APPMOB - AngularJS

Olivier Liechti & Simon Oulevay
COMEM Applications Mobiles

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Why AngularJS?

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

HTML is great for declaring static documents, but it falters when we try to use it for declaring dynamic views in web-applications. AngularJS lets you extend HTML vocabulary for your application. The resulting environment is extraordinarily expressive, readable, and quick to develop.

AngularJS: Directives & The Scope

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Angular Directives:
new dynamic HTML vocabulary.

Angular Controllers: instead of manipulating the view, work with the $scope.

```
<ul ng-controller="PeopleController">
  <li ng-repeat="person in people">
    {{ person.firstName }} {{ person.lastName }}
  </li>
</ul>
```

```
myApp.controller("PeopleController", function($scope) {
  $scope.people = [
    { firstName: "John", lastName: "Doe" },
    { firstName: "John", lastName: "Smith" }
  ];
});
```

Live example:

http://codepen.io/AlphaHydrae/pen/EadXWp/

AngularJS: Two-Way Binding

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

With the ngModel directive, it works the other way too!

```
<form ng-controller="FormController">
  <input type="text" ng-model="firstName" />
</ul>
```

```
myApp.controller("FormController", function($scope) {
  $scope.firstName = "John";
});
```

The binding goes two ways:

- if the user types in the field, the $scope variable is updated;
- if the $scope variable changes, the field is updated.

Live example:

http://codepen.io/AlphaHydrae/pen/YPJPOx/

AngularJS: Main Components

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Controllers

```
myApp.controller("PeopleController", function(PeopleService, $scope) {
  $scope.people = PeopleService.getPeople();
});
```

Services

```
myApp.factory("PeopleService", function() {
  return {
    getPeople: function() {
      return [
        { firstName: "John", lastName: "Doe" },
        { firstName: "John", lastName: "Smith" }
      ];
    }
  };
});
```

Templates

```
<ul ng-controller="PeopleController">
  <li ng-repeat="person in people">
    {{ person.firstName }} {{ person.lastName }}
  </li>
</ul>
```

Filters

```
myApp.filter("upcase", function() {
  return function(input) {
    return input.toUpperCase();
  };
});
```

```
<ul ng-controller="PeopleController">
  <li ng-repeat="person in people">
    {{ person.firstName }} {{ person.lastName | upcase }}
  </li>
</ul>
```

Directives

```
myApp.directive("personName", function() {
  return {
    type: "E",
    scope: {
      person: "="
    },
    template: "{{ person.firstName }} {{ person.lastName | upcase }}"
  };
});
```

```
<ul ng-controller="PeopleController">
  <li ng-repeat="person in people">
    <person-name person="person" />
  </li>
</ul>
```

AngularJS: Initialization

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Constants

```
myApp.constant("apiUrl", "https://api.example.com");
```

You can use constants to store reusable information. They can be injected into controllers, services, etc.

Config Blocks

```
myApp.config(function($logProvider) {
  $logProvider.setDebugEnabled(true);
});
```

Config blocks are run before your AngularJS application starts. Some modules can be configured there.

Run Blocks

```
myApp.run(function(VisitCounterService) {
  VisitCounterService.countVisitor();
});
```

Run blocks are run once immediately after your application has started.

AngularJS: Modules

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Your application is an Angular module:

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

myApp.controller("aController", function() { … });
myApp.factory("aService", function() { … });
myApp.filter("aFilter", function() { … });
myApp.directive("aDirective", function() { … });
```

```
<html ng-app="myApp">
  <head>…</head>
  <body>
    <div ng-controller="aController">
      …
    </div>
  </body>
</ul>
```

It can have include other AngularJS libraries:

```
var myApp = angular.module("myApp", ["satellizer", "ui.bootstrap", "ui.gravatar"]);
```

You can organize your app into separate modules:

```
var myApp = angular.module("myApp", ["myApp.security", "myApp.game"]);

var securityModule = angular.module("myApp.security", []);
securityModule.controller("LoginController", function(AuthenticationService) { … });
securityModule.factory("AuthenticationService", function() { … });

var gameModule = angular.module("myApp.game", []);
gameModule.controller("GameController", function() { … });
gameModule.directive("gameBoard", function() { … });
```

AngularJS: Dependency Injection

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

Angular is based around dependency injection.
The Angular injector is in charge of creating components, resolving their dependencies, and providing them to other components as requested.

```
myApp.factory("PeopleService", function($http) {
  return {
    getPeople: function() {
      return $http({
        url: "https://api.example.com/people",
      });
    },

    getFullName: function(person) {
      return person.firstName + " " + person.lastName;
    }
  };
});
```

You ask Angular to inject the **$http** service into **PeopleService**.

It's the same principle as injection with Java EE.

```
@Stateless
public class UserService implements IUserService {

    @EJB
    private UtilityService utilityService;
}
```

AngularJS: Dependency Injection

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

You can inject dependencies into any AngularJS component: controllers, services, directives, filters.

```
myApp.factory("PeopleService", function($http) {
  return {
    getPeople: function() { … },
    getFullName: function(person) { … }
  };
});
```

Inject your a service into a directive.

```
myApp.directive("personName", function(PeopleService) {
  return {
    type: "E",
    scope: {
      person: "="
    },
    controller: function($scope) {
      $scope.getFullName = PeopleService.getFullName;
    },
    template: "{{ getFullName(person) }}"
  };
});
```

Inject your a service into a controller.

```
myApp.controller("PeopleController",
  function(PeopleService, $scope) {
    PeopleService.getPeople().success(function(people) {
      $scope.people = people;
    });
  });
```

AngularJS: Automated Tests

heig-vd
Haute Ecole d'Ingénierie et de Gestion
du Canton de Vaud

AngularJS has excellent testing tools built-in:

https://docs.angularjs.org/guide/unit-testing

https://docs.angularjs.org/guide/e2e-testing

AngularJS: Unit Tests

heig-vd
Haute Ecole d'Ingénierie et de Gestion du Canton de Vaud

```
myApp.factory("PeopleService", function($http) {
  return {
    getPeople: function() {
      return $http({
        url: "https://api.example.com/people",
      });
    }
  };
});
```

To unit test a service which makes remote calls to an API…

It's easy to unit-test with dependency injection.

```
describe("PeopleService", function() {
  it("should return the list of people from the API", function() {

    var expectedList = [
      { firstName: "John", lastName: "Doe" },
      { firstName: "John", lastName: "Smith" }
    ];

    var fakeHttp = function() {
      return {
        success: function(callback) {
          callback(expectedList);
        }
      };
    };

    module(function($provide) {
      $provide.value("$http", fakeHttp);
    });

    var service;
    inject(function($injector) {
      service = $injector.get("PeopleService");
    });

    service.getPeople().success(function(people) {
      expect(people).toEqual(expectedList);
    });
  });
});
```

Tell Angular to use a **fake $http service**.

Inject it into your service.

That way your test only checks the functionality of the service itself. And since it's not calling the API, it runs **very fast**.