

Développement d'applications pour smartphones Firefox OS

version 271013

Brouillon

Ce document est publié sous licence Creative Commons

Vous êtes libres :

- de reproduire, distribuer et communiquer cette création au public
- de modifier cette création

Selon les conditions suivantes :



– **Paternité.** Vous devez citer le nom de l'auteur original de la manière indiquée par l'auteur de l'œuvre ou le titulaire des droits qui vous confère cette autorisation (mais pas d'une manière qui suggérerait qu'ils vous soutiennent ou approuvent votre utilisation de l'œuvre).



– **Pas d'Utilisation Commerciale.** Vous n'avez pas le droit d'utiliser cette création à des fins commerciales.

- À chaque réutilisation ou distribution de cette création, vous devez faire apparaître clairement au public les conditions contractuelles de sa mise à disposition.
- Chacune de ces conditions peut être levée si vous obtenez l'autorisation du titulaire des droits sur cette œuvre.
- Rien dans ce contrat ne diminue ou ne restreint le droit moral de l'auteur ou des auteurs.

Notes de l'auteur

Ce document est à l'origine destiné aux élèves de seconde, du lycée G Fichet de Bonneville (Haute-Savoie), qui ont choisi de suivre l'enseignement d'exploration PSN (Pratique Scientifique et Numérique). Il est aussi utilisable en spécialité ISN.

Dans tous les cas, des connaissances en HTML, en JavaScript et en CSS sont indispensables avant d'aborder la lecture de ce document. Pour les débutants en programmation, 19 activités, consacrées au trio JavaScript, HTML et CSS, sont téléchargeables : http://www.webisn.byethost5.com/apprendre_prog_avec_JS.pdf

Ce document n'a pas la prétention d'être exhaustif, il est uniquement là pour donner des bases aux élèves. Charge à eux, une fois les bases acquises, d'explorer les nombreuses ressources disponibles sur internet qui leur permettront d'enrichir leurs applications.

David Roche

D'après Wikipédia :

Firefox OS (précédemment connu sous le nom de Boot to Gecko ou B2G) est un système d'exploitation mobile libre proposé et développé par la Mozilla Corporation.

Il n'est pas fondé sur Android mais conçu pour s'adapter aux smartphones existants utilisant ce système (noyau Linux, libusb (en)...). Il utilise également le moteur de rendu Gecko pour faire fonctionner des applications web développées en format HTML5 : « notre ambition n'est pas d'imposer un énième système fermé, mais au contraire d'introduire plus d'ouverture, en apportant enfin tout le web sur les mobiles. »

Deux choses sont à retenir de l'extrait de l'article de Wikipédia :

- Firefox OS est un système d'exploitation mobile libre développé par Mozilla
- Firefox OS fait fonctionner des applications web développées en HTML5

À faire vous même

Faites des recherches sur la fondation Mozilla

Qu'entend-on par «libre» dans «un système d'exploitation mobile libre» ?

Pas de téléphone pour l'instant, mais un émulateur

À la date où je rédige ces lignes (octobre 2013), aucun téléphone fonctionnant sous Firefox n'est disponible à la vente en France (mais, au vu du succès de ces téléphones à l'étranger, il paraît évident que cela ne devrait pas tarder). Heureusement la fondation Mozilla propose des outils (dont un émulateur) très facilement installables puisqu'ils fonctionnent dans le navigateur Firefox (sous forme d'extension). Nous aurons l'occasion de revenir sur cet émulateur un peu plus loin dans ce document.

Applications web au format HTML5 ?

Les applications disponibles sous Firefox OS sont donc des « applications web au format HTML5 », mais qu'est-ce que cela signifie ?

Une application web (aussi appelée web app) est avant tout un site internet : cela implique donc l'utilisation du HTML, du JavaScript et du CSS.

Avec l'arrivée du nouveau standard HTML5 (et des API JavaScript qui lui sont associées), certains sites internet ressemblent à des applications «classiques» (applications qui s'installent sur l'ordinateur avec par exemple un fichier .exe sous Windows), sauf erreur de ma part une des premières applications web a été Gmail. Toujours chez Google, on peut aussi citer «Drive» avec sa suite bureautique qui, aujourd'hui, n'a presque plus rien à envier aux suites bureautiques classiques (au moins pour les utilisations basiques). Et pourtant, «Drive» est un site internet.

Première étape, créer des applications web

Avant de nous intéresser à la création d'applications pour Firefox OS, nous allons donc devoir nous intéresser à la création de web app, même si un document a déjà été consacré à ce sujet (voir <http://www.webisn.byethost5.com/jquery>). Je vais d'ailleurs reprendre ici des portions entières de ce document.

Utiliser les bibliothèques jQuery et jQuery mobile

Afin de nous faciliter la tâche, nous allons utiliser deux bibliothèques : jQuery et jQuery mobile.

Commençons par jQuery.

jQuery ? Qu'est-ce que c'est ?

jQuery est une bibliothèque JavaScript open source (rechercher la définition d'open source) développée par John Resig depuis 2006.

Cette bibliothèque facilite grandement le développement d'applications web (ou de pages web).

jQuery va nous permettre de :

- modifier très facilement l'aspect de nos pages (jouer sur les CSS)
- modifier le contenu de votre page (jouer sur le HTML)
- gérer les événements
- créer des animations sur votre page
- communiquer facilement avec un serveur

Nous allons, dans ce document, nous concentrer uniquement sur 2 aspects : « modifier le contenu de votre page » (dans la 1re partie) et « communiquer facilement avec un serveur » (dans la 2e partie). Il existe, sur internet, énormément de

ressources qui vous permettront d'approfondir tous les autres aspects (voir l'annexe pour les liens).

Notez bien que jQuery n'est pas un "logiciel", vous allez donc produire du code (HTML, JavaScript et CSS).

Première approche

jQuery permet de manipuler simplement le DOM (rechercher la signification de DOM), nous allons pouvoir modifier le code HTML d'une page, à la volée, en utilisant du JavaScript.

Un premier exemple

À faire vous même

code HTML fos01.html

```
<!doctype html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <title>Test jQuery</title>
    <script src="lib/jquery-2.0.3.min.js"></script>
    <script src="javascript/fos01.js"></script>
</head>
<body>
</body>
</html>
```

code JavaScript fos01.js

```
jQuery(function(){
    jQuery('body').append('<h1>hello world!</h1>');
});
```

Quelques remarques sur le contenu de la balise head du fichier HTML :

- Avant de commencer, vous devez télécharger la bibliothèque sur le site officiel (<http://jquery.com/download/>), Enregistrer le fichier «jquery-2.0.3.min.js» en faisant un « clic droit » suivi d'un « enregistrer sous » sur «Download the compressed, production jQuery 2.0.3» (attention le numéro de version sera sans doute différent quand vous lirez ces lignes)
- La ligne «<script src="lib/jquery-2.0.3.min.js"></script> » va vous permettre d'utiliser jQuery. À vous de l'adapter en fonction du dossier qui accueille le fichier « jquery-2.0.3.min.js ».
- La ligne suivante («<script src="javascript/fos01.js"></script>») va nous permettre d'accéder à notre fichier JavaScript «fos01.js», ici aussi, attention au chemin choisi.

Vous avez peut-être déjà constaté l'absence de contenu entre les balises «<body> » et «</body> » : notre page ne devrait donc rien afficher. À part cela, rien à signaler pour le fichier « fos01.html ».

Le fichier JavaScript «fos01.js» est un peu moins classique : listener

«jQuery(function(){ ... }); » : nos fichiers JavaScript (utilisant jQuery) commenceront systématiquement comme cela. Tout notre code devra se trouver à la place des « . . . », cette structure nous permet d'avoir l'assurance que le navigateur « exécute » le code HTML avant « d'exécuter » le JavaScript, ce qui, dans certains cas, est indispensable.

«jQuery('body') » : nous créons un objet jQuery à l'aide de la méthode jQuery(). Cette méthode a pour paramètre un sélecteur CSS.

Petite parenthèse : qu'est-ce qu'un sélecteur CSS ?

Considérons le fichier CSS suivant (si nécessaire revoir l'activité sur le CSS) :

```
body
{
    background-color : black ;
}
p
{
    margin : 10px ;
}
```

```

}

#maBalise
{
    background-color : green ;
}

.autresBalises
{
    font-size : 15px ;
}

```

body, p, #maBalise, et .autreBalise sont des sélecteurs CSS. Un sélecteur CSS vous permet d'identifier des balises HTML (body : la balise body, p : tous les paragraphes de la page, #maBalise : la balise ayant pour id maBalise, .AutresBalises : les balises ayant pour class AutresBalises).

«jQuery('body')» : nous permet de créer un objet jQuery qui sera « lié » à la balise body de notre code HTML.

«jQuery('body').append('<h1>hello world!</h1>')» : nous exécutons la méthode append de l'objet jQuery que nous venons de créer. La méthode «append» permet d'« ajouter » le code HTML («<h1>hello world!</h1>») à l'intérieur de la balise body.

Avant l'exécution de «jQuery('body').append('<h1>hello world!</h1>')» :

```

<body>
</body>

```

Après l'exécution de «jQuery('body').append('<h1>hello world!</h1>')» :

```

<body>
  <h1>hello world!</h1>
</body>

```

Attention : le fichier fos01.html n'est pas modifié, seul le DOM l'est (code HTML « chargé » dans le navigateur)

En vous aidant des explications qui viennent d'être fournies, décrire le résultat attendu si vous ouvrez le fichier fos01.html avec un navigateur internet (Firefox, Chrome....)

.....

.....

.....

Vérifiez votre hypothèse

ATTENTION : À partir de maintenant nous n'écrirons plus « jQuery », mais son alias (son raccourci) : le signe \$. Le code va donc devenir :

code JavaScript fos01.js

```

$(function(){
    $('body').append('<h1>hello world!</h1>');
});

```

À faire vous même

code HTML fos02.html

```

<!doctype html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <title>Test jQuery</title>
    <script src="lib/jquery-2.0.3.min.js"></script>
    <script src="javascript/fos02.js"></script>
</head>
<body>
<h1>Fiche</h1>
<div id='fiche'></div>
</body>
</html>

```

code JavaScript fos02.js

```
$(function(){
maFiche={
    id : "7845",
    nom : "Durand",
    prenom : "Jean-Pierre",
    dateNaissance : "17/08/1967"
}
$('#fiche').append('<p>id : '+maFiche.id+'</p>');
$('#fiche').append('<p>nom : '+maFiche.nom+'</p>');
$('#fiche').append('<p>prénom : '+maFiche.prenom+'</p>');
$('#fiche').append('<p>date de naissance : '+maFiche.dateNaissance+'</p>');
});
```

À chaque exécution d'append, le code HTML est ajouté à la suite du code déjà présent :
avant un `$(div).append('<p>titi</p>')` :

```
<div>
    <h1>toto</h1>
</div>
```

après un `$(div).append('<p>titi</p>')` :

```
<div>
    <h1>toto</h1>
    <p>titi</p>
</div>
```

La méthode « prepend » permet d'ajouter du code HTML en « début de balise ».
avant un `$(div).prepend('<p>titi</p>')` :

```
<div>
    <h1>toto</h1>
</div>
```

après un `$(div).prepend('<p>titi</p>')` :

```
<div>
    <p>titi</p>
    <h1>toto</h1>
</div>
```

Il est possible d'utiliser la concaténation : `$('#fiche').append('<p>id : '+maFiche.id+'</p>');` »,
maFiche étant un objet JavaScript (à quoi correspond maFiche.id ?).

Décrire le résultat attendu en cas « d'ouverture » du fichier fos02.html avec un navigateur internet (Firefox, Chrome....)

.....
.....
.....

Vérifiez votre hypothèse

Utilisation de la boucle for avec la méthode append

Imaginez que vous voulez afficher une liste de fruits à l'aide des balises HTML `` et `` (si vous ne connaissez pas ces balises recherchez sur internet des informations à leur sujet).

Nous aurions ce genre de code :

```
<ul>
    <li>banane</li>
    <li>pomme</li>
    <li>ananas</li>
    <li>pêche</li>
    <li>fraise</li>
</ul>
```

Cela peut vite devenir rébarbatif....

La combinaison d'un tableau, d'une méthode append et d'une boucle for va vous permettre d'automatiser la constitution de listes.

À faire vous même

code HTML fos03.html

```
<!doctype html>
<html lang="fr">
<head>
  <meta charset="UTF-8">
  <title>Test jQuery</title>
  <script src="lib/jquery-2.0.3.min.js"></script>
  <script src="javascript/fos03.js"></script>
</head>
<body>
  <h1>Voici ma liste de fruits</h1>
  <ul>
  </ul>
</body>
</html>
```

code JavaScript fos03.js

```
$(function(){
var mesFruitsTab=["banane","pomme", "ananas","pêche","fraise"];
for (var i=0;i<mesFruitsTab.length;i++){
  $('ul').append('<li>'+mesFruitsTab[i]+'</li>')
}
});
```

Décrire le résultat attendu en cas « d'ouverture » du fichier fos03.html avec un navigateur internet (Firefox, Chrome....)

.....
.....
.....

Vérifiez votre hypothèse

À faire vous même : miniprojet 1

Il est possible d'avoir un tableau contenant des objets :

```
monTableau=[
  {id : "7845",
  nom : "Durand",
  prenom : "Jean-Pierre",
  dateNaissance : "17/08/1967"},
  {id : "6578",
  nom : "Dupond",
  prenom : "Gérard",
  dateNaissance : "23/04/1984"},
  {id : "9876",
  nom : "Robert",
  prenom : "Gabriel",
  dateNaissance : "21/02/1991"}
]
```

Reprenez l'exemple traité dans fos02 (affichage d'une fiche de renseignements personnels). Sauf que cette fois-ci, vous devrez produire du code permettant d'afficher les 3 fiches de renseignements les unes sous les autres :

Fiche d'identité n°7845

Nom : Durand

Prénom : Jean-Pierre

Date de Naissance : 17/08/1967

Fiche d'identité n°6578

Nom : Dupond

Prénom : Gérard

Date de Naissance : 23/04/1984

Fiche d'identité n°9876

Nom : Robert

Prénom : Gabriel

Date de Naissance : 21/02/1991

Un peu d'interaction : l'événement 'click'

jQuery gère simplement les interactions avec l'utilisateur, nous parlerons d'événements.

Voici la structure de base de la gestion d'un événement :

```
$('#mon_sélecteur').on('type événement', fonction de callback)
```

Un exemple :

```
$('#mon_id').on('click', function(){alert('Hello World !');})
```

En cas de clic de souris sur l'élément ayant pour id «mon_id», la fonction «function(){alert('Hello World !');}» sera exécutée.

La notion de fonction de callback est essentielle : une fonction de callback sera exécutée seulement après un « événement » donné (ici, un clic de souris sur l'élément ayant pour id «mon_id»), pas avant.

Notre fonction de callback ne porte pas de nom (nous n'avons pas « toto=function(){...} » ou « function toto(){...} »), on parle de fonction anonyme. Le nom d'une fonction est important quand le programmeur désire appeler (exécuter) cette fonction un peu plus loin dans le code, avec par exemple un « toto() ». Ici, ce n'est pas le programmeur qui va « demander » l'exécution de la fonction, mais un événement : donner un nom à notre fonction est inutile

N. B. Il existe énormément d'événements, voici une liste (non exhaustive) :

dblclick => double-clic de la souris

hover => passage de la souris

mouseenter => rentrer dans un élément

mouseleave => quitter un élément

keydown => touche de clavier enfoncé

keyup => relâche une touche préalablement enfoncée

À faire vous même

code HTML fos04.html

```
<!doctype html>
<html lang="fr">
<head>
  <meta charset="UTF-8">
  <title>jQuery</title>
  <script src="lib/jquery-2.0.3.min.js"></script>
  <script src="javascript/fos04.js"></script>
</head>
<body>
  <button id="afficheListe">Afficher la liste</button>
```

```

        <ul>
        </ul>
</body>
</html>

```

code JavaScript fos04.js

```

$(function(){
var mesFruitsTab=["banane","pomme", "ananas","pêche","fraise"];
var firstClick=0;
$('#afficheListe').on('click',function(){
    if (firstClick==0){
        for(var i=0;i<mesFruitsTab.length;i++){
            $('ul').append('<li>'+mesFruitsTab[i]+'</li>');
        }
        firstClick=1;
    }
});
});

```

Décrire le résultat attendu en cas « d'ouverture » du fichier fos04.html avec un navigateur internet (Firefox, Chrome...). Quel est le rôle de la variable firstClick ?

.....

.....

.....

Vérifiez votre hypothèse

hide et show

La méthode «hide» permet de rendre invisible un élément, la méthode «show» permet de rendre visible un élément préalablement invisible.

```
$('#mon_id').hide()
```

rendra invisible l'élément ayant pour id «#mon_id».

À faire vous même

code HTML fos05.html

```

<!doctype html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <title>jQuery</title>
    <script src="lib/jquery-2.0.3.min.js"></script>
    <script src="javascript/fos05.js"></script>
</head>
<body>
    <button id="affiche">Cliquez ici</button>
    <ul>
    </ul>
</body>
</html>

```

code JavaScript fos05.js

```

$(function(){
var mesFruitsTab=["banane","pomme", "ananas","pêche","fraise"];
$('ul').hide();
var isVisible=false;
for (var i=0;i<mesFruitsTab.length;i++){
    $('ul').append('<li>'+mesFruitsTab[i]+'</li>');
}
$('#affiche').on('click',function(){
    if (isVisible==false){
        isVisible=true;
        $('ul').show();
    }
}

```

```

        else{
            isVisible=false;
            $('ul').hide();
        }
    });
});

```

Décrire le résultat attendu en cas « d'ouverture » du fichier fos05.html avec un navigateur internet (Firefox, Chrome....). Quel est le rôle de la variable isVisible.

.....

.....

.....

la balise input et la méthode val()

La méthode «val» permet de récupérer la valeur entrée par un utilisateur dans une balise input.

HTML :

```
<input type="text" id="monInput"/>
```

JavaScript :

```
$('#monInput').val()
```

Si on a :

```
maValeur=$('#monInput').val()
```

«maValeur» contiendra la chaîne de caractères entrée par l'utilisateur.

«\$('#monInput').val('')» permet d'effacer le champ input

À faire vous même

code HTML fos06.html

```

<!doctype html>
<html lang="fr">
<head>
    <meta charset="UTF-8">
    <title>jQuery</title>
    <script src="lib/jquery-2.0.3.min.js"></script>
    <script src="javascript/fos06.js"></script>
</head>
<body>
    <h1>Ma liste de fruits</h1>
    <p>Ajouter un fruit: <input type="text" id="fruit"/></p>
    <button id="ajout">Valider</button>
    <ul>
    </ul>
</body>
</html>

```

code JavaScript fos06.js

```

$(function(){
    var mesFruitsTab=[];
    $('#ajout').on('click',function(){
        var fruit=$('#fruit').val();
        if (fruit!=''){
            mesFruitsTab.push(fruit);
            $('ul').append('<li>'+fruit+'</li>');
            $('#fruit').val('');
        }
    });
});

```

Décrire le résultat attendu en cas « d'ouverture » du fichier fos06.html avec un navigateur internet (Firefox, Chrome....)

.....

.....

.....

faire des recherches

À faire vous même

code HTML fos07.html

```
<!doctype html>
<html lang="fr">
<head>
  <meta charset="UTF-8">
  <title>jQuery</title>
  <script src="lib/jquery-2.0.3.min.js"></script>
  <script src="javascript/fos07.js"></script>
</head>
<body>
  <h1>Ma liste de fruits</h1>
  <p id='resultat'></p>
  <p>rechercher un fruit <input type="text" id="recherche"/></p>
  <button id="butVal">Valider</button>
</body>
</html>
```

code JavaScript fos07.js

```
$(function(){
  var mesFruitsTab=["banane","pomme","fraise","kiwi","pêche","cerise","nectarine",
    "noix","framboise","noisette","raisin"];
  $('#butVal').on('click',function(){
    $('#resultat').html('');
    var rech=$('#recherche').val();
    var flag=true;
    var res;
    $('#recherche').val('');
    var i=0;
    while (i<mesFruitsTab.length && flag==true){
      if (rech==mesFruitsTab[i]){
        res=rech;
        flag=false;
      }
      i=i+1;
    }
    if (flag==true){
      $('#resultat').html('Désolé, aucun résultat trouvé');
    }
    else{
      $('#resultat').html(res+' fait bien parti de la liste');
    }
  });
});
```

Nous utilisons ici une nouvelle méthode jQuery : html

Cette méthode **remplace** le code HTML (au contraire de append qui **ajoute** du code HTML).

Avant un \$('div').html('Hello World!')

<div><p><Bonjour le monde</p></div>

Après un \$('div').html('Hello World!')

<div>Hello World!</div>

avec append à la place d'html, nous aurions eu :

<div><p><Bonjour le monde</p>Hello World!</div>

Il est possible d'écrire : \$('div').html('Hello World!') => pas de balise HTML, juste un texte

«`$('p').html(' ')` » permet de «vider» le contenu d'une balise
Après un `$('div').html(' ')`
`<div></div>`

Décrire le résultat attendu en cas « d'ouverture » du fichier `fos06.html` avec un navigateur internet (Firefox, Chrome....)

.....

.....

.....

Nous venons de voir les bases de jQuery, pour plus d'informations n'hésitez pas à consulter la documentation officielle :

<http://api.jquery.com/>

Le site openClassrooms (anciennement «site du zéro») propose des cours et une vidéo sur le sujet :

<http://fr.openclassrooms.com/informatique/cours/simplifiez-vos-developpements-javascript-avec-jquery>

jQuery mobile

jQuery mobile est une extension de jQuery. Cette librairie est développée par l'équipe qui développe jQuery.

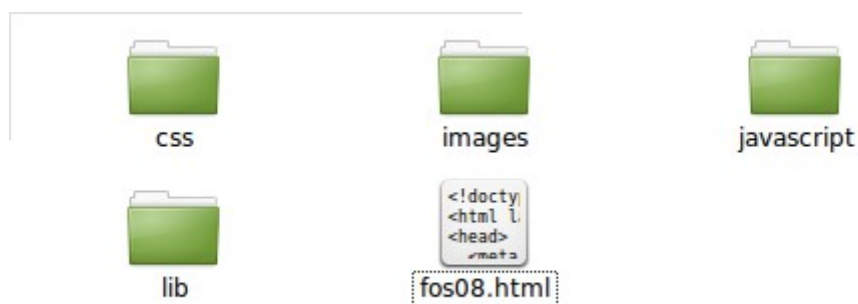
jQuery mobile est utilisée afin de faciliter la création d'interfaces utilisateurs dédiées aux appareils mobiles (smartphone et tablette). Il est nécessaire de comprendre que jQuery mobile est uniquement destinée à cette création d'interfaces utilisateurs, elle ne remplace pas jQuery, elle la complète.

Comme pour jQuery, jQuery mobile devra être téléchargée : <http://jquerymobile.com/download/>

Récupérez l'archive ZIP contenant les fichiers JavaScript, les fichiers CSS et les images (au moment de la rédaction de ce document : `jquery.mobile-1.3.2.zip`). Vous aurez aussi encore besoin du fichier `jquery-2.0.3.min.js`.

À faire vous même

Une fois le fichier téléchargé, vous allez mettre en place la structure (fichiers et dossiers) suivante :



- un fichier : `fos08.html`
- un dossier `css` qui contiendra le fichier `jquery.mobile-1.3.2.min.css` (à adapter en fonction de la version téléchargée)
- un dossier `images` qui contiendra les images récupérées du fichier ZIP jQuery mobile
- un dossier `lib` qui contiendra les fichiers `jquery.mobile-1.3.2.min.js` et `jquery-2.0.3.min.js` (à adapter en fonction des versions téléchargées)
- pour l'instant le dossier `javascript` restera vide

code HTML `fos08.html`

```
<!doctype html>
<html lang="fr">
<head>
  <meta charset="utf-8">
```

```

<meta name="viewport" content="width=device-width, initial-scale=1">
<title>jQuery Mobile pour Firefox OS</title>
<link rel="stylesheet" href="css/jquery.mobile-1.3.2.min.css">
<script src="lib/jquery-2.0.3.min.js"></script>
<script src="lib/jquery.mobile-1.3.2.min.js"></script>
</head>
<body>
</body>
</html>

```

Ce code devrait vous être familier à l'exception de la ligne «<meta name="viewport" content="width=device-width, initial-scale=1"> » est nécessaire pour que notre web app (ou notre site internet) s'adapte à la taille de l'écran de l'appareil mobile, nous ne rentrerons pas plus dans les détails.

Principe de jQuery mobile

Les principales « traces » de l'utilisation de jQuery mobile apparaîtront dans le fichier HTML. L'idée de base est de rajouter des attributs aux balises. Ces attributs seront tous de la forme data-xxx (en remplaçant xxx par différents termes que nous verrons plus loin).

Exemples : <div data-role='page'> ou encore <div data-theme='a'> (nous verrons plus loin la signification de ces attributs).

Premier pas avec jQuery mobile

Analysons ce code :

[code HTML fos09.html](#)

```

<!doctype html>
<html lang="fr">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>jQuery Mobile pour Firefox OS</title>
  <link rel="stylesheet" href="css/jquery.mobile-1.3.2.min.css">
  <script src="lib/jquery-2.0.3.min.js"></script>
  <script src="lib/jquery.mobile-1.3.2.min.js"></script>
</head>
<body>
  <div data-role='page'>
    <div data-role='header'><h3>Mon Application</h3></div>
    <div data-role='content'>
      <input id='texte' value='' placeholder='Entrez un élément' />
      <button id='valider' data-role='button' data-inline='true' data-theme="b">Valider</button>
    </div>
  </div>
</body>
</html>

```

Toutes les nouveautés se trouvent dans la balise body, entrons dans les détails :

«<div data-role='page'> »

Nous avons ici une balise div tout à fait classique, mais grâce à l'attribut «data-role='page'» cette balise div délimitera le contenu de notre page (nous verrons plus loin, dans les annexes, qu'il est possible de créer une application avec plusieurs pages, d'où l'intérêt de délimiter notre page grâce à cette balise div et son attribut «data-role='page'»).

«<div data-role='header'>»

Cette fois le contenu de notre div sera un en-tête. Il est possible de placer n'importe quel contenu dans cet en-tête : ici nous nous contentons d'un titre de type h3.

```
«<div data-role='content'> »
```

Le «content» est le cœur de la page (il est possible de placer plusieurs «content» dans une même page). Nous plaçons ici une balise input et un bouton.

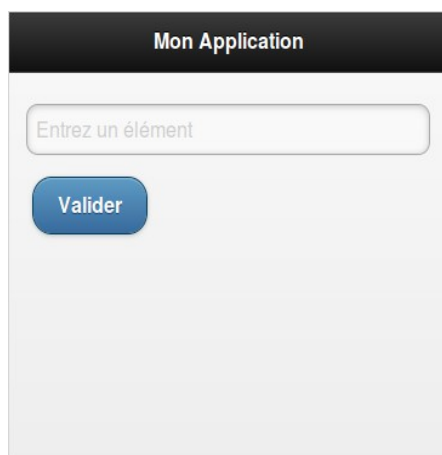
```
«<button id='valider' data-role='button' data-inline='true' data-theme='b'>Valider</button> »
```

Nous précisons le rôle de «button» (même si cela fait doublon avec la balise <button>, cela est nécessaire).

«data-inline='true'» permet d'éviter que le bouton prenne toute la largeur de la page.

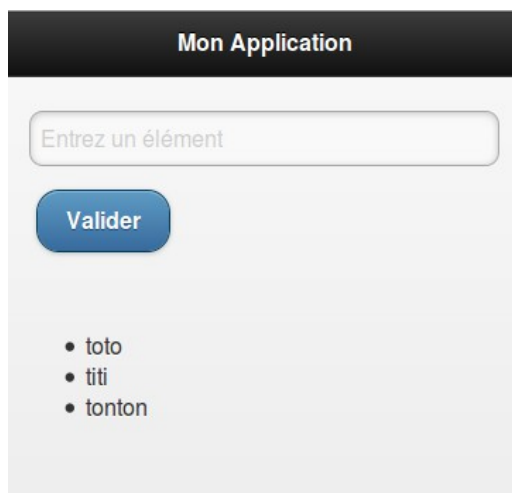
«data-theme='b'» permet de choisir un thème (couleur, police de caractère, forme...). Il existe différents thèmes, directement utilisables, nommés : a, b, c, d ou e. Il est possible de créer son propre thème, mais cela sort du cadre de ce document (il existe un outil dédié à cela : <http://jquerymobile.com/themeroller/index.php>)

Voici le résultat obtenu dans un navigateur (n'oubliez pas de réduire la largeur de la page afin de simuler le côté «mobile»):



À faire vous même

En vous basant sur l'exemple fos09, créer un fichier HTML et JavaScript permettant à l'utilisateur d'entrer un texte (à l'aide d'une balise input), de valider cette saisie (à l'aide d'un bouton) afin d'afficher les différents textes saisis par l'utilisateur sous forme de liste (n'oubliez pas d'utiliser ce que vous avez vu sur jQuery).



Ce document n'étant pas « un cours » sur jQuery mobile, nous n'irons pas plus loin dans la découverte de cette librairie (mais j'introduirai, par la suite, quand cela sera nécessaire, d'autres éléments jQuery mobile).

Avant de passer à la suite, voici quelques éléments qui vous permettront de poursuivre cette découverte par vous même :

Le site officiel jQuery mobile est très bien fait, il vous propose notamment des exemples très clairs et très bien documentés : <http://view.jquerymobile.com/1.3.2/dist/demos/>

Le site openClassrooms propose un cours consacré à jQuery mobile (même si au moment de la rédaction de ce document il n'est pas encore terminé) : <http://fr.openclassrooms.com/informatique/cours/un-site-web-dynamique-avec-jquery/le-framework-mobile-jquery-mobile>

Enfin, voici un livre consacré à jQuery mobile :

http://www.mobile-tuts.com/downloads/ebooks/D%C3%A9velopper_avec_jQuery_Mobile.pdf

Vous devez bien comprendre que jQuery mobile n'impose aucune contrainte, tout est modifiable, notamment grâce aux CSS, n'hésitez pas à « mettre les mains dans le cambouis ».

Enfin, je précise que l'utilisation de jQuery et de jQuery mobile n'est en rien obligatoire, il est possible de construire une web app (et donc une application Firefox OS) sans aucune librairie. Cependant cela demande des connaissances poussées, notamment au niveau des CSS (faites une recherche sur la notion de media queries).

Application Firefox OS

Nous allons maintenant installer les outils facilitant le développement d'applications pour Firefox OS.

Ouvrez Firefox (le navigateur !), dans le menu «Outils», choisissez «Modules complémentaires». Saisissez « Firefox OS » dans le champ de recherche situé en haut à droite. Vous devriez normalement voir apparaître plusieurs propositions, choisissez «Firefox OS Simulator 4.0 » (le numéro de version sera peut-être différent) en cliquant sur le bouton «Installer». Le module devrait alors se télécharger.

Voici la première application que nous allons faire fonctionner sous Firefox OS :

À faire vous même

code HTML fos10.html

```
<!doctype html>
<html lang="fr">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>jQuery Mobile with Firefox OS</title>
  <link rel="stylesheet" href="css/jquery.mobile-1.3.2.min.css">
  <script src="lib/jquery-2.0.3.min.js"></script>
  <script src="lib/jquery.mobile-1.3.2.min.js"></script>
  <script src="javascript/fos10.js"></script>
</head>
<body>
  <div data-role='page'>
    <div data-role='header'><h3>Mon Application</h3></div>
    <div data-role='content'>
      <input id='texte' value='' placeholder='Entrez un élément' />
      <button id='valider' data-role='button' data-inline='true' data-theme="b">Valider</button>
    </div>
    <div data-role='content'>
      <ul id='liste_nom'>
      </ul>
    </div>
  </div>
</body>
</html>
```

code JavaScript fos10.js

```
$(function(){
  $('#valider').on('click',function(){
```



```

        var nom=$('#texte').val();
        $('#texte').val('');
        if (nom!=''){
            $('#liste_nom').append('<li>'+nom+'</li>');
        }
        $('#valider').blur();
    })
});

```

Je pense que vous aurez remarqué qu'il s'agit ni plus ni moins de la correction du dernier «**A faire vous même**» en date.

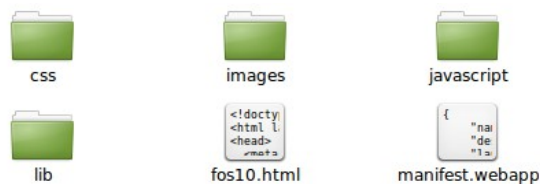
Nous n'avons plus que 2 choses à faire avant de pouvoir « profiter » de notre application sur le simulateur de smartphone Firefox OS :

- Une image qui jouera le rôle d'icône (icône qui permettra de « lancer » notre application sur le smartphone). Cette image doit avoir une taille de 128 pixels sur 128 pixels. Voici celle que j'ai choisie pour illustrer cet exemple :



Vous placerez votre image dans le dossier image

- Un fichier dénommé «manifest.webapp». Ce fichier contient toutes les informations nécessaires pour que Firefox OS «transforme» votre web app en application pour smartphone Firefox OS. Ce fichier devra être placé au même niveau que le fichier fos10.html



Voici le contenu de ce fichier :

```

{
  "name": "Mon application",
  "description": "premier test Firefox OS",
  "launch_path": "/fos10.html",
  "icons": {
    "128": "/images/icon128.png"
  },
  "developer": {
    "name": "LGF",
    "url": "LGF.com"
  },
  "default_locale": "fr"
}

```

Comme vous l'aurez sans doute remarqué, ce fichier est de type « objet JavaScript» (si nécessaire replongez-vous dans le document «19 activités pour apprendre à programmer avec JavaScript, HTML et CSS»)

Pour être tout à fait exact, nous avons un fichier de type JSON (nous aurons l'occasion de revenir sur le JSON un peu plus loin dans ce document). Analysons le fichier ligne par ligne :

- «"name": "Mon application"» : le nom de l'application !
- «"description": "premier test Firefox OS"» : la description de l'application !

- «"launch_path": "/fos10.html"» : indique la position du fichier HTML
- «"icons": {"128": "/images/icon128.png" }» : indique la position et le nom de l'image utilisés pour jouer le rôle d'icône. Il est possible d'avoir plusieurs tailles d'images
- «"developer": {"name": "LGF", "url": "LGF.com" }» : des informations sur le développeur de l'application (url : adresse de son site internet)
- «"default_locale": "fr"» : langue utilisée par défaut (important par exemple pour le clavier)

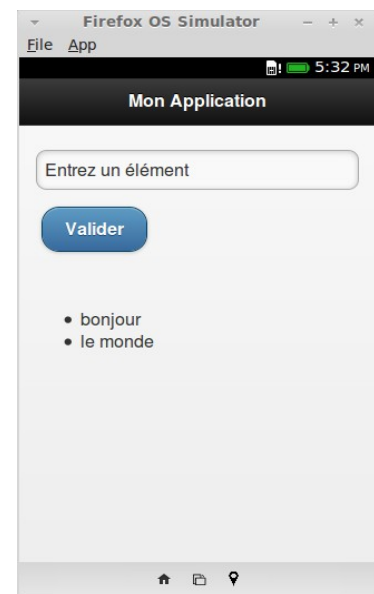
Nous aurons l'occasion de compléter ce fichier dans la suite de ce document, pour l'instant celui-ci nous suffit.

Voilà, tous les éléments sont maintenant en place, il ne nous reste plus qu'à «installer» notre application sur le smartphone (simulateur).

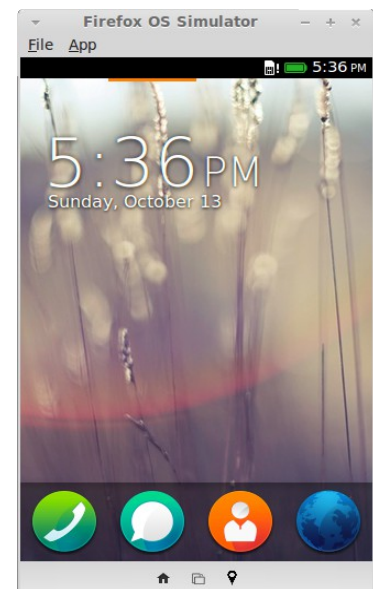
Pour utiliser le simulateur, sous Firefox allez dans «Outils» puis « Développeur web» et enfin «Firefox OS Simulator».

Sur la page qui vient de se charger, cliquez sur le bouton « Add Directory». Indiquez alors la position de votre fichier « manifest.webapp».

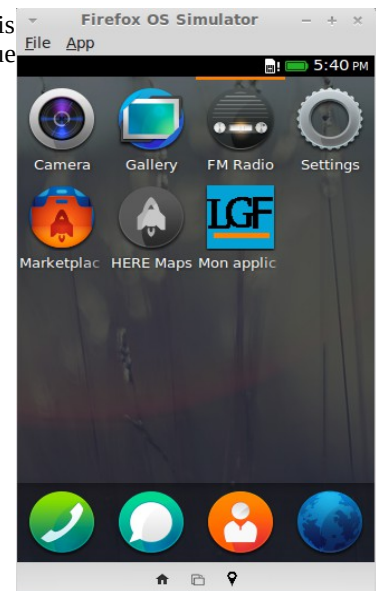
Le simulateur et votre application devraient se «lancer» automatiquement.



Pour revenir sur la page d'accueil du téléphone (home screen), il suffit de cliquer sur la petite maison en bas gauche du simulateur.



En balayant l'écran du simulateur de droite à gauche avec le curseur de la souris (équivalent du « balayage » de l'écran avec le doigt), vous devriez pouvoir constater que votre application est bien installée.



Vous pouvez maintenant quitter le simulateur, au prochain lancement votre application sera toujours « installée ».

Cela ne fonctionne pas : débogage d'une application avec le simulateur Firefox OS

Déboguer (de l'anglais bug (essayez de trouver l'origine de ce terme)) c'est essayer de trouver les erreurs qui provoquent le non (ou mauvais) fonctionnement d'une application.

Firefox OS fournit une «console» qui vous permettra de connaître les erreurs détectées par le système.

Pour faire apparaître cette console, il vous suffira d'appuyer sur le bouton «Connect»



La console devrait s'ouvrir et l'application se lancer (il sera parfois nécessaire de faire un « Refresh » pour voir apparaître les erreurs dans la console.

Attention, à cause de l'utilisation de jQuery mobile, la console est souvent polluée par des « fausses erreurs », prenez garde de ne pas tenir compte de ces « fausses erreurs » (elles sont relativement facilement détectables).

Stocker des informations sur le smartphone : le local storage

Si vous relancez votre application (après avoir redémarré le simulateur), vous constaterez que la liste de mots précédemment saisie a disparu.

Nous allons voir ici une méthode, issue des API HTML5, permettant de sauvegarder des données localement (sur le téléphone) afin de pouvoir les récupérer même après un arrêt du téléphone.

Ce système de sauvegarde repose sur la notion de paire clé/valeur.

La méthode «setItem» de l'objet «localStorage» permet de stocker cette paire clé/valeur. Imaginons qu'une personne se nomme toto, pour sauvegarder ce nom, il suffira d'écrire : «localStorage.setItem('nom','toto')». Dans cet exemple, «nom» est une clé et «toto» est la valeur associée à cette clé.

Il est ensuite possible de récupérer la valeur associée à une clé en utilisant la méthode «getItem».

«localStorage.getItem('nom')» sera égal à «toto».

Il est possible de connaître le nombre de paire clé/valeur dans le localStorage grâce à «length» :

```
localStorage.setItem('nom', 'toto');
```

```
localStorage.setItem('prenom', 'titi');
```

On aura ici «localStorage.length» qui sera égal à 2

Il est possible de modifier une valeur associée à une clé, il suffit de réutiliser la méthode «setItem» avec la même clé que précédemment : «localStorage.setItem('nom', 'tata')».

Encore 2 méthodes à connaître avant de passer à un cas concret :

- la méthode «removeItem» permet de supprimer une paire clé/valeur : `localStorage.removeItem('nom')`
- la méthode «clear» permet d'effacer le localStorage dans son intégralité (toutes les paires clé/valeur)

À faire vous même

Imaginons une application qui a besoin du nom et du prénom de l'utilisateur pour fonctionner. Sans le localStorage il faudrait redemander son nom et son prénom à l'utilisateur à chaque redémarrage du téléphone.

Écrivez une application qui après avoir demandé son nom et son prénom à l'utilisateur les affiche. Utilisez le localStorage pour que le nom et le prénom saisis restent affichés même après avoir redémarré le simulateur (si l'utilisateur entre un nouveau nom et/ou un nouveau prénom, cette nouvelle saisie devra remplacer l'ancienne).



Pour l'instant, nos applications travaillent en «local» (tout se passe sur le téléphone). La plupart des applications sur smartphone ont besoin d'aller chercher des informations «sur internet». Mais qu'est qu'internet ? Il est temps de faire un peu de «théorie» :

Notion de client-serveur

Que se passe-t-il lorsque vous tapez dans la barre d'adresse de votre navigateur «http://www.google.fr» ?

Votre ordinateur (que l'on appellera le client) va chercher à entrer en communication avec un autre ordinateur (que l'on appellera le serveur) se trouvant probablement à des milliers de kilomètres de chez vous. Pour pouvoir établir cette communication, il faut bien sûr que les 2 ordinateurs, le client et le serveur, soient « reliés ». On dira que nos 2 ordinateurs sont en réseau.

Il existe énormément de réseaux (les ordinateurs de la salle sont tous en « réseau »), certains réseaux sont reliés à d'autres réseaux qui sont eux-mêmes reliés à d'autres réseaux.....ce qui forme « des réseaux de réseaux de réseaux..... ». Savez-vous comment on appelle cet assemblage multiple de réseaux ? Internet !

Mon but ici n'est pas de vous expliquer comment font les ordinateurs pour se trouver dans cet « amas de réseaux », si ce sujet vous intéresse, vous rencontrerez certains termes, comme : « serveur DNS », « routeur », « adresse IP », « routage ».....

En tapant «http://www.google.fr», votre machine va chercher à entrer en communication avec le serveur portant le nom «www.google.fr» (en fait c'est plus compliqué. Pour les puristes nous dirons donc que la communication va être établie avec le serveur www du domaine google.fr, mais bon, pour la suite nous pourrions nous contenter de l'explication « simplifiée »).

Une fois la liaison établie, le client et le serveur vont échanger des informations en dialoguant :

client : bonjour www.google.fr (ou bonjour www se trouvant dans le domaine google.fr), pourrais-tu m'envoyer le code html contenu dans le fichier index.html ?

serveur : OK, voici le code html demandé

client : je constate que des images, du code css et du code JavaScript sont utilisés, peux-tu me les envoyer ?

serveur : OK, les voici

Évidemment ce dialogue est très imagé, mais il porte tout de même une part de « vérité ».

J'espère que vous commencez à comprendre les termes client (celui qui demande quelque chose) et serveur (celui qui

fournit ce qui a été demandé).

et le HTTP ?

Nous venons de voir que les ordinateurs communiquent entre eux, pour assurer cette communication, ils utilisent ce que l'on appelle des protocoles.

Selon Wikipedia, dans le cas général, protocole :

On nomme protocole les conventions qui facilitent une communication sans faire directement partie du sujet de la communication elle-même.

En électronique et en informatique (toujours selon Wikipedia) :

un protocole de communication est un ensemble de contraintes permettant d'établir une communication entre deux entités (dans le cas qui nous intéresse 2 ordinateurs)

Pour que la communication soit possible, le client et le serveur doivent avoir des règles communes, ces règles sont définies dans un protocole. Comme vous l'avez sans doute deviné, le protocole de communication employé ici se nomme HTTP.

Le protocole HTTP (HyperText Transfer Protocol) a été inventé par Tim Berners-Lee (1955-....) au début des années 1990. Tim Berners-Lee est aussi à l'origine du langage HTML et des « adresses web ». C'est la combinaison de ces 3 éléments (HTTP, HTML, « adresse web ») que l'on nomme aujourd'hui le « web » (« web » qu'il ne faut pas confondre avec l'internet (erreur ultra classique), même si le web utilise l'internet).

Le HTTP va permettre au client d'effectuer des requêtes à destination d'un serveur. En retour, le serveur va envoyer une réponse.

Voici une version simplifiée de la composition d'une requête :

- la méthode employée pour effectuer la requête
- l'URL de la ressource
- la version du protocole utilisé par le client (souvent HTTP 1.1)
- le navigateur employé (Firefox, Chrome) et sa version
- le type du document demandé (par exemple HTML)
-

Certaines de ces lignes sont optionnelles.

Voici un exemple de requête HTTP (la méthode, l'URL et la version du protocole se trouvent sur la première ligne) :

```
GET /mondossier/monFichier.html HTTP/1.1
```

```
User-Agent : Mozilla/5.0
```

```
Accept : text/html
```

Revenons uniquement sur 2 aspects (si nécessaire nous reviendrons sur les autres plus tard) : la méthode employée et l'URL.

Les méthodes des requêtes HTTP

Une requête HTTP utilise une méthode (c'est une commande qui demande au serveur d'effectuer une certaine action). Voici la liste des méthodes disponibles :

GET, HEAD, POST, OPTIONS, CONNECT, TRACE, PUT, PATCH, DELETE

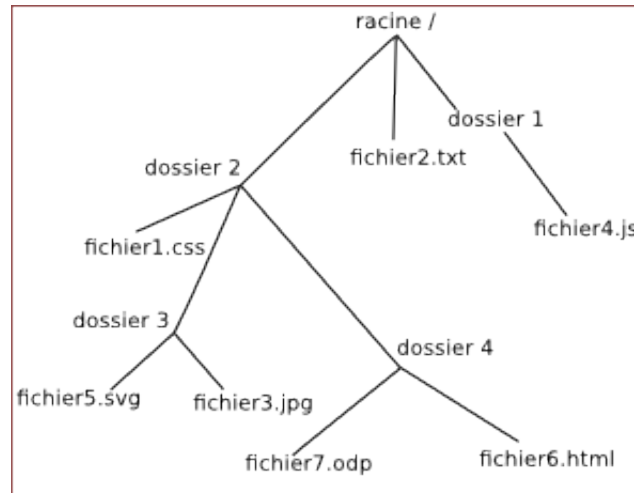
Détaillons 4 de ces méthodes :

- GET : C'est la méthode la plus courante pour demander une ressource. Elle est sans effet sur la ressource.
- POST : Cette méthode est utilisée pour soumettre des données en vue d'un traitement (côté serveur). Typiquement c'est la méthode employée lorsque l'on envoie au serveur les données issues d'un formulaire.
- DELETE : Cette méthode permet de supprimer une ressource sur le serveur.
- PUT : Cette méthode permet de modifier une ressource sur le serveur

L'URL (et l'URI)

Une URI (Uniform Resource Identifier) permet d'identifier une ressource sur un réseau, une URL est un cas particulier d'URI. Nous ne nous attarderons pas sur les subtiles différences entre une URI et une URL et à partir de maintenant je parlerai exclusivement d'URL (par souci de simplification).

L'URL indique « l'endroit » où se trouve une ressource sur le serveur. Un fichier peut se trouver dans un dossier qui peut lui-même se trouver dans un autre dossier.....on parle d'une structure en arborescence, car elle ressemble à un arbre à l'envers :



Comme vous pouvez le constater, la base de l'arbre s'appelle la racine de l'arborescence et se représente par un /

Chemin absolu ou chemin relatif ?

Pour indiquer la position d'un fichier (ou d'un dossier) dans l'arborescence, il existe 2 méthodes : indiquer un chemin absolu ou indiquer un chemin relatif. Le chemin absolu doit indiquer « le chemin » depuis la racine. Par exemple l'URL du fichier fichier3.txt sera : /dossier2/dossier3/fichier3.jpg

Remarquez que nous démarrons bien de la racine / (attention les symboles de séparation sont aussi des /)

Imaginons maintenant que le fichier fichier1.css fasse appel au fichier fichier3.jpg (comme un fichier HTML peut faire appel à un fichier CSS ou JavaScript). Il est possible d'indiquer le chemin non pas depuis la racine, mais depuis le dossier (dossier2) qui accueille le fichier1.css, nous parlerons alors de chemin relatif :

dossier3/fichier3.jpg

Remarquez l'absence du / au début du chemin (c'est cela qui nous permettra de distinguer un chemin relatif et un chemin absolu).

Imaginons maintenant que nous désirions indiquer le chemin relatif du fichier fichier1.txt depuis le dossier dossier4.

Comment faire ?

Il faut « reculer » d'1 « cran » dans l'arborescence (pour se retrouver dans le dossier dossier2 et ainsi pouvoir repartir vers la bonne « branche ». Pour ce faire il faut utiliser 2 points : ..

../dossier2/fichier3.jpg

Il est tout à fait possible de remonter de plusieurs « crans » : ../../ depuis le dossier dossier4 permet de « retourner » à la racine.

Remarque : la façon d'écrire les chemins (avec des slash (/) comme séparateurs) est propre aux systèmes dits « UNIX », par exemple GNU/Linux ou encore Mac OS. Sous Windows, ce n'est pas le slash qui est utilisé, mais l'antislash (\). Pour ce qui nous concerne ici, les chemins réseau (et donc le web), pas de problème, c'est le slash qui est utilisé.

Réponse du serveur à une requête HTTP

Une fois la requête reçue, le serveur va renvoyer une réponse, voici un exemple de réponse du serveur :

```
HTTP/1.1 200 OK
Date: Thu, 15 Feb 2013 12:02:32 GMT
Server: Apache/2.0.54 (Debian GNU/Linux) DAV/2 SVN/1.1.4
Connection: close
Transfer-Encoding: chunked
```

Content-Type: text/html; charset=ISO-8859-1

```
<!doctype html>
<html lang="fr">
<head>
<meta Charest="utf-8">
<title>Voici mon site</title>
</head>
<body>
<h1>Hello World! Ceci est un titre</h1>
<p>Ceci est un <strong>paragraphe</strong>. Avez-vous bien compris ?</p>
</body>
</html>
```

Nous n'allons pas détailler cette réponse, voici quelques explications sur les éléments qui nous seront indispensables par la suite :

Commençons par la fin : le serveur renvoie du code HTML, une fois ce code reçu par le client, il est interprété par le navigateur qui affiche le résultat à l'écran. Cette partie correspond au corps de la réponse.

La 1re ligne se nomme la ligne de statut :

- HTTP/1.1 : version de HTTP utilisée par le serveur
- 200 : code indiquant que le document recherché par le client a bien été trouvé par le serveur. Il existe d'autres codes dont un que vous connaissez peut-être déjà : le code 404 (qui signifie «Le document recherché n'a pu être trouvé»).

Les 5 lignes suivantes constituent l'en-tête de la réponse, une ligne nous intéresse plus particulièrement : «Server : Apache/2.0.54 (Debian GNU/Linux)».

Pourquoi vous ai-je raconté tout cela ? Simplement parce que les applications, pour récupérer (envoyer) des données sur (vers) un serveur, vont utiliser le protocole HTTP.

Nous allons donc maintenant nous concentrer sur les méthodes à utiliser pour effectuer ces requêtes.

An contraire de l'exemple exposé ci-dessus, le serveur ne va pas nous renvoyer une page HTML à afficher (il le pourrait, mais ce n'est pas ce cas qui va nous intéresser), il va nous renvoyer un fichier au format JSON.

JSON ?

D'après Wikipédia :

JSON (JavaScript Object Notation) est un format de données textuelles, générique, dérivé de la notation des objets du langage ECMAScript (et donc JavaScript). Il permet de représenter de l'information structurée. Il a été créé par Douglas Crockford (auteur d'un livre qui fait, selon moi, autorité dans le domaine du JavaScript : [«JavaScript garder le meilleur»](#)).

Pour faire simple, le JSON utilise des paires clé/valeur où la clé est une chaîne de caractères (et donc mise entre guillemets).

Voici un exemple de données au format JSON (toujours selon Wikipédia) :

```
{
  "menu": {
    "id": "file",
    "value": "File",
    "popup": {
      "menuitem": [
        { "value": "New", "onclick": "CreateNewDoc()" },
        { "value": "Open", "onclick": "OpenDoc()" },
        { "value": "Close", "onclick": "CloseDoc()" }
      ]
    }
  }
}
```

Comme vous pouvez le constater, il est possible d'avoir une certaine imbrication (la valeur associée à la clé popup est une clé ("menuitem") qui a pour valeur un tableau qui contient des paires clé/valeur...)

Il est évidemment possible d'avoir des données JSON beaucoup plus simples :

```
{
    "nom" : "toto",
    "prenom" : "titi",
    "age" : 42
}
```

Vous remarquerez que chaque paire clé/valeur est séparée par une virgule.

JSON ressemble beaucoup aux objets JavaScript, puisqu'un objet JavaScript équivalent serait :

```
{
    nom : "toto",
    prenom : "titi",
    age : 42
}
```

Il est très simple de transformer du JSON en objet JavaScript (qui sera lui directement exploitable dans un programme).

Si votre fichier contenant le JSON se nomme monFichier et que vous voulez, à partir de ce fichier, obtenir un objet JavaScript nommé monObjet, il suffira d'écrire :

```
monObjet = JSON.parse(monFichier)
```

Typiquement cette opération sera à effectuer quand le serveur vous enverra des données au format JSON.

Il est possible aussi que vous désiriez transformer un objet JavaScript en JSON (par exemple pour envoyer des données vers le serveur), il faudra alors utiliser «JSON.stringify(monObjet)».

exemple : JSON.stringify ({age:42}) renverra {"age":42}

Pour résumer, les données transiteront entre le client et le serveur sous forme JSON et seront exploitées côté client sous forme d'objet JavaScript.

Il existe un autre format de données permettant ces échanges client-serveur : le format XML

Voici des données au format XML :

```
<menu id="file" value="File">
  <popup>
    <menuitem value="New" onclick="CreateNewDoc()" />
    <menuitem value="Open" onclick="OpenDoc()" />
    <menuitem value="Close" onclick="CloseDoc()" />
  </popup>
</menu>
```

Ce format est historiquement antérieur au JSON et même s'il est encore très utilisé aujourd'hui, il est un peu en perte de vitesse (au profit du JSON). Nous ne l'étudierons pas ici, mais il fallait quand même l'évoquer (si vous avez un jour besoin de récupérer des données au format XML, vous trouverez les ressources nécessaires sur internet).

Effectuer une requête HTTP

Pour effectuer la requête, il faut suivre une certaine procédure que je vais décrire maintenant :

Il faut tout d'abord instancier un objet de type XMLHttpRequest :

```
var xhr = XMLHttpRequest ();
```

il faut ensuite utiliser la méthode open de cet objet de type XMLHttpRequest :

```
xhr.open('GET',url,true)
```

Quelques explications sur les paramètres de cette méthode :

- le premier paramètre correspond à la méthode HTTP à employer pour effectuer la requête (principalement GET pour « demander » des données au serveur et POST pour envoyer des données au serveur).
- url correspond à l'url du serveur à atteindre, nous verrons que cette url pourra contenir des paramètres afin de « personnaliser » la requête.

- true permet de préciser que la requête sera asynchrone (le client n'attendra pas la réponse du serveur pour continuer à vaquer à ses occupations, la requête sera donc « non bloquante »), si l'on désire une requête synchrone (bloquante), il faut mettre le paramètre à false.

Nous allons ensuite mettre en place la méthode «onreadystatechange» qui aura pour « mission » de « surveiller » la requête :

```
xhr.onreadystatechange = function () {
    if (xhr.readyState == 4 && xhr.status == 200){
        .....
    }
}
```

Quelques explications :

La requête HTTP va passer par différentes phases, chaque phase étant caractérisée par un chiffre :

0 : la requête n'a pas encore été initialisée

1 : la connexion avec le serveur est établie

2 : la requête a été reçue par le serveur

3 : le serveur traite la requête

4 : la requête est terminée

C'est l'attribut « readyState » de l'objet « xhr » (xhr.readyState) qui prend tour à tour la valeur 0 , la valeur 1,...et enfin la valeur 4 quand la requête est terminée.

À chaque évolution de la valeur de «xhr.readyState», la méthode «onreadystatechange» est appelée. Dans notre exemple ci-dessus, rien ne va se passer avant que «xhr.readyState» soit égale à 4 («if (xhr.readyState == 4»).

«xhr.status» contient le code de réponse du serveur (200 : tout est OK, 404 : le document n'a pas pu être trouvé)

Pour « entrer » dans le if (« if (xhr.readyState == 4 && xhr.status == 200)»), il faut donc satisfaire à 2 conditions :

- la requête est terminée («xhr.readyState == 4»)
- le serveur a renvoyé le code 200 («xhr.status == 200»)

Continuons à explorer le code permettant de mener à bien notre requête :

```
if (xhr.readyState == 4 && xhr.status == 200){
    result=JSON.parse(xhr.responseText) ;
}
```

Si tout s'est bien passé (4 et 200), nous récupérerons les informations demandées auprès du serveur grâce à «xhr.responseText» (pour nous ce sera des données au format JSON).

Enfin pour terminer il nous reste à envoyer la requête grâce à la méthode «send»:

```
xhr.send(null);
```

En cas d'utilisation de la méthode HTTP «POST», les données à envoyer au serveur remplaceront le paramètre «null».

AJAX ?

On parle de requête de type Ajax pour qualifier l'utilisation de l'objet XMLHttpRequest :

Asynchronous (la requête HTTP n'est pas bloquante) Javascript (on utilise du JavaScript) And XML (à l'origine on échangeait des données exclusivement au format XML).

Attention : pour ceux qui ont l'habitude d'utiliser jQuery pour effectuer leurs requêtes Ajax (par exemple avec « \$.ajax »), il semble qu'il ne soit pas possible d'utiliser ces méthodes dans les applications Firefox OS (mais cela peut évoluer).

Une dernière chose propre aux requêtes Ajax sous Firefox OS : afin de ne pas « subir » des échecs de nos requêtes à cause du « same-origin policy » (voir [ici](#) pour plus d'explications) nous devons :

- ajouter un paramètre à la méthode XMLHttpRequest : «var xhr = new XMLHttpRequest({mozSystem: true});»

- modifier le fichier manifest.webapp en ajoutant 2 lignes :
 "type": "privileged",
 "permissions": { "systemXHR": { "description": "Nécessaire pour faire des appels Ajax" } }

Pour illustrer tout cela, nous allons maintenant voir un cas concret.

Nous allons écrire une application météo : l'utilisateur entrera une localité française l'application lui fournira en retour le dernier bulletin disponible (pas les prévisions (même si cela est tout à fait possible), juste les conditions météo actuelles).

Nous allons utiliser le site <http://openweathermap.org/API> qui fournit une série d'API (voir la définition d'API sur Wikipédia) permettant de récupérer différentes données météo au format JSON (le XML est également disponible).

La demande d'informations se fera tout simplement par l'intermédiaire de l'url passée à la méthode open («xhr.open('GET',url,true)»)

À faire vous même

Allez visiter le site <http://openweathermap.org/API> (en anglais ;-)) afin de vous familiariser avec les différentes options.

Dans la barre d'adresse de votre navigateur favori, entrez l'url suivante :

<http://api.openweathermap.org/data/2.5/weather?q=bonneville,fr&lang=fr&units=metric>

si tout fonctionne, vous devriez obtenir quelque chose ressemblant à ceci :

```
{ "coord": { "lon": 6.40795, "lat": 46.0777 }, "sys": { "country": "France", "sunrise": 1382335229, "sunset": 1382373438 }, "weather": [ { "id": 803, "main": "Clouds", "description": "nuages éparses", "icon": "04d" } ], "base": "global stations", "main": { "temp": 15.65, "pressure": 1018, "humidity": 87, "temp_min": 14, "temp_max": 17 }, "wind": { "speed": 0.71, "deg": 119.002 }, "clouds": { "all": 75 }, "dt": 1382372400, "id": 3035563, "name": "Bonneville", "cod": 200 }
```

Comme vous pouvez le constater, nous récupérons bien des données au format JSON.

Étudions l'url :

- «<http://api.openweathermap.org/data/2.5/weather>» la première partie de l'url est tout à fait classique, rien de spécial à dire
- «[?q=bonneville,fr&lang=fr&units=metric](http://api.openweathermap.org/data/2.5/weather?q=bonneville,fr&lang=fr&units=metric)» c'est la partie la plus intéressante de l'url, c'est ici que nous personnalisons notre demande auprès du serveur grâce aux paramètres (délimitation des paramètres dans l'url grâce au point d'interrogation) : nous avons choisi la ville de Bonneville qui se trouve en France («q=bonneville,fr»), nous voulons avoir les informations en français («lang=fr») et nous voulons utiliser le système métrique («units=metric»). Ces différents paramètres sont séparés à l'aide du caractère &.

Chaque site fournissant des données possède sa propre API, il vous faudra donc étudier ces API afin de construire l'url. Cette méthode (faire une demande auprès d'un serveur grâce à l'url) est très souvent utilisée (API twitter, API Facebook,...)

Si la programmation côté serveur vous intéresse (et donc la mise au point d'API), je ne serai trop vous recommander la lecture du document « Création de web app avec jQuery, nodeJS et mongoDB » :

<http://www.webisn.byethost5.com/jquery>

À faire vous même

Après avoir étudié (en faisant des recherches sur internet) :

- la notion de timestamp (en informatique), attention, en JavaScript le timestamp est en milliseconde
- les objets «Date» en JavaScript (avec ses méthodes associées : getDate(), getMonth(), getFullYear()....)
- la méthode Math.floor en JavaScript

Étudiez les codes suivants :

code HTML fos11.html

```
<!doctype html>
<html lang="fr">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
```

```

<title>Firefox OS</title>
<link rel="stylesheet" href="css/jquery.mobile-1.3.2.min.css">
<script src="lib/jquery-2.0.3.min.js"></script>
<script src="lib/jquery.mobile-1.3.2.min.js"></script>
<script src="javascript/fos11.js"></script>
</head>
<body>
  <div data-role='page'>
    <div data-role='header'><h3>Météo</h3></div>
    <div data-role='content'>
      <input id='ville' value='' placeholder="Entrez le nom d'une ville"/>
      <button id='valider' data-role='button' data-inline='true' data-theme="b" data-mini="true">Valider</button>
      <div id='resultat'>
      </div>
    </div>
  </div>
</body>
</html>

```

code JavaScript fos11.js

```

$(function(){
$('#valider').on('click',function(){
var v=$('#ville').val();
var result={};
var url='http://api.openweathermap.org/data/2.5/weather?q='+v+',fr&lang=fr&units=metric';
$('#valider').blur();
if (v!=''){
//appel Ajax début
var xhr = new XMLHttpRequest({mozSystem: true});
xhr.open("GET", url, true);
xhr.onreadystatechange = function () {
if (xhr.readyState == 4){
if (xhr.status == 200){
result=JSON.parse(xhr.responseText);
$('#ville').val('');
$('#resultat').html('');
if (result.cod==200){
$('#resultat').append('<p>'+result.name+' le '+laDate(1000*result.dt)+' à '+lHeure(1000*result.dt)+'</p>');
var descri='';
for (var i=0;i<result.weather.length;i++){
descri=descri+' '+result.weather[i].description;
}
$('#resultat').append('<p>'+descri+'</p>');
$('#resultat').append('<p>Ciel couvert à '+result.clouds.all+' %</p>');
$('#resultat').append('<p> Température : '+Math.floor(result.main.temp)+' °C</p>');
$('#resultat').append('<p>Vitesse du vent : '+Math.floor(result.wind.speed)+' Km/h </p>');
var directionVent=dirVent(result.wind.deg);
if (Math.floor(result.wind.speed)!=0 && directionVent!=undefined){
$('#resultat').append('<p>Direction du vent : '+dirVent(result.wind.deg)+'</p>');
}
}
}
else {
$('#resultat').append('<p>Désolé aucun résultat disponible pour :</p><p>'+v+'</p>');
}
}
}

```

```

    }
}
else{
    ($('#resultat').append('<p>Désolé le serveur ne semble pas disponible</p>'));
}
}
}
xhr.send(null);
//fin Ajax
}
else {
    ($('#resultat').append('<p>Désolé, il faut entrer un nom</p>'));
}
});
laDate=function(timestamp){
    var maDate=new Date(timestamp);
    return maDate.getDate()+'/'+(1+maDate.getMonth())+'/'+maDate.getFullYear();
}
lHeure=function(timestamp){
    var maDate=new Date(timestamp);
    return maDate.getHours()+ ' h '+maDate.getMinutes();
}
dirVent=function(dir){
    if (dir>338 || dir<23){
        return 'Nord'
    }
    if (dir>22 && dir<68){
        return 'Nord-Est'
    }
    if (dir>67 && dir<113){
        return 'Est'
    }
    if (dir>112 && dir<158){
        return 'Sud-Est'
    }
    if (dir>157 && dir<203){
        return 'Sud'
    }
    if (dir>202 && dir<248){
        return 'Sud-Ouest'
    }
    if (dir>247 && dir<293){
        return 'Ouest'
    }
    if (dir>292 && dir<339){
        return 'Nord-Ouest'
    }
}
}
});

```

code manifest.webapp

```
{
  "name": "Météo",
  "description": "affiche info météo",
  "launch_path": "/fos11.html",
  "icons": {
    "128": "/images/icon128.png"
  },
  "developer": {
    "name": "LGF",
    "url": "LGF.com"
  },
  "default_locale": "fr",
  "type": "privileged",
  "permissions": {
    "systemXHR": { "description": "Nécessaire pour faire des appels Ajax" }
  }
}
```

À faire vous même

Comme vous l'avez sans doute remarqué, l'aspect esthétique de notre application laisse à désirer (et ce n'est pas peu dire). Améliorez cette application en utilisant les ressources proposées par jQuery mobile (par exemple, vous pourrez, si vous le désirez, utiliser les listes avec l'attribut «listview»).

GPS et géolocalisation

Les smartphones sont aujourd'hui tous équipés de GPS (Global Positioning System). Il est relativement simple de récupérer les coordonnées (longitude et latitude) du téléphone.


La première chose à faire est de vérifier que la géolocalisation est disponible, si cette géolocalisation est disponible «navigator.geolocation» renverra «true».

Il suffit ensuite d'utiliser la méthode `getCurrentPosition`. Cette méthode prend 2 fonctions de callback, une fonction qui sera appelée en cas de succès et une fonction qui sera appelée en cas d'échec :

«navigator.geolocation.getCurrentPosition(callBackSucces, callBackError)»

En cas de succès «position.coords.latitude» renverra la latitude et «position.coords.longitude » renverra la longitude.

GPS et simulateur Firefox OS

Le simulateur Firefox permet de simuler le GPS. Pour cela il vous faut appuyer sur le bouton  situé dans le bas du simulateur.

Vous aurez le choix entre :

- saisir les coordonnées
- laisser le système détecter la position de votre ordinateur (eh oui, c'est possible, même sans GPS, notamment grâce à votre adresse IP (rechercher ce terme si vous ne le connaissez pas)).

Modifier le fichier manifest.webapp

il faut modifier le fichier manifest.webapp afin d'autoriser l'utilisation du GPS :

```
"permissions": {
  "geolocation": {
    "description": "localisation"
  }
}
```

À faire vous même

Après avoir recherché l'utilité de «toFixed()» en JavaScript, vous analyserez le code suivant :

code HTML fos12.html

```
<!doctype html>
<html lang="fr">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Firefox OS</title>
  <link rel="stylesheet" href="css/jquery.mobile-1.3.2.min.css">
  <script src="lib/jquery-2.0.3.min.js"></script>
  <script src="lib/jquery.mobile-1.3.2.min.js"></script>
  <script src="javascript/fos12.js"></script>
</head>
<body>
  <div data-role='page'>
    <div data-role='header'><h3>Géoloc</h3></div>
    <div data-role='content'>
      <button id='maPos' data-role='button' data-inline='true' data-theme="b" data-mini="true">Ma position</button>
      <div id='resultat'></div>
    </div>
  </div>
</body>
</html>
```

code JavaScript fos12.js

```
$(function(){
$('#resultat').html('')
$('#maPos').on('click',function(){
  $('#resultat').html('')
  if (navigator.geolocation){
    navigator.geolocation.getCurrentPosition(callBackSucces, callBackError)
  }
  else {
    $('#resultat').html("Désolé, la géolocalisation n'est pas disponible actuellement sur votre téléphone");
  }
});
var callBackSucces=function (position){
  var lat=position.coords.latitude.toFixed(2) ;
  var long=position.coords.longitude.toFixed(2) ;
  $('#resultat').html('<p>Latitude :'+lat+'</p><p>Longitude :'+long+'</p>')
}
var callBackError=function (){
  $('#resultat').html("Désolé, une erreur ne permet pas de vous géolocaliser") ;
}
});
```

code manifest.webapp

```
{
  "name": "Géoloc",
  "description": "affiche coordonnées GPS",
  "launch_path": "/fos12.html",
  "icons": {
    "128": "/images/icon128.png"
  },
}
```

```

        "developer": {
            "name": "LGF",
            "url": "LGF.com"
        },
        "default_locale": "fr",
        "type": "privileged",
        "permissions": {
            "geolocation": {
                "description" : "localisation"
            }
        }
    }
}

```

Il est tout à fait possible d'utiliser les cartes proposées par Google, si cette possibilité vous intéresse, je vous encourage à consulter les API de Google Map (<https://developers.google.com/maps/documentation/javascript/tutorial?hl=fr-FR>) ou encore l'exemple consacré à cette question sur mon site webisn (<http://www.webisn.byethost5.com/geoloc>).

l'interface tactile

Qui dit smartphone dit écran tactile, il faut donc que « l'interface homme-machine » profite de cet écran tactile. jQuery mobile nous propose des événements « consacrés » au tactile (comme tous les autres événements, les événements « tactiles » sont à utiliser avec la méthode « on » (exemple : \$('#toto').on('click',fonctionCallback) ;)

Nous allons ici utiliser 4 « événements tactiles » :

- swipe : balayage (de l'écran) avec le doigt
- swipeleft : balayage avec le doigt vers la gauche
- swiperight : balayage avec le doigt vers la droite
- tap : appui sur l'écran
- taphold : appui prolongé (au moins 750 ms par défaut) sur l'écran (l'action cesse dès que l'appui s'arrête)

À faire vous même

Après vous être familiarisés avec la balise HTML5 canvas (par exemple en vous rendant [ici](#)), étudiez l'exemple suivant :

[code HTML fos13.html](#)

```

<!doctype html>
<html lang="fr">
<head>
    <meta charset="utf-8">
    <meta name="viewport" content="width=device-width, initial-scale=1">
    <title>Firefox OS</title>
    <link rel="stylesheet" href="css/jquery.mobile-1.3.2.min.css">
    <script src="lib/jquery-2.0.3.min.js"></script>
    <script src="lib/jquery.mobile-1.3.2.min.js"></script>
    <script src="javascript/fos13.js"></script>
</head>
<body>
    <div data-role='page'>
        <div data-role='header'><h3>Tactile</h3></div>
    <div id='surface' data-role='content'>
        <canvas id='canvas' width='290' height='300'></canvas>
    </div>
</div>

```

```
</body>
```

```
</html>
```

code JavaScript fos13.js

```
$(function(){
    var canvas = document.getElementById('canvas');
    var ctx = canvas.getContext('2d');
    dessinswleft=function(){
        ctx.fillStyle = "rgb(0,255,0)";
        ctx.fillRect(0,0,290,300);
    }
    $('#surface').on('swipeleft',function(){
        ctx.fillStyle = "rgb(0,255,0)";
        ctx.fillRect(0,0,290,300);
    });
    $('#surface').on('swiperight',function(){
        ctx.fillStyle = "rgb(255,0,0)";
        ctx.fillRect(0,0,290,300);
    });
    $('#surface').on('tap',function(){
        ctx.fillStyle = "rgb(0,0,255)";
        ctx.fillRect(0,0,290,300);
    });
    $('#surface').on('taphold',function(){
        ctx.fillStyle = "rgb(0,0,0)";
        ctx.fillRect(0,0,290,300);
    });
});
```

code manifest.webapp

```
{
    "name": "Tactile",
    "description": "test tactile",
    "launch_path": "/fos13.html",
    "icons": {
        "128": "/images/icon128.png"
    },
    "developer": {
        "name": "LGF",
        "url": "LGF.com"
    },
    "default_locale": "fr",
    "type": "privileged"
}
```

Application multipages

Il est possible avec jQuery mobile de faire des applications multipages.

Toutes les pages peuvent être créées dans un même fichier HTML, la structure est simple :

```
<div data-role='page' id='page_1'>
```



```
...
</div>
<div data-role='page' id='page_2'>
...
</div>
```

Seule la première page sera visible au démarrage de l'application. Les autres pages seront accessibles par l'intermédiaire de liens classiques :

pour passer de la page 1 à la page 2, dans la div « page1 », il suffira de mettre un lien qui pointerà vers la page 2 :

```
<a href='#page_2'>vers la page 2</a>
```

Le passage d'une page à l'autre peut se faire avec des transitions, il suffit d'ajouter l'attribut «data-transition» (voir <http://jquerymobile.com/demos/1.0a4.1/docs/pages/docs-transitions.html>) à la balise <a>

exemple : «vers la page 2»

La méthode «\$.mobile.changePage» permet aussi d'effectuer un changement de page :

exemple : «\$.mobile.changePage(\$('#page_2'))»

Il est possible, là aussi, d'utiliser des transitions :

exemple : «\$.mobile.changePage(\$('#page_2'),{transition:'slide'})»

À faire vous même

Écrire une application comportant 2 pages, le passage d'une page à l'autre se faisant grâce à un « geste tactile » (par exemple « swipeleft » pour passer de la page 1 à la page 2 et swiperight pour passer de la page 2 à la page 1).

Annexe 1

Utiliser un smartphone à la place du simulateur

Certaines fonctionnalités ne sont pas disponibles sur le simulateur. Il est donc nécessaire, pour tester certaines fonctions, d'utiliser un véritable smartphone.

Pour installer des applications sur votre smartphone (de marque ZTE) par l'intermédiaire d'un câble USB, je vous invite à suivre les consignes données ici : <https://hacks.mozilla.org/2013/08/pushing-a-firefox-os-web-app-to-zte-open-phone/>

Une fois « l'installation » terminée, vous devriez voir apparaître un bouton « PUSH » dans le simulateur, ce bouton vous permettra « d'envoyer » votre application vers le smartphone.

Annexe 2

Nous allons voir ici quelques fonctionnalités indisponibles dans le simulateur, mais qui fonctionnent très bien sur un véritable smartphone :

Utilisation de l'accéléromètre

voici un exemple d'utilisation de l'accéléromètre du smartphone :

fichier HTML

```
<!doctype html>
<html lang="fr">
<head>
  <meta charset="utf-8">
  <meta name="viewport" content="width=device-width, initial-scale=1">
  <title>Firefox OS</title>
  <link rel="stylesheet" href="css/jquery.mobile-1.3.2.min.css">
  <script src="lib/jquery-2.0.3.min.js"></script>
  <script src="lib/jquery.mobile-1.3.2.min.js"></script>
  <script src="javascript/fos14.js"></script>
</head>
<body>
  <div data-role='page'>
    <div data-role='header'><h3>Accéléromètre</h3></div>
    <div data-role='content' id='result'></div>
  </div>
</body>
</html>
```

fichier JavaScript

```
$(function(){
    window.addEventListener("deviceorientation", function (event) {
        var absolute = Math.floor(event.absolute);
        var alpha    = Math.floor(event.alpha);
        var beta     = Math.floor(event.beta);
        var gamma    = Math.floor(event.gamma);

        $('#result').html(alpha+' '+beta+' '+gamma)
    });
});
```

fichier manifest.webapp

```
{
  "name": "accéléromètre",
  "description": " test accéléromètre",
  "launch_path": "/fosA1.html",
  "icons": {
    "128": "/images/icon128.png"
  },
  "developer": {
    "name": "LGF",
    "url": "LGF.com"
  },
}
```

```
"default_locale": "fr",  
  "type": "privileged"  
}
```

Analyse du code JavaScript :

```
«window.addEventListener("deviceorientation", function (event) {...});»
```

Nous créons un listener qui réagira au changement d'orientation. Dès que le téléphone changera d'orientation, la fonction de callback sera appelée. Notez que l'on n'utilise pas le classique « on » de jQuery car visiblement l'événement «deviceorientation» n'est pas reconnu par ce dernier.

Intéressons-nous maintenant à la fonction de callback :

```
«function (event) {  
    var alpha    = Math.floor(event.alpha);  
    var beta     = Math.floor(event.beta);  
    var gamma    = Math.floor(event.gamma);  
    $('#result').html(alpha+' '+beta+' '+gamma)  
}»
```

Cette fonction prend en paramètre l'objet event qui contient l'action qui vient d'être déclenchée :

«event.alpha», «event.beta» et «event.gamma» sont les valeurs des angles liés à cette rotation.

Pour en savoir plus, consulter :

https://developer.mozilla.org/fr/docs/WebAPI/Detecting_device_orientation

Lancer des applications depuis votre application : les «Web Activities »

Si par exemple votre application a besoin de prendre une photo, il est (heureusement) inutile d'écrire le code destiné à gérer l'appareil photo, votre application va simplement déléguer cette tâche à l'application native du smartphone (qui se nomme «Camera»). Une fois la photo prise avec l'application «Camera», votre propre application pourra exploiter cette photo comme si c'était elle qui l'avait prise.

Vous trouverez en suivant le lien ci-dessous des explications détaillées sur l'utilisation des «webActivities» :

https://developer.mozilla.org/fr/docs/WebAPI/Web_Activities

Annexe 3

Utilisation d'indexDB

Vous avez appris à stocker « en local » (sans avoir besoin d'une connexion réseau) des données avec le localStorage. Mais ce type de stockage est très limité, si vous voulez stocker des données plus importantes en taille, il s'offre à vous 2 possibilités :

Stocker vos données sur un serveur (par exemple en utilisant une requête Ajax de type POST, les données transitant entre le client et le serveur sous forme de JSON).

Utiliser une « base de données locale ». Firefox OS propose par défaut une base de données relativement simple à utiliser : indexDB

Voici 2 liens qui devraient vous permettre d'apprendre à utiliser indexDB en autonomie :

https://developer.mozilla.org/fr/docs/IndexedDB/Basic_Concepts_Behind_IndexedDB

https://developer.mozilla.org/en-US/docs/IndexedDB/Using_IndexedDB

En cas de difficultés, n'hésitez pas à poser des questions.

Annexe 4

Publication des applications sur le « Firefox Marketplace »

À l'instar de Google et d'Apple, Mozilla propose « un supermarché aux applications » appelé « Firefox Marketplace »

Il est possible d'y trouver des applications gratuites et payantes.

Si vous jugez l'application que vous venez d'écrire digne d'intérêt, vous pouvez la soumettre pour qu'elle soit rendue disponible sur le « Firefox Marketplace » (n'hésitez pas c'est gratuit, ce qui n'est pas le cas chez Apple par exemple ;-)).

Voir le lien ci-dessous

https://developer.mozilla.org/fr/docs/Apps/Publishing/Proposer_une_application