Proyecto TIC: Desarrollo e implantación

RESTful Web Services
Java EE.

Xavi Piñol
xavi.pinyol@gmail.com

Index

Index

RESTful. Introduction.

RESTful web services are built to work best on the Web. Representational State Transfer (REST) is an architectural style that specifies constraints.

In the REST architectural style, data and functionality are considered resources and are accessed using Uniform Resource Identifiers (URIs), typically links on the Web. The resources are acted upon by using a set of simple, well-defined operations

RESTful. Introduction.

The REST architectural style constrains an architecture to a client/server architecture and is designed to use a **stateless** communication protocol, typically HTTP.

Index

RESTful principles.

The following principles encourage RESTful applications to be simple, lightweight, and fast:

Resource identification through URI: A RESTful web service exposes a set of resources that identify the targets of the interaction with its clients. Resources are identified by URIs, which provide a global addressing space for resource and service discovery.

RESTful principles.

Uniform interface: Resources are manipulated using a fixed set of four create, read, update, delete operations: PUT, GET, POST, and DELETE. PUT creates a new resource, which can be then deleted by using DELETE. GET retrieves the current state of a resource in some representation. POST transfers a new state onto a resource.

RESTful principles.

Self-descriptive messages: Resources are decoupled from their representation so that their content can be accessed in a variety of formats, such as HTML, XML, plain text, PDF, JPEG, JSON, and others.

RESTful principles.

Stateful interactions through hyperlinks: Every interaction with a resource is stateless; that is, request messages are self-contained. Stateful interactions are based on the concept of explicit state transfer. Several techniques exist to exchange state, such as URI rewriting, cookies, and hidden form fields. State can be embedded in response messages to point to valid future states of the interaction.

Index

Creating – Root Resource Class

Root resource classes are POJOs that are either annotated with @Path or have at least one method annotated with @Path or a request method designator, such as @GET, @PUT, @POST, or @DELETE. Resource methods are methods of a resource class annotated with a request method designator.

Index

Creating – Annotations.

Summary of JAX-RS Annotations

@ApplicationPath: Identifies the application path that serves as the base URI for all resource URIs provided by Path. May only be applied to a **subclass of Application.**

@Path: is a relative URI path indicating where the Java class will be hosted. You can also embed variables in the URIs to make a URI path template. /helloworld/{username}

Creating – Annotations.

@GET, @PUT, @POST, @DELETE: is a request method designator and corresponds to the similarly named HTTP method. The Java method annotated with this request method designator will process HTTP requests. The behavior of a resource is determined by the HTTP method to which the resource is responding .

Creating – Annotations.

@QueryParam: is a type of parameter that you can extract for use in your resource class. Query parameters are extracted from the request URI query parameters.

@Consumes: is used to specify the MIME media types of representations a resource can consume that were sent by the client.

@Produces: is used to specify the MIME media types of representations a resource can produce and send back to the client.

Creating – Annotations.

Values of MediaType:

APPLICATION_ATOM_XML
"application/atom+xml"

APPLICATION_FORM_URLENCODED
"application/x-www-form-urlencoded"

APPLICATION_JSON

"application/json"

APPLICATION_OCTET_STREAM
"application/octet-stream"

Creating – Annotations.

APPLICATION_SVG_XML
"application/svg+xml"

APPLICATION_XHTML_XML
"application/xhtml+xml"

APPLICATION_XML

"application/xml"

MULTIPART_FORM_DATA
"multipart/form-data"

Creating – Annotations.

TEXT_HTML

"text/html"

TEXT_PLAIN

"text/plain"

TEXT_XML

"text/xml"

WILDCARD

"*/*"

Index

Creating - Examples

Example Get:

```
@Path("/helloworld")
public class HelloWorldResource {
    @GET
    @Produces("text/plain")
    public String getClichedMessage()
     { return "Hello World"; }
}
```

Creating - Examples

Example Post:

```
@POST
@Consumes(MediaType.TEXT_PLAIN)
public void postClichedMessage(String
message) {
    // Store the message
}
```

Creating - Examples

Example with parameters:

```
@Path("/users/{username}")
public class UserResource {
  @GET
  @Produces("text/xml")
  public String getUser
(@PathParam("username") String userName)
 {  ...  }
}
```