# JavaScript / Web Development - TP2

## Introduction

Dans ce TP, nous allons développer un serveur Backend qui permettra d'enregistrer et de restituer des paris dans une base de données.

Nous allons utiliser un langage pour communiquer avec la base de données : SQL

La base de données sera constituée d'une seule table **paris** :

| id | auteur | cheval |
|---|---|---|
| 1 | Cyril | 1 |
| 2 | Georges | 4 |
| 3 | Elodie | 5 |

L'accès à la base de données se fera via une API REST.

Pour cela, nous allons utiliser différents outils/bibliothèques :

- NodeJS https://nodejs.org/en/docs/
- ExpressJS https://expressjs.com/fr/
- SQLiteDB http://www.sqlitetutorial.net/
- faker.js https://github.com/marak/Faker.js/
- Postman https://www.getpostman.com/apps

Le symbole  indique que vous devez faire quelque chose.

## Prérequis

Outils nécessaires :

- un navigateur récent (Firefox, Chromium)
- un éditeur de texte
- Postman
  - téléchargez la version Postman for Linux x64 : https://dl.pstmn.io/download/latest/linux64
  - décompresser le dossier Postman dans un dossier local (Documents par exemple)
  - lancer l'exécutable Postman en double-cliquant dessus
  - cliquer sur la mention **Take me straight to the app. I'll create an account another time.**
- node (version >= 14.20.1)
- npm (version >= 6.14.17)

## Démarrage

Dans le dossier `src`, vous trouverez :

- un fichier `package.json`
- un fichier `index.js` contenant une structure vide de code **JavaScript**
- un fichier `PHE-backend.postman_collection.json` qui contient des exemples pour Postman

Placez-vous dans le répertoire contenant ces fichiers et lancez la commande `npm install`. Cette commande va télécharger et installer les dépendances nécessaires pour notre TP. Si vous avez une erreur, essayez de libérer de l'espace dans votre espace personnel puis de relancer la commande.

## Lancer le serveur

Nous utilisons **nodemon**, un outil qui scrute les changements dans votre code et redémarre le serveur automatiquement pour les prendre en compte. Vous pouvez lancer le serveur avec la commande suivante :

```
node node_modules/nodemon/bin/nodemon.js index.js
```

Ou plus simplement avec :

```
npm run dev
```

## Détail des routes

L'objectif de ce TP est de développer 5 routes qui permettront par la suite de manipuler les paris depuis la partie Frontend :

- la route `/` avec le verbe HTTP `GET`. Cette route est juste disponible pour vérifier que le serveur répond. Vous pouvez renvoyer un simple message texte.
- la route `/paris` avec le verbe HTTP `GET`. Cette route permet de récupérer tous les paris présents dans la base de données avec le format suivant :

```
[
    {
        "id": 1,
        "auteur": "Océane",
        "cheval": 4,
        "date": "2018-11-19T15:15:03.977Z"
    },
    "..."
]
```

- la route `/paris` avec le verbe HTTP `POST`. Cette route permet d'ajouter un pari dans la base de données. Une requête HTTP `POST` utilise la notion de body pour transmettre des informations. Dans notre cas, seuls l'auteur et le cheval du pari seront transmis :

```
{
    "auteur": "Cyril",
    "cheval": 4
}
```

Le retour doit être de code `201` et peut contenir un message ("Pari créé" par exemple). - la route `/paris/generate` avec le verbe `POST`. Cette route va générer 10 paris dont le nom et le cheval seront aléatoires et les enregistrer en base de données. Le retour doit être de code `201` et peut contenir un message ("10 Paris générés aléatoirement" par exemple). Nous utiliserons la librairie Faker pour la génération. - la route `/paris/:id` avec le verbe `DELETE`. Cette route permet de supprimer un pari à partir de l'identifiant passé en paramètre. Le retour doit être de code `200` et peut contenir un message ("Pari supprimé" par exemple).

## Découvrir Express

Comme vu en cours, Express est une librairie permettant de développer rapidement une API HTTP.

Chaque route est définie de la manière suivante :

```
app.VERBE_HTTP(route, (req, res) => {
    // Il est possible de récupérer les paramètres dans req
    // Utiliser res pour créer le retour
});
```

Vous remarquez dans le fichier les lignes suivantes qui permettent d'importer la librairie, puis d'initialiser le serveur :

```
import express from 'express';

const app = express();
```

En utilisant la documentation d'ExpressJS, définissez les 5 routes décrites plus haut.

Pour l'instant, ces routes peuvent juste renvoyer un message différent ou logger des informations via `console.log()`.

Utilisez PostMan pour tester vos 5 routes.

## SQLite

Pour ne pas surcharger le TP, nous utilisons un type de base de données stockant les informations dans un fichier (ou en mémoire).

Vous remarquez dans le fichier les lignes suivantes qui permettent d'importer la librairie et d'initialiser une base dans le fichier database-phe.db :

```
import sqlite3 from 'sqlite3';
import { open } from 'sqlite';

db = await open({ filename: 'database-phe.db', driver: sqlite3.Database });
```

### Création de la table

En utilisant la documentation de SQLite, initialisez une table `paris` structurée avec 3 champs :

- auteur de type TEXT
- cheval de type INTEGER
- date de type TEXT

### Ajout d'un pari

Il faut maintenant pouvoir ajouter un pari depuis la route /paris avec le verbe POST. Pour lire les informations transmises dans le body, utilisez `req.body.propriété`

Ajoutez un pari dans la base de données à chaque appel sur cette route. Pour le champ **date**, utilisez la date actuelle (`new Date()`) ou le mot clé DATETIME de SQLite

Utilisez la requête **Ajouter Pari** depuis PostMan pour tester votre route et ajouter quelques paris.

### Récupération des paris

Maintenant que nous avons des paris dans la base de données, vous pouvez développer la récupération de tous les paris triés par date descendante.

Pour trier les paris, vous pouvez utiliser le mot clé `ORDER BY` de SQLite ou trier en Javascript les paris avant de les renvoyer.

Utilisez la requête **Récupérer Paris - 1 Pari** depuis PostMan pour tester votre route.

### Génération de 10 paris

Pourquoi ne pas injecter un ensemble de 10 paris pour tester notre application.

Utilisez le même principe que pour ajouter un pari, mais au lieu de récupérer les informations depuis le body de la requête, utilisez la librairie **faker** pour générer des informations.

```
faker.name.firstName();
faker.random.number({ min: 1, max: 10 });
```

Utilisez la requête **Générer 10 paris** depuis PostMan pour tester votre route et ajouter quelques paris.

**Suppression d’un pari**

Développez maintenant la route permettant de supprimer un pari (DELETE sur `/paris/:id`).

Utilisez la requête **Supprimer Pari** depuis PostMan pour tester votre route et supprimer un pari.

## Résultat

Vous avez développé en moins de 4 heures un serveur complet et fonctionnel permettant de stocker, gérer et récupérer des paris dans une base de données.

Dans le prochain TP, une partie Frontend viendra compléter notre application et utilisera le serveur. L’interaction entre le Frontend et le Backend se fait via une API REST HTTP.

## Conclusion

Dans ce TP, nous avons vu :

- comment ajouter une librairie, lire la documentation et se l’approprier pour développer rapidement
- comment interagir avec une base de données

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

Il pourrait être intéressant de gérer les cas d’erreurs :

- demande de suppression d’un pari qui n’existe pas
- demande d’ajout de pari sans auteur ou sans cheval

Écrire des requêtes SQL est puissant, mais peut demander beaucoup d’efforts pour écrire des requêtes classiques (ajout, suppression). Étudiez la notion d’ORM et remplacez les requêtes SQL par Sequelize (http://docs.sequelizejs.com/).