GSB avec Angular2, partie 3

Gestion des médecins, découverte des directives

Introduction

Dans les deux parties précédentes nous avons exploré les mécanismes principaux du Framework :

- l'organisation des projets,
- le binding,
- l'injection de dépendances,
- le routage,
- l'accès aux données avec la bibliothèque rXjs et l'utilisation d'un service dédié.

Nous allons poursuivre en développant la suite de l'application. Nous allons nous concentrer sur le cas d'utilisation de gestion des médecins, la mise à jour et l'édition des derniers rapports d'un médecin.
Vous pouvez utiliser le dépôt suivant comme projet initial :
https://github.com/patricegrand/GSBAngular2_V2.1.git

1) La recherche d'un médecin

Nous désirons obtenir l'enchainement suivant :

Fig 1

Fig 2

Fig 3

Fig 1 : lorsque l'on sélectionne l'option du menu « Gestion des médecins », la légende s'affiche ainsi que le champ de recherche.
Fig 2 : on saisit le début du nom du médecin, une liste s'affiche au-dessous
Fig 3 : lorsque l'on sélectionne un médecin de la liste, son nom, son prénom et son département s'affichent dans la zone de recherche ; un menu apparait, associé à ce médecin.

GSB avec Angular2, partie 3

Nous allons commencer à ajouter dans le service *DataService* le traitement nécessaire.

1.a) Le DataService mis à jour

Il faut ajouter une nouvelle méthode dans la classe *DataService* qui va permettre de récupérer tous les médecins dont le nom ***commence*** par la valeur de l'argument *nomMedecin* :

```
public chargerMedecins(nomMedecin : string) : Observable<string[]>{
        let url : string =  this.urlDomaine + "/medecins?nom=" + nomMedecin;
        let req = this.http
                    .get(url)
                    .map((r: Response)=>r.json());
        return req;

    }
```

Ce code peut être *copier/coller*
Remarque : ce code ne se distingue de la méthode *connexion* déjà présente que par l'URL du service REST (valeur de la variable *url*).

1.b) Le fichier HTML évolution1

L'ensemble du code HTML va être inséré dans le fichier app.medecin.html.
Cette évolution correspond à la figure 1 présente plus haut.

Navbar masquée sur la figure 1

```
<my-navbar></my-navbar>
<h2>
    {{legende}}
</h2>
<nav class="navbar navbar-inverse" [hidden]="estCacheMenu">
    <ul class="nav navbar-nav"
        style="display:flex; flex-direction:row;
             justify-content: center">
        <li><a (click)="derniersRapports()">Derniers rapports</a></li>
        <li><a (click)="majMedecin()">MAJ</a></li>
    </ul>
</nav>
<div class="col-lg-6">
    <div class="form-group">
         <input type="search" placeholder="Nom du médecin..."
         [(ngModel)]="nomMedecin" (keyup)="charger()" />
    </div>
</div>
```

Ce code peut être *copier/coller*.

Les trois éléments principaux sont pointés :

- le component *my-navbar,*
- le menu (noir à droite) qui ne sera visible que lorsque l'on sélectionnera le médecin, on a utilisé des classes *Bootstrap*,
- la zone de saisie.

Remarque : la méthode *charger()* est associée à l'événement *keyup*, préférable à l'événement *change*.
.

Travail à faire

En observant ce code HTML, énumérer les champs et les méthodes qu'il faudra déclarer dans la classe associée *MedecinsComponent*.

1.c) La classe MedecinsComponent évolution 1

La méthode *charger()* du fichier HTML, doit faire appel au service DataService qu'il faudra au préalable injecter dans la classe (voir la notion d'injection dans la partie 2) :

```
constructor( private dataService : DataService){}
charger() : void{
     this.dataService.chargerMedecins(this.nomMedecin)
                                .subscribe(
                                    (data)=>{this.lesMedecins = data;}
                                    ,(error)=>{}
                                            );
}
```

Ce code peut être *copier/coller*

Remarques :

- injection d'un objet dataService dans le constructeur,
- le deuxième argument (une fonction) de la méthode subscribe() est optionnel ; nous le conservons néanmoins pour éventuellement plus tard gérer des erreurs.

Un nouveau champ est utilisé ici, lesMedecins, qu'il faudra au préalable déclarer ; son type est :

```
lesMedecins : Array<any>;
```

Travail à faire

Mettez en œuvre ce qui précède afin de produire ce qui est présenté dans la figure 1 :
- déclarations et valorisations éventuelles des champs présents dans le fichier HTML,
- collage du code proposé au-dessus.

L'étape suivante correspond à ce qui est présenté dans la figure 2 : la présence d'une liste chargée avec les médecins concernés par le filtre saisi dans la zone supérieure.

1.d) Le fichier HTML évolution2

Cela va être l'occasion de découvrir une nouvelle notion, celle de *directive* qui enrichit le comportement du code HTML. C'est ainsi qu'il est possible, avec un code minimum, de proposer ce qui est montré dans la figure 2 :

```
<!--affichage des médecins -->
<div class="list-group">
    <ul>
        <li class="list-group-item" *ngFor="let medecin of lesMedecins">

        {{medecin.nom}} {{medecin.prenom}} ;dep : {{medecin.departement}}
        </li>
    </ul>
</div>
```

Ce code peut être *copier/coller*

Remarques :

- des classes Bootstrap sont utilisées,
- la *directive* ****ngFor*** est utilisée, elle permet d'itérer grâce à l'expression « **let** medecin **of** lesMedecins ». Dans un autre langage, php ici, on trouverait :

foreach($lesMedecins as $medecin)
echo '<li'. $medecin['nom '].' '.$medecin['prenom']. ' ; dep : ' .
*$medecin['departement'].
" ;*

N'oubliez pas que *lesMedecins* a été déclaré dans la classe et été valorisé par la requête REST.

Travail à faire

Ajouter dans le fichier HTML ce qui vient d'être décrit afin de réaliser ce qui est présenté dans la figure 2.

Il ne nous reste plus qu'à permettre la sélection d'un médecin afin de le faire apparaître dans la zone de saisie. Il faut pour cela ajouter un événement dans la balise li :

```
25          <li class="list-group-item" *ngFor="let medecin of lesMedecins"
26          (click)="selectionner(medecin)">
27          {{medecin.nom}} {{medecin.prenom}} ;dep : {{medecin.departement}}
28          </li>
29       </ul>
30    </div>
```

- ligne 26, on ajoute l'événement *click* qui appelle une méthode *selectionner* qui prend comme argument la variable *medecin*, déclarée dans l'expression du *ngFor ; en effet la *portée de la variable* est toute la structure où elle a été déclarée, ici l'élément *li*.

Dans la classe, il faut ajouter la méthode *selectionner* :

```
31      selectionner(med) : void{
32           this.medecin = med;
33      // code à compléter
34
35
36      }
```

On utilise un nouveau champ *medecin*, qu'il faudra déclarer dans la classe sous la forme d'un objet :

```
15      medecin : any;
```

Travail à faire

Compléter la méthode *selectionner* afin de gérer l'apparition du menu et le chargement dans la zone de saisi, comme présenté dans la figure 3

2) L'affichage des rapports du médecin

On désire obtenir l'enchainement suivant :

Fig 1

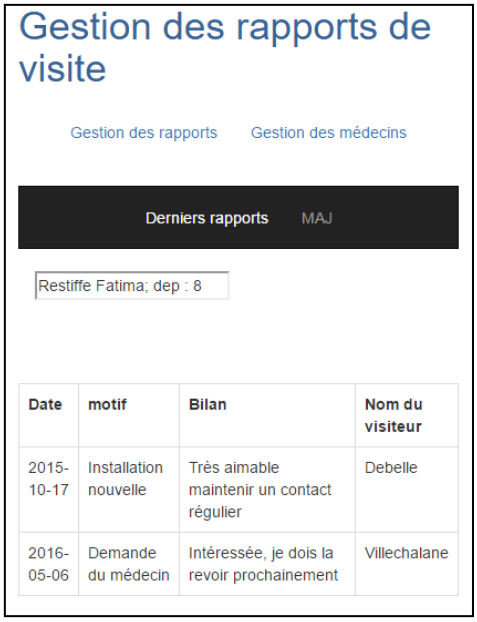

Fig 2

2.1) Le DataService

Il faut ajouter une nouvelle méthode appelant le service REST concerné qui retourne tous les rapports concernés par un médecin (par son id) :

```
public chargerRapports(idMedecin : Number ): Observable<string[]>{
       let url : string =  this.urlDomaine + "/rapports/" + idMedecin;
       let req = this.http
                    .get(url)
                    .map((r: Response)=>r.json());
        return req;
    }
```

Ce code peut être *copier/coller*.

Il n'y a rien à ajouter ; ce code ressemble à ce que nous avons déjà rencontré.

2.2) Le fichier HTML évolution 3

Il faut maintenant gérer l'affichage des rapports :

```
<!--affichage des rapports-->
<table class="table table-bordered" *ngIf="afficherRapports">
    <tr>
        <th>Date</th>
        <th>motif</th>
        <th>Bilan</th>
        <th>Nom du visiteur</th>
    </tr>
    <tr *ngFor="let rapport of lesRapports">
        <td>{{rapport.date }} </td>
        <td>{{rapport.motif }} </td>
        <td>{{rapport.bilan }} </td>
        <td>{{rapport.nom }} </td>
    </tr>
</table>
```

Ce code peut être *copier/coller*.

Remarques :

- nous utilisons encore des classes *bootstrap* pour l'affichage,
- on utilise à nouveau la directive ***ngFor**,
- on utilise pour la première fois une nouvelle directive ***ngIf** qui s'applique ici à la balise *table*. Cette directive prend une expression (évaluable) comme argument ; ici l'expression est réduite à un champ booléen *afficherRapports* – donc équivalent à « *afficherRapports == true* »-. La balise table et son contenu sera alors **ajoutée au DOM** si *afficherRapports* est *true*.

Ce procédé est bien plus performant que la gestion de la propriété *hidden*, que nous avons utilisé pour gérer la visibilité de certains champs.

Il faut bien sûr déclarer *afficherRapports* comme un champ de la classe.

Travail à faire

En vous aidant de ce que vous avez déjà réalisé, terminer cette partie en intervenant dans :
- le service DataService pour ajouter la méthode *chargerRapports* (présentée plus haut)
- la classe *MedecinsComponent* en ajoutant :
    a) le champ booléen *afficherRapport*
    b) le code de la méthode *dernierRapports() (*présent dans la navbar*)* qui doit appeler la *chargerRapports* présentée plus haut
    c) gérer les affichages.
- le fichier HTML en intégrant les éléments fournis à copier.

1) La mise à jour du médecin

C'est la dernière partie de ce module. On désire obtenir :

Fig 1

Fig 2

Fig 3

Fig 4

3.1) Le DataService évolution 3

On vous fournit le code de la nouvelle méthode majMedecin() à intégrer :

```
public majMedecin(id : string ,adresse : string, tel :string, spe : string) {
              let url : string =  this.urlDomaine + "/majmedecin?idmedecin=" +
id + "&adresse=";
                  url += adresse + "&tel=" + tel +"&specialite=" + spe;
              let req = this.http
                          .get(url)
          return req;
    }
```

Ce code peut être *copier/coller*.

Son rôle sera de récupérer les informations du médecin sélectionné. Ces données seront affichées dans un formulaire (fig 2) et pourront être mises à jour si besoin (fig 3 et 4), toujours via la méthode *majMedecin*.

3.2) Le fichier HTML évolution 3

On vous fournit le code à ajouter au fichier *app.medecins.html*, nécessaire à l'affichage du formulaire contenant les informations d'un médecin sélectionné :

```
<!--formulaire de MAJ du médecin-->
<form class="col-lg-6" name="frmMedecin" *ngIf="afficherMedecin"
(ngSubmit)="valider()">
    <div class="form-group">
        <label for="adresse">Adresse </label>
        <textarea rows="3" cols="30" required="true"
[(ngModel)]="medecin.adresse" name="adr">
        </textarea>
    </div>
    <div class="form-group">
        <label for="tel">Téléphone</label>
        <input type="text" required="true" [(ngModel)]="medecin.tel"
name="tel" />
    </div>
    <div class="form-group">
        <label for="specialite">Spécialité</label>
        <input type="text" [(ngModel)]="medecin.specialitecomplementaire"
name="spec" />
    </div>
    <button type="submit">Valider</button>
    <div class="alert alert-danger"
*ngIf="afficherMessage">{{lblMessage}}</div>
</form>
```

Ce code peut être *copier/coller*.

Remarques :

- Le binding (directive ngModel) concerne les champs de l'objet medecin déclaré (et valorisé) dans la classe !! sympa, non ?
- La directive *ngIf permet d'afficher le message (voir figure 4)

Travail à faire

Implémenter cette partie de mise à jour du médecin.

Le corrigé se trouve ici :

https://github.com/patricegrand/GSBAngular2_V3.1