Simple AngularJS thanks to Best Practices

Learn AngularJS the easy way

Level 100-300

CONNECT
DNN-Connect Association

What’s this session about?

1. AngularJS can be easy
   → when you understand basic concepts and best practices
2. But it can also be messy and difficult
   → if you follow most online examples for just about anything

→ I had to learn this the hard way ☺

→ I want to make it easier for you

→ With know-how and sample code

Quick Bad-Example Code

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

phonecatApp.controller('PhoneListCtrl', function ($scope, $http) {
  $http.get('phones/phones.json').success(function(data) {
    $scope.phones = data;
  });

  $scope.orderProp = 'age';
});
```

Taken from AngularJS documentation. This example use the old $scope concept which you should avoid nowdays.

Don't worry if you don't understand this – it's meant to keep advanced developers in the room ☺.

1. Demo
2. What is AngularJS
3. Deep Dive and Learn

Goal: Get to know AngularJS based on a real App built with as many simple best practices as I could think of.

Example AngularJS App

Think of a DNN-Module, just simpler

Live Demo

Feedback given by anonymous users
Feedback management for Admin users
Download current edition http://2sxc.org/apps

SoC Server and Client

Server Concerns

- Storage of Data
- Ability to deliver Data when needed
  - Sorted Categories
  - All/one Feedback Items
- Ability to do other CRUD
  - Create, Read, Update Delete
- REST
- Permissions on Content-Types (allow create…)

→ No code (zero c#)

Client Concerns

- Form UI for new items
- Instructs Server what to do (Create, Read, …)
- List-UI for admin
- Change UIs when editing
  - Messages
  - Dialogs
  - refresh

→ All code is on the client

Look at the wire (live glimpse)

Let's watch these processes

- Get Categories
- Create Feedback-Item
- Get All Feedback Items
- Status-Update Feedback-Item
- Update/Edit Feedback
- Delete

→ Standard REST calls
→ Special DNN headers

Recommended Tools

- Chrome Debug
- Firebug

Favorite: Fiddler (free, originally by Microsoft, now by Telerik)

- Allows detailed analysis of everything
- Allows live testing (modify request etc.)

Quick look at the Backend

- Data
  - Content-Types
  - REST
  - security
- Query to get sorted categories

That’s it ☺

So what is AngularJS?

AngularJS is a System to create simple and complex JS Apps

Think of MVC or any other MV** - live, not disconnected
on the server using ASP, PHP, Ruby…

Core things in solves

1. Splitting the App into small, clearly defined blocks
2. Conventions on solving many specific challenges like
   1. DOM control, Data Binding
   2. Communicating with the Server
   3. SoC and IoC (Dependency Injection)
3. Clear pattern based work style
4. Best-Practices on writing JavaScript code

(more features: url, testing, extensions, etc.)

#1 Splitting functional blocks (parts of Angular Apps – usually grouped)

Views
- Feedback Form
- Admin Form
- Partial Forms

Controllers
- Create Controller
- Admin Controller
- Edit Controller

Services
- Categories Service
- Feedback Service
- Content Service
- $http Service

AngularJS glues it all together (aka compiles). At runtime it's called an App. AngularJS provides additional features (built-in services, directives etc.)

Special DNN-Aware Initializers / Helpers
Extremely important to keep everything else simple

A simple View / Controller

```
<h2>These is where the categories will appear</h2>

<div sxc-app="AngularRestDemos" id="app-416" ng-controller="GetList as vm">
    <div>{{ vm.message }}</div>
    <ol>
        <li ng-repeat="c in vm.items"> {{ c.Title }} </li>
    </ol>
</div>
```

- Very easy to understand
- Clearly separates view from view-model
- View only operates with data provided by controller and doesn't try to retrieve data

```
function GetListController($http) {
    var vm = this;
    vm.message = "loading";

    $http.get('app-content/Feedback Category').then(function (result) {
        vm.items = result.data;
        vm.message = "The $http.get(...) worked - found " + result.data.length
            + " categories, the first one is '" + result.data[0].Title + "'";
    });
};
```

Comparing it to jQuery, knockoutJS etc.

Summary: jQuery is becoming obsolete thanks to new browsers; knockoutJS does about 20% of AngularJS.

In Detail: later, when you understand more…

Visions

1. Super fast AJAX Web Apps
2. Server-Agnostic: Create an App for DNN, re-use it in Drupal, Umbraco, DNN-X etc.
3. Almost no Server-Code
   1. Server doing 80%-100% with standard REST
   2. and maybe 20% with custom WebAPI
4. Google-Indexable JS Apps!

→ Read blogs about REST, Google/JS etc.

SoC Pattern
Separation of Concerns

Splitting JavaScript Apps into small parts is key to maintainable code

Jungleboy wisdom #1

But don’t overdo it ☺

Jungleboy Wisdom #2

Don’t create a service for a 1-line of Content-Get

Spend more time on architecture, less on code

Jungleboy wisdom #3

Break: Help us with 2sxc!

We need people passionate about bootstrap, knockoutJS, Ember, content-design, css3, … for the community

#1 Discover Views and View Best Practices

View Template for New Feedback (live)

Basics to understand

1. View handles lots of simple stuff like
   1. Show/Hide; if, switch, etc. *Directives*
   2. Data-Binding of "Core data" = Feedback
   3. Data-Binding of Helper-Data (Categories)
2. Very simple stuff is very simple (just like with knockoutJS, …)
3. Binding is "real" and works with dirty-checking = more performant in real-live scenarios than Observers in knockoutJS

Lessons Learned / Best Practices

1. Use simple HTML & CSS
2. View-Stuff is easy, the list of basic features is rather short/simple
3. Some things are a bit trickier
   1. Drop-down binding has many features, you have to figure out which variation you want
   2. Use custom Directives for anything advanced (SoC)

Best Practices

1. Use *Controller As* Syntax; avoid $scope
2. Try to work with a VM-item that is built like the server read/writes it in the REST
3. IMHO this is the only place where you should use the ModuleId (in an attribute).

Discover the Controller of New Feedback (Live)

Controllers Initialize VMs

```
function NewController(categoriesSvc, feedbackSvc) {
    // Training Notes: use the vm = this as recommended by
    var vm = this;
    vm.show = "form";
    vm.item = feedbackSvc.new();

    // ensure the categories are loaded and select the firs
     categoriesSvc.load().then(function (result) {
         vm.item.Category.Id = categoriesSvc._all[0].Id;
     });

    // the send-method, will first show "sending", then a thank-you message
    vm.save = function save() {
        vm.show = 'sending';
        feedbackSvc.add(vm.item).then(function () {
            vm.show = 'thanks'
        })
    }
};
```

Build / Create new View Model. Controller only does that; it's not needed any more after this one-time call.

Load Primary data

Load Helper Data

defines the actions available and attaches them to the view-model

Summary: It’s in charge of initializing one view-model (data, helper-data, actions)

Lessons Learned / Best Practice

1. Use vm = this; to support Controller As
2. Use simple Dependency Injection
3. The controller should do View-Model initialization and *nothing else*.
4. Avoid $scope in your controller and all DOM-manipulations.
   1. the $scope is the most common source for bugs, because it's much more complex than it look (partial scopes, etc.)
   2. Passing $scope around get's very messy and is against SoC and it's not necessary…
   3. It's a bit like a virus – like static methods in C#

Lessons Learned / Best Practice

1. Don’t access DNN-Internal stuff (like ServiceFramework) in your controller.
   1. Unofficial dependencies = bad
   2. Wrong SoC – should be in a service
2. Avoid using the module-ID in your controller
   1. Not for DOM-coding (you shouldn’t need $(…mid…) in the controller – use data binding)
   2. Not for $http config (not in controller)
   3. Not for initializing a service (use dependency injection)

Service – in charge of Data (Live: Categories Service)

Basics

1. The Service is in charge of Data (or other things like a toastr, calculations, interface to DNN, …)
2. It can do get/put
3. It can also keep data-states (like a cached list of categories)

Lessons Learned / Best Practices

1. Create a service for each purpose (categories, feedback, etc.) – it's simple and easy to manage
2. Keep resource calls simple – DON'T handle DNN-Specials at Service level…
3. So you usually don't need the ModuleID!
4. Rely on IoC (Dependency Injection) to give you a $http which already works

Advanced Topics

Application Initialization and SPA vs. MAP

Basic Concept

1. Initializing an App assembles all parts of the App together:
   1. Services, Views, Controllers
   2. Depended-On-Services like $http, …
2. AngularJS could start you App automatically, but it shouldn't, because
   1. DNN is strong in multiple "things" per page – which is a very common web-scenario – think of single-pagers. SPAs are not the normal scenario at all!
   2. Your Apps need DNN-specific Dependencies to work

What should Initialization do better?

See the live code 2sxc4ng.js

1. Initialize multiple Apps per Page
2. Provide important DNN-infos like the Module-ID to the app-parts; also handle mid-in-URL
3. Reconfigure $http to handle DNN specials
   1. Correct DNN-API URLs
   2. Include anti-forgery tokens
   3. Include ModuleId in header etc.

Solutions

1. Do your own bootstrapping…
2. …or use an advanced automatic bootstrapping like we did in 2sxc

I strongly recommend that you run your AngularJS within 2sxc (saves you a lot of work) OR that you imitate what we did there in the bootstrapping.

Dependencies and Dependency Injection (IoC)

Concept

1. Very often a part of your code needs other things (it depends on other code).
    1. Context like ModuleId
    2. Shared code/objects like code to access the server (Services)
2. The “old” way to do that is to pass this around: CreateFeedback(*moduleId*, data)
3. This doesn’t scale – as it needs more and more parameters for each call – often just to pass on to deeper calls

Solution: Dependency Injection

- All systems are “available when needed” – because another system (usually called the Dependency Injector) can supply them on demand.
- In AngularJS each module (view, service, etc.) can be specified as dependency, so Angular can coordinate that they are available.
- So new: CreateFeedback(data);

Example with $http

- $http is a service, the standard AngularJS XHR (AJAX) component.
- Other services register the need for using it. The *DO NOT NEED TO KNOW THAT $http IS DIFFERENT IN DNN*. The DI will automatically deliver the “right” $http.
- Any DNN AngularJS App which re-configures $http in it’s own services does it wrong by not adheering to SoC.

Summary

Summary

- Start small, don't over-engineer. Just be flexible enough to grow as needed.
- SoC: Keep Concerns very, very separate
- *Controller As* instead of $scope
- Read John Papas Best Practices
- SPA → MAP
- Use Dependency Injection
- Use JSON-View-Model for client, separate from helper data
- Give to Ceasar…:
Let the server do data, let the client do view
- And don't worry too much about MVC on the server ☺

Questions?

Offline Web Sites / Apps

@ 11:00

Vs. knockoutJS