# JavaScript / Web Development - TP5

## Introduction

Dans ce TP, nous allons ajouter des fonctionnalités à notre application de Pari Hippique Étudiant.

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 :

- Architecture de flux (temps réel) grâce à Socket.IO

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

Utilisez les deux dossiers fournis pour ce TP :

- phe-backend
- phe-frontend

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 run dev
```

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 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 vu dans les TPs précédents que notre application est composée de deux parties :

- un serveur backend responsable d'enregistrer les données (paris) et de les restituer. Pour cela, le serveur backend expose une API HTTP en mode REST.
- une application Frontend qui gère l'affichage et les actions de l'utilisateur.

À chaque interaction entre l'utilisateur et notre site web, la partie Frontend communique avec le backend :

- soit pour enregistrer des données saisies par l'utilisateur (envoi de pari)
- soit pour lire des données (affichage des paris des autres utilisateurs)

C'est toujours le Frontend (client) qui est à l'initiative de la communication vers le Backend (serveur).

Mais quand un utilisateur présent sur le site ajoute un pari, nous aimerions que tous les autres utilisateurs voient le nouveau pari instantanément.

Pour cela, nous allons utiliser la notion de Flux.

## Mise à jour des paris en temps réel

Socket.io est une librairie Javascript basée sur la technologie WebSocket. Les clients (Frontend) sont toujours connectés et en écoute avec le serveur.

Modifications à effectuer :

- configurer le backend pour mettre à disposition un flux
- configurer le frontend pour se connecter au flux
- lors de l'ajout/suppression d'un pari, émettre un évènement avec le bon type et les bonnes informations depuis le backend
- lors de la réception d'un de ces évènements, faire réagir le frontend en conséquence

### Configuration Backend

Dans le fichier **index.js** présent dans le backend, après la partie Démarrage du serveur, ajoutez les lignes suivantes :

```
import { Server } from 'socket.io';
const io = new Server(server, { cors: { origin: ['http://localhost:5173', 'http://127.0.0.1:5173'] }

io.on('connection', socket => console.log('Connection ' + socket.id));
```

La première ligne permet d'initialiser un flux et de le rattacher au serveur. La seconde ligne permet de réagir à l'évènement *connection* en affichant l'identifiant du client qui vient de se connecter.

### Configuration Frontend

Ajoutez dans le fichier **Race.vue** l'import de socket-io : `import io from 'socket.io-client';`

Ajoutez dans la déclaration de **data** la création du socket : `const socket = io('localhost:3000');`

### Emission d'évènements

https://socket.io/docs/#Using-with-Express

Dans le fichier **index.js** présent dans le backend, ajoutez pour les opérations suivantes l'émission d'un pari :

- route `/paris` avec le verbe POST (ajout de pari) => émettre un évènement de type **PARI_ADDED** contenant le pari
- route `/paris/generate` avec le verbe POST (ajout de plusieurs paris) => émettre un évènement de type **PARIS_ADDED**
- route `/paris` avec le verbe DELETE (suppression de pari) => émettre un évènement de type **PARI_DELETED** contenant l'identifiant du pari supprimé

### Réaction aux évènements

Ajoutez dans le fichier **Race.vue** le "hook" `onMounted` qui contiendra la réaction aux évènements émis par le backend :

```
socket.on('NOM_EVENEMENT', contenu => {
    // Actions
});
```

Ajoutez les actions nécessaires aux évènements émis par le backend :

- Ajout du pari sur l'évènement **PARI_ADDED**
- Ajout des paris sur l'évènement **PARIS_ADDED**
- Suppression du pari ayant le bon id sur l'évènement **PARI_DELETED**

Pensez à supprimer le rechargement des paris (appel à *chargerParis*) dans les méthodes *addPari*, *genererPari* et *supprimerPari*.

---

Pour tester, ouvrez deux onglets et connectez-vous avec deux utilisateurs différents. Lorsque vous envoyez un pari sur le premier onglet, la mise à jour du deuxième onglet doit être instantanée.

## Affichage des paris

Avant de passer à la suite, il serait plus user-friendly d'afficher un texte lorsque la liste des paris est vide.

## Séparation des responsabilités

En regardant la partie script du fichier *Race.vue*, on s'aperçoit que le code commence à s'agrandir et que la majorité concerne les appels au Backend. Il serait intéressant de déplacer ces fonctions dans une partie dédiée. Cela permettra de séparer les responsabilités et de rendre le code plus lisible. Également, si besoin, cela permettra de réutiliser ces fonctions dans d'autres composants.

Créez un fichier *paris.js* dans un nouveau dossier *api*. Copiez-y toutes les fonctions qui effectuent des appels au Backend présentes dans *Race.vue* et adaptez-les pour les rendre utilisables et accessibles. Vous pouvez créer une variable constante *BETS_URL* avec la valeur *http://localhost:3000/paris* que vous pourrez utiliser dans les différentes requêtes axios.

Vous pouvez maintenant importer les fonctions que vous avez créées, et les re-utiliser dans *Race.vue*. Vous pouvez également enlever la dépendance à *axios* dans ce fichier.

## Utilisation avancée de Vuex

Le but de cette partie est de déplacer la gestion du gain des paris dans Vuex. À la fin, notre composant *HorseRace.vue* n'émettra plus d'évènements, mais modifiera directement le state Vuex, et les composants qui réceptionnaient les évènements devront lire les données dans le store.

Commencez tout d'abord par modifier le state pour qu'il puisse contenir l'état courant des paris, ainsi que le cheval gagnant. Ensuite, rajoutez une mutation qui s'occupera de mettre à jour ces deux éléments. Puis créez trois actions différentes :

- une action qui sera appelée quand la course commence, afin de mettre à jour l'état des paris à *En cours*
- une action qui sera appelée quand la course termine, afin de mettre à jour l'état des paris à *Terminée* et de spécifier le cheval gagnant
- une action qui sera appelée quand les chevaux sont renvoyés aux box, afin de remettre l'état des paris à *En attente Paris*

Enfin, exposez à l'aide de deux getters l'état courant des paris et le cheval gagnant.

Modifiez maintenant le composant *HorseRace.vue* pour qu'il utilise les trois actions définies. Pour cela, repérez les trois appels à `emit('update:raceState')`, et remplacez-les par les actions correspondantes.

Nous allons maintenant utiliser nos deux getters. Le premier va servir dans *Race.vue* pour afficher l'état courant de la course. Le second va permettre de déterminer dans *Pari.vue* si le pari est gagnant ou non.

Le dernier détail consiste à supprimer tous les paris lorsque les chevaux sont renvoyés aux box et que la course est terminée. Rappelez-vous que les actions Vuex permettent de faire des appels asynchrones, et que vous pouvez donc appeler la fonction `deleteBets` de *paris.js* dans l'action correspondante.

Vous pouvez finir par retirer la définition et l'utilisation des deux évènements exposés par *HorseRace.vue*. Le fonctionnement devrait être le même que précédemment.

## Résultat

Dans ce TP, nous avons travaillé avec l'architecture temps réel avec Socket.io, nous avons déplacé tous les appels de notre API et avons été plus loin dans l'utilisation de Vuex.

## Conclusion

Ce TP clôt le Cours Développement d'Application Web / JavaScript.