AngularJS

Un framework Javascript MVC pour créer des Single Page Applications CRUD

Mais pas que....

Les frameworks MVC

Knockout.

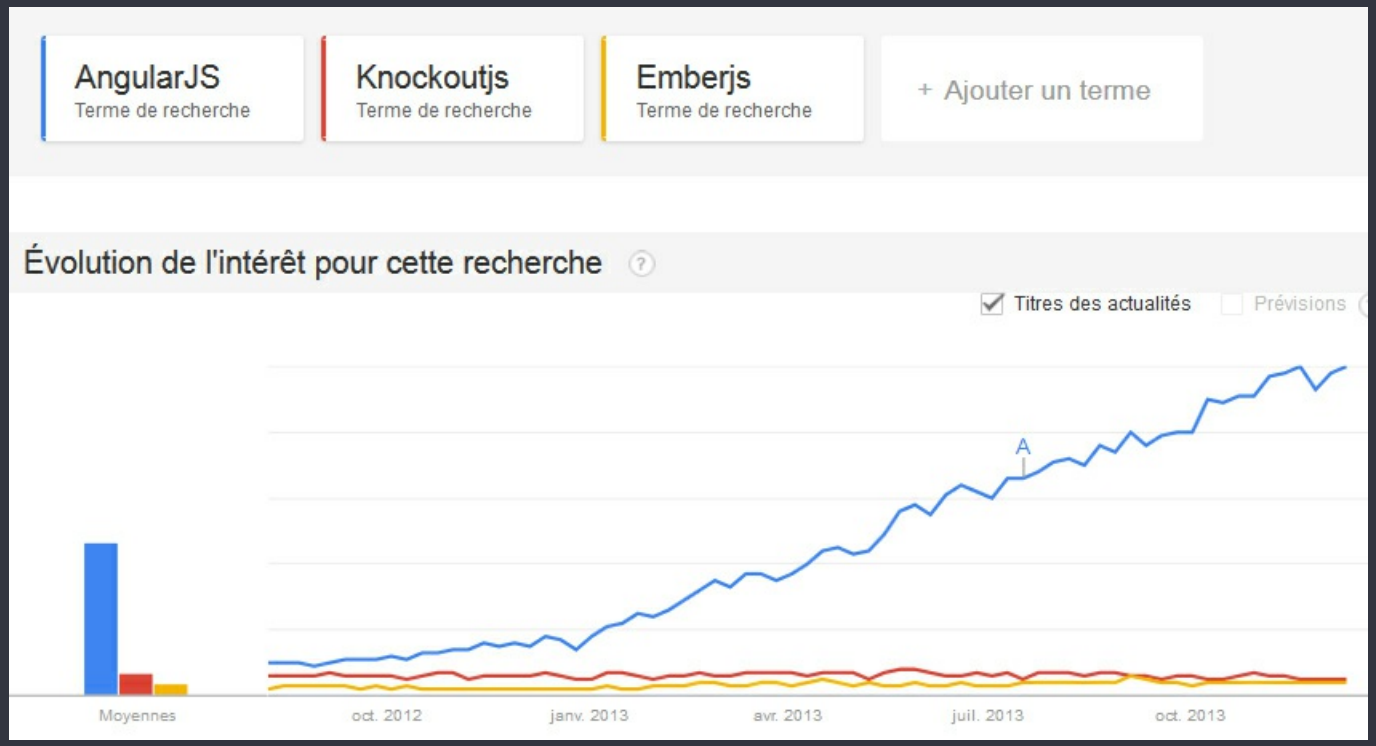
canjs

...

Pourquoi AngularJS

- Développé par Google
- Pensé dès le départ pour le test (unitaire et e2e)
- Data Binding (MVVM)
- Injection de dépendances
- HTML templating
- Étendre le HTML
- Agnostique
- ...

Le succès de AngularJS



Inconvénients

- La courbe d'apprentissage
- Les performances (Ça peut se discuter...)

Démarrer un projet

```
<!DOCTYPE html>
<html ng-app>
<head>
  <title>Mon application</title>
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.9/angular.js"></script>
</head>
<body>
    CONTENU
</body>
</html>
```

Les modules AngularJS

- ngRoute (angular-route.js)
- ngResource (angular-resource.js)
- ngTouch (angular-touch.js)
- ngAnimate (angular-animate.js)
- ngCookies (angular-cookies.js)
- ngSanitize (angular-sanitize.js)
- ngMock (angular-mocks.js)

Un simple Data Binding

```
<!DOCTYPE html>
<html ng-app>
<head>
  <title>Mon application</title>
  <script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.9/angular.js"></script>
</head>
<body>
  <input ng-model="name" type="text" placeholder="Ton nom">
  <h1>Hello {{ name }}</h1>
</body>
</html>
```

01.html

Directives ng-...

Mais, c'est quoi cette horreur: '*ng-app*' et '*ng-model*' ? Ça ne passera jamais la validation W3C !

```
data-ng-app

data-ng-model
```

Mais aussi

```
ng:app (namespace)
x-ng-app

ng:model
x-ng-app
```

Un simple Contrôleur

```
<body>
  <div class="container" ng-controller="myController">
    Démarré depuis: {{seconds}} seconde(s)
  </div>
</body>
```

```
function myController($scope) {
  $scope.seconds = 0;

  window.setInterval(function() { $scope.$apply($scope.seconds ++);}, 1000)
}
```

02.html

Les modules

- Parce qu'il est totalement déconseillé d'écrire du code fonctionnel dans le namespace global de Javascript.
- Afin d'encapsuler nos fonctionnalités dans des entités autonomes, facilement maintenables et testables.

Un module pour l'app.

```
<html ng-app="myApp">
...
</html>
```

```
var myApp = angular.module('myApp', []);
```

Le second paramètre permet de déclarer (injection de dépendance $injector) les autres modules dont dépend ce propre module.

03.html

Ajouter un contrôleur

```
var myApp = angular.module('myApp', []);

myApp.controller('MyController', function($scope){
   $scope.seconds = 0;

   window.setInterval(function() { $scope.$apply($scope.seconds ++);}, 1000);
})
```

03.html

$scope

Lorsque l'on crée un contrôleur, on lui injecte un objet $scope. Ainsi chaque contrôleur a son propre $scope.

On peut comparer le $scope d'un contrôleur avec le contexte Javascript d'une fonction.

Le $scope est la colle, fournie par le contrôleur, entre la vue (le HTML) et le modèle (les données). En fait, le $scope n'est rien d'autre que le VueModèle dans le patron MVVM.

Le HTML, par l’intermédiaire du Data Binding a donc accès à toutes les variables du $scope.

Les rôles du $scope

- Il fournit des "observers" afin de contrôler les changements sur le modèle (dirty checking). Le modèle va mettre à jour le $scope qui sera alors automatiquement répercuté sur la vue.
- Il fournit des fonctionnalité à la vue (fonctions).
- Les $scope peuvent être imbriqués (contrôleurs) mais ils peuvent aussi être isolé (directives).
- Il peut recevoir ou émettre des événements dans la hiérarchie des scopes ($emit, $broadcast, $on)
- Il fournit un environnement d’exécution (contexte) pour les expressions.
- $watch et $apply

Ex. Fonctionnalité

```
<button ng-click="alerte();">Message</button>
```

```
var myApp = angular.module('myApp', []);

myApp.controller('myController', function($scope) {
   $scope.alerte = function() {
      alert('Je suis là');
   }
});
```

04.html

$rootScope

De la même manière que Javascript possède un contexte global d’exécution, une application angularJS possède un $scope global : $rootScope.

Il est automatiquement crée avec l'application et il sera accessible dans les scopes imbriqués.

Il est, bien sûr, déconseillé d'utiliser directement le $rootScope. Il fournit cependant quelques fonctionnalités intéressantes ($locationChangeSuccess, $locationChangeStart, $routeUpdate, $ routeChangeStart...)

Ex. $rootScope

```
<div class="container" ng-controller="myController">
   Du fin fond des ténèbres nous entendions une voix: - <i>{{root}}</i>
</div>
```

```
var myApp = angular.module('myApp', []);

myApp.run(function($rootScope) {
   $rootScope.root = 'Je suis l\'alpha';
});

myApp.controller('myController', function($scope) {});
```

05.html

Remarque

Les nouvelles spécifications de Javascript ont introduit la méthode *Object.observe()* qui permet de déclencher une fonction lorsque que le contenu de l'objet, sur lequel est appliqué cet écouteur, est modifié.

AngularJS implémente cette fonctionnalité native de Javascript pour les navigateurs récents (donc plus rapide), dans le cadre du Data Binding.

C'est, entre autre, pour cette raison qu'il est déconseillé d'utiliser des types valeur (string, number, boolean) directement sur le $scope.

Exemple

```
<input type="text" ng-model="user.firstname">
<input type="text" ng-model="user.name">
Hello {{user.firstname + ' ' + user.name}} !
```

06.html

Les contrôleurs

- Un contrôleur est associé à une vue et permet de lui fournir un $scope (contexte partagé entre la vue et le modèle)
- Les contrôleurs peuvent être imbriqués et chaque contrôleur enfant peut accéder aux $scope parents

Contrôleurs imbriqués

```
<div class="container" ng-controller="parentController">
   <input type="text" ng-model="user.name">
   <div ng-controller="childController">
      <h4>{{content.title}}</h4>
      Hello {{user.name}} ! <br>
      <button ng-click="alerte();">Message</button>
   </div>
</div>
```

07.html

Contrôleurs imbriqués

suite...

```
var myApp = angular.module('myApp', []);

myApp.controller('parentController', function($scope) {
    $scope.alerte = function() {
        alert('Hello ' + $scope.user.name);
    }
});

myApp.controller('childController', function($scope) {
    $scope.content = {title: 'Bienvenue'};
});
```

07.html

Des contrôleurs simples

Afin de maximiser la maintenabilité et la testabilité de son code, on doit toujours faire en sorte que les contrôleurs soient les plus simples possible.

- Si un contrôleur devient trop long, on essaie de le scinder. Un contrôleur par fonctionnalité.
- Aucune manipulation de DOM dans les contrôleurs. C'est le rôle des directives.
- Les appels aux données doivent être refactorisées vers des services.

Les expressions

Elles sont souvent utilisées dans AngularJS, notamment pour le binding mais aussi comme paramètres pouvant êtres passés au directives.

On peut les rapprocher des expressions Javascript que l'on exécute avec *eval()* mais ce n'en sont pas. Elles ont leur propre syntaxe et sont parsées et exécutées par AngularJS (service *$parse*)

Leur contexte

Toute propriété ou méthode appelée dans une expression, fait référence au $scope (courant ou parents).

```
{{ma_variable}}
```

Revient à appeler

```
$scope.ma_variable;
```

Mots-clés, opérateurs

- true, false, null, undefined
- =, ==, ===, !=, <, <=, +, -, *, % ...
- Les boucles et structures conditionnelles Javascript ne peuvent pas être utilisées
- L'opérateur ternaire peut être remplacé par

```
condition && valeurTrue || valeurFalse
{true: valeurTrue, false: valeurFalse}[condition]
```

Filtres dans les express.

```
data | filterA:param1 | filterB | filterC:param1:param2
```

Voir chapitre sur les Filtres pour plus de détail.

Les Filtres

Les filtres permettent de formater les données que l'on va afficher à l'utilisateur.

Il sont appelé dans le binding HTML avec le caractère |

```
{{user.name | uppercase}}
```

Ils sont aussi accessibles dans le code JS avec le service $filter

```
$scope.user.name = $filter('uppercase')('Vincent'); /* VINCENT */
```

Les paramètres

Certains filtres acceptent des paramètres.

```
{{ data | myfilter:param1:param2 }]
```

```
$scope.data = $filter('myfilter')($scope.data, param1, param2);
```

Exemple:

```
{{123.33933 | number:2}} <!-- 123.34 -->
```

Les filtres AngularJS

AngularJS fournit un certain nombre de filtres

```
{{ today | date:'short' }}      <!-- 1/24/14 12:09 PM -->
{{ today | date:'yyyy' }}       <!-- 2014 -->

{{ 789123.4567 | number:2 }}    <!-- 789,123.46 -->

{{ 123 | currency }}            <!-- 123$ -->

{{ 'Vincent' | uppercase }}     <!-- VINCENT -->
{{ 'Vincent' | lowercase }}     <!-- vincent -->
```

Les filtres AngularJS

Suite...

```
{{ 'Je fais un essai' | limitTo: 2}}       <!-- Je -->
{{ 'Je fais un essai' | limitTo: -5}}      <!-- essai -->

{{ [1, 2, 3, 4] | limitTo: 2}}            <!-- [1,2] -->
```

Les filtres AngularJS

Suite...

```
{{ ['alpha', 'beta', 'gamma'] | filter:'e' }}   <!-- ['beta'] -->

{{  [{'nom': 'Nabet', 'prenom':'Vincent'},
    {'nom': 'Van Gogh', 'prenom':'Vincent'}] | filter:{'nom':'Nabet'}:true }}
    <!-- {'nom': 'Nabet', 'prenom':'Vincent'} -->  <!-- Rq: $ comme jocker clé -->

{{ ['alpha', 'beta', 'gamma'] | filter:fourCharac }}   <!-- ['beta'] -->
```

```
$scope.fourCharac = function(elem) { return elem.length === 4;}
```

Les filtres AngularJS

Suite...

```
{{  [{'nom': 'Durand', 'prenom':'Georges'},
    {'nom': 'Abitbol', 'prenom':'Georges'}] | orderBy:'nom' }}
    <!--Rq: tri inverse orderBy:'-nom' ou orderBy:'nom':true -->

{{  [{'nom': 'Durand', 'prenom':'Georges'},
    {'nom': 'Abitbol', 'prenom':'Georges'}] | orderBy:['nom', '-prenom'] }}

{{  [{'pieces':140, 'main':50}, {'pieces':30, 'main':150}] | orderBy:prixTotal }}
<!-- prixTotal renvoie la somme pieces+main -->
```

Chaîner les filtres

```
{{ 'le roi du monde' | uppercase | limitTo:6 }}  <!-- 'LE ROI' -->
```

Internationalisation

AngularJS fournit une longue liste de fichiers Javascript pour l'internationalisation de ses filtres (i18n).

```
<script src="vendor/angular/angular.min.js"></script>
<script src="vendor/angular-i18n/angular-locale_fr-fr.js"></script>
```

```
{{ 123456.33456 | currency}}          <!-- 123 456,33 € -->
{{ 1288323623006 | date:'fullDate'}}   <!-- vendredi 29 octobre 2010 -->
```

Créer ses propres filtres

```
myApp.filter('capitalize', function() {
   return function(text) {
      return text[0].toUpperCase() + text.slice(1);
   }
});
```

```
{{'jE SuiS Là' | lowercase | capitalize}}   <!-- Je suis là -->
```

08.html

Les formulaires

AngularJS permet de faire, très simplement, de la validation de formulaire:

- Pour contrôler les données qui y seront entrées et donc protéger notre application, côté client.
- Afin de fournir des indications visuelles à l'utilisateur.

ATTENTION: Cela ne dispense sûrement pas de faire des tests sur les données envoyées, côté serveur.

Pour cela, AngularJS utilise soit les versions surchargées des contrôles de validation HTML5 (required, min, max, email, url...), soit ses propres directives.

Formulaire de base

Le formulaire doit au moins avoir un *name*.

Il est conseillé d'ajouter l'attribut *novalidate* pour qu'il n'y ai pas de concurrence avec la validation native de HTML5.

```
<form name="form" novalidate>
</form>
```

Options de validation

```
<input type="text" required />
<input type="text" ng-minlength=5 />
<input type="text" ng-maxlength=20 />
<input type="text" ng-pattern="/^[a-zA-Z]+$/" />

<input type="email"/>
<input type="number" min="18" max="99"/>
<input type="url"/>
```

Note: Il est possible de créer ses propres directives de validation (*ngModel.$parsers, ngModel.$formatters, ngModel.$setValidity*)

Accès au formulaire

Dans la mesure où un formulaire a un attribut *name*, son contrôleur AngularJS associé est alors automatiquement accessible depuis le $scope courant, avec ce nom.

De la même manière, chaque champ sera accessible sur ce formulaire, grâce à son *name*.

```
<form name="form" novalidate>
    <input type="text" name="firstname">
</form>
```

```
$scope.form.firstname;
```

Contrôler le formulaire

On a alors accès à des propriétés qui vont nous permettre de contrôler les données entrées, en temps réel.

```
/* Formulaire modifié/non modifié */
formName.$dirty;                formName.$pristine;
formName.inputFieldName.$dirty;    formName.inputFieldName.$pristine;

/* Formulaire valide/invalide */
formName.$valid;                formName.$invalid;
formName.inputFieldName.$valid;    formName.inputFieldName.$invalid;

/* Objet contenant la liste des erreurs de validation */
formName.inputFieldName.$error; //ex: {number: false, min: true}
```

Appliquer un style

Lorsque AngularJS contrôle un formulaire, il ajoute automatiquement des classes CSS au formulaire et à ses champs, suivant leur état.

.ng-pristine, .ng-dirty, .ng-invalid, .ng-valid.

Pour une mise en forme plus avancée, il est possible de passer par la directive ng-class.

```
ng-class="{'has-error': form.email.$invalid && form.email.$dirty, 'has-success': form.email.$valid && form.email.$dirty}"
```

Exemple

```
<form name="form" novalidate>
  <div class="form-group">
    <label for="name" class="control-label">Nom</label>
    <input type="text" name="name" ng-minlength="3" ng-maxlength="20" ng-model="user.name" class="form-control">
    <div class="alert alert-danger" ng-show="form.name.$dirty && form.name.$invalid">
      <div ng-show="form.name.$error.minlength">Le nom doit faire au moins 3 caractères</div>
      <div ng-show="form.name.$error.maxlength">Le nom doit faire au max 20 caractères</div>
    </div>
  </div>...
```

09.html

Exemple

```
  <div class="form-group">
    <label for="email" class="control-label">Email</label>
    <input type="email" name="email" class="form-control" ng-model="user.email" required>
    <div class="alert alert-danger" ng-show="form.email.$dirty && form.email.$invalid">
      <div ng-show="form.email.$error.email">Adresse email incorrecte</div>
      <div ng-show="form.email.$error.required">Adresse email requise</div>
    </div>
  </div>
  <button type="submit" ng-click="submit()" class="btn btn-primary pull-right"
  ng-disabled="form.$invalid">Valider</button>
</form>
```

09.html

Les directives

Les directives sont l'une des fonctionnalités les plus puissantes de AngularJS.

Elles permettent d'étendre le HTML:

- En surchargeant des éléments existants comme <*form*>.
- En ajoutant de nouveau attributs comme *ng-disabled* ou encore *ng-class*.
- En créant de nouveaux éléments HTML.

Ainsi toute la base de AngularJS se fonde sur les directives car chaque élément de la page HTML (la vue) peut être traité comme une directive.

On parle ainsi parfois de "*composants*" pour parler des directives AngularJS.

Les directives natives

Elles sont nombreuses. Se référer à la documentation AngularJS.

- ng-href, ng-src, ng-disabled, ng-checked, ng-readonly, ng-selected, ng-class, ng-style
- ng-app, ng-controller, ng-view, ng-repeat
- ng-if, ng-switch...
- ng-bind, ng-model ...
- ng-change, ng-click, ng-select ...

Créer une directive

```
angular.application('myApp', [])
.directive('myDirective',
    // Factory
    function() {
      // Objet de définition de la directive
      return {
        // Options de définition
      };
});
```

Note: A la place de l'objet, on peut aussi renvoyer une fonction *link(scope, elem, attrs)*

Options de définition

Les options de définition de base d'une directive.

- restrict - E, A, C
- link(scope, elem, attrs) - Manipulation du DOM - jqLite
- template
- replace
- templateUrl
- scope - false, true, {} - @ (expression texte),= (expression valeur) ,& (expressions action)

10.html

Compile, link, controller

Compile est la phase durant laquelle la directive est compilée. Elle n'est exécutée qu'une seule fois quel que soit le nombre de fois où la directive sera utilisée. On y manipule le DOM mais on n'accède pas au scope. *Compile* doit renvoyer la fonction *link*.

Link est la phase durant laquelle le scope sera 'linké' sur le template HTML. Elle exécutée à chaque fois que la directive est utilisée. On peut manipuler le DOM et on accède au scope.

Controller. On utilise un contrôleur, en interne dans une directive, dans deux cas: Si plusieurs directives peuvent partager un même contrôleur. Ou, plus couramment, si la directive doit exposer une API à d'autres directives (directives imbriquées, voir l'option de definition '*require*')