A. Spécifications techniques

Full Stack JavaScript, stack MEVN :

Front :

- Nuxt
- VueJS
- Vuetify
- Moment

Back :

- NodeJS
- ExpressJS
- MongoDB
- Mongoose

Serveur :

- Front : Netlify
- Back : Heroku

Environnement de travail :

- Windows 10
- Google Chrome
- Visual Studio Code
- Git Bash
- GitHub

B. Réflexions initiales

Je connais déjà bien VueJS, c'est un framework avec lequel je travaille durant mon alternance. J'ai déjà utilisé Nuxt pour créer un site e-commerce pour mon dossier professionnel. Concernant le backend, j'ai déjà utilisé Express, Node et Mongo, toujours pour ce même site e-commerce.

J'ai décidé d'utiliser Vuetify pour m'aider à construire un design clair, simple et rapide. Je sais que le design n'est pas mon point fort, utiliser un framework comme Vuetify facilite grandement ce travail. De plus, cela permet également d'avoir un site qui s'adapte très facilement aux mobiles et aux desktops, il n'y a pas souvent besoin de rajouter à la main des breakpoints pour adapter son design.

Je commence d'abord par le frontend. Je décide de tout mettre en place avant d'ajouter les communications avec l'API. Je n'ai pas besoin de mettre en place un router, Nuxt s'en occupe pour moi : il suffit d'ajouter un fichier .vue dans le dossier "pages", et un router contenant le nom du fichier .vue est créé automatiquement.
Je crée également mon store, avec au début des données en dures dans l'application pour pouvoir travailler le design sur les pages. Une fois le frontend terminé, je passe au backend.

La partie backend est assez simple, je mets en place le serveur, et je commence par mes models pour les livres et les utilisateurs. Je mets ensuite en place les routes, puis les controllers. J'installe également multer pour pouvoir gérer facilement le traitement des images sur le serveur.

Une fois cela fait, je mets en place la logique de communication entre le frontend et le backend. Cela se passe majoritairement dans le store. Il y a un state, là où sont stockées les variables qui sont utilisées dans toute l'appli. Des getters, qui servent à avoir accès au state, sans pouvoir le modifier. Des mutations, qui servent à modifier le state. Des actions, qui servent à communiquer avec l'API.
Cette architecture permet une lecture simple et efficace du state. Les getters ne sont qu'en "read-only", les mutations en "write-only" et les actions appellent les mutations pour modifier le state.

Lorsque cela est mis en place, je rajoute de la sécurité avec l'implémentation de JWT dans le backend et le frontend.

C. Sécurité

Concernant la sécurité, c'est un point auquel j'ai particulièrement fait attention.

Dans le frontend, j'ai mis en place des middleware, qui permettent de sécuriser certaines routes (catalogue, emprunts, enregistrer un livre, valider les inscriptions) uniquement pour les utilisateurs connectés. Sur les pages disponibles uniquement pour les employés, un autre middleware est mis en place, qui permet de vérifier que l'utilisateur connecté est bien un employé de la médiathèque. De plus, un JWT est stocké dans le state pour vérifier l'authentification de l'utilisateur.
Dans le store, plus précisément les actions, au moment de passer des appels à l'API, il y a de nouveau une vérification qui est faite, pour vérifier que l'utilisateur est connecté pour certaines routes et pour vérifier que l'utilisateur est un employé pour d'autres.

Dans le backend, il y a également un middleware d'authentification pour chaque route sauf pour les routes "/login" et "/signin". Cela permet de vérifier le JWT de l'utilisateur, qu'il correspond bien à son user id. S'il ne correspond pas, l'utilisateur reçoit un erreur 403, lui interdisant l'accès à l'API.
Toujours dans le backend, au niveau des filtres pour le CORS, le header "Access-Control-Allow-Origin" est spécifié uniquement pour l'URL du frontend, pour éviter de pouvoir appeler l'API depuis n'importe quelle URL.

Bien sûr, lorsque l'utilisateur s'inscrit, son mot de passe est stocké en base de données seulement après avoir été crypté via le package "bcrypt". Aucun mot de passe n'est stocké "en clair".