AKADEMIA GÓRNICZO-HUTNICZA
IM. STANISŁAWA STASZICA W KRAKOWIE

Systemy rozproszone

RESTful Services
Laboratorium

Bartosz Kwolek, Marek Psiuk

Katedra Informatyki AGH

At background

- **Launch Spring Tool Suite (C:\STS) (on PCoIP terminal)** *
  - **Available at: https://spring.io/tools/sts**

- **If you prefer IntelliJ:**
  - **BYOD (i.e. your own laptop)**
  - **instal ULTIMATE version and configure it for WebApplication development**
    - **short instruction at the end of this presentation**

- **Create and open FRESH workspace in chosen IDE**
- **If you want to keep your work, copy your code after the class**

This presentation is available at Moodle

Rules of the game

- 5 simple exercises based on the **JAX-RS specification** and **Jersey implementation**

- ~~Your presence gives you 1 initial point~~
- ~~Exercises gives you up to 1,5 point (ex. 1: 1.5 pt. / ex.2-3: 1 pt. / ex.4: 0.5 pt.)~~
- ~~Total: 5 points~~

REST - SHORT RECAP

What is REST?
REpresentational State Transfer

- REST is an architectural style based on web standards and protocols
- In REST, everything is a resource accessible via a common interface based on HTTP (ROA)
- In REST, typically there is:
  - Server (application server) providing access to resources
  - Client (web browser/client app) accessing and modifying the REST resources
- Resource:
  - Is **identified** by URI (addressable)
  - Supports HTTP **operations** (typically: GET, PUT, POST, DELETE)
  - Can be **represented** in different ways: text, xml, json (client can require specific format)

Typical HTTP operations used in REST

HTTP Verb	Common Meaning	Safe	Idempotent
GET	Retrieve the current state of the resource	YES	YES
POST	Create a sub-resource	NO	NO
PUT	Initialize or update the state of a resource at given URI	NO	YES
DELETE	Clear a resource	NO	YES

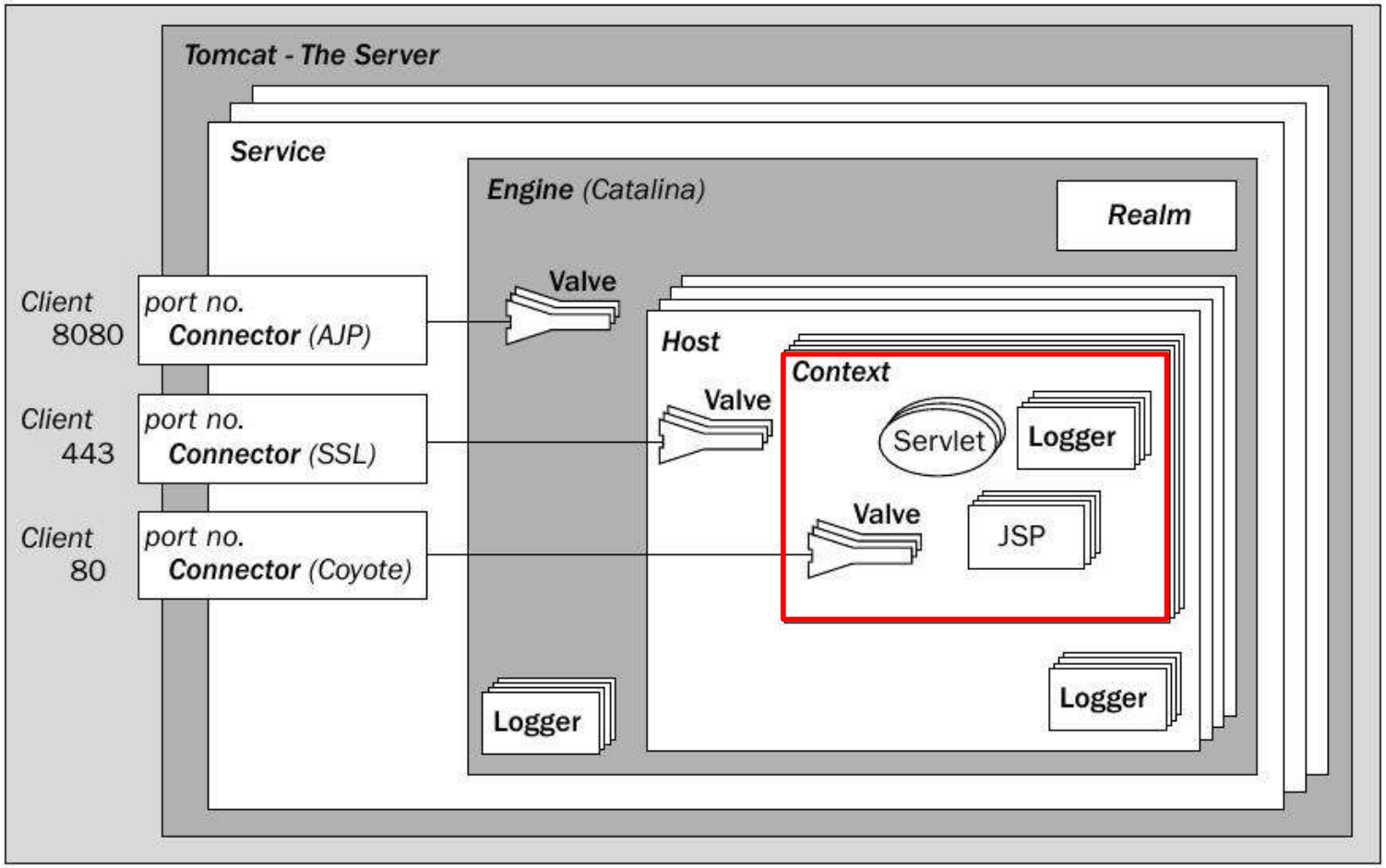
REST in Java

- Java defines REST support via JSR-311 called JAX-RS (The Java API for RESTful Web Services).
- JAX-RS helps exposing RESTful services by annotating Java classes
- **Jersey**, chosen for this laboratory, is a reference JAX-RS implementation (supports server and client)
- On the server side, Jersey exposes services through servlet (to be configured in *web.xml*)
- *web.xml* configuration points to packages, which are scanned for annotated Java classes
- A Jersey servlet analyzes incoming HTTP requests and selects the corresponding annotated class and method
- JAX-RS supports XML and JSON through JAXB

JAX-RS annotations

Annotation	Description
@PATH(your_path)	Sets the path to base URL + /your_path. The base URL is based on your application name, the servlet and the URL pattern from the web.xml" configuration file.
@POST	Indicates that the following method will answer to a HTTP POST request
@GET	Indicates that the following method will answer to a HTTP GET request
@PUT	Indicates that the following method will answer to a HTTP PUT request
@DELETE	Indicates that the following method will answer to a HTTP DELETE request
@Produces(MediaType.TEXT_PLAIN [, more-types])	@Produces defines which MIME type is delivered by a method annotated with @GET. In the example text ("text/plain") is produced. Other examples would be "application/xml" or "application/json".
@Consumes(type [, more-types])	@Consumes defines which MIME type is consumed by this method.
@PathParam	Used to inject values from the URL into a method parameter. This way you inject for example the ID of a resource into the method to get the correct object.

Servlet? Web server? Servlet container?

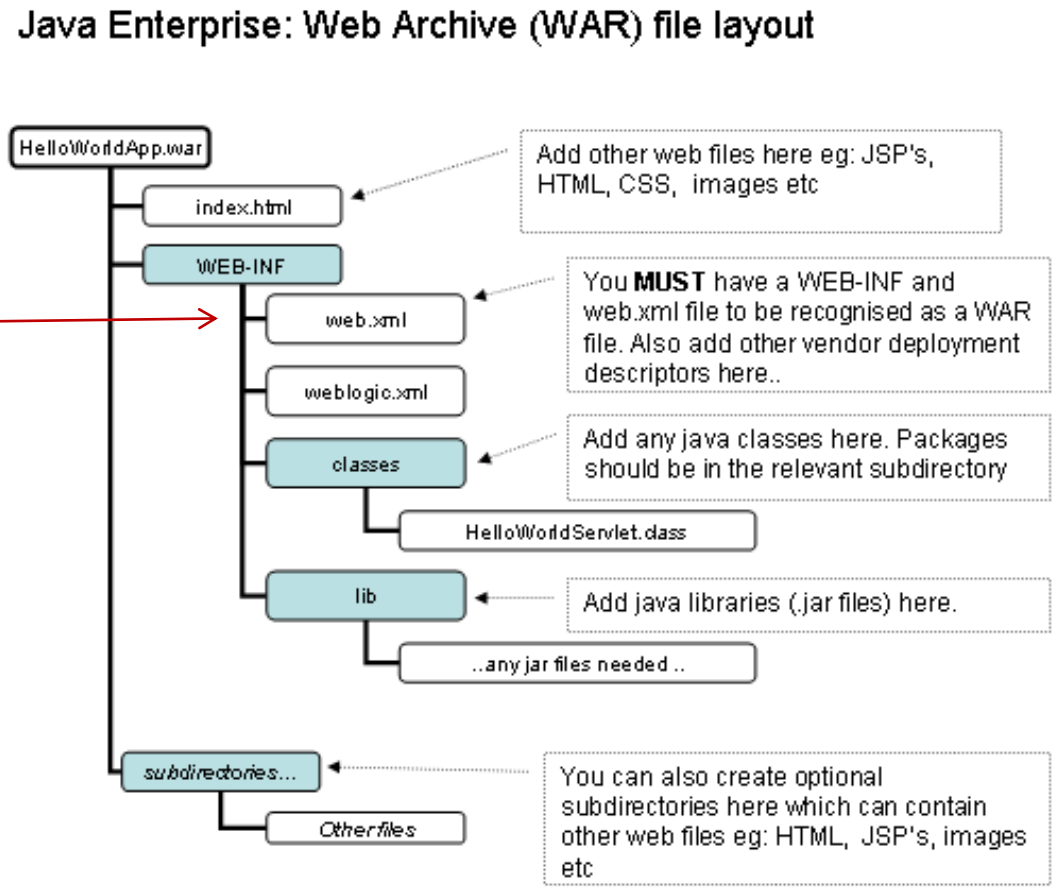


Tomcat's architecture.

http://bhujjikutty.blogspot.com/2010/06/tomcat-architecture-5.html

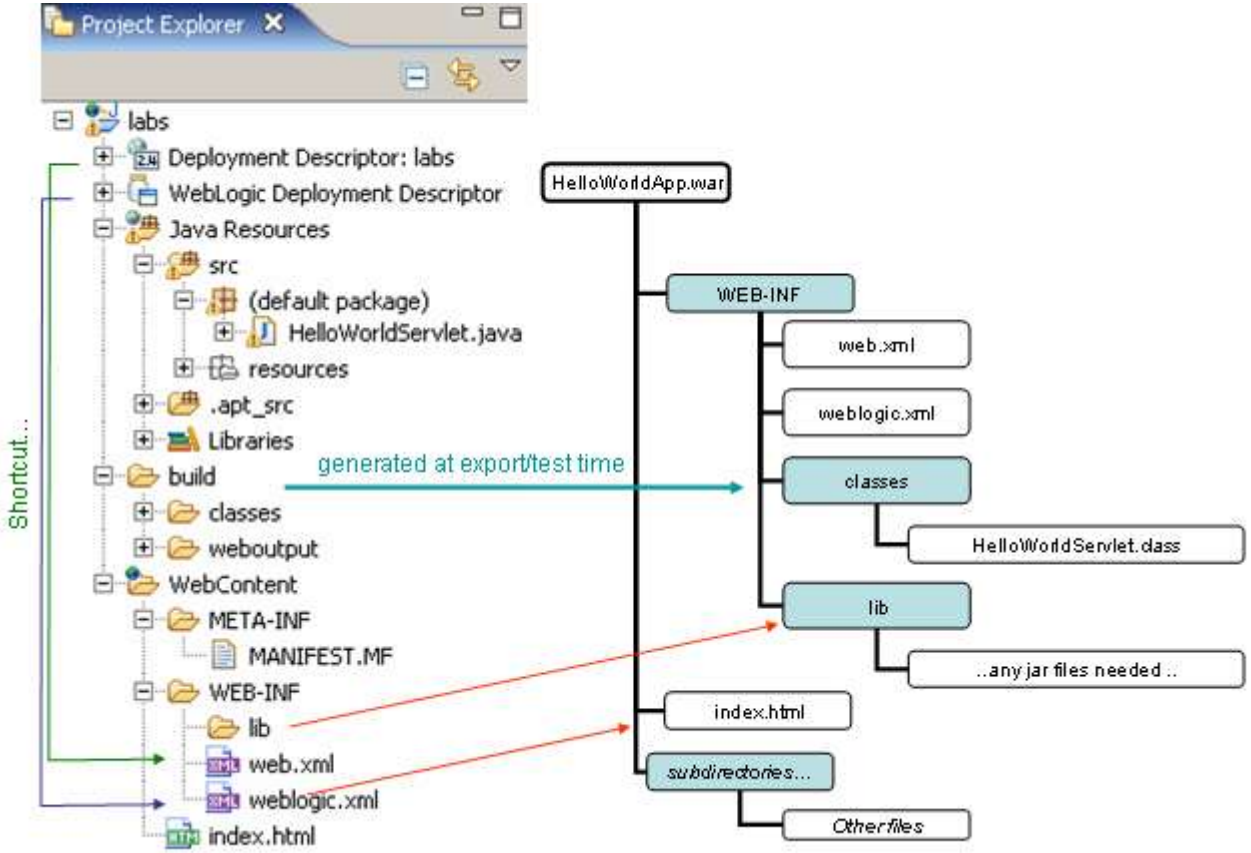
Web application (.war structure)

Deployment descriptor



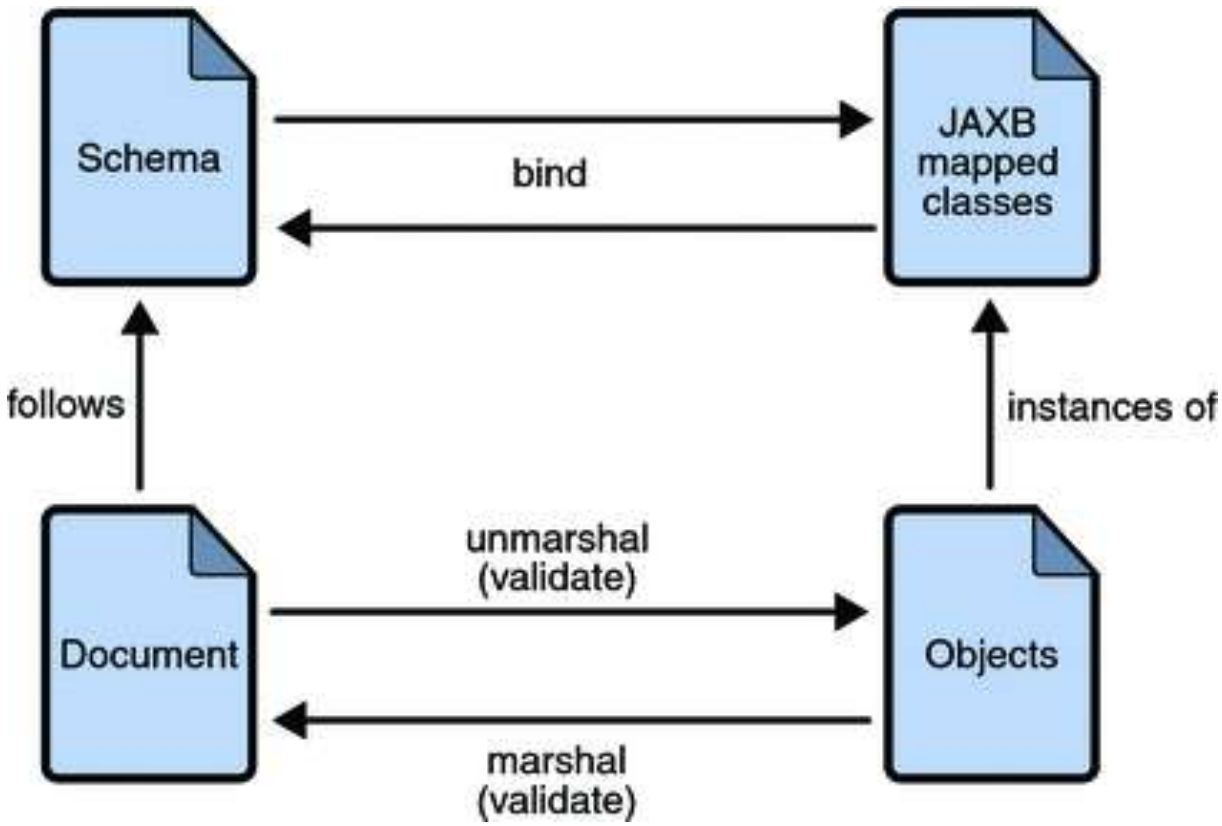
https://learn.it.uts.edu.au/aip/enrolled/03-servlets/lab-helloservlet.htm

Web application (.war structure)



https://learn.it.uts.edu.au/aip/enrolled/03-servlets/lab-helloservlet.htm

JAXB



https://docs.oracle.com/javase/tutorial/jaxb/intro/arch.html

EXCERCISES

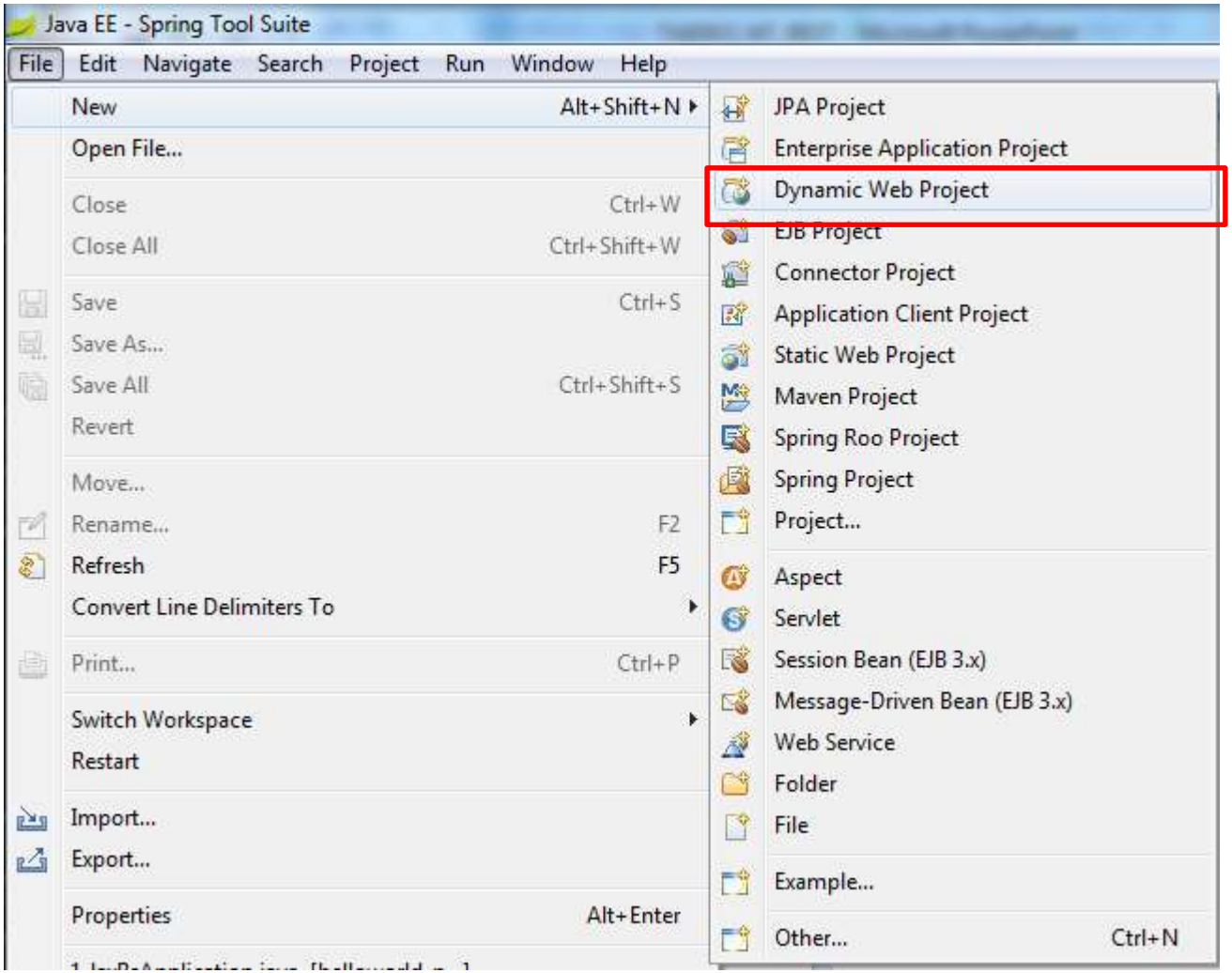
EXERCISES

1. Fundamentals:
   - Dynamic Web Project with *web.xml*,
   - Server,
   - JAX-RX annotations,
   - tracing resource requests
     (web browser and stand-alone client)
2. JAXB binded class
3. Simple CRUD (not only HTTP GET)
4. Client application for ex. 3
5. Inspection of requests with TCP/IP Monitor

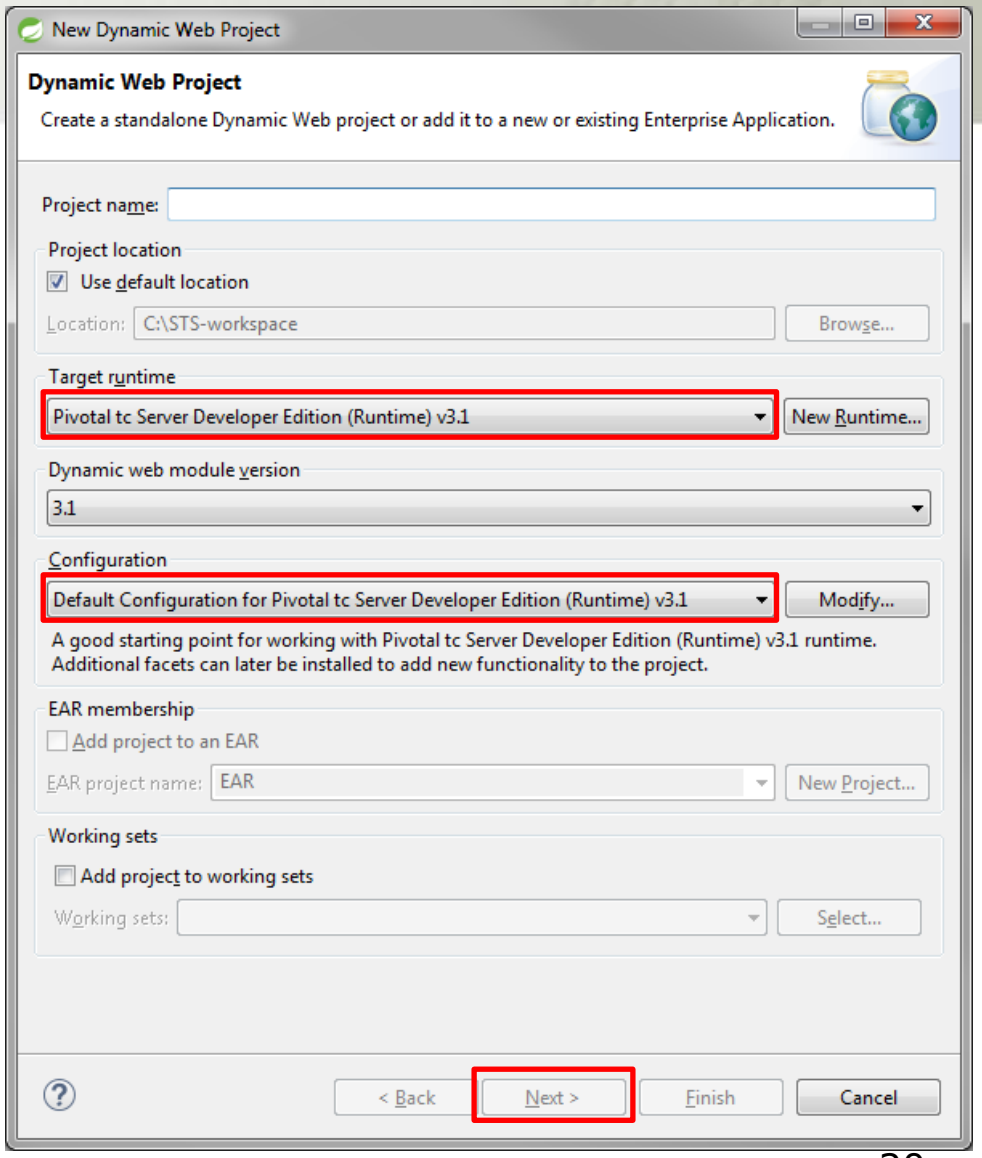
fundamentals

EXERCISE 1

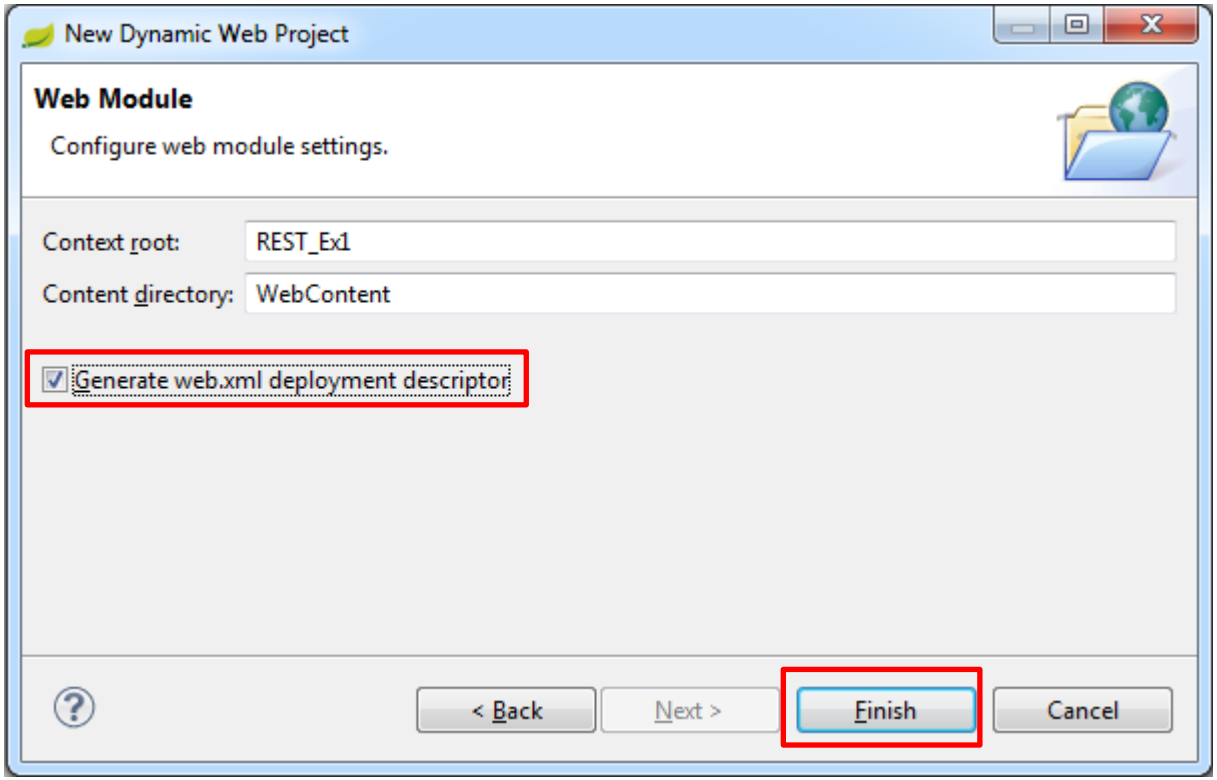
Exercise 1
Create Project



Exercise 1
Create Project

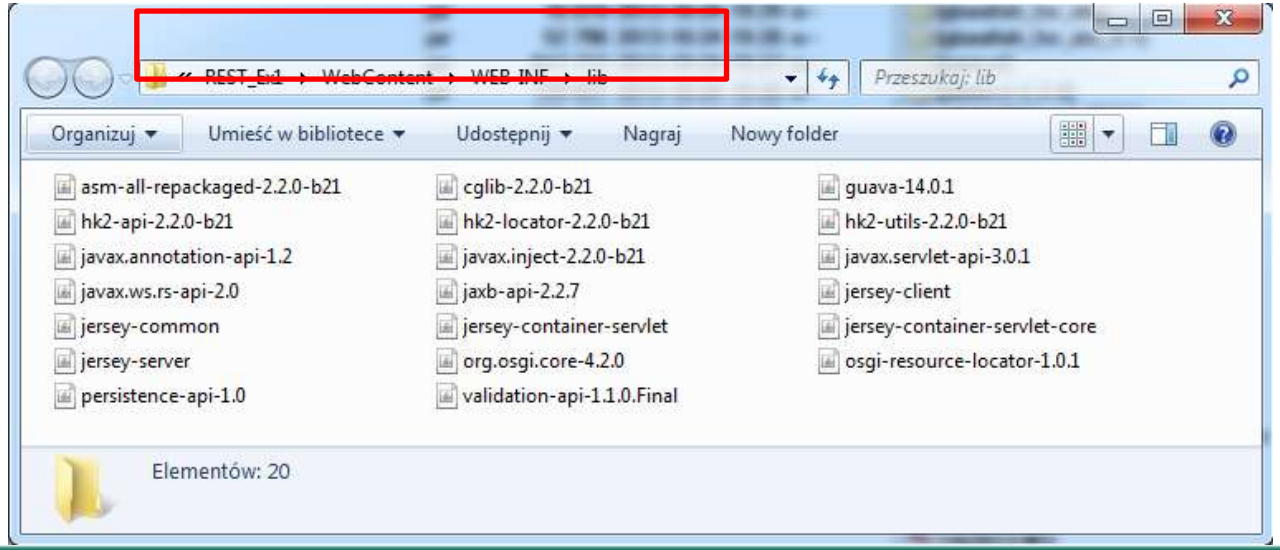


Exercise 1
Create Project

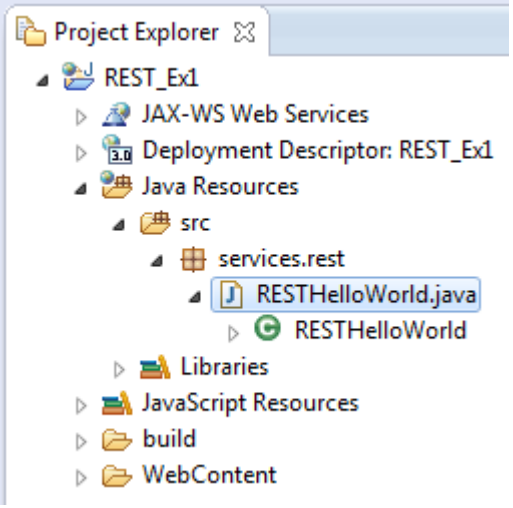


Exercise 1
Add Jersey Dependencies

- Download Jersey from: https://jersey.github.io/download.html
  (core Jersey module jars + required 3rd-party dependencies)
- Unzip all JARs to the project lib folder
- Add genson.jar https://owlike.github.io/genson



Exercise 1
Resource Implementation
CopyPaste Friendly



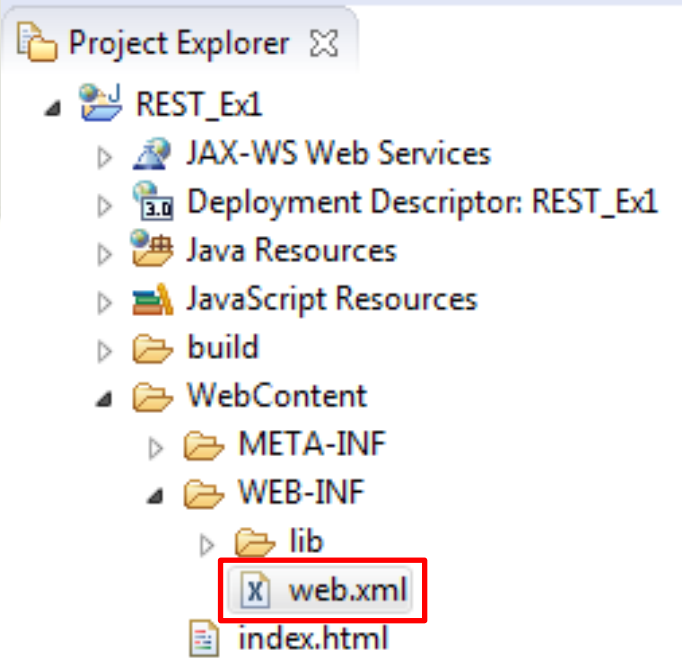

```
public class RESTHelloWorld {

        public String sayPlainTextHello() {
                return "Hello Jersey";
        }

        public String sayXMLHello() {
                return "<?xml version=\"1.0\"?>" + "<hello> Hello Jersey" + "</hello>";
        }

        public String sayHtmlHello() {
                return "<html> " + "<title>" + "Hello Jersey" + "</title>"
                                + "<body><h1>" + "Hello Jersey" + "</body></h1>" + "</html> ";
        }

}
```

Exercise 1
Resource Implementation

RESTHelloWorld.java

```java
package services.rest;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

@Path("/hello")
public class RESTHelloWorld {

    // This method is called if TEXT_PLAIN is request
    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String sayPlainTextHello() {
        return "Hello Jersey";
    }

    // This method is called if XML is request
    @GET
    @Produces(MediaType.TEXT_XML)
    public String sayXMLHello() {
        return "<?xml version=\"1.0\"?>" + "<hello> Hello Jersey" + "</hello>";
    }

    // This method is called if HTML is request
    @GET
    @Produces(MediaType.TEXT_HTML)
    public String sayHtmlHello() {
        return "<html> " + "<title>" + "Hello Jersey" + "</title>"
                + "<body><h1>" + "Hello Jersey" + "</body></h1>" + "</html> ";
    }

}
```

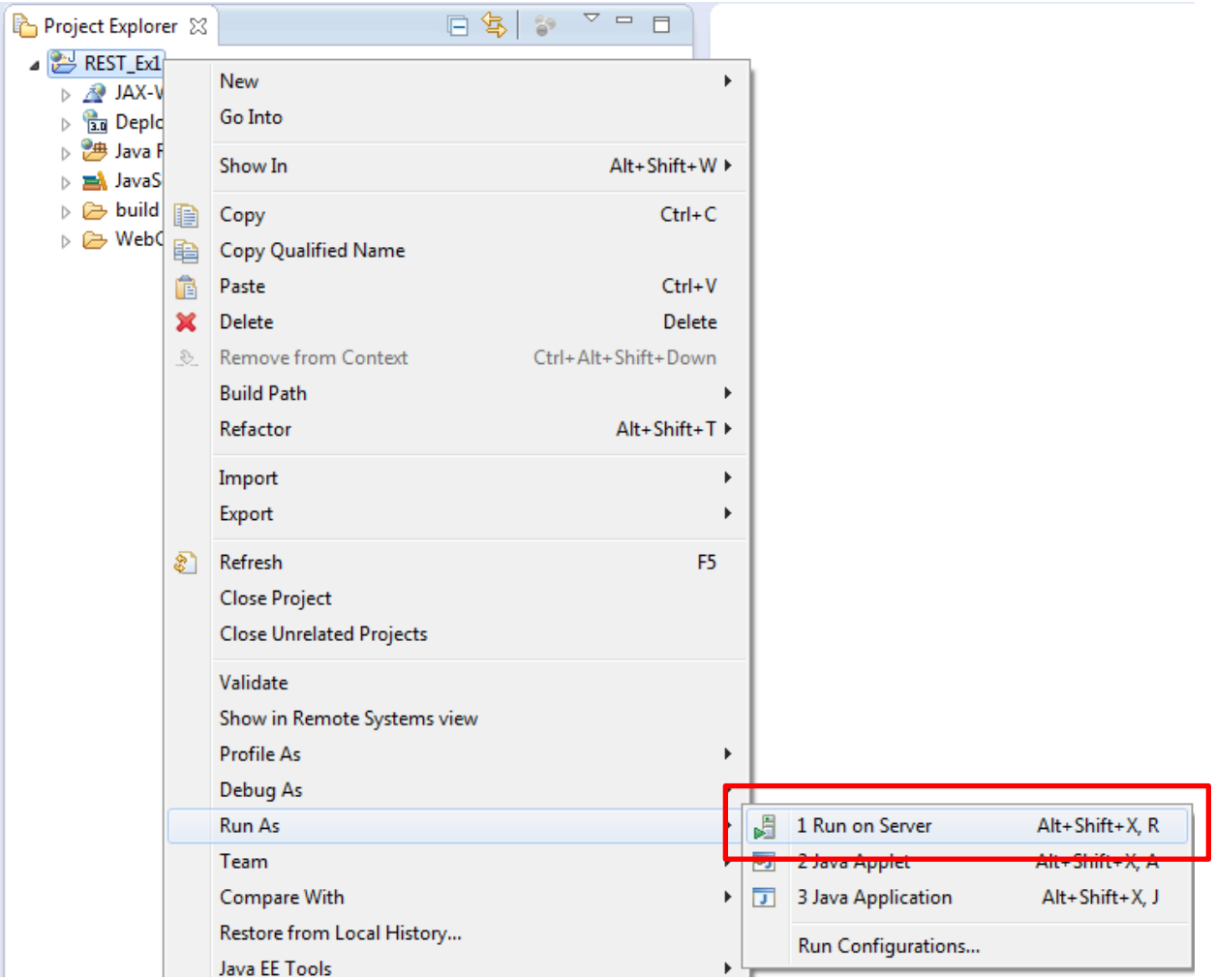
Exercise 1
web.xml



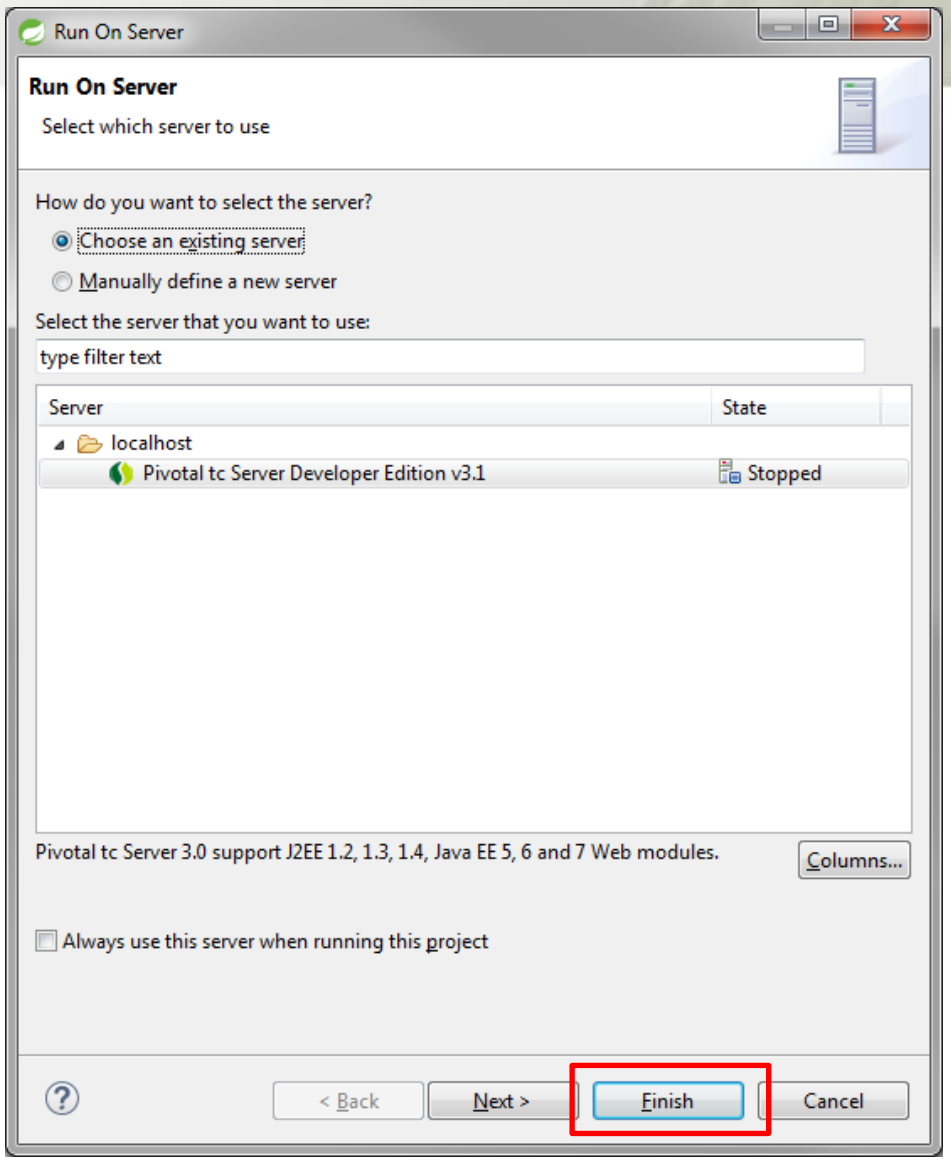
- Add this snippet to *web.xml* (CopyPaste Friendly)

```
<servlet>
   <servlet-name>Jersey REST Service</servlet-name>
   <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
   <init-param>
     <param-name>jersey.config.server.provider.packages</param-name>
     <param-value>services.rest</param-value>
   </init-param>
   <load-on-startup>1</load-on-startup>
 </servlet>
 <servlet-mapping>
   <servlet-name>Jersey REST Service</servlet-name>
   <url-pattern>/rest/*</url-pattern>
 </servlet-mapping>
```

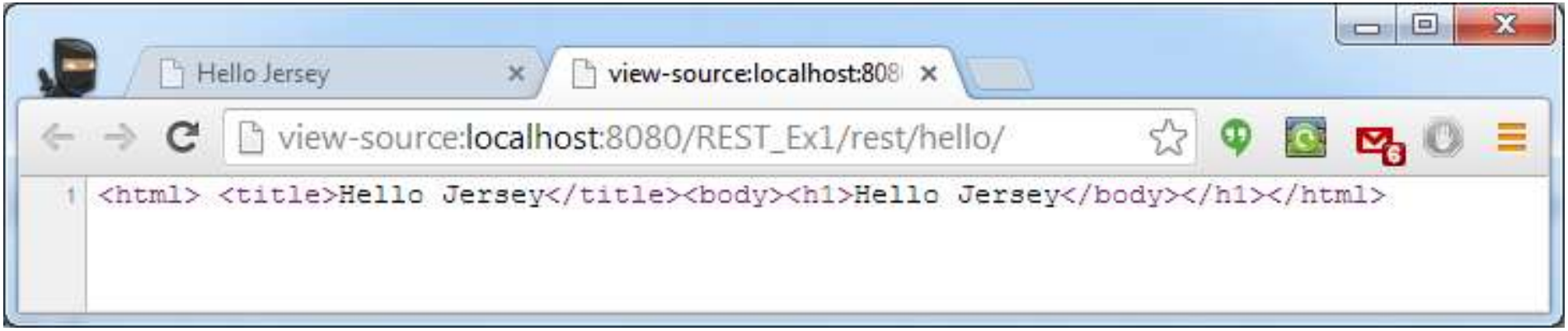
Exercise 1
Run Project



Exercise 1
Run Project



Exercise 1
Test the Project: Web Browser



Resource URL:

Context root web.xml

http://domain:port/context-root/URL-pattern/
path_from_REST_class

@Path

http://localhost:8080/**REST-Ex1**/**rest**/**hello**

Exercise 1
Inspect communication: Web Browser

Trace the HTTP dialog:

- Firefox:
  - (built-in) HTTP request tracer:
    Web Developer > Network (Ctrl+Shift+E)
  - (extension) *HTTPrequester…*
- Chrome:
  - (built-in)
    More tools > Developer tools (Ctrl+Shift+I)
  - (extension) *Postman, ARC, …*

Firefox built-in request inspector

http://localhos...rest/todosimple

localhost:8080/WS1/rest/todosimple

Podany plik XML nie zawiera żadnych informacji o stylach z nim związanych. Poniżej wyświetlone jest drzewo dokumentu.

```
– <todo>
    <description>This is my second todo</description>
    <summary>This is my second todo</summary>
  </todo>
```

Inspektor | Konsola | Debugger | Edytor stylów | Wydajność | Sieć

✓	Metoda	Plik	Domena
200	GET	todosimple	localhost:8080

Nagłówki | Ciasteczka | Parametry | Odpowiedź | Pomiary czasu

URL żądania: http://localhost:8080/WS1/rest/todosimple

Metoda żądania: GET

Kod stanu: 200 OK

Edytuj i wyślij ponownie | Nieprzetworzone nagłówki

Filtruj nagłówki

Nagłówki odpowiedzi (0,132 KB)

Content-Length: "158"

Content-Type: "application/xml"

Date: "Wed, 08 Apr 2015 15:26:21 GMT"

Server: "Apache-Coyote/1.1"

Nagłówki żądania (0,316 KB)

Host: "localhost:8080"

User-Agent: "Mozilla/5.0 (Windows NT 6.1; WOW64; rv:36.0) Gecko/20100101 Firefox/36.0"

Accept: "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8"

Accept-Language: "pl,en-US;q=0.7,en;q=0.3"

Wszystkie | HTML | CSS | JS | XHR | Czcionki | Obrazki | Media | Flash | Inne

Jedno żądanie, 0,15 KB, 0,10 s | Wyczyść

Sieć | CSS | JS | Bezpieczeństwo | Wpisy dziennika | Wyczyść

Filtruj wyjście

Chrome built-in request inspector

Hello Jersey!

localhost:8080/WS1/rest/hello

Hello Jersey!

Elements | Network | Sources | Timeline | Profiles | Resources | Audits | Console

Preserve log | Disable cache

Name
Path

hello
/WS1/rest

Headers | Preview | Response | Timing

General

Remote Address: [::1]:8080
Request URL: http://localhost:8080/WS1/rest/hello
Request Method: GET
Status Code: 200 OK

Response Headers view source

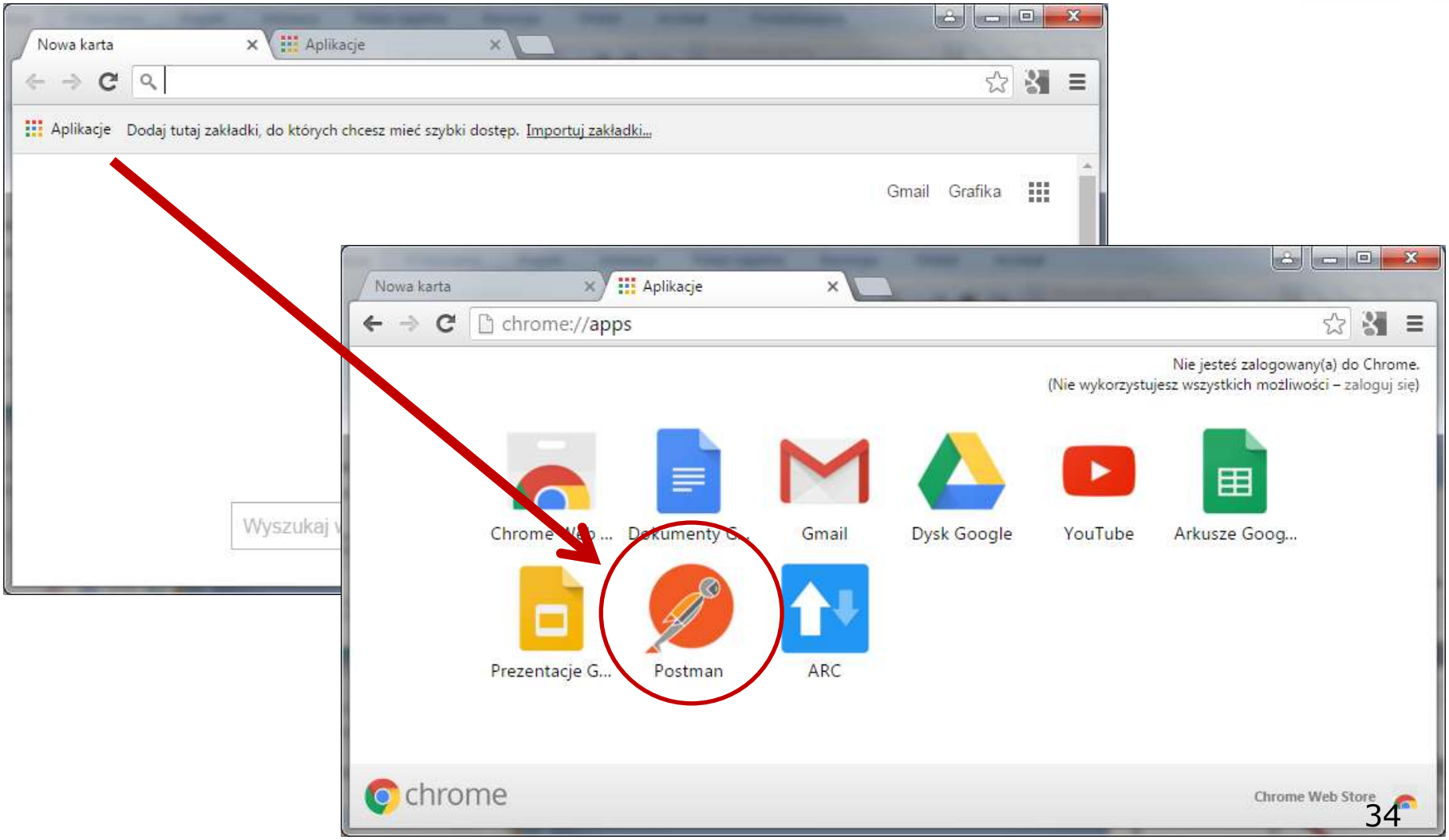
Content-Length: 78
Content-Type: text/html
Date: Fri, 10 Apr 2015 06:49:41 GMT
Server: Apache-Coyote/1.1

Request Headers view source

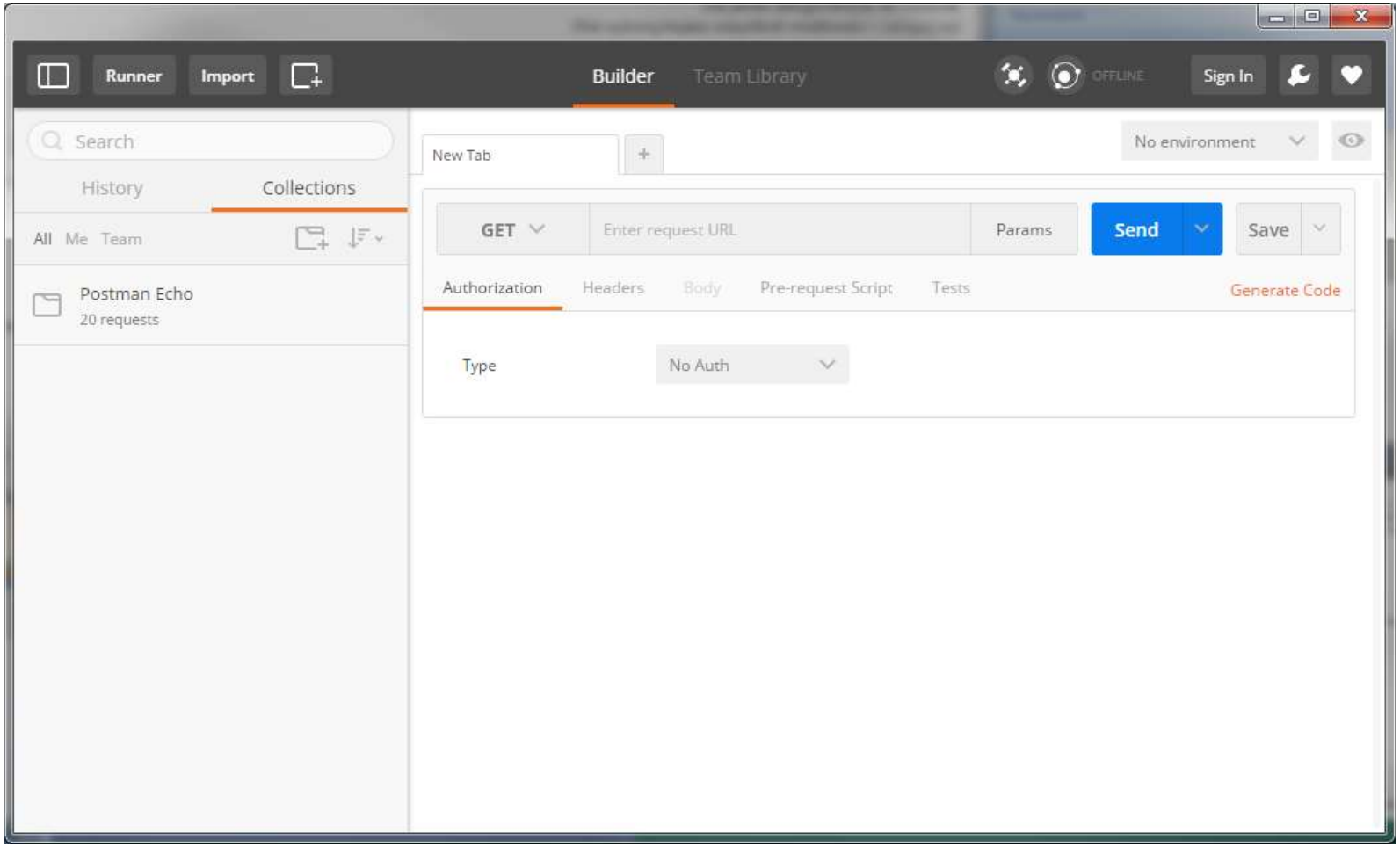
Accept: text/html,application/xhtml+xml,application/xml;q=0.9,image/webp,*/*;q=0.8
Accept-Encoding: gzip, deflate, sdch
Accept-Language: pl-PL,pl;q=0.8,en-US;q=0.6,en;q=0.4
Cache-Control: max-age=0
Connection: keep-alive
Host: localhost:8080
User-Agent: Mozilla/5.0 (Windows NT 6.1; WOW64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/41.0.2272.118 Safari/537.36

1 requests | 206 B transferred | 48 ...

Postman – Chrome application



Postman – Chrome application



REST client

- **Java** application:
  - JAX-RS Client API

- **JavaScript** (browser API):
  - implementation varied from browser to browser:
    - Microsoft – IXMLHTTPRequest,
    - Mozilla – **XMLHttpRequest** (*de facto* standard),

- **jQuery** framework (cross-browser)
  - `$.ajax();`

- **...**

Exercise 1
Test Project: JAX-RS Client API

```
package services.rest;

import java.net.URI;

import javax.ws.rs.client.Client;
import javax.ws.rs.client.ClientBuilder;
import javax.ws.rs.client.WebTarget;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.UriBuilder;

public class RESTHelloWorldClient {
```

Exercise 1
Test Project: JAX-RS Client API
CopyPaste Friendly

```
public static void main(String[] args) {
    Client client = ClientBuilder.newClient();
    WebTarget target = client.target(getBaseURI());

    // Fluent interfaces
    System.out.println(target.path("rest").path("hello").request()
    .accept(MediaType.TEXT_PLAIN).get(Response.class).toString());

    // Get plain text
    System.out.println(target.path("rest").path("hello").request()
    .accept(MediaType.TEXT_PLAIN).get(String.class));

    // Get XML
    System.out.println(target.path("rest").path("hello").request()
    .accept(MediaType.TEXT_XML).get(String.class));

    // Get HTML
    System.out.println(target.path("rest").path("hello").request()
    .accept(MediaType.TEXT_HTML).get(String.class));
}

private static URI getBaseURI() {
    return UriBuilder.fromUri("http://localhost:8080/REST_Ex1").build();
}
```

more than *Hello World* i.e. JAXB

EXERCISE 2

Exercise 2

```java
package services.rest.model;

import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement
public class Todo {
  private String id;
  private String summary;
  private String description;

  public Todo(){
  }
  public Todo (String id, String summary){
    this.id = id;
    this.summary = summary;
  }
```

```java
public String getId() {
  return id;
}
public void setId(String id) {
  this.id = id;
}
public String getSummary() {
  return summary;
}
public void setSummary(String summary) {
  this.summary = summary;
}
public String getDescription() {
  return description;
}
public void setDescription(String description) {
  this.description = description;
}
```

Exercise 2
TodoResourceSimple

```
package services.rest;

import javax.ws.rs.GET;
import javax.ws.rs.Path;
import javax.ws.rs.Produces;
import javax.ws.rs.core.MediaType;

import services.rest.model.Todo;

@Path("/todosimple")
public class TodoResourceSimple {
```

Exercise 2
TodoResourceSimple

```
// This can be used to test the integration with the browser
@GET
@Produces({ MediaType.TEXT_XML })
public Todo getHTML() {
    Todo todo = new Todo();
    todo.setSummary("This is my first todo");
    todo.setDescription("This is my first todo");
    return todo;
}

// This method is called if XMLis request
@GET
@Produces({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
public Todo getXML() {
    Todo todo = new Todo();
    todo.setSummary("This is my first todo");
    todo.setDescription("This is my first todo");
    return todo;
}
```

Exercise 2
Test Project: Web Browser

http://localhos...rest/todosimple

localhost:8080/WS1/rest/todosimple

Podany plik XML nie zawiera żadnych informacji o stylach z nim związanych. Poniżej wyświetlone jest drzewo dokumentu.

```
– <todo>
    <description>This is my second todo</description>
    <summary>This is my second todo</summary>
  </todo>
```

Inspektor | Konsola | Debugger | Edytor stylów | Wydajność | Sieć

✓	Metoda	Plik	Domena
200	GET	todosimple	localhost:8080

Nagłówki | Ciasteczka | Parametry | Odpowiedź | Pomiary czasu

URL żądania: http://localhost:8080/WS1/rest/todosimple
Metoda żądania: GET
Kod stanu: 200 OK

Edytuj i wyślij ponownie | Nieprzetworzone nagłówki

Filtruj nagłówki

Nagłówki odpowiedzi (0,132 KB)
- Content-Length: "158"
- Content-Type: "application/xml"
- Date: "Wed, 08 Apr 2015 15:26:21 GMT"
- Server: "Apache-Coyote/1.1"

Nagłówki żądania (0,316 KB)
- Host: "localhost:8080"
- User-Agent: "Mozilla/5.0 (Windows NT 6.1; WOW64; rv:36.0) Gecko/20100101 Firefox/36.0"
- Accept: "text/html,application/xhtml+xml,application/xml;q=0.9,*/*;q=0.8"
- Accept-Language: "pl,en-US;q=0.7,en;q=0.3"

Wszystkie | HTML | CSS | JS | XHR | Czcionki | Obrazki | Media | Flash | Inne

Jedno żądanie, 0,15 KB, 0,10 s | Wyczyść

Sieć | CSS | JS | Bezpieczeństwo | Wpisy dziennika | Wyczyść

Filtruj wyjście

Exercise 2
Test Project: JAX-RS Client API
CopyPaste Friendly

```
public static void main(String[] args) {
    Client client = ClientBuilder.newClient();
    WebTarget target = client.target(getBaseURI());
    // Fluent interfaces
    System.out.println(target.path("rest").path("todosimple").request()
    .accept(MediaType.TEXT_XML).get(Response.class).toString());

    // Get XML
    System.out.println(target.path("rest").path("todosimple").request()
    .accept(MediaType.TEXT_XML).get(String.class));

    // Get XML for application
    System.out.println(target.path("rest").path("todosimple").request()
    .accept(MediaType.APPLICATION_XML).get(String.class));

    // Get JSON for application
    System.out.println(target.path("rest").path("todosimple").request()
    .accept(MediaType.APPLICATION_JSON).get(String.class));
}

  private static URI getBaseURI() {
    return UriBuilder.fromUri("http://localhost:8080/REST_Ex1").build();
}
```

Exercise 2
Inspect communication: Web Browser

some CRUD i.e. not only HTTP GET

EXERCISE 3

Exercise 3
TodoDao – CopyPaste Friendly

```
package services.rest;

public class TodoDao {

        private static Map<String, Todo> contentProvider = new HashMap<String, Todo>();

        static {
                Todo todo = new Todo("1", "Learn REST theory");
                todo.setDescription("Attend the REST lecture on the TAI course");
                contentProvider.put("1", todo);
                todo = new Todo("2", "Learn REST practice");
                todo.setDescription("Attend the REST laboratory on the TAI course");
                contentProvider.put("2", todo);
        }

        public static Map<String, Todo> getModel() {
                return contentProvider;
        }

}
```

Exercise 3
TodoResource

```java
package services.rest;

import javax.ws.rs.Consumes;
import javax.ws.rs.DELETE;
import javax.ws.rs.GET;
import javax.ws.rs.PUT;
import javax.ws.rs.Produces;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Request;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.UriInfo;
import javax.xml.bind.JAXBElement;

import services.rest.model.Todo;

public class TodoResource {
    @Context
    UriInfo uriInfo;
    @Context
    Request request;
    String id;

    public TodoResource(UriInfo uriInfo, Request request, String id) {
        this.uriInfo = uriInfo;
        this.request = request;
        this.id = id;
    }
```

Exercise 3
TodoResource

```
// Application integration
@GET
@Produces({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
public Todo getTodo() {
    Todo todo = TodoDao.getModel().get(id);
    return todo;
}

// for the browser
@GET
@Produces(MediaType.TEXT_XML)
public Todo getTodoHTML() {
    Todo todo = TodoDao.getModel().get(id);
    return todo;
}
```

Exercise 3
TodoResource

```
@PUT
@Consumes(MediaType.APPLICATION_XML)
public Response putTodo(JAXBElement<Todo> todo) {
    Todo c = todo.getValue();
    return putAndGetResponse(c);
}

@DELETE
public void deleteTodo() {
    Todo c = TodoDao.getModel().remove(id);
}

private Response putAndGetResponse(Todo todo) {
    Response res;
    if (TodoDao.getModel().containsKey(todo.getId())) {
        res = Response.noContent().build();
    } else {
        res = Response.created(uriInfo.getAbsolutePath()).build();
    }
    TodoDao.getModel().put(todo.getId(), todo);
    return res;
}
```

Exercise 3
TodosResource

```java
package services.rest;

import java.io.IOException;
import java.util.ArrayList;
import java.util.List;

import javax.servlet.http.HttpServletResponse;
import javax.ws.rs.Consumes;
import javax.ws.rs.FormParam;
import javax.ws.rs.GET;
import javax.ws.rs.POST;
import javax.ws.rs.Path;
import javax.ws.rs.PathParam;
import javax.ws.rs.Produces;
import javax.ws.rs.core.Context;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Request;
import javax.ws.rs.core.UriInfo;

import services.rest.model.Todo;

//Will map the resource to the URL todos
@Path("/todos")
public class TodosResource {

    // Allows to insert contextual objects into the class,
    // e.g. ServletContext, Request, Response, UriInfo
    @Context
    UriInfo uriInfo;
    @Context
    Request request;
```

Exercise 3
TodosResource CopyPaste Friendly

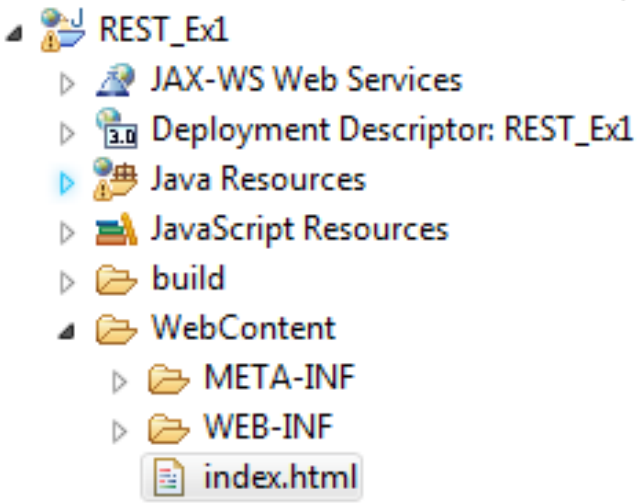
```
@GET
@Produces(MediaType.TEXT_XML)
public List<Todo> getTodosBrowser() {
  List<Todo> todos = new ArrayList<Todo>();
  todos.addAll(TodoDao.getModel().values());
  return todos;
}

// Return the list of todos for applications
@GET
@Produces({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
public List<Todo> getTodos() {
  List<Todo> todos = new ArrayList<Todo>();
  todos.addAll(TodoDao.getModel().values());
  return todos;
}

// Returns the number of todos
// Use http://localhost:8080/WS1/rest/todos/count
@GET
@Path("count")
@Produces(MediaType.TEXT_PLAIN)
public String getCount() {
  int count = TodoDao.getModel().size();
  return String.valueOf(count);
}
```

Exercise 3
TodosResource – CopyPaste Friendly

```
@POST
@Produces(MediaType.TEXT_HTML)
@Consumes(MediaType.APPLICATION_FORM_URLENCODED)
public void newTodo(@FormParam("id") String id,
                    @FormParam("summary") String summary,
                    @FormParam("description") String description,
                    @Context HttpServletResponse servletResponse) throws IOException{
        Todo todo = new Todo(id, summary);
        if (description != null) {
                    todo.setDescription(description);
        }
        TodoDao.getModel().put(id, todo);

        servletResponse.sendRedirect("../index.html");
}

// Defines that the next path parameter after todos is
// treated as a parameter and passed to the TodoResources
// Allows to type http://localhost:8080/REST_Ex1/rest/todos/1
// 1 will be treaded as parameter todo and passed to TodoResource
@Path("{todo}")
public TodoResource getTodo(@PathParam("todo") String id) {
        return new TodoResource(uriInfo, request, id);
}
```

Dopiero tutaj przydaje się klasa *TodoResource*

Exercise 3
index.html – CopyPaste Friendly




```
<!DOCTYPE html>
<html>
<head>
<meta charset="UTF-8">
<title>Insert title here</title>
</head>
<body>
  <form action="rest/todos" method="POST">
  <label for="id">ID</label>
  <input name="id" />
  <br/>
  <label for="summary">Summary</label>
  <input name="summary" />
  <br/>
  Description:
  <TEXTAREA NAME="description" COLS=40 ROWS=6></TEXTAREA>
  <br/>
  <input type="submit" value="Submit" />
  </form>
</body>
</html>
```

Exercise 3
Test Project: Web Browser

- Display all todos:
  http://localhost:8080/REST_Ex1/rest/todos/
- Display single todo:
  http://localhost:8080/REST_Ex1/rest/todos/2
- Display todo count:
  http://localhost:8080/REST_Ex1/rest/todos/count

- Submit a new todo through form:
  http://localhost:8080/REST_Ex1/
- and then check if it is visible in:
  all todos, single todo, count

Exercise 3 continuation – building a client

EXERCISE 4

Exercise 4
Test Project 3 through JAX-RS Client API

```
package services.rest;

import java.net.URI;

import javax.ws.rs.core.Form;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.UriBuilder;
import javax.ws.rs.client.Client;
import javax.ws.rs.client.ClientBuilder;
import javax.ws.rs.client.Entity;
import javax.ws.rs.client.WebTarget;

import services.rest.model.Todo;

public class TodosResourceClient {
    public static void main(String[] args) {
        Client client = ClientBuilder.newClient();
        WebTarget target = client.target(getBaseURI());
        (...)
    }

    private static URI getBaseURI() {
        return UriBuilder.fromUri("http://localhost:8080/REST_Ex1").build();
    }
```

Exercise 4
Test Project 3 through JAX-RS Client API

```
// Get the Todos
System.out.println(service.path("rest").path("todos")
        .request(MediaType.TEXT_XML).get(String.class));
// Get JSON for application
System.out.println(service.path("rest").path("todos")
        .request(MediaType.APPLICATION_JSON).get(String.class));
// Get XML for application
System.out.println(service.path("rest").path("todos")
        .request(MediaType.APPLICATION_XML).get(String.class));
```

Exercise 4
Test Project 3 through JAX-RS Client API

```
// Create a new todo through PUT
   Todo todo = new Todo("3", "Blablabla bla bla");
   Response response = target.path("rest").path("todos")
   .path(todo.getId()).request(MediaType.APPLICATION_XML)
   .put(Entity.xml(todo));

   // Return code should be: 201 == created resource
   // or 204 == No Content if resource is already present
   System.out.println(response.getStatus());
   System.out.println(response.getStatusInfo().toString());

   // Get the Todos, number 3 should be created
   System.out.println(target.path("rest").path("todos").request()
   .accept(MediaType.TEXT_XML).get(String.class));
```

Exercise 4
Test Project 3 through JAX-RS Client API

```
// Get the Todo with id 1
System.out.println(target.path("rest").path("todos/1")
.request(MediaType.APPLICATION_XML).get(String.class));

// Delete the Todo with id 1
target.path("rest").path("todos/1").request().delete();
System.out.println(response.getStatus());
System.out.println(response.getStatusInfo().toString());

// Get the all todos, id 1 should be deleted
System.out.println(target.path("rest").path("todos")
.request(MediaType.APPLICATION_XML).get(String.class));
```

Exercise 4
Test Project 3 through JAX-RS Client API

```
// Create a Todo through a Form
Form form = new Form();
form.param("id", "4");
form.param("summary", "Demonstration of the client lib for forms");
response = target.path("rest").path("todos")
.request()
.post(Entity.entity(form, MediaType.APPLICATION_FORM_URLENCODED));
System.out.println("Form response: " + response.readEntity(String.class));

// Get the all todos, id 4 should be created
System.out.println(target.path("rest").path("todos")
.request(MediaType.APPLICATION_XML).get(String.class));
```

During the tests pay attention to the *APPLICATION_FORM_URLENCODED* request

Exercise 4
Observe contextual params injection

- Add code into TodoResource.java (e.g. constructor)

```
System.out.println(request.getMethod());
System.out.println(uriInfo.getPath());
```

- and check the server console during tests

Inspect the client–server REST dialog

EXERCISE 5

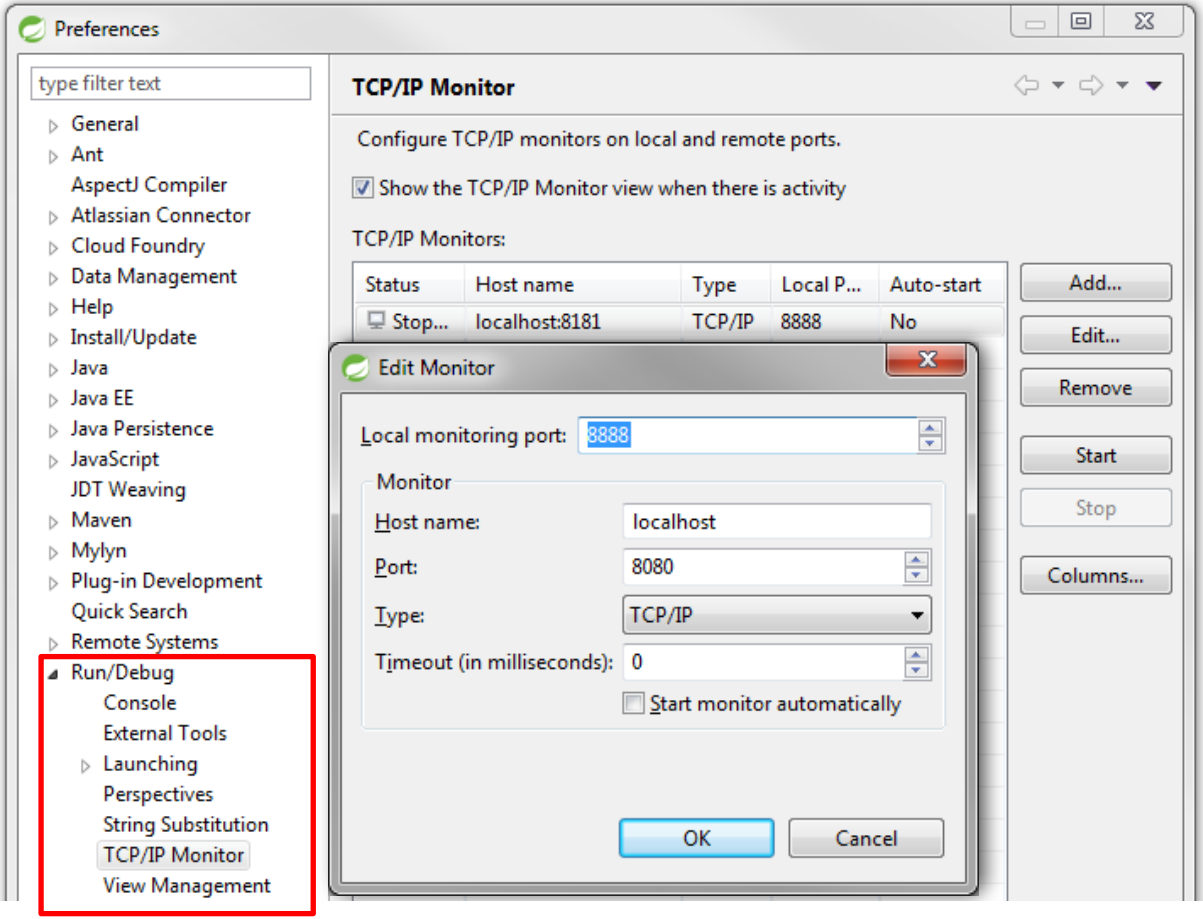
Exercise 5
Trace client-server REST dialog

- *Eclipse TCP/IP Monitor (built-in)* (http://www.mkyong.com/webservices/jax-ws/how-to-trace-soap-message-in-eclipse-ide/)

- Remember to change the port called by the client (in the web browser).

- OR
  TunelliJ plug-in on IntelliJ IDEA IDE

Eclipse TCP/IP Monitor

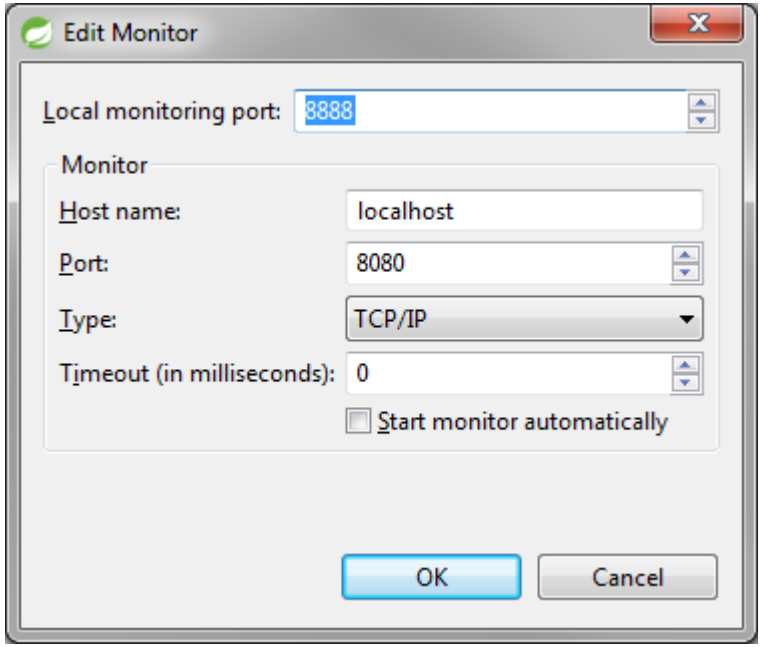


Windows → Preferences → Run/Debug → TCP/IP Monitor

Eclipse TCP/IP Monitor

1. **Client** → TcpMonitor:8888

2. TcpMonitor:8888 →
   → **Server**:8080



3. **Server**:8080 →
   → TcpMonitor:8888

4. TcpMonitor:8888 → **Client**

Exercise 5
Trace client-server REST dialog

Markers | Properties | Servers | Data Source Explorer | Snippets | Console | Progress | TCP/IP Monitor

localhost:8888
localhost:8080

Time of request: 2:09.8.202 PM
Response Time: 11 ms
Type: TCP/IP

Request viewer type: Byte
Request:
Size:
Header: GET /WS1/rest/todos/ HTTP/1.1
Encoding: <None>

Response viewer type: Byte
Response:
Size:
Header: HTTP/1.1 200 OK
Encoding: <None>

```
GET /WS1/rest/todos/ HTTP/1.1
Accept: application/x-ms-application, image/jpeg, app
Accept-Language: pl-PL
UA-CPU: AMD64
Accept-Encoding: gzip, deflate
User-Agent: Mozilla/5.0 (compatible; MSIE 10.0; Windo
Host: localhost:8888
Connection: Keep-Alive
Cookie: com.springsource.sts.run.embedded=true
```

```
HTTP/1.1 200 OK
Server: Apache-Coyote/1.1
Content-Type: text/xml
Content-Length: 331
Date: Tue, 31 Mar 2015 12:09:08 GMT

<?xml version="1.0" encoding="UTF-8" standalone="yes"?><todoes><todo><descrip
```

References

- http://www.vogella.com/articles/REST/article.html
- https://jersey.java.net/documentation/latest/index.html
- http://www.mkyong.com/webservices/jax-ws/how-to-trace-soap-message-in-eclipse-ide

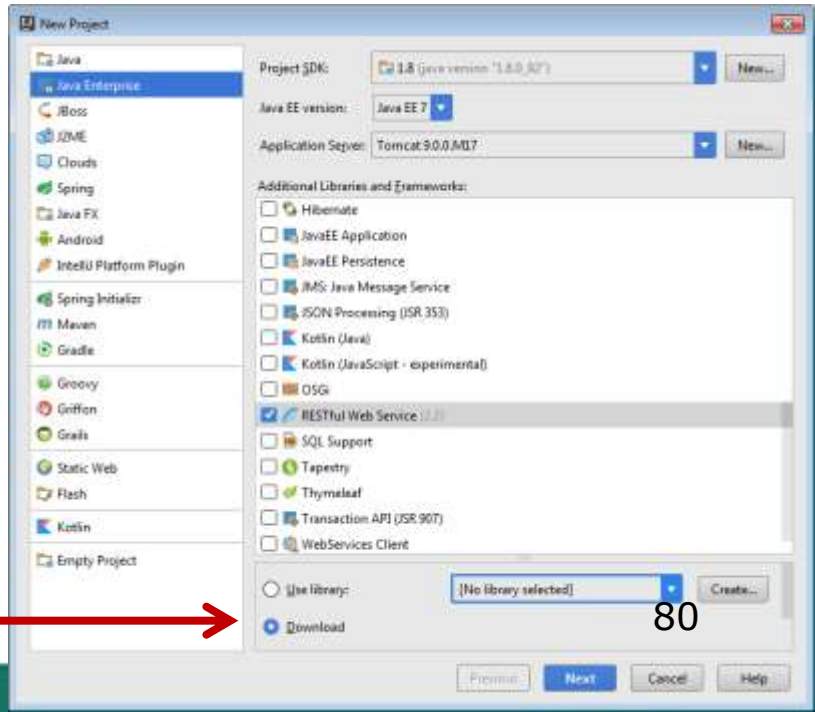
INTELLIJ ULTIMATE CONFIGURATION

If you prefer IntelliJ Ulimate

- Instal Apache Tomcat
  - http://tomcat.apache.org/
- Instal JDK
- Set application server in IntelliJ:
  - File>Settings>Build, Execution,Deployment> Application Servers
  - Add Tomcat (choose the folder)
- Set JDK in IntelliJ:
  - File>Project Structure>Platform Settings>SDK
  - Choose JDK (remember JAVA_HOME!)

If you prefer IntelliJ Ulimate

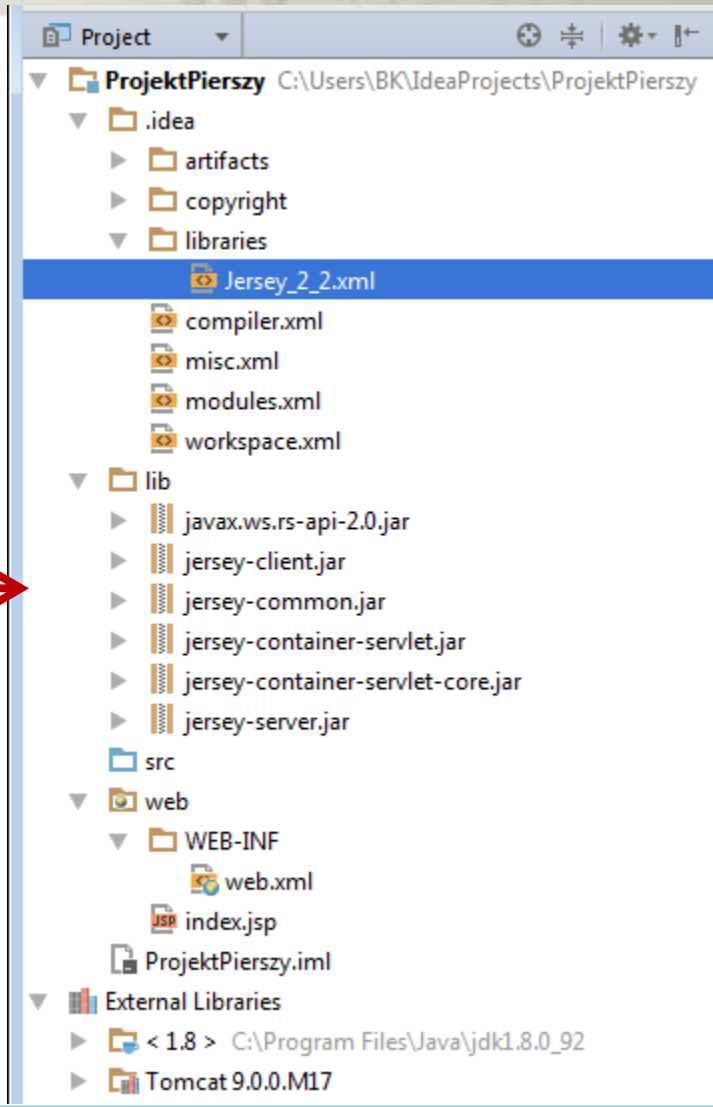
- New project: File>New>Project>
  - Java Enterprise
  - Choose JDK i Application Server
  - WebApplication + RESTful Web Service
    - Download
- Choose project name and location



If you prefer IntelliJ <u>Ulimate</u>

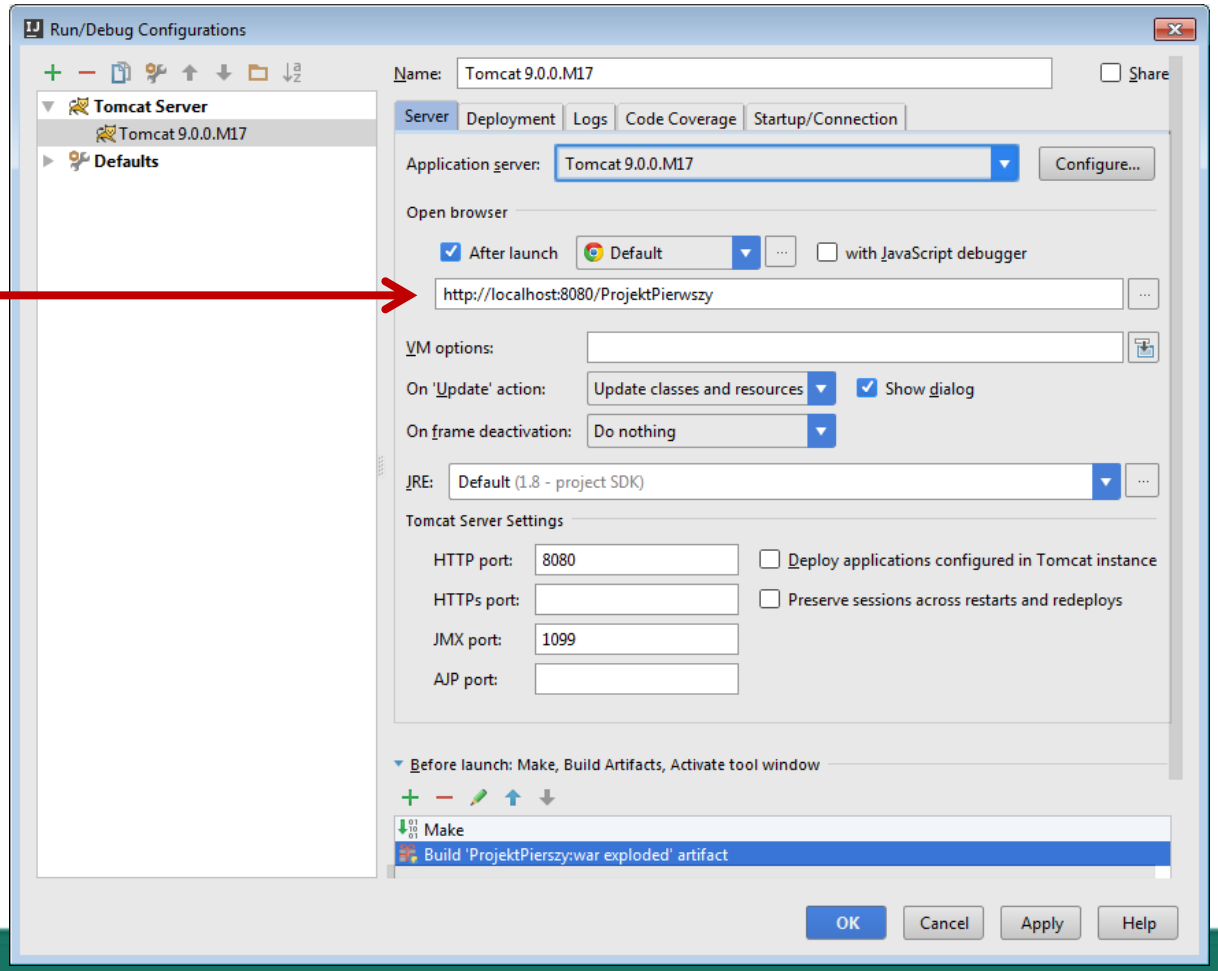
- Jersey library should be added

- Now follow the presentation



If you prefer IntelliJ Ulimate

- Run>EditConfiguration



If you prefer IntelliJ Ulimate

- Run>EditConfiguration