Création d’une api pour gérer une site de critique de livres

Chef de projet : M. Mveng

24 périodes (8 semaines)

Table des matières

[Introduction 2](#_Toc160543747)

[Analyse 3](#_Toc160543748)

[API (routes) 3](#_Toc160543749)

[Routes utilisateurs 3](#_Toc160543750)

[Routes livres 3](#_Toc160543751)

[Base de données (MCD, MLD) 4](#_Toc160543752)

[Architecture de la base de données 4](#_Toc160543753)

[Explications relations 4](#_Toc160543754)

[Réalisation 5](#_Toc160543755)

[Test 6](#_Toc160543756)

[Conclusion 7](#_Toc160543757)

[Gestion du code 7](#_Toc160543758)

[Qu’y a-t-il à retenir de ce projet 7](#_Toc160543759)

# Introduction

Ce projet a pour but d’approfondir nos connaissances sur les API et leurs constructions acquise en module, il consiste à créer une API comportant diverses routes permettant la gestion des données ainsi qu’un système d’authentification basé sur les jetons JWT, tout cela avec une gestion des statuts (200,3xx,4xx,5xx). Le bon fonctionnement de l’API est certifié par des tests via Insomnia.

Dans ce rapport nous allons vous expliquer comment nous avons fait pour travailler en groupe, la manière de gérer les tâches ainsi que la gestion du code, ensuite nous allons analyser et expliquer les divers incréments que nous proposons tel que l’API ou encore l’architecture de la base de données. Après nous allons expliquer la réalisation de l’API en passant par les points techniques et les fonctionnalités demandée. Finalement on conclura ce rapport en repassant sur la gestion du projet, et en l’analysant en globalité afin d’émettre des critiques constructives.

# Analyse

## API (routes)

### Routes utilisateurs

|  |  |  |  |
| --- | --- | --- | --- |
| **Action** | **Verbe http** | **URI** | **Informations à transmettre** |
| Récupère tous les utilisateurs | GET | /api/utilisateurs | Token d’authentification JWT |
| Récupère un utilisateur selon ID | GET | /api/utilisateurs/ :id | Token d’authentification JWT |
| Créer un utilisateur | POST | /api/utilisateurs | pseudo, mot de passe au format json + Token d’authentification JWT |
| Supprimer un utilisateur | DELETE | /api/utilisateurs/ :id | Token d’authentification JWT |

### Routes livres

|  |  |  |  |
| --- | --- | --- | --- |
| **Action** | **Verbe http** | **URI** | **Informations à transmettre** |
| Récupère tous les livres | GET | /api/livres | Token d’authentification JWT |
| Récupère un livre selon son ID | GET | /api/livres/ :id | Token d’authentification JWT |
| Ajout d’un livre | POST | /api/livres | Nom, extraitpdf(string), resume, anneeEdition, imageCouverture(string)  En json  + Token d’authentification JWT |
| Supprimer un livre | DELETE | /api/livres/ :id | Token d’authentification JWT |
| Mettre à jour un livre | PUT | /api/livres/ :id | Informations que l’on veut modifier en json + Token d’authentification JWT |

### Routes éditeurs

|  |  |  |  |
| --- | --- | --- | --- |
| **Action** | **Verbe http** | **URI** | **Informations à transmettre** |
| Récupère tous les éditeurs | GET | /api/editeurs | Token d’authentification JWT |
| Récupère un éditeur selon son ID | GET | /api/editeurs/ :id | Token d’authentification JWT |
| Ajout d’un éditeur | POST | /api/editeurs | Nom de l’éditeur dans un json+ Token d’authentification JWT |
| Supprimer un éditeur | DELETE | /api/editeurs/ :id | Token d’authentification JWT |
| Mettre à jour un éditeur | PUT | /api/editeurs/ :id | Informations que l’on veut modifier en json + Token d’authentification JWT |

### Routes commentaires

|  |  |  |  |
| --- | --- | --- | --- |
| **Action** | **Verbe http** | **URI** | **Informations à transmettre** |
| Récupère tous les commentaires | GET | /api/commentaires | Token d’authentification JWT |
| Récupère un commentaire selon son ID | GET | /api/commentaires / :id | Token d’authentification JWT |
| Ajout d’un commentaire | POST | /api/commentaires | Contenu commentaire et appréciation dans un json + Token d’authentification JWT |
| Supprimer un commentaire | DELETE | /api/commentaires / :id | Token d’authentification JWT |
| Mettre à jour un commentaire | PUT | /api/commentaires / :id | Informations que l’on veut modifier en json + Token d’authentification JWT |

### Routes catégories

|  |  |  |  |
| --- | --- | --- | --- |
| **Action** | **Verbe http** | **URI** | **Informations à transmettre** |
| Récupère tous les catégories | GET | /api/categories | Token d’authentification JWT |
| Récupère une catégorie selon son ID | GET | /api/categories/ :id | Token d’authentification JWT |
| Ajout d’une catégorie | POST | /api/categories | Nom catégorie dans un json + Token d’authentification JWT |
| Supprimer une catégorie | DELETE | /api/categories/ :id | Token d’authentification JWT |
| Mettre à jour une catégorie | PUT | /api/categories/ :id | Informations que l’on veut modifier en json + Token d’authentification JWT |

### Routes auteurs

|  |  |  |  |
| --- | --- | --- | --- |
| **Action** | **Verbe http** | **URI** | **Informations à transmettre** |
| Récupère tous les auteurs | GET | /api/auteurs | Token d’authentification JWT |
| Récupère un auteur selon son ID | GET | /api/auteurs/ :id | Token d’authentification JWT |
| Ajout d’un auteur | POST | /api/auteurs | Nom et prénom de l’auteur dans un json + Token d’authentification JWT |
| Supprimer un auteur | DELETE | /api/auteurs / :id | Token d’authentification JWT |
| Mettre à jour un auteur | PUT | /api/auteurs / :id | Informations que l’on veut modifier en json + Token d’authentification JWT |

### Routes login

|  |  |  |  |
| --- | --- | --- | --- |
| **Action** | **Verbe http** | **URI** | **Informations à transmettre** |
| Authentifie un utilisateur et lui donne un jeton JWT | POST | /api/login | username et password au format json |

**A REVOIR « informations à transmettre » peut-être un exemple à la place d’explication ?**

## Base de données (MCD, MLD)

### Architecture de la base de données

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

Description générée automatiquement

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

Description générée automatiquement

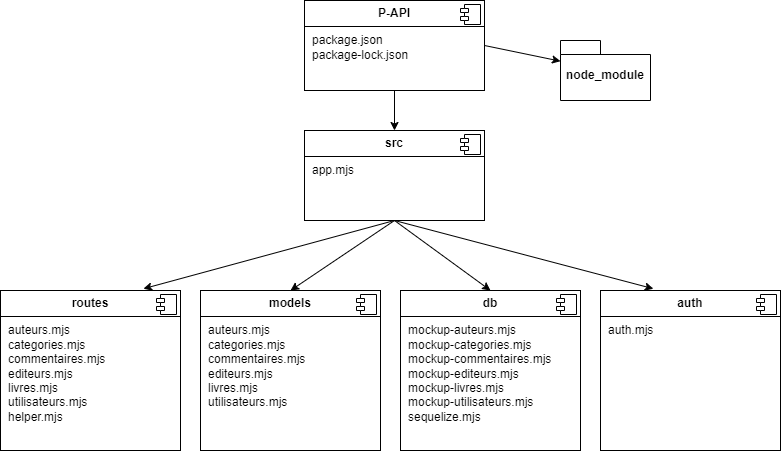
L’architecture de notre BDD (base de données) a été conçu en ayant en tête d’avoir une compréhension facile, des informations utiles et cohérentes.

### Explications relations

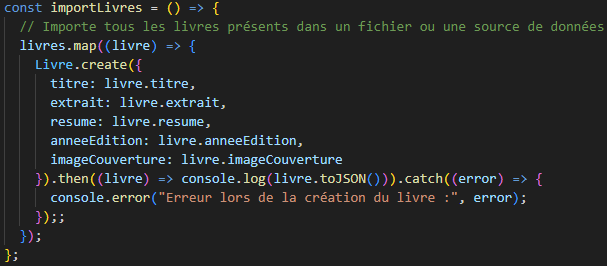
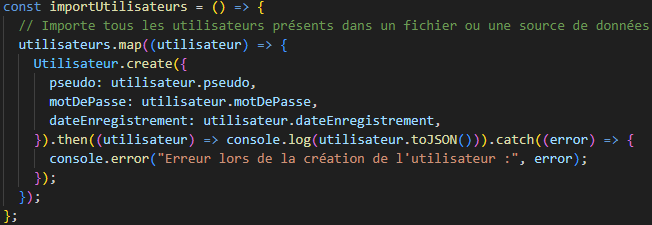
La table LIVRE contient les données des livres et est en relation avec toutes les autres tables, ces relations permettent d’obtenir plus d’informations sur certains de ses champs, par exemple on peut avoir plus d’informations sur l’auteur ou même d’obtenir des informations via des agrégations simples par exemple on peut obtenir la moyenne d’appréciation d’un livre grâce à la table commentaire

## Structure du code

Nous organisons notre code en regroupant tous les fichiers selon 4 fonctions, les codes des routes, les modèles de données (normes), ce qui est en lien avec la db et l’authentification.



### Importation des données de base

Premièrement afin d’avoir une base de données fonctionnels pour notre API nous l’a créons vide à la main puis via le fichier « sequelize.mjs » nous nous y connectons et importons les données présentes dans les divers mockups en utilisant des fonctions spécialement créer pour cette fonction :  

Il est important de noter qu’afin que ces fonctions fonctionnent correctement il faut que les noms(keys) utilisé soit les mêmes par rapport à ceux présent dans les mockups et les models.

Les « mockups » sont les fichiers qui contiennent les données aux formats json et les « models » sont des modèles de données qui définissent les normes des tables, leurs structures ainsi que des conditions, par exemple si un champ à le droit d’être vide ou pas, s’il doit contenir uniquement des chiffres, des caractères, etc…

### Les routes

Les routes sont des requêtes prédéfinies qui permettent d’effectuer certaines actions, dans notre cas pour chaque table on peut aller récupérer les données d’une table ou les données d’un champ selon id, rajouter un champ à une table, en modifier un ou encore en supprimer. Les routes elles se conforme aussi à leurs models, par exemple si on veut insérer des données dans une table il faut suivre son model.

### Authentification

Afin d’avoir un système plus sécurisé il y a un système d’authentification, premièrement il faut se connecter via la route login (pour se connecter il faut connaitre le nom d’utilisateur et le mot de passe du compte), une fois que c’est fait cela nous donne un token JWT, c’est ce token qui permet d’autoriser ou pas quelqu’un d’utiliser une route, dans notre cas toutes les routes le demande.

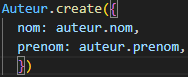
## API REST, BDD et ORM

Afin de facilité la communication entre l’API et la DB nous utilisons un ORM (Object Relational Mapping) nommé « sequelize » qui sert de pont de l’API à la DB, il contient des méthodes permettant la manipulation des informations dans une base de données, nous nous en servons afin d’interagir avec notre bdd dans nos routes ou encore à l’import des données.

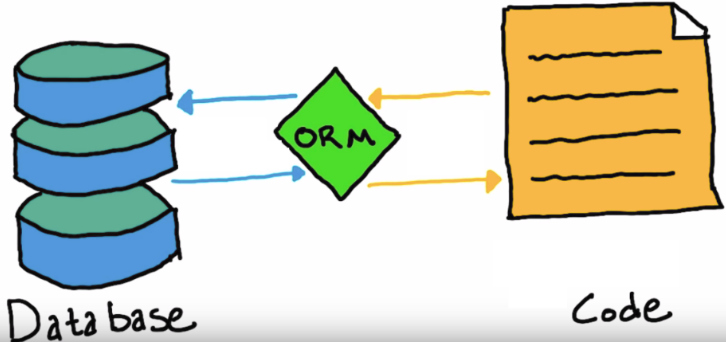
### Exemple des utilités de sequelize

Méthode sequelize permettant d’afficher toutes les données de la table « Editeur » :

Méthode sequelize qui ajoute un nouveau champ dans la table « Auteur » selon les données qu’on lui donne :



Voilà ci-dessus quelques exemples parmi tant d’autres des manières dont nous avons utilisé sequelize afin d’interagir avec la DB.



# Réalisation

### Algorithme pour gérer l’authentification

loginRouter.post("/", (req, res) => {

    User.findOne({ where: { username: req.body.username } }) // Recherche l'utilisateur par son pseudo

        .then((user) => { // Si l'utilisateur n'existe pas

            if (!user) { // On renvoie un message d'erreur

                const message = `L'utilisateur demandé n'existe pas`;

                return res.status(404).json({ message });

            }

            bcrypt // Si l'utilisateur existe

                .compare(req.body.password, user.password) // On compare le mot de passe envoyé avec celui enregistré

                .then((isPasswordValid) => {

                    if (!isPasswordValid) { // Si le mot de passe est incorrect on renvoie un message d'erreur

                        const message = `Le mot de passe est incorrecte.`;

                        return res.status(401).json({ message });

                    } else { // Si le mot de passe est correct on renvoie un message de succès et on génère un token JWT

                        // JWT

                        const token = jwt.sign({ userId: user.id }, privateKey, { // On signe le token avec la clé privée

                            expiresIn: "1y", // Le token expire dans 1 an

                        });

                        const message = `L'utilisateur a été connecté avec succès`;

                        return res.json({ message, data: user, token }); // On renvoie le token avec les données de l'utilisateur

                    }

                });

        })

        .catch((error) => { // Si une erreur survient on renvoie un message d'erreur

            const message = `L'utilisateur n'a pas pu être connecté. Réessayez dans quelques instants`;

            return res.json({ message, data: error });

        });

});

Voilà notre route de login, cette route demande 2 informations, un username et un password, ensuite elle cherche dans la base de données si le username correspondant à un username dans la bdd, si ce n’est pas le cas elle renvoie un message d’erreur, et si c’est le cas elle va comparer le mot de passe donné avec celui enregistré dans la bdd s’il ne correspond pas elle va avertir que le mot de passe est incorrect et si les deux sont identiques alors elle va renvoyée « L’utilisateur a été connecté avec succès » suivis du nom d’utilisateur et de son token, l’utilisateur n’aura donc plus qu’à utiliser son token pour certifier son identité aux yeux de la db.

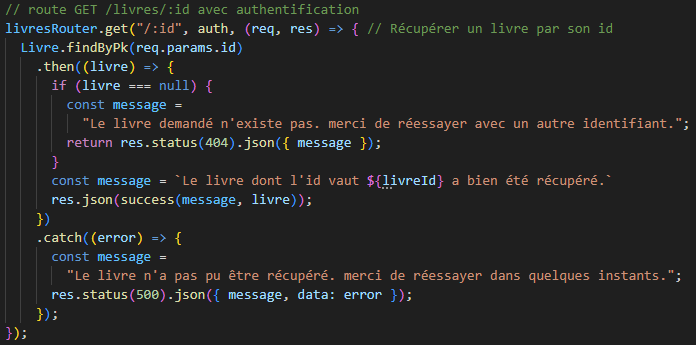
### Explication fonctionnalité technique demandé

#### Affiche toutes les données de tous les livres

Cette route affiche tous les champs de tout les livres stocker dans la table Livre ainsi qu’un message de succès ou d’erreur. Elle requiert une authentification, il faut donc lui donner un token d’un compte quand on veut l’utiliser.

### 

#### Affiche les données d’un livre

Cette route affiche tous les champs d’un livre selon l’id qu’on lui donne ainsi qu’un message de succès ou d’erreur. Elle requiert une authentification, il faut donc lui donner un token d’un compte quand on veut l’utiliser.

#### Affiche toutes les données des livres d’une certaine catégorie

#### 

Cette route affiche tous les champs de tous les livres d’une catégorie selon l’id(id\_Categorie) qu’on lui donne ainsi qu’un message de succès ou d’erreur. Elle requiert une authentification, il faut donc lui donner un token d’un compte quand on veut l’utiliser.

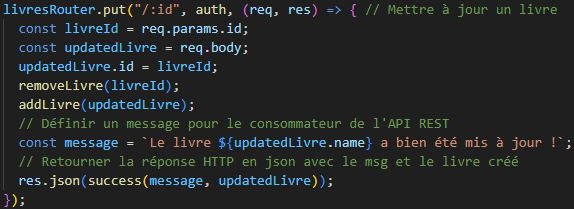
#### Ajouter un livre dans la db

#### 

Cette route permet d’ajouter un livre à la db, pour se faire il faut lui donner les informations requises selon le modèle de données au format json, elle renvoie un message de succès ou d’erreur selon si elle a fonctionné. Elle requiert une authentification, il faut donc lui donner un token d’un compte quand on veut l’utiliser.

#### Modifier les données d’un livre dans la db

Cette route permet de modifier les informations d’un livre, pour se faire il faut lui donner les informations requises selon le modèle de données au format json, il faut aussi dans l’URI lui spécifier l’id du livre qu’on veut modifier, elle renvoie un message de succès ou d’erreur selon si elle a fonctionné. Elle requiert une authentification, il faut donc lui donner un token d’un compte quand on veut l’utiliser.



#### Supprimer un livre de la db

#### 

Cette route permet de supprimer un livre de la db, pour se faire il faut lui donner l’id du livre qu’on le veut supprimer, elle renvoie un message de succès ou d’erreur selon si elle a fonctionné. Elle requiert une authentification, il faut donc lui donner un token d’un compte quand on veut l’utiliser.

# Test (voir insomnia)

# Conclusion

## Gestion du code

Dans ce projet pour pouvoir travailler simultanément sans se soucier des possibles conflit github ou autres nous avons en grande partie travailler sur des parties différentes en générales ou nous travaillons ensemble « en pair programming »

## Qu’y a-t-il à retenir de ce projet

Ce projet a été un bon moyen de consolider les connaissances fraichement acquises en module, des connaissances qui vont nous êtes très utile pour la suite, avec l’apprentissage de connaissances fondamentales sur les API, l’ORM « sequelize » qui nous paraissent déjà utile voir essentiel dans notre futur métier d’informaticien.

Webographie