Découverte du Framework Angular JS avec le contexte GSB

Partie 1 Les concepts de base

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 première partie.
Formation concernée	BTS SIO option SLAM
Matière	SLAM4
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	Un environnement de développement
Mots-clés	GSB, Angular JS, Ajax, MVVM
Durée	6 heures
Auteur(es)	Patrice Grand. Relecture de Cécile Nivaggioni et Yann Barrot.
Version	v 1.0
Date de publication	Novembre 2016

Présentation

Ce support se propose de présenter le *framework* Angular JS sous la forme de 5 parties en s'appuyant sur l'application de gestion des visites du contexte GSB :

1) Les concepts de base ;
2) Intégration de l'accès à la base de données, gestion des menus et de la mise à jour des rapports de visite ;
3) Gestion des rapports d'un médecin ;
4) Création d'un rapport de visite ;
5) Pour aller plus loin : création de directives et de services.

JavaScript est un langage qui évolue régulièrement mais il reste prisonnier de son statut originel : langage de script, complément dynamique du HTML. En même temps qu'il devient incontournable dans les applications web (principalement parce qu'il est présent dans tous les navigateurs), fleurissent à son côté des langages/*framework* qui proposent des « surcouches » à JS qui tentent de réduire les contraintes de ce dernier.

Angular JS se situe dans cette démarche de renouvellement. Il a pour lui la crédibilité de son initiateur Google et la participation de Microsoft qui fournit, dans la version 2.0 d'Angular, son langage de script TypeScript.

Ce support utilisera la version 1.5x avec JavaScript car la version 2.0 est très récente et peu mise en œuvre aujourd'hui.

Angular propose une architecture applicative près particulière qui peut se mettre en œuvre sur tous les *STA* (bureaux, tablettes, smartphone) ; aussi très peu de composants graphiques sont proposés par défaut. Nous allons peu nous soucier du style pour nous attacher à comprendre la logique du *framework*. Nous utiliserons néanmoins quelques éléments de style empruntés à Bootstrap, la bibliothèque de styles utilisée dans Twitter. Nous signalerons à chaque fois les emprunts que nous proposerons. Pour les composants que nous utiliserons nous privilégierons le mode *smartphone*.

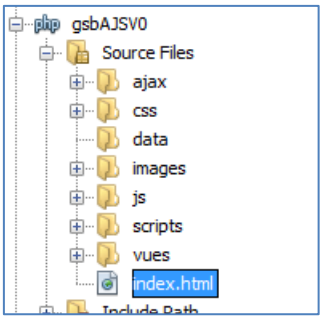
Le contexte

Il s'agit du contexte GSB et de la gestion des visites faites par les visiteurs médicaux auprès des médecins. Le document « Présentation du contexte » ***contenu dans le dossier contexte*** présente ce domaine de gestion et les cas d'utilisation que nous allons réaliser.

1 Pour commencer

Nous allons démarrer le projet avec la version gsbAJSV1.0, disponible dans un fichier zip fourni. Installez-le dans un répertoire de publication, après l'avoir décompressé.

Regardons d'abord l'architecture des répertoires. Celle qui est proposée ici n'est pas imposée, elle est seulement d'usage.



- Le répertoire *ajax* contiendra les différentes fonctions qui solliciteront le serveur pour obtenir des données (partie serveur en php).
- Le répertoire *css* contiendra (éventuellement un fichier CSS personnel).
- Le répertoire *data* contiendra la classe d'accès aux données (partie serveur en PHP).
- Le répertoire *js* contiendra tout le code javaScript créé pour l'application.
- Le répertoire *scripts* contient les scripts importés ; il peut ne pas exister si tous les scripts (Angular et Boostrap) sont téléchargés à partir de leurs sites de stockage. Nous sommes en phase de développement ; il est préférable de télécharger ces scripts dans l'application (voir annexe 1).
- Le répertoire *vues* contiendra tout le code HTML sous forme de différentes vues.

Le fichier index.html ne présente pas de difficultés particulières. Il charge les différentes ressources dans sa partie *head* mais contient deux *directives* inconnues, *ng-app* et *ng-view* :

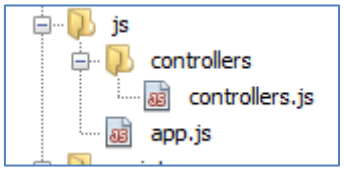
```
<!DOCTYPE html>
<html ng-app="gsbRapports">
    <head ...13 lines />
    <body >
        <div ng-view>
        </div>

    </body>
</html>
```

Les directives, sous Angular, sont l'une des clés de voute du système. Les balises HTML (ici *html* et *div*) peuvent être *décorées* par des directives qui vont en modifier le comportement par défaut.
Ainsi *ng-app* indique qu'Angular ne peut intervenir et donc interpréter toutes les directives que dans cette partie dont la description sera détaillée plus loin. On aurait tout aussi bien pu préciser que l'application était limitée seulement au *body*.
De même, *ng-view* indique dans quelle partie vont venir se placer les différentes vues HTML. Il n'y a qu'une seule directive *ng-view* par application.

Lancer l'application http://localhost/gsbAJSV1.0/ ; celle-ci s'ouvre avec un formulaire de connexion. Rien de spécial non plus… sauf que ce qui est proposé n'est pas dans index.html. Nous allons ouvrir et regarder le répertoire *js*.

1.1 L'application JavaScript

Tout le code JavaScript peut être écrit dans un seul fichier (comme pour jQuery) ; il est fortement conseillé de découper son code en plusieurs fichiers selon les fonctionnalités du code.



Le fichier *app.js* contient la création de l'application (module) et le mécanisme de *routage* :

```
1   var app = angular.module("gsbRapports",['ngRoute']);
2
3   app.config(function($routeProvider){...11 lines });
14
```

- **Création de l'application** : ligne 1, la variable *app* représentera dans tout le code JavaScript l'application (module) créée par l'objet angular. Ce module se nomme *gsbRapports*. En toute rigueur, disons qu'une application peut contenir plusieurs modules ; même si nous resterons sur un seul module, ce qui est bien suffisant. Par ailleurs, le second argument de la méthode module est un tableau ['ngModule'], ici à un seul élément, qui indique quelles ressources sont nécessaires au module ; dans notre cas, c'est une ressource de routage. Cette ressource doit être chargée dans la page d'index :

```
<script src="scripts/angular.min.js"></script>
<script src="scripts/angular-route.min.js"></script>
```

- **Le routage** ; déplions le code :

```
3  app.config(function($routeProvider){
4      $routeProvider
5          .when('/',{
6              templateUrl :'vues/connexion.html',
7              controller :'connexionController'
8          })
9          .otherwise({redirectTo: '/'});
10 });
11
```

Notre module (*app*) appelle une méthode *config* qui prend comme argument une fonction *anonyme* (fonction sans nom, qui s'exécute ainsi sans être appelée). L'argument de la fonction anonyme est *$routeProvider*. Cet objet a une méthode *when*.
La suite du code est assez facile à comprendre : si l'url demandée est '/' alors la vue retournée est connexion.html ; sinon (*otherwise*) on redirige vers '/'.

On comprend maintenant pourquoi l'URL http://localhost/gsbAJSV1.0/ *route* vers connexion.html

- **L'injection de dépendances**. Nous n'entrerons pas dans les détails du mécanisme ; disons simplement que l'idée est d'utiliser des services au moment où on en a besoin. Ainsi, l'objet $routeProvider (qui existe et a été créé) est injecté dans la fonction et peut donc être disponible ; la règle est de respecter son nom !! Ce mécanisme, de plus en plus courant dans les *frameworks*, permet d'utiliser des services (des classes en fait) à la volée sans avoir à se soucier de construire l'objet. Angular JS injecte des classes à partir du nom de l'objet, qu'il faudra respecter. **L'annexe 1** présente une lecture de la documentation et revient sur les services que vous pouvez *injecter*.

Remarque : JavaScript peut présenter une syntaxe un peu déroutante ; c'est ainsi que la structure entre accolades { } représente un objet qui contient deux champs *templateUrl* et *controller* dont les valeurs sont celles qui sont indiquées ; cette syntaxe privilégie la concision.

Le deuxième champ de l'objet évoqué plus haut est *controller* ; ainsi à un routage est associé un contrôleur, un objet nommé *connexionController*. Cet objet est défini dans le fichier *controllers.js*.

```
app.controller('connexionController',function($scope){

});
```

Notre objet *app* crée le contrôleur appelé *connexionController* et lance une fonction anonyme qui décrit le code exécuté au moment du chargement de la vue *connexion.html*. Pour l'instant, rien n'est écrit dans cette fonction ; nous allons le faire ensemble.

Remarques
- *Si nous gérons une autre vue, il faudra, lui lier un nouveau contrôleur et lui construire sa route en ajoutant un when dans app.config.*
- *Il est d'usage de nommer les contrôleurs en ajoutant un substantif à la fin précisant ainsi son rôle.*

1.2 La vue HTML connexion.html

Nous avons vu que le routage nous menait vers le fichier *connexion.html*. Si nous ouvrons ce fichier, nous trouvons, pour l'instant, du code classique ; notons que nous avons utilisé des classes de style Bootstrap, ceci reste un détail ici.

1.3 Le code du contrôleur *connexionController*, notion de *binding*

Nous avons évoqué l'importance des directives ; nous allons aborder le deuxième pilier d'Angular : le *binding* ou **liaison de données**.

Le contrôleur va permettre de fournir des données à sa vue (le fichier HTML) ; mais aussi de récupérer des données construites dans la vue (par des *inputs*, par exemple).

1.3.1 Le *Binding* contrôleur →vue

Dans le formulaire de connexion, nous avons un label qui affiche une légende « Login »:

```
<div class="form-group">
        <label for = "login" >Login  </label>
        <input type="text"   class = "form-control"  />
</div>
```

Cet affichage peut être réalisé par le contrôleur ainsi :

```
        <label for = "login" >{{lblLogin}}  </label>
        <input type="text"   class = "form-control"  />
```

La présence des doubles accolades indique à Angular de remplacer le contenu (*lblLogin*) par une valeur qui est présente dans le **contrôleur associé à cette vue**.
Comment faire cela ? En utilisant l'objet **$scope** qui a été injecté dans le contrôleur ; l'objet $scope permet de faire le lien entre la vue et son contrôleur :

```
app.controller('connexionController',function($scope){
    $scope.lblLogin = "Login";
});
```

Remarque : utiliser le *binding* pour afficher des légendes ici offre un intérêt limité ; il peut se comprendre si on souhaite totalement isoler le métier du *designer* dont le travail ne sera pas perturbé par des données. En ce qui nous concerne, nous l'avons mis en œuvre afin de montrer avec un scénario simple les possibilités du *framework*.

Travail à faire

Mettez en œuvre le *binding* pour les 3 légendes figurant dans la vue : le login (cf. plus haut), le mot de passe et le bouton valider.

1.3.2 Le *Binding* vue →contrôleur

Lorsque les données sont saisies dans des champs *input*, le binding va permettre de les récupérer ; pour cela la directive *ng-model* doit être utilisée :

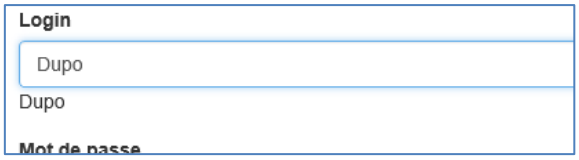
```
<div class="form-group">
        <label for = "login" >{{lblLogin}}  </label>
        <input type="text"  ng-model="login" class = "form-control"  />
</div>
```

En utilisant la directive *ng-model*, vous allez mettre en œuvre un mécanisme d'observation des mises à jour de la donnée inspectée. La donnée pourra être accessible dans le contrôleur, avec la syntaxe *$scope.login* ou bien aussi dans la vue, avec la syntaxe du binding comme vu plus haut :

Ainsi, si on écrit dans la vue :

```
<div class="form-group">
        <label for = "login" >{{lblLogin}}</label>
        <input type="text"  ng-model = "login" class = "form-control"/>
        {{login}}
</div>
```

La donnée saisie apparaîtra à chaque mise à jour :



Mais bien sûr, c'est en général dans le contrôleur que nous aimerons la récupérer.

Travail à faire

Mettre en œuvre ce *binding* pour les deux champs *login* et *mot de passe* ; testez en procédant comme il est proposé juste au-dessus (dans la vue).

1.3.3 Le *Binding* contrôleur →vue un peu particulier

Le mécanisme de *binding* est un plus incontestable, mais il a un coup en termes de performance ; en effet Angular doit inspecter régulièrement les données « bindées » pour prendre la décision de leurs mises à jour ou non. C'est pourquoi Angular a ajouté un *binding léger*, réservé aux données qui ne sont modifiées qu'une seule fois ; c'est le cas de nos différentes légendes. Il est possible de dire à la déclaration dans le code HTML que la donnée ne sera plus modifiée après son initialisation :

```
<label for = "login" >{{::lblLogin}}</label>
<input type="text"  ng-model = "login" class = "form-control"/>
{{login}}
```

Il suffit d'ajouter deux fois les deux points devant le nom de la donnée.

Travail à faire

Modifier le code HTML afin de mettre en œuvre ce *binding* pour les propriétés qui ne changeront pas de valeur.

1.4 Ajouter un fichier

Afin d'améliorer le *look* de l'application, on va ajouter le logo de l'entreprise GSB sur la page d'authentification ; ceci va nous permettre de découvrir une nouvelle directive. L'image est dans le répertoire images.
Une directive permet d'ajouter dans une balise un fichier HTML : *ng-include*.

Travail à faire

- Créer un fichier HTML *logo.html* ne contenant qu'une balise *img* permettant d'afficher le logo de GSB.
- Ajouter ce fichier en utilisant la balise *ng-include* dont la syntaxe est décrite sur le site officiel d'Angular https://angularjs.org/ (menu dévelop/API référence)

2 La validation de la connexion

2.1 La directive *ng-submit*

Cette directive permet de soumettre le formulaire grâce à son bouton *submit*, elle se déclare dans la balise *form* :

```
<form class = "col-lg-6" name = "frmLogin" ng-submit="valider()" >
```

La fonction *valider()* devra être implémentée dans le contrôleur. Elle s'exécutera lorsque l'on soumettra le formulaire. Notons que cette fonction peut utiliser un paramètre ; nous aurons l'occasion de profiter de cette option plus loin.

Dans le contexte GSB, le login et le mot de passe sont évalués en faisant appel à la base de données ; nous aborderons ceci dans la partie 2 de cette application. Concentrons-nous sur les fondamentaux de base d'Angular.
Ajoutons un test très simple dans la fonction valider :

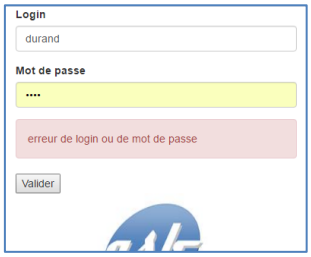
```
app.controller('connexionController',function($scope){
    $scope.lblLogin = "Login";
    $scope.lblMdp = "Mot de passe";
    $scope.lblBtn = "Valider";
    $scope.valider = function(){
        if($scope.login == "dupond" || $scope.mdp == "dddd")
            alert("ok");
        else
            alert("erreur");
    };
});
```

Remarque : pour JavaScript, *valider* est un champ de l'objet *$scope* (comme *lblLogin*, etc.) ; ce champ est par contre une fonction.

Travail à faire

Apporter les modifications proposées en recopiant le code dans votre application.

Plutôt que de faire apparaître une boite d'alerte on désire afficher un message indiquant que les identifiants de connexion ne sont pas valides. En cas d'erreur d'authentification, on désire obtenir :



Pour cela, on ajoute la balise *div* :

```
    <div class="form-group">
        <label for = "mdp" >{{::lblMdp}}</label>
        <input type = "password"  ng-model = "mdp" class = "form-control"  />
    </div>
    <div class="alert alert-danger" ng-show ="msgErreur" >{{::lblMessage}}</div>
    <input type ="submit" value ="{{::lblBtn}}">
</form>
```

On rencontre une nouvelle directive *ng-show*.

La directive *ng-show*, comme son nom l'indique permet de cacher le contenu de la balise selon la valeur booléenne d'une propriété, ici « *msgErreur* ». Cette propriété sera gérée à l'intérieur du contrôleur.
On désire que la *div* soit visible seulement lorsque l'utilisateur ne saisit pas ses identifiants corrects.

Travail à faire

Réaliser cette modification puis tester.

2.2 La navigation

Poursuivons notre découverte d'Angular pour aborder la navigation entre les pages ; ceci est nécessaire puisque l'on désire présenter une nouvelle page après la connexion.
Mais au préalable, un peu de « théorie ». Angular s'inscrit dans un choix d'application de type SPA (*Single Page Application*) : une seule page est présentée et ce qui distingue une page d'une autre c'est le contenu de la balise *ng-view* qui sera remplie par une vue. Ainsi, les en-têtes HTML (*meta*, *html*, *body*) ne sont présentes que dans la page index.html.

Le mécanisme de navigation va consister à demander à Angular d'afficher la bonne vue, un fichier HTML. Angular se chargera de la placer dans la balise *ng-view*.
Ainsi, pour naviguer vers une autre page (une vue en fait), il faut faire deux choses :

2.2.1 Ajouter une nouvelle route

Demandons au service de routage d'ajouter une nouvelle route vers une page d'accueil.

```
    .when('/accueil', {
        templateUrl :'vues/accueil.html',
        controller :'accueilController'
    })
```

La nouvelle route s'appelle *accueil*, le fichier associé est *accueil.html* et le contrôleur (optionnel) *accueilController.*
Ces lignes sont à insérer bien sûr après la première route.
Dans l'application cette page sera accessible à partir de sa route « accueil ».

2.2.2 Appeler cette route

Nous allons solliciter un nouveau service, *$location*, qui permet de naviguer vers la page indiquée grâce à sa méthode *url(" route")* ; par exemple *$location.url("mapage")* charge "mapage", si "mapage" a été définie comme une *route*.
Il faudra au préalable injecter l'objet dans le contrôleur :

```
app.controller('connexionController',function($scope,$location){
```

La page *accueil.html* ne contiendra, pour le moment, que le logo de GSB.

Travail à faire

- Ajouter la route « accueil » comme il est indiqué plus haut.
- Ajouter un nouveau fichier *accueil.html* qui ne contiendra que le logo de GSB ; pour cela, vous utiliserez la directive *ng-include*.
- Injecter le service *$location* dans le contrôleur de connexion (voir ci-dessus).
- Dans le contrôleur de connexion, appeler la méthode *url* sur l'objet *$location*, lorsque l'utilisateur se connecte correctement, afin de le diriger vers la page ‘‘*accueil*’’.
- Créer le contrôleur annoncé au moment de la définition de la nouvelle route. Ce contrôleur sera écrit dans le fichier *controllers.js* et ne contiendra aucun code.
- Tester

3 Le menu d'accueil

Cette page doit contenir un menu correspondant aux deux fonctionnalités évoquées en annexe : la gestion par le visiteur médical de « ses » médecins et la gestion de ses rapports de visite.

On désire obtenir une page qui ressemble à ceci :

Une barre de navigation apparaît en haut de la page ainsi que la légende au-dessus du logo.
Comme cette barre sera présente dans plusieurs pages, on créera un nouveau fichier HTML (*menuCommun.html*) qui ne contiendra (pour l'instant) que cette barre.
Pour ce faire, on va utiliser un composant *navBar* de BootStrap pour cela, visitez le site :

http://getbootstrap.com/components/ - navbar

Cette page d'accueil doit pouvoir ouvrir une page vers les médecins, et une vers les rapports de visite.

Comme c'est la fin de cette première partie, vous allez réaliser cet enchainement seul(e) en procédant de la manière suivante :

Travail à faire

- Créer le fichier *menuCommun.html* qui contiendra le composant *navBar* récupéré sur BootStrap.
- À l'intérieur de la balise *nav* vous ajouterez les *ul/li* afin de créer des liens *href* ordinaires ; on vous fournit la syntaxe pour l'un : *Gestion des rapports* (*rapports* est le nom de la route que vous devrez créer plus loin).
- Ajouter une propriété {{titre}} dans une balise *h3* afin de valoriser les légendes *gestion…*
- Ajouter les deux routes *rapports* et *medecins* qui pointeront sur les fichiers HTML *rapports.html* et *medecins.html*. Préciser le nom des deux contrôleurs *rapportsController* et *medecinsController*
- Créer les deux vues *rapports.html* et *medecins.html* qui ne contiendront que l'appel, grâce à *ng-include*, du fichier *menuCommun.html*
- Ajouter les deux contrôleurs évoqués qui ne feront que valoriser la propriété *titre*.

Ceci termine la partie 1 de la découverte d'Angular JS.
La correction se trouve dans le fichier décompressé **gsbAJSV1.1**

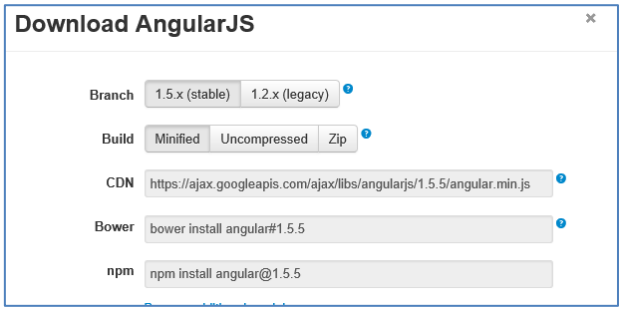
Annexe 1 : Utilisation du site officiel angularjs.org

Pour installer Angular JS, il faut se connecter au site officiel :

Nous utiliserons AngularJS 1 ; Angular 2 est très récent et peu pratiqué encore. Notons que la version 1 s'appelle Angular JS (JavaScript) mais que la version 2 s'appelle Angular ; en effet cette version recommande le langage TypeScript à la place de Javascript.

1 Installation

En cliquant sur AngularJS 1, plusieurs options sont disponibles :



Vous pouvez :

- Copier le lien CDN et le mettre dans votre application :
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.5.5/angular.min.js"> </script>
- Choisir, comme nous l'avons fait, de télécharger le fichier js. Dans ce cas vous avez 3 options :
  a) le code Minified, qui occupe moins de place sur le disque, mais qui est inexploitable,
  b) le code Uncompressed que l'on peut parcourir et éventuellement observer,
  c) Zip qui contient l'ensemble des *bibliothèques* entre autres.

Nous avons choisi de télécharger le fichier zip, ce qui nous permet en phase de développement de ne pas dépendre d'internet.

Le fichier zip contient toute la bibliothèque AngularJS et ceci aux deux formats, minified et uncompressed : d'abord le fichier angular.js (angular.min.js), noyau (le *core*) indispensable à tout projet mais aussi différentes bibliothèques à intégrer selon les besoins ; ainsi *angular-route.js* permet d'utiliser le service de routage d'AngularJS. Ainsi tout projet devra intégrer dans index.html, angular.js (ou angular.min.js) et seulement les bibliothèques utilisées.

Le zip contient également de la documentation dans le répertoire docs avec de nombreux exemples de code.

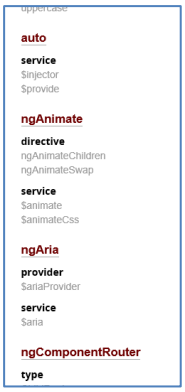
2 L'API

Vous avez un onglet dédié aux développeurs :

La partie API est celle qui fournit l'ensemble de la documentation officielle.

La partie de gauche commence par le paragraphe *ng* ; c'est le cœur d'Angular, celui qui est présent dans angular.js :

Les autres paragraphes (classés par ordre alphabétique) ne font pas partie du cœur ; il faut donc les ajouter à votre projet si vous en avez besoin :



Pour ajouter un fichier à votre projet, il faut le récupérer du fichier Zip téléchargé éventuellement plus haut ou faire figurer le lien, accessible lorsque vous sélectionnez un module supplémentaire :


```
<script src="//ajax.googleapis.com/ajax/libs/angularjs/1.5.3/angular-route.js"></script>
```

Dans cet exemple, la version 1.5.3 est utilisée.