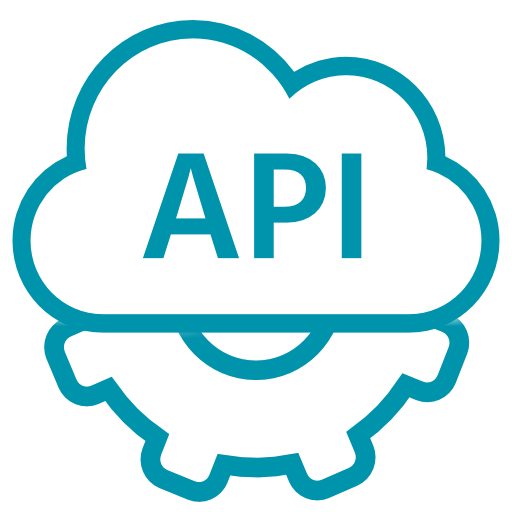
Logotipo, Icono

Descripción generada automáticamente

**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 4](#_Toc161145137)

[2. Analyse 4](#_Toc161145138)

[2.1 Planification 4](#_Toc161145139)

[2.2 API REST 5](#_Toc161145140)

[Créer un Livre : 5](#_Toc161145141)

[Supprimer un livre : 5](#_Toc161145142)

[Trouver tous les livres : 5](#_Toc161145143)

[Trouver tous les livres d’une catégorie : 5](#_Toc161145144)

[Trouver un livre à partir de l’id : 6](#_Toc161145145)

[Trouver un livre à partir de son titre : 6](#_Toc161145146)

[Modifier un livre : 6](#_Toc161145147)

[Créer une catégorie : 6](#_Toc161145148)

[Supprimer une catégorie : 7](#_Toc161145149)

[Modifier une catégorie : 7](#_Toc161145150)

[Trouver tous les livres 7](#_Toc161145151)

[2.3 Base de données 8](#_Toc161145152)

[MCD : 8](#_Toc161145153)

[MLD : 9](#_Toc161145154)

[MPD : 9](#_Toc161145155)

[2.4 Analyse de la structure du code 10](#_Toc161145156)

[2.5 Schéma d’interaction backend – frontend 10](#_Toc161145157)

[3. Réalisation 11](#_Toc161145158)

[3.1 Authentification et gestion des rôles 11](#_Toc161145159)

[3.2 Sécurité 11](#_Toc161145160)

[3.3 Technique 11](#_Toc161145161)

[4. Test 13](#_Toc161145162)

[5. Conclusion 13](#_Toc161145163)

[5.1 Gestion du code 13](#_Toc161145164)

[5.2 Conclusion générale 13](#_Toc161145165)

[5.3 Conclusion personnelle 13](#_Toc161145166)

[5.4 Critiques 13](#_Toc161145167)

[6. Webographie / Bibliographie / Glossaire 13](#_Toc161145168)

[7. Utilisation d’IA 13](#_Toc161145169)

# 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) 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 un 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



### 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**

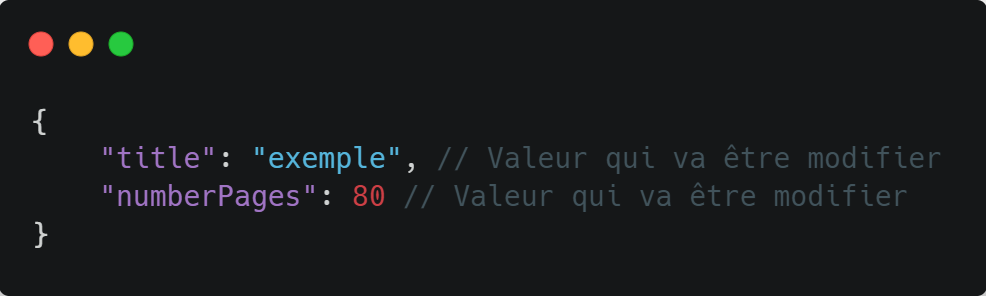
Cette route a été créé afin de récupérer un livre à partir de son titre

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

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

Une image contenant capture d’écran, texte, Police

Description générée automatiquement

### 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



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

### Trouver tous les livres

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

## 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 :

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

Description générée automatiquement

#### 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 :

Une image contenant texte, capture d’écran, diagramme, affichage

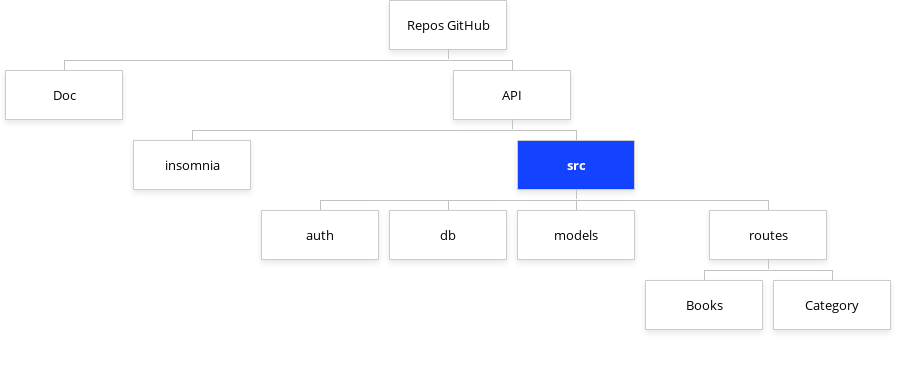
Description générée automatiquement

### MPD :

Vous le trouverez en cliquant sur [ce lien](../API/src/db/db_books.sql)

## 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 nous avons 4 sous dossiers :

* auth : ce dossier contient tout ce qui concerne l’authentification d’un utilisateur vers l’API enfin d’obtenir son "Token"
* db : ce dossier contient le fichier de configuration de sequelize et un ancien Mock des livres qui n’est plus utilisé.
* 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

## 2.5 Schéma d’interaction backend – frontend

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.)

TODO

# 3. Réalisation

## 3.1 Authentification et gestion des rôles

TODO

## 3.2 Sécurité

Afin de garantir une sécurité optimale il va être mis en place un système de cryptage des mots de passe des différents utilisateurs de l’API. Il a aussi été prévu de créer un utilisateur différent de celui pas 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

## 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. Tous les fichiers sont dans le même dossier « routes »

|  |  |
| --- | --- |
| 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 |

Un fichier « helper.mjs » est stocké dans le même répertoire, il contient des fonctions pouvant être utiles à aux routes.

|  |  |
| --- | --- |
| success | Permet d’écrire l’état de la requête et de l’afficher à l’utilisateur. |
| getUniqueId | Génère un nouvel id, l’id sera le plus grand id +1. Cette fonction a été utilisé lors de l’utilisation du mock |
| getBook | Retrouve un livre grâce à son id. Cette fonction a été utilisé lors de l’utilisation du mock |
| removeBook | Supprime un livre grâce à son id. Cette fonction a été utilisé lors de l’utilisation du mock |
| updateBook | Met à jour un livre grâce à son id. Cette fonction a été utilisé lors de l’utilisation du mock |

* 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.

200 pour les succès – 404 si l’URL n’est pas correcte – 500 s’il y a une erreur depuis la base de données.

Une erreur 404 à lieu si le chemin spécifié par l’utilisateur n’existe pas.

TODO ?

* Une recherche sur les livres, catégories, etc.

TODO

* Un système d’authentification basé sur les jetons JWT

TODO

* Une documentation Swagger la plus complète possible

TODO

* Des tests de votre API avec Insomnia ou Postman

TODO

* 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 le « main ».

## 5.2 Conclusion générale

Comprend une conclusion générale sur le projet (½ page minimum)

TODO

## 5.3 Conclusion personnelle

Comprend une conclusion personnelle sur le projet (½ page minimum)

TODO

## 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.