# R4.A.10 Complément Web Partie 1

## Avant-propos

Ce projet va vous permettre de mettre en œuvre, en 3 étapes, les concepts vus en TD et dans les premières séances de TP.

L'objectif de ce projet est de construire une WebApp qui affichera des informations sur les *Pays du Monde*, et permettra de rechercher, trier, filtrer les pays et leurs informations en fonction de divers critères qui vous seront détaillés ultérieurement.

Une *WebApp* est une page Web qui se comporte comme une application traditionnelle, à savoir un rafraîchissement dynamique du contenu affiché sans faire appel à des chargements de nouvelles pages HTML. Une telle application porte aussi le nom d'*Application Web Monopage* (Single Page App). Vous aurez à manipuler le DOM.

Les données qui serviront à remplir votre application proviendront d'une API WebService.

## Introduction

Dans cette partie, vous allez vous focaliser sur la lecture des données directement depuis un fichier contenant une structure JSON. Nous verrons dans une future partie comment récupérer ces mêmes informations depuis un WebService.

## Le fichier de données

Récupérez le fichier **`countries.json`** qui accompagne le sujet de ce projet.

Ce fichier contient 248 pays et beaucoup de données pour un seul pays. Analysez la structure JSON d'un pays (pour vous aider à localiser les données : le 1er est Afghanistan, le 2ème est Albanie...).

Nous n'allons pas utiliser toutes les informations issues de ce fichier. Voici celles qui nous intéressent et que vous devez prendre en compte (à vous d'identifier les noms des champs) :

- Code Alpha 3
- Superficie
- Pays frontaliers, attention il y a des pièges (cherchez Tuvalu)

- Capitale
- Continent
- Gentilé
- Drapeau (celui au format SVG)
- Nom (en français, anglais, allemand, espagnol et italien), attention il y a un piège !
- Population
- Top Level Domains (Internet)

Nous nous intéresserons aussi aux informations suivantes dans un second temps :

- Monnaies
- Langues

# La classe *Country*

Q1 - A partir de votre analyse de la structure JSON d'un pays, créez une classe **`Country`** pour y stocker vos données de travail. Prévoyez une fonction **`toString()`** synthétique (pas besoin de retourner toutes les informations).

**Note** : pour le moment, ne vous occupez ni des monnaies ni des langues.

Q2 - Ecrivez une fonction **`fill_db()`** qui lit la source de données et crée des objets **`Country`** que vous stockez dans un tableau (de classe) associatif **`all_countries`** dont les clés sont les valeurs trouvées dans le champ **`apha3Code`**.

Q3 - Complétez votre classe **`Country`** par les méthodes suivantes :

- **`getPopDensity()`** : retourne la densité de population du pays (hab. / $Km^2$)
- **`getBorders()`** : retourne une liste des pays frontaliers (les objets, pas les codes).

# La classe *Currency*

Vous allez maintenant prendre en compte les monnaies.

Q1 - Créez une classe **`Currency`** et un tableau (de classe) associatif **`all_currencies`** dont les clés sont les valeurs trouvées dans le champ **`code`** des monnaies des pays.

On s'intéresse au code, au nom (en anglais) et au symbole (UTF-8) des monnaies.

N'oubliez pas la méthode **`toString()`**.

Q2 - Dans la fonction **`fill_db()`**, vous allez devoir créer les objets **`Currency`** au fur et à mesure que vous créez vos objets **`Country`**, quand la monnaie n'existe pas déjà dans **`all_currencies`**. Dans la classe **`Country`**, vous devez stocker uniquement le code des monnaies, ce qui vous permettra de trouver les informations dans le tableau **`all_currencies`**, par la suite.

Q3 - Complétez votre classe **`Country`** par la méthode suivante :

- **`getCurrencies()`** : retourne une liste des monnaies (objets **`Currency`**)

# La classe *Language*

Vous allez maintenant prendre en compte les langues, de façon similaire aux monnaies.

On s'intéresse au code **`iso639_2`** et au nom (en anglais).

Q1 - Créez cette classe **`Language`**.

Le tableau (de classe) associatif des langues s'appelle **`all_languages`**.

Q2 - Complétez votre classe **`Country`** par la méthode suivante :

- **`getLanguages()`** : une liste des langues (objets **`Language`**)

# Tests

Écrivez des fonctions permettant de répondre aux questions suivantes. Vous testerez (afficherez) dans la console d'un navigateur ou dans la page HTML, à votre convenance.

Vous regrouperez, dans le fichier **`test.js`**, les fonctions suivantes.
Vous respecterez les noms des fonctions.
Vous serez aussi évalués sur le respect de ces consignes !

Q1 - **`outsideTheContinent()`** : Pays dont au moins un pays frontalier n'est pas dans le même continent.

Q2 - **`moreNeighbors()`** : Pays (possibilité de plusieurs) ayant le plus grand nombre de voisins. Affichez aussi les voisins.

Q3 - **`neighborless()`** : Pays n'ayant aucun voisin.

Q4 - **`moreLanguages()`** : Pays (possibilité de plusieurs) parlant le plus de langues. Affichez aussi les langues.

Q5 - **`withCommonLanguage()`** : Pays ayant au moins un voisin parlant l'une de ses langues. Affichez aussi les pays voisins et les langues en question.

Q6 - **`withoutCommonCurrency()`** : Pays sans aucun voisin ayant au moins une de ses monnaies.

Q7 - **`sortingDecreasingDensity()`** : Pays triés par ordre décroissant de densité de population.

Q8 - **`moreTopLevelDomains()`** : Pays ayant plusieurs Top Level Domains Internet.

Q9 - **`veryLongTrip(nom_pays)`** : En partant d'un pays donné (représenté ici par l'argument **`nom_pays`**), listez tous les pays que l'on peut visiter en passant de l'un à l'autre. Evidemment, seuls les pays frontaliers sont accessibles depuis un pays donné.
Exemple : France -> Espagne -> Portugal.

Q9 - Compléments :

- En guise de test vous ajouterez un appel à la fonction **`veryLongTrip(nom_pays)`** en utilisant **`France`** comme argument.
- Essayez de trouver un exemple de pays depuis lequel on peut ainsi visiter le plus de pays. Vous ajouterez aussi un appel à la fonction **`veryLongTrip(nom_pays)`** en utilisant ce pays comme argument.