire est envoyé via POST /comments et injecté instantanément dans le DOM pour un rendu réactif.

Il y a également la moyenne des appréciations qui est affichée.

#### 5.2.7 Droits administrateur

Le rôle est inclus dans le JWT (admin : « true »). L'administrateur a accès à toutes les pages et est autorisé à exécuter toutes les actions possibles.

### 5.3 Bilan personnel

#### 5.3.1 Alban

#### 5.3.2 Antoine

#### 5.3.3 Mateo

J'ai beaucoup apprécié travailler sur ce projet, surtout le fait qu'on ait commencé sur la partie backend et qu'on ait ensuite continué sur le même projet.

Je me sens quand même un peu mal pour le retard que j'ai engendré sur le projet à cause de mes deux semaines d'absence. J'ai quand même dû rattraper beaucoup de théorie sur les heures de projet mais j'ai eu la chance d'avoir un groupe qui m'a toujours soutenu et aidé.

## 6 ANNEXES

Tous les fichiers et documents annexes se trouvent dans le repository GitHub, à cette adresse : [https://github.com/ASETML/P\\_Web295](https://github.com/ASETML/P_Web295)

## 7 WEBOGRAPHIE

La relecture a été assistée par des outils d'intelligence artificielle (OpenAI ChatGPT)