JavaScript / Web Development - TP4

Introduction

Dans ce TP, nous allons continuer le développement de notre site Web.

Les TPs précédents ont permis de créer la partie Frontend et la partie Backend. Nous allons améliorer la partie Frontend en utilisant les concepts suivants :

- Découpage en composants : https://vuejs.org/guide/essentials/component-basics.html
- Vue Router : https://router.vuejs.org
- VueX : https://vuex.vuejs.org

Le symbole  indique que vous devez faire quelque chose.

Prérequis

Outils nécessaires :

- un navigateur récent (Firefox, Chromium)
- un éditeur de texte
- node (version 14.20.1)
- npm (version >= 6.14.17)

Démarrage

Vous trouverez 2 dossiers :

- phe-backend est une nouvelle version du serveur Backend intégrant la fonctionnalité de login
- phe-frontend est la base de travail de ce TP

Placez-vous dans le répertoire **phe-backend** et lancez la commande `npm install`. Cette commande va télécharger et installer les dépendances nécessaires pour notre TP.

Lancer ensuite le serveur backend :

```
npm start
```

Le serveur écoute par défaut sur le port 3000.

Ouvrez un autre terminal.

Placez-vous dans le répertoire **phe-frontend** et lancez la commande `npm install`. Cette commande va télécharger et installer les dépendances nécessaires pour notre TP.

Lancer le site web en mode développement

Lancer la commande suivante : `npm run dev`, le rendu sera disponible à l'adresse http://localhost:5173. La page se recharge automatiquement lorsque vous modifiez votre code.

Objectifs

Nous avons développé lors du TP 3 une salle de paris permettant à un utilisateur de voir les paris, d'en ajouter et de supprimer ses paris.

Nous allons maintenant développer de nouvelles fonctionnalités et de nouvelles pages.

Pour faire passer notre application de simple page à site web complet, nous allons implémenter :

- un routage avec une barre de navigation et la gestion de plusieurs pages
- le découpage de la page Race en plusieurs composants
- la gestion de la connexion de l'utilisateur

Le site Web a été volontairement découpé en plusieurs composants. Vous retrouverez le code écrit lors du TP 3 dans le composant Race.vue.

Vue-router

Lorsque vous arrivez sur le site, c'est le composant App.vue qui est chargé par défaut.

Nous voulons lier les routes suivantes aux composants correspondants :

- `/` => Home.vue (page d'accueil de notre site)
- `/race` => Race.vue (Page principale des paris)
- `/login` => Login.vue (Page de connexion des utilisateurs)

La définition du **router** se fait dans le fichier `src/router/index.js` et est importé dans `main.js`.

Dans ce fichier :

- importez les composants nécessaires
- déclarez les routes selon la liste ci-dessus

Dans le fichier App.vue :

- remplacez l'affichage du composant **Race** par la balise `<router-view></router-view>` (pensez à supprimer l'import du composant dans la partie script)
- à l'aide de la documentation d'Element Plus, ajoutez des liens dans la barre de navigation sur les items :
  - Pari Hippique Étudiant => `/`
  - Course => `/race`
  - Se connecter => `/login`
- testez la navigation en cliquant sur les différents liens de la barre de navigation

Components

Maintenant que nous avons accès à nos 3 pages, nous voudrions simplifier le code de `Race.vue` en créant un sous-composant `Pari.vue` qui s'occupera de l'affichage d'un pari.

Nous allons donc utiliser :

- le système de composant mono-fichier de VueJS
- le passage d'informations aux composants fils via les propriétés
- la remontée d'actions aux composants pères via les évènements

Création du nouveau composant Pari :

- créez un fichier `Pari.vue` dans le dossier `src/components`
- ajoutez les propriétés nécessaires (via la fonction `defineProps`)
- déportez le template
- transformez l'appel à la méthode `deletePari` par l'émission d'un évènement de type `deletePari` et en passant en paramètre l'id du pari

Dans le fichier **Race.vue** :

- ajoutez l'import du composant (en tant que fichier)
- utilisez le composant Pari.vue dans le template dans `section.paris`
- transmettez le pari via la directive `:prop="value"`
- ajoutez la directive `@deletePari` pour réagir à l'évènement émis par le composant fils **Pari.vue** (c'est toujours le composant père **Race.vue** qui gère les paris)

Une fois ce découpage terminé, testez que les fonctionnalités précédentes fonctionnent toujours (affichage des paris, ajout de pari, suppression de pari).

Vuex

Le serveur backend permet l'authentification des utilisateurs. Le composant **Login.vue** a été développé pour appeler le backend en transmettant les informations saisies par l'utilisateur.

Deux utilisateurs sont présents dans la base de donnée :

- alice / password
- bob / password

Nous voulons maintenant gérer l'état de connexion de l'utilisateur et stocker son nom.

Quelques prérequis

Dans le fichier **App.vue**, gérez l'affichage des boutons pour que :

- le bouton **Se connecter** apparaît uniquement si l'utilisateur n'est pas connecté
- les boutons **Course** et **Se déconnecter** apparaissent uniquement si l'utilisateur est connecté

Le fichier `src/store/index.js` représente notre **store** et c'est ici que nous allons ajouter les mutations, actions et getters.

- Modifiez la valeur par défaut de l'état **loggedIn** dans state à `false`

Ajoutez deux mutations :

- *login* qui permet de modifier le nom de l'utilisateur (`username`) et de positionner `loggedIn` à `true`
- *logout* qui permet de vider le nom de l'utilisateur (`username`) et de positionner `loggedIn` à `false`

Ajoutez deux actions :

- *login* qui va déclencher la mutation `login`
- *logout* qui va déclencher la mutation `logout`

Ajoutez deux getters :

- *username* qui retourne l'état `username`
- *loggedIn* qui retourne l'état `loggedIn`

Nous avons maintenant un store avec des accès en modification (via les actions) et en lecture (via les getters).

Dans le fichier **Login.vue**, le but est de réagir suite au retour du backend (en cas de succès). Ajoutez dans la méthode *login* :

- un appel à l'action *login* du store en transmettant le nom d'utilisateur
- une redirection vers la page de course (/race)

Pour tester, saisissez un nom d'utilisateur et un mot de passe valides et cliquez sur **Se connecter**.

Dans le fichier **App.vue**, implémentez la méthode *logout*. Elle doit :

- appeler l'action du store *logout*
- rediriger l'utilisateur vers la page d'accueil (grâce au routeur)

Lors du clic sur le bouton **Se déconnecter**, vous devriez être redirigé vers la page d'accueil.

Dans le fichier **Race.vue**, supprimez le champ de saisie de l'auteur du pari et modifiez le code du composant pour utiliser l'utilisateur sauvegardé dans le store.

Gérer le résultat de la course

Nous allons maintenant nous attaquer à la gestion du résultat de la course. Le premier objectif est d'indiquer quels sont les paris gagnants. Ensuite lorsque la course sera terminée et que les chevaux seront envoyés au box, nous supprimerons tous les paris existants.

Ajout du résultat de la course

La première étape consiste à récupérer le cheval qui a remporté la course et le transmettre au composant *Race.vue*. Pour cela, dans le fichier *HorseRace.vue* :

- créez une nouvelle variable permettant de stocker le cheval gagnant (ref)
- trouvez l'emplacement où la fin de la course est détectée et mettez à jour la variable créée précédemment
- lorsque l'évènement permettant d'informer que la course est terminée est émis, ajoutez le cheval gagnant

Dans la page *Race.vue*, puis la fonction `majEtatCourse` et si l'état est égal à *Terminée*, modifiez les paris pour ajouter un champ `gagne` qui sera `true` si le cheval est le gagnant de la course.

Enfin, modifiez le composant *Pari.vue* pour qu'il affiche une médaille à côté des paris gagnants. Vous pouvez utiliser l'icône *GoldMedal* définie par Element Plus.

Suppression des paris

Une nouvelle route a été ajoutée dans le Backend permettant de supprimer tous les paris. Regardez dans la définition des routes exposées afin de trouver l'URL et la méthode HTTP à utiliser.

Implémentez l'appel de cette route dans la fonction `supprimerParis` dans le fichier *Race.vue*.

Nous voulons appeler cette fonction lorsque les chevaux sont envoyés dans leur box et uniquement si la course est terminée. Pour cela, nous avons quelques modifications à effectuer dans le fichier *HorseRace.vue* :

- déclarez un nouvel évènement qui sera émis par le composant lorsqu'il faudra supprimer les paris
- trouvez la méthode qui est appelée lorsque l'on clique sur le bouton *Chevaux dans les box*
- ajoutez l'émission de l'évènement créé précédemment uniquement si la course est terminée

Il ne vous reste plus qu'à écouter ce nouvel évènement envoyé par le composant dans *Race.vue* et d'appeler la méthode permettant de supprimer tous les paris.

Résultat

Vous avez ajouté à notre page simple de course 3 concepts. L'utilisateur peut maintenant naviguer entre les différentes pages, se connecter et ajouter/supprimer des paris comme précédemment.

Conclusion

Dans ce TP, nous avons vu :

- comment utiliser vue-router pour gérer les accès aux différentes pages sans rechargement
- le découpage en composants pour structurer notre application
- la gestion des états avec VueX

J’ai fini plus tôt ou je veux en faire plus

À chaque fois que la page est rechargée par l’utilisateur, nous perdons la notion d’utilisateur connecté (le store est réinitialisé).

Le seul moyen de stocker cette information est d’utiliser les fonctionnalités de navigateur, comme les cookies ou le stockage local (localStorage).

À développer :

- Lorsque l’utilisateur se connecte avec succès, stockez dans le localStorage son nom d’utilisateur.
- Modifiez le store pour qu’il prenne en compte cette donnée si elle est présente quand l’utilisateur arrive sur notre site.
- N’oubliez pas de vider le localStorage lorsque l’utilisateur se déconnecte.
- La page `Race` ne devrait être accessible seulement quand l’utilisateur est connecté, à l’aide des guards redirigez vers la page de login si ce n’est pas le cas.
- Quant à la page `Login`, elle devrait être accessible seulement quand l’utilisateur n’est pas connecté.
- Redirigez vers la page d’accueil si l’utilisateur arrive sur une page inconnue.