

# Testabilité des applications JavaScript avec AngularJS



Romuald Coeffier  
@romualdcoeffier

## Testabilité => écrire du code testable

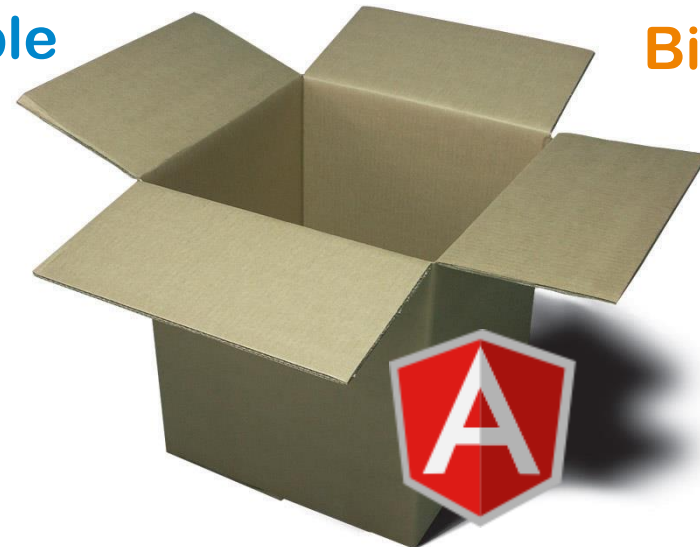
De mon expériences personnelles (Java/Spring et Flex/Swiz), les bons reflexes pour obtenir la testabilité sont:

- l'injection de dépendances
- les patterns d'ui MVVM ou MVP (binding)

Prévu pour écrire  
du code testable

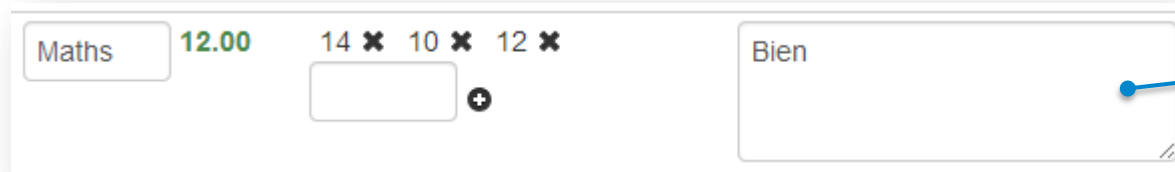
Injection de  
dépendances

Binding



AngularJS: framework JS pour les  
applications web single page

# Notre "application" jouet



The screenshot shows a web application interface for a subject. It includes a text input field containing "Maths", a green display of "12.00", a list of marks "14 ✖ 10 ✖ 12 ✖", an empty input field with a "+" button, and a larger text area containing "Bien". A blue arrow points from the "Subject:" list to the "Bien" text area.

## Subject:

- name
- average
- marks
- comments
- **addMark()**
- **removeMark()**

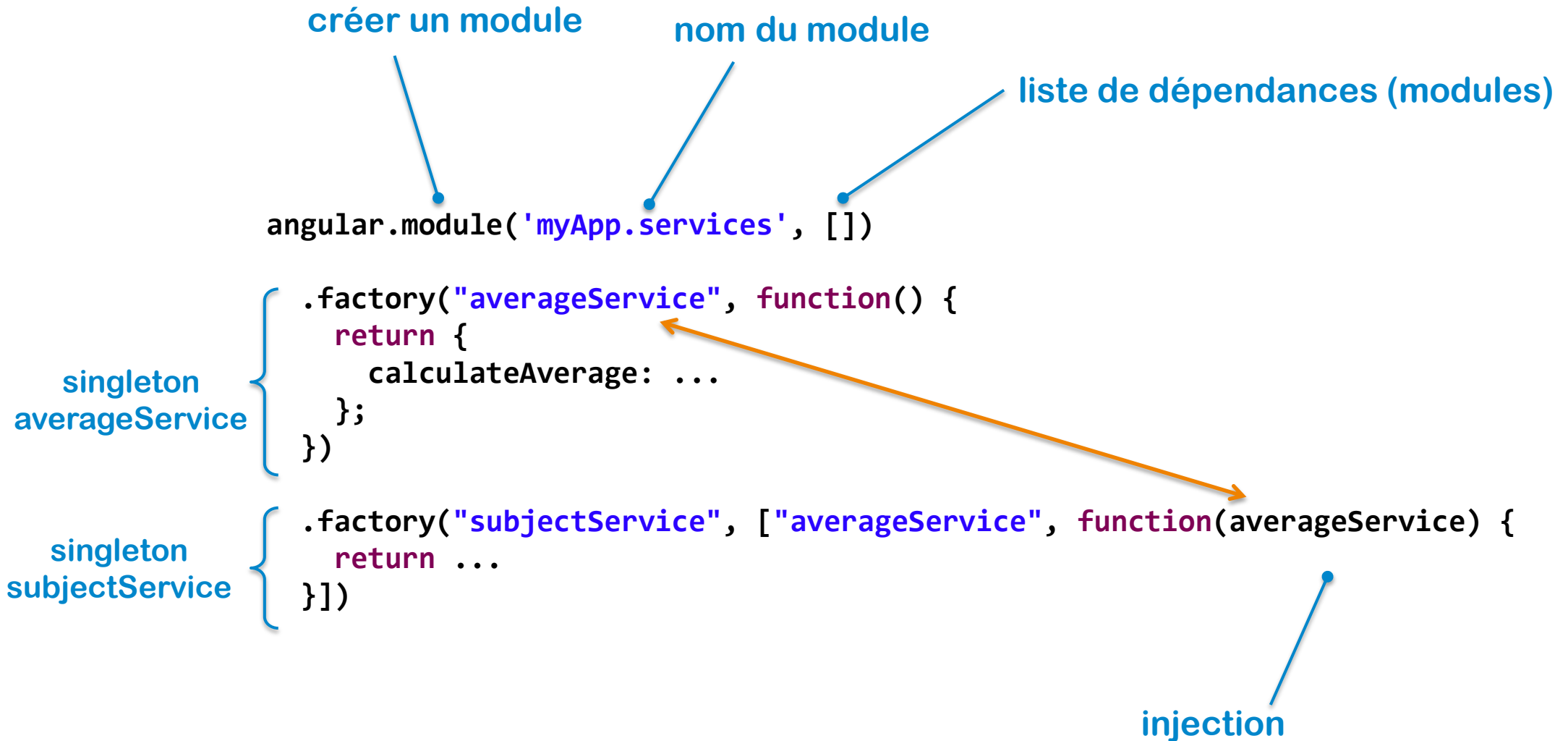
subjectService: ----->

- createSubject()

averageService:

- calculateAverage()

# Modules et injection de dépendances

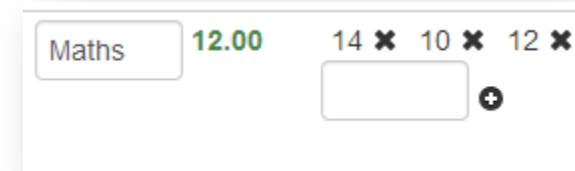


```
describe('service', function() {  
  beforeEach(module('myApp.services'));  
  
  describe('subject', function() {  
  
    it('test subject after creation', inject(function(subjectService) {  
      var subject = subjectService.createSubject('aSubjectName');  
      expect(subject.name).toEqual('aSubjectName');  
      expect(subject.marks).toEqual([]);  
      expect(subject.average).toEqual(null);  
      expect(subject.comments).toEqual('');  
    }));  
  });  
});
```

instanciation d'un module  
pour les tests

injection

```
<div ng-controller="SubjectController">  
  <input type="text" ng-model="subject.name">  
  <strong>{{subject.average}}</strong>  
  ...  
</div>
```



The screenshot shows a web application interface. It features a text input field containing the word "Maths". To its right is a numeric input field displaying "12.00". Further right are three small buttons, each with a number and a close icon (X): "14 X", "10 X", and "12 X". Below these buttons is a larger empty text input field with a plus icon (+) to its right.

### La vue:

- est décrite en HTML
- contient des **expressions** `{{...}}`
- contient des **directives** `ng-xxxx` (éléments, attributs, ...)

### Angular:

- ne travaille pas sur le code HTML
- **travaille directement sur l'arbre DOM**

Le rôle d'Angular est d'interpréter les expressions et directives de l'arbre DOM pour les remplacer par les données d'un objet \$scope.

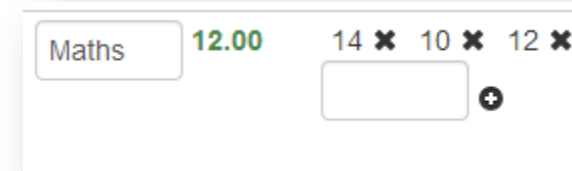
```
<div ng-controller="SubjectController">
  <input type="text" ng-model="subject.name">
  <strong>{{subject.average}}</strong>
  ...
</div>
```

binding

binding  
bidirectionnel

```
{
  name: 'Maths',
  average: 12,
  marks: [14, 10, 12]
}
```

```
$scope.subject = subjectService.xxxx();
```



L'objet \$scope contient:

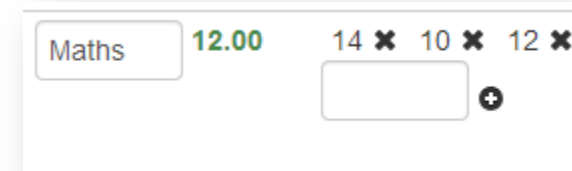
- le modèle
- l'état **complet** de la vue
- les méthodes pour traiter les actions de l'utilisateur

C'est le \$scope qui commande.

On ne code pas des opérations DOM depuis le scope (c'est le travail d'Angular).

```
<div ng-controller="SubjectController">
  <input type="text" ng-model="subject.name">
  <strong class="average-ok-{{averageOk}}>
    {{subject.average}}
  </strong>
  <input type="text" ng-model="newMark">
  <button ng-click="addMarkHandler()">
    ...
</div>
```

```
$scope.subject = subjectService.xxxx();
$scope.averageOk = false;
$scope.newMark = '';
$scope.addMarkHandler = function() {
  $scope.subject.addMark($scope.newMark);
}
```



L'objet \$scope contient:

- le modèle
- l'état **complet** de la vue
- les méthodes pour traiter les actions de l'utilisateur



```
<div ng-controller="SubjectController">  
  ...  
</div>
```

C'est dans un contrôleur que l'on définit le contenu d'un \$scope.

```
angular.module('myApp.controllers', ['myApp.services'])  
  .controller('SubjectController', ['$scope', subjectService',  
    function ($scope, subjectService) {  
      $scope.subject = subjectService.xxxx();  
      $scope.averageOk = false;  
      $scope.newMark = '';  
      $scope.addMarkHandler = function() {  
        $scope.subject.addMark($scope.newMark);  
      }  
      ...  
    }])
```

## Constations:

- les tests sur le DOM sont très couteux à écrire et à maintenir (ROI < 0)
- quelque soit le framework, on peut assez facilement tester:
  - les classes métier
  - les services
  - les utilitaires
- avec l'utilisation du binding => pas de manipulation DOM dans les contrôleurs

=> testabilité des contrôleurs (et des scopes associés)

```
it('test average state', function() {  
  // scope.subject.marks: [9]  
  expect(scope.averageOk).toBe(false);  
  scope.newMark = '12';  
  scope.addMarkHandler();  
  expect(scope.averageOk).toBe(true);  
});
```

AngularJS <http://angularjs.org/>

Angular-seed <https://github.com/angular/angular-seed>

Jasmine <http://pivotal.github.io/jasmine/>

Karma <http://karma-runner.github.io/>

Node.js <http://nodejs.org/>

Source

<https://github.com/romualdcoeffier/htg-angularjs-testabilite.git>

Questions ?