MICKAEL JEANROY

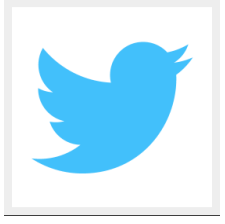
MICKAEL.JEANROY@ZENIKA.COM

INSA LYON - IF 2015

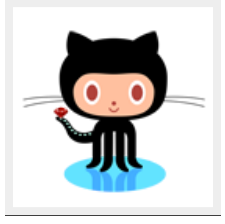
13 / 10 / 2015

INTRODUCTION

WHO AM I ?

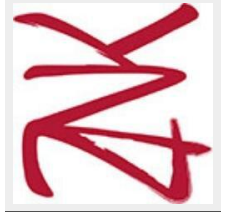


@mickaeljeanroy



github.com/mjeanroy

IF 2008



@ZenikaIT

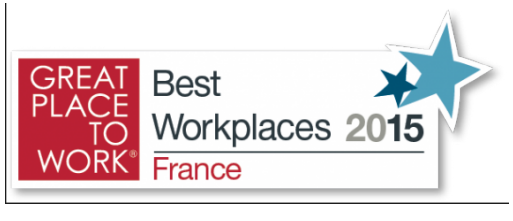
PROGRAMME

1. Zenika
2. Javascript - ES5 - ES6
3. Ecosystème & Industrialisation
4. Angular.js : principes
5. Angular.js : templates, contrôleurs, filtres

Du code

zenika
ARCHITECTURE INFORMATIQUE

- SSII fondée en 2006
- Présent à Paris, Lyon, Rennes, Nantes, Lille et Bordeau
- Développement, Architecture, Formations
- 4ème au classement "Great Place To Work" 2015
- 7ème au classement "Happy Trainees" 2015




zenika
ARCHITECTURE INFORMATIQUE


Java 8


spring
by Pivotal


HIBERNATE

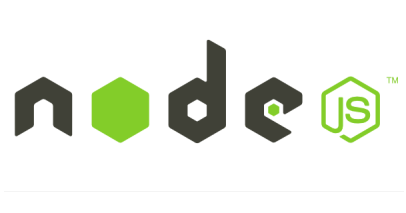

mongoDB


elasticsearch.


cassandra


HTML
5


ANGULARJS
by Google


node
JS

STAGES (HTTPS://JOBS.ZENIKA.COM/LES-OFFRES/)

- Générateur Yeoman (Lyon) : participation active à un projet Open Source sur le projet de générateurs Yeoman.
- Outil de gestion de Logs (Lyon) : Tenter de faire mieux que Logstash.
- Conception et à la réalisation de projets internes basés sur les meilleurs outils et Frameworks Java (Paris).
- Projet de R&D innovant en relation avec la Direction Technique (Paris).


ANY
QUESTIONS
?

JAVASCRIPT

INTRODUCTION

- Créé en 1995 par Brendan Eich pour Netscape
- Standardisé : ECMAScript
  - ECMAScript 3 : 1999
  - **ECMAScript 5** : 2009
  - ECMAScript 6 : 2015 !
- ECMAScript 5 est la version la mieux implémentée dans les navigateurs (Chrome, Firefox, Safari, IE10)
- ECMAScript 6 : besoin d'une phase de build (**babel**, traceur)

JAVASCRIPT

INTRODUCTION

- Langage de script
- S'exécute dans une VM (Virtual Machine)
  - Browser: V8 (Chrome) ; SpiderMonkey (Firefox) ; JavaScriptCore (Safari) etc.
  - Serveur: V8 (Node.JS) ; Rhino / Nashorn (Java) etc.
- La mémoire est gérée grâce à un **garbage collector**
- Langage **dynamique**
- Langage **faiblement typé**
- Langage **orienté prototype**

JAVASCRIPT

DEBUG

- Le plus basique: console.log
- Permet de laisser des traces sur les appels
- Certains navigateurs proposent des features en plus :
  - `console.table()`
  - `console.time()`
  - `console.memory()`
  - etc.

```
// Fonction nommée "foo"
function foo() {
  console.log('hello world', 'hello bar');
}
```

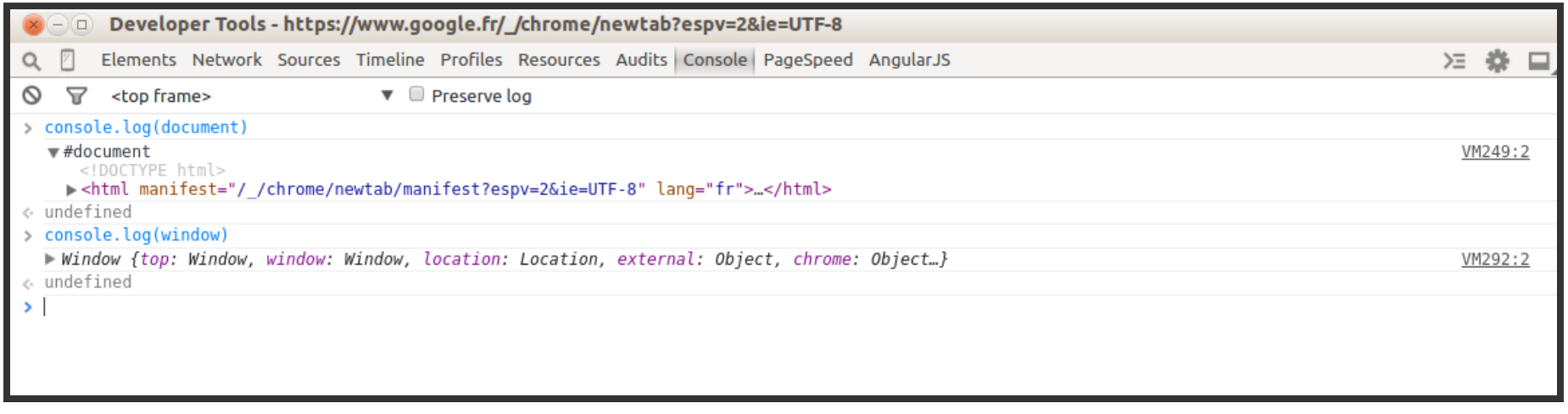
JAVASCRIPT

DEBUG

- **Chrome Dev Tools** : le meilleur !
- Firebug / Firefox Developer Edition
- Safari Web Inspector
- Opera DragonFly
- IE : présent depuis IE9

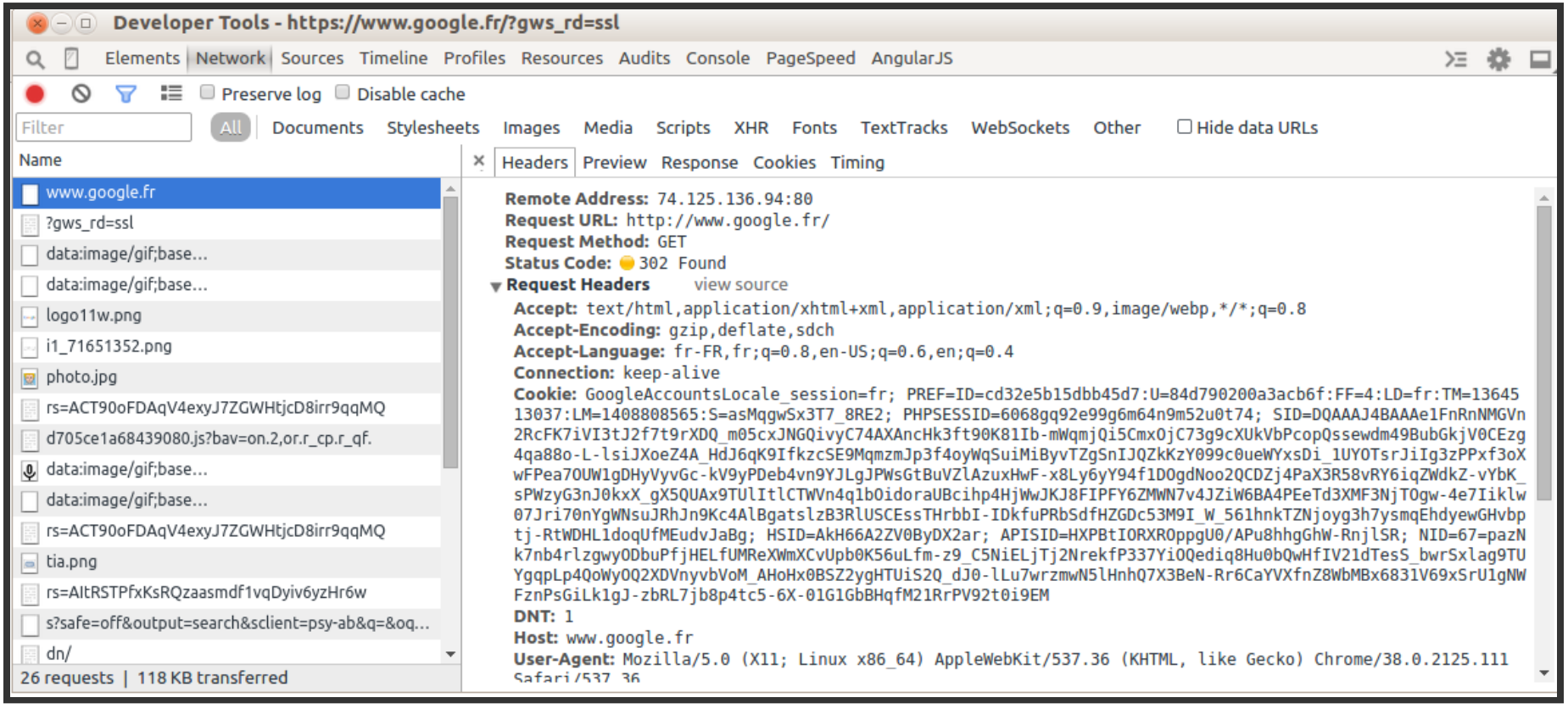
JAVASCRIPT

DEBUG



JAVASCRIPT

DEBUG

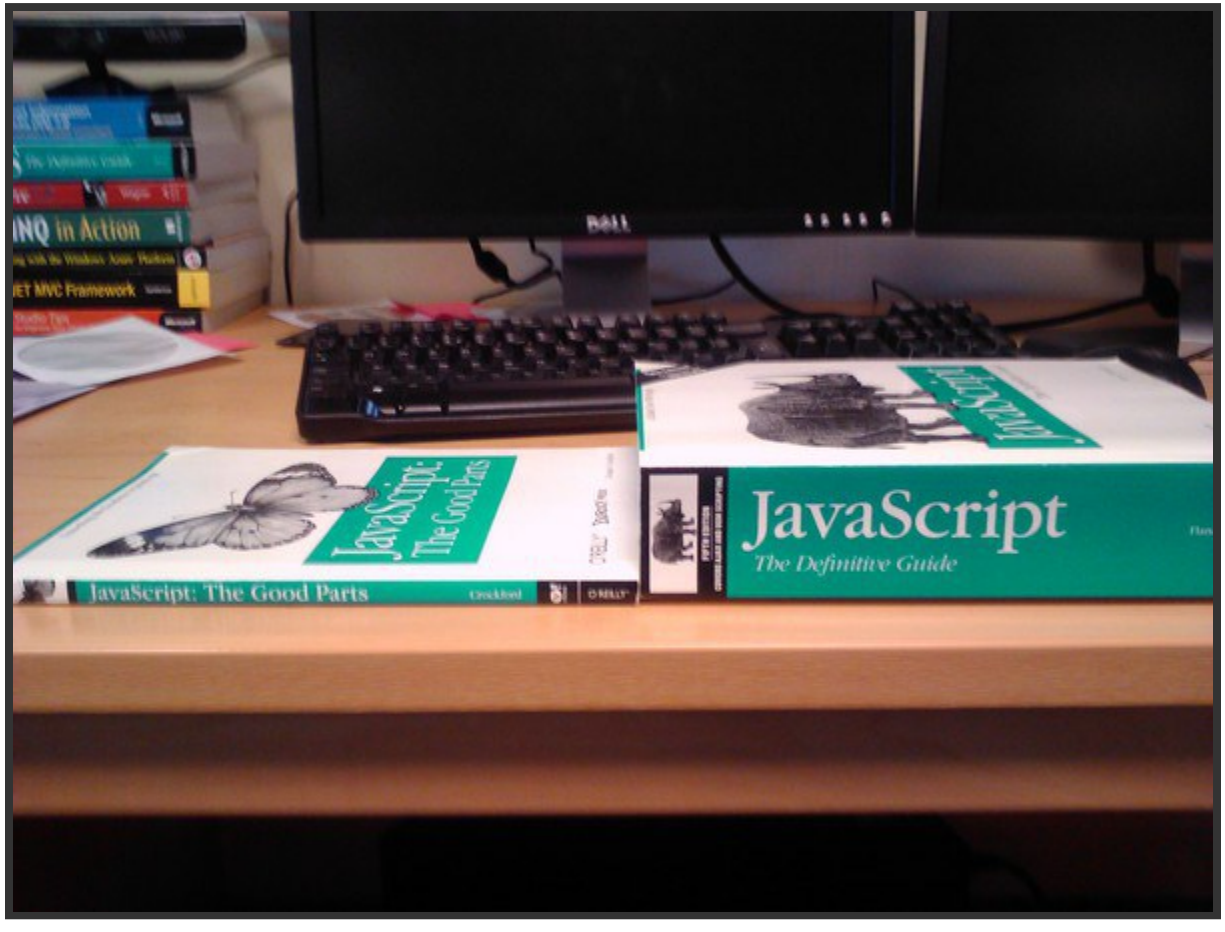


JAVASCRIPT

DEBUG

JAVASCRIPT

LE LANGAGE



JAVASCRIPT

UN LANGAGE FAIBLEMENT TYPÉ

- Le mot clé `var` permet de déclarer une variable
- Peu de type: `number`, `string`, `boolean`, `function`, `null`
- Une variable sans aucune valeur est `undefined`
- Les objets et les tableaux permettent de les composer
- Une variable n'a pas de type fixe (typage dynamique)

JAVASCRIPT

UN LANGAGE FAIBLEMENT TYPÉ

```
var foo;
console.log(foo); // 'undefined'
console.log(typeof foo); // 'undefined'

foo = 5;
console.log(typeof foo); // 'number'

foo = 'string';
console.log(typeof foo); // 'string'

foo = true;
console.log(typeof foo); // 'boolean'
```

JAVASCRIPT

LES COLLECTIONS

- Une seule structure de données: les tableaux
- Suite ordonnée d'éléments, chaque élément étant accessible par son index
- La propriété `length` donne la taille du tableau

```
var foo = [1, 2, 3];
console.log(foo); // [1, 2, 3]

foo.push('string', true);
console.log(foo); // [1, 2, 3, 'string', true]

console.log(foo.length); // 5
console.log(foo[4]); // true
console.log(foo[5]); // undefined
```

JAVASCRIPT

LES OBJETS

- Un objet est créé via une syntaxe simple : `var o = {};`
- Un objet est dynamique: on peut lui rajouter un attribut au runtime
- Ni plus, ni moins qu'une map clé valeur

JAVASCRIPT

LES OBJETS

```
var foo = {};
console.log(typeof foo); // 'object'
console.log(foo); // {}

foo = {
  id: 1
};

console.log(foo); // {id: 1}

foo.name = 'Mickael';
console.log(foo); // {id: 1, name: 'Mickael'}

foo.skills = ['Js', 'Java'];
console.log(foo.skills); // ['Js', 'Java']
console.log(foo['skills']); // ['Js', 'Java']
```

JAVASCRIPT

SPÉCIFICITÉS

- En JavaScript, on distingue égalité et égalité stricte
  - L'opérateur == (négation : !=) permet de comparer deux objets indépendamment du type
  - L'opérateur === (négation : !==) permet de comparer deux objets en prenant en compte les types
- Pour le cas des objets et des tableaux, c'est toujours une comparaison d'instance qui est faite !

JAVASCRIPT

SPÉCIFICITÉS

```
var foo = 1;
var bar = '1';
console.log(foo == bar);  // true
console.log(foo === bar); // false

console.log([1, 2, 3] == [1, 2, 3]);  // false
console.log([1, 2, 3] === [1, 2, 3]); // false
```

JAVASCRIPT

SPÉCIFICITÉS

Question : qu'affiche ce code ?

```
console.log('' == '0'); // ???
console.log('' == 0);   // ???
console.log('0' == 0);  // ???
```

JAVASCRIPT

SPÉCIFICITÉS

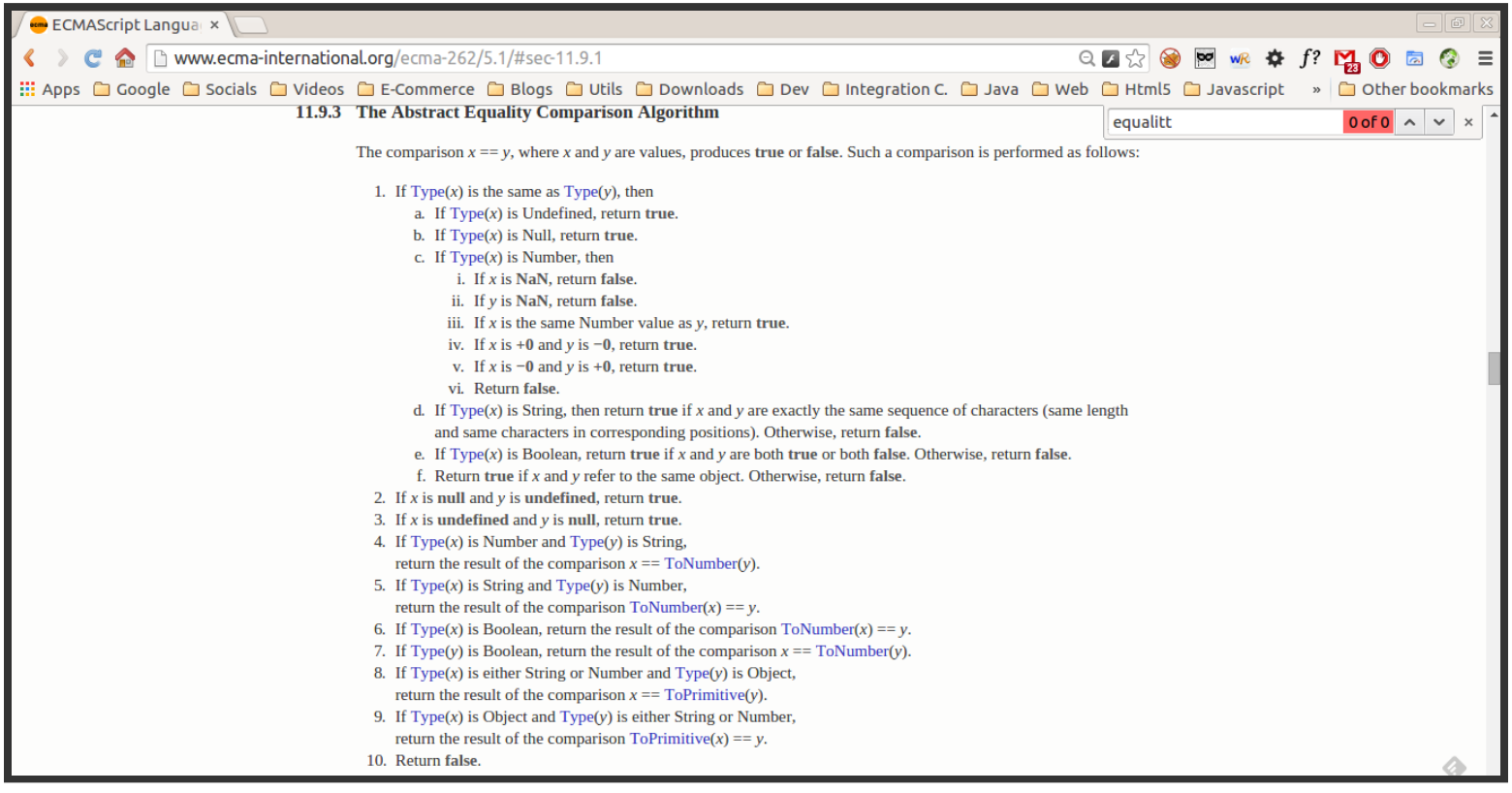
WTF ???

```
console.log('' == '0'); // false
console.log('' == 0);   // true
console.log('0' == 0);  // true
```

JAVASCRIPT

SPÉCIFICITÉS

http://www.ecma-international.org/ecma-262/5.1/#sec-11.9.1



JAVASCRIPT

SPÉCIFICITÉS

- Pour simplifier : JavaScript essaye de rapprocher les variables vers deux types identiques
- Cela peut amener à quelques spécificités un peu "étranges"
  - Opérateur non transitif
  - Opérateur non réflexif
- Simplifiez-vous la vie : **utilisez toujours l'égalité stricte !**

JAVASCRIPT

UN LANGAGE ORIENTÉ FONCTIONNEL

- En javascript, les fonctions sont partout
- Une fonction est un type comme un autre (objet de type `function`)
- Une fonction peut être nommée ou anonyme

```
// Fonction nommée "foo"
function foo() {
  console.log('hello world');
}

foo();

// Fonction anonyme qu'on affecte à une variable "foo"
var foo = function () {
  console.log('hello world');
};

foo();
```

JAVASCRIPT

UN LANGAGE ORIENTÉ FONCTIONNEL

- Une fonction a un **nombre de paramètre variable**
- Le mot clé `arguments` au sein d'une fonction permet de récupérer la liste des paramètres
- Une fonction peut ne rien retourner

```
// Fonction nommée "foo"
function foo(param1, param2) {
  console.log(param1, param2, 'arguments = ', arguments);

  if (param1 === 1) {
    return true;
  }
}

foo();          // undefined, undefinfed, arguments = []
foo(1);         // 1, undefinfed, arguments = [1]
foo(1, 2);      // 1, 2, arguments = [1, 2]
foo(1, 2, 3);   // 1, 2, arguments = [1, 2, 3]
```

JAVASCRIPT

UN LANGAGE ORIENTÉ FONCTIONNEL

- Une fonction peut être donnée en paramètre à une autre fonction
- Une fonction peut retourner une autre fonction
- En langage fonctionnel, de telles fonctions sont appelées **fonctions d'ordre supérieur**

JAVASCRIPT

UN LANGAGE ORIENTÉ FONCTIONNEL

```
// Fonction nommée "foo": cette fonction prend en paramètre
// une fonction à exécuter
function foo(func) {
  func();
}

foo(function () {
  console.log('hello world');
});

function myFunc() {
  console.log('hello world');
}

foo(myFunc);
```

JAVASCRIPT

UN LANGAGE ORIENTÉ FONCTIONNEL

```
// Fonction nommée "foo"
// Cette fonction renvoie une fonction permettant de "logguer" le
// contenu d'un tableau lorsqu'elle est exécutée
function add(op1) {
  return function (op2) {
    return op1 + op2;
  };
}

var r1 = add(5)(10);
console.log(r1); // 15

var addFn = add(5);
var r2 = addFn(10);
console.log(r2); // 15
```

JAVASCRIPT

UN LANGAGE ORIENTÉ FONCTIONNEL

Une fonction peut également être affectée à un **attribut d'un objet**

```
// On déclare un objet avec une fonction comme attribut
var batman = {
  name: 'Batman',
  speak: function () {
    console.log('I am Batman');
  }
};

console.log(batman); // {name: "Batman", speak: function}
console.log(typeof batman.speak); // 'function'

// Exécution de la fonction
batman.speak(); // 'I am Batman'
```

JAVASCRIPT

UN LANGAGE ORIENTÉ FONCTIONNEL

- Les fonctions sont le **coeur** du langage
- Avec Angular.js, on manipule des fonctions **tout le temps** :
  - Création d'un contrôleur
  - Création d'un service
  - Etc.

JAVASCRIPT

UN LANGAGE ORIENTÉ FONCTIONNEL

Comme beaucoup de langages fonctionnels, ECMAScript 5 rajoute des fonctions de manipulation de listes : `forEach`, `map`, `some`, `every`, `reduce` etc.

```
// Exécute une fonction sur chaque élément d'un tableau
[1, 2, 3].forEach(function (current, idx) {
  console.log(current, idx);
});

// Applique une transformation à chaque élément et retourne
// un nouveau tableau contenant tous les résultas
// Ex: Multiplication par deux de tous les éléments
var newArray = [1, 2, 3].map(function (current) {
  return current * 2;
});

console.log(newArray); // 2, 4, 6
```

JAVASCRIPT

UN LANGAGE ORIENTÉ FONCTIONNEL

```
// Vérifie une condition sur chaque élément d'un tableau
[2, 4, 6].every(function (current) {
  return current % 2 === 0;
});

// Vérifie qu'une condition est vérifié sur au moins un élément
[1, 2, 3].some(function (current) {
  return current % 2 === 0;
});

// Réduit le contenu du tableau à une valeur
var reduceValue = [1, 2, 3].reduce(function (memo, current) {
  return memo + current;
}, 0);

console.log(reduceValue); // 6
```

JAVASCRIPT

UN LANGAGE ORIENTÉ PROTOTYPE

- Le langage JavaScript n'est pas un langage orienté objet
- Mais, cela ne signifie pas qu'on ne peut utiliser des objets !
- Cela nécessite la manipulation de prototype
  - Compliqué, surtout quand on débute
  - ECMAScript 6 vient masquer cette complexité grâce à l'introduction des classes

JAVASCRIPT

UN LANGAGE ORIENTÉ PROTOTYPE

- Chaque objet dispose d'un `prototype`
- Un prototype n'est rien de plus qu'une liste de propriétés (a.k.a un objet) **partagé** par toutes les instances d'un même type

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

Hero.prototype = {
  speak: function () {
    console.log('I am ' + this.name);
  }
};

var batman = new Hero('Batman');     // I am Batman
var superman = new Hero('Superman'); // I am Superman
console.log(batman.speak === superman.speak); // true
```

JAVASCRIPT

ECMASCRIPT 6

- Introduit un ensemble de features visant à simplifier ECMAScript 5
- Beaucoup de sucre syntaxique au dessus d'ES5
  - Classes
  - Arrow Functions
  - Mots clés `let` et `const`
  - Etc.
- Mais pas que : modules, iterators, generators, etc.

JAVASCRIPT

ECMASCRIPT 6

Pourquoi apprendre ES6 ?

- Nouveau standard : **c'est l'avenir**
- Langage poussé sur les prochaines versions des librairies les plus populaires :
  - Angular 2
  - React
  - Twitter Bootstrap

JAVASCRIPT

ECMASCRIPT 6

Les classes

```
class Hero extends Human {
  constructor(name) {
    super();
    this.name = name;
  }

  speak() {
    console.log('I am ' + this.name);
  }
}

var batman = new Hero('Batman');
console.log(batman.speak()); // I am Batman
```

JAVASCRIPT

ECMASCRIPT 6

Arrow Functions : raccourci pour déclarer des fonctions anonymes

```
let array = [1, 2, 3];

array.forEach(x => console.log(x));
array.map(x => x + 1);
```


ANY
QUESTIONS
?

ECOSYSTÈME

FRAMEWORKS

ECOSYSTÈME

SINGLE PAGE APPLICATION

- Ces librairies visent à couvrir les besoins d'une application Web moderne :
  - Simplification des accès au DOM.
  - Mise en place d'une Single Page Application.
  - Interface avec une API REST.

ECOSYSTÈME

DOM

- DOM : Document Object Model.
- Page HTML vue sous la forme d'un arbre DOM.
- Le parcours de ce type d'arbre se fait en `javascript`.
  - Fastidieux.
  - Verbeux.
  - Différence entre les navigateurs.

ECOSYSTÈME

DOM

Exemple en pur javascript :

```
<div id="foo">
  <span class="message">Hello</span>
  <input name="inputName">
</div>
```

```
var div = document.getElementById('foo');
var span = document.getElementsByClassName('message')[0];
var input = document.querySelector('[name="inputName"]');
```

ECOSYSTÈME

DOM

Exemple avec jQuery :

```
<div id="foo">
  <span class="message">Hello</span>
  <input name="inputName">
</div>
```

```
var div = $('#foo');
var span = $('.message');
var input = $('[name="inputName"]');
```


ANY
QUESTIONS
?

ECOSYSTÈME

SINGLE PAGE APPLICATION

Wikipédia : Une application web monopage (en anglais single-page application ou SPA) est une application web accessible via une page web unique. Le but est d'éviter le chargement d'une nouvelle page à chaque action demandée, et de fluidifier ainsi l'expérience utilisateur.

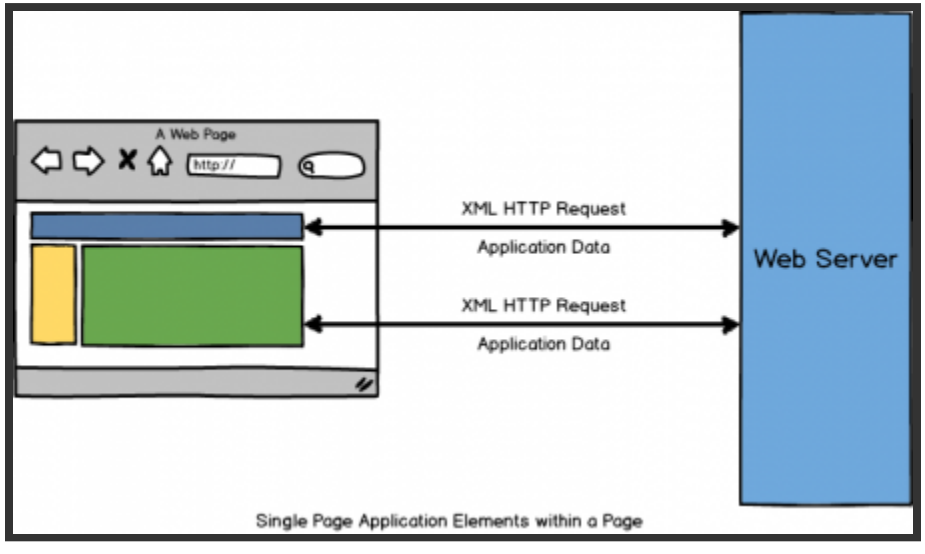
ECOSYSTÈME

SINGLE PAGE APPLICATION

- Dans une Single Page Application (SPA), la page html n'est jamais rechargée entièrement
- Seul le contenu dynamique, est mis à jour
- Exemples
  - Gmail
  - Facebook
  - Etc.

ECOSYSTÈME

SINGLE PAGE APPLICATION



ECOSYSTÈME

SINGLE PAGE APPLICATION

Problème : la gestion de l'historique !

ECOSYSTÈME

SINGLE PAGE APPLICATION

- Solutions:
  - Pendant longtemps, la solution a été d'utiliser le contenu situé aptès le caractère # (hash) : c'est la solution utilisée par Gmail
  - HTML5 standardise ce concept avec l'API push state : c'est la solution utilisée par Facebook
  - L'API push state est à préférer car c'est un standard et facilite l'indexation par les moteurs de recherches


ANY
QUESTIONS
?

ECOSYSTÈME

REST

- REpresentational State Transfer
- Standard d'échange entre un client et un serveur
- Style d'architecture
- Indépendant du protocole (http, etc.)
- Très rare de l'utiliser sur un autre protocole que http

ECOSYSTÈME

REST

- Doit respecter ces contraintes:
  - Client / Serveur
  - Stateless
  - Les échanges peuvent être mis en cache
  - Identification de resources

ECOSYSTÈME

REST

Exemple: HTTP

- Client / Serveur : OK par définition
- Stateless : OK par définition
- Les échanges peuvent être mis en cach : OK via les headers HTTP
- Identification de resources : OK via les URL

ECOSYSTÈME

REST

Exemple: Api Tweeter

- GET /tweets
- GET /tweets/1
- POST /tweets
- PUT /tweets/1
- DELETE /tweets


ANY
QUESTIONS
?

ECOSYSTÈME

BONNES PRATIQUES

- Pendant longtemps, le Javascript est resté très "artisanal"
  - Pas de tests automatisés
  - Pas de package manager
- Depuis quelques années, la communauté a développé tous les outils nécessaires !
- Ces outils sont basés sur node.js

ECOSYSTÈME

BONNES PRATIQUES

- En 2015, avant de livrer une application en production, il faut la "builder"
  - Concaténer ses fichiers javascript en un seul fichier
  - Minifier son code Javascript
  - Gzipper ses fichiers
  - Jouer les tests unitaires pour s'assurer de la stabilité du code
- Le but est de garantir la non régression, de réduire le poids de la page, et donc améliorer l'expérience utilisateur !
- Necessité **d'automatiser** ces tâches

ECOSYSTÈME

BONNES PRATIQUES

ECOSYSTÈME

BONNES PRATIQUES

ECOSYSTÈME

BONNES PRATIQUES


ANY
QUESTIONS
?

ANGULAR.JS

BY GOOGLE

- Angular.js est un framework javascript "full stack"
- Développé par Google (Miško Hevery, Igor Minar, Vojta Jina, etc.)
- Version 1.5
- Open source: https://github.com/angular/angular.js
- 43 000 stars sur GitHub

ANGULAR.JS

INTRODUCTION

- Data Binding
- Single Page Application (SPA)
- REST
- Dependency Injection (DI)
- MVC

ANGULAR.JS

INTRODUCTION

DEMO

ANGULAR.JS

FONCTIONNEMENT

- Les données à afficher à l'écran sont définies sur un objet `$scope`.
- Lors d'un événement, la phase de digest va comparer l'état des variables (dirty checking).
- Les watchers permettent d'appliquer les modifications dans le DOM.

ANGULAR.JS

FONCTIONNEMENT

Les watchers

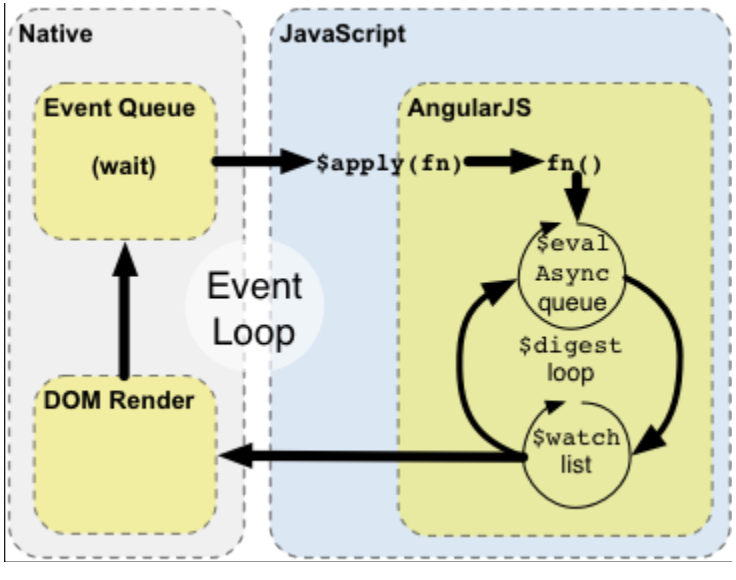
- Simple fonctions exécutées pendant la phase de digest.
- Permettent de déclencher un traitement à la modification d'une variable.
- C'est le coeur d'angular.
- Permet d'implémenter le data-binding.

```
$scope.$watch('foo', function(newValue, oldValue) {
  element.innerHTML = newValue;
});
```

ANGULAR.JS

FONCTIONNEMENT

- Les watchers sont exécutées lors de la phase de "digest"



ANGULAR.JS

FONCTIONNEMENT

Le code de la fonction $apply est en fait très simple :

```
$scope.$apply = function(fn) {
  try {
    fn();
  } finally {
    this.$digest();
  }
};
```

ANGULAR.JS

FONCTIONNEMENT

- La fonction `$apply` est, au final, très simple :
  - Exécution d'une fonction.
  - Déclenchement de la phase de digest.
- La fonction donnée en paramètre est généralement fournie par les directives :
  - Doit rester simple.
  - Ex: Mise à jour de variables dans le scope.

ANGULAR.JS

FONCTIONNEMENT

Exemple d'appel : mise à jour d'une variable sur un événement du DOM

```
element.on('keyup', function(e) {
  $scope.$apply(function() {
    $scope.name = e.target.value;
  });
});
```

ANGULAR.JS

FONCTIONNEMENT

- La fonction `digest` consiste à :
  - Parcourir les différents watchers.
  - Exécuter la fonction `listener` en cas de changement.
- C'est cette fonction qui implémente la "magie" d'Angular.

ANGULAR.JS

FONCTIONNEMENT

Zoom sur la fonction digest :

```
$scope.$digest = function() {
  this.$$watchers.forEach(function(watcher) {
    var oldValue = watcher.last;
    var newValue = watcher.watcherFn();

    watcher.last = newValue;

    if (oldValue !== newValue) {
      watcher.listenerFn();
    }
  });
};
```

ANGULAR.JS

FONCTIONNEMENT

- L'ajout de watchers se fait via la fonction `$watch` :
  - Le premier paramètre est le nom de la variable à observer.
  - Le deuxième paramètre est la fonction listener à exécuter lors d'un changement.
- La fonction listener va permettre de déclencher du traitement à chaque mise à jour : mise à jour du DOM.
- A inspirer l'api `Object.observe` qui sera disponible avec ECMAScript 7.

ANGULAR.JS

FONCTIONNEMENT

- Son implémentation est également assez simple : ajout du watchers dans une collection !

```
$scope.$$watchers = [];

$scope.$watch = function(getter, listener) {
  this.$$watchers.push({
    watcherFn: getter,
    listenerFn: listener
  });
};
```

ANGULAR.JS

FONCTIONNEMENT

Combinées ensemble, ces trois fonctions sont le coeur d'Angular.js

```
element.addEventListener('keyup', function(e) {
  $scope.$apply(function() {
    $scope.foo = e.target.value;
  });
});
```

```
$scope.$watch('foo', function(newValue) {
  element.innerHTML = newValue;
});
```

ANGULAR.JS

FONCTIONNEMENT

- C'est l'utilisation conjointe de ces trois fonctions qui permettent la "magie" d'Angular.js
- *C'est ce que nous allons implémenter dans l'exercice #1*


ANY
QUESTIONS
?

ANGULAR.JS

EXERCICE #1

- Implémenter la fonction `$apply` (todo #1).
- Implémenter la fonction `$digest` (todo #2).
- Implémenter le watcher de la directive `ng-bind` (todo #3).
- Implémenter l'appel à `$apply` de la directive `ng-model` (todo #4).

ANGULAR.JS

COMPOSANTS

- Modules
- Contrôleurs
- Formulaires
- Services et factory
- Filtres
- Directives
- Routeur

ANGULAR.JS

MODULE

- Contient les différents composants de votre application
- Toute application contient au moins un module
- On le déclare dans le code (avec l'attribut ng-app) et dans un fichier Javascript

```
// Création d'un module dans app.js
angular.module('myApp', []);
```

ANGULAR.JS

MODULE

- Equivalent d'un "main" (point d'entrée de votre application)
- Tous les composants de votre application seront créés au sein de votre module
- Le deuxième paramètre est la liste des modules dont votre application dépend
- Pour récupérer un module sans le créer, il suffit de ne pas préciser le second paramètre

ANGULAR.JS

MODULE

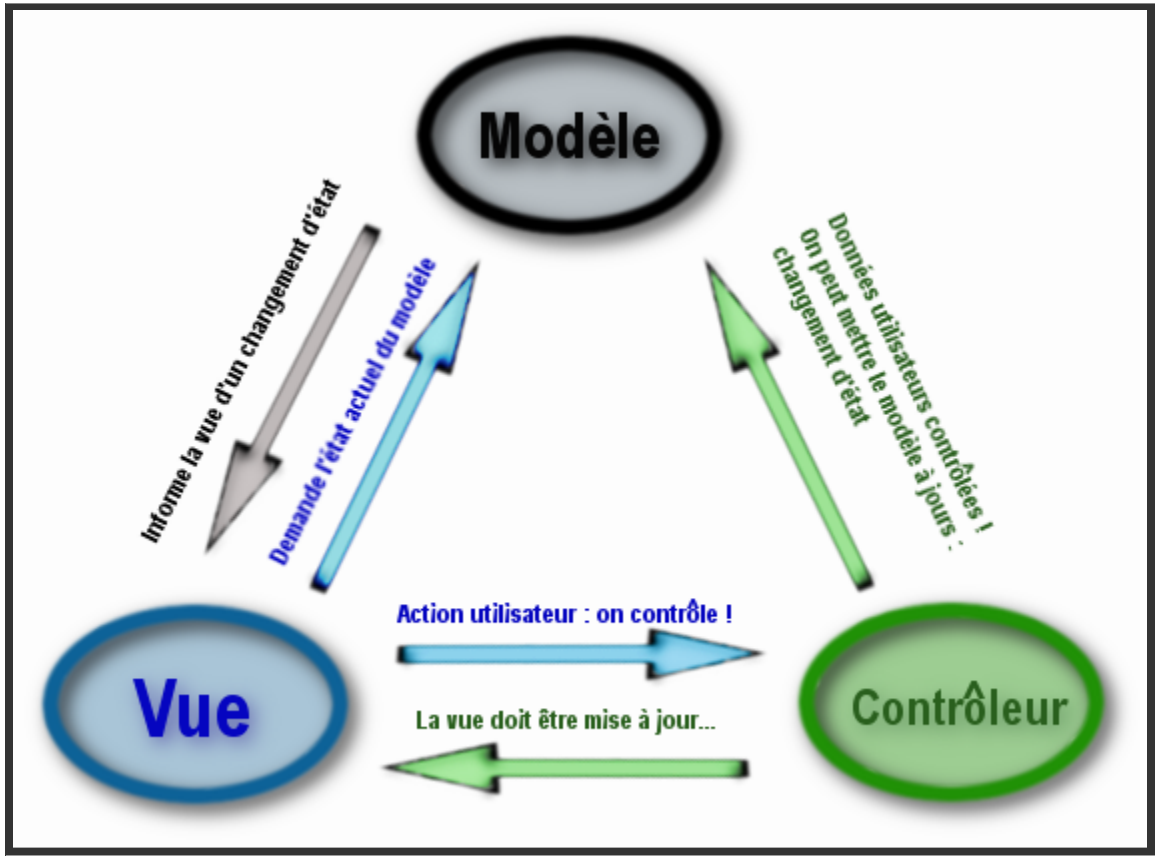
```
// Création d'un module
var m1 = angular.module('myApp', []);

// Récupération du module
var m2 = angular.module('myApp');

console.log(m1 === m2); // true
```

ANGULAR.JS

CONTRÔLEURS



ANGULAR.JS

CONTRÔLEURS

- Un contrôleur contient la logique (le comportement) associée à votre page HTML (qui représente la vue du modèle M**V**C)
- C'est un contrôleur conforme au pattern MV**C**
- Un contrôleur angular.js n'est rien d'autre qu'une fonction !

```
<div ng-controller="MyController">
  <p>Ma page HTML</p>
</div>
```

```
angular.module('myApp')
  .controller('MyController', function () {
    // Le code de votre contrôleur !
  });
```

ANGULAR.JS

CONTRÔLEURS

- Pour afficher des données gérées dans votre contrôleur, il faut les partager à la vue
- Pour partager des données, on utilise l'objet $scope
- C'est le M du modèle MVC

```
<div ng-controller="MyController">
  <p>Hello {{ name }}</p>
</div>
```

```
angular.module('myApp')
  .controller('MyController', ['$scope', function ($scope) {
    $scope.name = 'world';
  }]);
```

ANGULAR.JS

CONTRÔLEURS

- L'objet $scope a été instancié par Angular.js : notre application ne crée pas d'objets, mais **demande à Angular.js de nous les fournir**
- C'est **l'injection de dépendance**
- C'est un concept qu'on retrouve dans beaucoup d'autres langages et frameworks
  - Java: Spring, CDI
  - PHP: Symfony

ANGULAR.JS

$SCOPE

- L'objet $scope est un objet très **important**
  - Il contient les données accessible depuis la page HTML
  - Il hérite de ses parents
  - Il possède les méthodes `$watch` ; `$on` ; `$apply` ; `$digest`
  - Tout objet $scope hérite forcément du **$rootScope**
  - Toute application Angular.js possède **un (et un seul) $rootScope**

ANGULAR.JS

$SCOPE

Chaque $scope hérite de son scope parent (défini par l'arborescence du dom)

```
<div ng-controller="MyController">
  <div ng-controller="MyOtherController">
  </div>
</div>
```

```
angular.module('myApp')
  .controller('MyController', ['$scope', function ($scope) {
    $scope.name = 'world';
  }]);

angular.module('myApp')
  .controller('MyOtherController', ['$scope', function ($scope) {
    console.log($scope.name); // world
  }]);
```

ANGULAR.JS

$SCOPE

- Pour afficher les valeurs du scope, le plus simple est d'utiliser les "doubles moustaches"
- Une autre façon est d'utiliser la directive **ng-bind**

```
<div ng-controller="MyController">
  Hello {{name}}
  Hello <span ng-bind="name"></span>
</div>
```

```
angular.module('myApp')
  .controller('MyController', ['$scope', function ($scope) {
    $scope.name = 'world';
  }]);
```

ANGULAR.JS

$SCOPE

- La méthode $on permet d'écouter un évenement provenant d'un autre contrôleur
- Permet de "dispatcher" des événements
- Une façon de faire communiquer des contrôleurs (mais pas toujours la bonne façon)

```
angular.module('myApp')
  .controller('myController', ['$scope', '$rootScope',
    function ($scope, $rootScope) {
      $scope.$on('myEvent', function (event, param1) {
        console.log(param1);
      });

      $rootScope.$broadcast('myEvent', 'foo');
    }
  ]);
```

ANGULAR.JS

CONTROLLER AS

- Avec l'arrivée d'Angular 2, la syntaxe controllerAs est préconisée
- Les données sont précisées sur le contrôleur
- Le contrôleur est vu comme une classe

```
<div ng-controller="MyController as ctrl">
  Hello {{ctrl.name}}
  Hello <span ng-bind="ctrl.name"></span>
</div>
```

```
angular.module('myApp')
  .controller('MyController', function() {
    this.name = 'world';
  });
```

ANGULAR.JS

FORMULAIRES & NG-MODEL

- Pour récupérer les valeurs saisies dans un formulaire, on utilisera l'attribut **ng-model**
- Au fur et à mesure de la saisie, les variables du scope sont mises à jour par Angular.js
- Si la valeur est mise à jour programmatiquement, la valeur du champ est mise à jour
- C'est ce qu'on appelle le **double binding**

ANGULAR.JS

FORMULAIRES & NG-MODEL

Exemple

```
<div ng-controller="MyController as ctrl">
  <form>
    <input type="text" ng-model="ctrl.name">
    Message: {{ ctrl.name }}
  </form>
</div>
```

```
angular.module('myApp')
  .controller('MyController', function() {
    this.name = 'hello world';
  });
```

ANGULAR.JS

FORMULAIRES & NG-MODEL

La directive **ng-model** est l'une des features les plus importantes d'Angular.js

```
<div ng-controller="MyController as ctrl">
  <input id="my-input" type="text" ng-model="ctrl.name">
</div>
```

VS

```
var myValue = 'foo';

$('#my-input').on('keyup', function (e) {
  myValue = $(this).val();
});

myValue = 'bar';
$('#my-input').val(myValue);
```

ANGULAR.JS

FORMULAIRES & NG-MODEL

Permet de faire de la validation à partir des attributs "classiques" : `required`, `maxlength`, `pattern`, etc.

```
<div ng-controller="MyController as ctrl">
  <input id="my-input" type="text" name="inputName"
         ng-model="ctrl.name"
         required
         maxlength="100">
</div>
```

ANGULAR.JS

HTTP

- Angular.js fournit une api pour faire des requêtes HTTP
- S'interface parfaitement avec une **api REST**
- Prend en charge toutes les spécificités des navigateurs
- Il suffit d'injecter le service **$http** dans un contrôleur
- Permet de faire toutes les requêtes "classique"
  - GET
  - POST
  - PUT
  - DELETE

ANGULAR.JS

HTTP

Exemple : `GET /foo`

```
angular.module('myApp')
  .controller('MyController', ['$http',
    function ($http) {
      var vm = this;

      $http.get('/foo')
        .success(function (data) {
          vm.data = data;
        })
        .error(function (error) {
          vm.error = error;
        });
    }
  ]);
```

ANGULAR.JS

HTTP

- **Attention**, le résultat est **asynchrone**
  - Communication synchrone: vous demandez un résultat et l'avez "tout de suite" (i.e vous êtes bloqué tant que la réponse n'est pas là)
  - Communication asynchrone: vous demandez un résultat, mais vous pouvez passez à autre chose et vous serez averti de la réponse une fois disponible
- On récupère le résultat via un **callback** de succès
- On récupère l'erreur via un **callback** d'erreur

ANGULAR.JS

HTTP

Exemple de récupération des tweets :

```
<div ng-controller="TweetController as ctrl">
  <div ng-repeat="tweet as ctrl.tweets">{{ tweet.message }}</div>
</div>
```

```
angular.module('app').controller('TweetController', function($http) {
  var vm = this;

  $http.get('/tweets')
    .success(function(data) {
      vm.tweets = data;
    });
});
```

ANGULAR.JS

HTTP

Exemple de création d'un tweet :

```
<div ng-controller="TweetController as ctrl">
  <form name="tweetForm" ng-submit="ctrl.submit()">
    <input ng-model="ctrl.login" required>
    <textarea ng-model="ctrl.message" maxlength="140"></textarea>
    <button ng-disabled="tweetForm.$invalid">Tweeter !</button>
  </form>
</div>
```

```
angular.module('app').controller('TweetController', function($http) {
  var vm = this;

  vm.submit = function() {
    var tweet = { login: vm.login, message: vm.message };
    $http.post('/tweets', tweet)
      .success(function() {
        console.log('Success !');
      });
  };
```

ANGULAR.JS

FACTORY

- Une factory permet d'extraire des composants ré-utilisables
- On peut le récupérer dans un contrôleur par injection de dépendance

ANGULAR.JS

FACTORY

```
angular.module('myApp')
  .factory('myComponent', function () {
    return {
      helloWorld: function () {
        console.log('hello world');
      }
    };
  })

  .controller('MyController', ['myComponent',
    function (myComponent) {
      this.message = myComponent.helloWorld();
    }
  ]);
```

ANGULAR.JS

FACTORY

- La factory est exécutée la première fois où le composant est injecté (lazy initialization).
- Le composant ne sera créé qu'une et une seule fois (singleton).
- La solution idéale pour factoriser du code entre contrôleurs

ANGULAR.JS

FILTRES

- Un filtre est un composant permettant d'altérer l'affichage d'une valeur
- Permet de bien séparer l'affichage de la manipulation des données
- Un exemple est l'utilisation du filtre "date"

```
<div ng-conroller="MyController as ctrl">
  Current date: {{ ctrl.myDate | date:'dd/MM/yyyy' }}
</div>
```

```
angular.module('myApp')
  .controller('MyController', function () {
    this.myDate = new Date();
  });
```

ANGULAR.JS

FILTRES

- Angular fournit une collection de filtres prêts à l'emploi
  - `date`
  - `number`
  - `currency`
  - `uppercase`/`lowercase`
  - `orderBy`
- La documentation est très bien faite !

ANGULAR.JS

FILTRES

Les filtres peuvent se chaîner (comme un pipe unix)

```
<div ng-conroller="MyController as ctrl">
  Current date: {{ ctrl.myDate | date:'dd/MMMMM/yyyy' | uppercase }}
</div>
```

```
angular.module('myApp')
  .controller('MyController', function () {
    this.myDate = new Date();
  });
```

ANGULAR.JS

FILTRES

Ecrire son propre filtre consiste "juste" à écrire une fonction

```
<div ng-conroller="MyController as ctrl">
  {{ ctrl.myData | uppercase }}&gt;
</div>
```

```
angular.module('myApp')
  .controller('MyController', function () {
    this.myData = 'hello';
  })

  .filter('uppercase', function () {
    return function (param) {
      return param.toUpperCase();
    };
  });
```

ANGULAR.JS

FILTRES

Cas d'utilisation : affichage d'un login "twitter" :

```
<div ng-conroller="MyController as ctrl">
  {{ ctrl.tweet.login | twitterLogin }}
</div>
```

```
angular.module('myApp')
  .controller('MyController', function () {
    this.tweet = {
      login: 'mjeanroy',
      message: 'Hello Insa'
    };
  })

  .filter('twitterLogin', function () {
    return function (login) {
      return '@' + login;
    };
  });
```

ANGULAR.JS

DIRECTIVES

- Les directives représentent une extention du DOM : attribut / balise customisé
  - Permettent de centraliser la manipulation du DOM
  - Concept très proche des WebComponents
- Peut être utilisée comme un attribut ou une nouvelle balise
- Les attributs `ng-controller` / `ng-app` / `ng-model` sont des directives !

ANGULAR.JS

DIRECTIVES

Exemple de la directive ng-bind :

```
module.directive('ngBind', function () {
  return function (scope, element, attrs) {
    scope.$watch(attrs.ngBind, function (newValue) {
      element.html(newValue);
    });
  };
});
```

ANGULAR.JS

DIRECTIVES

- Les directives peuvent avoir un template html
  - Pratique pour séparer logique et présentation.
- Les directives disposent d'un scope :
  - Permet d'implémenter le data binding facilement !

ANGULAR.JS

DIRECTIVES

Exemple d'une directive custom :

```
<div ng-controller="TweetController as ctrl">
  <tweet tweet="ctrl.activeTweet"></tweet>
</div>
```

```
angular.module('app').directive('tweet', function () {
  return {
    template: '<div>{{tweet.login}} - {{tweet.message}}</div>',
    scope: {
      tweet: '='
    }
  };
});
```

ANGULAR.JS

DIRECTIVES

- Sujet vaste et compliqué qu'on n'abordera pas plus aujourd'hui
- Nombreux exemples sur Github (angular-ui, angular-strap)
- Beaucoup de directives Angular.js (ngBind, ngModel, etc.) sont simples à lire et sont très instructives

ANGULAR.JS

ROUTER

- Le routeur Angular.js est un composant très simple
- Il suffit de définir les routes disponibles et les templates html associés
- La page doit contenir un élément avec un attribut `ng-view`
- C'est ce bloc qui sera mis à jour à chaque changement de route

ANGULAR.JS

BEST PRACTICES

- Pas de manipulation du DOM dans un contrôleur : c'est le rôle d'une directive !
- La customisation de l'affichage doit être fait dans un filtre
- Gardez vos contrôleurs simple et "petit"
- Utiliser des services pour factoriser du code et extraire le code métier des contrôleurs
- **Faites des tests unitaires** : jasmine + karma = :)

ANGULAR.JS

BEST PRACTICES

- Evitez les composants trop complexe : principe de responsabilité unique
- Performances : évitez les pages trop lourdes


ANY
QUESTIONS
?

ANGULAR.JS

EXERCICE #2

- Récupérer le zip du tp
- Lancer le serveur en lançant la commande `node server/app.js`
- Enregistrer son login
- Valider avec moi l'exercice
- Récupérer le mot de passe de validation !

ANGULAR.JS

EXERCICE #2

1. Récupérer les tweets en faisant un GET sur `/tweets`
2. Brancher le post d'un tweet
3. Ecrire un filtre pour afficher le login précédé du caractère @
4. Capter l'événement 'tweet:new' pour afficher les tweets en temps réel
5. Refactorer l'affichage du tweet en utilisant une directive

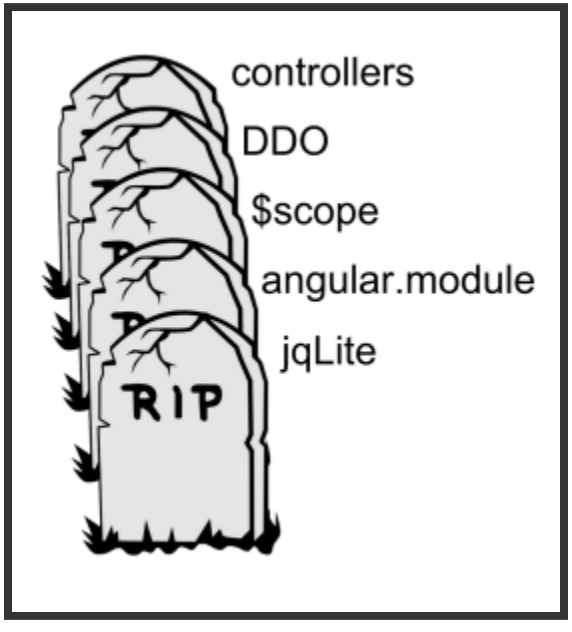
CONCLUSION

ANGULAR 2

- Angular 2 a été annoncé il y a environ un an.
- Ecrit en `typescript`
- Va changer beaucoup de choses :
  - Plus de contrôleurs & scopes.
  - Plus de modules (ce seront des modules ES6).
  - Plus de dirty checking (`Object.observe` !).
  - Tout sera directive !

CONCLUSION

ANGULAR 2



CONCLUSION

ANGULAR 2

- Disponible en version alpha
- Open Source
- https://github.com/angular/angular
- https://angular.io

CONCLUSION

ANGULAR 2

- Faut-il migrer tout de suite ?
  - Pas forcément...
  - Angular 1.X maintenue jusqu'en 2018 / 2019.
  - La communauté prendre certainement le relai.
- Pas encore de version stable pour Angular 2 : les premières versions seront-elles production-ready ?

CONCLUSION

ANGULAR 2

- Comment migrer progressivement ?
  - En adoptant les bonnes pratiques d'Angular 1.X (directives, component design).
  - En suivant progressivement les mises à jour de la versions 1.X !
  - Utilisation du module `ngUpgrade` pour une migration en douceur.

CONCLUSION

AUTOUR D'ANGULAR

- Angular.js n'est que le début :
  - ECMAScript 6.
  - Outils de builds (`gulp`, `bower`).
  - Tests Unitaires (`karma`, `jasmine`).
  - Etc.
- Le développement Web évolue tous les jours !

CONCLUSION

AUTOUR D'ANGULAR

- De nouvelles librairies apparaissent tous les jours :
  - React (by Facebook)
  - AureliaJS
  - EmberJS
- Chaque librairie apporte son lot d'idées intéressantes.
- Chacune est adaptée à un besoin particulier : il n'y a pas de "silver bullet" :
  - Indexation ?
  - Accessibilité ?
  - Performance ?

CONCLUSION

"Software is eating the world, the web is eating software, and JavaScript rules the web."

Eric Elliot

https://medium.com/javascript-scene/javascript-training-sucks-284b53666245


ANY
QUESTIONS
?