FREE eBook

LEARNING angular-ui-router

Free unaffiliated eBook created from
Stack Overflow contributors.

#angular-ui-router

Table of Contents

About

You can share this PDF with anyone you feel could benefit from it, downloaded the latest version from: angular-ui-router

It is an unofficial and free angular-ui-router ebook created for educational purposes. All the content is extracted from Stack Overflow Documentation, which is written by many hardworking individuals at Stack Overflow. It is neither affiliated with Stack Overflow nor official angular-ui-router.

The content is released under Creative Commons BY-SA, and the list of contributors to each chapter are provided in the credits section at the end of this book. Images may be copyright of their respective owners unless otherwise specified. All trademarks and registered trademarks are the property of their respective company owners.

Use the content presented in this book at your own risk; it is not guaranteed to be correct nor accurate, please send your feedback and corrections to info@zzzprojects.com

Chapter 1: Getting started with angular-ui-router

Remarks

Angular UI-Router is a client-side Single Page Application routing framework for AngularJS.

Routing frameworks for SPAs update the browser's URL as the user navigates through the app. Conversely, this allows changes to the browser's URL to drive navigation through the app, thus allowing the user to create a bookmark to a location deep within the SPA.

UI-Router applications are modeled as a hierarchical tree of states. UI-Router provides a state machine to manage the transitions between those application states in a transaction-like manner.

Taken from the UI-Router Github page

Versions

Version	Release Date
0.2.18	2016-02-07
0.2.17	2016-01-25
0.2.16	2016-01-24
0.2.15	2016-05-19
0.2.14	2016-04-23
0.2.13	2016-11-20
0.2.12	2016-11-13
0.2.11	2016-08-26
0.2.10	2016-03-12
0.2.9	2014-01-17
0.2.8	2014-01-16

Examples

Hello World Example

STEP 1: Installation

Before you can use Angular-UI Router you must include AngularJS itself in your project. For a detailed guide on that see this documentation.

You can download Angular-UI Router either from their GitHub-Page or from NuGet, NPM, Bower respectively.

After you have included the JS file in your webpage you can inject the `ui.router` module inside your application. In your script file you should have something like this:

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

and now we are going to inject Angular-UI Router into our own application like this:

```
var app = angular.module('app', ['ui.router']);
```

Now Angular-UI Router will be loaded with our application. The following steps will explain the basics behind Angular-UI Router and will show some of the basic functionality.

STEP 2: Defining simple states

You can configure the UI-Router inside the Angular `config` function. Use the `$stateProvider` to define your states. In the following example, each state has a url, controller and a template.

```
(function() {
  var app = angular.module('app', ['ui.router']);

  app.config(['$stateProvider', function($stateProvider) {
      $stateProvider
        .state('home', {
          url: "/home",
          templateUrl: "home.html",
          controller: "homeCtrl"
        })
        .state('kitchen', {
          url: "/kitchen",
          templateUrl: "kitchen.html",
          controller: "kitchenCtrl"
        })
        .state('den', {
          url: "/denCtrl",
          templateUrl: "den.html",
          controller: "denCtrl"
        })
        .state('itemDetail', {
          url: "/items/:itemName",
          templateUrl: "item.html",
          controller: "itemDetailCtrl"
        })

    }])
})();
```

in your HTML, you will need the `ui-view` directive so that the state views can be populated inside.

```
<div ui-view></div>
```

STEP 3: Accessing states

There are all together 3 ways to access a state that is defined in `$stateProvider`.

1. Via `ui-sref` directive

You can access states inside your HTML, by using the `ui-sref` directive

```
<li ui-sref-active="active">
    <a ui-sref="kitchen">Go to the Kitchen</a>
</li>
<li ui-sref-active="active">
    <a ui-sref="den">Enter the den</a>
</li>
<li ui-sref-active="active">
    <a ui-sref="itemDetail({itemName:'keyboard'})">Key Board</a>
</li>
```

2. Via `$state` service in the controller

you can also navigate to other states inside your controller by using the `$state` provided to the controller with the `.go` method.

```
.controller(function($scope, $state) {
    // ...
    $scope.navigateTo = function(stateName) {
        $state.go(stateName); // i.e. $state.go('den');
    };
})
```

3. Via the url in browser

Assuming you have a state called `kitchen` defined like this:

```
$stateProvider
  .state("kitchen", {
    url: "/kitchenUrl",
    ...
  });
```

Then accessing appdomain/kitchenUrl as the URL in your browser will go to your `kitchen` state, assuming that there are no nested states and `appdomain` is the server that hosts your application.

If you are still confused, here is a fully working Plnkr

Basic View

index.html

```
<html>
    <head>
        <title>Angular-UI Router Example</title>
        <script type="text/javascript"
src="https://cdnjs.cloudflare.com/ajax/libs/angular.js/1.4.9/angular.js"></script>
        <script type="text/javascript" src="https://cdnjs.cloudflare.com/ajax/libs/angular-ui-
router/0.3.1/angular-ui-router.js"></script>
        <script type="text/javascript" src="../js/script.js"></script>
    </head>
    <body ui-view="mainView"> <!-- Defining a container for our view -->
    </body>
</html>
```

script.js

```
var app = angular.module('app', ['ui.router']);
app.config(['$stateProvider', function($stateProvider){
    $stateProvider.state('home', {                      // Creating a state called 'home'
        url: '',                                        // An empty URL means that this
state will be loaded on the main URL when no other state is called
        views: {
            'mainView': {                               // Section for our view-container
that we defined in the HTML
                template: '<h1>It works!</h1>'          // Setting a template for this view
                /*templateUrl: '../templates/home.html'*/ //templateUrl would load the file
and uses it's content as the template
            }
        }
    });
}])
```

Defining a state with multiple view

In ui-router a state can hold multiple views, each with his own controller and a template

```
.state('dashboard', {
     name: 'dashboard',
     url: '/dashboard',
     views: {
         "view1": {
             templateUrl: "path/to/view1.html",
             controller: "view1Controller"
         },
         "view2": {
             templateUrl: "path/to/view2.html",
             controller: "view2Controller"
         }
     }
 })
```

Then inside your state's HTML, you can link these views

```
<div ui-view="view1"></div>
<div ui-view="view2"></div>
```

Resolving data into a state

You can `resolve` data into your state when you transition into it, usually it's useful when the state needs to use that data, or to resolve into a state when some provided input needs to be authenticated.

When you define your states, you will need to provide a map of values to be resolved into the `.resolve` property, each resolved value should have a function that returns a `promise`

```
.state('main', {
     url: "/main",
     templateUrl: "path/to/main.html",
     controller: 'mainCtrl',
     resolve: {
         serverData: function ($http) {
             return $http.get('some/url');
         }
     }
});
```

Now, inside the `mainCtrl` you can access the data (that is if the `$http` call resolved successfully).

```
.controller("mainCtrl", function($scope, serverData) {
    $scope.resolvedData = serverData.then(resp=> resp.data);
    ....
})
```

Using transition events

UI-Router exposes transition events that can be helpful for handling transition errors, handling/blocking transitions based on certain parameter values, custom authentication etc..

These events can be bound to `$rootScope` for a global effect or to `$scope` for a per controller effect.

`$stateChangeError` - This event is broadcasted when an attempt to change the state has failed and threw and error, this event fires a callback function with the following signature:

    callback(event, toState, toParams, fromState, fromParams, error)

event: the event object

toState: the target state

toParams: the parameters passed to the target state

fromState: current state

fromParams: the parameters passed to the current state

error: the error object

`$stateChangeStart` - This event is broadcasted when a state transition started, this event fires a callback function with the following signature:

    callback(event, toState, toParams, fromState, fromParams, options)

options: the state options object

`$stateChangeSuccess` - This event is broadcasted when a state transition completes, this event fires a callback function with the following signature:

    callback(event, toState, toParams, fromState, fromParams, options)

`$stateNotFound` - This event is broadcasted when a state you requested to transition to was not found, this event fires a callback function with the following signature:

    callback(event, unfoundState, fromParams, fromState)

unfoundState - an object representing the state that was not found

---

Example:

```
$rootScope.$on('$stateChangeSuccess', function (event, toState, toParams, fromState,
fromParams, options) {
    $log.debug("$stateChangeSuccess: event: %o toState: %o, toParams: %o, fromState: %o,
fromParams: %o, options: %o", event, toState, toParams, fromState, fromParams, options);
    // runs when the state has successfully changed
});

$rootScope.$on('$stateChangeStart', function (event, toState, toParams, fromState, fromParams,
options) {
    $log.debug("$stateChangeStart: event: %o toState: %o, toParams: %o, fromState: %o,
fromParams: %o, options: %o", event, toState, toParams, fromState, fromParams, options);
    // runs when the state has just started to transition
});

$rootScope.$on('$stateNotFound', function (event, unfoundState, fromParams, fromState) {
    $log.debug("$stateNotFound: event: %o unfoundState: %o, fromParams: %o, fromState: %o",
event, unfoundState, fromParams, fromState);
    // runs when the state wsa not found
});

$rootScope.$on('$stateChangeError', function (event, toState, toParams, fromState, fromParams,
error) {
    $log.debug("$stateChangeError: event: %o toState: %o, toParams: %o, fromState: %o,
fromParams: %o, error: %o", event, toState, toParams, fromState, fromParams, error);
    // runs when there was an error while attempting to transition
});
```

Read Getting started with angular-ui-router online: https://riptutorial.com/angular-ui-router/topic/1869/getting-started-with-angular-ui-router

Chapter 2: Custom parameter types

Parameters

Parameter	Details
`decode`	Converts URL value (string) to the value available in `$stateParams`
`encode`	Converts a value to the string that will be used in the URL
`equals`	Verifies if two values are equal from the type's point of view
`is`	Checks if the value can be used as defined parameter type
`pattern`	Ensures that the values from URL matches this pattern when route resolves

Examples

Page number parameter

Similar to `int` but accepts only positive integers (useful for pagination when there is a `page` parameter.

Define:

```
module.config(['$urlMatcherFactoryProvider', function($urlMatcherFactory) {
  $urlMatcherFactory.type('page', {
    decode: function(val) { return +val; },
    encode: function(val) { return Math.floor(val); },
    equals: function(a, b) { return this.is(a) && +a == +b; },
    is: function(val) { return angular.isNumber(val) && val >= 1; },
    pattern: /\d+/
  })
}]);
```

And use:

```
$stateProvider.state({
  url: '/my-route/{page:page}'
  template: '<my-page></my-page>'
});
```

Plunker and related SO answer.

Boolean parameter

Define:

```
module.config(['$urlMatcherFactoryProvider', function($urlMatcherFactory) {
  $urlMatcherFactory.type('boolean', {
    decode: function(val) { return val == true || val == "true" },
    encode: function(val) { return val ? 1 : 0; },
    equals: function(a, b) { return this.is(a) && a == b; },
    is: function(val) { return [true, false, 0, 1].indexOf(val) >= 0 },
    pattern: /false|true|0|1/
  })
}]);
```

And use:

```
$stateProvider.state({
  url: '/my-route/{showSidebar:boolean}'
  template: '<my-page></my-page>'
});
```

Plunker and related SO answer.

Path parameter (with not-encoded slash inside)

By default, ui-router encodes the slash `/` inside parameters. If you want to send a path in the URL, you need to define a custom parameter type.

Define:

```
module.config(['$urlMatcherFactoryProvider', function($urlMatcherFactory) {
  $urlMatcherFactory.type('path', {
    decode: function(val) { return val != null ? val.toString() : val; },
    encode: function(val) { return val != null ? val.toString() : val; },
    is: function(val) { return this.pattern.test(val); },
    pattern: /[^/]+\/[^/]+/
  })
}]);
```

And use:

```
$stateProvider.state({
  url: '/my-route/{directory:path}'
  template: '<my-page></my-page>'
});
```

Related question.

Read Custom parameter types online: https://riptutorial.com/angular-ui-router/topic/2031/custom-parameter-types

Chapter 3: State transition

Examples

Reload current state

You can reload the current state using the `$state.reload` method from your controller

```
$state.reload()
```

This is a shorthand for (code taken from the official docs)

```
$state.transitionTo($state.current, $stateParams, {
  reload: true, inherit: false, notify: false
});
```

Running a reload on your state will also restart your controller/s and re-resolve all your resolved values.

Use $state.go to transition between states

`$state.go` is shorthand method to `$state.transitionTo`

    $state.go(toState [, toParams] [, options])

This method automatically sets your options to `{ location: true, inherit: true, relative: $state.$current, notify: true }` (unless you override them) and allows you to transition with less code.

Examples:

Lets say we have an app with a 'main' state, with 2 child states: 'dashboard' and 'help', and 'dashboard' also has a child called 'about'.

Transition to another state

```
$state.go("main.dashboard") // from anywhere to 'main.dashboard'
```

Transition to parent state

```
$state.go("^") // from 'main.dashboard' to 'main'
```

You can also transit to another child of the parent state (sibling)

```
$state.go("^.help") // from 'main.dashboard' to main.help
```

Placing a `.` will allow you to transition to child states

```
$state.go(".about") // from 'main.dashboard' to 'main.dashboard.about'
```

Use $state.transitionTo to transition between states

Use `$state.transitionTo` to go form one state to another. This is a low level method to transition and `$state.go` is the recommended way for most common use cases as it uses this method internally.

    $state.transitionTo(toState [, toParams] [, options])

toState - the state to transition to

toParams (optional) - a map of parameters to send the target state

options (optional) - the state transition options

Examples:

```
$state.transitionTo("dashboard.history", {period: "week"})
// transitions to the history child state with a state parameter
```

Read State transition online: https://riptutorial.com/angular-ui-router/topic/3608/state-transition

Credits

S. No	Chapters	Contributors
1	Getting started with angular-ui-router	Ajeet Lakhani, Community, CozyAzure, Gaara, svarog, ThermalCube
2	Custom parameter types	fracz, lujcon
3	State transition	Matt Tester, svarog