*305CDE Developing the Modern Web 2*

# Lab 6: Introduction to AngularJS

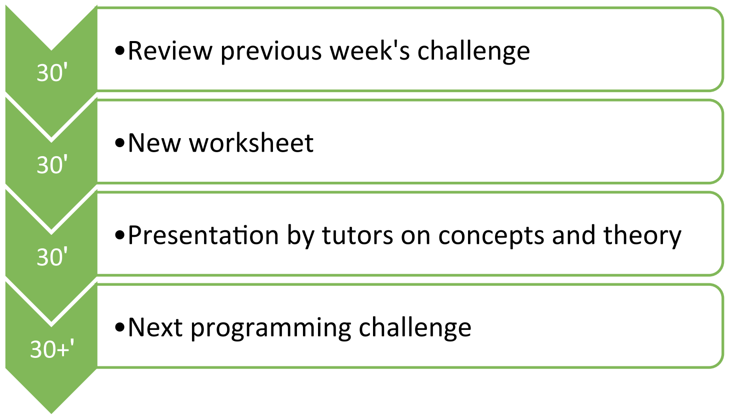
Lecturer: Dr. Jianhua Yang
Email: Jianhua.Yang@coventry.ac.uk
COM Programming Support: ECG-03, Tues. 13:00-15:00

ECG-14 05/11/2014

**OBJECTIVES**
This is the 1st of a series of 6 labs for module A202SGI, where we are going to spend most of our time on AngularJS. AngularJS is a JavaScript framework. It is quite new (since 2009) and is also becoming increasingly popular. It demonstrates some useful concepts in modern web development e.g. model–view–controller (MVC).
In this lab, we will introduce you AngularJS, get yourself familiar with basic directives and controllers. The objectives of this lab are: 1) Introduction of AngularJS; 2) Module and controller structure; 3) Some simple directives including ng-repeat; 4) First taste of server communication

## 1 LAB STRUCTURE AND MATERIAL

The overall structure of these labs is the same as before – it is divided into four (roughly) equal length parts: 1) review of previous challenge; 2) some new exercises based on the lab sheet; 3) presentations given on theoretical backgrounds; 4) a challenge for you to do in the lab and later in your own time.


30'
•Review previous week's challenge
30'
•New worksheet
30'
•Presentation by tutors on concepts and theory
30+'
•Next programming challenge

Lab sheets (the current document), presentation slides, and program source codes can be found on Colin's GitLab repository at https://gitlab.com/c0lin/305cde.git. Lab sheets and presentation slides are also available on Moodle.

## 2 ANGULARJS EXTENDS HTML WITH NEW ATTRIBUTES

### 2.1 AngularJS vs pure JS

AngularJS offers many advantages over pure JavaScript, one of which is two-way data binding. Before we dive into AngularJS, let's look at an example in pure JavaSCript.

(1) Create a new folder at where you normally save your work for 305CDE, give it a proper name for example `week6`.

(2) Create a new html file named `lab06_01_pureJS.html`

(3) Now open this file in Brackets, copy and paste the following code into this newly created file:

```
<!DOCTYPE html>
<html lang="en">

<body>
    <p>
        Hello
        <input type="text" id="input">
    </p>
    <button onclick="myFunction()">
        Click here
    </button>
    <p id="output"></p>
    <script>
        function myFunction() {
            var name = document.getElementById("input");
            document.getElementById("output").innerHTML = "Hello " +
name.value;
        }
    </script>
</body>

</html>
```

(4) Click the `Live Preview` button, input something into the text box for example your own name and then click `Click here`. Now you should be able to see what you just entered into the textbox.

This is a simple example, but it gives us a typical scenario of how JS works – the browser load HTML and JS codes, but it waits for further instructions on what to do! Now let's give a solution using AngularJS.

(1) Create a file called `lab06_02_angular_intro.html`.

(2) Copy and paste the following code into the file, and click `Live Preview`:

```
<!DOCTYPE html>
<html lang="en">

<head>
    <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.1/angular.min.js"></script
>
</head>
```

```
<body>
    <div ng-app="">
        <p>
            Name:
            <input type="text" ng-model="name">
        </p>
        <p>
            Hello, {{name}}!
        </p>
    </div>
</body>

</html>
```

What you can see now is that once you key in something into the text box, the result gets automatically updated! This is one of the advantages offered by AngularJS over pure JS– we as programmers concentrate on programming, while updating UI is done by AngularJS.

There are some concepts involved in the example above, e.g. separation of model–view–controller (MVC) and two-way data binding. We will come to that in the presentation section of the lab. Note here, however, how the AngularJS scripts was included within the HTML. It was included in the `<head>` section of the file. Below are some explanations why this is recommended, quoted from

> *A common advise for HTML applications, is to place all scripts at the very bottom of the <body> element. But, in many AngularJS examples, you will see the library in the <head> element. This is because calls to angular.module can only be compiled after the library has been loaded. Another solution is to load the AngularJS library in the <body> element, but before your own AngularJS scripts. – http://www.w3schools.com/angular/angular_modules.asp*

Also note that we used **angular.min.js** in the previous example, which is (you guessed it!) a minimized version of the file. You can also use angular.min.js instead. For a full list of what's available and the differences, go to the reference page at https://docs.angularjs.org/misc/downloading. Some of the differences are copied below for your convenience.

- **angular.js** — This is the human-readable, non-minified version, suitable for web development.
- **angular.min.js** — This is the minified version, which we strongly suggest you use in production.

Both `ng-app` and `ng-model` are examples of directives. The ng-app directive initializes an AngularJS application; while the ng-model directive binds the value of HTML controls (input, select, textarea) to application data, in our case a variable named "name". The part between two double braces is an expression – it is more or less like an ordinary JS expression, where the statement is evaluated and the results are given back. An alternative way to doing expression is using the `ng-bind` directive, as follows:

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

A less commonly used directive is `ng-init`, which often goes with `ng-app` to initialize variable such as

```
<div ng-app="" ng-init="Name='John'">
```

### 2.2 Test your understanding

Now it's time to do some little exercises. Using the directives mentioned above, take user inputs of their first and last name separately to give their full name. Within your text box, you should let the users know whether they need to key in first or last name using some pre-defined hints.

## 3 ORGANIZING DEPENDENCIES WITH MODULES

The previous example doesn't have many variables or functions. But once your program grows bigger, you'll need a way to control and organize. Here we introduce modules and controllers – this is the recommended way of doing things even for small applications.

### 3.1 Modules and controllers

You have seen the `ng-app` directive already. The purposes of this directive are:

(1) Put the current HTML element and all children elements under the control of AngularJS. This means that other outsider elements are not affected.

(2) Specify an AngularJS module to be used as the root module for the application. A module is a way to group dependencies and define controllers etc. You'll see an example later.

Controllers are the main body of any AngularJS application. As the name suggests, it controls how the data (your model) are being presented, and how users interact with your application. A module can have many controllers.

Now let's look at an example. The source code below can be found in `lab06_03_controllers.html`.

```
<!DOCTYPE html>
<html lang="en" ng-app="moduleToy" >

<head>
    <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.1/angular.min.js"></script
>
</head>

<body ng-controller="controllerToy as ctrl">
    <p>
        Hello
        <input type="text" ng-model="ctrl.name">
    </p>
    <button ng-click="ctrl.confirm()">
        Click here
    </button>
    <p id="output" ng-bind="ctrl.message"></p>

    <script type="text/javascript">
        angular.module('moduleToy', []).controller('controllerToy', [

                function () {
                var self = this;
                var lastName = 'last name is';
                self.confirm = function () {
                    self.message = 'Hello ' + self.name;
                };
```

```
                    }]);
    </script>
</body>

</html>
```

If you run this file, you'll see that the result look like the pure JS example. But indeed there are quite a lot going on.

(1) We start the application in the `html` tag, and pass on `ng-app` directive a module name `moduleToy`

(2) This module is defined using `angular.module('moduleToy', [])`, where squire brackets specify dependencies. Note this is not to be confused with `angular.module('moduleToy')`, where you simply retrieving a module instead of defining it.

(3) In a similar manner, you declare controller using `ng-controller` directive and define controller using `controller('controllerToy', [])`

(4) Note that the second argument to controller is actually an array containing dependencies and controller function definition. In our case, we don't have any dependencies, so it's a single-element array.

(5) As a good practice, we use `self` to point to `this` to avoid confusion. If you remembered it, we mentioned `this` and `that` already in previous lectures. Now you have `self`!

### 3.2 Test your understanding

- When you run the previous example, without putting anything in the textbox, if you click `Click here` it gives you `undefined`. Why is this? How to correct it?
- In the HTML, try to access variable named `lastName`, what happens? How to make this variable accessible?

## 4 THE REALLY HEAVY LIFTING

### 4.1 Working with arrays

We have so far been working with very simple data. Now let's turn to arrays, this is where things become very exciting. Take a look at the example below. This source code is in `lab06_04_arrays.html`. This example is adapted from AngularJS online documentation at https://docs.angularjs.org/api/ng/directive/ngRepeat.

```
<!doctype html>
<html lang="en">

<head>
    <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.1/angular.min.js"></script
>
</head>

<body>
    <div ng-app="" ng-init="friends = [
                                {name:'John', age:25, gender:'boy'},
                                {name:'Jessie', age:30, gender:'girl'},
                                {name:'Johanna', age:28, gender:'girl'}
                                ]">
        <p>
```

```
                I have {{friends.length}} friends. They are:
            </p>
            <ul>
                <li ng-repeat="friend in friends">
                    [{{$index + 1}}] {{friend.name}} who is {{friend.age}} years old.
                </li>
            </ul>
        </div>
</body>

</html>
```

There are several things to note in this example:

(1) The `ng-repeat` directive defines a scope where we can use the syntax `singleItem in array` to denote s member of the array.

(2) This `singleItem` variable only exists within the scope of the defining ng-repeat element

(3) `$index` is a helper variable, which gives the position of the array item. Similar helper variables exist e.g. `$first`, `$middle`, and `$last` which all return Boolean.

### 4.2 Test your understanding

- Move the array initialization into a module. What advantages do we have using a module instead of `ng-init` directive?
- Hide the array item if is not the first or last

## 5 SERVER COMMUNICATION USING $HTTP

In AngularJS, fetching data with $http is relatively easily. The code below shows an example of getting some wheather data from http://openweathermap.org/. We will go through this again in later labs. For the moment just remember basic syntax of retrieving remote JSON data. The source code is contained in file `lab06_05_http.html`.

```
<!doctype html>
<html lang="en" ng-app="weatherToy">

<head>
    <script
src="https://ajax.googleapis.com/ajax/libs/angularjs/1.3.1/angular.min.js"></script
>
</head>

<body ng-controller="MainCtrl as mainCtrl">
    <div>
        City Weather
    </div>
    <div ng-repeat="item in mainCtrl.items" class="item">
        <span ng-bind="item.name"></span>
        <span ng-bind="item.weather[0].description"></span>
    </div>

    <script>
        angular.module('weatherToy', []).controller('MainCtrl', ['$http',
            function ($http) {
                var self = this;
                self.items = [];


$http.get('http://api.openweathermap.org/data/2.5/box/city?bbox=12,32,15,37,10&clus
ter=yes').then(function (response) {
                    self.items = response.data.list;
                    console.log(self.items);
                }, function (errResponse) {
                    console.error('Error while fetching notes');
                });
        }]);
    </script>
</body>

</html>
```

## 6 CHALLENGE NO. 6

In Challenge No.2 you were asked to implement a user registration system. For the current challenge, i.e. Challenge No. 6 you need to complete a similar task, but all in pure AngularJS. Detailed requirements are:

- Implement a user registration system, using AngularJS and HTML.
- The system allows inputs of new user data e.g. name/age etc.
- The system contains a list of currently registered users.
- The system contains a search box to search the name field. It should updates its search results instantly based on search inputs.
- Only AngularJS allowed, no pure JavaScript or jQuery etc.
- AngularJS filters are not allowed.