**Introduction to Angular JS**

Himani Mense

CS Fall 2018

B00755952

Table of contents

1. Module 1
2. Learning outcomes
3. Introduction to AngularJS
4. Module 2
5. Angular Controllers
6. $scope
7. Sharing data across controllers
8. Module 3
9. Expressions
10. Module 4
11. Directives
12. Dependency Injection
13. Module 5
14. Routing, View Management and UI router
15. $RouteProvider
16. PrettyURLs
17. Module 6
18. Factory, Provider, Services
19. Best AngularJS practices

**MODULE 1:**

* Learning outcomes :

1. Explain what Angular JS is used for
2. Identify what Angular JS modules are
3. Discuss how Angular JS works
4. Identify why you would use Angular JS in your website development

* Introduction to Angular JS :

1. Angular JS is a front end Javascript framework for creating web application.
2. It is open source maintained by Google.
3. It handles common tasks such as DOM manipulation, updating UI based on data or input, registering callbacks.
4. It is declarative programming.

* Why use Angular ?

1. It is good for dynamic websites(CRUD based)
2. It allows to separate tasks, functions and functionalities preventing overriding.
3. It is easy to test since everything is modular(easy to break, hence testing is simple)

* How Angular JS works?

1. Angular JS will initialize when the DOM content is loaded.
2. It looks for the “ng-app” directive. If found that is the root of the app.
3. Directives can be declared by a variety of ways: ng-prefix, also data-ng
4. It will load the module associated with the directive if specified.

* Modules : These are containers for various parts of app and define dependencies for the app.

An AngularJS application must create a top level application module. This application module can contain other modules, controllers, filters, etc.

The angular.module() method creates an application module, where the first parameter is a module name which is same as specified by ng-app directive. The second parameter is an array of other dependent modules [].

Example: <!DOCTYPE html>

<html >

<head>

<script src="~/Scripts/angular.js"></script>

</head>

<body ng-app="myApp">

@\* HTML content \*@

<script>

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

</script>

</body>

</html>

**MODULE 2 :**

* **Angular Controllers :**

1. A controller is a set of javascript functions bound to a scope.
2. The ng-controller directive tells Angular to instantiate the new controller object and inject the new scope as a dependency.
3. Controllers are best for adding logic(but avoid using to manipulate DOM)

Short form: CTRL

Adding behavior: Through functions(getters and setters)

* **$scope :**

1. It is an object that refers to the app model.
2. It provides the execution context for expressions.
3. It acts as a glue between controller and the view.
4. It helps attach properties/functions to scope to make them available to the view.

* **Sharing data across controllers :**

1. Using Factory
2. Using values
3. Using services

**MODULE 3 :**

* **Expressions :**

1. These are javascript code snippets placed in bindings.
2. They are evaluated against scope object and can be used with filters to format data.
3. They are forgiving as they may not throw up errors.

Example : {{Hello World}}

**MODULE 4 :**

* **Directives :**

1. These are markers on the DOM that tell the angular compiler to attach behavior or transform the DOM element.
2. They can be placed on element, attribute or CSS.
3. One should prefer using the dash-delimited format (e.g. ng-model for ngModel). While working with an HTML validating tool, one could instead use the data-prefixed version (e.g. data-ng-model for ngModel).
4. Prefer using directives via tag name and attributes over comment and class names. Doing so generally makes it easier to determine what directives a given element matches.
5. While creating directives, it is recommended to prefix your own directive names to avoid collisions with future standard.
6. ng-click : The ng-click directive tells AngularJS what to do when an HTML element is clicked.

Example : ng-app, ng-controller, ng-src, ng-href, ng-class (built-in directives)

* Creating a custom directive :

1. You need some kind of HTML
2. You register directive on the module.

### **Dependency Injection**

The preferred way of injecting the dependencies is by passing the dependency to the constructor function rather than using one of the following other ways. In this way, the responsibility of creating the dependency object lies with other objects or function. This is straight forward for those who have worked with one or more dependency injection framework in the past. However, this could be useful information for the beginners. Following are other ways of doing dependency injection in Angular:

* Create it using the new operator.
* Look for it in a well-known place, also known as a global singleton.
* Ask a registry (also known as service registry) for it.

Restrict: It shouldn’t be used as element, class or attribute

**MODULE 5 :**

* **Routing, View Management and UI router :**

Routing with ng-route :

* 1. AngularJS ngRoute module provides routing, deep linking services and directives for angular applications.
  2. You dynamically load the content using $routeProvider and ng-view

if you are bundling this file into your application, then you can add it to your page with below code.

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

If you want to include it from Google CDN, then use below code.

<script src="https://ajax.googleapis.com/ajax/libs/angularjs/1.2.28//angular-route.min.js"></script>

Then load the ngRoute module in your AngularJS application by adding it as a dependent module as shown below.

angular.module('appName', ['ngRoute']);

**ngView** : this directive is used to display the HTML templates or views in the specified routes. Every time the current route changes, the included view changes with it according to the configuration of the $route service.

### **$routeProvider :**

$routeProvider is used to configure the routes. We use the ngRoute config() to configure the $routeProvider. The config() takes a function which takes the $routeProvider as parameter and the routing configuration goes inside the function.

$routeProvider has a simple API, accepting either the when() or otherwise() method.

# **Pretty URLs in AngularJS: Removing the #**

1. Pretty url get rid of hashtag on URLs that angular uses by default
2. To do so, we need to configure the $locationProvider and set base for relative URLs

**$locationProvider :** we use the $locationProvider to configure how the application deep linking paths are stored.

* **UI Router:**

1. Routing framework built by Angular team
2. It changes view based on application state, not just route URL
3. Links use ui-sref
4. It is separate and needs to be loaded.

**MODULE 6 : Factory, Provider, Services**

Angular provides us with three ways to create and register our own service.

1. Factory
2. Service
3. Provider

**Getting external data:**

$HTTP – is a built-in resource to make REST requests

Example :

**Factory:** The **factory** services are used to communicate with servers using HTTP and also used to create reusable code for application. The factories in Angularjs will always return a new instance for each object but it will not happen in services in Angularjs.

**Services :** AngularJS services are substitutable objects that are wired together using dependency injection (DI). You can use services to organize and share code across your app. AngularJS **services** are: Lazily instantiated – AngularJS only instantiates a service when an application component depends on it.

**Providers** : are the only service you can pass into your .config() function. Use a provider when you want to provide module-wide configuration for your service object before making it available.

**Best AngularJS practices :**

* **Small and Pure functions**

Use small functions because they are easy to test and maintain. Pure functions are which return output depending upon some input. These functions don't change input in any way and only job is to produce an output.

// **Pure** **function**

**function** add (a, b) {

**return** a + b;

}

// Not **pure** **function**

**function** add (a, b) {

var x = a + b;

console.log(x); // Doing other job than returning output

**return** x;

}

* **Modules and Single Responsibility**

Divide your code in seperate modules and every modules should be responsible for a single functionality. This increase usability.  
For example you are building a mailing application like gmail. You divide functionality in three pieces (filters, search bar and email listing). When you develop modules based on single responsibility then you can easily manage and test your code.

## **use controllerAs syntax**

When using controller anywhere in your angularJs application try using controller as syntax.

<**div** ng-controller="userController as user">

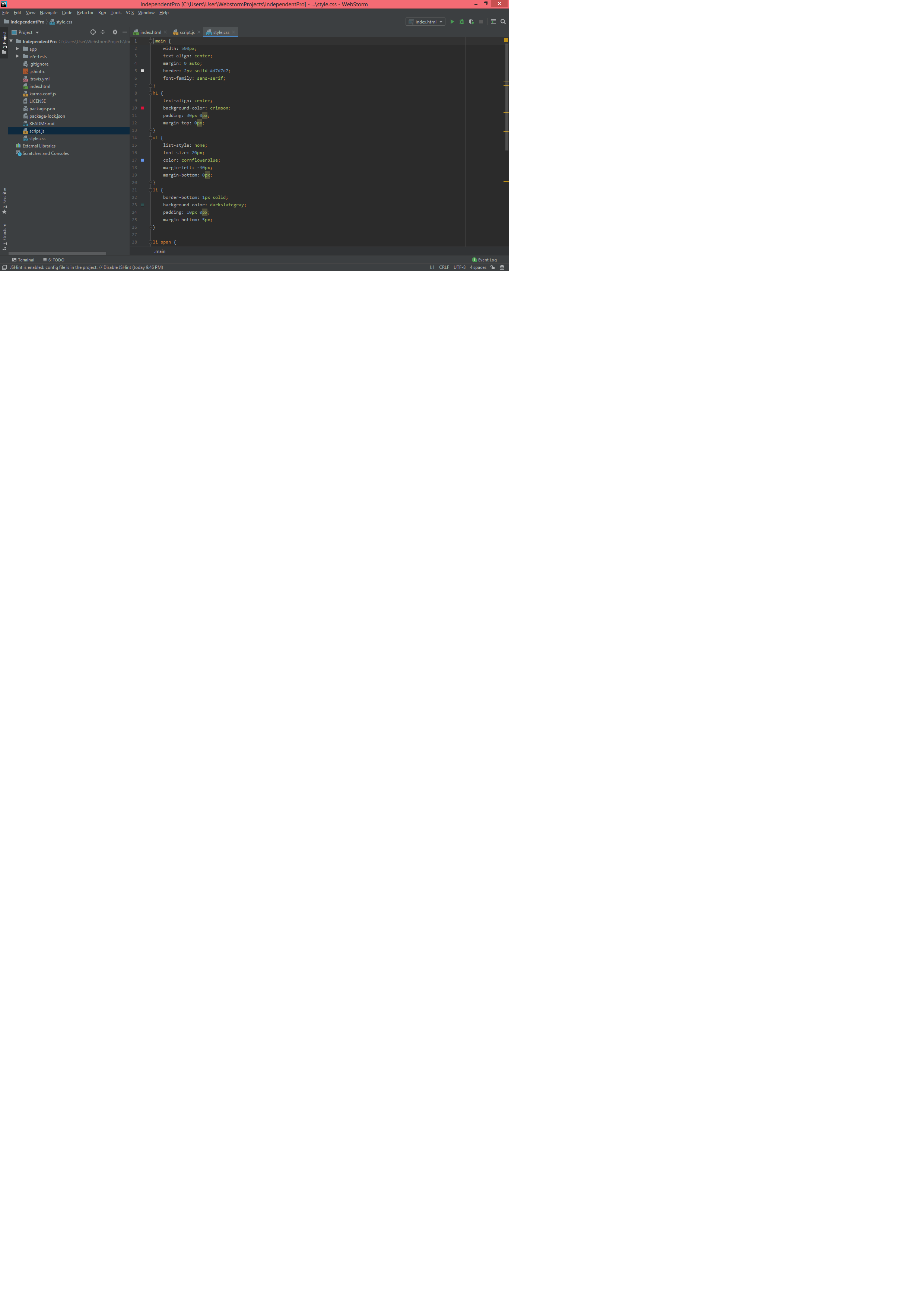
{{ user.name }}

</**div**>

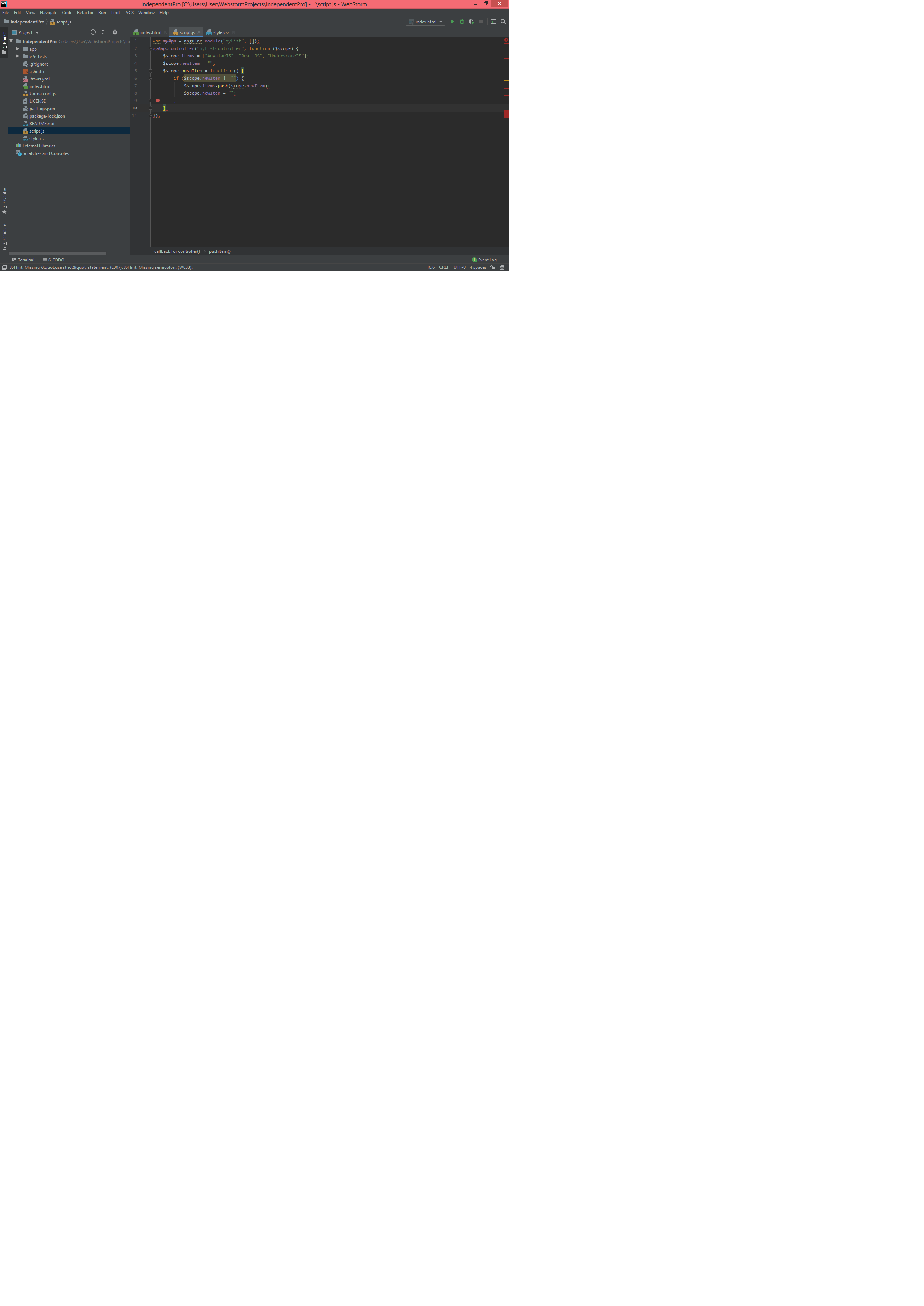
Angular App :

1. The website designed by me using AngularJS is a simple application that allows user to select various items from the available menu and also add a new item to existing menu.

Below are the snaps of the designed app :



A close up of a device

Description automatically generated

A close up of a logo

Description automatically generated



**Thank you**

**for**

**your guidance**

**and support.**