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. Finalement il y a aussi une documentation automatique crée grâce à Swagger (voir annexe).

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 |  |
| Récupère un utilisateur selon ID | GET | /api/utilisateurs/ :id |  |
| Créer un utilisateur | POST | /api/utilisateurs | pseudo, mot de passe au format json |
| Supprimer un utilisateur | DELETE | /api/utilisateurs/ :id |  |

### Routes livres

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

### Routes éditeurs

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

### Routes commentaires

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

### Routes catégories

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

### Routes auteurs

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

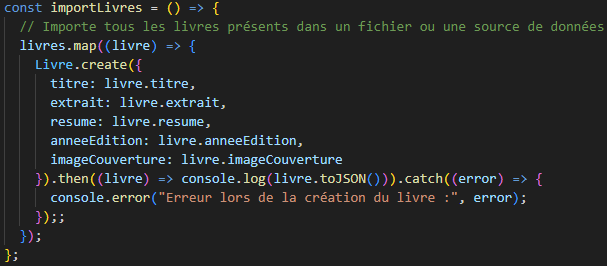
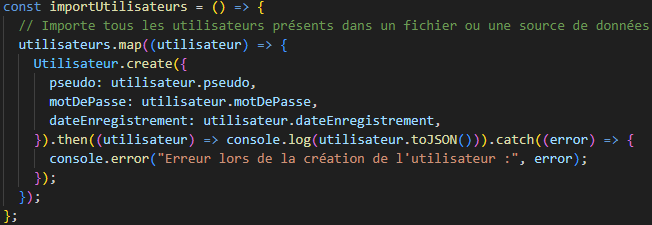
## Structure du code

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.

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

Description générée automatiquement

### 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.

**A CONTINUER (Auth + |relation/liaison|)**

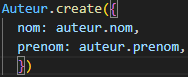
## API REST, BDD et ORM

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.

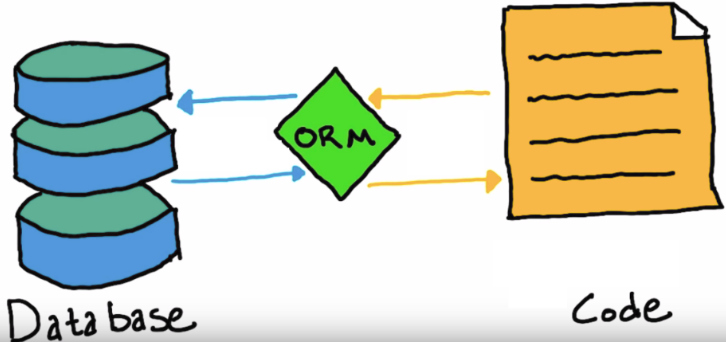
### Exemple des utilités de sequelize

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

# Test

# Conclusion

## Gestion du code

Afin de pouvoir travailler simultanément sur notre API nous avons choisi d’utiliser Github où nous travaillions sur la même API, cependant nous nous répartissions les tâches afin de ne pas travailler sur les mêmes parties du codes, grâces à cela nous avons rencontré peu de conflits facilement réglables directement.

**A REVOIR**

## Qu’y a-t-il à retenir de ce projet

Webographie