Travaux dirigés n°9
IMAC 1

HTML5 – API Javascript

Dans le TD précédent, nous avons découvert quelques-unes des nouvelles spécifications de HTML5 : elles concernaient le fond (HTML) et la forme (CSS3). Pour ce dernier TD, nous verrons comment la notion d'interactivité a évolué pour maintenant faire partie intégrante de HTML5.

Avant-propos

Comme nous l'avons vu précédemment, HTML5 va encore plus loin dans la séparation du fond et de la forme. De nouvelles balises, plus « sémantiques » ont fait leur apparition pour caractériser encore mieux le type de contenu qu'elles renferment. CSS3, quant à lui, propose de nouvelles propriétés permettant une mise en forme plus subtile.

Voyons aujourd'hui ce qu'il en est de l'interactivité. Avec HTML5, plusieurs surcouches ont été apportées au langage javascript (toujours lui). Ces nouvelles APIs sont directement présentes dans le navigateur et propose un certain nombre de nouvelles fonctionnalités : géolocalisation, stockage, communication avec des fichiers, etc...

Nous allons commencer par implémenter la géolocalisation afin d'afficher une carte centrée sur notre position courante, à l'aide de l'API de Google Maps. Puis, nous explorerons le stockage local directement au sein du navigateur grâce au Web Storage.

Note : Pour ce TD (comme tous les autres, finalement), il est vivement recommandé d'utiliser Google Chrome, ou Firefox + Firebug. Ceux-ci vous fourniront de véritables outils de débug.

Géolocalisation

- ### Fonctionnement de la géolocalisation

La géolocalisation est la possibilité de se placer géographiquement dans l'espace.
Avec l'introduction de la mobilité et des appareils reliés en permanence à Internet (smartphones, tablettes, ...), la possibilité de pouvoir définir avec précision sa propre position géographique, vis- à-vis d'environnement ou par rapport à d'autres utilisateurs, est devenue un enjeu important.

De nombreuses applications ont déjà vu le jour : des recherches contextualisées (restaurant japonais le plus proche), des calculs d'itinéraires (comment aller au Fifth Bar depuis la sortie du métro ?), le partage de positions géographiques sur les réseaux sociaux (se checker sur Facebook au stade de l'Abbé Deschamps)...

Il existe plusieurs moyens d'obtenir ces informations géographiques : par GPS, par triangulation des réseaux téléphoniques (GSM, 3G), par triangulation des réseaux Wifi à portée ou encore par adresse IP. Le navigateur peut combiner plusieurs de ces informations pour déterminer la position actuelle de l'utilisateur. HTML5 permet d'utiliser une **API de géolocalisation** récupérant ces données.

- Implémentation

Avant toute chose, vous devez détecter si le navigateur de l'utilisateur permet la géolocalisation. En effet, comme toutes les nouvelles APIs, seuls les navigateurs récents (Firefox 3.5+, Safari 5+, Chrome 5+, IE9+) ont accès à ces données.

Le test est très simple. Dans un document javascript, il vous suffit d'ajouter la condition suivante:

```
if(navigator.geolocation) {
      // L'API est disponible
} else {
      alert('La géolocalisation n'est pas prise en compte') ;
}
```

Vous l'aurez compris, l'objet **geolocation** présent directement dans le navigateur est justement l'API que l'on veut appeler. Celle-ci possède un certain nombre de méthodes. Ici, on va s'intéresser à **getCurrentPosition()** :

```
navigator.geolocation.getCurrentPosition(function(position) {
      // L'objet position contient les informations de géolocalisation
});
```

Vous notez que, lors de l'appel de cette méthode, le navigateur va vous demander s'il peut accéder à vos données de géolocalisation. En effet, pour des raisons de vie privée compréhensibles, l'utilisateur peut bloquer l'utilisation de l'API s'il le souhaite. Pensez donc à l'activer !

Note importante : sur le réseau de la fac (même sur le wifi), la géolocalisation semble bridée. Vous ne pourrez donc essayer de vous localiser qu'une fois chez vous... Vous pouvez tout de même passer à l'exercice suivant en mettant en place une carte Google Maps, mais vous n'arriverez peut être pas à la centrer sur votre position actuelle...

Exercice :

Grâce à la **documentation**, affichez sur votre page web (remplissez le contenu d'une div, par exemple), les coordonnées (latitude et longitude) de votre position actuelle et le niveau de précision de la longitude et de la latitude (en mètre).

- Google Maps API

Maintenant que nous avons récupéré ces données, nous allons pouvoir les réutiliser pour les afficher au sein d'une carte. Pour cela, nous pouvons utiliser l'**API de Google Maps**.

Commençons par inclure le fichier javascript :

```
<script type="text/javascript"
 src="https://maps.googleapis.com/maps/api/js?v=3.exp&sensor=false">
</script>
```

Créez ensuite une div ayant pour id « **map-canvas** » : il s'agira du conteneur de la carte. Cette div devra faire 100% du body en hauteur et largeur. Veillez également à ce que les balises html et body remplissent également à 100% la hauteur de la fenêtre (le but est d'avoir une carte en plein écran).

- Grâce à la **documentation**, continuez l'affichage de votre carte. Celle-ci sera pour l'instant centrée sur Paris (Lat : 48.8567 et Lng: 2.3508) avec un zoom de 12.
- Grâce à la méthode **setCenter()**, déplacez le centre de la carte vers votre position actuelle.
- Remplacez la méthode **setCenter()** par **panTo()** et constatez la difference.
- Ajoutez un **marqueur** à votre position.
- Modifiez les **options** du marqueur pour ajouter une animation "drop" et rendre le marqueur déplaçable.

Stockage des données locales (Web Storage)

Web Storage permet au navigateur de stocker des informations de façon locale, sans avoir à communiquer avec un quelconque serveur. Il s'agit-là d'une grande nouveauté, introduite par HTML5.

En effet, jusqu'alors, il était compliqué de stocker des informations complexes. L'utilisation des cookies, par exemple, n'était pas idéale : avec une taille limitée, ils n'étaient pas adaptés au stockage local. La solution la plus pratique était de stocker ces données en ligne, grâce à des appels AJAX : pratique lourde en ressources !

Désormais, Web Storage met en place deux espaces de stockage. Le premier, *Session Storage*, est destiné à la mémorisation de données ayant une faible durée de vie : celles-ci seront supprimées à la fermeture du navigateur. **Local Storage**, quant à lui, bénéficie d'une durée de vie plus longue possède une portée étendue à toutes les pages d'un même domaine. Au sein d'un même site, on peut donc mettre en place un vrai espace de stockage...

Mise en place

Là encore, la première chose à faire est de vérifier que le navigateur supporte Local Storage (que nous allons utiliser) :

```
if(typeof(localStorage) != 'undefined'){
 // On stocke une variable « data » qui prend comme valeur une chaîne
      localStorage.setItem('data', 'données de test');
// On récupère une variable
Var data = localStorage.getItem('data');
} else { alert('localStorage is not supported'); }
```

Comme vous pouvez le constater, son utilisation est très simple. Il vous suffit d'attribuer une variable à stocker (son nom et la valeur). Vous pourrez alors la récupérer quand vous le souhaitez : elle sera conservée par le navigateur pendant toute la navigation.

Exercice :

En suivant cet exemple, créez un compteur de visites d'une page web. Vous devrez afficher le nombre de venues (= actualisations de la page) de l'utilisateur.
Astuce : pensez à utiliser la fonction **Number()** pour convertir un objet en nombre.

- Stockage complexe

Web Storage ne permet pas uniquement de stocker des chaînes de caractères. Il peut également, et là c'est plus intéressant, mémoriser des objets entiers. Pour cela, nous allons utiliser la syntaxe JSON, que vous connaissez bien :

```
// Construction d'un objet JSON
var album = {}
// Remplissage de l'objet
album.artist = 'Orselsan';
album.title  = 'Le chant des Sirènes';
album.year   = '2011';
// On place l'objet en mémoire
// Pour cela, on doit transformer le JSON en chaîne de
caractères
localStorage.setItem('album', JSON.stringify(album));
// Pour le récupérer, on transforme la chaîne en JSON
var recupalbum = JSON.parse(localStorage.getItem('album'));
```

Comme vous pouvez le constater, les objets sont transformés en chaîne de caractère pour le stockage (grâce à la méthode **JSON.stringify()**). Une fois stockés, vous pourrez retrouver leur structure complète en re-parsant la chaîne de caractères en question grâce à **JSON.parse()**.

Exercice :

En vous appuyant sur cette technique, vous allez créer une application web de « to-do list ». Les tâches à réaliser seront mémorisées dans le navigateur (Local Storage) et listées sur la page, accompagnées d'un formulaire permettant à l'utilisateur d'en ajouter.

- Commencez par créer un formulaire pour l'ajout de tâche. Il sera composé du titre de la tâche, de la date à respecter pour sa réalisation (tiens, un datePicker ?), d'une description plus complète et d'un état d'avancement (0-100%).

- Traitez ce formulaire grâce à jQuery. A la soumission de ce dernier, récupérez-en les valeurs pour les stocker dans le Local Storage, sous forme d'objets JSON.

- Listez les tâches déjà ajoutées. Vous pourrez utiliser du CSS3 pour un rendu en forme de post-its, par exemple. Et pourquoi pas des animations au survol ?

- Faites en sorte de pouvoir modifier l'état d'avancement d'une tâche en cliquant sur un bouton (par exemple). Une fois effectuée, son affichage devra être modifié (contenu barré ? Changement de couleur ?)

- Ajoutez un bouton pour chaque tâche à fin de la supprimer.