Logotipo, Icono

Descripción generada automáticamente

**ECOLE TECHNIQUE  
ECOLE DES MÉTIERS LAUSANNE**

19 février 2024

Adrian toledo & mathis BOTTEUA

Enseignant : ANTOINE MVENG

« P\_WEB 295 »

LOVBOOKS – Passion Lecture

Réaliser le backend d’une application permettant de partager sa passion pour la lecture

Une image contenant texte, logo, Police, Graphique

Description générée automatiquement

Table des matières

[1. Introduction 2](#_Toc161101913)

[2. Analyse 2](#_Toc161101914)

[2.1. Planification 2](#_Toc161101915)

[2.2. Base de données 3](#_Toc161101916)

[2.3. API REST 5](#_Toc161101917)

[2.4. Structure du code 5](#_Toc161101918)

[2.5. Routes 6](#_Toc161101919)

[3. Réalisation 9](#_Toc161101920)

[3.1. System d’authentification 9](#_Toc161101921)

[3.2. Gestion des statuts HTTP 9](#_Toc161101922)

[3.3. Fonctionnalités techniques 10](#_Toc161101923)

[4. Test – Insomnia 11](#_Toc161101924)

[5. Conclusions 11](#_Toc161101925)

[6. Webographie 12](#_Toc161101926)

# Introduction

Le projet P\_Web\_295 est destiné à créer le backend d'un site web pour l'enregistrement et la recherche de livres par les utilisateurs.

Ce site, appelé LovBooks, offre un service de catalogue de livres géré par les utilisateurs. Sur cette page, vous trouverez une liste de livres, avec leurs informations, les évaluations et les commentaires faits par les utilisateurs enregistrés.

Pour la construction du backend du site web, une API REST a été développée afin de gérer les informations de la base de données et les utilisateurs avec leur système d'authentification.

Sur une période de 24 périodes, il est prévu d'obtenir un code API complet qui permette d'introduire des routes et d'accéder à la base de données pour utiliser ou modifier les données.

A travers ce rapport, nous allons introduire les notions appliquées dans ce projet qui ont été apprises dans le module Web\_295. Ce module dédié au développement web backend, nous offre les nations de l'utilisation et du codage pour les services d'une API REST.

# Analyse

## Planification

La planification de ce projet a été réalisée à l'aide de la plateforme Trello. Grâce à cette plateforme, nous avons divisé le développement du projet en suivant le modèle Kanban pour avoir une liste des tâches à faire (TO DO), une liste des tâches en cours (In Progress) et enfin une liste des tâches terminées (DONE).

Suivant ce modèle, nous avons divisé le projet en 5 sections principales :

- Planification et division du travail

- L'implémentation de GitHub pour le travail coopératif.

- Réalisation du rapport de projet

- Base de données : qui comprend le développement des modèles MCD, MLD, MPD et la connexion de l'API à un serveur.

- API REST, cette section est divisée en petites tâches de développement de l'API telles que : installation des extensions, création des routes, validation des données, recherche, système d'authentification, documentation Swagger, Test Insomnia.

## Base de données

La base de données est composée de 7 tables, chacune est spécifique à une autre, cela permet d’assurer une structure cohérente. L'objectif est de simplifier l'utilisation et la compréhension de la base de données. La table principale « t\_book » est liée avec les autres et permet d’obtenir n’importe quelle donnée d’une autre table.

Les tables "t\_Publisher" et "t\_Author" ont été isolées, car un auteur n'est pas toujours associé à un livre, et vice versa un éditeur peut ne pas avoir de livre lié. Cette séparation permet une représentation plus précise des relations entre les champs.

La table "t\_Category" permet d’assurer une cohérence entre les livres, cela permet aussi à une catégorie de regrouper plusieurs livres. Cela simplifie la gestion des informations liées aux catégories et à leur association avec les livres qui leurs correspondent.

La séparation des tables "t\_Comment" et "t\_Assessment" permet de faire un commentaire sans nécessairement effectuer une évaluation, et vice versa. Cela offre de l’agilité dans la collecte des retours des utilisateurs.

La table "t\_Customer", permet de garder la confidentialité de ses informations, ce qui évite toute interactions directes dans d'autres tables et renforçe la sécurité des données.

Les relations entre les tables "t\_Book", "t\_Comment" et "t\_Assessment" permettent d'obtenir des informations détaillées sur le moment des publications d'évaluations, des commentaires ou des livres. Cette liaison facilite la visualisation des événements des livres et des utilisateurs.

En résumé, la base de données à une structure cohérente et simple, elle prend en compte les champs de chaque table et les relations entre elles. Ce qui permet la compréhension et la visualisons de tous les événement des tables.

Ci-dessous les diagrammes de notre base de données en MLD et MCD :

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamenteDiagrama

Descripción generada automáticamente

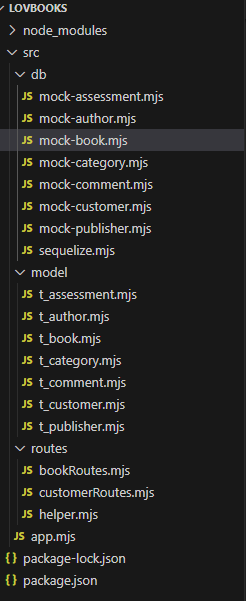
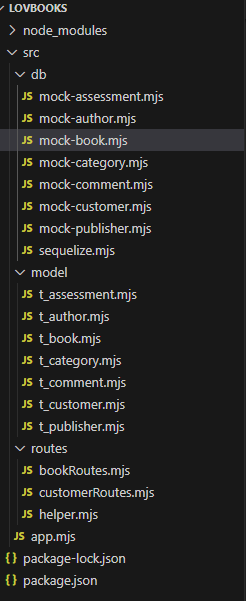
## API REST

LovBooks gère une grande quantité de données et l'utilisation d'une API nous permet de relier facilement nos données sur les livres et les utilisateurs à nos pages web.

Pour le développement de cette API, nous avons créé un code qui structure les tables de la base de données (Auteur, Livre, Catégorie, Commentaires, Publisher, Utilisateurs et Évaluation ), gère les routes pour l'interaction avec la base de données, un service d'authentification et une documentation du code en Swagger.

## Structure du code

L'api de LovBooks est divisée en 5 branches principales :



* Le projet est réalisé à partir de l’utilisation de node.js et des fichiers package.json. Un fichier pour la documentation en Swagger, et L'ensemble des routes de la page web et du serveur sont gérées par le script principal app.mjs.
* Une image contenant texte, capture d’écran, Police, menu

  Description générée automatiquementUn dossier pour la base de données qui contiendra les données préenregistrées des livres, des utilisateurs et de leurs critiques. En plus d'un sequelize qui se connectera à notre serveur et gérera l'entrée et la sortie des données.

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

Description générée automatiquement

* Une branche pour la création de modèles par table qui respectent les structures et les données de notre base MySQL db\_lovbooks.
* Une image contenant texte, capture d’écran, Police, nombre

  Description générée automatiquementPour le système d'enregistrement et de connexion, ce fichier d'authentification est utilisé.

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

* Enfin un dossier destiné à la création des routes selon leurs fonctions pour chaque table et connexion.

## Routes

La construction des routes est définie en fonction des fonctionnalités que nous souhaitons utiliser pour gérer les données dans nos tables et notre base de données. De cette manière, une méthode sera créée en fonction de la demande par itinéraire. Dans certaines fonctions comme la recherche de livres ou les interactions de commentaires, l'authentification de l'utilisateur est nécessaire.

En matière de sécurité, si l'utilisateur n'obtient pas l'accès à sa route ou problèmes de connexion, une erreur d'état http correspondante s'affichera. Vous trouverez ci-dessous une liste de tables avec la route à utiliser.

1. **BooksRouter**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Fonctionnalité** | **Route** | **JSON** | **Message de Réponse** | **Message d’Erreur** |
| Liste de tous les livres | GET http://localhost:3000  /api/books/ |  | «La liste des livres a bien été récupérée » | « Une erreur est survenue lors de la récupération des livres » |
| Rechercher un livre par son ID | GET http://localhost:3000  /api/books/(id) |  | « Le livre don't l'id vaut 3 a bien été récupéré » | « Le livre demandé n'existe pas. Merci de réessayer avec un autre identifiant » |
| Rechercher un livre par son nom | GET http://localhost:3000  /api/books?title=[nom] |  | « Il y a 1 livres qui correspondent au terme de la recherche » | « Il y a 0 livres qui correspondent au terme de la recherche » |
| Obtenir tous les commentaires d’un livre | GET http://localhost:3000  /api/books/  (id)/comments |  | « Il y a 6 comments qui correspondent au terme de la recherche » | « Le livre demandé n'existe pas. Merci de réesayer avec un autre identifiant » |
| Obtenir toutes les notes d’un livre | GET http://localhost:3000  /api/books/(id)/notes |  | « Il y a 2 notes qui correspondent au terme de la recherche | « Le livre demandé n'existe pas. Merci de réessayer avec un autre identifiant » |
| Créer un livre | POST http://localhost:3000  /api/books/ | {  "title" : "Heidi 2 : The end",  "number\_of\_pages" : 161,  "year\_of\_publication" : 1999,  "cover\_image":"exe",  "extract\_pdf":" exe ",  "category\_id":11,  "summary": " exe "  } | « Le livre Fondation dont l'id vaut 2 a été mis à jour avec success » | « Le livre demandé n'existe pas. Merci de réessayer avec un atre identifiant. » |
| Commenter un livre | POST http://localhost:3000  /api/books/(id)/comments |  | « Le commentaire 9 a bien été créé ! « |  |
| Evaluer un livre | POST http://localhost:3000  /api/books/(id)/notes |  |  |  |
| Modifier un livre | PUT http://localhost:3000  /api/users/(id) |  |  |  |
| Supprimer un livre | DELETE http://localhost:3000  /api/books/(id) |  |  |  |

1. **CustomersRouter**

|  |  |
| --- | --- |
| **Fonctionnalité** | **Route** |
| Liste de tous les utilisateurs | GET http://localhost:3000/api/users/ |
| Rechercher un utilisateur par son ID | GET http://localhost:3000/api /users/(Id) |
| Rechercher un utilisateur par son nom | GET http://localhost:3000/api/users?pseudo=[nom] |
| Obtenir tous les libres d’un utilisateur | GET http://localhost:3000/api//users/(id)/books |
| Créer un utilisateur | POST http://localhost:3000/api /users/ |
| Modifier un utilisateur | PUT http://localhost:3000/api /users/(id) |
| Supprimer un utilisateur | DELETE http://localhost:3000/api/users/(id) |

1. **LoginRouter**

|  |  |
| --- | --- |
| **Fonctionnalité** | **Route** |
| Vérification de la connexion | POST http://localhost:3000/api/login/ |

1. **CategoryRouter**

|  |  |
| --- | --- |
| **Fonctionnalité** | **Route** |
| Liste de tous les catégories | GET http://localhost:3000/api/categorys/ |
| Rechercher une catégorie par son ID | GET http://localhost:3000/api/categorys/(Id) |
| Rechercher une catégorie par son nom | GET http://localhost:3000/api/categorys?name=[nom] |
| Obtenir tous les livres de cette catégorie | GET http://localhost:3000/api/categorys/(id)/books |
| Créer une catégorie | POST http://localhost:3000/api/categorys / |
| Modifier une catégorie | PUT http://localhost:3000/api/categorys /(id) |
| Supprimer une catégorie | DELETE http://localhost 3000/api/categorys /(id) |

1. **AuthorsRouter**

|  |  |
| --- | --- |
| **Fonctionnalité** | **Route** |
| Liste de tous les auteurs | GET http://localhost:3000/api/authors/ |
| Rechercher un auteur par son ID | GET http://localhost:3000/api /authors/(Id) |
| Rechercher un auteur par son nom | GET http://localhost:3000/api/authors?name=[nom] |
| Rechercher un auteur par son prénom | GET http://localhost:3000/api/authors?first\_name=[prénom] |
| Obtenir tous les livres de cet auteur | GET http://localhost:3000/api/authors/(id)/books |
| Créer un auteur | POST http://localhost:3000/api/author / |
| Modifier un auteur | PUT http://localhost:3000/api authors/(id) |
| Supprimer un auteur | DELETE http://localhost:3000/api/authors /(id) |

1. **PublisherRouter**

|  |  |
| --- | --- |
| **Fonctionnalité** | **Route** |
| Liste de tous les éditeurs | GET http://localhost:3000/api/publishers/ |
| Rechercher un éditeur par son ID | GET http://localhost:3000/api/ publishers /(Id) |
| Rechercher un éditeur par son nom | GET http://localhost:3000/api/publishers?name=[nom] |
| Créer un éditeur | POST http://localhost:3000/api/publishers / |
| Modifier un éditeur | PUT http://localhost:3000/api /publishers /(id) |
| Supprimer un éditeur | DELETE http://localhost:3000/api/publishers /(id) |

1. **AssessmentRouter**

|  |  |
| --- | --- |
| **Fonctionnalité** | **Route** |
| Liste de tous les évaluations | GET http://localhost:3000/api/assessments/ |
| Rechercher une évaluation par son ID | GET http://localhost:3000/api/assessments/(Id) |
| Créer une évaluation | POST http://localhost:3000/api/assessments/ |
| Modifier une évaluation | PUT http://localhost:3000/api/assessments/(id) |
| Supprimer une évaluation | DELETE http://localhost:3000/api/assessments/(id) |

1. **CommentRouter**

|  |  |
| --- | --- |
| **Fonctionnalité** | **Route** |
| Liste de tous les évaluations | GET http://localhost:3000/api/assessments/ |
| Rechercher une évaluation par son ID | GET http://localhost:3000/api /assessments/(Id) |
| Créer une évaluation | POST http://localhost:3000/api /assessments/ |
| Modifier une évaluation | PUT http://localhost:3000/api /assessments/(id) |
| Supprimer une évaluation | DELETE http://localhost:3000/api/assessments/(id) |

# Réalisation

## System d’authentification

Les routes de les utilisateurs sont gérées à la demande de l'utilisateur admin et de ses droits d'accès. Pour réaliser cette fonction, un système d'authentification « auth » a été créé selon le modèle de la demande d'un nom d'utilisateur et d'un mot de passe confidentiel. En saisissant ces données, les utilisateurs reçoivent un jeton JWT à durée limitée de valeur Bearer qui leur donnera accès à l'itinéraire demandé. L’exécution de ses jetons est réalisée grâce à la dépendance Jswebtoken.

Un point important à suivre est la conservation et confidentialité des données des utilisateurs et des mots de passe. Pour sécuriser les données, nous avons utilisé à la dépendance Bcrypt pour le chiffrement et la vérification des données destinées au mot de passe de l’utilisateur. Dans le code, on utilise le fichier login.mjs qui, en fonction des informations saisies dans la route, va comparer les mots de passe et leur authentification avec la méthode compare().

## Gestion des statuts HTTP

Pour savoir si une requête HTTP a été correctement exécutée, il existe des statuts HTTP qui identifieront si la ligne de routage a été appliquée correctement ou si une erreur s’est produite.

Pour couvrir les éventuelles erreurs qui peuvent être commises dans une API web, nous avons couvert les routes les plus courantes.

L’erreur la plus courante causée par le client est l’erreur 404, qui se produit lorsque le serveur ne trouve pas la ressource demandée. Et aussi l’erreur 401 et 404 qui identifie les autorisations des utilisateurs et si existe.

Du côté des erreurs du serveur, un message d’erreur 500 s’affiche lorsque le serveur a rencontré un problème et n’est pas en mesure de répondre à la demande.

Dans le code, le « then & cath » est utilisé pour identifier l’erreur et faire un retour du «statut» de l’erreur et un message d’explication. Exemple :

return res.status(404).json({"Le livre demandé n'existe pas. Merci de réessayer avec un atre identifiant."})

return res.status(401).json({ `L'utilistaeur n'est pas autorisé à accéder à cette ressource.`});

return res.status(404).json({L'utilisateur demandé n'existe pas});

res.status(500).json({"Le livre n'a pas pu être mis à jour. Merci de réessayer dans quelques instants", data:error});

## Fonctionnalités techniques

* Documentation Swagger

Une image contenant texte, capture d’écran, logiciel, nombre

Description générée automatiquement

* Dépendances :

La création de ce code API a été réalisée grâce aux dépendances installées et utilisées. Voici une liste des dépendances et ses versions :

Une image contenant texte, capture d’écran, Police

Description générée automatiquement

# Test – Insomnia

Pour tester les routes de notre API, nous avons utilisé la plateforme Insomnia. Cette plateforme dédiée au test d’API en local et cloud, offre un environnement dédié à l’envoi et à la réception des réponses de nos routes en fonction de leurs opérations

Une image contenant texte, capture d’écran, logiciel, conception

Description générée automatiquementListe de routes sur Insomnia REST :

Une image contenant texte, capture d’écran, logiciel, Logiciel multimédia

Description générée automatiquementExemple de route :

# Conclusions

LovbooksAPI offre plusieurs routes pour la manipulation et l'utilisation des informations de notre base de données. Pour la réalisation de ce projet, il a été très utile de construire une modélisation des données dans un schéma de style MCD et MLD. Ce document a servi de guide pour la construction de l'API et de la base de données. Un autre point important a été la bonne communication au sein de l'équipe, la division du travail et l'utilisation de GitHub pour partager et créer le code ensemble.

Cette API entièrement fonctionnelle attend d'être utilisée pour être reliée à une page web et à son front-end. Ce projet offre une base de données locale déjà créée, de sorte qu'elle peut être testée pour connaître ses fonctionnalités à travers des supports tels qu'Insomnia Rest.

De plus, un modèle de données a été créé pour chaque type de table, ce qui assure une bonne intégrité de l'insertion des données et de la création des tables.

De plus, pour chaque demande de route, un système de validation et d'erreur a été créé. Celui-ci fournit à l'utilisateur un message en fonction de l'erreur rencontrée.

La dernière fonction importante est la création d'un système d'authentification avec un jeton de validation. Ce système nous a permis de créer des profils de connexion, et de tester la sécurité de nos routes destinées exclusivement aux utilisateurs connectés.

* Conclusion d'Adrian :

Mon rôle principal a été la création de la base de code, avec diverses fonctions telles que l'authentification, les routes de départ, la connexion à la base de données, l'intégration dans la plateforme Insomnia, affichage de message de validation d’erreur sur le model et la création de routes imbriquées. La plus grande difficulté de ce projet a été de pouvoir utiliser et créer des routes imbriquées, mais grâce à notre chef de projet et à la documentation de Sequelize, cela a été réalisé sans le moindre problème.

* Conclusion de Mathis

Mon rôle principal dans ce projet a été la gestion de la base de données, comprenant la gestion des modèles, des mocks, la connexion et l'insertion des données dans la base de données, ainsi que les relations entre les modèles afin que cela soit cohérent avec le MCD. Mon second rôle a été la création des routes avec la gestion des erreurs 404 et 500. Nous n'avons pas rencontré beaucoup de problèmes dans ce projet, car avec Adrian, nous avons communiqué sur toutes les tâches qui étaient faites et sur celles sur lesquelles nous travaillions chacun. Grâce à Trello, la planification et la visualisation de l'avancement du projet ont permis qu'il se déroule sans difficulté. Si je devais noter un problème, cela serait la création et l'utilisation des routes imbriquées ainsi que les erreurs de génération de données dans le mock. Mais cela a pu être résolu par la suite pour donner un produit fonctionnel

# Webographie

* Documentation Sequelize : <https://sequelize.org/>
* Guide de construction de l'API : « Steps[1-14] » fournies par le chef de projet
* Traducteur : <https://www.deepl.com/>
* ChatGPT : <https://chat.openai.com> - L'utilisation de chatGPT a été utilisée pour construire des documents Mocks fictifs afin de créer une fausse base de données.