# 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:

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

To create a new customer:

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

To get, 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

**org.glassfish.jersey.bundles : jaxrs-ri : 2.27**

implements JAX-RS 2.1 API

## REST IN NETBEANS

1. New maven web application
2. Add dependency org.glassfish.jersey.bundles : jaxrs-ri
3. Create new package rest
4. Create new restful web service from pattern (Simple root resource -> application/json)

## 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("p1") String job
```

DefaultValue (Default values if missing)

```
@DefaultValue("Nothing")
```

Context

```
@Context
private UriInfo uriInfo;
        System.out.println(uriInfo.getQueryParameters().toString());
        System.out.println(uriInfo.getQueryParameters().get("p1"));
@Context
private HttpHeaders httpHeaders
        System.out.println(httpHeaders.getMediaType());
SecurityContext / Request / …
```

Content (json / body)

```
@POST
@Consumes(MediaType.APPLICATION_JSON)
public void postJson(String json) { System.out.println(json) }
```

Response

```
return Response.ok("{}").build();
return Response.status(Response.Status.ACCEPTED).entity("{}").build();
```

## JSON

```
{ "name":"John", "age": 31, "member": false, “addresses”:[{“First Road 1”, “New York”},{“Second Road 2”, “Berlin”}] }
```

## GSON

When working with REST, there is a need to convert between json strings and java objects
Conversion between Java objects and JSON strings

**com.google.code.gson : gson : 2.8.5**

## GSON IN NETBEANS

1. Add dependency com.google.code.gson : gson

JSON Converting & JSON Parsing

*fromJson / toJson / JsonObject / JsonParser / JsonArray*

Creating a Gson object

```
Gson gson = new GsonBuilder().setPrettyPrinting().create();
```

Using Gson object

```
gson.fromJson(String);
gson.toJson(Object);
```

From json string to java object (Json property names must match class case sensitive property names)

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

To json string from java object

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

Creating a json object

```
JsonObject jo1 = new JsonObject();
Jo1.addProperty("firstName", "Hans");
Jo1.addProperty("lastName", "Hansen");
Jo1.addProperty("phoneNumber", 11223344);
Jo1.addProperty("id", 999);
Person p3 = new Gson().fromJson(jo1, Person.class);
```

Creating a JsonParser object

```
JsonParser jsonParser = new JsonParser();
```

Using JsonParser object

```
JsonObject jo2 = jsonParser.parse(json1).getAsJsonObject();
System.out.println(jo2.has("firstName"));
System.out.println(jo2.has("fName"));
System.out.println(jo2.get("firstName").getAsString());
System.out.println(jo2.get("id").getAsInt());
```

JsonArray to json string

```
JsonArray ja = new JsonArray();
ja.add(jo2);
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);
```

## BIDIRECTIONAL RELATIONSHIPS

Person has arraylist / object of type Address AND Address has arraylist / object of type Person

Problem when using objects with bidirectional relationships in new Gson().toJson(object)…

Alternative1: Add transient to reference in entity class

```
private transient List<Address> addresses = new ArrayList();
```

Alternative2: Create DTO class for entity class

Create DTO class with constructor and use typed query to map entity class to DTO class

## POSTMAN

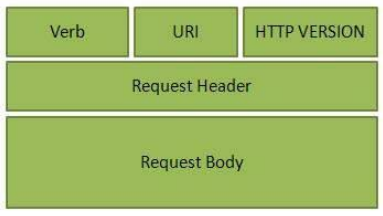
Postman can be used to define and send request to URLs, plus receive and inspect responses

URL / Method / Headers / Body

Responses / Status codes

## HTTP

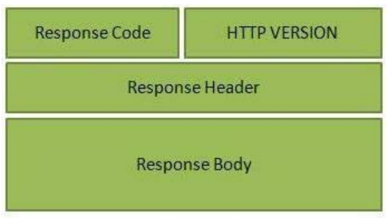
### HTTP REQUEST



HTTP Request

- Verb
  Indicate HTTP methods such as GET, POST, DELETE, PUT etc.
- URI
  Uniform Resource Identifier (URI) to identify the resource on server
- HTTP Version
  Indicate HTTP version, for example HTTP v1.1 .
- Request Header
  Contains metadata for the HTTP Request message as key-value pairs. For example, client (or browser) type, format supported by client, format of message body, cache settings etc.
- Request Body
  Message content or Resource representation.

### HTTP Response



HTTP Response

- Status/Response Code
  Indicate Server status for the requested resource. For example 404 means resource not found and 200 means response is ok.
- HTTP Version
  Indicate HTTP version, for example HTTP v1.1 .
- Response Header
  Contains metadata for the HTTP Response message as key-value pairs. For example, content length, content type, response date, server type etc.
- Response Body
  Response message content or Resource representation.

### HTTP status codes

**SUCCESSFUL**
**200, OK**
**201, Created**
**202, Accepted**
**204, No Content**
**205, Reset Content**
**206, Partial Content**
**REDIRECTION**
**301, Moved Permanently**
**302, Found**
**303, See Other**
**304, Not Modified**
**305, Use Proxy**
**307, Temporary Redirect**

**CLIENT_ERROR**
**400, Bad Request**
**401, Unauthorized**
**402, Payment Required**
**403, Forbidden**
**404, Not Found**
**405, Method Not Allowed**
**406, Not Acceptable**
**407, Proxy Authentication Required**
**408, Request Timeout**
**409, Conflict**
**410, Gone**
**411, Length Required**
**412, Precondition Failed**
**413, Request Entity Too Large**
**414, Request-URI Too Long**
**415, Unsupported Media Type**
**416, Requested Range Not Satisfiable**
**417, Expectation Failed**

**SERVER_ERROR**
**500, Internal Server Error**
**501, Not Implemented**
**502, Bad Gateway**
**503, Service Unavailable**
**504, Gateway Timeout**
**505, HTTP Version Not Supported**

## ERRORS / EXCEPTIONS

Wrong requests / Bad URLs / Exceptions thrown / Many others

Map java exceptions to http error responses and create responses with errors
Create error responses via exceptions and map existing exceptions to error responses
Errors can be reported to client by…

Creating and returning response object / Throwing exception

REST error handling: Take existing exception and map it to http response with status code and json error message
Thrown exceptions are handled by JAX-RS if an exception mapper has been registered

Use json for error messages / Use http status codes

```
return Response.status(Response.Status.LENGTH_REQUIRED).build();
return Response.status(404).build();
return Response.status(500).entity(gson.toJson(err)).type(MediaType.APPLICATION_JSON).build();
```

Exception handling in can be done with in several different ways

1: throw WebApplicationException
2: ExceptionDTO
3: ExceptionMapper

Only show original error / exception / stack trace in debug mode
Web.xml can be used to declare debug mode

Create web.xml in Web Pages/ WEB-INF/ folder
Add context parameter debug and set to either true or false

```
<context-param>
   <param-name>debug</param-name>
   <param-value>false</param-value>
</context-param>
```

Check debug mode

```
context.getInitParameter("debug").equals("true");
```

## TESTING

### JUNIT

Add dependencies…

junit : junit 4.12
org.hamcrest : hamcrest.core 1.3

- Tools -> Create / Update tests
- Set up junit tests

### DERBY

In memory database for testing
Add dependency…

org.apache.derby : derby 10.14.2.0

Add persistence unit for testing with subpackage value

```
<property name="eclipselink.canonicalmodel.subpackage" value="test"/>
```

## TESTING REST ENDPOINTS

Unit tests / Integration tests
Test early / Test often
Purpose: Show no errors exist / Find errors
Develop test cases / expectations / results

Execute tests and correct code
Maven requires all tests to be passed in order to build project.
Basic junit tests do not require server to be running
REST-Assured junit tests require server to be running

  Server cannot be running before project is compiled
  Project cannot be compiled unless server is running.

Plugins can be added in pom.xml to use an embedded tomcat server

## REST-ASSURED

REST Assured can be used together with Junit
Simple Java library for programmatically performing requests up against REST services with focus on testing
Supports XML and JSON Request / Responses
Supports POST, GET, PUT, DELETE, OPTIONS, PATCH and HEAD http methods
Specify and validate parameters, headers, cookies and body easily
given()

  What payload our method is going to have ( JSON data representing a Student )
  What kind of ContentType our payload has ( XML, PlainText, JSON, … )
  What kind of authentication we are having ( basic, token, Oauth, … )

when()

  The method we are going to call as well as the path
  POST, GET, PUT, DELETE, OPTIONS, PATCH or HEAD
  URL path ( /user, /api/person/add, … )

then()

  After we executed our method, we need to test on the results we get back
  Statuscode ( 200OK, 401NotFound, … )
  Messages returned - JSON, XML

### REST-ASSURED IN NETBEANS

1. Add dependency io.rest-assured : rest-assured 3.1.1
2. Add plugins to pom.xml
3. Create junit tests with rest assured code

Plugins for pom.xml…

```
<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-surefire-plugin</artifactId>
<version>2.22.0</version>
<configuration>
<excludes>
<exclude>**/*IntegrationTest*</exclude>
</excludes>
</configuration>
</plugin>

<plugin>
<groupId>org.apache.maven.plugins</groupId>
<artifactId>maven-failsafe-plugin</artifactId>
<version>2.22.0</version>
<configuration>
<argLine>-Dfile.encoding=UTF-8</argLine>
<includes>
<include>**/*IntegrationTest*</include>
</includes>
</configuration>
<executions>
<execution>
<goals>
<goal>integration-test</goal>
<goal>verify</goal>
</goals>
</execution>
</executions>
</plugin>
```

```
<plugin>
<groupId>org.apache.tomcat.maven</groupId>
<artifactId>tomcat7-maven-plugin</artifactId>
<version>2.2</version>
<configuration>
<port>7777</port>
<path>/</path>
</configuration>
<executions>
<execution>
<id>start-tomcat</id>
<phase>pre-integration-test</phase>
<goals>
<goal>run</goal>
</goals>
<configuration>
<fork>true</fork>
</configuration>
</execution>
<execution>
<id>stop-tomcat</id>
<phase>post-integration-test</phase>
<goals>
<goal>shutdown</goal>
</goals>
</execution>
</executions>
</plugin>
```