# AngularJS

**Beginner's guide - part 1**

POLITECNICO
DI TORINO

e-Lite

## AngularJS:

## AngularJS:

- **Superheroic JavaScript MVW Framework**

# AngularJS:

- **Superheroic JavaScript MVW Framework**

## AngularJS:

- **Superheroic JavaScript MVW Framework**
- Javascript framework for writing frontend web apps

# AngularJS:

- **Superheroic JavaScript MVW Framework**
- Javascript framework for writing frontend web apps
- It aims at **simplifying** both the **development** and the **testing** of **web application**

## Example 1

```html
<!DOCTYPE html>
<html lang="en" ng-app>
<head>
  <meta charset="UTF-8">
  <title>What's your name?</title>
  <script src="./angular.min.js"></script>
</head>
<body>

  <div>
    <label>Name</label>
    <input type="text" ng-model="name" placeholder="Enter
your name">
    <h1>Hello <span ng-bind="name"></span>!</h1>
  </div>
</body>
</html>
```

script loads and runs when the browser signals that the context is ready.

## Example 1

```
<!DOCTYPE html>
<html lang="en" ng-app>
<head>
  <meta charset="UTF-8">
  <title>What's your name?</title>
  <script src="./angular.min.js"></script>
</head>
<body>


  <div>
    <label>Name</label>
    <input type="text" ng-model="name" placeholder="Enter
your name">
    <h1>Hello <span ng-bind="name"></span>!</h1>
  </div>
</body>
</html>
```

Angular scans the HTML looking for the ng-app attribute. It creates a scope.

## Example 1

```
<!DOCTYPE html>
<html lang="en" ng-app>
<head>
  <meta charset="UTF-8">
  <title>What's your name?</title>
  <script src="./angular.min.js"></script>
</head>
<body>

  <div>
    <label>Name</label>
    <input type="text" ng-model="name" placeholder="Enter
your name">
    <h1>Hello <span ng-bind="name"></span>!</h1>
  </div>
</body>
</html>
```

The compiler scans the DOM for templating markups. Then, it fills them with info from the scope.

## Example 1

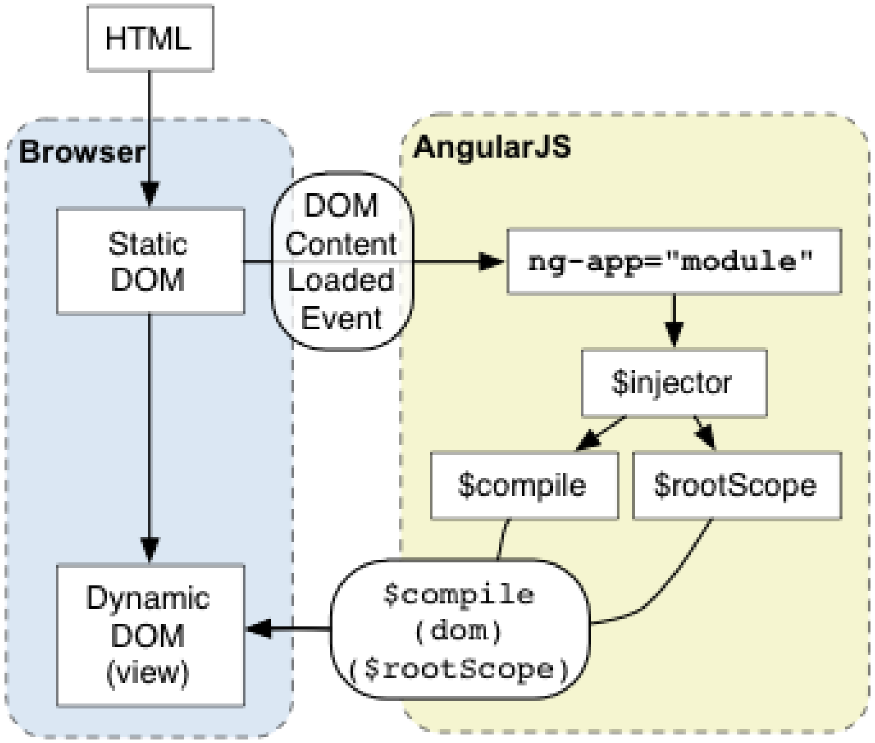
**Two-way data binding by ng-bind**

```
<!DOCTYPE html>
<html lang="en" ng-app>
<head>
  <meta charset="UTF-8">
  <title>What's your name?</title>
  <script src="./angular.min.js"></script>
</head>
<body>

  <div>
    <label>Name</label>
    <input type="text" ng-model="name" placeholder="Enter
your name">
    <h1>Hello <span ng-bind="name"></span>!</h1>
  </div>
</body>
</html>
```

name is replaced with the value inserted in the <input>

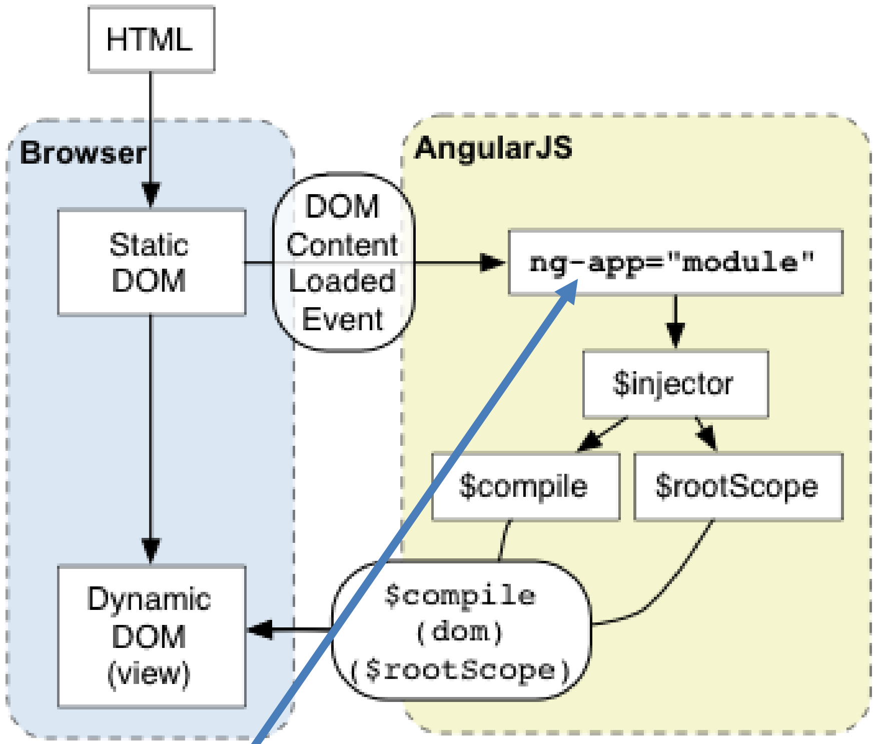
# AngularJS Initialization



https://docs.angularjs.org/guide/bootstrap

1. AngularJS initializes automatically upon DOMContentLoaded event
2. AngularJS looks in the **template** for the ngApp **directive** which designates our application root.
3. Loads the **module** associated with the directive.
4. Creates the application injector
5. Compiles the DOM treating the ngApp directive as the root of the compilation

# AngularJS Initialization



https://docs.angularjs.org/guide/bootstrap

1. AngularJS initializes automatically upon DOMContentLoaded event
2. AngularJS looks in the **template** for the ngApp **directive** which designates our application root.
3. Loads the **module** associated with the directive.
4. Creates the application injector
5. Compiles the DOM treating the ngApp directive as the root of the compilation

**Tip**: The ng-app attribute represents an AngularJS directive, named ngApp (AngularJS uses kebab-case for its custom attributes and camelCase for the corresponding directives which implement them)

# Concepts and Terminology

| | |
|---|---|
| **Template** | HTML with additional markup used to describe what should be displayed |
| **Directive** | Allows a developer to extend HTML with her own elements and attributes |
| **Module** | A container for parts of an application |
| **Scope** | Context where the model data is stored so that templates and controllers can access it |
| **Compiler** | Processes the template to generate HTML for the browser |
| **Dependency Injection** | Fetching and setting up all the functionality needed by a component |
| **Data Binding** | Syncing of data between the Scope and the HTML (two-way) |
| **Service** | Reusable functionality available for any view |

## Directives

- They are markers on a DOM element that tell Angular's HTML compiler to attach a specific behavior to that element
  - The **ng-app** directive defines an AngularJS application

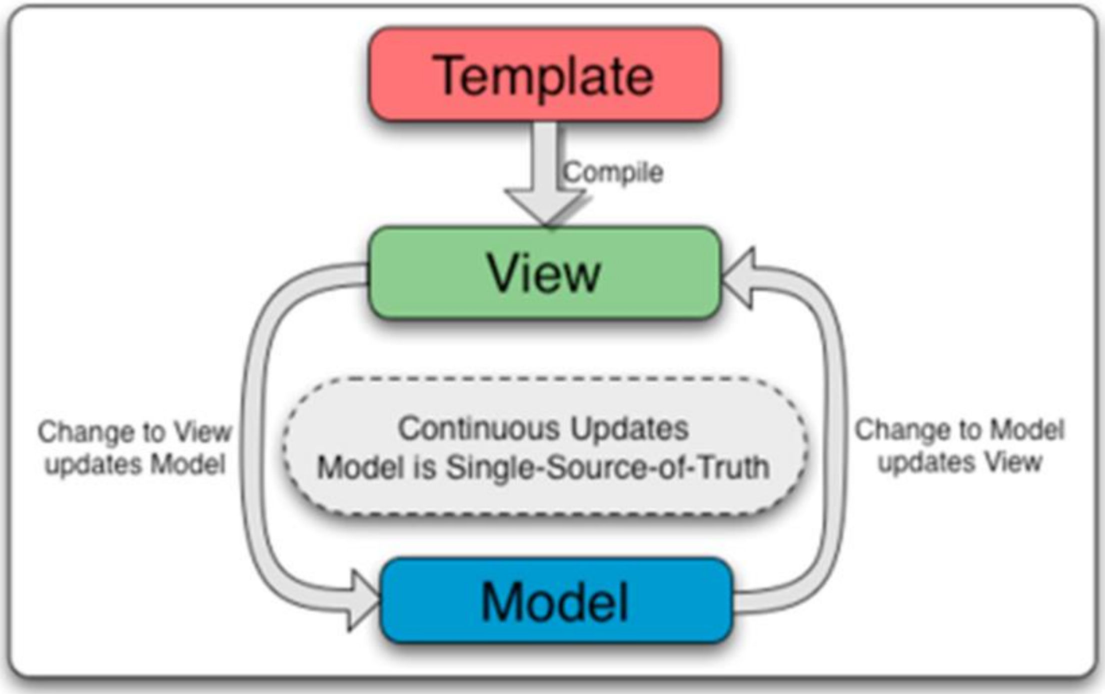
    `<html lang="en" `**`ng-app`**`>`
  - The **ng-model** directive binds the value of HTML controls (input, select, textarea) to application data.

    `<input type="text" `**`ng-model="name"`**` …>`
  - The **ng-bind** directive binds application data to the HTML view.

    `<span `**`ng-bind="name"`**`></span>`

## Two-Way Data Binding

- It is the automatic synchronization of data between the model and the view components


Template
Compile
View
Change to View updates Model
Continuous Updates Model is Single-Source-of-Truth
Change to Model updates View
Model

# Two-Way Data Binding

- It is the automatic synchronization of data between the model and the view components

**How can we do it?**

- AngularJS binds data to HTML using **Expressions:**
  - **ng-bind** directive

    ```
    <h1>Hello <span ng-bind=" expression "></span>!<\h1>
    ```

  - **Double braces**

    ```
    <h1>Hello {{ expression }}!<\h1>
    ```

- AngularJS will resolve the expression, and return the result exactly where the expression is written

## Example 2: double braces

Two-way data binding with double braces

```
<!DOCTYPE html>
<html lang="en" ng-app>
<head>
  <meta charset="UTF-8">
  <title>What's your name?</title>
  <script src="./angular.min.js"></script>
</head>
<body>

  <div>
    <label>Name</label>
    <input type="text" ng-model="name" placeholder="Enter
your name">
    <h1>Hello {{ name }}! Math says 5 + 4 is {{ 5+4 }}
</h1>
  </div>
</body>
</html>
```

name is replaced with the value inserted in the <input> and 5+4 is replace with 9

## Other Built-in Directives

- ng-init
  - Evaluates the given expression(s) to, for example, create a variable when initiating the application
- ng-src | ng-href
  - Add an image or a link, where the src is evaluated by AngularJS
- ng-repeat
  - instantiate a template once per item from a collection

```
<div ng-init="names = ['Laura', 'Teo', 'Gabriella']">
    <h3>Loop through names with ng-repeat</h3>
    <ul>
        <li ng-repeat="name in names">{{ name }}</li>
    </ul>
</div>
```

→ Example 3

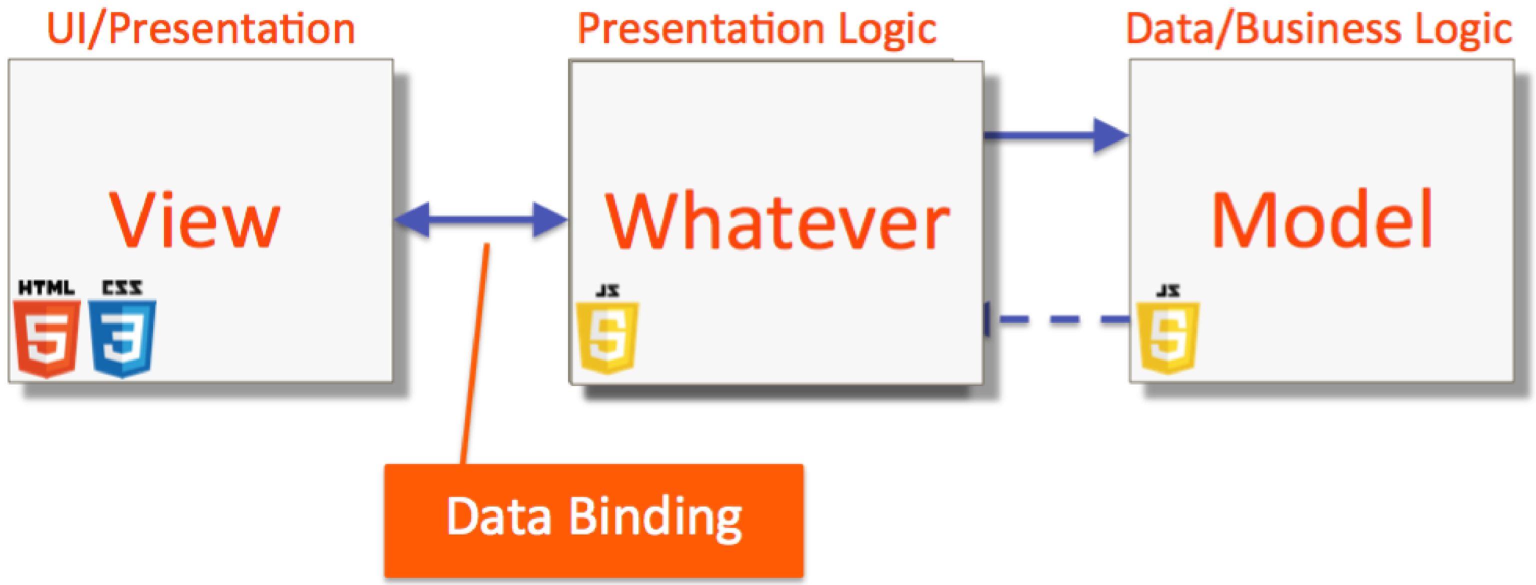
## Other Built-in Directives

- ng-show/ng-hide
  - show/hide DOM elements according to a given expression
- ng-if
  - include an element in the DOM if the subsequent expression is true
- ng-click
  - Angular version of HTML’s onclick
  - execute a given operation when the element is clicked

## Filters

- Formats the value of an expression for display to the user
  - `{{ name | uppercase }}`
- Keeps formatting into the presentation
- Syntax
  - `{{ expression | filter }}`
  - `{{ expression | filter1 | filter2 }}`
  - `{{ expression | filter:argument }}`

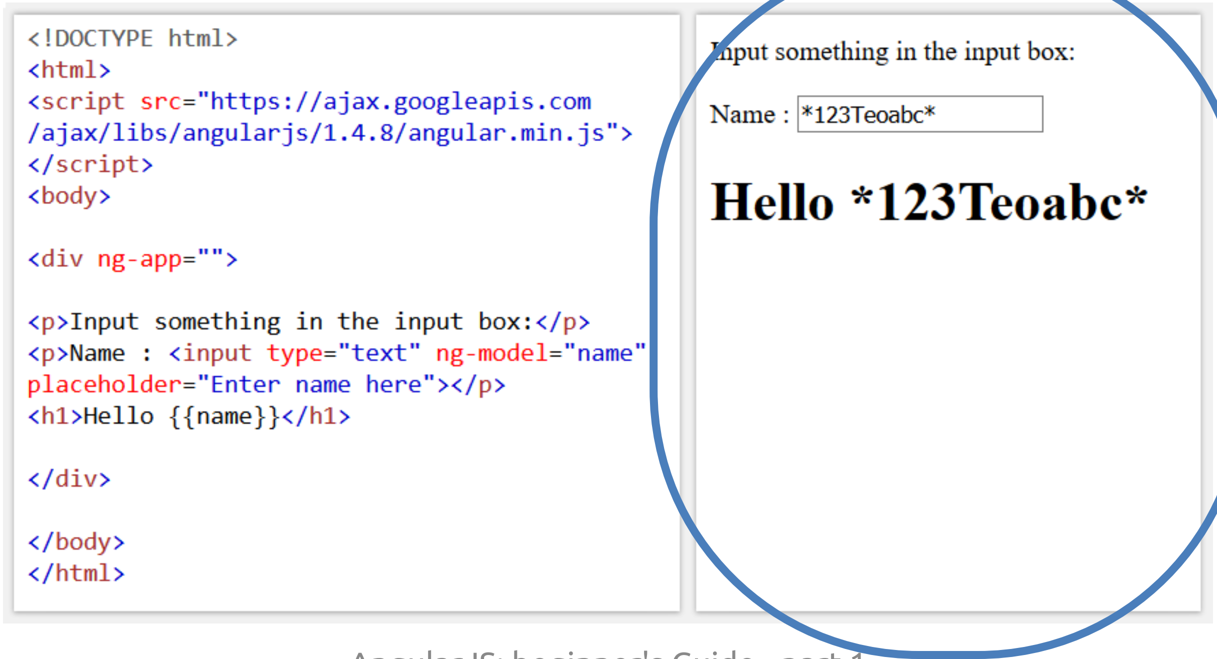
# AngularJS: MVW or MV*



# AngularJS: MVW or MV*

- View
  - It is the visible result generated by AngularJS by applying some directives to the HTML template

Example:



```
<!DOCTYPE html>
<html>
<script src="https://ajax.googleapis.com
/ajax/libs/angularjs/1.4.8/angular.min.js">
</script>
<body>

<div ng-app="">

<p>Input something in the input box:</p>
<p>Name : <input type="text" ng-model="name"
placeholder="Enter name here"></p>
<h1>Hello {{name}}</h1>

</div>

</body>
</html>
```

Input something in the input box:

Name : *123Teoabc*

**Hello *123Teoabc***

# AngularJS: MVW or MV*

- Whatever: the AngularJS **Controller**
  - A JavaScript function
  - It defines a new `$scope` that may
    - contain data (properties)
    - specify some behaviors (methods)
  - It should contain only the logic needed for a single view
    - i.e., each view should have one controller (and viceversa)

## AngularJS: MVW or MV*

- Whatever: the AngularJS **Controller**
  - It should be concerned – only! - with
    - consuming data,
    - preparing it for the view, and
    - transmitting it to service for processing
  - Best practices
    - services are responsible to hold the model and communicate with a server
    - declarations should manipulate the DOM
    - views do not have to know about their controller
    - a controller definitely does not want to know about its view

HTML

```
<!DOCTYPE html>
<html lang="en" ng-app="sonetExample" >
<head>
  <meta charset="UTF-8">
  <title>Introducing... Controllers!</title>
  <script src="./angular.min.js"></script>
</head>
<body ng-controller="MyCtrl">
  <div>
    <label>Name</label>
    <input type="text" ng-model="name" ...>
      <h1>{{ greeting }} {{name}}!</h1>
  </div>
[...]
```

module

controller

JS

```
angular.module("sonetExample", [])
 .controller('MyCtrl', function ($scope) {
    $scope.name = "";
    $scope.greeting = "Hello";
})
```

→ Example 4

# Modules

- Container to collect and organize components
  - in a *modular* way
- Multiple modules can exist in an application
  - “main” module (ng-app)
  - service modules, controller modules, etc.
- Best practices
  - a module for each feature
  - a module for each reusable component (especially directives and filters)
  - an application level module which depends on the above modules and contains any initialization code

## The Scope (`$scope`)

- The "glue" between a controller and a template
- The scope is shared among the controller and the view (the template): the framework passes it as a parameter to the controller

```
angular.module("sonetExample", [])
 .controller('MyCtrl', function ($scope) {
     ...
})
```

- It aims at defining the **data model** and exposing it to the view

# The Scope (`$scope`)

- Every property and/or function defined within the scope will be available in the view

HTML

```
[...]<body ng-controller="MyCtrl">
  <div>
    <label>Name</label>
    <input type="text" ng-model="name" ...>
      <h1>{{ greeting }} {{name}}!</h1>
      <h1>{{ Bye() }}!</h1>
  </div>
[...]
```

JS

```
angular.module("sonetExample", [])
 .controller('MyCtrl', function ($scope) {
    $scope.name = "";
    $scope.greeting = "Hello";
    $scope.Bye = function() {
            return "Bye" + " " + "!"
        };
})
```

⟶ Example 5

## The Scope (`$scope`)

- Thus, it can be seen as an execution context for expressions like
  `{{ pizza.name }}`
- Scopes can watch expressions and propagate events
- Scopes are arranged in a hierarchical structure that mimic the DOM structure of the application:
  - every application has ONE root scope: `$rootScope`
  - every controller has its own scope
  - all the scopes are descendant of the `$rootScope`
  -

continue

## The Scope (`$scope`)

– if we define nested controllers, the most internal scope inherits methods and properties of all the others

HTML

```
[...]
<div ng-controller="userController">
   <p>Name: <input type="text" ng-model="user.name"></p>
   <p>Surname: <input type="text" ng-model="user.surname"></p>
   <p ng-controller="greetingController">{{greeting()}}</p>
</div> [...]
```

JS

```
angular.module("myApp", [])
    .controller("userController",
        function($scope) {
            $scope.user = {name: "Mario", surname: "Rossi"};
        })
    .controller("greetingController",
        function($scope) {
            $scope.greeting = function() {
                return "Hello "+$scope.user.name+" "+$scope.user.surname+"!"
            };
        });
```

⟶ Example 6

## The Scope (`$scope`)

- More info about the Scope
  - ENG
    - https://docs.angularjs.org/guide/scope
  - ITA
    - http://www.html.it/pag/52713/scope-e-two-way-data-binding/
    - http://www.html.it/pag/52795/gerarchia-di-scope/

TO BE
CONTINUED...

## References

- AngularJS official guide
  - https://docs.angularjs.org/guide
- AngularJS API documentation
  - https://docs.angularjs.org/api
- AngularJS in 60 minutes [video]
  - https://www.youtube.com/watch?v=i9MHigUZKEM
- Learn Angular in your browser
  - http://angular.codeschool.com/

## 01QYAPD Social Networking: technologies and applications

My contact information:
Teodoro Montanaro (teodoro.montanaro@polito.it)

Thanks to **Luigi De Russis** (luigi.derussis@polito.it), the creator the slides I used as basis!

POLITECNICO DI TORINO
e-Lite
SOME RIGHTS RESERVED
BY NC SA

# License

- This work is licensed under the Creative Commons "Attribution-NonCommercial-ShareAlike Unported (CC BY-NC-SA 3,0)" License.
- You are free:
  - to **Share** - to copy, distribute and transmit the work
  - to **Remix** - to adapt the work
- Under the following conditions:
  - **Attribution** - You must attribute the work in the manner specified by the author or licensor (but not in any way that suggests that they endorse you or your use of the work).
  - **Noncommercial** - You may not use this work for commercial purposes.
  - **Share Alike** - If you alter, transform, or build upon this work, you may distribute the resulting work only under the same or similar license to this one.
- To view a copy of this license, visit http://creativecommons.org/license/by-nc-sa/3.0/