TP4 : Vue.js - Dr Mad et son laboratoire

Préambule

L'objectif principal de ce TP est d'implémenter une pseudo boutique en ligne, afin de mettre en œuvre de façon plus poussée les principes de vuex et de vue-router. Le scénario à implémenter est le suivant :

Quand l'utilisateur se trouve sur la page principale de la boutique il voit apparaître : - en haut, une barre de navigation avec un seul bouton : "Login" - au centre, un texte du genre "bienvenue à la boutique"

NB : pour simplifier l'exercice et notamment la gestion du panier, on part du principe que des boutons ne vont apparaître que si l'utilisateur est authentifié via la page de login.

Quand on clique sur "Login", le centre est remplacé par une page où il est possible de fournir un login/mot de passe pour la boutique. S'ils correspondent à un utilisateur valide, alors on considère que l'on récupère les informations liées à l'utilisateur, que l'on stocke dans le store, comme dans les TPs précédents. De plus : - le texte du bouton doit changer pour indiquer "Logout". - les boutons "Acheter", "Payer", et "Mes commandes" doivent apparaître.

Quand on clique sur "Acheter", le centre est remplacé par une page en 2 parties : - A gauche, une liste apparaît avec chaque nom de virus suivi d'un champ de saisie numérique permettant d'indiquer le nombre d'exemplaires voulus, et un bouton permettant de mettre le virus dans le panier. Il est également possible de sélectionner plusieurs virus et de tous les mettre dans le panier en une seule fois, grâce à un clic sur un bouton en bas de la liste. Dans les deux cas, les articles choisis et leur nombre sont stockés dans le store, dans les informations de l'utilisateur. - A droite, une liste apparaît à droite récapitulant les virus choisis se trouvant dans le panier, ainsi que le montant total (déductions de promotions comprises). En bas du panier, il est possible de cliquer sur un bouton "Acheter" pour créer une commande, dans l'état "waiting_payment". Une boite de dialogue apparaît faisant apparaître l'uuid de la commande.

Quand on clique sur "Payer", le centre est remplacé par une page où il est possible de saisir deux uuids : celui de la commande et celui d'une transaction bancaire. Quand on clique sur un bouton "Vérifier", on vérifie : - s'il existe une transaction bancaire avec cet uuid, - si le montant correspond au montant de la commande, - si le destinataire est bien le n° de compte de la boutique.

Si tout est ok, la commande passe en état "finalized".

Quand on clique sur "Mes commandes", le centre est remplacé par une page où il est possible de voir l'historique des commandes passées/en cours. Pour celles qui sont en état "waiting_payment", un bouton permet de l'annuler, et un autre permet de la payer. Un clic sur ce dernier redirige vers la page affichée lorsque l'on clique sur "Payer" dans la barre de navigation, mais avec le champ id de commande déjà rempli.

0°/ Mise en place

La structuration générale reste la même que pour les TPs précédents. De plus, certains composants vont être réutilisés. La mise en place initiale la plus simple consiste donc à : 1. Créer un nouveau projet avec vue-cli, 2. Remplacer le répertoire src de ce nouveau projet par celui du TP 3.

1°/ Modularisation du store

Comme l'application DrMad aura à terme des fonctionnalités bien séparées, il est important de modulariser le store. Pour cela : - Dans le répertoire store, créer deux fichiers : `shop.js` et `bank.js`, en reprenant l'exemple vu en TD sur les modules vuex. - Dans `shop.js` copier/coller depuis `index.js` tout ce qui concerne la gestion de la boutique, - Dans `bank.js` copier/coller depuis `index.js` tout ce qui concerne la gestion des comptes bancaires. - Modifier `index.js` pour importer `shop.js` et `bank.js` et créer le store à partir de ces 2 modules.

Pour les exercices suivants, il va falloir modifier le module `shop` afin d'implémenter les fonctionnalités demandées.

2°/ Le composant racine de la boutique

Dans le répertoire views, créer un composant `ShopView.vue`, avec un template du style :

```
<template>
  <div>
    <h1>Boutique</h1>
    <router-view name="shopmain"></router-view>
  </div>
</template>
```

Grâce à `<router-view>`, le composant va pouvoir afficher les différents composants pour gérer la boutique, grâce à une route à 2 niveaux. Il faut donc modifier `router/index.js` en conséquence. - La route `/shop` est la route racine. Elle doit afficher le composant `ShopView` (NB : grâce à `<router-view>` se trouvant dans `App.vue`) et elle a 5 enfants, qui vont tous s'afficher dans l'emplacement `shopmain`. - La route `/shop/home` doit permettre d'afficher `ShopHome.vue`. Elle a comme alias `/shop` (cf. https://router.vuejs.org/guide/essentials/redirect-and-alias.html pour voir des exemples d'alias). - La route `/shop/login` doit permettre d'afficher `ShopLogin.vue`. - La route `/shop/buy` doit permettre d'afficher `ShopBuy.vue`. - La route `/shop/pay/:orderId` doit permettre d'afficher `ShopPay.vue`. Le paramètre de cette route `orderId` doit être transformé en un props pour `ShopPay`. - La route `/shop/orders` doit permettre d'afficher `ShopOrders.vue`.

3°/ L'accueil

Dans le répertoire views, créer un composant `ShopHome.vue`. L'objectif de ce composant est juste d'afficher un texte de bienvenue à la boutique.

4°/ Authentification

Cette partie a déjà été implémentée dans les TPs précédents. Il suffit donc de reprendre le composant déjà écrit, et en faisant quelques modifications.

La première concerne le fichier `datasource/controller.js`. En effet, dans la solution proposée, la fonction qui recherche un utilisateur dans le fichier source de données `data.js` se contente de vérifier le login mais pas le mot de passe : il faut également faire cette vérification. Pour ce faire, vous devez : - installer le module `bcryptjs` : `npm install bcryptjs`, - importer ce module dans `controller.js`, - modifier la fonction de contrôle `shopLogin()` pour trouver un utilisateur dont le login ET le mot de passe correspondent, en appelant `bcrypt.compareSync(...)` pour vérifier si le mot de passe fourni correspond à celui de l'utilisateur (cf. doc bcryptjs pour son utilisation).

Remarques : - l'utilisateur "dr mad" (cf. `data.js`) a comme login/mdp : `drmad / drmad`. - il est possible que vous ayez un warning lors de la compilation de votre application. Normalement, cela n'a aucune influence sur le résultat. - La deuxième consiste à afficher un dialogue d'erreur si le login/mot de passe n'est pas valide.

Enfin, si l'authentification est correcte, on doit suivre la route `/shop/buy`.

5°/ Acheter des items

L'objectif est de permettre de sélectionner des virus et de les ajouter à un "panier d'achat", en plus ou moins grand nombre. (NB : cela impose de modifier les composants `CheckedList.vue` créés dans le TP précédent.)

5.1°/ Le composant racine des achats

Dans le répertoire views, créer un composant `ShopBuy.vue`. Ce composant se contente d'afficher à gauche un composant `ItemsList.vue` et à droite `BasketList.vue`.

5.2°/ Les items à acheter

Modifier le composant `CheckedList.vue` afin que : - il ait une props supplémentaire nommée `itemAmount` de type booléen, - un champ de saisie numérique soit ajouté devant chaque bouton se trouvant après un item (cf. `<input>` type number), si `itemAmount` vaut true, - quand on clique sur ce bouton, l'événement envoyé doit contenir l'indice de l'item ET la valeur du champ numérique si celui-ci est visible, - quand on clique sur le bouton se trouvant après la liste des items, l'événement envoyé doit contenir un tableau avec des couples indice/valeur champ numérique (si le champ est visible) pour chaque item sélectionné. Cela doit entraîner (dans le composant parent) la désélection des items.

Dans le répertoire components, créer un composant `ItemsList.vue`. Ce composant doit faire la même chose que `VirusesView.vue` des TPs précédents, mais en utilisant la version modifiée de `CheckedList.vue` pour : - que chaque item fasse apparaître le nom du virus, son prix, les promotions, le champ numérique de quantité et le bouton pour mettre au panier, - quand l'événement clic sur le bouton d'item est reçu, ajouter cet item ainsi que sa quantité dans le panier de l'utilisateur courant stocké dans le store (NB : dans `shopUser`). Attention, si cet item est déjà présent dans le panier, il faut juste augmenter sa quantité et non pas ajouter une nouvelle entrée dans le panier, - quand l'événement clic sur le bouton de fin de liste est reçu, ajouter tous les items et leur quantité dans

le panier de l'utilisateur courant stocké dans le store, - ne pas oublier de mettre à jour le total du panier dans `BasketList.vue`.

5.3°/ Le panier d'achat

Créer un composant `BasketList.vue`. Ce composant est responsable de l'affichage de la liste des items du panier avec leur prix et leur quantité. Il doit aussi afficher le total. Lorsqu'on clique sur le bouton "Acheter", une nouvelle commande doit être créée dans le store, avec comme état "waiting_payment", et un uuid doit être généré pour la commande (vous pouvez utiliser la méthode `uuidv4()` du module `uuid` installé dans le TP précédent).

Le panier doit être vidé de son contenu après avoir créé la commande.

6°/ Payer la commande

Dans le répertoire views, créer un composant `ShopPay.vue`. Ce composant est responsable de l'affichage de la page de paiement. Elle permet de saisir l'uuid de la commande ainsi que l'uuid de la transaction bancaire. Lorsque l'utilisateur clique sur "Vérifier", il doit être possible de vérifier si la transaction existe, et si son montant correspond à la commande. Si tout est correct, la commande doit passer en statut "finalized".

7°/ Voir les commandes passées

Dans le répertoire views, créer un composant `ShopOrders.vue`. Ce composant affiche une liste des commandes passées, avec leur statut. Quand une commande est en statut "waiting_payment", elle doit permettre d'être annulée ou payée. Si le bouton "Payer" est cliqué, il faut rediriger l'utilisateur vers la page de paiement, avec l'uuid de la commande déjà renseigné dans le formulaire.