Routing in Angular1 - ui.router

We can look at our angular app as a state machine, where each state in the application represents the current UI I see in front of. Usually on web application a certain UI is refered from a certain URL. For example the URL **/login/** can refer to the login page, and also you expect that even if you reload the page, or type the URL of the login page you will expect to enter the same login page everytime. So actually we might say that a state is a web application is the current UI I see and the given URL of that UI. Angular has a routing package called **ui.router** which helps us turn our web application into a state machine where each state responds to a URL and presents us with a UI. With Angular routing we can navigate from page to page by going from state to state, we can load controllers and templates into each state, we can also deal with **promises** and server calls that each controller requires and much more. It's a must have package in every web application that requires routing. In this tutorial we will the most common features you use with the **ui.router** package.

Bootstrap your angular app

first we will start a new angular project, create the root directory of your project and cd into that directory and in your terminal type:
> bower init
> bower install angular --save

The first command will create the **bower.json** file and the second one will install angular and place it inside the **bower_components** directory and also place the package version in the **bower.json** file. In case you want to read more about **bower** you can follow our tutorial in this link

Create an **index.html** file with the following HTML:

```
<!DOCTYPE html>
<html ng-app="AppModule">
<head>
  <meta charset="UTF-8">
  <title>ui.router</title>
</head>
<body>
  <script src="bower_components/angular/angular.js"></script>
  <script src="src/app.module.js"></script>
</body>
</html>
```

We are simply loading **angular** script and our **app.module.js** script.

Create a folder called **src** and place the file **app.module.js** in that folder with the following code:

```
angular.module('AppModule', []);
```

That's the general bootstrap of our app, let's continue with installing our **ui.router** package.

Install ui.router

using bower let's search for our package. In the root directory of your project type the following:
> bower search ui-router

you should see the package listed at the top so let's install our package:
> bower install ui-router --save

and in your **index.html** load the script after the angular script by placing this line:
<script src="bower_components/angular-ui-router/release/angular-ui-router.js"></script>

Now in your **app.module.js** change to look like this:

```
angular.module('AppModule', ['ui.router']);
```

we loaded the **ui.router** script after the angular script loads and also added dependency in our **root module** for the **ui.router** module.

ui-view

ui-view is a directive that tells the router where to place the templates.
ui-view can contain a name if you are using multiple view. make sure there is only a single **ui-view** in a template without a name.
We will place the **ui-view** directive in our **index.html** file. In the **index.html** right after the opening of the body place the following:
**<div ui-view> **

homepage

for now let's create our app with only the root URL and it will map to the homepage. first let's create the homepage controller and template.

in the **src** folder create another folder called **controllers** and create the file **home.controller.js** with the following code.

```
angular.module('AppModule')
  .controller('HomeController', function(){
     var self = this;

     self.greeting = 'Hello Home Page';
  });
```

in your **src** folder, create another folder called **templates** and create the file **home.template.html** with the following code:

```
<h1>
  {{ctrl.greeting}}
</h1>
```

we created a controller with a **greeting** property and we are printing that property on the template.

Load the script of the controller in the **index.html** right after the **module** script is loaded:
`<script src="src/controllers/home.controller.js"></script>`

Now let's config our app to contain the route for the homepage.

ui.router config

Configuring the router is done in the configuration state of our app, and it's done through a provider called **$stateProvider**.
in the **src** folder create a new file called **app.routes.js** and place the following code in that file**:**

```
angular.module('AppModule')
  .config(['$stateProvider', '$urlRouterProvider', '$locationProvider',
function($stateProvider, $urlRouterProvider, $locationProvider){
     $urlRouterProvider.otherwise('/');
     $locationProvider.html5Mode(true);
     $stateProvider.state('home', {
        controller: 'HomeController as ctrl',
        templateUrl: 'src/templates/home.template.html',
        url: '/'
     });
```

```
}]);
```

things to notice here:

- **$urlRouterProvider -** watched for **$location** changes and is working behind the scenes on our **ui.router** so you can use this service to instruct what to do when the current **url** route has no match in our routing. in this case we are directing the user to the homepage if route not found.
- **$locationProvider** allows us to config the **$location** service which is angular wrapper around the **location** object which takes care of the browser url location. We are configuring the **$location** to use HTML5 routes.
- **$stateProvider** - we are using the **$stateProvider** to configure the states of our app, we are calling the **state** method which get's as first argument the name of the state, in our case **home,** and as second argument the state config object.
- state config object - the state config object can have many properties but for now we will explore the 3 basic properties most commonly used:
    - **url -** the url of the state
    - **template/templateUrl -** we can enter the template directly with the attribute **template** or we can reference to the **templateUrl** which is a string of the url to load the template.
    - **controller -** If our controller is loaded we can just place the name of the controller or most recommended is to use to **controllerAs** syntax like we did which allows us to place our properties in the controller instance. Not recommended but you can also define your controller as a function to this attribute.

our router is now configured but for the router to work with **HTML5** mode we need to place the **base** tag in the **head** so the routing will know to calculate the route from this base set.

So place the following line in the **index.html head** section:

```
<base href="/" />
```

we also need to load the script of the **app.routes.js** in our **index.html** file so add the script loading right after the loading of our **module** script.

```
<script src="src/app.routes.js"></script>
```

launch your app and you should see the greeting of the homepage.

Let's explore the most common usage of the **$stateProvider.**

URL Params

let's add another page to our app, and this page has the following url:
/pokemon/<id>/?name=<name of pokemon>
where **<id>** is the primary key identifying our pokemon, and also we will pass get params describing the pokemon. The screen on this page will simply display the id that we got from the URL and all the other params.
To pass params for a page we need to inform in the **url** property of the **state config** that the url will have params.
Modify your **app.routes.js** file to look like this:

```
angular.module('AppModule')
  .config(['$stateProvider', '$urlRouterProvider', '$locationProvider',
function($stateProvider, $urlRouterProvider, $locationProvider){
    $urlRouterProvider.otherwise('/');
    $locationProvider.html5Mode(true);

    //homepage route
    $stateProvider.state('home', {
        controller: 'HomeController as ctrl',
        templateUrl: 'src/templates/home.template.html',
        url: '/'
    });

    //pokemon page
    $stateProvider.state('pokemon', {
        url: '/pokemon/:id/?name&power',
        controller: 'PokemonController as ctrl',
        templateUrl: 'src/templates/pokemon.template.html'
    });
  }]);
```

notice that we added a state called **pokemon** and in the state config object we placed a url with params syntax which is marked **:param** for dynamic url param and **?param1&param2…** for url get params.
We will now create the **PokemonController** so inside your **controllers** folder create a file called **pokemon.controller.js** with the following code:

```
angular.module('AppModule')
  .controller('PokemonController', ['$stateParams', function($stateParams){
    var self = this;
```

```
    self.id = $stateParams.id;
    self.name = $stateParams.name;
    self.power = $stateParams.power;
  }]);
```

notice that we inject the **$stateParams** to our controller which we can use to grab the params we sent. We are placing the params in the controller properties to present them in the template.

Let's create our template file, in the **templates** folder create a file called **pokemon.template.html** with the following code:

```
<h1>Hello pokemon page</h1>

<h4>id: {{ctrl.id}}</h4>
<h4>name: {{ctrl.name}}</h4>
<h4>power: {{ctrl.power}}</h4>
```

don't forget to include you **pokemon.controller.js** into your **index.html** file.
test your app and place in the browser the url:
/#/pokemon/3/?name=pikachu&power=10

you should see the data is printed in the template.

So we saw here how we can use the url to pass params and how to use those params in the controller.
We will probably want users in our app to navigate using link so let's create navigation for our app.

Navigating from state to state

ui-sref

we can use the **ui-sref** directive to navigate between our states.
the **ui-sref** binds a link tag (<a ...>) to a state.
We will now add a navigation to our app, so in our **index.html** file place the following code above the **<div ui-view> **:

```
<nav>
  <ul>
    <li>
      <a ui-sref="home" >Home</a>
```

```
    </li>
    <li>
        <a ui-sref="pokemon({id: 3, name: 'pikachu', power: 10})" >Pokemon page</a>
     </li>
  </ul>
</nav>
```

we can place in the **ui-sref** the name of the state (like the navigation link for the homepage) or we can specify the state name with the params to pass in the url (like the link to the pokemon page)

Launch your app and make sure you can navigate between the pages.

$state.go

you can also transfer to different state using code and not template directive. To do this you can inject to a controller the **$state** service and call **$state.go** to iterate between the states.
We will now demonstrate this in our app and we will place a button in the **pokemon** page and on the **homepage** that directs to the other page.

in the **home.template.html** place the following code:

```
<h1>
  {{ctrl.greeting}}
</h1>

<button ng-click="ctrl.gotoPokemon();">
  Go to Pokemon page
</button>
```

we added a button with a click event so let's check the implementation of the click event in the **home.controller.js**

```
angular.module('AppModule')
  .controller('HomeController', ['$state', function($state){
      var self = this;

      self.greeting = 'Hello Home Page';

      self.gotoPokemon = function gotoPokemon(){
          $state.go('pokemon', {id: 3, name: 'pikachu', power: 10});
      }
  }]);
```

notice that we injected the **$state** service and we are using it in the button function **gotoPokemon** where we are calling **$state.go** where the first argument is the name of the state we want to transition to, and the second argument are the params we pass in the url.

we are going to do a similar thing in the template and the controller of the pokemon page, in your **pokemon.template.html** change your code to look like this:

```
<h1>Hello pokemon page</h1>

<h4>id: {{ctrl.id}}</h4>
<h4>name: {{ctrl.name}}</h4>
<h4>power: {{ctrl.power}}</h4>

<button ng-click="ctrl.gotoHomepage();">
  Go to Home page
</button>
```

and in the **pokemon.controller.js** place the following code:

```
angular.module('AppModule')
  .controller('PokemonController', ['$stateParams', '$state', function($stateParams,
$state){
    var self = this;

    self.id = $stateParams.id;
    self.name = $stateParams.name;
    self.power = $stateParams.power;

    self.gotoHomepage = function gotoHomepage(){
      $state.go('home');
    }
  }]);
```

launch your app and you should see other than the navigation a button on each page the redirects to the other page.

So we saw how we can add navigation to our page through code and links, what if we want our controller to load only after some promise is resolved, for example let's say we have a todo tasks page and the page should load only after we get all the todo tasks from the server.
We will now cover how **ui.router** let's us easily accomplish this task.

resolve

resolve is one of the keys in the **state config** object, and this key get's an object with the keys as names that will be injected in the controller and the value of those names will be an injectable function which can return sync or async data to the controller. If we are passing **async** data via **promise** the controller and the page won't be loaded until the promise is resolved. The **resolve** is very useful for the cases where you want a page to present data from a server and not load that page until the data is returned. Let's try to demonstrate the usage of **resolve** by create a tasks page which will load todo tasks from the server at the url:
https://nztodo.herokuapp.com/api/task/?format=json
to achieve this we will use the **$http** service, if you want to learn more about **$http** service you can read about it in our tutorial about server communication on angular in the following link

In your **app.routes.js** change it to look like this:

```
angular.module('AppModule')
  .config(['$stateProvider', '$urlRouterProvider', '$locationProvider',
function($stateProvider, $urlRouterProvider, $locationProvider){
    $urlRouterProvider.otherwise('/');
    $locationProvider.html5Mode(true);

    //homepage route
    $stateProvider.state('home', {
        controller: 'HomeController as ctrl',
        templateUrl: 'src/templates/home.template.html',
        url: '/'
    });

    //pokemon page
    $stateProvider.state('pokemon', {
        url: '/pokemon/:id/?name&power',
        controller: 'PokemonController as ctrl',
        templateUrl: 'src/templates/pokemon.template.html'
    });

    //tasks page
    $stateProvider.state('tasks', {
        url: '/tasks/',
        controller: 'TasksController as ctrl',
        templateUrl: 'src/templates/tasks.template.html',
        resolve: {
            tasks: ['$http',function($http){
```

```
            return $http.get('https://nztodo.herokuapp.com/api/task/?format=json');
          }]
        }
      });
  }]);
```

we added a **tasks** state with the controller **TasksController** and template **tasks.template.html,** and this state belongs to the url **/tasks** .
We also added a **resolve** to our **state** that returns a promise by using **$http.get** and querying our **tasks** rest api.
We will now add the **TasksController** and use the data from the resolved promise.
Create the file **tasks.controller.js** in the **controllers** folder with the following code.

```
angular.module('AppModule')
  .controller('TasksController', ['tasks', function(tasks){
    var self = this;
    self.tasks = tasks.data;
  }]);
```

You can see that we can now inject **tasks** from the resolve in the **state config** object to our controller by using the name we chose in the resolve object.
The tasks will get a fulfilled promise with the data from the server so we can place that data as a controller property and print it in the template of the controller.
Don't forget to include this script to load in the **index.html** file.

now create a file called **tasks.template.html** inside the **templates** folder with the following code:

```
<h1>
  Tasks
</h1>
<ul>
  <li ng-repeat="task in ctrl.tasks">
    {{task.title}}
  </li>
</ul>
```

the template simply iterates on **ctrl.tasks** and prints the task title. You can also place a link in the navigation in the **index.html** that will refer to the tasks page.
Now launch your app and check how your tasks controller get the promise resolved. Your code is much cleaner now since you don't have to deal with promises and you can assume your controller get the resolved promise.

We referred to the **ui.router** as creating a state tree but for now we only saw simple pages and controller and we didn't really dealt with an actual tree of routes and parents and controllers relationship, so the next topic we will cover how we can nest our application routes.

Nesting states

In our app we usually create a tree of states, where we have an hierarchy of parents and children. This hierarchy is also reflected in our URL, for example in the current app we built we have a page representing our tasks page in the following url: **/tasks** let's mark this state as a parent state for the page representing a single task, so we will now create the URL **/tasks/<id>** which will represent a single task from the tasks list and will be the child of our tasks page.

In our **app.routes.js** add the following state before the closing of the **config** block.

```
//single task page
$stateProvider.state('tasks.singleTask', {
  url: ':id/',
  controller: 'SingleTaskController as ctrl',
  templateUrl: 'src/templates/single-task.template.html'
});
```

In the code above we are telling the **ui.router** that we want to create a child for the **tasks** state by writing the name in the syntax: **tasks.<name of child>** .
Also note that the **url** key is relative to the url of the parent, which means that for the state **tasks.singleTask** to enter the state the URL should be: **/tasks/<id>/**
It's important to note also that the child state will also inherit the resolve keys from the parent, meaning that also the child will have the tasks from the server.

Create the controller for the **singleTask** state and place the file **single-task.controller.js** in your **controllers** folder, with the following code:

```
angular.module('AppModule')
  .controller('SingleTaskController', ['tasks', '$stateParams', function(tasks,
$stateParams){
    var self = this;

    for(var i=0; i<tasks.data.length; i++){
      if(tasks.data[i].id == $stateParams.id){
        self.task = tasks.data[i];
      }
    }
  }]);
```

The **SingleTaskController** simply iterates over the list of tasks and finds the proper task by comparing the **id** of the task with the **id** we are getting from the url. we are placing the task we found as a property in our controller to be displayed later in the template.
Also make sure to include this **js** file in the **index.html**.

Let's create our template file so in the **templates** directory create a file called **single-task.template.html** with the following code:

```
<h1>
  Single Task
</h1>

<h4>
  id: {{ctrl.task.id}}
</h4>

<h4>
  title: {{ctrl.task.title}}
</h4>

<h4>
  description: {{ctrl.task.description}}
</h4>

<h4>
  group: {{ctrl.task.group}}
</h4>

<h4>
  when: {{ctrl.task.when}}
</h4>
```

the template simply prints the properties of the task we bind in the controller.

To iterate to a single task page we will add links in the **tasks.template.html** so add the following change in that file:

```
<h1>
  Tasks
</h1>
<ui-view></ui-view>
<ul>
  <li ng-repeat="task in ctrl.tasks">
```

```
    <a ui-sref="tasks.singleTask({id: task.id})" >{{task.title}}</a>
  </li>
</ul>
```

we simply change our list of tasks title to also link to the child page.
we also placed another **ui-view** directive since the child template will search for the parent for a **ui-view** directive to inject to.

Launch your app and try to click a single task in the list of tasks.
you will notice that you can now access a single task by clicking an item in the task list and it will direct you to a single task page, the only problem here is that the parent still presents us with all the tasks list, so to improve our app let's try and create a single task page which doesn't present the entire tasks list. To achieve this we will use **abstract** parent in our **ui.router** state tree.

Abstract Parent

we saw that if we actually put the **singleTask** state as a child of **tasks** then on the **singleTask** page we will see also a list of all the tasks, which is something we don't want, we want our **singleTask** state to just present us a single task. We will now create a mutual parent for the **tasks** and **singleTasks** state, the parent will have the responsibility of loading the tasks from the server, but the parent state can't be directly active and can only be active from one of it's children, we refer to this kind of state as an abstract state.

In our **app.routes.js** change the routes for **tasks** and **singleTask** to look like this:

```
//tasks parent
$stateProvider.state('tasks', {
  url: '/tasks',
  template: '<ui-view></ui-view>',
  abstract: true,
  resolve: {
    tasks: ['$http',function($http){
        return $http.get('https://nztodo.herokuapp.com/api/task/?format=json');
    }]
  }
});

//tasks list
$stateProvider.state('tasks.list', {
  url: '/',
  controller: 'TasksController as ctrl',
  templateUrl: 'src/templates/tasks.template.html'
```

```
});

//single task page
$stateProvider.state('tasks.singleTask', {
  url: '/:id/',
  controller: 'SingleTaskController as ctrl',
  templateUrl: 'src/templates/single-task.template.html'
});
```

notice that the **tasks** state is now an abstract state with no controller and a simple template with **ui-view** directive to inject the children into. We also placed the tasks resolve in the parent so it will be available for all it's children.
We then defined the two states we had before, one for displaying the tasks list in the state named: **tasks.list** and another which displays a single task in a state named **singleTask.**

Change your **tasks.template.html** and remove the **ui-view** directive since it's no longer a parent state.

Also change the navigation link in the **index.html** to direct to the tasks list page :

```
<a ui-sref="tasks.list">Tasks</a>
```

you can now launch your app and try to get into the list of tasks and into a single task, note that in a single task page we no longer see the entire list like before.

reloadOnSearch

The last thing we will cover about routing is called **reloadOnSearch**, whenever a state we are in will get new params then the state will be reloaded, we often won't want our state to reloaded when there is new params and the best example for that is in a search page.
We will know create in our tasks list page a text input to search the tasks list, user will be able to type a text and see all the tasks that match his search. Also we would like the state of the search to be saved in the url, meaning that if we search for the term **pokemon** we would expect our URL to look like: **/tasks/?q=pokemon** and even if our state reloads it should return exactly the same results every time. Let's try to achieve this in our current app. In your **app.routes.js** file, add the following change to our **tasks.list** state:

```
//tasks list
$stateProvider.state('tasks.list', {
  url: '/?q',
  controller: 'TasksController as ctrl',
  templateUrl: 'src/templates/tasks.template.html'
```

```
});
```

we simply added a url param called **q** to hold our search query.

Now in our **tasks.template.html** add the following to add a search input:

```
<h1>
  Tasks
</h1>

<h3>
  Search
</h3>
<input type="text" ng-model="ctrl.searchString" ng-change="ctrl.searchChanged();">
<br/>

<ul>
  <li ng-repeat="task in ctrl.tasks | filter:ctrl.searchString">
    <a ui-sref="tasks.singleTask({id: task.id})" >{{task.title}}</a>
  </li>
</ul>
```

We simply added a search section with an input, we are binding that input with the **ng-model** directive to the controller property **searchString,** we are also adding an **ng-change** event to modify the url when we type.
Also we are filtering the **tasks** list by using the **searchString** this will create the wanted search filtering on our list.

In the controller file **tasks.controller.js** change the controller to look like this:

```
angular.module('AppModule')
  .controller('TasksController', ['tasks', '$state', '$stateParams', function(tasks, $state,
$stateParams){
    var self = this;
    self.tasks = tasks.data;

    //init the search string from url params
    self.searchString = $stateParams.q;

    self.searchChanged = function searchChanged(){
        $state.go('tasks.list', {q: self.searchString});
    }
  }]);
```

notice that we are initiating the **searchString** from the **q** param in the URL, we also placed the **ng-change** function **searchChanged** to update the url when we type the search string.

notice that when you launch your app and type in the search box the app always reloads the tasks state and the result of this is a very clumsy search which always reloads the entire search and we lose the focus from the search box. Now to disable the reload when the params change you can add the property **reloadOnSearch** to false in the state config object like so:

```
//tasks list
$stateProvider.state('tasks.list', {
  url: '/?q',
  controller: 'TasksController as ctrl',
  templateUrl: 'src/templates/tasks.template.html',
  reloadOnSearch: false
});
```

try to launch your app again and the clumsiness of the search should be gone now.
The **reloadOnSearch** prevents the reload of the state when get params are changed.

Student Exercise

we will now add routing to our tasks application

- install the **ui.router** module and add it to our **index.html** and module
- our app will create two routes, one for the tasks list and one for a single task
- the tasks route will be in the the url **/tasks?q** this will be the default route of the application
- the url of the single task page is **/tasks/:id**
- both of the routes share the same abstract parent with resolve that gets all the tasks
- your task list page will contain a search input box that will add the search to the **q** param
- when you reload make sure the list is displayed partially.
- make sure that on a single task we see the entire task

Summary

there are more things to learn regarding the **ui.router** but we did cover in this tutorial the major and most usable features of this package. The **ui.router** is necessary for almost every angular app we are going to build.
Happy Coding!