**Institute of Technology, Tralee**

**Computing Department**

Distributed Computing – RestFul Web Services Lab

Representational State Transfer (REST) is a software architecture style that centers around the transmission of data over HTTP, using only the four basic HTTP verbs. It also eschews the use of any additional wrappers such as a SOAP envelope and the use of any state data. The goal of this lab is to show the student how to develop a RESTful web service using JAX-RS

The advantages of RESTful systems is that they are highly scalable and highly flexible. Because the resources are accessed and manipulated using the four HTTP verbs, the resources are exposed using a URI, and the resources are represented using standard grammars, clients are not as affected by changes to the servers. Also, REST style systems can take full advantage of the scalability features of HTTP such as caching and proxies.

RESTful architectures adhere to the following basic principles: Application state and functionality are divided into resources. Resources are addressable using standard URIs that can be used as hypermedia links. All resources use only the four HTTP verbs.

DELETE

GET

POST

PUT

All resources provide information using the MIME types supported by HTTP. The protocol is stateless. The protocol is cacheable. The protocol is layered.

*Resources* are central to REST. A resource is a source of information that can be addressed using a URI. In the early days of the Web, resources were largely static documents. In the modern Web, a resource can be any source of information. For example a Web service can be a resource if it can be accessed using a URI.

RESTful endpoints exchange *representations* of the resources they address. A representation is a document containing the data provided by the resource. For example, the method of a Web service that provides access to a customer record would be a resource, the copy of the customer record exchanged between the service and the consumer is a representation of the resource.

When designing RESTful services it is helpful to keep in mind the following:

* Provide a distinct URI for each resource you wish to expose.

For example, if you are building a system that deals with driving records, each record should have a unique URI. If the system also provides information on parking violations and speeding fines, each type of resource should also have a unique base. For example, speeding fines could be accessed through /speeding/*driverID* and parking violations could be accessed through /parking/*driverID*.

* Use nouns in your URIs.

Using nouns highlights the fact that resources are things and not actions. URIs such as /ordering imply an action, whereas /orders implies a thing.

* Methods that map to GET should not change any data.
* Use links in your responses.

Putting links to other resources in your responses makes it easier for clients to follow a chain of data. For example, if your service returns a collection of resources, it would be easier for a client to access each of the individual resources using the provided links. If links are not included, a client needs to have additional logic to follow the chain to a specific node.

* Make your service stateless.

Requiring the client or the service to maintain state information forces a tight coupling between the two. Tight couplings make upgrading and migrating more difficult. Maintaining state can also make recovery from communication errors more difficult.

**Exercise 1**

Investigate a live REST service

Look at the following link to get some experience of how restful web services work

<https://api.predic8.de/shop/docs>

**Exercise 2**

Investigate OpenAPI via Swagger Editor

Look at the following link to the Swagger Editor and play around with it (pet id 100, pet status available)

<https://editor.swagger.io/>

**Exercise 3**

Build your own REST service

Do the following trails in Java – do the first one only. Use Java Web EE 7 as the framework for the Glassfish server. I haven’t tested the second trail with Netbeans 11 – try it if you wish but not required.

You need to download and install Glassfish server 5 and latest Netbeans. Unzip Glassfish to a folder on your C: drive. Point Netbeans at this server when requested for a server.

<http://netbeans.org/kb/docs/websvc/rest.html>

(<https://netbeans.org/kb/72/websvc/jersey-rcp-client.html>)

4. Do the Swagger Hello World exercise in the Swagger Hello World folder

**Optional Exercises**

1. Do the IntelliJ RESTFUL Web Service example (<https://www.jetbrains.com/help/idea/restful-webservices.html>)
2. Look at Visual Studio tutorials for creating RESTful web services

<http://www.c-sharpcorner.com/UploadFile/0c1bb2/creating-wcf-rest-service/>

1. Look at Strip API (<https://stripe.com/docs/api>) – used by previous students Naz and Ciaran as good example of their own API

SpringBoot (project done by Naz and Ciaran for Fexco: <http://intranet.fexcosoftware.com:8443/#/> to allow customers request leave, used Hibernate back end, Spring Boot for Web Services etc) <https://spring.io/guides/gs/actuator-service/>

1. Could you develop a service called reverseWord as a RESTful web service? What would you need to do?
2. Compare and contrast SOAP v RESTful Web Services – homework