CS 454 – AngularJS & Node.js

CRUD APIs and App Structure

Albert Cervantes
Cydney Auman

AngularJS $resource

- Most Single Page Applications involve CRUD operations.
- In AngularJS you can leverage the power of the $resource service.
  - Built on the top of the $http service
  - Factory that lets you interact with RESTful backends easily.
- Not included by default!
  - Must include angular-resource.js

AngularJS $resource

- Your main app module should declare a dependency on the ngResource module in order to use $resource.
- Ex.
  - angular.module('cs454App',['ngResource']);

API Design

$resource expects a classic RESTful backend. This means you should have REST endpoints in the following format:

URL	HTTP Verb	POST Body	Result
http://cs454.yourdomain.com/api/issues	GET	empty	Returns all issues
http://cs454.yourdomain.com/api/issues	POST	JSON String	New issue created
http://cs454.yourdomain.com/api/issues/:id	GET	empty	Returns a single issue
http://cs454.yourdomain.com/api/issues/:id	PUT	JSON String	Updates an existing entry
http://cs454.yourdomain.com/api/issues/:id	DELETE	empty	Deletes existing entry

How does $resource work?

- To use $resource inside your controller/service you need to declare a dependency on $resource.
- Then, you call $resource() function with your REST endpoint.

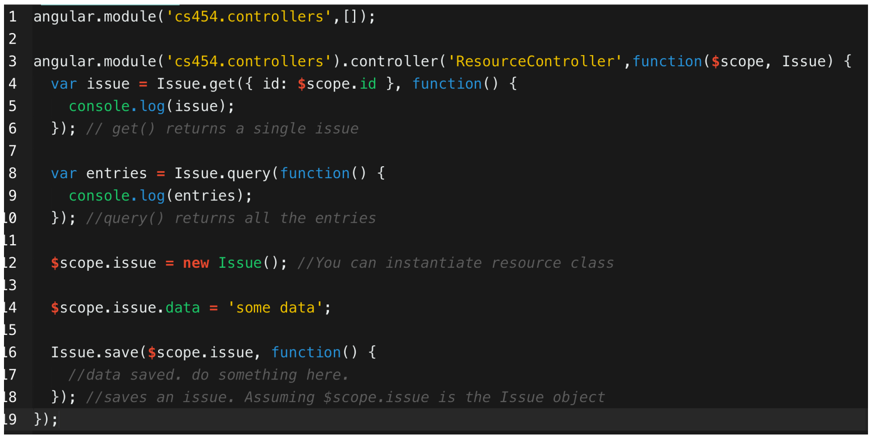
```
angular.module('myApp.services').factory('Issue', function($resource) {
  return $resource('/api/issues/:id'); // Note the full endpoint
address
});
```

- This returns a $resource class representation which can be used to interact with the REST backend.

How does $resource work?

- The following five methods are part of the resource class object:
  - get()
  - query()
  - save()
  - remove()
  - delete()

Using get(), query(), and save()



```
angular.module('cs454.controllers',[]);

angular.module('cs454.controllers').controller('ResourceController',function($scope, Issue) {
  var issue = Issue.get({ id: $scope.id }, function() {
    console.log(issue);
  }); // get() returns a single issue

  var entries = Issue.query(function() {
    console.log(entries);
  }); //query() returns all the entries

  $scope.issue = new Issue(); //You can instantiate resource class

  $scope.issue.data = 'some data';

  Issue.save($scope.issue, function() {
    //data saved. do something here.
  }); //saves an issue. Assuming $scope.issue is the Issue object
});
```

Using get(), query(), and save()

- The get() function in the above snippet issues a GET request to /api/issues/:id.
  - The parameter :id in the URL is replaced with $scope.id.
  - get() returns an empty object.
  - The object will be populated once the data is returned from the server.
  - The second argument to get() is a callback which is executed when the data arrives from server.
  - You can set the empty object returned by get() to the $scope and refer to it in the view.

Using get(), query(), and save()

- query() issues a GET request to /api/issues and returns an empty array.
  - Notice there is no :id
- Again, the array is populated when the data arrives from server.
- You can set the array to a reference on the $scope.
  - Once the data is populated, the view will be updated.

Using get(), query(), and save()

- The save() function issues a POST request to / api/issues.
  - The first argument is the POST body.
  - The second argument is a callback which is called when the data is saved.

Using get(), query(), and save()

- Recall that the return value of the $resource() function is a resource class.
- We can call new Issue() to instantiate an actual object out of this class
  - Once done, we can set various properties on it and finally save the object to backend.
- Ideally, you will only use get() and query() on the resource class (Issue in our case).
- All the non GET methods like save() and delete() are also available in the instance obtained by calling new Entry()
  - We'll call this a $resource instance.

Using get(), query(), and save()

- The difference is that these methods are prefixed with a $.
- The methods available in the $resource instance (as opposed to $resource class) are:
  - $save()
  - $delete()
  - $remove()

Using get(), query(), and save()

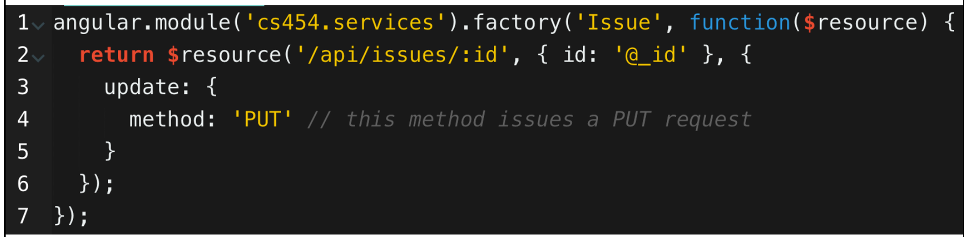
- For instance, the method $save() is used as follows:

```
$scope.issue = new Issue();
//this object now has a $save() method

$scope.issue.$save(function() {
  //data saved. $scope.issue is sent as the post
body.
});
```

What about update()?

- To support an update operation we need to modify our custom factory Issue as shown below:



```
angular.module('cs454.services').factory('Issue', function($resource) {
  return $resource('/api/issues/:id', { id: '@_id' }, {
    update: {
      method: 'PUT' // this method issues a PUT request
    }
  });
});
```

What about update()?

- The second argument to $resource() is a hash indicating what should be the value of the parameter :id in the URL.
- Setting it to @_id means whenever we call methods like $update() and $delete() on the resource instance, the value of :id will be set to the _id property of the instance.
- So now we can do the following:

```
$scope.issue.data = 'Some task to do…';
  $scope.issue.$update(function() {
    //updated in the backend
  });
```

What about update()?

- When the $update() function is called:
  - AngularJS knows the $update() function will trigger a PUT request to the URL /api/issues/:id.
- It reads the value of $scope.issue._id, assigns the value to :id and generates the URL.
- Sends a PUT request to the URL with $scope.issue as the post body.