RESTful CRUD Services

- Create
- Retrieve
- Update
- Delete

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

The Richardson Maturity Model

- According to the **Richardson Maturity Model**, which evaluates how RESTful an application is, apps that correctly implement CRUD are at **Level 2**

```
Level 0  Use a single verb, typically POST, and a single URI
         for all interactions. Most SOAP based services are
         like this

Level 1  Use a single verb, typically GET, but multiple URIs.
         These types of services often violate the idempotency
         of GET

Level 2  Use multiple verbs and multiple URIs. This includes
         CRUD RESTful apps

Level 3  Use Hypermedia as the Engine of Application State
```

You can read more about the RMM at:

http://martinfowler.com/articles/richardsonMaturityModel.html

According to the Wikipedia entry on "idempotence":

In computer science, the term idempotent is used more comprehensively to describe an operation that will produce the same results if executed once or multiple times.

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 Services

- 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 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

HTTP Review

- HTTP provides several **methods** as well as standard **return codes** and **headers**

```
HTTP Methods
------------
GET, POST,
PUT, DELETE,
HEAD, PATCH,
OPTIONS, TRACE,
CONNECT


HTTP Headers
------------
Accept
Content-Type
Status
...
```

```
HTTP Status Codes
-----------------
200 OK
201 Created
204 No Content
400 Bad Request
404 Not Found
409 Conflict
500 Internal Server Error
...
```

For more information on HTTP, see the specification:

`http://www.w3.org/Protocols/rfc2616/rfc2616.html`

Note that this is not a complete listing of status codes and headers, only those commonly used by 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

Media Types

- The IANA defines **media types (MIME)** that services can use to specify data types

```
Common Media Types
------------------

application/json
text/plain
text/xml
application/vnd.*     -- vendor specific
```

For more information on the Internet Assigned Numbers Authority (IANA) and media types, see:

https://www.iana.org/assignments/media-types

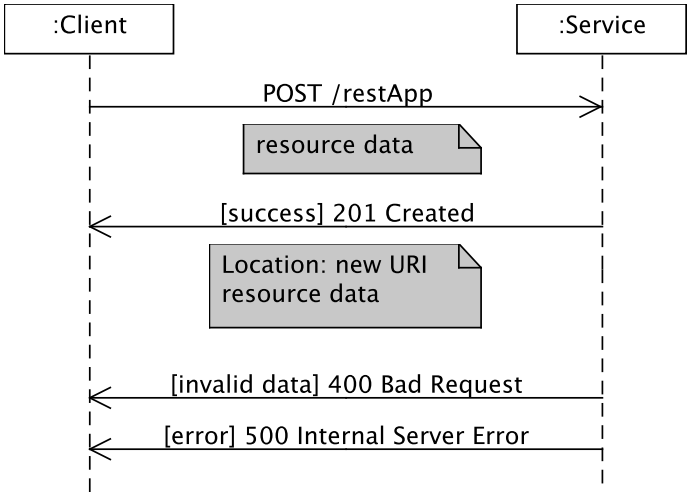
There is a lot of discussion in the REST community on whether to use standard media types like text/xml or application/json or to define custom media types for the application. See this article for a discussion:

http://www.infoq.com/news/2010/01/subbu-custom-media-types

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

Implementing Create

- Client should **POST** a resource representation and will receive a possibly modified resource representation
- Service returns response code **201 Created** on success or **400 Bad Request** or **500 Internal Server Error**



The response message also contains an HTTP Location header that specifies the newly created resource's RESTful URI.

The body of the response contains a possibly different resource representation than was passed, since it might contain auto-generated data like a database primary key.

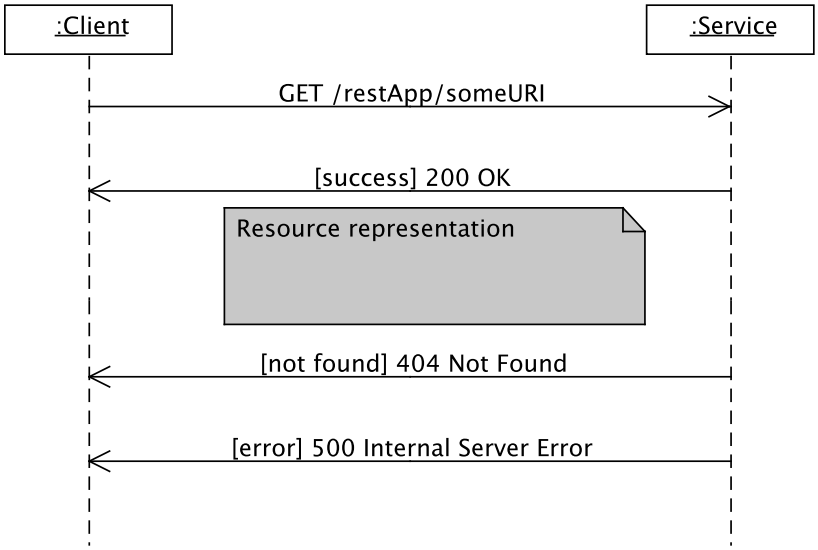
This page discusses using HTTP 201 vs 202:

`http://benramsey.com/blog/2008/04/http-status-201-created-vs-202-accepted/`

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

Implementing Retrieve

- Client should **GET** to a URI for the requested resource
- Service returns requested resource and **200 OK** or **404 Not Found** or **500 Internal Server Error**

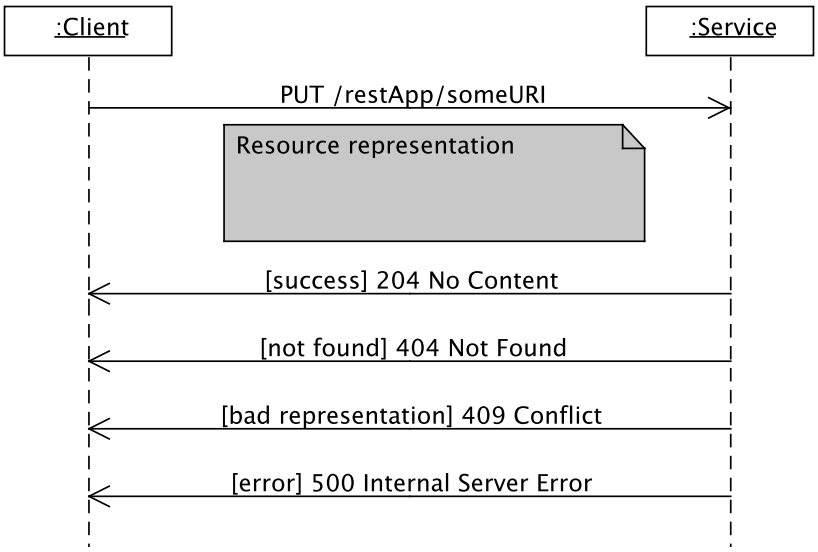


Even though the client can expect that the GET request itself doesn't change state on the server, it's not necessarily true that multiple GETs return the same data. For example, consider an Order-status request - the status might initially be "in progress" and eventually "complete".

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

Implementing Update

- Client should **PUT** a complete resource representation to the resource's URI
- Service updates the requested resource and returns **204 No Content**, **404 Not Found**, **409 Conflict** or **500 Internal Server Error**



Many RESTful applications follow this convention (established by Richardson and Ruby): <pre>- Use POST to create a resource identified by a service-generated URI. - Use POST to append a resource to a collection identified by a service-generated URI. - Use PUT to create or overwrite a resource identified by a URI computed by the client. </pre> The service could also return 200 OK, but the HTTP specification requires that such responses contain a response body, while 204 does not.

The service can return 409 Conflict if the supplied resource representation is incorrect, for example ill formed XML.

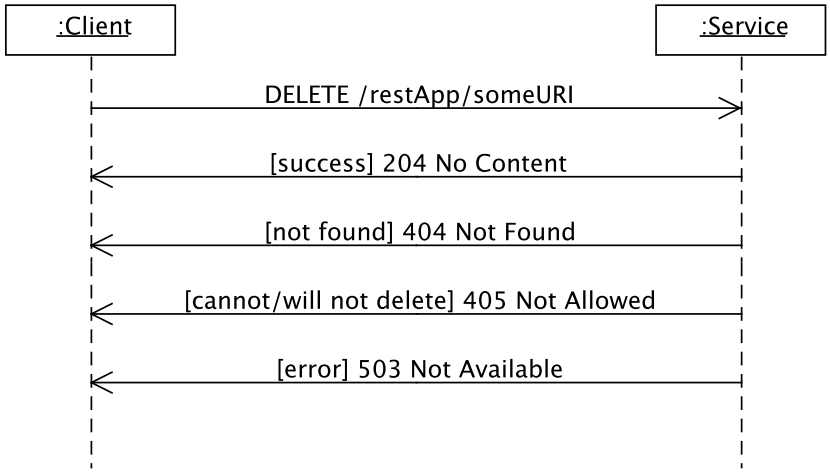
Note that the client must send complete resource representation. There is an emerging concept of using HTTP PATCH to provide just a delta to update.

Note that PUT should be idempotent since the client provides the entire resource. So if the service returns 500, it should be OK for the client to resend the request.

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

Implementing Delete

- Client uses HTTP **DELETE** specifying the resource's URI
- Service returns **204 No Content** on success or **404 Not Found**, **405 Method Not Allowed** or **503 Service Unavailable**



One reason the service might return 405 is if it refuses to delete for some reason. For example, the service might have a policy that once an order is fully complete, it cannot be deleted.

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

Complete JAX-RS CRUD Example

Student.java

MyContextListener.java

StudentApplication.java

StudentService.java

GetClient.java

PostClient.java

PutClient.java

DeleteClient.java

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

```
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     private Student findStudent(int id)
165     {
166         List<Student> students = (List<Student>) servletContext
167                 .getAttribute("students");
168
169         Student found = null;
170         for(Student temp : students)
171         {
172             if(temp.getStudentId() == id)
173             {
174                 found = temp;
175                 break;
176             }
177         }
178         return found;
179     }
180
```

```
181   }
```

```
1    package client;
2
3    import java.io.BufferedReader;
4    import java.io.InputStreamReader;
5    import java.net.HttpURLConnection;
6    import java.net.URL;
7
8    public class GetClient
9    {
10
11       public static void main(String[] args) throws Exception
12       {
13           URL url = new URL(
14             "http://localhost:9081/TestRestCRUDWeb/rest/students/12");
15           HttpURLConnection conn =
16             (HttpURLConnection) url.openConnection();
17           conn.setRequestMethod("GET");
18           conn.setRequestProperty("Accept", "application/json");
19
20           if (conn.getResponseCode() != 200)
21           {
22               throw new RuntimeException("Failed : HTTP error code : "
23                       + conn.getResponseCode());
24           }
25
26           BufferedReader br = new BufferedReader(new InputStreamReader(
27                   (conn.getInputStream())));
28
29           String output;
30           System.out.println("Output from Server .... \n");
31           while ((output = br.readLine()) != null)
32           {
33               System.out.println(output);
34           }
35
36           conn.disconnect();
37
38       }
39   }
```

```
1    package client;
2
3    import java.io.BufferedReader;
4    import java.io.InputStreamReader;
5    import java.io.OutputStream;
6    import java.net.HttpURLConnection;
7    import java.net.URL;
8
9    public class PostClient
10   {
11       public static void main(String[] args) throws Exception
12       {
13           URL url =
14             new URL(
15              "http://localhost:9081/TestRestCRUDWeb/rest/students");
16           HttpURLConnection conn =
17             (HttpURLConnection) url.openConnection();
18           conn.setDoOutput(true);
19           conn.setRequestMethod("POST");
20           conn.setRequestProperty("Content-Type", "application/json");
21
22           String input = "{\"name\":\"Harry Wolfe\",\"gpa\":2.22}";
23
24           OutputStream os = conn.getOutputStream();
25           os.write(input.getBytes());
26           os.flush();
27
28           if (conn.getResponseCode() != HttpURLConnection.HTTP_CREATED)
29           {
30               throw new RuntimeException("Failed : HTTP error code : "
31                       + conn.getResponseCode());
32           }
33
34           BufferedReader br = new BufferedReader(new InputStreamReader(
35                   (conn.getInputStream())));
36
37           String output;
38           System.out.println("Output from Server .... \n");
39           while ((output = br.readLine()) != null)
40           {
41               System.out.println(output);
42           }
43
44           System.out.println("Location header: "
45                   + conn.getHeaderField("Location"));
```

```
46
47              conn.disconnect();
48          }
49      }
```

```
1    package client;
2
3    import java.io.OutputStream;
4    import java.net.HttpURLConnection;
5    import java.net.URL;
6
7    public class PutClient
8    {
9        public static void main(String[] args) throws Exception
10       {
11           URL url =
12             new URL(
13               "http://localhost:9081/TestRestCRUDWeb/rest/students");
14           HttpURLConnection conn =
15             (HttpURLConnection) url.openConnection();
16           conn.setDoOutput(true);
17           conn.setRequestMethod("PUT");
18           conn.setRequestProperty("Content-Type", "application/json");
19
20           String input =
21             "{\"name\":\"Joe Blow\",\"gpa\":4.56, \"studentId\":12}";
22
23           OutputStream os = conn.getOutputStream();
24           os.write(input.getBytes());
25           os.flush();
26
27           if (conn.getResponseCode() !=
28              HttpURLConnection.HTTP_NO_CONTENT)
29           {
30               throw new RuntimeException("Failed : HTTP error code : "
31                       + conn.getResponseCode());
32           }
33
34           conn.disconnect();
35       }
36   }
```

```
1    package client;
2
3    import java.net.HttpURLConnection;
4    import java.net.URL;
5
6    public class DeleteClient
7    {
8        public static void main(String[] args) throws Exception
9        {
10           URL url =
11             new URL(
12               "http://localhost:9081/TestRestCRUDWeb/rest/students/44");
13           HttpURLConnection conn =
14             (HttpURLConnection) url.openConnection();
15           conn.setRequestMethod("DELETE");
16
17           conn.connect();
18
19           if (conn.getResponseCode() != HttpURLConnection.HTTP_NO_CONTENT)
20           {
21               throw new RuntimeException("Failed : HTTP error code : "
22                       + conn.getResponseCode());
23           }
24
25           conn.disconnect();
26       }
27   }
```

Chapter Summary

In this chapter, you learned:

- The fundamentals of REST applications that create, retrieve, update and delete resources
- How to implement a RESTful CRUD application using JAX-RS

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