Découverte du Framework jQuery Mobile en autonomie pour le contexte GSB (PARTIE 1)

Description du thème

Propriétés	Description
Intitulé long	Découverte en autonomie du Framework jQuery Mobile avec le contexte GSB dans sa partie gestion des rapports de visite
Formation concernée	BTS SIO option SLAM
Matière	SLAM4 ou phase préparatoire de PPE3 et PPE4
Présentation	Accompagnement dans la découverte de jQuery Mobile ; des sites de références sont proposés afin de développer pas à pas une application à partir du contexte GSB
Notions	• D4.1 - Conception et réalisation d'une solution applicative • D4.2 - Maintenance d'une solution applicative Savoir-faire • Programmer un composant logiciel • Exploiter une bibliothèque de composants • Adapter un composant logiciel • Valider et documenter un composant logiciel • Programmer au sein d'un framework
Prérequis	Les principes du développement web, PHP, SQL
Outils	SGBD MySql, un environnement de développement
Mots-clés	GSB, jQuery Mobile, Ajax
Durée	10 heures
Auteur(es)	Patrice Grand, Gildas Spéno relectrice attentive et éclairée
Version	v 1.0
Date de publication	Mars 2016

Présentation

La société GSB (contexte présenté sur le site du Certa) développe une forte activité commerciale au travers des rencontres faites par ses visiteurs chez les médecins ; ceci donne lieu à des rapports de visite. L'annexe 1 présente ce domaine de gestion ainsi que les nouveaux objectifs de la société.

L'objectif de ce support est de permettre de développer l'application mobile Cross-plateforme évoquée dans l'annexe 1 en utilisant le framework jQuery Mobile (jQM).

Liens indispensables

Nous allons guider la découverte de jQM.

Quelques liens utiles :

- http://jquerymobile.com/
- http://mobile.jquery-fr.com/demos/
- http://www.w3schools.com/jquerymobile/default.asp
- https://www.mobile-tuts.com/item/126-télécharger-le-livre-développer-avec-jquery-mobile : site actuellement en maintenance. L'auteur du livre Charles Edou Nze autorise la publication en annexe de son livre « Développer avec jQuery Mobile ».
- http://demos.jquerymobile.com/1.4.5/ // *les éléments d'un formulaire*

Par ailleurs, *Opera* fournit un émulateur à installer :
http://www.opera.com/fr/developer/mobile-emulator

Il est aussi possible d'utiliser simplement les émulateurs inclus dans les navigateurs :

- Chrome F12 et :

- Edge F12 + émulation

Pour démarrer (première partie)

Nous allons commencer par regarder le support « Développer avec jQuery Mobile » fourni en annexe sous forme de fichier PDF.
Après le sommaire, le document indique les fichiers jQuery à télécharger dans son application (page 6) ; nous allons plutôt utiliser des versions plus récentes des bibliothèques :

```
<link rel="stylesheet" href="http://code.jquery.com/mobile/1.4.5/jquery.mobile-1.4.5.min.css" />
<script src="http://code.jquery.com/jquery-1.11.3.min.js"></script>
<script src="http://code.jquery.com/mobile/1.4.5/jquery.mobile-1.4.5.min.js"></script>
```

Arrêtons-nous sur la page 8 et faisons ce petit exercice.

Travail à faire

1. Créer un nouveau projet dans l'environnement de développement (NetBeans, Eclipse …), puis dans le fichier *index.php* copier le code fourni page 8 (du support « Développer avec jQuery Mobile ») dans un environnement de développement. Vous utiliserez les versions plus récentes de jQuery comme indiqué au-dessus.
2. Exécuter ce code, de préférence en utilisant un émulateur. Il est aussi possible de l'exécuter directement dans le navigateur de votre smartphone si vous disposez d'un réseau Wifi (et l'adresse IP du serveur qui contient le code).

Dans ce code, vous découvrez un nouvel attribut data-role des div ; cet attribut sera utilisé très fréquemment. Ses valeurs possibles (ici page, header, content) permettent de fixer le rôle de la div, ce qui sera très important dans les applications mono-page HTML.

Les pages suivantes concernent jQuery ; vous pouvez :

- les consulter ou les sauter si vous l'avez déjà abordé ;
- ou bien encore lire l'annexe 2 qui présente succinctement mais suffisamment jQuery.

Ainsi, nous allons aller directement à la page 28 (du support « Développer avec jQuery Mobile ») qui présente la constitution d'une page jQuery Mobile.
Le paragraphe 3.1.2 présente l'organisation d'une page (déjà vue plus haut) et le paragraphe 3.1.3 celle d'une 'architecture' mono-page HTML mais multi-pages, au sens jQuery Mobile. Chaque page jQuery Mobile - c'est-à-dire la partie délimitée par * * et * * - apparaitra comme une vue sur le smartphone ou la tablette.
Notez la navigation offerte avec *href = ''#idPage''* (page 32).

Arrêtons-nous à la page 34 et commençons à mettre en œuvre ces premiers éléments pour notre projet GSB.

L'annexe 1 présente trois cas d'utilisation, mais nous traiterons la connexion plus tard ; ébauchons notre application avec un menu à deux entrées.

Nous désirons obtenir l'écran suivant :

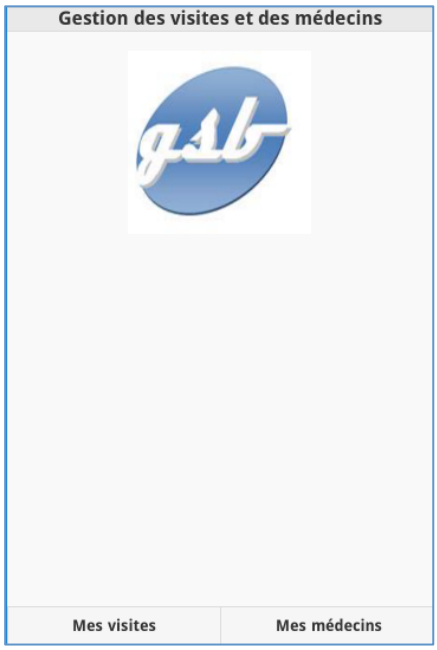


Fig 1

Vous organiserez le code de la manière suivante :

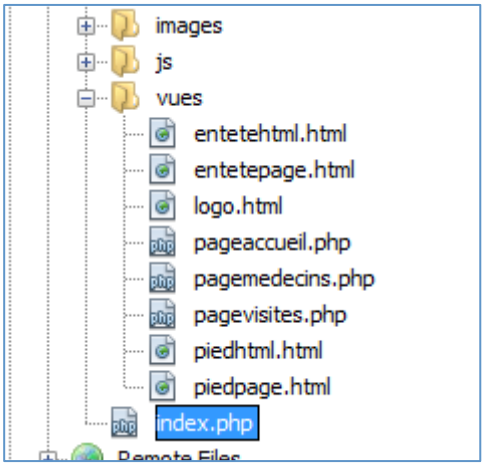


Fig 2

- *entetehtml.html* contient les *meta*, les chargements des bibliothèques jQuery et l'ouverture des balises *html* et *body*.
- *entetepage.html* contient une balise *div data-role = "header"* commune à toutes les pages, avec le *titre* gestion des rapports de visite.
- *logo.html* contient la balise affichant l'image du logo de GSB.

- *pageaccueil.php* est la *page* (au sens jQuery Mobile) qui s'affiche grâce à *div data-role = ''page''* dont l'id sera ''*pageaccueil*', on vous fournit le code qui vous servira de guide pour les autres pages :

```
<div data-role= "page" id = "pageaccueil">
      <?php
            include "vues/entetepage.html";
      ?>
    <div data-role = "content">
       <?php
            include  "vues/logo.html";
        ?>
    </div> <!-- /fin content -->
      <?php
           include "vues/piedpage.html";
      ?>
</div> <!-- /fin page -->
```

Fig 3

- *piedpage.html* va contenir la barre de navigation (abordé plus bas), c'est un composant de type *data-role = ''footer'*
- les pages *pagemedecins.php* et *pagerapports.php* ne contiennent rien, si ce n'est les mêmes *div data-role = ''header''* et *data-role = ''footer''*:
- piedhtml.html ne contient que la fermeture des deux balises *html* et *body*.

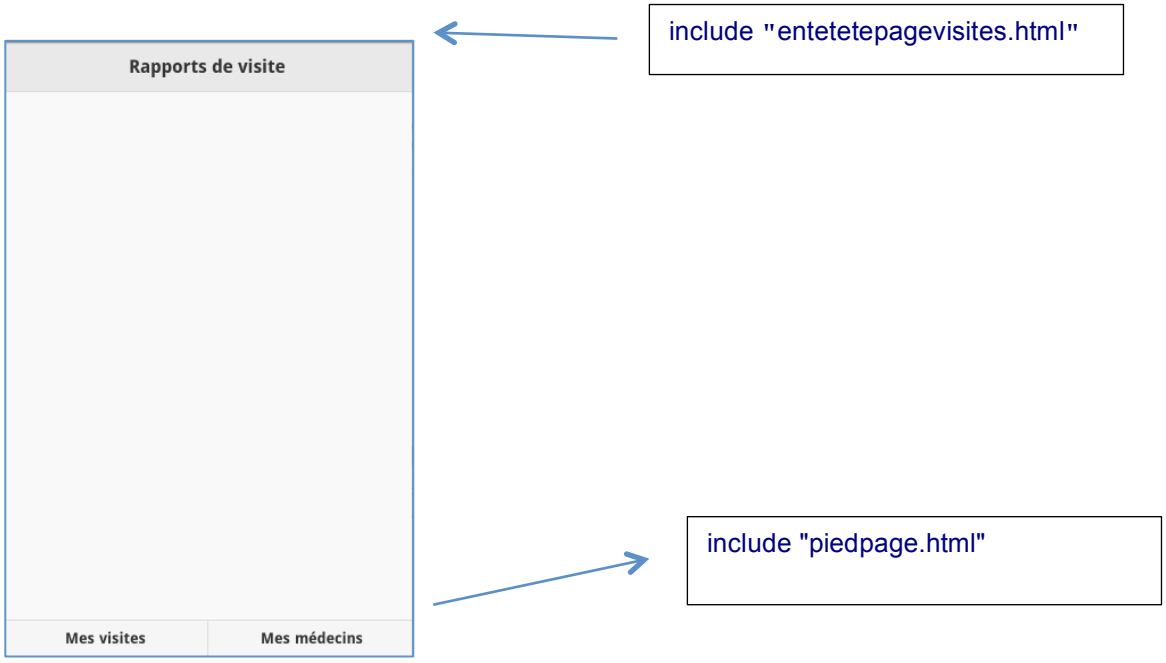


Fig 4

Ainsi, le fichier *index.php* ne contiendra que des appels *include_once* :

```
include_once "vues/entetehtml.html";
include_once "vues/pageaccueil.php";
include_once "vues/pagevisites.php";
include_once "vues/pagemedecins.php";
include_once "vues/piedhtml.html";
```

Fig 5

Quelques remarques :

- prenez soin de respecter des règles de nommage strictes (nom des pages, identifiants des pages, etc.) ;
- nous avons fait le choix de séparer les différentes vues inclues dans chaque page : *entête*, *content* et *footer* ; ce découpage peut bien sûr être différent. Ce choix peut être discuté, mais dans tous les cas il faut rester vigilant sur les utilisations des clauses *include* et *include_once* qui n'ont pas le même effet dans ce type d'architecture mono-page html et multi-page jQuery.

Nous fournissons en annexe 4 le code des fichiers afin de poursuivre.

Pour obtenir la barre de menu du pied de page, vous pouvez consulter un site officiel sur jQuery Mobile, vous y trouverez la syntaxe de tous les composants, ici il s'agit du *widget* ***Navbar*** :
http://demos.jquerymobile.com/1.4.5/

Les boutons de navigation

Il serait bienvenu de rajouter des boutons de navigation ''back'' et "accueil" sur les pages concernées (à partir de la seconde page). Nous aimerions obtenir :

Fig 6

Le paragraphe 3.3 de la page 43 du document présente les boutons et la page 63 présente les différentes icones associées aux boutons. Vous pouvez aussi vous rendre sur le site :
jquerymobile.com/demos/1.1.0-rc.1/docs/buttons/buttons-icons.html

Travail à faire

3. Ajouter un fichier *enteteavecboutons.html* qui contiendra ces deux boutons et sera inséré aux deux secondes pages.

Un peu de style

Lire les pages 71 à 73 qui abordent les problèmes de style et de thème. Ainsi, vous pouvez aussi créer votre propre thème.
Néanmoins, vous pouvez également utiliser un outil en ligne : http://themeroller.jquerymobile.com/
Cet outil permet de récupérer un fichier css à partir d'un environnement de conception visuel dans lequel vous définissez les styles des différents composants jQuery Mobile. Vous pourrez déposer le fichier css dans votre projet, y faire référence dans le fichier *entete.html* et l'utiliser dans le code avec l'instruction :
data-theme="a" (si vous avez créé un thème A) à l'intérieur d'une balise, par exemple :

```
<div data-role="page" id="pageaccueil" data-theme="a" >
```

Travail à faire

4. Définir un thème personnel, comme proposé plus haut.

Un formulaire de saisie

Nous allons ajouter une page de connexion, ce qui sera l'occasion de voir des composants de saisie ; nous désirons obtenir, au lancement de l'application, l'écran suivant :

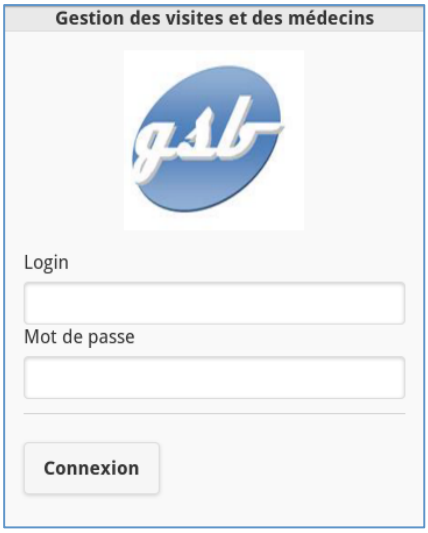


Fig 7

Il faut d'abord ajouter une nouvelle page, *pageconnexion.php* ; ne pas oublier d'ajouter son chargement dans l'index :

```
include_once "vues/entetehtml.html";
include_once "vues/pageconnexion.php";
include_once "vues/pageaccueil.php";
include_once "vues/pagevisites.php";
include_once "vues/pagemedecins.php";
include_once "vues/piedhtml.html";
```

Fig 8

À noter, que **la première page est celle qui s'affiche en premier**, ici la page de connexion.

Nous vous fournissons le code de cette page afin de détailler certains points :

```
1  <div data-role = "page" id = "pageconnexion">
2  <?php
3  include_once "vues/entetepage.html";
4  ?>
5      <div data-role = "content" id = "divconnexion">
6          <?php
7              include_once  "vues/logo.html";
8          ?>
9          <div class= "ui-field-contain">
10             <label for = "login" > Login </label>
11             <input type = "text" name = "login" id = "login" value = "" />
12             <label for = "mdp" > Mot de passe </label>
13             <input type = "password" name = "mdp" id = "mdp" value = "" />
14         </div>
15         <div id = "message" ></div>
16         <p>
17             <a href = "#" data-role = "button" id = "btnconnexion" data-inline="true"  > Connexion </a>
18         </p>
19     </div><!-- /content -->
20 </div><!-- /page -->
```

Fig 9

- À la ligne 9, la div utilise la classe *ui-field-contain* qui remplace le data-role *fieldcontain* obsolète depuis l'écriture du document pdf évoqué page 2.
- jQuery Mobile recommande d'associer un *label* à un *input* (rappel : l'attribut *for* du *label* doit avoir la même valeur que l'attribut *id* de l'*input*).
- Une div *message* (ligne 14) a été ajoutée ; elle sera utilisée en cas d'erreur de connexion.
- La ligne 17 définit le bouton, la valeur # de l'attribut *href* indique un retour sur la page ; l'attribut *data-inline* permet de réduire la dimension du bouton.

Vous avez sans doute remarqué, que ces saisies ne se font pas dans un formulaire (balise *form*) ; en effet, jQM est un grand consommateur d'Ajax …

Un peu d'Ajax

jQuery Mobile n'est pas fait pour faire des allers-retours serveur ; encore moins la technologie Mono-Page (html). L'essentiel des pages est chargé au démarrage et les données venant du serveur sont passées en Ajax.

Nous allons travailler sur le formulaire de connexion.

Principe : lorsque l'utilisateur va valider, jQuery va demander l'exécution d'une fonction (PHP) sur le serveur (sans rechargement de page) ; lorsque la fonction aura terminé son travail elle retournera une information, cette information sera récupérée par jQuery afin de prendre une décision.

Dans notre cas, la décision sera de fournir la page accueil ou de rester sur le formulaire de connexion avec un message d'erreur. Notons, et c'est très important, que les appels Ajax sont asynchrones, c'est-à-dire non bloquants pour l'utilisateur du téléphone.

Sur le site OpenClassrooms, le mécanisme est détaillé ; retenez surtout les versions simplifiées $.get() et $.post().

La page de connexion avec Ajax

Pour mettre en œuvre l'appel Ajax, on va restructurer le projet :

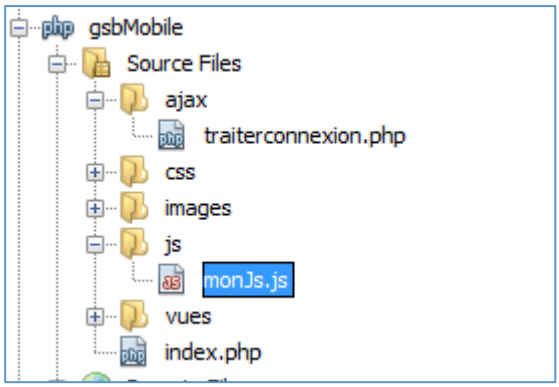


Fig 10

Le répertoire *ajax* contiendra toutes les pages PHP faisant des appels Ajax ; le fichier *monJs.js* centralise tout le code jQuery.

La *page traiterconnexion.php* sera exécutée au clic du bouton de connexion. Comme nous n'avons pas encore de base de données, nous écrivons un prototype simple :

```
1  <?php
2      $mdp = $_REQUEST['mdp'];
3      $login = $_REQUEST['login'];
4      $data = null;
5      $visiteur = array('id'=>"a213",'nom'=>'Durand','login'=>'jdurand','mdp'=>'aaaa');
6      if($visiteur['login'] == $login && $visiteur['mdp'] == $mdp){
7          $data = $visiteur;
8      }
9      echo json_encode($data);
   ?>
```

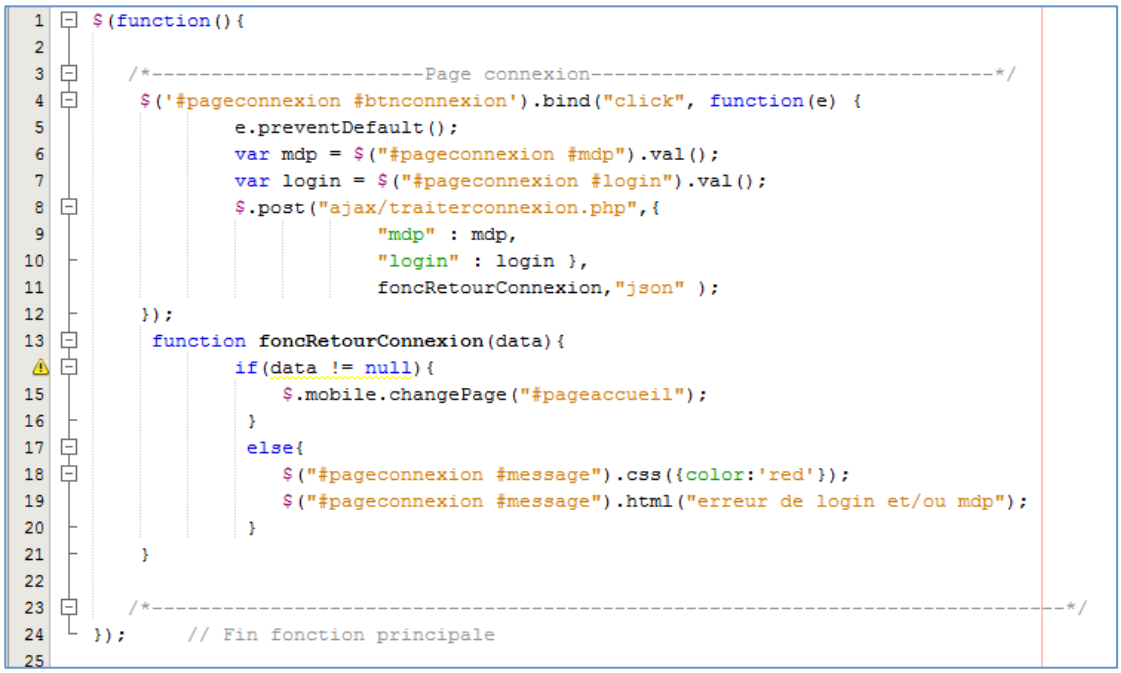
Fig 11

Remarque :

- La valorisation de la variable *$visiteur* est arbitraire, en attendant la base de données.
- À la ligne 9, le flux retourné à l'aide de *echo* est caractéristique des appels Ajax.
- Si le flux retourné est un tableau, il faut obligatoirement *l'encoder* JSON.

L'appel Ajax se fera au moment de la validation du formulaire, donc dans l'événement « click » du bouton.

Le code doit être écrit dans la page *monJs.js* et à l'intérieur de la fonction principale jQuery :



```
1  $(function(){
2
3      /*----------------------Page connexion---------------------------------*/
4      $('#pageconnexion #btnconnexion').bind("click", function(e) {
5              e.preventDefault();
6              var mdp = $("#pageconnexion #mdp").val();
7              var login = $("#pageconnexion #login").val();
8              $.post("ajax/traiterconnexion.php",{
9                      "mdp" : mdp,
10                     "login" : login },
11                     foncRetourConnexion,"json" );
12     });
13     function foncRetourConnexion(data){
14             if(data != null){
15                 $.mobile.changePage("#pageaccueil");
16             }
17             else{
18                 $("#pageconnexion #message").css({color:'red'});
19                 $("#pageconnexion #message").html("erreur de login et/ou mdp");
20             }
21     }
22
23     /*-------------------------------------------------------------------------*/
24 });     // Fin fonction principale
25
```

Fig 12

Remarques :

- ligne 1 : la fonction principale *$(function()* contiendra toutes les fonctions jQuery ;
- ligne 3 : il faut organiser et classer vos fonctions jQuery ;
- à la ligne 4, on utilise le sélecteur *$('#pageconnexion #btnconnexion')* qui désigne l'objet d'id *#btnconnexion* se trouvant dans le conteneur (ici, la page connexion) dont l'id est *#pageconnexion*. Il est fortement recommandé de ne pas référer directement un id mais de lui fournir au préalable la page qui le contient ; ceci évite également des erreurs de nommage entrainant des bugs difficilement décelables.
- à la ligne 4, on *abonne* le bouton à un événement « click » grâce à la méthode *bind*. On pouvait utiliser une syntaxe plus compacte mais moins claire :
  $('#pageconnexion #btnconnexion').click(function(e){…
- à la ligne 4, on utilise une fonction anonyme (function(e)) .

Voici la même chose avec une fonction nommée :

```
$('#pageconnexion #btnconnexion').bind("click",maFonction);

function maFonction(e){
        e.preventDefault();
        var mdp = $("#pageconnexion #mdp").val();
        var login = $("#pageconnexion #login").val();
        $.post("ajax/traiterconnexion.php",{
                    "mdp" : mdp,
                    "login" : login },
                    foncRetourConnexion,"json" );

}
```

Fig 13

- plus compact et plus performant car l'interpréteur jQuery dispose immédiatement du code à exécuter dans le cas de la fonction anonyme ;
- à la ligne 5, la méthode *preventDefault*, annule le fonctionnement par défaut du bouton, ici le chargement d'une page ;
- les lignes 6 et 7 récupèrent les données saisies dans les zones d'id ''mdp'' et ''login'' ;
- la ligne 8 lance la méthode Ajax $.post. Cette méthode prend (ici) 4 arguments : la page php à exécuter côté serveur, la liste des données et leur valeur à transmettre, le nom de la fonction qui s'exécutera lorsque des données seront retournées par le serveur et le format d'encodage ;
- à la ligne 13, justement voici cette fonction ; elle prend comme argument (nommé *data*, par tradition) la donnée qui sera retournée par le serveur à l'aide de la page *traiterconnexion.php* qui contient à sa dernière ligne une instruction *echo*.
- le reste est trivial.

Travail à faire

5. Reproduire ce code. Tester avec différentes valeurs de login et mdp. Voir l'annexe 3 qui rappelle l'utilisation de *Firebug*.

Le contrôle panel

Ce contrôle est très courant dans les applications mobiles, il permet de faire apparaître un menu à partir d'un bouton dédié ; la page ouverte couvre alors la moitié de l'écran et peut apparaître à droite ou à gauche.

Le site officiel incontournable http://api.jquerymobile.com/category/widgets/ présente le contrôle panel.

Nous désirons obtenir lors du chargement de la page de gestion des rapports de visite un écran qui ressemble à ceci :

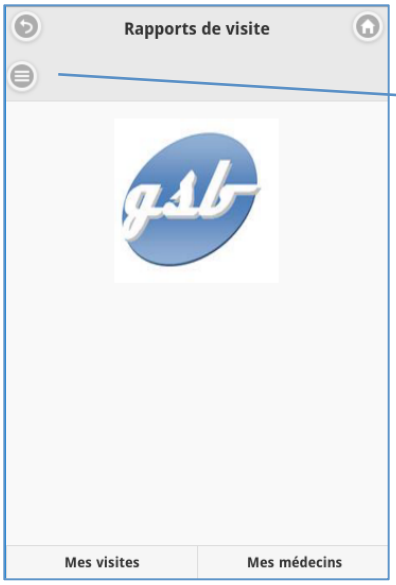


Fig 14

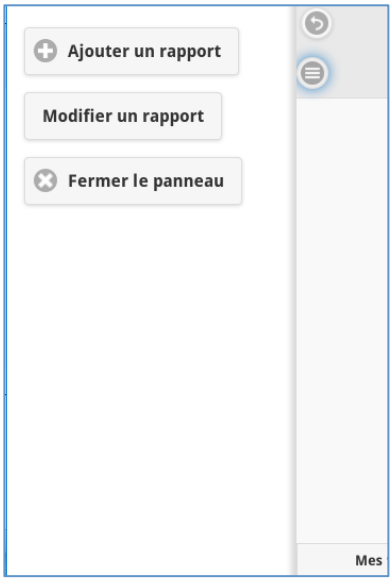


Fig 15

Les deux pages « ajouter un rapport » et « modifier un rapport » ne contiendront qu'un titre dans un premier temps.

Travail à faire

6. En étudiant les règles de mise en œuvre du contrôle Panel (sur le site officiel de jQuery Mobile, cf plus haut), ajouter ce contrôle ainsi que les deux pages évoquées. Pour la suite, nommer la page de mise à jour « *pagechoisirrapportamodifier* ».

La gestion des médecins est décrite dans le second cas d'utilisation (annexe 1). On désire démarrer sur une page qui ressemble (dans un premier temps) à ceci :

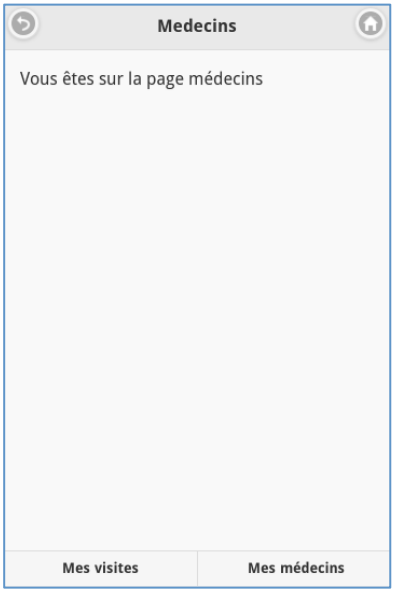


Fig 16

Travail à faire

7. Ajouter cette nouvelle page à l'application.

Ceci termine cette première partie de découverte.

Le corrigé de cette partie est disponible dans le répertoire *CorrigegsbMobilePartie1*.

Annexe 1 : présentation du contexte

PROJET APPLI-CR-MOBILE

Cahier des charges et description des données

Présentation du projet

L'activité à gérer

L'activité commerciale d'un laboratoire pharmaceutique est principalement réalisée par les visiteurs médicaux. En effet, un médicament remboursé par la sécurité sociale n'est jamais vendu directement au consommateur mais prescrit au patient par son médecin.

Toute communication publicitaire sur les médicaments remboursés est d'ailleurs interdite par la loi. Il est donc important, pour l'industrie pharmaceutique, de promouvoir ses produits directement auprès des praticiens.

Les Visiteurs Médicaux

L'activité des visiteurs médicaux consiste à visiter régulièrement les médecins généralistes, spécialistes, les services hospitaliers ainsi que les infirmiers et pharmaciens pour les tenir au courant de l'intérêt de leurs produits et des nouveautés du laboratoire.

Chaque visiteur dispose d'un portefeuille de praticiens, de sorte que le même médecin ne reçoit jamais deux visites différentes du même laboratoire.

Comme tous les commerciaux, ils travaillent par objectifs définis par la hiérarchie et reçoivent en conséquence diverses primes et avantages.

Pour affiner la définition des objectifs et l'attribution des budgets, il sera nécessaire d'informatiser les comptes rendus de visite.

L'activité des visiteurs

L'activité est composée principalement de **visites** : réalisées auprès d'un praticien (médecin dans son cabinet, à l'hôpital, pharmacien, chef de clinique…), on souhaite en connaître la date, le motif (6 motifs sont fixés au préalable), et savoir, pour chaque visite, les médicaments présentés et le nombre d'échantillons offerts. Le bilan fourni par le visiteur (le médecin a paru convaincu ou pas, une autre visite a été planifiée…) devra aussi être enregistré.

Les produits

Les produits distribués par le laboratoire sont des médicaments : ils sont identifiés par un numéro de produit (dépôt légal) qui correspond à un nom commercial (ce nom étant utilisé par les visiteurs et les médecins).

Comme tout médicament, un produit a des effets thérapeutiques et des contre-indications.

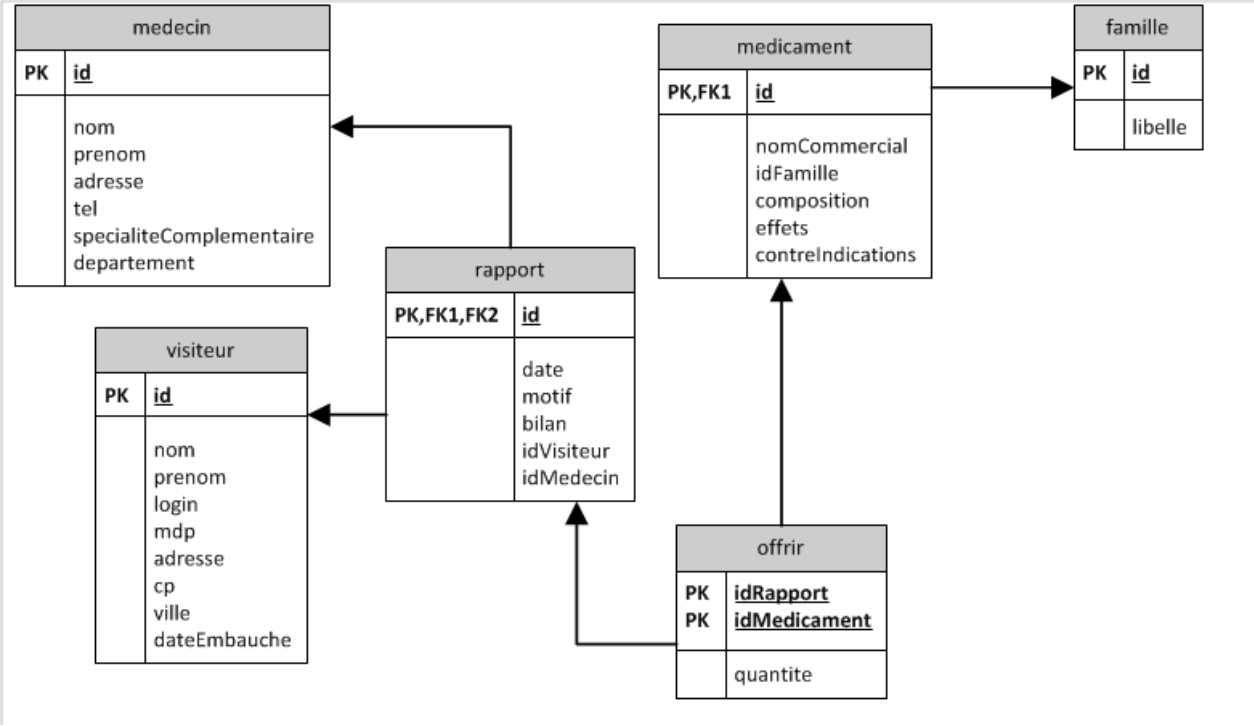
On connait sa composition (liste des composants et quantité) et les interactions qu'il peut avoir avec d'autres médicaments (éléments nécessaires à la présentation aux médecins).

La posologie (quantité périodique par type d'individu : adulte, jeune adulte, enfant, jeune enfant ou nourrisson) dépend de la présentation et du dosage.
Un produit relève d'une famille (antihistaminique, antidépresseur, antibiotique, …).
Lors d'une visite auprès d'un médecin, un visiteur présente un ou plusieurs produits pour lesquels il pourra laisser des échantillons.

Les médecins

Les médecins sont le cœur de cible des laboratoires. Aussi font-ils l'objet d'une attention toute particulière.
Pour tenir à jour leurs informations, les laboratoires achètent des fichiers à des organismes spécialisés qui donnent, les diverses informations d'état civil et la spécialité complémentaire.

La modélisation complète des données est présentée ci-dessous :



L’application à réaliser

L’entreprise envisage de permettre aux visiteurs de gérer ses visites par l’intermédiaire de son smartphone. L’application devrait permettre de réaliser les cas d’utilisation suivants :

Description textuelle des cas d’utilisation

Cas : gérer les rapports de visite

Scénario classique

1) Le visiteur demande à créer un nouveau rapport de visite
2) Le système retourne un formulaire avec la liste des médecins et des champs de saisie
3) Le visiteur sélectionne un médecin à partir de son début de nom, sélectionne la date et remplit les différents champs, sélectionne les médicaments et les quantités offertes et valide
4) Le système enregistre le rapport

Scénario étendu : modification d’un rapport

5) Le visiteur demande à modifier un rapport
6) Le système retourne un formulaire avec une date à sélectionner
7) Le visiteur sélectionne la date
8) Le système retourne les rapports *que le visiteur a effectués à cette date*
9) Le visiteur sélectionne un rapport de visite
10) Le système retourne les informations déjà saisies *concernant le motif et le bilan*
11) Le visiteur modifie les informations

12) Le système enregistre les modifications

Scénario alternatif

4.1) Des champs ne sont pas remplis, le système en informe le visiteur, retour à 3

Cas : gérer les médecins

Scénario classique

1) Le visiteur demande à voir les informations concernant un médecin
2) Le système retourne un formulaire avec un champ de recherche du médecin
3) Le visiteur sélectionne un médecin à partir de son début de nom et valide
4) Le système retourne les informations personnelles concernant ce médecin

Scénario étendu :

5) Le visiteur clique sur le numéro de téléphone du médecin
6) Le système compose le numéro
7) Le visiteur demande à voir tous les anciens rapports de visite concernant ce médecin
8) Le système retourne tous ses rapports
9) Le visiteur demande à modifier certains champs concernant des informations du médecin
10) Le système enregistre ces modifications.

Annexe 2 : présentation succincte de jQuery

Ce mini-framework se propose de simplifier l'utilisation de javascript et présente une organisation du code qui rompt avec celle de javascript.

Tous les appels jQuery sont faits dans une seule fonction et non plus éparpillés au milieu de code HTML. Tout le code est à l'intérieur de cette fonction qui débute par :

$(function(){ et se termine par });

Cette fonction est dite anonyme –sans nom- ; nous avons la garantie que le code qu'elle contient est exécuté après la construction du DOM.

Il peut y avoir des instructions, des déclarations de variables, mais le plus souvent il s'agit de fonctions exécutées sur des événements.

Comme jQuery centralise toutes ses instructions, il faut un mécanisme pour indiquer que nous désirons faire exécuter du code sur un événement associé à une ou des éléments particuliers du DOM. C'est le rôle du sélecteur jQuery qui va désigner un ou des éléments du DOM.

Trois types de sélecteurs sont prévus :

- À **partir de l'attribut id** de l'élément HTML :

  $(«#idDiv1 ») sélectionne la div d'id *idDiv1*

- À **partir du type d'élément** :

  $(«li ») sélectionne tous les éléments de type li ; ici, il est probable que plusieurs éléments seront sélectionnés
- À **partir de la classe** :

  $(« :cla1 ») sélectionne tous les éléments de classe cla1

Il est possible d'utiliser des opérateurs pour préciser les objets atteints par le sélecteur :

- $(« #idDiv1 >li ») sélectionne tous les éléments li, enfants de la div d'id idDiv1
- $(« input,label ») sélectionne les input et les labels

Nous ne fournirons pas ici la liste de tous les sélecteurs possibles avec leurs syntaxes, de nombreux sites le font de manière bien plus systématique. Ce qu'il faut retenir c'est l'importance de bien définir le sélecteur avant de lancer des méthodes ou ajouter un gestionnaire d'événement. De nombreuses méthodes appliquées à ces objets jQuery définis par leur sélecteur permettent de rendre de nombreux services, modifier le style (<mon sélecteur>.css(« description »), mettre un flux html (<mon sélecteur>.html(« code html »), retourner des valeurs (<mon sélecteur>.html(), etc.
On peut ajouter un gestionnaire d'événement en faisant :

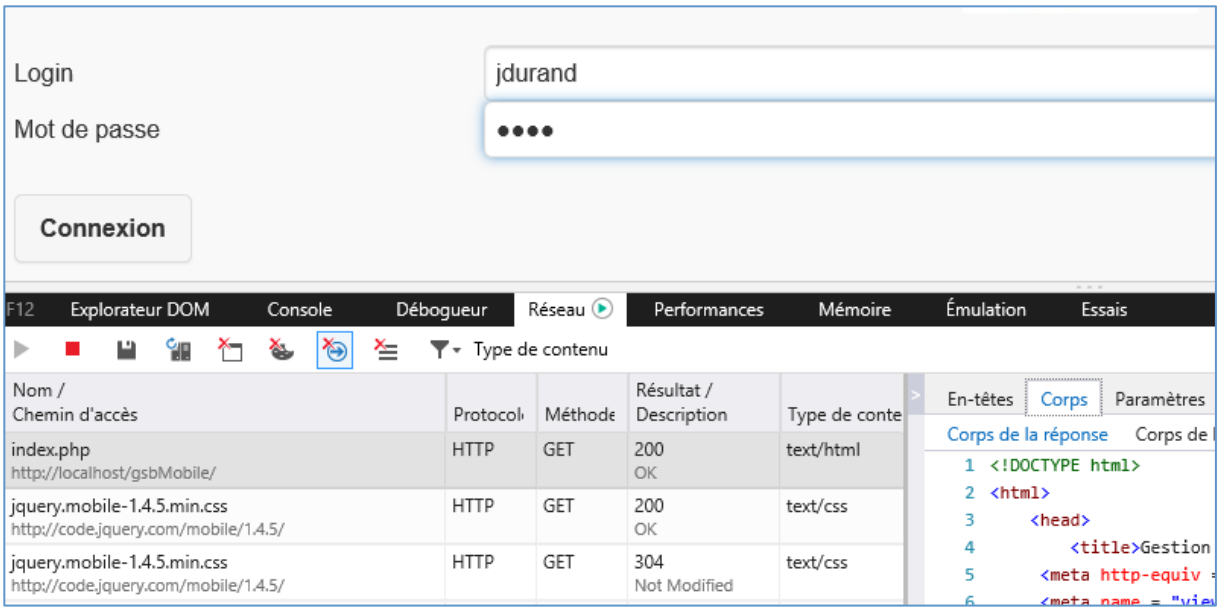
<mon sélecteur>.<nom événement>(function(){ code à exécuter }) ; // fonction anonyme

Tous les objets correspondant au sélecteur s'abonneront à la fonction anonyme.
Enfin, jQuery propose une mise en œuvre d'Ajax relativement simple.

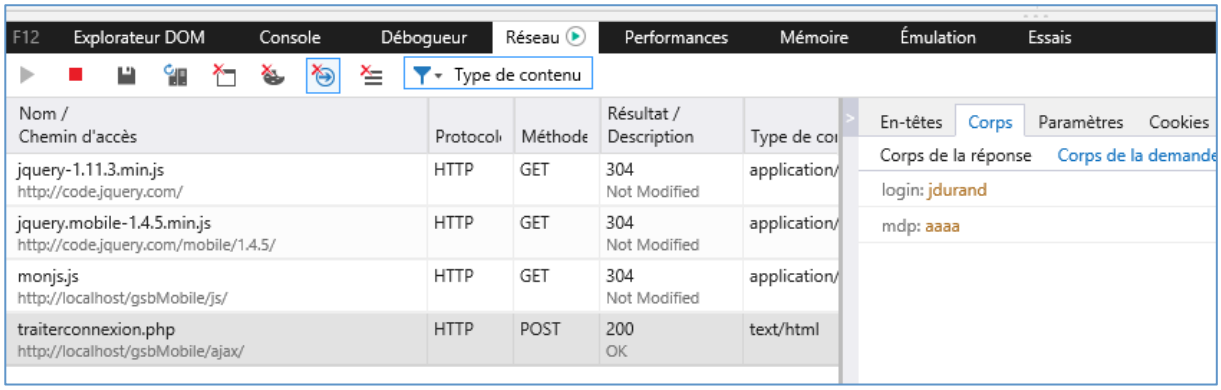
Annexe 3 : débugger son application avec FireBug

Nous utilisons Firebug, installé par défaut sur la plupart des navigateurs.
Nous allons détailler son utilisation sur le formulaire de connexion.

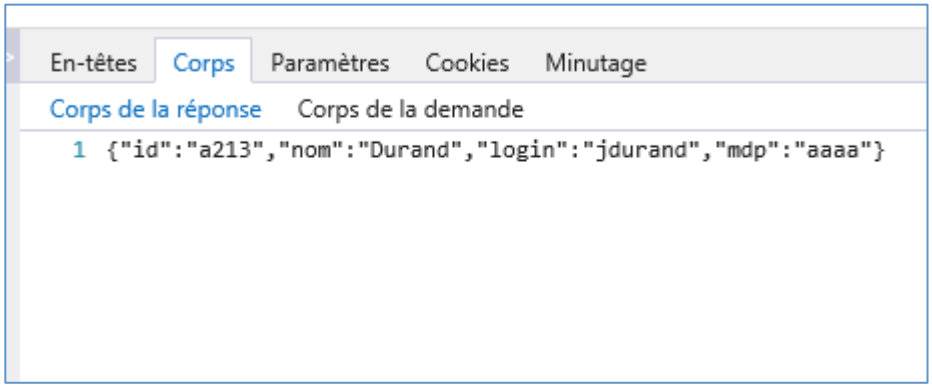
1- Lancer l'application jusqu'à la page à tester, ici *avant* d'avoir validé.
2- Appuyer sur la touche *F12*. Vous avez un écran qui ressemble à ceci (ici Edge), si vous avez sélectionné l'onglet *Réseau* :



3- Valider la connexion, vous avez un écran qui ressemble à ceci : si vous avez sélectionné à gauche l'onglet *Réseau* et *traiterconnexion.php* / à droite Corps de la demande.



Nous voyons que la page a envoyé les deux données ; si nous sélectionnons maintenant *Corps de la réponse*, nous obtenons :



Ainsi, la fonction Ajax a retourné (dans le paramètre *data*) ce tableau JSON ; il sera traité par la fonction jQuery *foncRetourConnexion*.

Annexe 4 : les premiers fichiers

Entetehtml.html

```
<!DOCTYPE html>
<html>
    <head>
        <title>Gestion des rapports de visite beta 1</title>
        <meta http-equiv = "Content-Type" content = "text/html; charset=UTF-8">
        <meta name = "viewport" content = "width=device-width, initial-scale=1.0">
        <link rel = "stylesheet" href = "http://code.jquery.com/mobile/1.4.5/jquery.
        <script src = "http://code.jquery.com/jquery-1.11.3.min.js"></script>
        <script src = "http://code.jquery.com/mobile/1.4.5/jquery.mobile-1.4.5.min.j
        <script src="./js/monjs.js"></script>
    </head>
  <body>
```

entetepage.html

```
<div data-role = "header" >
    <center> Gestion des visites et des médecins </center>
</div><!-- /header -->
```

pagevisites.php

```
<div data-role = "page" id = "pagemedecins">
<?php
    include "vues/entetepagemedecins.html";
?>
   <div data-role = "content">
      Vous êtes sur la page médecins
   </div> <!-- /fin content -->
<?php
    include "vues/piedpage.html";
?>
</div><!-- /fin page -->
```

pagemedecins.php

```
<div data-role = "page" id = "pagevisites">
<?php
    include "vues/entetepagevisites.html";
?>
 <div data-role = "content">
      <?php
          include  "vues/logo.html";
        ?>
    </div>
<?php
    include "vues/piedpage.html";
?>
</div><!-- /fin page -->
```

logo.html

```
<center>
    <img src = "images/logoGSB.jpg" alt = "logo" width = "155" height = "155"/>
</center>
```

piedpage.html

```
<div data-role = "footer"  data-position = "fixed" data-id= "pied">

</div><!-- /footer -->
```

piedhtml.html

```
    </body>
</html>
```