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-further

cd foodme-further

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/>

# run the local webserver

## Where are we?

A running Angular application running from the file system; displaying a filterable, sortable list of restaurants and delivery info form.

## Goals

* Serve the application from a local webserver

## Tasks

### Install a simple http server: npm install -g http-server

Start the server:

$ http-server

Browse to the application: http://localhost:8080/step-01

# Unit testing the AppController

## Where are we?

A running Angular application running from a local webserver; displaying a filterable, sortable list of restaurants and delivery info form.

## Goals

* Ensure karma is installed Configure a karma test config
* Create and run initial unit tests for AppController

## Topics

* Karma
* Jasmine
* angular.mock.inject
* angular.mock.module
* $controller

## Tasks

### Install karma-cli and karma libraries in the root of the repository

$ cd foodme-further

$ npm install -g karma-cli

$ npm init

...

Press ^C at any time to quit.

name: (foodme-further)

version: (1.0.0)

description:

entry point: (index.js)

test command:

git repository: (https://github.com/petebacondarwin/foodme-further.git)

keywords:

author:

license: (ISC)

About to write to /Users/pete/dev/angular/foodme-further/package.json:

{

"name": "foodme-further",

"version": "1.0.0",

"description": "(explain about http server and XHR issues) \* Step 0 - run the local webserver",

"main": "index.js",

"scripts": {

"test": "echo \"Error: no test specified\" && exit 1"

},

"repository": {

"type": "git",

"url": "https://github.com/petebacondarwin/foodme-further.git"

},

"author": "",

"license": "ISC",

"bugs": {

"url": "https://github.com/petebacondarwin/foodme-further/issues"

},

"homepage": "https://github.com/petebacondarwin/foodme-further"

}

Is this ok? (yes)

### Initialize karma config in the step folder

$ cd step-01

$ karma init

Which testing framework do you want to use ?

Press tab to list possible options. Enter to move to the next question.

> jasmine

Do you want to use Require.js ?

This will add Require.js plugin.

Press tab to list possible options. Enter to move to the next question.

> no

Do you want to capture any browsers automatically ?

Press tab to list possible options. Enter empty string to move to the next question.

> Chrome

>

What is the location of your source and test files ?

You can use glob patterns, eg. "js/\*.js" or "test/\*\*/\*Spec.js".

Enter empty string to move to the next question.

> ../shared/js/angular.js

> ../shared/js/angular-messages.js

> ../shared/js/angular-message-format.js

> ../shared/js/angular-mocks.js

> \*.js

>

Should any of the files included by the previous patterns be excluded ?

You can use glob patterns, eg. "\*\*/\*.swp".

Enter empty string to move to the next question.

>

Do you want Karma to watch all the files and run the tests on change ?

Press tab to list possible options.

> yes

### Create app.spec.js Jasmine spec file

describe('AppController controller', function() {

var controller;

beforeEach(module('app'));

beforeEach(inject(function($controller) {

controller = $controller('AppController', {});

}));

it('should initialize controller properties', function() {

expect(controller.deliveryFormVisible).toBe(true);

expect(controller.user).toEqual(jasmine.any(Object));

expect(controller.sortProperty).toEqual('name');

expect(controller.sortDirection).toBe(false);

});

});

### Start the karma test runner

$ karma start

## Extras

Test that the AppController filters property has been correctly initialized

# more AppController unit tests

## Where are we?

A running Angular application running from a local webserver; displaying a filterable, sortable list of restaurants and delivery info form; with really simple unit tests for AppController.

## Goals

* Create more unit tests for AppController methods
* Use mocks to isolate unit test behaviour of localStorage
* Use ngMock.$httpBackend to test making server requests

## Topics

* Mocking services
* ngMock.$httpBackend
* $rootScope.$digest

## Tasks

### Add tests for AppController methods

describe('showDeliveryForm', function() {

it('should set deliveryFormVisible to true', function() {

controller.deliveryFormVisible = null;

controller.showDeliveryForm();

expect(controller.deliveryFormVisible).toBe(true);

});

});

describe('hideDeliveryForm', function() {

it('should set deliveryFormVisible to false', function() {

controller.deliveryFormVisible = null;

controller.hideDeliveryForm();

expect(controller.deliveryFormVisible).toBe(false);

});

});

### Mock the localStorage object so that we can test the user object

beforeEach(inject(function($controller, localStorage) {

localStorage['foodMe/user'] = '{ "name": "Test User", "address": "Test Address" }';

controller = $controller('AppController', {});

}));

### Test that the user object has been initialized correctly

it('should initialize controller properties', function() {

...

expect(controller.user).toEqual({ "name": "Test User", "address": "Test Address" });

});

### Mock the $http service to return test restaurant data for testing

beforeEach(inject(function($controller, localStorage, $httpBackend) {

localStorage['foodMe/user'] = '{ "name": "Test User", "address": "Test Address" }';

$httpBackend.when('GET', '../shared/data/restaurants.json').respond([

{ id: 'test1' },

{ id: 'test2' },

{ id: 'test3' }

]);

controller = $controller('AppController', {});

}));

### Flush the mock $http service to trigger the response and test the restaurants property on the controller

it('should attach the restaurant data when it arrives', inject(function($httpBackend) {

$httpBackend.flush();

expect(controller.restaurants).toEqual([

{ id: 'test1', price: 1, rating: 3 },

{ id: 'test2', price: 2, rating: 4 },

{ id: 'test3', price: 3, rating: 5 }

]);

}));

### Modify the filters and test that the filteredRestaurants collection updates

describe('filteredRestaurants', function() {

beforeEach(inject(function($httpBackend) {

$httpBackend.flush();

}));

it('should contain the full list after the restaurants have loaded', function() {

expect(controller.filteredRestaurants).toEqual([

{ id: 'test1', price: 1, rating: 3 },

{ id: 'test2', price: 2, rating: 4 },

{ id: 'test3', price: 3, rating: 5 }

]);

});

it('should update the list when the filters change', inject(function($rootScope) {

controller.filters.price = 2;

$rootScope.$digest();

expect(controller.filteredRestaurants).toEqual([

{ id: 'test1', price: 1, rating: 3 },

{ id: 'test2', price: 2, rating: 4 }

]);

controller.filters.rating = 4;

$rootScope.$digest();

expect(controller.filteredRestaurants).toEqual([

{ id: 'test2', price: 2, rating: 4 }

]);

}));

});

## Extras

* Test AppController sortBy and getSortClass methods

# rating filter unit tests

## Where are we?

A running Angular application running from a local webserver; displaying a filterable, sortable list of restaurants and delivery info form; with unit tests for AppController.

## Goals

* Create unit tests for rating filter

## Topics

* Testing filters
* $filter
* underscore injection sugar

## Tasks

### Use $sce and $filter to test the ratingFilter

describe('rating filter', function() {

beforeEach(module('app'));

it('should return a trusted HTML string', inject(function($sce, $filter) {

var ratingFilter = $filter('rating');

var trustedValue = ratingFilter(3, 'star');

var realValue = $sce.getTrustedHtml(trustedValue);

expect(realValue).toEqual(

'<span class="glyphicon glyphicon-star"></span>' +

'<span class="glyphicon glyphicon-star"></span>' +

'<span class="glyphicon glyphicon-star"></span>'

);

}));

});

### Inject the rating filter (as ratingFilter) directly into the test

it('should return a trusted HTML string', inject(function($sce, ratingFilter) {

...

}));

### Use underscores to assign the ratingFilter outside of the current function context

var ratingFilter

beforeEach(inject(function(\_ratingFilter\_) {

ratingFilter = \_ratingFilter\_;

}));

## Extras

* Try writing tests for the localStorageBinding service

# fmRating directive unit tests

## Where are we?

A running Angular application running from a local webserver; displaying a filterable, sortable list of restaurants and delivery info form; with unit tests for AppController and rating filter.

## Goals

* Create unit tests for rating directive

## Topics

* Testing directives
* $compile
* $rootElement
* custom Jasmine matchers

## Tasks

### Create an instance of the rating directive to unit test using $compile

Create rating.spec.js:

describe('fmRating directive', function() {

var $scope, element;

beforeEach(module('rating'));

beforeEach(inject(function($compile, $rootScope, $rootElement) {

$scope = $rootScope.$new();

element = $rootElement;

element.html('<fm-rating glyph="star" rating="model.value"></fm-rating>');

$compile(element)($scope);

$scope.$apply('model.value = 0');

}));

});

### Create a toHaveClass Jasmine matcher

beforeEach(function() {

jasmine.addMatchers({

toHaveClass: function () {

return {

compare: function(element, className, expectedCount) {

var actualCount = 0;

var result = { pass: true };

for(var i = 0, ii = element.length; i < ii; i++) {

if (element.eq(i).hasClass(className)) {

actualCount ++;

}

}

if ( actualCount !== expectedCount ) {

result = {

pass: false,

message: 'Expected ' + className + ' on ' + expectedCount + ' elements but it appeared on ' + actualCount + ' elements'

};

}

return result;

}

};

}

});

});

### Use the custom matcher to test the number of icons displayed

it('should contain five star glyphs', function() {

expect(element.find('li').length).toEqual(5);

expect(element.find('span')).toHaveClass('glyphicon-star', 5);

})

### Change the rating programmatically to test that the HTML changes

it('should update the view when the rating changes', function() {

$scope.$apply('model.value = 3');

expect(element.find('li')).toHaveClass('selected', 3);

});

### Simulate clicking the directive to test changing the rating

it('should update the rating when clicked', function() {

element.find('li').eq(2).triggerHandler('click');

expect($scope.model.value).toEqual(3);

expect(element.find('li')).toHaveClass('selected', 3);

});

# e2e testing

## Where are we?

A unit-tested Angular application running from a local webserver

## Goals

* Ensure protractor is installed
* Configure a protractor test config
* Create and run initial e2e tests

## Topics

* Protractor

## Tasks

### Install protractor and webdriver

$ npm install -g protractor

$ webdriver-manager update

### Ensure that the http server is running in the root of the repository

$ cd foodme-further

$ http-server

### Create protractor.conf.js in the step folder

var path = require('path');

var stepPath = path.basename(\_\_dirname);

exports.config = {

specs: ['e2e/\*.spec.js'],

baseUrl: 'http://localhost:8080/' + stepPath + '/',

directConnect: true

};

### Exclude the protractor.conf.js from the karma config

// list of files to exclude

exclude: [

'protractor.conf.js'

],

### Create e2e/app.spec.js protractor spec file

describe('app', function() {

beforeEach(function() {

browser.get('index.html');

});

it('should update the delivery info box when the deliveryForm is changed', function() {

var userNameInput = element(by.model('app.user.name'));

userNameInput.clear();

userNameInput.sendKeys('test user');

var hideDeliveryFormLink = element(by.css('a[ng-click="app.hideDeliveryForm()"]'));

hideDeliveryFormLink.click();

var userNameDisplay = element(by.binding('app.user.name'));

expect(userNameDisplay.getText()).toContain('test user');

});

it('should display a list of restaurants', function() {

var restaurantList = element.all(by

.repeater('restaurant in app.filteredRestaurants')

.column('restaurant.name'));

expect(restaurantList.count()).toEqual(39);

expect(restaurantList.get(0).getText()).toEqual('Angular Pizza');

});

});

### Execute the protractor specs from the step folder

$ cd step-05

$ protractor protractor.conf.js

## Extras

* Try running protractor with a standalone Selenium server:

# e2e testing

## Where are we?

An Angular application running from a local webserver; with unit tests and initial e2e tests

## Goals

* Use a PageObject to make e2e tests clearer

## Topics

* Protractor PageObjects

## Tasks

### Create a PageObject for the app

function HomePage() {};

HomePage.prototype = {

get: function() {

browser.get('index.html');

},

setUserName: function(name) {

var e = element(by.model('app.user.name'));

e.clear();

e.sendKeys(name);

},

getDisplayedUserName: function() {

return element(by.binding('app.user.name'));

},

hideDeliveryForm: function() {

element(by.css('a[ng-click="app.hideDeliveryForm()"]')).click();

},

showDeliveryForm: function() {

element(by.css('a[ng-click="app.showDeliveryForm()"]')).click();

},

getRestaurantList: function(column) {

var repeater = by.repeater('restaurant in app.filteredRestaurants');

if (column) {

repeater = repeater.column(column);

}

return element.all(repeater);

}

};

module.exports = HomePage;

### Import and use the HomePage object in the app.spec.js e2e file

var HomePage = require('./home.page');

describe('app', function() {

var homePage;

beforeEach(function() {

homePage = new HomePage();

homePage.get();

});

it('should update the delivery info box when the deliveryForm is changed', function() {

homePage.setUserName('test user');

homePage.hideDeliveryForm();

expect(homePage.getDisplayedUserName().getText()).toContain('test user');

});

it('should display a list of restaurants', function() {

var restaurantList = homePage.getRestaurantList('restaurant.name');

expect(restaurantList.count()).toEqual(39);

expect(restaurantList.get(0).getText()).toEqual('Angular Pizza');

});

});

## Extras

* Try adding checks that test the validation of the deliveryInfoForm
* Try adding checks that test the filtering of the restaurant list

# add routing

## Where are we?

An Angular application running from a local webserver; with unit tests and e2e tests

## Goals

* Implement routing to enable multiple URL driven views
* Move the current HTML into a new view

## Topics

* ngRoute
* $routeProvider

## Tasks

### Load the ../js/angular-route.js file

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

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

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

### Add the angular-route.js file to the files to load in the karma config

files: [

'../shared/js/angular.js',

'../shared/js/angular-messages.js',

'../shared/js/angular-message-format.js',

'../shared/js/angular-route.js',

'../shared/js/angular-mocks.js',

'\*.js'

],

### Create a new components/restaurants/index.html template for the current HTML

Copy the restaurant filtering form and the restaurant list HTML from index.html

<!-- Restaurant List -->

<div class="row">

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

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

<legend>Filter Restaurants</legend>

...

</form>

</div>

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

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

{{ app.filteredRestaurants.length, plural,

=0 {No restaurants found!}

=1 {Only one restaurant found!}

other {# restaurants found.}

}}

</div>

<table class="table table-striped">

...

</table>

</div>

</div>

### Remove the restaurant specific HTML from index.html and replace with an ng-view directive

<ng-view autoscroll></ng-view>

### Configure the application to display this component when we navigate to the root of the application

.config(function($routeProvider) {

$routeProvider

.when('/restaurants', {

templateUrl: 'components/restaurants'

})

.otherwise('/restaurants');

})

### Check that the karma and protractor tests still pass

$ karma start --single-run

$ protractor protractor.conf.js

# Add more routes

## Where are we?

An Angular application running from a local webserver; with unit tests and e2e tests; and routing

## Goals

* Add additional static views
* Add navigation links to these views

## Tasks

### Create a new components/help/index.html template for the help view

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

<div class="fm-panel">

<div class="fm-heading">Help</div>

<div class="fm-content">

<h4>Until how late do you deliver?</h4>

<p>We deliver as late as 9pm from all restaurants.</p>

<h4>What payment methods do you accept?</h4>

<p>We gladly accept all major credit cards.</p>

<h4>I got an extra chocolate muffin with my order, what is this about?</h4>

<p>That's just a small thank-you gift from FoodMe. Enjoy it!</p>

</div>

</div>

</div>

### Create a new components/about-us/index.html template for the about-us view

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

<div class="fm-panel">

<div class="fm-heading">Who we are</div>

<div class="fm-content">

<p>We are purple unicorns jockeys and knitters of woolen socks!</p>

</div>

</div>

</div>

### Create a new components/how-it-works/index.html template for the how-it-works view

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

<div class="fm-panel">

<div class="fm-heading">How it works</div>

<div class="fm-content">

<p>It's simple:</p>

<ol>

<li>Enter delivery address</li>

<li>Pick a great restaurant</li>

<li>Select yummy food</li>

<li>Enter delivery time</li>

<li>Enter payment details</li>

<li>And the food will be on the way!</li>

</ol>

</div>

</div>

</div>

### Configure the application to display these views

.config(function($routeProvider) {

$routeProvider

.when('/restaurants', {

templateUrl: 'components/restaurants'

})

.when('/about-us', {

templateUrl: 'components/about-us'

})

.when('/help', {

templateUrl: 'components/help'

})

.when('/how-it-works', {

templateUrl: 'components/how-it-works'

})

.otherwise('/restaurants');

})

### Add links to these views in the navigation panel

<!-- Navigation Bar -->

<div class="navbar navbar-default">

<div class="container-fluid">

<div class="navbar-header">

<a class="navbar-brand" href="#/">FoodMe</a>

</div>

<div class="collapse navbar-collapse">

<ul class="nav navbar-nav">

<li><a href="#/">Home</a></li>

<li><a href="#/how-it-works">How it works</a></li>

<li><a href="#/about-us">Who we are</a></li>

</ul>

<ul class="nav navbar-nav navbar-right">

<li><a href="#/help">Help</a></li>

</ul>

</div>

</div>

</div>

### Check that the karma and protractor tests still pass

$ karma start --single-run

$ protractor protractor.conf.js

## Extras

* Try adding more static views of your own

# move restaurant logic into component

## Where are we?

An Angular application running from a local webserver; with unit tests and e2e tests; and routing

## Goals

Move restaurants route config and logic into its own module

## Tasks

### Create a restaurants module in components/restaurants/index.js

angular.module('restaurants', [])

### Load this new index.js file in index.html

<script src="components/restaurants/index.js"></script>

### Add the restaurants module as dependency of the app module

angular.module('app', ['ngMessages', 'ngMessageFormat', 'ngRoute',

'localStorage', 'rating', 'restaurants'])

### Add a new RestaurantsController to the restaurants module

.controller('RestaurantsController', function($http, $rootScope) {

var that = this;

// var url = 'https://foodme.firebaseio.com/.json'; // CORS enabled server

var url = '../shared/data/restaurants.json'; // Local webserver

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

that.restaurants = response.data;

});

this.sortProperty = 'name';

this.sortDirection = false;

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');

}

};

this.filters = {

price: null,

rating: null

};

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);

});

### Remove this logic from the AppController

### Move the restaurants route config into the restaurants module and load the new controller

angular.module('restaurants', [])

.config(function($routeProvider) {

$routeProvider

.when('/restaurants', {

templateUrl: 'components/restaurants',

controller: 'RestaurantsController as component'

});

})

### Update the restaurants view to use the new RestaurantsController

(Replace all instances of app. with component.)

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

...

<fm-rating id="priceFilter" rating="component.filters.price" glyph="gbp"></fm-rating>

...

<th><a href ng-click="component.sortBy('name')">Name <span ng-class="component.getSortClass('name')">

...

### Update the Protractor HomePage PageObject to reference this new controller

getRestaurantList: function(column) {

var repeater = by.repeater('restaurant in component.filteredRestaurants');

if (column) {

repeater = repeater.column(column);

}

return element.all(repeater);

}

### Check that the protractor tests still pass

$ protractor protractor.conf.js

### Move the restaurant specific unit tests from the AppController into the RestaurantsController

describe('restaurants', function() {

beforeEach(module('restaurants'));

describe('RestaurantsController', function() {

var controller;

beforeEach(inject(function($controller, $httpBackend) {

$httpBackend.when('GET', '../shared/data/restaurants.json').respond([

{ id: 'test1', price: 1, rating: 3 },

{ id: 'test2', price: 2, rating: 4 },

{ id: 'test3', price: 3, rating: 5 }

]);

controller = $controller('RestaurantsController', {});

}));

it('should initialize controller properties', function() {

expect(controller.sortProperty).toEqual('name');

expect(controller.sortDirection).toBe(false);

expect(controller.filters).toEqual({ price: null, rating: null});

});

it('should attach the restaurant data when it arrives', inject(function($httpBackend) {

$httpBackend.flush();

expect(controller.restaurants).toEqual([

{ id: 'test1', price: 1, rating: 3 },

{ id: 'test2', price: 2, rating: 4 },

{ id: 'test3', price: 3, rating: 5 }

]);

}));

describe('filteredRestaurants', function() {

beforeEach(inject(function($httpBackend) {

$httpBackend.flush();

}));

it('should contain the full list after the restaurants have loaded', function() {

expect(controller.filteredRestaurants).toEqual([

{ id: 'test1', price: 1, rating: 3 },

{ id: 'test2', price: 2, rating: 4 },

{ id: 'test3', price: 3, rating: 5 }

]);

});

it('should update the list when the filters change', inject(function($rootScope) {

controller.filters.price = 2;

$rootScope.$digest();

expect(controller.filteredRestaurants).toEqual([

{ id: 'test1', price: 1, rating: 3 },

{ id: 'test2', price: 2, rating: 4 }

]);

controller.filters.rating = 4;

$rootScope.$digest();

expect(controller.filteredRestaurants).toEqual([

{ id: 'test2', price: 2, rating: 4 }

]);

}));

});

});

})

### Update the karma config to load these new files

files: [

'../shared/js/angular.js',

'../shared/js/angular-messages.js',

'../shared/js/angular-message-format.js',

'../shared/js/angular-route.js',

'../shared/js/angular-mocks.js',

'\*.js',

'components/\*/\*.js'

],

## Extras

* Try moving the config for the static views into their own modules

# add animated navigation indicators

## Where are we?

An Angular application running from a local webserver; with unit tests and e2e tests; and componentized routing

## Goals

* Add CSS classes to show which is the currently selected route
* Use ngAnimate to animate these class changes with CSS transitions

## Topics

* ngClass
* ngAnimate
* CSS transitions

## Tasks

### Create a NavigationController in app.js, which provides routeIs() helper

.controller('NavigationController', function($location) {

this.routeIs = function(routeName) {

return $location.path() === routeName;

};

}]);

### Add unit tests for this controller to app.spec.js

describe('NavigationController', function() {

var controller;

beforeEach(module('app'));

beforeEach(inject(function($controller) {

controller = $controller('NavigationController', {});

}));

describe('routeIs', function() {

it('should return true if the current route matches the parameter', inject(function($location) {

$location.path('/test-route');

expect(controller.routeIs('/other-route')).toBe(false);

expect(controller.routeIs('/test-route')).toBe(true);

}));

})

});

### Check that the unit tests still pass

$ karma start --single-run

### Add this controller to the navigation bar using ng-controller

<!-- Navigation Bar -->

<div class="navbar navbar-default" ng-controller="NavigationController as nav">

<div class="container-fluid">

<div class="navbar-header">

<a class="navbar-brand" href="#/">FoodMe</a>

</div>

<div class="collapse navbar-collapse">

<ul class="nav navbar-nav">

<li ng-class="{active: nav.routeIs('/restaurants')}"><a href="#/">Home</a></li>

<li ng-class="{active: nav.routeIs('/how-it-works')}"><a href="#/how-it-works">How it works</a></li>

<li ng-class="{active: nav.routeIs('/about-us')}"><a href="#/about-us">Who we are</a></li>

</ul>

<ul class="nav navbar-nav navbar-right">

<li ng-class="{active: nav.routeIs('/help')}"><a href="#/help">Help</a></li>

</ul>

</div>

</div>

</div>

### Load angular-animate.js in index.html

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

### Add ngAnimate as a dependency of the app module

angular.module('app', ['ngMessages', 'ngMessageFormat', 'ngRoute', 'ngAnimate',

'localStorage', 'rating', 'restaurants'])

### Note that the shared/css/app.css file has relevant CSS transition styles

.navbar-nav > li, .navbar-nav > li > a {

transition: 0.5s ease-out all;

}

.navbar-nav > li.active-add-active a {

background-color: white !important;

}

### Add angular-animate.js to the Karma config

files: [

'../shared/js/angular.js',

'../shared/js/angular-messages.js',

'../shared/js/angular-message-format.js',

'../shared/js/angular-route.js',

'../shared/js/angular-animate.js',

'../shared/js/angular-mocks.js',

'\*.js',

'components/\*/\*.js'

],

### Check that the unit tests still pass

$ karma start --single-run

## Extras

* Define your own transitions for the navigation
* Define CSS classes to animate the restaurant list changing as it is filtered

# add detailed menu view

## Where are we?

An Angular application running from a local webserver; with unit tests and e2e tests; and componentized routing

## Goals

* Refactor restaurant data management into a service

## Topics

* route resolves

## Tasks

### Create a restaurantListPromise service in the restaurants module

.factory('restaurantListPromise', function($http) {

// var url = 'https://foodme.firebaseio.com/.json'; // CORS enabled server

var url = '../shared/data/restaurants.json'; // Local webserver

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

return response.data;

});

})

### Add a resolve to the /restaurants route to inject the list of restaurants into the controller

.config(function($routeProvider) {

$routeProvider

.when('/restaurants', {

templateUrl: 'components/restaurants',

controller: 'RestaurantsController as component',

resolve: {

restaurants: 'restaurantListPromise'

}

});

})

### Update the RestaurantsController to use this injected value

.controller('RestaurantsController', function(restaurants, $rootScope) {

var that = this;

...

var filterRestaurants = function() {

that.filteredRestaurants = [];

angular.forEach(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; }

], filterRestaurants);

});

### Move the unit tests from the RestaurantController into restaurantListPromise

describe('restaurantListPromise', function() {

beforeEach(inject(function($httpBackend) {

$httpBackend.when('GET', '../shared/data/restaurants.json').respond([

{ id: 'test1', price: 1, rating: 3 },

{ id: 'test2', price: 2, rating: 4 },

{ id: 'test3', price: 3, rating: 5 }

]);

}));

it('should return a promise to restaurant data', inject(function($httpBackend, restaurantListPromise) {

var restaurants;

restaurantListPromise.then(function(data) {

restaurants = data;

});

$httpBackend.flush();

expect(restaurants).toEqual([

{ id: 'test1', price: 1, rating: 3 },

{ id: 'test2', price: 2, rating: 4 },

{ id: 'test3', price: 3, rating: 5 }

]);

}));

});

### Inject mock restaurant data directly into the RestaurantsController under test

describe('RestaurantsController', function() {

var controller, restaurants;

beforeEach(inject(function($controller) {

restaurants = [

{ id: 'test1', price: 1, rating: 3 },

{ id: 'test2', price: 2, rating: 4 },

{ id: 'test3', price: 3, rating: 5 }

];

controller = $controller('RestaurantsController', { restaurants });

}));

...

describe('filteredRestaurants', function() {

it('should initially contain the full list of restaurants', inject(function($rootScope) {

$rootScope.$digest();

expect(controller.filteredRestaurants).toEqual([

{ id: 'test1', price: 1, rating: 3 },

{ id: 'test2', price: 2, rating: 4 },

{ id: 'test3', price: 3, rating: 5 }

]);

}));

...

});

});

### Check that the unit tests still pass

$ karma start --single-run

### Check that the e2e tests still pass

$ protractor protractor.conf.js

# add detailed menu view

## Where are we?

An Angular application running from a local webserver; with unit tests and e2e tests; and componentized routing

## Goals

* Add a new view for a single restaurant showing its menu

## Topics

* $routeParams
* route resolves

## Tasks

### Add a new restaurant menu route, with a route parameter :id

.when('/restaurants/:id', {

templateUrl: 'components/restaurants/menu.html',

controller: 'MenuController as component',

resolve: {

restaurants: 'restaurantListPromise'

}

});

### Create a new view for this route at components/restaurants/menu.html

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

<div class="row fm-restaurant">

<div class="col-md-2">

<img ng-src="../shared/img/restaurants/{{component.restaurant.id}}.jpg" class="img-rounded">

</div>

<div class="col-md-10">

<h3>{{component.restaurant.name}}</h3>

<div class="row">

<div class="col-md-2">

<div>{{component.restaurant.address}}</div>

<div ng-bind-html="component.restaurant.rating | rating : 'star'"></div>

<div ng-bind-html="component.restaurant.price | rating : 'gbp'"></div>

</div>

<div class="col-md-4">

<div>{{component.restaurant.description}}</div>

</div>

</div>

</div>

</div>

<div class="row">

<div class="col-md-8">

<div class="fm-panel fm-menu-list">

<div class="fm-heading">Menu</div>

<div class="fm-content">

<ul>

<li ng-repeat="menuItem in component.restaurant.menuItems">

<a href>

<span>{{menuItem.name}}</span>

<span>{{menuItem.price | currency}}</span>

<i class="glyphicon glyphicon-plus"></i>

</a>

</li>

</ul>

</div>

</div>

</div>

<div class="col-md-4">

<div class="fm-panel fm-cart">

<div class="fm-heading">Your order</div>

<div class="fm-content">

<div class="fm-restaurant">{{ component.restaurant.name }}</div>

</div>

</div>

</div>

</div>

</div>

### Create a new controller for this route, which uses the resolved restaurant data

.controller('MenuController', function(restaurants, $routeParams, $location) {

var restaurantId = $routeParams.id;

for(var i=0; i<restaurants.length; i++) {

if (restaurants[i].id == restaurantId) {

this.restaurant = restaurants[i];

break;

}

}

if (!this.restaurant) {

console.log('missing restaurant', restaurantId);

$location.path('/');

}

});

### Update the restaurant list view to link each restaurant to its menu

<td class="description">

<div class="media">

<a class="pull-left" href="#/restaurants/{{restaurant.id}}">

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

</a>

<div class="media-body">

<h4 class="media-heading">

<a href="#/restaurants/{{restaurant.id}}">{{restaurant.name}}</a>

</h4>

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

</div>

</div>

</td>

## Extras

* Write unit tests for the MenuController
* Write new e2e test that demonstrate navigating to a menu

# create custom component directive

## Where are we?

An Angular application running from a local webserver; with unit tests and e2e tests; and componentized routing

## Goals

* Refactor the delivery info form and display box into a reusable fmDeliverTo directive
* Use fmDeliverTo in different places in the restaurant list and menu views

## Topics

* Directives
* Isolated Scope
* Testing Directives
* karma-ng-html2js-preprocessor

## Tasks

### Define fmDeliverTo module with fmDeliverTo directive in fmDeliverTo.js

angular.module('fmDeliverTo', [])

.directive('fmDeliverTo', function() {

return {

restrict: 'E',

templateUrl: 'fmDeliverTo.template.html',

controller: 'FmDeliverToController as directive',

bindToController: {

user: '=deliverTo'

},

scope: {}

};

});

### Move the fmDeliverTo logic from the AppController into a new FmDeliverToController

.controller('FmDeliverToController', function() {

this.deliveryFormVisible = true;

this.showDeliveryForm = function() {

this.deliveryFormVisible = true;

};

this.hideDeliveryForm = function() {

this.deliveryFormVisible = false;

};

})

### Move the fmDeliverTo HTML from the index.html file to the fmDeliverTo.template.html file

<!-- Delivery Info Form -->

<div class="row delivery-info-form" ng-show="directive.deliveryFormVisible">

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

<form role="form" class="well" name="directive.deliveryForm">

<a href="" class="pull-right" ng-click="directive.hideDeliveryForm()">Hide</a>

<legend>Delivery Details</legend>

<div class="form-group" ng-class="{'has-error': directive.deliveryForm.userName.$invalid}">

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

<input type="text" id="customerName" class="form-control" ng-model="directive.user.name" name="userName" required ng-minlength="5">

</div>

<div ng-messages="directive.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 class="form-group" ng-class="{'has-error': directive.deliveryForm.userAddress.$invalid}">

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

<input type="text" id="address" class="form-control" ng-model="directive.user.address" name="userAddressInput" required ng-minlength="10">

</div>

<div ng-messages="directive.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>

</form>

</div>

</div>

<!-- Delivery Info -->

<div class="row delivery-info-box" ng-hide="directive.deliveryFormVisible">

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

<div class="well">

<a href="" class="pull-right" ng-click="directive.showDeliveryForm()">Change</a>

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

<span>{{ directive.user.name }}</span><br>

<span>{{ directive.user.address }}</span>

</div>

</div>

</div>

### Move the unit test logic from the AppController spec to the FmDeliverToController spec

describe('fmDeliverTo module', function() {

beforeEach(module('fmDeliverTo'));

describe('FmDeliverToController', function() {

beforeEach(inject(function($controller) {

controller = $controller('FmDeliverToController', {});

}));

it('should initialize controller properties', function() {

expect(controller.deliveryFormVisible).toBe(true);

});

describe('showDeliveryForm', function() {

it('should set deliveryFormVisible to true', function() {

controller.deliveryFormVisible = null;

controller.showDeliveryForm();

expect(controller.deliveryFormVisible).toBe(true);

});

});

describe('hideDeliveryForm', function() {

it('should set deliveryFormVisible to false', function() {

controller.deliveryFormVisible = null;

controller.hideDeliveryForm();

expect(controller.deliveryFormVisible).toBe(false);

});

});

});

});

### Install the Karma HTML to JavaScript preprocessor to help with testing the directive

npm install karma-ng-html2js-preprocessor --save-dev

### Update the karma config for this HTML to JavaScript preprocessor

preprocessors: {

'\*\*/\*.template.html': ['ng-html2js']

},

ngHtml2JsPreprocessor: {

},

### Write unit tests for the fmDeliverTo directive, loading the module containing the template

describe('fmDeliverTo module', function() {

beforeEach(module('fmDeliverTo'));

...

describe('fmDeliverTo directive', function() {

var scope, element;

beforeEach(module('fmDeliverTo.template.html'));

beforeEach(inject(function($compile, $rootScope) {

$rootScope.testUser = {

name: 'Test Name',

address: 'Test Address'

};

scope = $rootScope;

element = $compile('<fm-deliver-to deliver-to="testUser"></fm-deliver-to>')($rootScope);

$rootScope.$digest();

}));

it('should bind the user info to the form inputs', function() {

var userNameInput = element[0].querySelector('input[ng-model="directive.user.name"]');

expect(userNameInput.value).toEqual('Test Name');

var userAddressInput = element[0].querySelector('input[ng-model="directive.user.address"]');

expect(userAddressInput.value).toEqual('Test Address');

});

it('should bind the user info display boxes', function() {

var displayBox = element[0].querySelector('.delivery-info-box');

expect(displayBox.textContent).toContain('Test Name');

expect(displayBox.textContent).toContain('Test Address');

});

it('should update the model given by the `deliver-to` expression when the inputs are changed', function() {

var userNameInput = angular.element(element[0].querySelector('input[ng-model="directive.user.name"]'));

userNameInput.val('Other Name');

userNameInput.triggerHandler('change');

expect(scope.testUser.name).toEqual('Other Name');

var userAddressInput = angular.element(element[0].querySelector('input[ng-model="directive.user.address"]'));

userAddressInput.val('Other Name');

userAddressInput.triggerHandler('change');

expect(scope.testUser.address).toEqual('Other Name');

});

});

});

### Check that the unit tests still pass

$ karma start --single-run

### Use the fmDeliverTo directive in the left side bar of the restaurant list view

<!-- Restaurant List -->

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

<fm-deliver-to deliver-to="app.user"></fm-deliver-to>

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

<legend>Filter Restaurants</legend>

...

### Use the fmDeliverTo directive in the right side bar of the menu view

<div class="col-md-4">

<fm-deliver-to deliver-to="app.user"></fm-deliver-to>

<div class="fm-panel fm-cart">

<div class="fm-heading">Your order</div>

<div class="fm-content">

<div class="fm-restaurant">{{ component.restaurant.name }}</div>

</div>

</div>

</div>

### Update the HomePage Protractor page object for the new user bindings

Rename the app.user.name, app.hideDeliveryForm and app.showDeliveryForm expressions with directive.user.name, directive.hideDeliveryForm and directive.showDeliveryForm expressions respectively.

### Check that the e2e tests still pass

$ protractor protractor.conf.js

## Extras

* Update the e2e test that demonstrates navigating to a menu, if you wrote one

# add Shopping Cart functionality

## Where are we?

An Angular application running from a local webserver; with unit tests and e2e tests; reusable directives and componentized routing

## Goals

* Provide a Shopping Cart facility to store items from the menu

## Topics

* Services

## Tasks

### Create a shoppingCart module, which depends upon localStorage

angular.module('shoppingCart', ['localStorage'])

### Load the shoppingCart module in index.html

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

### Add the shoppingCartModule as a dependency of the app module

angular.module('app', ['ngMessages', 'ngMessageFormat', 'ngRoute', 'ngAnimate',

'localStorage', 'rating', 'restaurants', 'fmDeliverTo',

'shoppingCart'])

### Create an alert service to wrap the browser's alert function for easier testing

.value('alert', window.alert)

### Create a shoppingCart service that stores the cart info in localStorage

.factory('shoppingCart', function(localStorageBinding) {

return localStorageBinding('fmShoppingCart', {

items: [],

restaurant: {}

});

})

### Create a ShoppingCartController that exposes the shoppingCart and helper methods

.controller('ShoppingCartController', function(shoppingCart, alert) {

this.cart = shoppingCart;

this.items = function() {

return this.cart.items;

};

this.restaurant = function() {

return this.cart.restaurant;

};

this.add = function(choice, restaurant) {

if ( !this.cart.restaurant.id ) {

this.cart.restaurant = restaurant;

}

if ( this.cart.restaurant.id !== restaurant.id ) {

alert('You cannot mix items from different restaurant - clear the shopping cart first.');

return;

}

angular.forEach(this.cart.items, function(item) {

if (choice && choice.name === item.name) {

item.amount += 1;

choice = null;

}

});

if (choice) {

this.cart.items.push({

name: choice.name,

price: choice.price,

amount: 1

});

}

};

this.remove = function(cartItem) {

var index = this.cart.items.indexOf(cartItem);

if ( index !== -1 ) {

this.cart.items.splice(index, 1);

}

if (this.cart.items.length === 0) {

this.cart.restaurant = {};

}

};

this.total = function() {

var sum = 0;

angular.forEach(this.cart.items, function(item) {

sum += Number(item.price \* item.amount);

});

return sum;

};

this.reset = function() {

this.cart.items = [];

this.cart.restaurant = {};

};

});

### Attach the ShoppingCartController to the menu view using ng-controller

<div class="row" ng-controller="ShoppingCartController as shoppingCart">

<div class="col-md-8">

<div class="fm-panel fm-menu-list">

<div class="fm-heading">Menu</div>

...

### Add a new item to the cart from the menu items when the plus sign is clicked

<li ng-repeat="menuItem in component.restaurant.menuItems">

<a href>

<span>{{menuItem.name}}</span>

<span>{{menuItem.price | currency}}</span>

<i class="glyphicon glyphicon-plus" ng-click="shoppingCart.add(menuItem, component.restaurant)"></i>

</a>

</li>

### Display the shopping cart items and total in the side panel, with option to remove items by clicking

<div class="fm-content">

<div class="fm-restaurant">{{ shoppingCart.restaurant().name }}</div>

<ul class="list-unstyled">

<li ng-repeat="cartItem in shoppingCart.items()">

<a ng-click="shoppingCart.remove(cartItem)">

<i class="glyphicon glyphicon-remove"></i>

</a>

{{cartItem.amount}} &times; {{ cartItem.name }}

</li>

</ul>

<div class="pull-right">

<a href class="btn btn-primary">Checkout</a>

</div>

<div class="fm-total">Total: {{shoppingCart.total() | currency}}</div>

</div>

## Extras

* Write unit tests for the ShoppingCartController (mocking alert)
* Write e2e tests for adding items to the cart

# add injection annotations

## Where are we?

An Angular application running from a local webserver; with unit tests and e2e tests; reusable directives and componentized routing

## Goals

* Add injection annotations to prevent minification issues

## Topics

* ng-annotate
* gulp
* gulp-ng-annotate

## Tasks

### Install gulp and gulp-ng-annotate

$ npm install -g gulp $ npm install --save-dev gulp gulp-ng-annotate

### Create a gulp build file to add the annotation adder

var gulp = require('gulp');

var ngAnnotate = require('gulp-ng-annotate');

var annotateOptions = {

remove: true,

add: true,

single\_quotes: true

};

gulp.task('default', function () {

return gulp.src([

'\*.js',

'components/\*/\*.js',

'!protractor.conf.js',

'!gulpFile.js',

'!karma.cong.js'

])

.pipe(ngAnnotate(annotateOptions))

.pipe(gulp.dest('.'));

});

### Run the default gulp task

$ gulp

### Add ng-strict-di directive to ensure all components are correctly annotated

<body class="container" ng-app="app" ng-strict-di ng-controller="AppController as app">

# Next Steps

* Add a checkout and thank you view
* Use Gulp and Uglify to minify the JS code