

Distribution and Integration Technologies

RESTful Services



REST style for web services

- ❖ **REST – Representational State Transfer, considers the web as a data resource**
 - Services accesses and modifies this data (CRUD operations)
 - This model and style was first described in 2000 in a thesis called “Architectural Styles and the Design of Network-based Software Architectures” (by Roy Fielding)
 - Usual implementations uses Uri templates for naming the resources and data items
 - Ex: `http://finance.google.com/finance/info?q={stock_nick}`
 - `http://some_organization/CRM/customer/{id}`
 - Operations are performed using the HTTP protocol operation verbs. HTTP results are also important
 - GET, POST, PUT, DELETE, HEAD
 - Information can be transfered in a variety of formats (representations)
 - HTML, XML, JSON, RSS, CSV, ...

Operation verbs

❖ GET

- Retrieve the resource identified by the URI

❖ POST

- Send (to create) a resource to the server identified by the URI

❖ PUT

- Store (to modify) a resource in the server using the supplied URI

❖ DELETE

- Removes the resource identified by the URI

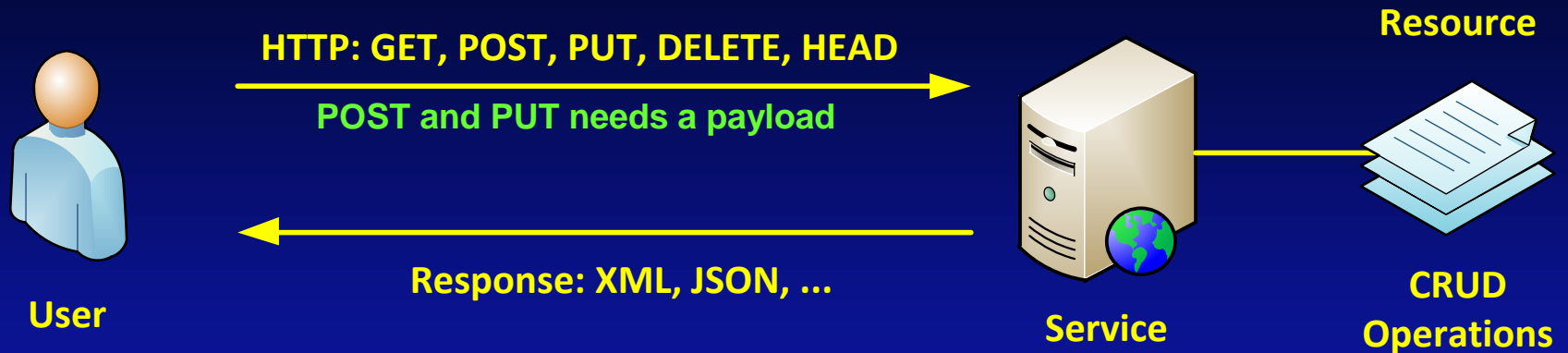
❖ HEAD

- Retrieve metadata (more information) about the resource identified by the URI

Information formats

- ❖ Usually simple formats, not supporting the complications of SOAP and their associated functionality
- ❖ The most used
 - XML in a simple schema
 - The services that use this format are also called POX (plain old XML) services
 - JSON (JavaScript Object Notation) used in literal representations of java script objects
 - RSS (Really Simple Syndication) and ATOM are content syndication formats and can also be used for services

REST operations



Clients:

- . Send the HTTP web request building the appropriate:
URI, using the operation template and parameters
In the case of a POST or PUT you need the payload, encoded appropriately
- . Collect the response from the server
- . Check the HTTP result
- . Decode the contents

REST services in .NET

Are built as the other WCF services

1. Start with a WCF Service Library project and define the contract and implementation
2. The operations in the contract must be associated with the HTTP operations using the attributes [WebGet] (for GET) and [WebInvoke] (for other HTTP operations)
3. You can define parameters in these attributes like 'UriTemplate', 'RequestFormat' and 'ResponseFormat'. Formats can be XML or JSON. Parameters in UriTemplate (between { }) must match method parameters
4. A [Description] attribute can be added for an automatic help page

Example:

```
[ServiceContract]
public interface IRestService {
    [WebGet(UriTemplate="/users", ResponseFormat=WebMessageFormat.Json)]
    [Description("Gets all users stored so far.")]
    [OperationContract]
    Users GetUsers();

    [WebInvoke(Method = "POST", UriTemplate = "/users", RequestFormat = WebMessageFormat.Json,
              ResponseFormat = WebMessageFormat.Json)]
    [Description("Adds one user.")]
    [OperationContract]
    int AddUser(User user);

    [WebInvoke(Method="DELETE", UriTemplate="/users/{id}")]
    [Description("Deletes one user by id.")]
    [OperationContract]
    void DeleteUser(string id);
}
```

Binding for REST services

REST Services use the existing **webHttpBinding** and a behavior specifying the communication (**webHttp**)

A minimal configuration file is:

```
<system.serviceModel>
  <services>
    <service name="RestService">
      <endpoint address="http://localhost:8700/Rest/" behaviorConfiguration="RestBehavior"
        binding="webHttpBinding"
        name="RestWebService" contract="IRestService" />
    </service>
  </services>
  <behaviors>
    <endpointBehaviors>
      <behavior name="RestBehavior">
        <webHttp helpEnabled="true" />
      </behavior>
    </endpointBehaviors>
  </behaviors>
</system.serviceModel>
```

Hosting a REST service

The host can be an application or the IIS server

In any case the class implementing the host should be **WebServiceHost** instead of simply **ServiceHost** (**WebServiceHost** already assumes the **webHttp** binding),

Example: A console application host

```
class RestHost {  
    static void Main() {  
        WebServiceHost host = new WebServiceHost( typeof(RestService) );  
        host.Open();  
        Console.WriteLine("Rest service running");  
        Console.WriteLine("Press ENTER to stop the service");  
        Console.ReadLine();  
        host.Close();  
    }  
}
```


A client for a REST service

The client can use a proxy or generate the request directly

This kind of service doesn't generate a description and because of that we can not build a proxy automatically

But it is simple to have one in a client (assuming a config file with a named endpoint):

```
...
ChannelFactory<IRestService> factory =
    new ChannelFactory<IRestService>("RestClient"); // client endpoint name

IRestService proxy = factory.CreateChannel();

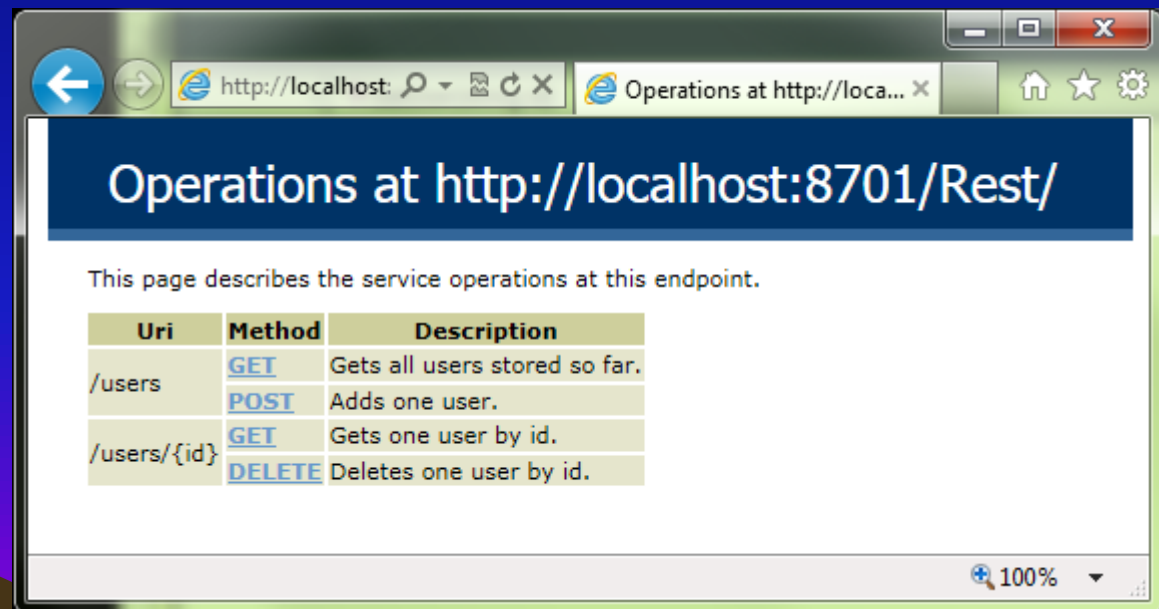
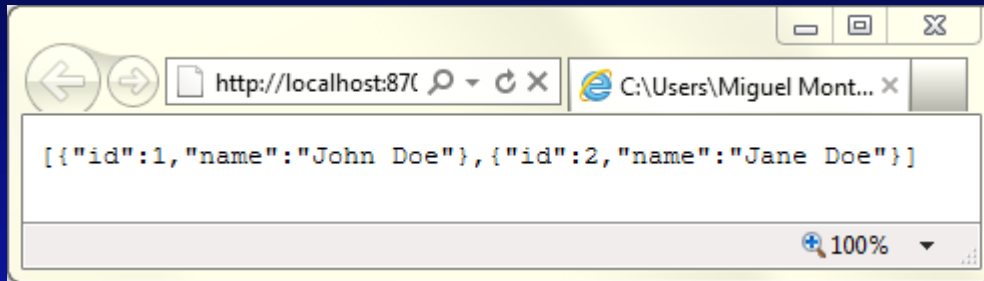
User user = proxy.GetUser("1");

...
```

REST service operation

GET methods can be invoked in a browser

Also the host can generate a help page (in the base address)



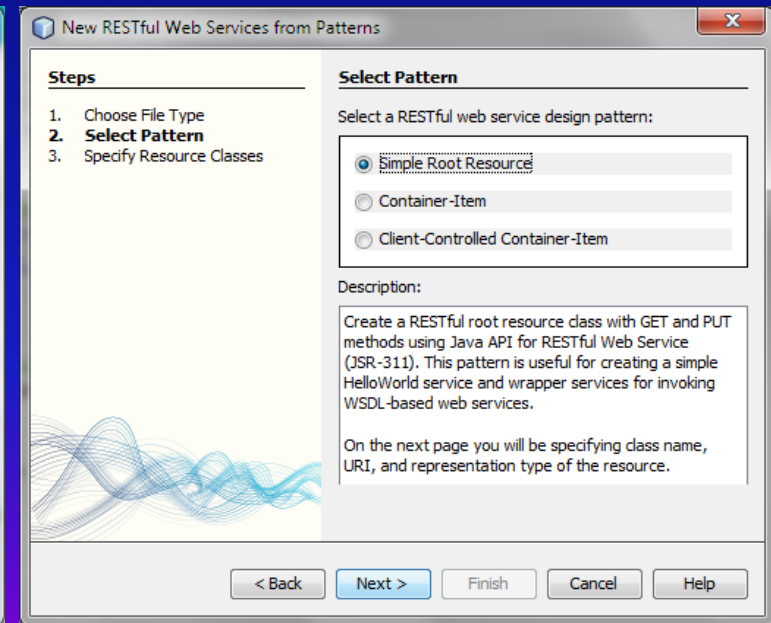
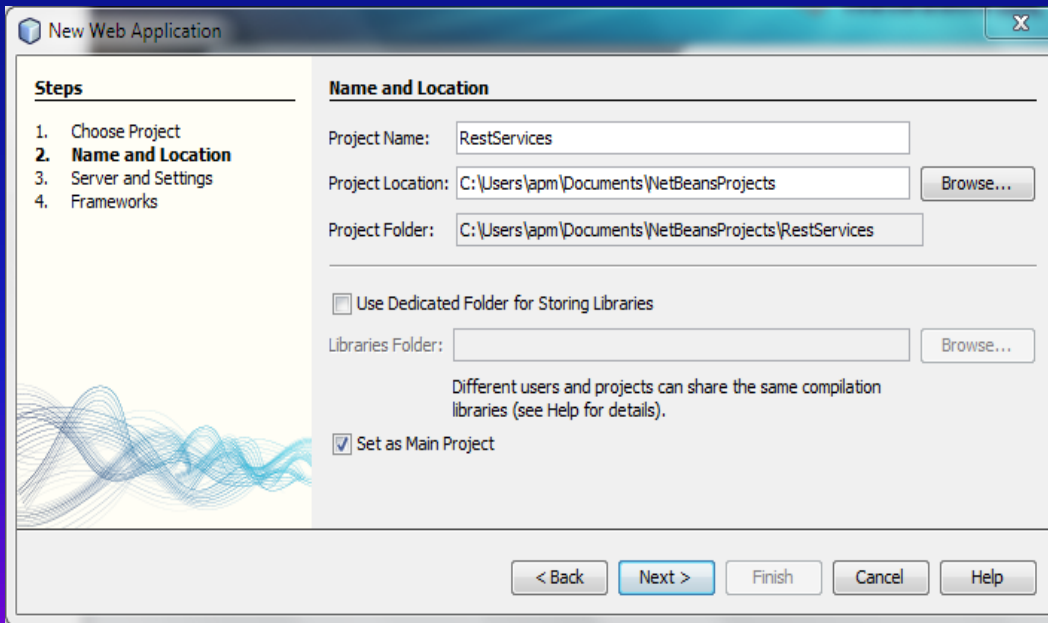
REST services in Java

- ❖ A standard specification for REST services (JAX-RS) is included in and after the specification Java EE 6
- ❖ The Jersey library is the reference implementation of JAX-RS
 - It's included in the Glassfish application server
 - The NetBeans IDE supports developing REST services with Jersey
 - A REST service is a POJO with a number of annotations specifying URIs, parameters, representations, operation verbs
 - In NetBeans is seen as a Web Application resource
 - REST services should be application stateless (no sessions), but not the resource they interact with
 - They can use other server resources (EJBs, Entities, ...) through dependency injection

Using NetBeans to create a REST service

❖ Create a Web Application in Glassfish

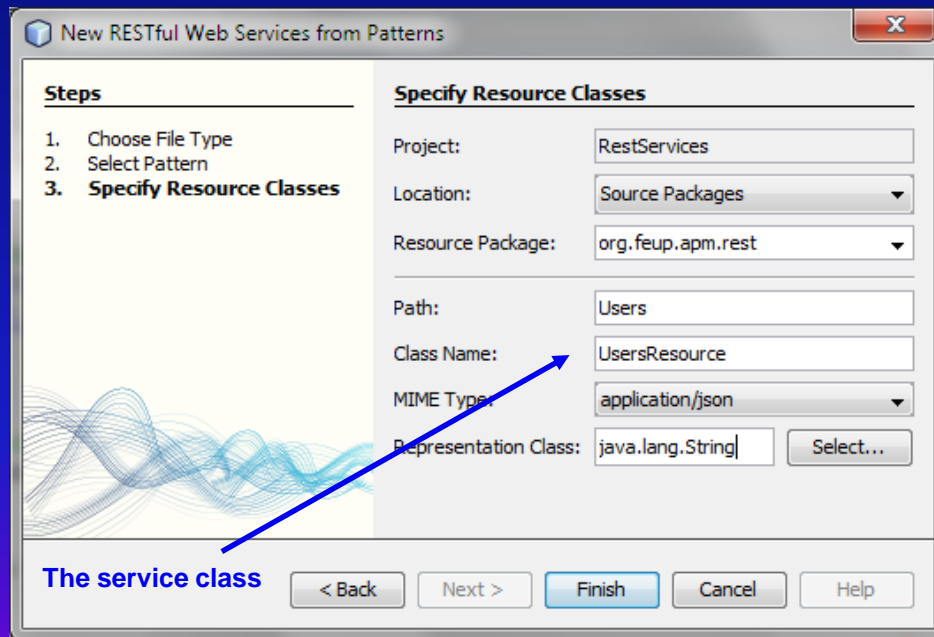
- You can use the web page to give some presentation and help about the service
- Add a new resource to the web application specifying a “RESTful Web Service from patterns ...” and a “Simple Root Resource”



REST service specification

❖ Specify the service inside the Web application

- You need a class for the operations (the service)
- Probably you will need also a class to represent the resource that the service will access



The dialog box 'New RESTful Web Services from Patterns' shows the 'Specify Resource Classes' step. A blue arrow points from the text 'The service class' to the 'Class Name' field, which contains 'UsersResource'.

Steps

1. Choose File Type
2. Select Pattern
3. **Specify Resource Classes**

Specify Resource Classes

Project: RestServices

Location: Source Packages

Resource Package: org.feup.apm.rest

Path: Users

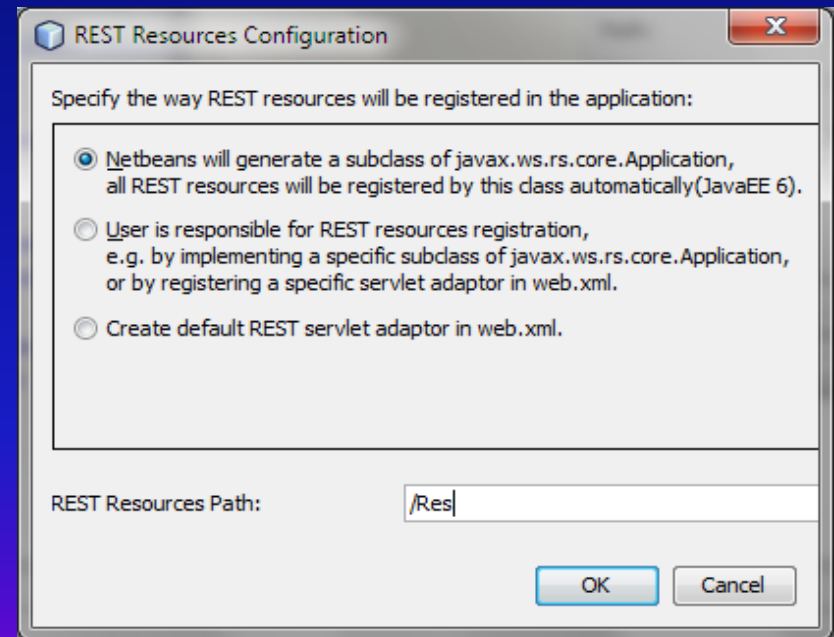
Class Name: UsersResource

MIME Type: application/json

Representation Class: java.lang.String [Select...]

The service class

< Back Next > Finish Cancel Help



The dialog box 'REST Resources Configuration' allows specifying how REST resources are registered. The first option is selected.

REST Resources Configuration

Specify the way REST resources will be registered in the application:

- ☒ Netbeans will generate a subclass of javax.ws.rs.core.Application, all REST resources will be registered by this class automatically (JavaEE 6).
- ☐ User is responsible for REST resources registration, e.g. by implementing a specific subclass of javax.ws.rs.core.Application, or by registering a specific servlet adaptor in web.xml.
- ☐ Create default REST servlet adaptor in web.xml.

REST Resources Path: /Res

OK Cancel

REST annotations

The class implementing the service (containing the operations as methods) must be annotated with `@Path(<base address>)`

The `<base address>` is a string specifying the base path to the resource represented by the service. This path is appended to the web application address + REST Resource (specified when we added the service to the web application)

```
@Path("users")
public class UsersResource {

    ... // service methods

}
```

The classes representing the resource or its components and appearing as a parameter or as a result of an operation (therefore transferred between clients and service) must be annotated with `@XmlElement`

```
@XmlElement
public class User {
    public int Id;
    public String Name;

    public User() {
    }

    public User(int id, String name) {
        Id = id;
        Name = name;
    }
}
```

Methods annotations

Service methods should be annotated with the http operation verb: **@GET**, **@POST**, **@PUT**, **@DELETE** or **@HEAD**

Also each method can have an additional path specification to add to the class path. This specification can include a template (between { })

Post and Put operations can have a payload: The format of that payload can be specified in a **@Consumes(<format>)** annotation and is assigned to a parameter of the service.

All operations can produce a result also encoded in a specified format in **@Produces(<format>)** annotation

Other parameters for the methods can be obtained from the path (**@PathParam(<template>)** annotation), from a parameter in the URI (appended like **?name=value**) (**@QueryParam(<name>)** annotation) or from a cookie (**@CookieParam(<name>)** annotation)

```
@Path("users")
public class UsersResource {

    @GET
    @Produces("application/json")
    public ArrayList<User> getUsers() {
        ...
    }

    @GET
    @Path("{id}")
    @Produces("application/json")
    public User getUser(@PathParam("id") String id) {
        ...
    }

    @POST
    @Consumes("text/plain")
    public Response addUser(String name) {
        ...
    }

    @DELETE
    @Path("{id}")
    public void deleteUser(@PathParam("id") String id) {
        ...
    }
}
```

Data formats and state

The most common supported formats that can be specified in the `@Produces` and `@Consumes` annotations include the following MIME types:

- application/xml
- application/json
- text/plain
- text/html
- text/xml
- image/*

We can use other resources in our REST service, like EJBs, including singletons. JAX-RS and the Jersey library support the resource injection available in Java EE application servers

```
//Stateful resource (singleton)
```

```
@Singleton
public class UsersStorage {
    ...
}
```

```
@Stateless           // needed for dependency injection
@Path("users")
public class UsersResource {

    @Context           // from the context call
    private UriInfo uriInfo;

    @EJB               // the singleton EJB
    private UsersStorage listOfUsers;

    ....
}
```