AngularJS

AN INTRODUCTION

Introduction to the AngularJS framework

POLITECNICO
DI TORINO

e-Lite

AngularJS

- Javascript framework for writing frontend web apps
  - DOM manipulation, input validation, server communication, URL management, etc.
- Inspired by the Model-View-Controller pattern
  - HTML templating approach with two-way binding
- Focus on supporting large and single page applications
  - modules, reusable components, etc.
- Widely used
  - a major rewrite is coming out (version 2)
  - we will use version 1.x

AngularJS

- Javascript framework for writing frontend web apps
  - DOM manipulation, input validation, server communication, URL management, etc.
- Inspired by the Model-View-Controller pattern
  - HTML templating approach with two-way binding
- Focus on supporting large and single page applications
  - modules, reusable components, etc.
- Widely used
  - a major rewrite is coming out (version 2)
  - we will use version 1.x

AngularJS

- Developed by Google
- Website: https://angularjs.org/
  - download version 1.x
- Included in our “starter kit”
  - https://github.com/SoNet-2016/starter-kit

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
Scope	Context where the model data is stored so that templates and controllers can access
Compiler	Processes the template to generate HTML for the browser
Data Binding	Syncing of data between the Scope and the HTML (two-way)
Dependency Injection	Fetching and setting up all the functionality needed by a component
Module	A container for parts of an application
Service	Reusable functionality available for any view

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

  <div>
    <label>Name</label>
    <input type="text" ng-model="name" placeholder="Enter
your name">
    <h1>Hello {{ name }}!</h1>
  </div>

  <script src="./angular.min.js"></script>
</body>
</html>
```

→ Example 1

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

  <div>
    <label>Name</label>
    <input type="text" ng-model="name" placeholder="Enter
your name">
    <h1>Hello {{ name }}!</h1>
  </div>

  <script src="./angular.min.js"></script>
</body>
</html>
```

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

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

  <div>
    <label>Name</label>
    <input type="text" ng-model="name" placeholder="Enter
your name">
    <h1>Hello {{ name }}!</h1>
  </div>

  <script src="./angular.min.js"></script>
</body>
</html>
```

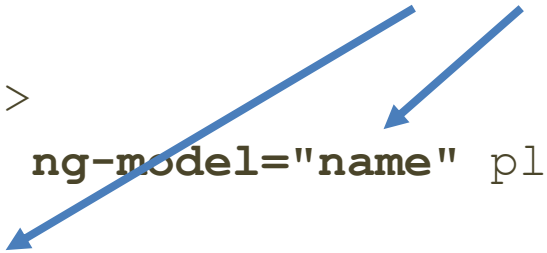
Angular scans the HTML looking for the ng-app attribute. It creates a scope.

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

  <div>
    <label>Name</label>
    <input type="text" ng-model="name" placeholder="Enter
your name">
    <h1>Hello {{ name }}!</h1>
  </div>

  <script src="./angular.min.js"></script>
</body>
</html>
```

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



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

  <div>
    <label>Name</label>
    <input type="text" ng-model="name" placeholder="Enter
your name">
    <h1>Hello {{ name }}!</h1>
  </div>

  <script src="./angular.min.js"></script>
</body>
</html>
```

Two-way binding

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

Directives and Data Binding

- Directives
  - markers on a DOM element that tell Angular's HTML compiler to attach a specific behavior to that element
  - `<html lang="en" **ng-app**>`
  - `<input type="text" **ng-model="name"** …>`
- Data Binding
  - the automatic synchronization of data between the model and the view components
    - `{{ name }}`

Other Built-in Directives

- ng-src | ng-href
  - delay <img> src | <a> href interpretation to get handled by Angular
- ng-repeat
  - instantiate a template once per item from a collection

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

Example 2

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 }}`

Controllers

- A controller 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

Controllers

- 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)

HTML

```
<!DOCTYPE html>
<html lang="en" ng-app="sonetExample" >
<head>
  <meta charset="UTF-8">
  <title>Introducing... Controllers!</title>
</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 = "Ciao";
})
```

→ Example 3

Modules

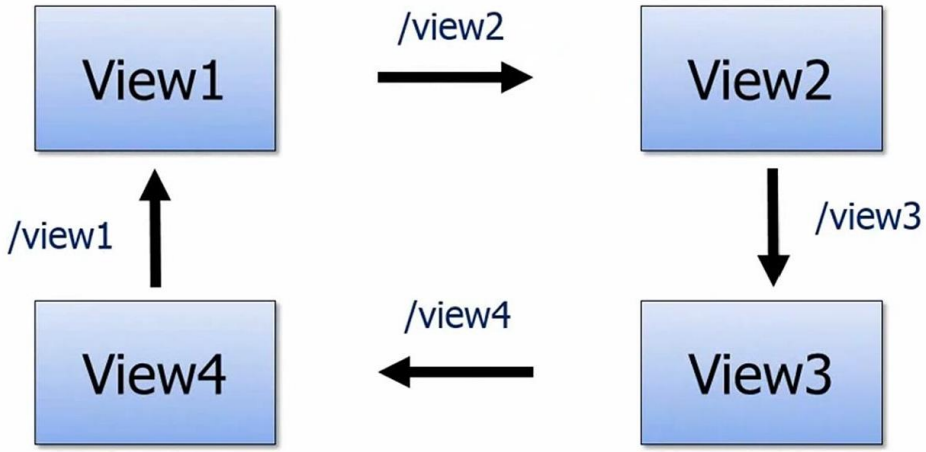
- Container to collect and organize components
  - in a *modular* way
- Multiple modules can exist in an application
  - “main” module (ng-app)
  - service modules, controller modules, etc.
- Best practices
  - a module for each feature
  - a module for each reusable component (especially directives 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
- An execution context for expressions like `{{ pizza.name }}`
- Scopes are arranged in a hierarchical structure that mimic the DOM structure of the application
- Scopes can watch expressions and propagate events
- `$rootScope`
  - every application has ONE root scope, all other scopes are its descendant

Routing

- Single page applications need routing
  - to mimic static addresses (http://mysite.com/...)
  - to manage browser history
- Goal: load different views into the single page app



ngRoute

- Angular API for routing
- Provides
  - a directive to indicate where view component should be inserted in the template (`ngView`)
  - a service that watches `window.location` for changes and updates the displayed view (`$route`)
  - a configuration that allows the user to specify the URL to be used for a specific view (`$routeProvider`)
- It is located in a dedicated js file
  - not in `angular.(min).js`

Using ngRoute

- In a template

```
<div ng-view></div>
```

- In your Angular module, add ngRoute as a dependency

```
angular.module('sonetExample', [ngRoute])
```

- Configure the routing table in a config block

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

Example 4

Passing parameters in URLs

URLs:

- #/pizzas/my-pizza
- #/pizzas/another-pizza

Routing table configuration:

JS

```
.config(['$routeProvider', function ($routeProvider){
  $routeProvider
    .when('/pizzas/:pizzaId', {
      templateUrl: 'single-pizza.html',
      controller: 'PizzaCtrl',
    })
    [...]
}])
```

Passing parameters in URLs

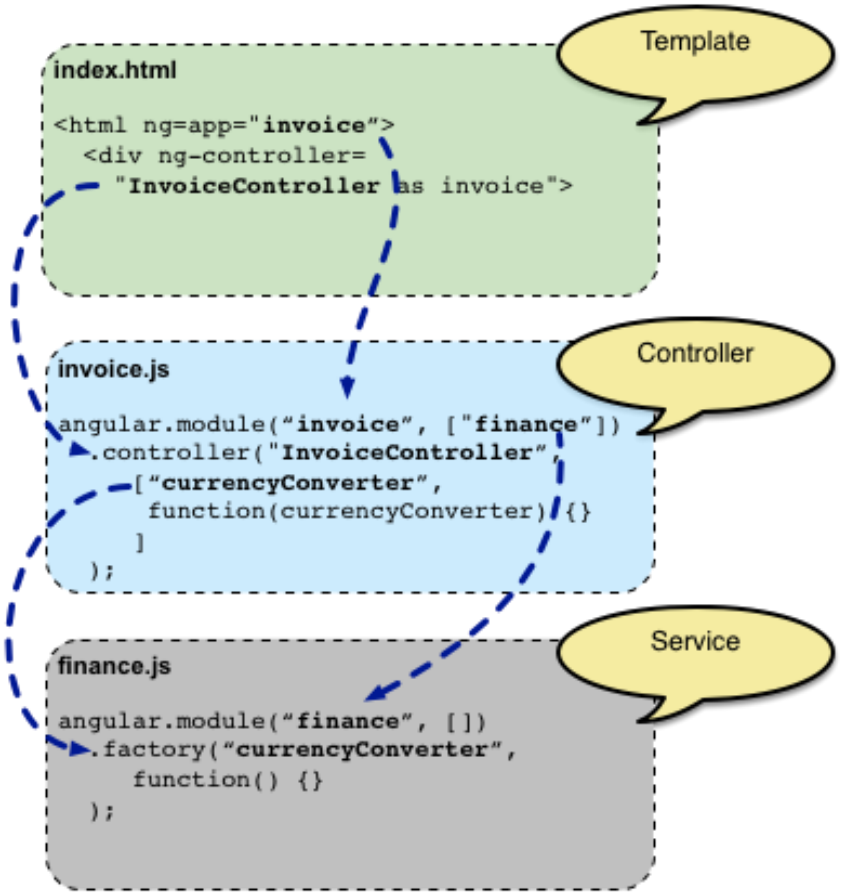
Controller:

JS

```
.controller('PizzaCtrl', ['$routeParams',
  function ($routeParams) {
    $routeParams.pizzaId
    // will be my-pizza or another-pizza
})
```

Services

- Responsible to hold the model and communicate with a server
- Singleton objects that are instantiated only once per app and lazy loaded
  - controllers are instantiated only when they are needed and discarded otherwise



Built-in Services

- Server communication
  - $http, $resource, …
- Wrapping DOM access
  - $location, $window, $document, …
- JavaScript functionality
  - $animate, $log, …

The $http Service

- Uses the XMLHttpRequest object (or JSONP) to communicate with a server
- How it works
  - takes a single argument: a configuration object
  - returns a promise with two methods: `success()` and `error()`
- It can use the `.then()` method, too
  - it registers a dedicated callback for the success and error method

The $http Service

promise

JS

```
$http({
    method: 'GET',
    url: '/someUrl'})
   .success(function(data, status, headers, config) {
     // this callback will be called asynchronously
     // when the response is available
   })
   .error(function(data, status, headers, config) {
     // called asynchronously if an error occurs
     // or server returns response with an error status.
   });
});
```

callbacks

The $http Service

JS

```
var sonet = angular.module('sonetExample', []);

sonet.controller('MyCtrl', ['$scope', '$http',
    function ($scope, $http) {
        var promise = $http.get('/pizzas.json');
        promise.success(function(data) {
            $scope.pizzas = data;
        });
    }
]);
```

The $http Service

JS

```
var sonet = angular.module('sonetExample', []);

sonet.controller('MyCtrl', ['$scope', '$http',
    function ($scope, $http) {
        var promise = $http.get('/pizzas.json');
        promise.success(function(data) {
            $scope.pizzas = data;
        });
    }
]);
```

Shortcut methods:
.get(), .post(), .put(), .delete(), etc.

$http in a Custom Service

JS

```
// custom service
sonet.factory('Pizza', function($http) {
    var pizzaService = {
        getData: function () {
            return $http.get('/pizzas.json')
              .then(function (response) {
                return response.data;
            });
        }
    };
    return pizzaService;
})

// controller
sonet.controller('PizzaCtrl', ['$scope', 'Pizza',
    function ($scope, Pizza) {
        Pizza.getData().then(function(pizzaList) {
            $scope.pizzas = pizzaList;
        });
    }
]);
```

⟶ Example 5

The $resource Service

- Replace the "low level" $http service
  - it has methods with high-level behaviors
  - but the server URLs must be RESTful
- REST: REpresentational State Transfer
  - works on the concept of “resource”
- CRUD: Create, Read, Update and Delete

read a collection of resources:	GET /pizzas
read a single resource:	GET /pizzas/:id
add a new resource:	POST /pizzas
update an existing resource:	PUT /pizzas/:id
delete an existing resource:	DELETE /pizzas/:id

The $resource Service

- Requires the ngResource module to be installed
- No callbacks needed!
  - e.g., `$scope.pizzas = pizzaService.query();`
- Available methods
  - get(), method: GET, single resource
  - query(), method: GET, is array (collection)
  - save(), method: POST
  - remove(), method: DELETE,
  - delete(), method: DELETE

Custom Directives

- Application specific
  - replace HTML tags with “functional” tags you define e.g., DatePicker, Accordion, ...
- Naming
- Camel case naming (`ngRepeat`)
  - automatically mapped to `ng-repeat`, `data-ng-repeat` in the templates
    e.g., `<div ng-repeat></div>`

Custom Directives

- A directive returns an “object” (directive definition) with several properties
  - details: https://docs.angularjs.org/api/ng/service/$compile

JS

```
var sonet = angular.module('sonetExample', []);

sonet.directive('helloHeader', function() {
    return {
        restrict: 'E',
        template: '<h2>Hello World!</h2>'
    };
});
```

Custom Directives

JS

```
var sonet = angular.module('sonetExample', []);

sonet.directive('helloHeader', function() {
    return {
        restrict: 'E',
        template: '<h2>Hello World!</h2>'
    };
});
```

Restricts the directive to a specific HTML "item".
If omitted, the defaults (Elements and Attributes) are used.

Custom Directives

JS

```
var sonet = angular.module('sonetExample', []);

sonet.directive('helloHeader', function() {
    return {
        restrict: 'E',
        template: '<h2>Hello World!</h2>'
    };
});
```

replace or wrap the contents of an element with this template

Compile and Link

- Named after the two phases Angular uses to create the live view for your application
- Compile
  - looks for all the directives and transforms the DOM accordingly, then calls the compile function (if it exists)
- Link
  - makes the view dynamic via directive link functions
  - the link functions typically create listeners on the DOM or the model; these listeners keep the view and the model in sync at all times
  - scopes are attached to the compiled link functions, and the directive becomes live through data binding

Link Function: Example

JS

```
var sonet = angular.module('sonetExample', []);

sonet.directive('backButton', ['$window', function ($window) {
    return {
        restrict: 'A',
        link: function (scope, elem, attrs) {
            elem.bind('click', function () {
                $window.history.back();
            });
        }
    };
}])
```

Other best practices in AngularJS

BEST PRACTICES

A dot in the model

"There should always be a dot in your model"

Parent scope: `$scope.name = "Anon"`
Child scope: `<input type="text" ng-model="name">`

The application will display "*Anon*" initially, but once the user change the value a *name* will be created on the child scope and the binding will read and write that value.
The parent's name will remain "*Anon*".
This can cause problems in large applications.
To avoid them:

```
<input type="text" ng-model="person.name">
```

camelCase vs dash-case

You can use variable named like

```
my-variable
myVariable
```

- The latter can cause problems in HTML
  - it is case-insensitive
- Angular solution: use either, it will map them to the same thing

Use dash-case in HTML and camelCase in JavaScript

Concepts behind AngularJS. Briefly.

BACKGROUND

Model-View-Controller Pattern

- Model
  - manage the application data
  - e.g., Javascript objects representing pizza names, pictures, comments, etc.
- View
  - what the web page looks like
  - e.g., HTML/CSS for viewing pizzas, view pizza photos, etc.
- Controller
  - fetch models and control views, handle user interactions
    - e.g., Javascript code for DOM event handlers, server communication, etc.

View Generation (frontend)

- HTML/CSS is generated in the frontend
- Templates are commonly used
- Basic idea
  - write a HTML document containing parts of the page that do not change
  - add some code that generate the parts computed for each page
  - the template is expanded by executing the code and substituting the result into the document
- Angular has a rich templating language

Models in Angular

- Angular does not specify model data
  - i.e., as Javascript objects
- It provides support for fetching data from a web server
  - support for REST APIs
  - JSON is frequently used

Single Page Applications

- Web applications in which the appropriate resource is dynamically loaded and added to the page when necessary
- Instead of changing the page, a view inside the page is changed
  - “everything is in a single web page”
  - exploits the templating approach
- The entire page does not reload
  - it uses AJAX

Single Page Applications

Problems:

1. mimic static addresses (http://mysite.com/...) and manage browser history
2. mix HTML and JavaScript
3. handle AJAX callbacks

Solutions:

[managed by Angular and similar frameworks]

1. routing (http://mysite.com/#...)
2. templating
3. server backend + HTML5 functionalities

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/

Questions?

01QYAPD Social Networking: Technologies and Applications

Luigi De Russis
luigi.derussis@polito.it

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/