



## Présentation de la formation

AngularJS est un Framework Javascript qui est devenue, en peu de temps un outil incontournable des développeurs front. Il surprend par sa simplicité à créer des applications riches, structurées et évolutives. Développé et soutenu par Google, il est distribué sous licence open-source et est en téléchargement libre.

Au cours de cette formation, vous découvrirez le fonctionnement du Framework AngularJS, ses concepts, la création de modules, contrôleurs, vues, l'utilisation du routage, la mise en place de services et tous les outils utiles à la réalisation de vos applications web.

La pratique étant aussi importante que la théorie, des travaux dirigés seront régulièrement pratiqués.

## Public

- Développeur web

## Pré-requis de la formation

- Connaissance des technologies web
- Connaître le langage Javascript (notions avancées)

## Objectif de la formation

- Comprendre l'intérêt d'un Framework dans une application moderne
- Acquérir les réflexes de base et devenir autonome
- Développer un prototype d'application single page

# HTML5

## Balises et sémantique

Le HTML5 est une nouvelle version du standard HTML qui apporte une plus grande sémantique ainsi qu'une meilleure lecture et accessibilité.

Le HTML5 permet de mieux dissocier les parties d'une page web avec plus de logique.

## Exemple de balises

Balise	Usage
<header>	Zone représentant l'entête du document
<hgroup>	Permet de grouper un titre avec ses sous-titres (Déprécié)
<nav>	Représente la navigation
<section>	Division d'une page représenté par une section
<article>	Représente un article
<figure>, <figcaption>	Associe une légende à un illustration
<input>	Prend en charge de nouveaux types : <i>date, time, email, url, tél, ...</i>

# Architecture REST

## Qu'est-ce qu'une architecture REST ?

- REST est un style d'architecture facilitant la communication entre machines
- REST est utilisé dans le cadre du web permet de créer des applications et services
- REST n'est pas une technologie à part entière, mais plutôt une méthodologie et un ensemble de conventions à respecter
- REST profite des spécifications originelles du protocole HTTP

## Historique

REST est l'acronyme de « **R**epresentational **S**tate **T**ransfert » imaginé/créé par Roy Thomas Fielding en 2000.

## REST combine

- le protocole HTTP pour gérer les ressources
- la norme URI pour identifier les ressources
- le type mime pour représenter les ressources (*Content-Type: application/json*)

## Exemple d'URI

- GET /blog/articles (Réponse : *200 OK* ou *404 Not found*)
- POST /blog/article/add (Réponse : *201 Created*)

## Table de correspondance entre CRUD et HTTP

SYSTEME (CRUD)	Usage	SYSTEME (Méthodes HTTP)
CREATE	Ajouter/créer une ressource	POST
READ	Récupérer/Lire une ressource ou une liste de ressources	GET
UPDATE	Modifier/Mettre à jour une ressource	PUT
DELETE	Supprimer une ou plusieurs ressources	DELETE

# Introduction à AngularJS

## Présentation du framework AngularJS

AngularJS fait partie de la nouvelle vague de Frameworks Javascript. C'est une innovation coté client vous permettant de développer des applications web riche.

Il impose une **architecture** qui vous assure d'avoir du code bien **structuré** et **réutilisable**.

Soutenu par Google, la popularité d'AngularJS ne cesse de grimper nous offrant ainsi, une grande communauté pour échanger sur des forums de discussion et partager du code.

## Historique

AngularJS a été créé en 2009 par **Miško Hevery** alors qu'il travaillait sur un projet pour Google, il se rendit compte des limites d'un langage peu structuré et réutilisable. Il eu donc l'idée, après avoir testé son implémentation de créer un framework. Trouvant le résultat satisfaisant il décida de distribuer son code sous licence open-source.

# MVVM

## MVC : Modèle Vue Contrôleur

Le patron de conception MVC est un design pattern destiné à répondre aux besoins des applications interactives en séparant les problématiques liées aux différents composants au sein de leurs architectures respectives.

- Model**      Modèle de données, statut des données
- View**        Présentation des données, interface utilisateur (output)
- Controller**   Gestion des actions, logique de contrôle (input)

## Pourquoi MVVM

La forte dépendance, imbrication du DOM et du HTML rend difficile la séparation de la logique avec la vue.

## Meilleure adaptation au développement Front : MVVM

- Model**        Modèle de données, statut des données
- View**        Présentation des données, interface utilisateur (output)
- ViewModel**   Lien entre la vue et le modèle (binding)

# Single Page Application

## Définition

Une **Single Page Application (SPA)** est une application web accessible via une page web unique, qui consiste à charger toute la logique et visuel sur une seule page et permet de fluidifier l'expérience utilisateur.

Les données complémentaires seront alors chargées dynamiquement, par l'application en AJAX généralement fourni via une API REST.

## Ce qu'il ne faut pas faire

- Utiliser JQuery dans tous les cas
- Utiliser une fonction – un plugin JQuery

## Ce qu'il faut faire

- Créer une application modulaire
- Utiliser les bons patrons de conceptions
- Garder une relation client serveur saine

## AngularJS et les autres

### Y-a-t-il d'autres Frameworks Javascript ?

Oui, il existe d'autres Frameworks Javascript, équivalents ou complémentaires :

- Backbone
- Ember
- Express
- React
- Sails





# AngularJS et les autres

## L'écosystème des outils pour Javascript

Avec l'arrivée de **NodeJS**, une multitude d'outils se sont développés pour rendre la vie des développeurs plus agréable.

Il est désormais possible de piloter un projet codé en javascript et bénéficier d'outils de tests, de gestionnaire de paquet et d'automatiser des tâches récurrentes.

Voici une liste de quelque outils parmi les plus utilisés :

- NodeJS (Environnement d'exécution événementiel multi-plateforme utilisant Javascript)
- Npm (Node Package Manager) utilitaire de téléchargement de modules NodeJS
- Grunt (Outil d'automatisation des tâches)
- Bower (Gestionnaire de paquet pour le web)
- Yeoman (Générateur de site web utilisant Grunt et Bower)



# Installer AngularJS

## Depuis le site de l'éditeur

Télécharger AngularJS directement via le site : <https://angularjs.org>

Il existe plusieurs types de fichiers :

- Minified (compressé pour la production)
- Uncompressed (Code brut pour le développement)

Choisissez le fichier qui vous convient mais dans tous les cas ils auront le même comportement.

Faites attention au type de version ! La **legacy** représente une version stable, tandis que **latest** représente une version en cours de développement et débogage.

Exemple d'intégration dans du code HTML :

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>AngularJS</title>
  </head>
  <body>
    <script src="js/angular.min.js"></script>
  </body>
</html>
```

# Installer AngularJS

## Depuis un gestionnaire de paquet

Un gestionnaire de paquet permet d'installer des librairies et ses dépendances via un utilitaire en ligne de commande.

L'intérêt est de pouvoir choisir la version que l'on souhaite utiliser, et ainsi de pouvoir mettre aisément à jour ses librairies avec leurs dépendances. De cette manière il ne sera pas nécessaire de modifier du code ou d'aller télécharger les librairies utiles à l'application.

L'installation proposée ici se fera sur une distribution Linux (xubuntu), mais il est possible de le faire sur d'autres systèmes d'exploitation comme Windows et Mac de façon plus ou moins similaire.

### Liste des étapes à réaliser :

1. Installation du gestionnaire de version **Git**
2. Installation de **NodeJS**
3. Installation du gestionnaire de paquet **Npm**
4. Installation du gestionnaire de version **Bower**
5. Installation d'**AngularJS**

Les lignes de commandes sont simples et nous utiliserons **sudo** dans cet exemple qui permet d'avoir des droits plus avancés pour installer des paquets sous Linux.

### Mise à jour du gestionnaire de paquets (APT) Linux :

```
$ sudo apt-get update && sudo apt-get upgrade
```

### Installation de Git :

```
$ sudo apt-get install git
```

### Installation de NodeJS :

```
$ sudo apt-get install nodejs
```

# Installer AngularJS

L'installation terminée, il faut créer un alias **node** qui pointera vers **nodejs**. La raison de cette alias est de satisfaire les autres applications utilisant le nom **node** pour fonctionner.

```
$ sudo ln -s /usr/bin/nodejs /usr/bin/node
```

Pour vérifier que tout s'est bien déroulé, il faut taper la commande suivante :

```
$ node --version
```

## Installation de npm :

```
$ sudo apt-get install npm  
$ npm --version
```

## Installation de Bower avec npm :

```
$ sudo npm install bower -g  
$ bower --version
```

## Installation d'AngularJS :

```
# création du répertoire ou sera placé le projet :  
$ mkdir monprojet  
$ cd monprojet  
  
# installation d'AngularJS :  
$ bower install angular
```

L'installation via bower, créera automatiquement un répertoire nommé **bower\_components** dans lequel se trouvera les fichiers d'AngularJS.

# Installer AngularJS

Exemple d'intégration dans du code HTML (index.html) :

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>AngularJS</title>
  </head>
  <body>
    <script src="bower_components/angular/angular.js"></script>
  </body>
</html>
```

A l'avenir il sera facile de mettre à jour ou d'installer d'autres librairies directement avec Bower :

```
$ bower update
```

# Ses premiers pas

## Créer sa première application

Les premières impressions sont souvent les meilleures :

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="utf-8">
    <title>AngularJS</title>
  </head>
  <body ng-app>

    <h1>Bonjour {{ name }}</h1>
    <input type="text" ng-model="name">

    <script src="bower_components/angular/angular.js"></script>
  </body>
</html>
```

# Déclaratif & Impératif

## Programmation impérative

- les instructions conditionnent le fonctionnement du programme
- chaque état est prédéfinie pour interagir avec le programme
- la structure du langage permet d'exécuter dynamiquement des commandes

## Programmation déclarative

- Décrit un état statique
- HTML est déclaratif, parce qu'il ne peut pas modifier son état

## AngularJS et la programmation déclarative

- Etendre le HTML

### Afficher la valeur d'un champ de formulaire texte :

- avec JQuery

```
<h1 id="name">Bonjour <span></span></h1>
<input type="text" id="textfield">

$("#textfield").on("keyup", function(e) {
    $("#name span").text($(this).val());
});
```

- avec AngularJS

```
<h1>Bonjour {{ name }}</h1>
<input type="text" ng-model="name">
```

# Notion de directive

## Qu'est-ce qu'une directive ?

- une directive permet de connecter et appliquer un comportement à une section de la page
- les directives se branchent au document via des balises, des classes et ou attributs HTML



## Exemple :

```
<!-- Connexion via une balise HTML -->  
<ma-directive></ma-directive>  
  
<!-- Connexion via attribut -->  
<input type="text" ma-directive="name">  
  
<!-- Connexion via une classe -->  
<div class="ma-directive"></div>
```



# Data Binding

## Le principe

Le data binding avec AngularJS est la synchronisation automatique des données entre le modèle et la vue.

Il permet d'assigner une valeur à un modèle de données, d'en modifier cette valeur et en d'en faire l'affichage.

## En pratique

Le data binding est la relation entre les données et la vue par l'intermédiaire du modèle

```
<!-- assignation du modèle portant le label : nomDuModel -->  
<input type="text" ng-model="nomDuModel">  
  
<!-- affichage de la valeur du modèle -->  
<h1>Bonjour {{ nomDuModel }}</h1>  
  
<!-- Le modèle prendra la valeur entrée dans le champ input par l'utilisateur -->
```

## Fonctionnement

- le data binding utilise le principe du **Dirty checking**
- le **Dirty checking** consiste à surveiller l'état des données d'une zone qui lui a été attribué
- une boucle permet de vérifier tout changement au sein des données
- `$digest()` résume l'état des données et procède à la mise à jour en cas de changement

## Performance (one-time binding)

Depuis la version **1.3** d'AngularJS, il est possible de procéder à l'évaluation des données qu'une seule fois, stoppant ainsi la surveillance pour un gain de performance.

La syntaxe permettant d'annuler la surveillance est « `::` » placé devant le nom du modèle dans la vue.

```
<!-- assignation du modèle portant le label : nomDuModel -->  
<input type="text" ng-model="nomDuModel">  
  
<!-- les deux points « :: » annuleront la surveillance -->  
<h1>Bonjour {{ ::nomDuModel }}</h1>
```

# Les expressions

## Du javascript dans les templates ?

Les expressions dans AngularJS sont des portions de code ressemblant à du code javascript.

```
<!-- addition entre deux nombres -->
<div>
  <input type="number" ng-model="a"> +
  <input type="number" ng-model="b"> = {{ a + b }}
</div>

<!-- remplacement des espaces par des tirets -->
<div>
  <input type="text" ng-model="slug" ng-init="slug='un titre avec espace'">
  {{ d = slug.split(' '); d.join('-') }}
</div>

<!-- autre exemple ... -->
<div>
  <input type="text" ng-model="chars" ng-init="chars='AngularJS'">
  Nombre de caractères : {{ chars.length }}
</div>
```

Cette ressemblance avec Javascript comporte quelques différences :

VS	
ANGULARJS	JAVASCRIPT
Evaluées dans une partie de l'objet <b>window</b>	Evaluées et valides dans tout l'objet <b>window</b>
L'évaluation de propriétés non définies ne génère pas d'erreurs ( <b>undefined</b> et <b>null</b> )	Evaluer des propriétés non-définies génère des erreurs de type <b>ReferenceError</b> ou <b>TypeError</b>
Il n'est pas permis d'utiliser les boucles, structures conditionnelles, ...	Les structures de contrôles peuvent être utilisées
Des filtres peuvent être utilisés pour formater les données en sortie	Des fonctions peuvent être utilisés pour formater les données en sortie

# Le templating

## Inclusion de templates

La vue peut être subdivisée en utilisant les templates qui seront inclus dynamiquement.

## Les plus

- éviter les répétitions
- chargement dynamique des templates
- maintenabilité et réutilisabilité du code

## Exemple

```
<html>
  <body ng-app>

    <ng-include src="'tpl.html'"></ng-include>
    <ng-include src="'tpl.html'"></ng-include>

    <script type="text/ng-template" id="tpl.html">
      <h1>Ma template</h1>
      <div>Contenu de ma template</div>
    </script>

    <script src="js/angular/angular.min.js"></script>
  </body>
</html>
```

## Fonctionnement du chargement de templates

1. tente de charger une template dans le cache d'AngularJS
2. tente de charger une template par une balise script identifié par un attribut id
3. tente de charger une template via une requête ajax

# Modules

## Notion de module

On crée des applications modulaires pour séparer, organiser et maintenir plus facilement le code.

### Avant

- `$('#elementID')`
- code spaghetti
- difficilement réutilisable
- difficilement testable

### Eventuellement

- un fichier par plugin
- utilisation de bibliothèques interdépendantes

# Modules

## Modularisation fonctionnelle

**Important pour les applications contenant beaucoup de fonctionnalités**

- un module par fonctionnalité (blog, user, cms, catalogue produit)
- chaque module fonctionnel peut être découpé en module technique

## Modularisation technique

**Segmentation en fonction des besoins techniques**

- application(s)
- contrôleurs
- templates
- includes
- filtres
- services
- directives
- configuration
- animations

# Modules – Exemples d'arborescences

## Modularisation technique

```
app/  
  config/  
  resources/  
  src/  
    controllers/  
    directives/  
    filters/  
    services/  
  vendor/  
  views/
```

## Modularisation fonctionnelle

```
app/  
  module1/  
    config/  
    resources/  
    src/  
      controllers/  
      directives/  
      filters/  
      services/  
  module2/  
    config/  
    resources/  
    src/  
      controllers/  
      directives/  
      filters/  
      Services/  
  default/  
    config/  
    resources/  
    src/  
      controllers/  
      directives/  
      filters/  
      services/  
  vendor/  
  views/
```

# Modules

## Principe de l'injection de dépendance

L'injection de dépendance permet de charger certaines parties de l'application à la demande.

Un module ayant besoin d'autres fonctionnalités peut se voir injecter, interchanger les composants d'un autre module.

## Avantage

- possibilité d'utiliser des modules ensemble ou de manière indépendante
- réduit le fort couplage entre chaque composants
- remplacement de « vrai » objets par des mocks
- testable

# Modules

## Utilisation de l'injection de dépendance

Exemple d'injection de dépendance avec un module :

```
// le module ngRoute (module de routage) a été injecté à monModule  
angular.module('monModule', ['ngRoute']);
```

Exemple d'injection de dépendance avec un contrôleur :

```
// les services $scope et monService ont été injectés au contrôleur  
angular.controller('mainController', function ($scope, monService) {  
    // faire quelque chose ici ...  
});
```



# Contrôleurs et Scopes

## Rôle d'un contrôleur

Le contrôleur permet de prendre en charge une zone de la page HTML et de lui appliquer un comportement via une fonction Javascript.

## Créer et initialiser son premier contrôleur

```
<html>
  <body ng-app="monModule">

    <div ng-controller="monPremierController"></div>

    <script src="js/angular/angular.min.js"></script>
    <script>
      var app = angular.module('monModule', []);
      app.controller('monPremierController', function () {
        console.log('Contrôleur 1');
      });
    </script>
  </body>
</html>
```

## Associer un contrôleur à un template

```
<html>
  <body ng-app="monModule">

    <div ng-controller="monPremierController">
      Bonjour {{ prenom }}
    </div>

    <script src="js/angular/angular.min.js"></script>
    <script>
      var app = angular.module('monModule', []);
      app.controller('monPremierController', function ($scope) {
        $scope.prenom = 'toto';
      });
    </script>
  </body>
</html>
```

# Contrôleurs et Scopes

## Imbrication des contrôleurs

Les contrôleurs peuvent être imbriqués.

```
<html>
  <body ng-app="monModule">

    <div ng-controller="controller1">
      Contrôlé par controller1
      <div ng-controller="controller2">
        Contrôlé par controller2 et controller1
      </div>
    </div>

    <script src="js/angular/angular.min.js"></script>
    <script>

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

      app.controller('controller1', function () {
        console.log('Contrôleur 1');
      });

      app.controller('controller2', function () {
        console.log('Contrôleur 2');
      });

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

# Contrôleurs et Scopes

## Concept de Scope

- le scope représente un espace dans lequel une action sera exécuté
- le contrôleur accède au service **\$scope** via l'injection de dépendance
- les valeurs initialisées dans le scope sont directement accessibles dans la vue

## Le scope et le partage de données

```
<div ng-controller="catalogue">
  <h1>{{ boutique }}</h1>
  <ul>
    <li>{{ product.name }}</li>
    <li>{{ product.description }}</li>
  </ul>
</div>

<script>
  app.controller('catalogue', function ($scope) {
    $scope.boutique = 'AngularJS store';
    $scope.product = {
      name: 'NG-STAR',
      description: 'A JS Framework'
    };
  });
</script>
```

## Le scope et le partage de méthodes

```
<div ng-controller="afficheNom">
  <h1>Hello, {{ toUpper('toto') }}</h1>
</div>

<script>
  app.controller('afficheNom', function ($scope) {
    $scope.toUpper = function (str) {
      if (!str) return '';
      return str.toUpperCase();
    };
  });
</script>
```

# Contrôleurs et Scopes

## Scope - Héritage

- un scope enfant hérite des propriétés du scope parent
- seuls les objets sont partagés et peuvent être propagées en cas de modification
- pour être sûr que les propriétés de **\$scope** soient partagées et mises à jour dans tous les scopes qui la référencent, utilisez des objets

```
<div ng-controller="parentCtrl">
  <input type="text" ng-model="prenom">
  <input type="text" ng-model="user.prenom">

  <div ng-controller="childCtrl">
    <input type="text" ng-model="prenom">
    <input type="text" ng-model="user.prenom">
  </div>
</div>

<script>
  app.controller('parentCtrl', function ($scope) {
    $scope.prenom = 'foo';
    $scope.user = { prenom: 'bar' };
  });

  app.controller('childCtrl', function () {});
</script>
```

## Exemple avec la directive ng-repeat

```
<div ng-controller="parentCtrl">
  {{ prenom }}
  <ul>
    <li ng-repeat="item in items">
      <input ng-model="prenom">
    </li>
  </ul>
</div>

<script>
  app.controller('parentCtrl', function ($scope) {
    $scope.prenom = 'foo';
    $scope.items = [1, 2, 3];
  });
</script>
```

# Contrôleurs et Scopes

## Contrôleur et alias

L'utilisation de contrôleur avec la directive ng-controller peut se faire via un alias.

```
<div ng-controller="nomController as nomAlias"></div>
```

## Utilisation d'un alias

```
<div ng-controller="parentCtrl as parent">
  <input type="text" ng-model="parent.prenom"> {{ parent.prenom }}

  <div ng-controller="childCtrl as child">
    <input type="text" ng-model="child.prenom">
      {{ child.prenom }} - {{ parent.prenom }}
    </div>
  </div>

<script>
  app.controller('parentCtrl', function ($scope) {
    $scope.parent.prenom = 'foo';
  });

  app.controller('childCtrl', function ($scope) {
    $scope.child.prenom = 'bar';
  });
</script>
```

# Contrôleurs - Scopes & Événement

## Déclencher une action

Pour interagir avec l'utilisateur il est possible d'appeler une fonction du scope d'exécution via des directives événementielles déclaratives (*ng-click*, *ng-keyup*, *ng-mouseup*, ...)

```
<div ng-controller="monCtrl">
  <a href="" ng-click="action()">Like</a>
</div>

<script>
  app.controller('monCtrl', function ($scope) {
    $scope.action = function () {
      alert('Hello');
    };
  });
</script>
```

# Contrôleurs – Scopes & Événement

## Événements

AngularJS propose deux méthodes attachées au scope pour propager un événement et une méthode pour les écouter, intercepter et lancer une action.

- `$broadcast` (Diffusion)
- `$emit` (Émission)
- `$on` (Écouter)

## Écoute

- `$scope.$on`
- écouteur et lanceur d'événements

## Émission

- `$scope.$emit`
- propage un événement par ébullition vers les ancêtres

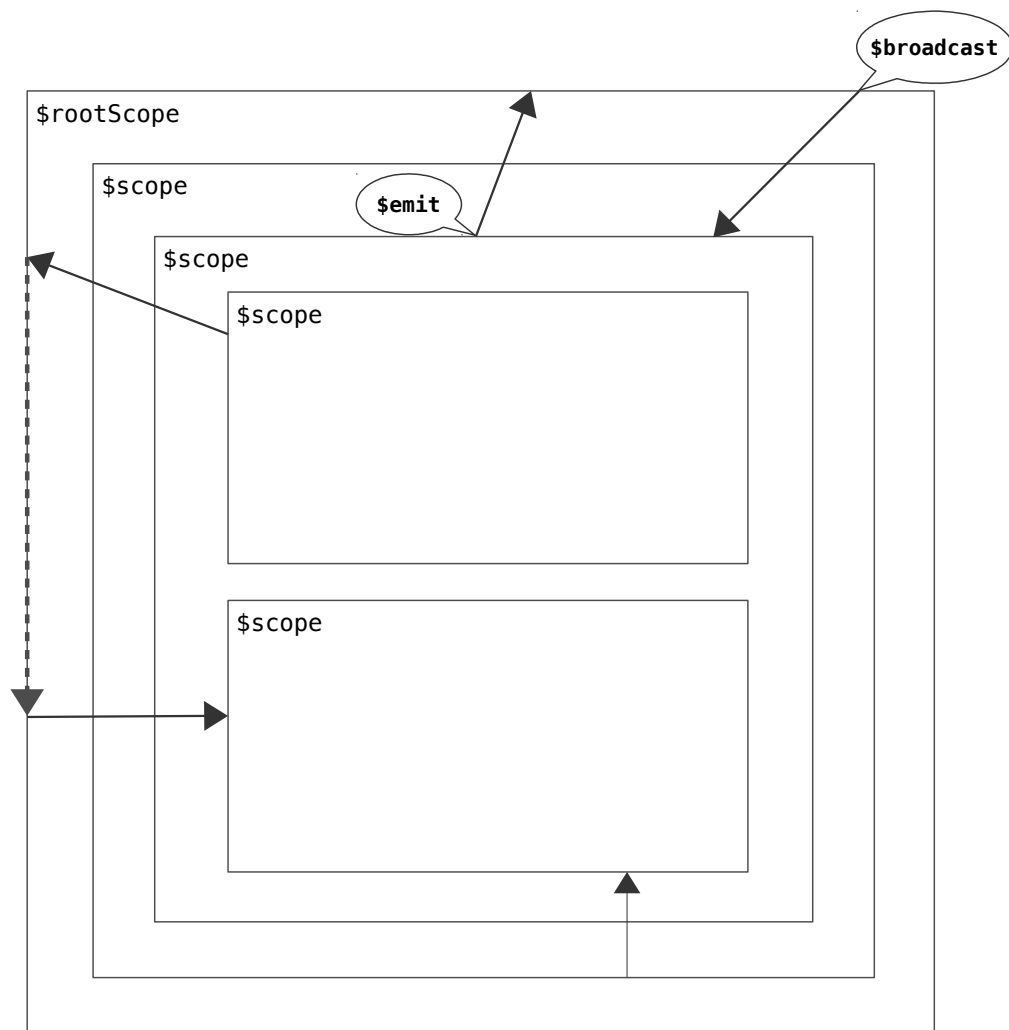
## Diffusion

- `$scope.$broadcast`
- propage un événement vers les enfants
- performance : préférez le `$rootScope` pour un événement global

# Contrôleurs – Scopes & Événement

## Événements

Propagation d'événements





# Contrôleurs – Scopes & Événement

## Événements

Utilisation des événements avec AngularJS

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

  // PARENT
  app.controller('parentCtrl', function ($scope) {
    $scope.$broadcast('evt', 'Message aux descendants');

    $scope.$on('evt', function (evt, msg) {
      console.log('Reçu dans parent : ' + msg);
    });
  });

  // CHILD
  app.controller('childCtrl', function ($scope) {
    $scope.$on('evt', function (evt, msg) {
      console.log('Reçu dans child : ' + msg);
    });
  });

  // GRANDSON
  app.controller('childCtrl', function ($scope) {
    $scope.$on('evt', function (evt, msg) {
      console.log('Reçu dans grandson : ' + msg);
    });
  });

</script>
```

# Contrôleurs et Scopes

## Surveiller l'état des données

AngularJS met à disposition une surveillance des données grâce au **\$scope** et à sa méthode **\$watch**

```
<div ng-controller="monCtrl">
  <input ng-model="pays">
</div>

<script>
  app.controller('monCtrl', function ($scope) {
    $scope.$watch('pays', function (newValue, oldValue) {
      console.log(newValue);
    });
  });
</script>
```

## Surveiller une expression

```
<div ng-controller="monCtrl">
  <input ng-model="pays">
</div>

<script>
  app.controller('monCtrl', function ($scope) {
    $scope.$watch("pays === 'France'", function (newValue, oldValue) {
      if (newValue) console.log(newValue);
      if (oldValue && !newValue) console.log('Bye');
    });
  });
</script>
```

# Contrôleurs et Scopes

## Watch et valeur

```
<div ng-controller="monCtrl">
  <input ng-model="user.name">
</div>

<script>
  app.controller('monCtrl', function ($scope) {
    $scope.$watch('user', function (newValue, oldValue) {
      if (newValue) console.log('Ref: ', user.name);
    });

    $scope.$watch('user', function (newValue, oldValue) {
      if (newValue) console.log('Valeur: ', newValue.name);
    }, true);
  });
</script>
```

## Watchgroup (introduit en 1.3)

Depuis la version 1.3 il devient possible de surveiller une collection

```
$scope.$watchGroup([item1, item2, item3], function (newItem, oldItem) {
  // exécutée dès qu'un item change
});
```

# Les Filtres

## Formatage

- une entrée + des paramètres
- une sortie

## Transformation, suppression, trie

- Array
- String
- Date
- ...

# Les Filtres

## Définition

Un filtre permet de traiter/trier des données selon des critères définis.

## Syntaxe

- | (pipe UNIX)
- une expression angular

```
<div>{{ expression | nomDuFiltre }}</div>
```

## Chaînage

- Il est possible de chaîner les filtres à l'aide du pipe « | »
- attention aux types en entrée / sortie

```
<div>{{ expression | filtreA | filtreB | filtreC }}</div>
```

## Paramètres

- il est possible de passer des paramètres à un filtre
- il faut utiliser « : » comme séparateur

```
<div>{{ expression | filtre:param1:params2 }}</div>
```

# Les Filtres

## Utilisation hors template

- utiliser un filtre hors d'un template peut s'avérer utile

```
.controller('monCtrl', function ($scope, $filter) {  
    $scope.txt = $filter('uppercase')('ce text sera en majuscule');  
});
```

## Liste des filtres fournie par défaut

- currency
- date
- filter
- json
- limitTo
- lowercase
- number
- orderBy
- uppercase

# Les Filtres

## Création d'un filtre personnalisé

```
angular.module('monModule', [])  
  
.filter('truncate', function () {  
    return function (input, length) {  
        length = length || input.length;  
        return input.substr(0, length) + '...';  
    };  
});
```

## Utilisation d'un filtre personnalisé

```
<p>{{ 'Chargement en cours' | truncate:10 }}</p>  
  
<!-- sortie après filtrage : Chargement... -->
```

## Précaution sur l'utilisation des filtres

Un filtre étant en générale appelé sur une expression, il sera exécuté à chaque changement de cette expression.

Il est préférable d'utiliser les filtres avec parcimonie.

# Routage

## Notion de routage

Le routage est similaire aux applications web client / serveur traditionnelles.

- le routeur fait la correspondance d'une route à un contrôleur
- le contrôleur met en relation les données avec la vue

## Installation du module Route

Le module route n'est plus disponible par défaut dans AngularJS depuis la version 1.2 et son intégration se fait maintenant par le biais d'une dépendance.

Le module route est téléchargeable sur le site [angularjs.org](http://angularjs.org), ou peut-être chargé avec bower :

```
$ bower install angular-route
```

Le téléchargement terminée, il faut l'inclure dans la page HTML :

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

Injection de la dépendance dans angular :

```
angular.module('monModule', ['ngRoute']);
```



# Routage

## Configuration des routes

```
angular.module('monModule', ['ngRoute'])

.config(function ($routeProvider) {

    $routeProvider.when('/', {
        templateUrl: 'views/accueil.html',
        controller: 'accueilController'
    })

    .when('/page', {
        templateUrl: 'views/une_page.html',
        controller: 'unePageController'
    });

});
```

## Format de routes

Le protocole HTTP permet une gestion d'ancrage avec le symbole « # » en fin d'url. Tout ce qu'il y a après ce dièse est appelé un fragment et AngularJS, l'utilise à son avantage sans que cela génère un rafraichissement de la page.

### Exemples de routes possible via un navigateur

Route	Depuis un serveur	Depuis un fichier
/	<i>votresite.com/#/</i>	<i>file:///chemin/fichier/index.html#/</i>
/page	<i>votresite.com/#/page</i>	<i>file:///chemin/fichier/index.html#/page</i>

# Routage

## Routage, layout et template

Des templates sont définies dans la configuration des routes, mais où les intégrer dans la page ?

- créer les templates dans un répertoire views (*home.html*, ...)
- placer la directive *ng-view* comme dans l'illustration



# Routage

## Routage, layout et template

Exemple d'un layout au format HTML

```
<!DOCTYPE html>
<html>
  <body>
    <header>
      <!-- exemple d'inclusion -->
      <ng-include src="'views/nav.html'"></ng-include>
    </header>

    <!-- ici, intégration dynamique des vues par angular -->
    <ng-view></ng-view>

    <footer>
      <p>©2014 – monsite.com</p>
    </footer>
  </body>
</html>
```

Utilisation de la directive comme nom de balise :

```
<ng-view></ng-view>
```

Utilisation de la directive avec un attribut :

```
<div ng-view></div>
```

# Routage

## Déclaration d'un template

les templates peuvent être déclarés de deux manières différentes

1. chargé depuis une url
2. déclaré sous forme de chaîne de caractères

il n'est pas possible d'utiliser les deux façons simultanément

```
angular.module('monModule', ['ngRoute'])

.config(function ($routeProvider) {

    $routeProvider.when('/', {
        templateUrl: 'views/accueil.html',
        controller: 'accueilController'
    })

    .when('/sayhello', {
        template: '<h1>Hello</h1>',
        controller: 'sayHelloController'
    });

});
```

## Déclaration d'un contrôleur

AngularJS donne le choix de déclarer un contrôleur pour chaque route de deux façons

1. en déclarant le contrôleur par son nom après avoir été enregistré au-près du module
2. en définissant une fonction directement en valeur

```
angular.module('monModule', ['ngRoute'])
.config(function ($routeProvider) {

    $routeProvider.when('/', {
        templateUrl: 'views/accueil.html',
        controller: 'accueilController'
    })
    .when('/sayhello', {
        template: '<h1>Hello {{ prenom }}</h1>',
        controller: function ($scope) { $scope.prenom = 'toto'; }
    });

});
```

# Routage

## Route par défaut

Définition d'une route par défaut dans le cas où la route demandée serait inexistante

```
angular.module('monModule', ['ngRoute'])

.config(function ($routeProvider) {

  $routeProvider.when('/', {
    templateUrl: 'views/accueil.html',
    controller: 'accueilController'
  })

  .when('/sayhello', {
    template: '<h1>Hello {{ prenom }}</h1>',
    controller: function ($scope) { $scope.prenom = 'toto'; }
  })

  <!-- Route par défaut avec « redirectTo » -->
  .otherwise({
    redirectTo: '/'
  });

});
```

# Routage

## Paramètres de route

Dans le cas d'une route possédant des paramètres variables, AngularJS fournit des paramètres, des variables qu'il faut placer dans l'url à l'endroit attendu.

**Exemple d'url** : *website.com/#/product/67*

```
angular.module('monModule', ['ngRoute'])

.config(function ($routeProvider) {

    $routeProvider.when('/product/:id', {
        templateUrl: 'views/product.html',
        controller: 'productController'
    });

});
```

## Options de paramètres

- */product/:id?* (signifie que le paramètre n'est pas obligatoire)
- */product/:id\** (paramètre représentant un groupe de caractères formant un fragment d'url)

## Exemple de correspondance de routes

Configuration de routes	Usage navigateur	Résultat
/product/:id	/product/14	id == 14
/todo/items/:id?	/todo/items/23	Id == 23 ou id == undefined
/api/v1/:param*	/api/v1/key/37HD9	param == key/37HD9
/color/:color/to/:code*Vedit	/color/brown/to/with/slashes/edit	color == brown, code == with/slashes

# Routage

## Récupération de paramètres dans un contrôleur

```
// exemple : /product/:id == /product/8  
  
angular.module('monModule', ['ngRoute'])  
  
.controller(function ($routeParams) {  
    console.log($routeParams.id);  
});
```

# Routage

## Mode de routage

L'utilisation d'une application « Single Page » n'est pas sans conséquence

- référencement plus difficile pour les moteurs de recherche
- une url plus complexe à retenir pour les utilisateurs avec le « # »

## Le Hashbang (!)

- est un hack
- il sert à tromper le navigateur pour lui faire croire qu'il n'a pas changé de page
- il permet d'informer les moteurs de recherche que l'application est en Ajax

```
// exemple : http://monsite.com/#!/products/promo

angular.module('monModule', ['ngRoute'])

// configuration du préfix « ! »
.config(function ($locationProvider) {
  $locationProvider.hashPrefix('!');
});
```



# Routage

## Le passage vers les URLs HTML5

Le HTML5 offre une multitude de fonctionnalités et parmi elles, notre attention se portera sur API History.

## API History

- création d'une application Ajax avec des urls pour chaque état
- plus besoin de hashbang

## HTML5 mode

```
// exemple : http://votresite.com/products/promo

angular.module('monModule', ['ngRoute'])

.config(function ($locationProvider) {

    $locationProvider.html5mode(true);

});
```

## Conséquences HTML5

- URLs correctement définies (*user friendly*)
- plus lisible pour les robots des moteurs de recherches et l'humain
- rétro compatible avec les navigateurs plus anciens
- le serveur devra toujours rediriger vers le fichier index.html

# Routage

## Les événements de routes

AngularJS donne la possibilité d'une part de gérer les événements de routes et d'autre part d'intercepter ces derniers pour interagir à un moment donné dans le routage

## Liste des événements broadcastés (diffusés)

Événement	Description
<code>\$routeChangeStart</code>	Diffusé avant que la route change
<code>\$routeChangeSuccess</code>	Diffusé après que la route ai été résolu
<code>\$routeChangeError</code>	Diffusé si l'une des promesses a été rejeté
<code>\$routeChangeUpdate</code>	Si la propriété <b><i>reloadOnSearch</i></b> a été mise à <b><i>false</i></b> alors l'instance du contrôleur courant sera réutilisée

## Utilisation des événements

```
$rootScope.$on('$routeChangeStart', function (event, next, current) {  
    // code  
});
```

## Exemple

```
angular.module('monModule', [])  
  
.config(function ($routeProvider) { // .when('/login') ... })  
.run(function ($rootScope, $location, auth) {  
    $rootScope.$on('$routeChangeSuccess', function (event, next, current) {  
        if (!auth.isConnected()) {  
            if (next.templateUrl !== 'views/login.html') {  
                event.preventDefault();  
                $location.path('/login');  
            }  
        }  
    });  
});
```

# Routage

## Résolution d'une route avant instanciatoon du contrôleur

Il peut être intéressant de pouvoir effectuer un traitement, initialiser des paramètres avant l'instanciation d'un contrôleur avec l'utilisation d'une route.

Avec AngularJS il est possible de rajouter une troisième propriété portant le nom de : **resolve**.

- la valeur de la propriété **resolve** est un objet
- l'objet **resolve** peut avoir plusieurs propriétés
- les noms des propriétés de l'objet **resolve** sont laissés au choix du développeur
- les valeurs de l'objet **resolve** doivent être le nom d'un service ou une fonction
- les propriétés résolus sont injectables dans le contrôleur

```
angular.module('monModule', ['ngRoute'])

.config(function ($routeProvider) {

    $routeProvider.when('/', {
        templateUrl: 'views/main.html',
        controller: 'mainController',
        resolve: {
            keyA: 'unService',
            keyB: function () {
                return 'une valeur';
            }
        }
    });

})

.controller('mainController', function ($scope, keyA, keyB) {
    // code
});
```

# Routage

## Resolve et promesse

- une promesse peut être une propriété de l'objet resolve
- l'instanciation du contrôleur dépendra de la résolution cette promesse
- en cas de non résolution le contrôleur n'est pas instancié et lancera un événement de type `$routeChangeError`

# Formulaires

## Création de formulaires

Avec la venue du HTML5, AngularJS vient faciliter et étendre la gestion des formulaires

## Les attributs nécessaires

- ajouter l'attribut **name** sur tous les champs y compris la balise form
- ajouter la directive **ng-model** sur tous les champs pour lesquels vous souhaitez récupérer une valeur

## Exemple

```
<form name="monForm" ng-submit="submitForm();">
  <div>
    <label for="username">Identifiant</label>
    <input type="text" name="username" id="username" ng-model="auth.username">
  </div>
  <div>
    <label for="email">Email</label>
    <input type="text" name="email" id="email" ng-model="auth.email">
  </div>
  <div>
    <input type="submit">
  </div>
</form>
```

# Formulaires

## Validation

- mélange de validation par AngularJS et HTML5
- utilisation de nouveaux types de champs HTML5
- utilisation de l'attribut **required** pour rendre un champ obligatoire
- utilisation des directives **ng-minlength**, **ng-maxlength**, **ng-pattern**
- il est conseillé d'utiliser l'attribut **novalidate** pour bypasser la validation native HTML5

## Directives de validation

AngularJS possède déjà quelques directives de validation dont certaines reproduisent le comportement du HTML5 :

```
<input type="text"
      ng-model="{string}"
      [name="{string}"]
      [required]
      [ng-minlength="{number}"]
      [ng-maxlength="{number}"]
      [ng-pattern="{string}"]
      [ng-change="{string}"]>
```

## Exemple avec des directives de validation

```
<form name="monForm" ng-submit="submitForm();">
  <div>
    <label for="username">Identifiant</label>
    <input type="text"
          name="username"
          id="username"
          ng-minlength="5"
          ng-maxlength="10"
          ng-model="auth.username">
  </div>
  <div>
    <label for="email">Email</label>
    <input type="text" name="email" id="email" ng-model="auth.email">
  </div>
  <div>
    <input type="submit">
  </div>
</form>
```

# Formulaires

## État d'un champ de formulaire

- \$pristine (aucun changement, vierge)
- \$dirty (modifié, à changé, sale)
- \$valid
- \$invalid
- \$error
- \$touched (focus puis blur sur le champ)
- \$pending (nouveau en 1.3, une validation asynchrone du champ est en cours)
- \$submitted (nouveau en 1.3)

## Exemple avec état du formulaire

```
<div ng-controller="authFormCtrl">
  <form name="authForm" ng-submit="authenticate(authForm, auth);">
    <div>
      <input type="text" name="username" ng-model="auth.username" required>
      <input type="password" name="password" ng-model="auth.password" required>
      <input type="submit" ng-disabled="userForm.$invalid">
    </div>
  </form>
</div>

<script>
  angular.module('monModule', [])

  .controller('authFormCtrl', function ($scope) {
    $scope.auth = {
      username: '',
      email: ''
    };

    $scope.authenticate = function (form, auth) {
      console.log('pristine : ' + form.username.$pristine);
      console.log('dirty: ' + form.username.$dirty);
      console.log('valid: ' + form.username.$valid);
      console.log('invalid: ' + form.username.$invalid);
      console.log('error: ' + form.username.$error); // un objet
      console.log('required error: ' + form.username.$error.required);
      console.log('touched: ' + form.username.$touched);
    };
  });
</script>
```

# Formulaires – Affichage des erreurs

## Avec ng-if

Il est important de pouvoir aider l'utilisateur au bon remplissage du formulaire. Si il commet des erreurs, il est possible de les lui afficher avec la directive **ng-if**

```
<form name="monForm">
  <div>
    <input name="pays" ng-model="pays" minlength="3" maxlength="20">
  </div>

  <div ng-if="monForm.pays.$touched">
    <div ng-if="monForm.pays.$error.required">Message A</div>
    <div ng-if="monForm.pays.$error.minlength">Message B</div>
    <div ng-if="monForm.pays.$error.maxlength">Message C</div>
  </div>

  <input type="submit">
</form>
```



# Formulaires – Affichage des erreurs

## Avec ng-messages

La version 1.3 propose un nouveau module permettant d'afficher les messages (ngMessages).

### Installation de la dépendance :

```
$ bower install angular-messages
```

### Inclusion du script :

```
<script src="bower_components/angular/angular-messages.min.js"></script>
```

### Injection de la dépendance :

```
angular.module('monModule', ['ngMessages']);
```

## Intégration de ngMessages

```
<form name="monForm" novalidate>
  <input name="pays" required ng-model="pays" minlength="3" maxlength="20">

  <div ng-messages="monForm.pays.$error">
    <div ng-message="required">Message A</div>
    <div ng-message="minlength">Message B</div>
    <div ng-message="maxlength">Message C</div>
  </div>
</form>
```

## Formulaires – Style CSS

### Décorer vos élément avec des styles CSS

En plus d'offrir des fonctionnalités à la validation de formulaires, AngularJS prévoit aussi des classes CSS permettant d'ajouter du style CSS.

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

```
<style>
  input.ng-pristine { background: white;      }
  input.ng-valid    { background: lightgreen; }
  input.ng-invalid  { background: pink;      }
</style>

<form name="monForm" novalidate>
  <input name="pays" required ng-model="pays" minlength="3" maxlength="20">

  <div ng-messages="monForm.pays.$error">
    <div ng-message="required"></div>
    <div ng-message="minlength"></div>
    <div ng-message="maxlength"></div>
  </div>
</form>
```

## Formulaires - Performance

### Réduire le déclenchement d'opérations coûteuses

Certaines opérations peuvent se révéler coûteuses si elle doivent intervenir à chaque saisie

- interroger un serveur
- filtrer un tableau volumineux

### Avec ng-model-options (introduit en 1.3)

L'option **debounce** ajoute un délai d'attente entre chaque saisie de caractères

```
ng-model-options="{debounce: {'default': 500, 'blur': 0}}"
```

L'option **updateOn** restreint au strict minimum les événements à écouter

```
ng-model-options="{updateOn: 'blur'}"
```

Exemple :

```
<div ng-controller="monContrôleur">
  <form name="userForm">
    <input type="text" name="userName"
      ng-model="user.name"
      ng-model-options="{debounce: 1000, updateOn : 'blur'}"
      ng-keyup="cancel($event)">

    Autres données :
    <input type="text" ng-model="user.data">
  </form>
  <pre>user.name = <span ng-bind="user.name"></span></pre>
</div>

// contrôleurs (escape key)
angular.module('monModule', [])

.controller('monContrôleur', function ($scope) {
  $scope.user = {name: 'say', data: ''};
  $scope.cancel = function (e) {
    if (e.keyCode == 27) {
      $scope.userForm.userName.$rollbackViewValue();
    }
  };
});
```

# Services

## Qu'est-ce qu'un service ?

Un service avec AngularJS, permet de partager du code dans toute votre application grâce à l'injection de dépendances.

## Liste de services proposé par AngularJS

- `$anchorScroll`
- `$animate`
- `$cacheFactory`
- `$compile`
- `$controller`
- `$document`
- `$exceptionHandler`
- `$filter`
- `$http`
- `$httpBackend`
- `$interpolate`
- `$interval`
- `$locale`
- `$location`
- `$log`
- `$parse`
- `$q`
- `$rootScope`
- `$sce`
- `$sceDelegate`
- `$templateCache`
- `$templateRequest`
- `$timeout`
- `$window`

# Services

## Type de services ?

Il existe deux types de services avec AngularJS :

### Simple

- factory
- service
- variables

### Configurable

- constant
- provider

# Services

## Factory

La méthode **factory** de l'objet angular permet de créer un service. Elle prend en paramètres deux arguments.

- le nom du service
- une fonction

Durant l'exécution de l'application, la fonction n'est appelée qu'une seule fois. Des services peuvent lui être injectés.

La fonction peut retourner tout type de données

- un objet
- une fonction
- une valeur
- un tableau
- ...

L'élément retourné sera transmis aux services appelants.

```
angular.module('monModule', [])

.factory('meteoService', function () {
  return {
    temperature: 28,
    etatDuTemps: function () {
      return 'Beau temps';
    }
  };
});

angular.module('monModule', [])

.controller('meteoCtrl', function ($scope, meteoService) {
  $scope.temperature = meteoService;

  $scope.afficheLeTemps = function () {
    return meteoService.etatDuTemps();
  };
});
```

# Services

## Service

La méthode **service** donne l'accès à un objet créé à partir d'un constructeur. Contrairement à la méthode **factory** le mot clé *this* peut être utilisé pour faire référence aux propriétés de l'objet.

```
var ProductService = function () {  
    var products = [];  
  
    this.getProducts = function () {  
        return products;  
    };  
};  
  
angular.module('monModule', [])  
  
    .service('productService', ProductService)  
  
    .controller('storeCtrl', function ($scope, productService) {  
        $scope.products = productService.getProducts();  
    });
```

## Services

### Service ou Factory ?

***“If you want your function to be called like a normal function, use factory. If you want your function to be instanced with the new operator, use service. If you don't know the difference, use factory”***

<http://iffycan.blogspot.fr/2013/05/angular-service-or-factory.html>



# Services

## Value

Il est possible de créer un service simples et d'avoir accès directement à sa valeur.  
Ce service n'est pas injectable en dépendances mais peut être récupérer par un décorateur.

```
angular.module('monModule', [])  
  
.value('config', {  
  application: {  
    name: 'Mon Application',  
    version: '1.3.2'  
  }  
})  
  
.value('langue', 'fr');
```

# Services

## Constant

Constant est pareil à value à la différence que ce service est injectable en dépendances mais ne peut être récupérer par un décorateur.

```
angular.module('monModule', [])  
  
  .constant('config', {  
    application: {  
      name: 'Mon Application',  
      version: '1.3.2'  
    }  
  });
```

# Les promesses

## Définition d'une promesse (promise)

Une promesse avec AngularJS est un objet correspondant au résultat différé d'une opération asynchrone.

## Utilisation

Une promesse peut prendre trois arguments optionnels dans l'ordre suivant :

1. une fonction callback si la promesse est résolu avec succès
2. une fonction callback en cas d'erreur
3. une fonction callback de notification pour fournir l'état des mise à jour en cours

```
promise.then(  
  function (valeur) {  
    console.log('Valeur: ' + valeur);  
  },  
  function (err) {  
    console.log('Err: ' + err);  
  },  
  function (valeur) {  
    console.log('Valeur: ' + valeur);  
  }  
)  
  
// Chaînage de promesses  
.then()  
.then()
```

# Services

## Exemple d'utilisation avec le service \$http

Le service **\$http** est un bon exemple d'utilisation des promesses car il en retourne une lors de l'émission d'une requête.

```
angular.module('monModule')

.controller('productController', function ($scope, $http) {
  return $http.get('http://www.google.fr').then(
    function (response) {
      console.log(response);
    },
    function (err) {
      console.log(err);
    }
  );
});
```

# Services

## Créer une promesse avec le service manager \$q

Le service **\$q** donne la possibilité de créer ses propres promesses.

```
angular.module('monModule')

function maPromesse(prenom) {
  var deferred = $q.defer();

  // code asynchrone à exécuter
  setTimeout(function () {
    if (prenom) {
      deferred.resolve(prenom);
    } else {
      deferred.reject('Prénom non défini');
    }
  }, 3000);

  // en cours ...
  deferred.notify('En cours...');

  // on retourne la promesse
  return deferred.promise;
};

// usage
promesse().then(
  function (valeur) {
    // en cas de succès
    console.log(valeur);
  },
  function (raison) {
    // en cas d'erreur
    console.log(valeur);
  },
  function (valeur) {
    // durant l'exécution
    console.log(valeur);
  }
);
```

## Service \$http

### Communication avec le serveur

Une application angularJS est chargée et exécutée côté client et les données proviennent du chargement de l'application et de l'utilisateur final.

AngularJS rend accessible la communication avec un serveur.

Le service **\$http** enveloppe l'objet XMLHttpRequest, simplifie son usage, et nous fait bénéficier en retour d'une promesse.

```
$http({
  method: 'GET',
  url: 'http://www.google.com',
})
.then(
  function (response) {
    // en cas de succès
  },
  function (error) {
    // en cas d'erreur
  }
);
```

AngularJS améliore la gestion des promesses en proposant deux méthodes nommées **success** et **error** et fourni l'accès directement aux données sans passer par l'objet réponse.

```
$http({
  method: 'GET',
  url: 'http://www.google.com',
})
.success(function (data, status) {
  // en cas de succès
})
.error(function (data, status) {
  // en cas d'erreur
});
```

# Service \$http

## Méthodes du service \$http

- `$http.get(url, [config])`
- `$http.head(url, [config])`
- `$http.post(url, [config])`
- `$http.put(url, [config])`
- `$http.delete(url, [config])`
- `$http.jsonp(url, [config])`
- `$http.patch(url, [config])`

```
$http.get('http://monsite.com/products.json')
  .success(
    function (data) {}
  )
  .error(
    function (error) {}
  );
```

## Options

- `method` (GET, POST, DELETE, ...)
- `url`
- `params`
- `data`
- `headers`
- `xsrHeaderName`
- `xsrCookieName`
- `transformRequest` (function)
- `transformResponse` (function)
- `cache` (boolean)
- `timeout`
- `withCredential` (CORS)
- `responseType` ("json", "document")

## Service \$http

### Configuration global du service \$http

#### Les entêtes

```
angular.module('monModule', [])  
  
.config(function ($httpProvider) {  
    $httpProvider.defaults.headers.post['X-Power-By'] = 'MonApplication';  
});
```

#### Configuration du cache

```
angular.module('monModule', [])  
  
.config(function ($httpProvider) {  
    $httpProvider.defaults.cache = false;  
});
```



# Service \$http

## Interceptors \$http

- request
- response
- requestError
- responseError

```
angular.module('monModule', [])  
  
.factory('monHttpInterceptor', function ($q) {  
  return {  
    response: function (response) {  
      return response;  
    },  
    responseError: function (response) {  
      console.log(response.data);  
      return $q.reject(response);  
    }  
  }  
})  
  
.config(function ($httpProvider) {  
  $httpProvider.interceptor.push('monHttpInterceptor');  
});
```

# Les directives

## Introduction

Les directives sont en charges de manipuler le DOM et d'étendre le HTML.

## Application dans l'application

- scope
- controller

## Vocabulaire

- Document
- Node
- Element
- Tag
- Attributs

# Les directives

## Syntaxe d'une directive

```
angular.module('monModule', [])  
  
.directive('maDirective', function () {  
    return {  
  
    };  
});
```

## Utilisation d'une directive

```
<ma-directive></ma-directive>  
  
<div ma-directive></div>  
  
<div class="ma-directive"></div>
```

# Les directives - template

## Création d'une directive

Déclaration d'une directive dans une vue

```
<say-hello></say-hello>
```

Résultat à obtenir

```
<div>Hello, World</div>
```

### Option 1 : Avec la propriété **template**

```
angular.module('monModule', [])  
  
.directive('maDirective', function () {  
    return {  
        restrict: 'E',  
        template: '<div>Hello, World</div>'  
    };  
});
```

### Option 2 : Avec la propriété **templateURL**

```
angular.module('monModule', [])  
  
.directive('maDirective', function () {  
    return {  
        restrict: 'E',  
        templateURL: 'chemin/du/template.html'  
    };  
});
```

# Les directives - restrict

## Définition

Il est possible de restreindre la façon de déclarer les directives avec la propriété **restrict**.

Lettre	Valeur	Contrainte par défaut
E	Element	
A	Attribut	OUI
C	Class	

## Combinaisons

Table de combinaisons possible à l'application des directives

Combinaison	Attribut	Element	Class
EA	OUI	OUI	NON
EC	NON	OUI	OUI
AC	OUI	NON	OUI
EAC	OUI	OUI	OUI

# Les directives

## Link

La manipulation du DOM dans AngularJS est contre indiqué mais parfois nécessaire. Il existe une zone dans laquelle manipuler les éléments du DOM est possible avec la propriété **link**.

```
angular.module('monModule', [])

.directive('maDirective', function () {
  return {
    restrict: 'E',
    template: '<div>Hello, World</div>',
    link: function (scope, element, attrs) {
      // manipulation du DOM
    }
  };
});
```

## Liste des paramètres de la fonction Link

Paramètre	Usage
scope	scope courant de la directive
element	le nœud du DOM sur lequel la directive a été appliquée
attrs	liste des attributs de l'élément courant sur lequel la directive a été appliqué

## Exemple

```
<ma-directive></ma-directive>
```

```
angular.module('monModule', [])

.directive('maDirective', function () {
  return {
    restrict: 'E',
    template: '<div>Hello, World</div>',
    link: function (scope, element, attrs) {
      element.css('color', attrs.color || '#F64738');
    }
  };
});
```

# Les directives

## Mise à jour de l'application vers le DOM

Pour mettre à jour le DOM, en fonction des changements de données, il suffit d'observer tout changement sur le scope et de réagir en conséquence.

```
<ma-directive color="#CCC"></ma-directive>
```

```
angular.module('monModule', [])

.directive('maDirective', function () {
  return {
    restrict: 'E',
    template: '<div>Hello, World</div>',
    link: function (scope, element, attrs) {
      scope.$watch('color', function () {
        element.css('color', attrs.color || '#F64738');
      });
    }
  };
});
```

# Les directives

## Cycle de vie

AngularJS met à disposition le service `$compile` permettant la compilation du DOM à fin d'interpréter les directives successivement dans l'ordre naturel de l'arborescence, rendant le DOM exécutable par angular.

Pour que la compilation fonctionne il faut fournir une fonction `link` à chaque directive permettant d'établir un lien entre angular et le HTML.

Il est possible d'intervenir à plusieurs niveaux du cycle de vie d'une directive pour appliquer des transformations. En pratique, intervenir dans le cycle de vie consiste à déclarer une fonction ou un objet dans la configuration de la directive. Le plus courant est d'utiliser la fonction `link` car elle est appelée quand les choses sont prêtes et que l'on souhaite interagir dessus.

### Étape 1 : compile

AngularJS récupère une fonction `link`, étape durant laquelle il est possible d'interagir sur le DOM mais pas encore avec AngularJS.

### Étape 2 : controller

Une instanciation de contrôleur est créée pour la directive et se fait avant son interprétation. Attention, cela ressemble au concept des contrôleurs sans pour autant être la même chose. Le contrôleur peut-être récupéré depuis une autre directive pour communiquer et favoriser une coopération entre elles par le jeu des dépendances entre directives.

### Étape 3 : link

Deux propriétés permettent d'intervenir avant ou après l'interprétation du contenu de la directive :

- **pre-link**
- **post-link**

La fonction `post-link` est la plus souvent utilisée pour réaliser le principal des opérations puisque, elle est exécutée après la génération du contenu ce qui lui permet d'agir dessus.



## Les directives

### Créer une directive avec la fonction compile

Utilisation de la propriété **compile** en remplacement de **link**, mais pas les deux en même temps car la fonction compile retourne elle même une fonction link.

```
<div ng-app="monModule">
  <ma-directive></ma-directive>
</div>
```

```
angular.module('monModule', [])

.directive('maDirective', function () {
  return {
    restrict: 'E',
    template: '<div>Hello, World</div>',
    compile: function (element, attrs) {
      return function (scope, element, attrs) {
        // contenu de la fonction link
      };
    }
  };
});
```

# Les directives

## Scope

- Représente la relation de la directive avec le scope
- par défaut il n'y a pas de nouvelle création de scope
- le scope utilisé dans la fonction link d'une directive est le scope courant.

```
<div ng-app="monModule">
  {{ name }}

  <ma-directive></ma-directive>
</div>
```

```
angular.module('monModule', [])

.directive('maDirective', function () {
  return {
    restrict: 'E',
    template: '<div>Hello, World</div>',
    compile: function (scope, element, attrs) {
      scope.name = 'toto';
    }
  };
});
```

# Les directives

## Scope enfant

La création d'un scope fils héritant du scope courant est possible en positionnant **scope** à **true**.

```
<div ng-app="monModule">
  {{ name }}

  <ma-directive></ma-directive>
</div>
```

```
angular.module('monModule', [])

.directive('maDirective', function () {
  return {
    restrict: 'E',
    scope: true,
    template: '<div>Hello, World</div>',
    link: function (scope, element, attrs) {
      scope.name = 'toto';
    }
  };
});
```

# Les directives

## Scope isolé

Une Directive bien conçue doit être réutilisable et indépendante du contexte. Il est nécessaire de garder le contrôle entre les échanges de la directive avec le reste de l'application.

Il suffit pour cela de déclarer un scope isolé, c'est à dire détacher le scope de tout parent.

```
<div ng-app="monModule" ng-init="name='toto'">
  <ma-directive></ma-directive>
</div>
```

```
angular.module('monModule', [])

.directive('maDirective', function () {
  return {
    restrict: 'E',
    scope: {},
    template: '<div>Hello, World</div>',
    link: function (scope, element, attrs) {
      console.log(scope.name);
    }
  };
});
```

# Les directives

## Copie d'une valeur du scope

Il est possible de recopier la valeur d'un attribut compris dans le scope de la directive.

```
<div ng-app="monModule" ng-init="name='toto'">
  <ma-directive txt="Hello, World"></ma-directive>
</div>
```

```
angular.module('monModule', [])

.directive('maDirective', function () {
  return {
    restrict: 'E',
    scope: {
      txt: '@'
    },
    template: '<div>{{ txt }}</div>'
  };
});
```

# Les directives

## Binding

La déclaration du symbole « = » dans une propriété du scope permet de lier une variable du scope avec une variable d'une directive

```
<div ng-app="monModule" ng-controller="monController">
  <ma-directive txt="Hello, World"></ma-directive>
  <input ng-model="page.url">
</div>
```

```
angular.module('monModule', [])

.controller('monController', function ($scope) {
  $scope.page = {
    name: 'home',
    url: 'http://angularjs.org'
  };
})

.directive('maDirective', function () {
  return {
    restrict: 'E',
    scope: {
      txt: '='
    },
    template: '<input ng-model="txt">'
  };
});
```

# Les directives

## Evaluation d'une expression

Il est possible d'évaluer une expression avec le symbole « & »

```
<div ng-app="monModule">
  <ma-directive calc="resultat = resultat + 1"></ma-directive>
  {{ resultat }}
</div>
```

```
angular.module('monModule', [])

.directive('maDirective', function () {
  return {
    restrict: 'E',
    scope: {
      calc: '&'
    },
    template: '<div><button ng-click="calc();">Result</button></div>'
  };
});
```

# Les directives

## Expression avec paramètres

```
<div ng-app="monModule">
  <ma-directive calc="func(num)"></ma-directive>
  {{ resultat }}
</div>
```

```
angular.module('monModule', [])

.controller('monController', function ($scope) {
  $scope.func = function (param) {
    $scope.resultat = param;
  };
})

.directive('maDirective', function () {
  return {
    restrict: 'E',
    scope: {
      calc: '&'
    },
    template: '<div><button ng-click="calc({num:1});">Result</button></div>'
  };
});
```



# Les directives

## Transclusion

Inclusion par référence d'un document ou d'une partie d'un document dans un autre document.

```
<div ng-app="monModule">
  <ma-directive>
    zone à transcluser
  </ma-directive>
</div>
```

```
angular.module('monModule', [])

.directive('maDirective', function () {
  return {
    restrict: 'E',
    transclude: true,
    template: '<div>Ma Directive<div ng-transclude></div></div>'
  };
});
```

# Les directives

## Transclusion et scope

Le scope utilisé dans le code transcluté est le scope courant et non pas celui de la Directive

```
<div ng-app="monModule">
  <input ng-model="txt">

  <ma-directive>
    zone à transcluser : {{ txt }}
  </ma-directive>
</div>
```

```
angular.module('monModule', [])

.directive('maDirective', function () {
  return {
    restrict: 'E',
    transclude: true,
    scope: {},
    template: '<div>Ma Directive {{ txt }}<div ng-transclude></div></div>'
  };
});
```

# Les directives - require

## Contrôleur

Liaison et communication entre les directives

```
<div ng-app="monModule">
  <ma-directive></ma-directive>
</div>
```

```
angular.module('monModule', [])

.directive('maDirective', function () {
  return {
    restrict: 'E',
    controller: function ($scope) {
      $scope.res = 'Foo';
      this.func = function () {
        $scope.res += 'Bar';
      };
    }
    template: '<div>{{ result }}</div>'
  };
});
```

## Les directives – autres options

### replace

Prend deux états possible

- **vrai**, le template remplace l'élément sur lequel la Directive est appelée
- **faux**, le template remplace le contenu de l'élément sur lequel la Directive est appelée

### priority

Ordre dans lequel les directives doivent s'appliquer. Une priorité haute donne la précedence dans l'ordre de l'application.