Le projet - WeatherApp
Day 2 - Rendre l'application dynamique
Instructions

Objectif de la journée

Rendre l'application dynamique, c'est à dire que le back end aura la responsabilité de gérer les villes saisies. Pour cela il modifier le backend de l'application, pour mettre en place deux nouvelles routes qui permettront de prendre en compte les requêtes suivantes:

- Ajout d'une ville
- Suppression d'une ville

Il faudra ensuite modifier la partie front end pour que celle ci puisse exploiter les deux routes.

1- Préparer l'environnement de travail

- Création du répertoire **part2** dans le répertoire **./lacapsule/project/weatherapp**
- Reprendre le livecode **part1** et le déposer dans le répertoire **part2**
- Se positionner sur le répertoire **part2** via le terminal

```
cd ./lacapsule/project/weatherapp/part2
```

2- Stocker les informations des villes

- Vous devez stocker plusieurs villes, il faut donc une structure vous permettant de lister l'ensemble des villes.
  Note:
  Chaque ville contient plusieurs informations qui lui sont propres: nom, une image (en réalité une url), un descriptif du temps qui fait, une température min et une température max. Il faudra également trouver une structure qui vous permettra de représenter simplement l'ensembles des informations d'une ville

- Ecrivez cette structure que l'on nommera **cityList** au dessus de la route `/` et remplissez la avec les informations utilisées dans le fichier index.ejs

- Transmettez les informations de **cityList** au fichier **index.ejs** via `res.render()`

3- Dynamiser la vue

- Dans le fichier **index.ejs**, exploitez les données transmises par le back end, en les parcourant à l'aide d'une boucle.
  Note:
  Le bloc de code répété par la boucle sera une portion de code html représentant une ville.
  Attention dans un fichier ejs, tout bloc javascript devra être délimité par les caractères `<%` pour déclarer le début et `%>` pour déclarer la fin.

- Supprimez les autres parties HTML faisant références aux autres villes.

- A ce stade, notre boucle va répéter autant de fois qu'il y a de ville dans notre tableau la même portion de code HTML. Vous allez donc avoir plusieurs fois la même ville.
  L'objectif est d'exploiter les données des villes présentes dans chaque élément du tableau, et de les afficher aux endroits prévus dans cet effet.
  Note:
  Attention pour afficher le contenu d'une variable JS il faudra l'entourer des caractères `<%=` et `%>`

4- Rappel sur les échanges client/serveur

Deux solutions possibles pour communiquer avec le serveur:

- Le lien, il s'utilise lorsque l'on veut envoyer simplement une demande au backend
- Le formulaire, il est nécessaire lorsqu'on souhaite envoyer des informations que l'utilisateur a saisie

Deux types d'envoi possible:

- Envoi **GET** permet d'envoyer des informations complémentaires à la question envoyée au serveur. Le problème est que ce mode d'envoi est limité en nombre d'information. On pourrait assimiler ce mode de communication par un envoi par lettre entre les deux parties.

- L'envoi **POST** permet à travers cette limitation d'envoyer des informations volumineuses comme par exemple des photos. On pourrait assimiler ce mode de communication par un envoi par colis entre les deux parties.

Contraintes à retenir:

- ★ Un lien est forcement en **GET**.
- ★ Un formulaire est par défaut en **GET** mais peut se paramétrer en **POST** grâce à l'attribut method à placer dans la balise `<form>`

5- Ajouter une ville

Côté back end

- La structure qui représente les villes est la variable globale **cityList**. L'objectif est de mettre à jour cette variable en y ajoutant une nouvelle ville.

- Pour gérer cette demande, il faut une nouvelle route que l'on nommera `/add-city` qui se chargera d'ajouter une nouvelle ville et de renvoyer au client le contenu de la page **index.ejs**

- Dans la fonction de callback de la route de `/add-city`, utilisez une instruction permettant d'ajouter dans la structure **cityList** les informations d'une ville (name, desc, img, temp_min et temp_max).

- Pour le moment, validez cette étape en y mettant des valeurs fictives comme par exemple pour name mettez , desc : , img: **/picto.png**, temp_min: et temp_max:

Côté front end

- Utilisez les bonnes balises HTML permettant d'envoyer la demande `/add-city` au backend ainsi que le nom de la ville saisi par l'utilisateur.

- Testez la mécanique, a ce stade, chaque clic sur le bouton "Enregistrer" enverra une demande au backend. Celui-ci ajoutera une nouvelle ville avec les informations saisi en "dur" côté backend.

On revient côté back end

- Revenez sur la fonction de callback de la route `/add-city`, et récupérez le nom de la ville envoyée par le front end.
  Note:
  Vérifiez cette étape en réalisant un `console.log()` de cette valeur puis relancez le serveur pour tester le scénario.

- Utilisez cette valeur pour remplacez la valeur fictive utilisée pour remplir la partie name

5- Supprimer une ville

La structure qui représente la liste des villes est la variable globale **cityList**.
L'objectif est de mettre à jour cette variable en y supprimant la ville sélectionnée lors du clic sur l'icône de la "croix".

Côté back end

- Pour gérer cette demande, il faut une nouvelle route que l'on nommera `/delete-city` qui se charge de supprimer de **cityList** la ville sélectionnée et de renvoyer au client le contenu de la page **index.ejs**

- Dans la fonction de callback de la route de `/delete-city`, utilisez une instruction permettant de supprimer un élément dans la structure **cityList**.

- Pour le moment, validez cette étape en y mettant des valeurs fictives comme par exemple supprimer toujours le premier élément de a structure **cityList**.

Côté front end

- Utilisez la bonne balise HTML permettant d'envoyer la demande `/delete-city` au backend

- Testez la mécanique, a ce stade chaque clic sur l'icône enverra une demande au backend. Celui-ci supprimera toujours le 1er élément de la structure **cityList**.

- Rendez cette étape dynamique en ajoutant à la requête l'information permettant de cibler l'entité sélectionnée.

On revient côté back end

- Revenez sur la fonction de callback de la route `/delete-city`, et récupérez la valeur envoyée par le front end.
  Note:
  Vérifiez cette étape en réalisant un console.log de cette valeur puis relancez le serveur pour tester le scénario.

- Remplacez la position fictive mise dans l'instruction pour supprimer un élément de **cityList** par la valeur envoyée par le frontend