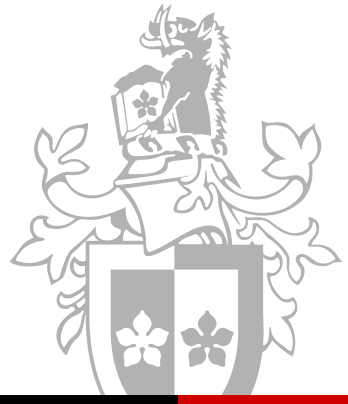
COS30041 Creating Secure and Scalable Software

Lecture 10 RESTful Web Services



SWINBURNE UNIVERSITY OF TECHNOLOGY

Learning Objectives

- After studying the lecture material, you will be able to
  - Understand and describe what RESTful Web Services are
  - Understand and describe the architectural properties and constraints of RESTful Web Services
  - Understand the issues involved in programming the RESTful Web Services
  - Program the RESTful Web Services via JAX-RS
  - Program client applications that call the RESTful Web services

Pre-requisite

- Object Oriented Programming
- Some experiences on XML would be an advantage

Outline

- RESTful Web Services
- Programming RESTful Web Services
  - Server side component
  - Client program

Roadmap

- **RESTful Web Services**
- Programming RESTful Web Services
  - Server side component
  - Client program

RESTful Web Services

- REpresentational State Transfer (REST)
- A way to provide interoperability between “software” on the internet
- Based on “Resources” and their operations
  - Standardize “operations”
  - Map C / R / U / D to HTTP’s POST / GET / PUT / DELETE
- Using HTTP
- Using HTML, XML or JSON for representation of Resources
- Using URI (Uniform Resource Identifier)

REST

- “Throughout the HTTP standardization process, …

  …

  …

  That process honed my model down to a core set of principles, properties, and constraints that are now called REST”

  Roy Fielding (source REST’s wiki)

- An architectural style (mostly known)

Architectural Properties

- Performance
- Scalability
- Simplicity
- Modifiability
- Visibility
- Portability
- Reliability

Six Architectural Constraints

- Client-server
  - Client [UI concerns] vs Server [data storage concerns]
- Stateless
  - Session state is held in client
- Cacheable
- Layered system
  - Client can't tell whether it is connected directly to the end server
- Code on demand (optional)
- Uniform interface

Six Architectural Constraints (cont'd)

- Uniform interface
  - Identification of resources – URI
  - Manipulation of resources through representation
  - Self-descriptive messages
  - Hypermedia as the engine of application state (HATEOAS)

Roadmap

- RESTful Web Services
- **Programming RESTful Web Services**
  - Server side component
  - Client program

Prog. RESTful WS using JAX-RS

- Similar to “Big” WS
  - Software Modules in Web Tier
  - Client and Server communicate via HTTP (basically text)
  - No actual DTO passing
    - Sorry, I do not consider JSON as an object in OO Programming sense!
    - To me, JSON is just “name”-”value” pairs formatted in “{“ and “}”
    - If JSON is an object, the corresponding XML doc can also be called XMLON!

Roadmap

- RESTful Web Services
- Programming RESTful Web Services
  - **Server side component**
  - Client program

Prog. RESTful WS using JAX-RS – Server side

- For Web Service Java Class
  - @Path(“xxxx”) – specify the RESTful web service “object” for a Java class
    - @Path(“entity.myuser”)
      public class MyuserFacadeREST
- For methods in Web Service Java Class
  - @Path(“{vvvv}”) – specify a parameter to be used in a method
  - @PathParam(“vvvv”) – specify which parameter to get the values, in a method call
    - @DELETE
      @Path(“{userid}”)
      public void deleteRecord(@PathParam(“userid”) String userid)

Prog. RESTful WS using JAX-RS – Server side (cont'd)

- Standardize method naming
  - Use @POST / @GET / @PUT / @DELETE in method that corresponds the HTTP verbs POST / GET / PUT / DELETE
    - @POST
      ```
      @Consumes({MediaType.APPLICATION_XML})
      public void create(Myuser myuser)
      ```
    - @GET
      ```
      @Path("{userid}")
      @Produces({MediaType.APPLICATION_XML})
      public Myuser getRecord(@PathParam("userid") userid)
      ```
- See Parameter Passing slides (later) for a discussion on "@Consumes" and "@Produces"

Example (server side) – MyuserFacadeREST

```java
/**
 *
 * @author elau
 */
@Stateless
@Path("entity.myuser")
public class MyuserFacadeREST extends AbstractFacade<Myuser> {

    @PersistenceContext(unitName = "ED-Myuser-RS2-warPU")
    private EntityManager em;

    public MyuserFacadeREST() {
        super(Myuser.class);
    }

    @POST
    @Override
//    @Consumes({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
    @Consumes({MediaType.APPLICATION_XML})
    public void create(Myuser entity) {
        super.create(entity);
    }

    @GET
    @Path("{id}")
//    @Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
    @Produces({MediaType.APPLICATION_XML})
    public Myuser find(@PathParam("id") String id) {
        return super.find(id);
    }
```

Prog. RESTful WS using JAX-RS – Parameter Passing

- Similar to “Big” WS, the parameters sit on client and server
- Use @Path and @PathParam in server side module to specify the parameters to be used from HTTP request
- Primitive data type – Easy (no extra work at all)
- Java Object / User-defined Class [Type] – More work
  - Object → XML / JSON – @Produces “the message format”
  - XML / JSON → Object – @Consumes “the message format”

JAX-RS – Parameter Passing – Message format

XML

- MediaType.APPLICATION_XML or “application/xml”

JSON

- MediaType.APPLICATION_JSON or “application/json”
- Not supported at the moment
  - Even though JavaEE Tutorial says it is supported [via Apache Maven project – a different setting]

- Use JAXB as in “Big” WS
  - @XmlRootElement
  - @XmlAccessorType(XmlAccessType.FIELD)
  - @XmlElement(requited = true)

Example (Passing Object, JAXB) – Myuser

```
 * @author elau
 */
@Entity
@Table(name = "MYUSER")
@XmlRootElement(name = "Myuser")
@XmlAccessorType(XmlAccessType.FIELD)
@NamedQueries({
    @NamedQuery(name = "Myuser.findAll", query = "SELECT m FROM Myuser m")
    , @NamedQuery(name = "Myuser.findByUserid", query = "SELECT m FROM Myuser
    , @NamedQuery(name = "Myuser.findByName", query = "SELECT m FROM Myuser m
    , @NamedQuery(name = "Myuser.findByPassword", query = "SELECT m FROM Myus
    , @NamedQuery(name = "Myuser.findByEmail", query = "SELECT m FROM Myuser
    , @NamedQuery(name = "Myuser.findByPhone", query = "SELECT m FROM Myuser
    , @NamedQuery(name = "Myuser.findByAddress", query = "SELECT m FROM Myuse
    , @NamedQuery(name = "Myuser.findBySecqn", query = "SELECT m FROM Myuser
    , @NamedQuery(name = "Myuser.findBySecans", query = "SELECT m FROM Myuser
public class Myuser implements Serializable {

    private static final long serialVersionUID = 1L;
    @Id
    @Basic(optional = false)
    @NotNull
    @Size(min = 1, max = 6)
    @Column(name = "USERID")
    @XmlElement(required=true)
    private String userid;
    @Size(max = 30)
    @Column(name = "NAME")
    @XmlElement(required=true)
    private String name;
```

Roadmap

- RESTful Web Services
- Programming RESTful Web Services
  - Server side component
  - **Client program**

Prog. RESTful WS client using JAX-RS

- Need to know the following
  - URI for the required RESTful web services
  - how to “access” those resources you need (e.g. database records)
- All these can be found from the RESTful WS WADL
  - Example:
    http://localhost:8080/ED-Myuser-RS-WAR/webresources/application.wadl

Example (WADL)

```
<resources base="http://localhost:8080/ED-Myuser-RS-WAR/webresources/">
  <resource path="entity.myuser">
    <method id="create" name="POST">
      <request>
        <ns2:representation xmlns:ns2="http://wadl.dev.java.net/2009/02" xmlns="" element="Myuser" mediaType="application/xml"/>
      </request>
    </method>
    <method id="findAll" name="GET">
      <response>
        <ns2:representation xmlns:ns2="http://wadl.dev.java.net/2009/02" xmlns="" element="Myuser" mediaType="application/xml"/>
      </response>
    </method>
    <resource path="{id}">
      <param xmlns:xs="http://www.w3
      <method id="remove" name="DELE
      <method id="find" name="GET">
        <response>
          <ns2:representation xmlns:n
        </response>
      </method>
      <method id="edit" name="PUT">
        <request>
          <ns2:representation xmlns:n
        </request>
      </method>
    </resource>
    <resource path="{from}/{to}">
      <param xmlns:xs="http://www.w3
      <param xmlns:xs="http://www.w3
      <method id="findRange" name="G
        <response>
          <ns2:representation xmlns:n
        </response>
      </method>
    </resource>
    <resource path="count">
      <method id="countREST" name="G
        <response>
          <representation mediaType="
        </response>
      </method>
    </resource>
  </resource>
</resources>
```

```
<resources base="http://localhost:8080/ED-Myuser-RS-WAR/webresources
  <resource path="entity.myuser">
    <method id="create" name="POST">
      <request>
        <ns2:representation xmlns:ns2="http://wadl.dev.java.net/2009/
      </request>
    </method>
    <method id="findAll" name="GET">
      <response>
        <ns2:representation xmlns:ns2="http://wadl.dev.java.net/2009/
      </response>
    </method>
    <resource path="{id}">
      <param xmlns:xs="http://www.w3.org/2001/XMLSchema" name="id" st
      <method id="remove" name="DELETE"/>
      <method id="find" name="GET">
        <response>
          <ns2:representation xmlns:ns2="http://wadl.dev.java.net/200
        </response>
      </method>
      <method id="edit" name="PUT">
        <request>
```

Example (URI and paths)

- [URI]: http://localhost:8080/ED-Myuser-RS-WAR/webresources/
- ${URI}/entity.myuser do something with Myuser records
  - POST – create a new myuser
  - GET – return all myuser
- ${URI}/entity.myuser/{id} do something with the Myuser record whose userid = “{id}”
  - GET – return the record
  - POST – update the record
  - DELETE – delete the record

Prog. RESTful WS client using JAX-RS (cont'd)

- How to call the method that we want?
- Steps [Generic]
  - 1. use ClientBuilder.newClient() to construct a Client object
  - 2. use Client.target() and Client.path() to point the Client object to the right URI for the resources, the result is a WebTarget object
  - 3. use the WebTarget.request() to "formulate" a request
  - 4. use the Builder.post() / .get() / .put() / .delete() method to perform the required HTTP POST / GET / PUT / DELETE for the CRUD operation

Example (Client's calling method)

Generic

Specific [Get a Myuser record whose userid is "000099"

URI is http://localhost:8080/ED-Myuser-RS-WAR/webresources/entity.myuser/000099

- 1. use ClientBuilder.newClient() to construct a Client object
- 2. use Client.target() and .path() to point the Client object to the right URI for the resources, the result is a WebTarget object
- 3. use the WebTarget.request() to "formulate" a request
- 4. use the Builder.post() / .get() / .put() / .delete() method to perform the required HTTP POST / GET / PUT / DELETE for the CRUD operation

- Client client = ClientBuilder.newClient();
- WebTarget wt =
  client.target("http://…/webresources")
  .path("entity.myuser")
  .path(Message.format("{0}",
  new Object[] {id}));
- Builder b4Get =
  wt.request(MediaType.APPLICATION_XML);
- Myuser_ClientSide myuser =
  b4Get.get(Myuser_ClientSide.class);

Example (RESTful Client – my version)

```
 * @author elau
 */
public class MyuserRESTClient {

    private WebTarget webTarget;
    private Client client;
    private static final String BASE_URI = "http://localhost:8080/ED-Myuser-RS-WAR/webresources";

    public MyuserRESTClient() {
        client = ClientBuilder.newClient();
        /**
         * build the web target using the BASE_URI and append the path with
         * "entity.myuser"
         *
         * Note: '/' is assumed when append a new sub-path
         */
        webTarget = client.target(BASE_URI).path("entity.myuser");
    }
public void myDisplayUser(String userid) {
    // set the web target to "{id}" for getting record
    WebTarget webTarget4Get = webTarget.path(MessageFormat.format("{0}", new Object[]{userid}));
    // build a request that accepts XML
    Builder builder4Get = webTarget4Get.request(MediaType.APPLICATION_XML);
    // accept a Myuser_ClientSide object;
    Myuser_ClientSide myuser = builder4Get.get(Myuser_ClientSide.class);

    System.out.println("=== User Info ===");
    myuser.displayAllInfo();
    System.out.println("=== End user info ===");
}
```

Example (RESTful Client – NB8.2)

```
 * @author elau
 */
public class MyuserRESTClient {

    private WebTarget webTarget;
    private Client client;
    private static final String BASE_URI = "http://localhost:8080/ED-Myuser-RS-WAR/webresources";

    public MyuserRESTClient() {
        client = ClientBuilder.newClient();
        /**
         * build the web target using the BASE_URI and append the path with
         * "entity.myuser"
         *
         * Note: '/' is assumed when append a new sub-path
         */
        webTarget = client.target(BASE_URI).path("entity.myuser");
    }
  public <T> T find(Class<T> responseType, String id) throws ClientErrorException {
        // set the web target to "{id}" for getting record - substitute the actual id
        WebTarget webTarget4Get = webTarget.path(MessageFormat.format("{0}", new Object[]{id}));
        // build a request that accepts XML
        Builder builder4Get = webTarget4Get.request(MediaType.APPLICATION_XML);
        // accepts the responsType;
        T result = builder4Get.get(responseType);
        return result;
  }
```

```
public void displayUser(String userid) {
    Myuser_ClientSide myuser = this.find(Myuser_ClientSide.class, userid);
    if (myuser == null) {
        System.out.println("No such user whose id is " + userid);
        return;
    }
    System.out.println("=== User Info ===");
    myuser.displayAllInfo();
    System.out.println("=== End user info ===");
}
```

Example (RESTful Client – Java Type)

```java
/**
 *
 * @author elau
 */
@XmlRootElement(name = "Myuser")
@XmlAccessorType(XmlAccessType.FIELD)
public class Myuser_ClientSide {

    @XmlElement(required=true)
    private String userid;
    @XmlElement(required=true)
    private String name;
    @XmlElement(required=true)
    private String password;
    @XmlElement(required=true)
    private String email;
    @XmlElement(required=true)
    private String phone;
    @XmlElement(required=true)
    private String address;
    @XmlElement(required=true)
    private String secqn;
    @XmlElement(required=true)
    private String secans;

    public Myuser_ClientSide() {
    }

    public String getUserid() {
        return userid;
    }

    public void setUserid(String userid) {
        this.userid = userid;
    }
```

References

- Wikipedia – REpresentational State Transfer (en.wikipedia.org/wiki/Representational_state_transfer)