P\_Web 295 :



Naudet Maël, Morier Mina, Fleurdelys Brendan– FID2

24p

Durée

G. Charmier

Table des matières

[1. Introduction 3](#_Toc212816695)

[2. Analyse 4](#_Toc212816696)

[2.1 Planification 4](#_Toc212816697)

[2.2 Modélisation 5](#_Toc212816698)

[2.3 Maquettes 7](#_Toc212816699)

[2.4 Structure du code 13](#_Toc212816700)

[2.5 API REST 15](#_Toc212816701)

[2.6 Schéma de l'architecture 19](#_Toc212816702)

[3. Réalisation 21](#_Toc212816703)

[3.1 Gestion de l’authentification 21](#_Toc212816704)

[3.2 Mesures de sécurités 22](#_Toc212816705)

[3.3 Fonctionnalités demandées 23](#_Toc212816706)

[3.4 Eco-Conception 24](#_Toc212816707)

[4. Conclusion 24](#_Toc212816708)

[4.1 Organisation du groupe 24](#_Toc212816709)

[4.2 Conclusion générale 25](#_Toc212816710)

[4.3 Conclusion personnelle 25](#_Toc212816711)

[4.4 Conclusion planification 25](#_Toc212816712)

[5. Utilisation de l’IA 26](#_Toc212816713)

[6. Webographie / Bibliographie / Glossaire 26](#_Toc212816714)

# Introduction

Passion Lecture est un site web qui permettra à ses utilisateurs de rechercher des livres qui leur plaisent, les noter de 1 à 5 et de les commenter pour approfondir leur avis.

L’objectif principal est de créer une communauté active d’utilisateurs qui échangent autour de leurs lectures et découvrent de nouveaux ouvrages grâce aux évaluations et recommandations des autres.

Ce projet se concentre principalement sur la conception et le développement de l’API REST du site Passion Lecture.

La partie frontend, quant à elle, sera réalisée dans un projet ultérieur, afin de séparer clairement les responsabilités entre la couche serveur (backend) et la couche cliente (interface utilisateur).

Pour répondre à ces besoins techniques et garantir une architecture robuste, le développement du backend s’apuie sur le framework AdonisJS6, accompagné de son ORM Lucid pour la gestion des données.

Cette combinaise permet de structurer efficacement l’application et de gérer les différentes entités du systèmes telles que le catalogue d’ouvrages, leurs auteurs, leurs catégories, des utilisateurs, ainsi que les commentaires et évaluations associés.

L’objectif est de fournir un backend propre, maintenable et sécurisé, prêt à alimenter un frontend web ou mobile conforme aux maquettes UI définies dans le cahier des charges.

Afin de réaliser ce projet, un dépôt Github a été créé[[1]](#footnote-2) qui contient le code source complet du backend, le cahier des charges du projet, les maquettes servant de référence pour l’intégration du frontend et les journaux de travail retraçant l’avancement du développement et les choix techniques effectués.

# Analyse

## Planification

La planification du projet a entièrement été effectuée sur Github Projects afin d’avoir une vraie centrialisation des outils de gestion et de développement. Cela permet de regrouper en un seul espace le suivi de tâches, le dépôt de code, les issues, ainsi que la documentation, garantissant une meilleure visibilité sur l’avancement du projet et une collaboration plus fluide.

Ce projet est disponible ici : [Passion lecture API](https://github.com/users/Fbrend23/projects/9)

## Modélisation

Afin de mener à bien ce projet, une modélisation de la base de données à été donnée à chacun :

Une image contenant texte, capture d’écran, diagramme, ligne

Le contenu généré par l’IA peut être incorrect.

Figure 1 – MCD de la db

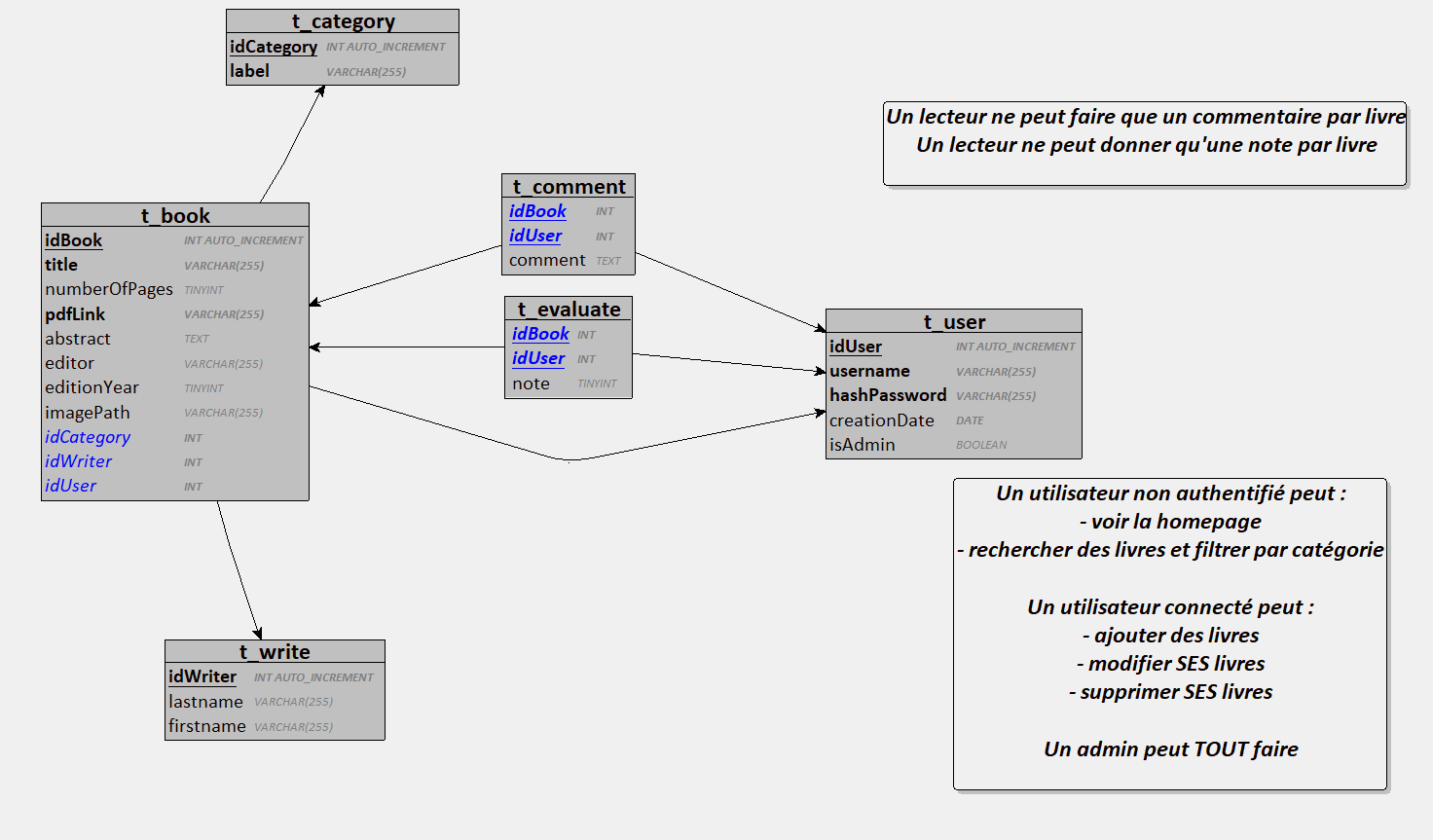


Figure 2 – MLD de la db

Ces modèles conceptuel et logique de données illustre la structure relationnelle de la base de données utilisées par l’API. Il vise à modéliser les principales entités nécessaires à la gestion des livres, de leurs auteurs, des catégories, ainsi que des utilisateurs et de leurs interactions.

Entités principales

* BOOK  
  Représente un ouvrage. Chaque livre possède un identifiant unique (*idBook*), un titre, un résumé, un nombre de pages, un lien PDF, une année d’édition, un éditeur et éventuellement une image de couverture.

Un livre est écrit par un seul auteur mais un auteur peut écrire plusieurs livres.

De plus, chaque livre appartient à une catégorie, tandis qu’une catégorie peut regrouper plusieurs ouvrages.

* WRITER

Correspond aux auteurs. Chaque auteur est identifié par un *idWriter*, un nom et un prénom.

La relation **“**write**”** relie l’auteur à ses livres (cardinalité 1,n côté livre).

* CATEGORY

Permet de classer les livres selon un genre ou un thème.

Chaque catégorie (*idCategory*, *label*) peut regrouper plusieurs ouvrages, mais chaque livre n’appartient qu’à une seule catégorie.

* USER  
  Représente les utilisateurs du site, identifiés par *idUser*.

Un utilisateur possède un *username*, un mot de passe chiffré (*hashPassword*), une date de création de compte (*creationDate*) et un statut (*isAdmin*).

Les utilisateurs sont à la fois lecteurs et potentiellement contributeurs (ajout de livres).

Relations utilisateur – livre

Deux relations traduisent les interactions principales entre les utilisateurs et les ouvrages :

* Comment

Un utilisateur peut laisser un seul commentaire par livre, enregistré dans le champ *comment* (type TEXT).

La relation est donc de cardinalité (0,n) côté utilisateur vers (0,n) côté livre, avec une contrainte d’unicité (1 commentaire max par couple user/livre).

* Evaluate

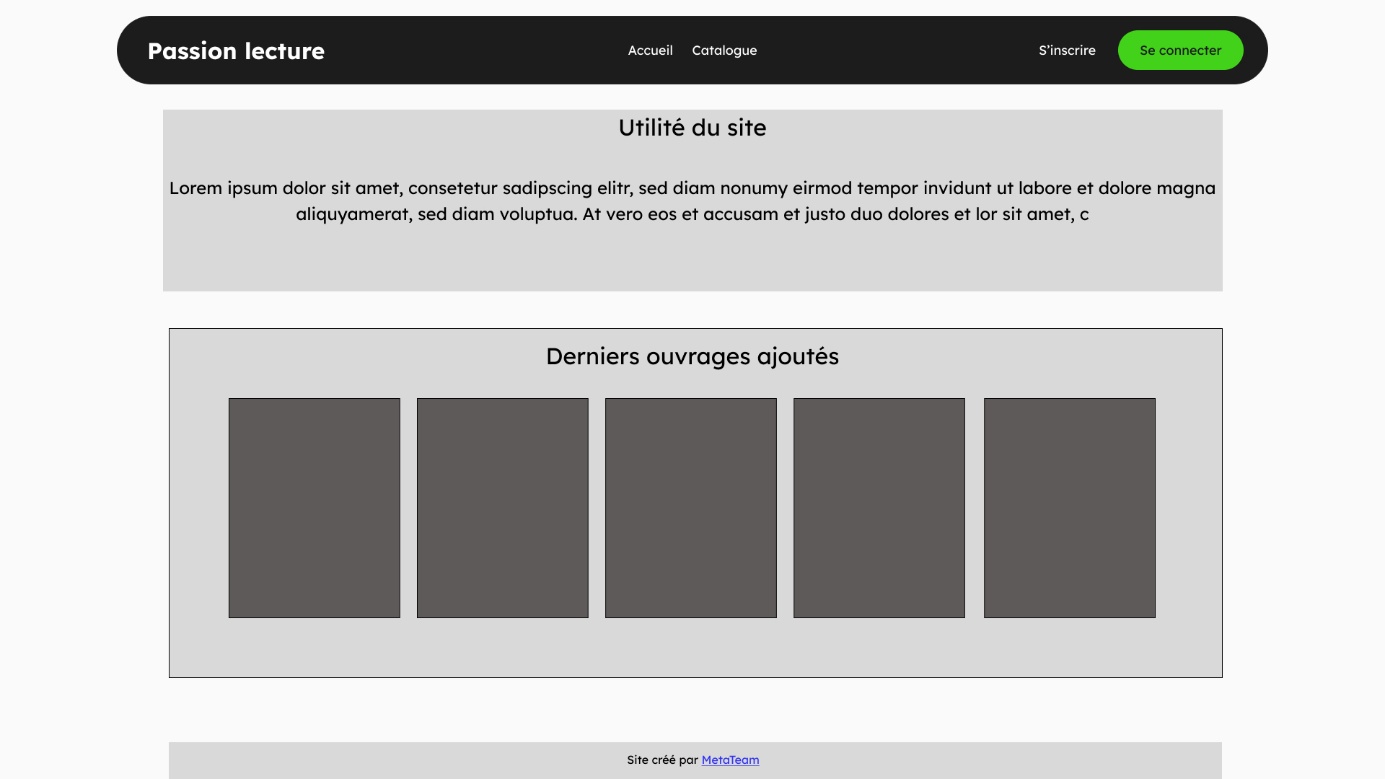
Un utilisateur peut attribuer une seule note par livre, enregistrée dans le champ *note* (type TINYINT).

De même, la relation est de type (0,n) ↔ (0,n) avec une contrainte d’unicité similaire.

Il y a également un prémice du système d’authentification et de ce qu’un utilisateur peut faire ou non, ce dernier sera expliqué plus tard, dans une section pour.

## Maquettes

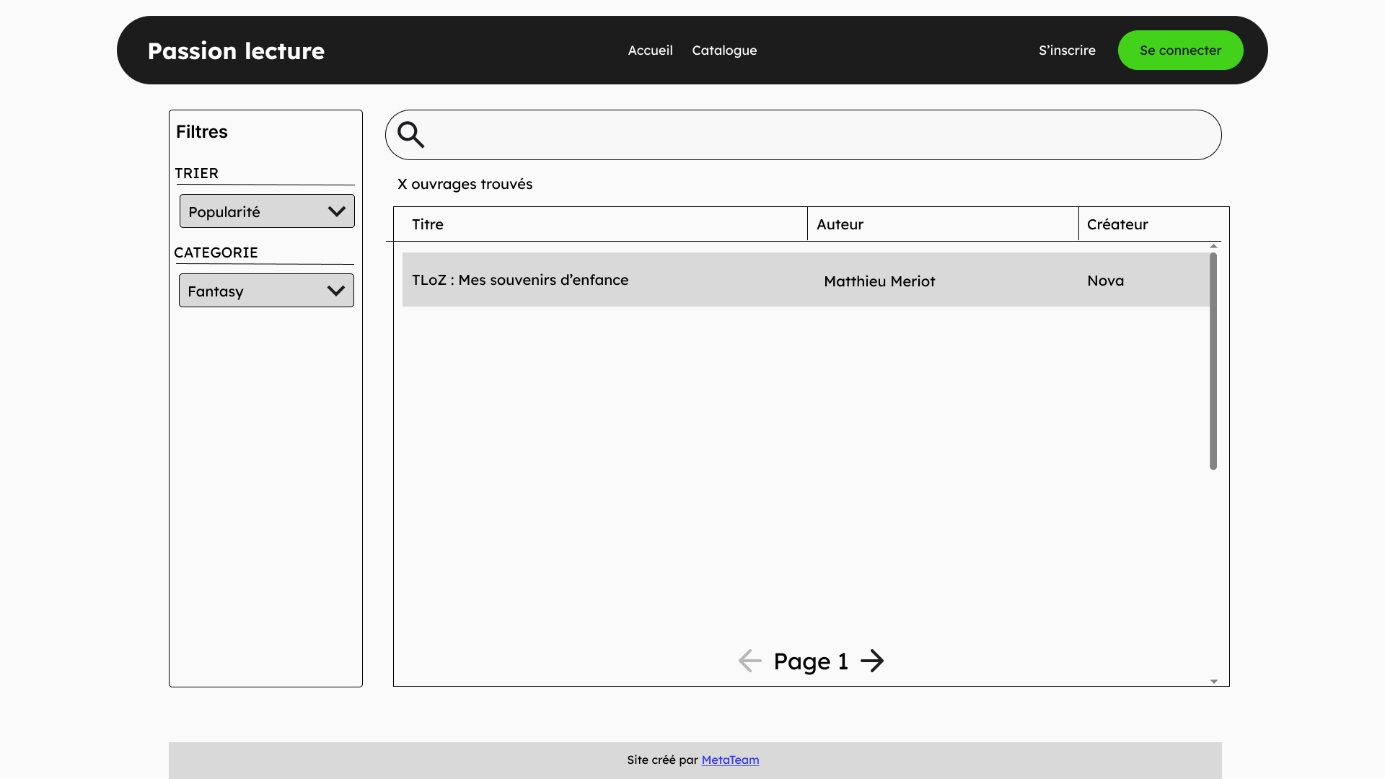
Page d’accueil



La page d’accueil contient une explication de l’utilité du site et les cinq derniers ouvrages ajoutés.

* Accessibilité : Tout public

Catalogue



Le catalogue contient la liste des ouvrages par catégorie, permet de visualiser les titres, l’auteur et le pseudo de la personne qui a ajouté l’ouvrage.

* Accessibilité : La recherche est tout public mais pour voir un ouvrage en détail il faut se connecter.

Ouvrage





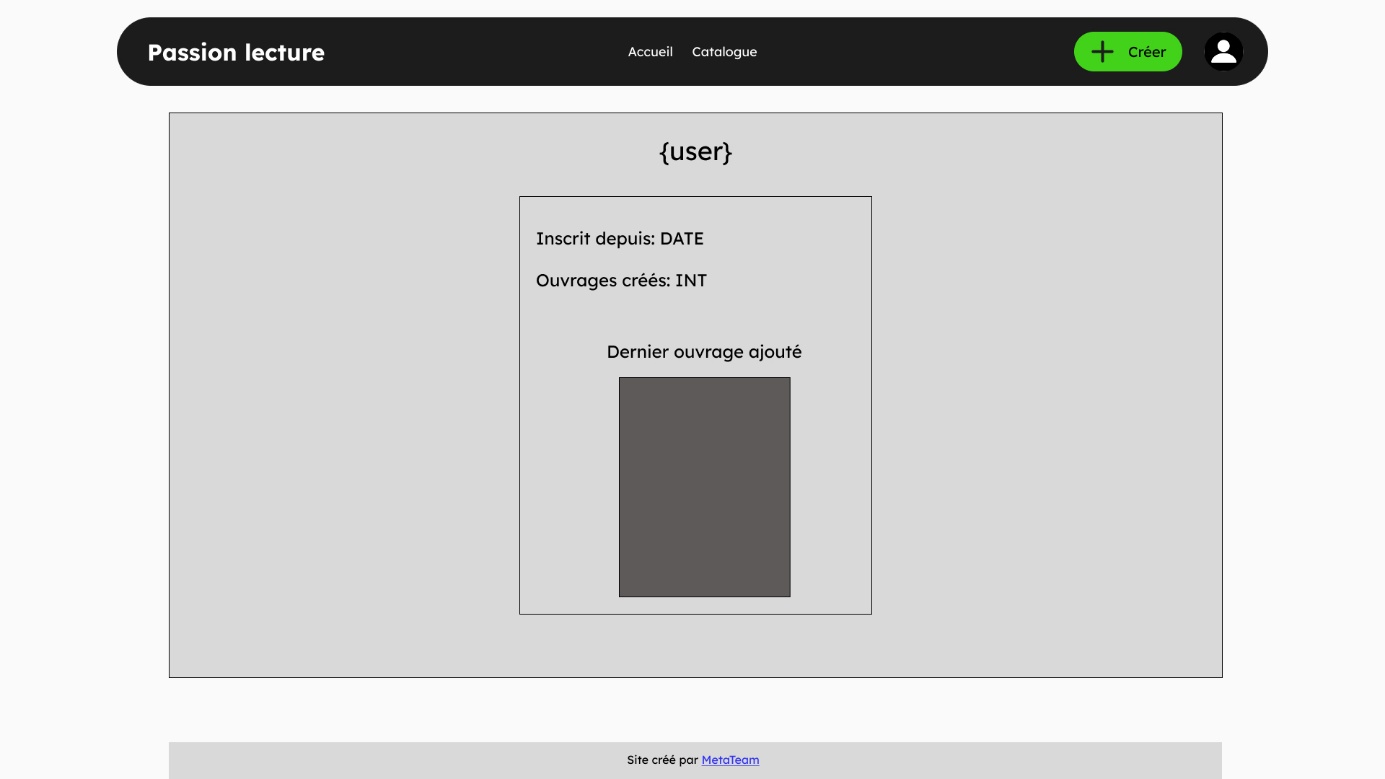
La page d’ouvrage permet de visualiser les détails d’un ouvrage, d’ajouter un commentaire, de donner une note à l’ouvrage.

Cette page doit contenir :

1. Le titre de l’ouvrage
2. La catégorie
3. Le nombre de pages
4. Un lien vers un extrait pdf
5. Un résumé
6. Nom et prénom de l’écrivain auteur de l’ouvrage
7. L’éditeur
8. L’année d’édition
9. La moyenne des notes des utilisateurs
10. Les commentaires
11. Une image de la couverture de l’ouvrage

Accessibilité : Utilisateurs connectés.

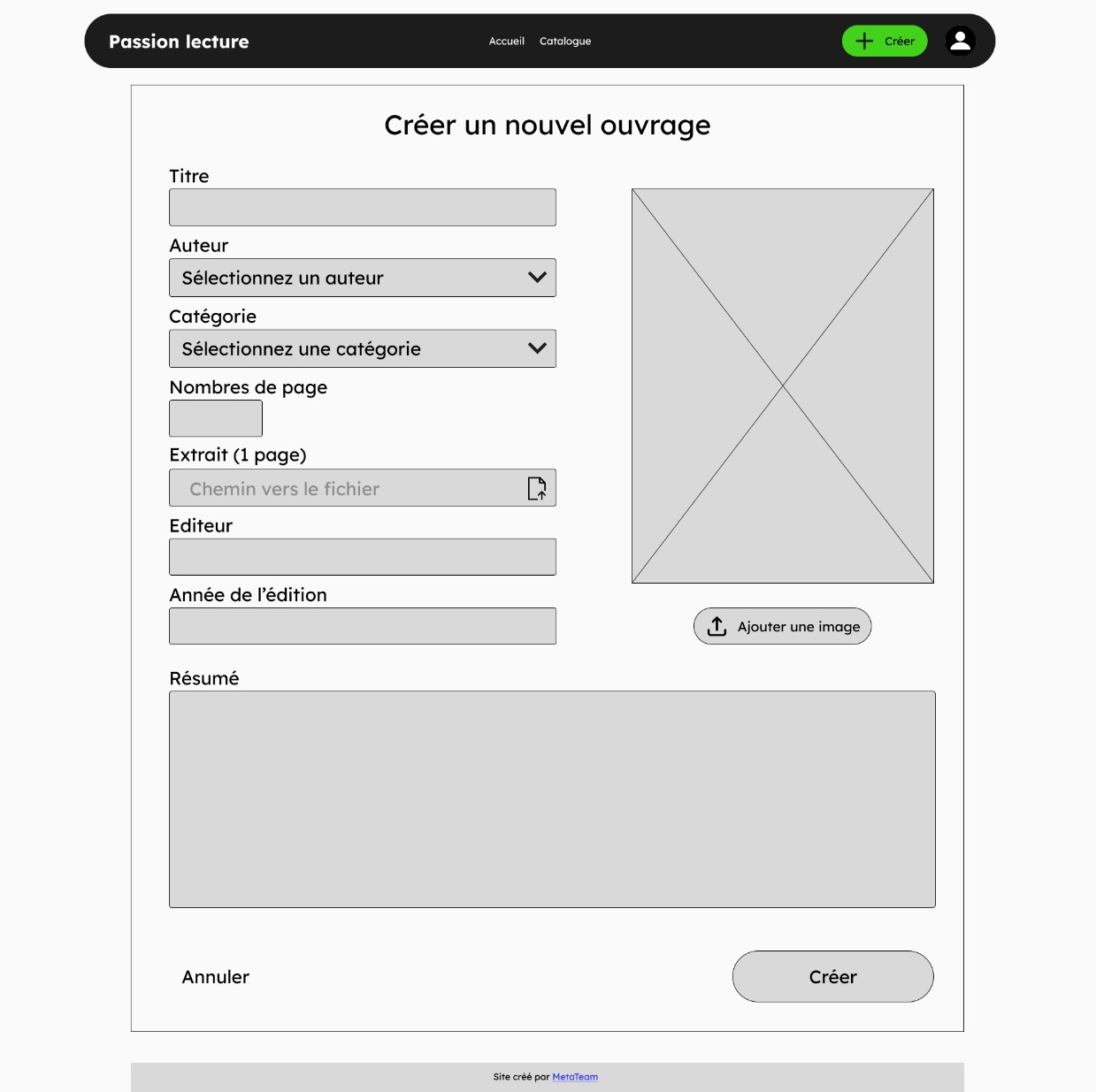
Profil utilisateur



Cette page permet de voir le profil d’un utilisateur, affiche son pseudo, la date de son inscription, le nombre total d’ouvrages créer ainsi que le dernier qu’il a ajouté au site.

Accessibilité : Utilisateurs connectés.

Ajout d’un ouvrage

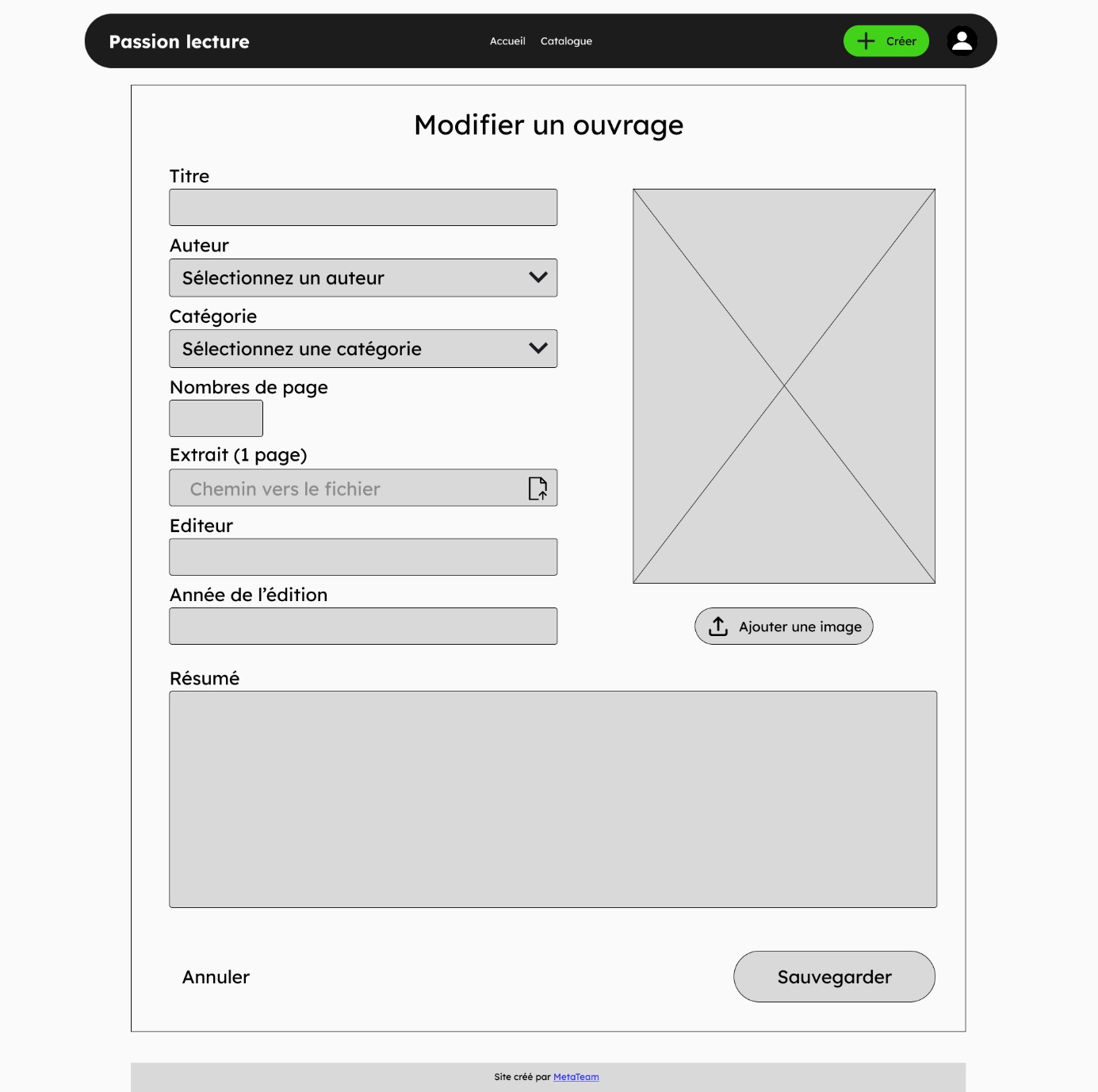


Cette page permet d’ajouter un ouvrage à la base de données. Pour cela, il suffit d’ajouter :

* Le titre
* L’auteur
* La catégorie
* Le nombre de pages
* Un extrait du livre
* L’éditeur
* L’année d’édition
* Un résumé du livre
* Une image d’illustration

Accessibilité : Utilisateurs connectés.

Modification d’un ouvrage



Cette page permet de modifier un ouvrage présent dans la base de données. Pour cela, il est possible de modifier :

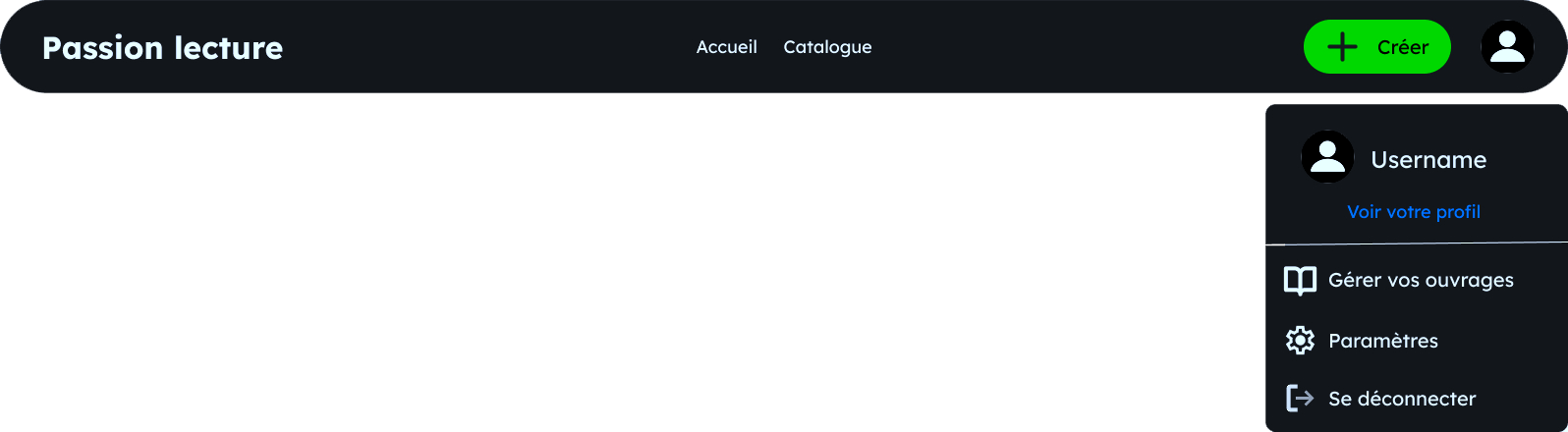
* Le titre
* L’auteur
* La catégorie
* Le nombre de pages
* Un extrait du livre
* L’éditeur
* L’année d’édition
* Un résumé du livre
* Une image d’illustration

Accessibilité : L’utilisateur qui a créé l’ouvrage.

Liste des ouvrages d’un utilisateur

La liste des ouvrages par utilisateur permet d’avoir tous les ouvrages qui ont été ajoutés par des utilisateurs.

Accessibilité : Chaque utilisateur n’a accès qu’a sa propre liste d’ouvrages ajoutés.

Navbar

La barre de navigation permet de retourner à l’accueil, voir le catalogue, ajouter un livre, voir ses livres, voir les paramètres de l’application et se déconnecter.

## Structure du code

Le backend étant fait avec le framework AdonisJS, nous devons suivre la structure suivante :

* **app/controllers/** :

Logique REST par ressource (books, authors, categories, comments, evaluations, users, auth). Chaque contrôleur regroupe les méthodes nécessaires à la gestion du cycle CRUD. Les contrôleurs orchestrent les appels entre les modèles, les validateurs et les middlewares, tout en gérant les réponses http (JSON).

* **app/models/** :

Ce répertoire contient les modèles Lucid, qui représentent les tables de la base de données et définissent leur relations.

Chaque modèle gère la correspondance entre les données stockées en base MySQL et les objets manipulés par le code.

* **app/validators/** :

Ce dossier regroupe les validateurs VineJS, utilisés pour contrôler la conformité des données reçues dans les requêtes http.

Ces validations sont indispensables avant toute modification ou création d’entité dans la base de données.

* **app/middleware/** :

Ce dossier contient les middlewares utilisés pour interceptier les requêtes avant qu’elles n’atteignent les contrôleurs.

Les middlewares permettent d’assurer la sécurité, la cohérence et la gestion des accès à l’application.

* **database/migrations/** :

Ce répertoire contient les migrations définissant la structure de la base de données MySQL (tables, colonnes, clés primaires et étrangères, contraintes).

Elles permettent de versionner le schéma et de le reconstruire facilement sur n’importe quel environnement (développement, test, production).

* **database/ seeders|factories:**

Ces fichiers sont utilisés pour préremplir la base de données avec des données d’exemple ou des données de test.

Les factories définissent les modèles de génération, tandis que les seeders exécutent ces modèles pour injecter des entrées automatiques (ex : livres fictifs, catégories…)

* **start/routes.ts** :

Fichier central de l’API où sont déclarées les routes REST.

Il définit :

* les endpoints (ex : /books, /authors/ :id)
* Les groupes de routes (par ressource ou par type d’accès)
* Les middlewares associés à chaque groupe (authentification, validation, etc…)
* Ce fichier est le point d’entrée principal reliant les requêtes http aux contrôleurs correspondants.

## API REST

Voilà toutes les routes dans notre API.

Premièrement, voilà les routes qui ont été implémentées.

| **Domaine** | **URI** | **Verbe** | **JSON** | **Description & Remarques** | **Accessibilité** |
| --- | --- | --- | --- | --- | --- |
| Auth | /login | POST | { username, password } | Authentifie l’utilisateur et émet un **access token** | Tout le monde |
| Auth | /logout | POST | (token dans Authorization) | Révoque le token actif | Utilisateur connecté |
| Auth | /register | POST |  | Inscription au site | Tout le monde |
| Authors | /authors | GET |  | Liste des auteurs | Utilisateur connecté |
| Authors | /authors | POST | { firstName, lastName } | Création (validator authorValidator) | Utilisateur connecté |
| Categories | /categories | GET |  | Liste des catégories | Tout le monde |
| Categories | /categories | POST | { label } | Création (validator categoryValidator) | Utilisateur connecté |
| Books | /books | GET | Voir sous le tableau | Retourne tous les livres | Tout le monde |
| Books | /books | POST | Voir sous le tableau | Création d’un livre (validator bookValidator). | Utilisateur connecté |
| Books | /books/:id | GET | Voir sous le tableau | Lecture d’un livre | Utilisateur connecté |
| Books | /books/:id | PUT/DELETE | Voir sous le tableau | Modification et suppression d’un livre | Utilisateur ayant ajouté le livre |
| Categories→Books | /categories/:category\_id/books | GET |  | Tous les livres d’une catégorie | Tout le monde |
| Users | /users/:id | GET |  | Retourne l’utilisateur + nombre d’ouvrages + dernier ouvrage crée | Utilisateur connecté |
| Users→Books | /users/:user\_id/books | GET |  | Tous les ouvrages créés par l’utilisateur | Propre utilisateur |
| Comments | /books/:book\_id/comments | GET |  | Liste des commentaires d’un livre | Utilisateur connecté |
| Comments | /books/:book\_id/comments | POST | { content, userId } | Crée un commentaire lié à book\_id | Utilisateur connecté |
| Evaluations | /books/:book\_id/evaluations | GET |  | Donne la moyenne des notes d’un livre | Utilisateur connecté |
| Evaluations | /books/:book\_id/evaluations | POST | { note (1..5), userId } | Crée une évaluation lié à book\_id | Utilisateur connecté |

|  |  |
| --- | --- |
| POST /books  PUT, DELETE /books/:id | GET /books |
| {  "title": "string (2..255)",  "categoryId": number (valide),  "authorId": number (valide),  "numberOfPages": number,  "pdfLink": "string (2..255)",  "editor": "string (2..255)",  "editionYear": number,  "abstract": "string (2..255)",  "imagePath": "string (2..255 | nullable)",  "userId": number  } | **Query** : page, limit, sort, order, categoryId, authorId, userId, search |

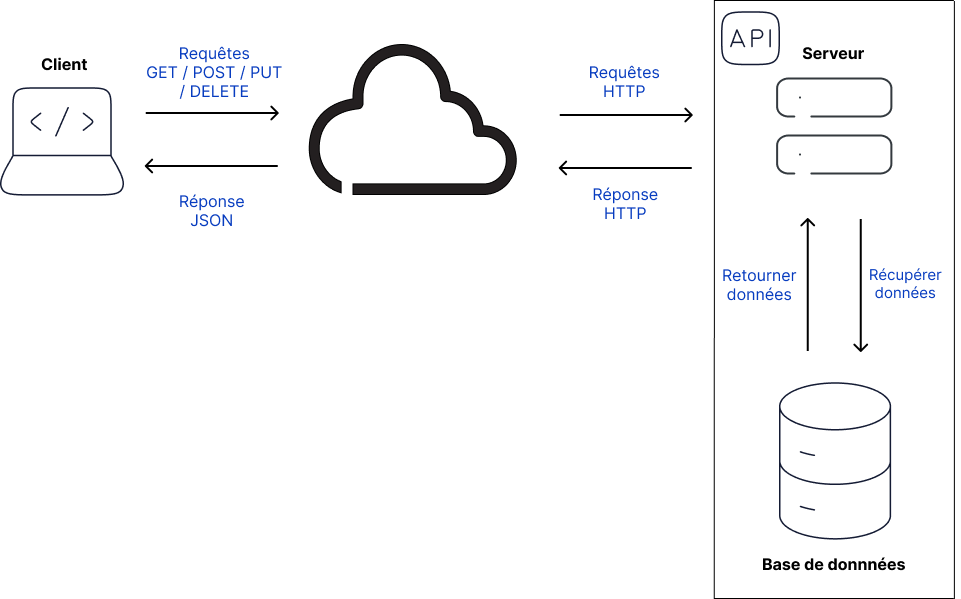
Chaque route est également accessible par l’admin.

Afin de bien réaliser ce projet, il a d’autres routes qui n’ont pas été implémentées car non obligatoires, mais qui sont très utiles pour la suite du projet lors de la réalisation de la partie visuel du site. Les voici :

| **Domaine** | **URI** | **Verbe** | **JSON** | **Description & Remarques** | **Accessibilité** |
| --- | --- | --- | --- | --- | --- |
| Books | /books ?limit=5 | GET |  | Permet de faire des recherches de livres par filtres dynamiquement (ici en limitant le nombre par exemple) | Tout le monde |
| Comments | /books/:book\_id/comments/:comment\_id | PUT | {content} | Modifier un commentaire | Utilisateur qui l’a créé |
| Comments | /books/:book\_id/comments/:comment\_id | DELETE |  | Supprimer un commentaire | Utilisateur qui l’a créé |
| Evaluations | /books/:book\_id/evaluations/:evaluation\_id | PUT |  | Modifier une note sur un livre | Utilisateur qui l’a créé |
| Authors | /authors/:author\_id | PUT | { firstName, lastName } | Modifier un auteur | Admin |
| Authors | /authors/:author\_id | DELETE |  | Supprimer un auteur | Admin |
| Category | /category/:category\_id | PUT | { label } | Modifier une catégorie | Admin |
| Category | /category/:category\_id | DELETE |  | Supprimer une catégorie | Admin |
| Users | /users/:id | DELETE |  | Supprimer un compte | Utilisateur qui l’a créé |
| Users | /users | GET |  | Voir tous les utilisateurs | Admin |

## Schéma de l'architecture

Le schéma ci-dessous illustre l'architecture de l'application **Passion Lecture**, en se concentrant sur les interactions entre le **frontend** (qui sera développé ultérieurement) et le **backend** (l'API REST).



*Description de l'architecture*

L'architecture est basée sur une séparation claire des responsabilités entre le client (frontend) et le serveur (backend).

*Frontend (Côté Client)*

Il est représenté par le client web(ordinateur). Le frontend envoie des requêtes HTTP (utilisant les verbes GET, POST, PUT, DELETE, etc.) pour interagir avec l'API REST. En retour, il reçoit des Réponses JSON contenant les données demandées ou le statut de l'opération.

*API REST (Backend)*

L’API est le cœur du système. Elle a été développée avec le framework AdonisJS6. L'API reçoit les requêtes HTTP du client et agit comme point d'entrée central.

On retrouve la base de données dans cette structure. L'API interagit avec la base de données pour récupérer et retourner les données (livres, utilisateurs, commentaires, etc.). Cette interaction est facilitée par l'ORM Lucid d'AdonisJS.

Cette structure garantit la maintenabilité et la scalabilité du système, préparant l'API à alimenter efficacement le futur frontend web ou mobile, tout en assurant l'intégrité des données via l'ORM et les contraintes de la base.

# Réalisation

## Gestion de l’authentification

L’authentification constitue un élément central de la sécurité de l’API REST. Elle permet de s’assurer que seules les personnes autorisées peuvent effectuer certaines actions, notamment la modification ou la suppression de ressources sensibles.

L’API expose une route POST /login qui reçoit les identifiants de l’utilisateur sous la forme :

{

"username": "exemple",

"password": "motdepasse"

}

Ces identifiants sont vérifiés grâce au provider Lucid, en comparant le mot de passe fourni avec le hash scrypt stocké en base de données.

En cas de succès, l’API génère un access token, persistant dans la table auth\_access\_tokens. Ce token est ensuite renvoyé au client et doit être utilisé pour accéder aux routes protégées.

La route POST /logout permet de révoquer le token en cours, assurant ainsi qu’il ne peut plus être utilisé pour accéder aux ressources protégées.

Pour gérer les permissions, le modèle users expose un champ is\_admin permettant de distinguer les administrateurs des utilisateurs classiques.

La politique d’accès aux routes est disponibles dans la section sur l’[API REST](#_API_REST).

L’implémentation reste simple et efficace : les routes nécessitant un rôle particulier sont protégées par le middleware d’authentification (auth), et la vérification des droits se fait via auth.user!.is\_admin directement dans les actions correspondantes.

## Mesures de sécurités

Dans ce chapitre, nous présentons les principales mesures mises en place pour garantir la sécurité, la validation et l’intégrité des données au sein de l’API, assurant à la fois la protection des utilisateurs et la fiabilité des informations manipulées.

* Les mots de passe des utilisateurs sont stockés de manière sécurisée grâce au hashscrypt, via le providerLucidUser d’AdonisJS. Cette méthode garantit que les mots de passe ne sont jamais stockés en clair, protégeant ainsi les comptes même en cas de fuite de données.
* Toutes les données reçues par l’API sont validées avec VineJS, afin de garantir leur conformité :
  + Types : chaînes, nombres, booléens, etc.
  + Limites : longueurs minimales/maximales, valeurs numériques.
  + Listes autorisées : par exemple, catégories valides pour un livre.

Cette validation stricte empêche les erreurs et les injections de données incorrectes ou malveillantes.

* Pour uniformiser la communication entre backend et frontend, toutes les réponses de l’API sont envoyées en JSON. Cela est assuré par un middleware dédié, qui convertit ou rejette automatiquement tout type de réponse non conforme.
* Les relations entre les entités (livres, catégories, utilisateurs, évaluations) sont protégées par des clés étrangères (FK) avec l’option ON DELETE CASCADE.
  + Exemple : si un livre est supprimé, toutes les évaluations ou commentaires associés sont automatiquement supprimés, évitant les données orphelines.

Cette stratégie garantit que la base de données reste cohérente et fiable.

* Pour éviter que le même utilisateur n’évalue plusieurs fois le même livre, une contrainte unique est appliquée sur les colonnes (book\_id, user\_id) de la table evaluations.
  + Toute tentative de créer une évaluation en double renverra une erreur409Conflict, protégeant l’intégrité logique des données et la fiabilité des moyennes de notes.

## Fonctionnalités demandées

L’application backend développée consiste en une API REST complète, permettant à un consommateur (par exemple un frontend) d’interagir avec les données de l’application. Chaque fonctionnalité a été conçue afin de respecter les bonnes pratiques du développement web et d’offrir un service fiable et sécurisé.

**1. Gestion des routes**

L’API propose un ensemble de routes permettant de gérer les principales entités de l’application, telles que les livres, les catégories et les utilisateurs. Chaque entité dispose de routes permettant certaines opérations CRUD (Create, Read, Update, Delete). Les routes sont organisées de manière logique et documentées pour faciliter l’intégration côté frontend.

**2. Validation des données**

Toutes les données reçues par l’API sont validées afin de garantir leur conformité et d’éviter les erreurs côté serveur. Cette validation comprend la vérification des types, des formats, des valeurs autorisées et des contraintes spécifiques aux champs (par exemple, un titre de livre obligatoire ou un email au format valide).

**3. Gestion des statuts HTTP et des erreurs**

L’API utilise les codes de statut HTTP pour informer le consommateur du résultat des requêtes :

* **2xx** pour indiquer le succès (ex. 200 OK, 201 Created),
* **3xx** pour la redirection si nécessaire,
* **4xx** pour les erreurs côté client (ex. 400 Bad Request, 401 Unauthorized),
* **5xx** pour les erreurs côté serveur (ex. 500 Internal Server Error).

Un mécanisme de gestion des erreurs centralisé permet de renvoyer des messages clairs et cohérents, facilitant le débogage et l’intégration.

**4. Recherche sur les entités**

L’API offre des fonctionnalités de recherche sur les livres, les catégories et autres entités, permettant de filtrer les résultats selon différents critères (ex. titre, auteur, catégorie). Ces fonctionnalités améliorent l’expérience utilisateur et permettent une navigation efficace dans les données.

**5. Authentification basée sur les jetons OAT**

Pour sécuriser l’accès aux données, l’API utilise un système d’authentification par jetons OAT (Open Authorization Token). Chaque utilisateur doit s’authentifier pour obtenir un jeton valide, qui sera ensuite utilisé pour accéder aux routes protégées. Cela garantit que seules les personnes autorisées peuvent effectuer certaines actions.

**6. Gestion des règles CORS**

L’API est configurée pour gérer correctement les Cross-Origin Resource Sharing (CORS), permettant à des applications frontend hébergées sur des domaines différents de consommer l’API en toute sécurité. Les règles CORS définissent quels domaines sont autorisés à accéder à l’API et quels types de requêtes sont acceptés.

## Eco-Conception

L’éco-conception de l’API repose sur plusieurs principes visant à réduire la consommation de ressources serveur et réseau tout en garantissant la fiabilité et la sécurité des données. L’un des aspects les plus importants est la validation stricte des données. Grâce à VineJS, toutes les données reçues par l’API sont vérifiées avant traitement, ce qui permet d’éviter les calculs et opérations inutiles sur des informations mal formées ou incorrectes. Cette approche réduit la charge du serveur et limite l’utilisation de la mémoire et du CPU, tout en protégeant l’intégrité des données.

En complément, l’API limite les volumes de données envoyés aux clients grâce à la pagination et aux filtres, évitant ainsi l’envoi de réponses trop volumineuses en une seule fois. Les requêtes SQL sont optimisées pour récupérer uniquement les colonnes nécessaires, ce qui réduit le transfert de données et la consommation de mémoire côté serveur. L’utilisation de contraintes uniques et de relations référentielles assure qu’aucune donnée inutile n’est stockée ou calculée, évitant les doublons et les traitements superflus.

La sécurité contribue également à cette éco-conception : les mots de passe sont hachés avec scrypt et l’authentification par jetons garantit que seules les requêtes autorisées déclenchent des traitements, limitant ainsi les opérations inutiles sur le serveur. Enfin, le code est structuré de manière réutilisable, avec des tests automatisés pour détecter rapidement les anomalies, ce qui réduit les déploiements répétés et minimise l’empreinte énergétique liée au développement et à l’exploitation.

Ainsi, l’ensemble de ces mesures permet à l’API de fonctionner de manière plus légère, efficace et respectueuse des ressources, tout en offrant aux utilisateurs une expérience fiable et sécurisée.

# Conclusion

## Organisation du groupe

L’organisation du groupe a été simple et rapide car habituer à travailler ensemble.

Un repo commun a été créer avec les bases du projet (node, adonisJS…)

Il a été ensuite complété avec tous les fichiers choisis par le groupe.

Ensuite chacun travaille sur un aspect du projet dans une branche secondaire. Quand une personne estime avoir terminé son travail, le groupe regarde ensemble la pull request et confirme les changements à l’unanimité avant de merge dans la branche main.

## Conclusion générale

Le projet s’est bien déroulé, nous avons pu réaliser toutes les fonctionnalités demandées. Ce projet nous a permis de bien progresser en backend et plus précisement en AdonisJS. Le travail de groupe a été un vrai plus, ça a permis de découvrir Github de manière partagée.

## Conclusion personnelle

* Brendan

Projet intéressant mais peu de séquences avec des sessions bien trop longues pour travailler correctement (3 jours), beaucoup de travail pour très peu de temps, un rapport détaillé à faire alors que le projet en lui-même prend déjà tout le temps restant.

A permis de mieux appréhender le début d’un projet en ayant une vision globale des choses à faire. Un bon aperçu de ce que peut-être le backend et son interaction avec le frontend.

Premier projet en groupe qui a permis de voir le fonctionnement de github et le pair progamming.

* Mina

J’ai beaucoup apprécié ce projet. C’est à mon avis un des projets les plus proches de ce que l’on pourrait faire en entreprise en tant que développeur d’application et cela me semble très enrichissant. Le fait de lier le frontend et le backend me semble très formateur, comme c’est le cas également pour l’aspect travail d’équipe et organisations.

Un petit bémol est à poser sur le temps alloué au projet face à la demande de fonctionnalité et l’ampleur du rapport. En effet, considérer le temps alloué, les demandes du rapport me paraissent trop conséquentes.

De manière générale, c’est de manière plutôt positive que j’ai abordé ce projet.

* Maël

J’ai bien aimé le projet, c’était intéressant d’avoir une vraie approche vers la construction d’une application. Le travail en groupe m’a permis de m’habituer aux différents mécanismes que je vais devoir utiliser dans mon métier, notamment avec l’utilisation de Github à plusieurs, avec la création de branche.

Après avoir eu un premier pas vers la construction d’une application, j’ai hâte de terminer ce projet avec la partie frontend dans le prochain module.

## Conclusion planification

Le travail d’équipe s’étant déroulé sans encombre, la planification a été simple et efficace. Les tâches ont été réparties équitablement en fonction des compétences et affinités de chacun. Ainsi, le temps consacré au projet a été efficace et bien répartit.

En conclusion, l'approche de planification basée sur GitHub Projects et une approche collaborative (peer programming, etc.) ont été très efficaces pour structurer le projet et gérer l'évolution du code, permettant de livrer un backend propre et fonctionnel.

# Utilisation de l’IA

L’IA a uniquement été utilisée pour la réalisation de ce rapport. Le rapport prenant beaucoup de temps, il a été décidé que l’intelligence artificielle aiderait grandement à la réalisation de ce dernier afin d’accélérer cette partie du projet.

Les différentes utilisations de l’IA sont assez simples, une petite partie de chapitre avec les axes principaux, est écrite à la main puis, une demande à ChatGPT d’approfondir le chapitre en lui donnant ce qui était demandé de base dans le livrable, afin d’avoir une structure plus claire et compréhensible. Cette approche IA a été utilisée pour les chapitres introduction, modélisation, structure du code, schéma de l’architecture, gestion de l’authentification, mesures de sécurités, fonctionnalités demandées, eco-conception.

Pour ce dernier chapitre, il a aussi été demandé à l’IA de donner tous les concepts d’eco-conception de AdonisJS et de créer un chapitre à partir d’un des concepts choisi.

Voilà un exemple de prompt utilisé :

*Approfondis ce chapitre : Comprend une explication de l'algorithme utilisé pour gérer l’authentification et la gestion des rôles (1 page)*

*Voila le deja fait :*

*3.1 Gestion de l’authentification*

*1. POST /login reçoit { username, password }.*

*2. Vérification des identifiants (hash scrypt) via le provider Lucid.*

*3. À la réussite : création d’un access token (persisté en DB auth\_access\_tokens) retourné au client.*

*4. POST /logout révoque le token courant.*

*Rôles et autorisations :*

*Le modèle users expose is\_admin.*

* *Ø Lecture publique (GET) ;*
* *Ø Écriture (POST/PATCH/DELETE) réservée aux authentifiés ;*
* *Opérations sensibles (p.ex. CRUD auteurs/catégories, suppression de livres) réservées aux admin.*

*Implémentation simple : garde auth + vérification auth.user!.is\_admin dans les actions nécessitant le rôle admin.*

# Webographie / Bibliographie / Glossaire

[Doc AdonisJS](https://docs.adonisjs.com/guides/preface/introduction)

[Doc Lucid](https://lucid.adonisjs.com/docs/introduction)

[Doc VineJS](https://vinejs.dev/docs/introduction)

[Kanban du projet](https://github.com/users/Fbrend23/projects/9/views/1)

[Repo Github](https://eduvaud-my.sharepoint.com/personal/pl76tup_eduvaud_ch/Documents/github.com/Fbrend23/P_Web-295)

[ChatGPT](https://chatgpt.com/)

1. [github](https://github.com/Fbrend23/P_Web-295) [↑](#footnote-ref-2)