



INTRODUCTION À ANGULAR JS

Sébastien Chassande-Barrioz

Architecte

sebastien.chassande@soprasteria.com

PLAN

1. Introduction
2. Les modules
3. Controllers & Scope
4. Les directives
5. Les services
6. Le routage
7. Outillage
8. Conclusion
9. Bonus





Angular JS : Introduction



Introduction

C'est quoi Angular JS ?

- Framework (pas une librairie)
- Construction d'applications web
- Organisation / Architecture du code
- 100% client
- 100% JS
- Créé par Google en 2009
- Navigateur : FF, Chrome, IE9+, Safari, Opéra, IOS/Android browser)



Introduction

Les fonctionnalités intéressantes d'Angular JS

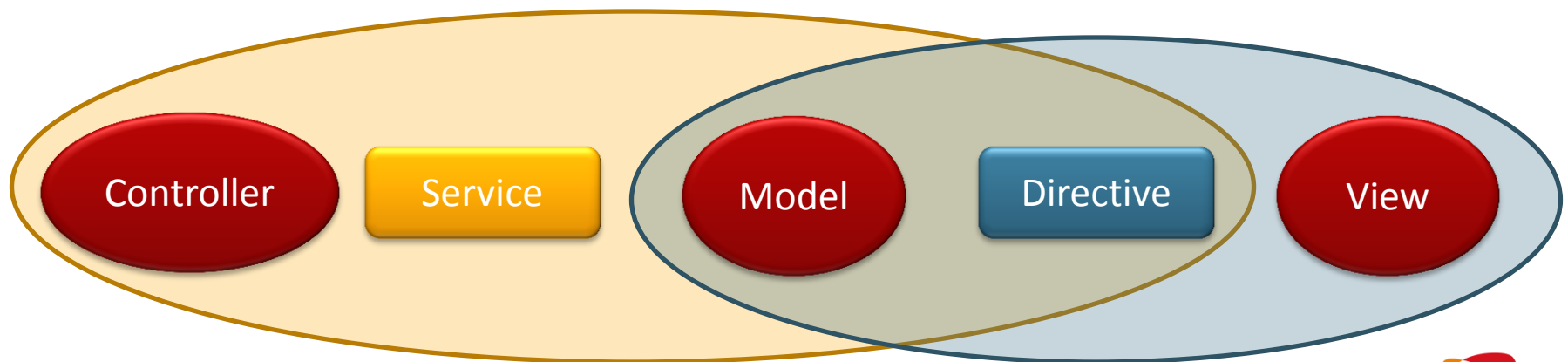
- Binding bi-directionnel
- Pattern MVW
- Modularité
- Injection de dépendances
- Routage
- Validation



Introduction

Les concepts

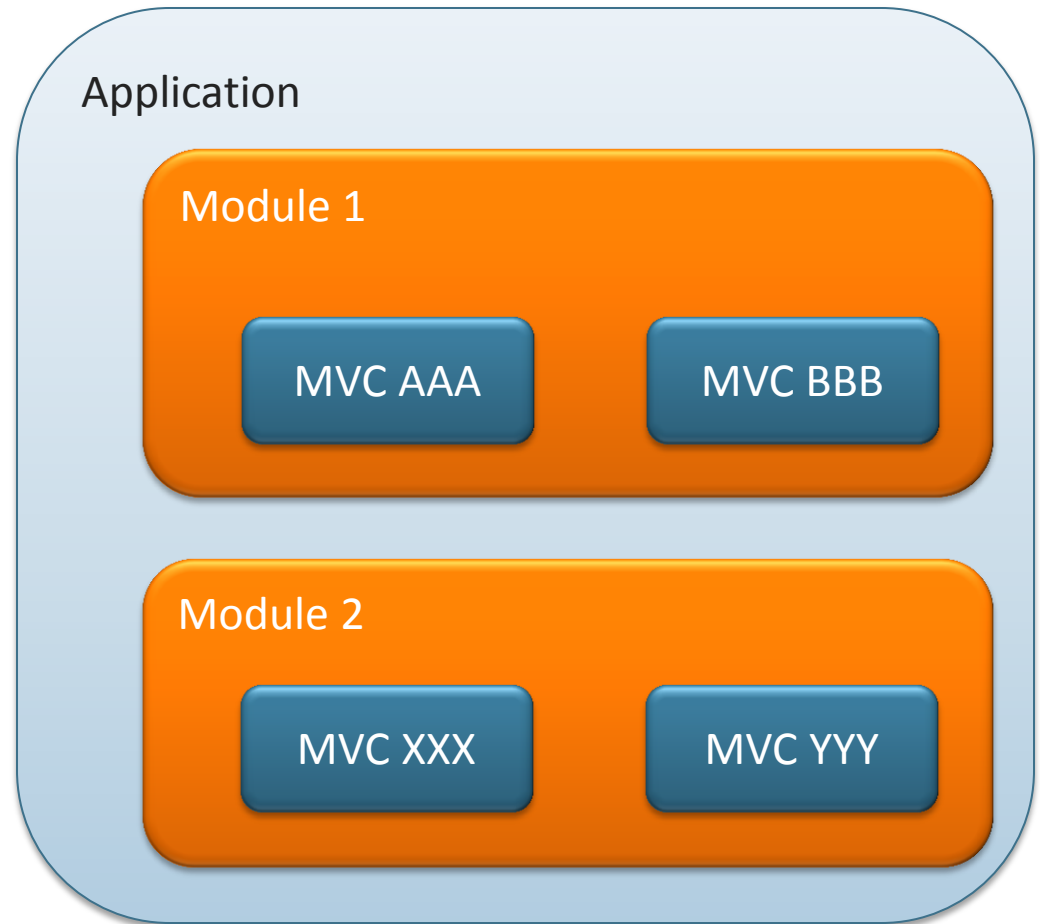
- **Model** : purs objets JSON (POJO) contenant les données
- **View** : templates HTML accédant aux données exposées
- **Controller** : fonction JS exposant le modèle aux vues via l'objet \$scope
- **Directive** : concept spécifique Angular permettant d'ajouter du comportement aux applications. Fait souvent le lien entre la vue et le model.
- **Service** : permet de concentrer la logique hors des controllers qui doivent rester compacts et compréhensibles



Introduction

Organisation logique d'Une application

- Une application = { module }
- Module = { Controller }
- module $\neq$ namespace
 - Nom de controller unique



Introduction

Organisation des fichiers

- 1 fichier par composant :
 - Application,
 - Controller,
 - View,
 - Module,
 - Service
- Organisation hiérarchique
 - Module / Famille fonctionnalité / Fonctionnalité
- Nommage des fichiers : le + complet
 - Objet + Type

```
App
  schedule
    scheduleDetails
      scheduleCtrl.js
      scheduleSvc.js
      schedule.html
    scheduleTile
      scheduleTile.js
      scheduleTile.html
      scheduleTileSvc.js
  common
    scheduleResource.js
```



Introduction

Une application avec Angular JS

- Récupérer angular.js depuis le site web
- 2 conditions sur une page html :
 - Ajouter angular.js dans une balise script

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

- Ajouter un attribut ng-app sur une balise
 - Définit le nom du module de l'application
 - Indique le début du code à traiter par AngularJS
 - Un seul ng-app par page !
 - ng = raccourci pour Angular JS

```
<body ng-app="app">  
...  
</body>
```



Angular JS : Les modules



LES MODULES

Pourquoi des modules ?

- AngularJS utilise une logique modulaire pour déclarer les différents composants de l'application.
- Le chargement des modules peut être fait dans n'importe quel ordre
- Les bibliothèques externes sont souvent récupérées sous forme de module
Permet d'avoir des fichiers séparés par responsabilité (un fichier / service, controller, module)

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

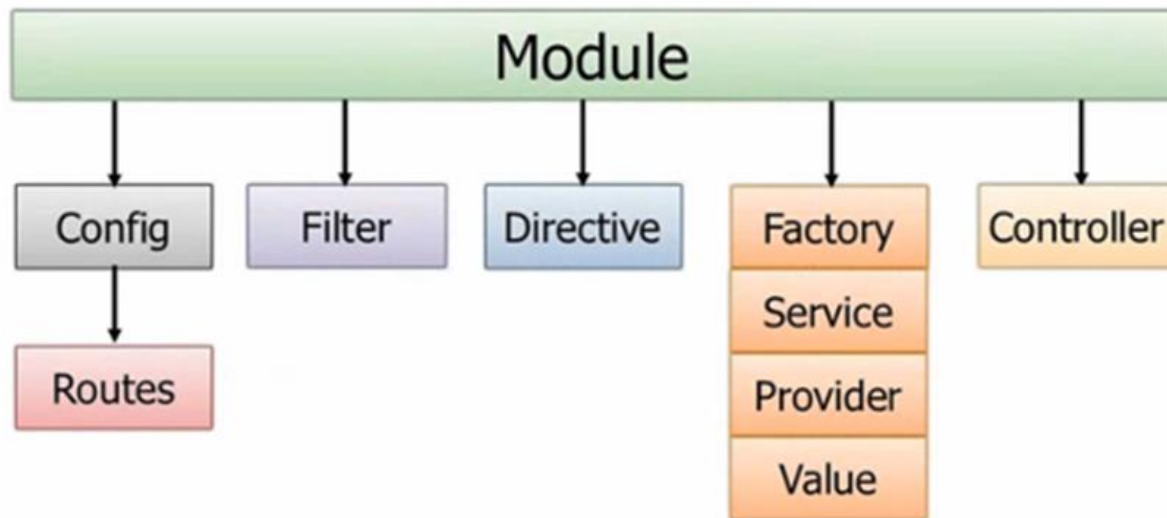
- Même si un module n'a pas de dépendance,
il ne faut pas omettre les crochets. Sinon c'est l'appel d'un module existant.



LES MODULES

Que contient un module ?

```
<html ng-app="moduleName">
```



- Déclaration des membres d'un module :
 - `var accountModule = angular.module('myapp-account', []);`
 - `accountModule.controller('accountCtrl', function () {});`
 - `accountModule.service('accountService', function () {});`
 - `var myapp = angular.module('myapp', ['myapp-account']);`



LES MODULES

Déclaration et dépendances

- Une application Angular démarre par un module racine don't toutes les dépendances sont attachées/déclarées dessus :

```
var app = angular.module('fra', ['fra-core', 'fra-layout', 'fra-home', 'fra-endpoints', 'fra-requests']);
```

- 'fra' utilise le module 'fra-core' :

```
angular.module('fra-core', [ 'ngMessages', 'ui.router', 'ui.bootstrap', 'restangular']);
```

- On ne répète pas les dépendances récursives



LES MODULES

Les modules facultatifs

- *ngCookies*, pour la gestion des cookies
- *ngTouch*, pour les événement liés au mobile
- *ngRoute*, pour le routage
- *ngResource*, pour interagir avec des ressources REST
- *ngMessages*, pour l'affichage de messages notamment pour les champs de formulaires
- *ngAnimate*, pour les animations
- *ngSanitize*, pour vérifier le HTML



LES MODULES

Les modules essentiels

- *AngularUI Bootstrap*
Adaptation du module JS de Bootstrap en directives Angular
- *Angular Material*,
Implémentation de la spécification Material Design de Google en Angular.
- *AngularUI Router*
Module de gestion de routes beaucoup plus puissant que le module officiel (ngRoute)
- *Angular Translate*
Permet une gestion plus poussée de l'i18n.





Angular JS : Controllers & Scope



CONTROLLERS & SCOPE

Role et définition d'un controller

- Les contrôleurs sont chargés de manipuler le modèle et de faire lien avec la vue (page HTML).
- Il est possible de définir plusieurs contrôleurs sur un template.
- AngularJS gère le cycle de vie des contrôleurs.

JS

```
function CustomerCtrl() {  
    var vm = this;  
    vm.name = {};  
}
```

HTML

```
<div ng-controller="CustomerCtrl as customer">  
    {{ customer.name }}  
</div>
```

CONTROLLERS & SCOPE

Déclaration d'un controller sans variable globale

- Utilisation d'un IFE

```
(function () {  
    "use strict";  
  
    var module = angular.module("pizzaApp", ["ngRoute"]);  
  
    module.controller("orderController", function () {  
        var order = this;  
  
        order.orderCount = 0;  
  
        order.addOrder = function () {  
            order.orderCount++;  
        };  
    });  
  
})();
```

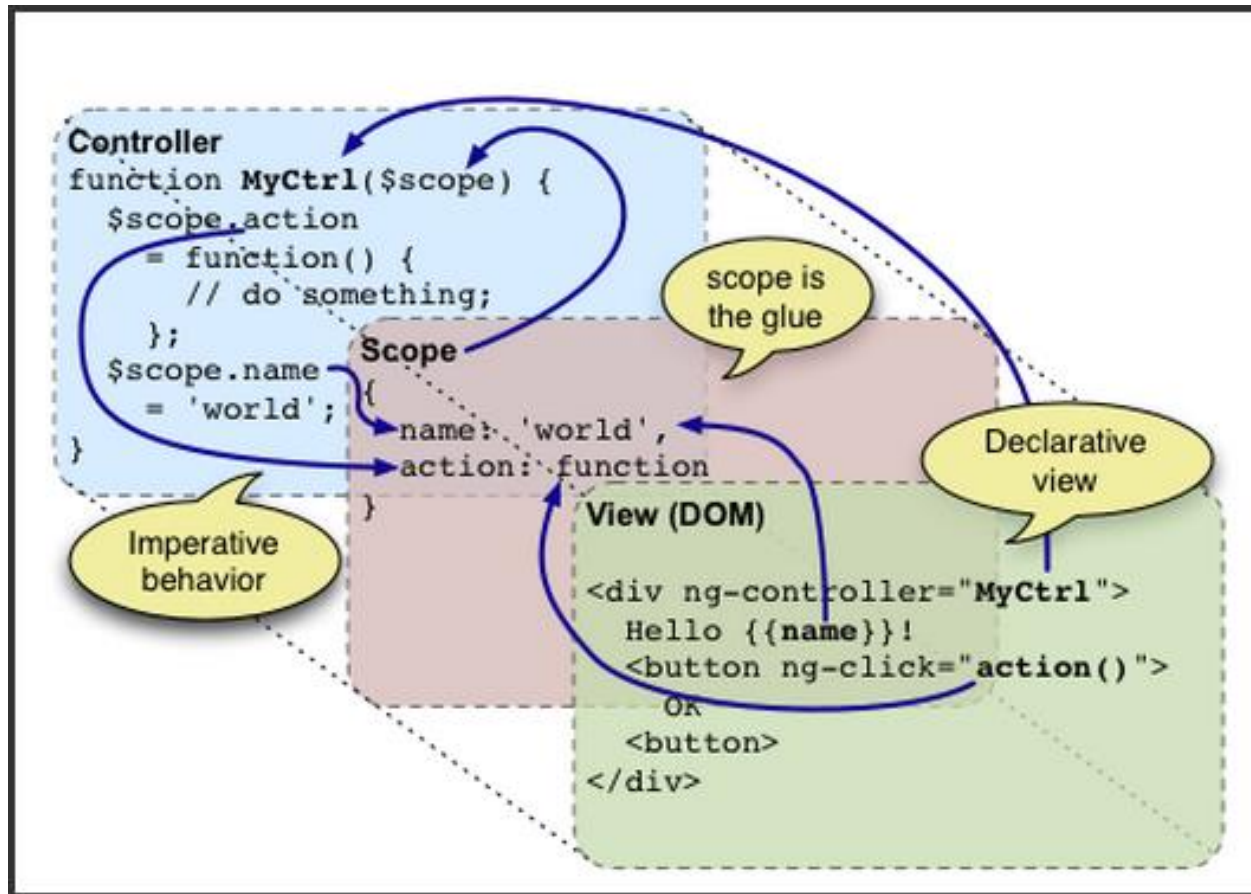
- Site de référence pour le coding style :
<https://github.com/johnpapa/angular-styleguide#controllers>



CONTROLLERS & SCOPE

Scope (1/2)

- Le scope est lien entre le contrôleur et la vue.
- Chaque contrôleur créé aura son propre scope.



CONTROLLERS & SCOPE

Scope (2/2)

```
<input type="text" ng-model="hello"/>
<h1>{{ hello }} world !!!</h1>
```

- Comment est évaluée l'expression {{ hello }} ?
- AngularJS va chercher un attribut nommé 'hello' dans l'objet scope, si celui-ci est trouvé alors la valeur de l'attribut sera affichée sinon une chaîne de caractères vide.
- Le scope est ainsi utilisé pour initialiser des valeurs ou récupérer les valeurs modifiées par un utilisateur (cf double data binding).



CONTROLLERS & SCOPE

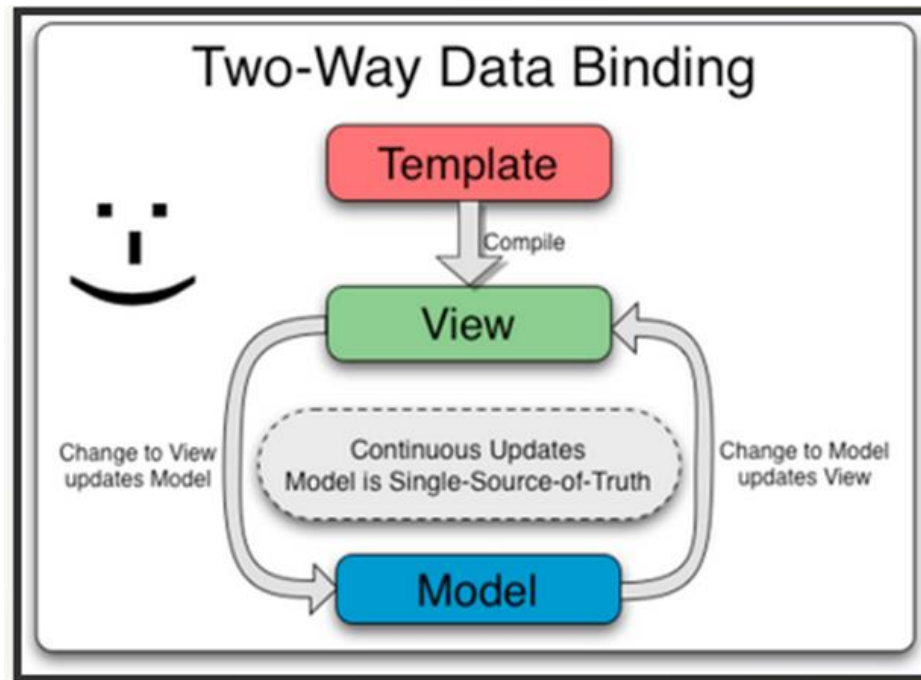
Expressions

- Angular interprète les templates, ce qui permet d'écrire de l'HTML contenant des expressions dynamiques :
 - `<h1>{{ 1+1 }}</h1> // 2`
 - `<h1>{{ "Hello " + "world !!!" }}</h1> // Hello world`
- Les expressions pardonnent les null ou undefined :
 - une chaîne de caractères vide sera affichée
- RÈGLES
 - Les variables des expressions sont évaluées depuis l'objet \$scope et non l'objet window.
 - Pour respecter la séparation du code entre les vues et les contrôleurs, il est impératif de garder les expressions simples.
 - Il est préférable de faire appel à une fonction exposée par un contrôleur.
 - Il est possible d'utiliser des filtres dans les expressions



CONTROLLERS & SCOPE

Double data binding



- Le double data-binding d'Angular permet de maintenir le modèle en accord avec la vue et vice-versa.

```
<input type="text" ng-model="hello.myclass"/>
<h1 class="{ { hello.myclass } }">Hello world !!!</h1>
```



Angular JS : Les directives



LES DIRECTIVES

Les directives les plus courantes (1/2)

- Directive = attribut dans une balise de l'arbre DOM.
- Permet d'agir sur la vue ou le modèle
- ng-class : affecter des style selon des conditions
Exemple : Affecte à la balise <p>, le style *style4* lorsqu'il sera défini et le style *orange* si *warning* est vrai

```
<p ng-class="[style4, {orange: warning}]">
```

- ng-click : Permet de spécifier une fonction à appeler lorsque l'objet est cliqué.

```
<p ng-click="vm.update(film)">
```



LES DIRECTIVES

Les directives les plus courantes (2/2)

- ng-model : Permet de lier une variable du modèle à une balise input, select ou textarea

```
<input ng-model="user.name" type="text">
```

- ng-repeat : Permet de répéter le contenu d'une balise

```
Liste de mes amis: <ul>  
  <li ng-repeat="friend in friends">  
    [{{$index + 1}}] {{friend.name}} who is {{friend.age}} years old.  
  </li>  
</ul>
```

LES DIRECTIVES

LES FILTRES

- Permet de sélectionner un sous-ensemble d'un tableau sous la forme d'un nouveau tableau.
- Syntaxe dans le HTML :
`{{ filter_expression | filter : expression : comparator }}`
- Syntaxe dans le Javascript :
`$filter('filter')(array, expression, comparator)`

- Les types filtres :

Name	
currency	<code>{{amount currency:"USD\$"}}</code>
date	<code>{{startDate date:'short'}}</code>
filter	<code>repo in repos filter:searchTerm</code>
json	<code>{{ repo json }}</code>
limitTo	<code>repo in repos limitTo:10</code>
lowercase, uppercase	<code>{{ user.name uppercase }}</code>
number	<code>{{ stars number }}</code>
orderBy	<code>repo in repos filter:searchTerm orderBy:'name'</code>



LES DIRECTIVES

Bien d'autres directive

- <https://docs.angularjs.org/api/ng/directive/>

ng-app	ng-bind	ng-blur	ng-change
ng-class	ng-click	ng-copy	ng-dblclick
ng-disabled	ng-focus	ng-hide	ng-href
ng-if	ng-include	ng-init	ng-keydown
ng-keypress	ng-keyup	ng-model	ng-mouseenter
ng-mouseleave	ng-mousemove	ng-mouseover	ng-paste
ng-repeat	ng-style	ng-switch	ng-transclude





Angular JS : Les services



LES SERVICES

Utiliser un service

- Service = ensemble de fonctions communes/génériques non spécifique à un controller
- Un controller devrait utiliser des services pour accéder au monde extérieur
- Commence toujours par \$
- Exemples : \$http, \$log, \$interval ...
- Déclarer le service dans le paramètre du controller et dans ses dépendances :



LES SERVICES

Création d'un service

```
(function () {  
    'use strict';  
  
    angular.module('film-session').service('FilmSession', FilmSession);  
  
    /*Object Pattern (a service is created with new)*/  
    function FilmSession(FilmDao) {  
  
        var that = this;  
  
        this.films = [];  
  
        // public API is defined on this  
        this.findAllFilms = findAllFilms;  
  
        // private API  
        function findAllFilms() {  
            FilmDao.findAll().then(function (response) {  
                that.films = response.data;  
            }, errorCallback);  
        }  
  
        // private API  
        function errorCallback(error) {...}  
    }  
  
    }) ();
```

LES SERVICES

Pourquoi créer un service ?

- Partage de code métier
- Partage de données
 - Le service est un singleton
- Gérer la complexité





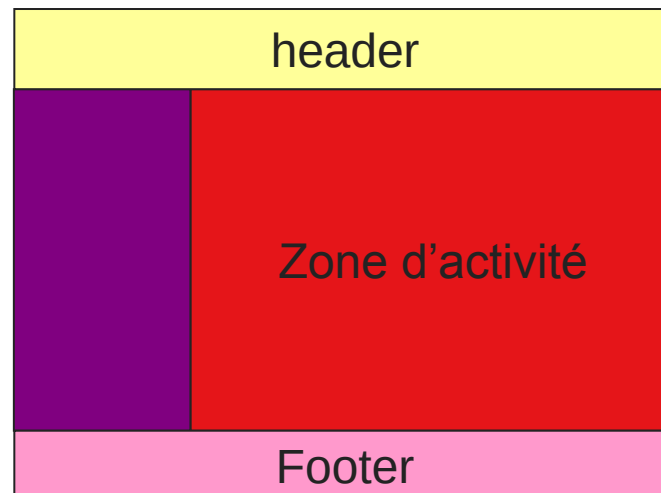
Angular JS : Routage



Le routage

Principes

- Une application = { activité }
- Organisation de la page contient souvent une zone pour les activités
- Une activité active en même temps
- Activité a une url : encodé dans l'ancre de l'unique html page



Le routage

En angular (1/2)

- Ajouter la dépendance JS : angular-route.js

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

- Ajouter ngRoute dans les dépendances du module

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

- Déclaration de la zone qui contiendra les activités : utilisation de la directive *ng-view* à placer sur une balise
- La doc : https://docs.angularjs.org/tutorial/step_07



Le routage

En angular (1/2)

- Déclaration du mapping des URLs :

```
myApp.config(['$routeProvider', function($routeProvider) {  
  $routeProvider  
    .when('/phones',  
      { templateUrl: 'partials/phone-list.html',  
        controller: 'PhoneListCtrl' })  
    .when('/phones/:phoneId',  
      { templateUrl: 'partials/phone-detail.html',  
        controller: 'PhoneDetailCtrl' })  
    .otherwise({ redirectTo: '/phones' });  
});
```

Url de l'activité

Template html

Référence vers
la variable du
controller

Mapping par
défaut





Outil pour AngularJS



Outil pour AngularJS

DEEP COPY

- Deep Copy : `angular.copy`
 - Creates a deep copy of source, which should be an object or an array.

Usage

```
angular.copy(source, [destination]);
```

- If no destination is supplied, a copy of the object or array is created and returned by the function
- If a destination is provided, all of its elements (for arrays) or properties (for objects) are deleted and then all elements/properties from the source are copied to it.
- If `source` is not an object or array (inc. `null` and `undefined`), `source` is returned.
- If `source` is identical to 'destination' an exception will be thrown.



Outil pour AngularJS

Difference \$watch \$on

- The **\$watch** function is used to watch variables on the scope. Scope inheritance allows you to watch parent scope variables as well, so that is definitely the way to go for your use case.

```
$scope.$watch(function(scope) { return scope.data.myVar },
    function(newValue, oldValue) {
        document.getElementById("").innerHTML =
            "" + newValue + "";
    }
);
```

- **\$on** is used to watch for events, which you can **\$broadcast** to child scopes or **\$emit** to parent scopes. This gives you much more control, but it might cause more mistakes while coding, since you could get an update to a scope variable from a point which you don't monitor and forget to notify the listeners.

```
$rootScope.$on('$stateChangeStart', stateChangeStart);

function stateChangeStart(event, toState, toParams, fromState, fromParams) {
    console.log('STATE change start', event, toState, toParams, fromState, fromParams);
}
```



Outil pour AngularJS

\$parse

- [https://docs.angularjs.org/api/ng/service/\\$parse](https://docs.angularjs.org/api/ng/service/$parse)
- To transform a function literal (or attribute literal) into a living access

```
var myFunctionStr = 'myExecute(myParam)';
```

```
var fn = $parse(myFunctionStr);
```

```
fn($scope);
```

```
$scope.myExecute = function (bar) {  
    console.log(bar);  
};
```

Returns

function(context, locals)

a function which represents the compiled expression:

- `context` – {object} – an object against which any expressions embedded in the strings are evaluated against (typically a scope object).
- `locals` – {object=} – local variables context object, useful for overriding values in `context`.

```
fn($scope , { myParam : 'overriding MyParam with this value !' });
```





- Karma (independent from Angular, but ...)
 - run your test in a console using NodeJs : unit test or e2e (integration : Selenium like)
 - Can run test from multiple testing framework : Jasmine (default), Mocha, Qunit
 - multiple browser targets : Chrome, IE
- Protractor
 - something like Selenium for e2e testing (and better than Karma because dedicated to e2e tests)

- Jasmine
 - Testing framework



- Mocks :
 - angular-mock (ngMocks)
 - Sinon

Sinon.JS





Conclusion



AngularJS

Feelings over time

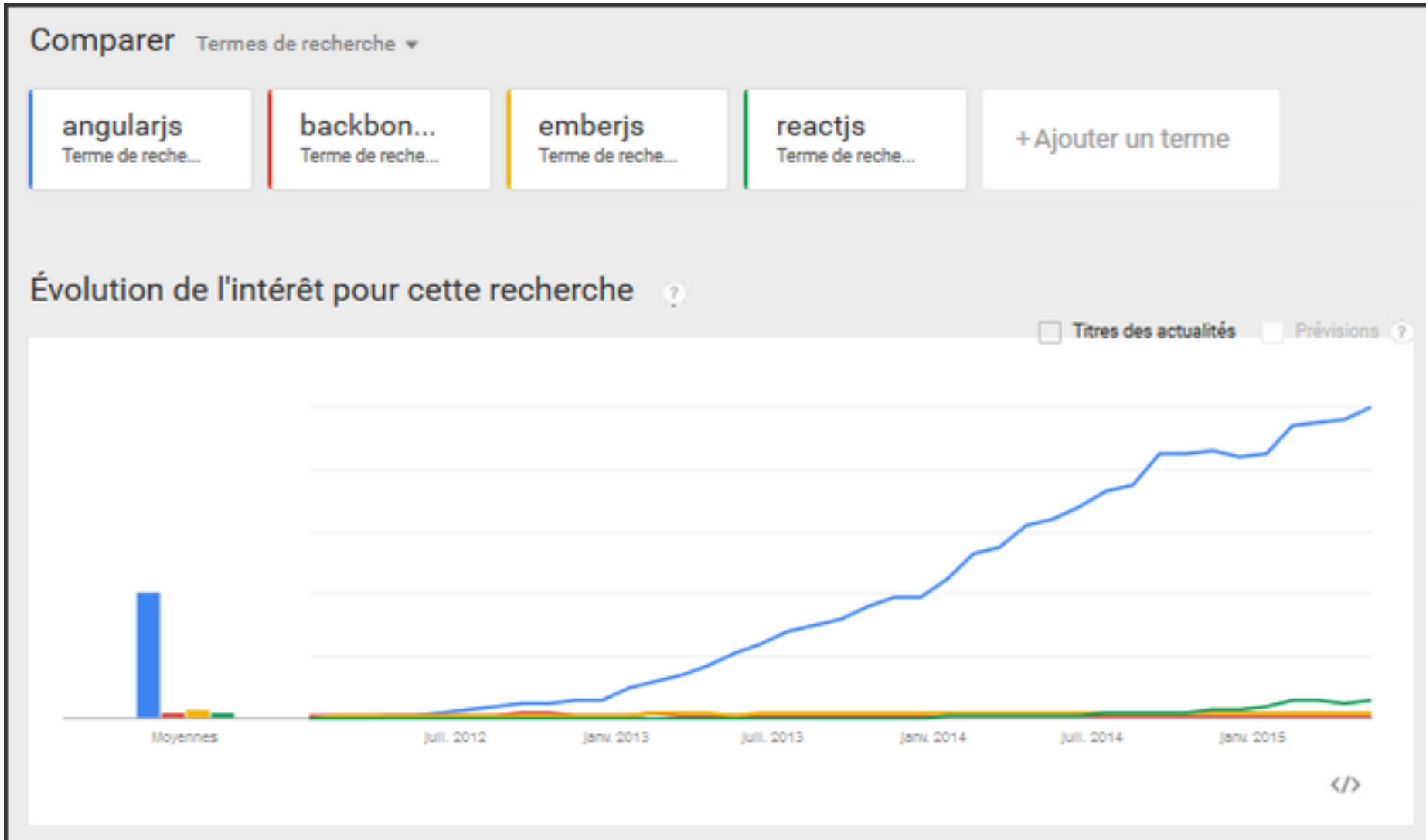


My Feelings About AngularJS Over Time



Introduction

Encore un autre framework JS ?



CONCLUSION

TOUT UN ECOSYSTÈME

- apporte un peu de rationalisation dans le monde du développement JS
- Tout un environnement
- S'intègre aussi dans une stack full JS :
 - M : MongoDB
 - E : Express
 - A : Angular JS
 - N : Node JS
- A le vent en poupe, forte communauté
- Un must à connaître pour le futur





Bonus



Bonus

Lazy loading

- Angular Services are LAZY loaded
- If you don't explicitly call a service, it won't be created unless you tell Angular to do so :

```
angular.module('app').run(function (myEagerService) {  
  |   console.log("MyEagerService is initialized !");  
});
```



Bonus

Konwing Lifecycles as always make your life easier !



BIG rules :

- **Controllers** are instantiated with every change of route ng-route or UI-Router. Singleton but not long-lived.
- **Services** singleton will persist through route change and could be thought of as http session on server side
 - => Services are independent of routes
 - => Services could also play the M Controller being the MV (model data restricted for a specific view)
- Long lived data could be stored in local storage :
 - i.e : User login info
 - Use : Angular Storage



AngularJS

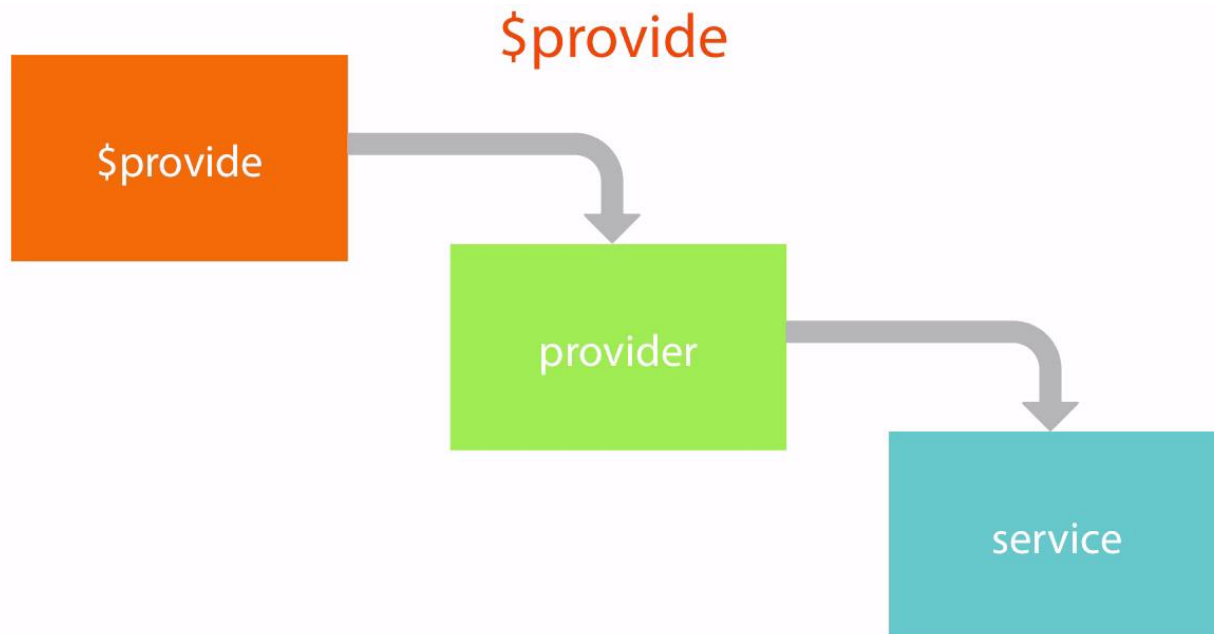
- value
 - build what you want BUT without any need for injections
 - - constant
 - - factory of objects
 - - pure business API
- factory
 - injection of dependencies : \$http ...
 - js styled : revealing module pattern
- service
 - injection of dependencies : \$http ...
 - object styled : this.prop, this.func
 - angular uses a new on your function
- provider
 - before creation : param injection is possible



Bonus

Les providers (1/3)

- Providers are Objects knowing how to create injectable services



AngularJS

Les providers (2/3)

- The service will be named: 'book'
- The provider will be named: 'bookProvider'
- \$get is required
- Advantage of the provider syntax : configurable during the module configuration phase

```
$provide.provider('books', function () {  
    this.$get = function () {  
        var appName = 'Book Logger';  
        return {  
            appName: appName  
        };  
    }  
});
```



AngularJS

Les providers (3/3)

- Built as a provider a service is configurable during Module Configuration Phase
- service : 'book' and provider : 'bookProvider'

```
app.provider('books', function () {
  var includeVersionInTitle = false;
  this.setIncludeVersionInTitle = function (value) {
    includeVersionInTitle = value;
  };

  this.$get = function () {
    var appName = "Super Todos App!";
    if (includeVersionInTitle) {
      appName += ' ' + version;
    }
    return {
      appName: appName
    };
  };
});
```

```
app.config(['booksProvider', function (booksProvider) {
  booksProvider.setIncludeVersionInTitle(true);
}]);
```



FACTORY SYNTAX

```
(function () {  
    'use strict';  
    angular.module('film-session').service('FilmSession', filmSession);  
  
    function filmSession(FilmDao) {  
  
        var films = [];  
  
        var publicApi = {  
            findAllFilms: findAllFilms  
        };  
  
        return publicApi;  
  
        // private function  
        function findAllFilms() {  
            FilmDao.findAll().then(function (response) {...}, errorCallback);  
        }  
  
        // private function  
        function errorCallback(error) {...}  
    }  
}) ();
```



Bonus

Starting sequence

- Angular Start

- `app.config()`
- `app.run()`
- *directive's compile functions (if they are found in the dom)*
- `app.controller()`
- *directive's link functions (again if found)*
- demo : <http://jsfiddle.net/ysq3m/>

- **Run blocks -**

- get executed after the injector is created and are used to kickstart the application. Only instances and constants can be injected into run blocks. This is to prevent further system configuration during application run time.
- **Run blocks are the closest thing in Angular to the main method.** A run block is the code which needs to run to kickstart the application. It is executed after all of the service have been configured and the injector has been created.
- **Run blocks typically contain code which is hard to unit-test**, and for this reason should be declared in isolated modules, so that they can be ignored in the unit-tests.

