Workshop Instructions

Please see the **Prerequisites Document** before you get started

Get the code

If you have installed git you can clone the repository from github:

git clone https://github.com/petebacondarwin/foodme-intro

cd foodme-intro

Folder Structure

There are two folders for each step. One (step-xx) contains the starting point for that step. The other (step-xx-solution) contains the end point for that step. There is a shared folder that contains the common resources for each step: CSS stylesheets, AngularJS libraries, images and data.

Background Reading

Many of the steps have a Topics section. In here you will see keywords of things to read up on that are related to this step. You can find out about these topics at the Angular documentation site:

<https://docs.angularjs.org/>

# Basic Angular App

We start with an initial static HTML mock up, such as you could have received from a designer.

## Goals

* Convert the static HTML to an Angular application
* Bind the Customer Info form to a model on the Angular scope.

## Topics

* Angular basics
* Directives
* Scope
* Data binding

## Tasks

### Load the AngularJS JavaScript file in index.htm

<head>

...

**<script src="../shared/js/angular.js"></script>**

</head>

### Add ng-app to the index page

<html lang="en" **ng-app**>

### Bind the user name input element to **user.name**

Change **index.html**:

<input type="text" ... **ng-model="user.name"**>

### Bind the user address input element to **user.address**

Change **index.html**:

<input type="text" ... **ng-model="user.address"**>

### Display the user info using **{{ ... }}** curly brace interpolated bindings

Change **index.html**:

<div class="well">

<a href="" class="pull-right">Change</a>

<strong>Deliver to:</strong><br>

**{{ user.name }}<br>**

**{{ user.address }}**

</div>

# Conditionally display customer info form

We have a deliveryForm, which is bound to the AppController; and a display box for this info.

## Goals

* Only show one of the form or the display box at a time
* Show or hide these items when the user clicks on links

## Topics

* ngShow/ngHide
* ngClick

## Tasks

### Show the form when deliveryFormVisible is true, using ng-show directive

Change **index.html**:

<div class="row" **ng-show="deliveryFormVisible"**>

### Hide the delivery info display box when deliveryInfoVisible is true, using ng-hide

Change **index.html**:

<div class="row" **ng-hide="deliveryFormVisible"**>

### Change deliveryInfoVisible when clicking "Change" and "Hide"

Change **index.html**:

<a href="" class="pull-right" **ng-click="deliveryForm.visible = true"**>Change</a>

...

<a href="" class="pull-right" **ng-click="deliveryForm.visible = false"**>Hide</a>

# Move logic into a Controller

We have a deliveryInfoForm and display box that are shown and hidden respectively when links are clicked.

## Goals

* Move the logic for the display of the deliveryInfoForm and display box into a controller
* Show or hide these items when the user clicks on links

## Topics

* Modules
* Controllers
* ngController

## Tasks

### Create an **app** module and a **AppController** controller in app.js

Create **app.js**:

angular.module('app', [])

.controller('AppController', function() {

});

### Load the app.js file in the index.html page

Change **index.html**:

<script src="app.js"></script>

### Initialize the controller with a **deliveryFormVisible** and a **user** property

Change **app.js**:

this.deliveryFormVisible = true;

this.user = {

name: 'Jo Bloggs',

address: '123, Some Place, Some Where'

};

### Add showDeliveryForm() and hideDeliveryForm() methods to AppController

Change **app.js**:

this.showDeliveryForm = function() {

this.deliveryFormVisible = true;

};

this.hideDeliveryForm = function() {

this.deliveryFormVisible = false;

};

### Connect the AppController to the view using ng-controller

Change **index.html**:

<body ... ng-app**="app" ng-controller="AppController as app"**>

### Update the ng-model, ng-show, ng-hide and interpolation bindings to reference the controller

Change **index.html**:

<div class="row" **ng-show="app.deliveryFormVisible"**>

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

<form role="form" class="well">

<a href="" class="pull-right"

**ng-click="app.hideDeliveryForm()"**>Hide</a>

<legend>Delivery Details</legend>

<div class="form-group">

<label for="customerName" class="control-label">Name</label>

<input type="text" id="customerName"

class="form-control" **ng-model="app.user.name"**>

</div>

<div class="form-group">

<label for="address" class="control-label">Address</label>

<input type="text" id="address"

class="form-control" **ng-model="app.user.address"**>

</div>

</form>

</div>

</div>

...

<div class="row" **ng-hide="app.deliveryFormVisible"**>

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

<div class="well">

<a href="" class="pull-right"

**ng-click="app.showDeliveryForm()"**>Change</a>

<strong>Deliver to:</strong><br>

**{{ app.user.name }}**<br>

**{{ app.user.address }}**

</div>

</div>

</div>

# Display a list of restaurants

We are able to view and update the deliver info.

## Goals

* Display a data driven list of restaurants using a repeated template

## Topics

* ngRepeat
* ngSrc

## Tasks

### Initialize a mock list of restaurants in the **AppController**.

Change **app.js**:

this.restaurants = [

{

"price": 3,

"rating": 3,

"id": "esthers",

"name": "Esther's German Saloon",

"location": "22 Teutonic Ave.",

"description": "German home-cooked meals and fifty-eight different beers on tap. To get more authentic, you'd need to be wearing lederhosen."

},

{

"price": 4,

"rating": 5,

"id": "robatayaki",

"name": "Robatayaki Hachi",

"location": "8 Hawthorne Ln.",

"description": "Japanese food the way you like it. Fast, fresh, grilled."

},

{

"price": 5,

"rating": 4,

"id": "bateaurouge",

"name": "Le Bateau Rouge",

"location": "2 South Park Dr.",

"description": "Fine French dining in a romantic setting. From soupe à l'oignon to coq au vin, let our chef delight you with a local take on authentic favorites."

}

];

### Bind the template to the list of restaurants using **ng-repeat** directive

Change **index.html**:

<tr **ng-repeat="restaurant in app.restaurants"**>

<td class="description">

<div class="media">

<a class="pull-left">

<img class="img-rounded"

**ng-src="../shared/img/restaurants/{{restaurant.id}}.jpg"**>

</a>

<div class="media-body">

<h4 class="media-heading">**{{restaurant.name}}**</h4>

<p>**{{restaurant.description}}**</p>

</div>

</div>

</td>

<td class="price">

**{{restaurant.price}}**

</td>

<td class="rating">

**{{restaurant.rating}}**

</td>

</tr>

# Enable sorting of restaurant list

We have a data driven list of restaurants.

## Goals

* Use the **orderBy** filter to sort the restaurant list by price or rating

## Topics

* Filters
* orderBy
* ngClass

## Tasks

### Initialize sortProperty and sortDirection on the controller for sorting columns

Change **app.js**:

this.sortProperty = 'name';

this.sortDirection = false;

### Add a orderBy filter to the ng-repeat using these properties

Change **index.html**:

<tr ng-repeat="restaurant in app.restaurants **| orderBy : app.sortProperty : app.sortDirection"**>

### Create helper methods in the controller, sortBy(property) and getSortClass(property)

Change **app.js**:

this.sortBy = function(property) {

if ( this.sortProperty === property ) {

this.sortDirection = !this.sortDirection;

} else {

this.sortProperty = property;

this.sortDirection = false;

}

};

this.getSortClass = function(property) {

if ( this.sortProperty === property ) {

return 'glyphicon glyphicon-chevron-' + (this.sortDirection ? 'down' : 'up');

}

};

### Convert the table headings to clickable anchors using ng-click

Change **index.html**:

<tr>

<th><a href **ng-click="app.sortBy('name')"**>Name ...</a></th>

<th><a href **ng-click="app.sortBy('price')"**>Price ...</a></th>

<th><a href **ng-click="app.sortBy('rating')"**>Rating ...</a></th>

</tr>

### Display sort direction up/down markers using ng-class

Change **index.html**:

Name **<span ng-class="app.getSortClass('name')"></span>**

...

Price **<span ng-class="app.getSortClass('price')"></span>**

...

Rating **<span ng-class="app.getSortClass('rating')"></span>**

# Improve price and rating display

We have a sortable list of restaurants

## Goals

* Implement a custom filter to display price and ratings using icons rather than numbers

## Topics

* Filters
* ngBindHtml
* Strict Contextual Escaping ($sce)

## Tasks

### Create a custom rating filter in the app module - we must use $sce.trustAsHtml since we are generating HTML.

Change **app.js**:

.filter('rating', function($sce) {

return function(value, glyph) {

var output = "";

while(value>0) {

output += '<span class="glyphicon glyphicon-' + glyph + '"></span>';

value -= 1;

}

return $sce.trustAsHtml(output);

};

});

### Use the filter in the price and rating fields, with the ng-bind-html directive

Change **index.html**:

<td class="price" **ng-bind-html="restaurant.price | rating : 'gbp'"**>

</td>

...

<td class="rating" **ng-bind-html="restaurant.rating| rating : 'star'"**>

</td>

# Add Validation to the DeliveryInfoForm

We have a basic deliveryForm, which is bound to the AppController.

## Goals

* Add validation to the input elements on the form
* Show error messages when an input is invalid
* Change the CSS style of the form elements when there are invalid inputs

## Topics

* Form, input and validation directives
* ngModelController and FormController
* ngClass
* ngMessages

## Tasks

### Give the form a name so we can access its validity

Change **index.html**:

<form ... **name="app.deliveryForm"**>

Now we can access validity properties of the form on the scope such as **app.deliveryForm.$valid**.

### Add name, required and ng-minlength attributes to the input elements

Change **index.html**:

<input type="text" ng-model="app.user.name"

**name="userName"** **required ng-minlength="5"**>

…

<input type="text" ng-model="app.user.address"

**name="userAddress"** **required ng-minlength="10"**>

Notice that, when the inputs are invalid, they now have the **ng-invalid** CSS class on them. In the **../shared/css/app.css** stylesheet, there is a rule that gives the input box a red border when this happens.

### Change the CSS classes of the form element when validity changes

Change **index.html**:

<div class="form-group"

**ng-class="{'has-error': app.deliveryForm.userName.$invalid}"**>

...

</div>

<div class="form-group"

**ng-class="{'has-error': app.deliveryForm.userAddress.$invalid}"**>

...

</div>

### Load the angular-message.js file

Change **index.html**:

<script src="../shared/js/angular-messages.js"></script>

### Add the **ngMessages** module as a dependency of our **app** module

Change **app.js**:

angular.module('app', [**'ngMessages'**])

### Use the **ng-messages** directive to display errors for the form elements

Change **index.html**:

<div ng-messages="app.deliveryForm.userName.$error">

<div ng-message="required" class="alert alert-warning"

role="alert">You must enter a name.</div>

<div ng-message="minlength" class="alert alert-warning"

role="alert">Your name must be at least 5 characters long.</div>

</div>

...

<div ng-messages="app.deliveryForm.userAddress.$error">

<div ng-message="required" class="alert alert-warning"

role="alert">You must enter an address.</div>

<div ng-message="minlength" class="alert alert-warning"

role="alert">Your address must be at least 10 characters long.</div>

</div>

# Persist the delivery Info in localStorage

We have a delivery info form with validation.

It is nice if the customer does not have to enter their information every time. We can use the browser’s local storage to persist this information between visits to the site.

## Goals

* Create angular services that persist values to the browser's **localStorage**.
* Use these services to persist changes to the **user** property of the **AppController**.

## Topics

* Angular Services
* Dependency Injection
* Scope Watches

## Tasks

### Create a new **localStorage** module

Create **localStorage.js**:

angular.module('localStorage', [])

### Create a **localStorage** service to wrap the browser's **localStorage** object

Change **localStorage.js**:

.value('localStorage', window.localStorage)

Wrapping the **localStorage** browser object like this makes it easier to mock out when testing.

### Create a **localStorageBinding** service that binds an object to the **localStorage**

Change **localStorage.js**:

.factory('localStorageBinding', function(localStorage, $rootScope) {

return function(key, defaultValue) {

defaultValue = JSON.stringify(defaultValue || {});

var value = JSON.parse(localStorage[key] || defaultValue);

$rootScope.$watch(function() { return value; }, function() {

localStorage[key] = JSON.stringify(value);

}, true);

return value;

};

})

The **localStorageBinding** service is a function that will return an object that is persisted to the **localStorage** every time it changes. This uses the **$rootScope**.**$watch** mechanism to know when the object has changed.

### Load the new **localStorage.js** file

Change **index.html**:

<script src="localStorage.js"></script>

### Add the new **localStorage** module as a dependency to our **app** module

Change **app.js**:

angular.module('app', ['ngMessages', **'localStorage'**])

### Inject the localStorageBinding service into the AppController

Change **app.js**:

.controller('AppController', function(**localStorageBinding**) {

### Bind the user property to the **localStorage** using the **localStorageBinding** service

Change **app.js**:

this.user = **localStorageBinding('foodMe/user', {**

**name: 'Jo Bloggs',**

**address: '123, Some Place, Some Where'**

**});**

# Load restaurant list from a server

We have a data driven restaurant list, with the data hard coded in the **AppController**.

## Goals

* Use an HTTP request to get the restaurant data from a server

## Topics

* Same-origin security policy
* Cross-origin resource sharing (CORS)
* HTTP Requests (**$http**)
* Promises

## Tasks

### Add the $http dependency to the AppController

Change **app.js**:

.controller('AppController', function(localStorageBinding, **$http**) {

### Replace the static data with a call to get the restaurant data from server

Change **app.js**:

**var that = this;**

**var url = '...';**

**$http.get(url).then(function(response) {**

that.restaurants = **response.data;**

**});**

### Specify the url to the restaurant data on the CORS enabled server

Change **app.js**:

var url = **'https://foodme.firebaseio.com/.json';**

### Alternatively, start up a local server and change the url to this server

In a **console/terminal**:

npm install –g http-server

http-server

You can browse to the application at:

http://localhost:8080/step-09

Change **app.js**:

var url = **'../shared/data/restaurants.json';**

# Filter Restaurants by Price and Rating

There are now 39 restaurants in the list. It would be good to filter these based on their rating and how expensive they are.

## Goals

* Filter the restaurant list by rating and price

## Topics

* $watchGroup

## Tasks

### Add a new form to the left of the restaurant list

Change **index.html**:

<div class="col-md-3">

<form role="form" class="well" name="app.filterForm">

<legend>Filter Restaurants</legend>

<div class="form-group">

<label for="priceFilter" class="control-label">Price (no more than)</label>

<input type="number" id="priceFilter" name="priceFilter" class="form-control" ng-model="app.filters.price">

</div>

<div class="form-group">

<label for="ratingFilter" class="control-label">Rating (at least)</label>

<input type="number" id="ratingFilter" name="ratingFilter" class="form-control" ng-model="app.filters.rating">

</div>

</form>

</div>

<div **class="col-md-9"**

...

</div>

### Initialize the filters to null in the AppController

Change **app.js**:

this.filters = {

price: null,

rating: null

};

### Inject the $rootScope service into the AppController

Change **app.js**:

.controller('AppController', function(localStorageBinding, $http, **$rootScope**) {

### Watch the price, rating and restaurant list for changes and filter the list accordingly

Change **app.js**:

var filterRestaurants = function() {

that.filteredRestaurants = [];

angular.forEach(that.restaurants, function(restaurant) {

if ( ( !that.filters.rating ||

restaurant.rating >= that.filters.rating ) &&

( !that.filters.price ||

restaurant.price <= that.filters.price ) ) {

that.filteredRestaurants.push(restaurant);

}

}

};

**$rootScope.$watchGroup**([

function() { return that.filters.price; },

function() { return that.filters.rating; },

function() { return that.restaurants; }

], filterRestaurants);

### Change the **ng-repeat** directive to use the **filteredRestaurants** collection

Change **index.html**:

<tr ng-repeat="restaurant in **app.filteredRestaurants** | orderBy : app.sortProperty : app.sortDirection">

# Display the number of filtered restaurants

We have a sortable, filterable list of restaurants loaded from a server.

## Goals

* Add a message showing the number of restaurants in the filtered list

## Topics

* ICU Message Format
* ngMessageFormat

## Tasks

### Load the **angular-message-format.js** file

Change **index.html**:

<script src="../shared/js/angular-message-format.js"></script>

### Add the **ngMessageFormat** module as a dependency of our **app** module

Change **app.js**:

angular.module('app', ['ngMessages', **'ngMessageFormat'**, 'localStorage'])

### Add a message bound to the length of the **filteredRestaurants** collection using the message format syntax.

Change **index.html**:

<div class="alert alert-info" role="alert">

{{ app.filteredRestaurants.length, plural,

=0 {No restaurants found!}

=1 {Only one restaurant found!}

other {# restaurants found.}

}}

</div>

# Pretty Rating Chooser Directive

We have a basic form for selecting the price and rating for filtering.

## Goals

* Implement a reusable directive to display a more attractive price and rating chooser

## Topics

* Directives
* Isolated Scope

## Tasks

### Create a new rating.js file containing a rating module

Create **rating.js**:

angular.module('rating', [])

### Load the rating.js file

Change **index.html**:

<script src="rating.js"></script>

### Add the rating module as a dependency of our app module

Change **app.js**:

angular.module('app', ['ngMessages', 'ngMessageFormat', 'localStorage', **'rating'**])

### Define a **fmRating** directive in the **rating** module

Change **rating.js**:

.directive('fmRating', function() {

return {

restrict: 'E',

scope: {

glyph: '@',

rating: '='

},

link: function(scope, element, attrs) {

scope.ratings = [1,2,3,4,5];

scope.select = function(value) {

scope.rating = value;

};

scope.isSelected = function(value) {

return scope.rating >= value;

};

},

template:

'<ul class="fm-rating">' +

' <li ng-repeat="value in ratings" ng-click="select(value)" ng-class="{selected: isSelected(value)}">' +

' <span class="glyphicon glyphicon-{{glyph}}"></span>' +

' </li>' +

'</ul>' +

'<a ng-click="rating = null">clear</a>'

};

});

### Use this directive in the Filter Restaurants form instead of the input boxes

Change **index.html**:

<fm-rating rating="app.filters.price" glyph="gbp"></fm-rating>

...

<fm-rating rating="app.filters.rating" glyph="star"></fm-rating>