12/03/2024

Rapport de projet

P_Web_C295

Tiago Rodrigues Sousa, Evin Paramanathan
ETML – VENNES
CHEF DE PROJET : ANTOINE MVENG
SALLE : B11

Rapport de projet

Table des matières

Introduction

L'objectif de ce projet est de mettre en pratique le module ICT C295, qui consiste à réaliser le backend pour des applications. Pour ce faire, nous devons mettre en place une API qui permet de partager notre passion pour la lecture. Nous travaillons en binôme pour mener à bien ce projet.

Pour réaliser ce projet, nous disposons d'un PC, de VS Code, d'un accès à Internet, de supports de cours, ainsi que de technologies telles que Node.js, npm, Express.js, Sequelize, MySQL, Swagger et Insomnia. Nous avons à notre disposition 24 périodes pour mener à bien ce projet.

Avant d'entrer dans les détails du projet, regardons simplement ce qu'est une API.

API est l'abréviation d'Application Programming Interface ou Interface de Programmation d'Application en français. Une API est une interface logicielle qui permet de relier un service à un autre ou un logiciel à un autre. Cela permet d'échanger des données et des fonctionnalités.

Analyse

1. Planification

La planification des taches à réaliser pour le projet a été fait sur Trello, ça permet de créer un tableau et de l'adapter pour faire un suivit du projet plus personnalisé.

https://trello.com/invite/b/CKSDCYuq/ATTI9eb058082f456ec9f8f1b74df1999ac35786C719/pweb295

2. Verbe http

Verb	Action
GET	Recherche
POST	Créer
PUT	Mettre à jours
DELETE	Supprimer

3. Routes :

Livre :

Nom du fichier	Verbe	URI	Action
addLivre	POST	http://localhost:3000/api/livres	Permet d'ajouter un livre
deleteLivre	DELETE	http://localhost:3000/api/livres/3	Permet de supprimer un livre
getAllLivresCategorie	GET	http://localhost:3000/api/categories/1/livres	Permet de rechercher les livres

			d'une catégorie spécifique
getLivre	GET	http://localhost:3000/api/livres	Permet de rechercher un livre spécifique via un paramètre (titre) ou bien tous les livres
getLivreId	GET	http://localhost:3000/api/livres/2	Permet de rechercher un livre spécifique grâce à son ID
updateLivre	PUT	http://localhost:3000/api/livres/2	Permet de mettre à jour un livre

Catégorie :

addCategorie	POST	http://localhost:3000/api/categories	Permet d'ajouter une catégorie
getCategorie	GET	http://localhost:3000/api/categories	Permet de rechercher une catégorie via un paramètre (nom) ou bien de récupérer toutes les catégories
getCategorieId	GET	http://localhost:3000/api/categories/1	Permet de rechercher une catégorie à partir d'un id

Commentaire :

addCommentaires	POST	http://localhost:3000/api/commentaires	Permet d'ajout un commentaire
getCommentaire	GET	http://localhost:3000/api/commentaires	Permet de récolter tous les commentaires

Login :

Login	POST	http://localhost:3000/api/login	Permet de générer un token pour l'api

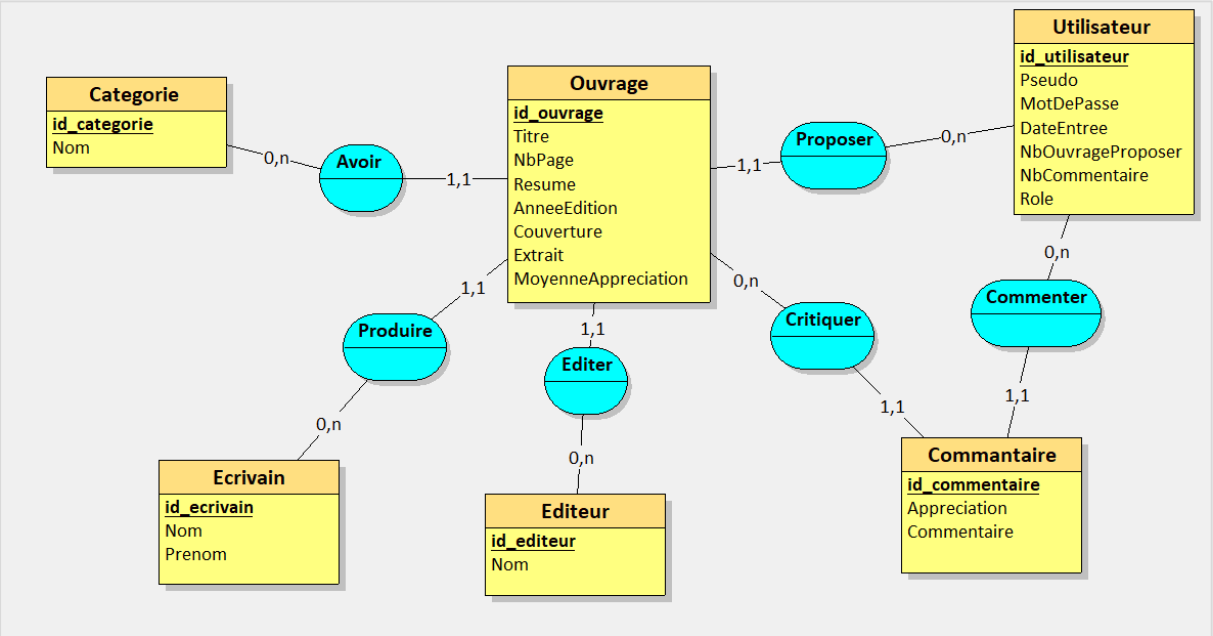
4. Base de données

Pour faire la base de données nous avons décidé de faire 6 tableaux, comme on peut le voir dans l'image ci-dessous.

Premièrement, il y a la table principale "ouvrage", celle-ci est reliée à 5 autres tables. Ensuite, il y a la table "catégorie" que nous avons décidé de ne pas mettre directement dans "ouvrage", car si un livre a plusieurs catégories, cela pourrait poser un problème pour les recherches.

Deuxièmement, il y a les deux tables "écrivain" et "éditeur", pour chaque écrivain et chaque éditeur qui contient le nom pour les deux tables, et dans la table "écrivain", il y a le prénom en plus du nom. Ensuite, il y a la table "utilisateur" dans laquelle il y a le pseudo, le mot de passe, la date d'entrée, le nombre d'ouvrages proposés, le nombre de commentaires et leur rôle.

Et pour finir, il y a la table "commentaire" dans laquelle il y a l'appréciation et le commentaire.



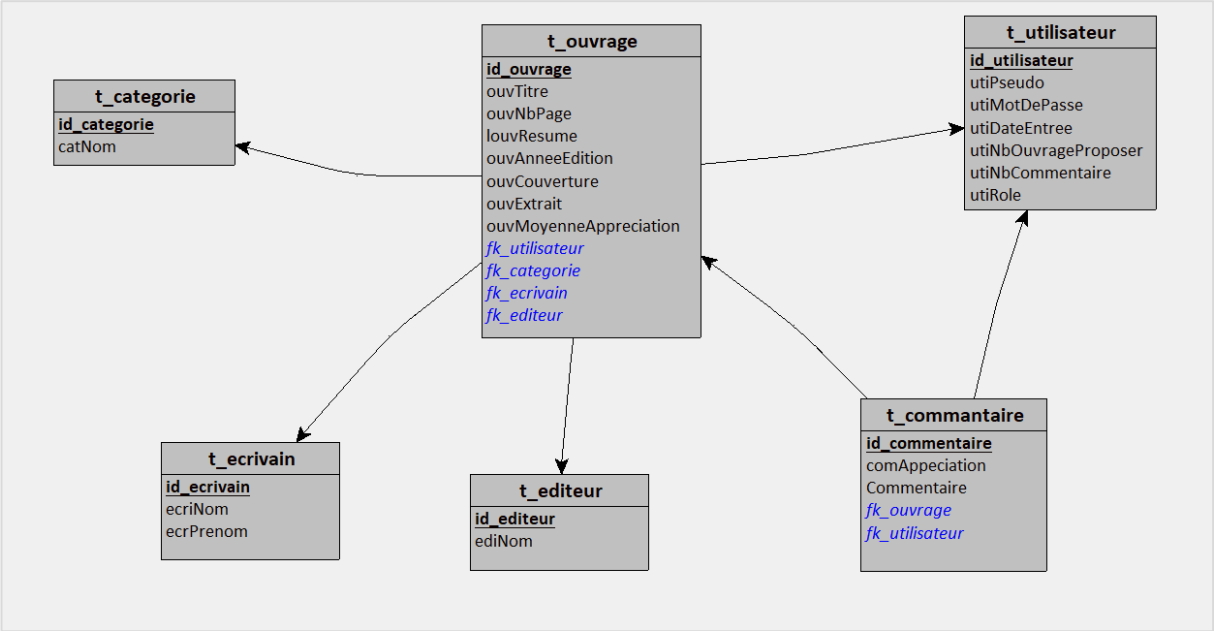
La table catégorie et la table ouvrage sont reliées avec les cardinalités 0,n et 1,1, car une catégorie peut avoir autant de 0 ouvrages qu'une infinité, et un ouvrage doit forcément avoir une catégorie.

La table écrivain et ouvrage sont reliées avec les cardinalités 0,n et 1,1 parce que un écrivain n'as pas forcement produit une ouvrage et au contraire il peut en avoir produit plusieurs. Un ouvrage doit forcément avoir un écrivain. Et c'est pareil pour l'éditeur, les cardinalités sont les mêmes et la raison de c'est cardinalité sont aussi le même un éditeur n'as pas forcément édité de livre comme il y très bien pu éditer beaucoup d'ouvrage. Et un ouvrage doit forcément avoir un éditeur.

La table Commentaire et la table ouvrage ont comme cardinalités 1,1 et 0,n car un commentaire peut seulement critiquer un ouvrage alors qu'un ouvrage peut ne pas avoir de commentaire ou peut avoir plein de commentaire.

La table commentaire et aussi relier à la table utilisateur avec les cardinalités 1,1 et 0,n parce que un commentaire est forcément écrit par un utilisateur, alors qu'un utilisateur n'a pas forcement de commentaire, mais il peut tout de même écrire 1 ou plusieurs commentaire.

Et pour finir il y a la table utilisateur qui est relier à la table ouvrage avec les cardinalités 0,n et 1,1 car un utilisateur n'est pas obligé de proposer des ouvrages mais si il veut il peut en proposer plusieurs. Et un ouvrage est et peu seulement être proposer par un seul utilisateur.



5. Structure du code

Le projet (partie code) est organisé de cette façon : dans le dossier API qui est la racine de l'API il y a un gitignore, les packages json et un dossier src, à l'intérieur de celui-ci il y a un fichier app.mjs et 4 dossiers.

Premièrement il y a le dossier auth contient deux fichiers pour faire le système d'authentification auth.mjs et private_key.mjs.

Ensuite il y a le dossier db qui contient aussi deux fichiers, mocks-livre.mjs et sequelize.mjs

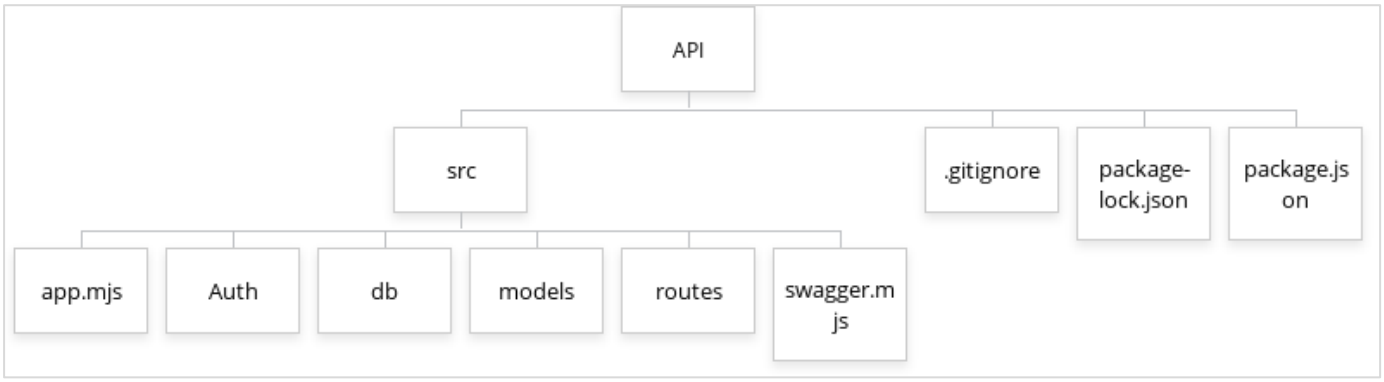
L'avant dernière dossier est le dossier models qui contient 6 fichiers, chaque fichier est une table de la base données : donc il y a les fichiers suivant t_categorie.mjs, t_commentaire.mjs, t _ecrivain.mjs, t_editeur.mjs, t_ouvrage.mjs et t_utilisateur.

Pour finir il y a le dossier routes qui contient le plus de fichier. Elle contient toutes les fonctionnalités de l'API donc il y a 12 fichiers dans ce dossier voici la liste de fichiers :

- addCategorie.mjs
- addCommentaires.mjs
- addLivre.mjs
- deleteLivre.mjs

- getAllLivreCategorie.mjs
- getCategorie.mjs
- getCategorieId.mjs
- getCommantaire.mjs
- getLivre.mjs
- getLivre.mjs
- getLivreId.mjs
- helper.mjs
- login.mjs
- updateLivre.mjs

6. Schéma de l'architecture



Réalisation

1. Gestion des rôles

Pour l'instant, il n'y a qu'un champ utiRole dans la table utilisateur où il y a "admin" et "user". Ils ne sont pour l'instant pas utilisés pour faire la gestion des rôles. Cela n'a pas été fait par manque de temps.

2. Sécurité et Authentification

Le système de token a été mis en place. Il est obligatoire pour pouvoir utiliser l'API et il ne peut être obtenu qu'en se connectant via la route /api/login. Nous avons également mis en place un hachage pour le mot de passe afin d'éviter que celui-ci soit stocké en clair dans la base de données. Ce sont les seules sécurités qui ont été mises en place en raison d'un manque de temps.

3. Swagger

Il y a la documentation des routes via swagger, la route est : /api-docs.

4. Système de recherche

Il y a un système de recherche pour les ouvrages et les catégories. Pour l'ouvrage on peut utiliser le paramètre titre pour trier les livrer par titre, il faut au minimum 2 lettre pour pouvoir effectuer une recherche. Pour la catégorie, il faudra utiliser le paramètre nom pour chercher par le nom de la catégorie, il faut au minimum 2 lettre pour pouvoir faire une recherche.

Pour les 2 il y a un paramètre limite pour limiter le nombre de résultats.

5. Insomnia

Un json est dans le dossier insomnia qui permet de tester toutes les routes.

6. Gestion des statuts http

Les statuts ont été gérés dans les routes là où cela était pertinent. Voici les types d'erreurs gérées

- 200 : succès
- 404 : le chemin spécifié n'existe pas
- 401 : un token est manquant
- 500 : lorsque le serveur n'est pas capable de délivrer les pages web

ChatGPT

Pour la partie code, ChatGPT nous a servi à comprendre des détails que nous ne savions pas ou que nous ne comprenions pas. ChatGPT nous a également servi à créer des données factices pour les insérer dans la base de données. Du côté du rapport, ChatGPT nous a seulement aidés à corriger des fautes d'orthographe et de grammaire.

Conclusion

1. Gestion du code

Nous avons utilisé GitHub comme gestionnaire de versionning. Il y a 4 branches :

- main : rassemble toutes les branches
- MiseEnPlaceAPI : c'est là où est stocké le code de l'api et insomnia
- Database : il y a le MCD/MLD/MPD
- Documentation : stocke la documentation

2. Général

Le projet est plutôt abouti, toutes les fonctionnalités créées dans le dossier fonctionnent, mais le projet n'est pas fini à 100% car la charge de travail est plutôt élevée par rapport au nombre de périodes dont nous disposons (24 périodes) et aussi par rapport au nombre de personnes par groupe. Il y a plusieurs choses qui pourraient être améliorées : Plusieurs routes (celles qui étaient obligatoires ont été faites), un système de gestion de rôles efficace, la mise en place de tests unitaires et la dockerisation.

3. Personnelle

a) Tiago

Ce projet m'a permis d'approfondir mes connaissances acquises lors du module C-295 et m'a fait découvrir la gestion des relations entre les tables. Le projet c'est bien passé et la communication avec mon collègue c'est très bien passé. Je trouve dommage de ne pas avoir eu plus de temps pour aboutir à une api complète.

b) Evin

Pour ma part, ce projet m'a permis d'apprendre de nouvelles choses et d'appliquer ce que nous avons vu durant le module. Du côté de la gestion du projet, je trouve que nous avons réussi à bien nous organiser, et la communication entre mon collègue et moi était claire. Comme l'a dit Tiago dans sa partie, malheureusement, nous n'avons pas pu totalement finir le projet car la charge de travail était trop élevée par rapport au temps à disposition.

4. Planification du projet

Pour la planification nous avons utilisé Trello comme dit dans le point 1 dans analyse, et cela nous a permis de décortiquer le projet pour mieux s'organiser, et vers la fin du projet quand on a compris que ça va être compliqué de tout finir, le fait d'avoir découper le projet auparavant nous a permis de nous réorganiser pour faire les points les plus important pour atteindre le minimum pour réussir ce projet.

Webographie

1. Docker :
   https://welovedevs.com/fr/articles/docker-node-api/