582-31F-MA – Programmation d'interface Web 2 Jonathan Martel
Exercice Pondération : 5%

Création d'un module pour consulter la liste des maires de Montréal

Objectif :
Créer une application qui utilise les requêtes asynchrones et un modèle de programmation orientée objet avec des classes.

Contexte :
Changement de plan pour votre application sur les maires de Montréal! Celle-ci doit gérer les requêtes asynchrones vers un service Web qui retournera les données déjà traitées. Votre module doit toujours permettre aux utilisateurs de l'application d'afficher la liste complète des maires (et mairesse) de Montréal triée par nom ou par date (année) et en ordre ascendant ou descendant. Il devra aussi de permettre d'effectuer des recherches par date (année) et par nom/prénom. Cependant, les données seront retournées en format JSON par un service Web. Comme votre code est déjà greffé à l'application, votre classe doit retourner les mêmes données, mais à partir de requêtes asynchrones (xhr et fetch)

Comme votre travail doit se greffer à celui de vos collègues, vous devez suivre des spécifications techniques. De plus, vous devez recréer une mini-application qui permet de montrer vos résultats à vos collègues. Le tout doit être réaliser en suivant un patron de conception orientée objet (tel que vue en classe).

Spécifications :
Vous devez adapter votre classe Maire pour qu'elle fasse les requêtes correctement.

La classe Application gère l'interface et les appels sur la classe Maire. Cette dernière ne traite que les données. Pour l'instant, comme l'application backend n'est pas complète, vous devez travailler avec des données statiques en format JSON (fourni).

La classe Maire doit être structurée tel que spécifié :

Classe Maire (Module)

Méthodes publiques :
Maire.rechercheMaires({type:[date, nom], valeur:"recherche", callback:function(data){}})
En fonction du type de recherche et de la valeur cherchée, la méthode utilisera la fonction de callback pour retourner un tableau qui contient les résultats de recherche. Chaque maire doit être représenté par un objet littéral. Le callback doit être associé à la fonction de traitement des données lorsque celles-ci seront reçues.

Maires.liste({type:["date", "nom"], ordre:["ASC", "DESC"], callback:function(data){}})
En fonction du type de recherche et de l'ordre du tri, , la méthode utilisera la fonction de callback pour retourner un tableau qui contient l'ensemble des maires triés. Chaque maire doit être représenté par un objet littéral. Le callback doit être associé à la fonction de traitement des données lorsque celles-ci seront reçues.

Maire.getNombreMaires()
Retourne le nombre de maires. Cette méthode doit utiliser les données de la classe pour retourner le nombre de maire immédiatement.

Notez que vous pouvez ajouter des méthodes privées selon vos besoins.

La classe Application pourra être fait selon votre choix, mais notez que le code doit être bien structuré, facile à lire et commenté. La classe Application est instanciée par le fichier principal (main.js)

Autres précisions :

- Votre objet doit être dans le fichier nommé NomClasse.js (ou .mjs)
- Code commenté et en français
- Vous devez remettre deux versions de la classe Maire, une avec fetch, l'autre avec xhr.

Travail individuel.
Remise : Voir Léa

Détails de l'API
url : https://jmartel.webdev.cmaisonneuve.qc.ca/wsmaires/maires/

Faire une recherche : /recherche
?type = [date, nom]
?valeur = chaine de recherche
Ex : /maires/recherche?type=date&valeur=2012

Obtenir la liste : /liste
?tri=[date, nom]
?ordre=[asc, desc]
Ex : /maires/liste?tri=date&ordre=asc

Obtenir le nombre de maires : /nombre
Ex : /maires/nombre

Le serveur retourne les données en JSON.