# Projet Adonis : Passion Lecture



## Introduction

## Étapes de développement

* Introduction

Comprend une brève explication du projet (½ page)[[1]](#footnote-1)

* 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).

### 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.

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 | – |

💡 **Remarque :** 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)

Comprend une explication sur chaque fonctionnalité technique demandée.

* Test

Comprend une explication des tests réalisés (test automatique) (1 page)

* Conclusion

Comprend une explication de l’organisation du groupe quant à la gestion du code avec git/github.

Comprend une conclusion générale sur le projet (½ page)

Comprend une conclusion personnelle sur le projet (½ page)

Comprend une critique constructive sur la planification du projet (½ page)

* Webographie / Bibliographique / Glossaire

1. Le nombre de page en () est le minimum à effectuer. [↑](#footnote-ref-1)