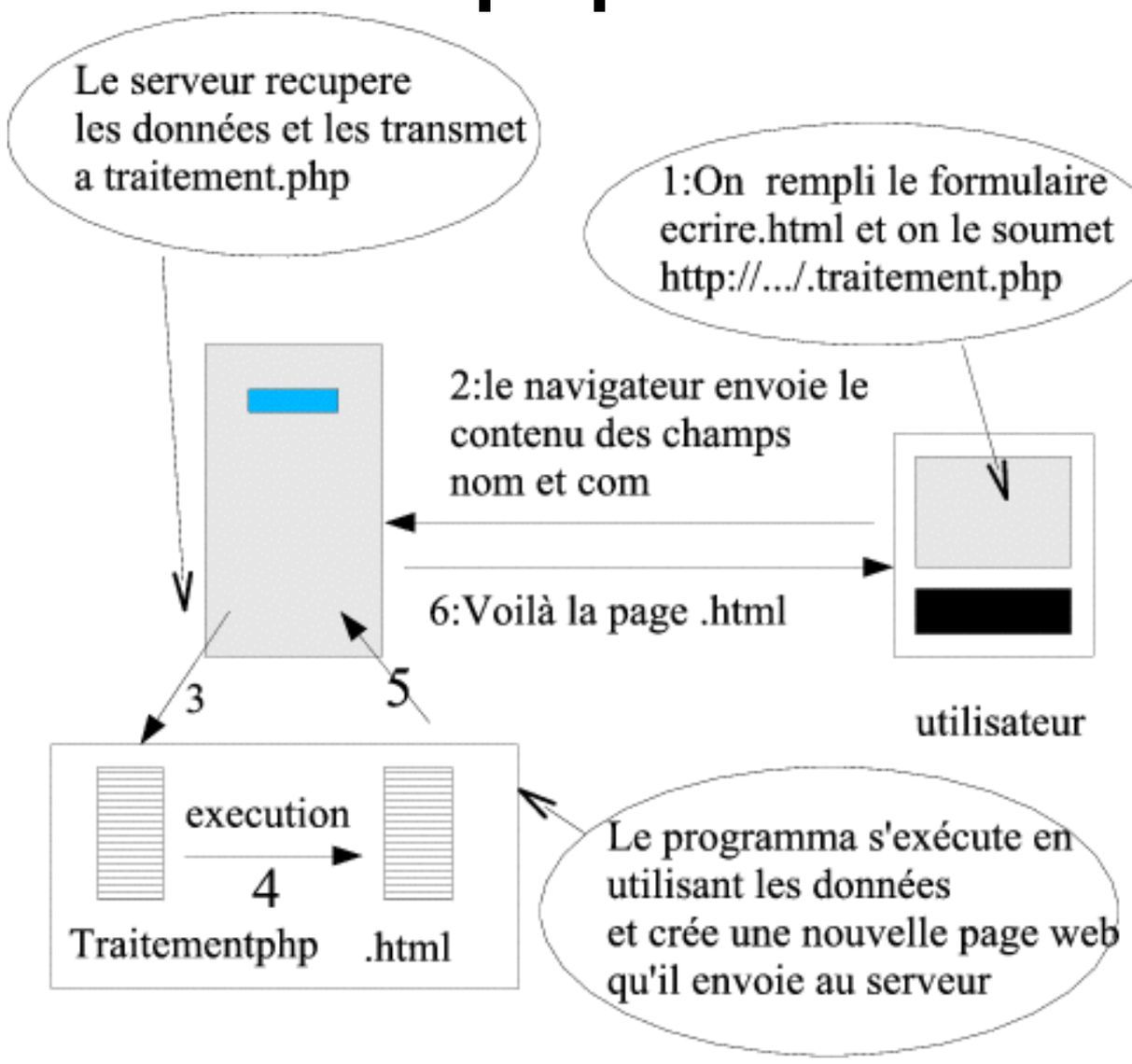
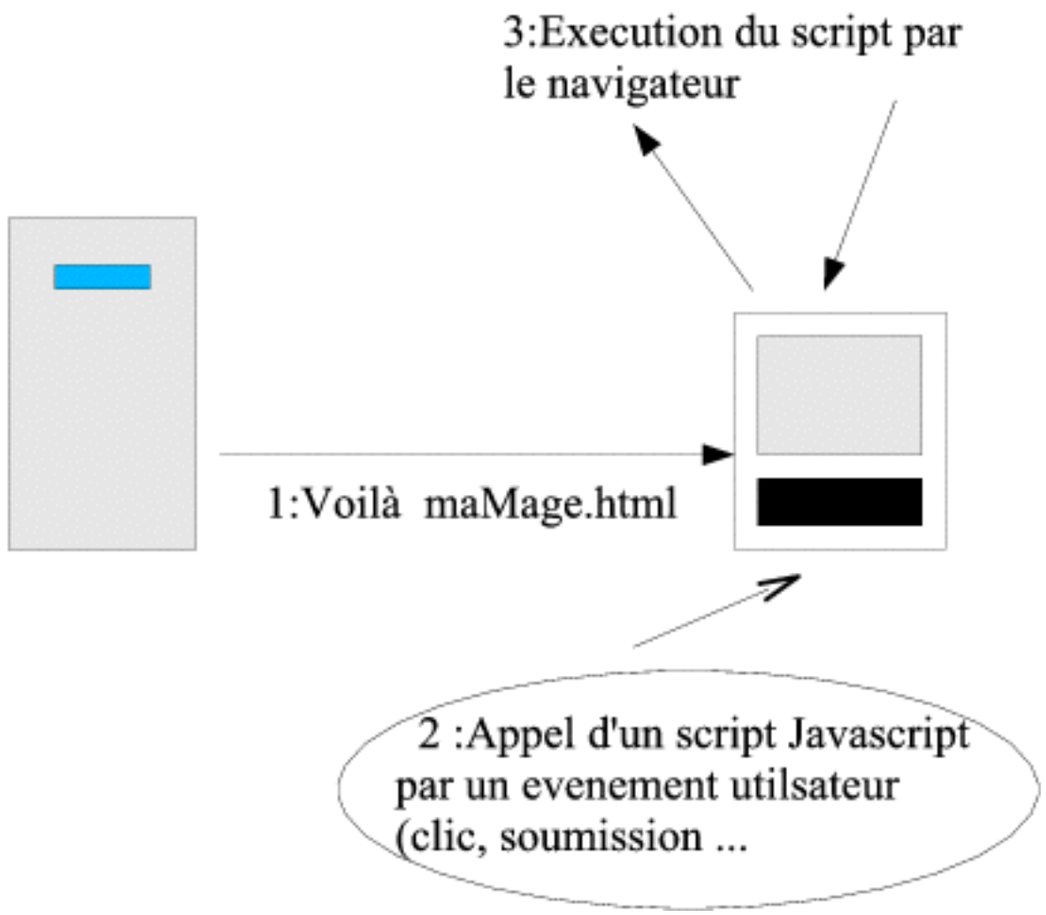
Javascript/Ajax
Web dynamique côté client
NFE102

Serge Rosmorduc
CNAM, 2015

Rappels


Le serveur recupere
les données et les transmet
a traitement.php
1:On rempli le formulaire
ecrire.html et on le soumet
http://.../.traitement.php
2:le navigateur envoie le
contenu des champs
nom et com
6:Voilà la page .html
3
5
utilisateur
execution
4
Traitementphp
.html
Le programma s'exécute en
utilisant les données
et crée une nouvelle page web
qu'il envoie au serveur


3:Execution du script par
le navigateur
1:Voilà maMage.html
2 :Appel d'un script Javascript
par un evenement utilsateur
(clic, soumission ...

Traitement « classique »

Javascript et dynamisme côté client

Javascript

Le DOM

AJAX

- en temps normal: une requête ➜ un rechargement de page

- pas toujours pratique

- ➜ Ajax: possibilité

  - d’envoyer une requête en javascript sans quitter la page courante

  - de traiter le résultat pour *modifier* le contenu de la page

Ajax

- **Asynchronous JavaScript and XML**
  - Asynchronous: ne bloque pas l'application.
  - Javascript : ben, c'est en javascript…
  - XML : *à l'origine,* les données circulaient plutôt en XML. Aujourd'hui: JSON

Ajax: fonctionnement

- objet XMLHttpRequest permet de manipuler la connexion
- création dépend du navigateur :

```
if (window.XMLHttpRequest) {
        xhr = new XMLHttpRequest();
} else if (window.ActiveXObject) {
        xhr = new ActiveXObject("Microsoft.XMLHTTP");
}
```

Objet XMLHttpRequest

- Une fois l'objet créé:
- la méthode open permet d'ouvrir une connection
- la méthode send permet d'envoyer une requête
- après la requête l'object contient (entre autre)
  - le code de retour
  - le document résultant de la requête

Ajax, exemple

```
<head><title> exemple</title>
  <script type="text/javascript">
  function exemple1(){
        var req = new XMLHttpRequest();
        req.open('GET', 'test.php', false);
        req.send(null);
        alert(req.status);
       if(req.status == 200)
            alert(req.responseText);
  }
  </script>
  </head>
  <body>
    <h1 onclick='exemple1()'>go</h1>
    <h1 onclick='alert("debloqué")'>bloquer</h1>
  </body>
```

Méthodes

- open (*méthode , URL , type*)
  - méthode: GET, POST…
  - URL
  - *type:* true ou false : asynchrone(true) ou synchrone(false)
- send (argument)
  - argument: unnom=unevaleur&autrenom=autrevaleur
- abort
- getResponseHeader(header)
- setRequestHeader(header, value)

  si méthode POST changer le MIME :

```
httpRequest.setRequestHeader('Content-Type',
    'application/x-www-form-urlencoded');
```

Attributs

- status (200, 404 ...)
- statusText(OK,notfound...)
- responseText : du texte
- responseXML : du xml
- readyState : état de l’objet (entier de 0 à 4)
- onreadystatechange

Passage de paramètres (GET)

- En mode GET: passer les valeurs des paramètres dans l'URL :

```
var params= "nom=" + encodeURIComponent(nom) +
                    "&prenom="+ encodeURIComponent(prenom);
req.open("POST", "http://site/nfe102/page.php?"+params, true);
```

Passage de paramètres (POST)

- En mode POST: deux manipulations

```
req.open("POST", "http://site/page.php", true);
// content-type obligatoire…
req.setRequestHeader("Content-type",
                     « application/x-www-form-urlencoded");
// on encode comme pour get…
req.send("nom=" + encodeURIComponent(nom));
```

Exemple Asynchrone

- Non bloquant
- n'attend pas la réponse pour continuer son exécution,
- doit pouvoir connaître l' état de traitement de la requête : **readystate**
- doit être prévenu des changements et décider quoi faire : onreadystatechange

readyState

- Les états de readyState :
  - 0 : non initialisé.
  - 1 : connexion établie.
  - 2 : requête reçue.
  - 3 : réponse en cours.
  - 4 : terminé. (le plus utile)

```
<head><title> exemple </title>
  <script type="text/javascript">
    function exemple2(){
     var req = new XMLHttpRequest();
     req.onreadystatechange=function(){
          if(req.readyState  == 4){
               alert(req.status);
               if(req.status == 200)
                    alert(req.responseText);
          }
    }
    req.open('GET', 'test.php',true);//
    req.send(null);}
</script> </head> <body>
    <h1 onclick=‘exemple2()'>GO!</h1>
    <h1 onclick='alert("pas")'>pas bloqué</h1>
</body>
```

Interaction avec la page

- via le DOM ou innerHTML

```
<head><title> exemple </title>
  <script type="text/javascript">
    function exemple2(){
          var req = new XMLHttpRequest();
          req.onreadystatechange=function(){
              if(req.readyState  == 4){
                  if(req.status == 200)
                        document.getElementById("unid").innerHTML=
                              req.responseText;
          }}
    req.open('GET', 'test.php',true);//
    req.send(null);}
</script> </head> <body>
    <h1 onclick=‘exemple2()'>go!</h1>
    <div id="unid"></div>
</body>
```

Extraction du JSON

- Si le résultat est du JSON, on doit l'extraire:

var résultat = **JSON.parse(req.responseText);**

JQuery

- En pratique: on utilise des bibliothèques (masquent les différences entre navigateur)
- JQuery
  - accès au DOM simplifié (moins avec HTML5)
  - méthodes simplifiées pour les requêtes ajax
  - décoration *a posteriori* du html

JQuery : guide de survie

- $("...") permet de désigner un ou plusieurs éléments d'une page à partir d'un sélecteur css. Exemple $("#unId")
- Sur un élément (ou ensemble d'éléments) :
  - .text() : texte de l'élément
  - .html(): innerHtml
  - .val() : valeur d'un élément de formulaire

JQuery : guide de survie

- utilisables dans les deux sens:
  - var t= $("#id").text();
  - ou
  - $("#id").text("un texte…");

JQuery : guide de survie

- Pour créer de nouveaux éléments:
- $("#unId").empty() : vide un élément
- $("<h1></h1>").text("du texte") : crée un h1, avec un texte donné. L'élément n'est pas ajouté
- $('#unId').append($("<h1></h1>").text("du texte")) : ajoute l'élément créé à un autre élément.

JQuery : guide de survie

- $("#unId").click(fonction…) : fixe le "on click" d'un élément…
- plein d'autres possibilités.
- fonctionne sur un ensemble d'éléments:
  - $("p").empty() : vide *tous* les paragraphes !

Le même exemple avec JQuery

```
<head><title> exemple </title>
   <script src="http://code.jquery.com/jquery-1.10.2.js"></script>
   <script type="text/javascript">
      function demoJQuery() {
         $("#unid").text("en attente...");
         $.post(
               "lente.php", [],
               function (data, status) {
                  $("#unid").html("<ul></ul>");
                  for (var i=0; i < data.length; i++) {
                     $("#unid ul").append($("<li>").text(data[i]));
                  }
               });
       }
      }
   </script> </head> <body>
   <h1 onclick='demoJQuery()'>cliquer</h1>
   <div id="unid"></div>
</body>
```

Formulaire dynamique

- Certaines parties du formulaire sont remplies à partir d'interrogations http
    - exemples
        - recherche google
        - remplissage de champs (code postal/ville)
        - …
- formulaire à nombre variable de champs

Affichage de données

- Remplissage d’une information à partir d’une requête JSON
- pratique, par exemple, pour un fil d’information
- exemple possible : Facebook

Problèmes de référencement

- Inconvénient possible de l'utilisation massive d'Ajax
  - forum en ligne, tout en une page
  - l'affichage d'un message est fait en ajax/json
  - tous les messages ont la même URL
  - impossible à indexer pour google.

Frameworks

Javascript

fonctionalités des frameworks

- liens modèle-vue (données /visualisation)
- actions
- navigation: modèle de routage

AngularJS (version 1.2)

Serge Rosmorduc
Conservatoire National des Arts et Métiers

qu'est-ce qu'angular JS ?

- Un framework applicatif Javascript
- Fournit :
  - une architecture MVC **monopage**
  - **un moyen d’interagir facilement en REST**
  - **plein d’autres choses:**
    - **modularité**
    - **routage**

Rappels sur Javascript

- **Langage créé pour Netscape en 1995 par ;**
- **syntaxe inspirée de C**
- **nom inspiré de java**
- **sémantique inspirée de scheme (donc en partie fonctionnel)**

Notion de clôture

- **Def: variable libre dans une fonction: variable qui n’est ni un paramètre, ni une variable locale**

- **Fermeture (ou clôture): fonction dotée d’un environnement comprenant des variables libres.**

Clôture

```
function accumulateur(depart) {
    var s= depart;
    var f= function () {
        s= s + 1;
        return s;
    }
    return f;
}

var f1= accumulateur(10);
var f2= accumulateur(20);
afficher("f1 " + f1()); // 11
afficher("f1 "+f1()); // 12
afficher("f2 "+ f2()); // 21
afficher("f1 "+f1()); // 13
```

s variable libre pour f

chaque appel d'accumulateur retourne une **fonction** f qui a dans sa clôture une variable libre s.

f1 et f2 ont chacune leur valeur de s; chaque appel modifie la valeur dans la clôture.

Objet en javascript

- **Un objet peut être défini directement :**
  - **var p= {nom: "Wirth", prenom: "Niklaus"}**
- Ou à travers un *constructeur*

```
var p= new Personne("Wirth","N.");

alert(p.nom);
```

```
function Personne(n,p) {
  this.nom= n;
  this.prenom= p;
}
```

- ne pas oublier **new**
- **on peut ajouter des propriétés après coup:**

```
p.age= 60;
```

Méthode en javascript

- **Une fonction qui est une propriété de this.**
- **appel sur l'objet:**

```
var p= new
Personne("a", "b");

var s= p.nomComplet();
```

```
function Personne(n,p) {
  this.nom= n;
  this.prenom= p;
  this.nomComplet= function() {
    return this.prenom+
           " "+ this.nom;
  }
}
```

Intéraction objet/clôture

- **toutes les propriétés de this sont publiques**
- **on peut utiliser la clôture pour « simuler » des variables privées**
- **nom et prénom sont ici invisibles de l'extérieur.**
- **on peut seulement appeler nomComplet()**

nom et prenom dans la clôture

```
function Personne(n,p) {
  var nom= n;
  var prenom= p;
  this.nomComplet= function() {
     return prenom+ " "+ nom;
  }
}
```

pas de **this** !

Objet et prototype

- **La fonction constructeur représente la classe**

- On lui associe un **prototype,** où on ira chercher les variables d'instances ou méthodes non déclarées

```
function Personne(n,p) {
  this.nom= n;
    this.prenom= p;
}

Personne.prototype.nomComplet= function() {
  return this.prenom+ " "+ this.nom;
}
```

Objets et prototypes

- quand on cherche **p.maPropriété** :
  - on cherche d'abord directement dans p
  - puis dans le prototype de p
  - puis dans le prototype du prototype
- en lecture, pas de problème
- en modification: p.maPropriété= 40; ne modifie pas le prototype.
- attention aux objets dans les objets… manipulation par adresse !

Ecosystème javascript

- (encore un)
- node (npm) : interpréteur javascript étendu
- bower : gestionnaire de dépendance
- grunt : « make » de javascript
- karma : framework de tests

On revient à angular

Idées principales

- le document html sert de template
    - html est étendu par de nouveaux attributs (et balises, etc…)
    - angular va lire le document html d'origine, et modifier le DOM pour créer l'affichage final
    - interprétation de la template côté client
    - l'essentiel des communications avec le serveur se fait en ajax…

Exemple 0

```
<!DOCTYPE html>
<html>
    <head>
        <title>Exemple…</title>
        <meta charset="UTF-8">
        <script src="js/bower/angular/angular.min.js"></script>
    </head>
    <body ng-app>
        {{ 4 + 5}}
    </body>
</html>
```

expression calculée par angular. protégée contre Cross-site-scripting, etc.

délimite la zone où angular va modifier le DOM

Exemple 1

```
<!DOCTYPE html>
<html>
    <head>
        <script src="js/bower/angular/angular.min.js"></script>
         <script>
             var app = angular.module("AppGLG", []);
             var controller = app.controller(
                     "GLGController",
                     function ($scope) {
                         $scope.hello= "salut";
                     }
                 );
         </script>
    </head>
    <body ng-app="AppGLG">
        <div ng-controller="GLGController">
             <input ng-model="hello"/>
             <input ng-model="hello"/>
             {{hello}}
        </div>
    </body>
</html>
```

le contrôleur permet l'accès au modèle.

affichage de la variable « hello » dans le scope.

ces deux inputs sont synchronisés sur la même entrée du modèle!

Exemple 2

```
<!DOCTYPE html>
<html>
    <head>
        <script src="js/bower/angular/angular.min.js"></script>
         <script>
             var app = angular.module("AppGLG", []);
             var controller = app.controller("GLGController",
                     function ($scope) {
                         $scope.personne
                                 = {nom: "Lovelace", prenom: "Ada"};
                     }
             );
         </script>
    </head>
    <body ng-app="AppGLG">
        <div ng-controller="GLGController">
            <fieldset>
                <input ng-model="personne.nom"/>
                <input ng-model="personne.prenom"/>
                {{personne.prenom}} {{personne.nom}}
            </fieldset>
        </div>
    </body>
</html>
```

Le scope peut aussi contenir des objets !

concepts principaux

- **module** : unité architecturale de base. En particulier, une application est un module.

- **contrôleur** : fait le lien entre le modèle d'une part et la visualisation (html) d'autre part.

- **directives** : extensions du langage html fournies par angular. On peut en créer soi-même. C'est **la** manière de manipuler le DOM avec angular.

- **service** : fonctionnalités transverses, qui ne sont ni des contrôleurs, ni des directives… généralement utilisées par ceux-ci.

Une application angular

```
var monApp =
    angular.module("demoApp",["ngRoute"]);
```

une application est un module

nom de l'application, à utiliser comme valeur de l'attribut **ng-app**

liste des modules dont on dépend

Les modules servent de fabrique pour les services, les contrôleurs, les directives…

Un contrôleur

```
var controller =
 monApp.controller(
  "personneCtrl",
    function ($scope) {
       $scope.personne=
          {nom: "",
           prenom: ""
          };
    }
);
```

créé par l'application

nom du contrôleur

fonction d'initialisation du contrôleur

- les arguments de la fonction sont injectés par **nom.**
- **$scope** est un objet qui permet:
  - d'accéder aux données du modèle
  - de définir les fonctions appelées lorsque l'utilisateur interagit avec l'interface

La vue

nom de l'application

nom du contrôleur

```
<body ng-app="demoApp">
  <div ng-controller="personneCtrl">
     <input ng-model="personne.nom"/>
      <input ng-model="personne.prenom"/>
        {{personne.prenom}} {{personne.nom}}
  </div>
</body>
```

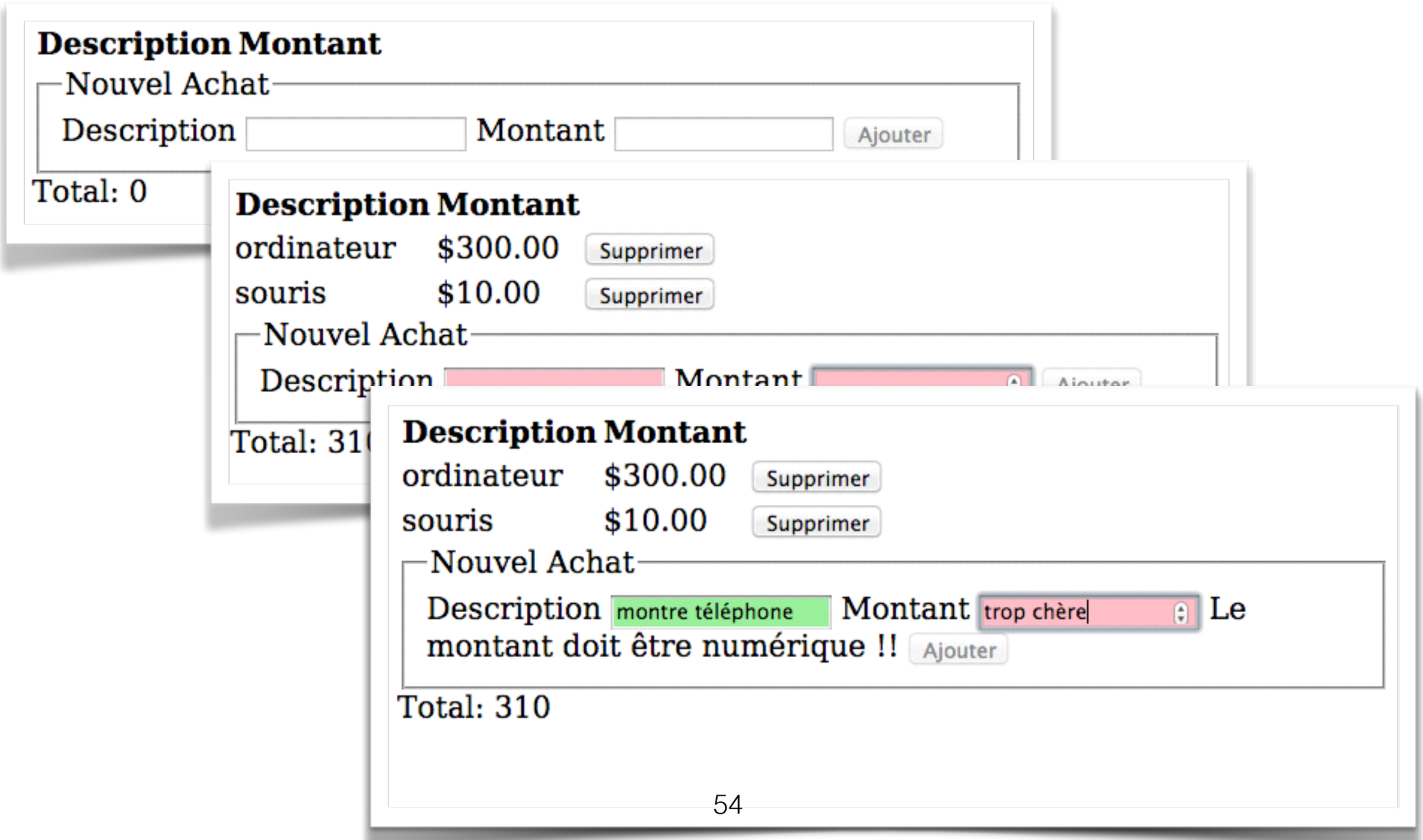
- **ng-app** : zone prise en charge par l'application. Souvent sur html
- **ng-controller**: zone prise en charge par un contrôleur. Il peut y avoir plusieurs contrôleurs sur une même page
- **ng-model**: lie un contrôle html à une variable du $scope

Services et injection de dépendance

- La fonction définie par le contrôleur reçoit automatiquement les **services** dont les noms sont passés comme paramètre, comme **$scope**

- **dans certains cas, il faut avoir inclus le module qui définit le service en question (à la fois comme code javascript et l'avoir déclaré comme dépendance de l'application)**

Exemple: liste d'achats



Modèle

```
function Achat(description, montant) {
                this.description = description;
                this.montant = montant;
}
function ListeAchats() {
   var list = [];

this.ajouter = function (a) {
     list.push(a);
}

this.products = function () {
  return list;
}
…
```

Modèle

```
function ListeAchats() {
  …
 this.enlever= function (p) {
   var pos= -1;
   for (var i=0; i < list.length; i++) {
     if (list[i]=== p) {
       pos= i;
       break;
     }
   }
   if (i !== -1)
     list.splice(pos, 1);
 }
 this.total = function () {
   var r = 0;
   for (var i = 0; i < list.length; i++) {
     r += list[i].montant;
   }
   return r;
 }
}
```

Contrôle

```
var app = angular.module("additionApp", []);
var controller = app.controller(
  "appCtrl",
  function ($scope) {
   $scope.data = {};
   $scope.data.achat = new Achat("", "");
   $scope.data.achats = new ListeAchats();
   $scope.ajouter = function () {
         $scope.data.achats.ajouter($scope.data.achat);
         $scope.data.achat = new Achat("", "");
      }
   $scope.enlever= function (p) {
         $scope.data.achats.enlever(p);
   }
  }
);
```

- définit dans le $scope:
  - une variable achats pour la liste d'achats
  - une variable achat pour un nouvel achat
  - une fonction ajouter pour gérer l'ajout d'un nouvel achat
  - une fonction enlever pour supprimer un achat
- aucune information sur la visualisation…

Vue: partie liste

```
<html ng-app="additionApp"> … <body>

<div ng-controller="appCtrl">
  <table>
  <tr ng-repeat="a in data.achats.products()">
    <td>{{a.description}}</td>
    <td>{{a.montant| currency}}</td>
    <td><button ng-click="enlever(a)">Supprimer</button></td>
  </tr>
</table>
```

- **ng-repeat**: attribut qui répète l'élément qui le porte
- a in data.achats.products() : data.achats = propriété du $scope

Vue: partie liste

```
<html ng-app="additionApp"> … <body>
<div ng-controller="appCtrl">
  <table>
  <tr ng-repeat="a in data.achats.products()">
    <td>{{a.description}}</td>
    <td>{{a.montant| currency}}</td>
    <td><button ng-click="enlever(a)">Supprimer</button></td>
  </tr>
</table>
```

- **{{a.description}} :** a est un achat
- **{{a.montant l currency}}** applique le filtre « currency » à a.montant. Affiche une unité monétaire devant le montant.

Vue: partie liste

```
<html ng-app="additionApp"> … <body>

<div ng-controller="appCtrl">
  <table>
  <tr ng-repeat="a in data.achats.products()">
    <td>{{a.description}}</td>
    <td>{{a.montant| currency}}</td>
    <td><button ng-click="enlever(a)">Supprimer</button></td>
  </tr>
</table>
```

- **ng-click="enlever(a)":** quand on presse le bouton, on appelle $scope.enlever(a), où a est le produit courant.
- le modèle est mis à jour… et l'affichage aussi! (cf. le total)

Vue: formulaire (version simple)

```
Description <input name="description"
                ng-model="data.achat.description" required/>
Montant <input name="montant"
                ng-model="data.achat.montant"
                type="number" required/>
<button ng-click="ajouter()">Ajouter</button>
```

- quand on presse sur le bouton, on appelle la méthode ajouter(), qui utilise les valeurs stockées dans le $scope pour ajouter un achat à la liste

- l'affichage est automatiquement mis à jour.

Vue: formulaire (version avancée)

```
<form name="addAchat" novalidate ng-submit="ajouter()">
  Description <input name="description"
                ng-model="data.achat.description" required/>
  Montant <input name="montant"
                ng-model="data.achat.montant"
                type="number" required/>
    <span ng-show="addAchat.montant.$error.number">
      Le montant doit être numérique !!
    </span>
    <button type="submit" ng-disabled="addAchat.$invalid">Ajouter</
button>
  </form>
```

- validation par angular
- utilise beaucoup les attributs « normaux » de html

Forumulaire, les détails...

```
<form name="addAchat" novalidate ng-submit="ajouter()">
```

- pour bénéficier de la validation par angular, le formulaire et ses champs doivent avoir un nom **(name)**

- **novalidate:** empêche la validation par le navigateur -> c'est angular qui s'en charge

- ng-submit: code du $scope appelé lorsqu'on a pressé le bouton submit

Formulaire (encore des détails)

```
Description <input name="description"
                   ng-model="data.achat.description" required/>
Montant <input name="montant"
                   ng-model="data.achat.montant"
                   type="number" required/>
```

- montant et description sont des champs liés à des variables du $scope

- required et type=number: informations utilisées pour la validation

Formulaire...

```
<button type="submit" ng-disabled="addAchat.$invalid">
                Ajouter</button>
</form>
```

- angular crée automatiquement une variable du même nom que le formulaire dans le $scope

- **ng-disabled**: le bouton submit est désactivé si la propriété **addAchat.$invalid** est vraie.

Formulaire, message d'erreur

```
Montant <input name="montant"
              ng-model="data.achat.montant"
              type="number" required/>
<span ng-show="addAchat.montant.$error.number">
  Le montant doit être numérique !!
</span>
```

- **ng-show:**
  - le span n'est affiché que si addAchat.montant.$error.number est vrai
  - donc si la contrainte **type='number'** du champ montant n'est pas vérifiée

Vue : total

```
Total: {{data.achats.total()}}
```

- automatiquement mis à jour quand le modèle est modifié

Ajax et Angular

- Intéraction avec le web: service **$http,** injecté dans le contrôleur

```
app.controller("personListCtrl", function ($scope, $http) {
    $scope.loadList = function () {
        var taillePage = $scope.data.taillePage;
        var page = $scope.data.page;
        var filtre = $scope.data.filtre;

        $http({
            method: 'get',
            url: 'json/person',
            params: {
                offset: (page - 1) * taillePage,
                limit: taillePage,
                filtre: filtre
            }
        }).success(function (resultat) {
            $scope.data.liste = resultat.data;
            $scope.total = resultat.total;
        });
};
```

$http

- la fonction prend comme argument une description de la requête

- ou fonctionne comme un objet pour appeler directement post ou get

- elle retourne une **promise**

  - **à laquelle on peut appliquer les fonctions success() ou error() pour dire ce qui se passe quand la requête se termine**

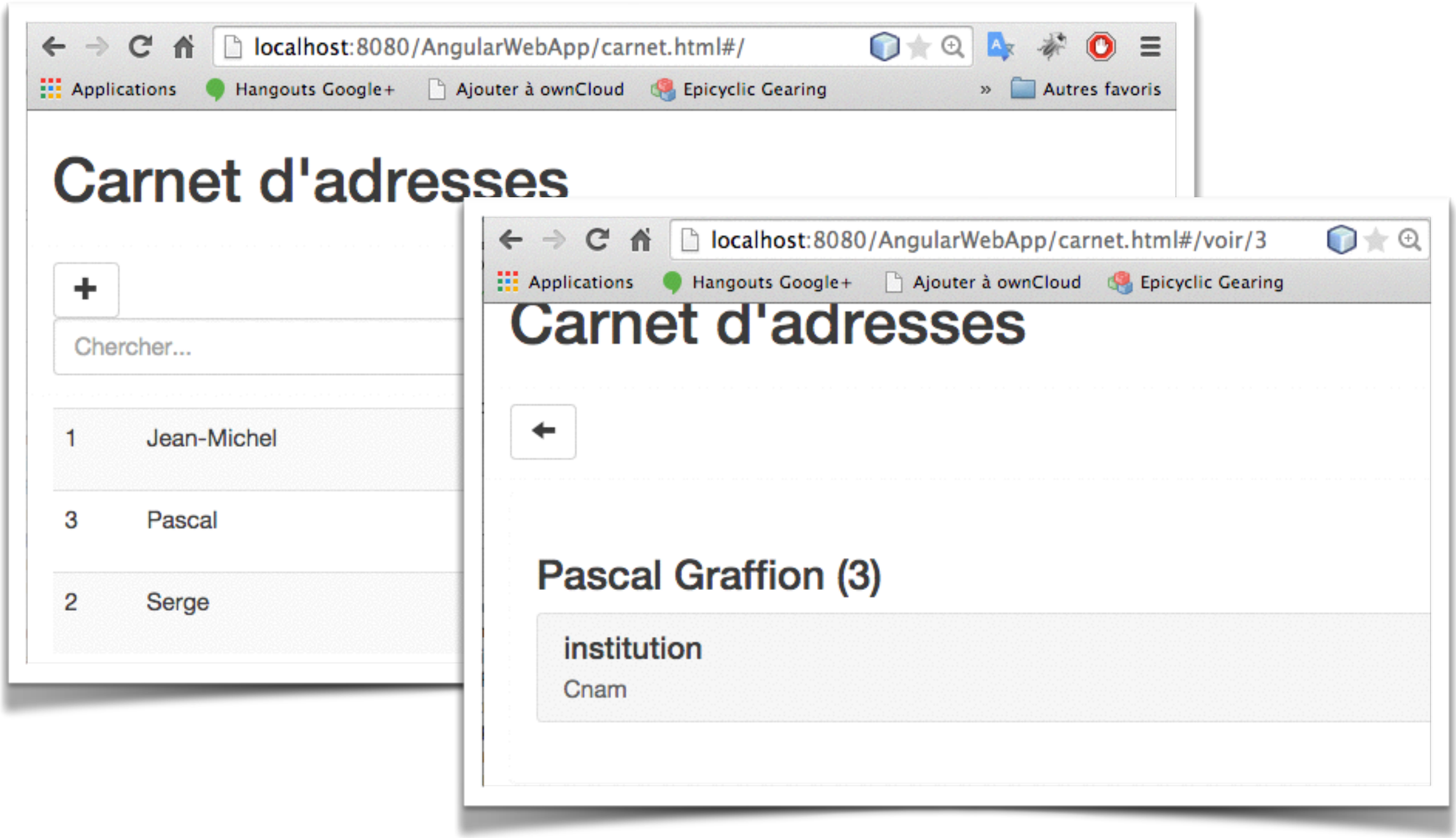
$http

- **typiquement:**
  - **on envoie une requête vers un serveur REST**
  - **paramètres tirés des données du $scope, envoyés en JSON**
  - **au retour: dans success, on fournit une fonction qui est appelée en cas de succès. idem pour error**
  - **premier argument de ces fonctions: les données retournées, typiquement du JSON.**

Routage

- **Application mono-page: c’est un peu limité!**
- **on veut dans doute pouvoir bookmarquer certains écrans**
- **exemple: carnet d’adresse…**

Routage


localhost:8080/AngularWebApp/carnet.html#/
Applications
Hangouts Google+
Ajouter à ownCloud
Epicyclic Gearing
Autres favoris
Carnet d'adresses
+
Chercher...
1 Jean-Michel
3 Pascal
2 Serge
localhost:8080/AngularWebApp/carnet.html#/voir/3
Applications
Hangouts Google+
Ajouter à ownCloud
Epicyclic Gearing
Carnet d'adresses
Pascal Graffion (3)
institution
Cnam

Routage

- **Il est utile d'avoir des URL différentes sur certains écrans…**
- **mais un « vrai » changement de page ne permet pas de conserver les données javascript…**
- **du coup, idée: changer d'URL sans changer de page…**
- **comment ???????????**

Routage

- **astuce ! le '#' en fin d'adresse:**
  - si on est sur http://www.cnam.fr/page.html
  - et qu'on saute à http://www.cnam.fr/page.html#chapitre1 …
  - on reste sur la même page : pas de rechargement, les données javascript sont intactes
- mais l'URL est différente !!!

Routage

- URL de l’application :
  - http://localhost:8080/AngularWebApp/carnet.html#/
- URL de la page de P. Graffion:
  - http://localhost:8080/AngularWebApp/carnet.html#/voir/3

Routage

- Ça fonctionne
- C’est moche
- C’est pratique: une seule vraie URL, donc les URL de ressources peuvent être relatives
- On peut utiliser aussi les fonctionnalités de HTML5 pour la gestion de l’historique, sauf que ça fonctionne actuellement moins bien

Routage: mise en place

- inclure angular-route:

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

- ajouter le module ngRoute à l'application :

```
var app = angular.module("carnetApp", ["ngRoute"]);
```

Routage

- Configurer le routage:

```
app.config(['$routeProvider', function ($routeProvider) {
        $routeProvider.when("/", {
            templateUrl: 'carnet/list.html',
            controller: 'personListCtrl'
    });
    $routeProvider.when("/creer",
      {templateUrl: 'carnet/editer.html'});
    $routeProvider.when("/voir/:id", {
        templateUrl: 'carnet/voir.html',
        controller: 'personDisplayCtrl'
    });
    $routeProvider.otherwise({
        redirectTo: "/"
    });
}]);
```

Règle simple

```
$routeProvider.when("/creer",
                    {templateUrl: 'carnet/editer.html'});
```

- Quand l'adresse est … maPage.html#/creer
- on va utiliser la *template* définie par le fichier carnet/editer.html
- les templates sont des bouts de code html, pas des pages entières

Utilisation des templates

- Dans la page principale, on donne la position de la template grâce à l'attribut **ng-view**:

```
<!DOCTYPE html>
<html ng-app="carnetApp">
    <head>
        <title>Carnet d'adresses</title>
        <script src="js/bower/angular/angular.min.js"></script>
        <script src="js/bower/angular-route/angular-route.min.js"></
script>
        <script src="js/app/carnet.js"></script>
    </head>
    <body>
        <h1>Carnet d'adresses</h1>
        <div ng-view>
            (contenu...)
        </div>
    </body>
</html>
```

la template sera injectée ici

ce contenu sera remplacé

Templates

```
$routeProvider.when("/", {
  templateUrl: 'carnet/list.html',
  controller: 'personListCtrl'
});
```

- on peut préciser en même temps la template et le contrôleur (autre solution: faire le lien directement dans la template)

Templates

```
$routeProvider.otherwise({
     redirectTo: "/"
 });
```

- $routeProvider.otherwise: donne la page par défaut
- redirectTo: renvoie vers une autre page.

Templates

```
$routeProvider.when("/voir/:id", {
                templateUrl: 'carnet/voir.html',
                controller: 'personDisplayCtrl'
});
```

- On peut ajouter des arguments dans le path, comme ici « :id »
- le contrôleur de la page en question pourra y accéder grâce à **$routeParams :**

```
app.controller("personDisplayCtrl", function ($scope, $http,
$routeParams) {
    var id = $routeParams.id;
    $http.get("json/person/" + id).success(function (d) {
        $scope.person = d;
    });
});
```

filtres

- **permettent de modifier ce qui est affiché:**
  - **prennent en charge la mise en forme des variables du scope**
  - **filtrent les listes pour n’en afficher qu’une partie, les trier, etc…**
- **on peut créer ses propres filtres**

Filtres

```
<script>
  var app = angular.module("demoFiltreApp", []);
  var controller = app.controller("demoFiltreCtrl",
    function ($scope) {
      $scope.produits= [
        {des: 'livre1', prix: 30},
        {des: 'livre3', prix: 20},
        {des: 'livre4', prix: 5},
        {des: 'abat-jour', prix: 26},
      ];
      $scope.prix= 133.34423344;
      $scope.texte = "Un beau soir d'été";});
</script>
<ul>
  <li>{{texte| uppercase}}</li>
  <li>{{texte| lowercase}}</li>
  <li>{{prix| number:2| currency}}</li>
  <li>Produits par prix décroissants:</li>
  <li ng-repeat="p in produits | orderBy:'-prix'">
          {{p.des}} {{p.prix| currency}}
  </li>
</ul>
```

Filtre perso (1)

- **Création d'un filtre simple: entoure la valeur à afficher d'étoiles**

```
var app = angular.module("demoFiltreApp", []);
app.filter("etoiles", function () {
  return function (value) {
    return ‘**’ + value + ‘**’;
  }
});
```

- **Utilisation:**

```
<li>{{texte| etoiles}}</li>
```

Ajout d’un argument

```
app.filter("etoiles", function () {
  return function (value, motif) {
     if (angular.isUndefined(motif)) {
        motif = '*';
     }
     return motif + motif + value + motif + motif;
  }
});
```

- **Utilisation:**

```
<li>{{texte| etoiles}}</li>
<li>{{texte| etoiles:'+'}}</li>
```

- **un petit texte**
- ++un petit texte++

Filtre dans un ng-repeat

- **hg-repeat n'a pas de boucle « numérique », il n'itère que sur des collections…**

- **donc on crée le filtre:**

```
app.filter('repetition', function () {
                return function (value) {
                    var r = [];
                    for (var i = 0; i < value; i++) {
                        r.push(i);
                    }
                    return r;
                }
            }
);
```

filtre ‘repetition’, usage:

```
<li ng-repeat="i in val| repetition">valeur {{i}}</li>
<li ng-repeat="j in 5| repetition">autre valeur {{j}}</li>
```

- valeur 0
- valeur 1
- valeur 2
- autre valeur 0
- autre valeur 1
- autre valeur 2
- autre valeur 3
- autre valeur 4

Les directives

- **les directives sont les attributs et les balises introduites par angular (par exemple ng-repeat)**

- **on peut créer ses propres directives**

- **c'est là et seulement là qu'on fait des manipulations sur le DOM dans Angular.**

Quelques directives standards

Répétition: ng-repeat

- ...

ng-class

ng-click

Création de nouvelles directives

- **on définit une directive dans un module (qui peut être l'application, ou non)**

- **les modules ont une méthode directive, qui prend deux arguments**

- **le nom de la directive, en camelCase. La directive « maDirective » sera utilisée dans le html sous la forme ma-directive ;**

- **une fonction, qui servira de fabrique pour la directive; cette fonction peut retourner des résultats très divers**

La fonction directive

- **la fonction passée en argument à directive peut retourner:**
  - **un objet, dont nous allons détailler les champs**
  - **une « fonction-link », ce qui est utilisable pour créer rapidement une directive**

Exemple simpliste

```
<!DOCTYPE html>
<html>
    <head>
        <script src="js/bower/angular/angular.min.js"></script>
         <script>
             angular.module('demoSimple', [])
                     .controller('controleur', function ($scope) {
                         $scope.personne = {
                                 nom: ‘Lovelace', prenom: 'Ada'};})
                     .directive('demoPersonne', function () {
                         return {
                             template:
                     '{{personne.prenom}} <b>{{personne.nom}}</b>'
                         };
                     });
        </script>
    </head>
    <body ng-app="demoSimple" ng-controller="controleur">
        <div> <span demo-personne></span></div>
    </body>
</html>
```

Exemple simpliste

```
monModule.directive('demoPersonne', function () {
  return {
    template:
      '{{personne.prenom}} <b>{{personne.nom}}</b>'
  };
});
```

- **la fonction retourne un objet avec un champ template**

```
<span demo-personne></span>
```

- la directive est appliquée au span
- noter les noms différents
- le **contenu** du span est remplacé par la template.

Note sur les attributs

```
<span demo-personne></span>
```

- **N’est pas du html valide.**
- **On peut ajouter « data- » devant les attributs non standard, auquel cas c’est autorisé par HTML**

```
<span data-demo-personne></span>
```

- **Vu par angular, l’attribut s’appelle toujours demo-personne.**

Une barre de progression

- **On va écrire une directive qui sera capable de manipuler le DOM de manière fine**

- **La façon la plus simple c’est de retourner une link-fonction directement**

- **la fonction en question sert normalement à mettre en place des listeners, mais elle est aussi utilisable pour manipuler le dom…**

Une barre de progression…

Exemple

Exemple : on veut créer un contrôle "progress bar". Il contient une div de couleur différente.

Première valeur

30 30

Seconde valeur

70 70

Deux barres contrôlées par la première valeur

Une barre contrôlées par la seconde valeur

http://stackoverflow.com/questions/7190898/progress-bar-with-html-and-css pour le HTML et les CSS

La fonction lien

```
moduleDirectives.directive(
  "qenherProgress",
  function () {
            return function (scope, elt, attrs) {…}
);
```

- la fonction reçoit trois arguments:
  - le scope (pas $scope, scope), qui permet de consulter et modifier les données ;
  - l'élément (objet JQuery, ou plus exactement JQLite), qui permet les manipulations DOM
  - attrs : le tableau des attributs html de l'élément annoté

La fonction lien

```
return function (scope, elt, attrs) {…}
```

```
<div data-qenher-progress data-progress-value="completion"></div>
```

l’objet DOM elt passé à la fonction représentera cet élément

dans le tableau attrs, on trouvera attrs["progress-value"] qui vaudra ‘completion’

Notre barre de progression

- Idée HTML:
  - on crée deux div imbriquées
  - celle à l'intérieur a une taille proportionnelle à la complétion
  - on joue sur les couleurs, etc… avec la feuille de style suivante:

```
.progressbar {
  background-color: black;
  border-radius: 13px;
}

.progressbar > div {
  background-color: orange;
  width: 0%;
  height: 20px;
  border-radius: 10px;
}
```

La directive

```
moduleDirectives.directive(
        "qenherProgress",
        function () {
            return function (scope, elt, attrs) {
                elt.addClass("progressbar")
                var propName = attrs["progressValue"];
                var subDiv = angular.element("<div>");
                elt.append(subDiv);
                scope.$watch(propName, function () {
                    subDiv.css("width", scope[propName] + "%");
                });
            };
        }
);
```

La directive

```
return function (scope, elt, attrs) {

  elt.addClass("progressbar")

  var propName = attrs["progressValue"];

  var subDiv = angular.element("<div>");

  elt.append(subDiv);

  scope.$watch(propName, function () {
    subDiv.css("width", scope[propName] + "%");
  });

};
```

nom de la variable du scope affichée par la barre de progression

ajoute la div interne (la barre)

met à jour l'affichage quand la valeur est modifiée

scope.$watch

- permet d’exécuter une fonction quand la valeur d’une expression change

Le code dans son ensemble

```
var moduleDirectives= angular.module("mesDirectives", []);
var myApp = angular.module("myApp", ['mesDirectives']);
var controller = myApp.controller("myController",
        function ($scope) {
           $scope.completion = 30;
            $scope.val1 = 70;
        }
);
moduleDirectives.directive(
        "qenherProgress",
        function () {
            return function (scope, elt, attrs) {
                elt.addClass("progressbar")
                var propName = attrs["progressValue"];
                var subDiv = angular.element("<div>");
                elt.append(subDiv);
                scope.$watch(propName, function () {
                    subDiv.css("width", scope[propName] + "%");
                });
            };
        }
);
```

Noter l'utilisation de deux modules

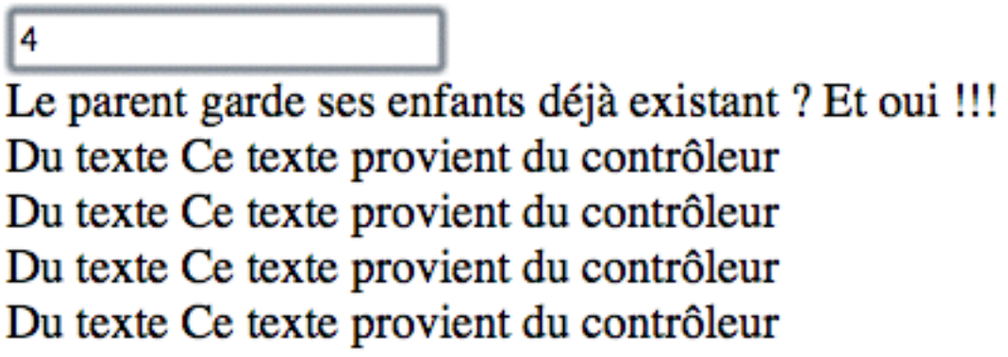
Le html

```
 <body ng-controller="myController">
        <input ng-model="completion" /> {{completion}}
        <div data-qenher-progress data-progress-value="completion"
style="width: 10em;"></div>

</body>
```

- La div est transformée en barre de progression, la valeur de la progression étant donnée par la variable « completion » du scope.

Exemple de directive complexe: répétition d'un élément n fois



- On peut aussi manipuler le contenu de l'élément annoté par la directive

- Ça demande d'écrire une fonction compile

- (je pense qu'on n'aura pas le temps d'en discuter, du coup je propose de revenir sur ces transparents en fin de séance)

Répétition

```
<!DOCTYPE html>
<html lang="fr" ng-app="myApp">
    <head>
        <script src="js/bower/angular/angular.min.js"></script>
         <script src="js/app/ncopies.js"></script>
         <script>
             var myApp = angular.module("myApp", ["MesUtilitaires"]);
             var controlleur = myApp.controller("MonControlleur",
              function ($scope) {
                 $scope.texte = "Ce texte provient du contrôleur";
                 $scope.rep = 10;});
         </script>
    </head>
    <body ng-controller="MonControlleur">
        <input ng-model="rep"/>
        <div>
            Le parent garde ses enfants déjà existant ?
            <div data-n-copies data-nb-items='rep'>
                Du texte {{texte}}
            </div>
       </div>
    </body>
</html>
```

Répétition

```
var utilsModule = angular.module("MesUtilitaires", []);
utilsModule.directive(
  "nCopies",
  function () {
    return {
      transclude: 'element',
      scope: true,
      compile: function (element, attrs, transcludeFn) {
        return function ($scope, $element, $attr) {
          $scope.$watch($attr["nbItems"], function () {
            var n = $scope[$attr["nbItems"]];
            var parent = $element.parent();
            parent.children().remove();
            for (var i = 0; i < n; i++) {
              transcludeFn($scope,function (clone){
                 parent.append(clone);
});});});}}});
```

Tests

- Angular est conçu pour simplifier le test unitaire
- les constructions de javascript sont remplacées par des objets qu'on peut redéfinir (ex. $window à la place de window), et qui sont injectés

- Angular fournit le module ngMock pour simplifier le test.

Angular 2.0

- Casse tout…
- basé sur typescript et non plus javascript
- Beaucoup de code déjà en angular 1.x…

Notes diverses

- Si on nomme l’application, ng-app=« … », il ne faut pas oublier de créer le module correspondant (pb. uniquement quand on fait de petites pages)

Bibliographie

- https://docs.angularjs.org/tutorial
- Adam Freeman, *Pro Angular JS,* Apress, 2014
  - très complet techniquement, avec beaucoup d'exemples
- Pawel Kozlowski & Peter Bacon Darwin, *Mastering Web Application Development with AngularJS,* Packt publishing, 2013
  - orienté problème, très bien fait aussi
- Brad Green & Shyam Seshadri, *AngularJS*, O'Reilly 2013
  - précis et concentré, plus utilisable comme référence que les précédents

Backbone JS

- Concurrent d'angular
- ne repose pas sur une modification du html
- laisse plus de liberté au programmeur
- intégration plus facile avec l'existant
- moins concis, du coup