Exercise Unit 1 – Laying out A Page

In this exercise, you will convert a basic HTML 4 page to an HTML5 page. The

CSS documents have already been created for you, such that, when you're finished, your HTML5 page should render exactly like the HTML 4 page.

1. Open html5-laying-out-a-page/Exercises/html4-layout.html.
2. Save the file as html5-layout.html.
3. Turn the page from an HTML 4 page into an HTML5 page. Make sure to

change the stylesheet reference to point to style-html5.css.

Exercise Unit 2 – Create the App

In this exercise, you will convert a basic HTML 4 page to an HTML5 page. The

CSS documents have already been created for you, such that, when you're finished, your HTML5 page should render exactly like the HTML 4 page.

1. Add an exercise?

Exercise Unit 3

a. Sections

In this exercise, you will modify an HTML page we worked on earlier in the course to replace meaningless div elements with meaningful section and article elements.

1. Open html5-sections/Exercises/html5-layout.html.
2. Replace meaningless div elements with meaningful section and article

elements. Note that there is room for interpretation here, so there is no one correct solution.

1. To keep the page looking as it did before, you will also need to modify html5-sections/Exercises/style-html5.css.

b. Determining the Outline

In this exercise, you will try to determine the outline of an HTML page.

1. Review the code below.

2. Create a list either on paper or in a text editor or word processor that shows the HTML outline as specified by the HTML5 specification.

Exercise Unit 4 – Services

1. Create a “film” service in a new file
   1. Put film-service.js into a new scripts/services folder
2. Have the controller get the list of films from the service
   1. Move the films array into the service
   2. Define a getter function
   3. Remember to inject the service into your controller
3. Have the controller call the service in order to add items to the array
4. Create a remove-film function inside your service. Reference it in your controller.
5. Create an “omdb” factory-service to be able to search for films by title
   1. You will need to make a GET request to http://www.omdbapi.com/
   2. Your search value should be assigned to the t key in the query string params (i.e. /?t=searchString)
   3. The call will return a film object
   4. Make sure you are returning the film object to the controller if successful, not the entire response object
   5. Remember to handle any errors
6. Add the ability for the user to be able to search for a film and add it to their list
   1. Add an input box that will call a function on the controller that will search the OMDB
   2. Display the returned results, including the movie poster, for the user
   3. Add a button that will add the new film to the list

Exercise Unit 5 – Directives

1. Create a template expanding “film” directive that wraps the display of each of the films in our list
   1. Put film.js into a new scripts/directives folder
   2. Put the film.html template file into a new views/directives folder
   3. Make sure that you explicitly pass in any properties you want to use, not the entire film object
2. Change your ng-repeat to use this directive

Exercise Unit 6 – Filters

1. Add a search field for your films list that will dynamically filter the list

1. Order the list alphabetically by title
2. Add a “watched” Boolean property to your films
   1. Make sure that you give the user the ability to toggle this value for each film
3. If the film is marked as watched, hide it from the list using a built in Angular directive
4. *Challenge:* Create a custom filter that hides any film marked as watched from the list

Exercise Unit 7 – Routes

1. Add the ngRoutes module to your app
   1. Install the dependency with Bower

$ bower install angular-route --save

* 1. Include the file in your index.html

<script src=”bower\_components/angular-route/angular-route.min.js”></script>

* 1. Inject the dependency into your app module

1. Refactor your current view into a new route
   1. Define the /to-watch route in app.js
   2. Make /to-watch the default route
   3. Place the new to-watch.html file in the views folder
   4. Add the ng-view directive to index.html
2. Create new pages, controllers, and routes for /faves, and /watched
3. Add a nav bar using some Bootstrap code:to

<nav class="navbar navbar-default navbar-fixed-top">

<div class="container-fluid navigation-container">

<div class="navbar-header">

<button type="button" class="navbar-toggle collapsed">

<span class="icon-bar"></span>

<span class="icon-bar"></span>

<span class="icon-bar"></span>

</button>

<a class="navbar-brand" href="#/to-watch">Movie To Do</a>

</div>

<div class="collapse navbar-collapse">

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

<li><a class="navbar-link" href="#/to-watch">watchlist</a></li>

<li><a class="navbar-link" href="#/watched">watched</a></li>

<li><a class="navbar-link" href="#/faves">favourites</a></li>

</ul>

</div>

</div>

</nav>

1. Create your “faves” page showing a list of the films that you have marked as favourite
   1. Make sure you add a search box at the top of the list
   2. Remember to use the directive you built and to pull the data from the service
2. Create your watched page showing a list of the films that you have marked as watched
   1. This will be very similar to your faves page
3. **Hint:** Perhaps you should make separate controllers for your watched and favourites pages.

Exercise Unit 8 – Forms

1. Add some validation to your Add a Movie form
   1. Make sure that the text colour changes if there is an error with a field
   2. Make sure that helpful tips are displayed if there are errors with the form
   3. The user should not be able to submit the form if there are errors
   4. Add some code into to-watch controller that resets your Add Film form once you add a film

Exercise Unit 9 – Animation

1. Add the ngAnimate module to your app

$ bower install angular-animate –save

1. Add a CSS3 transition to your app so that when the routes change, the new view slides in from the right side while the old view slides out the left side
2. *Challenge:* Add an animation to your lists so that they move and fade when items are added and removed

Exercise Unit 10 – Unit Testing

Initialize our project with npm:

$ npm init

*name:* ***angularjs-fundamentals***

*version:* ***0.0.1***

*description:* ***This is my first angular app***

*entry point:* ***index.html***

*test command:* ***karma start karma.conf.js***

*git repository:*

*keywords:*

*author:* ***Your Name <me@somedomain.com>***

*license:* ***MIT***

*Is this ok?* ***Yes***

Install Karma, Jasmine, and Phantom:

$ npm install jasmine-core karma phantomjs karma-jasmine karma-phantomjs-launcher –-save-dev

Install the ngMock module:

$ bower install angular-mocks --save

Initialize our project with Karma:

$ karma init

*Which testing framework do you want to use?* ***jasmine***

*Do you want to use Require.js?* ***No***

*Do you want to capture any browsers automatically?* ***PhantomJS***

*What is the location of your source and test files?*

***test/\*\*/\*.js***

***scripts/\*\*/\*.js***

***app.js***

*Should any of the files included 3by the previous patterns be excluded?*

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

To include the angular dependency files in our tests, add the following lines to the files array in karma.conf.js:

'bower\_components/angular/angular.js',

'bower\_components/angular-mocks/angular-mocks.js',

1. Create unit tests for your controllers, directives, and services
   1. Your \*.spec.js files should be put in the following directory structure with the same name as the source file to allow for easy navigation of the test suite

test

spec

controllers

directives

services

Exercise Unit 11 – Custom Modules

1. The app currently resets its list of films whenever we refresh the page. Take a look at including the ngStorage module ( https://github.com/gsklee/ngStorage ) to have the film service store and retrieve the list from the browsers localStorage
   1. Make sure that the app loads its initial list from localStorage if there is one there and updates that stored list whenever changes are made
2. The mobile dropdown list in our navigation bar is currently broken. No amount of clicking on the hamburger button will fix it. Take a look at including the collapse directive from UI Bootstrap ( <https://angular-ui.github.io/bootstrap/> ) and a new controller to make the dropdown appear on mobile.
   1. Also remember that the dropdown should go away when we click on one of the navigation links