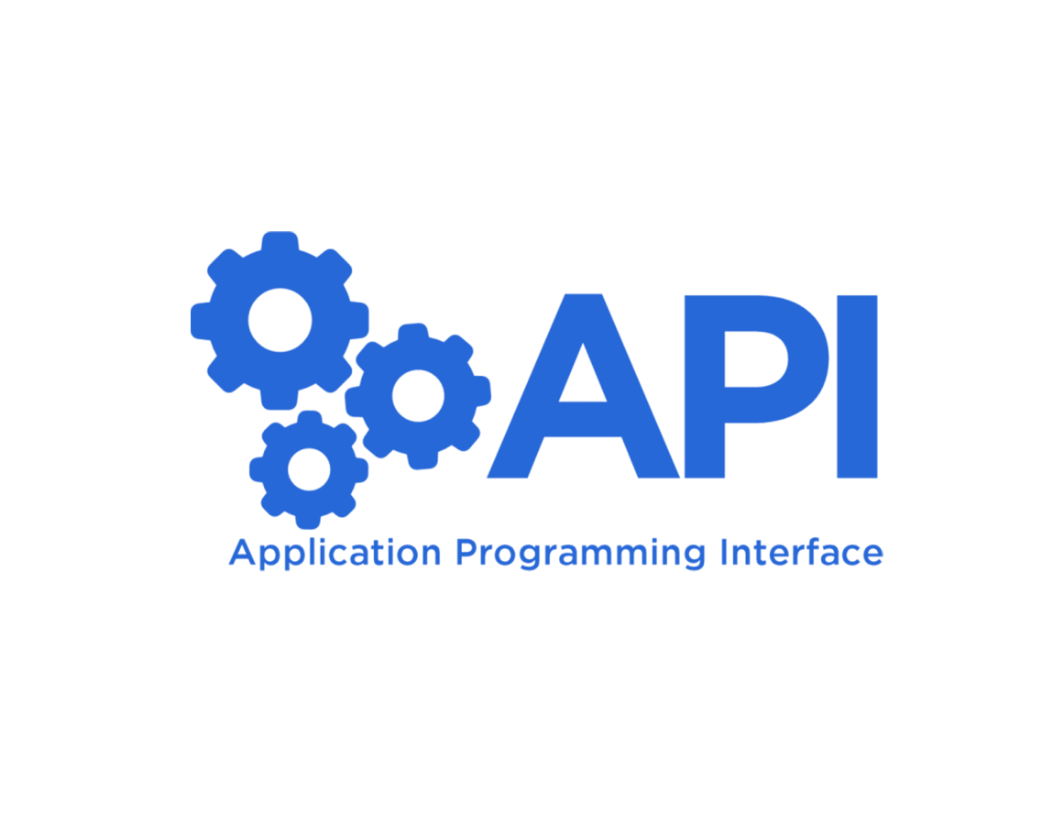
P\_WEB – C295



Sofien Habib Belkhiria - Lucas Lordon

Vennes – ETML

19.02.24 – 12.03.24

Chef de projet : Antoine Mveng

Table des matières

[Introduction 1](#_Toc160533968)

[Analyse 1](#_Toc160533969)

[Routes 1](#_Toc160533970)

[Réalisation 2](#_Toc160533971)

[Test 2](#_Toc160533972)

[Conclusion 2](#_Toc160533973)

[Webographie/Bibliographie/Glossaire 2](#_Toc160533974)

# Introduction

Ce projet à été réaliser par Lucas Lordon et Sofien Habib Belkhiria.

Le but de ce projet est de faire une application backend qui consiste à la mise en place d’une API REST. L’API REST doit reprendre les notions suivantes vu dans le cours : les routes, la gestion des statuts, un système de recherche, un système d’authentification JWT, une documentation Swagger, des tests via Insomnia, des tests automatisés avec vitest, GitHub actions et une dockerisation du backend.

Le développement de cette application permettra de consolider les bases de nos connaissances sur le développement Backend et l’utilisation de technologies tels que « Sequilize » et « Insomnia ».

Le projet à été réaliser entre le 19.02.24 et le 12.03.24 pour un total de 24 périodes.

L’ensemble du projet et présent dans le repository GitHub P\_Web295, mis à par la planification qui a été réaliser sur trello.

To Do : Comprend une brève explication du projet (½ page)2

# Analyse

To Do : 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)

## Routes

Livre

|  |  |  |  |
| --- | --- | --- | --- |
| Verbes | Utilisation | Exemple URI | JSON |
| Get | Retourne tous les livre se trouvant dans la db. | http://localhost/api/books |  |
| Get | Retourne un livre précis de la db à condition que la personne entre l’identifiant du livre. | http://localhost/api/books/id |  |
| POST | Permet d’ajouter un livre à la db. | http://localhost:3000/api/books | [ref 1](Refs/ref1.png) |
| DELETE | Permet de supprimer un livre à condition de connaitre son ID. | http://localhost:3000/api/books/:id |  |
| PUT | Permet de mettre à jour un livre de la db à condition de connaitre son id. | http://localhost:3000/api/books/:id | [ref 2](Refs/ref2.png) |
| Get | Permet de connaitre les livres en fonction de leur catégorie. | http://localhost:3000/api/categorys/booksCategorys?category=fant |  |

Utilisateur

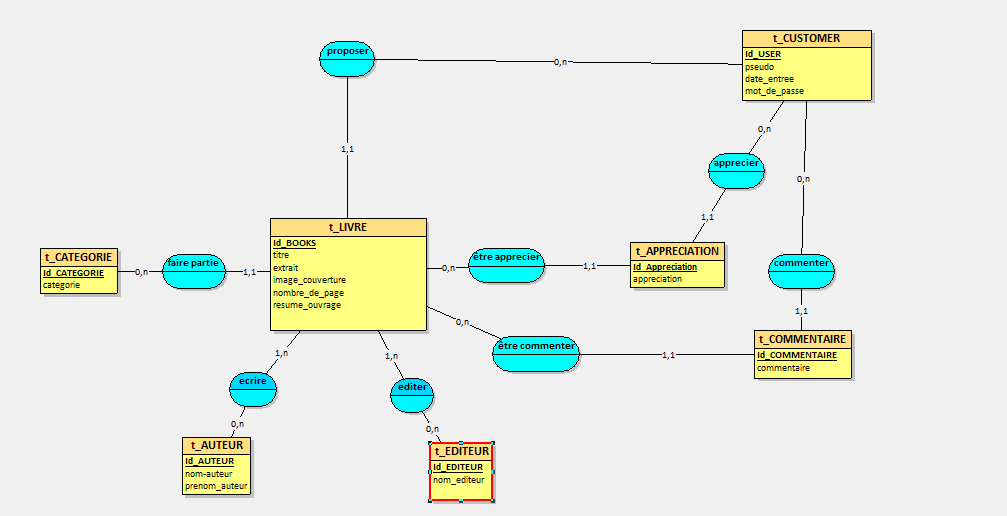
|  |  |  |  |
| --- | --- | --- | --- |
| Verbes | Utilisation | Exemple URI | JSON |
| Get | Retourne tous les utilisateurs de présent dans la db. | http://localhost:3000/api/users |  |
| Get | Retourne un utilisateur précis de la db à condition de connaitre son id. | http://localhost:3000/api/users/:id |  |
| POST | Permet d’ajouter un livre à la db. | http://localhost:3000/api/users | [ref 3](Refs/ref3.png) |
| DELETE | Permet de supprimer un livre de la db | http://localhost:3000/api/users/:id |  |
| PUT | Permet de mettre à jour une entrée de la DB à condition de connaitre son identifiant. | http://localhost:3000/api/users/:id | [ref 4](Refs/ref4.png) |

Catégorie

|  |  |  |  |
| --- | --- | --- | --- |
| Verbes | Utilisation | Exemple URI | JSON |
| Get | Retourne toutes les catégories de la db | http://localhost:3000/api/categorys/ |  |
| Get | Retourne une catégorie précise de la db à condition de connaitre son id. | http://localhost:3000/api/categorys/:id |  |
| POST | Permet d’ajouter une catégorie à la db. | http://localhost:3000/api/categorys/ | [ref 5](Refs/ref5.png) |
| DELETE | Permet de supprimer une catégorie de la db | http://localhost:3000/api/categorys/:id |  |
| PUT | Permet de mettre à jour une entrée de la DB à condition de connaitre son identifiant. | http://localhost:3000/api/categorys/:id | [ref 6](Refs/ref6.png) |

## Base de données

MCD



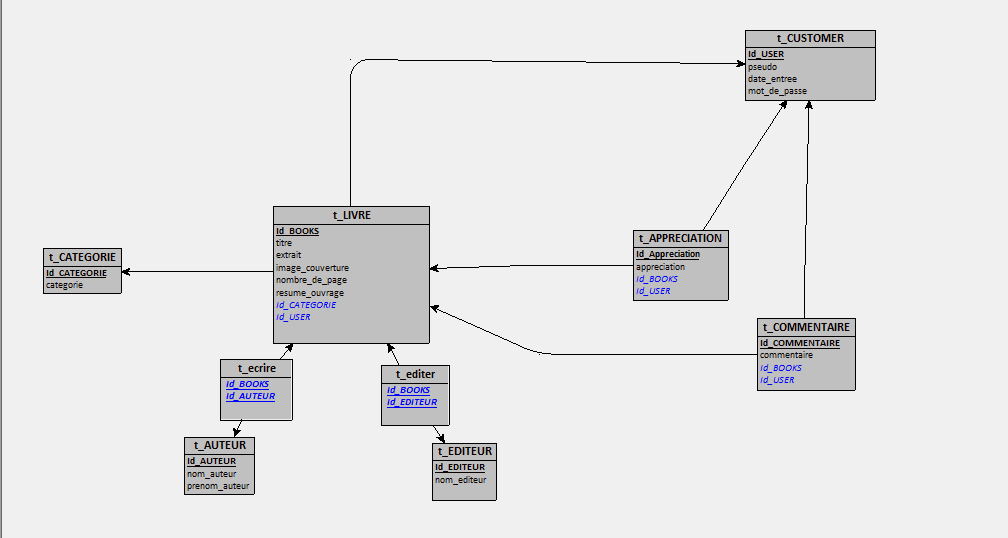
Table

La base de données comprend 7 tables vous trouverez ci-dessous les raisons des cardinalités de chacune d’entre elle ainsi que les raisons de leur emplacement.

* T\_CUSTOMER :
  + Contient le pseudo, le mot de passe et la date d’entrée de l’utilisateur.
  + Est associer à la table livre afin que l’utilisateur puissent proposer des livres, chacun des livres de l’ayant été proposer par un utilisateur a donc un lien vers l’id de l’utilisateur l’ayant proposé.
  + Est associer à la table appréciation afin de permettre à l’utilisateur de pouvoir laisser des appréciations et de connaitre quel utilisateur à laisser quelle appréciation.
  + Est associer à la table commentaire afin de permettre à l’utilisateur de pouvoir laisser des commentaires et de connaitre quel utilisateur à laisser quel commentaire.
* T\_LIVRE
  + Contient le titre, le prix, la/les catégorie(s), le nombre de pages, le résumé d’un livre, la table contient aussi l’ID de l’utilisateur qui à proposer le livre.
  + Est associer à la table éditeur afin de permettre à l’utilisateur de connaitre quel éditeur a édité le livre.
  + Est associer à la table catégorie afin de permettre à l’utilisateur de connaitre quel sont les catégories associer au livre.
  + Est associer à la table auteur afin de permettre à l’utilisateur de connaitre qui est l’auteur du livre.
* T\_AUTEUR
  + Contient le nom et le prénom de l’auteur.
* T\_CATEGORIE
  + Contient le nom des catégories
* T\_EDITEUR
  + Contient le nom de l’éditeur
* T\_COMMENTAIRE
  + Contient le commentaire de l’utilisateur
* T\_APPECIATION
  + Contient l’appréciation de l’utilisateur

MLD

* Vous trouverez-ci-dessous la logique de connexion entre les tables.

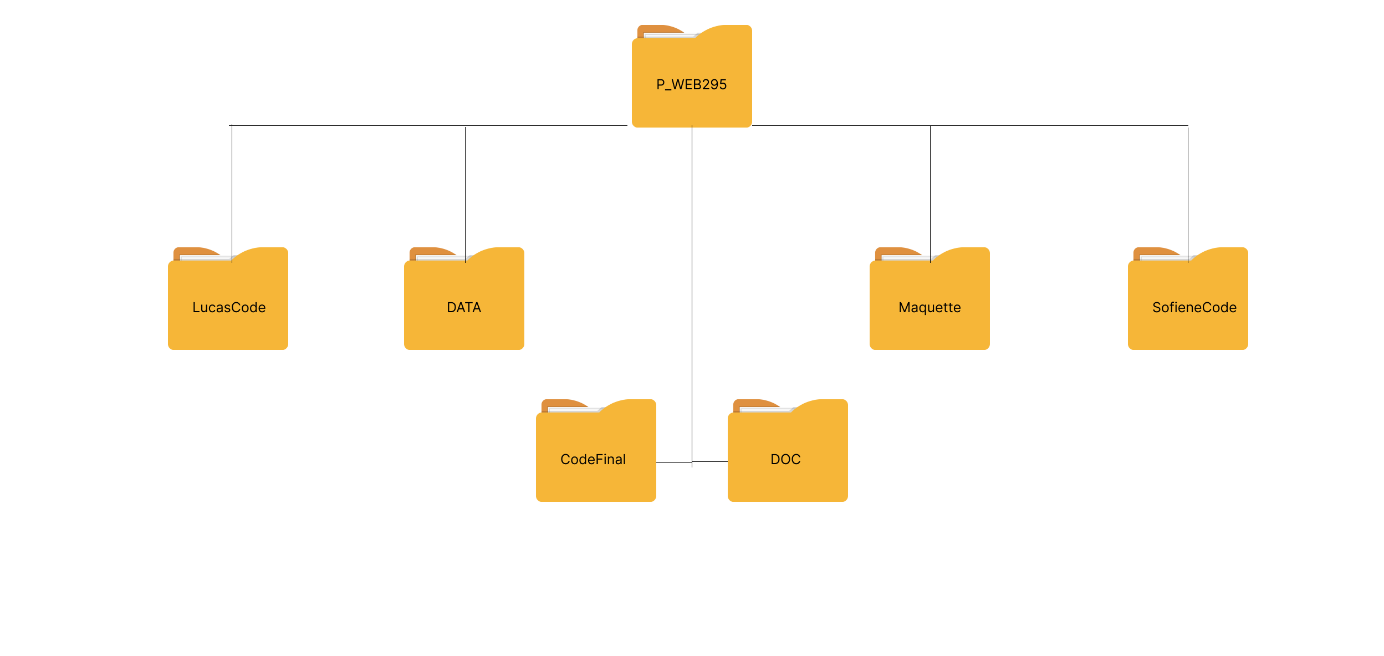


## Structure du projet

But

* Vous trouverez ci-dessous une analyse du code de l’API en commençant par la structure des fichiers, passant par les modules utiliser et finissant par le contenu des fichiers.

Structure des Dossiers



LucasCode

* Dossier contenant le code de l’API appartenant à Lucas.

SofieneCode

* Dossier contenant le code de l’API Appartenant à Sofiène.

Data

* Dossier contenant tous les dossiers concernant la base de données et les requêtes tel que :
  + MCD/MLD looping.
  + Image du MCD et MLD du looping.
  + Fichier SQL contenant les données de la db.
  + Fichier Excel contenant les informations des livres.
  + Fichier Insomnia contenant toutes les requête de l’API.

Doc

* Dossier contenant la documentation du projet comme le rapport ainsi que des PDF utile à l’API en plus des journaux de travaux.

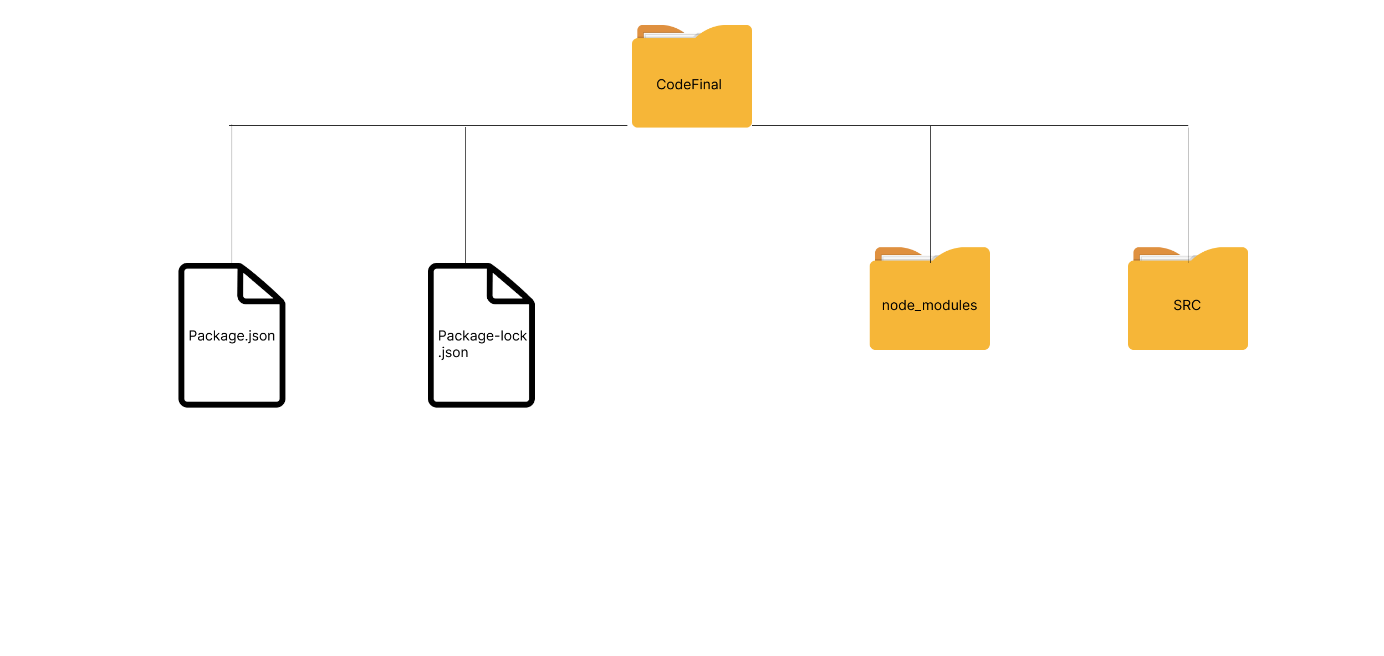
Maquette

* Dossier contenant les maquette du site Web.

Code Final

* Code final de l’API, fusion du code de Lucas et Sofiène.

## Structure dossier codeFinal



Package.json

* Fichier de base de l’API, contient la référence des modules utiliser, ainsi que les informations de base de l’API

Package-lock.json

* Contient tous les imports des différents modules utiliser dans l’API

Node\_modules

* Contient tous les fichiers nécessaires au bon fonctionnement des modules installé

SRC

* Contient tous les dossiers du code de l’API (voir précision plus bas).

## Structure dossier SRC

Model

* Contient les différents modèles de chacune des tables de l’API, ces model permette à l’API de connaitre sur quelles tables elle va agir ainsi que les données acceptées sur ces tables.

Routes

* Contient toutes les routes utilisables par l’API, ces dans ce fichier que se trouve le code s’appliquant quand le client exécute une des URI vus plus [haut](#_Routes).

Auth

* Contient le code gérant l’authentification des utilisateurs via leur token, il est aussi responsable d’empêcher certaines routes dans le cas ou l’utilisateur n’est pas authentifier.

DB

* Dossier contenant les fichiers servant de données de base à introduire dans la DB, contient aussi le code permettant de lier l’API à la base de données en plus de créer les tables de la DB en fonction des model vu précédemment.

# Réalisation

# Test

# Conclusion

## Auto-évaluation :

Dans cette partie de la conclusion, la note que nous nous sommes attribués sera expliquer critère par critère :

|  |  |  |
| --- | --- | --- |
| Critère | Note | Explication |
| Rythme de travail | ACQUIS | Nous avons acquis les exigences demandées, mais nous n’avons pas eu le temps de faire toutes les routes bonus. |
| Qualité du travail | ACQUIS | Nous estimons avoir un travaille transmissible mais le manque des commentaires via swagger reste un point à faire. |
| Niveau de maîtrise technique | ACQUIS | Nous estimons maitriser l’ensemble des technologies de ce projet mais loin de la maitrise totale du Backend d’API. |
| Autonomie | LARGEMENT ACQUIS | Dans l’ensemble nous avons été très autonomes, nous avons su surmonter les difficultés. |
| Processus de travail | LARGEMENT ACQUIS | Nous avons respecté les contraintes imposer durant ce projet. De plus l’ensemble des documents demander ont été fournis. |
| Expression orale et écrite Technique de présentation | ACQUIS | Dans l’ensemble nous avons bien documenter via insomnia notamment mais le commentaire reste un manque dans notre projet. |
| Approche de la durabilité et de l’innovation | ACQUIS | Nous avons respecté la durabilité sans apporter de réelle innovation. |
| Aptitude au travail en équipe Gestion des conflits Communication | LARGEMENT ACQUIS | Nous sommes contents d’avoir travailler ensemble car il n’y a eu aucun problème est que nous étions les deux autant impliquer. |
| Notes | **ACQUIS** | **/** |

Les fichier PDF de notre autoévaluation est disponible sous :

P\_WEB295\Doc\ AutoEvaluation\_P\_WEB-SBA-LLN

# Webographie/Bibliographie/Glossaire