AngularJS le MVC côté client

Par Marc AUTRAN

Date de publication : 2 avril 2015

Dernière mise à jour : 29 avril 2015

CONFIRMÉ

Durée : 1heure

Ce tutoriel présente le développement d'applications Web avec AngularJS. L'objectif de cet article est principalement de montrer comment concevoir une application Modèle - Vue - Contrôleur (MVC) à l'aide de ce Framework.**Commentez**

I - introduction

Le but de ce tutoriel est de présenter ce qu'offre AngularJS pour créer des applications basées sur le paradigme MVC côté client. Je ne présenterai pas AngularJS qui est en train de s'imposer comme le framework JavaScript de référence côté client.

Je ne présenterai pas non plus le paradigme MVC qui sépare le Modèle (les données) de la Vue (l'IHM). Le Contrôleur, quant à lui, assure la logique de contrôle et la gestion des événements. La connaissance de ce pattern est néanmoins un prérequis pour comprendre ce tutoriel.

On implémentera l'exemple basique d'une liste de clients que l'on pourra afficher et enrichir avec de nouveaux clients. Cet exemple nous permettra d'illustrer deux points essentiels d'AngularJS : le Databinding et l'injection de dépendances.

AngularJS étend le HTML5 pour le rendre dynamique en développant ses propres balises. Mais pour autant si l'on veut concevoir une véritable application MVC, il ne faut pas concevoir une vue pour la rendre dynamique, mais penser l'application en couches.

II - Installation

Le framework peut être téléchargé sur le site **https://angularjs.org/**.

Le framework réside entièrement dans le fichier trivialement dénommé « *angular.js* ».

L'application d'illustration sera une « Single Page Application » que l'on pourra écrire à l'aide de n'importe quel éditeur de texte. Pour ma part, j'ai utilisé Sublime Text.

L'exemple que nous suivrons sera de type Standalone pour des raisons pédagogiques. Mais nous aurions très bien pu héberger les données dans une base de données distante ou avoir recours à un Web Service. D'ailleurs, si cet article ne se concentre que sur le Databinding et l'injection de dépendances, et se veut donc résolument simple (tout s'exécute dans un navigateur), la consommation de Web Services en AngularJS fera l'objet d'articles futurs.

Il est également à noter que dans le cas d'une application en production, on ne téléchargera pas le fichier *angular.js,* mais on choisira plutôt d'indiquer son chemin sur le site AJAX de Google et sous sa forme minimisée (sans espace ni saut de ligne) autant pour des raisons de performances que pour disposer de la dernière version du framework. Cette ligne de code sera la suivante :

```
<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.16/angular.min.js"></script>
```

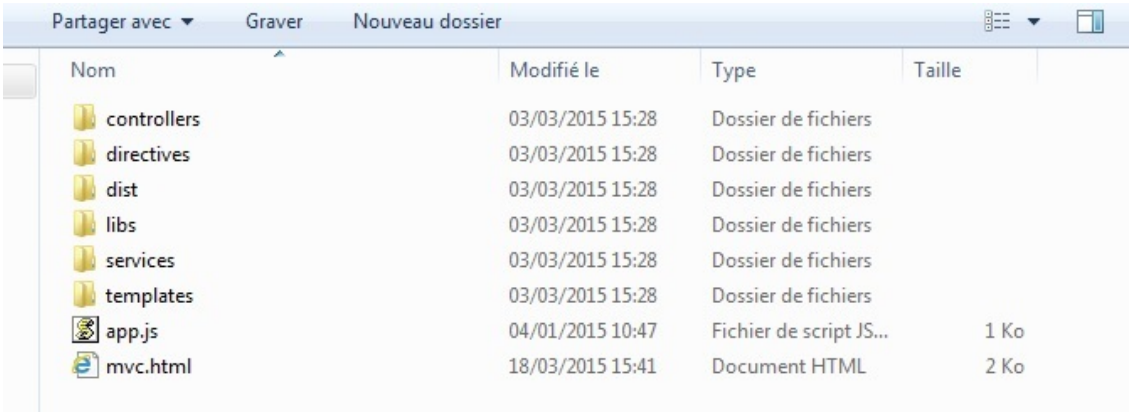
III - Le cahier des charges

On créera une application basique permettant dans une page HTML5 (la vue) d'afficher la liste des clients déjà présents dans le modèle et d'y ajouter de nouveaux clients. Comme ci-dessous :

Le seul objectif de cet exemple d'illustration est de manipuler des clients qui constituent le « modèle » et de les faire apparaître dans la « vue » grâce à la magie du « contrôleur ». On verra également comment ce même principe permet de créer des clients dans la vue et les mettre en persistance dans le modèle.

IV - L'architecture AngularJS

Voici l'architecture standard conseillée pour une application MVC depuis la racine de l'application :



On a l'habitude, pour développer avec AngularJS, de ranger les différents fichiers dans des répertoires conventionnels.

Le fichier *mvc.html* sera le fichier HTML5 que l'utilisateur connaîtra comme URL unique.

le fichier *app.js* est le fichier qui définit les modules AngularJS de notre application. Ces modules peuvent être vus comme des containers pour les différentes parties de l'application (au sens fonctionnel). On pourrait imaginer un module commande, facturation ou logistique. Ici nous n'aurons qu'un seul module, donc le fichier *app.js* ne contiendra que l'unique ligne de code suivante : `angular.module("app", []);` « *app* » sera le nom de notre module et on met entre *[]* les dépendances (éventuelles) du module.

Le modèle sera dans le répertoire *services.*

Le répertoire *directives* sera utilisé pour créer de nouvelles balises (directives) à utiliser dans la vue (*mvc.html*).

Le répertoire *templates* sera utilisé pour stocker les consignes d'affichage (esthétique) des *directives.*

Le répertoire *controllers* abritera les contrôleurs permettant de lier la vue et le modèle.

Le répertoire *libs* contiendra le fichier *angular.js* et le répertoire *dist* hébergera la bibliothèque *bootstrap* de Twitter que j'utilise pour la présentation HTML5.

V - Le modèle

Le modèle se trouve dans le fichier /services/clientsFactory.js :

```
1. angular.module("app").factory("clientsFactory", function ()
2. {
3.     var clients =     [
4.                         {id:1, nom: 'dupont', prenom: 'pierre'},
5.                         {id:2, nom: 'dupont', prenom: 'paul'},
6.                         {id:3, nom: 'dupont', prenom: 'jacques'},
7.                         {id:4, nom: 'dupont', prenom: 'alice'}
8.                     ];
9.     var getClients = function()
10.     {
11.         return clients;
12.     };
13.     var addClient = function(client)
14.     {
15.         var client = prepareClient(client);
16.         clients.push({id:client.id, nom:client.nom, prenom:client.prenom});
17.     };
18.     function prepareClient(client)
19.     {
20.         client.id = clients.length + 1;
21.         return client;
22.     }
23.     return {
24.         getClients: getClients,
25.         addClient: addClient
26.     };
27. });
```

Nous créerons en réalité un service (*clientsFactory*) que nous pourrons appeler partout dans notre application.

Ce service est créé grâce à la fonction *factory()* du framework qui permet réellement l'injection de dépendances.

Cette fonction *factory()* prend deux paramètres en entrée :

- le nom du service *clientsFactory ;*
- et une fonction dite de *callback.*

Depuis l'avènement d'AJAX et de son célèbre *XmlHttpRequest*, les fonctions de callback sont la clef de voûte des frameworks JavaScript côté client (comme AngularJS) ou côté serveur (comme NodeJS). Et tout ce que fait notre service se passe dans cette fonction.

La syntaxe de ce code source peut surprendre les habitués de la programmation impérative, car il s'agit là de programmation déclarative. Sans pousser trop loin la théorie, ce qu'il faut retenir de ce service, c'est que dans le *return* de sa fonction de *callback* il expose deux fonctions que l'on pourra appeler :

- *getClients* qui liste tous les clients de la liste ;
- *addClient* qui ajoute un client à la liste.

VI - Le contrôleur

Notre contrôleur se trouve dans le fichier *controller/mainController* :

```
1. angular.module("app").controller("mainController", function ($scope, clientsFactory)
2. {
3.     $scope.clients = clientsFactory.getClients();
4.
5.     $scope.addClient = function(client)
6.     {
7.         clientsFactory.addClient(client);
8.         $scope.newClient.nom='';
9.         $scope.newClient.prenom='';
10.     }
11. });
```

Nous créons un contrôleur (*mainController*) et là encore tout se passe dans la fonction de *callback*. Cette fonction prend deux paramètres d'entrée :

- *$scope* : défini par la communauté AngularJS comme un vecteur vers le modèle et comme un contexte d'exécution pour les expressions dans la vue ;
- et le service défini au chapitre précédent.

Dans le corps de la fonction, le contrôleur définit dans l'objet *$scope* la liste des clients et une méthode permettant de rajouter un client à cette liste. *$scope* étant la glu entre la vue et le contrôleur, la liste des clients et la méthode pour en ajouter sont disponibles dans la vue.

VII - Les directives

Nous allons créer une directive dont nous nous servirons dans la vue. Comme nous manipulons des clients, il serait intéressant de créer une nouvelle balise <client> </client>.

Nous créerons cette directive dans le fichier *directives/client.js* :

```
1. angular.module("app").directive("client", function ()
2. {
3.     return{
4.         restrict: 'E',
5.         templateUrl: 'templates/client.html'
6.     }
7. });
```

On remarque que la création de la directive *client* est accompagnée de sa fonction traditionnelle. Dans cette fonction, on précise que cette directive ne s'appliquera qu'aux éléments (*restrict : 'E'*) et où se trouve le modèle HTML qu'on souhaite appliquer à cette directive.

Jetons maintenant un coup d'œil à ce modèle *templates/client.html* :

client.html
```
1. <h3>{{client.nom}}</h3>
2. prenom:{{client.prenom}}
```

Cette notation entre deux accolades permet d'évaluer dans la vue des expressions AngularJS. Ici en particulier, on affichera le nom et le prénom du client.

VIII - La vue

La vue est constituée pour notre application d'une simple page HTML5 *mvc.html* :

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <link href="dist/css/bootstrap.css" rel="stylesheet">
        <title>Comprendre Angularjs</title>
        <script src="libs/angular.js"></script>
        <script src="app.js"></script>
        <script src="services/clientsFactory.js"></script>
         <script src="controllers/mainController.js"></script>
         <script src="directives/client.js"></script>
    </head>
    <body>
        <section  class="container" ng-app="app" ng-controller="mainController">
                <form name="newClientForm" class="well form-inline pull-right col-lg-5">
                    <legend>Ajouter un nouveau client</legend>
                    <fieldset class="row">
                        <label for="nom" class="col-lg-3">Nom :</label>
                        <input id="nom" type="text" style="width:150px" class="input-sm form-
control " ng-model="newClient.nom">
                    </fieldset>
                    <h1></h1>
                    <fieldset class="row">
                        <label for="prenom" class="col-lg-3">Prenom :</label>
                        <input id="prenom" type="text" style="width:150px" class="input-sm form-
control" ng-model="newClient.prenom">
                    </fieldset>
                    <h1></h1>
                    <fieldset class="row">
                        <button class="btn btn-
primary" type="submit" ng-click="addClient(newClient)"><span class="glyphicon glyphicon-ok-
sign"></span> Ajouter </button>
                        <button class="btn btn-
primary" type="reset"><span class="glyphicon glyphicon-remove-sign"></span> Annuler </button>
                    </fieldset>
                </form>
                <h2>Liste des Clients</h2>
                <article ng-repeat="client in clients">
                    <client />
                </article>
        </section>
    </body>
</html>
```

Dans la section d'entête, on trouve l'importation des scripts précédents et de la bibliothèque CSS Bootstrap.

Dans le corps du HTML, on remarque que toutes les instructions AngularJS sont préfixées par *ng-* . On déclare dans la balise <section> le module et le contrôleur.

On remarque comme il est aisé de mettre en place un Databinding bidirectionnel entre les propriétés du client et celles du formulaire à l'aide de l'instruction *ng-model.*

De même, on fera une boucle sur la liste des clients avec l'instruction *ng-repeat* afin d'afficher les informations des clients grâce à la directive <client> que l'on a créée.

IX - Conclusion

Cet article a permis d'exposer comment mettre en place simplement un MVC avec AngularJS. Ce n'est cependant qu'un début pour deux raisons :

- nous ne sommes pas dans cet exemple sur une architecture de type Internet. En effet, il faudrait que les données du modèle soient issues d'un serveur tiers, via un service REST qu'AngularJS sait parfaitement consommer à travers son objet *$resource* ;
- nous n'utilisons qu'une petite partie de ce qu'AngularJS sait faire. Parmi les apports intéressants d'AngularJS, ne citons que les *routes*, les *promises* et la manipulation d'AJAX.

Nous tenons à remercier **yahiko** pour la relecture technique et **milkoseck** pour la relecture orthographique de cet article.