# 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](#_Toc212792034)

[Introduction 3](#_Toc212792035)

[Étapes de développement 3](#_Toc212792036)

[Conception de liste de routes 4](#_Toc212792037)

[List Routes 6](#_Toc212792038)

[Writer validator 6](#_Toc212792039)

[Sécurité contre les injections XSS 11](#_Toc212792040)

[Conclusion 11](#_Toc212792041)

[Conclusion personnelle 11](#_Toc212792042)

[Webographie / Bibliographique / Glossaire 12](#_Toc212792043)

## Introduction

## Étapes de développement

* Introduction

Comprend une brève explication du projet (½ page)

* Analyse

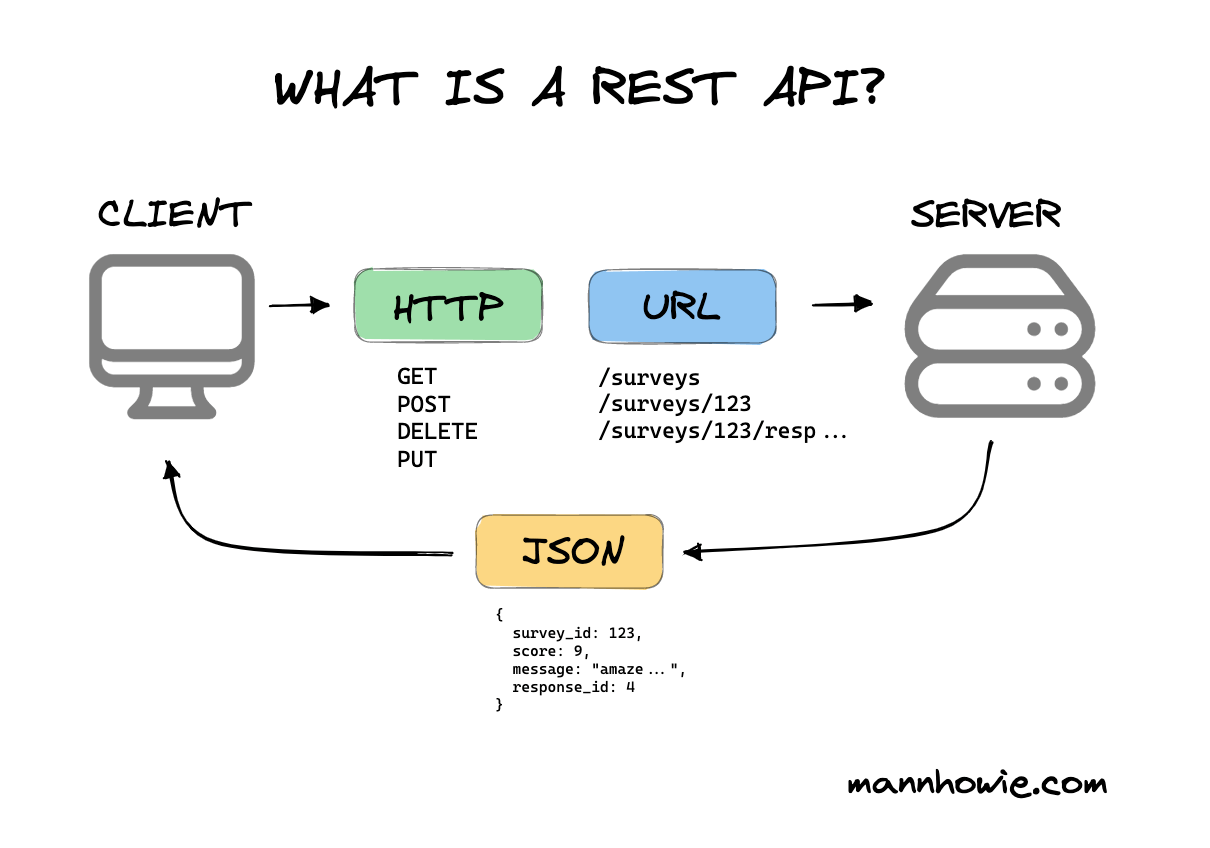
Planification de toutes les tâches à réaliser : Vous devez utiliser un outil permettant au chef de projet de voir l’évolution des tâches (github projet ou trelllo).

Contiendra une analyse de l’API REST présentant pour chaque route :

Le verbe http, l’URI et si nécessaire le json ou autre information à transmettre (1 page)

Contiendra une analyse de de la base de données à réaliser (MCD, MLD, MPD) (1 page)

Contiendra une analyse de la structure du code qui sera effectuée (Organisation du code, …) (1 page)

Schéma de l’architecture représentant les interactions entre le frontend et le backend tout en situant les différents composants (API REST, la base de données, l’ORM, etc).

### Conception de liste 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 |

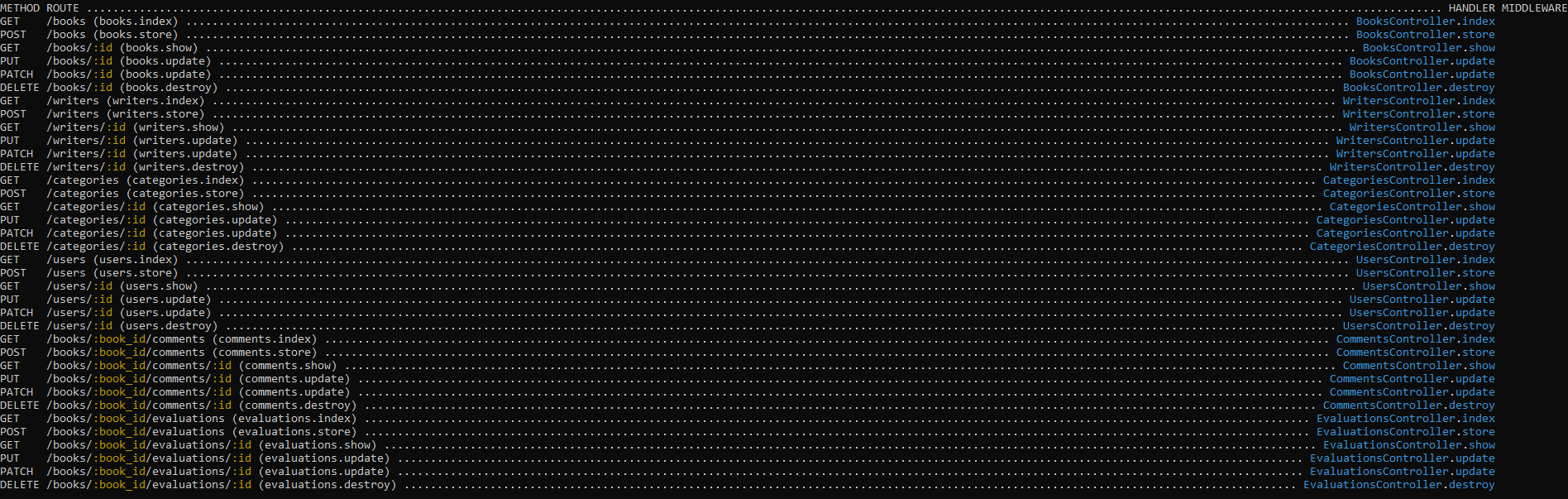
#### Gestion des erreurs & validation

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.

|  |  |
| --- | --- |
| **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…). |

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).
* Réalisation

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

Comprend une explication des mesures prises pour les aspects de sécurité. (1 page)

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.

### Conclusion

L’organisation du groupe s’est appuyée sur Git et GitHub Project, où nous avons utilisé la méthode Kanban pour planifier et suivre l’avancement des tâches. Chaque fonctionnalité était représentée par une carte dans le tableau, ce qui nous permettait de visualiser clairement les étapes “À faire”, “En cours” et “Terminées”. Cette organisation a favorisé une meilleure répartition du travail et une collaboration fluide entre les membres du groupe.

Le projet Passion Lecture a permis de concevoir une API REST complète et sécurisée avec AdonisJS. Nous avons appliqué les bonnes pratiques du développement backend : séparation du code en contrôleurs, modèles et validateurs, gestion des rôles et authentification sécurisée, ainsi que protection contre les failles XSS et SQL. Ce travail nous a permis d’approfondir nos compétences techniques et de comprendre l’importance d’une architecture claire et maintenable.

Sur le plan de la planification, même si la méthode Kanban a été très utile, certaines phases auraient pu être mieux équilibrées, notamment entre le développement et les tests. Une évaluation plus précise du temps nécessaire à chaque tâche aurait permis d’éviter les périodes de surcharge en fin de projet. Malgré cela, cette approche a grandement contribué à la réussite et à la clarté du projet.

### 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.

### 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