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](#_Toc160435507)

[2. Analyse 2](#_Toc160435508)

[3. Réalisation 2](#_Toc160435509)

[4. Test 2](#_Toc160435510)

[5. Conclusions 2](#_Toc160435511)

[6. Webographie 2](#_Toc160435512)

# Introduction

# 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.

## BD

Diagrama

Descripción generada automáticamente

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamente

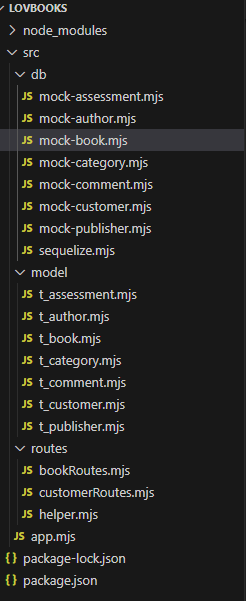
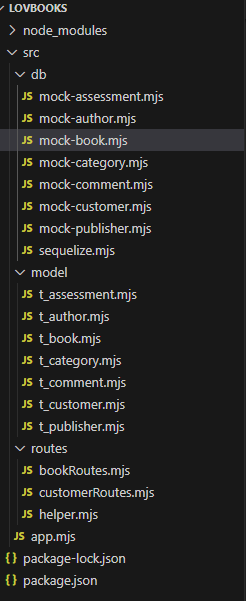
## API REST

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, menu

  Description générée automatiquementEnfin 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. **BookRoutes**

|  |  |
| --- | --- |
| **Fonctionnalité** | **Route** |
| Liste de tous les livres | GET /api/books/ |
| Rechercher un livre par son ID | GET /api /books/(Id) |
| Créer un livre | POST /api /books/ |
| Modifier un livre | PUT /api /users/(id) |
| Supprimer un livre | DELETE /api/books/(id) |

1. **CustomerRoutes**

|  |  |
| --- | --- |
| **Fonctionnalité** | **Route** |
| Liste de tous les utilisateurs | GET /api/users/ |
| Rechercher un utilisateur par son ID | GET /api /users/(Id) |
| Créer un utilisateur | POST /api /users/ |
| Modifier un utilisateur | PUT /api /users/(id) |
| Supprimer un utilisateur | DELETE /api/users/(id) |

1. **LoginRoutes**

|  |  |
| --- | --- |
| **Fonctionnalité** | **Route** |
| Vérification de la connexion | POST /api/login/ |

# Réalisation

## System d’authentification

Les routes sont gérées à la demande de l'utilisateur et de ses droits d'accès. Pour réaliser cette fonction, un système d'authentification 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
* Dependences
* Connexion DB
* Correction de nombres introducidos
* La création de ce code API a été réalisée grâce aux dépendances installées et utilisées. Voici une liste des dépendances et ses versions :

# Test

* Presentacion de Insomnia

# Conclusions

* General
* Mathis
* Adrian

# Annex

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

Description générée automatiquement

Imagen que contiene Interfaz de usuario gráfica

Descripción generada automáticamenteDiagrama

Descripción generada automáticamente

# Webographie