Tutorial

Excerpt

This AngularJS tutorial is going to take you through a streamlined set of instructions and cram pack you full of AngularJS knowledge within 30 minutes. After this short but detailed tutorial you will be able to quickly create powerful and intuitive web applications.

Leon Revill

Nov 29, 2014

javascript, angularjs.

Easy

30 Minutes

Share

View Demo (http://codepen.io/RevillWeb/full/KwVoMv/)

View Code (https://github.com/RevillWeb/angularjs-tutorial)

Introduction

I started using AngularJS over 3 years ago when it was new and relatively unknown. Since then AngularJS has become one of the most popular JavaScript frameworks and for good reason. The AngularJS team have done superb work and are very dedicated to ensuring AngularJS is the best it can be.

This AngularJS tutorial will be using **version 1.3.4** and will cover just the basics and follow the best practices recommended by Google and by me through my own experience.

Setup

To get started create a folder called `angularjs-tutorial` and create the following files copying the directory structure below.

```
- angularjs-tutorial
    | - main.ctrl.js
    | - app.js
    | - index.html
```

Open `index.html` and add the following simple HTML to create a basic web page that includes AngularJS and Twitter Bootstrap

from a CDN. We are going to use Twitter Bootstrap as the CSS framework so its easy for us to quickly create the layout and focus more on the AngularJS. If you would like to know more about Twitter Bootstrap please check out my Twitter Bootstrap Tutorial (/tutorials/twitter-bootstrap-tutorial).

```
<!DOCTYPE html>
<html>
<head lang="en">
    <meta charset="UTF-8">
    <title>AngularJS Tutorial<
    <link rel="stylesheet" hre
    <script src="https://ajax.
    <script src="app.js"></scr
    <script src="main.ctrl.js'
</head>
<body>

</body>
</html>
```

This HTML document also includes `app.js` and `main.ctrl.js` which you have just created.

Angularfy your web page

Within our `app.js` file we need to instantiate an AngularJS module which we will use for our app. Copy the follow code into `app.js`

```
angular.module('app', []);
```

The `angular.module()` function acts as a setter and getter for angular modules. The first argument is the name of the module where we chose `app`, for the second we provided an array. If an array is provided then angular will create the app module as opposed to retrieving it which you'll see later when we create our controller.

> ### Info
>
> In more complex applications you will provide an array of module dependencies as opposed to an empty array as with this example.

Now we need to tell angular where we want to use this module, open up `index.html` and

add `ng-app='app'` to the `<body>` tag as shown below.

```
<!DOCTYPE html>
<html>
<head lang="en">
    <meta charset="UTF-8">
    <title>AngularJS Tutorial<
    <link rel="stylesheet" hre
    <script src="https://ajax.
</head>
<body ng-app="app">

</body>
</html>
```

This tells angular that everything within the DOM under body is part of our AngularJS application called `app`.

The controller

AngularJS provides a loose MVC structure more accurately described as MVVM (http://en.wikipedia.org/wiki/Model_View_ViewModel). The controller is where your business logic should be handled, essentially the controller 'controls' the data and provides it to the view for display. To declare our main controller open up `main.ctrl.js` and add the following JavaScript code.

```
angular.module('app').controll
    var vm = this;
});
```

In the code above we first retrieve the app module we created early and then use the `controller` function to instantiate a new controller. The `controller` function takes two parameters. The first is the controller name and the second is a function where we will place our controlling code.

Before we can use the controller we need to tell angular which part of our HTML document this controller has governance over. You can have many controllers per document and many controllers governing the same section of a document, you can even have nested controllers. In our very simple example we are going to have a single controller governing the entire page. We will use the `controllerAs` syntax to declare this (more on this later), open `index.html` and add the following to the body tag.

```
<!DOCTYPE html>
<html>
<head lang="en">
    <meta charset="UTF-8">
    <title>AngularJS Tutorial<
    <link rel="stylesheet" hre
    <script src="https://ajax.
</head>
<body ng-app="app" ng-controll

</body>
</html>
```

Understanding scope

Traditionally you would inject `$scope` into the controller and assign all controller variables to the scope object which will then make them accessible within the view. However, the better approach and recommended by the AngularJS team is the controllerAs syntax (https://github.com/johnpapa/angularjs-styleguide#controlleras-controller-syntax). This provides a more semantic approach to declaring controllers within your HTML and also prevents issues such as scope bleed when multiple-nested controllers are present.

We capture the controller instance into the `vm` variable which stands for ViewModel as recommended by John Papa's style guide (https://github.com/johnpapa/angularjs-styleguide). We will then assign all controller variables that we need access to within the view to this object so angular can work its magic, such as two-way data binding.

To understand the scope lets declare a simple variable called `title` which will hold some title text we want to display within our web page. Open `main.ctrl.js` and add the following.

```
angular.module('app').controll
    var vm = this;
    vm.title = 'AngularJS Tutc
});
```

To display this text within our web page we use the following syntax.

```
<!DOCTYPE html>
<html>
<head lang="en">
    <meta charset="UTF-8">
    <title>AngularJS Tutorial<
    <link rel="stylesheet" hre
    <script src="https://ajax.
</head>
<body ng-app="app" ng-controll
    <div class="container">
        <h1>{{main.title}}</h1
    </div>
</body>
</html>
```

In the above example we are displaying the title text inside a `h1` tag using a set of curly braces so that angular knows that this is a variable. We access the controllers scope by prefixing the variable with main matching our `ControllerAs` definition. Now imagine that we had multiple controllers in the same document and you were referencing multiple variables using the traditional method where you would simple do `{{title}}`. It would be very difficult to see which scope the variables were present in. Additionally it would also be easy to accidentally have multiple variables with the same name (i.e scope bleed).

If you open `index.html` within a web page you should see the title text output as a header on a very boring web page.

Understanding binding

Two-way data binding is one of the main selling points of AngularJS so lets take a look what its all about. Within `main.ctrl.js` add another variable called `searchInput` as shown below.

```
angular.module('app').controll
    var vm = this;
    vm.title = 'AngularJS Tutc
    vm.searchInput = '';
});
```

Open `index.html` and add a very simple search input using the following code.

```
<!DOCTYPE html>
<html>
<head lang="en">
    <meta charset="UTF-8">
    <title>AngularJS Tutorial<
    <link rel="stylesheet" hre
    <script src="https://ajax.
    <script src="app.js"></scr
    <script src="main.ctrl.js'
</head>
<body ng-app="app" ng-controll
<div class="container">
    <h1>{{main.title}}</h1>
    <div class="input-group">
        <span class="input-grc
            <span class="glyph
        </span>
        <input type="text" cla
    </div>
    <p>{{main.searchInput}}</p
</div>
</body>
</html>
```

In addition to adding a search input we have used the curly braces again to output the `searchInput` variable. On the text input we have added the `ng-model` directive and specified the `searchInput` variable from our main controller. This input is now bound to this variable making the inputs value available

from within the controller code and also from anywhere within the HTML document that is governed by the controller. Now if you open `index.html` in a browser and type within the search box you should instantly see the search input value printed underneath as you type. Have a go using the live preview below.

Using ngRepeat

AngularJS provides a wide range of built in directives for you to help you perform common tasks. The ngRepeat directive allows you to iterative through items and display them on the page. Let's create an array of objects we can iterate through. Update `main.ctrl.js` to have the following array.

```
angular.module('app').controll
    var vm = this;
    vm.title = 'AngularJS Tutc
    vm.searchInput = '';
    vm.shows = [
        {
            title: 'Game of Th
            year: 2011,
            favorite: true
        },
        {
            title: 'Walking De
            year: 2010,
            favorite: false
        },
        {
            title: 'Firefly',
            year: 2002,
            favorite: true
        },
        {
            title: 'Banshee',
            year: 2013,
            favorite: true
        },
        {
            title: 'Greys Anat
            year: 2005,
            favorite: false
        }
    ];
});
```

(/)

Follow @revillweb

(/) (/blog/)

Tutorials (/tutorials/)

Series (/series/)

Articles (/articles/)

AngularJS Tutorial - Learn A

Info

In a real-world scenario data such as this would probably be loaded via an API using a service to interact and store the data locally.

Now we have an array we can use the ngRepeat directive to loop through each TV show and create an unordered list within our web page. Open `index.html` and add the following code.

```
<!DOCTYPE html>
<html>
<head lang="en">
    <meta charset="UTF-8">
    <title>AngularJS Tutorial<
    <link rel="stylesheet" hre
    <script src="https://ajax.
    <script src="app.js"></scr
    <script src="main.ctrl.js'
</head>
<body ng-app="app" ng-controll
<div class="container">
    <h1>{{main.title}}</h1>
    <div class="input-group">
        <span class="input-grc
            <span class="glyph
        </span>
        <input type="text" cla
    </div>
    <h3>A list of TV shows</h3
    <ul class="list-group">
        <li class="list-group-
    </ul>
</div>
</body>
</html>
```

In this example ngRepeat will copy the list item element for each item in our shows array. We can then access properties of

each show using `{{show.[property]}}` to display them within the list element.

Hiding and showing elements

Another set of directives that AngularJS provides allow you to easily hide and show DOM elements based on criteria or an expression. These directives are ngHide (https://docs.angularjs.org/api/ng/directive/ngHide), ngShow (https://docs.angularjs.org/api/ng/directive/ngShow), ngSwitch (https://docs.angularjs.org/api/ng/directive/ngSwitch) & ngIf (https://docs.angularjs.org/api/ng/directive/ngIf). Even though these directives provide a similar result they are useful in different situations. The ngSwitch directive is useful when you have a case like situation, such as a radio button toggle. ngHide, ngShow and ngIf are very similar except with a fundamental difference. ngIf will remove the affected element from the DOM

where ngHide and ngShow will essentially add a `display: none;` to it hiding it from view.

Therefore ngIf should be used where possible as this will save the browsers memory due to a decluttered DOM. ngHide and ngShow are particularly useful when the items are animated as the additional overhead of adding the element back to the DOM with ngIf can often affect the animations.

For our simple example we would like to add a star next to each TV show in the list only if the show is marked as a `favorite`. Update `index.html` to use ngIf as follows.

```
<h3>A list of TV shows</h3>
<ul class="list-group">
    <li class="list-group-item
</ul>
```

This simply added the star icon to the DOM when the `show.favorite` property is equal to true. It's worth mentioning that expressions can be used here so something like

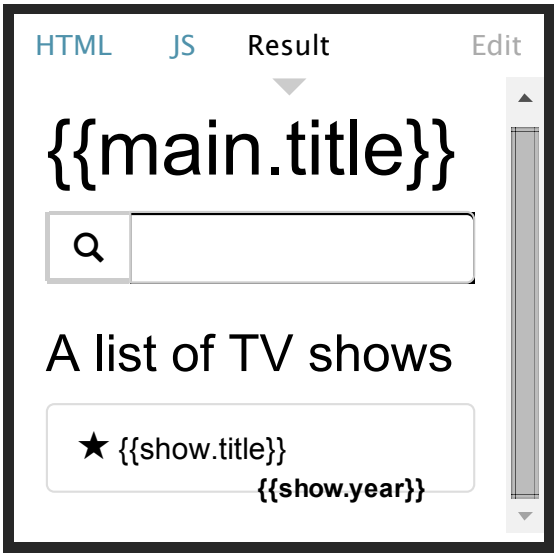
`show.favorite == true` or `show.favorite != false` would also work.

Using filters with ngRepeat

AngularJS provides the ability to easily filter repeated content. Let's use the search input we created earlier to allow users to filter the TV show list.

```
<h3>A list of TV shows</h3>
<ul class="list-group">
    <li class="list-group-item
</ul>
```

It's that easy! Imagine trying to do something like that with jQuery. Have a play around with this implementation below.



Let's make things a little more complicated and add the option to change the order of the list items. Update `main.ctrl.js` with another set of objects specifying the order types available.

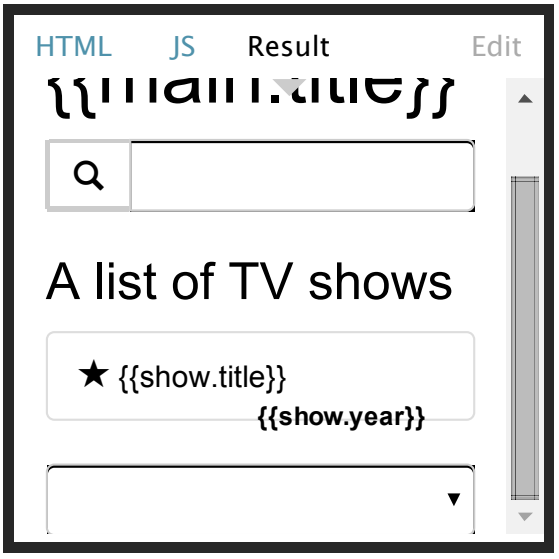
```
vm.orders = [
    {
        id: 1,
        title: 'Year Ascending
        key: 'year',
        reverse: false
    },
    {
        id: 2,
        title: 'Year Descendir
        key: 'year',
        reverse: true
    },
    {
        id: 3,
        title: 'Title Ascendir
        key: 'title',
        reverse: false
    },
    {
        id: 4,
        title: 'Title Descendi
        key: 'title',
        reverse: true
    }
];
vm.order = vm.orders[0];
```

Now add a select menu to `index.html` so the user is able to choose the list order and add an additional filter to the ngRepeat we added earlier.

```
<body ng-app="app" ng-controll
<div class="container">
    <h1>{{main.title}}</h1>
    <div class="input-group">
        <span class="input-gro
            <span class="glyph
        </span>
        <input type="text" cla
    </div>
    <h3>A list of TV shows</h3
    <ul class="list-group">
        <li class="list-group-
    </ul>
    <select class="form-contro
</div>
</body>
```

Above we have added an `orderBy` filter to the ngRepeat specifying which key of the show object we want to order by, in this case `title`. We then specified the direction of the order which is stored within the orders objects as `reverse`. To populate the select menu we use the ngOptions directive which you can read more about here (https://docs.angularjs.org/api/ng/directive/select).

See it in action below by changing the selected value in the select menu.



Form validation

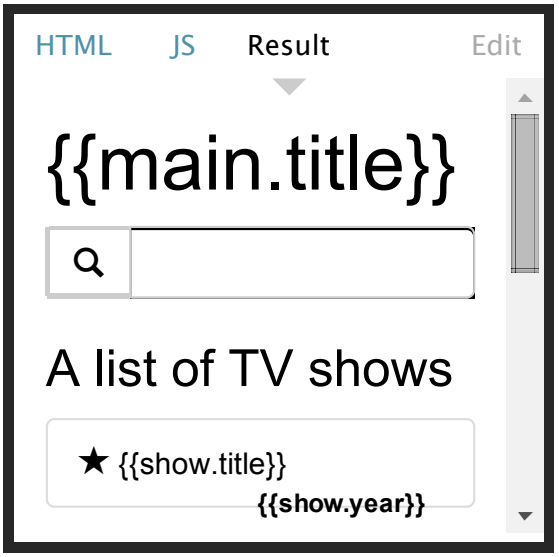
Collecting data from a user makes up a large part of web applications. AngularJS makes it easy to validate form data before it is used by your application. In this simple example we will create a form allowing the user to add a new TV show to the list. Open `index.html` and add the following HTML to create a simple form.

```
<!DOCTYPE html>
<html>
<head lang="en">
    <meta charset="UTF-8">
    <title>AngularJS Tutorial<
    <link rel="stylesheet" hre
    <script src="https://ajax.
    <script src="app.js"></scr
    <script src="main.ctrl.js'
</head>
<body ng-app="app" ng-controll
<div class="container">
    <h1>{{main.title}}</h1>
    <div class="input-group">
```

```
        <span class="input-gro
            <span class="glyph
        </span>
        <input type="text" cla
    </div>
    <h3>A list of TV shows</h3
    <ul class="list-group">
        <li class="list-group-
    </ul>
    <select class="form-contro
    <div class="clearfix"></di
    <h3>Add a new TV Show</h3>
    <form name="main.addForm"
        <div class="form-group
            <label>Title</labe
            <input type="text"
        </div>
        <div class="form-group
            <label>Year</label
            <input type="numbe
        </div>
        <div class="row">
            <div class="col-xs
                <label>Favorit
            </div>
            <div class="col-xs
                <button class=
            </div>
        </div>
    </form>
</div>
</body>
</html>
```

This simple form will allow the user to enter a show title, year and specify if it is a favorite. On the title and year fields we have added a `required` attributed marking this field as required. Additionally we have specified

the field type for the year input as `number` and added the additional `min` and `max` properties. AngularJS provides out of the box form validation and will automatically provide feedback to the user when they click the add button. If the user has not entered a value in either of the required fields they will be prompted with an error message. Also, if the user has not entered a number between 1900 and 2030 they will be instructed to do so. Try it yourself using the pen below.



Read more about forms with AngularJS here (https://docs.angularjs.org/guide/forms).

Responding to simple events

The final part of this AngularJS tutorial will show you how events are used within the framework. Events have always been an important part in JavaScript frameworks and libraries and it is no different here, even with Angular's two-way data binding.

The form we just created is useless if the user is unable to actually submit the data and add it to the list of shows. Add the following code to the bottom of `MainController`.

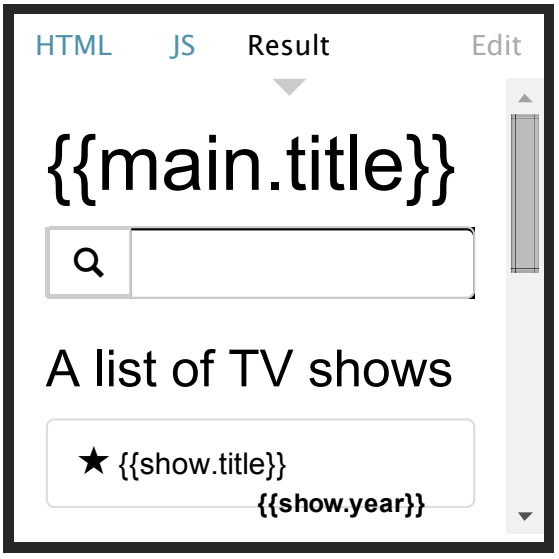
```
vm.new = {};
vm.addShow = function() {
    vm.shows.push(vm.new);
    vm.new = {};
};
```

The above code declares an object called new which the form uses to store its input values denoted by the ngModel value (i.e. `ng-model="main.new.year"`). Also a function `addShow` is created which pushes the new object into the shows array and adds it to the list within the view.

AngularJS provides various event directives such as ngClick, ngChange, ngFocus, etc. that we can use to react upon. In our simple example we will use the ngSubmit directive to trigger the `addShow` function when the form is successfully submitted. Open `index.html` and amened your form declaration to include this directive as shown below.

```
<form name="main.addForm" clas
```

Now, when the form passes validation and the user clicks the add button this `addShow` function will get called, adding the new show to the list. Have a go!



This is an extremely simple example of one in many event directives AngularJS provides,

head over to the documentation (https://docs.angularjs.org/api) to learn more.

Conclusion

In this quick tutorial you should now have a good working understand of many AngularJS concepts which will allow you to build simple single page applications. AngularJS offers so much more than I can show you in 30 minutes, if you liked this tutorial then please subscribe below to receive updates regarding up coming advanced AngularJS tutorials.

Subscribe to our mailing list

@ email ac Subscribe

View Demo (http://codepen.io/RevillWeb/full/KwVoMv/)

View Code (https://github.com/RevillWeb/angularjs-tutorial)

Comments Community

Recommended 2 Sort by Best

Join the discussion…

Homayoon Fayez
• 3 months ago

Nice tutorial. By the way you are pushing an empty vm.new object to the vm.shows array that's why octavianusb had update problems. I just read your Angular tutorial. I don't know what is the right angular way but this solved my update problem :)

vm.new.title = vm.new.title;
vm.new.year = vm.new.year;

• Reply • Share ›

octavianusb
• 4 months ago

Hi Leon! I made your tutorial, it's a very good one, thank you. Anyway I have a question, after I insert a new show, my vm.shows object isn't updated, is there a way to update it? Because after I reload the page I loose all the new shows added. As I know the push() method should do that, but in your example it don't work.

• Reply • Share ›

revillwebdesign
Mod
octavianusb
• 4 months ago

Hi, glad you liked my tutorial. Anything you do with JavaScript is in memory so as soon as

you refresh the browser everything will reset. If you want to persist the data you need to use something like localStorage. Take a look at this article to learn how: http://www.revillweb.co

• Reply • Share ›

Kai • 4 months ago

Very cool tutorial, following from beginning to the end in an hour or so. Thanks a lot!

• Reply • Share ›

Chris • 4 months ago

I believe the form validation you show here is actually being done with HTML5 and has nothing to do with AngularJS.

• Reply • Share ›

CEO // headlinesbd.com • 5 months ago

very nice and helpful tutorial, loved it!

• Reply • Share ›

Liviu • 5 months ago

What I like most about this tutorial is the small steps to make progress. It is so easy to follow because every new code part has clear instructions to where it should be added.
Just love it! Thanks!

• Reply • Share ›

Frederic FLECHE • 5 months ago

Just to let you know that there is a little error in your code

concerning the fillters. You should replace the first block by the second one :

id: 4,
title: 'Title Ascending',
---
id: 4,
title: 'Title Descending',

• Reply • Share ›

revillwebdesign
Mod ➚ Frederic FLECHE
• 5 months ago

Hi Frederic. Thanks for taking the time to let me know, it's been spotted before and I;ve finally got round to sorting it. The code samples should have the fix now and the post itself will be updated as part of the next post. Thank you!

• Reply •

(https://twitter.com/revillweb)

(https://www.facebook.com/pages/Revillweb/264778196907648)

(https://plus.google.com/113128976126646245587)

Subscribe to our mailing list

@ email address Subscribe