Groupe F - Synthèse

Hudayfa Koujdal - Hugo Lignères - Soukaina Mourabit - Samantha Ortega

UE L315 - Semaine 4

Table des matières

1 Introduction

Pour ce mini-projet, nous devions implémenter une médiathèque, qui permet à un utilisateur d'emprunter et de rendre les documents affichés.
Les données à utiliser proviennent d'une collection «`documents`», stockée dans une base de données de type NoSQL, elle-même hébergée dans un cluster MongoDB Atlas.
Pour créer la logique de l'application et interagir avec les données de la collection, nous avons utilisé Node.js avec le framework Express.js.

Nous allons d'abord voir comment nous nous sommes organisés pour ce travail de groupe. Puis, dans une deuxième partie, nous reviendrons sur ce que nous avons produit. Enfin, en conclusion, chaque membre du groupe fera un retour d'expérience.

2 Organisation générale

2.1 Répartition du travail

Pour ce projet, nous avons décidé de répartir les rôles de cette manière :

Nom	**Mission**
Hugo	**- Affichage des documents :** - Création de l'interface pour afficher les documents ; - Affichage des informations principales (titre, auteur, etc.) ; **- Mise en place du projet et du dépôt GitHub** **- Rédaction du rapport**
Samantha	**- Emprunt et retour des documents :** - Création de fonctionnalités pour emprunter et rendre des documents - Mise en place d'un mécanisme pour suivre le statut d'un document
Soukaina	**- Gestion des utilisateurs :** - Implémentation du système d'authentification ; - Gestion de la connexion des utilisateurs.
Hudayfa	**- Rôle administrateur :** - Création de l'interface d'administration (ajout, suppression) ; - Création de l'interface de gestion des documents (ajouter, supprimer).

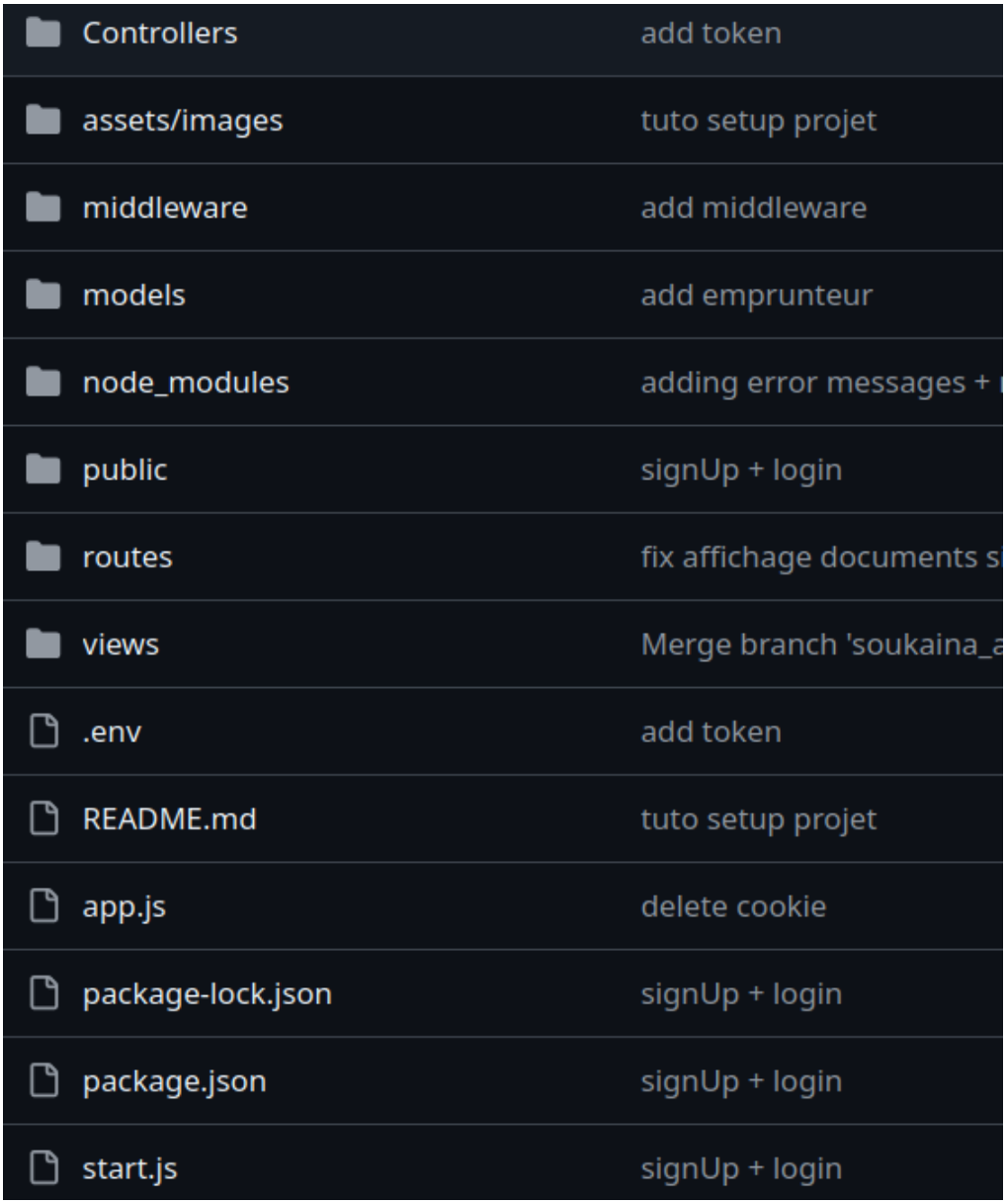
Répartition des rôles au sein du groupe

2.2 Méthode d'organisation

Pour le dépôt GitHub, nous avons utilisé la fonctionnalité des branches pour faciliter le travail en asynchrone. Nous nous sommes mis d'accord pour que ces branches doivent respecter la nomenclature suivante : `"nom_fonctionnalité"`. Par exemple, `"hudayfa_admin"` ou `"soukaina_auth"`.
Pour la communication interne, nous avons décidé d'utiliser l'outil Teams. Nous y avons intégré un cahier de bord, pour que chaque membre puisse y noter tous les changements apportés au projet.

2.3 Dépôt GitHub du projet

Lien vers le dépôt du projet



Dépôt GitHub du projet

Une rapide explication des principaux répertoires du projet et de leur rôles :

- `Controllers/AuthController.js` → On trouve dans ce fichier les fonctions qui gèrent l'authentification des utilisateurs (`signUp(), login(), logout()`).
- `models` → C'est dans ce répertoire que se trouvent les fichiers de schémas de données utilisés par MongoDB via l'ORM Mongoose.
- `routes/index.js` → C'est ici que les routes de l'application sont définies.
- `views` → Dans ce répertoire se trouvent les fichiers de vues, qui utilisent le moteur de templates `EJS` pour générer le contenu des pages dans le navigateur.

Il y a également les fichiers `app.js` et `start.js`. Le premier permet de configurer les différents éléments du projet : Express.js, les templates EJS, les routes, et les fichiers BootStrap. La ligne `module.exports = app;` permet d'utiliser ces éléments configurés dans tout le code de l'application.
Le deuxième est responsable de la connexion à la base de données et du démarrage du serveur Express.

3 Fonctionnalités implémentées

Dans cette partie, nous allons aborder les principales fonctionnalités implémentées, ainsi que certaines parties importantes du code.

3.1 Affichage et emprunt des documents

Ces fonctionnalités permettent d'afficher les documents et de gérer les emprunts et les retours de documents en utilisant Express, Mongodb et EJS. Plusieurs routes ont été mises en place :

- `GET /documents` pour afficher tous les documents disponibles
- `POST /emprunter/:id` pour emprunter un document
- `POST /rendre/:id` pour retourner un document.

Un middleware d'authentification a été intégré afin de restreindre certaines actions aux utilisateurs connectés. Dans les vues EJS, l'affichage a été adapté pour que tous les utilisateurs puissent voir les documents, mais que seul un utilisateur connecté puisse emprunter un document. Les documents empruntés s'affichent uniquement pour l'utilisateur concerné. Un champ `emprunteur` a été ajouté dans la base de données afin d'associer chaque emprunt à un utilisateur spécifique.

Page d'affichage des documents disponibles à l'emprunt

```
// route pour emprunter un document
router.post('/emprunter/:id', authenticate, async (req, res) => {
    const { id } = req.params;
    const userId = req.user.id;  // récupère l'ID de l'utilisateur connecté

    try {
        const document = await Document.findById(id);

        if (document.emprunteur) {
            return res.status(400).send('Ce document est déjà emprunté.');  // vérifie si le document est déjà emprunté
        }

        document.emprunteur = userId;
        await document.save();

        res.redirect('/documents');  // redirige vers la page des documents
    } catch (error) {
        console.error('Erreur lors de l\'emprunt du document:', error);
        res.status(500).send('Erreur lors de l\'emprunt du document');
    }
});
```

Exemple de route

3.2 Interface administration

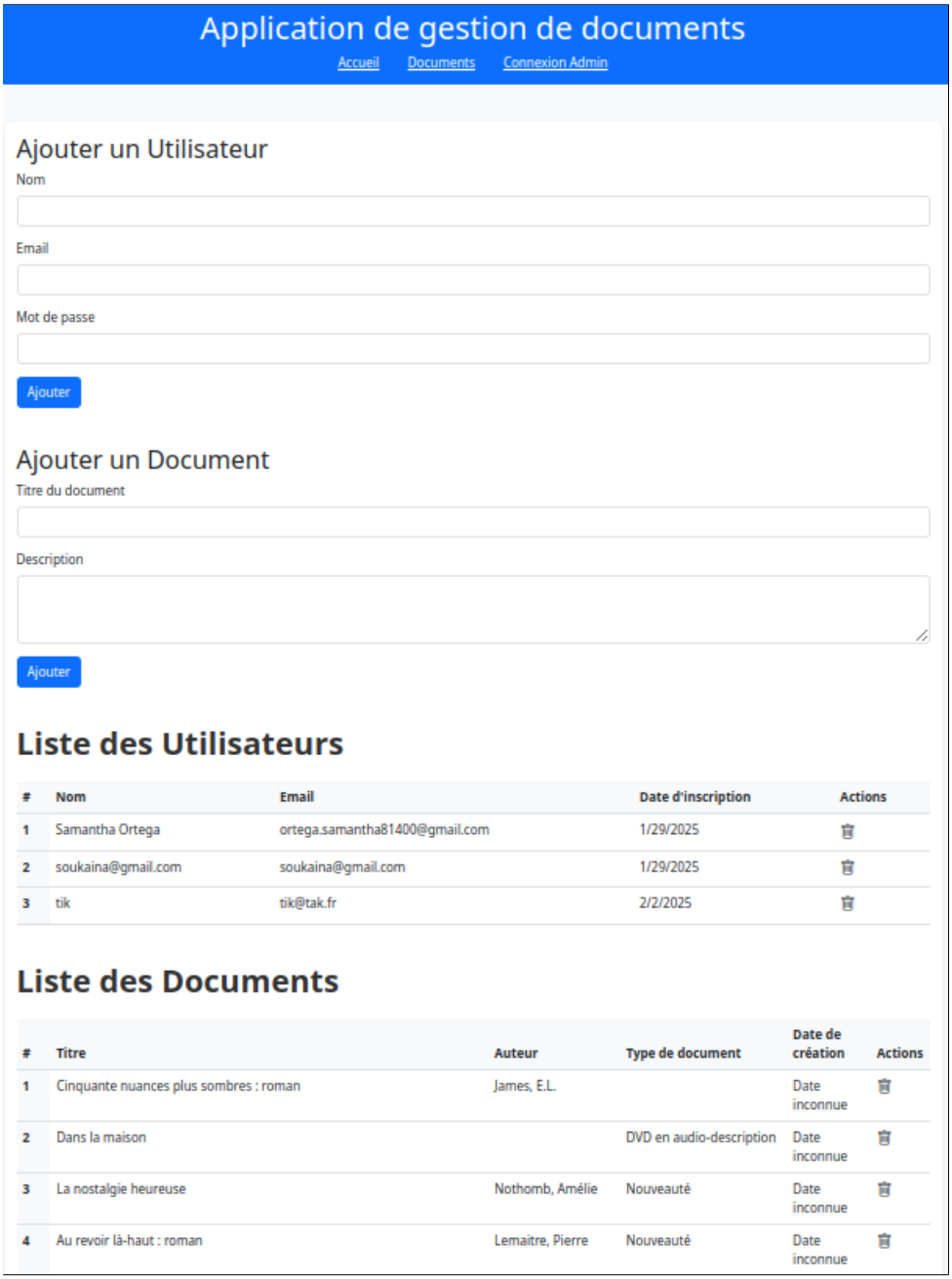
Le travail a consisté à implémenter un système d'authentification pour un administrateur et à ajouter des fonctionnalités de gestion des utilisateurs et des documents. Le modèle `Admin` a été créé avec un hachage du mot de passe, et le contrôleur `AdminAuthController` gère l'inscription, la connexion et la déconnexion.

Des routes ont été ajoutées au fichier `index.js` pour permettre à l'administrateur de s'inscrire, de se connecter et de se déconnecter, ainsi que pour ajouter et supprimer des utilisateurs et des documents, qui sont :

- `POST /admin/add-user` ;
- `POST /admin/add-document` ;
- `DELETE /admin/delete-user/:id` ;
- `DELETE /admin/delete-document/:id`.

Les vues ont été mises à jour pour refléter ces fonctionnalités : `admin/dashboard.ejs` permet la gestion des utilisateurs et des documents, tandis que `admin/login.ejs` et `admin/signup.ejs` offrent des interfaces pour la connexion et l'inscription des administrateurs. Le header a été modifié pour inclure un lien vers la page de connexion, facilitant l'accès. Ce système offre une gestion sécurisée et efficace du contenu pour l'administrateur.

Voici le rendu sur la page d'administration :



Page d'administration

4 Témoignages des membres du groupe

Hudayfa

«Le plus compliqué était l'ajout des utilisateurs et des documents, surtout pour bien gérer les erreurs et les redirections. La connexion admin aussi, avec la vérification des identifiants et des sessions, ça m'a pris du temps. Au total, j'ai travaillé sur le projet à peu près 6h.»

Hugo

«Globalement, je n'ai pas ressenti de difficultés particulières dans les missions. Le plus compliqué a été de comprendre le fonctionnement de Mongoose pour construire le fichier `Document.js`*. En tout, je pense que le projet m'a pris environ 5 heures.»*

Samantha

«J'ai eu du mal à mettre en place les fonctionnalités d'emprunt et de retour. J'avais réussi à faire mes fonctions sans la page de connexion, mais l'intégration du token m'a posé problème pour récupérer l'ID de l'utilisateur et déterminer ce qu'il pouvait emprunter ou rendre. Au total, j'ai passé environ 4 heures sur le projet.»

Soukaina

«J'ai passé environ 2h30 sur le projet. J'ai eu du mal au début avec la connexion à la base de données. Je pensais devoir la configurer avec la mienne parce que je n'avais pas accès à celle d'Hugo, mais j'ai finalement découvert que celle de Hugo fonctionnait sans problème.
Pour les fonctions d'inscription et de connexion, ainsi que l'ajout des routes et des templates, ça ne m'a pas pris de temps du tout, car je les avais déjà faits en .NET, donc c'est la même logique.»