Team: PixPerfect

Member: Alan Chan, Shuyang Yu, Shuoran Zhang

Class: CMPE 280 Web UI Design

Professor: Chandrasekar Vuppalapati

**Implementing AngularJS to A Single Page Web Application**

The goal of our project is to offer users a quick and easy way to explore real estate price history in different regions. As silicon valley is becoming a hot place where people from all over the world are moving to, there are huge housing needs growing year after year. We plan to build a powerful real estate historical price search website with easy usability and clear analysis insight. Users can locate the place by zip code, address and city names, and view the past real estate price records. Easy use, clean design, and fast speed are the three core values that we want to deliver. Our team wants to build a single page web application, and AngularJS is powerful and efficient on fulfilling this task.

In this days and ages, users do not browse websites the way they used to. They do not just sit in front of a desktop in a room and surf the web. Roughly 77% of the American own a smartphone [1]. Mobile web browsing has officially surpassed desktop in October 2016. Out of the 2.5 million websites across the internet [2], 51.3% of traffic was from mobiles or tablets instead of desktops [3]. Naturally, websites have to somehow condense all the information to fit in that little 5inches screen. Single-page application (SPA) came into the picture. It describes an application that was built for the web. In the good old days, the back-end server would be serving the website page by page. Users would see the whole page refreshing for each link/button they click. SPA reduces the amount of full page refresh scientifically by using AJAX. This technology provides a rich interface, as the user interacts with the website, the browser will make certain server request to the server side to obtain more data to display to the users. This represent a fat client model, which outdated mobile or tablet might have a hard time rendering the web page. Developer can have the server render the whole page and serve it as it is to the client to help with the load [4].

So, why use AngularJS one might ask. As mentioned previously, AngularJS is powerful and efficient. It is one of the best open source frameworks to develop Single Page Web Application. AngularJS was originally developed by a Google employee back in 2009 and Google eventually took over the project, and made it open source. With Google and the community backing it, improvement of the framework is constantly being made, questions can be answered by experts or the community. It also does not require a developer to pickup any other language as it is purely HTML and JavaScript. As mentioned above, one of the characteristic SPA is the reduced amount of full page refresh, and AngularJS can turn a static web page to a dynamic web page [5]. AngularJS allow all the pages to be preloaded and cached on the client side, this means a user can still “browse” the site if the network is down [6]. AngularJS is powerful because of its ability to allow developers to decouple the control logic and the data of an UI. The separation enhances the testability of the web page logic, and it also permits the web developers to focus on the desired UI development solely. The framework give the developer/designer to have animation on their web page with little development code [7].

Now, let us take a closer look at how AngularJS helps coders to set up a single page web application. First of all, we know that single page web application is building all html content under a single page as we mentioned earlier. Everything we do is just adding, deleting or swapping html content inside this single page. In AngularJS, a directive library “ng-view” helps us to achieve this goal. In order to use this directive library, we need to set up $route service for the web application as well. These two AngularJS technologies build up the initial single page web application structure together. So let us divide the concept into two parts, first we will dig into ng-view directive and explain what is AngularJS directives, and second we will explain what $route service is and what functionality it offers.

Firstly, what are AngularJS Directives? Generally, directives are marks such as comments, CSS classes, element name, or attributes on a DOM element which inform AngularJS’s HTML $compiler to attach a corresponding specific behavior to that DOM element. Once AngularJS is installed, there are already a variety of built-in directives for users to use. For instance, ngIf, ngShow, ngClass, ngBind, etc. Creating directives is like creating controllers, users have the freedom to create customized directives. Now let us go through some code examples on how to use the built-in directives.

Class style directives:

Matching the input elements to ngModel directive:

<input ng-model="foo">

Binding the input elements to ngBind directive:

<input ng-bind="foo">

Custom label style directives:

<person>{{fullName}}</person>

Beside these built-in directives, we are most interested in how to build customized directives to fulfill specific needs. When we create directives, we need to use the ‘module.directive’ API. Also, here are some important basic rules to follow when we create directives. In order to avoid conflicts with future AngularJS built-in directives, there are mainly two things to pay attention to. Firstly, we need to add customized prefix for directives name, so we can avoid conflicting with future possible tags introduced by HTML. Secondly, we need to avoid naming our directives starting with ‘ng’, so we do not conflict with AngularJS built-in directives. Now, let us go through several common methods of building customized directives and their best practices.

Template-expanding drective:

<div ng-controller="Controller">  
 <div my-customer></div>  
</div>

angular.module('docsSimpleDirective', [])  
.controller('Controller', ['$scope', function($scope) {  
 $scope.customer = {  
 name: 'Naomi',  
 address: '1600 Amphitheatre'  
 };  
}])  
.directive('myCustomer', function() {  
 return {  
 template: 'Name: {{customer.name}} Address: {{customer.address}}'  
 };  
});

Isolating the Scope of a Directive:

**Name: {{customer.name}} Address: {{customer.address}}**

<div ng-controller="NaomiController">  
 <my-customer></my-customer>  
</div>  
<hr>  
<div ng-controller="IgorController">  
 <my-customer></my-customer>  
</div>

angular.module('docsScopeProblemExample', [])  
.controller('NaomiController', ['$scope', function($scope) {  
 $scope.customer = {  
 name: 'Naomi',  
 address: '1600 Amphitheatre'  
 };  
}])  
.controller('IgorController', ['$scope', function($scope) {  
 $scope.customer = {  
 name: 'Igor',  
 address: '123 Somewhere'  
 };  
}])  
.directive('myCustomer', function() {  
 return {  
 restrict: 'E',  
 templateUrl: 'my-customer.html'  
 };  
});

Creating a Directive that Manipulates the DOM:

**<div ng-controller="Controller">  
 Date format: <input ng-model="format"> <hr/>  
 Current time is: <span my-current-time="format"></span>  
</div>**

angular.module('docsTimeDirective', [])  
.controller('Controller', ['$scope', function($scope) {  
 $scope.format = 'M/d/yy h:mm:ss a';  
}])  
.directive('myCurrentTime', ['$interval', 'dateFilter', function($interval, dateFilter) {  
  
 function link(scope, element, attrs) {  
 var format,  
 timeoutId;  
  
 function updateTime() {  
 element.text(dateFilter(new Date(), format));  
 }  
  
 scope.$watch(attrs.myCurrentTime, function(value) {  
 format = value;  
 updateTime();  
 });  
  
 element.on('$destroy', function() {  
 $interval.cancel(timeoutId);  
 });  
  
 // start the UI update process; save the timeoutId for canceling  
 timeoutId = $interval(function() {  
 updateTime(); // update DOM  
 }, 1000);  
 }  
  
 return {  
 link: link  
 };  
}]);

Also the restrict options and their meanings:

A: only matches attribute name

E: only matches element name

C: only matches class name

M: only matches comment

AEC: matches attribute or element or class name [8]

Now that we have a decent understanding of AngularJS directives, it is time for us to move to router services. We mainly want to introduce two routing technologies here: ng-router, ui-router. First, we need to understand why we need routing technology to build single page web application. As mentioned earlier, we are putting everything into one single page, and we add, delete or swap content according to our needs. When we have lots of subpages, we maintain the same header and footer, and we keep swapping the ngView content with a subpage we decide to jump into. Routing service helps us to notify the AngularJS html compiler to switch html content according to the subpage user directed at. There is a routing map inside of routing service which claims which html template and JS controller is bond with which url. When certain url jump event got triggered, the corresponding HTML template and JS controller will be placed into the ngView div. Let us see an example from ngRouter and uiRouter separately. [9]

Ng-router:

<body ng-app="myApp">

<p><a href="#/!">Main</a></p>

<a href="#!red">Red</a>

<a href="#!green">Green</a>

<a href="#!blue">Blue</a>

<div ng-view></div>

<script>

var app = angular.module("myApp", ["ngRoute"]);

app.config(function($routeProvider) {

$routeProvider

.when("/", {

templateUrl : "main.htm"

})

.when("/red", {

templateUrl : "red.htm"

})

.when("/green", {

templateUrl : "green.htm"

})

.when("/blue", {

templateUrl : "blue.htm"

});

});

</script>

</body>

[10]

Ui-router:

<html>  
 <head>  
 <script src="lib/angular.js"></script>  
 <script src="lib/angular-ui-router.js"></script>  
 <script src="helloworld.js"></script>  
  
 <style>.active { color: red; font-weight: bold; }</style>  
 </head>  
  
 <body ng-app="helloworld">  
 <a ui-sref="hello" ui-sref-active="active">Hello</a>  
 <a ui-sref="about" ui-sref-active="active">About</a>  
  
 <ui-view></ui-view>  
 </body>  
</html>

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

myApp.config(function($stateProvider) {  
 var helloState = {  
 name: 'hello',  
 url: '/hello',  
 template: '<h3>hello world!</h3>'  
 }  
  
 var aboutState = {  
 name: 'about',  
 url: '/about',  
 template: '<h3>Its the UI-Router hello world app!</h3>'  
 }  
  
 $stateProvider.state(helloState);  
 $stateProvider.state(aboutState);  
});

[11]

In AngularJS, when a controller is added to DOM by the “ng-controller”, ng will call that controller’s constructor function to generate an object of the controller. In this way, a new scope is generated. We will use the controller to initialize the $scope object, and also add the new function/method to this $scope object. The controller of AngularJS has scope, which is the DOM that contains the controller tag. The function/method in this controller will bond with the associate DOM within this scope, for example:

1. <div class="col-md-3" id="login" ng-controller="LoginCtrl">
2. <input type="text" ng-model="UserInfo.UserName" />
3. <input type="password" ng-model="UserInfo.Password" />
4. <button ng-click="Login()">submit</button>
5. </div>
6. <script>
7. var loginApp = angular.module("loginApp", []);
8. loginApp.controller('LoginCtrl', function ($scope, $http) {
9. $scope.UserInfo={
10. UserName:"admin",
11. Password:"admin"
12. }
13. $scope.Login=function() {
14. $http.post('/login.php', $scope.UserInfo).success(function (data,status) {
15. console.log('Success logged in');
16. }).error(function (data,status) {
17. console.log('Login failed');
18. });
19. }
20. });
21. </script>

The div defined the controller called “loginCtrl”, and the scope of the controller is within the div which id=login. The “loginCtrl” contains the model UserInfo {UserName, Password}, it also has a method called Login, which functionality is to POST form.

With all the AngularJS technology we researched, we pictureing our real estate website as a powerful, clean, and fast single page web application. All HTML content should render into a big html file. And all Javascript logic should also render into a big JS file. When we explore the resources of our website, we should expect the whole website content is unified into one file. And after the first time pre-load, everything should be stored and content swapping should be efficient and quick. We gained hands-on experience from this project, and this will benefits us all along our career life.

**Bibliography**:

[1]A. Smith and A. Smith, "Record shares of Americans now own smartphones, have home broadband", *Pew Research Center*, 2017. [Online]. Available: http://www.pewresearch.org/fact-tank/2017/01/12/evolution-of-technology/.

[2]J. Titcomb, "Mobile web usage overtakes desktop for first time", *The Telegraph*, 2017. [Online]. Available: http://www.telegraph.co.uk/technology/2016/11/01/mobile-web-usage-overtakes-desktop-for-first-time/.

[3]S. Gibbs, "Mobile web browsing overtakes desktop for the first time", *the Guardian*, 2017. [Online]. Available: https://www.theguardian.com/technology/2016/nov/02/mobile-web-browsing-desktop-smartphones-tablets.

[4]"Single-page Applications | Code School", *Code School*, 2017. [Online]. Available:<https://www.codeschool.com/beginners-guide-to-web-development/single-page-applications>.

[5]"What is AngularJS", *Tutorialsteacher.com*, 2017. [Online]. Available:<http://www.tutorialsteacher.com/angularjs/what-is-angularjs>.

[6]"Single Page Application using AngularJs Tutorial - Tests4Geeks", *Tests4Geeks*, 2017. [Online]. Available:<https://tests4geeks.com/single-page-application-using-angularjs-tutorial/>.

[7]2017. [Online]. Available: <https://www.linkedin.com/pulse/why-angularjs-preferred-single-page-applications-spas-rahul-sudeep/>.

[8]2017. Directives[Online]. Available: https://docs.angularjs.org/guide/directive#what-are-directives-

[9]2017. ngView[Online]. Available: https://docs.angularjs.org/api/ngRoute/directive/ngView

[10]2017. ngRoute[Online]. Available: https://docs.angularjs.org/api/ngRoute#example

[11]2017. UI-Route[Online]. Available: https://ui-router.github.io/ng1/tutorial/helloworld#live-demo