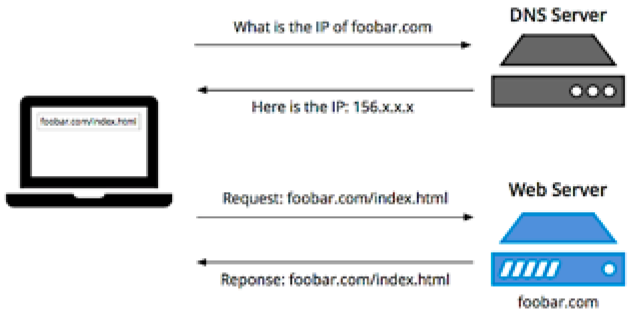
ANGULARJS

What happened when you write www.google.com to your browser?

Client (BROWSER) sends a request to the *Server* and Server starts to send what the client asks. Think about you are sending a picture with 4 parts, if one of the part is lost, browser knows and can not opens it.

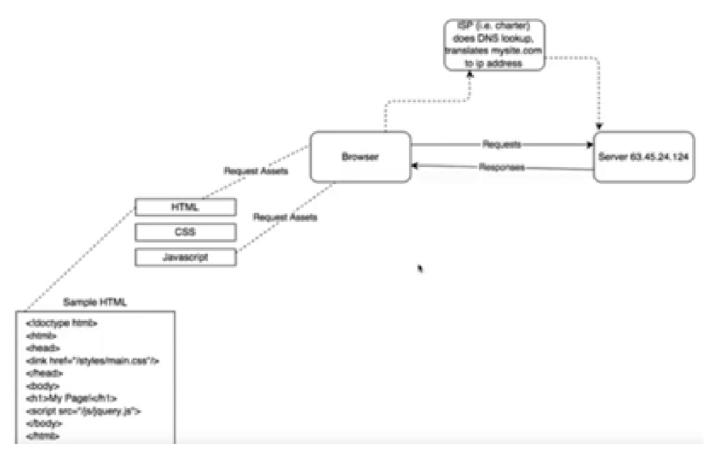
How web works?

DNS (Domain Name Service) – DNS changed the name to an IP address

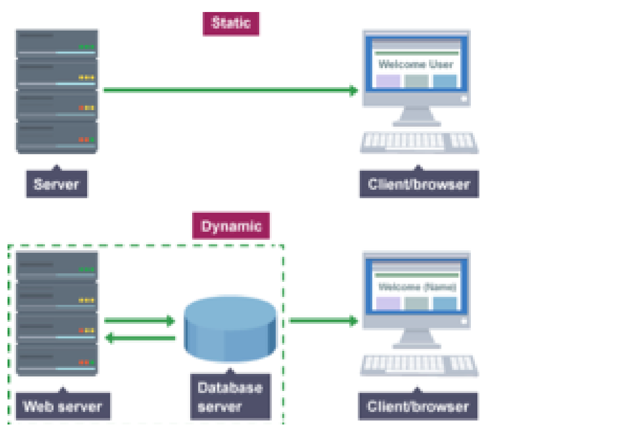


What is the IP of foobar.com
DNS Server
foobar.com/index.html
Here is the IP: 156.x.x.x
Request: foobar.com/index.html
Web Server
Reponse: foobar.com/index.html
foobar.com

TCP (Transmission Control Protocol)
IP (Internet Protocol)
HTTP (Hyper Text Transfer Protocol)



```
<!doctype html>
<html>
<head>
<link href="/styles/main.css"/>
</head>
<body>
<h1>My Page</h1>
<script src="/js/jquery.js">
</body>
</html>
```



FRONT END & BACK END

<u>Front End;</u>

- Client side (Browser side)
- Web Designer
- HTML
- CSS
- JS

<u>Back End;</u>

- Server side
- What user cant see
- Worry about
    - Security
    - Content management
    - Structure
- Constantly changing updated in real time
- Dynamic website requires database

ANGULARJS

The problem is that we spend a lot of time updating the document object model (DOM), the representation of the HTML that sits inside the browser's memory.

That's the way for us to manipulate the HTML and code, and also to read it and see what all the HTML on the page is attributes and various values.

So we need to do a lot of management ourselves without AngularJS, after a while, this gets really difficult to deal with and it can become just overwhelming.

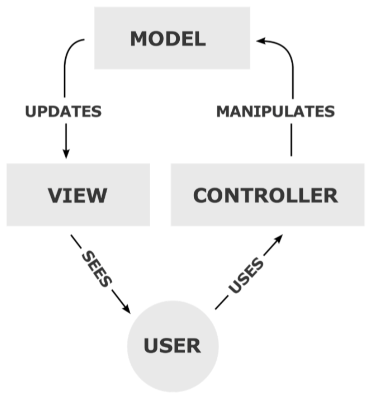
So, with AngularJS, if we could just update one side and the other side updated automatically. I mean the HTML and JS updates.

That's the basic idea behind the various frameworks like AngularJS have become popular.

So, we need to connect those two things.

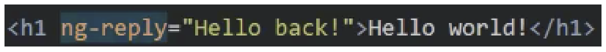
We want whatever happens on one to effect the other and vice versa automatically without us having to do it manually.

So, a concept called **Model View Controller** or **MVC**.

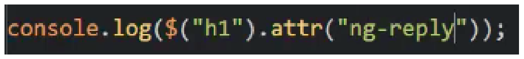
 MODEL UPDATES MANIPULATES VIEW CONTROLLER SEES USES USER	*MODEL* is our data (JS it is a variable for example) *VIEW* is whatever people see (in the case of web application, that is HTML, or in reality the DOM; the representation of the HTML in the browser's memory.) Something that ties or binds the model and the view. So, whatever happens to the model automatically affects the view … without us. *BINDS*; involves Controllers, View Models, …

HTML ASIDE : Custom Attributes

1) Add a <script> tag pointing to angular.js
2) Add an ng-app attribute in your HTML



```
<h1 ng-reply="Hello back!">Hello world!</h1>
```



```
console.log($("h1").attr("ng-reply"));
```

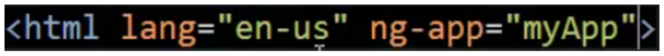
JS ASIDE : Global Namespace

Encapsulation, making sure that we're not polluting the global namespace, is a very important aspect of building extensible, reusable and complex software.

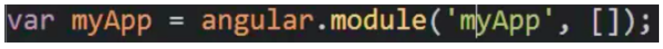
Angular module is a collection of services, directives, controllers, filters, and configuration information. Container for the application not to leak identifiers out and clutter up someone else's code

MODULES, APPS and CONTROLLERS

- ng-app is an Angular directive

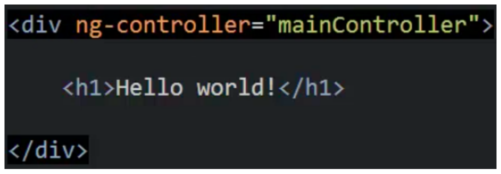


```
<html lang="en-us" ng-app="myApp">
```

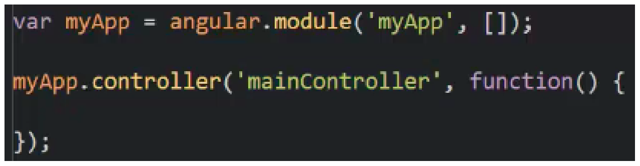


```
var myApp = angular.module('myApp', []);
```

Declare Controller



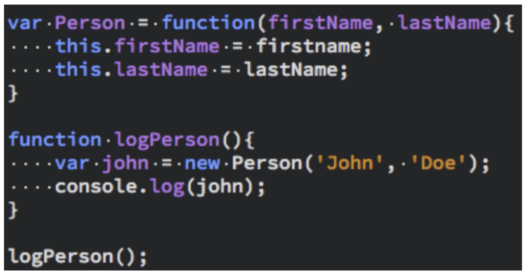
```
<div ng-controller="mainController">

    <h1>Hello world!</h1>

</div>
```



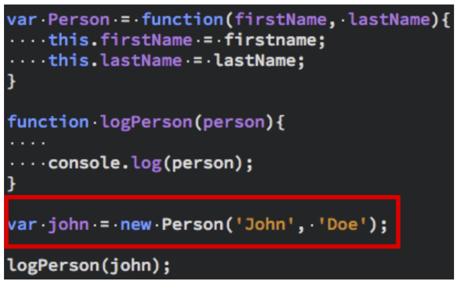
```
var myApp = angular.module('myApp', []);

myApp.controller('mainController', function() {

});
```

JS ASIDE - Dependency Injection

- Giving a function an object
- Rather than creating an object inside a function, you pass it to the function



```
var Person = function(firstName, lastName){
    this.firstName = firstname;
    this.lastName = lastName;
}

function logPerson(){
    var john = new Person('John', 'Doe');
    console.log(john);
}

logPerson();
```



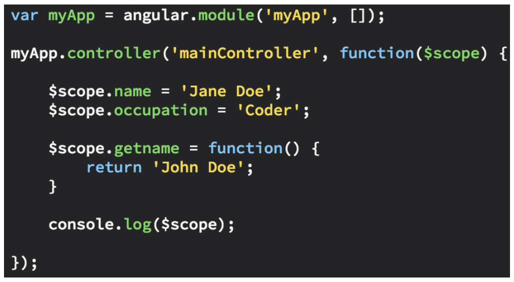
```
var Person = function(firstName, lastName){
    this.firstName = firstname;
    this.lastName = lastName;
}

function logPerson(person){
    
    console.log(person);
}

var john = new Person('John', 'Doe');

logPerson(john);
```

Think about we define an object using a function. Then inside of our object, we add a first and last name before we create a new object. Then we create a new function and think about we create a new object inside this function, it works. But my first function is dependent my new object because it is inside of my second function. It is a dependency. If something were to change about a new object, we have to change inside the function. It makes for, really complicated or difficult to deal with code.

So instead, we use dependency injection. If we create our new object outside the function and pass it to the function, we will get the same result and so our function is not dependent the new object.

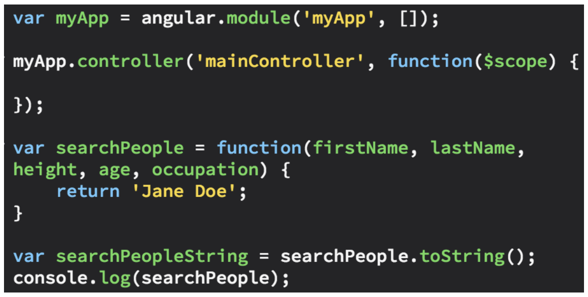
The Scope Service

It involves dependency injection. $scope is an object of AngularJs.

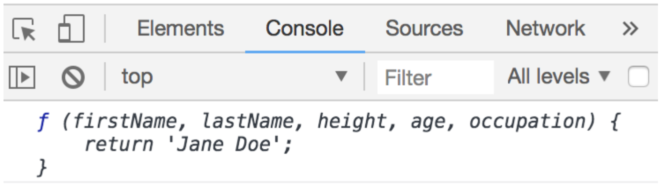


```
var myApp = angular.module('myApp', []);

myApp.controller('mainController', function($scope) {

    $scope.name = 'Jane Doe';
    $scope.occupation = 'Coder';

    $scope.getname = function() {
        return 'John Doe';
    }

    console.log($scope);

});
```

JS ASIDE - Functions and Strings



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

myApp.controller('mainController', function($scope) {

});

var searchPeople = function(firstName, lastName,
height, age, occupation) {
    return 'Jane Doe';
}

var searchPeopleString = searchPeople.toString();
console.log(searchPeople);
```

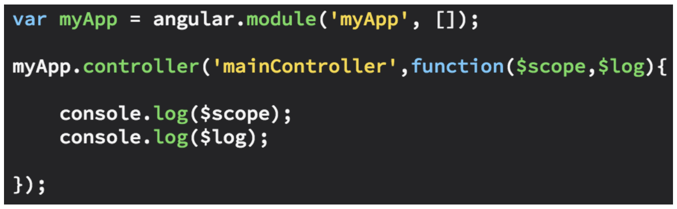


```
ƒ (firstName, lastName, height, age, occupation) {
    return 'Jane Doe';
}
```

So, if I can take any function in JS and get its string representation, that is the function actually typed out the way the coder typed it out, then this is just a string.

Getting Other Services

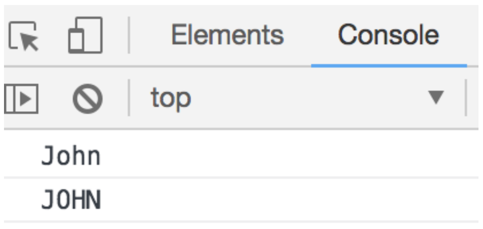
Steps: download the AngularJS version, add it to <script>, add to module dependency and add the service to the controller



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

myApp.controller('mainController',function($scope,$log){

    console.log($scope);
    console.log($log);

});
```

It means; I can simply get any of these services by simply including them in my parameter list, to the controller. Thanks to AngularJS's dependency injection.

```
var myApp = angular.module('myApp', ['ngMessages', 'ngResource']);

myApp.controller('mainController',function($scope, $log, $filter) {

    $scope.name = "John";
    $scope.formattedname = $filter("uppercase")($scope.name);

    $log.info($scope.name);
    $log.info($scope.formattedname);
});
```



```
John
JOHN
```

Thanks the idea of modules, where we are not polluting the global namespace and we have array list of dependencies. So, it knows we're gonna use them, and then this dependency injection concept to make it easy for me to get the object in a clean way.

JS ASIDE - Arrays and Function

My array can contain functions !

```
var things =
    [1,"2",function(){
        alert(`Hello!`)
    }]

things[2]();
console.log(things);
```

Dependency Injection and Minification

Minification: Shrinking the size of files for faster download.

```
myApp.controller('mainController', function($scope, $log) {

    $log.info($scope);

});
```

```
myApp.controller("mainController",function(a,b){b.info(a)});
```

Minifier's code

So, when the minifier changed the name of the variable, it broke AngularJS's dependency injection. So, they provided us a second method for specifying dependency injection:

- A minifier never change the values inside a string.

```
myApp.controller('mainController',["$scope","$log",function($scope,$log){

    $log.info($scope);

}]);
```

```
myApp.controller("mainController",
["$scope","$log",function(a,b){b.info(a)}]);
```

a=$scope ; b=$log

DATA BINDING and DIRECTIVES

Scope and Interpolation

<u>Interpolation</u>: Creating a string by combining strings and placeholders.

"My name is" + name

```
<div ng-controller="mainController">

    <h1>Hello {{ name + '.How are you?' }}!</h1>

</div>
```

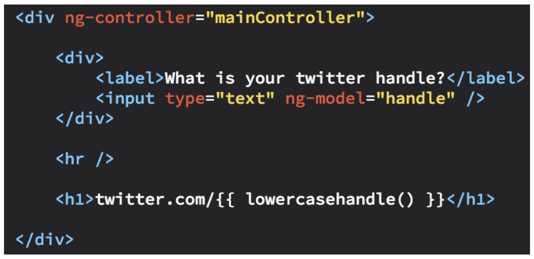
```
myApp.controller('mainController',["$scope",function($scope){

    $scope.name = "Eyup";

}]);
```

Directives and Two Way Data Binding

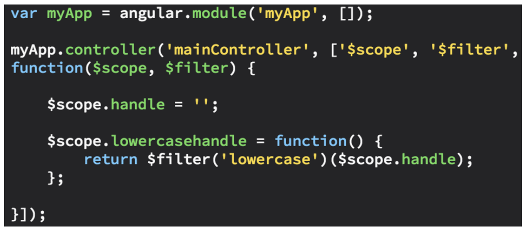
<u>Directive</u>: An instruction to AngularJS to manipulate a piece of the DOM.
This could be "Add a Class", "Hide This", "Create This", etc.

We may have manually changed what was going on in the DOM, in memory representation of the HTML. AngularJS prefers that we use directives, because it makes it much more powerful and much more flexible, much more easier to do.

So we are directing what is going on in the DOM, we are directing that things change on the web page.

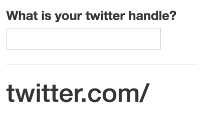


```
<div ng-controller="mainController">

    <div>
        <label>What is your twitter handle?</label>
        <input type="text" ng-model="handle" />
    </div>

    <hr />

    <h1>twitter.com/{{ lowercasehandle() }}</h1>

</div>
```



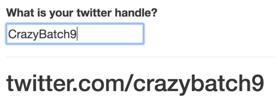
```
var myApp = angular.module('myApp', []);

myApp.controller('mainController', ['$scope', '$filter',
function($scope, $filter) {

    $scope.handle = '';

    $scope.lowercasehandle = function() {
        return $filter('lowercase')($scope.handle);
    };

}]);
```

ng-model → is a custom attribute and it is the ng-model directive. This says; I want this element to be bound to a specific property or variable in the scope.



What is your twitter handle?

twitter.com/

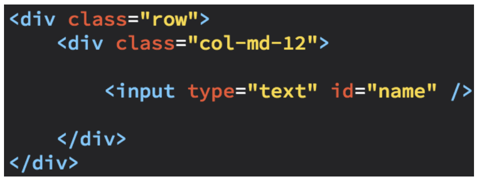


What is your twitter handle? CrazyBatch9

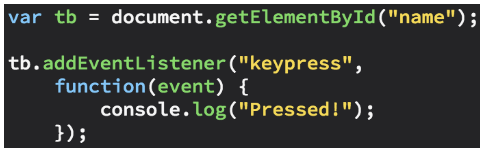
twitter.com/crazybatch9

So, **ng-model** (view) changed the model, (lowercasehandle) model changed the view
That is two-way data binding.

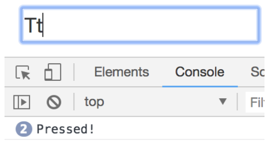
JS ASIDE - The Event Loop



```
<div class="row">
    <div class="col-md-12">

        <input type="text" id="name" />

    </div>
</div>
```



```
var tb = document.getElementById("name");

tb.addEventListener("keypress",
    function(event) {
        console.log("Pressed!");
    });
```

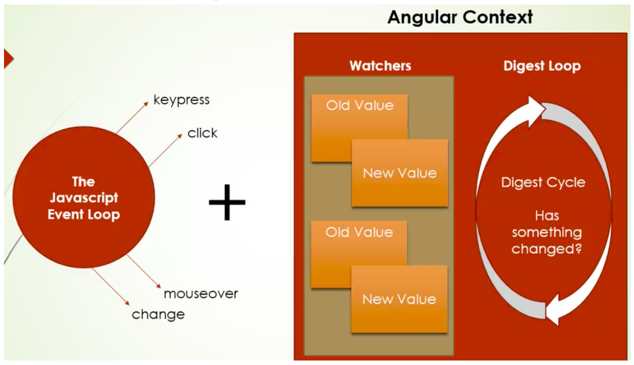


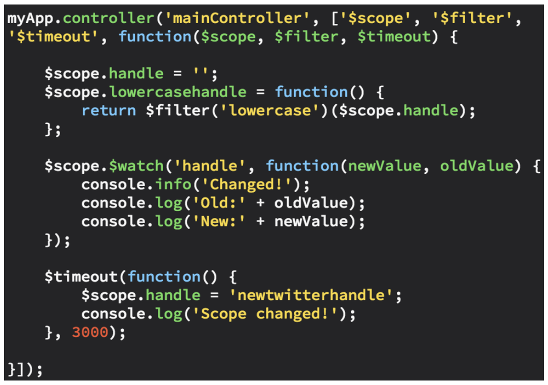
Tt

```
2 Pressed!
```

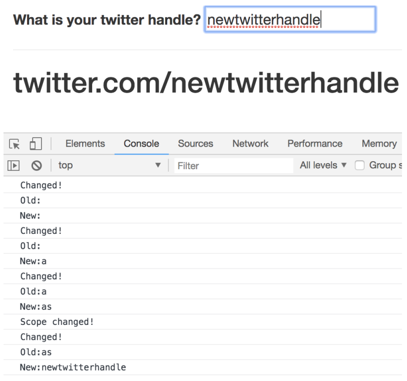
This event just occurred, and JS has smtg called Event Loop, which basically means it sits there and waits for events to get thrown. It throw events all in this kind of continuing cycle. We have got EventListener waiting for the 'keypress' event on the text box. The browser throw this event when we press the key while inside the text box.

Watchers and Digest Loop





```
myApp.controller('mainController', ['$scope', '$filter',
'$timeout', function($scope, $filter, $timeout) {

    $scope.handle = '';
    $scope.lowercasehandle = function() {
        return $filter('lowercase')($scope.handle);
    };

    $scope.$watch('handle', function(newValue, oldValue) {
        console.info('Changed!');
        console.log('Old:' + oldValue);
        console.log('New:' + newValue);
    });

    $timeout(function() {
        $scope.handle = 'newtwitterhandle';
        console.log('Scope changed!');
    }, 3000);

}]);
```



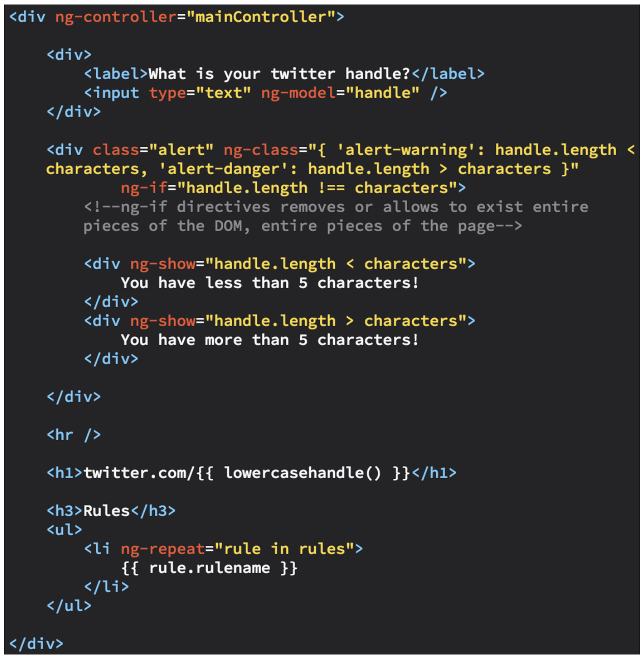
What is your twitter handle? newtwitterhandle

twitter.com/newtwitterhandle

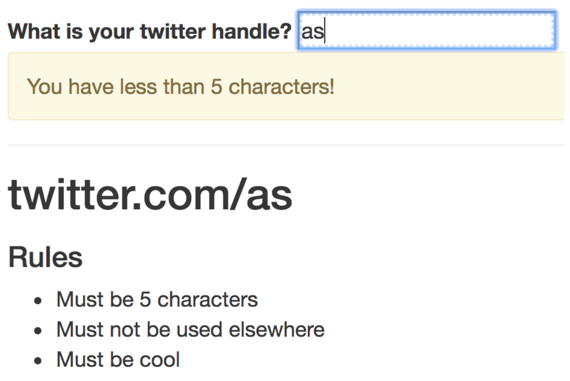
```
Changed!
Old:
New:
Changed!
Old:
New:a
Changed!
Old:a
New:as
Scope changed!
Changed!
Old:as
New:newtwitterhandle
```

AngularJS is attaching events and looking at variables. Every time something changes or something even could have changed, it checks all those values and then updates the page and the other variables for me.

Common Directives



```
<div ng-controller="mainController">

    <div>
        <label>What is your twitter handle?</label>
        <input type="text" ng-model="handle" />
    </div>

    <div class="alert" ng-class="{ 'alert-warning': handle.length <
    characters, 'alert-danger': handle.length > characters }"
            ng-if="handle.length !== characters">
        <!--ng-if directives removes or allows to exist entire
        pieces of the DOM, entire pieces of the page-->

        <div ng-show="handle.length < characters">
            You have less than 5 characters!
        </div>
        <div ng-show="handle.length > characters">
            You have more than 5 characters!
        </div>

    </div>

    <hr />

    <h1>twitter.com/{{ lowercasehandle() }}</h1>

    <h3>Rules</h3>
    <ul>
        <li ng-repeat="rule in rules">
            {{ rule.rulename }}
        </li>
    </ul>

</div>
```



What is your twitter handle? as

You have less than 5 characters!

twitter.com/as

Rules

- Must be 5 characters
- Must not be used elsewhere
- Must be cool

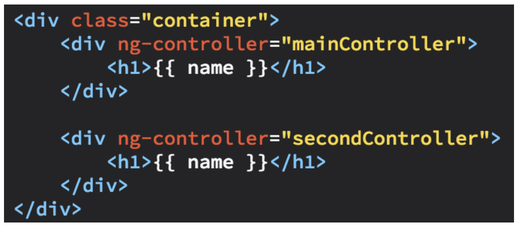
SINGLE PAGE APPLICATIONS (SPA)
Single Page Apps is an important part of AngularJS, one html page and dynamically updates it, so it does not require to reload the web page.

SPA is using AJAX method, which is asynchronous JS XML. Exchange the data without refresh the whole page.

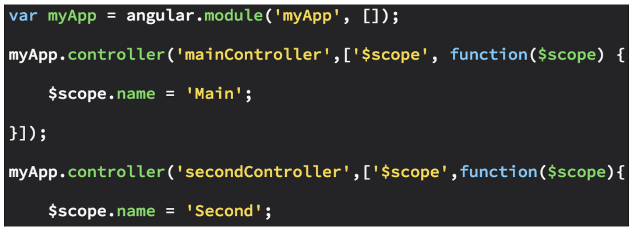
Thanks to AngularJS's dependency injection because we can use Multiple Controller and multiple views with using SPA.

SPA involves several fundamental concepts.
Routing: ng-route it is a module. It is a router, help us route whatever is in the hash, and then run the proper code and grab the proper HTML.
Templates, **and**
Controllers

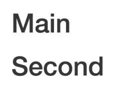
Angular Aside - Multiple Controllers, Multiple Views



```
<div class="container">
    <div ng-controller="mainController">
        <h1>{{ name }}</h1>
    </div>

    <div ng-controller="secondController">
        <h1>{{ name }}</h1>
    </div>
</div>
```



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

myApp.controller('mainController',['$scope', function($scope) {

    $scope.name = 'Main';

}]);

myApp.controller('secondController',['$scope',function($scope){

    $scope.name = 'Second';
```

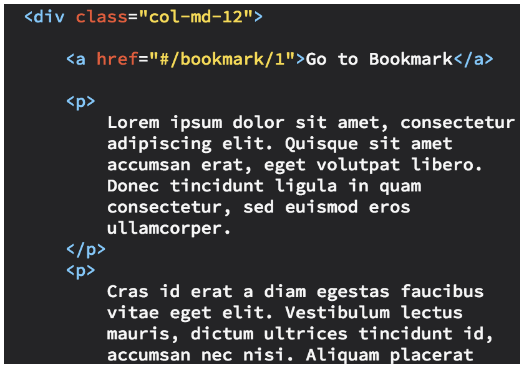


Main

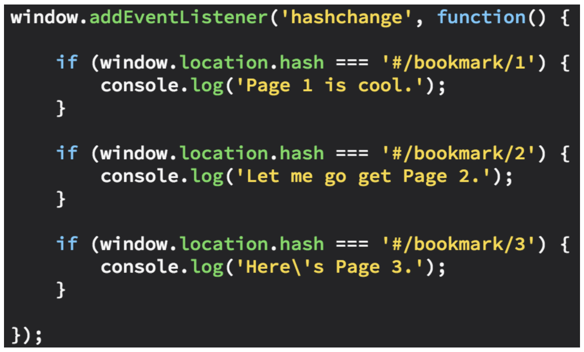
Second

Thanks to AngularJS's dependency injection. We asked for a scope object by placing it into the function but AngularJs actually gives us a new version of the object, a new instance of the scope object, For each time that we request it in a separate controller. Even if we put the same property name on it in the instance and we outputted it similarly <h1>{{ name }}</h1>. So the templates we created are connected individually via the ng-controller. This is how AngularJS implements the idea of single page applications.

HTML and JS Aside - Single Page Apps and the Hash



```html
<div class="col-md-12">

    <a href="#/bookmark/1">Go to Bookmark</a>

    <p>
        Lorem ipsum dolor sit amet, consectetur
        adipiscing elit. Quisque sit amet
        accumsan erat, eget volutpat libero.
        Donec tincidunt ligula in quam
        consectetur, sed euismod eros
        ullamcorper.
    </p>
    <p>
        Cras id erat a diam egestas faucibus
        vitae eget elit. Vestibulum lectus
        mauris, dictum ultrices tincidunt id,
        accumsan nec nisi. Aliquam placerat
```

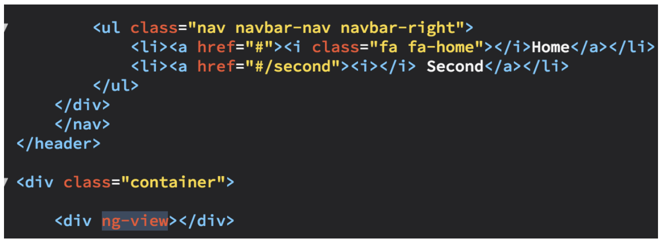


```javascript
window.addEventListener('hashchange', function() {

    if (window.location.hash === '#/bookmark/1') {
        console.log('Page 1 is cool.');
    }

    if (window.location.hash === '#/bookmark/2') {
        console.log('Let me go get Page 2.');
    }

    if (window.location.hash === '#/bookmark/3') {
        console.log('Here\'s Page 3.');
    }

});
```

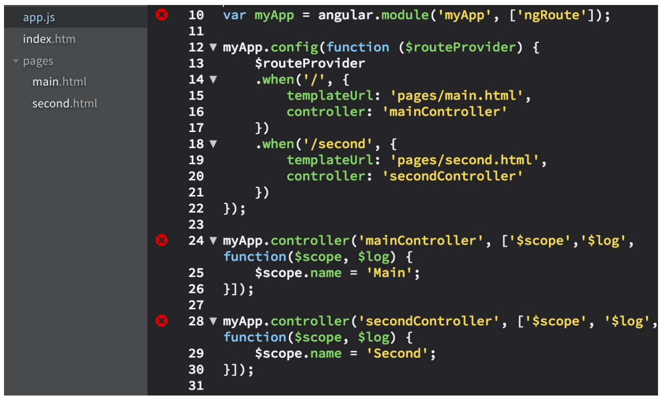
Routing, Templates, and Controllers

Single Page Apps involves several fundamental concepts.

Routing: It is a router, help us route whatever is in the hash, and then run the proper code and grab the proper HTML. It is a module ngRoute and provides a couple of different elements such as a routeProvider.



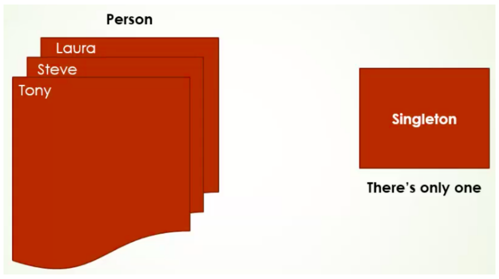
```html
        <ul class="nav navbar-nav navbar-right">
            <li><a href="#"><i class="fa fa-home"></i>Home</a></li>
            <li><a href="#/second"><i></i> Second</a></li>
        </ul>
    </div>
    </nav>
</header>

<div class="container">

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



```javascript
var myApp = angular.module('myApp', ['ngRoute']);

myApp.config(function ($routeProvider) {
    $routeProvider
    .when('/', {
        templateUrl: 'pages/main.html',
        controller: 'mainController'
    })
    .when('/second', {
        templateUrl: 'pages/second.html',
        controller: 'secondController'
    })
});

myApp.controller('mainController', ['$scope','$log',
function($scope, $log) {
    $scope.name = 'Main';
}]);

myApp.controller('secondController', ['$scope', '$log',
function($scope, $log) {
    $scope.name = 'Second';
}]);

```

JS and Angular Aside - Singletons and Services

Singleton: The one and only copy of an object. It is a pattern in OOP, it means I only have one of these objects ever. $scope (child) is an exception of this rule.



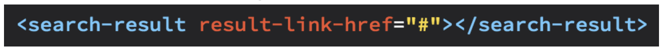
Even though, we are navigating from page-to-page, we are inside a SINGLE PAGE APPLICATION, we are still inside the SAME JS MEMORY SPACE. I have all the variables, content, and values that we had when we started. Even after I navigate, so that means I can share the content across pages, as well as use the services to encapsulate and functionality that I use across different controllers.

HTML Aside - Reusable Components

There is no quick way to make your own tags in HTML. AngularJS brings us it with Custom Directives.

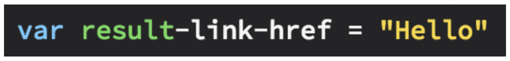
JS Aside - Normalization

We create our own tag;



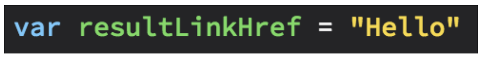
```html
<search-result result-link-href="#"></search-result>
```

There was an error here, JS read "-" symbol as a minus;



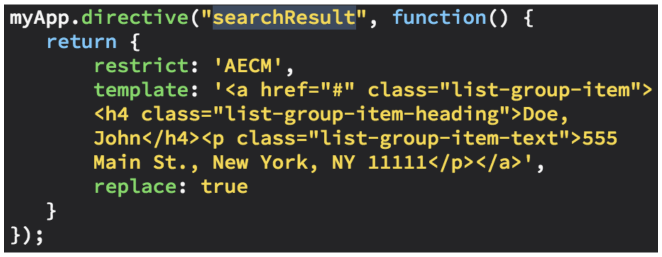
```javascript
var result-link-href = "Hello"
```

So, solution is;

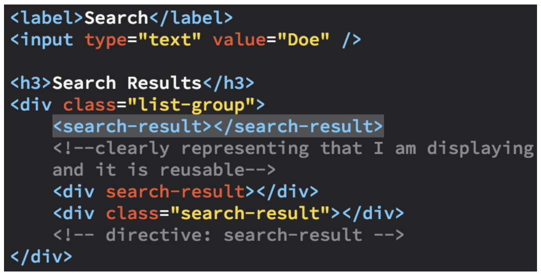


```javascript
var resultLinkHref = "Hello"
```

Creating a Directive

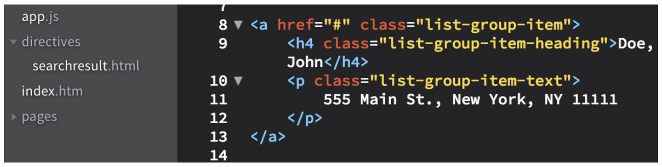


```
myApp.directive("searchResult", function() {
   return {
       restrict: 'AECM',
       template: '<a href="#" class="list-group-item">
       <h4 class="list-group-item-heading">Doe,
       John</h4><p class="list-group-item-text">555
       Main St., New York, NY 11111</p></a>',
       replace: true
   }
});
```

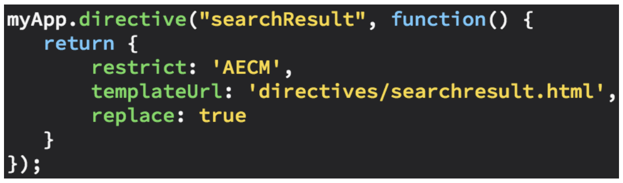


```
<label>Search</label>
<input type="text" value="Doe" />

<h3>Search Results</h3>
<div class="list-group">
    <search-result></search-result>
    <!--clearly representing that I am displaying
    and it is reusable-->
    <div search-result></div>
    <div class="search-result"></div>
    <!-- directive: search-result -->
</div>
```

Templates



```
<a href="#" class="list-group-item">
    <h4 class="list-group-item-heading">Doe,
    John</h4>
    <p class="list-group-item-text">
        555 Main St., New York, NY 11111
    </p>
</a>
```



```
myApp.directive("searchResult", function() {
   return {
       restrict: 'AECM',
       templateUrl: 'directives/searchresult.html',
       replace: true
   }
});
```

<u>SOME TERMS:</u>

Compile and Link: When you're writing code, and you build it, the <u>compiler</u> converts that code into a lower-level language, then the <u>linker</u> generates a file the computer will actually interact with.

Transclusion: Include one document inside another. Place a copy of one document at a particular point inside another.

Transpile: Converting source code of one programming language into another.