Découverte du Framework Angular JS avec le contexte GSB

Partie 2 L'accès aux données

Description du thème

|  |  |
| --- | --- |
| Propriétés | Description |
| **Intitulé long** | Découverte du Framework Angular JS avec le contexte GSB dans sa partie gestion des rapports de visite. Ceci est la partie 2 |
| **Formation concernée** | BTS SIO option SLAM |
| **Matière** | SLAM 4 |
| **Présentation** | Accompagnement dans la découverte d’Angular. Développement pas à pas d’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  • Programmer au sein d’un Framework |
| **Prérequis** | Les principes du développement web, PHP, SQL, JavaScript |
| **Outils** | SGBD MySQL, un environnement de développement |
| **Mots-clés** | GSB, Angular JS, Ajax, MVVM |
| **Durée** | 10 heures |
| **Auteur(es)** | Patrice Grand. Relecture de Cécile Nivaggioni et Yann Barrot. |
| **Version** | v 1.0 |
| **Date de publication** | Novembre 2016 |

Présentation

La première partie s’est attachée à présenter les concepts principaux d’Angular : les directives, le *binding*, le modèle vue/contrôleur, le mécanisme de routage, les injections d’objets.

Dans cette deuxième partie nous resterons sur ces acquis en introduisant l’accès aux données et la technologie Ajax.

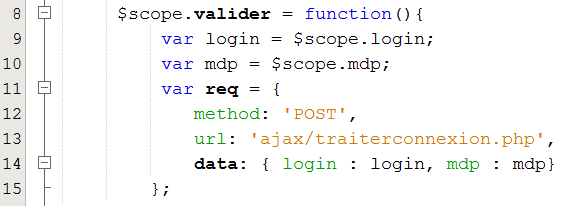
Comme ce n’est pas l’objet de ce support, nous fournirons une application de départ avec la classe d’accès aux données entièrement écrite. Cette application est disponible dans le répertoire **gsbAJSV2.0** (pour ceux qui souhaitent partir des sources obtenues à la fin de la partie précédente, les éléments à récupérer dans ce répertoire seront indiqués lorsque cela sera nécessaire); elle est identique à la correction de la première partie à laquelle on a ajouté la classe *pdo* (fichier *pdogsbrapports.php* dans le répertoire « *data* ») ainsi que différents fichiers qui seront présentés dans cette partie.

Vous aurez bien sûr besoin d’installer la base de données dont un script se trouve dans le fichier *gsbRapports.sql*.

##### Le formulaire de connexion

Nous allons reprendre l’authentification en intégrant l’accès aux données. Nous utiliserons une requête Ajax privilégiée dans les applications de type *Single Page Application*.

Dans le contrôleur de connexion, sur la propriété *valider* en lieu et place du code initial, après avoir récupéré des propriétés *login* et *mdp*, nous commençons par ajouter la requête Ajax :



Ligne 8 : c’est sur le click du bouton valider que la requête doit être menée

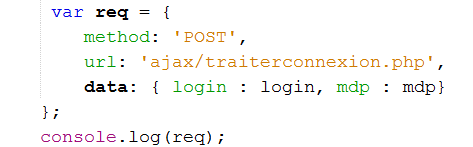
Ligne 9, 10 : on récupère les propriétés login et mdp du *scope*

Ligne 11 : un ***objet*** req est créé (présence des accolades) contenant 3 champs :

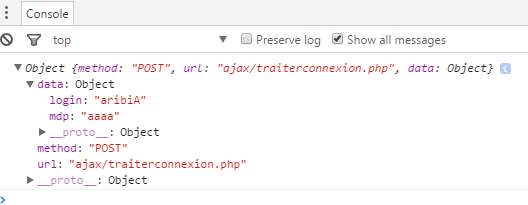
* + un champ *method* dont la valeur (présence des « : ») est ‘POST’
  + un champ *url* dont la valeur est ‘*ajax/traiterconnexion.php’*
  + un champ *data* dont la valeur est un objet qui contient deux champs…

Ainsi, cette requête va générer une méthode POST et exécuter le code PHP qui se trouve dans le fichier *traiterconnexion.php*. Elle fournit une donnée à ce fichier PHP sous la forme de l’objet *data*.

À ce moment, l’appel Ajax n’est pas encore fait, seul un objet vient d’être créé. Pour s’en persuader, *logons* l’objet *req*:



Et lançons l’application en exécutant *FireBug* (touche F12 pour la plupart les navigateurs ; celui intégré au navigateur Safari est accessible à partir d’un clic droit puis en sélectionnant « inspecter l’élément») ; dans le menu console, on peut voir l’objet :



On retrouve bien la structure de l’objet *req*; notons au passage que JavaScript ajoute à chaque objet, ici *req* et *data*, un autre objet \_\_proto\_\_, mais nous n’avons pas besoin de nous attarder sur la création des objets avec JavaScript qui pourrait faire l’objet d’un cours particulier…

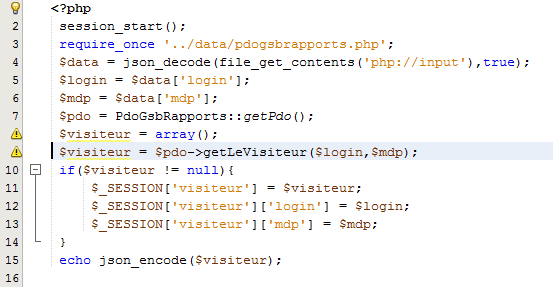
Pour effectivement lancer la requête Ajax, il faut faire appel à un service dédié *$http*. Vous aurez compris qu’il faut d’abord *injecter* ce service au niveau du contrôleur :



Nous pourrions ainsi utiliser le service *$http*:

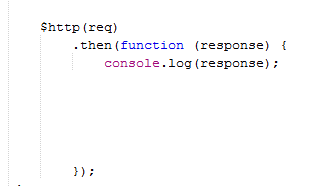
*$http(req)*;

Au préalable, il faut créer le fichier annoncé dans la requête Ajax : *traiterconnexion.php* qui doit se trouver dans le répertoire Ajax (usage suivi).



* + ligne 2 : on se met en mode session car nous allons enregistrer des données sur le serveur.
  + ligne 3 : appel de la classe pdo.
  + ligne 4 : récupération dans un tableau, *$data,* des données qui sont envoyées par la requête Ajax, (celles définies dans le champ data de l’objet *req* déclaré dans le contrôleur) ; ces données doivent être encodées dans un format compatible pour PHP, c’est le rôle de la fonction *json\_decode.*
  + lignes 5 et 6 extraction des valeurs du tableau *$data.*
  + ligne 7 : création d’un objet *pdo*.
  + lignes 8 et 9 : création d’un tableau et récupération des données de la base.
  + lignes 10 à 14 : on souhaite ici, pour des traitements ultérieurs mettre en session les données.
  + ligne 15 : toute fonction Ajax se termine par *echo* qui retourne une information ; cette information sera récupérée par JavaScript.

Nous pouvons maintenant lancer la requête Ajax ; nous vous fournissons le début du code :



* + *$http* a une méthode *then* qui prend comme paramètre une fonction anonyme
  + nous avons ajouté un *log* sur ce que récupère cette fonction, *response*
  + il est d’usage d’écrire la méthode *then* sur une autre ligne.

**Travail à faire**

* Créer la base de données « gsbrapports » et importer le fichier de scripts gsbrapports.sql.
* Créer le fichier *traiterconnexion.php* tel qu’il est présenté ci-dessus.
* Modifier la méthode *$scope.valider* en y intégrant l’appel de la requête Ajax décrite en page 2.
* En vous aidant du *log* de *response*, terminer en écrivant le test qui permettra d’afficher le message d’erreur ou de charger la page d’accueil.
* Tester (rappel : les codes d’accès des visiteurs sont consultables dans la table *visiteur*).

**Remarque :** en cas d’erreur 500 renvoyée par le serveur web, pensez à vérifier les paramètres d’accès à la base de données dans le fichier *pdogsbrapports.php.*

##### Gestion des menus

Nous désirons obtenir les enchaînements des menus suivants :

|  |  |  |
| --- | --- | --- |
|  | Gestion des rapports |  |
| Fig 1 | Fig 2 | Fig 3 |
|  | Gestion des médecins |  |
| Fig 4 | Fig 5 | Fig 6 |

* Les figures 1 et 4 montrent la même page d’accueil sans sous-menu.
* Les figures 2 et 5 montrent un bouton permettant d’ouvrir un sous-menu.
* Les figures 3 et 6 montrent les options du sous-menu.

Les figures 2 et 3 détaillent le fonctionnement du menu avec un bouton qui permet d’ouvrir le sous-menu. Nous utiliserons à chaque fois ce type d’enchainement ; la barre de navigation (*navbar)* haute sera présente dans toutes les pages et chaque page aura son propre sous-menu.

**L’application dans sa version 2.0 contient le code HTML des sous-menus, dans le fichier *menuCommun.html*. Nous allons commenter certains points** :



* La barre de sous-menu est un tableau HTML (L10 à L18).
* A gauche le bouton avec deux propriétés :
  + *btnVisible* associée à la directive *ng-show* (que vous connaissez et qui permettra de masquer le bouton pour la seule page d’accueil) ;
  + *menu* associée à *ng-click* qui a le même rôle que *ng-submit* du formulaire de connexion. Les classes CSS sont empruntées à BootStrap et légèrement enrichies par le fichier CSS *casGSB.css.*
* On retrouve la propriété titre (ligne 16) qui changera de valeur selon la page chargée.
* Ligne 21, un composant Angular « *uib-collapse* » est utilisé pour cacher ou afficher le sous-menu. Le site <https://angular-ui.github.io/bootstrap/> propose quelques *composants* mais qui restent très éloignés en qualité de ceux proposés par jQuery. Ces composants sont développés sous forme de directive ; le fichier *index.html* de *gsbAJSV2.0* intègre les fichiers nécessaires (*collapse.js* et *angular-animate.js*).
* Les lignes 21 et 23 font apparaître deux propriétés *isCollapsed* et *srcMenu* qui devront être valorisées dans les contrôleurs. La directive *ng-include* permettra de charger le bon fichier du sous-menu ; ce dernier ne contiendra que des liens *href*, options des sous-menus.

Remarque : si vous travaillez avec NetBeans dont la compatibilité avec AngularJS est limitée, vous pourrez avoir parfois des erreurs sur des attributs de balise du Framework AngularJS ; c’est le cas de l’attribut *uib-collapse*. Il suffit d’accepter cet attribut en cliquant sur l’erreur et de choisir la bonne option.

Si vous ouvrez le fichier *app.js* de gsbAJSV2.0, vous constatez que les injections nécessaires ont été réalisées :



**Remarque** : nous avons injecté *ngAnimate* pour bénéficier, à l’ouverture du sous-menu, d’une légère animation.

**Dans le cas où vous travaillez à partir de vos propres sources obtenues à la fin de la partie 1**, vous devez :

* + mettre à jour votre fichier *app.js* afin d’y intégrer cette injection ;
  + récupérer les scripts *collapse.js* et *angular-animate.js* (disponible dans le répertoire « js » de gsbAJSV2.0) puis les déclarer dans *index.html*;
  + compléter menuCommun.html pour y intégrer le sous-menu (comme indiqué ci-dessus).

###### Gestion du menu *accueil*

Ce menu ne propose pas de modification d’apparence (cf. fig 1), néanmoins, il faudra un peu modifier le code du contrôleur afin de ne pas faire apparaitre le bouton et le sous-menu.

**Travail à faire**

Compléter le code du contrôleur de la page d’accueil en conséquence.

###### Gestion du menu *rapports*

Les figures 2 et 3 présentent ce qui est attendu. Il faut ajouter un fichier construisant le sous-menu (fig 3) attendu, *menuRapports.html*.

**Travail à faire**

* Créer le fichier *menuRapports.html* contenant les deux liens « Nouveau rapport » et « mise à jour » (ce seront pour l’instant des liens *morts*).

Dans le contrôleur associé :

* valoriser les propriétés *btnVisible* (booléen) et *isCollapsed* (booléen),
* ajouter la fonction associée à la propriété *menu* qui agira sur la propriété *isCollapsed*. Le *clic* sur le bouton ouvrira ou fermera le sous-menu selon sa situation
* tester.

###### Gestion du menu *medecins*

**Travail à faire**

En vous inspirant de ce que vous venez de réaliser pour la question précédente, effectuer les modifications et ajouts nécessaires pour le sous-menu *médecins,* puis tester.

##### Mise à jour d’un rapport de visite

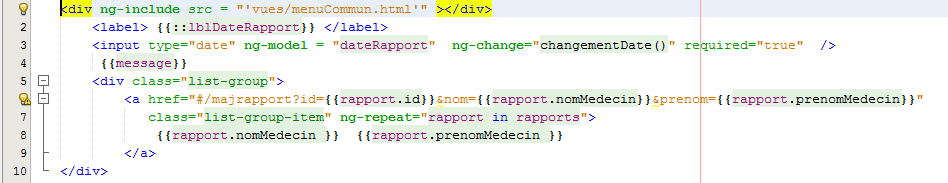
Conformément au cas d’utilisation décrit dans la première partie, on désire avoir les enchainements suivants :

|  |  |
| --- | --- |
| Fig 1 | Fig 2 |
| Fig 3 | Fig 4 |

###### ****Choisir le rapport à modifier (figures 1, 2 et 3)****

Lorsque l’on sélectionne la mise à jour d’un rapport, on ouvre une nouvelle page (fig 1 et 2). Nous devons commencer par ajouter une vue et son contrôleur.

On vous fournit le code de cette vue, *choisirRapport.html* :



* + à la ligne 1 on inclut le menu commun vu plus haut.
  + la ligne 2 ajoute un *label* dont la valeur sera chargée par le contrôleur « Choisir le rapport… ».
  + la ligne 3 ajoute un *input date*, la propriété associée dans le contrôleur est *dateRapport* ; on découvre la directive *ng-change* (équivalent au *onChange* de JavaScript).
  + la ligne 4 va proposer un message dans le cas où aucun rapport n’existe à la date sélectionnée.

**Remarque** : l’élément HTML *input type = ‘date’* n’est disponible actuellement que sur les navigateurs Chrome et Edge ; pour les autres navigateurs l’utilisateur devra saisir les dates au format AAA/MM/JJ.

Cette première partie de saisie de la date n’est pas très compliquée.

Maintenant, observons attentivement les lignes 6 à 8 :

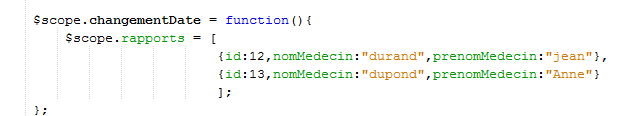
* + La directive *ng-repeat*, comme son nom le laisse imaginer, va itérer la balise *<a href>*. Si l’on comparait avec un langage de programmation, on aurait quelque chose comme :  
    *foreach( rapport in rapports){ <a href ..> rapport.nomMedecin rapport.prenomMedecin</a>}*  
    la propriété *rapports* doit être valorisée dans le contrôleur, la variable *rapport* est interne à la balise <a> où la directive *ng-repeat* est utilisée.
  + Ne prêtez pas trop d’attention aux classes CSS ; elles sont fournies par Angular et permettent un affichage un peu plus sympathique.

**Travail à faire**

* Ajouter une nouvelle route dans le fichier *app.js* nommée *choisirrapport* ; le contrôleur à indiquer sera nommé *choisirRapportController.*
* Ajouter ce contrôleur qui ne fera que valoriser les propriétés présentes dans la vue ; inspirez-vous de ce que vous avez écrit au paragraphe 2.

Nous pouvons maintenant revenir sur la directive *ng-repeat* en testant son fonctionnement.

Dans le contrôleur *choisirRapportController* nous allons valoriser la propriété *rapports* avec un tableau arbitraire d’objets :



**Travail à faire**

* Ajouter ce code dans le contrôleur.
* Vérifier que vous obtenez bien ce qui suit (les champs *id* doivent apparaitre en bas au survol des noms figurant dans la liste).

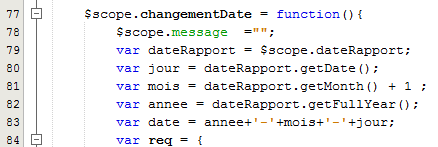


Nous sommes prêts à mener la requête qui utilisera la base de données. Ceci se fera en utilisant une requête Ajax.

**Le principe sera le suivant :**

1. lorsqu’une date est sélectionnée dans la vue *choisirRapport.html*, la fonction anonyme associée à l’événement *changementDate()* du contrôleur *choisirRapportController* est appelée ;
2. la requête Ajax définie dans cette fonction est chargée de transmettre au fichier *traiterlesvisitesaunedate.php*, en méthode POST, la date sélectionnée par l’utilisateur ;
3. *traiterlesvisitesaunedate.php* fait appel à la méthode *getLesVisitesUneDate()* définie dans la classe *PdoGsbRapports* afin de récupérer la liste des rapports de visites, puis retourne ces informations au format JSON à la fonction *changementDate()*;
4. la réponse à la requête Ajax est ensuite traitée dans la fonction anonyme:
   1. si un ou plusieurs rapport sont récupérés, ils sont affectés à la propriété « rapports » (utilisées dans la directive « ng-repeat » de la vue *choisirRapport.html* pour générer autant de lien qu’il y a de rapports retournés) ;
   2. dans le cas où il n’y a aucun rapport retourné pour une date, le message « désolée, pas de rapport ce jour … » sera affiché.

Nous vous fournissons le début :



Remarque : les lignes 80 à 83 récupèrent la date et la formatent pour être utilisée avec PHP. Il existe peut-être un autre procédé plus rapide, mais je ne l'ai pas trouvé…

**Travail à faire**

* Apporter les modifications nécessaires afin de respecter le fonctionnement de la requête Ajax énoncé ci-dessus (pensez à vous aider de l’implémentation de la gestion de l’authentification).
* Tester (rappel : l’identifiant d’un visiteur est défini dans la table visiteur et les rapports de visites sont stockés dans la table rapport).

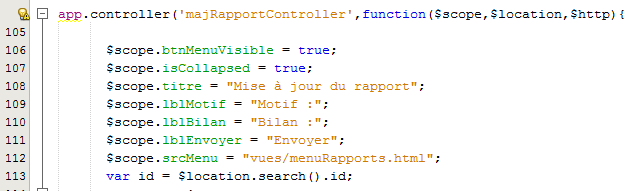
###### ****La mise à jour du rapport****

En fonction de la date de création de rapport recherchée, la vue affichera la liste des médecins pour lesquels un rapport de visite a été réalisé ou le message « désolée, pas de rapport ce jour … ».

Chaque nom sera cliquable (un lien *href* contenant l’identifiant du rapport a été ajouté pour chacun) et permettra d’éditer et mettre à jour le rapport correspondant (voir Fig.3 et 4)

**Comme cette partie ne présente pas de notions nouvelles, le code est fourni.**

**Le fichier HTML est disponible dans le répertoire vues (*majRapport.html*) ; de même le contrôleur est écrit mais sous forme de commentaire :**



**À la ligne 113, on récupère l’*id* du rapport en appelant la méthode *search* du service *$location*, qu’il faut donc *injecter* en ligne 104.**

**Le service *$location* permet de modifier ou de récupérer l’URL de navigation. Dans notre cas, il est nécessaire de récupérer la valeur de l’identifiant du rapport sélectionné, c’est le rôle de la méthode *search*. Elle s’appuie sur le nom du (ou des paramètres) envoyé(s) via l’URL, ici, il s’agit de *id*.**

**Les deux fonctions PHP appelées dans les appels Ajax sont également disponibles.**

**Dans le cas où vous travaillez à partir des sources obtenues à la fin de la partie 1**, vous devez :

* + récupérer la vue majRapport.html (disponible dans le répertoire « vues » de gsbAJSV2.0) puis la stocker dans votre répertoire « vues » ;
  + ajouter le contrôleur *majRapportController* (disponible en commentaire dans le fichier *controller.js* de gsbAJSV2.0) dans votre fichier *controller.js* ;
  + récupérer les fichiers *traiterchoixrapport.php* et *traitermajrapport.php* (disponibles dans le répertoire « ajax » de gsbAJSV2.0) puis les stocker dans votre répertoire « ajax ».

**Travail à faire**

**Ajouter la route afin que tout ceci soit opérationnel puis tester.**

**On souhaite faire une légère modification ; lorsque l’on sélectionne le médecin, on souhaite revoir le nom et le prénom du médecin dans la page de mise à jour :**



**Rappel : le nom et le prénom du médecin sont disponibles sur la vue *choisirRapport*.**

**Travail à faire**

**Réaliser cette modification.**

Ceci termine la partie 2 dont le corrigé se trouve dans le fichier décompressé **gsbAJSV2.1**