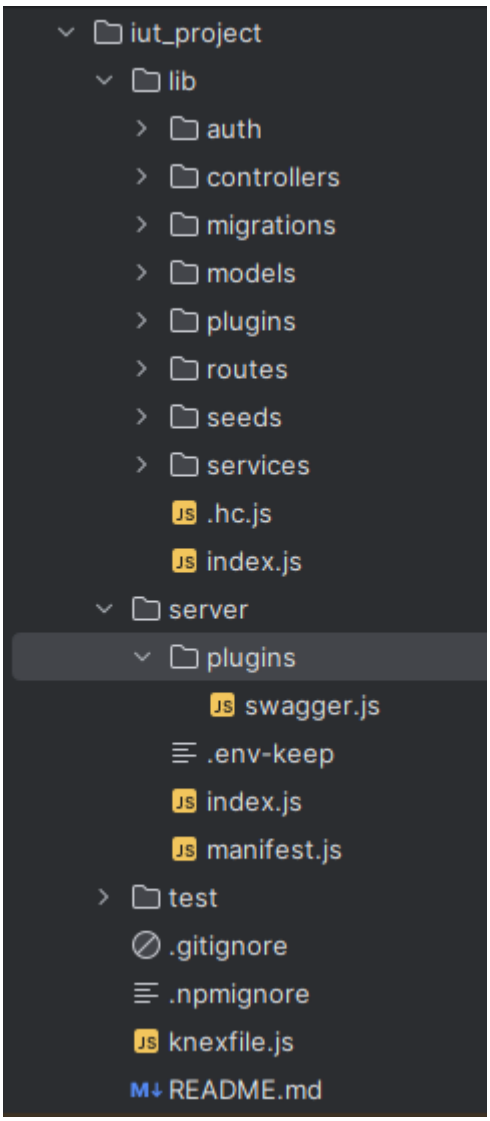
Compte Rendu du Projet

Aigueperse Pauline G8

Introduction :

Ce projet est une application de streaming IPTV développée dans le cadre du module R6A.05 de la troisième année du BUT Informatique à Limoges. Son objectif est de permettre aux utilisateurs d'accéder à une bibliothèque de films, d'en gérer les favoris et de recevoir des notifications par e-mail. Seul le back-end de cette application a été faite, il n'y a aucun front-end.

Architecture du projet du back-end :



Fonctionnalités :

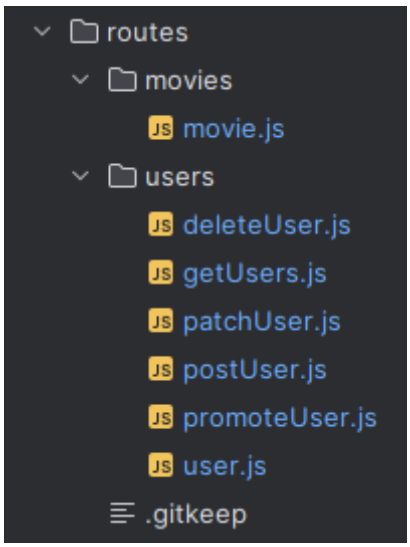
L'application IPTV propose plusieurs fonctionnalités essentielles :

- **Gestion des utilisateurs et de l'authentification** :
    - Inscription et connexion des utilisateurs avec JWT pour sécuriser les sessions.
    - Gestion des rôles, permettant aux administrateurs de gérer les films et les utilisateurs.
- **Gestion des films** :
    - Ajout, modification et suppression de films.
    - Consultation des films disponibles et des détails de chaque film.
    - Gestion d'une liste de favoris pour chaque utilisateur.
- **Export CSV via un message broker (RabbitMQ)** :
    - Un administrateur peut demander un export CSV de la liste des films.
    - Le fichier CSV est généré et envoyé via RabbitMQ.
- **Envoi de notifications par e-mail avec Nodemailer** :
    - Notifications aux utilisateurs pour des mises à jour spécifiques.
    - Envoi du fichier CSV de l'export aux administrateurs.
- **API REST sécurisée avec Hapi.js** :
    - Documentation disponible via Swagger.
    - Gestion des accès sécurisée grâce à des middlewares et JWT.

Choix Techniques :

Routes et endpoints

Concernant le dossier des routes j'ai séparé les différentes règles métier en deux dossiers pour plus de logique. J'avais fait un fichier pour chaque endpoint de l'entity User, mais pour Movie j'ai mis tous les endpoints dans le même fichier. Personallement je préfère largement séparer chaque endpoint en plusieurs fichiers, mais cela va dépendre des outils utilisés. J'ai pour habitude d'utiliser Api Platform et tout les endpoints sont déclarés dans la même entity. Du coup ici j'ai décidé de faire des deux façons différentes, je ne crois pas qu'il y a une façon meilleure qu'une autre, cela dépend surtout de ses goûts personnels :



Back-end avec Hapi.js

Le choix de Hapi.js pour le back-end s'explique par sa flexibilité et sa robustesse dans la gestion des routes et des plugins. Contrairement à Express, Hapi offre un cadre plus structuré pour la validation des requêtes et la gestion de l'authentification.

Message Broker avec RabbitMQ

L'utilisation de RabbitMQ pour la gestion des exports CSV permet de découpler les traitements lourds de l'API. Ainsi, la génération et l'envoi des fichiers CSV ne bloquent pas les requêtes des utilisateurs et garantissent une meilleure scalabilité.

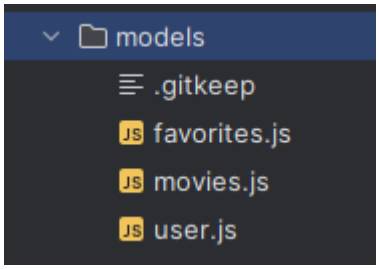
AdminController

J'ai fait un fichier AdminController qui est en soit un service mais je ne l'ai pas mis dans le dossier service car il permet aussi de gérer les accès, c'est-à-dire qu'il va renvoyé une erreur si dans la request l'user n'a pas le rôle admin.

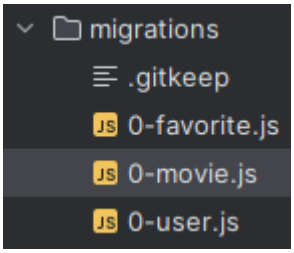
```
module.exports = class AdminController extends Service {
  async requestCsvExport(request, h) :Promise<...> { Show u
```

Modèles

J'ai de plus fait un modèle favorite, je ne sais pas si c'était vraiment pertinant (on aurait pu faire une liason doctrine ManyToOne avec un attribut favorite et une array de movie pour un utilisateur).



Migrations



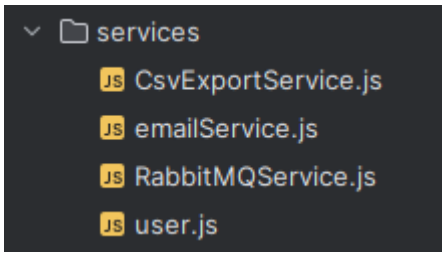
Globalement, tout les fichiers de migrations sont sous cette forme :

```
'use strict';

module.exports = {

  async up(knex) :Promise<void> {

    await knex.schema.createTable('movies', (table) :void => {
      table.increments('id').primary();
      table.string('title').notNullable();
      table.text('description').notNullable();
      table.date('release_date').notNullable();
      table.string('director').notNullable();
      table.timestamp('created_at').defaultTo(knex.fn.now());
      table.timestamp('updated_at').defaultTo(knex.fn.now());
    });
  },

  async down(knex) :Promise<void> {
    await knex.schema.dropTableIfExists('movies');
  }
};
```

On met obligatoirement une promise up et down, pour les commandes npx.

Services

J'ai fait 4 différents services ; un pour générer le fichier Csv, un pour gérer l'envoie de différents mail (bienvenue, update etc.), un pour gérer le message broker et pour finir un service qui va créer un user et le fetch dans la database.



Déploiement et Exécution

L'application est conçue pour être facilement déployable grâce à Docker et npm. Elle repose sur un fichier `.env` pour la configuration des variables sensibles. Les étapes d'installation et d'exécution sont détaillées dans le fichier README du projet.

Conclusion

Le choix de RabbitMQ pour la gestion des tâches asynchrones et l'intégration de Hapi.js permettent une gestion efficace des utilisateurs et des films. Ce projet constitue une bonne base pour une plateforme de streaming évolutive et sécurisée. Il permet de plus de voir toutes les bases de NodeJS, avec une (H)API, des endpoints, routes, controllers, services, message broker, migrations etc. La mise en place de génération de token Jwt est très interessante car toute web application utilise à minima de genre de connexion. Ce projet pourrait être compléter avec une partie front-end, mais là n'était pas le principal sujet.