# Projet Adonis : Passion Lecture

Lien du repos <https://github.com/Meronada749/P_Web_C295-M-R-C>



Table des matières

[Projet Adonis : Passion Lecture 1](#_Toc212801739)

[Introduction 3](#_Toc212801740)

[Analyse 3](#_Toc212801741)

[Planification des tâches 3](#_Toc212801742)

[Analyse de l’API REST 4](#_Toc212801743)

[Conception des listes de routes 7](#_Toc212801744)

[Base de données à réaliser 10](#_Toc212801745)

[Structure du code 11](#_Toc212801746)

[Schéma api rest 12](#_Toc212801747)

[Réalisation 13](#_Toc212801748)

[Algorithme de gestion de l’authentification 13](#_Toc212801749)

[Sécurité 13](#_Toc212801750)

[Éco-conception 14](#_Toc212801751)

[Conclusion 15](#_Toc212801752)

[Conclusion personnelle 15](#_Toc212801753)

[Critique 15](#_Toc212801754)

[Webographie / Bibliographique / Glossaire 16](#_Toc212801755)

## Introduction

Le projet Passion Lecture a été développé dans le cadre d’un apprentissage autour du framework AdonisJS, avec pour objectif de concevoir une API REST complète et fonctionnelle dédiée à la gestion d’une plateforme de lecture en ligne. L’idée centrale du projet était de permettre aux utilisateurs d’explorer, d’ajouter et de commenter des livres, tout en offrant aux administrateurs la possibilité de gérer les auteurs, les catégories et les contenus associés. Cette API sert ainsi de fondation à une éventuelle interface frontend, qui pourrait consommer ses données via des requêtes HTTP, en respectant les principes d’une architecture RESTful.

Dès le début du projet, nous avons cherché à concevoir une structure claire, maintenable et conforme aux bonnes pratiques du développement backend. L’organisation du code s’appuie sur la séparation des responsabilités entre les contrôleurs, les modèles et les validateurs, garantissant une meilleure lisibilité et une évolutivité du système. Les routes sont définies de manière cohérente et hiérarchisée, notamment avec des relations imbriquées comme celles entre les livres et leurs commentaires. Cette approche a permis de mettre en place une API structurée, capable de gérer les opérations CRUD de manière sécurisée et efficace.

Une attention particulière a également été portée à la sécurité et à la fiabilité du système, notamment grâce à la gestion de l’authentification et des rôles utilisateurs, ainsi qu’à la validation stricte des données à l’aide des outils intégrés à AdonisJS, comme VineJS. Ces mécanismes permettent de prévenir les erreurs et les failles potentielles, tout en assurant la cohérence des données échangées entre le client et le serveur.

## Analyse

### Planification des tâches

La planification du projet Passion Lecture a été réalisée à l’aide de GitHub Projects, en suivant la méthode Kanban. Cet outil nous a permis d’organiser le travail de manière claire et collaborative tout au long du développement. Le tableau Kanban était composé de plusieurs colonnes représentant les différentes étapes du cycle de production : “Backlog”, “Ready”, “In Progress”, “In Review” et “Done”. Ce fonctionnement a facilité la visualisation de l’avancement du projet, depuis la création d’une tâche jusqu’à sa validation finale.

Chaque fonctionnalité ou élément à développer faisait l’objet d’une issue sur GitHub, décrivant précisément la tâche à accomplir, les objectifs attendus et les critères de réussite. Ces issues étaient ensuite assignées aux membres de l’équipe selon leurs rôles et compétences, afin d’assurer une répartition équilibrée du travail. Dès la création des tâches, nous avons pris soin de les découper de manière cohérente, en veillant à ce qu’aucune ne soit trop importante ni trop minime. Cette approche a permis de maintenir un rythme de progression régulier et d’éviter la surcharge ou la fragmentation du travail.

Tout au long du projet, nous mettions régulièrement à jour le tableau Kanban afin de suivre l’évolution de chaque tâche. Lors des séances de travail, les issues passaient d’une colonne à l’autre selon leur état d’avancement : prêtes à être développées, en cours de réalisation, en phase de relecture ou terminées. Ce processus a favorisé la communication, la transparence et la coordination entre les membres du groupe, tout en permettant d’identifier rapidement les points de blocage éventuels.

L’utilisation du Kanban sur GitHub s’est donc révélée essentielle pour la gestion du projet. Elle nous a offert une vision claire et partagée de l’ensemble des étapes de développement, facilitant à la fois la planification, le suivi et la réussite du projet Passion Lecture.

### Analyse de l’API REST

L’API REST a été conçue selon une architecture claire et standardisée, en respectant les principes RESTful. Elle expose plusieurs ressources principales : books, comments, writers et categories. Chaque ressource est gérée par un contrôleur dédié et suit la convention CRUD (Create, Read, Update, Delete) à l’aide de la méthode router.resource() d’AdonisJS.

#### 1. Ressource : Books

Contrôleur : BooksController  
URI racine : /books  
Méthodes disponibles :

|  |  |  |  |
| --- | --- | --- | --- |
| Verbe HTTP | URI | Description | Corps attendu / Réponse |
| GET | /books | Récupère la liste de tous les livres | – |
| GET | /books/:id | Récupère un livre spécifique via son ID | – |
| POST | /books | Crée un nouveau livre | { "title": "string", "writer\_id": "number", "category\_id": "number", "description": "string" } |
| PUT/PATCH | /books/:id | Met à jour un livre existant | Même structure que POST |
| DELETE | /books/:id | Supprime un livre existant | – |

#### 2. Ressource : Comments (imbriquée sous Books)

Contrôleur : CommentsController  
URI racine : /books/:book\_id/comments  
Méthodes disponibles :

|  |  |  |  |
| --- | --- | --- | --- |
| Verbe HTTP | URI | Description | Corps attendu / Réponse |
| GET | /books/:book\_id/comments | Liste tous les commentaires liés à un livre | – |
| GET | /books/:book\_id/comments/:id | Récupère un commentaire spécifique | – |
| POST | /books/:book\_id/comments | Ajoute un commentaire à un livre | { "content": "string", "user\_id": "number" } |
| PUT/PATCH | /books/:book\_id/comments/:id | Met à jour un commentaire existant | { "content": "string" } |
| DELETE | /books/:book\_id/comments/:id | Supprime un commentaire | – |

Les routes des commentaires sont imbriquées afin de refléter la relation hiérarchique *Book → Comments*.

#### 3. Ressource : Writers

Contrôleur : WritersController  
URI racine : /writers  
Méthodes disponibles :

|  |  |  |  |
| --- | --- | --- | --- |
| Verbe HTTP | URI | Description | Corps attendu / Réponse |
| GET | /writers | Liste tous les auteurs | – |
| GET | /writers/:id | Récupère un auteur via son ID | – |
| POST | /writers | Crée un nouvel auteur | { "name": "string", "bio": "string" } |
| PUT/PATCH | /writers/:id | Met à jour un auteur existant | Même structure que POST |
| DELETE | /writers/:id | Supprime un auteur | – |

#### 4. Ressource : Categories

Contrôleur : CategoriesController  
URI racine : /categories  
Méthodes disponibles :

|  |  |  |  |
| --- | --- | --- | --- |
| Verbe HTTP | URI | Description | Corps attendu / Réponse |
| GET | /categories | Liste toutes les catégories | – |
| GET | /categories/:id | Récupère une catégorie spécifique | – |
| POST | /categories | Crée une nouvelle catégorie | { "label": "string" } |
| PUT/PATCH | /categories/:id | Met à jour une catégorie existante | { "label": "string" } |
| DELETE | /categories/:id | Supprime une catégorie | – |

#### 5. Architecture et conventions

* L’API respecte le modèle RESTful avec des routes explicites et cohérentes.
* Chaque ressource dispose d’un contrôleur dédié pour séparer la logique métier.
* Les routes imbriquées (comme les *comments* dans *books*) facilitent la gestion des relations entre entités.
* Les échanges s’effectuent en JSON, tant pour les requêtes que pour les réponses.
* L’API est prévue pour être consommée par un frontend via des appels HTTP (ex. Axios ou Fetch API).

### Conception des listes de routes

Authentification

|  |  |  |  |
| --- | --- | --- | --- |
| Méthode | URI | Description | Accès |
| POST | /auth/register | Créer un compte utilisateur | Public |
| POST | /auth/login | Connexion, retourne un token | Public |
| POST | /auth/logout | Déconnexion de l’utilisateur | Authentifié |
| GET | /auth/me | Récupérer les infos de l’utilisateur connecté | Authentifié |

#### Utilisateurs

|  |  |  |  |
| --- | --- | --- | --- |
| Méthode | URI | Description | Accès |
| GET | /users | Liste de tous les utilisateurs | Admin |
| GET | /users/:id | Détails d’un utilisateur | Authentifié |
| PUT | /users/:id | Modifier les infos de l’utilisateur | Propriétaire ou Admin |
| DELETE | /users/:id | Supprimer un utilisateur | Admin |

#### Catégories

|  |  |  |  |
| --- | --- | --- | --- |
| Méthode | URI | Description | Accès |
| GET | /categories | Liste des catégories | Public |
| GET | /categories/:id | Détails d’une catégorie | Public |
| POST | /categories | Ajouter une catégorie | Admin |
| PUT | /categories/:id | Modifier une catégorie | Admin |
| DELETE | /categories/:id | Supprimer une catégorie | Admin |

#### Livres

|  |  |  |  |
| --- | --- | --- | --- |
| Méthode | URI | Description | Accès |
| GET | /books | Liste de tous les livres | Public |
| GET | /books/latest | Récupère les 5 derniers livres ajoutés | Public |
| GET | /books/:id | Détails d’un livre (auteur, résumé, notes, etc.) | Public |
| GET | /books/category/:categoryId | Liste des livres d’une catégorie | Public |
| POST | /books | Ajouter un livre | Authentifié |
| PUT | /books/:id | Modifier un livre (si propriétaire ou admin) | Authentifié |
| DELETE | /books/:id | Supprimer un livre (si propriétaire ou admin) | Authentifié |
| GET | /books/search?q= | Recherche d’un livre par titre, auteur, ou catégorie | Public |

#### Appréciations et Commentaires

|  |  |  |  |
| --- | --- | --- | --- |
| Méthode | URI | Description | Accès |
| POST | /books/:id/ratings | Ajouter une note à un livre | Authentifié |
| GET | /books/:id/ratings | Voir les notes d’un livre | Public |
| POST | /books/:id/comments | Ajouter un commentaire à un livre | Authentifié |
| GET | /books/:id/comments | Voir les commentaires d’un livre | Public |
| DELETE | /comments/:id | Supprimer un commentaire (propriétaire ou admin) | Authentifié |

#### Administration

|  |  |  |  |
| --- | --- | --- | --- |
| Méthode | URI | Description | Accès |
| GET | /admin/dashboard | Statistiques générales (nombre de livres, utilisateurs, commentaires) | Admin |

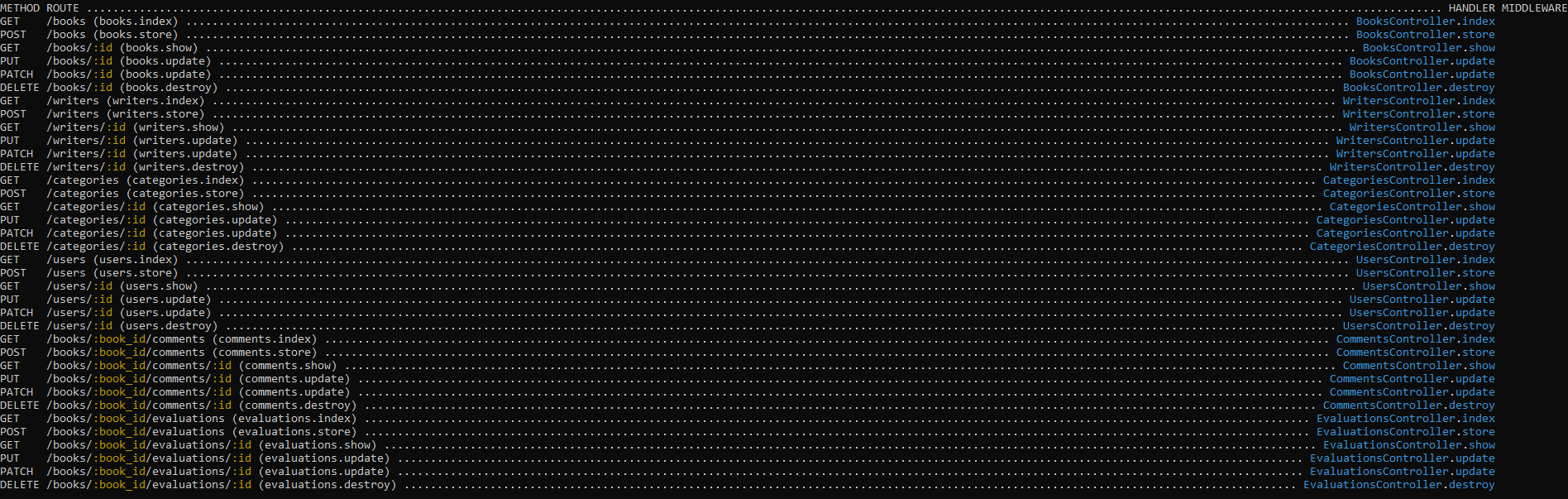
#### Gestion des erreurs & validation

Tous les formulaires sont validés (ex. : champs obligatoires, formats, longueurs).

Retour des statuts HTTP appropriés :

* 200 OK / 201 Created / 204 No Content
* 400 Bad Request
* 401 Unauthorized
* 403 Forbidden
* 404 Not Found
* 500 Internal Server Error

#### List Routes

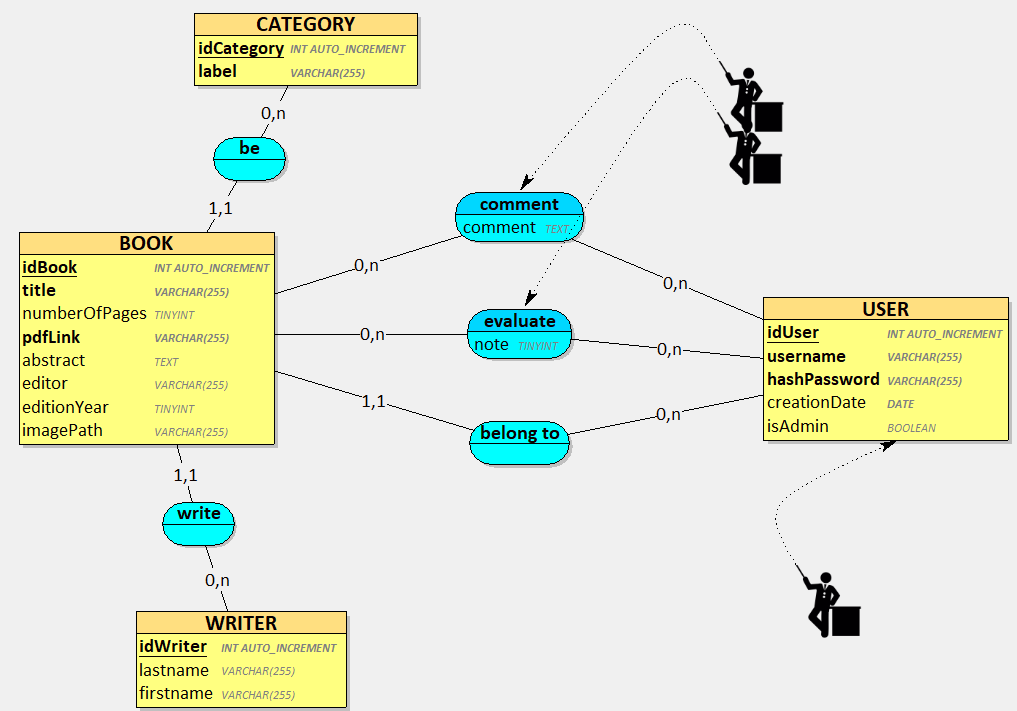
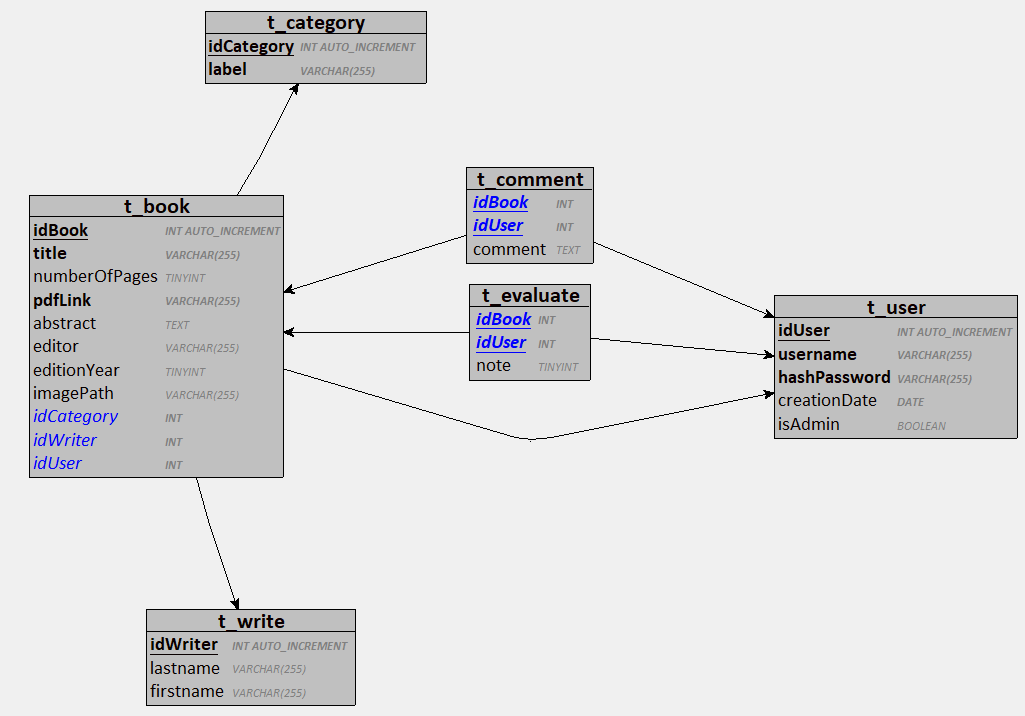
Category validator

Sur le MCD, le champ label est de type string et peut contenir jusqu’à 255 caractères, sans contrainte minimale. Afin d’assurer la cohérence des données, il est pertinent de définir une longueur minimale — 2 caractères semble être un choix judicieux pour un label, afin d’éviter la validation d’entrées trop courtes ou non significatives.

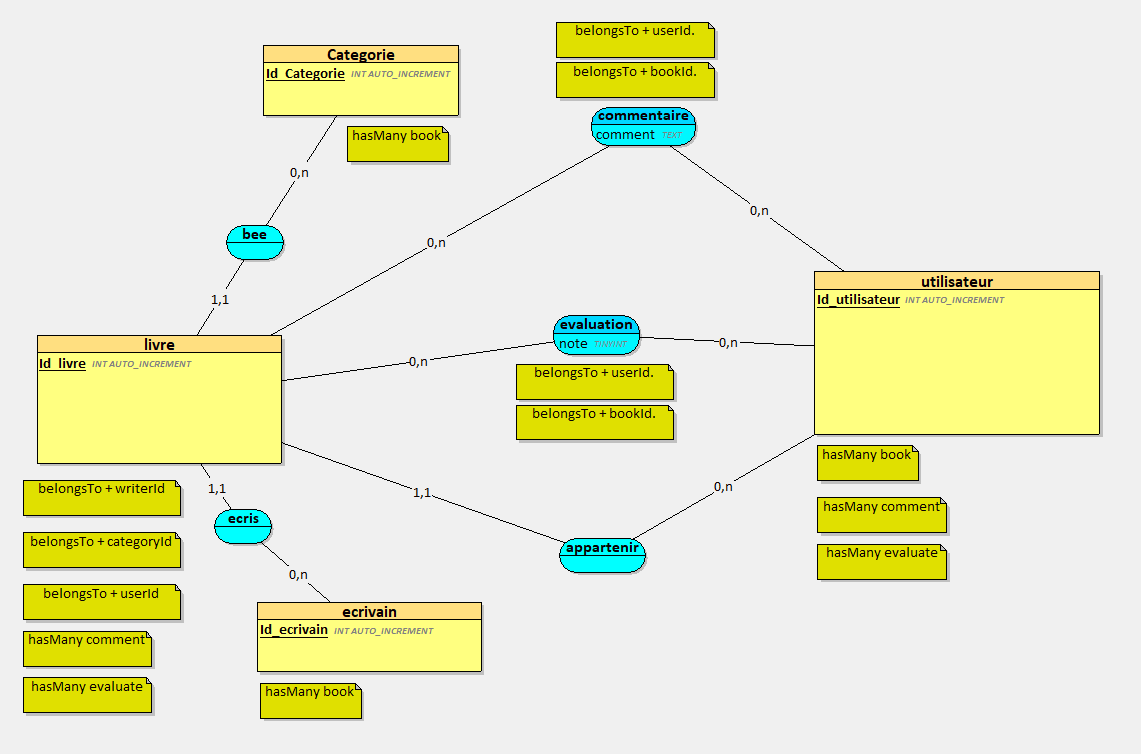
#### Writer validator

Sur le MCD, les champs nom et prénom sont de type string et peuvent contenir jusqu’à 255 caractères, sans contrainte minimale. Afin d’assurer la cohérence et la validité des données, il est recommandé de définir une longueur minimale. Une valeur de 2 caractères semble appropriée, puisqu’il n’existe généralement ni nom ni prénom plus court. Cela permet d’éviter la validation d’entrées trop brèves ou non significatives.

### Base de données à réaliser

La MCD/MLD suivant a été fourni :

Après analyse nous avons trouvé les appartenances suivantes :

 Après une analyse approfondie du modèle conceptuel de données (MCD) et de son passage au modèle logique de données (MLD), nous avons pu identifier avec précision les appartenances entre les différentes entités. Cette étape a été particulièrement utile pour clarifier les relations entre les tables ainsi que pour repérer les clés étrangères nécessaires.

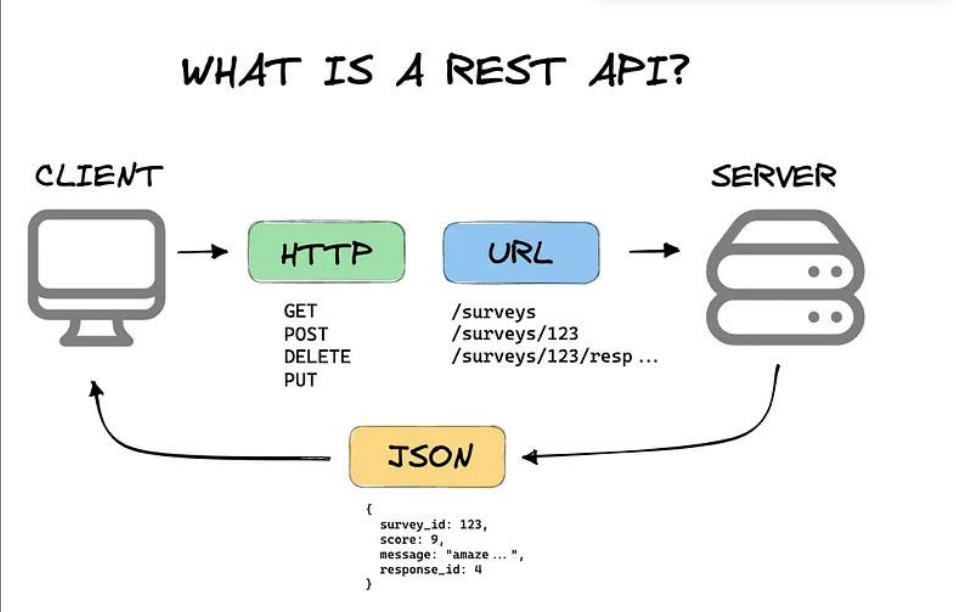
Grâce à cette démarche, nous avons non seulement amélioré notre compréhension globale de la structure de la base de données, mais nous avons également optimisé la phase de développement avec AdonisJS. En effet, la connaissance préalable des relations entre les entités nous a permis de définir rapidement et correctement les associations via les méthodes belongsTo et autres relations du framework. Cela nous a fait gagner un temps considérable, tout en réduisant les risques d’erreurs liées aux jointures et aux relations entre tables.

En résumé, cette analyse a joué un rôle clé dans la mise en place d’une architecture claire et cohérente, facilitant la transition entre la conception théorique et la réalisation pratique avec AdonisJS.

### Structure du code

|  |  |
| --- | --- |
| Dossier / Fichier | Rôle principal |
| /app/controllers | Contient les contrôleurs, qui gèrent la logique métier et les interactions entre le client (frontend) et les modèles. Chaque contrôleur correspond généralement à une ressource (ex. BooksController, WritersController, etc.). |
| /app/validators | Contient les validateurs, utilisés pour contrôler la validité des données reçues dans les requêtes HTTP (ex. vérifier qu’un champ est obligatoire, qu’une chaîne ne dépasse pas une longueur, etc.). Cela permet d’assurer la cohérence et la sécurité des données avant leur traitement. |
| /app/models | Contient les modèles représentant les entités de la base de données (ex. Book, Writer, Category). Ces modèles utilisent l’ORM Lucid d’AdonisJS pour interagir avec la base (lecture, écriture, relations…). |
| /database/migrations | Contient les migrations qui décrivent la structure des tables de la base de données. Chaque fichier de migration définit les colonnes, les types et les relations entre les tables. |
| /database/seeders | Permet de définir des jeux de données de test insérés automatiquement dans la base, utiles pour le développement ou les démonstrations. |
| /start/routes.ts | Fichier principal de définition des routes API. Il associe chaque URI et verbe HTTP à la méthode d’un contrôleur correspondant. |
| /config | Contient les fichiers de configuration du projet (base de données, application, sécurité, etc.). |
| /public | Contient les fichiers statiques accessibles directement (images, fichiers CSS/JS, etc.). |
| /resources/views | Utilisé si le projet comprend des vues côté serveur (non essentiel dans une API pure). |
| .env | Fichier d’environnement contenant les variables sensibles (configuration de la base de données, clé d’application, port du serveur…). |

### Schéma api rest

Ce schéma de l’architecture représente les interactions entre le frontend et le backend tout en situant les différents composants.

## Réalisation

### Algorithme de gestion de l’authentification

**Algorithme de connexion (login)**

1. **Validation des données :**

* L’utilisateur envoie username et password.
* Le loginValidator vérifie :
* que le username existe,
* que le mot de passe respecte certaines règles (ex. longueur minimale).

1. **Vérification des identifiants :**

* User.verifyCredentials(username, password) compare le mot de passe fourni avec le mot de passe stocké (généralement hashé avec bcrypt ou argon2).
* Si l’identifiant ou le mot de passe est incorrect, une erreur est renvoyée.

1. Génération du token d’accès :

* Si l’utilisateur est validé, un token JWT ou token d’accès est généré (User.accessTokens.create(user)).
* Ce token permet d’authentifier les requêtes futures sans renvoyer le mot de passe.

Réponse :

* Le serveur renvoie au client :
* le token d’accès,
* et les informations de l’utilisateur (nom, email, rôle, etc.).

**Algorithme de déconnexion (logout)**

* Récupérer l’utilisateur authentifié (auth.getUserOrFail()).
* Identifier le token utilisé pour cette session (auth.user?.currentAccessToken.identifier).
* Supprimer ce token de la base (User.accessTokens.delete(user, token)).
* Le token devient invalide, l’utilisateur est déconnecté.

### Sécurité

La sécurité est un aspect essentiel dans le développement d’une API REST. Le framework AdonisJS intègre nativement plusieurs mécanismes et bonnes pratiques pour protéger les données et assurer la fiabilité du système.

Dans ce projet, plusieurs mesures ont été prises à différents niveaux : validation des données, authentification, configuration du serveur et gestion des erreurs.

L’une des premières lignes de défense repose sur la validation systématique des entrées utilisateur à l’aide des validateurs AdonisJS.

* Chaque requête (POST, PUT, PATCH) est associée à un validateur qui définit les règles de type et de contrainte.
* Par exemple, un champ *string* est restreint à une longueur précise (minLength, maxLength) et ne peut pas contenir d’autres types de données.
* Les champs numériques, dates ou identifiants sont également validés avant d’être insérés en base.

AdonisJS utilise Lucid ORM, un ORM sécurisé qui construit automatiquement les requêtes SQL.

Cela permet d’éviter les injections SQL, car les valeurs sont toujours transmises sous forme de paramètres préparés, et jamais concaténées directement dans les requêtes.

Sécurité contre les injections XSS  
Validation et filtrage des entrées utilisateur.

Tous les champs textuels (ex. title, description, content d’un commentaire) sont validés et restreints via les validateurs d’AdonisJS.

Les champs de type string ont une longueur maximale et ne doivent contenir que des caractères autorisés.

Cette validation empêche l’insertion directe de balises ou de scripts malveillants dans la base de données.

## Éco-conception

Pour ce projet, nous avons également réfléchi à l’éco-conception de notre API REST. Même si ce n’était pas notre objectif principal, nous avons essayé de limiter les ressources utilisées et de rendre le système plus léger. Par exemple, nous avons structuré nos routes et nos contrôleurs de façon claire pour éviter les traitements inutiles et réduire le nombre d’appels à la base de données. L’ORM Lucid nous a permis de charger seulement les données nécessaires grâce aux relations entre les tables, ce qui évite de récupérer des informations superflues. Nous avons également organisé le code et les modules pour ne charger que ce qui est utile, ce qui réduit un peu la charge sur le serveur. Nous avons donc lors de la réalisation de ce projet, fait attention à l’efficacité et à la consommation des ressources dans le développement backend avec AdonisJS.

## Conclusion

Tout au long du projet Passion Lecture, notre groupe s’est organisé autour de Git et GitHub pour assurer la gestion du code et le suivi de l’avancement. Nous avons rapidement compris l’importance de la communication et de la rigueur dans un travail collaboratif. Pour éviter les conflits et les pertes de code, nous nous sommes assurés de toujours effectuer un pull avant chaque push, en vérifiant que chacun travaillait sur la version la plus à jour du projet. Cette méthode, simple mais efficace, nous a permis de progresser sans trop de difficultés et d’apprendre à utiliser Git de manière concrète. L’utilisation de kanban sur GitHub Projects nous a également aidés à suivre les tâches à réaliser et à garder une vision claire de ce qui restait à faire, ce qui a renforcé la coordination entre les membres de l’équipe.

Sur le plan technique, Passion Lecture a été pour nous une véritable initiation au développement backend et à la création d’une API REST avec AdonisJS. La plupart d’entre nous découvraient le framework et les concepts liés à la structure d’une API : contrôleurs, modèles, routes et validateurs. Ce projet nous a donc permis d’apprendre en pratiquant, de comprendre comment organiser le code pour qu’il soit clair et réutilisable, et de découvrir les bases de la sécurité avec la validation des données et l’authentification. Nous avons aussi mieux compris comment une base de données interagit avec un serveur à travers un ORM comme Lucid. Même si certaines parties ont demandé du temps et de la recherche, cette expérience nous a beaucoup aidés à progresser collectivement et à mieux comprendre le fonctionnement global d’un backend moderne.

### Conclusion personnelle

Ce projet m’a permis de mieux comprendre le fonctionnement d’une API REST et d’approfondir mes compétences en développement backend avec AdonisJS. J’ai particulièrement apprécié la collaboration au sein du groupe, qui m’a aidé à progresser dans la gestion de projet et l’utilisation de Git. Malgré quelques difficultés d’organisation, ce travail m’a permis de renforcer ma méthodologie et ma capacité à résoudre des problèmes techniques de manière autonome.

### Critique

Tout au long du projet, nous avons bénéficié d’un bon encadrement et d’un suivi régulier, ce qui nous a permis de progresser et de comprendre les concepts malgré notre niveau débutant. Les consignes et l’accompagnement ont été précieux pour nous guider dans la réalisation technique et dans l’organisation du travail. Cependant, certaines parties du projet restaient un peu complexes pour notre niveau, ce qui a parfois nécessité plus de temps pour assimiler et appliquer correctement les notions.

Un autre point à mentionner concerne le rapport écrit. Bien que la documentation détaillée soit importante pour structurer le projet et expliquer nos choix, le volume exigé était très important, ce qui a considérablement augmenté la charge de travail rédactionnel. Cette exigence a eu pour effet de réduire le temps disponible pour le développement pratique et la programmation, et a parfois rendu difficile la conciliation entre rédaction et réalisation technique. Une approche plus équilibrée, avec un rapport moins volumineux, aurait permis de consacrer davantage de temps aux aspects pratiques et à l’expérimentation.

Dans l’ensemble, malgré ces contraintes, le projet a été formateur et nous a appris à mieux gérer notre temps, à organiser notre travail et à progresser techniquement tout en respectant un cadre imposé.

## Webographie / Bibliographique / Glossaire

* AdonisJS : <https://docs.adonisjs.com/guides/preface/introduction>
* VineJS : <https://vinejs.dev/docs/introduction>
* ORM Lucid : <https://lucid.adonisjs.com/docs/introduction>
* Support de cours
* Recours à chatgpt pour l’éco-conception, prompt:
  + “en développement backend, quels sont les points d'éco-conceptions possible pour les api rest”