REST

WEB SERVICE

Web Service:

A method of communication between two electronic devices over a network.
A software function provided at a network address over the web, with the service always on.

Used to exchange data

W3C definition of web service:

***A software system designed to support interoperable machine-to-machine interaction over a network*.**

ALTERNATIVES

SOAP

Simple Object Access Protocol
Protocol for sending and receiving messages
XML

WSDL

Web Service Description Language
Model for describing web services
XML

REST

REpresentational State Transfer web service
Architectural style for web resources
JSON

RESTFUL WEB SERVICE

REpresentational State Transfer
Has become one of the most important technologies for web applications.
REST Server provides access to resources and REST client accesses and modifies the resources.
Resources can be text files, html pages, images, videos or dynamic business data and each resource is identified by URLs.
API oriented / Maintainable / Scalable / Light weight communication
HTTP methods are used to extract / manipulate resources: GET, POST, PUT, DELETE

Representational
Clients possess the information necessary to identify, modify, and/or delete a web resource.
A resource can consist of other resources.
Resources are represented by a format and identified using URLS.
Resources can be represented by formats, such as json and xml.
Both client and server should be able to comprehend communication format.
State
No state information is stored on the server / All state information is stored on the client
Transfer
Client state is passed from the client to the service through HTTP

Architectural constraints
Client-Server

- The clients and the server are separated from each other:
- The client is not concerned with the data storage thus the portability of the client code is improved

- The server is not concerned with the client interference, thus the server is simpler and easy to scale.

Stateless

- Each request can be treated independently.
- REST interactions store no client context on the server between requests.
- All information necessary to service the request is contained in the URL, query parameters, body or headers.
- The client holds session state.

Cacheable

- The responses must define themselves as, cacheable or not, to prevent the client from sending the inappropriate data in response to further requests.
- Caching can be controlled using HTTP headers.

Uniform interface

- The uniform interface constraint is fundamental to the design of any REST service.
- The uniform interface simplifies and decouples the architecture.
- Each resource has at least one URI.

Layered System (Resources are decoupled from their representation)

- At any time clients cannot tell if they are connected to the end server or to an intermediate.
- Neither can clients see (and should not consider), the technologies used to implement a REST API
- When resources are decoupled from their representation their content can be accessed in a variety of formats.

REST API URL EXAMPLE

To create a new customer:

POST - HTTP://WWW.EXAMPLE.COM/CUSTOMERS

To read, update and delete a customer with Customer ID# 33245:

GET - HTTP://WWW.EXAMPLE.COM/CUSTOMERS/33245
PUT - HTTP://WWW.EXAMPLE.COM/CUSTOMERS/33245
DELETE - HTTP://WWW.EXAMPLE.COM/CUSTOMERS/33245

To create a new product:

POST HTTP://WWW.EXAMPLE.COM/PRODUCTS

JAX-RS

JAX-RS stands for JAVA API for RESTful Web Services.
JAX-RS is a Java programming language API designed to make it easy to develop applications that use the REST architecture.
The JAX-RS API uses Java programming language annotations to simplify the development of RESTful web services.
JAX-RS annotations are runtime annotations; therefore, runtime reflection will generate the helper classes and artifacts for the resource.

JERSEY

Open source framework for implementing RESTful web services.
Consist of components such as core server, core client, JAXB support and JSON support.
JAX-RS -> Specification
JERSEY -> Implementation

REST IN NETBEANS

1. New maven web application
2. Add dependency org.glassfish.jersey.bundles : jax-rs-ri
3. Add dependency com.google.code.gson : gson
4. Create package rest
5. Create new restful web service from pattern (json)

CONNECTING JPA

1. Add dependency mysql-connector-java
2. Create database
3. Create connection to database
4. Add persistence unit
5. Create entity, facade, facadeinterface
6. Generate schema
7. Use facade to connect restful web service and jpa

APPLICATIONCONFIG

Automatically updated with resource classes when adding new restful web services

ApplicationPath (Where to access server root REST API)

ANNOTATIONS

Annotations are used in the restful web service to configure the communication between server and client.

ApplicationPath (Server root REST API path)

```
@javax.ws.rs.ApplicationPath("api")
```

Path (Resource path) - Can be used both on complete class and individual methods

```
@Path("person/all")
```

Get / Post/ Put / Delete (HTTP method used)

```
@GET / @POST / @PUT / @DELETE
```

Produces / Consumes (Body type: JSON / XML / HTML)

MediaType.APPLICATION_JSON / MediaType.APPLICATION_XML / MediaType.TEXT_HTML

```
@Consumes(MediaType.APPLICATION_JSON)
@Produces(MediaType.APPLICATION_JSON)
```

PathParam (Path parameters)

```
@Path("all/{id}")
@PathParam("id") int id
```

QueryParam (Query parameters)

```
?job=None
@QueryParam("job") String job
```

DefaultValue (Default values if missing)

```
@DefaultValue("Worker")
```

Context

```
@Context
private UriInfo context;
                System.out.println(context.getQueryParameters().toString());
                System.out.println(context.getQueryParameters().get("id"));
@Context
private HttpHeaders headers
                System.out.println(headers.getMediaType());
SecurityContext / Request / …
```

Send JSON Method

```
@POST
@Consumes(MediaType.APPLICATION_JSON)
public void postJson(String content)
{}
```

JSON

```
{ "name":"John", "age": 31, "member": false }
```

CONVERSION (GSON / JSON)

When working with REST, there is a need to convert json string / java object, when communication occurs.

From json string to java object

```
String json1 = "{\"firstName\":\"Ole\",\"lastName\":\"Olsen\",\"phoneNumber\":12345678}";
Person p1 = new Gson().fromJson(json1, Person.class);
Json property names must match class property names
```

To json string from java object

```
Person p2 = new Person("Mads", "Madsen", 87654321);
String json2 = new Gson().toJson(p2);
```

Creating a json object

```
JsonObject jo3 = new JsonObject();
jo3.addProperty("firstName", "Hans");
jo3.addProperty("lastName", "Hansen");
jo3.addProperty("phoneNumber", 11223344);
jo3.addProperty("id", 999);
Person p3 = new Gson().fromJson(jo3, Person.class);
```

JsonParser

```
JsonObject body = new JsonParser().parse(json1).getAsJsonObject();
System.out.println(body.has("fName"));
System.out.println(body.has("firstName"));
System.out.println(body.get("firstName").getAsString());
System.out.println(body.get("id").getAsInt());
```

JsonArray to json string

```
JsonArray ja = new JsonArray();
ja.add(jo3);
ja.add(jo3);
String jsonArrayString = new Gson().toJson(ja);
```

Java list to json string

```
ArrayList<Person> Persons = new ArrayList();
Persons.add(p1);
Persons.add(p2);
Persons.add(p3);
String json = new Gson().toJson(Persons);
```

JPA <-> FACADE <-> REST <-> JQUERY

Make connection from JPA entities to facades and use the facades in REST web services
AJAJ

getJson / ajax

Array / Prototype -> Table / Listeners

POSTMAN

Postman can be used to send, receive and inspect responses to urls.
Method / Headers / Body