Semaine 13 - Stockage de données locales avec HTML5

Introduction

Avant HTML5 et l'**API Web Storage**, les données d'application devaient être stockées dans les **cookies avec des limites**.... Contrairement aux cookies dont la taille maximale est limitée à **4ko**, la limite de stockage avec **Web Storage** est bien plus grande (au moins **5 Mo** pour la majorité des navigateurs) et l'information n'est jamais transmise à un serveur web distant.

Le stockage local est par **domaine**. Toutes les pages, d'un même domaine, peuvent stocker et accéder aux mêmes données. Les données stockées ne sont toutefois pas accessibles d'un navigateur à l'autre.

Les données ne sont pas cryptées, il ne faut donc pas y placer d'informations sensibles…

L'API Web Storage fournit deux objets pour stocker des données sur le poste du client dont la seule différence concerne la persistance des données:

- **window.localStorage** qui permet de stocker des données « permanente » sans date d'expiration;
- **window.sessionStorage** qui stocke les données pour une session uniquement (les données sont perdues lorsque l'onglet du navigateur est fermé).

Il est à noter que cette note de cours ne concerne que le stockage local (localStorage). Mais, les méthodes et propriétés sont communes aux deux objets, ce n'est que la durée de vie qui change…

LocalStorage

L'objet **localStorage** stocke les données **sans date d'expiration**. Ces données sont sauvegardées sous forme de **chaînes de caractères** (des paires clé/valeur) et, elles ne seront pas supprimées lorsque le navigateur sera fermé *(si on les a bien sûr sauvegardées avant...)*. Elles seront disponibles si la page Web est de nouveau ouverte, soit la journée-même, la semaine suivante ou même l'année suivante...

Les opérations courantes sont le stockage, la récupération et la suppression des données ou l'obtention d'informations (nombre de paires, récupération des clés..). Les méthodes de l'objet **localStorage** sont les suivantes :

- **setItem**(clé,valeur) : stocke une paire clé/valeur;
- **getItem**(clé) : retourne la valeur associée à une clé;
- **removeItem**(clé) : supprime la paire clé/valeur en indiquant le nom de la clé;
- **key**(index): retourne la clé stockée à l'index spécifié;
- **clear**(): efface toutes les paires de clé/valeur.

Par ailleurs, la propriété **length** renvoie le nombre de paires stockées.

Stockage des données

La méthode setItem() permet de stocker une donnée. La syntaxe est la suivante:

- **localStorage.setItem("clé", "valeur");**

Le **premier paramètre** de setItem() est la **clé** (toujours de type String). Elle précise l'endroit où sont stockées les données afin de pouvoir les y retrouver ultérieurement. Le deuxième est la **valeur attribuée** (aussi de type String). Par exemple:

```
//Stockage  setItem("nomPropriété", "valeur")
localStorage.setItem("couleur", "vert");
```

Il est à noter que puisque **localStorage** est un **objet**, on pourrait également utiliser la **notation à point** ou les **crochets** pour stocker une nouvelle donnée. Exemple:

```
localStorage.couleur= "vert";
//ou
localStorage["couleur"] = "vert";
```

À noter que bien que ces deux dernières façons sont fonctionnelles, il est fortement recommandé de ne pas les utiliser et de favoriser l'utilisation des méthodes de l'API.

Récupération des données

La méthode getItem() permet de récupérer la valeur d'une donnée stockée. La syntaxe est la suivante:

- **localStorage.getItem("clé");**

Grâce à la **clé** initialement créée avec **setItem()** il est possible de récupérer facilement les données. Ces dernières sont **toujours** retournées sous la forme d'une **chaîne de caractères**. Exemple :

```
console.log(localStorage.getItem("couleur"));//Retourne : "vert"
```

Suppression d'une donnée

La méthode removeItem() permet de supprimer une donnée. La syntaxe est la suivante:

- **localStorage.removeItem("clé");**

La méthode **removeItem()** permet de supprimer une clé créée précédemment. Par exemple, dans le code qui suit on supprime la clé "couleur" créée ci-haut. Si on essaie d'accéder de nouveau à cette clé, la valeur retournée sera **null** :

```
localStorage.removeItem("couleur");
console.log(localStorage.getItem("couleur"));//Retourne : null
```

Suppression de toutes les données sauvegardées

La méthode **localStorage.clear()** supprime **TOUTES** les données stockées dans la navigateur du poste client

Conversion des données

REMARQUE IMPORTANTE: la paire clé / valeur est toujours stockée sous forme de chaînes. Il ne faut donc pas oublier de les convertir dans un autre format en cas de besoin!

L'exemple simple qui suit met en place un compteur de visites. Dans ce code, si la valeur retournée par **localStorage.getItem("visites")** est **null**, c'est la première visite. Autrement, la chaîne de valeur reçue est convertie en un nombre afin d'être en mesure d'augmenter le compteur:

```
//Vérifier si l'utilisateur a déjà visité le site
let nbVisites = localStorage.getItem("visites");
if (nbVisites == null) { //Aucune donnée stockée avec cette clé
    //C'est la première visite
    nbVisites = 1;
} else {
    //La chaîne est convertie en nombre avant son incrémentation
    nbVisites = Number(nbVisites) + 1;
}
//On stocke localement le nombre de visites
localStorage.setItem("visites", nbVisites);
console.log("Nombre de visites:", nbVisites);
```

Mémoriser des objets de données

L'API Web Storage est bien pratique pour **stocker de simples chaînes de texte**. Lorsqu'il s'agit de manipuler des données plus **complexes**, entre autres des **objets JavaScript**, il faut les convertir au préalable en chaînes de texte puisque **localStorage** n'accepte que ce type de données.

Le format **JSON** (JavaScript Object Notation) est la solution de prédilection pour cela. **JSON** est un format de données **textuelles** dérivé de la notation des objets du langage JavaScript. Deux méthodes sont disponibles : **JSON.stringify()** qui prend en paramètre un objet (ou un tableau) et renvoie une chaîne de caractères *linéarisée*, et son inverse **JSON.parse()** qui reforme un objet à partir de la chaîne de caractères. Exemple d'utilisation :

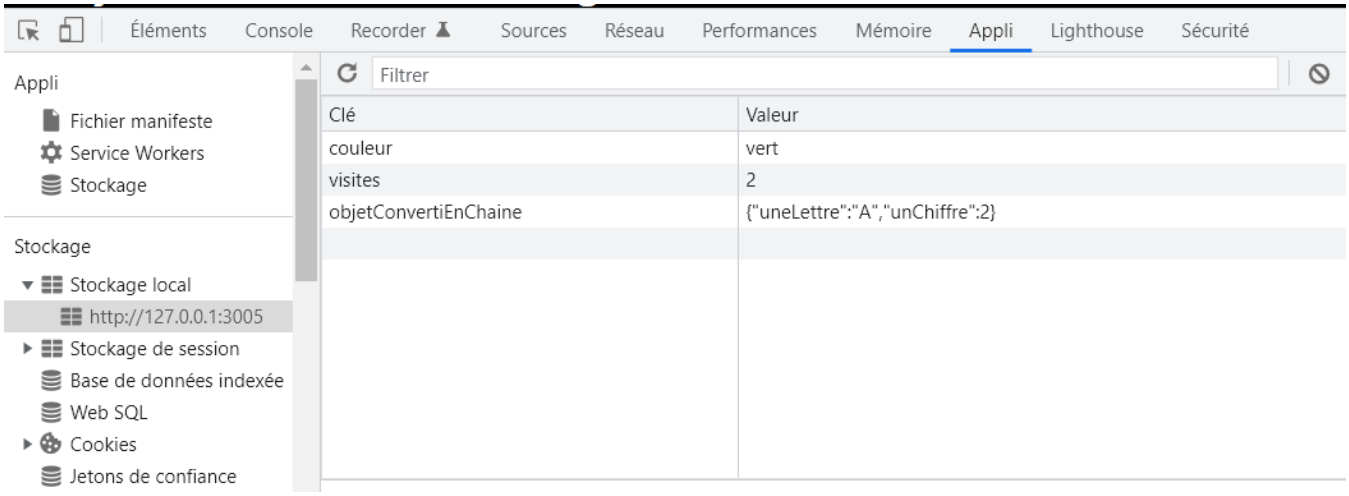
```
//Déclaration de l'objet
let monObjet = {
    uneLettre: "A",
    unChiffre: 2
};

//*************** CONVERSION ET STOCKAGE
let monObjet_EnChaine = JSON.stringify(monObjet);
localStorage.setItem("objetConvertiEnChaine", monObjet_EnChaine);
console.log(typeof monObjet_EnChaine);//Retourne : string


//*************** LECTURE ET CONVERSION
monObjet_EnChaine = localStorage.getItem("objetConvertiEnChaine");
monObjet = JSON.parse(monObjet_EnChaine);
console.log(typeof monObjet);//Retourne : object
```

Examen et espace alloué

L'espace local alloué, bien que considérablement supérieur à celui disponible via un *cookie*, est cependant limité. Personne ne tient à voir son disque dur submergé de données créées par des milliers de sites consultés. Il faut donc gérer cet espace raisonnablement avec efficacité et parcimonie. Une exception **QUOTA_EXCEEDED_ERR** est déclenchée si le quota est atteint.

Visualisation des données stockées

Précisions que les outils de développement et diverses consoles peuvent donner accès à des vues détaillées du stockage. Par exemple sous Chrome, l'onglet « *Appli* » :



Quelques références utiles

- Alsacreations (Janvier 2017). Stockage des données locales - Web Storage. Repéré à http://www.alsacreations.com/article/lire/1402-web-storage-localstorage-sessionstorage.html
- Mozilla(Novembre 2022). Utiliser l'API Web Storage. Repéré à: https://developer.mozilla.org/fr/docs/Web/API/Web_Storage_API/Using_the_Web_Storage_API