Angular JS for the Real World

Why AngularJS

- Advantages
  - Speed up web development
  - Specifically designed for building SPA
  - Don’t need to refresh the browser to see updated content
    - Loaded dynamically in real time
  - Declarative, not imperative
    - Tells what needs to be shown and not how to do that
  - MVC done right and 2-way data binding
  - Extensible
    - Directives can be used to create your own custom elements and attributes
    - Reusable code across the application
      - Directive can be used to create your own custom elements and attributes
      - Reusable code across the application
  - Template
    - Written in traditional HTML plus custom angular elements and attributes
    - What need to shown. Angular thinks about the how
  -

MV*

- MVC Pattern
  - UI (view) is separated from the business data (model)
  - The Controller handles the business logic of our application and is responsible to update the Model
  - The Changes in the model are reflected in the view
- MVVM
  - A ViewModel exposes the application data (Model) to the View in a meaningful way
  - The Model represent our data. The model can have data in its raw state (example holding a UNIX time stamp instead of a human readable timestamp).
  - The View is the UI
  - The DataBinding happens between the View and the View Model in the ViewModel
    - The ViewModel is the State/Operations
  - Create functions to update the view

First Angular Application

- ng-app
  - A directive defines the scope of our application which defines the scope of our application
  - Can be used on any tag and limit the scope of the angular application
- Directives – offer a new way to build our HTML to crate your own specific DSL
  - HTML markers used to create new specific behaviors for DOMs Objects
  - Modify DOMs object behavior
  - Two families of directives
    - Built-in
    - Custom
  - Common Directives
    - ng-app
      - Needs a = “<name>” which defines a module
      - Module container used for the app
      - Useful to define modules where we want to rwrite different components and inject them
    - ngBind, ngModel –> Used to bind DOM object to the scope (see example on next page)
    - ngController
    - ngRepeat
    - nvView → Used with the route service to render the correct template according to the current state
    - ngShow, ngHide, ngClass → Used to show or hide DOM elements
  - The Input[Text]
    - PlaceHolder → Text show in the Entry box when the user did not type in anything.
  - Example
    -   Gets myName from the controller
  - Model → A property from your scope.
  - {{ 5 + 3}} will produce 8

First Angular Application

- Example
  - Angular Code
    - <html ng-app= “myfirstApp”>
      - <input type=”text” placeholder=”Enter your name” ng-model=”myName” >
      - <h1>Hello </h1>
    - </html>
  - JavaScript
    - var myApp = angular.module(‘myFirstApp’, []); // parameters – module name , dependencies
- A model is a property from your scope
- ng-model – binds HTML control’s value to a property on the $scope Object
- ng-bind → evaluates an expression

Controllers

- Holds the presentational logic and scope initialization for of the application
- Best way to deal with business logic is to use factories and services.
- Controllers should not 1st Column, the second column contains the better way to do it.
  - Implement business logic — Services for business logic
  - Manipulate the DOM — Data Binding and Directives
  - Format input — forms controls
  - Filter output — filters
- Scope is the data (application) model
- Can do {{ printName }} // Will call the scope.printName
- Example
  - JavaScript
    - MyApp.conftroller(‘myController’, function(scope) { // Need name and the scope variable
      - $scope.firstName = ‘Test’;
      - $scope.lastName = ‘Russo’;
      -
      - $scope.printName = functionName() { return $scope.firstName + ‘ ‘ + return $scope.lastName }
    - })
  - HTML
    - // Encloses everything we want managed from that controller
      - <div // For First Name
      - // For last name
      - {{ printName() }} // Calls a function from the controller

Scope: Our Application Data-Model

- Scope is where we are storing our application data model (our properties, our live data)
- $scope is an object that refers to the application model.
- Scope are arranged hierarchically (Prototypical Inheritance)
  - Can access Parent Controller by $parent notation
- Scope are the Intermediary between view and controllers
- Two way to display our scope inside the html
  - NgModel
    - Directive used to bind from controls to the model (data)
    - NgModel supports 2-way data binding.
  - NgBind
    - Display the values only
    - Directive that replaces the content of the specified HTML with a value of a give expression
    - Coded two ways
      -
      - {{Name}}
-

Controllers

- Example
  - HTML
    -
    - Your name:
      - <input type=”text” ng-model=”username”>
      - <button ng-click=’sayHello()’>greet</button>
      - <hr>
      - {{greeting}}
    -
  - JS
    - Angular.module(‘scopeExample’, [])
      - .controller(‘MyController’, [‘$scope’, function($scope) { // Injecting the scope
        - $scope.username = “World”;
        -
      - $scope.sayHello = function() {
        - $scope.greeting = ‘Hello’ + $scope.username + ‘!’;
      - };
    - });

Best Practice : Controller as Syntax

- Example of parent/son controller
  - HTML
    - <div ng-controller=”ParentCtrl as Prt”>
      - <h1>Hello {{parent.name}}</h1>
      - <div ng-controller=”SonCtrl as Son” >
        - <h1> {{ son.name }}, Son of { ${parent.name}}</h> // Access a value from the parent scope
- Instead of using the scope service you can use the this keyword
- Stores the controller instance in the scope property
- Example Controller as Syntax
  - Javascript
    - angular.module(‘DemoApp’).controller(‘ParentCtrl’, function() {
      - var parent = this; // this represent the current object and store it in the scope
      - parent.name = ‘John’;
    - }).controller(‘SonCtrl’, function() {
      - var son = this;
      - son.name = ‘Sam’;
    - })
  - HTML
    - < div ng-controller=”ParentCtrl as prt”> // I think this line stores the controller instance in the scope property
      - <h1>Hello {{ Son.name}}, Son of {{ parent.name}}</h1>

Services, Factories, Providers

- Code is shared using ng dependency injection
- A server is wrapper for factory and factory is a wrapper for providers
- Services
  - Where to Put the application business logic
  - Share and reuse code
  - code is shared using ng dependency injection
  - Services are lazy loaded in the applications
  - Services are Singleton Object. A single object being created will be returned as reference
- Angular provided services : http, log, timout, q (execute asynchronous functions), filter, locale, animate
- Custom Services
  - Services → Angular creates the object with the new keyword → We can access all methods and properties declare with this keyword
  - Example
    - var app – angular.module(‘myAp’, []);
    - App.factory(‘trainee’, function() {
      - Return { getTranineeName:function() { return “John Doe”; }
      - };
    - });
    - app.controller(‘myControllerWithService’, [“$scope’, ‘trainee’ function($scope, trainee) { // invoke the scope and traijnee service
      - $scope.TraineeName = trainee.getRaineeName();
      - }
    - <body stykle=font-size: 40px;”>
      - <div style=”font-size: 40px;”>
      - {pre {{ traninееName}} </pre?
    -
    -

Services, Factories, Providers

- Types
  - Services
    - Services : Angular creates the object with the new key word. For this reason we can access all properties and methods declared with the this keyword
    - app.service('myService', function() {
      - this.someValue = "some value";
      - this.getSomeValue = function() { return this.someValue; }
    - });
    - app.controller('myServiceController', function($scope, myService) {
      - $scope.something = myService.getSomeValue()
    - }
  - Factory
    - Factory will get a reference to the service object created. We can access only properties and methods returned by the factory
    - app.factory('myFactory', function() { // Whatever is defined in the srv will be returned in the SRV
      - var srv = {}
      - var thisWontBeReturned = "thisWontBeReturned";
      - srv.someValue = "SomeValue"
      - srv.getSomeValue = function() { return srv.someValue; }
      - return srv;
    - });
    - app.controller('myFactoryController, function($scope, myFactor, myFactory) {
      - $scope.printName = myFactory.getSomeValue(); // If the value is changed using 2 way bind the GUI is not updated
    - }
    - In order to have two way function working
      - $scope.printName = function() { return myFactory.getSomeValue(); }
    - In the HTML we use {{printName}}

Services, Factories, Providers

- Providers
  - The most verbose version of a Service. Can be configured (angular.config) phase
  - Example
    - app.provider(‘myProvider’, function() {
      - this.someValue = “SomeValue”;
      - this.someOtherValue =”Some other Value”;
      - this.get = function() {
        - return {
          - getSomeValue: function() { return “any value”; },
          - someOtherValue: this.someOtherValue
          - }
        - }
      - });
    - app.controller(‘myProviderController’, function($scope, myProvider) {
      - $scope.someThing = myProvider.getSomeValue();
      - $scope.someOtherValue = myProvider.someOtherValue;
- Look at 5:11 again
- Example

Best Practice : Avoiding Globals when declaring a module

- Example
  - // can remove myApp because the angular instance is declared in the global scope so use a setter
  - angular.module('myFirstApp', []) .factory('personService', function() {
    - Var person = {}
    -
    - person.printName = function(firstName, lastName) {
      - Return firstName + " " + lastName;
    - }
  - }).controller('myController', function($scope, personService) {
    - $scope.firstName = 'Trainee';
    - $scope.lastName = 'Russo';
    -
    - $scope.printName = function() { return personService.printName() }
- Move the Control into a separate file
  - angular.module('myFirstApp') with the dependency injection is a getter not declaring a module
  - In the controller.js file
  - angular.contorller('myController', function($scope, personService) {
    - $scope.firstName = "Trainee";
    - $scope.lastName = 'Russo';
    - $scope.printName = function() { return personService.printName();
  - }
- HTML
  - add script line <script src='controller.js'></script>

NgRoute and NgView

- Single View Apps → Whatever view will be rendered, our browser will never be refreshed.
- ngroute is the AngularJS core module for routining
  - Provides dynamic routing services
  - Used for deep linking (All internal links to our website).
  - It couples with the ngView directive
    - Render out content
- Setup routing during the configuration phase of the module
  - Allows to configure the routine service before the application is bootstrapped
  - We can do this because in a module's <<,config>> block we can inject constants and providers only and the routing component is a provider
  - The configuration phase begins at the very beginning and the next phase is the run phase
- NgRoute Configuration
  - Include the angular-route.min.js in the HTML
  - Add ngRoute to the module's dependency
  - declare a "config" block and inject the routeProvider
    - Example
      - app.config(['$routeProvider", function($routeProvider){
        - // Your Code here
      - }]);

NgRoute and NgView

- Angular.module(‘myFirstApp’, [])
- .config([‘$routeProvider’, function($routeProvider) { // Define our routes here
  - $routeProvider.when(‘/about’ . {
    - TemplateUrl: ‘views/about.html’, // physical path where we read our view
    - })
    - .when(‘/contact’, {
      - TemplateUrl: ‘views/contact.html’
    - })
    - .otherwise({templateUrl: ‘view/404.html’})
- }])
- In the HTML code add the routes adding the ng-href attribute to the
  - <script src=’https://code.angularjs.org/1.3.15/angular-route.js></script>
  - <a ng-href=”#/about”>About
  - The way we written our route provider is non friendly html so we the / to work
- To show the html on the same page use <ng-view></ng-view>
- Can use .otherwise({‘reirectTo: ‘/’})

NgRoute and NgView (2)

- In the HTML
  - <a ng-ref='#/about'>About
  - The way we have written our route provider, we deal with non html and non friendly urls
- Ng-view
  - Includes the rendered template of the current route int the main layout (index.html)
  - Every time the current view changes the included view changes with it according to the configuration to the $route service
  - Example
    - Put the code in the html file where you want to see the output
    - <ng-view></ng-view>
      - <div ng-view> and are also the same

Enabling HTML 5 Mode for Friendly URLs

- The Links defined by Angular have the # which is not URL
- Recap of $routeProvider
  - When → Used to map to URLs and takes two parameters templateUrl, controller (defined the the controller binded to the URL)
  - Other parameters : controllerAs, Template, resolve, redirectTo
- Location Provider Service
- Example
  - Angular.module('myFirstApp', [])
  - .config(['$routeProvider', function($routeProvider. $'locationProvider') { // Added the locationProvider Service
    - $routeProvider
    - .when('/about' . {
    - TemplateUrl: 'views/about.html', // physical path where we read our view
    - })
    - .when('/contact', {
      - TemplateUrl: 'views/contact.html'
    - })
    - .otherwise({templateUrl: 'view/404.html'}) // Use the HTML 5 History API
    - locationProvider.html5Mode(true)
    - }])
  - In the HMTL
    - In the head tag set the <base href="/"> // Use HTML 5 Mode
    - When you click on the links they now are 127.0.01:49722/contact instead of /#/contact

HTML 5 Mode and Subfolders

- The server points to the Angular project folder to be the root
-
- Experiment : You move your view (html files) from the view directory to the views/subfolder instead of view
  - Will cause a forbidden error. The problem is with the base tag, href attribute.
    - All files will use the route as the base so instead of looking in subfoler it look where href attribute is pointing
  - Then modify the base <base href="/subfolder/"> so the html files can be viewed
  - When using the ng-href you should never prifix the partial url with the base
    - The slash is overriding the base tag, href attribute (using / instead of /subfolder)
    - Solution remove the slash from the ng-href attirbute value
    - Example
      - <li class="active">Home
        - <ng-href="about">About</lib> url is localhost:/subfolder/about
      -

Route Controller and the $routeParam Service

- $routeParam can be used outside the block
- Example
  - Angular.module(‘myFirstApp’, [])
  - .config([‘$routeProvider’, function($routeProvider. $’locationProvider’) { // Added the locationProvider Service
  - $routeProvider
  - .when(‘/about:param1’)’ . {
  - TemplateUrl: ‘views/about.html’,
  - controller: ‘aboutCtrl’
  - .when(‘/contact’, {
  - TemplateUrl: ‘<h1>I am the contact page</h1>’ // Displays the HTML Page
  - Controller: ‘ContactCtrl’
  - })
  - .otherwise({templateUrl: ‘view/404.html’})
  - locationProvider.html5Mode(true)
  - }])
  - HTML
    - <li class=”active”>Home // Has the value alex
    - {{input}} // Displays the value input
  - To use in the controller
    - .controller(‘ContactCtrl’, [‘$scope’, ‘$routeParam’, fucntion($scope, $routeParam) {
      - $scope.input = $**param1** // Will get the value alex
    - }]);

Ng-include: Templating in Angular

- Good for loading HTML Fragments in our page
- ng-include
  - An Angular core directive
  - Used to load and compile the template specified in src parameter
  - The template name is enclosed with double quotes and single quotes because the parameters applied to the src are an expression which will be evaluated
- Different Types of directives : Attribute , Element, Class
- Example
  - <ng-include src="'views/first-view-param.html'"
    - onload="string"
    - autoscroll="string">
  - </ng-include>
- Use it to load files from the same domain otherwise you have to deal with CORS errors.
- The directive can be treated as attribute and as element (better), but it can be used as CSS class too
- 2 optional attributes
  - onload: an expression to evaluate whenever the view updates
  - autoscroll: used to scroll the viewport partial after the view is loaded
- Can nest them ng-include one file and then inside that file have another ng-include
- Ng-view can be used in different way
  - <ng-view></ng-view>
  - <div data-ng-view> // Fix errrors in IE8
  -

NG Form – How to quicly Angularize it

- Used for Debugging : {{ eventForm | json}} // Shows the Scope as JSON
-

The NgSubmit Directive

- NG-Submit → binds the onSubmit to the expression
- Example
  - <form class=”form” id=”addEventForm” ng-submit=’submit(eventForm)’ name=’addEventForm’></form>
  - Angular.module(‘eventApp’)
  - .controller(‘formCtrl’, funciton($scope) {
    - $scope .event = [];
    - $scope.submitForm = function(form) {
      - $scope.event.push() // passed by reference, when the event array will have an object for each

        // time the button is pressed, but thety wll all contain the same data

        // solution sue angular.copy(form)
    - }
    -

Refactoring and Annotation

- Example
  - Angular.module(‘eventApp’) {
    - .controller(‘formCtrl’, [‘eventFactory’,$scope, function(a,b) P
    - Var eventFactory = {}
    - EventFactory.createEvent = function(event, eventList) {
      - EventList.createEvent(angular.copy(form), $event);
      - Console.log(form));
      - Return EventList
    - Return eventFactory; });

Retrieving Data From a Service

- Nginspector → Useful chrome plugin.
- Problem → Add two events and then change the page and add another event
  - See the last event in the scope.
  - The previous scope has been destroyed since we moved away from the page
  - Solution
    - Use the rootScope, but it is like using global variables
    - Example
      - Angular.module('eventApp')
      - .factory('eventFactory', function() {
        - Var eventFactory = {};
        - Var events = [] // Local to the service no matter what page you are on.
        -
        - EventFactory.getAllEvent = function() {
          - Returtn events; }
        -
        - Events.fatory.createEvent = function(event, eventList) {
          - Events.push(event);
          - EventList = events
          -
          - Return eventList;
        - }
        -
        - Return eventFactory; })
      - .controller('formCtrl', ['eventFactory','$scope', function(eventFactory, $scope) {
        - $scope.event = fucntion(form)
          - EventFactory.createEvent(angular.copy(form), $scope, event);
          - Console.log($scope, event); }
        - }])

Ng-Options and Grouping and trackby

- Ng-options
  - Select as label for value in array
    - ex. select as category.name for category in eventCtl.categories
    - Ng-modal=’eventForm.category’ → Where the dat ais put
  - Can also have groups
    - In your json assign a group for each value (ex group: ‘Main;’)
    - ng-options=”category select as category.name by category.group for category in eventCtl.categories”
    - Show a select box where each group is bolded and the item of the groupar idented after it.
- Default Category
  - Add the track by category.id
  - Ng-model is a new variable that contains the current selection.
  - When doing a post or form or saving it then new viriable that contians the current selection into you model (structure with all your data)

The NG-Value Directive

- Example
  - <label>
    - <input type=”radio” ng-model” eventForm.specialEvent’ ng-value=’eventCtl.specialValue’ value = ‘True’ /> Yes
  - </label>
  - <label>
    - <input type=”radio” ng-model=’eventForm.specialEvent’ name=’eventSpecial’>No
  - </label>
  - In this controller this.specialEvent = ‘true’ // Sets the default.
  - Ng-value → An expression that is evaluated
    - In the controller this.specialValue = { id:1, value: ‘somethingSpecial’ }
    - The Object is contained in the option selected.
  - Ng-value → An expression that can store an value
    - EXAPLE
      - IN THE CONTROLLER : This.specialValue = { id:1, value:’Something Special’ };
      - In the html : <input type=”radio” ng-model=’eventCtl.specialEvent’ ng-value=’eventCtl.specialValue’>Yes</label>

Ng-true-value, ng-false and ng-click

- Example
  - In controller : this.specialType = [{ name: ‘Age Restricted’, checked:false}, {name: ‘luxury’, checked:false}]
  - In html :
    - <input type=”checkbox” name=”eventSpecialType” ng-model=’type.checked’>{{type.name}}l</input>
    - <input type=”eheckbox” ng-click=”eventCtl.selectAllTypes()” name=”selectAllTypes”>Both</input>
      - In controller this.selectAllTypes = function() {
      -
        - Var self = this; // Can always used in nested functions without worrying about scope.
        - If (eventCtl.bothSelecteted) {
          - self.bothSelected = true
        - }
        - Else {
          - self.selected = false;
        - }
        -
        - Amgular.foreach(thsi.specialType, fucntion(item) {
          - self.checked = this.bothSelected;

          }
      - })
  - Ng-false-value, ng-true-value can set different values instead of true or false for a boolean.

Displaying Display Content and Exploring the Date Directive

- <div class=”form-group” ng-if=”eventCtl.eventForm.specialEvent === true”>
- Hides the element and removes it from the object element
- Date
  - <input type=”date” ng-model=’eventCtl.eventForm.date’ class=’form-control’ name=”eventDate” placeHolder=”Event Date” min=’2015-06-30’>
    - Min allows the calender to not select any date less than min
  - Date is in ISO Format
  - Filter {{ event.date | date : ‘yyyy-MM-dd’ }}
  - Can also have a time that allow you to an hour/minute
  - Set a place holder for the user so the date format will be known (some browsers done support the calendar).
-

Angular Validation : Introduction

- The form tag is actually an angular directive
- A directive can have a controller
  - The form controller can check all the children
-

Tracking a Form Validity and ng-show

- Variables of a form for validation
  - $pristine True when the form (or any of its input) has not been touched
  - $dirty The reverse of pristine,true when the user have touched the form
  - $valid True when all the form fields are valid
  - $invalid True when the form (or any of its input) are invalid
  - $touched True when the form control has been interacted with
  - $untouched True when the form control has not been interacted with
- Another nice plugin for Chrome is Angular Plugin.
- Ng-show (Hide an element if the condition is not true)
  - Ng-show condition is true it will display
  - Ng-hide condition is false it will hide
- Ng-minlength → Takes the mininium length
  - Ng-show “addEventForm.$dirty && addEventForm.eventName.$invalid”
- Ng-maxlength → Take a number for the mac length
  - Ng-show “addEventForm.$dirty && addEventForm.eventName.$invalid”

The ngPattern Directive: RegExp driven validation and Preventing invalid data from begin submitted

- Example in the div
  - Ng-pattern=’^[0-9]{4}—(0[1-9]|1[0-2])’
- If the regular expression is not matched the text will be displayed
- Preventing invalid data from being submitted
  - Use $valid
    - ng-sbumit=’addEventForm.$valid && eventCtrl.submitForm(eventCtl.eventForm)’
    - Does not alert you that the form is not valid
  - Ng-disabled
    - ng-disabled=”addEventForm.$invalid” in the input type = submit
    - When true, the submit button will be disabled.
- CSS
  - For each of the validation variables angular js provides a ng-pristine for the $pristine
  - Input.ng-touched.ng-invalid { Add to your css
    - Border : 1px solid red;
    - }
  - When the field is invalid it will change to red since the ng-touched.ng-invalid will be an css attribute added to the tag
- ng-class={‘has-error’: addEventForm.eventName.$touched && addEventForm.eventName.$invalid }
  - Apply the class when the condition is mentioned
  - Class : <rule>

Ng-messages – Improved Validation message

- Use the script angular-message.js
- Inject ngMessages in the app.js module
- // Use the plural to define the field
  - This is required // Use the singular for the specific message
- Generic template of messages that can be used on the fields
  - <script type=”text/ng-template” id=”error-messages”>
    - This filed is required // Adds an error message to the default template
    - Too short
  - </script>
  -
    - / / A div with condition should wrap this current div
  -

Basic Filtering in Data

- Example
  - Search <input type+’text” ng-model=”search”>
    - Stores the search value int the search variable
  - <tr ng-repeat=”event in managerCtl.eventList | filter: search”>
    - Will filter basic on that String for each row.
  - <Search by Event Name: <input type=”text” ng-model=”search.name”>
    - Will look at the name field
  - Clear filter
    - <button ng-click=”search = undefined”>Clear Fitler</button>

Strict Search and Order-By

- Strict → An exact match or no results returned
- Add the keyword true
  - <tr ng-repeat in managerCtl.eventList | filter search : true”>
  - By parameter
    - Exact Match <input typ=”checkbox” ng-model=”strict”>
    - <tr ng-repeat in managerCtl.eventList | filter search : strict>
- Order keywords by alphabetical
  - <tr ng-repeat in managerCtl.eventList | filter search : true” | orderBy: ‘name’ “>
  - Reverse Order : <tr ng-repeat in managerCtl.eventList | filter search : true” | orderBy: ‘-name’ “>