



Université Jendouba



Faculté des Sciences Juridiques  
Économiques et de Gestion

# Développement Web: HTML 5 Les APIs avancées

Manel Ben Sassi

Université de Jendouba, FSJSEG

*bensassi.manel@gmail.com*

# Sommaire

## 1 Géo-localisation

- Comprendre un peu le concept
- La géolocalisation par les méthodes !
- La géolocalisation : La gestion des erreurs
- La géolocalisation : Google Maps

## 2 Drag and Drop

## 3 Le Stockage coté client

- Les cookies
- Web Storage
  - Session Storage
  - Local Storage

## 4 Les Web Sockets

## 5 Bibliographie

# Les API de HTML5 .... sa force



# HTML5 : Les API de HTML 5

- L'apogée de l'**AJAX** et du **web 2.0** ainsi que des bibliothèques **Javascript** telles que **jQuery** a rendu son utilisation bien plus agréable et efficace. De plus, de moins en moins d'utilisateurs désactivent le Javascript et il est même souvent impossible de le désactiver sur certains navigateurs.



L'API Javascript (les objets et les méthodes utilisables) a été généreusement enrichi avec de nouvelles fonctionnalités. En voici quelques unes ....

## 2. La Géolocalisation



# La Géo-localisation

Il ne s'agit pas strictement d'une spécification de l'HTML5, mais elle y est souvent associée ()

Il est possible grâce à l'API de géo-localisation d'accéder aux coordonnées de l'utilisateur si celui-ci a accepté de partager sa position via le bandeau s'affichant en haut de page

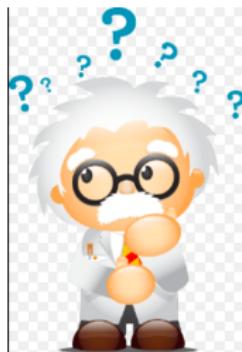
*Géolocalisation !*

*Dis moi où tu es, je te dirai ce que je peux te vendre*

Ses usages sont nombreux et souvent corrélés avec des bases de données de renseignements géographiques :

- Plans/cartes, calculs de position et d'itinéraires
- Renseignements locaux en mobilité (points d'intérêts proches)
- Résultats contextualisés sur les moteurs de recherche
- Méta-information jointes aux photos/vidéos

# De quelles informations a-t-on besoin pour se géolocaliser ?



# La Géo-localisation : Les informations nécessaires !

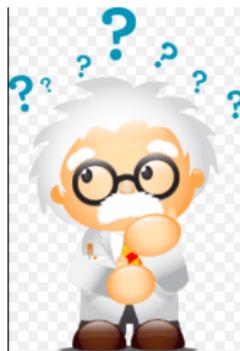
Dans l'espace, nous avons besoin de 3 coordonnées nécessaires



On peut alors très facilement disposer d'informations telles que :

- La latitude, la longitude, et l'altitude de l'utilisateur.
- Son orientation par rapport au Nord.
- La vitesse à laquelle il se déplace.

# De quels moyens dispose un navigateur pour se géolocaliser ?



# La Géo-localisation

Différentes techniques sont mises à contribution avec + ou - de précision pour obtenir les coordonnées de géolocalisation. Elles peuvent être combinées pour affiner le résultat au cours du temps.

De quels moyens dispose un navigateur pour se géolocaliser ?

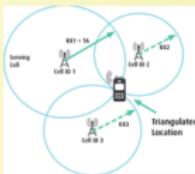


GPS (global position station)  
(mobiles)



Adresse IP  
(bases de données)

Triangulation Wifi  
(mobiles et bases de données)



Triangulation GSM ou 3 ou 4 G  
(mobiles et /ou bases de données)

Vous devez connaître que ....

- ▶ Plusieurs techniques peuvent être combinées (les techniques mobiles comme GPS et base de données par exemple)

# La Géo-localisation

La géolocalisation ... il faut coder et tester pour comprendre !

CODER AND TESTER

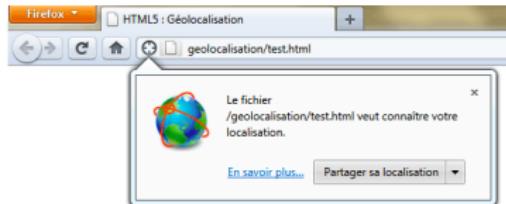


# La Géo-localisation : un exemple

Disponibilité de l'API ?

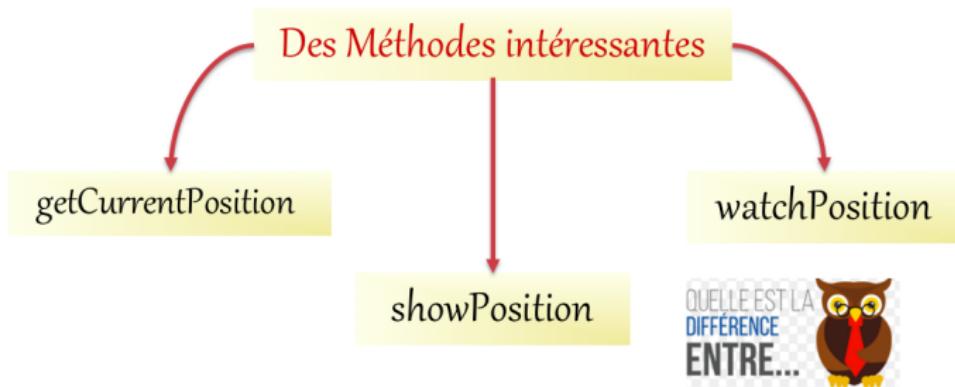
```
If(navigator.geolocation) {  
    // L'API est disponible  
} else {  
    // Pas de support, proposer une alternative ?  
}
```

Confidentialité de la position !! => Des notifications pour demander l'accès à vos coordonnées :)



# La Géo-localisation : un exemple

Les méthodes importantes qu'il faut connaître !

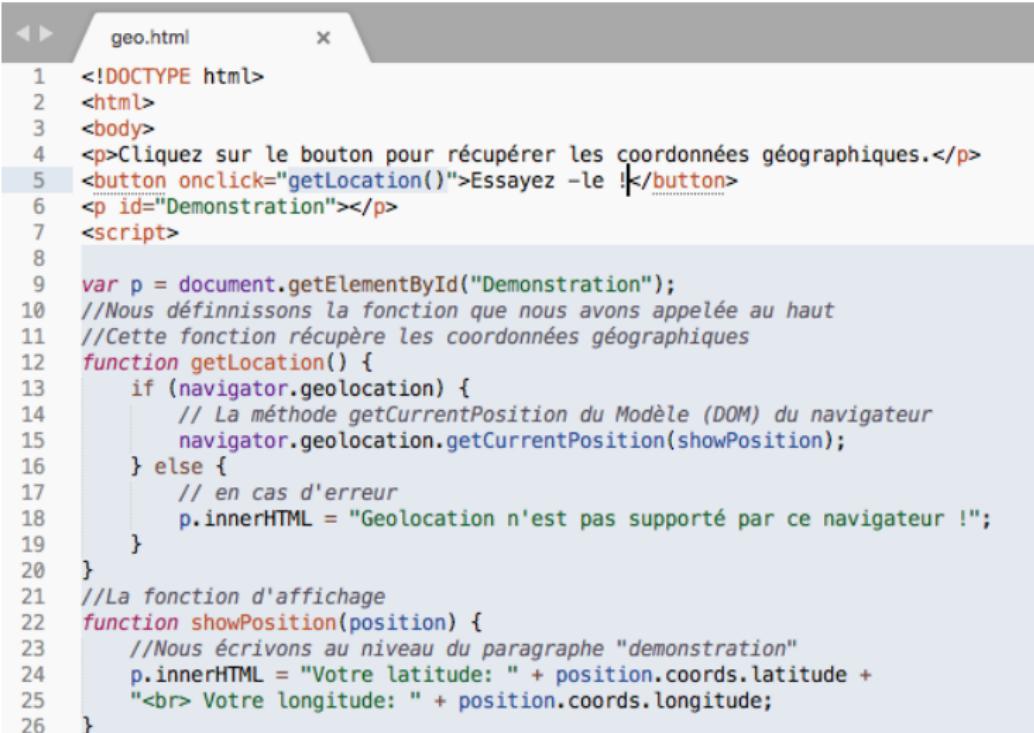


La géolocalisation repose sur 3 méthodes intéressantes :

- ▶ **getCurrentPosition** permettant un ciblage ponctuel
- ▶ **showPosition** permet d'afficher les coordonnées résultant de la méthode "*getCurrentPosition*"
- ▶ **watchPosition** pour un suivi continu

# La Géo-localisation : un exemple

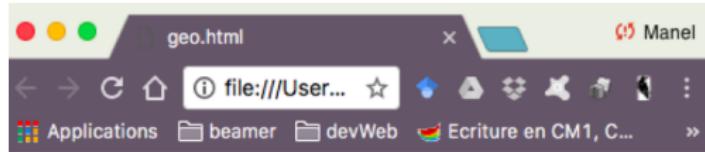
Essayons de comprendre le Script !



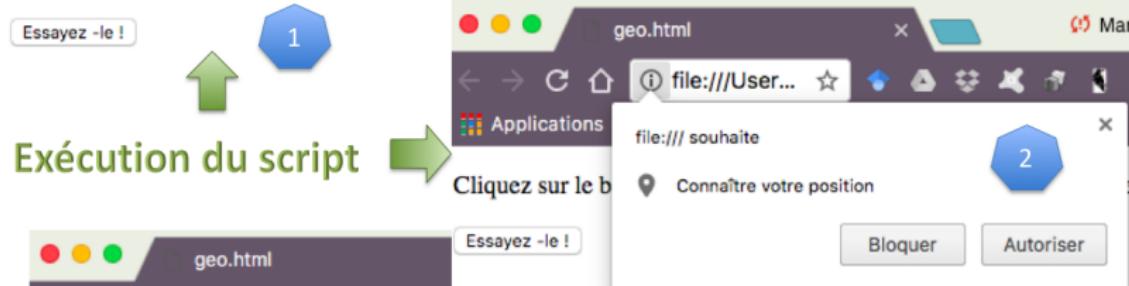
The screenshot shows a browser developer tools window with the title bar "geo.html". The code editor contains the following script:

```
1  <!DOCTYPE html>
2  <html>
3  <body>
4  <p>Cliquez sur le bouton pour récupérer les coordonnées géographiques.</p>
5  <button onclick="getLocation()">Essayez -le !</button>
6  <p id="Demonstration"></p>
7  <script>
8
9  var p = document.getElementById("Demonstration");
10 //Nous définissons la fonction que nous avons appelée au haut
11 //Cette fonction récupère les coordonnées géographiques
12 function getLocation() {
13     if (navigator.geolocation) {
14         // La méthode getCurrentPosition du Modèle (DOM) du navigateur
15         navigator.geolocation.getCurrentPosition(showPosition);
16     } else {
17         // en cas d'erreur
18         p.innerHTML = "Geolocation n'est pas supporté par ce navigateur !";
19     }
20 }
21 //La fonction d'affichage
22 function showPosition(position) {
23     //Nous écrivons au niveau du paragraphe "demonstration"
24     p.innerHTML = "Votre latitude: " + position.coords.latitude +
25     "<br> Votre longitude: " + position.coords.longitude;
26 }
27 </script>
28 </body>
```

# La Géo-localisation : un exemple



Cliquez sur le bouton pour récupérer les coordonnées géographiques.



Exécution du script →

Cliquez sur le bouton



Cliquez sur le bouton pour récupérer les coordonnées géographiques.

Essayez -le !

Votre latitude: 36.806494799999996

Votre longitude: 10.181531600000001

3

# La Géo-localisation : GetCurrentPosition

Les données retournées par la méthode `getCurrentPosition` :

- Retournées obligatoirement : `coords.latitude`, `coords.longitude`,  
`coords.accuracy`,
- Rétournées si disponibles :
  - `coords.altitude` (l'altitude en mètres par rapport à la mer)
  - `coords.altitudeAccuracy` (précision de l'altitude)
  - `coords.heading` (mesure de l'angle par rapport au nord)
  - `coords.speed` (mètre par seconde)
  - `timestamp` (date et durée de la réponse à la requête)

# La Géo-localisation : Pour suivre un élément mobile

Pour suivre la position d'un élément, il suffit de remplacer **getCurrentPosition()** par la méthode **watchPosition()**.

```
1  <!doctype html>
2  <html lang="fr">
3  <head>
4  <meta charset="utf-8">
5  <title>HTML5 : Géolocalisation</title>
6  </head>
7  <body>
8
9  <!-- Un élément HTML pour recueillir l'affichage -->
10 <div id="infoposition"></div>
11 <script>
12 // Fonction de callback en cas de succès
13 function surveillePosition(position) {
14     var infopos = "Position déterminée :\n";
15     infopos += "Latitude : "+position.coords.latitude +"\n";
16     infopos += "Longitude: "+position.coords.longitude+"\n";
17     infopos += "Altitude : "+position.coords.altitude +"\n";
18     infopos += "Vitesse : "+position.coords.speed +"\n";
19     document.getElementById("infoposition").innerHTML = infopos;
20 }
21
22 // On déclare la variable survId afin de pouvoir par la suite annuler le suivi de la
position
23 var survId = navigator.geolocation.watchPosition(surveillePosition);
24
25 if(navigator.geolocation)
26     navigator.geolocation.getCurrentPosition(maPosition);
27 </script>
28 </body>
29 </html>
```

# La Géo-localisation : Pour suivre un élément mobile

Résultat !



Position déterminée : Latitude : 36.613883746801484 Longitude: 8.968781021019542 Altitude : null Vitesse : null

Pour stopper ce suivi continu, il faut réexploiter la variable obtenue (qui est en quelque sorte un pointeur vers le processus de suivi) avec la méthode **clearWatch()**.

// Annule le suivi de la position si nécessaire.

```
navigator.geolocation.clearWatch(survId);
```

# La Géo-localisation : Un exercice

## La géolocalisation ... il faut coder et tester Maintenant !

CODER AND TESTER

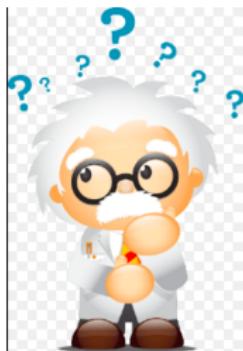


Figure – Exercice

### Exercice

*Modifiez et enrichissez le code de suivi en temps réel avec deux boutons pour lancer et arrêter l'appel*

# La géolocalisation et la gestion des erreurs ?



# La Géo-localisation : la gestion des erreurs

## Retenez !

- Les appels aux méthodes de géolocalisation sont **asynchrones** => La détection peut nécessiter **un temps variable et incertain**.
- **Asynchrones** : les appels retournent tout de suite pour donner la main à la suite du code, tout en déclenchant les techniques permettant de réunir toutes les informations nécessaires.
- Une fois ces informations obtenues de la part du navigateur, une fonction de **callback** définie par le développeur (vous !) peut être appelée. Celle-ci recevra en argument toutes les valeurs réunies dans les propriétés d'un objet JavaScript.

# La Géo-localisation : la gestion des erreurs

Les fonctions Callback ou "de rappel" par définition ...

- C'est une fonction qui est passée en paramètre à une autre fonction.
- Elle est exécutée au "bon moment".
- Elle est souvent utilisée pour la programmation évenementielle ou encore la programmation Asynchrone (exemple l'AJAX).

## Un exemple de fonction auto-appellante pour se rappeler

```
1 //fonction d'affichage
2 var show = function affichage(c){
3     console.log("somme "+c);
4 }
5 //focntion somme
6 function sum(a,b, affichage){
7     var c = a+b;
8     affichage(c);
9     return c;
10 }
11 // appel de la fonction
12 var c = sum(10,4,show);
13 //afficher la valeur
14 console.log(c);
```

# La Géo-localisation : la gestion des erreurs

La méthode `getCurrentPosition()` peut prendre un second paramètre : *le callback d'erreur*. C'est la fonction qui sera appelée dans les cas où :

- L'utilisateur refuse l'autorisation d'accès à sa position
- L'emplacement de l'utilisateur n'ait pas pu être déterminé
- Le service de géolocalisation ne réponde pas assez vite

Les retours d'erreurs potentiels **sont très importants** et méritent d'être pris en compte dans toute application web. Ils permettent de savoir si l'utilisateur a refusé d'être géolocalisé, ou si la position n'a pu être obtenue.

# La Géo-localisation : la gestion des erreurs

Le code pour la gestion de l'erreur !

```
27 function showError(error) {
28     switch(error.code) {
29         case error.PERMISSION_DENIED:
30             x.innerHTML = "L'utilisateur a rejeté la requête pour la géolocalisation!"
31             break;
32         case error.POSITION_UNAVAILABLE:
33             x.innerHTML = "Les informations de localisation ne sont pas disponibles!"
34             break;
35         case error.TIMEOUT:
36             x.innerHTML = "Temps de traitement de la requête est dépassé."
37             break;
38         case error.UNKNOWN_ERROR:
39             x.innerHTML = "Une erreur inconnue est survenue"
40             break;
41     }
42 }
```

## Exercice

Enrichissez votre code initial avec la gestion des erreurs et testez avec des navigateurs différents !

# La Géo-localisation et Google Maps

L'API Google Maps V3 est très aisée à exploiter en combinaison à la géolocalisation. Elle comprend de nombreuses fonctionnalités pour afficher une carte géographique, positionnée et équipée de marqueurs.

**Etape 1** Réserver une zone pour l'affichage de la carte

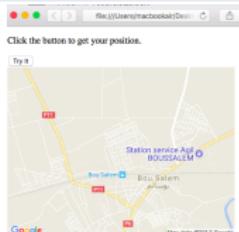
**Etape 2** Faire appel à un "service map" comme "**Google Maps**"

**Etape 3** Récupérer l'altitude et la logitude et l'afficher sur la carte  
(sur une image statique)

# La Géo-localisation et Google Maps

## Le code source et un exemple de rendu !

```
22 function showPosition(position) {  
23     var latlon = position.coords.latitude + "," + position.coords.longitude;  
24     var img_url = "https://maps.googleapis.com/maps/api/staticmap?center="  
25     +latlon+"&zoom=14&size=400x300&key=AIzaSyBu-916DdpKAjTmJNlgnngS6HL_kDIKU0aU";  
26     document.getElementById("ZoneMap").innerHTML = "<img src='"+img_url+"'>";  
27 }  
28 }
```



# La Géo-localisation et Google Maps

## Un Bonus : Le marqueur !

```
3 marker = new google.maps.Marker({  
4     position: latlng,  
5     map: map,  
6     title:"Vous êtes ici",  
7     icon: "fleche.png"  
8});
```



Important Notice

## 2. Drag and Drop



# Le Drag and Drop

Il est également possible d'effectuer des "glisser-déposer" dans une page web. Les étapes à suivre si l'on souhaite pouvoir déplacer un élément d'une liste :

- ➊ Déclarer cet élément avec l'attribut "**draggable**" à true et l'évènement associé :

```
<li draggable ="true" ondragstart=""> Élément de ma liste </li>
```

- ➋ Définir une zone potentielle de destination (ou la zone de réception) :

```
<div ondrop="drop(this, event)">
```

- ➌ Définir les fonctions Javascript correspondant :

- *ondragstart* : sélectionne un élément déplaçable
- *ondragend* : permet de signaler à l'objet déplacé que son déplacement est terminé, que le résultat soit un succès ou non.
- *ondrag* : fait glisser l'élément vers une zone de réception
- *ondrop* : relâche l'élément dans la zone de réception

# Le Drag and Drop

## Initialisation d'un déplacement avec l'objet `dataTransfer`

- L'objet `dataTransfer` est utilisé généralement avec `dragstart` et `drop`.
- Il permet de définir et de récupérer les informations relatives au déplacement en cours d'exécution.
- L'objet `dataTransfer` permet de réaliser trois actions (toutes facultatives) :
  - ⇒ Sauvegarder une chaîne de caractères qui sera transmise à l'élément HTML qui accueillera l'élément déplacé. La méthode à utiliser est `setData()`.
  - ⇒ Définir une image utilisée lors du déplacement. La méthode concernée est `setDragImage()`.
  - ⇒ Spécifier le type de déplacement autorisé avec la propriété `effectAllowed`.

# Le Drag and Drop

## Retenez lors du déplacement de l'élément ...

Il faut savoir quel élément est déplacé et dans quelle zone il est déposé ⇒  
Il est préférable d'associer **un identifiant aux éléments déplaçables** et de récupérer cette information au moment de déposer l'élément dans la zone de réception (**Utile lorsque vous travaillez sur deux pages HTML et pour communiquer des données !**)

⇒ En employant les fonctions :

1. **dataTransfer.setData** à l'appel de l'évènement *ondragstart*
2. **dataTransfer.getData** à l'appel de l'évènement *ondrop*

On fait également appel à la fonction "**preventDefault**" qui empêche la propagation de l'élément.

# Le Drag and Drop

## La méthode setDragImage() ...

**Personnalisation de l'affichage** Elle permet de définir une image qui se placera sous le curseur pendant le déplacement de l'élément concerné

**Utilisation** La méthode prend trois arguments en paramètres. Le premier est un élément <img> contenant l'image souhaitée, le deuxième est la position horizontale de l'image et le troisième est la position verticale



Comment ?

# Le Drag and Drop

## Exemple

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8" />
    <title>Exemple Drag and Drop d'une image</title>
  </head>
  <body>
    <div draggable="true" id="1">Une deuxième zone !</div>
    <script>
      var dragImg = new Image();
      dragImg.src = 'question.png';
      document.querySelector('*[draggable="true"]').addEventListener('dragstart', function(e){
        e.dataTransfer.setDragImage(dragImg, 20, 20);
      })
    </script>
  </body>
</html>
```

Figure – Interprétez le code !

# Le Drag and Drop

## La zone de drop...

Un élément en cours de déplacement ne peut pas être déposé n'importe où, il faut pour cela définir une zone de "*drop*" (zone qui va permettre de déposer des éléments) qui ne sera, au final, qu'un simple élément HTML.

Les zones de drop prennent généralement en charge quatre événements :

**dragenter** qui se déclenche lorsqu'un élément en cours de déplacement entre dans la zone de drop.

**dragover** qui se déclenche lorsqu'un élément en cours de déplacement se déplace dans la zone de drop.

**dragleave** qui se déclenche lorsqu'un élément en cours de déplacement quitte la zone de drop.

**drop** qui se déclenche lorsqu'un élément en cours de déplacement est déposé dans la zone de drop.

# Le Drag and Drop

## La zone de drop et le comportement du navigateur...

Par défaut, le navigateur interdit de déposer un quelconque élément où que ce soit dans la page Web.

Objectif ⇒ Annulation d'une action par défaut, **preventDefault()** !

Cette méthode va devoir être utilisée au travers de l'événement dragover.

```
<div id="draggable" draggable="true">Je peux être déplacé !</div>


Essayez ce code !!



```
function allowDrop(ev) {
    ev.preventDefault();
}
```



35 / 78


```

# Le Drag and Drop

La fonction drop !

```
function drop(ev) {  
    ev.preventDefault();  
    var data = ev.dataTransfer.getData("text");  
    ev.target.appendChild(document.getElementById(data))  
}
```

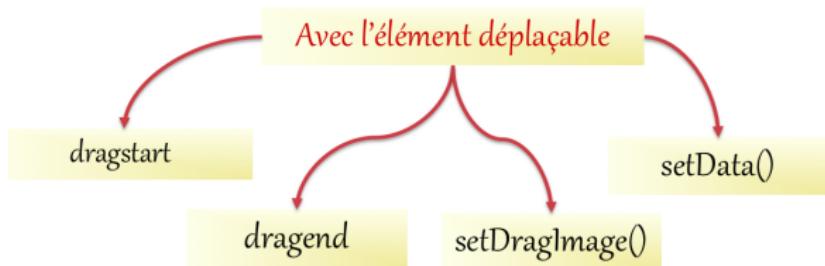
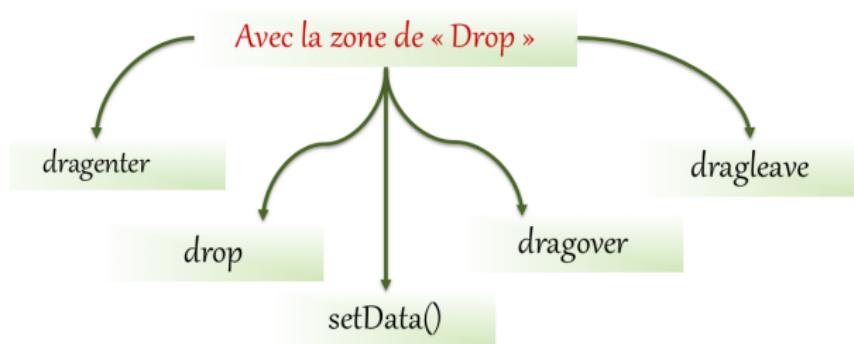
# Le Drag and Drop

```
1  <!DOCTYPE HTML>
2  <html>
3  <head>
4  <style>
5  #div1, #div2 { *** }
6  }
7  </style>
8  <script>
9  function allowDrop(ev) {
10    ev.preventDefault();
11  }
12
13  function drag(ev) {
14    ev.dataTransfer.setData("text", ev.target.id);
15  }
16
17  function drop(ev) {
18    ev.preventDefault();
19    var data = ev.dataTransfer.getData("text");
20    // Ajout de l'élément cloné à la zone de drop actuelle
21    ev.target.appendChild(document.getElementById(data));
22  }
23
24</script>
25</head>
26<body>
27
28<h2>Drag and Drop</h2>
29<p>Drag the image back and forth between the two div elements.</p>
30
31<div id="div1" ondrop="drop(event)" ondragover="allowDrop(event)">
32  
33</div>
34
35<div id="div2" ondrop="drop(event)" ondragover="allowDrop(event)"></div>
36
37</body>
38</html>
```



# Le Drag and Drop : Conclusion

Pour conclure ! ⇒ Les méthodes associées à chaque éléments

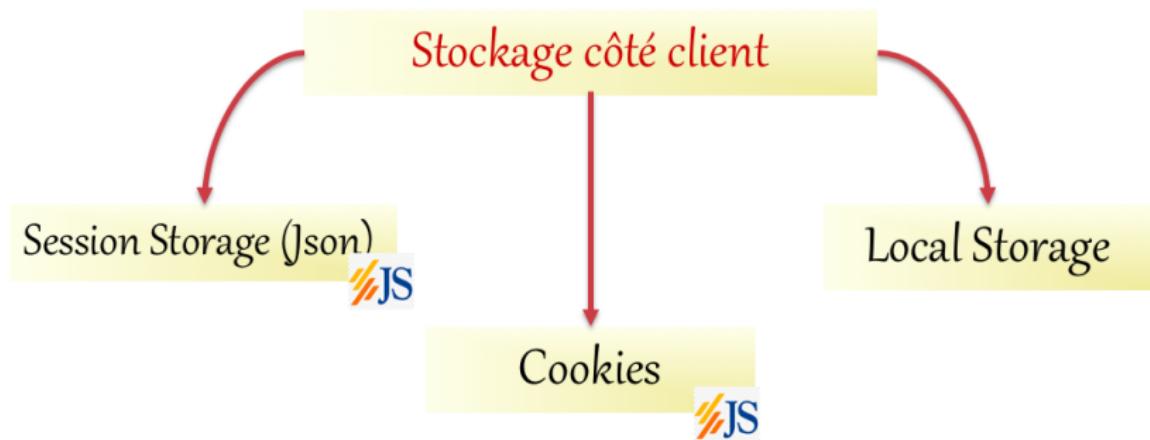


### 3. Le Stockage coté client

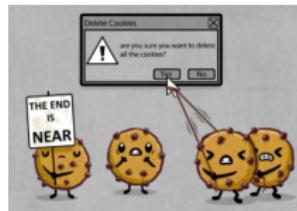


# Stockage côté client

Le stockage côté client peut se faire de plusieurs manières.



# Les cookies



# Les cookies ...

## Un "Cookie" ...

- est une chaîne de caractères qu'une page HTML (contenant du code JavaScript) peut écrire à un emplacement UNIQUE et bien défini sur le disque dur du client.
- Cette chaîne de caractères ne peut être lue que par le seul serveur qui l'a générée.

## Que faire avec un cookie ?

# Les cookies ...

## Un "Cookie" ...

- est une chaîne de caractères qu'une page HTML (contenant du code JavaScript) peut écrire à un emplacement UNIQUE et bien défini sur le disque dur du client.
- Cette chaîne de caractères ne peut être lue que par le seul serveur qui l'a générée.

## Que faire avec un cookie ?

- ▶ Transmettre des valeurs (contenu de variables) d'une page HTML à une autre.
  - Par exemple, créer un site marchand et constituer un "**caddie**" pour le client. Caddie qui restera sur son poste et vous permettra d'évaluer la facture finale au bout de la commande. **Sans faire appel à quelque serveur que ce soit.**
- ▶ Personnaliser les pages présentées à l'utilisateur en reprenant par exemple son nom en haut de chaque page.

# Les cookies ...

## Structure d'un cookie

**Nom=Contenu ; expires=expdate ; path=Chemin ;  
domain=NomDeDomaine ; secure**

- La variable **Nom** contient le nom à donner au cookie.
- La variable **Contenu** contient le contenu du cookie
- **Exemple** macookie="oui :visite"
- Le mot réservé **expires** suivi du signe "**=**". Derrière ce signe, une date d'expiration représentant la date sous la forme (Wdy, DD-Mon-YYYY HH :MM :SS GMT) à laquelle le cookie sera supprimé du disque dur du client. *Utiliser les fonctions de l'objet Date*
- **path** représente le chemin de la page qui a créé le cookie.

# Les cookies ...

## Structure d'un cookie

Nom=Contenu ; expires=expdate ; path=Chemin ;  
domain=NomDeDomaine ; secure

- **domain** représente le nom du domaine de cette même page
- **secure** prend les valeurs "true" ou "false" permet de spécifier que le cookie sera envoyé uniquement si la connexion est sécurisée selon que le cookie doit utiliser des protocoles HTTP ou HTTPS.
- *Les arguments path, domain et secure sont facultatifs.* Lorsque ces arguments sont omis, les valeurs par défaut sont prises. Pour secure, la valeur est "False" par défaut.

# Gestion des cookies ...

Ecrire un cookie Un cookie est une propriété de l'objet document (la page HTML chargée dans le navigateur) alors l'instruction d'écriture de cookie est :

```
document.cookie = Nom + "=" + Contenu + "; expires=" +
                  expdate.toGMTString();
```



```
var Nom = "MonCookie" ; // nom du cookie
var Contenu = "Hé... Vous avez un cookie sur votre disque !" ; // contenu du cookie
var expdate = new Date () ; // crée un objet date indispensable puis rajoutons lui 10 jours d'ex.
expdate.setTime (expdate.getTime() + ( 10 * 24 * 60 * 60 * 1000)) ;
document.cookie = Nom + "=" + Contenu + "; expires=" + expdate.toGMTString() ;
```

# Gestion des cookies ...

**Lecture d'un cookie** Accéder à la propriété cookie de l'objet document.



```
var LesCookies ; // pour voir les cookies  
LesCookies = document.cookie ; // on met les cookies dans la variable LesCookies
```

**Modification d'un cookie** Modifier le contenu de la variable **Contenu** puis réécrire le cookie sur le disque dur du client



```
Contenu = "Le cookie a été modifié..." ; // nouveau contenu  
document.cookie = Nom + "=" + Contenu + ";" + expires=" + expdate.toGMTString() ; // écriture
```

# Gestion des cookies ...

**Suppression d'un cookie** Positionner la date de péremption du cookie à une valeur inférieure à celle du moment où on l'écrit sur le disque.



```
// on enlève une seconde (ça suffit mais c'est nécessaire)
expdate.setTime(expdate.getTime() - (1000)) ;

// écriture sur le disque
document.cookie = Nom + "=" + Contenu + "; expires=" + expdate.toGMTString() ;
```

# Les cookies ...

## Les "Cookies" ... des limites

On ne peut pas écrire autant de cookies que l'on veut sur le poste de l'utilisateur (client d'une page). Il y a des limites :

- ▶ **Limites en nombre** : Un seul serveur (ou domaine) ne peut pas être autorisé à écrire plus de 20 cookies.
- ▶ **Limites en taille** : un cookie ne peut excéder 4 Ko.
- ▶ **Limites du poste client** : Un poste client ne peut stocker plus de 300 cookies en tout.

## Où sont stockés les cookies ?

# Les cookies ...

## Les "Cookies" ... des limites

On ne peut pas écrire autant de cookies que l'on veut sur le poste de l'utilisateur (client d'une page). Il y a des limites :

- ▶ **Limites en nombre** : Un seul serveur (ou domaine) ne peut pas être autorisé à écrire plus de 20 cookies.
- ▶ **Limites en taille** : un cookie ne peut excéder 4 Ko.
- ▶ **Limites du poste client** : Un poste client ne peut stocker plus de 300 cookies en tout.

## Où sont stockés les cookies ?

- En général, ils sont pour *Netscape*, dans le répertoire de l'utilisateur (si il y a des profils différents) sous le nom de "**cookie.txt**".
- **Microsoft Internet Explorer** stocke les cookies dans des répertoires tels que "*C :\windows\ Cookies*" ou encore "*C :\windows\ tem\ Cookies*".

# Web Storage



# Le stockage du web ou le "Web Storage"

## Le Web Storage

- Est une solution adaptée aux besoins actuels de stockage de données variées, dans le navigateur (côté client !)
- Est une technique *plus puissante* que les cookies qui :
  - sont limités en taille (quelques Ko contre plusieurs Mo pour Web Storage)
  - engendrent un trafic HTTP supplémentaire pour chaque requête (que ce soit pour demander la page web, une image, une feuille de styles, un fichier javascript, etc)



# Session Storage



# Le stockage des sessions

## La sessionStorage

- ⇒ L'interface `sessionStorage` mémorise les données sur la durée d'une session de navigation,
- ⇒ Sa portée est limitée à la fenêtre ou l'onglet actif
- ⇒ Lors de sa fermeture, les données sont effacées
- ⇒ Pas **d'interférence** ⇒ Chaque stockage de session est limité à un domaine

# Local Storage



# Local Storage

Le **Local Storage** est une manière élégante de stocker dans le navigateur des informations facilement.

Par exemple, pour écrire puis lire une valeur dans le Local Storage il suffit d'écrire :

```
1 | localStorage.setItem("name", "John");
2 | alert(localStorage.getItem("name"));
```

- La variable sera toujours disponible si l'utilisateur ferme puis ré-ouvre son navigateur  $\neq$  *Session storage*
- La portée de **Local Storage** est de facto **plus large** : il est possible de l'exploiter à travers plusieurs onglets ouverts pour le même domaine ou plusieurs fenêtres **du même navigateur**.
- L'utilisation du **Local Storage** est *proche de celle des cookies*, mais  $\neq$  aux cookies, **ces informations ne sont jamais communiquées au serveur**. Elles sont ainsi particulièrement adaptées aux applications **offline !!!**

# Local Storage : Usages et précautions

*Le stockage de données dans le navigateur web se prête à différentes applications*

- **Stocker** rapidement des données en cache sans faire intervenir le serveur (*par exemple via AJAX*),
- **Augmenter la performance** ressentie et **faciliter** les développements,
- Une alternative aux cookies et au trafic HTTP supplémentaire qu'ils représentent (*un cookie est envoyé à chaque requête effectuée sur un domaine*),
- **Exploiter** un espace alloué plus important que la limite imposée par les cookies (fixée à 4 Ko),
- **Retrouver des données immédiatement** à la reconnexion ou les mémoriser pour éviter la perte s'il y a déconnexion

## Attention

*Les données ne sont pas cryptées, facilement accessible et modifiable en accédant au navigateur !*

# Local Storage

Quelles méthodes ? Comment y accéder ?

Les méthodes d'accès sont communes :

**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

**.length** : La propriété **.length** renvoie le nombre de paires stockées.

La console JavaScript des navigateurs est un outil essentiel pour tester et vérifier le bon fonctionnement de Web Storage.

# Local Storage

## Stockage

Le 1<sup>er</sup> argument est une chaîne de caractère la clef



```
sessionStorage.setItem("couleur", "vert");
```

```
localStorage.setItem("couleur", "rouge");
```

Very important



# Local Storage

## Récupération

Grâce à la clef définie par `setItem`, il est possible de récupérer les données.



```
var Mycolor = sessionStorage.getItem("couleur » );
```

```
Var Mycolor2 = localStorage.getItem("couleur");
```



Very important



# Local Storage

## Suppression partielle



```
sessionStorage.removeItem("couleur ");  
localStorage.removeItem("couleur");
```

Figure – Suppression d'un élément (une variable)



# Local Storage

## Suppression totale



```
sessionStorage.clear();  
localStorage.clear();
```

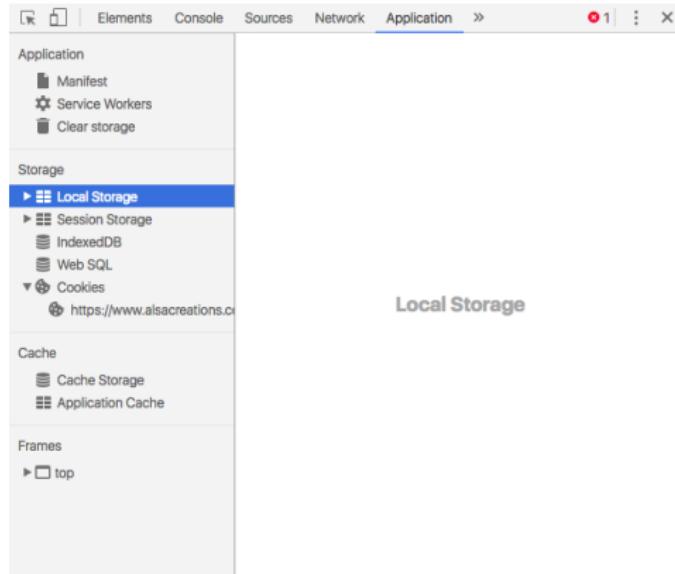
Figure – Suppression de toutes les variables



# Local Storage

## Outils de développement

Les outils de développement et diverses consoles peuvent donner accès à des vues détaillées du stockage. Par exemple sous Chrome, l'onglet "Application" :



# Exercices et Démonstration

## CODER AND TESTER



# Utilisation de JSON

## Utilisation de JSON

**N'OUBLIEZ PAS !** Web Storage est bien pratique pour stocker de simples chaînes de texte.



Comment manipuler des données complexes telles que les Objets de JS

# Utilisation de JSON

## Utilisation de JSON

**N'OUBLIEZ PAS !** Web Storage est bien pratique pour stocker de simples chaînes de texte.



Comment manipuler des données complexes telles que les Objets de JS



Il faut les linéariser au préalable en chaînes de texte

# JSON ...

Le format JSON est la solution !

JSON un petit rappel ...

- Format léger et vraiment très simple

Usages

- ▶ Sert à stocker des données (ex : fichier de configuration)
- ▶ Sert à échanger des données à travers le réseau, entre clients et serveurs (ex : sérialisation / désérialisation)
- ▶ Sert à décrire un objet JS mais sous une forme textuelle dans le code source

# Exemple de définition JSON

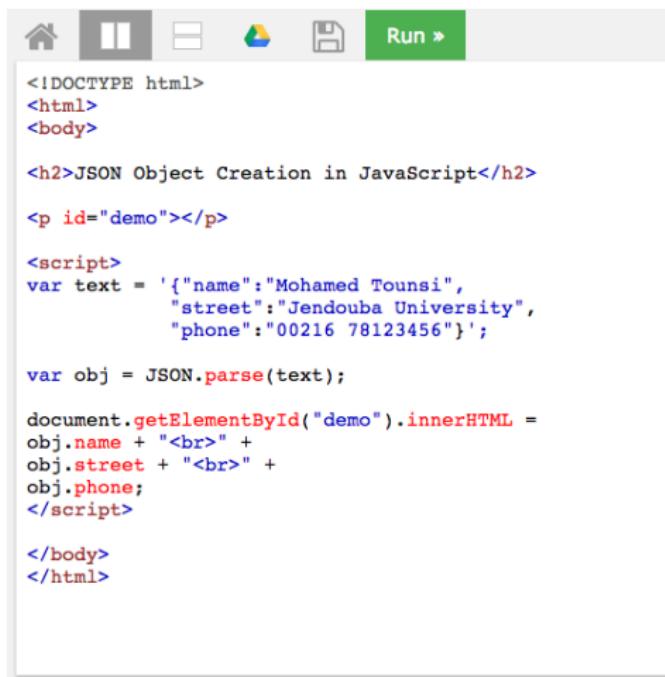
```
{  
  "fruits": [  
    { "kiwis": 3,  
      "mangues": 4,  
      "pommes": null  
    },  
    { "panier": true }  
  ],  
  "legumes":  
    { "patates": "amandine",  
      "figues": "de barbarie",  
      "poireaux": false  
    }  
}
```

courses.json

```
var courses = {  
  "fruits": [  
    { "kiwis": 3,  
      "mangues": 4,  
      "pommes": null  
    },  
    { "panier": true }  
  ],  
  "legumes":  
    { "patates": "amandine",  
      "figues": "de barbarie",  
      "poireaux": false  
    }  
};  
  
if (courses.legumes.poireaux) {  
  console.log("j'ai des poireaux !");  
}
```

processCourses.js

# Exemple d'utilisation JSON



The screenshot shows a browser's developer tools interface with a code editor. The toolbar includes icons for home, back, forward, search, and run. The code editor contains the following HTML and JavaScript:

```
<!DOCTYPE html>
<html>
<body>

<h2>JSON Object Creation in JavaScript</h2>

<p id="demo"></p>

<script>
var text = '{"name":"Mohamed Tounsi",
            "street":"Jendouba University",
            "phone": "00216 78123456"}';

var obj = JSON.parse(text);

document.getElementById("demo").innerHTML =
obj.name + "<br>" +
obj.street + "<br>" +
obj.phone;
</script>

</body>
</html>
```

## JSON Object Creation in JavaScript

Mohamed Tounsi  
Jendouba University  
00216 78123456

# Utilisation de JSON

Le format JSON est la solution !

Deux méthodes équipent les navigateurs modernes :

- **JSON.stringify()** prend en paramètre un objet et renvoie une chaîne de texte linéarisée
- **JSON.parse()** L'inverse de la 1<sup>re</sup> qui reforme un objet à partir de la chaîne linéarisée.

Des frameworks tels que **jQuery** sont équipés de fonctions similaires (*parseJSON*) pour les anciens navigateurs qui ne seraient pas équipés en natifs de telles méthodes de conversion.



# JSON pour Le web Storage

## JSON pour Le web Storage par les exemples !

### Stockage

```
// Déclaration de l'objet
var monobjet = {
    propriete1 : "valeur1",
    propriete2 : "valeur2"
};
//
var monobjetjson = JSON.stringify(monobjet);
// Déclaration de l'objet
sessionStorage.setItem("objet",monobjetjson);
```

# JSON pour Le web Storage

## JSON pour Le web Storage par les exemples !

### Lecture

```
// Récupérer la variable de la session  
var monobjetjson = sessionStorage.getItem("objet");  
// Réformer l'objet à partir de la chaîne de caractère  
var monobjet = JSON.parse(monobjetjson);  
// Affichage dans la console  
console.log(monobjet);
```



# JSON pour Le web Storage

**Exercice :** Proposez une solution pour stocker en session les informations introduites par l'utilisateur

## Web Storage

### Démonstration

Cette page utilise JSON en complément de Web Storage.

Nom

Prénom

Ville

Usage :

1. Remplir le formulaire avec quelques informations
2. Cliquer sur Mémoriser
3. Effacer tout
3. Cliquer sur Récupérer pour obtenir les informations à partir de Web Storage

## 4. Web Sockets



# Le Web Sockets

- ▶ C'est un nouveau protocole ambitieux de communication avec le serveur.
- ▶ Un navigateur ne peut habituellement qu'effectuer des requêtes au serveur puis recevoir sa réponse ⇒ C'est une communication **unidirectionnelle (canal simplex)**.
- ▶ Les Web Sockets apportent la communication **bi-directionnelle (full-duplex)** entre le client et le serveur.

## Question ?

Demandez-vous comment vous réaliseriez une page dont le contenu devrait toujours être à jour (comme un chat par exemple) ?



# Le Web Sockets

Euh... On vérifierait s'il y a eu des changements sur le serveur toutes les n secondes en AJAX ?



Oui, mais ...

Le client travaille, le serveur travaille, et la plupart du temps il n'y a rien à mettre à jour ⇒ **Pas optimal !**

**La solution :**

Le serveur va pouvoir nous envoyer les changements dès qu'ils se produisent, et le tout est effectué dans une requête spéciale dépourvue d'en-têtes HTTP ! ⇒ Les données téléchargées sont **ultra légères et ne contiennent que l'information qui nous intéresse**



# Références Bibliographiques



# Références Bibliographiques

- ① HTML5 et CSS3 Cours et exercices corrigés, Jean Engels, Edition Eyrolles, 2012
- ② CSS avancées : Vers HTML5 et CSS3, Raphaël Goetter, 2<sup>me</sup> Edition, Eyrolles, 2012
- ③ Bootstrap 3 pour l'intégrateur web - CSS et Responsive Web Design, Christophe Aubry, Edition Eni, Avril 2014, 370 pages
- ④ HTML5 pour les web designers, Rachel Andrew et Jeremy Keith, 2<sup>me</sup> Edition, Eyrolles, Juillet 2016, 104 pages