The Angular $http Service

- Introduction to Ajax
- Promises
- The $http Service

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is AJAX?

- AJAX (Asychronous JavaScript and XML) uses a combination of existing technologies:
  - JavaScript
  - The XMLHttpRequest Object
  - HTML Document Object Model
- AJAX lets you create Web applications with a higher degree of interactivity



The notion of asynchronous request processing using the XMLHttpRequest object has been around for several years, but the term "AJAX" was coined by Jesse James Garrett of Adaptive Path.

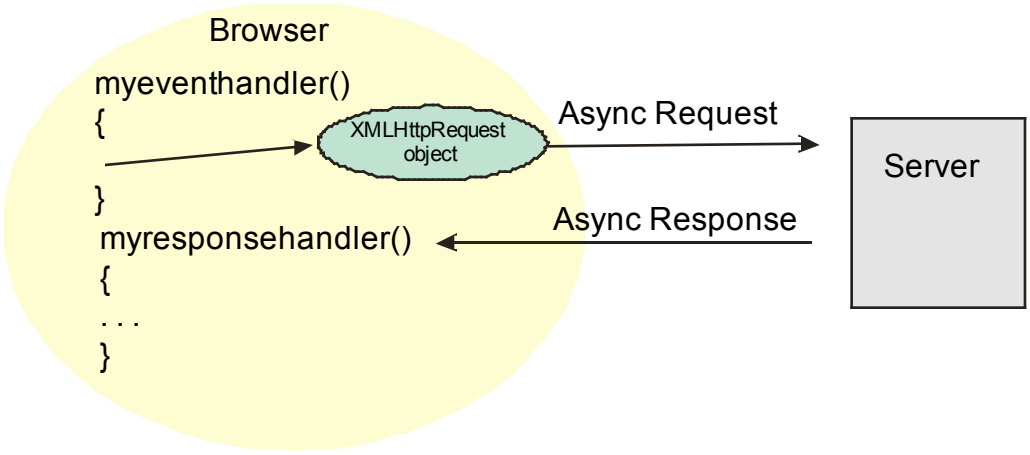
You can read his essay at www.adaptivepath.com/publications/essays/archives/000385.php.

Note that to use AJAX, you're not required to use XML.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

The XMLHttpRequest Object

- The **XMLHttpRequest** object was first implemented by Microsoft Internet Explorer, but is now a part of other browsers such as Firefox and Opera
- This object defines an API for making remote requests, either synchronously or asynchronously



The XMLHttpRequest object is the foundation of AJAX.

This object lets a JavaScript event handler (e.g. from a radio-button click), send a request to the server and establish another handler to process the response. The response handler can use the DOM to modify the page content based on the returned data. This technique is what lets us write AJAX applications that perform requests without requiring a full page reload.

The XMLHttpRequest object has an interesting history. First implemented by Microsoft Internet Explorer, it has since been "ported" to other browsers and has become a standard part of modern browsers.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Standard JavaScript Ajax Example

standardAjax.html

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
<html>
  <head>
    <title>Current Time and Date</title>
    <script>

      var req;

      function setDate(text)
      {
        var dateTxt=document.getElementById('dateTxt');
        dateTxt.firstChild.nodeValue=text;
      }

      function changeFmt(text)
      {
        var url = "GetDate?fmt=" + escape(text);
        //sendRequest (url, null, "GET");
        sendAsyncGetRequest(url);
      }

      function sendRequest(url,params,HttpMethod)
      {
        if (!HttpMethod)
        {
            HttpMethod="POST";
        }
        req=getXMLHttpRequest();
        if (req)
        {
            req.open(HttpMethod,url,true);
            req.setRequestHeader("Content-Type",
                "application/x-www-form-urlencoded");
            req.onreadystatechange = callback;
            req.send(params);
        }
    }

    function sendAsyncGetRequest(url)
    {
       req = getXMLHttpRequest();
       if (req)
       {
          req.open("GET", url, true);
          req.onreadystatechange = callback;
          req.send(null);
```

```
46          }
47      }
48
49      function callback()
50      {
51          if (req.readyState == 4)
52          {
53              if (req.status == 200)
54              {
55                  var text = req.responseText;
56                  setDate(text);
57              }
58              else
59              {
60                  alert("Problem: " + req.statusText);
61              }
62          }
63      }
64
65        function getXMLHttpRequest()
66        {
67          var xRequest=null;
68          if (window.XMLHttpRequest)
69          {
70              xRequest=new XMLHttpRequest();
71          }
72          else if (typeof ActiveXObject != "undefined")
73          {
74              xRequest=new ActiveXObject("Microsoft.XMLHTTP");
75          }
76          return xRequest;
77        }
78      </script>
79    </head>
80    <body>
81      <p>Choose format:</p>
82      <p><input type="radio" name="fmt" value="short"
83            onclick="changeFmt('short')">Short</p>
84       <p><input type="radio" name="fmt" value="medium"
85            onclick="changeFmt('medium')">Medium</p>
86       <p><input type="radio" name="fmt" value="long"
87            onclick="changeFmt('long')">Long</p>
88       <p id="dateTxt">No date</p>
89    </body>
90  </html>
```

Issues with Standard JavaScript Ajax

- Working with the XHR object directly is cumbersome, error-prone and requires a lot of code
- The XHR object itself is browser-specific, though modern browsers have converged on a common API

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

AngularJS and Ajax

- AngularJS provides the **$http** *service* that lets you issue HTTP GET, POST, UPDATE and DELETE requests

```
1    myModule.controller('MyController',
2       function($scope, $http) {
3     $scope.name="Placeholder";
4
5     $scope.getData = function() {
6       var response = $http.get("namedata.js");
7       response.success(function(data) {
8         $scope.name = data.name;
9       });
10
11      response.error(function() {
12        alert("Ajax error!");
13      });
14    };
15  });
```

An AngularJS "service" is a singleton object that implements some useful function or provides data that's shared by multiple controllers. Predfined AngularJS services have names that start with $.

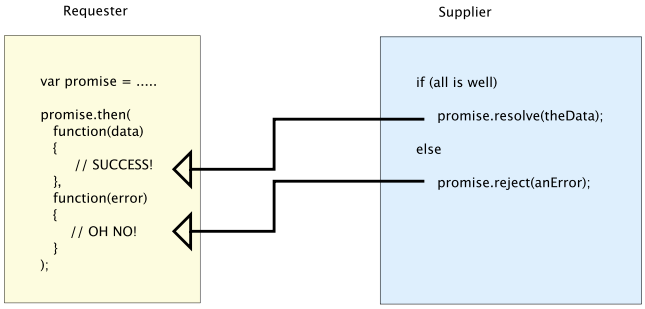
Note how we update the controller's signature to accept a reference to the $http service - this is a form of "dependency injection" - we essentially tell AngularJS that we need that object, and AngularJS obliges by finding its reference and passing it to our controller.

Since we often use AngularJS as the user interface for apps that access RESTful services, we can use the $http service to implement the classic CRUD (Create, Retrieve, Update, Delete) operations.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

What is a Promise?

- **Promises** is an emerging API for writing asynchronous code
- Many popular JavaScript frameworks, including AngularJS, implement this API
- A **promise** is an object that represents an asynchronous action
- The "supplier" calls **resolve()** or **reject()** to fulfill or reject the promise
- The "requester" calls **then()** to register one or two callbacks



You can read more about the concept of promises at https://www.promisejs.org

It's likely that ECMAScript v6 will provide a standardized promise API. For details, see:

`https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promis`

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

AngularJS and Promises

- AngularJS provides a **$q promise** service that makes it easier to work with asynchronous programming
- The **$http** service uses promises behind the scenes

```
1    // define the async request
2    var myAsyncWorker = $q.defer();
3
4    // retrieve the promise object
5    var promise = myAsyncWorker.promise;
6
7    // register success and error callbacks
8    promise.then(function(name)  {
9      $scope.name = name;
10   },
11   function(error) {
12     $scope.name = error;
13   });
```

testPromise.html

The AngularJS $q service is a lightweight implementation of a standalone JavaScript library of the same name. See https://github.com/kriskowal/q.

The $q service provides the defer() method to define an asynchronous object. This object defines the "promise" object on which we can register callbacks, and the resolve() and reject() methods that we use to indicate success and failure of the promise.

In the example shown here, we are working with some hypothetical service that provides an asynchronous result. In the service, we'd find code like:

```
if(gotGoodData)
  myAsyncWorker.resolve(theData);
else
  myAsyncWorker.reject("Bad data");
```

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    <!DOCTYPE html>
2    <html lang="en" ng-app="MyModule">
3      <head>
4        <script src="scripts/angular.min.js"></script>
5        <script>
6        var myModule = angular.module("MyModule", []);
7
8        myModule.controller("MyController",
9           function($scope, $timeout, $q)
10       {
11           $scope.name = "PLACEHOLDER";
12
13           // define the async request
14           var myAsyncWorker = $q.defer();
15
16           // retrieve the promise object
17           var promise = myAsyncWorker.promise;
18
19           // register success and error callbacks
20           promise.then(function(name) // OK
21           {
22               $scope.name = name;
23           },
24           function(error) // error
25           {
26               $scope.name = error;
27           });
28
29           // simulate async operation with timer
30           $timeout(function()
31           {
32               var random = Math.random();
33               console.log("Random number: " + random);
34
35               if (random > 0.5)
36                   myAsyncWorker.resolve("Sue");
37               else
38                   myAsyncWorker.reject("Bad boy");
39           }, Math.random() * 2000);
40       });
41       </script>
42     </head>
43     <body>
44       <div ng-controller="MyController">
45           <h1>Hello, {{name}}</h1>
```

```
46        </div>
47      </body>
48    </html>
```

The $http Service and Promises

- The $http service uses promises behind the scenes, but also lets you specify **success()** and **error()** functions
- By default, the service assumes JSON data format, but you can supply a **custom transformation** to support other formats

```
1    var response = $http.get("rest/students/12");
2
3    response.success(function(data, status, header, config)
4      {
5         . . .
6      });
7
8    response.error(function(data, status, header, config)
9      {
10        alert("Ajax error!");
11     });
```

2 - 9

The documentation for the $http service is at: https://docs.angularjs.org/api/ng/service/$http.

The success() method accepts optional parameters that give you access to the HTTP status code, HTTP headers and a configuration object.

The "header" parameter is a function that acts as a "getter" for headers. For example:

```
response.success(function(data, status, header, config)
{
   console.log("Location header: " + header("Location"));
});
```

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

RESTful CRUD Service Overview

- Many applications use the four **CRUD** operations: **Create**, **Retrieve**, **Update** and **Delete**
- The REST community has come up with guidelines for how to implement CRUD using standard HTTP facilities
- The AngularJS **$http** service provides methods so can you invoke the four CRUD methods from a Web page

---

The Rest in Practice book by Leonard Richardson is a very good reference for these guidelines.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Invoking the C in CRUD

- The $http service provides the **post()** convenience method that you can use to create a RESTful resource

```
1    var theStudent = {name: "Joe", gpa:  3.44};
2
3    var response = $http.post("rest/students", theStudent);
4    response.success(function(data, status, header, config)
5    {
6       console.log("New student ID: " + data.studentId);
7       alert("New student's RESTful URI: "
8         + header("Location"));
9    });
10
11   response.error(function(data, status)
12   {
13     alert("Ajax error, " + status);
14   });
```

2 - 11

The standard operation of CREATE is to use an HTTP POST to the service's URL and provide a resource representation. The server will perform the create and return a possibly modify resource in the response body. For example, the server might assign a unique student ID.

The service will return HTTP response code 201 Created on success or 400 Bad Request on error. The server will also set the response's Location header to the newly created resource's RESTful URI so you can retrieve it later.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Invoking the R in CRUD

- The $http service provides the **get()** convenience method that you can use to retrieve a RESTful resource

```
1    var response = $http.get("rest/students/12");
2
3    response.success(function(data)
4    {
5       console.log("Name: " + data.name);
6       console.log("ID: " + data.studentId);
7       console.log("GPA: " + data.gpa);
8    });
9
10   response.error(function(data, status)
11   {
12      alert("Ajax error: " + status);
13   });
```

The standard operation of RETRIEVE is to use an HTTP GET specifying the resource's RESTful URI. The server will return a resource representation in the response body with HTTP response code 200 OK.

If the server cannot locate the specified resource, it will return 404 Not Found.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Invoking the U in CRUD

- The $http service provides the **put()** convenience method that you can use to update a RESTful resource

```
1    var theStudent = {studentId: 12, name: "Sam", gpa:  2.22};
2
3    var response = $http.put("rest/students", theStudent);
4    response.success(function(data, status, header, config)
5    {
6       alert("Student successfully updated");
7    });
8
9    response.error(function(data, status, header, config)
10   {
11      alert("Update failed: " + status);
12   });
```

The standard operation of UPDATE is for the client to provide a complete resource representation containing the properties to update, using HTTP PUT. If successful, the server will update the resouce and return 204 No Content.

If the update fails, the server will return 404 Not Found or 409 Conflict.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Invoking the D in CRUD

- The $http service provides the **delete()** convenience method that you can use to delete a RESTful resource

```
1    var response = $http.delete("rest/students/12");
2
3    response.success(function(data, status, header, config)
4    {
5       alert("Student successfully deleted");
6    });
7
8    response.error(function(data, status, header, config)
9    {
10       alert("Delete failed: " + status);
11   });
```

The standard operation of DELETE is for the client to specify the RESTful URI of the resource to delete using HTTP DELETE. If succesful, the server will delete the resource and return 204 No Content.

If the delete fails, the server will return 404 Not Found or 405 Method Not Allowed.

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

Complete Example

The JAX-RS Service

Student.java

MyContextListener.java

StudentApplication.java

StudentService.java

AngularJS Clients

the-c-in-crud.html

the-r-in-crud.html

the-u-in-crud.html

the-d-in-crud.html

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum

```
1    package edu.bigcollege;
2
3    import java.io.Serializable;
4
5    import javax.xml.bind.annotation.XmlRootElement;
6
7    @SuppressWarnings("serial")
8    @XmlRootElement
9    public class Student implements Serializable
10   {
11       private int studentId;
12       private String name;
13       private double gpa;
14
15       public Student()
16       {
17       }
18
19       public Student(int studentId, String name, double gpa)
20       {
21           super();
22           this.studentId = studentId;
23           this.name = name;
24           this.gpa = gpa;
25       }
26
27       public int getStudentId()
28       {
29           return studentId;
30       }
31
32       public void setStudentId(int studentId)
33       {
34           this.studentId = studentId;
35       }
36
37       public String getName()
38       {
39           return name;
40       }
41
42       public void setName(String name)
43       {
44           this.name = name;
45       }
```

```
46
47      public double getGpa()
48      {
49          return gpa;
50      }
51
52      public void setGpa(double gpa)
53      {
54          this.gpa = gpa;
55      }
56
57      @Override
58      public int hashCode()
59      {
60          final int prime = 31;
61          int result = 1;
62          result = prime * result + studentId;
63          return result;
64      }
65
66      @Override
67      public boolean equals(Object obj)
68      {
69          if (this == obj)
70              return true;
71          if (obj == null)
72              return false;
73          if (getClass() != obj.getClass())
74              return false;
75          Student other = (Student) obj;
76          if (studentId != other.studentId)
77              return false;
78          return true;
79      }
80
81      @Override
82      public String toString()
83      {
84          StringBuilder builder = new StringBuilder();
85          builder.append("Student [studentId=").append(studentId)
86                  .append(", name=").append(name).append(", gpa=").append(gpa)
87                  .append("]");
88          return builder.toString();
89      }
90  }
```

```
1    package edu.bigcollege.listeners;
2
3    import java.util.ArrayList;
4    import java.util.List;
5
6    import javax.servlet.ServletContextEvent;
7    import javax.servlet.ServletContextListener;
8    import javax.servlet.annotation.WebListener;
9
10   import edu.bigcollege.Student;
11
12   @WebListener
13   public class MyContextListener implements ServletContextListener
14   {
15
16       @Override
17       public void contextDestroyed(ServletContextEvent ctx)
18       {
19       }
20
21       @Override
22       public void contextInitialized(ServletContextEvent ctx)
23       {
24           System.out.println("Context initialized");
25           List<Student> students = new ArrayList<Student>();
26
27           Student c = new Student();
28           c.setGpa(3.44);
29           c.setName("Sue Smith");
30           c.setStudentId(12);
31           students.add(c);
32
33           c = new Student();
34           c.setGpa(3.87);
35           c.setName("Bill Jones");
36           c.setStudentId(44);
37           students.add(c);
38
39           ctx.getServletContext().setAttribute("students", students);
40       }
41
42   }
```

```
1    package edu.bigcollege.service;
2
3    import java.util.HashSet;
4    import java.util.Set;
5
6    import javax.ws.rs.ApplicationPath;
7    import javax.ws.rs.core.Application;
8
9    @ApplicationPath("/rest/")
10   public class StudentApplication extends Application
11   {
12       @Override
13       public Set<Class<?>> getClasses()
14       {
15           Set<Class<?>> s = new HashSet<Class<?>>();
16           s.add(StudentService.class);
17
18           return s;
19       }
20   }
```

```
1    package edu.bigcollege.service;
2
3    import java.net.URI;
4    import java.net.URISyntaxException;
5    import java.util.List;
6    import java.util.Random;
7
8    import javax.servlet.ServletContext;
9    import javax.ws.rs.Consumes;
10   import javax.ws.rs.DELETE;
11   import javax.ws.rs.GET;
12   import javax.ws.rs.POST;
13   import javax.ws.rs.PUT;
14   import javax.ws.rs.Path;
15   import javax.ws.rs.PathParam;
16   import javax.ws.rs.Produces;
17   import javax.ws.rs.core.Context;
18   import javax.ws.rs.core.Response;
19   import javax.ws.rs.core.Response.ResponseBuilder;
20   import javax.ws.rs.core.Response.Status;
21
22   import edu.bigcollege.Student;
23
24   @Path("students")
25   public class StudentService
26   {
27       @Context
28       ServletContext servletContext;
29
30       @SuppressWarnings("unchecked")
31       @DELETE
32       @Path("{id}")
33       public Response delete(@PathParam("id") int id)
34       {
35           ResponseBuilder rb = null;
36
37           System.out.println("In delete: " + id);
38
39           Student found = findStudent(id);
40           if(found != null)
41           {
42               List<Student> students = (List<Student>) servletContext
43                       .getAttribute("students");
44
45               students.remove(found);
```

```
46              rb = Response.noContent();
47          }
48          else
49          {
50              rb = Response.status(Status.NOT_FOUND);
51          }
52
53          return rb.build();
54      }
55
56      @PUT
57      @Consumes({ "text/xml", "application/json" })
58      public Response update(Student s)
59      {
60          System.out.println("In update: " + s);
61          ResponseBuilder rb = null;
62
63          if(s != null)
64          {
65              Student found = findStudent(s.getStudentId());
66
67              if(found != null)
68              {
69                  found.setGpa(s.getGpa());
70                  found.setName(s.getName());
71                  rb = Response.noContent();
72              }
73              else
74              {
75                  rb = Response.status(Status.NOT_FOUND);
76              }
77          }
78          else
79          {
80              rb = Response.status(Status.CONFLICT);
81          }
82
83          return rb.build();
84      }
85
86
87      @SuppressWarnings("unchecked")
88      @POST
89      @Consumes({ "text/xml", "application/json" })
90      // need @Produces since we send back updated Student
```

```
91      // representation in response body
92      @Produces({ "text/xml", "application/json" })
93      public Response insert(Student s)
94      {
95          System.out.println("In insert: " + s);
96          ResponseBuilder rb = null;
97
98          if(s != null)
99          {
100             List<Student> students = (List<Student>) servletContext
101                     .getAttribute("students");
102
103             Random r = new Random();
104             s.setStudentId(Math.abs(r.nextInt()));
105
106             students.add(s);
107
108             try
109             {
110                 rb = Response.created(new URI("/students/" +
111                     s.getStudentId()));
112                 // marshal the student into the response body
113                 rb.entity(s);
114             }
115             catch (URISyntaxException e)
116             {
117                 System.out.println("ERROR inserting: " +
118                   e.getMessage());
119                 rb = Response.serverError();
120             }
121         }
122         else
123         {
124             rb = Response.status(Status.BAD_REQUEST);
125         }
126
127         return rb.build();
128     }
129
130     @SuppressWarnings("unchecked")
131     @GET
132     @Path("{id}")
133     @Produces({ "text/xml", "application/json" })
134     public Response getStudent(@PathParam("id") int id)
135     {
```

```
136         Response ret = null;
137
138         List<Student> students = (List<Student>) servletContext
139                 .getAttribute("students");
140
141         Student theStudent = null;
142         for (Student c : students)
143         {
144             if (c.getStudentId() == id)
145             {
146                 theStudent = c;
147                 break;
148             }
149         }
150
151         if (theStudent != null)
152         {
153             ret = Response.ok(theStudent).build();
154         }
155         else
156         {
157             ret = Response.status(Response.Status.NOT_FOUND).build();
158         }
159
160         return ret;
161     }
162
163     @SuppressWarnings("unchecked")
164     @GET
165     @Produces({ "text/xml", "application/json" })
166     public Response getAllStudents()
167     {
168         Response ret = null;
169
170         List<Student> students = (List<Student>) servletContext
171                 .getAttribute("students");
172
173         Student[] theStudents = new Student[students.size()];
174
175         int i = 0;
176         for(Student s : students)
177             theStudents[i++] = s;
178
179         ret = Response.ok(theStudents).build();
180
```

```
181         return ret;
182     }
183 
184     @SuppressWarnings("unchecked")
185     private Student findStudent(int id)
186     {
187         List<Student> students = (List<Student>) servletContext
188                 .getAttribute("students");
189 
190         Student found = null;
191         for(Student temp : students)
192         {
193             if(temp.getStudentId() == id)
194             {
195                 found = temp;
196                 break;
197             }
198         }
199         return found;
200     }
201 
202 }
```

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="ISO-8859-1">
    <title>AngularJS Create RESTful Client</title>
    <script src="angular.min.js"></script>
    <script>
      var myModule = angular.module('MyModule', []);

      myModule.controller('MyController', function($http)
      {
         var mc = this;

         mc.name="Name Placeholder";
         mc.gpa = "GPA Placeholder"
         mc.newId = "New ID Placeholder";
         mc.newURI = "New URI Placeholder"

         // respond to form submit by making an Ajax request
         mc.createStudent = function()
         {
           // create object from form data
           var theStudent =
           {
              name: mc.name,
              gpa:  mc.gpa
           };

           var response = $http.post("rest/students", theStudent);
           response.success(function(data, status, header, config)
           {
                console.log("New student's RESTful URI: " + header("Location"));

                mc.newId = data.studentId;
                mc.newURI = header("Location");
           });

           response.error(function(data, status, header, config)
           {
             alert("Ajax error!");
           });
         };
      });
    </script>
  </head>
```

```
46      <body data-ng-app="MyModule">
47        <div data-ng-controller="MyController as mc">
48          <h1>Create Student</h1>
49          <form data-ng-submit="mc.createStudent()">
50            <div>
51               New student name: <input data-ng-model="mc.name" type="text">
52            </div>
53            <div>
54               New student GPA: <input data-ng-model="mc.gpa" type="number" min="0.0"
55            </div>
56            <input type="submit" value="Create Student on Server">
57          </form>
58          <div>
59            New student's ID: <span data-ng-bind="mc.newId">Placeholder</span>
60          </div>
61          <div>
62            New student's RESTful URI: <span data-ng-bind="mc.newURI">Placeholder</spa
63          </div>
64        </div>
65      </body>
66    </html>
```

```
1    <!DOCTYPE html>
2    <html>
3      <head>
4        <meta charset="ISO-8859-1">
5        <title>AngularJS Retrieve RESTful Client</title>
6        <script src="angular.min.js"></script>
7        <script>
8          var myModule = angular.module('MyModule', []);
9
10         myModule.controller('MyController', function($http)
11         {
12            var mc = this;
13
14            mc.name="Name Placeholder";
15            mc.id = "ID Placeholder";
16            mc.gpa = "GPA Placeholder"
17
18            mc.getStudent = function()
19            {
20              var response = $http.get("rest/students/12");
21              response.success(function(data)
22              {
23                mc.name = data.name;
24                mc.id = data.studentId;
25                mc.gpa = data.gpa;
26              });
27
28              response.error(function(data, status, header, config)
29              {
30                alert("Ajax error!");
31              });
32            };
33
34            mc.allStudents = [];
35
36            mc.getAllStudents = function()
37            {
38                var response = $http.get("rest/students");
39                response.success(function(data)
40                {
41                    mc.allStudents = data.student;
42                });
43
44                response.error(function(data, status, header, config)
45                {
```

```
46                  alert("Ajax error!");
47                });
48
49            };
50
51        });
52      </script>
53    </head>
54    <body data-ng-app="MyModule">
55      <h1>Get Student #12</h1>
56      <div data-ng-controller="MyController as mc">
57        <div>
58           Name: {{mc.name}}
59         </div>
60        <div>
61           ID: {{mc.id}}
62         </div>
63        <div>
64           GPA: {{mc.gpa}}
65         </div>
66        <form data-ng-submit="mc.getStudent()">
67          <input type="submit" value="Get Student #12 from Server">
68        </form>
69        <h1>All Student IDs</h1>
70        <form data-ng-submit="mc.getAllStudents()">
71          <input type="submit" value="Get All Student IDs from Server">
72        </form>
73        <ul>
74          <li data-ng-repeat = "student in mc.allStudents">{{student.studentId}}</li
75        </ul>
76      </div>
77    </body>
78  </html>
```

```
1    <!DOCTYPE html>
2    <html>
3      <head>
4        <meta charset="ISO-8859-1">
5        <title>AngularJS Update RESTful Client</title>
6        <script src="angular.min.js"></script>
7        <script>
8          var myModule = angular.module('MyModule', []);
9
10         myModule.controller('MyController', function($http)
11         {
12            var mc = this;
13
14            mc.id = "ID Placeholder";
15            mc.name="Name Placeholder";
16            mc.gpa = "GPA Placeholder"
17
18            mc.response = "Placeholder";
19
20            // respond to form submit by making an Ajax request
21            mc.updateStudent = function()
22            {
23              // create object from form data
24              var theStudent =
25              {
26                 studentId: mc.id,
27                 name: mc.name,
28                 gpa:  mc.gpa
29              };
30
31              var response = $http.put("rest/students", theStudent);
32              response.success(function(data, status, header, config)
33              {
34                  mc.response = "Student successfully updated";
35              });
36
37              response.error(function(data, status, header, config)
38              {
39                  mc.response = "Update failed: " + status;
40              });
41            };
42         });
43       </script>
44     </head>
45     <body data-ng-app="MyModule">
```

```
46      <div data-ng-controller="MyController as mc">
47        <h1>Update Student</h1>
48        <form data-ng-submit="mc.updateStudent()">
49          <div>
50             ID of student to update: <input data-ng-model="mc.id" type="number">
51          </div>
52          <div>
53             Student's new name: <input data-ng-model="mc.name" type="text">
54          </div>
55          <div>
56             Student's new GPA: <input data-ng-model="mc.gpa" type="number" min="0.0
57          </div>
58          <input type="submit" value="Update Student on Server">
59        </form>
60        <div>
61          Response from service: <span data-ng-bind="mc.response">Placeholder</span>
62        </div>
63      </div>
64    </body>
65  </html>
```

```
<!DOCTYPE html>
<html>
  <head>
    <meta charset="ISO-8859-1">
    <title>AngularJS Delete RESTful Client</title>
    <script src="angular.min.js"></script>
    <script>
      var myModule = angular.module('MyModule', []);

      myModule.controller('MyController', function($http)
      {
         var mc = this;

         mc.id = "ID Placeholder";

         mc.response = "Placeholder";

         // respond to form submit by making an Ajax request
         mc.deleteStudent = function()
         {
           var response = $http.delete("rest/students/" + mc.id);
           response.success(function(data, status, header, config)
           {
               mc.response = "Student successfully deleted";
           });

           response.error(function(data, status, header, config)
           {
               mc.response = "Delete failed: " + status;
           });
         };
      });
    </script>
  </head>
  <body data-ng-app="MyModule">
    <div data-ng-controller="MyController as mc">
      <h1>Delete Student</h1>
      <form data-ng-submit="mc.deleteStudent()">
        <div>
           ID of student to delete: <input data-ng-model="mc.id" type="number">
        </div>
        <input type="submit" value="Delete Student on Server">
      </form>
      <div>
        Response from service: <span data-ng-bind="mc.response">Placeholder</span>
```

```
46          </div>
47        </div>
48      </body>
49    </html>
```

Chapter Summary

In this chapter, you learned:

- How to use AngularJS's **$http** service to invoke RESTful services

Copyright © Descriptor Systems, 2001-2014. Course materials may not be reproduced in whole or in part without prior written consent of Joel Barnum