ECOLE TECHNIQUE
ECOLE DES MÉTIERS LAUSANNE

API REST

P_WEB 295

Joachim Berchel & Thomas Nardou

MID2 - 2024

02.02.2024 – 15.03.2024

Antoine Mveng

Vennes - ETML

Table des matières

1. Introduction

L'objectif de ce projet est de réaliser l'API d'une application avec Node, Express Js et une base de données MySQL. Le travail s'effectue en groupe de 2. Chacun doit avoir travaillé sur chaque aspect du projet (programmation, rapport). Le projet s'effectue du 2 février 2024 jusqu'au 15 mars 2024. Le projet est lié au module C295 « Réaliser le backend pour des applications ».

Pour ce projet il a été mis à disposition : Un ordinateur avec le système d'exploitation Windows 10, l'éditeur de texte Visual Studio Code, un serveur local pour pouvoir exécuter les routes de l'API, deux conteneurs Docker qui servent à faire tourner MySQL et PhpMyAdmin, un navigateur Web pour pouvoir effectuer des recherches et un accès à internet.

Pour pouvoir réaliser au mieux ce projet il est nécessaire de savoir ce qu'est une API.

Une API (Application Programming Interface en anglais ou interface de programmation d'application en français) est un ensemble routes qui permettent à différents logiciels de communiquer entre eux. En d'autres termes, c'est un pont qui permet à différentes applications informatiques de partager des données et de fonctionner ensemble de manière cohérente.

2. Analyse

2.1 Planification

Pour la planification il a été décider d'utiliser le logiciel "Trello" car il s'agit d'un outil très simple d'utilisation et qui a déjà été utilisé dans le passé. Pour voir la planification qui a été faite, cliquer sur ce lien :

https://trello.com/invite/b/1XpbzDPe/ATTI15cb97128bf2b139ef7df8a7c1b23815DCDEEB8D/papi

2.2 API REST

Voici les routes qui ont été mises en place avec le verbe http et l'URL d'utilisation comme exemple ainsi qu'un exemple de ce qu'il faut fournir avec si besoin

Créer un Livre :

POST localhost:3000/api/books

Il a été décidé de créer cette route car il est fort probable que de nouveau livre soient ajouter et cette route permettra ainsi de simplifier l'insertion d'un nouveau livre

Voici un exemple de JSON qu'il faut fournir

```
{
    "title": "exemple",
    "numberPages": 80,
    "excerpt": "ceci est un exemple",
    "summary": "exemple de résumé",
    "writer": "Michel exemple",
    "editor": "exempleShuesha",
    "releaseYear": 2024,
    "avgRating": 2.1,
    "coverImage": "http://exemple.com/images.png",
    "fk_user": 1,
    "fk_category": 1
}
```

Supprimer un livre :

DELETE localhost:3000/api/books/**1**

Cette route a été créé dans le but de supprimer un livre afin de simplifier leur suppression.

Pour cette route il n'y a pas besoin de fournir un JSON il est juste nécessaire de fournir l'id du livre dans l'URL (en rouge).

Trouver tous les livres :
GET localhost:3000/api/books

Cette route permet de récupérer tous les livres de la base de données

Il n'y a rien à fournir ni à spécifier dans l'URL

Trouver tous les livres d'une catégorie :
GET localhost:3000/api/ categories/**1**/books

Cette route a été créé dans le but de récupérer un livre à partir de l'identifient de sa catégorie.

Pour cette route il n'y a pas besoin de fournir un JSON il est juste nécessaire de fournir l'id de la catégorie dans l'URL (en rouge).

Trouver un livre à partir de l'id :
GET localhost:3000/api/books/**1**

Cette route a été créé dans le but de récupérer un livre à partir de son identifiant.

Pour cette route il n'y a pas besoin de fournir un JSON il est juste nécessaire de fournir l'id du livre dans l'URL (en rouge).

Trouver un livre à partir de son titre :
GET http://localhost:3000/api/books/?title=**exemple**

Cette route a été créé afin de récupérer un livre à partir de son titre

Pour cette route il n'y a pas besoin de fournir un JSON il est juste nécessaire de fournir le titre du livre dans l'URL (en rouge).

Modifier un livre :
PUT localhost:3000/api/books/**1**

Il a été décidé de créer cette route car il est fort probable que des livres soient modifier et cette route permettra ainsi de simplifier la modification d'un livre

Voici un exemple de JSON qu'il faut fournir :

```
{
    "title": "exemple", // Valeur qui va être modifier
    "numberPages": 80 // Valeur qui va être modifier
}
```

D'accord, voici une reformulation du texte à la troisième personne :

L'exemple précédent n'est qu'un exemple. Seuls les champs que l'utilisateur souhaite modifier peuvent être inclus.

Il est aussi nécessaire de fournir l'id du livre dans l'URL (en rouge).

Créer une catégorie :
POST localhost:3000/api/categories

Il a été décidé de créer cette route car il est probable que de nouvelles catégories soient ajoutées et cette route permettra ainsi de simplifier l'insertion de la nouvelle catégorie

Voici un exemple de JSON qu'il faut fournir

```
{
    "name": "exempleCategory"
}
```

Supprimer une catégorie :
DELETE localhost:3000/api/categories/**1**

Cette route a été créé dans le but de supprimer une catégorie car si celle-ci doit être supprimé cette route permettrai de simplifier sa suppression.

Pour cette route il n'y a pas besoin de fournir un JSON il est juste nécessaire de fournir l'id de la catégorie dans l'URL (en rouge).

Modifier une catégorie :
PUT localhost:3000/api/categories/**1**

Il a été décidé de créer cette route car il est probable que des catégories soient modifiées et cette route permettra ainsi de simplifier sa modification

Voici un exemple de JSON qu'il faut fournir

```
{
    "name": "exempleCategoryUpdate"
}
```

Il est aussi nécessaire de fournir l'id du livre dans l'URL (en rouge).

Trouver toutes les catégories
GET localhost:3000/api/categories

Cette route permet tout simplement de récupérer toutes les catégories de la base de données

Il n'y a rien à fournir ni à spécifier dans l'URL

Trouver une catégorie à partir de son id

GET localhost:3000/api/categories/**1**

Cette route permet tout simplement de récupérer une catégorie à partir de son identifiant

Pour cette route il n'y a pas besoin de fournir un JSON il est juste nécessaire de fournir l'id de la catégorie dans l'URL (en rouge).

Récupérer son "Token"

POST localhost:3000/api/login

Cette route permet de se connecter à l'API et donc de récupérer son Jeton d'authentification.

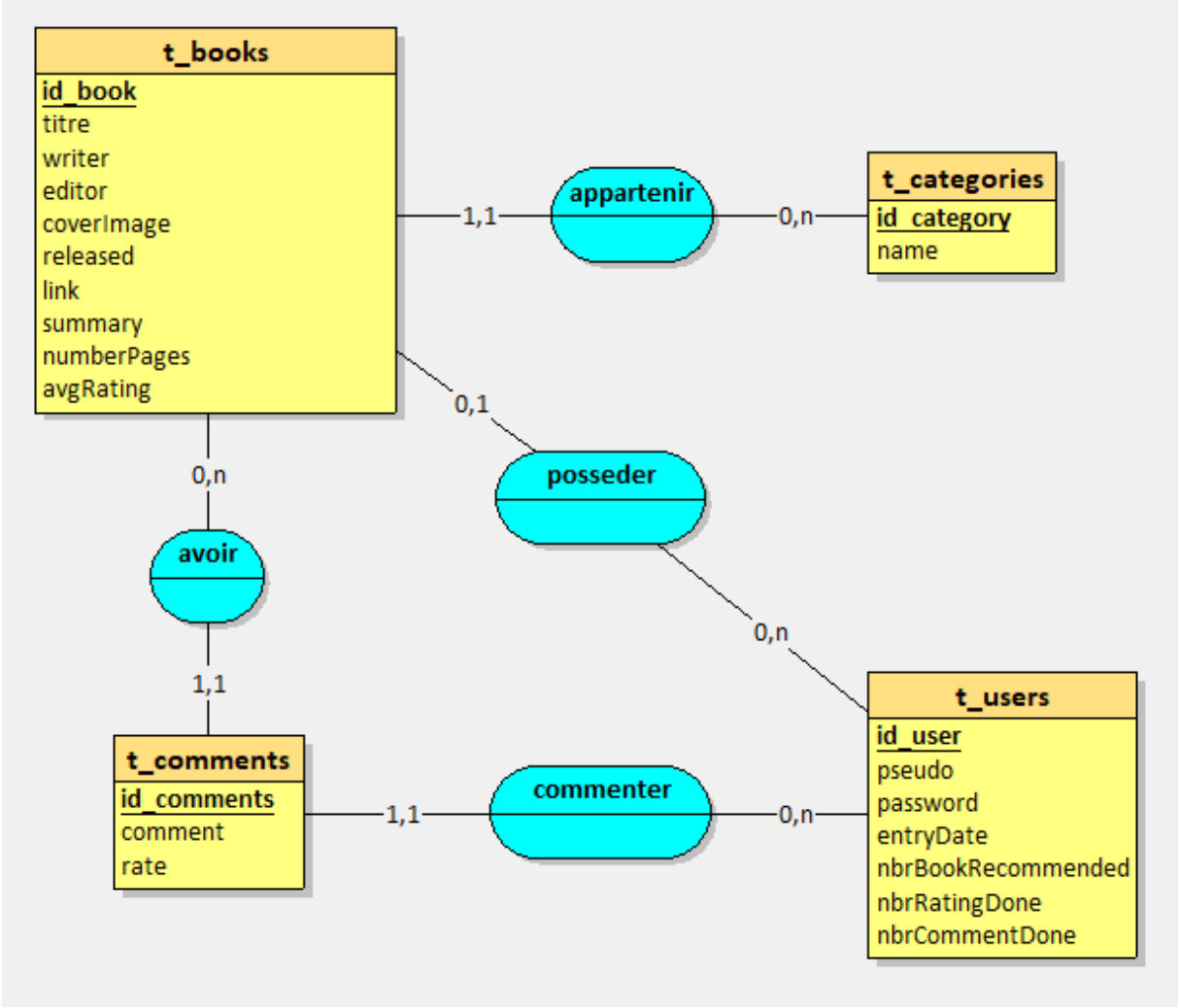
Voici un exemple du JSON qu'il faut fournir :

Le "username" correspond au nom de l'utilisateur et "password" à son mot de passe il n'y a rien besoin de spécifier dans l'URL

2.3 Base de données

Afin de faciliter la modélisation de la base de données il a été décider de créer un MCD (modèle conceptuel de donnée) et un MLD (Modèle logique de donnée) que voici :

MCD :



Détails :

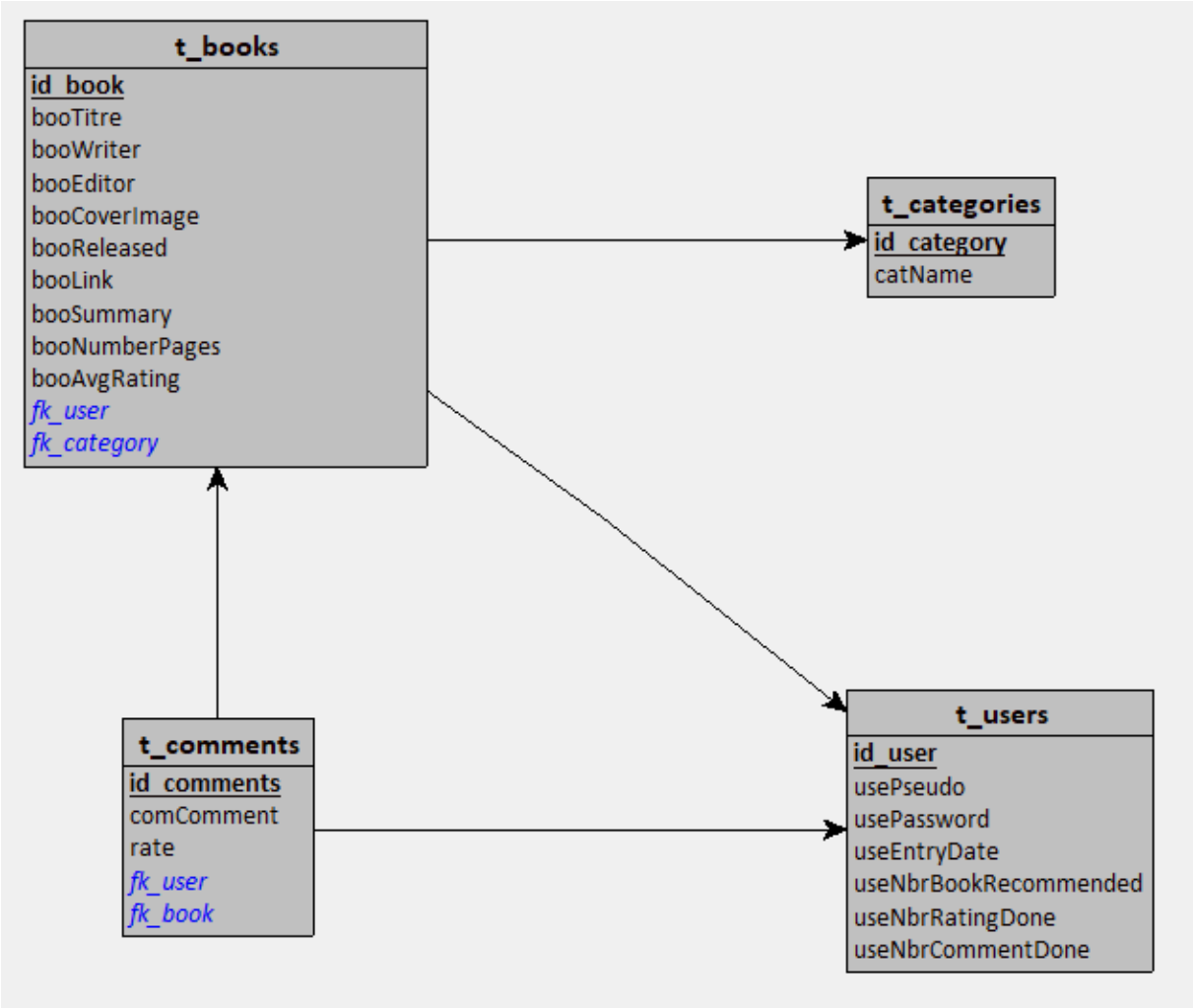
Il a été décider d'avoir 4 entités ("t_users", "t_categories", "t_comments", "t_books") chacune relié par une association qui vont être détailler :

- t_comments et t_users : il a été décidé de mettre le verbe "commenter" avec les cardinalités "1,1" et "0,n" car l'utilisateur peut commenter 0 ou plusieurs commentaires et un commentaire a été commenter par un et un seul utilisateur.
- t_comments et t_books : il a été décider d'utiliser le verbe "avoir" et pour les cardinalités il a été décidé d'utiliser les cardinalité "1,1" et "0,n" car un livre peut avoir 0 ou plusieurs commentaires et un commentaire peux appartient à un seul et unique livre
- t_books et t_users : il a été décider d'utiliser le verbe "posséder" avec les cardinalités "0,1" et "0,n" car un utilisateur peut posséder 0 ou plusieurs livres et un livre peut être posséder par 0 ou un utilisateur.

- t_books et t_categories : il a été décider d'utiliser le verbe "appartenir" avec les cardinalités "1,1" et "0,n" car un livre appartient une et une seule catégorie et une catégorie possède 0 ou plusieurs livres.

MLD :

Voici la conversion du MCD en MLD :

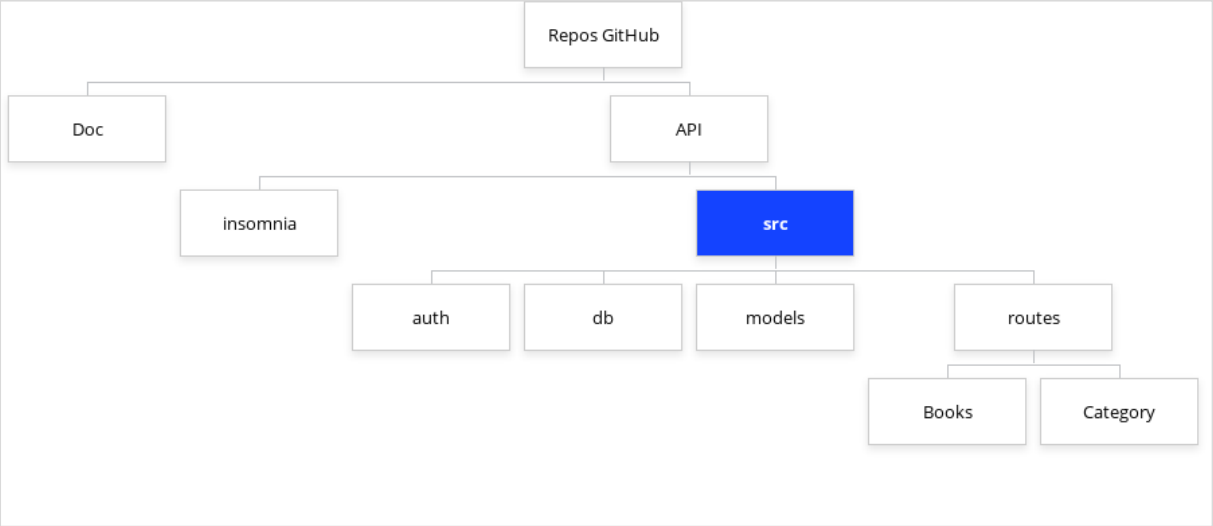


MPD :

Vous le trouverez en cliquant sur ce lien

2.4 Analyse de la structure du code

Tout d'abord voici un diagramme de notre arborescence :



A la racine se trouve le dossier du repositories de GitHub c'est lui qui contient tout notre travail mais passons directement au dossier "src" qui se trouve dans "API". Dans le dossier source il a 4 sous dossiers :

- auth : ce dossier contient tout ce qui concerne l'authentification d'un utilisateur vers l'API comme la clé privé et le système de vérification du "Token"
- db : ce dossier contient le fichier de configuration de sequelize et un ancien Mock des livres qui n'est plus utilisé et le MPD.
- models : ce dossier contient tous les modèles (tables) de la base de données.
- routes : ce dossier contient deux autres sous-dossiers qui contiennent les routes qui leurs correspondent dans le cas présent il y a un dossier "Books" qui contient les routes en lien avec les livres. Le dossier "Category" contient les routes consternant les catégories. Ce dossier contient aussi la route "login" qui permet de récupérer le "Token" (jeton).

3. Réalisation

3.1 Authentification et gestion des rôles

Une route "login" a été créée pour permettre aux utilisateurs de s'authentifier et de récupérer un "jeton" afin d'accéder à l'API. Cependant, la gestion des rôles n'a pas pu être implémentée en raison des contraintes de temps.

3.2 Sécurité

Afin de garantir une sécurité optimale il a été mis en place un système de cryptage des mots de passe des différents utilisateurs de l'API lors de sa création. Il a aussi été prévu de créer un utilisateur pour la base de données différent de celui par défaut (root) afin d'éviter toute grosse modification sur la base de données. Le package "dotenv" va aussi être mis en place afin d'éviter que d'autres personnes aient accès à l'information de connexion à la base de données. Mais par manque de temps ces fonctionnalité n'ont pas pu être mise en place.

3.3 Technique

Comprend une explication sur chaque fonctionnalité technique demandée.

- Un ensemble de routes permettant la gestion des livres, des catégories, des utilisateurs, etc.
  Chaque route s'effectue sur un fichier module javascript à part.

createBooks.mjs	Route permettant de rajouter des livres à l'API.
deleteBooks.mjs	Route permettant de supprimer des livres de l'API.
findAllBooks.mjs	Route permettant d'afficher tous les livres de l'API.
findBooksByCategory.mjs	Route permettant d'afficher les livres de l'API selon leur catégorie, la catégorie est choisie par l'utilisateur.
findBooksById.mjs	Route permettant d'afficher les livres de l'API selon leurs id, l'id est choisie par l'utilisateur.
findBooksByTitle.mjs	Route permettant d'afficher les livres de l'API selon leur titre, le titre est choisi par l'utilisateur.
login.mjs	Route permettant la connexion à l'API
updateBooks.mjs	Route permettant la modification des livres de l'API.
createCategory.mjs	Cette route permet de créer une nouvelle catégorie. Le nom doit être fournis par l'utilisateur
deleteCategory.mjs	Cette route permet de supprimer une catégorie. L'id de la catégorie doit être fournis dans l'URL
findAllCategory.mjs	Route permettant de récupérer toutes les catégories.
updateCategory.mjs	Route permettant de changer les valeurs de la catégorie. L'id doit être spécifier dans l'URL et les nouvelles informations doivent être fournis par l'utilisateur en format JSON

- Une validation de toutes les données fournies par le consommateur de l'API.

  Les fichiers dans le répertoire "models" permettent de valider certaines conditions.
  Voilà l'analyse de la table "t_books" :

title (string)	Ne peut pas être vide ou « null »
numberPages (int)	Ne peut pas être vide ou « null »
excerpt (string)	Ne peut pas être vide ou « nul »
summary (string)	Ne peut pas être vide ou « null »
writer (string)	Ne peut pas être vide ou « null »
editor (string)	Ne peut pas être vide ou « null »
releaseYear (int)	Ne peut pas être vide ou « null »
avgRating (float)	Ne peut pas être null
coverImage (string)	Ne peut pas être vide ou « null »
fk_user (int)	Ne peut pas être « null »
Fk_category (int)	Ne peut pas être « null »

  Analyse de la table "t_category" :

name (string)	Ne peut pas être vide ou « null

  Analyse de la table "t_comment" :

Comment (string)	Ne peut pas être vide ou « null »
Rate (int)	Ne peut pas être « null »
Fk_user (int)	Ne peut pas être « null »
fk_book (int)	Ne peut pas être « null »

  Analyse de la table "t_user" :

username (string)	Ne peut pas être vide ou « null »
password (string)	Ne peut pas être « null »
nbrBookRecommended (int)	Ne peut pas être « null »
nbrRatingDONE (int)	Ne peut pas être « null »
nbrCommentsDone (int)	Ne peut pas être « null »

- Une gestion des statuts http (200, 3xx, 4xx, 5xx) et des erreurs.

  Une erreur 404 a lieu si le chemin spécifié par l'utilisateur n'existe pas.
  L'erreur 401 a lieu s'il n'y a pas de token d'identification.
  L'erreur 500 a lieu lorsqu'il y a un problème avec la base de données.
  En cas de succès on retourne l'information 200.

- Une recherche sur les livres, catégories, etc.

  L'utilisateur peut effectuer une recherche non stricte du titre du livre ou peut utiliser des routes différentes afin de rechercher par catégorie ou par id.

- Un système d'authentification basé sur les jetons JWT

  Les routes sont protégées par un système d'authentification basé sur les jetons JWT, Json Web Token. Afin de récupérer un token d'identification il faut utiliser

la route nommé « login » dans insomnia, l'utilisateur est « etml » et le mot de passe « etml ».

- Une documentation Swagger la plus complète possible

  La documentation swagger est accessible sur « http://localhost:3000/api-docs ».

- Des tests de votre API avec Insomnia ou Postman

  Il y a un fichier json à importer dans insomnia avec tous les tests effectués. Le fichier se trouve à la racine de l'API dans le dossier « insomnia », même emplacement que le dossier « src » et « node_modules ».

- Des tests automatisés avec vitest

  Par manque de temps cette fonctionnalité n'a pas été implémenté.

- Une intégration continue dans github actions

  Par manque de temps cette fonctionnalité n'a pas été implémenté.

- Une « dockerisation » du backend

  Par manque de temps cette fonctionnalité n'a pas été implémenté.

4. Test

Les tests automatiques étaient censés s'exécuter grâce au package "vitest" mais par manque de temps il n'a pas été possible de les implémenter.

5. Conclusion

5.1 Gestion du code

GitHub a été utilisé lors de ce projet afin de gérer le code. Il n'y a pas eu de méthodologie de travail spécifique, n'étant que 2 à travailler sur ce projet nous avons été capable de nous coordonner afin de ne pas créer d'erreurs en travaillant sur la branche "main".

5.2 Conclusion générale

Ce projet n'a pas pu être terminée dans son entièreté dû à la trop grosse charge de travail demandée dans le cahier des charges cependant nous avons pu mettre en place les fonctionnalités principales comme les routes pour les catégories et les livre et le système d'authentification pour récupérer le Token (jeton). Pour ce qui est des fonctionnalités manquantes il s'agit notamment la gestion de rôles et les points de sécurités discutés plus haut. La dockerisation et l'automatisation des test unitaires n'ont pas pu être mis en place aussi dû à la charge de travail trop grande et le manque de temps

5.3 Conclusion personnelle

Thomas :

Ce projet m'a beaucoup appris notamment sur sequelize car je ne savais pas comment faire des références entre les tables avec les "fk" (foreign key) et comment faire des jointures entre mes tables. Ce projet m'a aussi appris comment je peux réaliser des routes imbriquées avec Express JS. Si ce projet était à refaire je le referais avec grand plaisir en changeant juste chose qui est le journal de travail car à certain moment j'oubliais de notifier que j'étais entrain de travailler sur une tâche. Sinon je ne pense pas que je changerais grand-chose dans ma méthodologie de travail.

Joachim

Ce projet m'a permis de mettre en pratique la théorie vue durant le module associé au projet. Ainsi cela m'a permis de bien comprendre ce que j'avais vue en théorie et à régler des imprécisions. Je considère que ce projet m'a beaucoup aidé dans mon apprentissage du backend et des API. J'ai également appris à utiliser swagger et j'ai consolider mes bases en javascript.

5.4 Critiques

La planification du projet effectué sur Trello a été efficace afin de répertorier les taches nécessaires et manquantes mais elle n'est pas assez spécifique pour connaître précisément le temps passé sur une tâche.

6. Webographie / Bibliographie / Glossaire

- https://sequelize.org/docs/v6/advanced-association-concepts/eager-loading/
- https://sequelize.org/docs/v6/core-concepts/assocs/
- Support de cours du module I-295

7. Utilisation d'IA

L'intelligence artificiel "ChatGPT" a été utilisé sur le rapport afin d'améliorer la compréhension du texte.