In this step, you will learn how to create a layout template and how to build an app that has multiple views by adding routing, using an Angular module called 'ngRoute'.

- When you now navigate to `app/index.html`, you are redirected to `app/index.html#/phones` and the phone list appears in the browser.
- When you click on a phone link the url changes to one specific to that phone and the stub of a phone detail page is displayed.

Workspace Reset Instructions ➤

The most important changes are listed below. You can see the full diff on GitHub

---

Dependencies

The routing functionality added by this step is provided by angular in the `ngRoute` module, which is distributed separately from the core Angular framework.

We are using Bower to install client side dependencies. This step updates the `bower.json` configuration file to include the new dependency:

```
{
"name": "angular-phonecat",
"description": "A starter project for AngularJS",
"version": "0.0.0",
"homepage": "https://github.com/angular/angular-phonecat",
"license": "MIT",
"private": true,
"dependencies": {
  "angular": "~1.3.0",
  "angular-mocks": "~1.3.0",
  "jquery": "2.1.1",
  "bootstrap": "~3.1.1",
  "angular-route": "~1.3.0"
}
}
```

The new dependency `"angular-route": "~1.3.0"` tells bower to install a version of the angular-route component that is compatible with version 1.3.x. We must tell bower to download and install this dependency.

If you have bower installed globally then you can run `bower install` but for this project we have preconfigured npm to run bower install for us:

```
npm install
```

Multiple Views, Routing and Layout Template

Our app is slowly growing and becoming more complex. Before step 7, the app provided our users with a single view (the list of all phones), and all of the template code was located in the `index.html` file. The next step in building the app is to add a view that will show detailed information about each of the devices in our list.

To add the detailed view, we could expand the `index.html` file to contain template code for both views, but that would get messy very quickly. Instead, we are going to turn the `index.html` template into what we call a "layout template". This is a template that is common for all views in our application. Other "partial templates" are then included into this layout template depending on the current "route" — the view that is currently displayed to the user.

Application routes in Angular are declared via the $routeProvider, which is the provider of the $route service. This service makes it easy to wire together controllers, view templates, and the current URL location in the browser. Using this feature we can implement deep linking, which lets us utilize the browser's history (back and forward navigation) and bookmarks.

A Note About DI, Injector and Providers

As you noticed, dependency injection (DI) is at the core of AngularJS, so it's important for you to understand a thing or two about how it works.

When the application bootstraps, Angular creates an injector that will be used to find and inject all of the services that are required by your app. The injector itself doesn't know anything about what `$http` or `$route` services do, in fact it doesn't even know about the existence of these services unless it is configured with proper module definitions.

The injector only carries out the following steps :

- load the module definition(s) that you specify in your app
- register all Providers defined in these module definitions
- when asked to do so, inject a specified function and any necessary dependencies (services) that it lazily instantiates via their Providers.

Providers are objects that provide (create) instances of services and expose configuration APIs that can be used to control the creation and runtime behavior of a service. In case of the `$route` service, the `$routeProvider` exposes APIs that allow you to define routes for your application.

> **Note:** Providers can only be injected into `config` functions. Thus you could not inject `$routeProvider` into `PhoneListCtrl`.

Angular modules solve the problem of removing global state from the application and provide a way of configuring the injector. As opposed to AMD or require.js modules, Angular modules don't try to solve the problem of script load ordering or lazy script fetching. These goals are totally independent and both module systems can live side by side and fulfill their goals.

To deepen your understanding of DI on Angular, see Understanding Dependency Injection.

Template

The `$route` service is usually used in conjunction with the ngView directive. The role of the `ngView` directive is to include the view template for the current route into the layout template. This makes it a perfect fit for our `index.html` template.

> **Note:** Starting with AngularJS version 1.2, `ngRoute` is in its own module and must be loaded by loading the additional `angular-route.js` file, which we download via Bower above.

`app/index.html`:

```
<!doctype html>
<html lang="en" ng-app="phonecatApp">
<head>
...
  <script src="bower_components/angular/angular.js"></script>
  <script src="bower_components/angular-route/angular-route.js"></script>
  <script src="js/app.js"></script>
  <script src="js/controllers.js"></script>
</head>
<body>

  <div ng-view></div>

</body>
</html>
```

We have added two new `<script>` tags in our index file to load up extra JavaScript files into our application:

- `angular-route.js` : defines the Angular `ngRoute` module, which provides us with routing.
- `app.js` : this file now holds the root module of our application.

Note that we removed most of the code in the `index.html` template and replaced it with a single line containing a div with the `ng-view` attribute. The code that we removed was placed into the `phone-list.html` template:

app/partials/phone-list.html :

```
<div class="container-fluid">
<div class="row">
  <div class="col-md-2">
    <!--Sidebar content-->

    Search: <input ng-model="query">
    Sort by:
    <select ng-model="orderProp">
      <option value="name">Alphabetical</option>
      <option value="age">Newest</option>
    </select>

  </div>
  <div class="col-md-10">
    <!--Body content-->

    <ul class="phones">
      <li ng-repeat="phone in phones | filter:query | orderBy:orderProp" class="thumbnail">
        <a href="#/phones/{{phone.id}}" class="thumb"><img ng-src="{{phone.imageUrl}}"></a>
        <a href="#/phones/{{phone.id}}">{{phone.name}}</a>
        <p>{{phone.snippet}}</p>
      </li>
    </ul>

  </div>
</div>
</div>
```

We also added a placeholder template for the phone details view:

app/partials/phone-detail.html :

```
TBD: detail view for <span>{{phoneId}}</span>
```

Note how we are using the `phoneId` expression which will be defined in the `PhoneDetailCtrl` controller.

The App Module

To improve the organization of the app, we are making use of Angular's `ngRoute` module and we've moved the controllers into their own module `phonecatControllers` (as shown below).

We added `angular-route.js` to `index.html` and created a new `phonecatControllers` module in `controllers.js`. That's not all we need to do to be able to use their code, however. We also have to add the modules as dependencies of our app. By listing these two modules as dependencies of `phonecatApp`, we can use the directives and services they provide.

`app/js/app.js`:

```
var phonecatApp = angular.module('phonecatApp', [
'ngRoute',
'phonecatControllers'
]);

...
```

Notice the second argument passed to `angular.module`, `['ngRoute', 'phonecatControllers']`. This array lists the modules that `phonecatApp` depends on.

```
...

phonecatApp.config(['$routeProvider',
  function($routeProvider) {
    $routeProvider.
      when('/phones', {
        templateUrl: 'partials/phone-list.html',
        controller: 'PhoneListCtrl'
      }).
      when('/phones/:phoneId', {
        templateUrl: 'partials/phone-detail.html',
        controller: 'PhoneDetailCtrl'
      }).
      otherwise({
        redirectTo: '/phones'
      });
  }]);
```

Using the `phonecatApp.config()` method, we request the `$routeProvider` to be injected into our config function and use the `$routeProvider.when()` method to define our routes.

Our application routes are defined as follows:

- `when('/phones')`: The phone list view will be shown when the URL hash fragment is `/phones`. To construct this view, Angular will use the `phone-list.html` template and the `PhoneListCtrl` controller.
- `when('/phones/:phoneId')`: The phone details view will be shown when the URL hash fragment matches '/phones/:phoneId', where `:phoneId` is a variable part of the URL. To construct the phone details view, Angular will use the `phone-detail.html` template and the `PhoneDetailCtrl` controller.
- `otherwise({redirectTo: '/phones'})`: triggers a redirection to `/phones` when the browser address doesn't match either of our routes.

We reused the `PhoneListCtrl` controller that we constructed in previous steps and we added a new, empty `PhoneDetailCtrl` controller to the `app/js/controllers.js` file for the phone details view.

Note the use of the `:phoneId` parameter in the second route declaration. The `$route` service uses the route declaration — `'/phones/:phoneId'` — as a template that is matched against the current URL. All variables defined with the `:` notation are extracted into the `$routeParams` object.

Controllers

`app/js/controllers.js` **:**

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

phonecatControllers.controller('PhoneListCtrl', ['$scope', '$http',
  function ($scope, $http) {
    $http.get('phones/phones.json').success(function(data) {
      $scope.phones = data;
    });

    $scope.orderProp = 'age';
  }]);

phonecatControllers.controller('PhoneDetailCtrl', ['$scope', '$routeParams',
  function($scope, $routeParams) {
    $scope.phoneId = $routeParams.phoneId;
  }]);
```

Again, note that we created a new module called `phonecatControllers`. For small AngularJS applications, it's common to create just one module for all of your controllers if there are just a few. As your application grows it is quite common to refactor your code into additional modules. For larger apps, you will probably want to create separate modules for each major feature of your app.

Because our example app is relatively small, we'll just add all of our controllers to the `phonecatControllers` module.

Test

To automatically verify that everything is wired properly, we wrote end-to-end tests that navigate to various URLs and verify that the correct view was rendered.

```
...
 it('should redirect index.html to index.html#/phones', function() {
  browser.get('app/index.html');
  browser.getLocationAbsUrl().then(function(url) {
      expect(url.split('#')[1]).toBe('/phones');
    });
});

describe('Phone list view', function() {
  beforeEach(function() {
    browser.get('app/index.html#/phones');
  });
...

describe('Phone detail view', function() {

  beforeEach(function() {
    browser.get('app/index.html#/phones/nexus-s');
  });


  it('should display placeholder page with phoneId', function() {
    expect(element(by.binding('phoneId')).getText()).toBe('nexus-s');
  });
});
```

You can now rerun `npm run protractor` to see the tests run.

Experiments

Try to add an `{{orderProp}}` binding to `index.html`, and you'll see that nothing happens even when you are in the phone list view. This is because the `orderProp` model is visible only in the scope managed by `PhoneListCtrl`, which is associated with the `<div ng-view>` element. If you add the same binding into the `phone-list.html` template, the binding will work as expected.

Summary

With the routing set up and the phone list view implemented, we're ready to go to step 8 to implement the phone details view.

Previous   Live Demo   Code Diff   Next