Software Craftsmanship Guild

526 S. Main St, Suite 609

Akron, OH 44311

AngularJS Fundamentals

Software Craftsmanship Guild

Lesson Goals

Learn about SPA (single page applications) and how Google's AngularJS Framework can help with the complexity.

What is a SPA App?

Single Page Applications are pages that contain multiple views that are shown and hidden dynamically based on events.

SPAs do not do full postbacks to the server to fully reload pages. They request data via ajax in the background.

Gmail is a good example of a complex SPA application.

SPA Challenges

- DOM Manipulation
- History
- Module Loading
- Routing
- Caching
- Object Modeling
- Data Binding
- Ajax/Promises
- View Loading

AngularJS Features

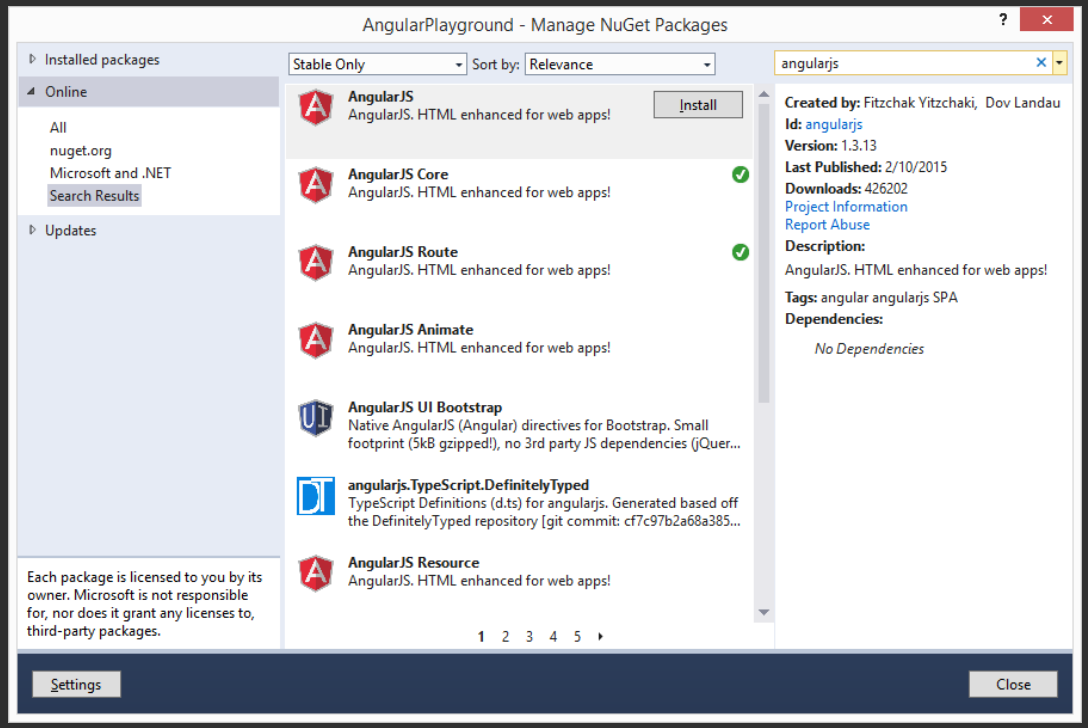
- Data Binding
- MVC
- Routing
- Testing
- View Models
- Views
- Controllers
- jqLite
- Templates
- History
- Factories
- Directives
- Services
- Dependency Injection
- Validation

Download AngularJS

AngularJS is available on NuGet. The first package has all of the modules, but if you're just doing some basic things you only need Core and if you are doing dynamic views you also need Route.

For these examples we will only be using Core and Route.

Note that angular does a lot of plug-ins so there are many things you can download, similar to jQuery.



Add angular to the layout

We will be using AngularJS in all of our pages, so first let's update our _Layout to include the angular scripts.

```
<script src="~/Scripts/jquery-2.1.3.min.js"></script>
<script src="~/Scripts/bootstrap.min.js"></script>
<script src="~/Scripts/angular.min.js"></script>
<script src="~/Scripts/angular-route.min.js"></script>
@RenderSection("scripts", false)
```

Directives

A directive is a way to wire up angular to html elements to extend their behavior.

Angular directives by convention are named ng-* where * is the directive name.

For a simple example, we're going to add a ng-app directive to our row div and tag a text input with a ng-model directive called name.

Then anywhere in our html we can use double braces {{ name }} to bind to the value of the textbox model. This is called a "data binding expression"

```
<div class="row">
    <div class="col-sm-4">
        <p>
            Here we can see how a ng-app and ng-model
            can be added to a page to set up two way binding
        </p>
    </div>
</div>
<div class="row" ng-app>
    <div class="col-sm-4">
        <label>Name: </label>
        <input type="text" ng-model="name" />
        <p>Your name is: {{name}}</p>
    </div>
</div>
```

The ng-repeat Directive

The ng-repeat directive instructs angular to repeat an html element for every item in a collection.

Here we have initialized so data (ng-init creates default data on page load) and we want to loop through that array and create a list item for each element in the array.

```
<div class="row" ng-app ng-init="primaryColors=['red','yellow','blue']">
    <div class="col-sm-4">
        <ul>
            <li ng-repeat="color in primaryColors">{{ color }}</li>
        </ul>
    </div>
</div>
```

ng-repeat is very similar to a foreach loop.

All directives are in the angularjs documentation:

http://docs.angularjs.org/api/ng.directive:ngRepeat

Using Filters

In a data binding expression, we can use the pipe character to use filters that are built into angularjs.

Let's say we have an array of friend objects. We can use filters to use the value of a text input to filter the list and perform other common tasks like ordering data.

We can also put filters on data binding expressions.

Try doing {{ friend.name | uppercase }}

There is a list of filters in the documentation with examples.

```
<div class="row" ng-app ng-init="friends=[{name:'John', phone:'555-1212', age:10},
      {name:'Mary', phone:'555-9876', age:19},
      {name:'Mike', phone:'555-4321', age:21},
      {name:'Adam', phone:'555-5678', age:35},
      {name:'Julie', phone:'555-8765', age:29}]">
    <div class="col-sm-4">
        <input type="text" ng-model="friendFilter">
    </div>
    <div class="col-sm-4">
        <ul>
            <li ng-repeat="friend in friends | filter:friendFilter | orderBy:'name'">
            {{ friend.name }}, {{ friend.phone }}, age: {{ friend.age }}</li>
        </ul>
    </div>
</div>
```

Views, Controllers, and Scope

So far we have been working only in views. Views contain all your formatting and data binding syntax.

Controllers are intended to contain all your calculations and business logic.

Scope, or $scope as angular calls it, is the transport object that shuttles information back and forth between the controller and view.

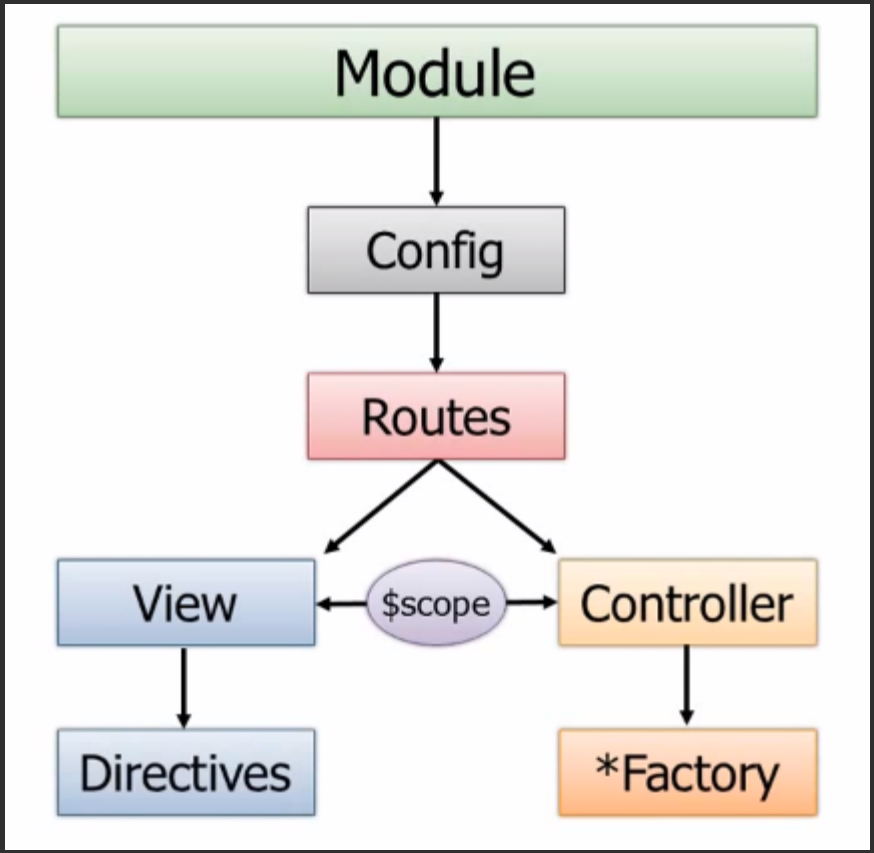
Modules and Routes

Modules in AngularJS are responsible for configuring routes.

Routes can register views and controllers for a given URL.

Controllers often call out to factory or service classes to handle their CRUD operations.

Modules contain all these things and are referenced by name in ng-app



* Image from Dan Wahlin

Creating a Module

The angular.module command creates a named module and allows for passing in an array of other modules if you want to build in complex dependencies.

Now all we need to do is add the controller to the module and then reference demoApp as the ng-app.

It's typical to have many controllers in a module, particularly in a singe page application.

```
<div class="row" ng-app="demoApp">
    <div class="col-sm-4" ng-controller="SimpleController">
        <ul>
            <li ng-repeat="player in players">
                {{ player.name }} - {{ player.city }}
            </li>
        </ul>
    </div>
</div>


@section scripts
{
    <script>
        var myApp = angular.module('demoApp', []);

        myApp.controller('SimpleController', function ($scope) {
            $scope.players = [
                { name: 'Jordan Cameron', city: 'Cleveland' },
                { name: 'Drew Brees', city: 'New Orleans' },
                { name: 'Calvin Johnson', city: 'Detroit' },
                { name: 'Marshawn Lynch', city: 'Seattle' }
            ];
        });

    </script>
}
```

Routes and AJAX

When you want to dynamically load views with Angular you can load the routing module.

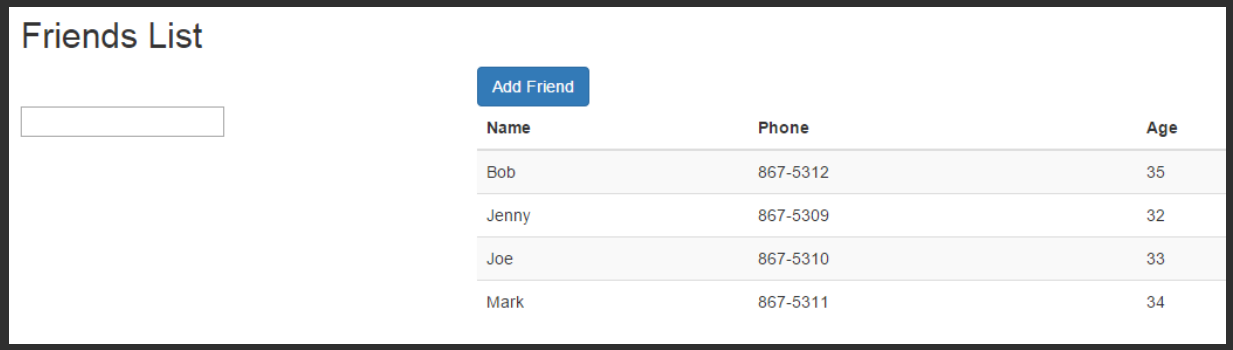
Naturally in a real application we will also want to request our data from the server instead of hard coding it as we have been.

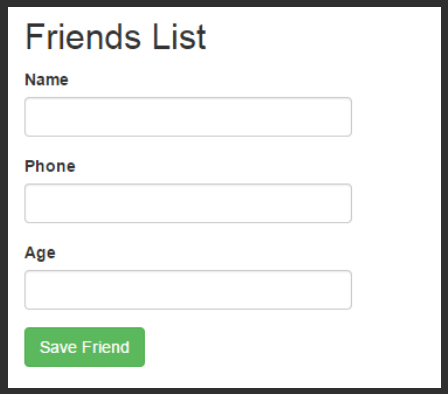
So let's take a look at how that works in practice.

Our Goal

Create a filterable friends list that also allows us to add a friend. When we click the Add Friend button we want it to switch to the add form without reloading the whole page (like Gmail).

We want to load and save data via WebAPI as well.





Setting up the server-side

Let's add a new Web API controller for our friends and use a fake database class as well as create a model for our friends.

```
public class FakeFriendsDb
{
     private static readonly List<Friend> _friends = new List<Friend>();

     static FakeFriendsDb()
    {
        _friends.AddRange(new[]
        {
            new Friend() {Age = 32, Name = "Jenny", Phone = "867-5309"},
            new Friend() {Age = 33, Name = "Joe", Phone = "867-5310"},
            new Friend() {Age = 34, Name = "Mark", Phone = "867-5311"},
            new Friend() {Age = 35, Name = "Bob", Phone = "867-5312"},
        });
    }

    public List<Friend> GetAll()
    {
        return _friends;
    }

    public void Add(Friend friend)
    {

        _friends.Add(friend);
    }
}
```

```
public class Friend
{
    public string Name { get; set; }
    public string Phone { get; set; }
    public int Age { get; set; }
}
```

Setting up the Controller

For this example we just want to get the list of our friends and be able to post a new friend. So we will create a simple new WebAPI controller and create a Get and Post like so:

```
public class FriendsController : ApiController
{
    public List<Friend> Get()
    {
        var db = new FakeFriendsDb();
        return db.GetAll();
    }

    public HttpResponseMessage Post(Friend friend)
    {
        var db = new FakeFriendsDb();
        db.Add(friend);

        return new HttpResponseMessage(HttpStatusCode.OK);
    }
}
```

Using AJAX in Angular

Where jQuery has $.ajax, Angular has $http to do Ajax requests with.

When we create a new Angular module, we can register factory methods that create objects for us. Our goal is to create a factory method that returns an object that has functions to use $http to communicate with our WebAPI.

Notice in the angular.module statement we are informing angular that we will be using the ngRoute plugin. If you have multiple plugins you just comma separate them.

* notice how simple ajax calls are with angular.

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

myApp.factory('friendsFactory', function ($http) {
    // create a new object
    var webApiProvider = {};

    var url = '/api/Friends/';

    // attach a get request to the object
    webApiProvider.getFriends = function () {
        return $http.get(url);
    };

    // attach a post request to the object
    webApiProvider.saveFriend = function (friend) {
        return $http.post(url, friend);
    };

    // the object is ready to use, return it to
    // whatever controller needs it.
    return webApiProvider;
});
```

Handling Routes

Routes allow our app to dynamically load different views into our page based on what the user does.

In order to enable routing, we must download the angular-route.js files into our script folders (nuGet).

Don't forget to link angular-route.js into your shared layout page!

When we use angular routes, we reference an object called $routeProvider. Angular looks at your function parameters and attempts to fill them with objects. Ones that are built in tend to start with $ while the ones we create (like friendsFactory) tend not to.

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

myApp.factory('friendsFactory', function ($http)...);

myApp.config(function($routeProvider) {
    $routeProvider
        .when('/Routes',
        {
            controller: 'FriendsController',
            templateUrl: '/AngularViews/FriendsList.html'
        })
        .when('/AddFriend',
        {
            controller: 'AddFriendController',
            templateUrl: '/AngularViews/AddFriend.html'
        })
        .otherwise({ redirectTo: '/Routes' });
});
```

Breaking it down

The .config() method of an angular app allows us to set things up when the app is loaded. Think of it like a constructor since it does configuration setup.

The route provider will look at the URL in the browser and when it is /Routes (our starting URL) it will load the html from /AngularViews/FriendsList.html and place it into the page.

If the URL changes to /AddFriend it will remove the FriendsList.html and replace it with AddFriend.html.

Each template is assigned a controller that will handle its data and interactions.

```
myApp.config(function($routeProvider) {
    $routeProvider
        .when('/Routes',
        {
            controller: 'FriendsController',
            templateUrl: '/AngularViews/FriendsList.html'
        })
        .when('/AddFriend',
        {
            controller: 'AddFriendController',
            templateUrl: '/AngularViews/AddFriend.html'
        })
        .otherwise({ redirectTo: '/Routes' });
});
```

Adding the FriendsController

Notice in our $routeProvider we assigned our FriendsList to the FriendsController. Let's add the controller.

In the controller, we will need access to our ajax factory (friendsFactory). In angular if you just make a function parameter with the same name you registered it with, angular will automatically inject the right object for you.

We will invoke our getFriends function and on success put the data into the controller scope so the page can bind it.

```
myApp.controller('FriendsController', function ($scope, friendsFactory) {
    friendsFactory.getFriends()
        .success(function(data) {
            $scope.friends = data;
        })
        .error(function(data, status) {
            alert('oh noes!  status: ' + status);
        });
});
```

Adding the AddFriendController

The controller for the entry form is a bit more complicated. When we are done adding the new friend, we want to redirect to the other view. The $location object in angular allows us to set a path and trigger the view change.

Here we also see how to register a function, save() to the scope so that we can bind it to our button in the page.

We also can re-use the friendsFactory object for our ajax.

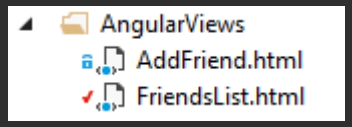
```
myApp.controller('AddFriendController', function ($scope, $location, friendsFactory) {
    $scope.friend = {};

    $scope.save = function() {
        friendsFactory.saveFriend($scope.friend)
            .success(function() {
                $location.path("/Routes");
            })
            .error(function (data, status) {
                alert('oh noes!  status: ' + status);
            });
    }
});
```

Handling Routes – Adding Views Part 1

Now let's add our views. In the Route provider, we said we would have a FriendsList.html and a AddFriend.html in the /AngularViews/ folder of our website.

Notice the add friend link. We've used bootstrap to make it look like a button, but the href is #/AddFriend which will allow the local page's route provider to pick it up and know to load the AddFriend view instead of making a server request.



FriendsList.html

```
<div class="row">
    <div class="col-xs-8 col-xs-offset-4">
        <a href="#/AddFriend" class="btn btn-primary">Add Friend</a>
    </div>
</div>
<div class="row">
    <div class="col-xs-4">
        <input type="text" ng-model="friendFilter" />
    </div>
    <div class="col-xs-8">
        <table class="table table-striped">
            <thead>
                <tr>
                    <th>Name</th>
                    <th>Phone</th>
                    <th>Age</th>
                </tr>
            </thead>
            <tbody>
                <tr ng-repeat="friend in friends | filter:friendFilter | orderBy:'name'">
                    <td>{{friend.Name}}</td>
                    <td>{{friend.Phone}}</td>
                    <td>{{friend.Age}}</td>
                </tr>
            </tbody>
        </table>
    </div>
</div>
```

Handling Routes – AddFriend.html

The second view will be our form to add a friend to our database. Notice the ng-click directive that wires up our save() function in the controller.

```
<div class="row">
    <div class="col-xs-4">
        <div class="form-group">
            <label>Name</label>
            <input type="text" class="form-control" ng-model="friend.Name"/>
        </div>
        <div class="form-group">
            <label>Phone</label>
            <input type="text" class="form-control" ng-model="friend.Phone" />
        </div>
        <div class="form-group">
            <label>Age</label>
            <input type="text" class="form-control" ng-model="friend.Age" />
        </div>
        <button class="btn btn-success" ng-click="save()">Save Friend</button>
    </div>
</div>
```

Last Step

Wherever in our page we decide to inject the views, we simply use the ng-view directive as a placeholder.

```
@{
    ViewBag.Title = "Routes";
}

<h2>Friends List</h2>

<div ng-app="friendsApp" ng-view="">

</div>
@section scripts
{

    <script src="~/Scripts/app/routes.js"></script>
}
```

Conclusion

SPA frameworks like Angular, Ember, Durandal, etc. can bring a lot of convenience to the table.

There is a lot of “magic” that goes on in them and it definitely takes time and effort to wrap your head around them.

Once you get your head around them though, you can do some really amazing things with not much code at all.