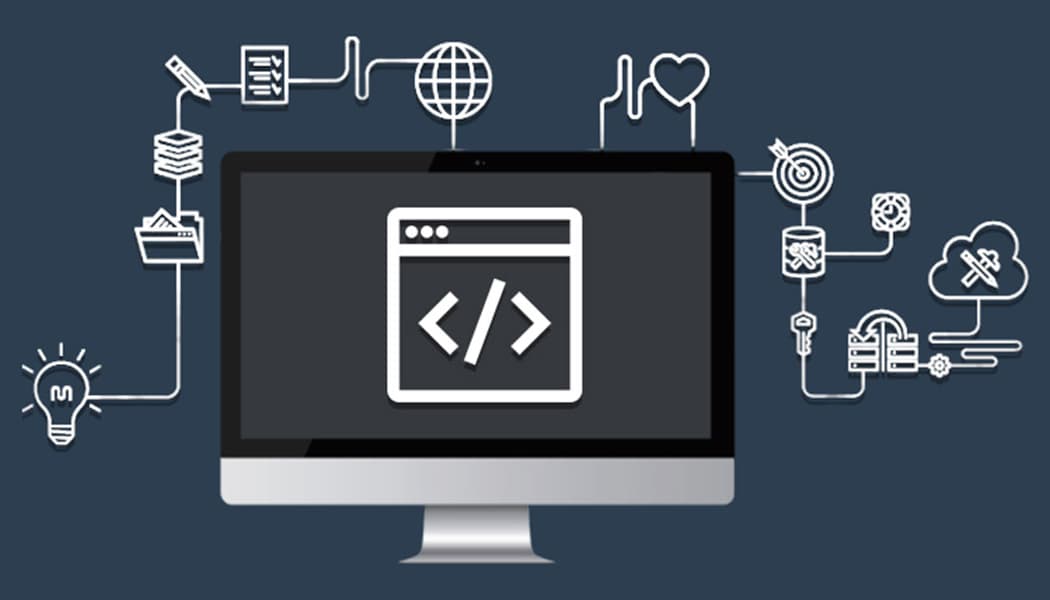
**P\_WEB295**



(Logo Backend)

Antoine Piguet  
Yosef Nademo  
Charles-Henri Moser  
 CID2B

Vennes - Lausanne

24 périodes

Chef de Projet – Gregory Charmier

Table des matières

[1 Introduction 3](#_Toc195003251)

[2 Analyse 4](#_Toc195003252)

[2.1 Planification – GitHub Project 4](#_Toc195003253)

[2.2 Routes – API REST 5](#_Toc195003254)

[2.3 Conception de la base de données 7](#_Toc195003255)

[3 Réalisation 8](#_Toc195003256)

[3.1 Algorithmes utilisés 8](#_Toc195003257)

[3.2 Maquettes – Figma 8](#_Toc195003258)

[3.3 Démarche d’éco-conception et Responsabilité Numérique 8](#_Toc195003259)

[4 Tests 9](#_Toc195003260)

[4.1 Dossier des tests 9](#_Toc195003261)

[5 Conclusion 9](#_Toc195003262)

[5.1 Conclusion générale 9](#_Toc195003263)

[5.2 Bilan personnel 9](#_Toc195003264)

[5.2.1 Conclusion Charles-Henri 9](#_Toc195003265)

[5.2.1 Conclusion Antoine Piguet 9](#_Toc195003266)

[5.2.2 Conclusion Yosef Nademo 9](#_Toc195003267)

[6 Divers 10](#_Toc195003268)

[6.1 Webographie/ Bibliographie 10](#_Toc195003269)

[7 Annexes 10](#_Toc195003270)

# Introduction

Le projet Passion Lecture a pour but de dévelpper un backend d'une application dédiée aux amateurs de lecture. Le projet est intégré au module 295 et devrait permettre aux utilisateurs de partager, voir et évaluer les livres. L'application sera basée sur une API REST sécurisée qui devrait permettre de gérer les livres, les catégories, les utilisateurs et les relations entre eux.

L’API devra intégrer une authentification sécurisée avec JWT, une validation rigoureuse des données, ainsi qu’une gestion des erreurs avec des statuts HTTP appropriés. Pour une expérience utilisateur fluide, une documentation détaillée sera générée via Swagger, et des tests seront réalisés avec Insomnia, Postman et des tests automatisés avec Vitest.

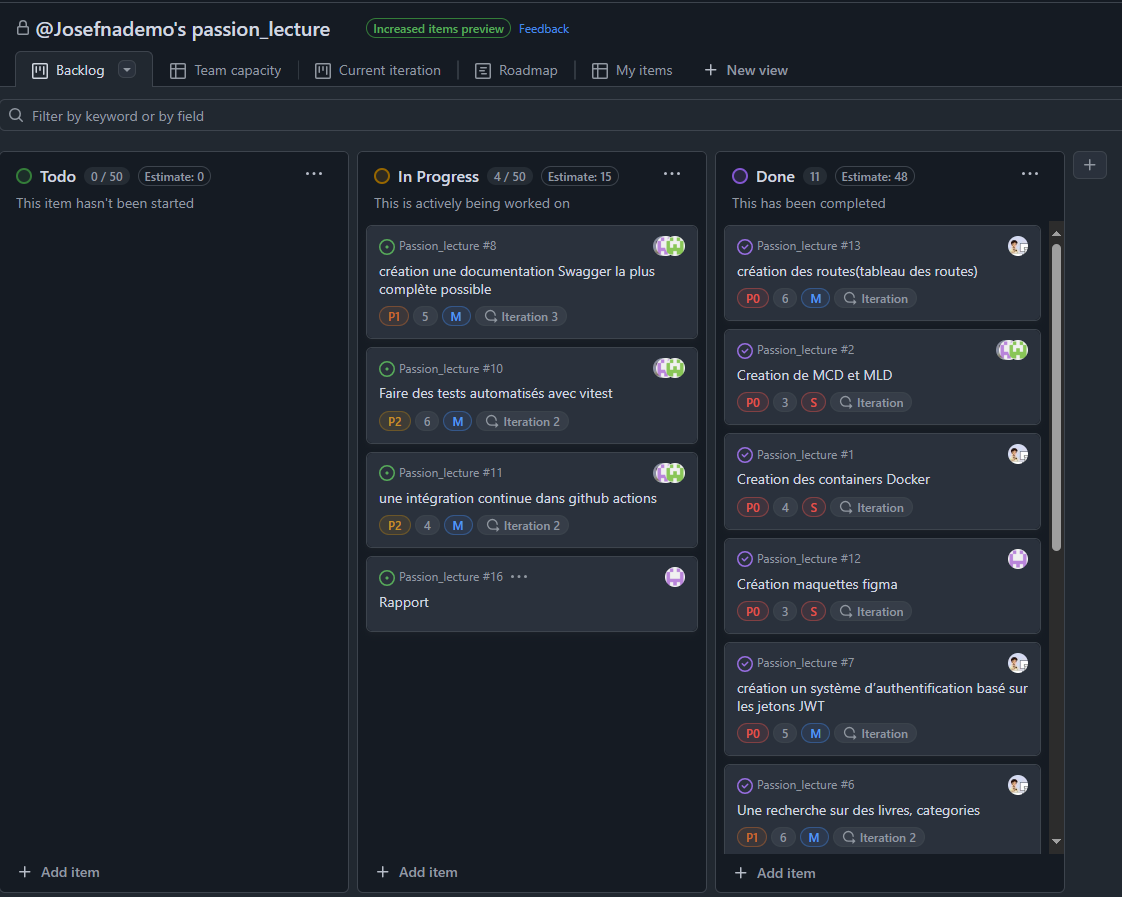
Le back-end sera développé avec Node.js, Express.js et Sequelize pour gérer une base de données MySQL. Le projet utilisera l'intégration continue avec GitHub Actions et la conteneurisation avec Docker pour assurer la portabilité. Finalement, le design écologique sera pris en compte, afin de proposer une application optimisée et respectueuse des normes de développement durable.

# Analyse

## Planification – GitHub Project

Durant le projet, nous avons utilisé GitHub Project afin d’organiser nos tâches et de se les répartir afin d’avoir une exécution plus efficace de notre projet.

Comme vous pouvez le voir sur cette image (ci-dessous), Nous avons utilisé le modèle de tableau Kanban avec trois colonnes : le backlog, c’est ici que nous avons mis toutes les tâches que nous devions accomplir, la colonne -In progress- ou nous mettons les tâches que nous sommes actuellement entrain de faire, et la colonne -Done-, ici nous updatons nos tâches afin de les définir comme finis AKA Done.



(Image tableau Kanban)

## Routes – API REST

|  |  |  |  |
| --- | --- | --- | --- |
| Action | Chemin | Json ? | Description |
| POST | /books/ | Toute la t\_livre + image | Ajoute un livre. |
| DELETE | /books/ :id/ | Non | Supprimer un livre. |
| PUT | /books/ :id/ | Toute la t\_livre | Mettre à jour un livre. |
| DELETE | /books/:id/notes/:id/ | Non | Supprimer un commentaire et une note par ID |
| PUT | /books/:id/notes/:id/ | La note + commentaire + UserID | Mettre à jour un commentaire et une note par ID |
| GET | /books/:id/notes/:id/ | Non | Obtenir une évaluation par son ID |
| POST | /books/:id/notes/ | La note + commentaire + UserID | Poste une note |
| GET | /books/:id/notes/ | Non | Regarde toutes les notes d’un livre |
| GET | /books/ :id/ | Non | Récupère un ivre par id |
| GET | /books/ | Non | Cherche un livre avec les paramètres demandés. |
| GET | /authors/ | Non | Récupère tous les auteurs |
| GET | /authors/:id/ | Non | Récupère les auteurs correspondant à un ID |
| GET | /authors/:id/books/ | Non | Récupère tous les livres d’un auteur |
| GET | /categories/ | Non | Récupère toutes les catégories |
| GET | /categories/:id/books/ | Non | Récupère tout livres qui correspond à une catégorie |
| GET | /categories/:id/ | Non | Récupère les categories qui correspondent à l’id |
| GET | /users/ | Non | Récupère tous les user |
| DELETE | /users/:id | Non | Supprimer un user par son ID |
| POST | /users/ | username + password | Ajouter un user |
| POST | /auth/login/ | username + password | Se connecter |

## Conception de la base de données

Le modèle relationnel de données a été conçu avec MySQL et Sequelize :

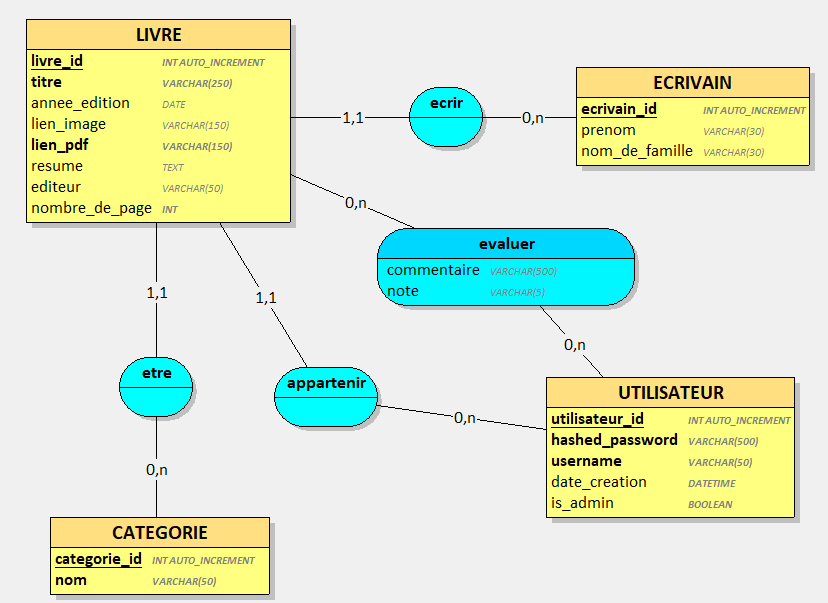
Tables principales : Utilisateur, Livre, Categories, Ecrivain

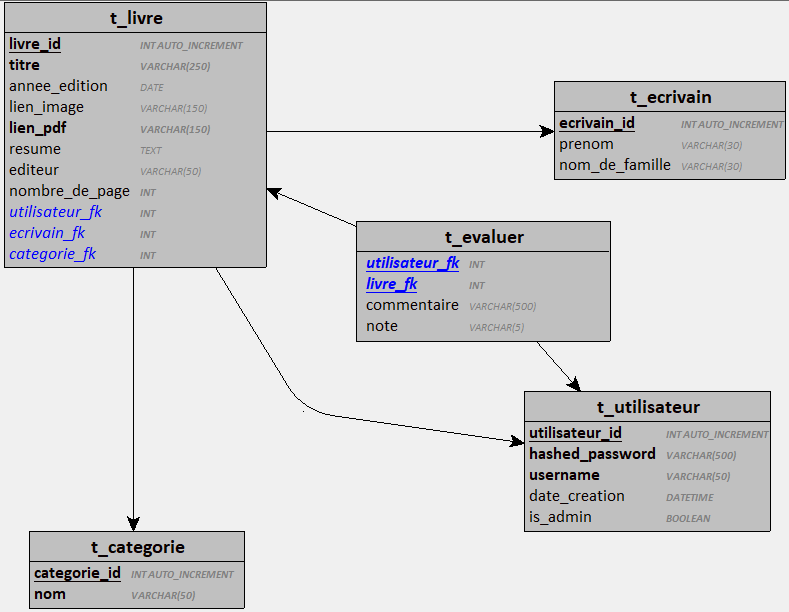
Relations :

Un utilisateur peut ajouter plusieurs livres

Un livre appartient à une catégorie et a un écrivain

Un utilisateur peut noter et commenter plusieurs livres





**Analyse de la structure du code**

Nous avous développé notre application en plusieurs fichiers .js, les fichiers contenans des opérations importantes comme les controllers, les routes ou les modèles utilisent l’extension .js.

Voici une image de la structure générale du projet :

Une image contenant texte, capture d’écran, Police, logiciel

Le contenu généré par l’IA peut être incorrect.

On peut voir les différents dossiers importants au bon fonctionnement du projet commme le dossier db, routes, controllers ou encore models. Ainsi que le fichier app qui permet d’initialiser les routes et le server lorsque l’on lance le projet, le helper qui sert juste a retourner un message + l’erreur dans les cas d’erreur ainsi que le fichier swagger utile a la docummentation.

**Dossier controller :**

**Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.**

le dossier controller contient tous les fichers contenant des méthodes qui permettront d’effectuer diverse opérations pour par exemple vérifier le token d’un user, ajouter un user dans la db etc, en fait toutes ces méthodes représentent toutes les actions effectuables dans l’application. Ils contiennent aussi un fichier private\_key qui contient une clé privée qui va servir au chiffrement et déchiffrement des tokens jwt.

**Dossier routes :**

Une image contenant texte, capture d’écran, Police, conception

Le contenu généré par l’IA peut être incorrect.

Le dossier routes contient une arborescence de dossiers qui répertorient les différentes routes par leur usage.

Ces routes permettent d’accéder à des fonctions des controllers sous certaines conditions (être admin ou être connecté).

**Dossier models**

Une image contenant texte, capture d’écran, Police, nombre

Le contenu généré par l’IA peut être incorrect.

Le dossier models contient les différents models qui vont servir à l’ORM (sequelise dans notre cas) de définir les données lorsque l’on va vouloir entrer des données dans la db, par exemple si une données est obligatoire ou le type d’une donnée qui au final représentent les attributs d’une table dans la db. Chque fichier représente donc une table et ses attributs. Les models seront ensuite utilisés par nos controllers pour entrer des données conforme dans la db.

**Dossier db :**

Une image contenant texte, capture d’écran, Police, nombre

Le contenu généré par l’IA peut être incorrect.

Le dossier db contient des mocks pour chaque table de la db aisni que le fichier sequelise.

Les mocks sont simplement des données factices que l’on va entrer dans la db lors de son initialisation pour ne pas qu’elle soit vide et donc avoir des données de test.

Le fichier sequeilise contient la méthode initDB qui va initialiser notre db avec les mocks via plusieurs méthodes (une méthode par mock) qui utilisent les donnée des mocks ainsi que les models pour créer les données dans la db.

Les dossiers validators et views sont inutiles mais seront utile dans la suite du projet.

# Réalisation

## Algorithmes utilisés

L'authentification repose sur les JWT (JSON Web Token) pour assurer une gestion sécurisée des accès utilisateur.

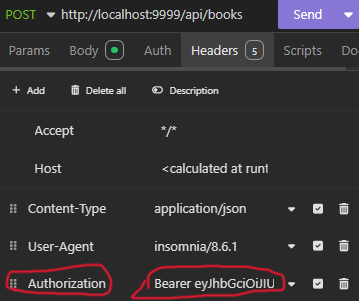
Voici comment cela fonctionne dans notre code :

Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.

On commence par importer jwt qui va servir a encoder/décoder des jetons jwt, ainsi que la privatekey que la fonction jwt.verify va utiliser pour décoder le token et ainsi verifier si il est valide.

La fonction auth est déclarée puis la constante authorisationHeader est déclarée, elle va prendre le header appelé « authorization », le contenu est simple, c’est un string avec le bearer + token comme ainsi :



Puis si authorisationHeader est null (donc qu’il n’y a pas de token) un message d’erreur est envoyé.

Ensuite si la variable n’est pas null:

Une image contenant texte, capture d’écran, logiciel, Police

Le contenu généré par l’IA peut être incorrect.

Une constante « token » va être crée contenant uniquement le token et pas le mot « bearer ».

Ensuite, le token va être vérifié avec la fonction jwt,verify et la privatekey, si il y a une erreur, le processus s’arrête et un message est envoyé.

Puis si l’id user de la requête correspond avec l’id trouvé dans le token et qu’il y a un id, la fonction next() s’execute (elle permet a la route de passer a la fonction suivante) sinon, un message d’erreur est envoyé.

J’ai choisi d’utiliser la fonction next car c’est ce qui me paraissait le plus court et adapté pour les routes, pour que chaque route doive passer par la fonction d’authentification et ainsi garantir la sécurité de celle-ci.

**L'algorithme utilisé pour gérer la gestion des rôles**

Lorsqu'un utilisateur est créé, il est défini comme non-administrateur par défaut.

Il est impossible de créer un utilisateur Admin (sauf depuis les mocks)

Pour obtenir le rôle d'administrateur, il faut le modifier dans la base de données, ce qui ne peut être fait que par un administrateur DB.

Cela permet d'empêcher un utilisateur d'envoyer une requête se définissant lui-même comme administrateur.

Une image contenant texte, capture d’écran, Police

Le contenu généré par l’IA peut être incorrect.

Pour plus de précision sur d’autres parties du code, je vous invite a consulter notre documentation swagger ou les commentaires de notre code.

## Maquettes – Figma

Pour visualiser la maquette de notre projet, vous pouvez consulter notre prototype interactif disponible sur Figma en cliquant sur le lien ci-dessous : [Maquettes PDF](Figma/Passion-Livres.pdf)

## Démarche d’éco-conception et Responsabilité Numérique

**Plusieurs principes ont été intégrés** :

Nous avons utilisé l'ORM Sequelize pour optimiser le processus de base de données

Nous avons également utilisé Express, un Framework très léger.

Nous avons utilisé uniquement les frameworks strictement nécessaires, afin d’alléger le projet

Nous avons utilisé une documentation bien structurée afin d’assurer une meilleure maintenabilité du projet, ainsi qu’une compréhension aisée et graphique.

# Tests

## Dossier des tests

Les tests ont été réalisés à l'aide de l'outil **Insomnia**, permettant d'effectuer des tests manuels sur les différentes routes de l’API.  
Une **vidéo démonstrative** est également fournie, attestant du bon fonctionnement de l’ensemble des routes testées.

# Conclusion

## Conclusion générale

Grace au temps supplémentaire accordé, nous avons pu tester toutes les routes (qui sont maintenant fonctionnelles) ainsi qu’améliorer grandement le rapport.

## Bilan personnel

### Conclusion Charles-Henri

En conclusion, j’ai beaucoup apprécié le projet, car il m’a permis d’en apprendre énormément sur le backend et comment est-ce qu’il fonctionne, comme la structure et les différentes parties du code (modèle, middleware, Controller, routes, …) Et ceci est extrêmement propice pour notre futur en tant que développeurs. Car après se projet, nous allons faire le frontend, ce qui veux dire que nous faisons la réalisation complète du frontend e du backend. Si c’étais à refaire, je pense que de séparer le frontend et le backend en deux trimestres différent pourrait être bénéfique à la meilleure compréhension totale du projet. Mais finalement, j’ai beaucoup appris et apprécié ce projet backend.

### Conclusion Antoine Piguet

J’ai aussi apprécié le projet car j’ai appris plein de choses entre autres sur l’utilisation des ORM et de plusieurs fonctionnalités de JS. J’ai l’impression d’avoir bien progressé et d’avoir acquis des connaissances.

Il faudra néanmoins que l’on apprenne à mieux communiquer en équipe je pense car nous av

ons eus quelques soucis pour finir pour la deadline car Yosef n’a pas update de tout le weekend alors que le code et le rapport n’étaient pas terminés, c’est pourquoi il n’a pas de conclusion.

### Conclusion Yosef Nademo

Ce projet m'a permis de développer mes compétences en backend, en particulier avec Node.js, Express.js, et JWT pour l'authentification. J'ai aussi appris l'importance de la communication et de la gestion du temps au sein de l'équipe. Bien que la majorité des fonctionnalités aient été réalisées, des retards sur les tests avec Vitest ont retardé la fin du projet. Si c'était à refaire, je m'assurerais d'une meilleure coordination pour respecter les délais. Ce projet m'a préparé à la phase frontend et a enrichi ma compréhension du développement complet d'une application.

# Divers

## Webographie/ Bibliographie

Documentation Sequelize : <https://sequelize.org/docs/v6/>

Documentation Express.js : <https://expressjs.com/>

Swagger API Docs : <https://swagger.io/docs/>

Tutoriels JWT et sécurité des API : <https://jwt.io/introduction/>

# Annexes

Repo GitHub: <https://github.com/Josefnademo/Passion_lecture.git>

Figma : [Passion-Livres – Figma](https://www.figma.com/design/iG5BIscos6hSl4IACsA2Ek/Passion-Livres?node-id=0-1&p=f&t=zx3GCyWmoT17IoEr-0)

GitHub Project : [Backlog · @Josefnademo's passion\_lecture](https://github.com/users/Josefnademo/projects/2/views/1)