Bramanouis *Projet WEB Avancé* *Mr. Cherel*

Date ;	lundi 11 novembre 2019
Auteur(s) ;	KANTAR OSMAN, CHASLE LOUIS, BOURHESDIS BRAHIM
Titre document ;	Projet WEB Avancé : Bramanouis
Version ;	1.1

I. L'URL

https://github.com/kalanyr95/bramanouis

II. Identifiants

Il est possible de créer des utilisateurs dans le site. Par défaut nous avons le compte (Nom utilisateur : test, Mot de passe : test) pourra accéder au site.

III. Description

Notre projet de base consistait à faire une application de gestionnaire de produits. C'est un site à potentiel de développement et comme base à des projets commerciaux.

Bramanouis est un gestionnaire de bibliothèque virtuelle. Il est possible d'ajouter, de supprimer des livres, de voir et de modifier les livres. Il y'a un système d'authentification qui permet aux utilisateurs d'interagir avec cette bibliothèque. Les livres présents dans cette bibliothèque sont conservés dans un fichier de base de données « lowdb » présent dans le projet. Ce fichier est un fichier JSON appelé « db.json ».

Dans un premier temps, nous avons voulu faire cette gestion avec le middleware passport, mais la difficulté et le manque de temps nous a orienté vers un projet de gestion de bibliothèque qui reste assez proche de ce que nous voulions de base et qui est toujours ouvert à un développement futur pour atteindre notre objectif premier.

Finalement les views de notre projet sont en. ejs pour la gestion des listes d'objets de notre fichier .json.
Notre projet en ejs est dans le master de notre github, il est possible de voir notre frontend (avec vue.js) que nous n'avons pas pu intégrer mais qui marche avec un faux backend dans le dossier 1 – Ancien.

IV. Conception, Réalisation, Déploiement

Projet de gestion de bibliothèque

Le répertoire se trouve dans le dossier 2 – Projet de notre git.

Pour lancer : npm install

npm run dev

Tous les éléments qui constituent notre projet se retrouvent dans cette partie fonctionnelle, avec la possibilité de s'authentifier et de s'inscrire grâce à la base de donnée lowdb en .json.

L'authentification se trouve en haut à droite, une fois connecté, le « login » prend le nom d'utilisateur. On peut cliquer dessus pour avoir le profil avec l'id et l'username.

Nous avons la fonctionnalité CRUD sur les livres, il suffit d'aller dans la page « Books » en haut à gauche :

- Ajouter un livre qui sera stocké également dans la bdd « db.json »
- Cliquer sur un livre existant pour :
    - Le supprimer
    - Mettre à jour son nom

Projet Frontend Vue-Vuex

Le répertoire se trouve dans le dossier 1 – Ancien de notre git.

Pour lancer : npm install

npm start

Authentification

auth-header.js est une fonction qui renvoie un header d'autorisation HTTP contenant le jeton Web JSON (JWT) de l'utilisateur actuellement connecté à partir du localStorage. Si l'utilisateur n'est pas connecté, un objet vide est renvoyé. Il est utilisé pour faire des requêtes HTTP authentifiées à l'API du serveur en utilisant l'authentification JWT.

fake-backend.js est utilisé pour exécuter notre connexion sans une API de serveur. Il contrôle la fonction fetch () pour intercepter certaines requêtes d'API et imiter le comportement d'une véritable API. Toutes les demandes qui ne sont pas interceptées sont transmises à la fonction fetch ().

Toute la gestion se fait grâce à une simulation de token JWT, les vérifications passeront par ce token dans notre partie authentification.

Ici nous avons 4 fonctionnalités principales :

- L'authentification
    - Prend les paramètres de la requête POST
    - Vérifie si les logins entrés en paramètre correspondent à un utilisateur
        - Si existent : on retourne les infos de l'utilisateur et un token
        - Sinon : erreur.
- Get users
    - Vérifie le token dans le header
        - Si valide : affiche la liste des utilisateurs dans l'arraylist d'utilisateurs
        - Sinon : erreur.
- L'inscription
    - On vérifie d'abord si le login d'utilisateur existe déjà dans la liste
    - Si le login est valide, avec la fonction push, on ajoute l'utilisateur à la liste.
- Supprimer un utilisateur existant

- Vérifie le token dans le header
    - Supprime sur choix de l'utilisateur, l'utilisateur choisi.

router.js contient la partie qui renvoi vers la page de login si l'utilisateur essaye d'accéder à une page restreinte.

user-service.js encapsule tous les appels api du backend pour effectuer des opérations CRUD sur les données user, ainsi que pour se connecter et se déconnecter de l'application. Les fonctions de service sont exportées via l'objet userService en haut du fichier et l'implémentation de chaque méthode est située dans les fonctions ci-dessous.

- Les fonctions login, logout utilisent localStorage afin de garder en mémoire les données de l'utilisateur et le token afin de le garder connecté entre les rafraichissements de la page.
- Dans la fonction handleResponse, le service vérifie si la réponse http de l'API est 401 non autorisée et déconnecte automatiquement l'utilisateur. Cela gère si le jeton JWT expire ou n'est plus valide pour une raison quelconque.

CRUD :

➔ C : Nous avons la création d'un utilisateur avec registration.
➔ R : Nous lisons la liste array d'utilisateurs.
➔ U : La liste se met à jour dans la page.
➔ D : Nous avons la possibilité de delete un utilisateur une fois connecté.

VUEX

Bibliothèque de gestion des states

Account.module.js Il contient des actions pour enregistrer un nouvel utilisateur, se connecter et se déconnecter de l'application

L'état de connexion initial de l'utilisateur est défini en vérifiant si l'utilisateur est enregistré dans la mémoire local storage, ce qui permet à l'utilisateur de rester connecté si le navigateur est actualisé et entre les sessions du navigateur.

Users.module.js Il contient des actions permettant d'extraire tous les utilisateurs de l'API et de supprimer un utilisateur.

V. Difficultés

La principale difficulté était de réussir à coder chacun de notre côté. Nous n'avions pas de vrai squelette de projet pour tous nous fier à la même chose, de ce fait, assembler le travail de chacun et l'adapter aux contraintes était notre obstacle principal.

Étant donné les dépendances à gérer ainsi le fait que cela soit nouveau pour nous, nous avons perdu beaucoup de temps dans l'adaptation du code de chacun.

Au final, nous n'avons pas réussi à rassembler les travaux réalisés par chacun de nous. Les travaux étaient bien trop complexes à rassembler.

Nous avons aussi eu beaucoup de mal et n’avons pas réussi à déployer le projet sur Glitch. Il s’agissait d’un nouvel outil pour nous qu’on ne connaissait pas auparavant. La compréhension de cet outil a été longue et n’a pas été suffisante pour mettre notre site web sur Glitch.