Découverte du Framework jQuery Mobile en autonomie pour le contexte GSB (PARTIE 2)

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

Poursuivons notre découverte de jQuery Mobile ; vous avez compris l'organisation des pages dans ce type d'application nommée SPA (Single Page Application), vous avez découvert les premiers *widgets*, vous avez mené une requête Ajax. Nous allons, lors de cette deuxième partie, intégrer la base de données et utiliser de nouveaux *widgets*.

Query Mobile avec une base de données

Les tables

Le modèle logique est donné dans l'annexe 1 de la partie 1 ; le script SQL est fourni dans le fichier ***gsbrapports.sql***.

Travail à faire

8. Installer les tables sur un serveur MySQL.

Nous allons développer la couche d'accès aux données.

Commençons par créer un répertoire *data* ainsi qu'un fichier qui contiendra notre classe *pdo* :

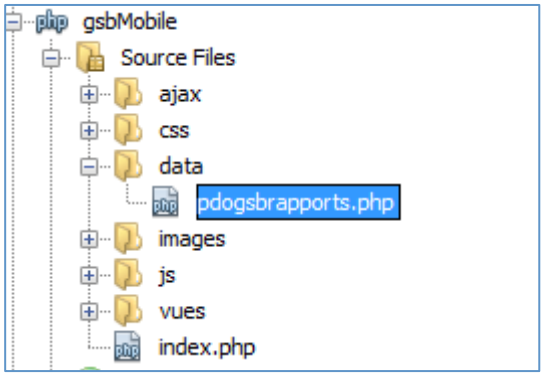

Fig 17

Nous utiliserons :

- une classe qui reprend l'architecture de la classe *pdo* du contexte GSB, elle est disponible dans le fichier *pdogsbrapportV0.php*
- des *requêtes préparées* qu'il faut privilégier pour tout accès aux données.

La connexion du visiteur avec la base de données

Nous allons faire une première expérimentation de l'architecture Ajax avec accès à une base de données ; soyez très attentifs ☺

1) Côté pdo

Le code est classique ; la classe *pdoGsbRapports* est fournie. À chaque fois que vous aurez besoin d'accéder à la base de données, il faudra intégrer ce fichier (bibliothèque de classe dans votre code). La méthode *getLeVisiteur* retourne l'id, le nom et le prénom du visiteur concerné par un login et un mot de passe ou NULL.

2) Côté Ajax

Le code initial est un peu modifié (il n'y avait pas d'accès aux données) :

```php
<?php
    session_start();
    require_once '../data/pdogsbrapports.php';
    $mdp = $_REQUEST['mdp'];
    $login = $_REQUEST['login'];
    $pdo = PdoGsbRapports::getPdo();
    $visiteur = $pdo->getLeVisiteur($login,$mdp);
    if($visiteur != NULL){
        $_SESSION['visiteur'] = $visiteur ;
        $_SESSION['visiteur']['mdp'] = $mdp;
        $_SESSION['visiteur']['login'] = $login;
    }
    echo json_encode($visiteur);
?>
```

Fig 18

Remarques :

- À la ligne 2, nous installons le mode session, afin de stocker le visiteur (ligne 9). Ainsi, tous les appels Ajax pourront récupérer le visiteur*, à la seule condition bien sûr, d'activer le mode session,* cf point suivant.
- La ligne 3 inclut le fichier de la classe ; à ce propos, il faut bien comprendre que l'appel Ajax est une requête http « classique », donc en mode déconnectée ; ainsi il faudra pour chaque page PHP répondant à un appel Ajax « reconstruire » son environnement applicatif par exemple inclure une bibliothèque (comme ici), ou créer un objet *pdo* (comme ici) ou bien encore se mettre en « mode session » avec *session_start()*.
- La ligne 6 crée un objet *pdo*, cf le point précédent.
- La ligne 7 est classique et appelle la méthode de la classe.
- Les lignes 9, 10 et 11 ajoutent à la session les login et mot de passe ; en effet il est conseillé de ne pas faire circuler dans le sens serveur/client ces données sensibles. Le login et mot de passe seront utilisés plus loin pour sécuriser des demandes de mises à jour de données.

3) Côté jQuery

Pas de modification par rapport à la version sans base de données.

La mise à jour des rapports

Nous continuons par la mise à jour d'un rapport existant ; le cas d'utilisation « gérer les rapports de visite » dans son scénario étendu (annexe 1) décrit le fonctionnement attendu. Nous souhaiterions avoir des écrans qui ressemblent à ceci :

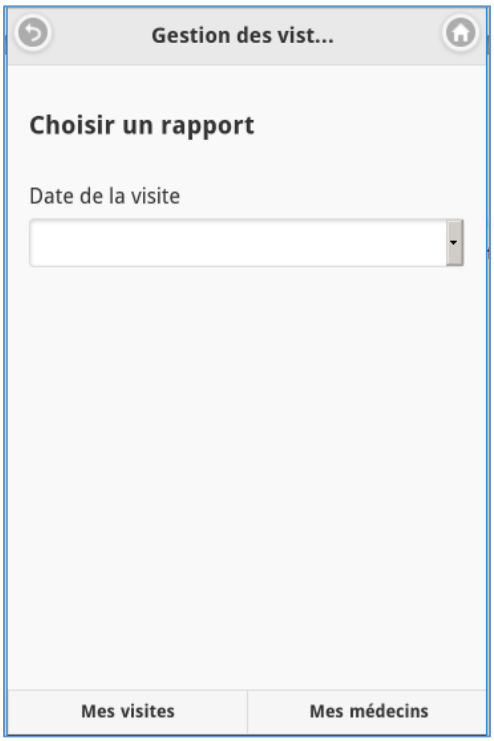

Fig 19

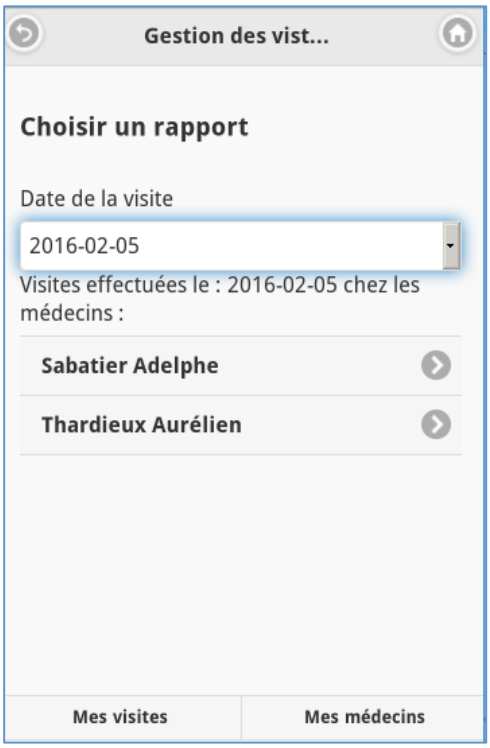

Fig 20

Les médecins visités ce jour apparaissent lorsque la date a été sélectionnée.

Lorsque le visiteur sélectionne un médecin visité, l'écran fait apparaître les champs à modifier éventuellement :

Fig 21

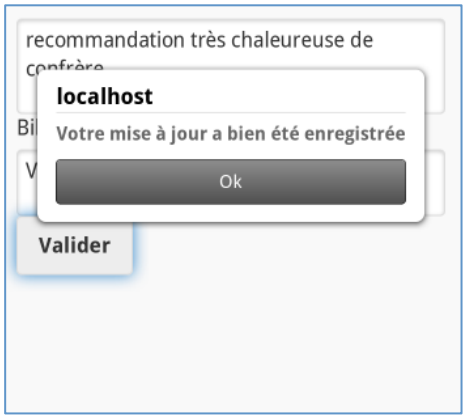

Fig 22

À la validation, les informations sont enregistrées en base, un message en alerte le visiteur.

Nous allons avancer ensemble.

Vos pages vues doivent contenir au moins :

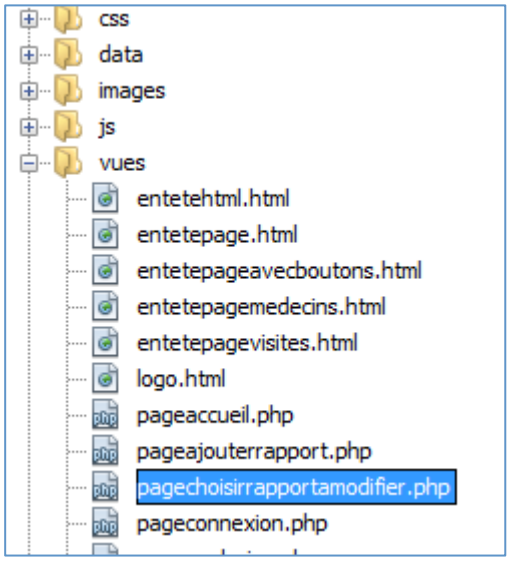

Fig 23

1) La page *page*choisirrapportamodifier

1.a Côté HTML

Elle contient la zone de saisie pour la date et la liste des médecins, construite au moment où on sélectionne la date (Fig 19 et 20). Cette page est simple à écrire, elle fait appel à une *div ui-field-content* (voir Fig 9) contenant :

- la zone de saisie de type *date (*renseigner l'attribut id de cet élément),
- et la zone juste au-dessous de type label.

Concernant la liste, vous allez découvrir la listview. Allons consulter notre site préféré http://api.jquerymobile.com/category/widgets/.
Dans cette page, en plus des entêtes classiques (data-role) il n'y aura que la balises ul. Les balises li seront insérées en jQuery.

Travail à faire

9. Créer cette page.

1.b Côté jQuery

C'est un peu plus délicat… Il faut utiliser l'événement *change* de la zone de saisie de la date. Prenez soin de préfixer dans le sélecteur l'id de l'élément HTML par sa page conteneur :
$(#idPage #idElementHTML).bind(…)

Travail à faire
10. Écrire le code jQuery de manière à faire seulement apparaître ce qui est présenté figure 24.

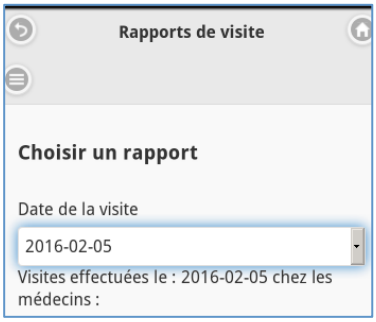

Fig 24

Dans la suite de l'événement *change*, nous allons faire un appel Ajax (un POST) qui pointera sur une page PHP avec un argument, la date choisie. La fonction de retour jQuery (que nous avons appelée *foncRetourListeRapports*) ne fera rien, pour l'instant ; elle devra néanmoins être présente.
Nous avons appelé le fichier PHP *traiterlesvisitesaunedate.php*… ne soyez pas avare de nom très significatifs ☺

Le code de cette nouvelle page PHP est relativement trivial, il :

- ouvre le mode session ;
- intègre la bibliothèque de la classe ;
- récupère le champ *date* passé en Ajax ;
- récupère le login et le mot de de passe du visiteur (on pouvait utiliser son *id*) de la session ;
- crée un objet *pdo ;*
- demande à cet objet d'exécuter la méthode *getLesVisitesUneDate* (pas encore écrite) *$lesVisites = $pdo->getLesVisitesUneDate($login, $mdp, $date)* ;
- retourne (*echo*) *$lesVisites* en JSON.

 **Pour tester efficacement cette page, vous devez disposer de plusieurs visites pour un visiteur le même jour, dans la base de données ; rajouter 2/3 visites dans la base.**

Travail à faire

11. Écrire le code du fichier *traiterlesvisitesaunedate.php*
12. Écrire aussi la méthode *getLesVisitesUneDate* (qui doit retourner id du rapport, le nom et le prénom du médecin).

Il faudra tester tout cela en utilisant *FireBug* (annexe 3 de la partie 1) ; ceci est **indispensable**.

Conseils :

Concernant la méthode *getLesVisitesUneDate*, prenez l'habitude de ne retourner que ce qui est nécessaire, ainsi n'écrivez pas systématiquement select *.

Lorsque vous avez une requête de jointure, vous avez plusieurs champs id ; prenez l'habitude de créer à cette occasion un alias (id as idRapport) ; vous communiquez ainsi à son « utilisateur » une signification au champ retourné.

Bien, nous supposons que l'appel Ajax fonctionne. Passons à ce qui est le plus délicat : notre objectif est de remplir les éléments ul (qui sont écrits en dur dans la page) par des éléments li, créés dynamiquement par jQuery.

CE N'EST PAS TRES COMPLIQUE SI VOUS ETES METHODIQUE. Dans la fonction *foncRetourListeRapports,* il faut :

- faire une boucle for sur le nombre de rapports (*lesVisites.length*, si *lesVisites* est l'argument de votre fonction) ;
- récupérer le rapport courant *var unRapport = lesVisites[i]* ;
- récupérer chaque champ, l'id du rapport (unRapport['id']), le nom du médecin et son prénom ;
- écrire une chaîne de texte qui commence par *var html = "<li id =* …

  La syntaxe d'une balise li étant la suivante :

  Chaine_a_afficher

  Dans notre contexte, chaque balise <li> devra afficher les noms et prénoms des médecins (sous forme d'hyperliens) :

  

  L'attribut *id* de cette balise <li> aura comme valeur l'id du rapport, nécessaire pour faire apparaître un rapport particulier lorsqu'on la sélectionnera.
- nous ajoutons cette chaîne à notre liste *ul*

  $("#pagechoisirrapportamodifier #listerapports").append(html);
- Et nous fermons la boucle, et c'est tout, enfin presque…

Travail à faire

12. Écrire ce code.

Si vous voyez quelque chose qui ressemble à une liste, c'est très bien. Nous allons en profiter pour aborder un aspect lié à jQuery Mobile.

Vous avez dû constater, lorsque vous ouvrez *FireBug* sur le code HTML, que jQuery Mobile ajoute du code à celui que vous avez écrit ; il rajoute les classes *css* pour figurer les éléments HTML dans un *look* mobile (c'est le rôle de l'attribut *data-role*). Si votre code est écrit *en dur*, il n'y a pas de problème ; par contre, si votre code provient de jQuery, donc après le chargement de la page, il y a un problème. C'est le cas de nos balises *li* qui sont créées après le chargement de la page.
Il faut donc demander un rafraichissement de la page.
Ainsi, pour donner à votre liste le *look* mobile, vous devez terminer votre code par :

```
$("#pagechoisirrapportamodifier #listerapports").listview('refresh');
```

Notons, que ce problème, est source de bien des soucis car tous les navigateurs ne réagissent pas de la même manière, tout comme les différentes versions de jQuery Mobile. Il faut souvent s'armer de patience et visiter les forums qui traitent du problème qui malheureusement n'ont pas toujours la même réponse selon l'élément HTML à rafraichir ou construire ou reconstruire… et le navigateur.
Un dernier petit souci, cohérent celui-ci : si l'on refait deux fois une recherche, la liste n'est pas vidée. La solution est simple, il faut la vider au début de la fonction :

```
$("#pagechoisirrapportamodifier #listerapports").empty();
```

2) La page *pagerapportamodifier*, fig 21 et 22

Il faut ajouter cette page à nos vues.

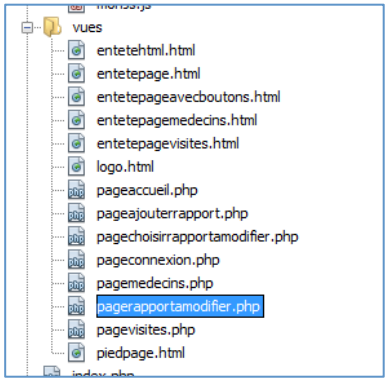

Fig 25

Dans cette vue PHP, le code correspond à la figure 21, les zones de texte sont de type *textarea*.

Travail à faire

13. Créer cette page.

Côté jQuery

Le code est à écrire sur l'événement "click" ; on vous donne l'entame :

```
$("#pagechoisirrapportamodifier").on("click","li", function(e) {
        var idRapport = $(this).attr("id");
        window.idRapport = idRapport;
```

Fig 26

Remarques :

- La première ligne indique que cet événement *click* est ajouté à toutes les balises *li* de la page indiquée. La seconde ligne récupère la valeur courante -*$(this)*- de l'attribut *id* (le rapport sélectionné pour modification) ;
- La troisième ligne stocke cette valeur dans une variable accessible en dehors de la fonction (portée script).

Pour la suite, il faut :

- récupérer le contenu de type *text* de la balise (c'est le nom et le prénom du médecin) grâce à cette instruction :
  var medecin = $(this).text();
- le mettre dans une variable *window* (cf figure 26) ;
- lancer la requête Ajax (POST) qui prendra l'*idRapport* comme argument, qui pointera sur un fichier PHP (pas encore écrit) *traiterchoixrapport.php*. La fonction de retour sera appelée *foncRetourChoixRapport*.

Travail à faire

14. Terminer cette fonction JQuery. Attendre pour tester.

15. Écrire la page PHP *traiterchoixrapport.php* qui récupère l'id du rapport et retourne (*echo*) les champs du rapport à l'aide d'une nouvelle méthode à écrire dans la classe pdo *getLeRapport($idRapport)*.

16. Tester maintenant avec FireBug.

17. Il ne reste plus qu'à écrire le code de la fonction *foncRetourChoixRapport* ; elle commence par un appel de la page du rapport à modifier :

```
$.mobile.changePage("#pagerapportamodifier");
```

La mise à jour du rapport

La figure 22 propose le fonctionnement attendu ; enregistrement en base et apparition d'une boite de dialogue.

Le code ici, reprend ce que nous avons déjà vu :

- côté jQuery, l'ajout d'un événement click sur le bouton valider. Son code commence à récupérer l'id du rapport, le contenu des champs *motif* et *bilan*, poursuit par une fonction Ajax qui prend ces trois arguments ;
- côté pdo, une méthode qui permet de mettre à jour la base *majRapport($idRapport,$bilan,$motif).*
- côté Ajax, la récupération des trois arguments et l'appel de la méthode du pdo ;
- dans la fonction de rappel, juste le message de confirmation et le chargement de la page de choix du rapport à modifier.

Remarque :

La fonction pdo *majRapport* retourne une valeur qui indique si *la mise à jour de la* base s'est bien passée. C'est cette valeur (1) qui sera testée dans la fonction de retour pour afficher la boite de dialogue.

Travail à faire

18. Écrire le code correspondant à ce qui est décrit plus haut.

Ceci termine la deuxième partie, disponible dans le dossier *CorrigegsbMobilePartie2*.