Découverte du Framework jQuery Mobile en autonomie pour le contexte GSB (PARTIE 3)

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

Voici la troisième partie qui vous permettra de consolider vos connaissances, de découvrir deux nouveaux widgets bien utiles, l'un permet de gérer les recherches dans une liste et l'autre présente des données sous forme de tableau façon jQuery Mobile.

Gestion des médecins

Nous poursuivons cette découverte approfondie de jQuery Mobile par le **menu de gestion des médecins** ; en effet, nous terminerons par l'ajout d'un rapport qui est la partie la plus complexe.

Sélection d'un médecin

Nous voulons obtenir un écran qui ressemble à ceci, lorsque l'utilisateur sélectionne « Mes médecins » :

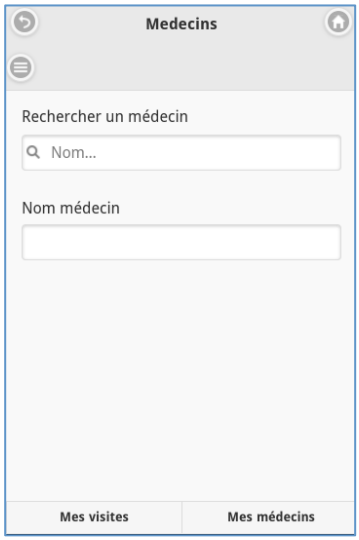


Fig 27

Comme indiqué dans le cas d'utilisation de l'annexe 1, à l'étape 3, il est évoqué la possibilité de sélectionner le médecin à partir d'une liste construite en tapant les premières lettres du nom du médecin. Scénario classique sur les champs de recherche sur internet.
Cela pourra se présenter ainsi :

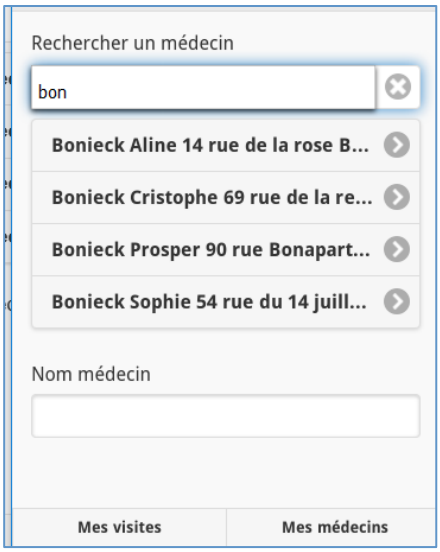


Fig 28

Lorsque l'utilisateur choisira le médecin, la liste disparait et le nom, prénom et adresse apparaissent dans *l'input text* :

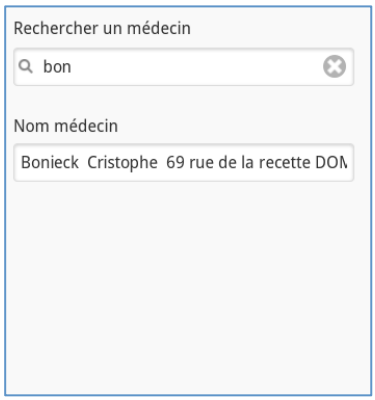


Fig 29

Une nouvelle recherche peut être effectuée en effaçant le contenu de la zone de recherche (icone *delete*).

Heureusement, jQuery Mobile va bien nous faciliter la tâche…

Vous avez remarqué dans la figure 28, que le champ de recherche générait une *listview* ; widget que nous avons un peu manipulé pour les rapports.

Le site suivant propose des exemples de code :
http ://demos.jquerymobile.com/1.4.2/listview-autocomplete-remote/#&ui-state=dialog

a) Côté HTML

Regardez attentivement le code proposé sur ce site, qui distingue bien la partie HTML et jQuery.
Sur la partie HTML, seules les propriétés, *id*, *data-role* et *data-filter* sont nécessaires ; la propriété *data-filter-placeholder* sert seulement à faire apparaître un message par défaut.

Travail à faire

19. Ajouter à la page *pagemedecins.php* dans la partie content, un *label* pour faire apparaître le texte « Recherche du médecin », la *listview*, un *label* et un *input text* ; ces 4 éléments dans une div *class="ui-field-contain*.

b) Côté jQuery

Observons ensemble les éléments principaux du code fourni dans l'exemple :

- ligne 2 : l'événement `filterablebeforefilter` est ajouté à la *listview*. Cet événement a lieu à chaque fois que le contenu de la zone de recherche est modifié ;

- lignes 4 et 5 : grâce au paramètre *data* de la fonction anonyme on récupère le contenu de la saisie avec *data.input.val()* (écrit sur deux lignes dans l'exemple) ;
- ligne 8 : on teste si la saisie est non vide avec plus de 2 caractères, avant de lancer une requête Ajax ;
- ligne 19, on donne à la requête Ajax l'argument saisi ;
- lignes 20 ,21 et 22, on boucle sur la *response* en utilisant une boucle *each* pour construire les balises *li* ;
- lignes 24 et 25, on rafraichit la *listview* qui a été générée dynamiquement.

Remarques :

- dans cet exemple de code, on utilise la forme complète *$.ajax* ; vous pourrez privilégier un appel *$.post*, plus simple. De même, la ligne 19 utilise une fonction anonyme ; nous avons pris l'habitude quant à nous de privilégier dans nos appels Ajax des fonctions nommées
- la boucle *each* peut être remplacée par un *for* classique.

Pour écrire le code jQuery, il faudra suivre les étapes suivantes :

- commencer par le gestionnaire d'événement `filterablebeforefilter` sur la liste, aidez-vous du code de l'exemple ligne2 ;
- cette méthode contiendra très peu de code, la récupération de la saisie, un test sur l'existence et la longueur de la saisie (>=1) et l'appel d'une fonction Ajax :

```
$.post("ajax/traiterrecherchemedecins.php",{
    "nom" : nom
}, foncRetourRecherchePageMedecins,"json" );
```

La fonction *foncRetourRecherchePageMedecin* contient le code qui va générer les balises *li*.

Écrire son code en suivant les étapes suivantes :

- commencer par initialiser une variable chaîne vide (var html = "" par exemple) ;
- poursuivre par une boucle qui parcourt le paramètre *data* de la fonction ; aidez-vous de ce que vous avez écrit pour générer la liste des rapports dans la partie 2. Vous imaginez que le paramètre *data* retourne des lignes de médecin (avec tous leurs champs) ; **voir si besoin l'annexe qui explique comment *loguer* des variables JavaScript.** Récupérer dans des variables chaque id, nom, prénom et adresse de médecin. Ajouter à votre chaine *html* la balise *li* construite avec les informations que vous avez stockées dans vos variables. La balise *li* doit pouvoir afficher les nom, prénom et adresse du médecin et a comme id l'id du médecin.
- après la boucle, ajouter votre chaîne html à votre balise *ul* ;
- terminer par un *refresh* de la liste.

Travail à faire

20. Écrire le code jQuery en suivant les étapes indiquées ci-dessus.

c) Côté PHP

Le code de la fonction *traiterrecherchepagemedecins* est très simple, il récupère le nom retourné en POST par jQuery, appelle une méthode *pdo $pdo->getLesMedecins($nom)* et retourne en JSON le résultat.

Travail à faire
21. Écrire cette fonction.

Il ne reste plus qu'à écrire la fonction *pdo* : elle retourne tous les champs des médecins **triés sur leur nom** et ***dont le nom commence par l'argument $nom***.

Travail à faire
22. Écrire cette méthode.

Pour cette page, il nous reste à faire apparaître le nom, prénom et adresse du médecin dans *l'input text* de la page (figure 29).
Lors de la sélection du médecin dans la liste de recherche, il faut commencer par récupérer l'*id* du médecin à partir de la liste (comme vous l'avez fait pour les rapports). Ainsi, vous allez ajouter du code sur l'événement "click" des balises *li* de la *listview*. On vous donne l'entête :

```
$("#pagemedecins #listeMedecins").on("click","li", function(e) {…
```

Le code à écrire est très proche de celui mis en œuvre pour la récupération du rapport, sans appel Ajax puisque nous disposons des informations ; vous n'oublierez pas de sauvegarder au niveau *window* l'id du médecin ainsi que de rafraichir la *lisview* .

Travail à faire
23. Écrire cette fonction. Tester.

La figure 27 contient un en-tête où figure un bouton traditionnel d'appel à un panel ; celui-ci propose un menu, qui ressemble à cet écran :

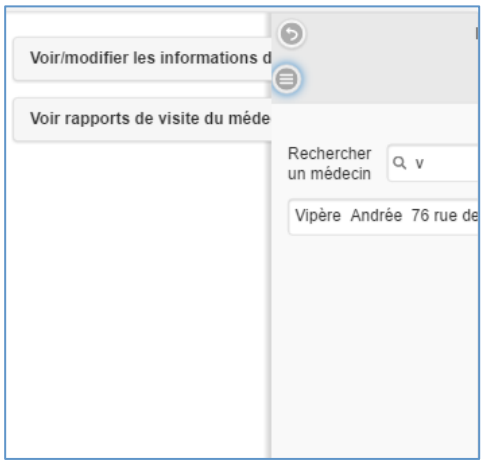


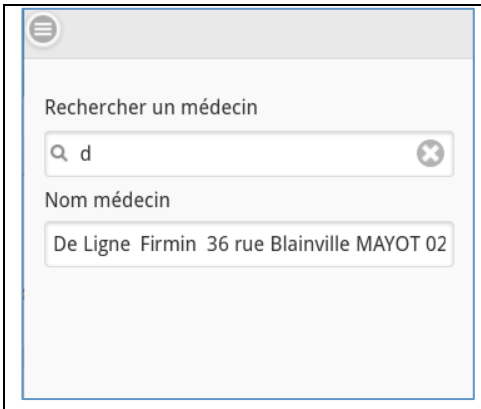
Fig 30

Travail à faire

24. Ajouter un nouveau fichier HTML *entetepagemedecins.html* qui contiendra le code permettant de créer ce qui est montré figure 30. Ajouter ce fichier (clause *include*) au fichier *pagemedecins.php*.

Gestion des médecins : mise à jour d'un médecin

La mise à jour d'un médecin passe par l'ouverture d'un formulaire suite à la sélection d'un médecin (cf. figure 30) ; 'on souhaite obtenir des écrans qui ressemblent à :



On sélectionne le médecin et l'on demande à voir ses informations à partir du panel (fig 30)

Fig 31

La page de mise à jour contient aussi un *widget panel*, mais avec un menu plus réduit :

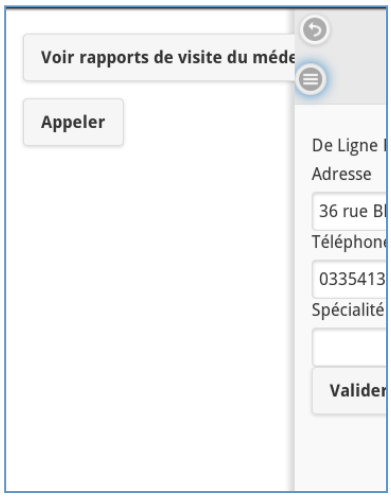


Fig 32

a) Côté HTML

Travail à faire

25. Ajouter cette nouvelle page à votre projet, *pagemajmedecin.php*. Tester l'enchainement des pages.

Vous avez le choix, concernant l'en-tête, de le mettre directement dans la page où dans un fichier *entetepagemajmedecin.html*.

b) Côté jQuery

Lorsque l'on clique sur le panel (figure 30) en choisissant de *voir/modifier les informations*, la page de mise à jour s'ouvre en remplissant les champs de cette page (figure 31). On va ajouter du code sur l'événement "click".

$("#pagemedecins #btnMajMedecin").on("click", function (e) {...

Le code à écrire doit permettre :

- de tester que le champ *input text médecin* de la page *medecins* est bien rempli (ce qui témoigne que l'utilisateur a bien sélectionné son médecin ; dans le cas contraire on affiche une boite d'alerte (et on arrête le fonctionnement normal avec *e.preventDefault()*) ;
- dans le cas favorable, on lance une requête Ajax qui va récupérer toutes les informations du médecin :

```
$.post("ajax/traitergetmedecin.php",{
        "idMedecin" : window.idMedecin
  }, foncRetourGetMedecin,"json" );
```

- de remplir les champs *input text* de la page de présentation des informations du médecin à partir de la fonction de retour ; voici un début :

```
function foncRetourGetMedecin(data){
    var adresse = data["adresse"];
    $("#pagemajmedecin #adresse").val(adresse);
```

c) Côté PHP

La page PHP appelée par l'appel Ajax est simple : son objectif est de retourner le médecin à partir de son id récupéré par POST ou REQUEST.

Il faut bien sûr associer une méthode pdo.

Travail à faire

26. Écrire la page PHP et la méthode pdo.

Dernière étape pour cette page : l'enregistrement des modifications.
Rien à faire de plus côté HTML.

a) Côté jQuery

C'est sur l'événement *click* du bouton *Valider* qu'il faut intervenir pour :

- récupérer les contenus des champs ;

```
$("#pagemajmedecin #btnEnregistrerMajMedecin").bind("click",function(){
    var adresse = $("#pagemajmedecin #adresse").val();
```

- lancer une requête Ajax en fournissant en entrée toutes les informations, y compris l'id ;

```
$.post("ajax/traitermajmedecin.php",{
        "idMedecin" : window.idMedecin,
```

- écrire la fonction de retour qui ne fait rien si ce n'est d'afficher un message positif si l'argument data contient bien 1 (valeur retournée par pdo dans le cas où l'update s'est bien passée).

Travail à faire

27. Écrire le code jquery à partir de la fonction :
$("#pagemajmedecin #btnEnregistrerMajMedecin").bind("click",function(){…

b) Côté PHP

Classique maintenant : récupération des valeurs *postées* par l'appel Ajax, appel d'une méthode *pdo* à écrire qui réalise *l'update* en base ; la valeur de retour est l'appel de la méthode *pdo execute*.

Travail à faire

28. Écrire le code de la page PHP et la fonction pdo.

Les précédents rapports de visite du médecin

Nous arrivons presque au bout de la gestion des médecins. Lla dernière partie va nous faire découvrir un *widget* bien sympathique.

Cette nouvelle page est appelée à partir des panels et du menu « Voir rapports du médecin » (fig 32).

On désire obtenir la vue suivante :

Fig 33

Nous avons utilisé ici le mode tablette avec l'émulateur d'*Opéra*.

Ainsi, la vue montre tous les anciens rapports du médecin sélectionné auparavant.

Le bouton à droite permet de sélectionner les colonnes que l'on veut afficher, ce qui est bien pratique en mode *smartphone* :

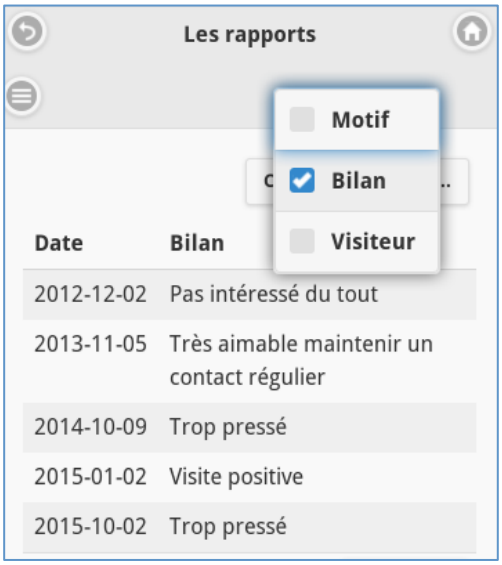


Fig 34

Le *widget* que nous allons utiliser nous demandera un minimum d'effort…

Il s'agit du *widget table*, dont vous pouvez voir la mise en œuvre sur un de nos sites préférés :

http://demos.jquerymobile.com/1.4.5/table-column-toggle/

En lisant la documentation, on constate que certaines propriétés sont indispensables dans la déclaration de la balise table : l'*id*, le *data-role* et le *data-mode* ainsi que la class *table-stripe*; le reste n'est pas obligatoire. On constate également que la structure ressemble à ce que vous connaissez concernant les tableaux HTML.

Le panel figurant à gauche permet de voir le menu suivant :

Fig 35

a) Côté HTML

Travail à faire

29. Ajouter une nouvelle vue *pagevoirlesrapprts.php* qui contiendra :

- l'en-tête ou son appel permettant d'insérer le panel ;
- la balise table avec les seules parties *thead* remplie et *tbody* vide mais avec un *id* (son contenu sera chargé par jQuery) ;
- et bien sûr le pied de page commun.

b) Côté jQuery

Le code sera contenu dans la fonction maintenant bien connue :

$("#pagemedecins #btnVoirRapports").on("click", function (e) {…

Le code à écrire doit permettre :

- de tester si un médecin a été sélectionné ;

- de lancer dans le cas favorable une requête Ajax avec en entrée l'*id* du médecin.

```
$.post("ajax/traitergetlesrapports.php",{
        "idMedecin" : window.idMedecin
    }, foncRetourGetLesRapports,"json" );
```

La *fonction de retour* de l'appel Ajax doit permettre de remplir la balise table dans sa partie *tbody* en imaginant que l'argument *data* contient la liste des rapports et pour chacun le nom et le prénom du visiteur. Ainsi, dans une boucle *for* sur le nombre de rapports, vous allez générer les balises *tr* et *td* comme vous le feriez tout un tableau html. N'oubliez pas de rafraichir la table à la fin comme il l'était nécessaire pour les *listview*.

Travail à faire

30. Écrire le code de ces deux fonctions.

c) Côté PHP

Le code de la fonction PHP appelée par Ajax n'est plus à détailler maintenant, il devra y avoir un appel d'une méthode *pdo* :

$lesRapports = $pdo->getLesRapports($idMedecin);

La méthode *pdo* doit retourner les rapports ainsi que les nom et prénom des visiteurs concernés, rapports triés par date.

Travail à faire

31. Écrire le code de la page PHP et la méthode pdo.

Si vous êtes à ce point sans trop d'encombre, bravo !!!

Si l'on teste la partie médecin en détail, on rencontre un problème lorsque l'on passe, après la sélection du médecin à la page des rapports et ensuite à celle de la mise à jour du médecin. En effet, les événements "click" n'ont pas été associés à du code pour ces deux panels là.

Une solution est *d'abonner* au gestionnaire existant les deux boutons. On utilise la syntaxe suivante :
$("#pagemedecins #btnMajMedecin,#pagevoirlesrapports #btnMajMedecin ").on("click", function (e) {…

Une virgule sépare les deux boutons ; ainsi chacun exécutera ce gestionnaire d'événement. On dit que deux boutons se sont abonnés au même gestionnaire.

Travail à faire

32. Réaliser cette modification ainsi que celle concernant la page *pagemajmedecin*.

Remarque

On peut trouver un peu redondant d'avoir à déclarer trois panels qui font, à peu près la même chose. Il est possible avec la version 1.3.x de jQuery Mobile de ne déclarer qu'un seul panel dans une page et de l'appeler dans une autre. Avec la version 1.4.x ce n'est plus possible ; un mécanisme d'*external panel* est proposé mais sa mise en œuvre n'est pas très convaincante sur le site jQuery.

On touche là à une particularité de jQuery Mobile (ce n'est pas le seul langage dans ce cas) ; il s'agit de sa grande évolubilité ; à chaque nouvelle version un grand nombre d'instructions sont déclarées obsolètes et remplacées par d'autres ou d'autres mécanismes. C'est assez déroutant mais c'est le prix à payer pour profiter de la puissance mais aussi la jeunesse du produit.

Une dernière chose à faire concernant la gestion des médecins ; l'appel téléphonique en direction de celui qui est sélectionné dans la *listview*.

La gestion de l'appel téléphonique en direction du médecin

a) Côté HTML

Commençons par modifier l'icône des panels qui pointent sur l'appel téléphonique :

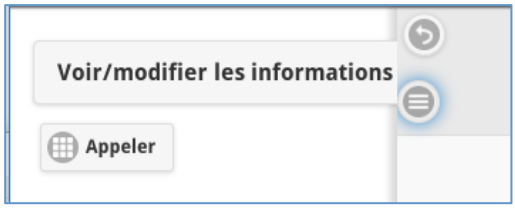


Fig 36

Travail à faire

33. Procéder à cette modification en recherchant sur le site officiel le *data-icon* qui convient.

b) Côté jQuery

Pour pouvoir effectuer directement l'appel, il faut disposer du numéro de téléphone. Nous allons choisir d'ajouter dans *listview* un champ caché dont la valeur prendra le numéro de téléphone de chaque médecin.

Travail à faire

34. Ajouter ce champ caché dans la balise *li* construite dans la fonction de rappel :

function foncRetourRechercheMedecinPageMedecins(data)

35. Tester, en utilisant FireBug (touche F12), que le numéro apparaît bien :

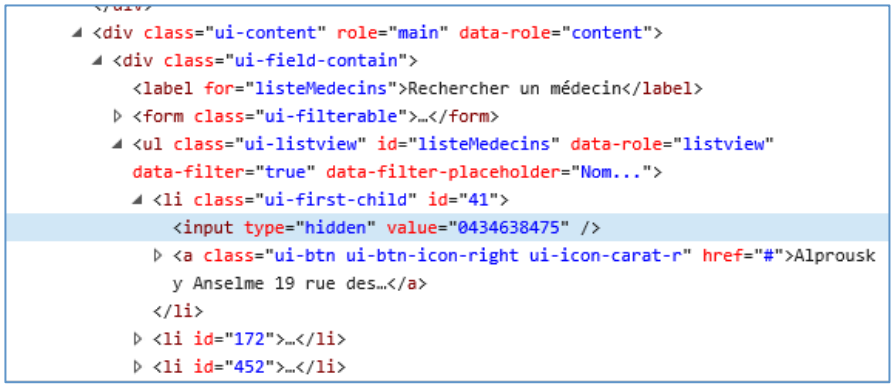


Fig 37

Lors du clic sur un élément de la *listview*, il faut récupérer la valeur du champ caché et la mettre dans les liens *href* des panels (propriété *href*) ; c'est dans le code de la fonction

$("#pagemedecins #listeMedecins").on("click","li", function(e)

qu'il faut intervenir.

On peut récupérer la valeur du champ caché en utilisant la méthode *find* :

var numeroTel = $(this).find("input:hidden").val() ;

Travail à faire

36. Effectuer les modifications proposées.

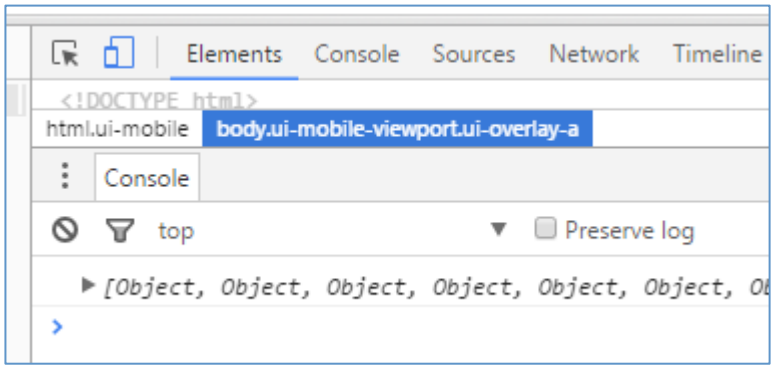
Ceci termine la partie 3 ; le corrigé est disponible dans le répertoire **CorrigegsbMobilePartie3.**

Annexe : débogage avec JavaScript

Lorsque l'on souhaite déboguer du code JavaScript, il faut éviter (en général) d'utiliser la boite de dialogue *alert*. Celle-ci doit être utilisée dans de très rares cas, par exemple voir si on « passe » dans un morceau de code.
Il faut utiliser les outils de débogage prévus à cet effet, il faut utiliser par exemple l'objet *console* dans le code :

```
function foncRetourRechercheMedecinPageMedecins(data){
    console.log(data);
```

En mode débogage (F12 sur tous les navigateurs), avec l'onglet console :



On accède aux détails de la donnée loguée.