

---

# Angular-js

## Table des matières

<b>I - Angular-Js (présentation du framework).....</b>	<b>3</b>
1. Présentation de AngularJs.....	3
<b>II - Bases de Angular-Js.....</b>	<b>6</b>
1. Structure fondamentale (template+ctrl+model).....	6
2. Quelques repères syntaxiques :.....	10
3. Héritage entre "scope".....	10
4. Binding de valeur(du modèle) via ng-bind.....	11
5. Pattern M-V-VM et "Binding" bi-directionnel.....	11
6. Filtrage avec "binding" bi-directionnel.....	12
7. affichage et activation selon conditions.....	14
8. Gestion des événements (réflexes).....	15
<tr ng-repeat="p in pays.liste"> <td ng-click="pays_doClick(p, \$event)"> .... </tr> .....	16
}.....	16
(selon le type de composant HTML).....	16
<b>III - Injection de dépendances , \$log et accès http.....</b>	<b>17</b>
1. Injection de dépendance (js , angular-js).....	17
2. Service \$log pour générer des lignes de logs.....	21
3. Appels distants (\$http , XHR).....	22
<b>IV - Navigations (route, changement de vues).....</b>	<b>23</b>
1. Changement de vues (via \$routeProvider).....	23

2. Switch de vue à un (sous-)sous-niveau quelconque.....	25
<b>V - Accès aux services distants (REST).....</b>	<b>27</b>
1. Appels de services REST depuis Angular-Js.....	27
2. invocation de web-services "REST" via \$resource.....	28
<b>VI - Aspects divers et avancés de Angular-Js.....</b>	<b>35</b>
1. Validation et soumission contrôlée de formulaire.....	35
2. Tests unitaires (avec jasmine et angular-mocks).....	38
3. Programmation de directives personnalisées.....	41
4. Animations graphiques avec ngAnimate.....	45
<b>VII - Annexe – Rappels HTML et CSS.....</b>	<b>48</b>
1. Pages HTML.....	48
2. Feuilles de styles (CSS, ...).....	49
<b>VIII - Apports de HTML5.....</b>	<b>51</b>
1. Versions officielles et interprétations effectives.....	51
2. Grandes lignes de l'évolution vers HTML5.....	51
3. Mise en page html5 (sémantique + styles css).....	51
4. apports et précisions au niveau des formulaires.....	54
5. API "Canvas" pour les images vectorielles.....	56
6. Graphisme évolué et multimédia.....	56
<b>IX - Annexe – Web Services REST (coté serveur).....</b>	<b>57</b>
1. Généralités sur Web-Services REST.....	57
2. Limitations Ajax sans CORS.....	63
3. CORS (Cross Origin Resource Sharing).....	64
<b>X - Annexe – Env. Dev. , MongoDB , JsonP.....</b>	<b>67</b>
1. Environnement de développement AngularJs.....	67
2. MongoDB.....	77
3. JONP (JSON Padding) – requêtes inter-domaines.....	80
<b>XI - Annexe – énoncés (précis) de TP.....</b>	<b>85</b>
1. Calcul de Tva (m-v-vm) et expérimentations \$scope.....	85
2. Test d'événements + ng-show , ng-disabled.....	86
3. Test de \$route vers "subview".....	89

# I - Angular-Js (présentation du framework)

## 1. Présentation de AngularJs

### 1.1. AngularJs et MVVM

**AngularJs** est un framework web de google qui est basé sur le langage javascript et sur le design pattern "MVVM" (*Model-View-ViewModel*) .

Proche des patterns MVC (Model-View-Controller) et MVP (Model-View-Presenter) , le pattern MVVM permet de synchroniser automatiquement la vue (présentation HTML) par rapport à un modèle de données "javascript" (en arrière plan dans le navigateur web). Ce modèle "javascript" sera généralement mis à jour suite à des appels de web services "REST" (avec des résultats en "JSON") .

Etant donné que la racine du modèle doit avoir une partie stable et que celle-ci est liée à une page html, la navigation s'effectue généralement en passant d'une sous-vue à une autre (contenu interchangeable de `<div ng-view> ...</div>`).

Les principaux intérêts de la technologie angularJs sont les suivants :

- une grande partie des traitements web s'effectue coté client (dans le navigateur) et le serveur se voit alors déchargé d'une lourde tâche (refabriquer des pages , gérer les sessions utilisateurs, ...) ---> bien pour tenir la charge .
- meilleurs performances/réactivités du coté affichage/présentation web (navigateur) : c'est directement l'arbre DOM qui est réactualisé/rendu à partir des modifications apportées sur le modèle javascript (plus de html à transférer/ré-analyser).
- séparation claire entre la partie "présentation" (js) et la partie "services métiers" (java ou ".net" ou ".php" ou ...) . Google présente d'ailleurs angularJs comme un framework MVW (Model-View-**Whatever**) .

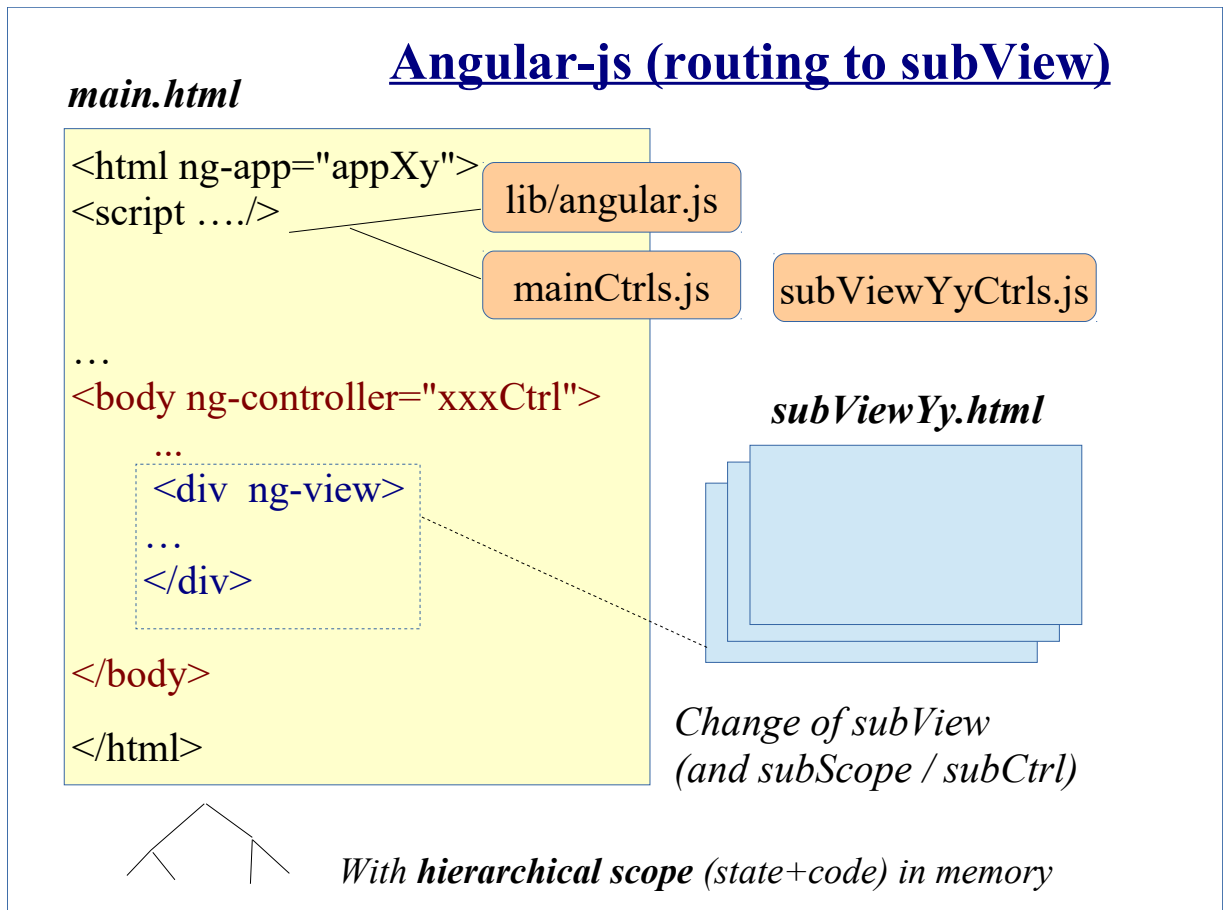
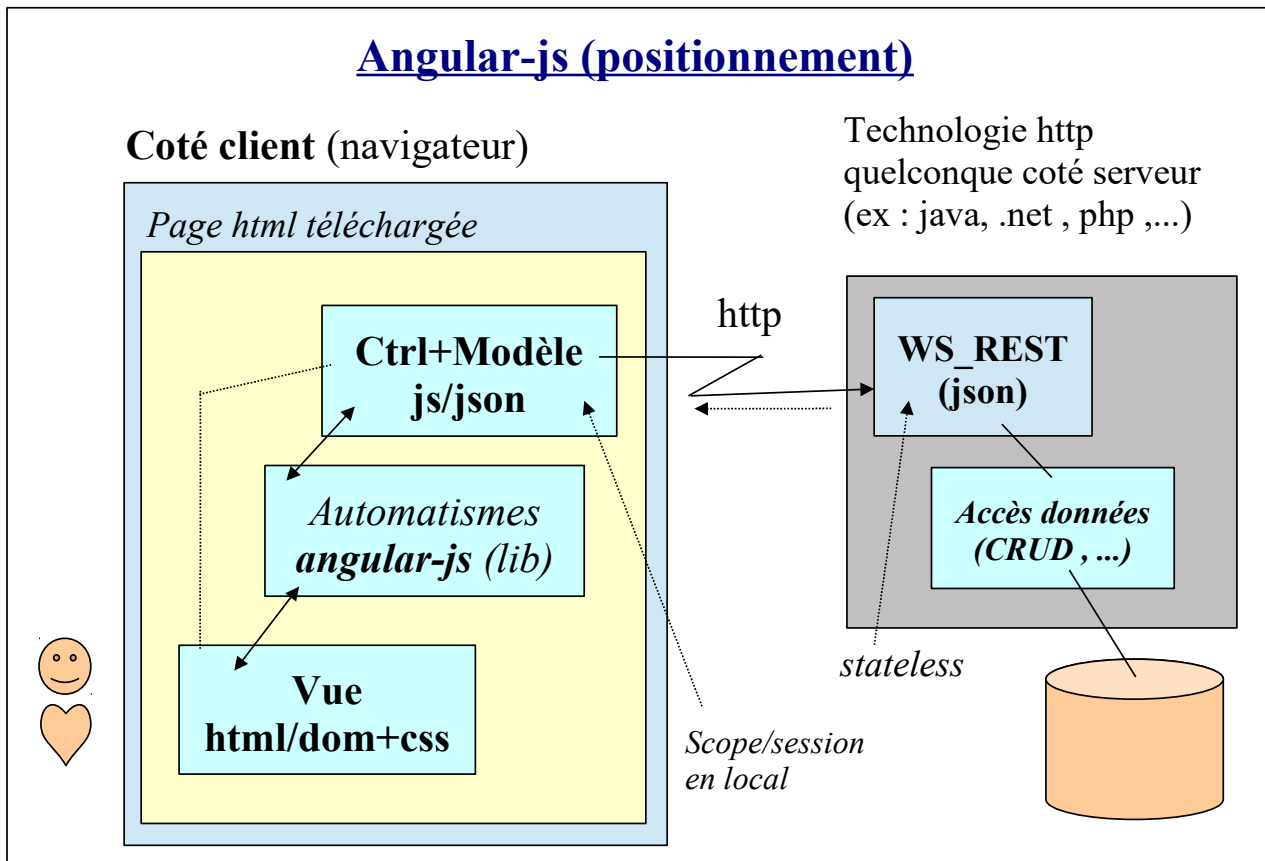
Les petits inconvénients de la technologie angularJs sont :

- un langage "javascript" qui n'est pas très simple à débbugger et besoin de connaître deux langages ("javascript" coté client et "java ou php ou c# ou ..." coté serveur (pour WS REST)) .
- besoin d'ajouter éventuellement un mécanisme pour cacher le code source (s'il est un peu confidentiel)
- besoin d'ajuster certains paramétrages de sécurité au niveau des services REST (authentification , ....).

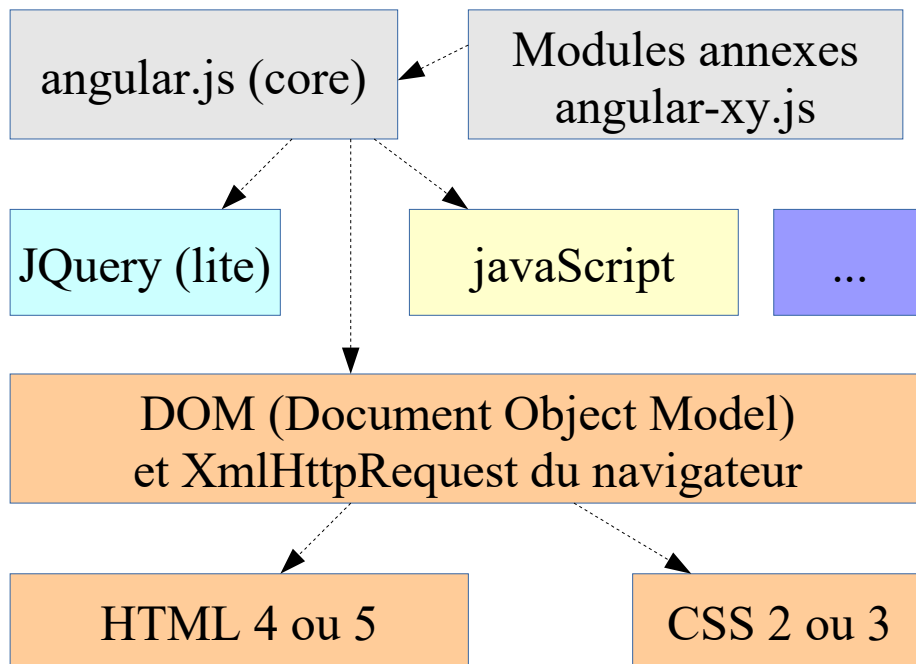
NB : \* angularJs s'appuie en interne sur la partie light de jquery .

\* framework proche de Angular-js : knockout-js .

## 1.2. Architecture de AngularJs



## Dépendances d'angular-Js



**NB:** jQuery (lite) est inclus dans angular.js (core) , le reste (HTML , CSS , DOM , javascript) est géré par le navigateur internet

Structure conseillée (arborescence des fichiers) :

```
main.html
subView1.html
subView2.html
lib/
  angular/
    angular.js
data/xxx.json
  yyy.json
js/mainCtrls.js
  Ctrls1.js
  Ctrls2.js
```

```
main.html
<html ng-app="....App">
  <head>
    <script src="lib/angular/angular.js"></script>
    <script src="js/mainCtrls.js"></script>
  </head>
  <body ng-controller="....Ctrl">
    ....
  </body>
</html>
```

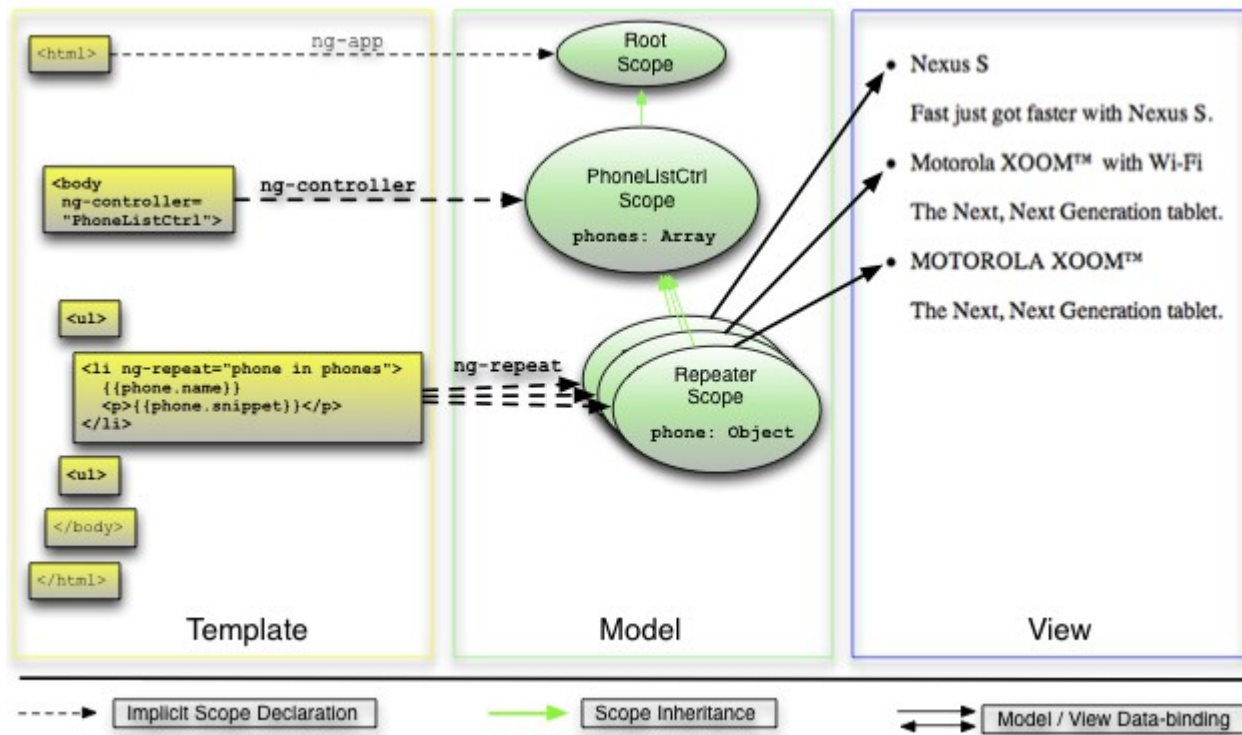
## II - Bases de Angular-Js

### 1. Structure fondamentale (template+ctrl+model)

#### 1.1. Principaux éléments d'une application "angular-js"

<i>Eléments</i>	<i>Positionnements/syntaxes</i>	<i>Utilités/rôles</i>
<b>Template</b>	Page html avec parties en {{ }}	Modèle de vue(s) html 4 ou 5 à générer
<b>Model</b>	Structure de données en javascript souvent orientée objet (code js , json , ...) rattachée à un \$scope et généralement gérée par un contrôleur	<ul style="list-style-type: none"> <li>* Structure de données brutes (généralement sans mise en forme) à afficher</li> <li>* souvent récupérées par des appels XHR/ajax vers des services REST/json</li> </ul>
<b>Contrôleur</b>	Bloc de code javascript prenant en charge un modèle de données et quelques gestionnaires d'événements	<ul style="list-style-type: none"> <li>* Point d'entrée pour le code des structures de données</li> <li>* Comportement de l'application (ou d'une de ses sous parties)</li> </ul>
<b>Scope</b>	Zone mémoire (javascript) comportant des données (à état) qui sont associées à une partie (ou sous partie) de la page HTML	<ul style="list-style-type: none"> <li>* Permet d'exprimer des chemins relatifs simples vers des données à états</li> <li>* espace de noms logiques associés à des données structurées</li> </ul>

NB : Les "scopes" et les "contrôleurs" de angular-js sont explicitement ou implicitement liés à un nœud de l'arbre DOM représentant la structure de la page HTML (issue du template).  
Les "scopes" sont ainsi automatiquement organisés de façon hiérarchique :



**Exemple simple** (inspiré du tutoriel officiel de angular-js) :

#### *simpleProductList.html*

```

<html ng-app="myAjsApp">
  <head>
    <script src="lib/angular/angular.js"></script>
    <script src="js/mainCtrls_v1.js"></script>
  </head>
  <body ng-controller="ProductListCtrl">
    <div id="divProd">
      <h3> {{title}} </h3>
      <ul>
        <li ng-repeat="prod in products">
          {{prod.name}}
          <p>{{prod.label}} , date:{{prod.date}} ,
            price:{{prod.price}}</p>
        </li>
      </ul>
    </div>
    <div id="divStatus">
      <hr/>
      total number of products :
      <b>{{products.length}}</b>
    </div>
  </body>
</html>

```

#### **list of smartphones**

- iphone 5C

very good smartphone , date:2013-10-12 ,  
price:600

- galaxy S4 mini

good smartphone of Samsung with  
medium screen , date:2013-05-23 ,  
price:350.5

- ultym 5

cheap and good smartphone by Bougues-  
Telecom , date:2014-05-23 , price:120

---

total number of products : 3

js/mainCtrls\_v1.js

```

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

myAjsApp.controller('ProductListCtrl', function ($scope) {

  $scope.products = [
    { "name": "iphone 5C",
      "label": "very good smartphone",
      "date" : "2013-10-12",
      "price" : 600 },
    { "name": "galaxy S4 mini",
      "label": "good smartphone of Samsung with medium screen",
      "date" : "2013-05-23",
      "price" : 350.50 },
    { "name": "ultym 5",
      "label": "cheap and good smartphone by Bougues-Telecom",
      "date" : "2014-05-23",
      "price" : 120 }
  ];

  $scope.title = "list of smartphones";
});

```

### Quelques exemples d'expressions angular-js :

```

<html ng-app="myCarreApp">
  <head>
    <script src="lib/angular/angular.js"></script>
    <script>
      var myAjsApp = angular.module('myCarreApp', []);
      myAjsApp.controller('BasicCtrl', function ($scope) {
        // basic (empty) controller
      });
    </script>
  </head>
  <body ng-controller="BasicCtrl">
    <table border="1">
      <tr> <th>i</th> <th>i*i</th> </tr>
      <tr ng-repeat="i in [0,1,2,3,4,5,6,7,8,9]" >
        <td>{{i}}</td> <td>{{i*i}}</td>
      </tr>
    </table>
  </body>
</html>

```

i	i*i
0	0
1	1
2	4
3	9
4	16
5	25
6	36
7	49
8	64
9	81



```
</body>
</html>
```

Autre version (fonctionnellement identique):

```
<html ng-app="" ng-init="numbers=[0,1,2,3,4,5,6,7,8,9]">
  <head>
    <script src="lib/angular/angular.js"></script>
  </head>
  <body>

    <table border="1">
      <tr> <th>i</th> <th>i*i</th> </tr>
      <tr ng-repeat="i in numbers" >
        <td>{{i}}</td> <td>{{i*i}}</td>
      </tr>
    </table>

  </body>
</html>
```

La directive **ng-init** permet d'**initialiser** des valeurs au sein du modèle .

## 2. Quelques repères syntaxiques :

Toute balise (ou attribut de paramétrage) spécifique à angular-js est appelé(e) "**directive**".

Toutes les directives prédéfinies de angular-js commencent par "**ng-**" (ex : **ng-controller** , **ng-app** , **ng-repeat** , ....) . il est cependant possible de programmer de nouvelles directives personnalisées (qui ne devraient normalement pas commencer par "ng-").

Etant donné que la valeur (de paramétrage) d'une directive est toujours interprétée par angularJs , il n'est pas nécessaire d'imbriquer la syntaxe `{{ }}` entre les simples ou doubles quotes :

Exemple:

**ng-model="montantHt"** mais pas **ng-model="{{montantHT}}"**

A l'inverse , le paramétrage des attributs HTML standard (qui ne commencent pas par ng- ) doit s'appuyer sur une syntaxe en `{{...}}` pour pouvoir accéder aux éléments du modèle du scope courant.

Exemples:

`<body bgColor="{{myBackgroundColor}}" ...>`

`<span style="font-style : {{myFontStyle}}" >...</span>`

## 3. Héritage entre "scope"

Soit une page classiquement organisée en plusieurs "niveaux" (de contrôleurs , ...)

```
...
<body ng-controller="MainCtrl">
  <h1> {{title}} .... </h1>
  <div id="divXy" ng-controller="XyCtrl">
    <h3> {{title}} ... </h3>
  ...
```

====> Les valeurs affichées par `{{title}}` correspondent à celles qui sont gérées par le contrôleur du niveau courant et peuvent être différentes.

NB: Si un sous-niveau n'affecte pas explicitement de valeur spécifique à un élément du modèle existant au niveau parent , il y a alors un héritage implicite de type "copy on write" :

- le sous-niveau partage (en lecture seule) l'élément de son parent (sans préfixe à préciser)
- si le sous-niveau affecte explicitement une valeur à un élément hérité , une nouvelle zone mémoire distincte sera alors construite et utilisée dans le sous-niveau.

NB: **\$scope.\$parent.xy** permet d'accéder (en lecture/écriture) à la donnée "xy" du scope parent.

## 4. Binding de valeur(du modèle) via ng-bind

```
<p>Enter your Name: <input type="text" ng-model="name"></p>
```

<!-- le nom saisi est automatiquement envoyé dans la partie "name" du modèle -->

```
<p>Hello <span ng-bind="name"></span></p>
```

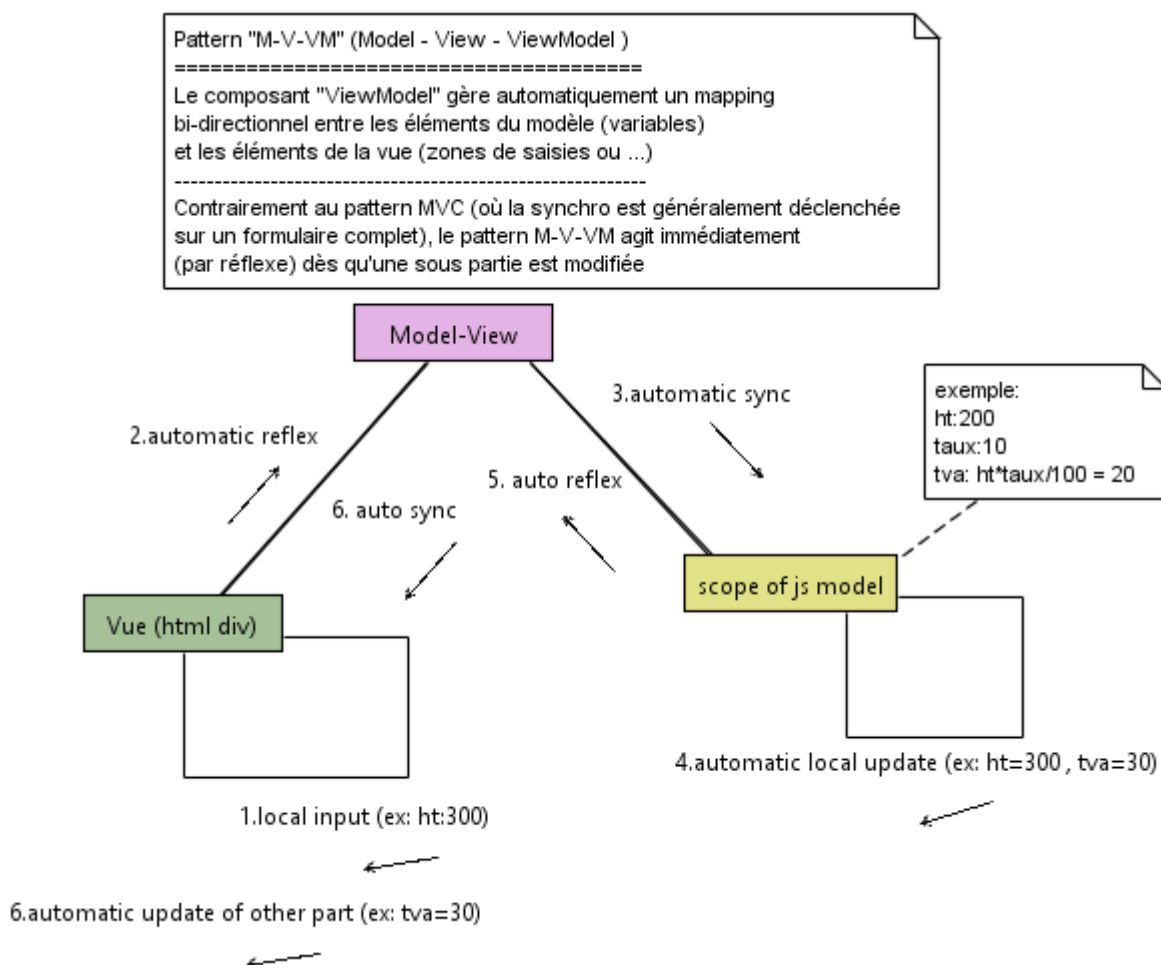
<!-- la valeur de la partie "name" du modèle est reliée à la balise "span" avec réactualisation automatique de la valeur →

Autre exemple :

```
<p>Enter a number: <input type="text" ng-model="num"></p>
```

```
<p><span ng-bind="num"></span> * 10 = <span ng-bind="num * 10"></span></p>
```

## 5. Pattern M-V-VM et "Binding" bi-directionnel



## 6. Filtrage avec "binding" bi-directionnel

### 6.1. Filtrage

```
...
<body ng-controller="ProductListCtrl">
<div id="divParam">
  filter_query : <input ng-model="filter_query" /> <br/>
</div>
  <div id="divProd">
    <h3> {{title}} </h3>
    <ul>
      <li ng-repeat="prod in products | filter:filter_query">
        {{prod.name}}
        <p>{{prod.label}} , date:{{prod.date}} , price:{{prod.price}}</p>
      </li>
    </ul>
  </div>
</body>
...
```

Via l'attribut (directive angular) **ng-model="..."**, la valeur saisie dans la zone de texte préfixée par le label "filter\_query :" est automatiquement placée dans la variable de nom "**filter\_query**".

Le suffixe " | **filter: nomVariable** " placé au bout de l'expression de la liste associée à la directive **ng-repeat="..."** permet de filtrer les produits en ne retenant que ceux qui comportent la sous expression saisie.

filter\_query :

**list of smartphones**

- galaxy S4 mini

good smartphone of Samsung with medium screen , date:2013-05-23 , price:350.5

### 6.2. Binding bi-directionnel (explications)

Le nom de variable "filter\_query" précisée au niveau de la directive **ng-model="..."** est en fait interprété comme une partie nommée du modèle de données à mettre à jour en arrière plan (au sein du scope courant).

Les automatismes "Model-View-ViewModel" de angular-js font que la valeur saisie au niveau de la vue se retrouve automatiquement injectée dans le modèle en arrière plan.

Inversement, le nom de variable d'une partie de l'expression " | **filter: nomVariable** " de la directive **ng-repeat** du template est interprété comme une partie nommée du modèle de données dont

il faut tenir compte pour (re-)générer ou actualiser la vue HTML.

Les automatismes "Model-View-ViewModel" sont bi-directionnels et une modification partielle de la vue (valeur saisie ou sélection ou ...) engendre une mise à jour automatique du modèle et *par ricochet* une autre partie de la vue se retrouve alors automatiquement ré-actualisée.

### 6.3. ré-ordonnancement (tri)

*ex2b\_product\_list\_with\_filter\_and\_order.html*

```
<html ng-app="myAjsApp">
  <head> <script src="lib/angular/angular.js"></script>
    <script src="js/mainCtrls_v2_orderProp.js"></script>
  </head>
  <body ng-controller="ProductListCtrl">
    <div id="divParam">
      filter_query : <input ng-model="filter_query" /> <br/>
      Sort by: <select ng-model="orderProp">
        <option value="name">Alphabetical</option>
        <option value="-date">Newest</option>
        <option value="price">Cheaper</option>
      </select>
    </div>
    <div id="divProd"> <h3> {{title}} </h3>
    <table border='1'>
      <tr> <th>name</th><th>label</th><th>date</th><th>price</th></tr>
      <tr ng-repeat="prod in products | filter:filter_query | orderBy:orderProp">
        <td> {{prod.name}} </td><td> {{prod.label}} </td>
        <td> {{prod.date}} </td><td> {{prod.price}} </td>
      </tr> </table> </div>
    </body>
  </html>
```

filter\_query :

Sort by:

#### list of smartphones

| name           | label   | date       | price |
|----------------|---|------------|-------|
| ultym 5        | cheap and good smartphone by Bougues-Telecom  | 2014-05-23 | 120   |
| iphone 5C      | very good smartphone                          | 2013-10-12 | 600   |
| galaxy S4 mini | good smartphone of Samsung with medium screen | 2013-05-23 | 350.5 |

*mainCtrls\_v2\_orderProp.js*

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

myAjsApp.controller('ProductListCtrl', function ($scope) {
  $scope.products = [{...},{...},{...}];
  $scope.title = "list of smartphones";
  $scope.orderProp = '-date'; //by default , -date : ordre décroissant , date : ordre croissant
```

});

## 7. affichage et activation selon conditions

### 7.1. Directive ng-show

La directive **ng-show** permet d'exprimer une condition booléenne qui sera automatiquement interprétée (ou ré-interprétée) par *AngularJs* pour afficher (ou pas) une certaine zone de la page html. La directive **ng-hide** existe également (avec une interprétation inversée de l'expression booléenne).

Exemple :

```
operation:<select ng-model="operation">
    <option>multiplication</option>
    <option>division</option>
</select>
<hr/>
<div ng-show="operation == 'multiplication'">
    <h4>multiplication</h4>
    <input type="text" ng-model="a"> * <input type="text" ng-model="b"> = {{a*b}}
</div>
<hr/>
<div ng-show="operation == 'division'">
    <h4>division</h4>
    <input type="text" ng-model="a"> / <input type="text" ng-model="b"> = {{a/b}}
    <span style="color:red" ng-show="b==0"> division par zero interdite !!! </span>
</div>
```

operation:

**division**

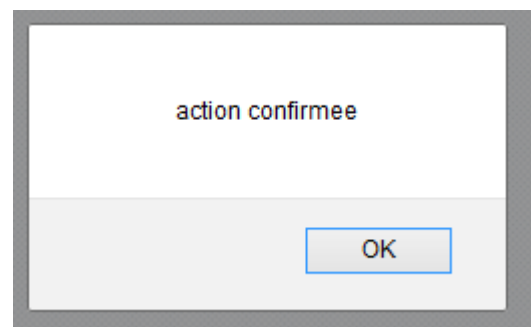
/  = null **division par zero interdite !!!**

## 7.2. Directive ng-disabled

Selon le même principe que ng-show , la directive **ng-disabled** permet de rendre une zone "inactive / grisée" si une expression booléenne est à "true".

```
<input type="checkbox" ng-model="confirmation" > confirmation </input>
<input type="button" value="action" ng-disabled="!confirmation"
  onClick="alert('action confirmee')"/>
```

☐ confirmation  ☒ confirmation



## 8. Gestion des événements (réflexes)

### 8.1. Exemple simple

```
...
<hr/>
  selected prodId: <input ng-model="prodId" /> <br/>
  <button ng-click="refreshSelectedProdWithRest()">
    refresh selected Prod (with or without rest)</button><br/>
  selected product: [ {{prodId}} ] {{selectedProd}}
<hr/>
...
```

```
...
myAjsApp.controller('ProductListCtrl', [ '$scope' , function ($scope) {
  ...
  $scope.refreshSelectedProdWithRest = function (){
    $scope.selectedProd = ..... ; // get or simulate values from $scope.prodId
  };
  ...
}]);
```

...

## 8.2. Fonction avec objet et/ou événement en paramètre :

```
<tr ng-repeat="p in pays.liste"> <td ng-click="pays_doClick(p, $event)"> .... </tr>
```

```
$scope.pays_doClick = function(item, event) {  
    alert("clicked on " + item.nom + " @ " + event.clientX + ": " + event.clientY);  
}
```

## 8.3. Principaux événements gérés par AngularJS :

(selon le type de composant HTML)

- **ng-click**
- **ng-dbl-click**
- **ng-mousedown**
- **ng-mouseup**
- ng-mouseenter
- ng-mouseleave
- ng-mousemove
- **ng-mouseover**
- **ng-keydown**
- **ng-keyup**
- **ng-keypress**
- **ng-change**



## III - Injection de dépendances , \$log et accès http

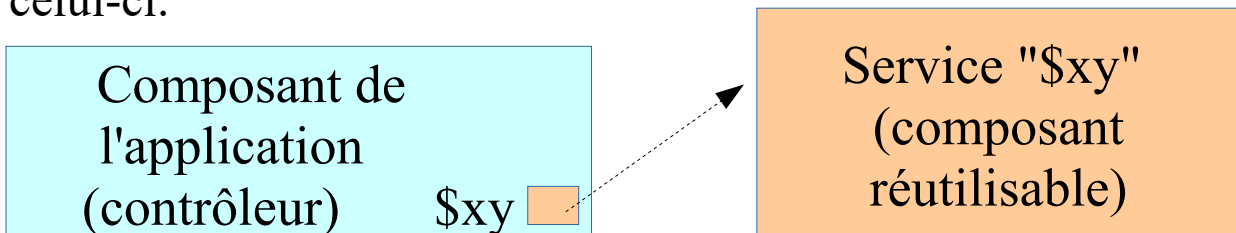
### 1. Injection de dépendance (js , angular-js)

#### 1.1. Rappel du principe d'injection de dépendance

##### Principe d'injection de dépendance

**L'injection de dépendance consiste à relier entre eux deux niveaux de composants (codes orientés objets) compatibles.**

Le composant de haut niveau utilise le sous composants (dont il dépend) en arrière plan via une référence variable vers celui-ci.



L'initialisation de cette référence peut éventuellement être automatiquement prise en charge par un framework (ex : spring ou angular-js) selon certaines règles ou certains paramètres.

#### 1.2. Injection de dépendance adaptée au langage javascript

Certains langages de programmation (tels que C++ , java , C#) sont fortement typés et les injections de dépendances automatiques peuvent être basées sur des mises en correspondance entre des références et des composants existants qui ont des types de données compatibles (ex : instance d'une classe *ServiceXxImpl* implémentant l'interface *IServiceXx* compatible et une référence à injecter de type *IServiceXx* ).

Javascript étant un langage faiblement typé , les injections automatiques ne peuvent simplement être basées que sur des correspondances de noms (ex : nom de paramètre d'une fonction et nom d'un composant/service existant quelque-part) et des exceptions sont à prévoir en cas d'incompatibilité.

Soit f1 une fonction javascript , on peut récupérer tout son code source via `f1.toString()` ;  
Un framework javascript (tel que angular-js) peut donc analyser (via un découpage basé sur des expressions régulières) les noms des paramètres pour ensuite effectuer des mises en correspondances automatiques.

## 1.3. Service pré-définis de angular-js (pouvant être injectés)

Par convention, les scopes et les services pré-définis de angular-js ont des noms qui commencent par le caractère \$ .

### Principaux services de angular-js :

|   |  |
|---|--|
| <b>\$http</b>                                     | Communication <b>http</b> (modes get,post,put,delete,...) via le composant <b>XmlHttpRequest</b> du navigateur (habituellement utilisé pour ajax)  |
| <b>\$resource</b><br>( <i>ngResource module</i> ) | Service de haut niveau fabriquant/gérant des ressources (données distantes) en s'appuyant en interne sur le sous service \$http et sur les conventions des web services "REST" .   |
| <b>\$location</b>                                 | Service d'accès en lecture/écriture à l'url courante (window.location)   |
| <b>\$log</b>                                      | Service de log (par défaut vers la console)  |
| <b>\$route</b><br>( <i>ngRoute module</i> )       | Service pour associer une URL de sous vue (html partial) à un contrôleur angular-js .permet d'effectuer un "switch" d'une sous partie de la page.<br><br>(NB : ce service est généralement configuré via son fournisseur <b>\$routeProvider</b> . Un " <i>provider</i> " ne peut être injecté que dans la fonction <b>config()</b> d'un module applicatif) . |
| ...   |  |

Ces différents services sont situés et trouvés dans le "scope" racine (hérité par les autres "scopes") et pourront être injectés dans les paramètres (de mêmes noms) des fonctions "contrôleurs" .

## 1.4. Injection de dépendance au niveau de angular-js

**L'injection de dépendance effectuée par angular-js** s'effectue en fonction d'**une mise en correspondance automatique entre les noms des paramètres d'une fonction javascript** prise en charge par le framework (**contrôleur**) **et les noms des objets/composants/services instanciés dans le scope courant** (en tenant compte des éléments hérités).

→ au final , le nom des paramètres est plus important que l'ordre de ceux ci .

Le mécanisme d'injection d'angular-js , déclenche si besoin l'instanciation d'un composant/service nécessaire via une fabrique spéciale (service factory) .

D'autre part , il peut éventuellement y avoir plusieurs niveaux d'injections (déclenchés automatiquement par transitivité) .

Exemple :

```
myAjsApp.controller('ProductListCtrl', function ($scope,$http) {  
  
    $http.get('data/products.json').then( function(response) {  
        $scope.products = response.data;  
    });  
    $scope.title = "list of smartphones";  
});
```

Dans l'exemple ci-dessus , lorsque la fonction **function (\$scope,\$http) { ... }** est prise en charge par angular-js , les paramètres \$scope et \$http sont automatiquement initialisé en y injectant le scope courant (\$scope) et le service \$http pré-défini .

## 1.5. Contournement des problèmes liés à la minification (minimisation / compression) du code

Certains mécanismes de minimisation/compression du code javascript sont quelquefois utilisés (en production) pour optimiser la bande passante et les performances au niveau des transferts de fichiers entre le serveur Http et le navigateur.

Ces mécanismes dits de "minification" retirent tous les aspects non essentiels (commentaires , espaces inutiles , ...) et simplifient le noms des paramètres des fonctions (habituellement non significatifs en javascript).

Etant donné que les mécanismes d'injections de dépendances de angular-js sont basés sur des correspondances de noms au niveau des paramètres de certaines fonctions , la "minification" du code peut à priori poser de sérieux problèmes.

De façon à contourner ce problème potentiel, Angular-Js propose des syntaxes alternatives qui sont robustes et fiables vis à vis de la "minification" du code javascript :

```
...  
function ProductListCtrl ($scope,$http) {  
    ... // fonction javascript ordinaire avec paramètres ordonnés  
}  
  
ProductListCtrl.$inject=[ '$scope' , '$http' ]; //paramétrage explicite des futures injections  
  
myAjsApp.controller('ProductListCtrl', ProductListCtrl) ;//prise en charge par AngularJs .
```

ou bien

```
...  
function ProductListCtrl ($scope,$http) {  
    ... // fonction javascript ordinaire avec paramètres ordonnés  
}  
  
myAjsApp.controller('ProductListCtrl', [ '$scope' , '$http' , ProductListCtrl ] ) ;  
//tableau avec chaînes des éléments à explicitement injecter + fonction.
```

ou bien encore (avec fonction "inline" anonyme) :

```
...  
myAjsApp.controller('ProductListCtrl', [ '$scope', '$http', function ($scope,$http) {  
...  
} ]);
```

Dans toutes les variantes ci-dessus , les noms des éléments à injecter sont précisés sous forme de chaîne de caractères (littérales) qui ne seront jamais modifiées/simplifiées par une éventuelle minimisation/compression du code javascript.

Avec cette approche explicitant clairement les éléments à injecter , les paramètres de la fonction '....Ctrl' pourraient éventuellement être renommés autrement mais il vaut mieux garder les mêmes noms pour garantir une bonne lisibilité .

## 2. Service \$log pour générer des lignes de logs

Le service prédéfini **\$log** de angularJs permet de générer simplement des lignes de logs qui s'affichent dans la console web du navigateur.

Ce service comporte différentes méthodes (classiques) associées aux différents niveaux de logs :

|                         |   |
|-------------------------|---|
| <b>.log</b> (message)   | ligne de log quelconque                         |
| <b>.warn</b> (message)  | warning   |
| <b>.info</b> (message)  | information (comportement normal)               |
| <b>.error</b> (message) | erreur / exception (avec contexte "stackTrace") |
| <b>.debug</b> (message) | trace pour le debug                             |

Exemple :

```
angular.module('logExample', [])
.controller('LogController', ['$scope', '$log', function($scope, $log) {

$scope.title = "list of smartphones";

$log.log("a new logged line");
$log.info("ok - normal info");
$log.warn("my warning : attention ");
$log.error("my erreur : ... ");
$log.debug("title (as debug):" + $scope.title);
}]);
```

Résultats (dans la console web du navigateur) :

```
"a new logged line"
"ok - normal info"
"my warning : attention "
"my erreur : ... "
"title (as debug):list of smartphones"
```

### 3. Appels distants (\$http , XHR)

#### 3.1. XHR (XmlHttpRequest) via service \$http et données "json"

data/products.json

```
[
{
  "name": "iphone 5C",
  "label": "very good smartphone (Mac)",
  "date" : "2013-10-12",
  "price" : 600
},
{
  "name": "galaxy Ace",
  "label": "old good Android smartphone of Samsung",
  "date" : "2011-05-23",
  "price" : 130
}
]
```

mainCtrls\_v3\_xhr\_json\_data.js

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

myAjsApp.controller('ProductListCtrl', function ($scope,$http) {

  jsonDataUrl = 'data/products.json';
  // jsonDataUrl = 'services/rest/products/allJson'; // vers ws rest java ou ...

  //NB : l'ancien $http.get(...).success (function(data){ .... }) ; est "deprecated"
  //      et n'existe carrément plus dans les version récentes d'AngularJs
  $http.get(jsonDataUrl).then( function(response) {
    $scope.products = response.data;
  });

  $scope.title = "list of smartphones";
});
```

Les données (au format "json") récupérées par ajax (\$http) au bout de l'url "data/products.json" sont placées dans la variable nommée "products" du modèle (de la portée courante \$scope) .

## IV - Navigations (route, changement de vues)

### 1. Changement de vues (via \$routeProvider)

La page html principale à besoin d'être stable de façon à chapeauter toute une application "angular-js". Au sein de cette page principale "main.html", au moins une sous partie marquée via `<div ng-view />` pourra automatiquement être remplacée par une sous vue (template html partiel) et gérée par un contrôleur spécifique.

L'url relative de la sous vue active sera automatiquement actualisée au niveau de la barre d'adresse du navigateur et des "bookmarks" précis pourront ainsi être enregistrés.

Toutes ces fonctionnalités sont gérées par le **module** annexe "**ngRoute**" de angular-js (à charger depuis le fichier lib/angular/**angular-route.js**).

Ce module comporte **\$routeProvider** pour la configuration et les services \$route et \$routeParams.

**NB:** depuis la version 1.6 les URL permettant un changement de route sont en "**#!...**" plutôt qu'en "**#/....**" ( v <=1.5 ).

#### 1.1. Structuration globale

```
main.html
partials/subview1.html
      subview2.html
js/mainApp.js
  controllers.js
```

**main.html**

```
<html ng-app="myAjsApp">
  <head>
    <script src="lib/angular/angular.js"></script>
    <script src="lib/angular/angular-route.js"></script>
    <script src="js/mainApp.js"></script>
    <script src="js/controllers.js"></script>
  </head>
  <body>
    <div ng-view>
    </div>
  </body>
</html>
```

## 1.2. Configuration des sous vues via \$routeProvider

*js/mainApp.js*

```
var myAjsApp = angular.module('myAjsApp', [ 'ngRoute','myAjsControllers' ] );

// app module 'myAjsApp' depends on [ 'ngRoute','myAjsControllers' ] other modules

myAjsApp.config(['$routeProvider',
  function($routeProvider) {
    $routeProvider.
      when('/products', {
        templateUrl: 'partials/product_list_view.html',
        controller: 'ProductListCtrl'
      }).
      when('/products/:prodId', {
        templateUrl: 'partials/product_details_view.html',
        controller: 'ProductDetailCtrl'
      }).
      otherwise({
        redirectTo: '/products'
      });
  });

// when local relative url '/products' , '/products/:prodId' (in web browser) will be
// interpreted by Angular-js and ngRoute module (with $route and $routeParams services) ,
// partial template will be loaded in <div ng-view> and specific controller will be associated
```

## 1.3. Sous templates partiels et contrôleurs associés

*js/controllers.js*

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

myAjsControllers.controller('ProductListCtrl', function ($scope,$http) {

  $http.get('data/products.json').success(function(data) {
    $scope.products = data;
  });

  $scope.orderProp = '-date'; //by default , -date : ordre decroissant , date : ordre croissant

  $scope.title = "list of smartphones";

});

myAjsControllers.controller('ProductDetailCtrl', ['$scope', '$routeParams',
  function($scope, $routeParams) {
    $scope.prodId = $routeParams.prodId;
```



```
});
```

partials/*product\_details\_view.html*

```
<i>detail view</i> for product number <b>{{prodId}}</b> <br/>
...
```

partials/*product\_list\_view.html*

```
<div id="divProd">
  <h3> {{title}} </h3>
  <ul>
    <li ng-repeat="prod in products">
      <a href="#/products/{{prod.id}}">{{prod.name}}</a> <!-- url depuis v1.6 avec #!/... -->
      <p>{{prod.label}}, date: {{prod.date}}, price: {{prod.price}}</p>
    </li>
  </ul>
</div>
<div id="divStatus">
  <hr/>
  total number of products : <b>{{products.length}}</b>
</div>
```

/ex4\_mainAppRoutingToSubView.html#/products

### list of smartphones

- [iphone 5C](#)

;/ex4\_mainAppRoutingToSubView.html#/products/3

detail view for product number 3

...

## 2. Switch de vue à un (sous-)sous-niveau quelconque

action: <select ng-model="*renderPath*" >

```
<option value="a1"> liste des comptes </option>
```

```
<option value="a2">effectuer un virement</option>
```

```
</select>
```

```
<div ng-switch on="renderPath">
```

```
<div ng-switch-when="a1">
```

```
<ng-include src=" 'partials/pourEspaceXy/a1.html' " />
```

```
</div>
```

```
<div ng-switch-when="a2">
```

```
<ng-include src=" 'partials/pourEspaceXy/a2.html' " />
```

```
</div>
```

```
<div ng-switch-default > <p> *** </p> </div>
```

`</div>`

La directive **ng-switch** permet de permuter explicitement de "sous-vue" (sous "div") à un niveau (ou sous-niveau) quelconque donné .

**ng-switch** est pratique pour un **sous-sous-niveau** .

Il n'y a par défaut pas d'effet sur l'url apparente au niveau du navigateur mais on a en contre-partie une très grande liberté dans l'organisation des templates et des contrôleurs :

- soit un même grand contrôleur pour plusieurs sous-sous-niveaux alternatifs ("switchables").  
soit plusieurs petits contrôleurs.
- soit des imbrications directes de balises ( à l'intérieur de `<div ng-switch-when="....">` )  
soit des inclusions de sous-sous-templates "html/angularJs" .

NB:

`<ng-include src=" expression retournant url" />`  
*avec éventuelle valeur fixe via '...' au sein de "...."*

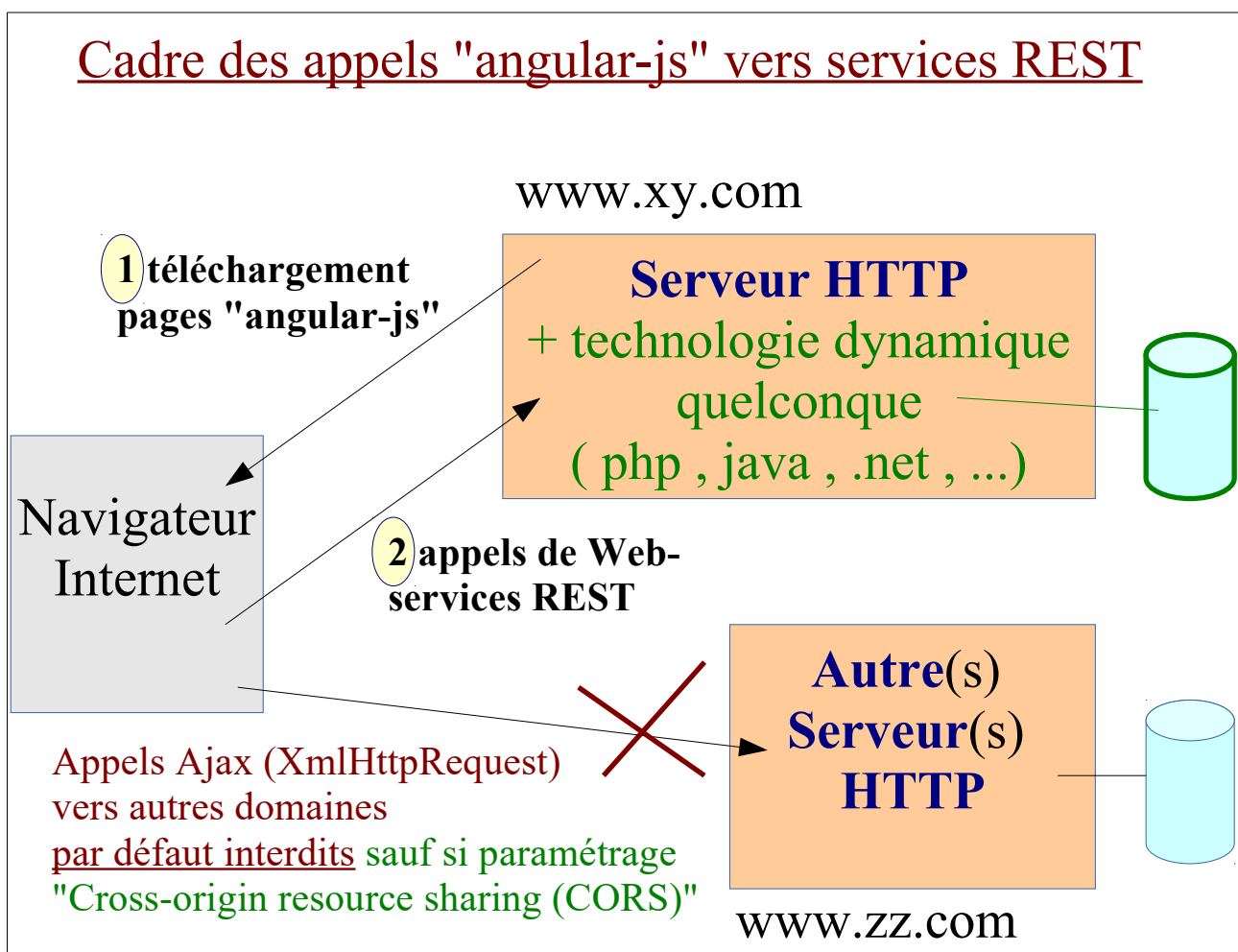
# V - Accès aux services distants (REST)

## 1. Appels de services REST depuis Angular-Js

La technologie angular-js positionnée du côté client (navigateur) doit pouvoir communiquer efficacement avec une technologie http coté serveur (ex : java/jee , .net , php , ...) pour indirectement accéder en lecture/écriture à une base de données centrale.

Angular-js étant basé sur le langage javascript , la façon la plus naturelle et efficace pour communiquer avec le coté serveur (http) consiste à invoquer des services web "REST" retournant des données au format json .

### Cadre des appels "angular-js" vers services REST



## 1.1. Rappel: récupération de données json via \$http

*mainCtrls\_v3\_xhr\_json\_data.js*

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

myAjsApp.controller('ProductListCtrl', function ($scope, $http) {

  //jsonDataUrl = 'data/products.json';
  jsonDataUrl = 'services/rest/products/allJson'; // vers ws rest java ou ...

  $http.get(jsonDataUrl).then( function(response) {
    $scope.products = response.data;
  });
});
```

## 2. invocation de web-services "REST" via \$resource

Le module **ngResource** d'angular.js comporte un service central dénommé **\$resource** qui permet de manipuler simplement des "proxy" vers des ressources distantes "REST" (en s'appuyant sur le service de bas niveau \$http).

### 2.1. Structure de code conseillée/classique

```
<html ng-app="myAjsApp">
  <head>
    <script src="lib/angular/angular.js"></script>
    <script src="lib/angular/angular-resource.js"></script>
    <script src="js/mainCtrls_with_rest_service.js"></script>
    <script src="js/myServices.js"></script>
  ...
</html>
```

Le module "ngResource" codé dans le fichier lib/angular/**angular-resource.js** doit être chargé. Un module personnalisé comportant des services d'accès aux données "REST" doit être codé et chargé (ici : *js/myServices\_for\_v5.js* ).

## 2.2. Module de services pour l'accès aux données distantes

js/myServices.js

```
var myServices = angular.module('myServices', ['ngResource' ]

myServices.factory('Product', ['$resource', function($resource){

  return $resource('services/rest/json/products/:idRes' /*default common url*/,
    { idRes:'@id' } /*default params */,
    {
      queryXy: { url : 'services/rest/json/xxx/yyyy/:value' /*redefined url */,
                method:'GET' , isArray:true} ,
      update : {method:'PUT'},
    } /* actions / methods */
  );
}
]);

myServices.factory('XY', ... ) ;
```

La classe d'objet fabriquée via `productServices.factory('Product', ...)` et `return $resource(...)` ici appelée '**Product**' correspond à une sorte de **mixte** entre:

- \* une **classe de données** 'Product' (dont la structure est récupérée via REST) et qui a (via \$resource) des *facultés de persistance à distance (via REST)* : méthode `$save()` , `$remove()` , ... , `$update()`
- \* une **classe de service** permettant de déclencher des recherches : `Product.query(...)` , `Product.queryXy()` , `Product.get()` , ...

Sans aucune précision/redéfinition , une classe fabriquée via `$resource(...)` hérite des actions/méthodes suivantes par défaut :

```
{ 'get': {method:'GET'},
  'save': {method:'POST'},
  'query': {method:'GET', isArray:true},
  'remove': {method:'DELETE'},
  'delete': {method:'DELETE'} };
```

qui peuvent (entre autres syntaxes possibles) s'utiliser simplement de la façon suivante :

```
ClasseXy.get ( {idRes : 123 } ).$promise.then(...) ; //findByPrimaryKey (objbyId )

obj.$delete(); or obj.$remove(); //suppression à distance (ws-rest)

obj.price = $scope.newPrice ;
obj.$save() ; //saveOrUpdate

//recherche (tout ou bien selon critères ) et récupère un résultat de type arrayOfObjects:
ClasseXy.query ( ) .$promise.then(...) ; // or .query ( { searchCriteria1 : value1 , ... } ) ;
```

Le principal paramétrage de `$resource(...)` est l'URL commune (ou par défaut) associée à un

### service REST .

Rappel : d'après les conventions "REST" et d'après les attentes / pré-supposés d'angular-js , les opérations classiques CRUD se déclenchent habituellement comme ceci :

Type de requêtes	Méthode HTTP	URL classique identifiant la/les ressource(s) distante(s)	Contenu du corps (body) invisible de la requête	Réponse JSON
<b>Recherche multiple</b>	<b>GET</b>	.../products .../products?crit1=v1&crit2=v2	vide	Liste/tableau d'objets
<b>Recherche par id</b>	<b>GET</b>	.../products/idRes (idRes=1,...)	vide	Objet JSON
Ajout (seul)	<b>POST</b>		Objet JSON	Objet JSON avec id quelquefois calculé (incr)
Mise à jour (seule)	<b>PUT</b>	.../products/idRes	Objet JSON	Statut ou ...
<b>SaveOrUpdate</b>	<b>POST</b>	.../products/idRes	Objet JSON	Objet JSON modifié (id)
<b>suppression</b>	<b>DELETE</b>	.../products/idRes	vide	Statut ou ...
Autre	...	.../products/.../:action/....	...	...

L'url de base a bien souvent un format stable . Seule la partie finale varie (souvent en précisant ou pas d'identifiant précis d'une ressource à gérer et en précisant ou pas des critères de recherche).

Quelques exemples d'url (avec syntaxe attendue par angular-js) :

".../products/:idRes.:format" avec :format = "xml" ou "json"  
 ".../products/:idRes.json"  
 ".../json/products/:idRes" //le plus simple suffit souvent !!!

### Remarques importantes :

Si un seul paramètre (préfixé par:) est présent en fin de l'url (ex : "idRes") et que plusieurs valeurs sont fixées par défaut ou durant l'appel (ex : ex : { idRes= 'xxx' , a = '12' , b = '3' } )  
 alors l'url sera complétée par des critères de recherche (ex : ".../json/products/xxx?a=12&b=3" ).

Si la valeur affectée à un paramètre est préfixée par @ , alors la valeur résultante correspondra à la valeur le l'attribut/champ/propriété de l'objet dont le nom est préfixé par @ (ex : @id pour partie ".id" d'un objet de type 'Product' )

la valeur d'un paramètre variable en fin d'url ne peut pas comporter de "/" .  
 D'où le besoin de redéfinir quelquefois une url avec un "/" en plus au niveau d'une action/méthode  
 On peut également définir explicitement une fin d'url complexe du type  
 ".../products/:idRes/:action/:idAction" mais c'est difficile à gérer et pas très classique .

La méthode save() prend toujours en entrée l'objet (json) à sauvegarder et renvoie en retour l'objet

(json) dont les valeurs sont potentiellement modifiées (ex: clef primaire auto-incrémentée) .  
Plusieurs syntaxes sont possibles lors de l'appel :

```
obj.$save() ;
Product.save(obj) ;
Product.save({idRes = obj.id } , obj )
```

...

Par défaut (d'un point de vue angular-js) , la méthode save() en mode "POST" a une sémantique de **"saveOrUpdate"** mais rien n'empêche de considérer save() comme un createNew et ajouter update() en mode "PUT" . update() ( en mode PUT) est généralement invoqué en passant l'id en fin d'URL

delete() et remove() ont la même sémantique , remove est moins problématique si delete est interprété comme un mot clef. Il faut généralement passer l' id en fin d'URL pour préciser ce qu'il faut supprimer.

---> **Product.remove({idRes=3})** ; ou bien **obj.\$remove()** ; //avec idRes valant @id par défaut

### 2.3. contrôleur utilisant un service basé sur \$resource

js/mainCtrls\_with\_rest\_service.js

```
var myAjsApp = angular.module('myAjsApp' , ['productServices' ]
myAjsApp.controller('ProductListCtrl', [ '$scope' , 'Product' , function ($scope,Product) {

$scope.maximumPrice = '2000';//by default

// génère une url = .../products?maxPrice=2000
Product.query( { maxPrice : $scope.maximumPrice } ).$promise.then(
  function (value) { $scope.products = value ; }
);

$scope.prodId='0'; //by default

Product.get( { idRes : $scope.prodId } ).$promise.then(
  function (value) { $scope.selectedProd = value ; }
);

...

$scope.newPrice='200'; //by default

$scope.updatePrice = function (){
  $scope.selectedProd.price = $scope.newPrice;
  $scope.selectedProd.$update();
};

}]);
```

Autres exemples :

```
//Product.remove({idRes:3}); //ok
```

```
//$scope.products[2].$delete(); //ok
```

```
/*
var newProd =new Product();
newProd.name='xxx';
newProd.id=0; //peut être mieux à faire ? (en concordance avec interprétation coté serveur) ?
newProd.label="yyyyy";
newProd.price=$scope.newPrice;
newProd.date="2014-09-06";
//Product.save(newProd);
newProd.$save();//ok with id=0 in object (or maybe with String id whose value may be null)
*/
```

.....html (partiel)

```
...
selected prodId: <input ng-model="prodId" /> <br/>
<button ng-click="refreshSelectedProdWithRest()">refresh selected Prod (...) </button> <br/>
selected product: [ {{prodId}} ] {{selectedProd}}
<hr/>
new price : <input ng-model="newPrice" /> <br/>
<button ng-click="updatePrice();refreshSelectedProdWithRest()">update price</button> <br/>
...
```

selected prodId:

selected product: [ 2 ] { "id":2,"name":"galaxy S4 mini","label":"good screen","date":"2014-01-01","price":253 }

new price :



## 2.4. Appels asynchrones avec résultats récupérés via "\$promise"

Le code basic et intuitif suivant ne fonctionne pas bien :

```
$scope.doVirementAndRefresh = function () {
    $scope.transfert = Transfert.save($scope.transfert);
}
```

**Problème:** l'objet transfert n'est pas immédiatement mis à jour (en mode synchrone) lors de l'appel à Transfert.save(...) et celui ci garde toujours sa valeur initiale (juste avant l'appel).

L'explication du "non fonctionnement apparent" tient tout simplement dans le fait qu'un appel (direct ou **indirect**) via XMLHttpRequest s'effectue de manière asynchrone et le résultat doit alors être récupéré en différé via une fonction "callback" .

```
$scope.doVirementAndRefresh = function () {
    Transfert.save($scope.transfert).$promise.then(
        //success
        function( value ){
            $scope.transfert = value;
            if($scope.transfert.ok){
                $scope.message="virement de " + $scope.transfert.montant +
                " euros bien effectue du compte "
                + $scope.transfert.numCptDeb + " vers le compte "
                + $scope.transfert.numCptCred;
                //+ RAZ du montant pour éviter deux transferts consécutifs (par erreurs)
                $scope.transfert.montant=0; $scope.transfert.ok=null;
                ....
            }
        },
        //error
        function( error ){
            alert(error);
        }
    ); //fin de $promise.then()
};
```

**\$promise.then()** permet de préciser **deux méthodes callback** permettant de **récupérer le résultat différé théoriquement promis en fin d'opération asynchrone**.

La **première méthode "callback"** gère un **résultat positif (succès)** et la **deuxième "callback"** est prévue pour **gérer une erreur**.

Ces 2 "callbacks" constituent des paramètres facultatifs (mais néanmoins conseillés) de la méthode then() . On pourra ainsi omettre le traitement d'erreur dans des cas ultras simples (sans fiabilité importante exigée). Le paragraphe suivant (qui détaillera un peu l'API "promise") montrera qu'en cas d'enchaînement , en omettant volontairement la callback de succès dans le dernier "then()" , on se retrouve alors dans logique de catch d'exception.

## 2.5. Détails sur l'API "\$promise"

L'API des promises garantit que :

1. la méthode *then()* d'une promise prend deux callbacks en paramètres, tous deux optionnels ; le premier sera appelé si la promise est résolue avec succès, le second en cas d'erreur.
2. on peut éventuellement enregistrer plusieurs paires de callbacks (succès et erreur), en appelant plusieurs fois la méthode *then()* d'une promise
3. si plusieurs paires de callbacks ont été enregistrées sur une même promise, la résolution de cette promise déclenchera une seule fois la callback adéquate de chacune des paires, et **forcément dans l'ordre où les paires de callbacks ont été enregistrées**
4. on peut appeler la méthode *then()* d'une promise déjà résolue, dans ce cas la callback adéquate sera appelé **immédiatement** - mais jamais avant la fin de la méthode *then()*
5. une promise ne peut être résolue qu'une seule fois, donc toutes les callbacks verront le même résultat, ou la même erreur
6. la méthode *then()* renvoie une nouvelle promise, qui représente le résultat différé de la callback appelée . → ceci permet de facilement enchaîner les opérations asynchrones (avec une écriture quasi linéaire !!!)
7. les promises peuvent être utilisées dans les vues AngularJS comme des données différées, qui seront automatiquement prises en compte dans les bindings lors de la résolution de chaque promise

Le point 4 est pratique : on peut enregistrer une callback n'importe quand, sans se préoccuper de savoir si la promise est déjà résolue, autrement dit si l'opération asynchrone s'est déjà terminée ou est encore en cours d'exécution. Peu importe, on enregistre une paire de callbacks sur la promise, et on est sûr que la callback appropriée sera appelée dès que possible.

```
$promise.then(step1)
    .then(step2)
    .then(step3)
    .then(step4)
    .then(null, function (error) {
        // Handle any error from step1 through step4
    });
```

# VI - Aspects divers et avancés de Angular-Js

## 1. Validation et soumission contrôlée de formulaire

### 1.1. Structure "html + angularJs" d'un formulaire

```
myApp.controller('customerController',function($scope) {
    $scope.reset = function(){
        $scope.customer = {
            firstName : "didier",
            lastName : "Defrance",
            email : "didier-defrance@wanadoo.fr"
        };
    }
    $scope.reset();
});
```

```
<div ng-controller="customerController">
  <form name="customerForm" novalidate>
    <table border="0">
      <tr><td>Enter first name:</td>
        <td><input name="firstname" type="text"
          ng-model="customer.firstName" required>...</td>
      </tr>
      ...
      <tr><td><button ng-click="reset()">Reset</button></td>
        <td><button ... ng-click="submit()">Submit</button></td>
      </tr>
    </table>
  </form>
</div>
```

NB : cette version (à l'ancienne) utilise un tableau html invisible pour l'alignement. On peut préférer une utilisation d'une combinaison de styles css .

### AngularJS application with form

Enter first name:	<input type="text" value="didier"/>	
Enter last name:	<input type="text" value="Defrance"/>	
Email:	<input type="text" value="didier-defrance#wanadoo.fr"/>	Invalid email address.
<input type="button" value="Reset"/>	<input type="button" value="Submit"/>	



## 1.2. Indicateurs pour contrôle/validation

Les indicateurs suivants (automatiquement mis à jour par AngularJS) peuvent aider à pister et caractériser une erreur:

- ☐ **\$dirty** – indique que la valeur a été changée / modifiée.
- ☐ **\$invalid** - indique que la valeur saisie est invalide .
- ☐ **\$error**- précise les détails de l'erreur.

La syntaxe complète est "**nomFormulaire.nomInput.\$error**" ou ...

On peut ainsi effectuer précisément des tests pour savoir si une erreur est présente.

Exemple 1:

```
<span ng-show="customerForm.lastname.$error.required">
Last Name is required.</span>
```

Exemple 2 :

```
<span style="color:red"
  ng-show="customerForm.email.$dirty && customerForm.email.$invalid">
  <span ng-show="customerForm.email.$error.required">Email is required.</span>
  <span ng-show="customerForm.email.$error.email">Invalid email address.</span>
</span>
```

## 1.3. Soumission contrôlée d'un formulaire

```
<form name="customerForm" novalidate>
...
...
<button ng-disabled=
  "customerForm.firstname.$dirty && customerForm.firstname.$invalid
  || customerForm.lastname.$dirty && customerForm.lastname.$invalid
  || customerForm.email.$dirty && customerForm.email.$invalid"
  ng-click="submit()">Submit</button>
</form>
```

## 2. Tests unitaires (avec jasmine et angular-mocks)

La documentation de référence d' AngularJs fait souvent référence à des tests unitaires (avec la fonction "describe()" qui sont prévus pour fonctionner sous le contrôle de la technologie "jasmine".

Mode opératoire :

- 1) télécharger (<https://github.com/jasmine/jasmine/tree/master/dist>) le zip de la technologie jasmine et extraire dans un répertoire choisi le contenu de l'archive .
- 2) Assembler les éléments de "jasmine" avec ceux de "angular-js" au sein d'une arborescence mixte telle que la suivante :

```
main.html
JasmineUnitTestSpecRunner.html
subView1.html
subView2.html
lib/
  angular/
    angular.js
    angular-mocks.js ← indispensable !!!
  jasmine-2.1.3/
    jasmine.js
    jasmine.css
    jasmine-html.js
    boot.js
    console.js
    ...
test/
  unit/
    MyControlerSpec.js
    MyControler2Spec.js
data/xxx.json
  yyy.json
js/mainCtrls.js
  Ctrls1.js
```

Editer ensuite le fichier *SpecRunner.html* ici renommé *JasmineUnitTestSpecRunner.html* en ajustant tous les chemins relatifs nécessaires (librairies , code source , spécifications des tests)

Exemple :

*JasmineUnitTestSpecRunner.html*

```
<!DOCTYPE html>
<html>
<head>
  <meta charset="utf-8">
  <title>Jasmine Spec Runner v2.1.3</title>

  <link rel="shortcut icon" type="image/png" href="lib/jasmine-2.1.3/jasmine_favicon.png">
  <link rel="stylesheet" href="lib/jasmine-2.1.3/jasmine.css">
  ...
```

```

<script src="lib/jasmine-2.1.3/jasmine.js"></script>
<script src="lib/jasmine-2.1.3/jasmine-html.js"></script>
<script src="lib/jasmine-2.1.3/boot.js"></script>

<script src="lib/angular/angular.js"></script>
<script src="lib/angular/angular-mocks.js"></script>

<!-- include source files here... -->
<script src="js/mainCtrls_v1.js"></script>

<!-- include spec files here... -->
<script src="test/unit/MyControlerSpec.js"></script>

</head>
<body> </body>
</html>

```

Ecrire ensuite le contenu d'une spécification de test unitaire en s'inspirant de l'exemple suivant et en étudiant la documentation de jasmine et de angularJs.

#### *test/unit/MyControlerSpec.js*

```

describe('ProductListCtrl', function() {

  beforeEach(module('myAjsApp'));

  it('should create "products" model with 3 products', inject(function($controller) {
    var scope = {},
        ctrl = $controller('ProductListCtrl', {$scope:scope});

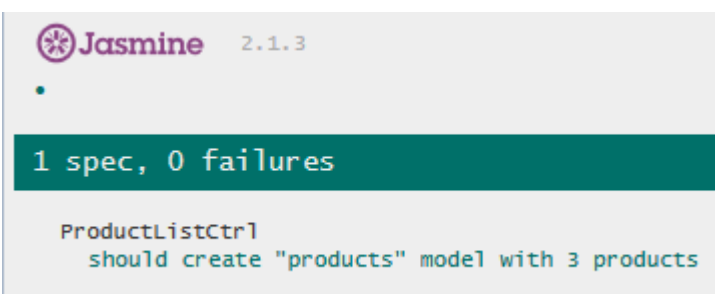
    expect(scope.products.length).toBe(3);
  }));

});

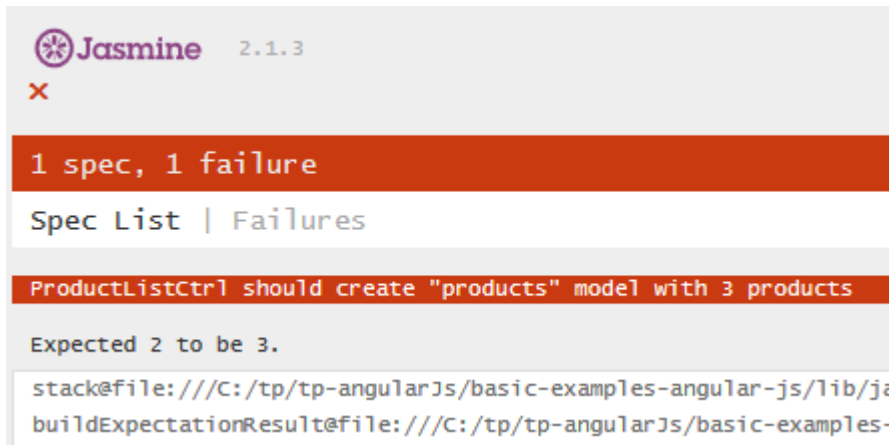
```

Lancer l'affichage de *JasmineUnitTestSpecRunner.html* dans un onglet d'un navigateur web et effectuer régulièrement des "refresh" pour ré-évaluer les tests unitaires.

Exemple de résultat en cas de succès :



Exemple de résultat en cas d'échec :





## 3. Programmation de directives personnalisées

### 3.1. Syntaxe d'utilisation des directives

<p>Enter your Name: <input type="text" ng-model="name"></p>

<p>(ng-bind / classic) <span ng-bind="name"></span></p>

<p>(x-ng-bind / no html5 validation error) <span x-ng-bind="name"></span></p>

<p>(data-ng-bind / no html5 validation error) <span data-ng-bind="name"></span></p>

**ng-bind**, **x-ng-bind** et **data-ng-bind** correspondent à 3 syntaxes alternatives (équivalentes) pour utiliser la directive ng-bind sous forme d'attribut accroché à une balise html.

Soit "**my-basic-directive**" une nouvelle directive personnalisée, celle ci pourra (sauf restriction explicite) être utilisée sous l'une des formes suivantes :

<div my-basic-directive="..."></div>      <!-- en tant qu' **Attribut** -->  
 <my-basic-directive .... />      <!-- en tant qu' **Elément** (balise) XML/HTML →  
 <div class="my-basic-directive: expression;"></div>      <!-- en tant que **Classe** css →  
 <!-- directive : my-directive expression --> <!-- en tant que **coMmentaire** -->

### 3.2. Directive triviale (pour la syntaxe)

Dans <script></script> ou dans sous fichier .js :

```
var myAjsApp = angular.module('myApp', []);
myAjsApp.directive('myBasicDirective', function() {
    return {
        restrict: 'AE', /* A: as Attribute, C: as css Class, E: as Element, M: as coMment */
        template: 'myDirective default text'
    };
});
```

Dans ".html" :

```
<html ng-app="myApp">...
<body>
<div my-basic-directive></div> <!-- as Attribute (A) -->
<div x-my-basic-directive></div>
<div data-my-basic-directive></div>
<p/>
<my-basic-directive /> <!-- as Element (E)-->
</body>
```

→ résultat : affiche 4 fois myDirective default text

Point clef:

- 'myBasicDirective' est orthographié coté javascript en "*Camel case*"

- (minuscule + majuscule en "Bosse de chameau")
- Côté utilisation ".html" `my-basic-directive` est orthographié avec en "snake case" (- - de serpent)

### 3.3. Directives élémentaires (sans isolation de scope)

NB: cet exemple n'a d'intérêt que pour être critiqué et comparé avec la meilleur variante qui suivra.

```
<html ng-app="myApp">
<head>
<script src="lib/angular/angular.js"></script>
<script>
    var myAjsApp = angular.module('myApp', []);
    myAjsApp.controller('p1Controller', ['$scope', function($scope) {
        $scope.person = { name:"nom1", address : "adr1"};
    }]);
    myAjsApp.controller('p2Controller', ['$scope', function($scope) {
        $scope.person = { name:"nom2", address : "adr2"};
    }]);
    myAjsApp.directive('myBadCustomerDirective', function() {
    return {
        restrict: 'E', /* A: as Attribute , C:as css Class , E:as Element */
        /* if no scope : no isolated scope for directive and same scope as parent */
        /* scope: {
            }, */
        template: '<p>Name: {{person.name}} Address: {{person.address}}</p>'
    };
    });
</script>
</head>
<body>
    ***<br/>
    <div ng-controller="p1Controller">
        <my-bad-customer-directive /> <!-- pas de paramétrage explicite -->
    </div>
    **<br/>
    <div ng-controller="p2Controller">
        <my-bad-customer-directive /> <!-- dépendance vis à vis du contexte / scope parent -->
    </div>
    *
</body>
</html>
```

Résultats :

\*\*\*

Name: **nom1** Address: **adr1**

\*\*

Name: **nom2** Address: **adr2**

\*

---> besoin de définir plusieurs contrôleurs avec variantes: ce n'est pas la bonne approche !!!

### 3.4. Directives avec isolation de scope

```
<html ng-app="myApp">
  <head>
    <script src="lib/angular/angular.js"></script>
    <script>
      var myAjsApp = angular.module('myApp', []);
      myAjsApp.controller('scopeExampleController', ['$scope', function($scope) {
        $scope.person1 = { name:"nom1", address : "adr1"};
        $scope.person2 = { name:"nom2", address : "adr2"};
      }]);
      myAjsApp.directive('myCustomer', function() {
        return {
          restrict: 'E', /* A: as Attribute , C:as css Class , E:as Element */
          scope: {
            customerInfo: '=info' /* in this "scope of directive" customerInfo
                                   have the value of attr 'info' of directive call */
          },
          template: ' <p>{{customerInfo.name}} - {{customerInfo.address}}</p>'
        };
      });
    </script>
  </head>
  <body ng-controller="scopeExampleController">
    <my-customer info="person1" />      <!-- passage explicite d'information -->
    <my-customer info="person2"></my-customer>
  </body>
</html>
```

Résultats :

nom1 - adr1

nom2 - adr2

Variante (au comportement équivalent) :

```
....
scope: {
  customerInfo: '=' /* same name in directive scope and in calling / parent scope
                     just a difference of case : "camel" vs "snake" */
}, ...
```

et

```
<my-customer customer-info="person1"></my-customer> <!-- with snake case -->
<my-customer customer-info="person2"></my-customer>
```

### 3.5. Directive manipulant l'arbre DOM

```

<html ng-app="myApp">
<head><script src="lib/angular/angular.js"></script>
<script>
var myAjsApp = angular.module('myApp', []);
myAjsApp.controller('Controller', ['$scope', function($scope) {
    $scope.format = 'M/d/yy h:mm:ss a';
}]);

/* like $timeout , injected $interval service can fire code every n ms ,
   predefined 'dateFilter' is injected as well */
myAjsApp.directive('myCurrentTime', ['$interval', 'dateFilter',
    function($interval, dateFilter) {
        /* function link(scope,element,attr) is called by angular for DOM linking where
           scope is angular scope, element is DOM (jqLite) element and attrs is a map of attributes */
        function link(scope, element, attrs) {
            var format, timeoutId; //local var declarations

            /* utility sub function wich update DOM element with current time */
            function updateTime() {
                element.text(dateFilter(new Date(), format));
            }

            /* surveiller si l'attribut "myCurrentTime / my-current-time" change
               et appeler updateTime() dans ce cas */
            scope.$watch(attrs.myCurrentTime, function(value) {
                format = value;
                updateTime();
            });

            /* for prevent memory leak */
            element.on('$destroy', function() {
                $interval.cancel(timeoutId);
            });

            // start the UI update process; save the timeoutId for canceling
            timeoutId = $interval( function() {
                updateTime(); // update DOM
            }, 1000 /* every 1000ms */);
        } //end of link function

        //return association to the "link" function to be used by angular
        return {
            restrict: 'A',
            link: link
        };
    }]); //end of directive customisation
</script>
</head>

```

```

<body>
  <div ng-controller="Controller">
    Date format: <input ng-model="format"> <hr/>
    Current time is: <span my-current-time="format"></span>
  </div>
</body>
</html>

```

Résultat :

Date format:

---

Current time is: 3/31/15 8:59:16 PM

avec actualisation automatique de la valeur chaque seconde.

## 4. Animations graphiques avec ngAnimate

Le module annexe (facultatif) **ngAnimate** d'angularJs permet de **contrôler automatiquement des effets de style css (animations sur transitions)** .

Attention:

- La compréhension de ces transitions s'appuie sur une assez grande maîtrise supposée des syntaxes css et css3.
- La documentation du module angular-animate (ngAnimate) précise les conventions de noms sur les classes de style css attendues.

Exemple simple (la compréhension passe par un essai et une analyse des styles css) :

**exemple\_animation.html**

```

<html ng-app="app">
<head>
  <meta charset="utf-8">
  <title>My Simple Top Animation</title>
  <script>document.write('<base href="' + document.location + '" />');</script>
  <link rel="stylesheet" href="css/styles.css">
  <script src="lib/angular/angular.js"></script>
  <script src="lib/angular/angular-animate.js"></script>
</head>

```

```

<script>
var app = angular.module('app', ['ngAnimate']);

app.controller('Ctrl', function($scope) {
  $scope.names = ['Alain Therieur', 'Alex Therieur', 'Olie Condor', 'Naomi Black', 'John Scott'];
});
</script>

<body ng-controller="Ctrl">
  <div style="margin-top: 30px; width: 200px; overflow: hidden;">
    <form > <div > <input type="text" ng-model="search" style="width: 80px">
      <button type="submit" >Search</button> </div>
    <ul > <li class="my-anim" ng-repeat="name in names | filter:search">
      <a href="#"> {{name}} </a>
    </li> </ul>
  </form>
</div> </body> </html>

```

En associant la **classe css "my-anim"** à la balise `<li ...>` contrôlée par **ng-repeat="name in names | filter:search"** de AngularJs , on obtient automatiquement une animation dès que l'on modifie le critère de filtrage "search" et dès que les mécanismes d'angular-Js rafraichissent cette zone de la page.

### css/styles.css

```

.my-anim.ng-enter,
.my-anim.ng-leave
{
  -webkit-transition: 500ms cubic-bezier(0.250, 0.250, 0.750, 0.750) all;
  -moz-transition: 500ms cubic-bezier(0.250, 0.250, 0.750, 0.750) all;
  -ms-transition: 500ms cubic-bezier(0.250, 0.250, 0.750, 0.750) all;
  -o-transition: 500ms cubic-bezier(0.250, 0.250, 0.750, 0.750) all;
  transition: 500ms cubic-bezier(0.250, 0.250, 0.750, 0.750) all;
  position: relative;
  display: block;
  overflow: hidden;
  text-overflow: clip;
  white-space: nowrap;
}

```

```
}  
  
.my-anim.ng-leave.my-anim.ng-leave-active,  
.my-anim.ng-enter {  
  opacity: 0;  
  width: 0px;  
  height: 0px;  
}  
  
.my-anim.ng-enter.ng-enter-active,  
.my-anim.ng-leave {  
  opacity: 1;  
  width: 150px;  
  height: 30px;  
}
```

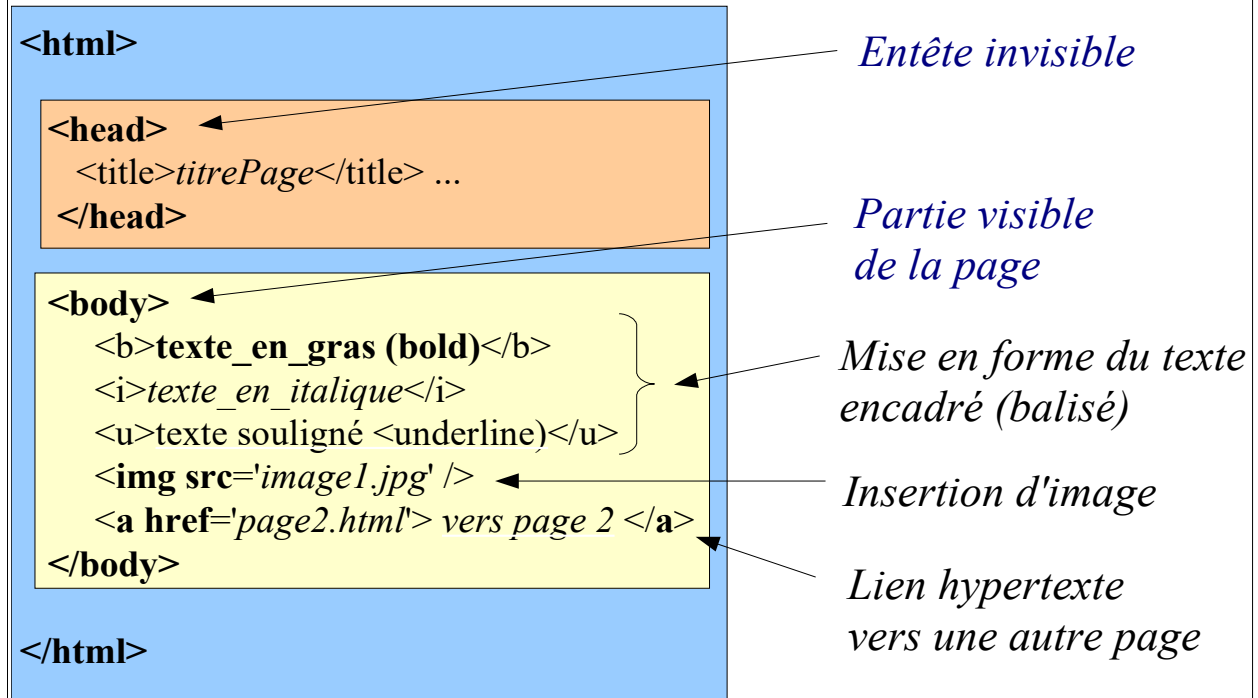
# ANNEXES

# VII - Annexe – Rappels HTML et CSS

## 1. Pages HTML

### HTML (HyperText Markup Language)

*Page html interprétée (pour affichage)  
dans un navigateur internet*





**HTML** (formulaires et tableaux)*Page html dans navigateur*`<html> <head>...</head>``<body> ...``<form action="s1" method="GET" ou "POST" >`couleur : Nom: `</form>``<table border='2' >``<tr><th>années</th><th>valeurs</th></tr>``<tr><td>2008</td><td>100</td></tr>``<tr><td>2009</td><td>120</td></tr>``</table>``</body>``</html>`**Formulaire de saisie****Liste déroulante**`<select name="couleur">
 <option>rouge</option>
 <option>bleu</option>
</select>`**Zone de saisie**`<input type="text" .../>`**Bouton poussoir**`<input type="submit" .../>`**tableau**

années	valeurs
2008	100
2009	120

## 2. Feuilles de styles (CSS, ...)

### 2.1. Intérêts des feuilles de styles

- Clairement **découpler** (séparer) le **contenu** de la **mise en forme** .
- Permettre différentes représentations d'un même contenu (via l'application de différents styles)
- Basculer très rapidement d'un look à l'autre (en changeant les attributs d'une feuille globale).
- Maintenance du site simplifiée , évolutivité garantie .

### 2.2. CSS ( **C**ascading **S**tyle **S**heet )

Ces **feuilles de styles** peuvent être organisées **en cascade** car on peut établir tout un tas d'inclusions entre différentes expressions complémentaires des styles :

**Feuille de styles (CSS=Cascading Style Sheet)**

Par défaut, tous  
les **<h3>** en rouge  
et souligné

*style\_xy.css*

```
p { color:black;font-size:12; }
.c1 { color:blue; }
```

*style1.css*

```
@import style_xy.css
h3 { color:red;text-decoration:underline; }
.c2 { font-style:italic; ....}
```

*style2.css*

```
@import style_xy.css
.....
....
```

*page1.html*

```
<html><head>
  <link rel="stylesheet" href="style1.css" />
</head><body>
  <h3>titre_xy</h3>
  <p class='c2'> texte </p>
</body> </html>
```

*page2.html*

*page3.html*

*pageN.html*

Les **styles CSS** permettent d'appliquer automatiquement (via des règles) certains **attributs de mise en forme** (couleurs, polices, dispositions, ...) à des éléments du document source.

La **sélection d'une règle** repose principalement sur les **noms des balises** ou sur les noms des **classes de styles** (attribut *class* d'une balise).

# VIII - Apports de HTML5

## 1. Versions officielles et interprétations effectives

Principales versions officielles de HTML (W3C) :

- **HTML 4** (1997 , 1999)
- **XHTML 1.0** (1999, 2000) (HTML vu comme un cas particulier du rigoureux XML)
- **HTML 5** (2014 , ...)

Principaux navigateurs actuels :

- **IE 8,9,10,11,...** (*historique* : traitement particulier des événements "javascript" )
- **Firefox** (moteur "gecko/moz" : mozilla)
- **Google Chrome** (moteur "webkit" )
- **Opera , Safari,**
- **Android , ...**

Attention, attention : tous les navigateurs n'interprètent pas bien (pour l'instant) les nouvelles balises de HTML5 . D'autre part les interprétations variables des styles CSS et de javascript font que dans les faits, le rendu exact est au cas par cas (mais ça progresse dans le bon sens) .

En règle générale , l'essentiel de HTML5 fonctionne à partir de IE10 (pas IE9) .

## 2. Grandes lignes de l'évolution vers HTML5

- Utilisation grandement conseillée des **styles CSS** pour la mise en forme.
- Utilisation de **balises "sémantiques"** qui structurent logiquement le document (entête , section/article/paragraphe , pied de page) sans imposer un look précis .
- Plus de **précision sur les champs des formulaires** (number ,date , ...)
- API "**canvas**" pour des "images vectorielles" gérées dynamiquement par des instructions "javascript"
- Meilleure prise en charge des contenus "**multimédia**" (vidéo , audio , ...)
- Meilleure gestion du **graphisme** (rotations , coins arrondis , dégradés , ...)

## 3. Mise en page html5 (sémantique + styles css)

**HTML5** apporte de nouvelles balises sémantiques permettant de mieux structurer le contenu d'une page. Ces balises ne sont par défaut associées à aucune mise en page très précise. Il faut absolument associer des styles CSS particuliers pour contrôler les dispositions et le look.

<i>balises</i>	<i>significations</i>
<b>section</b>	Sous partie d'une page ou article ou section (pouvant

	éventuellement comporter "header" ... , "footer" )
<b>article</b>	(Sous-contenu) assez indépendant du reste de la page (ex : blog-post) (pouvant éventuellement comporter "header" ... , "footer" )
<b>aside</b>	Panneau latéral ou équivalent
<b>h1 , h2 , h3 , ... , h6</b>	Titre d'une (sous-)section quelconque
<b>footer</b>	Pied de page ou de section/article (potentiellement multiple(s))
<b>header</b>	entête de page ou de section/article (potentiellement multiple(s))
<b>nav</b>	barre de navigation avec liens (menu simple ou mode plan ou ...)
address	Adresse (contact) liée à l'auteur de la page ou de l'article englobant

<section> et <article> sont très proches (à choisir en fonction du contexte)

sémantique pure (associée à aucune mise en page) :

<i>balises</i>	<i>significations</i>
<main> ou en attendant <div id="main">	Unique partie principale d'une page (dans <body> , éventuellement à coté de <header> et <footer> ou englobant <header> et <footer> mais jamais à l'intérieur de <footer>/<header>/<section>/<article>)

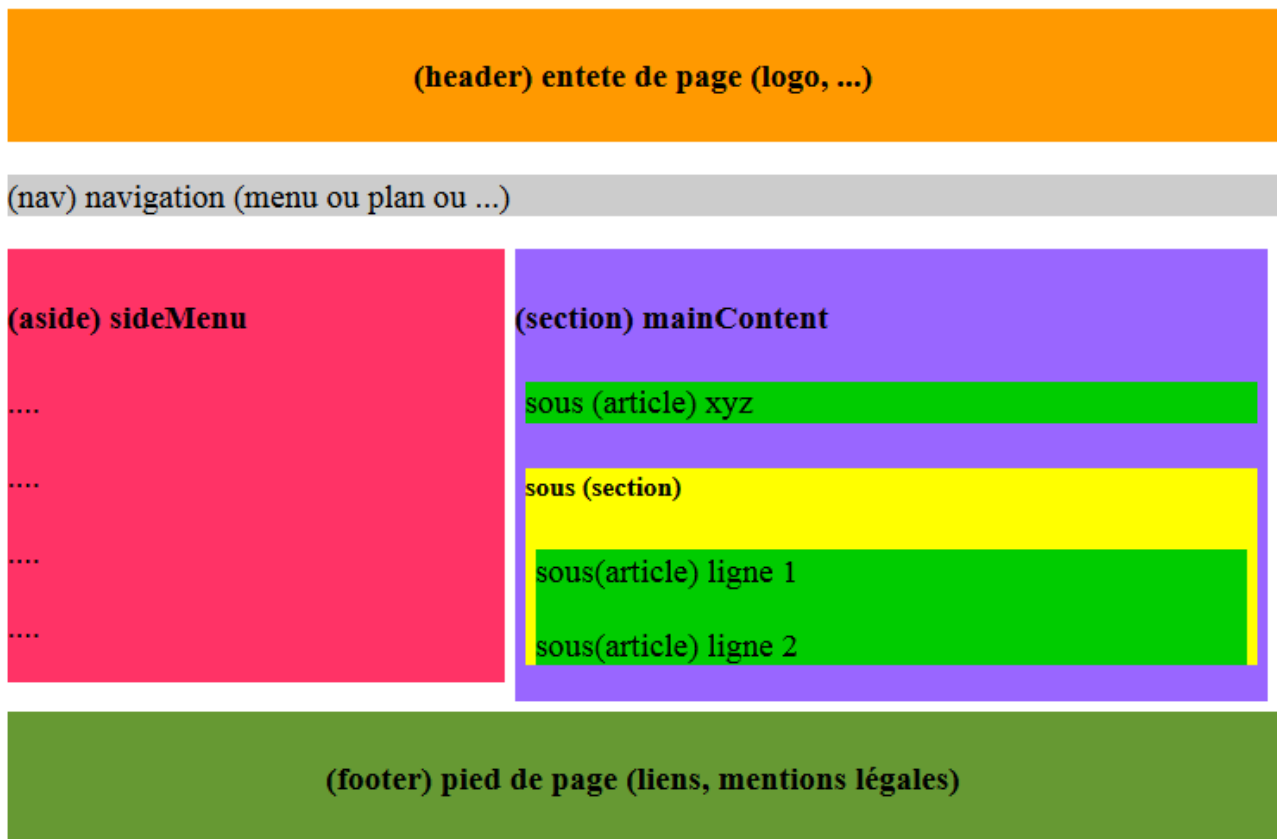
**NB** : bien que des styles CSS (2 ou 3) soient conseillés pour paramétrer la mise en page d'un site web , on peut éventuellement utiliser (à l'ancienne) des tableaux pour contrôler l'alignement des éléments mais ceci doit absolument se faire en précisant la sémantique **role="presentation"** au niveau de la balise <table> pour qu'il n'y ait pas confusion avec un tableau de données .

Exemple (parmi pleins d'autres possibles) :

```
<html>
<head>
  <link rel="stylesheet" type="text/css" href="layout.css"/>
  <title>essai layout html5</title>
</head>
<body>
  <header id="mainHeader" role="banner">
    <h4> (header) entete de page (logo, ...)</h4>
  </header>
  <nav id="mainNav">
    <p>(nav) navigation (menu ou plan ou ...)</p>
  </nav>
  <main>
```

```
<aside id="mainSideMenu">
  <h4> (aside) sideMenu </h4> <p> .... </p> <p> .... </p> <p> .... </p> <p> .... </p>
</aside>
<section id="mainContent">
  <h4> (section) mainContent</h4>
  <article>
    <p>sous (article) xyz</p>
  </article>
  <section>
    <h5>sous (section)</h5>
    <article>
      <p>sous(article) ligne 1</p>  <p>sous(article) ligne 2</p>
    </article>
  </section>
</section>
</main>
<footer id="mainFooter" >
  <h4>(footer) pied de page (liens, mentions légales)</h4>
</footer>
</body>
</html>
```

ce qui donne :



avec les styles css du fichier suivant :

*layout.css*

```
#mainHeader, #mainSideMenu, #mainContent, #mainFooter { padding:1px 0;}
#mainHeader { background-color:#FF9900; text-align:center; }
#mainNav { background-color : #cccccc; }
#mainSideMenu { float:left; width:240px; background-color:#FF3366; overflow:auto; }
/* margin-left de #mainContent doit etre un peu plus grand que width de #mainSideMenu flottant */
#mainContent { margin-left:245px; background-color:#9966FF; }
/* clear:both; essentiel pour que #mainFooter soit toujours en dessous du #mainSideMenu flottant
et jamais sur la meme ligne */
#mainFooter { background-color:#669933; text-align:center; clear:both; }
article { margin:5px; background-color : #00cc00; }
section { margin:5px; background-color : yellow; }
```

## 4. apports et précisions au niveau des formulaires

### 4.1. Nouveaux types de "input"

Select your favorite color: <input type="color" name="favcolor">

Select your favorite color:

Birthday: `<input type="date" name="bday">` , existera aussi `type="datetime"` .

Birthday:

`<input type="email" name="email">` avec message d'erreur automatique s'il manque "@"

Birthday (month and year): `<input type="month" name="bdaymonth">`

Birthday (month and year):

Quantity (between 1 and 5): `<input type="number" name="quantity" min="1" max="5">`

Quantity (between 1 and 5):  , plus message d'erreur automatique si non numérique ou hors intervalle , min et max sont facultatifs

Points: 0 `<input type="range" name="points" min="1" max="10">` 10

Points: 0  10

`<input type="search" name="googlesearch">` qui se comporte comme `type=text` .

Autres types :

- `type="tel"` pour numéro de téléphone

## 4.2. Datalist

Zone combinée (liste + saisie)

`<input list="browsers" name="browser">`

`<datalist id="browsers">`

`<option value="Internet Explorer">`

`<option value="Firefox">`

`<option value="Chrome">`

```

<option value="Opera">
<option value="Safari">
</datalist>

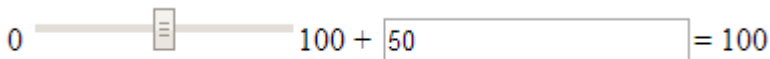
```

### 4.3. output

```

<form oninput="x.value=parseInt(a.value)+parseInt(b.value)">
0<input type="range" id="a" value="50">100 +
<input type="number" id="b" value="50">= <output name="x" for="a b" />
</form>

```



### 4.4. keygen

<keygen name="security"> génère une clef privée (utilisée coté client) et une clef publique envoyée au serveur .

2048 (haute sécurité) ▼

La clef publique pourra par exemple être utilisée pour générer un certificat client pour authentifier l'utilisateur dans le futur.

## 5. API "Canvas" pour les images vectorielles

## 6. Graphisme évolué et multimédia



## IX - Annexe – Web Services REST (coté serveur)

### 1. Généralités sur Web-Services REST

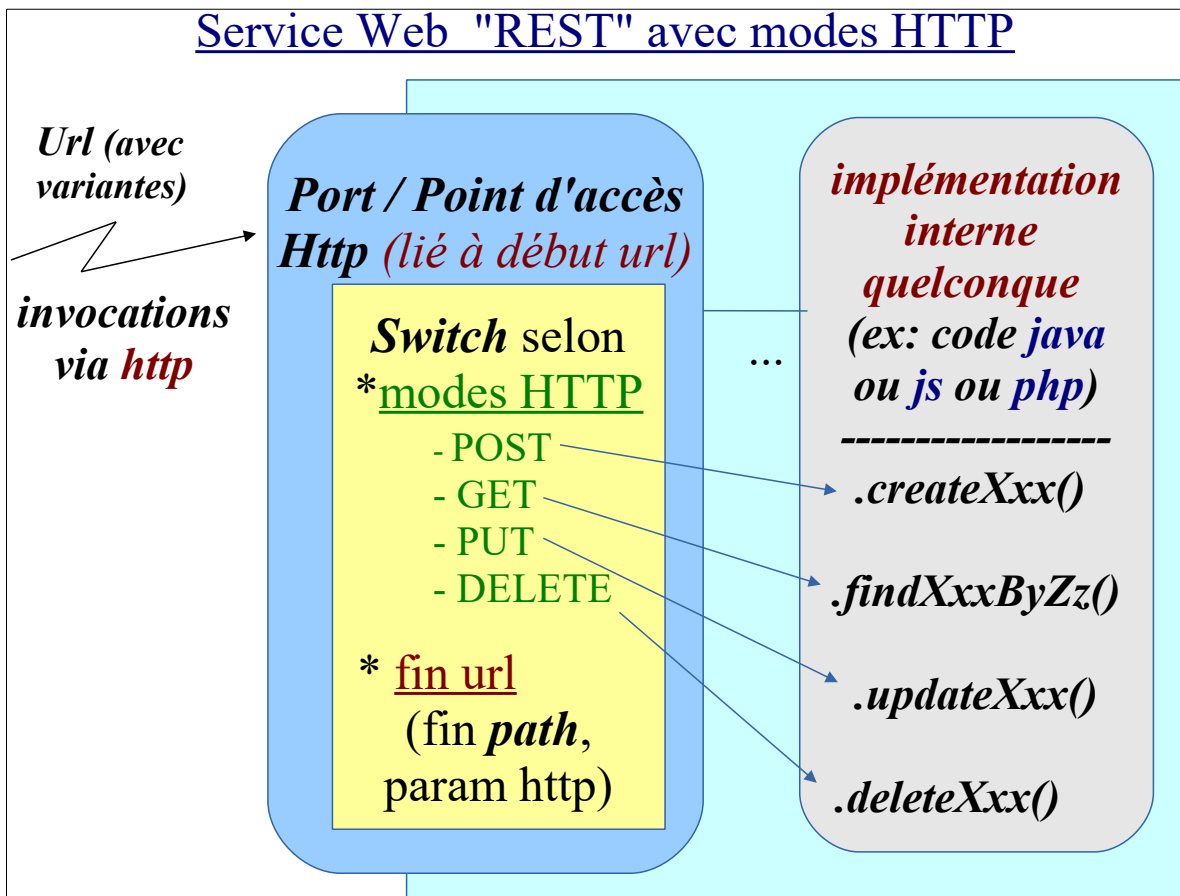
2 grands types de services WEB: SOAP/XML et REST/HTTP

#### ***WS-\* (SOAP / XML)***

- "Payload" systématiquement en **XML** (*sauf pièces attachées / HTTP*)
- **Enveloppe SOAP** en XML (*header facultatif pour extensions*)
- **Protocole de transport au choix (HTTP, JMS, ...)**
- Sémantique quelconque (*appels méthodes*) , **description WSDL**
- **Plutôt** orienté Middleware SOA (*arrière plan*)

#### ***REST (HTTP)***

- "Payload" au choix (XML , HTML , **JSON**, ...)
- Pas d'enveloppe imposée
- **Protocole de transport = toujours HTTP.**
- Sémantique "**CRUD**" (*modes http PUT,GET,POST,DELETE*)
- **Plutôt** orienté IHM Web/Web2 (*avant plan*)



## Points clefs des Web services "REST"

Retournant des données dans un format quelconque ("**XML**", "**JSON**" et éventuellement "**txt**" ou "**html**") les web-services "REST" offrent des **résultats qui nécessitent généralement peu de re-traitements** pour être mis en forme au sein d'une IHM web.

Le format "**au cas par cas**" des données retournées par les services REST permet peu d'automatisme(s) sur les niveaux intermédiaires.

Souvent associés au format "**JSON**" les web-services "REST" **conviennent parfaitement** à des appels (ou implémentations) au sein du **langage javascript** .

La **relative simplicité des URLs d'invocation des services "REST"** permet des **appels plus immédiats** (un simple **href="..."** suffit en mode **GET** pour les recherches de données) .

La **compacité/simplicité des messages "JSON"** souvent **associés à "REST"** permet d'obtenir **d'assez bonnes performances** .

## REST = style d'architecture (conventions)

**REST** est l'acronyme de **R**epresentational **S**tate **T**ransfert.

C'est un **style d'architecture** qui a été décrit par *Roy Thomas Fielding* dans sa thèse «*Architectural Styles and the Design of Network-based Software Architectures*».

L'information de base, dans une architecture REST, est appelée **ressource**.  
Toute information (à sémantique stable) qui peut être nommée est une ressource: un article, une photo, une personne, un service ou n'importe quel concept.

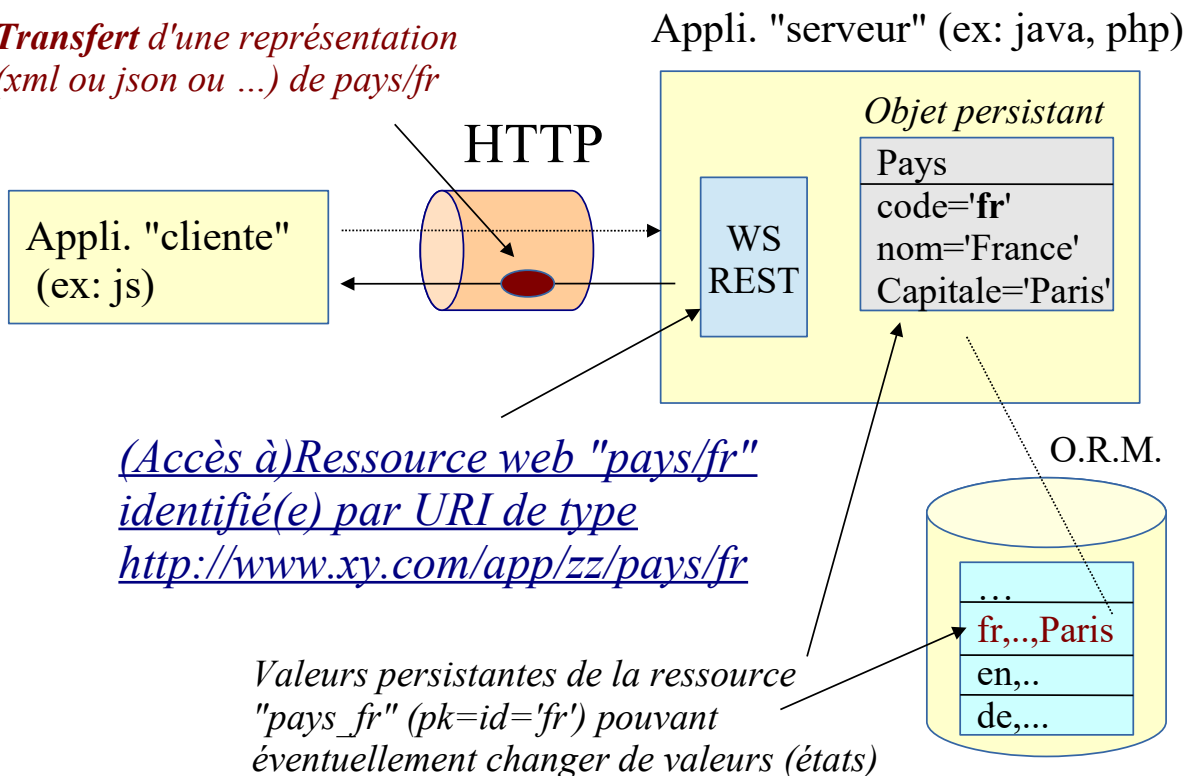
Une ressource est identifiée par un **identificateur de ressource**. Sur le web ces identificateurs sont les **URI** (Uniform Resource Identifier).

**NB:** dans la plupart des cas, une ressource REST correspond indirectement à un enregistrement en base (avec la *clef primaire* comme partie finale de l'uri "identifiant").

Les composants de l'architecture REST manipulent ces ressources en **transférant à travers le réseau** (via HTTP) des **représentations de ces ressources**.  
Sur le web, on trouve aujourd'hui le plus souvent des représentations au format **HTML, XML ou JSON**.

## REST : transferts de représentations de ressources

*Transfert d'une représentation  
(xml ou json ou ...) de pays/fr*



## **REST et principaux formats (xml,json)**

Une invocation d'URL de service REST peut être accompagnée de données (en entrée ou en sortie) pouvant prendre des formats quelconques :

**text/plain , text/html , application/xml , application/json , ...**

Dans le cas d'une lecture/recherche d'informations , le format du résultat retourné pourra (selon les cas) être :

- **imposé (en dur) par le code du service REST .**
- **au choix (xml , json) et précisé par une partie de l'url**
- **au choix (xml , json) et précisé par le champ "Accept :" de l'entête HTTP de la requête. (exemple: Accept: application/json) .**

Dans tous les cas, la réponse HTTP devra avoir son format précisé via le champ habituel ***Content-Type: application/json*** de l'entête.

## Format JSON (JSON = *JavaScript Object Notation*)

Les 2 principales caractéristiques de JSON sont :

- Le principe de clé / valeur (map)
- L'organisation des données sous forme de tableau

```
[
  {
    "nom": "article a",
    "prix": 3.05,
    "disponible": false,
    "descriptif": "article1"
  },
  {
    "nom": "article b",
    "prix": 13.05,
    "disponible": true,
    "descriptif": null
  }
]
```

Les types de données valables sont :

- tableau
- objet
- chaîne de caractères
- valeur numérique (entier, double)
- booléen (true/false)
- null

*une liste d'articles*

*une personne*

```
{
  "nom": "xxxx",
  "prenom": "yyyy",
  "age": 25
}
```

## REST et méthodes HTTP (verbes)

Les méthodes HTTP sont utilisées pour indiquer la sémantique des actions demandées :

- **GET** : **lecture/recherche** d'information
- **POST** : **envoi** d'information
- **PUT** : **mise à jour** d'information
- **DELETE** : **suppression** d'information

Par exemple, pour récupérer la liste des adhérents d'un club, on peut effectuer une requête de type **GET** vers la ressource **<http://monsite.com/adherents>**

Pour obtenir que les adhérents ayant plus de 20 ans, la requête devient **<http://monsite.com/adherents?ageMinimum=20>**

Pour supprimer numéro 4, on peut employer une requête de type **DELETE** telle que **<http://monsite.com/adherents/4>**

Pour envoyer des informations, on utilise **POST** ou **PUT** en passant les informations dans le corps (invisible) du message HTTP avec comme URL celle de la ressource web que l'on veut créer ou mettre à jour.

**Exemple concret de service REST : "Elevation API"**

L'entreprise "**Google**" fournit gratuitement certains services WEB de type REST. "**Elevation API**" est un service REST de Google qui renvoie l'altitude d'un point de la planète selon ses coordonnées (latitude, longitude).

La documentation complète se trouve au bout de l'URL suivante :

<https://developers.google.com/maps/documentation/elevation/?hl=fr>

Sachant que les coordonnées du Mont blanc sont :

Lat/Lon : 45.8325 N / 6.86417 E (GPS : 32T 334120 5077656)

Les invocations suivantes (du service web rest "api/elevation")

<http://maps.googleapis.com/maps/api/elevation/json?locations=45.8325,6.86417>

<http://maps.googleapis.com/maps/api/elevation/xml?locations=45.8325,6.86417>

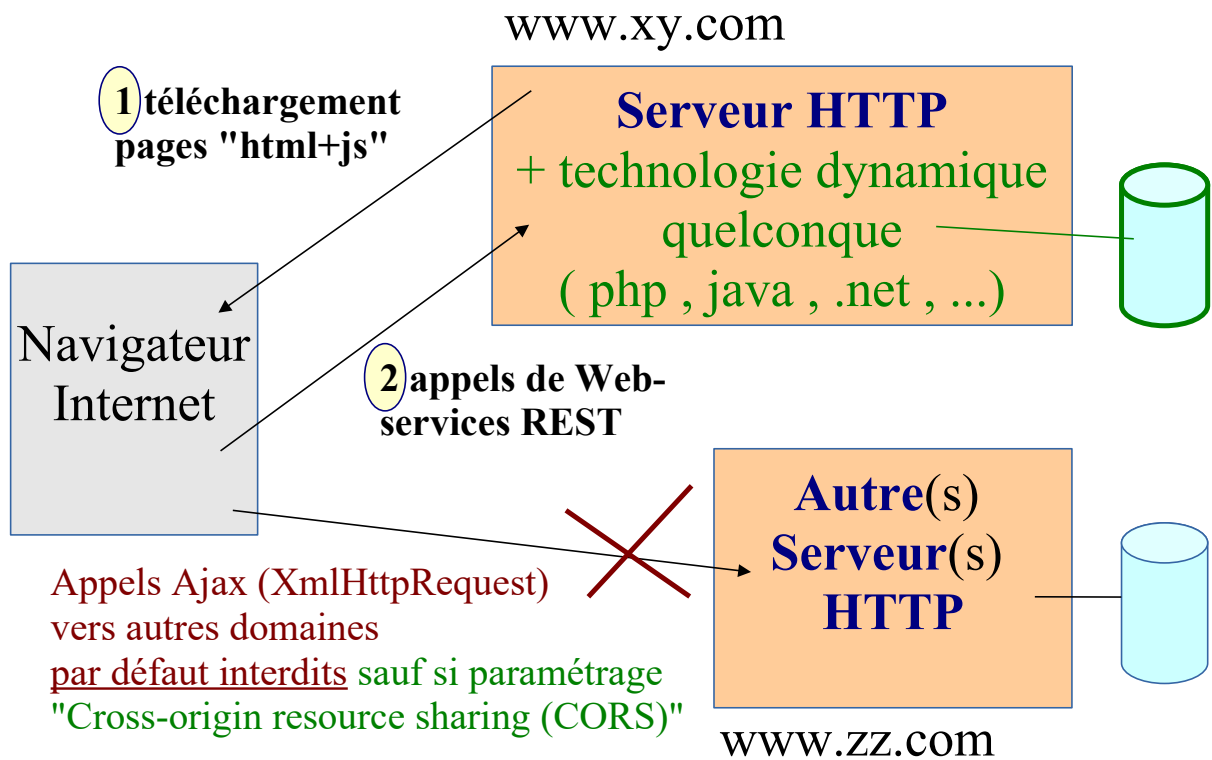
donne les résultats suivants "json" ou "xml":

```
{ "results" : [  
  {  
    "elevation" : 4766.466796875,  
    "location" : {  
      "lat" : 45.8325,  
      "lng" : 6.86417  
    },  
    "resolution" : 152.7032318115234  
  },  
], "status" : "OK"  
}
```

```
?xml version="1.0" encoding="UTF-8"?>  
<ElevationResponse>  
  <status>OK</status>  
  <result>  
    <location>  
      <lat>45.8325000</lat>  
      <lng>6.8641700</lng>  
    </location>  
    <elevation>4766.4667969</elevation>  
    <resolution>152.7032318</resolution>  
  </result>  
</ElevationResponse>
```

## 2. Limitations Ajax sans CORS

### Cadre des appels "html/js/ajax" vers services REST



### 3. CORS (Cross Origin Resource Sharing)

## CORS=Cross Origin Resource Sharing

CORS est une **norme du W3C** qui précise certains **champs** à placer dans une **entête HTTP** qui serviront à échanger entre le navigateur et le serveur des informations qui serviront à décider si une requête sera ou pas acceptée.

(utile si domaines différents) , dans requête simple ou bien dans pré-échange préliminaire quelquefois déclenché en plus :

*Au sein d'une requête "demande autorisation" envoyée du client vers le serveur :*

**Origin:** <http://www.xy.com>

*Dans la "réponse à demande d'autorisation" renvoyée par le serveur :*

**Access-Control-Allow-Origin:** <http://www.xy.com>

*Ou bien*

**Access-Control-Allow-Origin:** \* *(si public)*

*→ requête acceptée*

*Si absence de "Access-Control-Allow-Origin :" ou bien valeur différente  
---> requête refusée*



## CORS=Cross Origin Resource Sharing (2)

NB1: toute requête "CORS" valide doit absolument comporter le champ "**Origin** :" dans l'entête http. Ce champ est toujours construit automatiquement par le navigateur et jamais renseigné par programmation javascript.

*Ceci ne protège que partiellement l'accès à certains serveurs car un "méchant hacker" utilise un "navigateur trafiqué".*

*Les mécanismes "CORS" protège un peu le client ordinaire (utilisant un vrai navigateur) que dans la mesure où la page d'origine n'a pas été interceptée ni trafiquée (l'utilisation conjointe de "https" est primordiale) .*

NB2 : Dans le cas (très classique/fréquent) , où la requête comporte "**Content-Type: application/json**" (ou **application/xml** ou ...) , la norme "CORS" (considérant la requête comme étant "pas si simple") impose un pré-échange préliminaire appelé "**Preflighted request/response**" .

## Paramétrages CORS à effectuer coté serveur

L'application qui coté serveur, fourni quelques Web Services REST , peut (et généralement doit) autoriser les requêtes "Ajax / CORS" issues d'autres domaines ("\*" ou "www.xy.com" ).

Attention: ce n'est pas une "sécurité coté serveur" mais juste **un paramétrage autorisant ou pas à rendre service à d'autres domaines et en devant gérer la charge induite (taille du cluster, consommation électrique, ...)** .

*// Exemple : CORS enabled with express/node-js :*

```
app.use(function(req, res, next) {
  res.header("Access-Control-Allow-Origin", "*"); // "*" ou "xy.com , ..."
  res.header("Access-Control-Allow-Methods",
    "POST, GET, PUT, DELETE, OPTIONS"); //default: GET, ...
  res.header("Access-Control-Allow-Headers",
    "Origin, X-Requested-With, Content-Type, Accept , Authorization");
  next();
});
```

# Paramétrages CORS avec CXF et JAX-RS

```

<bean id="corsFilter" class="org.apache.cxf.rs.security.cors.
CrossOriginResourceSharingFilter">
  <!-- <property name="allowCredentials" value="true"/> -->
</bean>
...
<jaxrs:server id="myRestServices" address="/rest">
  <jaxrs:providers>
    <ref bean='jacksonJsonProvider' />
    <ref bean='corsFilter' />
  </jaxrs:providers>
  <jaxrs:serviceBeans> ...
    <ref bean="serviceClientsRest" />
  </jaxrs:serviceBeans> ...

```

config  
spring/cxf

```

@Path("/json/gestionclients")
@Produces("application/json")
@Consumes("application/json")
@CrossOriginResourceSharing(allowAllOrigins = true)
// ou bien autorisations plus fines
public class ClientRestJsonService {
...}

```

code java

# X - Annexe – Env. Dev. , MongoDB , JsonP

## 1. Environnement de développement AngularJs

### 1.1. Env. (sandbox) en ligne "jsFiddle ou Plunker"

**JsFiddle** et **Plunker** sont des mini environnements de développement en ligne centrés autour de javascript + html + css.

Les principales fonctionnalités de ces IDE en ligne sont :

\* **éditeur (de portion) de code javascript** (colorisation des mots clefs , des parenthèses , des accolades , ...)

\* **tests immédiats (interprétation + rendu)**

\* **partage d'exemples en ligne entre développeurs "javascript"**

( envoi de l'url d'un exemple par mail )

<b>jsFiddle</b>	<a href="https://jsfiddle.net/">https://jsfiddle.net/</a>	Simple , intuitif , vue d'ensemble (HTML,CSS,JS)
<b>plunker</b>	<a href="http://plnkr.co/">http://plnkr.co/</a>	Plus évolué (plusieurs fichiers) Auto complétion au niveau des éditeurs html , js

→ très pratique pour se familiariser avec la syntaxe "JS" , "HTML/CSS" , "Angular-Js" sans avoir à installer de logiciel.

→ très pratique pour exposer/partager des exemples "Js , Angular , ..." .

### 1.2. jsFiddle

Exemple d'utilisation basique de JsFiddle

(avec jQuery comme bibliothèque js sélectionnée) :

The screenshot shows the JsFiddle interface with three panels for code editing and one for the live preview.

- HTML Panel:**

```

1 <b class='c1'>coucou</b>
2 <form id="f1">
3   nom: <input type='text' id="nom"/>
4 </form>
5

```
- CSS Panel:**

```

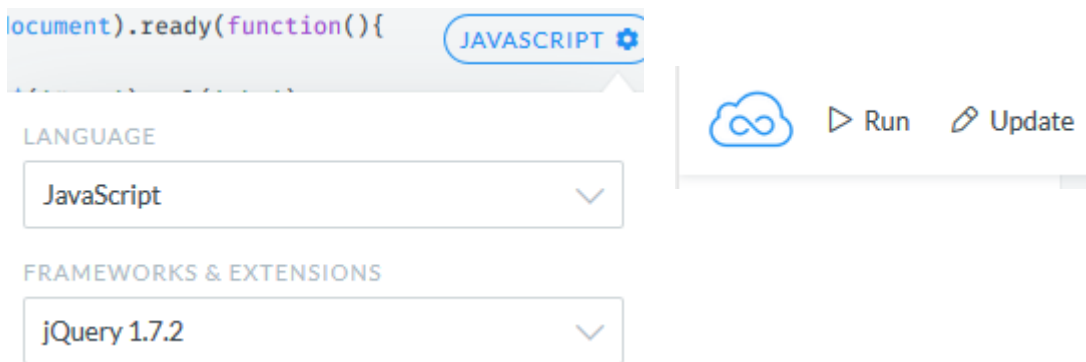
1 .c1 { color: red; }

```
- JAVASCRIPT Panel:**

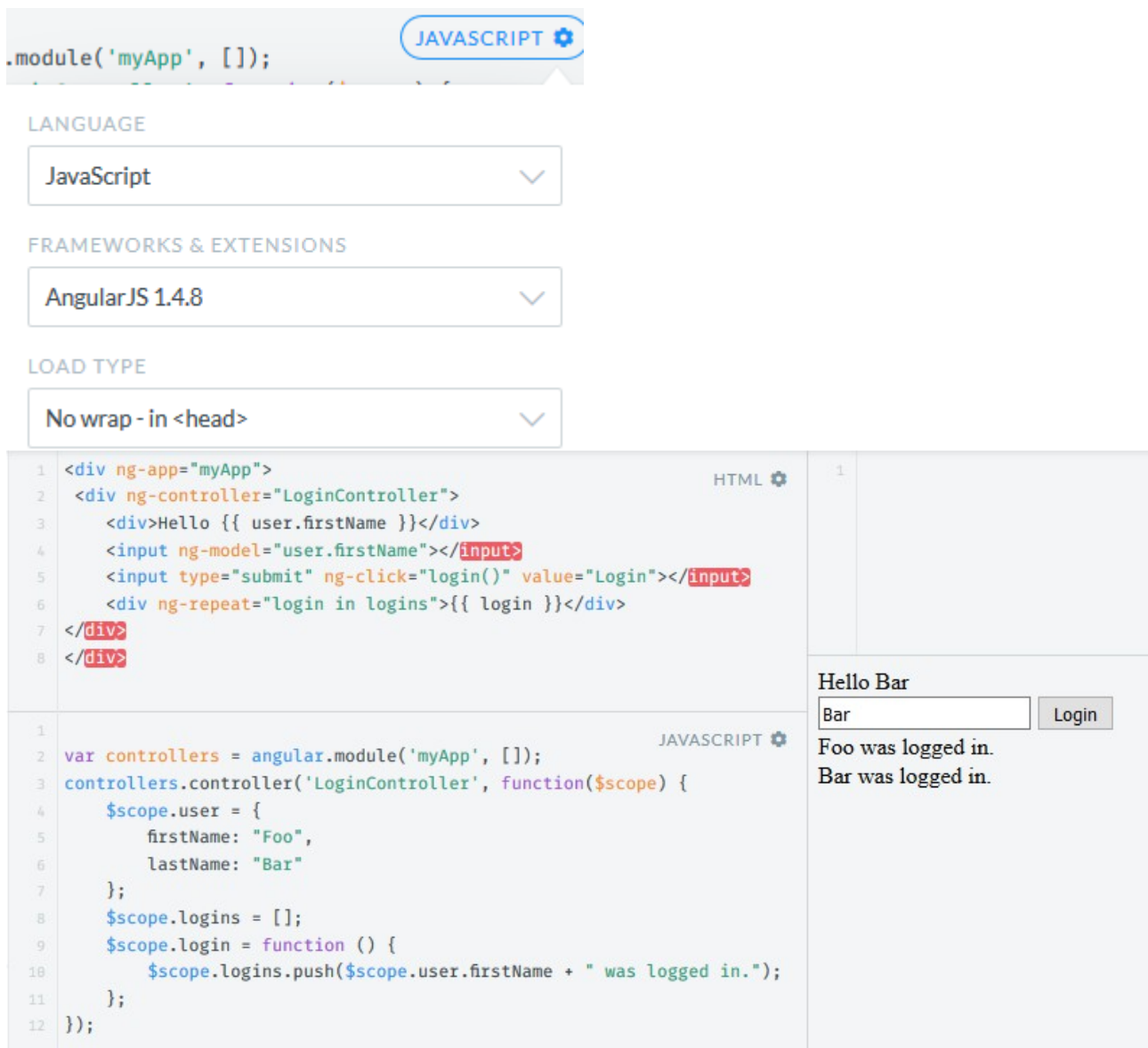
```

1 $(document).ready(function(){
2
3   $('#nom').val('abc');
4
5 });|

```
- Preview Panel:** Shows the rendered output with the text "coucou" in red and a text input field labeled "nom:" containing the value "abc".



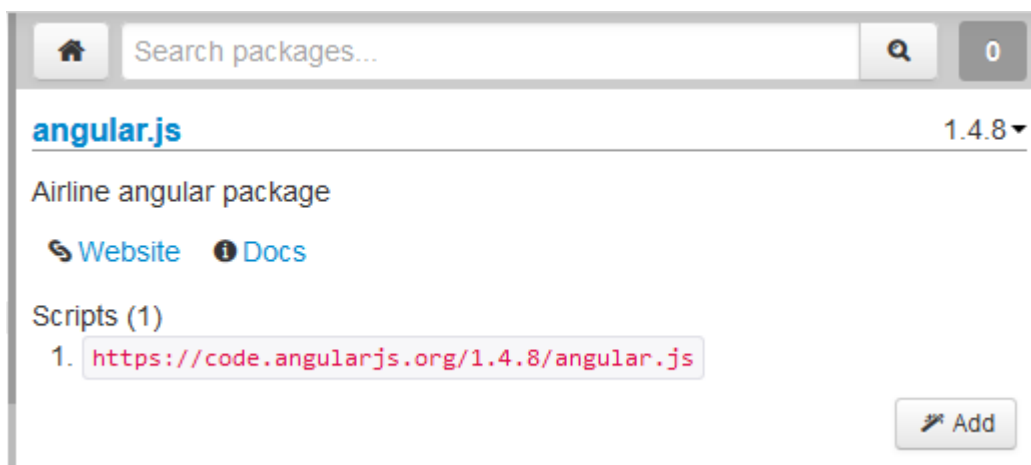
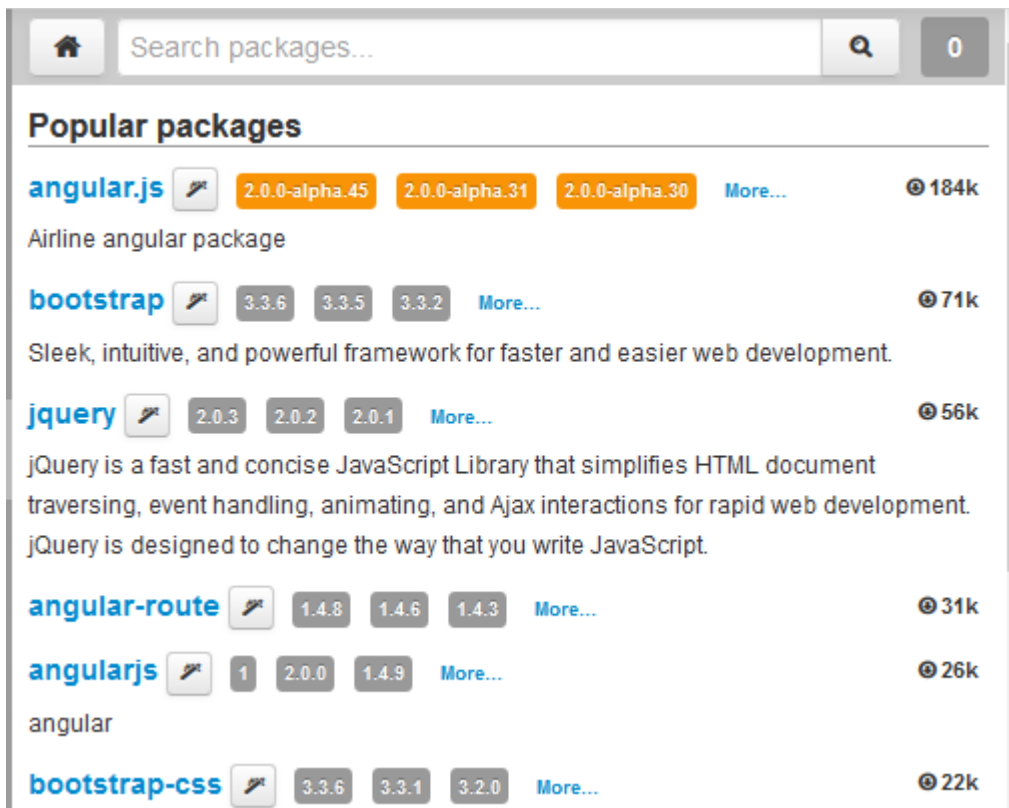
Exemple "jsFiddle" avec Angular-js" :



URL de l'exemple (original / ancienne version): <http://jsfiddle.net/Lt7aP/4/>

## 1.3. Plunker

Nouvel éditeur (nouveau projet) , choix d'une (ou plusieurs "librairies" / "packages" )



Saisie des contenus des fichiers puis "Save" , "Run" , "Stop"

The screenshot shows the Plunker v0.8.18 web interface. On the left, there's a sidebar with a 'FILES' section containing 'index.html', 'README.md', 'script.js', and 'style.css'. Below it is a 'VERSIONS' section with a count of 3. The main area is a code editor with a light blue background. It contains JavaScript code for an Angular.js application. The code defines a module 'myApp' and a controller 'LoginController'. The controller sets a user object with 'firstName: "Foo"' and 'lastName: "Bar"', initializes a 'logins' array, and defines a 'login' function that pushes a message to the array. The preview on the right shows the rendered HTML: 'Hello Plunker!', 'Hello Bar', a text input with 'Bar', a 'Login' button, and two lines of feedback text: 'Foo was logged in.' and 'Bar was logged in.'

```
// Code goes here

var controllers = angular.module('myApp', []);
controllers.controller('LoginController', function($scope) {
  $scope.user = {
    firstName: "Foo",
    lastName: "Bar"
  };
  $scope.logins = [];
  $scope.login = function () {
    $scope.logins.push($scope.user.firstName + " was logged in.");
  };
});
```

Freeze Fork New Stop

```
1 <!DOCTYPE html>
2 <html ng-app="myApp">
3
4 <head>
5   <script data-require="angular.js@1.4.8"
6     data-semver="1.4.8"
7     src="https://code.angularjs.org/1.4.8/angular.js"></script>
8   <link href="style.css" rel="stylesheet" />
9   <script src="script.js"></script>
10 </head>
11
12 <body>
13   <h1>Hello Plunker!</h1>
14
15   <div ng-controller="LoginController">
16     <div>Hello {{ user.firstName }}</div>
17     <input ng-model="user.firstName"/>
18     <input type="submit" ng-click="login()" value="Login" />
19     <div ng-repeat="login in logins">{{ login }}</div>
20   </div>
21
22 </body>
23 </html>
```

**Hello Plunker!**

Hello Bar

Bar Login

Foo was logged in.  
Bar was logged in.

URL de l'exemple (validité temporaire) : <http://plnkr.co/edit/5sDbHWMzAXRYDgJFHRJM>

## 1.4. NPM (d'origine "NodeJs")

**NPM** (fausse ou vraie abréviation de "Node Package Manager" ) est le gestionnaire de projet de la technologie "NodeJs" .

A l'origine prévue pour "nodeJs" et maintenant intégrée à l'installation de la technologie "nodeJs" , **npm** peut être utilisé avec d'autres technologies "javascript" telles que Angular 1 et 2 .

NPM (qui est un peu en javascript ce que maven est en java) permet de travailler sérieusement sur de véritables projets d'entreprise .

Les principales fonctionnalités de npm sont les suivantes :

- téléchargement des bibliothèques/packages/modules nécessaires à partir des dépendances exprimées dans le fichier de configuration "package.json" .
- aide au lancement de certains utilitaires (pre-processeurs , mini serveur-web , nodejs , tests, ...)

## 1.5. Grunt (Javascript Task Runner)

"**Grunt**" s'appuyant lui même sur npm pour le téléchargement/installation , permet d'**automatiser des tâches de développement répétitives** telles que :

- **nettoyage** de certains répertoires et fichiers temporaires (via *grunt-contrib-clean*)
- **analyse/vérification** du code javascript (via *grunt-contrib-jshint*)
- éventuelle **pré-compilations** (ex : scss → css ,  
ts (typeScript/anagular2) → js ,  
...) via *grunt-typescript* , ...
- éventuelle concaténation de fichiers javascript (via *grunt-contrib-concat*)
- éventuelle **compression** (*minification*) du code (via *grunt-contrib-uglify* ou ...)
- éventuels déclenchements automatiques dès qu'un fichier a changé (via *grunt-contrib-watch*)
- **lancement de tests** (via *grunt-karma* ou ....)

la configuration de "**Grunt**" s'effectue via un fichier javascript de configuration (fonction de configuration javascript pilotant des modules "**npm/nodeJs**" et avec données/paramètres de syntaxe proche JSON ) .

## 1.6. Tests autour d'une application "Angular-js"

**Karma** sert à **lancer facilement des tests** (basés sur jasmine ou autre) . Karma peut lancer plusieurs navigateurs (ex : Chrome et FireFox) pour ensuite communiquer avec eux de façons à re-lancer automatiquement certains tests et à actualiser les résultats (statuts , ...).

**Jasmine** sert à **interpréter des tests unitaires écrits en "javascript"** .

Plusieurs types de tests (lançables via karma):

- unitaire : test d'un seul (sous-) composant (ex : contrôleur) avec éventuel(s) "mock"
- **End To End** (intégration) : test de l'ensemble (ihm + dialogue XHR + ...)
- intermédiaire (ex : service avec appels XHR , ...)

## 1.7. Karma

L'installation de "karma" s'effectue à partir de npm :

```
npm install -g karma
```

Remarque importante: il faut spécifier l'option **-g** pour une installation "globale" (non spécifique au projet) avec stockage des librairies téléchargées dans un répertoire

C:\Users\username\AppData\Roaming\npm\node\_modules\ sous windows .

pour une éventuelle installation locale:

```
npm install karma --save-dev
```

puis

```
./node_modules/.bin/karma
```

installation du plugin important **karma-jasmine-html-reporter** via NPM :

```
npm install -g karma-jasmine-html-reporter
```

Future utilisation via `reporters: ['kjhtml']` ou `reporters: ['progress', 'kjhtml']` dans `karma.conf.js` ou bien via `--reporters kjhtml` sur la ligne de commande de lancement de karma

### Configuration de karma:

1) se placer dans le répertoire racine d'un projet existant

avec par exemple `js/*.js` pour le *code source*

et `test/unit/*Spec.js` pour le code des tests unitaires basés par exemple sur jasmine

2) **karma init karma.conf.js**

==> saisir par exemple les valeurs suivantes de manière interactive:

jasmine

no (for Require.js)

Chrome

Firefox

js/\*.js

test/\*\*/\*Spec.js

(nothing to exclude)

yes (watch changes)

==> **génération du fichier de configuration KARMA** "*karma.conf.js*"

Exemple de fichier "**karma.conf.js**" (configuration karma):

```
module.exports = function(config) {
  config.set({

    // base path that will be used to resolve all patterns (eg. files, exclude)
```



```

basePath: ",

// frameworks to use
// available frameworks: https://npmjs.org/browse/keyword/karma-adapter
frameworks: ['jasmine'],

// list of files / patterns to load in the browser
files: [
  'js/*.js',
  'test/**/*.Spec.js'
],

// list of files to exclude
exclude: [
],

// 'kjhtml' correspond au plugin karma-jasmine-html-reporter
reporters: ['progress', 'kjhtml'],

// web server port
port: 9876,

// enable / disable colors in the output (reporters and logs)
colors: true,

// level of logging , possible values: config.XY
// avec XY = LOG_DISABLE || LOG_ERROR || LOG_WARN || LOG_INFO || LOG_DEBUG
logLevel: config.LOG_INFO,

// enable / disable watching file and executing tests whenever any file changes
autoWatch: true,

// start these browsers
// available browser launchers: https://npmjs.org/browse/keyword/karma-launcher
browsers: ['Chrome', 'Firefox'],

// Continuous Integration mode
// if true, Karma captures browsers, runs the tests and exits
singleRun: false,

// Concurrency level
// how many browser should be started simultaneous
concurrency: Infinity
})
}

```

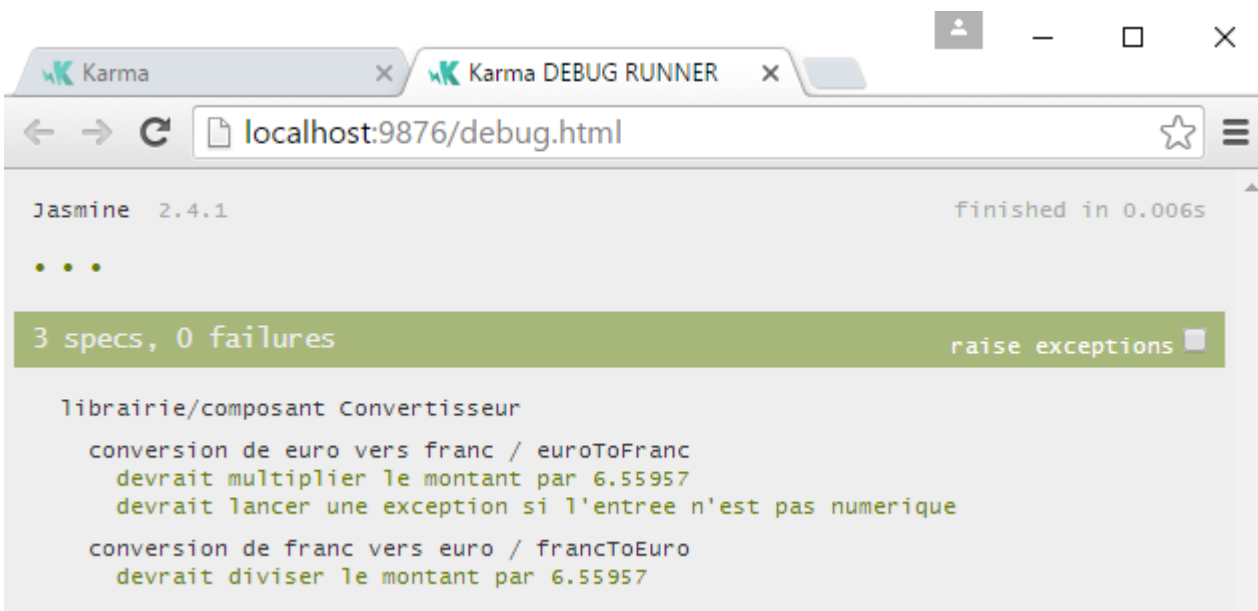
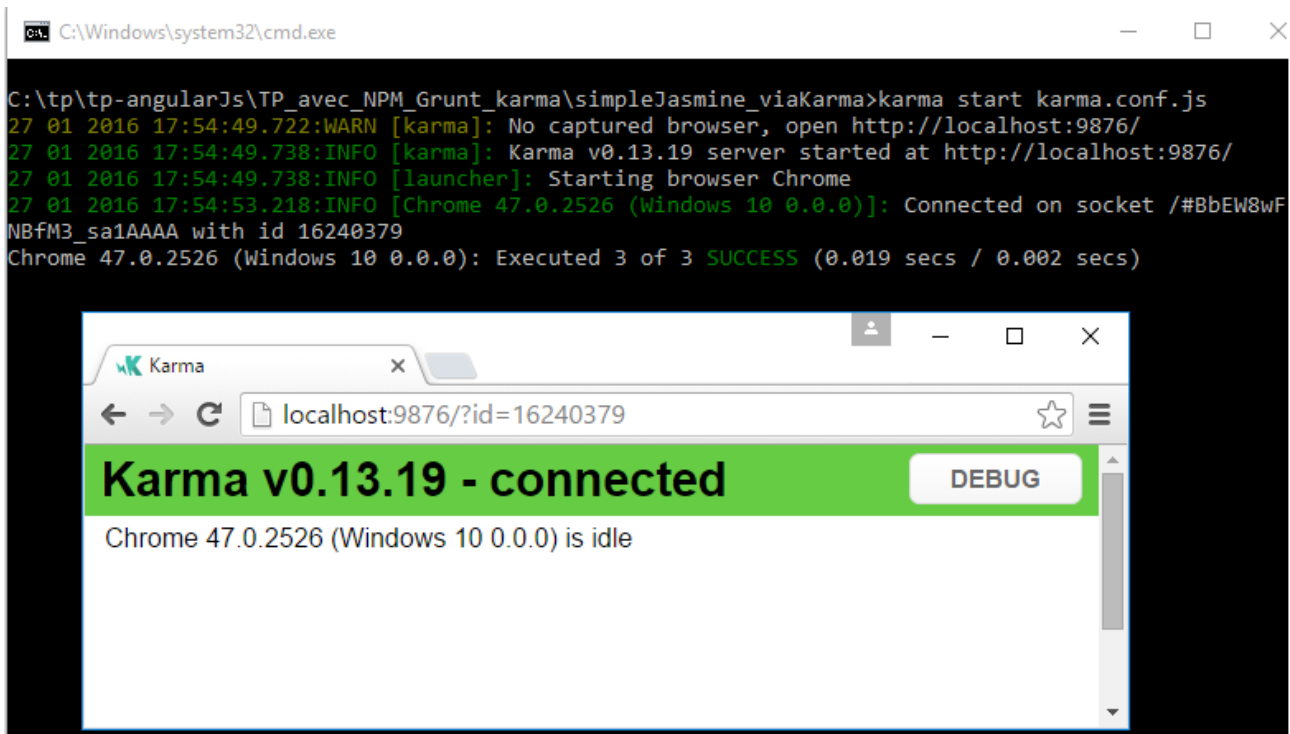
Nb.: il est souvent utile d'ajouter le "reporter" *kjhtml* dans le fichier *"karma.conf.js"*

Démarrage de karma :

```
karma start karma.conf.js
```

Il faut cliquer sur "**DEBUG**" pour faire apparaître les détails/résultats de "jasmine" dans un *nouvel onglet du navigateur* (karma-jasmine-html-reporter , "kjhtml")

L'arrêt de karma s'effectue via **ctrl-c**



## 1.8. Configuration et utilisation de Grunt

Installation (en mode global) du lanceur de Grunt en ligne de commande :

**npm install -g grunt-cli**

Installation (locale au projet) de quelques plugins souvent utiles :

**npm install grunt grunt-contrib-uglify grunt-karma --save-dev**  
ou bien édition de `package.json` + `npm update`

Installation (locale au projet) de quelques autres plugins souvent utiles :

fichier `package.json`

```
{
  "name": "karma-through-grunt",
  "version": "0.0.1",
  "description": "...",
  "main": "...",
  "scripts": {
    "test": "echo \"Error: no test specified\" && exit 1"
  },
  "author": "...",
  "devDependencies": {
    "karma-jasmine": "~0.3.6",
    "karma-chrome-launcher": "~0.2.2",
    "karma-jasmine-html-reporter": "~0.2.0",
    "grunt-contrib-clean": "~0.7.0",
    "grunt-contrib-jshint": "~0.12.0"
  }
}
```

**npm update**

Configuration de GRUNT via le fichier `gruntfile.js` :

```
module.exports = function(grunt) {
  // Configuration de Grunt
  grunt.initConfig({

    //NB: npm update (after change dependencies in package.json)
    pkg: grunt.file.readJSON('package.json'),

    clean: {
      // Deletes all old .js files, but skips old min.js files
      js: ["path/to/dir/old/*.js", "!path/to/dir/old/*.min.js"],

      // delete all files and directories here
      build: ["build/xx/yy", "dist", "dest"],
    },
  },
```

```

jshint: {
    all: ['gruntfile.js', 'js/**/*.js', 'test/**/*.js']
    // options in .jshintrc file
},

uglify: {
    my_target: {
        files: [
            { src: 'js/*.js', dest: 'dest/common.js'},
            { src: 'test/unit/*.js', dest: 'dest/test.js'}
        ]
    }
},

karma: {
    unit: {
        configFile: 'karma.conf.js'
    }
}
});

// A very basic logging task :
grunt.registerTask('basic_log', "", function() {
    grunt.log.write('Logging some stuff...').ok();
});

// Définition des tâches/plugins Grunt :
grunt.loadNpmTasks('grunt-contrib-clean');
grunt.loadNpmTasks('grunt-contrib-jshint');
grunt.loadNpmTasks('grunt-contrib-uglify');
grunt.loadNpmTasks('grunt-karma');

grunt.registerTask('default', [ 'basic_log' , 'clean' , 'jshint' , 'uglify' , 'karma' ]);
};

```

Exemple de contenu du fichier [.jshintrc](#)

```

{ "curly": true,
  "eqnull": true,
  "eqeqeq": true,
  "undef": true,
  "globals": {
    "jQuery": true
  }
}

```

Lancement de GRUNT (depuis le répertoire contenant gruntfile.js) :

**grunt**

## 2. MongoDB

### 2.1. Présentation

**MongoDB** est une base de donnée "Not Only SQL" orientée document.

Le format interne des données correspond à des **collections de documents "JSON"** (binarisés en BSON ).

### 2.2. Installation et Configuration

*mongodb-win32-x86\_64-2008plus-ssl-3.2.1-signed.msi* (environ 100 Mo à télécharger depuis "<https://www.mongodb.org/downloads#production>") permet une installation sous windows .

L'installation s'effectue par défaut dans **C:\Program Files\MongoDB\Server\3.2** .

Pour configurer une base de données sur l'ordinateur local , on peut :

- 1) préparer un répertoire (par exemple : C:\Prog\MongoDB )
- 2) préparer (par exemple) les sous répertoires suivants :
  - data/db** (pour les fichiers internes des données de la base)
  - data/log** (pour les fichiers de log)
  - dataset** (pour import/export de fichiers ".json" )

- 3) écrire un petit fichier de configuration "**mongodb.cfg**" :

```
systemLog:
  destination: file
  path: C:\Prog\MongoDB\data\log\mongod.log
storage:
  dbPath: C:\Prog\MongoDB\data\db
```

- 4) lancer éventuellement l'installation de "mongoDB" en tant que service windows (en tant qu'administrateur) :

```
set MONGO_HOME=C:\Program Files\MongoDB\Server\3.2
"%MONGO_HOME%\bin\mongod.exe" --config "C:\Prog\MongoDB\mongodb.cfg" --install
pause
```

- 5) démarrer si nécessaire le service "mongoDB" (en tant qu'administrateur) :

```
net start MongoDB
pause
```

ou bien effectuer un démarrage direct (sans service windows préalablement installé) :

```
"%MONGO_HOME%\bin\mongod.exe" --dbpath "C:\Prog\MongoDB\data\db"
```

- 6) lancer un "mongo shell" pour gérer la base de données :

```
"%MONGO_HOME%\bin\mongo.exe"
```

Exemples de commandes (mongo shell) :

```
db.collectionName.find();
db.clients.find();
```

...

NB: la base locale par défaut "test" est sans username et sans mot de passe.

## 2.3. Import de données au format ".json"

**import\_comptes\_dataset\_in\_test\_db.bat**

```
set MONGO_HOME=C:\Program Files\MongoDB\Server\3.2
cd /d "%~dp0"
%MONGO_HOME%\bin\mongoimport" --db test --collection comptes --drop
--file dataset/comptes.json --jsonArray
pause
```

une collection "mongoDB" est un peu l'équivalent d'une table (mais avec une structure complètement différente).

L'option **--drop** demande à d'abord supprimer tous les éléments de la collection avant d'importer le contenu du fichier .json .

L'option **--jsonArray** permet d'importer un tableau d'objet "json" si le fichier à importé est structuré de cette façon (par défaut le fichier .json ne comporte qu'un document/objet à importer).

## 2.4. Hébergement cloud via MongoLab

L'entreprise "**MongoLab**" se propose d'héberger des bases de données au format "mongoDB" sur un "Cloud" (ex : Amazon , Google , Azure) .

Cette entreprise propose une version gratuite "**free / sandBox / limitée à 500Mo**" qui est suffisante pour effectuer quelques essais rapides en mode "développement" .

**Mode opératoire :**

- Créer (gratuitement) un compte pour l'administration de la base . Mémoriser les paramètres.
- Valider l'adresse e-mail pour activer le compte "mongoLab" . Se connecter.
- Choisir une souscription de service (ex : " free/sandbox" ou ...)
- Créer une nouvelle base de données (ex : "**my\_mongo\_db**")
- Créer un utilisateur (ex : "*powerUser*" , "*myPwd*" )

- Lire les paramètres (URL de la base , ...)
- Administrer et gérer la base à distance (via mongo , mongoimport , ...)

Exemples de paramètres (pour base distante hébergée par MongoLab):

```
mongoimport -h ds041494.mongolab.com:41494 -d my_mongo_db  
            -c <collection> -u <user> -p <password> --file <input file>
```

exemple :

```
"%MONGO_HOME%\bin\mongoimport" -h ds041494.mongolab.com:41494 --db my_mongo_db  
--collection comptes --drop -u powerUser -p myPwd --file ../dataset/comptes.json --jsonArray
```

MongoShell (à distance) :

```
mongo ds041494.mongolab.com:41494/my_mongo_db -u <dbuser> -p <dbpassword>
```

URI de connexion (ex : depuis code nodeJs) :

```
mongodb://<dbuser>:<dbpassword>@ds041494.mongolab.com:41494/my_mongo_db
```

## 3. JONP (JSON Padding) – requêtes inter-domaines

### 3.1. Utilité et Principes

Un appel AJAX ne peut (par défaut) être exécuté que si l'URL est du même domaine que la page en cours (préalablement téléchargée via HTTP).

Pour dépasser la contrainte/limite "same origin policy" codée en dur dans les navigateurs WEB , on peut :

- soit effectuer des paramétrages "CORS" du coté serveur (ce qui ne fonctionne qu'avec des navigateurs récents)
- soit utiliser l'astuce "JSONP" exposée ci-après (fonctionnant avec tout type de navigateur mais limité au mode "GET")

JSONP signifiant "JSON Padding" est un mécanisme servant à contourner la contrainte "same origin policy" et autrement dit à permettre d'effectuer des requêtes inter-domaines .

Au lieu d'utiliser XMLHttpRequest pour effectuer un appel HTTP en mode GET pour récupérer des données JSON on peut s'appuyer sur la balise HTML <script> de façon à télécharger (en HTTP/GET) le code d'un appel de fonction javascript avec des données JSON en argument.

Exemple (issu de wikipedia) :

```
<script type="application/javascript"
  src="http://server.example.com/Users/1234?callback=parseResponse">
</script>
```

Réponse construite et retournée par le coté serveur (ex : PHP , NodeJs+ExpressJs , servlet java) :

```
parseResponse ( { "Name": "Foo", "Id": 1234, "Rank": 7 } );
```

#### Explications :

la balise <script> de HTML peut éventuellement être utilisée avec une URL absolue de façon à directement télécharger une librairie depuis le domaine source. Exemple :

```
<script src="http://crypto-js.googlecode.com/svn/tags/3.0.2/build/rollups/md5.js"> </script>
<script> var passhash = CryptoJS.MD5(password); ...</script>
```

Ce même principe de téléchargement peut être utilisé pour récupérer des données JSON encadrées par un appel de fonction javascript (appelée "padding" autour de JSON) .

L'URL comporte généralement un paramètre "callback" (ou "jsonp") permettant de spécifier au coté serveur quel doit être le nom de la fonction javascript qui doit encadrer les données JSON.



Lorsque le bloc `<script src="...?callback=parseResponse"></script>` est coté navigateur , remplacé par l'appel de la fonction javascript `"parseResponse()"` ou autre préparée dans la page par un autre bloc `<script></script>` , cet appel de fonction s'exécute alors et les données JSON sont affichées d'une manière ou d'une autre.

"JSONP" est une solution assez radicale qui ne fonctionne qu'en mode "GET" dans le cas où l'on fait confiance au coté serveur .

### 3.2. Exemple de code "NodeJs+ExpressJs" (du coté serveur) :

```
var express = require('express');
var app = express();

// GET http://localhost:8282/devises/1?callback=parseResponse pour devise de codeDevise=1
app.get('/devises/:numero', function(req, res,next) {

    var numDevise = Number(req.params.numero) ;
    var jsonp_callback = req.query.callback;

    var jsDeviseEuro = {
        codeDevise : 1 ,
        nom : 'Euro' ,
        change : 1.12
    };
    if(numDevise!==undefined && numDevise > 0){
        var jsDeviseObject = null;
        if(numDevise==1)
            jsDeviseObject = jsDeviseEuro;
        if(jsonp_callback!==null && jsonp_callback!==undefined){
            res.setHeader('Content-Type', 'application/javascript');
            res.write(jsonp_callback+"("+JSON.stringify(jsDeviseObject) + ");");
        }else {
            res.setHeader('Content-Type', 'application/json');
            res.write(JSON.stringify(jsDeviseObject));
        }
    }
    res.end();
});

app.listen(8282 , function () {
    console.log("nodeJs server listening at 8282");
});
```

<http://localhost:8282/devises/1?callback=parseResponseXy>

□ `parseResponseXy({"codeDevise":1,"nom":"Euro","change":1.12});`

### 3.3. Exemple d'appel via HTML+javascript (sans jQuery)

Cet exemple basique permet de bien comprendre les mécanismes JSONP :

```
<html >
<head>
  <script>
    function parseResponseXy(response){
      document.getElementById('jsonpResXy').innerHTML = JSON.stringify(response);
    }

    function dynamicAddJSONPscript(url_with_callback){
      var script = document.createElement('script');
      script.src = url_with_callback;
      var headDomNode = document.getElementsByTagName('head')[0];
      headDomNode.appendChild(script);
      headDomNode.removeChild(script);
    }
  </script>
</head>
<body>

<input type='button' value="dynamic JSONP Call (without jQuery)" onclick=
"dynamicAddJSONPscript('http://localhost:8282/devises/1?callback=parseResponseXy');" />

<br/><br/>
<div id="jsonpResXy"></div>
</body>
</html>
```

dynamic JSONP Call (without jQuery)

```
{"codeDevise":1,"nom":"Euro","change":1.12}
```

### 3.4. Exemple d'appel JSONP avec jQuery

```
<html >
<head>
```

```

<script src="lib/jquery-2.1.4.min.js"></script>

<script>

function parseResponseXy(response){
    document.getElementById('jsonpResXy').innerHTML = JSON.stringify(response);
}

function dynamicJQueryJsonpCall(url, query, secondLevelCallbackName){
    $.ajax({
        url: url,

        // The name of the callback parameter
        //('callback' for ?callback=jqueryDynamicIndirectCallback)
        jsonp: "callback",

        // Tell jQuery we're expecting JSONP
        dataType: "jsonp",

        data: query,

        // Work with the response (after indirect jquery Dynamic jsonp Callback)
        success: function( response ) {
            console.log( response ); // server response
            //parseResponse3(response); //my second level callback
            window[secondLevelCallbackName](response);
        }
    });
}

</script>
</head>
<body>

<input type='button' value="dynamic JSONP Call (with jQuery)" onclick=
"dynamicJQueryJsonpCall('http://localhost:8282/devises/4','', 'parseResponseXy');"/>

<br/><br/>
<div id="jsonpResXy"></div>
</body>
</html>

```

dynamic JSONP Call (with jQuery)

```

{"codeDevise":4,"nom":"Yen","change":122}

```

### 3.5. Exemple d'appel JSONP avec AngularJs 1 et \$http

js/mainCtrls\_jsonp\_data.js

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

myAjsApp.controller('DeviseCtrl', function ($scope,$http) {

    $scope.onJsonpFct = function (){

        $http.jsonp("http://localhost:8282/devises/"+$scope.numDevise
            +"?callback=JSON_CALLBACK")
            .success(function(data) {
                $scope.msg = "... with success";
                $scope.data = data; //JSON
            })
            .error(function (data) {
                $scope.msg = "Request failed";
            });

    };

});
```

*numDevise (1,2,3,4) :* 2

JSONP Call with \$http of AngularJs

**http://localhost:8282/devises/4?callback=JSON\_CALLBACK  
with success**

```
{"codeDevise":2,"nom":"Dollar","change":1}
```

```
code:2 , nom:Dollar , change:1
```

# XI - Annexe – énoncés (précis) de TP

## 1. Calcul de Tva (m-v-vm) et expérimentations \$scope

Développer *tva\_angularJs.html* en apportant les fonctionnalités suivantes :

### Titre principal ( MaMarque2)

#### calcul de TVA ( MaMarque2)

ht :

taux :  (20 %)

tva: 40

ttc (via basic expression): 240

ttc (via function call in expression): 240.00

newBrandName :

Pour l'expérimentation des "scopes", on pourra utiliser une structure proche de celle-ci :

```
...
<body ng-controller="MainCtrl">
  <h1> {{title}} ( {{brandName}})</h1>
  <div id="divTva" ng-controller="TvaCtrl">
    <h3> {{title}} ( {{brandName}})</h3>
    ...
    newBrandName : <input ng-model="newBrandName"/> <br/>
    <input type='button' ng-click="updateBrandName()"
      value="update BrandName" /> <br/>
    <input type='button' ng-click="updateBrandNameInParentScope()"
      value="update BrandName in parent scope" /> <br/>
  </div>
...
```

avec title ayant une valeur différente dans les 2 niveaux (body , divTva)  
et brandName ayant une valeur commune (héritée).

**NB:** **\$scope.\$parent.xy** permet d'accéder (en lecture/écriture) à la donnée "xy" du scope parent.

```
$scope.calcul_ttc= function(){
  var ttc = $scope.ht* (1 + $scope.taux/100);
  return ttc.toFixed(2); //arrondi à 2 chiffres après virgule.
};
```

## 2. Test d'événements + ng-show , ng-disabled

Partie1 :

**Evenements (exemples)**

☒ avec details

my background color :

\*\*\*

#####

a:

b:

operation:

res: 8

---

texte :

texte inverse (apres declenchement evenement): cba

Ne montrer la partie

*my background color* : `<input type='color' ng-model="my_background_color" />` `<br/>`

que si la case "avec details" est cochée via `ng-show="...expression vraie ou fause..."`

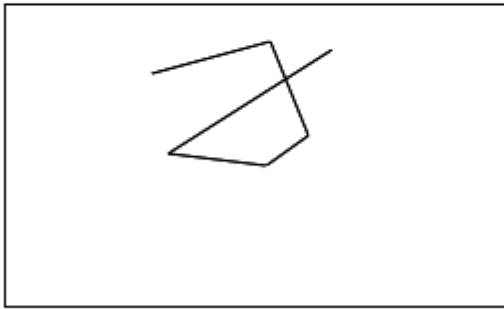
Rendre grisé le bouton de l'opération "a/b" (division) si la valeur de b est "undefined" , "" ou égale à 0 via `ng-disabled="...expression vraie ou fause..."`

Gérer le reste avec des gestions d'événements (ng-click , ng-change , ....)

Rappel `<body bgColor='...' > ....</body>`

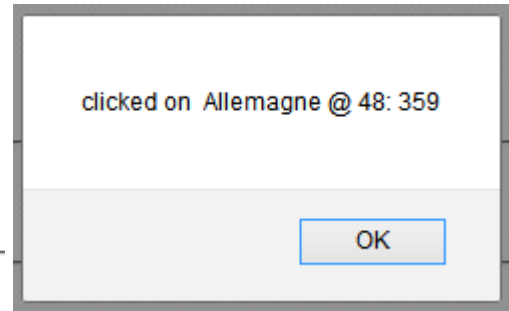
Partie2 :

France	Paris
Allemagne	Berlin
Espagne	Madrid
Italie	Rome



clear

click at x=163 y=22



```
<tr ng-repeat="p in pays.liste"> <td ng-click="pays.doClick(p, $event)"> .... </tr>
```

est compatible avec la structure suivante :

```
$scope.pays={};
$scope.pays.liste=[{nom:"France",capitale : "Paris"},{nom:"Allemagne",capitale : "Berlin"} ,
    {nom:"Espagne",capitale : "Madrid"},{nom:"Italie",capitale : "Rome"}];
$scope.pays.doClick = function(item, event) {
    alert("clicked on " + item.nom + " @ " + event.clientX + ": " + event.clientY);
}
```

```
<canvas id="myCanvas" width="250" height="150" style="border:1px solid #000000;"
    ng-click="log_coords($event)">
</canvas> <input type="button" value="clear" ng-click="clear_canvas()" />
```

Code permettant d'effacer le contenu d'un canvas html5 :

```
$scope.clear_canvas = function(){
    var canvasElement = document.getElementById("myCanvas");
    var ctx = canvasElement.getContext("2d");
    ctx.clearRect ( 0 , 0 , canvasElement.width, canvasElement.height );
}
```

code technique permettant d'ajuster les coordonnées en mode "compatible canvas html5" :

```
function x_y_canvas(e,canvasElement){
    var x;
    var y;
    if (e.pageX || e.pageY) {
        x = e.pageX;
        y = e.pageY;
    }
    else {
        x = e.clientX + document.body.scrollLeft + document.documentElement.scrollLeft;
        y = e.clientY + document.body.scrollTop + document.documentElement.scrollTop;
    }
    x -= canvasElement.offsetLeft;
    y -= canvasElement.offsetTop;

    $scope.xC = x;
    $scope.yC = y;
}
```

```
$scope.log_coords=function(event){
    var canvasElement = document.getElementById("myCanvas");
    x_y_canvas(event,canvasElement);//compute yC,yC and store in $scope
    var msg="click at x=" + $scope.xC + " y=" + $scope.yC;
    //$log.log(msg); //if myApp.controller('MainCtrl', function ($scope,$log){...}
    // to log in navigator web-console
    $scope.my_status_msg=msg;
    // window.status=msg; not working with recent navigator

    var ctx = canvasElement.getContext("2d");
    ctx.beginPath();
    ctx.moveTo($scope.x,$scope.y);//from last x,y
    ctx.lineTo($scope.xC,$scope.yC);//to new xC,yC
    ctx.closePath();
    $scope.x=$scope.xC; $scope.y=$scope.yC;//store last coord for next line
    ctx.stroke();
};
```



### 3. Test de \$route vers "subview"

Partie1 :

## Minibank App

---

**welcome minibank**

[identification client](#)

---

... status , mentions legales , ... [retour accueil](#)

Avec sous vue "welcome\_minibank.html"

## Minibank App

---

**identification client (minibank)**

numClient:

password:

[vers espace client identifie](#)

---

... status , mentions legales , ... [retour accueil](#)

avec sous vue "identificationClient.html"

**NB :** le lien hypertexte ne sera montré (via ng-show="...") que si le password est testé et considéré comme correct (par exemple égal à "pwd" + numClient) .

## Minibank App

---

**client identifie : 1**

alain Therieur

adresse: 123 - rue elle 75000 Par ici


telephone: 0102030405


email: alain.therieur@ici-ou-la-bas.fr


---

avec sous vue "clientIdentifie.html"

Dans partials , trois "subView" :

 clientIdentifie.html

 identificationClient.html

 welcome\_minibank.html

Dans une première version (sans service "REST") on pourra s'appuyer sur des jeux de données (à télécharger via \$http dans le répertoire "data") :

**client.json**

```
{
  "numero" : 1,
  "nom": "Therieur",
  "prenom": "alain",
  "adresse" : { "idAdr" :1 , "rue" : "123 - rue elle" , "codePostal" : "75000", "ville" : "Par ici" },
  "telephone" : "0102030405" ,
  "email" : "alain.therieur@ici-ou-la-bas.fr"
}
```

**Partie2 :**

En s'appuyant temporairement sur les données suivantes

**comptes.json**

```
[
  { "numero" : 1, "label" : "compte 1 (courant)", "solde" : 600.0 },
  { "numero" : 2, "label" : "compte 2 (LDD)", "solde" : 3200.0},
  { "numero" : 3, "label" : "compte 3 (PEL)", "solde" : 6500.0}
]
```

**operations.json**

```
[
  { "numero" : 1, "label" : "achat xy", "montant" : -50, "dateOp" : "2015-01-20"},
  { "numero" : 2, "label" : "achat zz", "montant" : -90, "dateOp" : "2015-02-08"},
  { "numero" : 3, "label" : "salaire", "montant" : 2000, "dateOp" : "2015-03-18"}
]
```

peaufiner la sous vue "clientIdentifie.html" avec un switch sur 3 "sous-sous-parties" :

- liste des comptes du clients (avec soldes)
- dernières opérations d'un compte sélectionné
- paramétrage d'un virement

```
$scope.transfert = {
  montant : 50,
  numCptDeb : 1,
  numCptCred: 2
};
```

action:  ▼

## liste des comptes

numero	label	solde
1	compte 1 (courant)	600
2	compte 2 (LDD)	3200
3	compte 3 (PEL)	6500

*Un click sur un numero de compte permet d'obtenir la liste des dernieres operations*

... status , mentions legales , ... [retour accueil](#)

action:  ▼ affichage des operations du compte selectionne : 2

## dernieres operations du compte 2

numero	label	montant	date
1	achat xy	-50	2015-01-20
2	achat zz	-90	2015-02-08
3	salaire	2000	2015-03-18

... status , mentions legales , ... [retour accueil](#)

action:  ▼

## parametrage virement

montant :   
 numCptDeb :   
 numCptCred :

... status , mentions legales , ... [retour accueil](#)

avec affichage d'un message "le montant de 50 a bien ete transfere du compte 1 vers le compte 2 "

Pour basculer de sous-sous-parties , une solution possible consiste à utiliser les directives **ng-switch** et **ng-include** comme le montre le code ci-après :

```

action: <select ng-model="renderPath" ng-click="message="">
    <option value="listeComptes"> liste des comptes </option>
    <option value="paramVirement">effectuer un virement</option>
    <!-- <option value="dernieresOperations">dernieres Operations</option>
         indirectement apres selection dans tableau -->
</select> {{message}}
<hr/>

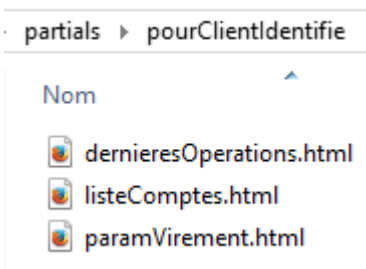
<div ng-switch on="renderPath">
    <div ng-switch-when="paramVirement">
        <ng-include src=" 'partials/pourClientIdentifie/paramVirement.html' " />
    </div>

    <div ng-switch-when="listeComptes">
        <ng-include src=" 'partials/pourClientIdentifie/listeComptes.html' " />
    </div>

    <div ng-switch-when="dernieresOperations" >
        <ng-include src=" 'partials/pourClientIdentifie/dernieresOperations.html' " />
    </div>

    <div ng-switch-default > *** </div>
</div>

```



<ng-include src=" *expression retournant url*" />  
 avec éventuelle valeur fixe via '...' au sein de "...."