AngularJS *101*

Everything you need to know to get started

+stephane.begaudeau @sbegaudeau stephanebegaudeau.tumblr.com

THE FOLLOWING **PRESENTATION** HAS BEEN APPROVED FOR

ALL AUDIENCES

BY ME, MYSELF and I, INC.

www.twitter.com/sbegaudeau

stephanebegaudeau.tumblr.com

Why Angular?

to create properly architectured and maintainable web applications

Expressions

To create the views of your applications, you can use expressions within your HTML

- Javascript like code
- Used for small operations in the HTML page

```
<div>1+1 = {{1+1}}</div>
```

1+1 = 2

Expressions are nice for small operations, for real applications, we have something more powerful

Directives

what HTML would have been, had it been designed for building web-apps

Directives

Extends HTML to structure your application

- Declarative
- Use the data available in the scope (more on that later)
- Create the DOM of the fly

Let's have a look at an example: ngRepeat.

```
<div>
    <div ng-repeat="user in users">
        <h3>{{user.name}}</h3>
        <p>{{user.description}}</p>
    </div>
</div>
```

It iterates on a collection in the scope to create the DOM

Directives - ngRepeat

```
<div>
    <div ng-repeat="user in users">
        <h3>{{user.name}}</h3>
        <p>{{user.description}}</p>
    </div>
</div>
```

Tyrion Lannister

Youngest son of Lord Tywin

Daenerys Targaryen

Only daughter of Aerys II Targaryen

Arya Stark

Younger daughter of Eddard Stark

Jon Snow

Bastard Son of Lord Eddard Stark

Cersei Lannister

Daughter of Lord Tywin Lannister

For each elements in the collection "users" a new has been created with all its children

Directives - ngShow

AngularJS comes with a collection of standard directives that can be combined

```
<div>
    <div ng-repeat="user in users" ng-show="user.gender == 'female'">
        <h3>{{user.name}}</h3>
        <p>{{user.description}}</p>
    </div>
</div>
```

Daenerys Targaryen

Only daughter of Aerys II Targaryen

Arya Stark

Younger daughter of Eddard Stark

Cersei Lannister

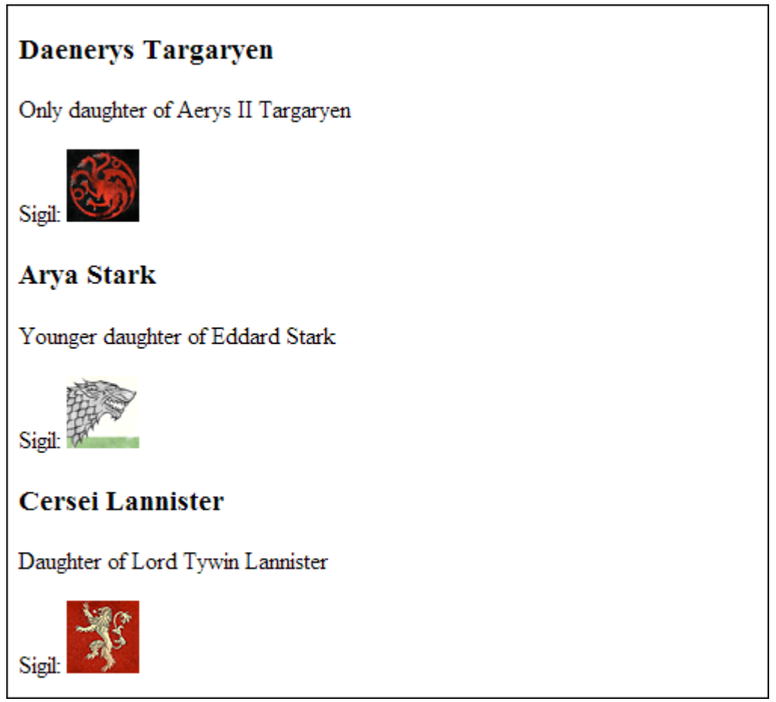
Daughter of Lord Tywin Lannister

ngShow let you hide elements that do not validate a given predicate
Here, the users that do not have the gender "female" have generated a hidden

Directives - ngSwitch

AngularJS also provides you with complex directives like ngSwitch

```
<div>
    <div ng-repeat="user in users"
         ng-show="user.gender == 'female'"
         ng-switch="user.house">
        <h3>{{user.name}}</h3>
        <p>{{user.description}}</p>
        Sigil:
        <img src="images/targaryen.gif" ng-switch-when="Targaryen">
        <img src="images/stark.gif" ng-switch-when="Stark">
        <img src="images/lannister.gif" ng-switch-when="Lannister">
    </div>
</div>
```



With those directives, you can create the basic structure of your web application easily

Directives

A final word on directives

- All the directives of the AngularJS standard library are named "ngMyAwesomeDirective"
- You can use them with "ng-my-awesome-directive"
- Some directives can be used as attributes, comments, DOM elements name or even CSS classes

```
<div>
    <div ng-my-awesome-directive></div>
    <ng-my-awesome-directive></ng-my-awesome-directive>
    <div class="ng-my-awesome-directive"></div>
    <!-- directive: ng-my-awesome-directive -->
</div>
```

And of course, you can create your own directives (we will create a very basic one later)

Data Binding

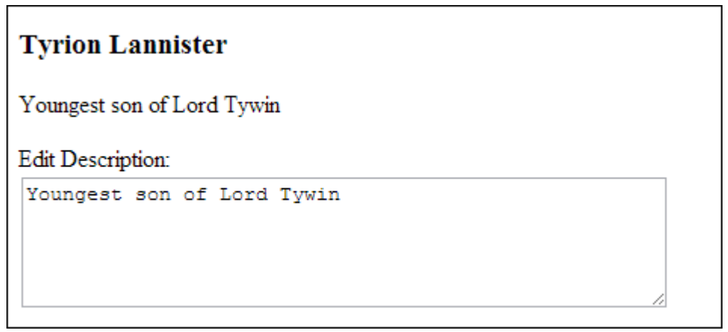
connect your models and your views

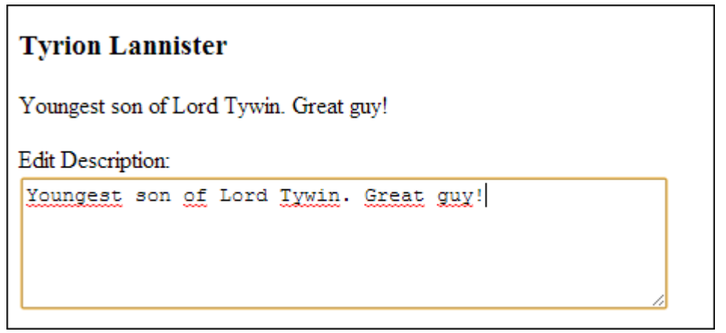
Data Binding

Angular gives you the ability to define the binding between the data in your scope and your views

- Most directives that are using expressions are creating a bidirectionnal data binding for you
- You can create manually new bindings with the directive ngModel

```
<div>
    <div ng-repeat="user in users">
        <h3>{{user.name}}</h3>
        <p>{{user.description}}</p>
        Edit Description:
        <textarea rows="5" cols="50" ng-model="user.description">
    </div>
</div>
```





The changes are visible in real-time in all the expressions

Filters

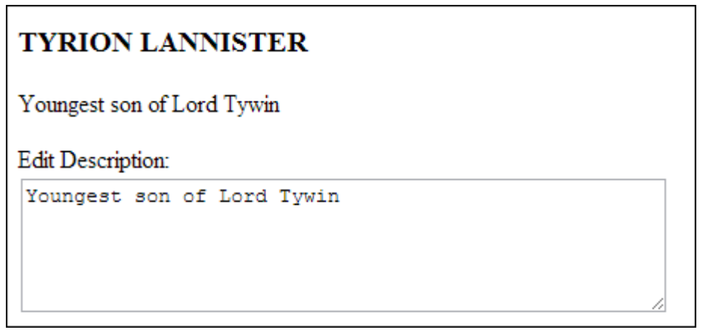
change the way your expressions are displayed

Filters - uppercase

Angular comes with a collection of filters that can change the way your data are displayed

- Usage: {{expression | filter}}

```
<div>
    <div ng-repeat="user in users">
        <h3>{{user.name | uppercase}}</h3>
        <p>{{user.description}}</p>
        Edit Description:
        <textarea rows="5" cols="50" ng-model="user.description">
    </div>
</div>
```



You can also easily create your own filters

Partial Views

single page web applications at best

Partial Views

Everything you need to build single page applications

- Angular handles the History management
- You can easily bind your views to the routes

All you need to do is

- Creates the page that will hold the structure of the application
- Bind the page to a specific AngularJS module thanks to ngApp
- Select where the views will be included with ngView
- Writes the different views
- Binds the view and the routes

Partial Views - ngView

```
<!DOCTYPE html>
<html>
    <head><title>AngularJS</title></head>
    <body ng-app="AngularJSModule">
        <div ng-view>Loading...</div>

        <!-- vendors -->
        <script src="angular.min.js"></script>
        <!-- modules -->
        <script src="app.js"></script>
        <!-- controllers -->
        <script src="users.js"></script>
    </body>
</html>
```

index.html

```
<div>
    <div ng-repeat="user in users">
        <h3>{{user.name | uppercase}}</h3>
        <p>{{user.description}}</p>
    </div>
</div>
```

users.html

- When the route changes, Angular will load the partial view in the DOM thanks to ngView
- The views can be created in other HTML files
- We will see later how we can bind a view to a route

Modules

the structure of your application

Modules and ngApp

In AngularJS, applications are structured in modules

You can define a new module very easily thanks to the function "module'

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

You just have to declare the name of the module and an array containing the name of the modules that you wre depending on

Inside of a module, you can create:

- controllers
- services
- filters
- directives

Modules and ngApp

Use ngApp, in order to tell Angular that a part of your application will be manage by a module

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

app.js

```
<!DOCTYPE html>
<html>
    <head><title>AngularJS</title></head>
    <body ng-app="AngularJSModule">
        <div ng-view>Loading...</div>

        <!-- vendors -->
        <script src="angular.min.js"></script>
        <!-- modules -->
        <script src="app.js"></script>
        <!-- controllers -->
        <script src="users.js"></script>
    </body>
</html>
```

index.html

Dependency Injection

building a testable and maintainable application

Dependency Injection

In Angular, most of the operation of the framework are using dependency injection (DI)

For example, in order to configure the routes of your module, you will have to inject the service $routeProvider in your configuration function.

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

angularJSModule.config(function ($routeProvider) {
    // do something
});
```

All the components that you will create will specify their dependencies thanks to DI
This way, you will have a collection of small specialized components that can be easily tested

Dependency Injection

The basic way of doing dependency injection in Angular uses the name of the parameters of the function

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

angularJSModule.config(function ($routeProvider) {});
```

This solution will not work with the minification of the code

Each time you can use dependency injection in Angular, you can use this way instead

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

angularJSModule.config(['$routeProvider', function ($routeProvider) {}]);
```

The string '$routeProvider' would not change with the minification of the code

Configure The Module

binding routes, views and controllers

Configure the module

In order to create the routes of your application, you will use the $routeProvider

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

angularJSModule.config(['$routeProvider', function ($routeProvider) {
    $routeProvider.when('/users', {
        templateUrl: 'views/users.html',
        controller: 'UsersCtrl'
    });

    $routeProvider.otherwise({
        templateUrl: 'views/404.html'
    });
}]);
```

If the end user visit the URL "http://www.ourdomain.com/#/users", the view "views/users.html" will be injected in the page, otherwise the view "views/404.html" will loaded.

Controllers

data provider for our views

Controllers

In Angular, the controller is used to provide data for the view

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

angularJSModule.controller('UsersCtrl', ['$scope', function ($scope) {
    // do something
}]);
```

In order to provide the data to the view, the controller can be injected with the scope of the view

In the module, we have binded the view, the route and the controller, so the controller can receive the scope of the view when the route is modified

```
<div>
    <div ng-repeat="user in users">
        <h3>{{user.name}}</h3>
        <p>{{user.description}}</p>
    </div>
</div>
```

Controllers

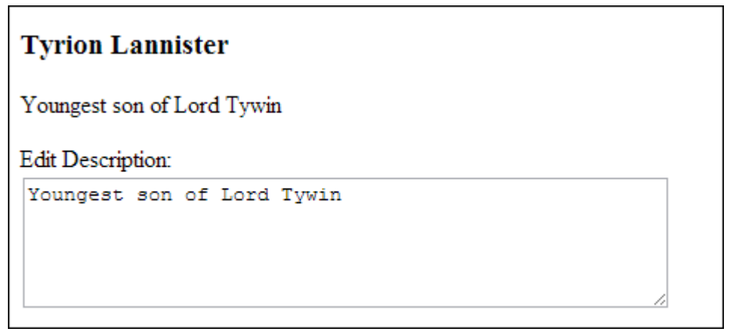
```
var angularJSModule = angular.module('AngularJSModule', []);

angularJSModule.controller('UsersCtrl', ['$scope', function ($scope) {
    $scope.users = [
        {
            name: 'Tyrion Lannister',
            description: 'Youngest son of Loard Tywin',
            gender: 'male',
            house: 'Lannister',
        }
    ];
}]);
```

controller

```
<div>
    <div ng-repeat="user in users">
        <h3>{{user.name}}</h3>
        <p>{{user.description}}</p>
        Edit Description:
        <textarea rows="5" cols="50" ng-model="user.description">
    </div>
</div>
```

view



result

Scope

the backbone of the views

Scope

The scope is used to link the controllers and the views to which they are binded.

A controller can add data and function in its scope and then they will be accessible in the view.

In our case, we have used in the view a variable named “users" which was created in the scope by the controller “UsersCtrl".

A scope can have child scopes which can see the content of the parent scopes.

Each directive can create and manage its own scope.

The scope also comes with additional operations that can be quite useful to build your application

Scope - $watch

AngularJS provides the necessary tool to observe the changes on the data of the scope from the controller.

With the operation "$watch", a controller can add a listener on an expression of the scope.

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

angularJSModule.controller('UsersCtrl', ['$scope', function ($scope) {
    $scope.users = [];

    $scope.$watch('users', function (newValue, oldValue) {
        console.log('The value of "users" has changed');
    }, true);
}]);
```

Scope - $broadcast and $on

AngularJS also gives you access to a system of events and listeners on the scope.

You can use the operation "$broadcast" in order to fire an event on a specific scope, then the event will be transmitted to the selected scope and all its children. With $on, you can receive the event.

If you want to send an event to the whole application, you can use the root scope by injected the service $rootScope.

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

angularJSModule.controller('UsersCtrl', ['$scope', function ($scope) {
    $scope.$on('AngularJSModule.Event', function () {
        console.log('An event "AngularJSModule.Event" has been fired!');
    });

    $scope.$broadcast('AngularJSModule.Event');
}]);
```

Services

utility components of your application

Services

Controllers contains the data of the application that should be available to the view.

If you have some code that you want to re-use or that you want to separate from a controller, use a service and inject it thanks to dependency injection.

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

angularJSModule.factory('UsersServer', [function () {
    return {};
}]);
```

Services can be injected into other services, filters, controllers or directives to build more complex components.

Services - public API

By using the function "factory", you can create your service and return it manually

You can then define the public API of your service and the private functions and variables

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

angularJSModule.factory('UsersServer', [function () {
    var privateFunction = function () {};

    var usersServer = {};
    usersServer.publicFunction = function () {
        privateFunction();
    };

    return usersServer;
}]);
```

There are other ways to create your services but we won't see them here.

Directives

today's HTML components

Directives

Directives are insanely powerful in Angular but it is a bit complex to create an advanced directive.

I won't explain here all the options available to create a directive, but here is a simple example.

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

angularJSModule.directive('whereIsThePower, [function () {
    return {
        template: '<p>Power resides where men believe it resides. No more and no less</p>',
        replace: true
    };
}]);
```

```
<div where-is-the-power></div>
```

Power resides where men believe it resides. No more and no less.

One more thing

You should **never** ever ever manipulate the DOM from a controller!

respect the separation of concerns, the data in the controller, the behavior in the directives
it's easier to test and easier to maintain

Wrapping things up

As you have seen it during this presentation, Angular is a MVC framework with a strong opinion on how things should be done. It helps you build an application

- testable with dependency injection
- maintainable with small specialized components
- with reusable components
- well architectured (data in controllers, behavior in the directives, utility stuff in services)

Frameworks, libraries and tools

If you want to build an Angular application, you should have a look at those tools too

- Batarang - chrome extension to debug Angular applications
- Bower - dependency management tooling for front end applications by Twitter
- Grunt - tasks management (minification, autoreload, SASS or CoffeeScript compilation)
- Yeoman - generate preconfigured kickass web applications
- Karma - Angular tests runner
- Angular-ui - collection of Angular directives (datepicker, Google Maps, Bootstrap)
- Restangular - improved Angular services for REST communication

Thanks!

For more information and regular news about AngularJS, follow me on Twitter or Google+