



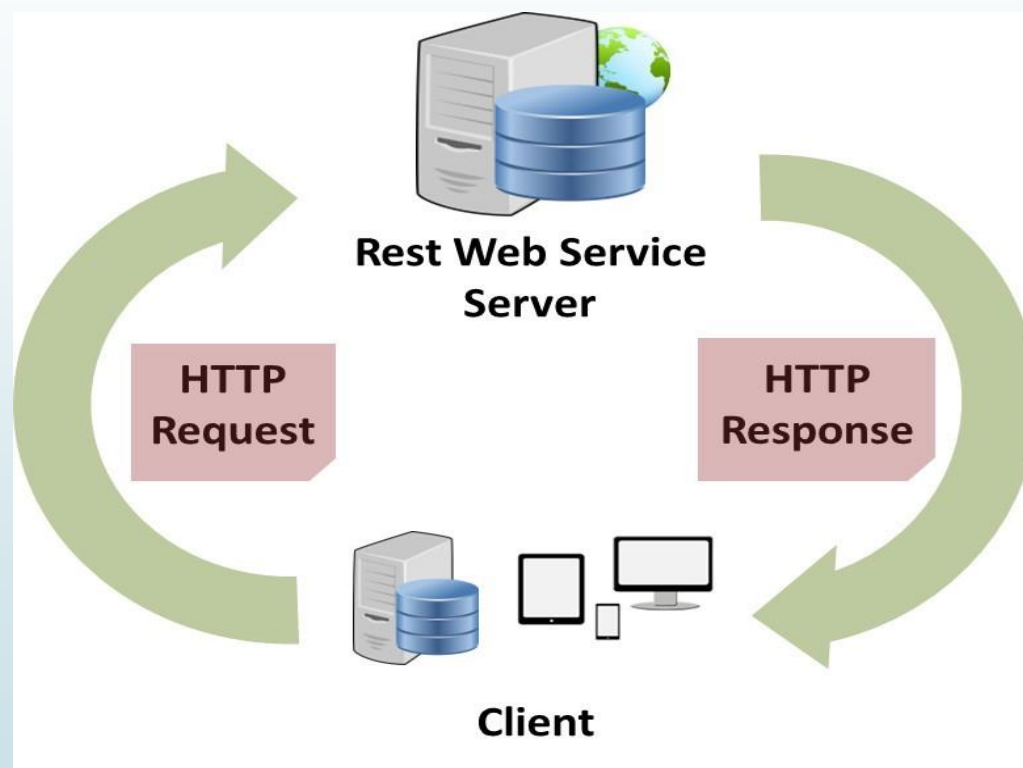
RESTful Web Services



What is REST?

- REST stands for **RE**presentational **S**tate **T**ransfer.
- REST is an architecture principle for data transmission based on Web-standards and HTTP protocol.
- There are resources which are hosted using REST server and can be accessed by a common interface using HTTP standard methods.
- REST allows representation of a resource in Text, JSON and XML formats.
- RESTful Web Services make use of HTTP protocols as a medium of communication between client and server.

How it works...



A client sends a message in form of a HTTP Request and the server responds in the form of an HTTP Response.



What is a Resource?

- REST architecture treats every content as a resource.
- A resource in REST is a similar Object in Object Oriented Programming or is like an Entity in a Database.
- Once a resource is identified then its representation is to be decided using a standard format so that the server can send the resource in the above said format and client can understand the same format.

Resource Representation

▶ JSON

```
{  
  "id":1,  
  "name":"Peter",  
  "age":45,  
  "profession":"Teacher"  
}
```

▶ XML

```
<person>  
  <id>1</id>  
  <name>Peter</name>  
  <age>45</age>  
  <profession>Teacher</profession>  
</person>
```

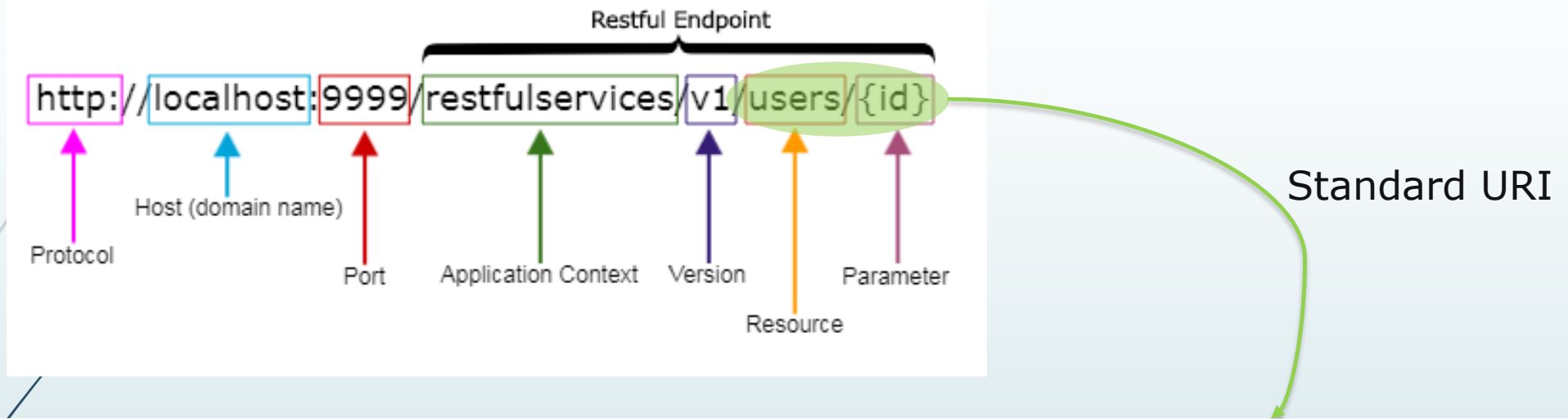


HTTP Methods

The following HTTP methods are most commonly used in a REST based architecture.

- **GET** – Provides a read only access to a resource.
- **PUT** – Used to create a new resource.
- **DELETE** – Used to remove a resource.
- **POST** – Used to update an existing resource or create a new resource.
- **OPTIONS** – Used to get the supported operations on a resource.

Addressing Endpoint – Methods



Users API for users in the system		⌵
POST	/users Create a new user in system	
GET	/users Get all users in system	
GET	/users/{userId} Get user with given ID	
DELETE	/users/{userId} Delete user with given ID	
PUT	/users/{userId} Update user with give ID	



HTTP Request

- **Verb** – Indicates the HTTP methods such as GET, POST, DELETE, PUT, etc.
- **URI** – Uniform Resource Identifier (URI) to identify the resource on the server.
- **HTTP Version** – Indicates the 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 the client, format of the message body, cache settings, etc.
- **Request Body** – Message content or Resource representation.

Verb

URI

HTTP Version

Request Header

Request Body

Last Response Data

Request Response Status: 201 + Time: 1024 ms +

Headers (6) Body Raw

POST http://localhost:5000/api/users HTTP/1.1

Content-Type: application/json
User-Agent: Telerik Test Studio for APIs
Host: localhost:5000
Content-Length: 66
Expect: 100-continue
Connection: Keep-Alive

```
{
  "userName": "jsmith",
  "email": "john.smith@example.com"
}
```



HTTP Response

- **Status/Response Code** – Indicates the Server status for the requested resource. For example, 404 means resource not found and 200 means response is ok.
- **HTTP Version** – Indicates the 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.

Last Response Data

Request Response Status: 201 + Time: 1024 ms +

Headers (5) Body Raw

HTTP/1.1 201 Created

Transfer-Encoding: chunked
Content-Type: application/json; charset=utf-8
Date: Fri, 09 Jun 2017 22:11:19 GMT
Location: /api/users/299fcf2c-9098-4ab6-aff8-2ac8a68d7cb7
Server: Kestrel

```
{ "avatarImage": null, "avatarImagePath": null, "avatarImageContentType": null, "id": "299fcf2c-9098-4ab6-aff8-2ac8a68d7cb7", "userName": "jsmith", "normalizedUserName": "JSMITH", "email": "john.smith@example.com", "normalizedEmail": "JOHN.SMITH@EXAMPLE.COM", "emailConfir
```

HTTP Version

Status/Response Code

Response Header

Request Body

HTTP Status Codes

HTTP defines forty standard status codes that can be used to convey the results of a client's request.

CATEGORY	DESCRIPTION
1xx: Informational	Communicates transfer protocol-level information.
2xx: Success	Indicates that the client's request was accepted successfully.
3xx: Redirection	Indicates that the client must take some additional action in order to complete their request.
4xx: Client Error	This category of error status codes points the finger at clients.
5xx: Server Error	The server takes responsibility for these error status codes.




Developing RESTful Web Services with JAX-RS




JAX-RS

- JAX-RS is a Java programming language API designed to make it easy to develop applications that use the REST architecture.
- JAX-RS API uses Java programming language annotations to simplify the development of RESTful web services.
- JAX-RS annotations define the resources and the actions that can be performed on those resources.
- **javaee-api-*.*.jar**



Annotation	Description
@GET	The @GET annotation 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 GET requests. The behavior of a resource is determined by the HTTP method to which the resource is responding.
@POST	The @POST annotation 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 POST requests. The behavior of a resource is determined by the HTTP method to which the resource is responding.
@PUT	The @PUT annotation 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 PUT requests. The behavior of a resource is determined by the HTTP method to which the resource is responding.
@DELETE	The @DELETE annotation 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 DELETE requests. The behavior of a resource is determined by the HTTP method to which the resource is responding.
@HEAD	The @HEAD annotation 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 HEAD requests. The behavior of a resource is determined by the HTTP method to which the resource is responding.



Annotation	Description
@Path	The @Path annotation's value is a relative URI path indicating where the Java class will be hosted: for example, /helloworld. You can also embed variables in the URIs to make a URI path template. For example, you could ask for the name of a user and pass it to the application as a variable in the URI: /helloworld/{username}.
@PathParam	The @PathParam annotation is a type of parameter that you can extract for use in your resource class. URI path parameters are extracted from the request URI, and the parameter names correspond to the URI path template variable names specified in the @Path class-level annotation.
@QueryParam	The @QueryParam annotation 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	The @Consumes annotation is used to specify the MIME media types of representations a resource can consume that were sent by the client.
@Produces	The @Produces annotation is used to specify the MIME media types of representations a resource can produce and send back to the client: for example, "text/plain".
@Provider	The @Provider annotation is used for anything that is of interest to the JAX-RS runtime, such as MessageBodyReader and MessageBodyWriter. For HTTP requests, the MessageBodyReader is used to map an HTTP request entity body to method parameters. On the response side, a return value is mapped to an HTTP response entity body by using a MessageBodyWriter. If the application needs to supply additional metadata, such as HTTP headers or a different status code, a method can return a Response that wraps the entity and that can be built using Response.ResponseBuilder.

The following code sample is a very simple example of a root resource class that uses JAX-RS annotations:

```
import javax.ws.rs.GET;  
import javax.ws.rs.Produces;  
import javax.ws.rs.Path;
```

```
// The Java class will be hosted at the URI path "/helloworld"
```

```
@Path("/helloworld")
```

```
public class HelloWorldResource {
```

```
    // The Java method will process HTTP GET requests
```

```
    @GET
```

```
    // The Java method will produce content identified by the MIME Media
```

```
    // type "text/plain"
```

```
    @Produces("text/plain")
```

```
    public String getClichedMessage() {
```

```
        // Return some cliched textual content
```

```
        return "Hello World";
```

```
    }
```

```
}
```