

M4103C

Programmation Web – Client riche

Support de cours – Partie 2

DUT Informatique – 2nd année

Matthieu Le Lain, Michel Adam, Nicolas Le Sommer, Goulven Kerbellec
IUT de Vannes

Plan global

- 1. Rappels sur les langages de programmation Web
- 2. AJAX & JQuery
- 3. AngularJS
- 4. API HTML5
- 5. Cordova



Plan – Chapitre 3 & 4

- 3 - AngularJS
 - 1. Introduction
 - 2. Binding
 - 3. Watch
 - 4. Directives
 - 5. Filtre
 - 6. Service
- 4 - API HTML5
 - 1. Géolocalisation
 - 2. Mise en cache
 - 3. LocalStorage

3 – AngularJS Introduction



AngularJS - Introduction

3.1

AngularJS est un Framework JavaScript libre, open source développé par Google.



Ses principes sont :

- Découpler les manipulations du DOM et la logique métier
- Découpler le côté serveur et client d'une application.
- Considérer que le test d'une application est aussi important que l'écriture de l'application elle-même.
- Favoriser le couplage faible entre la présentation, la logique métier et les données (MVC / MVVM)
- Version actuelle : 1.5

Tutoriel AngularJS : Découverte du Framework, création d'une Todo

<https://www.youtube.com/watch?v=E1NIJjTYq6U>

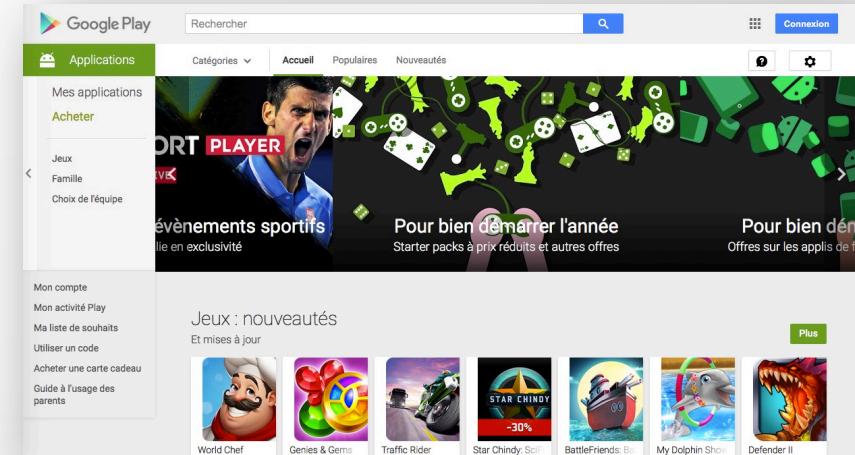
AngularJS – Pourquoi ?

3.1

Ce genre de Framework permet notamment de mettre en œuvre des SPA (Single Page Application). Vous utilisez ce concept d'application au quotidien lors de vos visites sur le Play Store, l'Apple Store, Gmail, etc....

À la différence d'un site web classique composé d'un ensemble de page sur lesquelles l'utilisateur peut naviguer, avec une logique applicative centralisé sur le serveur;
une SPA n'est composé que d'une unique page !

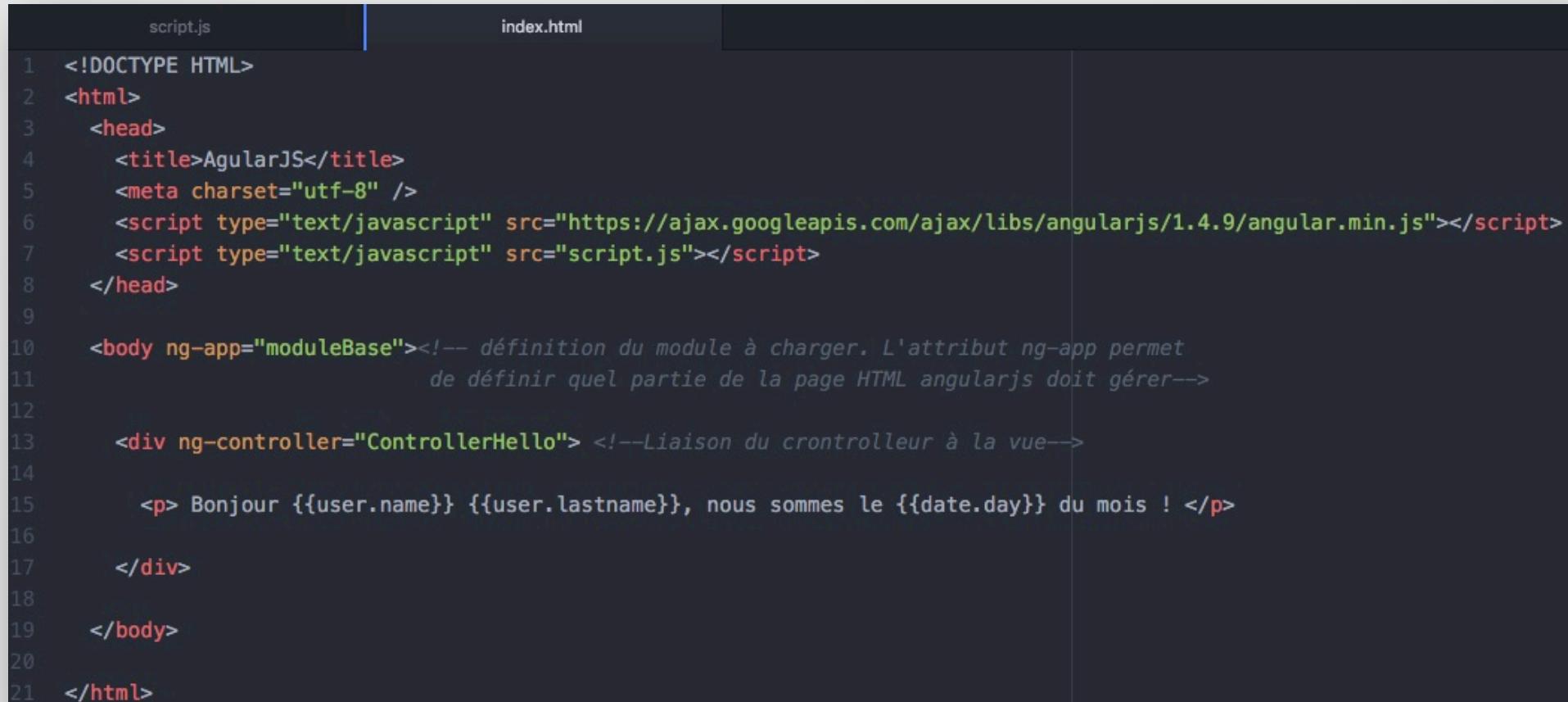
L'utilisateur navigue sur différentes vues, mais sur la même page.
La logique applicative est ainsi déportée sur le client.



AngularJS - Base

3.1

Comme tout script JavaScript, il suffit de le référencer sur la page HTML (local ou CDN) pour l'utiliser et définir sa portée sur via les balise du DOM. **La page HTML est la vue.**



The screenshot shows a code editor with two tabs: "script.js" and "index.html". The "index.html" tab is active, displaying the following code:

```
1  <!DOCTYPE HTML>
2  <html>
3      <head>
4          <title>AngularJS</title>
5          <meta charset="utf-8" />
6          <script type="text/javascript" src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.9/angular.min.js"></script>
7          <script type="text/javascript" src="script.js"></script>
8      </head>
9
10     <body ng-app="moduleBase"><!-- définition du module à charger. L'attribut ng-app permet
11                     de définir quel partie de la page HTML angularjs doit gérer-->
12
13     <div ng-controller="ControllerHello"> <!--Liaison du contrôleur à la vue-->
14
15         <p> Bonjour {{user.name}} {{user.lastname}}, nous sommes le {{date.day}} du mois ! </p>
16
17     </div>
18
19 </body>
20
21 </html>
```

AngularJS - Base

3.1

```
script.js           index.html

1
2
3
4 /**
5 * La fonction déclarée ci-dessous utilise le $scope.
6 * Ce mécanisme permet ainsi de présenter les données aux vues.
7 * Ainsi chaque contrôleur prend en paramètre un $scope, ce qui
8 * permet de rendre accessibles aux vues les fonctionnalités ou les
9 * propriétés. Les scopes possèdent une relation d'héritage, ainsi
10 * tous les scopes de l'application ont accès au $rootscope
11 */
12 function sayHello($scope) {
13   var ephemericid = new Date();
14
15   $scope.user = { name : "Jessica", lastname : "Fletcher" };
16   $scope.date = { day : ephemericid.getDate() };
17 }
18
19
20 /**
21 * Le code ci-dessous permet de créer un module, en l'occurrence
22 * ici le module moduleBase.
23 * -> Au sein de ce module, nous déclarons un
24 *     contrôleur appelé ControllerHello qui appelle la fonction sayHello
25 */
26 angular.module("moduleBase", [])
27   .controller("ControllerHello", sayHello);
```

Le modèle

se représente par un objet ou une valeur.

Le contrôleur est une classe JS qui prend l'Object \$scope en paramètre. Toutes les propriétés ou fonction qui seront ajoutées par le contrôleur à l'objet \$scope seront accessible par la vue.

Chaque contrôleur est ajouté au **module qui représente l'application**.

AngularJS - Binding

3.2

Le binding est un mécanisme qui permet de synchroniser les données du modèle et l'affichage de ces dernières dans les vues. AngularJS gère trois type de binding :

- **one-way** : Mise à jour des vues lors d'une modification du modèle.

```
<p> {{ mavariable }} </p>
<p ng-bind="mavariable"> </p>
```

- **two-way** : Mise à jour des vues lorsque le modèle change. Et inversement, mise à jour du modèle lorsque les vues changent.

*Au sein d'un formulaire <input type="text" ng-model="username" />
L'input est binder à la propriété username du modèle et le contrôleur pour avoir accès au propriétés via \$scope.username*

- **one-time** : Affichage d'une donnée du modèle dans la vue, puis désactivation du lien de binding.

```
<p>{{:value}} </p>
```

Conditions

- **ng-hide="solde > 0"** : cacher/afficher un élément en fonction d'une valeur
- `<div ng-if="connected"> Bienvenue {{username}} </div>`
- **ng-switch** avec ses sous éléments ng-switch-when

Liens

- `<a ng-href="/detail/{{id}}> Detail `
- ``

Listes

```
<li ng-repeat="item in items> {{item.name}}</li>
```

Evènements

ng-click, ng-change, ng-submit..

*Documentation
en ligne:
<https://code.angularjs.org/1.4.9/docs/api>*

AngularJS – Watch

3.3

Les contrôleurs sont notifiés dès lors qu'une action a été effectué depuis la vue, uniquement. Ainsi, si une modification survient au niveau du modèle ou d'une autre source le ou les contrôleurs ne seront pas notifiés.

Pour pallier à cela, AngularJS utilise le mécanisme de watch qui permet de surveiller tout changement d'une des propriétés du modèle et d'être notifié à chaque modification. Son utilisation passe par la méthode \$watch de l'objet \$scope.

Ce type de mécanisme permet par exemple d'appliquer une réduction du montant total dès lors que la personne saisie son numéro de client, qu'une commande atteint 100€, qu'il y a plus de 5 articles dans le panier, etc..

AngularJS – Watch

3.3



```
4 function sayHelloController($scope){  
5     $scope.username = null;  
6     $scope.log = null;  
7  
8     /**  
9      Le changement du modèle, ici la valeur de username,  
10     déclenchera les actions en conséquences grâce à  
11     l'observation de l'élément.  
12     **/  
13     $scope.$watch("username", function(newValue, oldValue) {  
14         $scope.log = "Nouvelle valeur : " + newValue + " | Ancienne valeur : " + oldValue;  
15     });  
16 }
```

AngularJS – Directive

3.4

Pour **rajouter de la dynamicité** aux pages HTML, AngularJS utilise des directives. Vous les avez déjà rencontrés précédemment, il s'agit des balises ou attributs « ng » déclarés sur la page HTML.

```
17      <div ng-controller="ControllerHello">
18          <div class="actu" ng-repeat="bloc in blocs">
```

Chaque directive a son fonctionnement. Par exemple, la balise ng-controller attache un contrôleur à la vue et la balise ng-repeat répète un template en bouclant sur chacun des éléments de la collection « blocs ». Vous retrouverez la liste des directives existantes et leur fonctionnement sur la documentation officielle.

Documentation en ligne:

<https://docs.angularjs.org/guide/directive>

AngularJS – Filtre

3.5

Tout comme on souhaite typer les données d'entrées dans le développement de nos applications Android pour afficher le clavier adapté, nous avons besoin parfois de formater nos données à afficher. Pour cela AngularJS utilise des filtre. **Ce formatage se fait essentiellement dans la vue.**

```
 {{ expression | leFiltre }}
```



```
36  <h2>Solde de votre compte</h2>
37  <p>{{ solde | currency }}</p>
```

Ici, la valeur de notre variable solde sera affiché au format monétaire avec deux chiffres après la virgule et le symbole €.

AngularJS – Filtre

3.5

Filtre	Effet
currency	Monétaire
date	Date
number	Nombre
json	Objet JS sous forme de chaîne de caractère JSON
Lowercase (upper)	Chaîne de caractère en minuscule
<i>filter</i>	<i>Éléments correspondant à un critère</i>
<i>limitTo</i>	<i>Récupération de X éléments</i>
<i>orderBy</i>	<i>Ordonner la liste selon un critère</i>



Certains filtres sont utilisés pour manipuler les listes. On y associera également la directive « ng-repeat ».

AngularJS – Services

3.6

Les Services sont des objets JS uniques (singletons) dont l'objectif est de factoriser du code en exposant des méthodes et propriétés qui peuvent être injectées dans un contrôleur, une directive... grâce au **mécanisme d'injection de dépendance**. On identifie facilement les service, ils commencent par \$.

```
6  function getWeather($scope, $http) {  
7      $http.get("http://api.openweathermap.org/data/2.5/forecast?q=vannes&mode=json&APPID=idkeyhere")  
8          .success(function(data){  
9              alert("Ville : "+ data.city.name + ", Long : "+ data.city.coord.lon + ", Lat : "+data.city.coord.lat);  
10         });  
11     }  
12  
13     angular.module("moduleBase", [])  
14         .controller("ControllerAjax", getWeather);
```

Vous avez déjà rencontré l'objet \$http dans le tutoriel visionné précédemment. L'objet \$http est chargé d'effectuer des requêtes. Par exemple, \$location permet d'interagir avec l'URL du navigateur.

4 – Utilisation de l'API HTML5

HTML



Web API

L'utilisation des langages récents (HTML5, JS, ...), permet d'utiliser les Web API disponibles, telles que la géolocalisation, le web storage, drag'n drop etc..

De ce fait, cela permettra aussi aux applications d'interagir ou de consulter les différents capteurs des plateformes mobiles, et de consulter les gestionnaires de données.

Communication

Bluetooth

Wifi

SMS

Téléphonie

Données & autres

Contacts

Stockage data

Planification de notif.

Web Payement

Matériel

Caméra

Batterie

Géolocalisation

Vibration

API de Géolocalisation – JS Natif

4.1.1

Pour votre application météo, ils seraient appréciable de récupérer la géolocalisation automatiquement grâce à l'API Web. Une fois récupérée nous pourrions également l'afficher dans une notification.

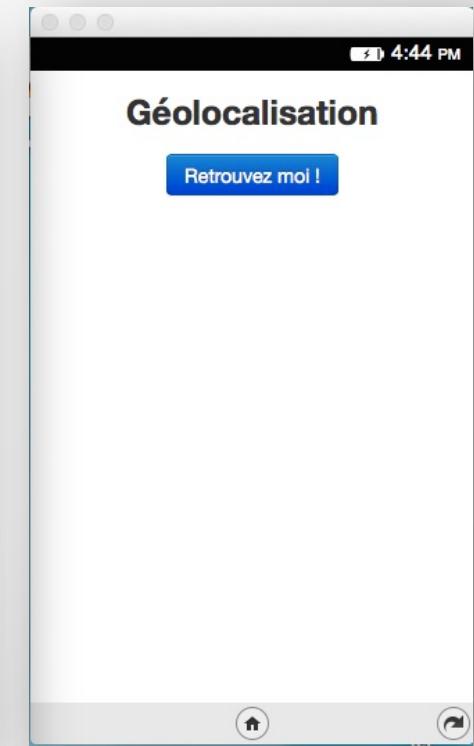
```
<!DOCTYPE html>
<html>
<head>
    <meta charset="UTF-8">
    <title>Is it rain today ?</title>
    <link rel="icon" type="image/png" href="images/icon_app-60.png">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <link href="styles/style.css" type="text/css" rel="stylesheet">
    <link href="styles/bootstrap.css" type="text/css" rel="stylesheet">
</head>

<body>
    <div class="container-fluid text-center">

        <h3>Géolocalisation</h3>
        <div class="col-xs-12" >
            <!-- Un simple bouton appellant notre fonction getLocation()-->
            <p><button onclick="getLocation()" class="btn btn-primary">Retrouvez moi !</button></p>
            <div id="result"></div>
        </div>
    </div>

</body>
<script src="scripts/geoloc.js"></script>
<script src="scripts/bootstrap.js"></script>
</html>
```

index.html



API de Géolocalisation – JS Natif

4.1.1

```
//Fonction de géolocalisation
function getLocation() {

    var result = document.getElementById("result");

    //Vérification du support de la géolocalisation par le navigateur
    if (!navigator.geolocation){
        result.innerHTML = "<p>La géolocalisation n'est pas supportée par votre navigateur.</p>";
        return;
    }

    //Récupération de la position
    navigator.geolocation.getCurrentPosition(success, error);

    //En attendant le retour de la méthode, on affiche un message d'attente
    result.innerHTML = "<p>Localisation en cours..</p>";

    //Si la géolocalisation est correctement effectué
    function success(position) {
        var latitude = position.coords.latitude;
        var longitude = position.coords.longitude;

        result.innerHTML = '<p>Latitude : ' + latitude + '° <br>Longitude : ' + longitude + '°</p>';

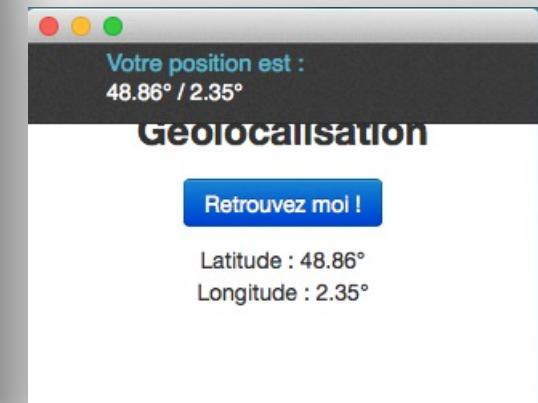
        //On créer une notification pour l'utilisateur
        createNotification(latitude+"° / "+longitude+"°");
    };

    //Si la géolocalisation ne s'est pas correctement effectuée
    function error() {
        result.innerHTML = "Impossible de récupérer votre position";
    };
}
```

API de Géolocalisation – JS Natif

4.1.1

```
function createNotification(message) {  
  
    // On vérifie que le navigateur supporte les notifications  
    if (!("Notification" in window)) {  
        console.log("Ce navigateur ne supporte pas les notifications.");  
    }  
  
    else if (Notification.permission === "granted"){// refusé = denied  
        var notification = new Notification('Votre position est : ', { body: message });  
        // vibration 500ms  
        window.navigator.vibrate(500);  
    }  
  
    else{  
        alert('Les notifications sont désactivées');  
    }  
}
```



On pourra également, au sein de notre application, demander à l'utilisateur la permission d'utiliser les notifications si elles sont désactivées en appliquant `requestPermission()` sur l'objet `Notification`. Il ne reste donc plus qu'à intégrer cette fonctionnalité dans votre application météo.

Documentation Web API : <https://developer.mozilla.org/fr/docs/Web/API>

API de Géolocalisation – AngularJS

4.1.2

Reprendons notre feuille HTML de base pour y afficher la position géographique. L'objet position que l'on va récupérer contiendra comme précédemment les coordonnées en longitude et en latitude.

```
index.html • script.js

1 <html>
2
3   <head>
4     <title>AngularJS – Geoloc</title>
5     <meta charset="utf-8" />
6     <script type="text/javascript" src="https://ajax.googleapis.com/ajax/libs/angularjs/1.4.9/angular.min.js"></script>
7     <script type="text/javascript" src="script.js"></script>
8   </head>
9
10  <body ng-app="moduleBase">
11
12    <div ng-controller="geoloc">
13      <p>Longitude {{position.coords.longitude}}, Latitude : {{position.coords.latitude}}</p>
14    </div>
15
16  </body>
17
18 </html>
```

API de Géolocalisation – AngularJS

4.1.2



```
index.html • script.js

1
2
3  function getPosition($scope){
4      if (navigator.geolocation) {
5          navigator.geolocation.getCurrentPosition(function(position){
6              $scope.$apply(function(){
7                  $scope.position = position;
8              });
9          });
10     }
11     else{
12         alert("geolocalisation non active");
13     }
14 }
15
16
17
18 angular.module("moduleBase", [])
19     .controller("geoloc", getPosition);
```

Application hors connexion

4.2

Avec HTML5 il est possible de rendre accessible son application hors connexion. Il faut pour cela utiliser deux mécanismes : le cache et le storage.

Par ailleurs, pour le storage, HTML5 permet d'enregistrer les données de trois manières différentes WebSQL, WebStorage que nous utiliserons et IndexedDB.

Ces mécanismes nous permettront ensuite de packager notre application avec Cordova pour la destiné aux plateformes mobiles.

Hors connexion - Mise en cache

4.2.1

La mise en cache de ressources, qu'ils s'agissent de pages HTML, CSS, JS, etc..) se réalise par la création d'un fichier de manifeste qui décrit les règles et les ressources à mettre en cache.

Fallback précise quel fichiers sont à charger en cas d'erreur et *Network* à récupérer nécessairement sur le réseaux.

```
script.js           index.html          cache.manifest
1 CACHE MANIFEST
2 #v1 26/01/2016
3
4 CACHE:
5 index.html
6 script.js
7
8 Fallback:
9 index.html script.js
10
11 NETWORK:
12 https://ajax.googleapis.com/ajax/libs/angularjs/1.4.9/angular.min.js
```

```
script.js           index.html          cache.manifest
1 <!DOCTYPE HTML>
2 <html manifest="cache.manifest">
3 <!-- Si la balise HTML contient "manifest", le navigateur
4 va récupérer le fichier de manifeste et effectuer des requêtes
5 pour chacune des ressources à mettre en cache. -->
6 <head>
```

Hors connexion – Web Storage

4.2.2

Le storage, comme son nom l'indique permet de sauvegarder de manière persistante et localement des données à afficher sur le navigateur lorsque le réseau n'est pas disponible et de les synchroniser lorsqu'il le sera de nouveau. Les données sont placées par domaine.

Le web storage se fait sous forme de clé/valeur. Il est utilisé pour enregistrer des données légères comme des préférences utilisateurs, des configurations ou des listes d'éléments. Selon les navigateurs 5 à 10 Mo sont disponibles sans validation utilisateur.

La gestion du stockage se fait avec l'objet JavaScript *localStorage*.

Hors connexion – Web Storage

4.2.2

```
32 /**
33 * Enregistrement des préférences de l'utilisateur
34 */
35 var userPreferences = {
36   theme: 'black',
37   language: 'fr-FR'
38 }
39
40 var serializedPreferences = JSON.stringify(userPreferences);
41 localStorage.setItem("userPreferences", serializedPreferences);
42
43
44 /**
45 * Récupération des préférences de l'utilisateur
46 */
47 var serializedPreferences = localStorage.getItem("userPreferences");
48 var userPreferences = JSON.parse(serializedPreferences);
```

Outils, références

- **W3C:** <http://www.w3.org/standards/>
- **Alsacreations :** <http://www.alsacreations.com/>
- **jQuery:** <https://jquery.com/>
- **API jQuery :** <https://api.jquery.com>
- **Angular JS:** <https://angularjs.org/>

Bibliographie

Une liste, non exhaustive, d'ouvrage vous est présentée ici. Il est fortement conseillé de se tenir à jour des évolutions des divers langages présentés dans ce cours. Soit par l'intermédiaire d'ouvrages comme ci-après et/ou grâce à la documentation officielle en ligne, mentionné sur le diapositive (outils & références).

- **HTML5 et CSS3 – Maîtrisez les standards des applications web.** Luc Van Lancker– Ed. Eni. ISBN-13 : 978-2746079700
- **jQuery – Simplifiez et enrichissez vos développements JavaScript.** J. Chaffer et K. Swedberg– Ed. Pearson. ISBN : 978-2-7440-2381-1
- **AngularJS – Développez aujourd’hui les applications web de demain.** S. Ollivier et PA Gury – Editions ENI. ISBN : 978-2-7460-9334-8