

Tutorial REST Web Service pada Java Menggunakan Framework (JAX-RS) Jersey

Husni@trunojoyo.ac.id

Daftar Isi

1. REST: Representational State Transfer	3
1.1. Apa itu REST?	3
1.2. Metode-metode HTTP	3
1.3. RESTful Web Service	3
2. JAX-RS dengan Jersey	4
2.1. JAX-RS	4
2.2. Jersey	4
2.3. Anotasi JAX-RS	4
3. Instalasi Jersey	5
3.1. Setup Jersey Secara Manual di Proyek Eclipse	5
4. Web Container	5
5. Prasyarat	6
6. Membuat RESTful Web Service Pertama	6
6.1. Membuat Proyek Web Baru	6
6.2. Mengkonfigurasi Pemanfaatan Jersey	7
6.3. Kelas Java REST API	8
6.4. Mendefinisikan Servlet Dispatcher dari Jersey	10
6.5. Menjalankan Layanan REST	11
7. Membuat Client dari REST API	13
7.1 Menggunakan Pustaka Jersey	13
7.2 Menggunakan Kelas URL	20
8. RESTful Web Service dan JAXB	21
8.1. Membuat Proyek REST API JAXB	21
8.2. Mengkonfigurasi Pemanfaatan Jersey	21
8.3. Membuat File-file Web Service	21
8.4. Membuat Client	23
9. CRUD RESTful Web Service	24
9.1. Memulai Proyek	24

9.2. Membuat Form HTML Sederhana	26
9.3. Rest API Provider.....	26
9.4. Menjalankan REST API	29
9.5. Membuat Client	32
9.6. Mengakses Layanan REST via Halaman HTML	34
10. Referensi	34

RESTful web services dengan Java (Jersey / JAX-RS). Tutorial ini akan menjelaskan bagaimana mengembangkan suatu RESTful web services dalam Java menggunakan Jersey, suatu implementasi referensi JAX-RS. Tutorial ini menggunakan beberapa software, yaitu Eclipse 4.5 (Mars) atau Neon 3, JDK 1.8 atau dikenal sebagai Java 8, Tomcat 8.0 atau 8.5 dan JAX-RS 2.0 (dengan Jersey 2.11).

1. REST: Representational State Transfer

1.1. Apa itu REST?

REST merupakan suatu style arsitektural yang didasarkan pada standar web dan protokol HTTP. REST pertama kali dijelaskan oleh Roy Fielding pada tahun 2000. Dalam suatu arsitektur berbasis REST segalanya adalah resource (sumber daya). Suatu resource diakses melalui antarmuka (interface) umum berbasis pada metode-metode standar HTTP. Dalam suatu arsitektur berbasis REST kita mempunyai suatu server REST (API Provider) yang mempunyai akses ke sumber daya dan menyediakan beberapa akses terbatas ke sumber daya untuk penggunaannya, dan client REST (API Customer, pengguna dari server REST) yang meminta akses dan perubahan terhadap sumber daya REST.

Setiap sumber daya harus mendukung operasi umum dari HTTP. Sumber daya ini diidentifikasi oleh global ID (yang biasanya berupa URI). REST memungkinkan bahwa sumber daya mempunyai representasi berbeda seperti text, XML, JSON, dll. Client dari REST dapat meminta representasi spesifik melalui protokol HTTP (negosiasi konten).

1.2. Metode-metode HTTP

Metode *PUT*, *GET*, *POST* dan *DELETE* biasa digunakan dalam arsitektur berbasis REST. Rincian berikut memberikan penjelasan mengenai operasi-operasi ini:

- *GET* mendefinisikan suatu akses pembacaan sumber daya tanpa efek samping. Sumber daya tidak pernah diubah melalui request *GET*, misalnya request hanya meminta detail mengenai sumber daya nomor tertentu, tidak meminta perubahan satu baris record pun di provider. Operasi bersifat idempotent.
- *PUT* membuat suatu sumber daya baru. Operasi juga harus bersifat idempotent.
- *DELETE* menghapus sumber daya tertentu. Operasinya bersifat idempotent. Operasi ini dapat diulang tanpa mengakibatkan hasil yang berbeda.
- *POST* mengupdate sumber daya yang telah ada atau membuat suatu sumber daya baru.

1.3. RESTful Web Service

Suatu RESTful web services didasarkan pada metode-metode HTTP dan konsep dari REST. RESTful web service khususnya mendefinisikan URI basis bagi layanan, tipe MIME yang didukung (XML, text, JSON, user-defined, ...) dan himpunan operasi (*POST*, *GET*, *PUT*, *DELETE*) yang didukung.

2. JAX-RS dengan Jersey

2.1. JAX-RS

Java mendefinisikan dukungan REST melalui *Java Specification Request (JSR) 311*. Spesifikasi ini dinamakan juga JAX-RS (Java API for RESTful Web Services). JAX-RS menggunakan anotasi untuk mendefinisikan relevansi REST dari kelas-kelas Java.

2.2. Jersey

Jersey merupakan suatu implementasi referensi untuk spesifikasi JSR 311.

Implementasi Jersey menyediakan suatu pustaka untuk mewujudkan RESTful webservice dalam suatu container servlet Java.

Pada sisi server Jersey menyediakan implementasi servlet yang mengamati kelas-kelas predefined untuk mengidentifikasi sumber daya RESTful. Di dalam file konfigurasi web.xml kita mendaftarkan (register) servlet ini untuk aplikasi web kita.

Implementasi Jersey juga menyediakan suatu pustaka client untuk berkomunikasi dengan RESTful web service (komunikasi customer dengan provider REST API).

URL basis dari servlet ini adalah

http://nama_domain:port/display-name/url-pattern/path_dari_kelas_REST

Servlet ini menganalisa request HTTP yang masuk dan memilih kelas dan metode yang tepat untuk merespon request ini. Pemilihan ini didasarkan pada anotasi dalam kelas dan metode tersebut.

Suatu aplikasi web REST terdiri dari (karena itu) kelas-kelas data (sumber daya) dan layanan-layanan. Dua jenis ini khasnya dikelola dalam paket-paket berbeda sebagai servlet Jersey akan diinstruksikan melalui web.xml untuk memeriksa paket tertentu bagi kelas-kelas data.

JAX-RS mendukung pembuatan XML dan JSON melalui Java Architecture for XML Binding (JAXB).

2.3. Anotasi JAX-RS

Anotasi paling penting dalam JAX-RS dijelaskan di dalam Tabel 1.

Tabel 1. Anotasi dalam JAX-RS

Anotasi	Penjelasan
@PATH(path_kita)	Atur path ke URL basis + /path_kita. URL basis didasarkan pada nama aplikasi, servlet dan pola URL (URL pattern) di dalam file konfigurasi web.xml.
@POST	Menunjukkan bahwa metode mengikutinya akan menjawab suatu request POST HTTP.
@GET	Menunjukkan bahwa metode yang mengikutinya akan menjawab suatu request GET HTTP.

Tabel 1. Anotasi dalam JAX-RS

Anotasi	Penjelasan
@PUT	Menunjukkan bahwa metode yang mengikutinya akan menjawab suatu request PUT HTTP.
@DELETE	Menunjukkan bahwa metode yang mengikutinya akan menjawab suatu request DELETE HTTP.
@Produces(MediaType.TEXT_PLAIN[, tipe_lain])	@Produces mendefinisikan tipe MIME apa yang dihantarkan oleh suatu metode yang dianotasi dengan @GET. Dalam contoh dihasilkan teks ("text/plain"). Contoh lain dapat berupa "application/xml" atau "application/json".
@Consumes(tipe[, tipe-lain])	@Consumes mendefinisikan tipe MIME apa yang dikonsumsi oleh metode ini.
@PathParam	Digunakan untuk memasukkan nilai-nilai dari URL ke dalam suatu parameter metode. Dengan cara ini kita memasukkan, misalnya, ID dari suatu sumber daya ke dalam metode untuk mendapatkan obyek yang tepat.

Path lengkap untuk suatu sumber daya didasarkan pada URL basis dan anotasi @PATH di dalam kelas program REST API yang dibuat.

http://nama_domain:port/display-name/url-pattern/path_dari_kelas_REST

3. Instalasi Jersey

3.1. Setup Jersey Secara Manual di Proyek Eclipse

Download distribusi Jersey sebagai file .zip dari situs Jersey (<https://jersey.java.net/download.html>).

File .zip tersebut mengandung JAR implementasi JAR dan ketergantungan intinya. Jersey ini tidak menyediakan ketergantungan (dependensi) terhadap JAR pihak ketiga selain dukungan untuk JSON dan JavaDoc.

Ekstrak file .zip dari Jersey yang telah didownload dan salin semua file JAR yang diperoleh ke dalam folder WEB-INF/lib dari proyek yang memerlukan Jersey. Lebih jelasnya dapat dilihat [ada bagian 6 dari tutorial ini.

4. Web Container

Untuk tutorial ini kita dapat menggunakan web container apapun, misalnya Tomcat (paling banyak digunakan) atau Google App Engine.

Jika ingin menggunakan Tomcat sebagai container servlet sebaiknya anda mempelajari bagaimana membangun aplikasi web di Eclipse di:

<http://www.vogella.com/tutorials/EclipseWTP/article.html>

dan bagaimana instalasi dan pemanfaatan Tomcat sebagai container servlet di:

<http://www.vogella.com/tutorials/ApacheTomcat/article.html>

Sebagai alternatif, kita dapat juga menggunakan Google App Engine untuk menjalankan bagian server dari contoh-contoh REST API yang akan kita buat. Jika anda memilih menggunakan Google App Engine, maka Tomcat tidak harus diinstal dan dikonfigurasi.

Jika anda menggunakan GAE/J, maka anda harus membuat proyek App Engine di Eclipse, bukan memilih *Dynamic Web Project*. Penjelasan di dalam bagian 6 dan seterusnya didasarkan pada container Apache Tomcat.

5. Prasyarat

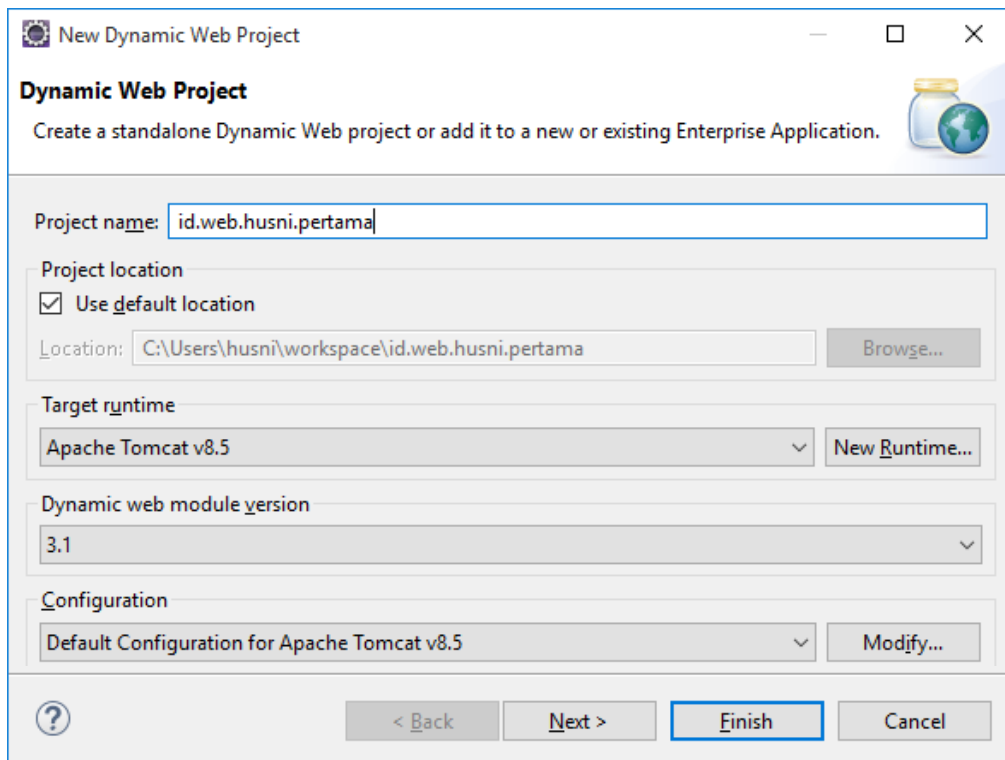
Penjelasan praktik mulai bagian 6 dari tutorial ini menganggap bahwa kita sudah akrab dengan pembuatan aplikasi web menggunakan Java dan Eclipse. Pengantar bagaimana membuat aplikasi web dengan Eclipse dapat diperoleh pada alamat:

<http://www.vogella.com/tutorials/EclipseWTP/article.html>.

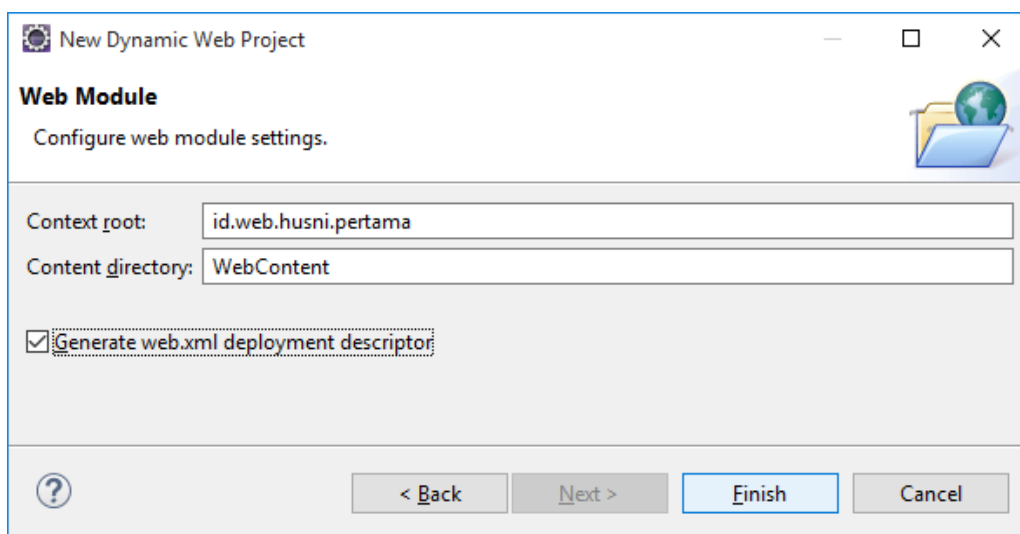
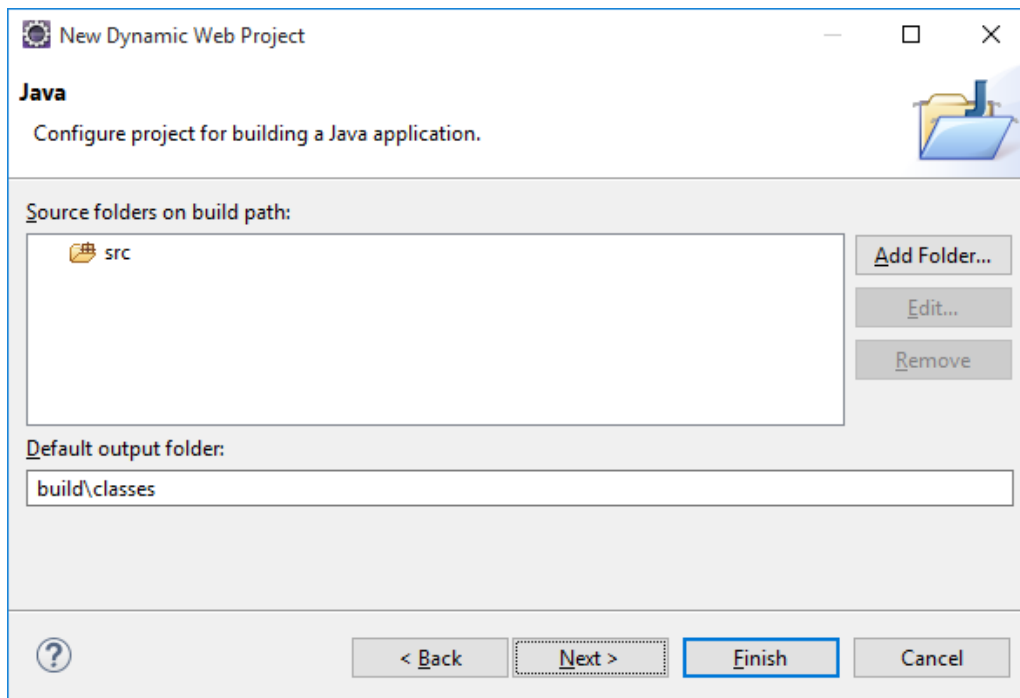
6. Membuat RESTful Web Service Pertama

6.1. Membuat Proyek Web Baru

Mari kita mulai pembuatan REST Web Service dengan Java dan Jersey. Buatlah suatu *Dynamic Web Project* baru dan namai sebagai (misalnya) *id.web.husni.pertama*.



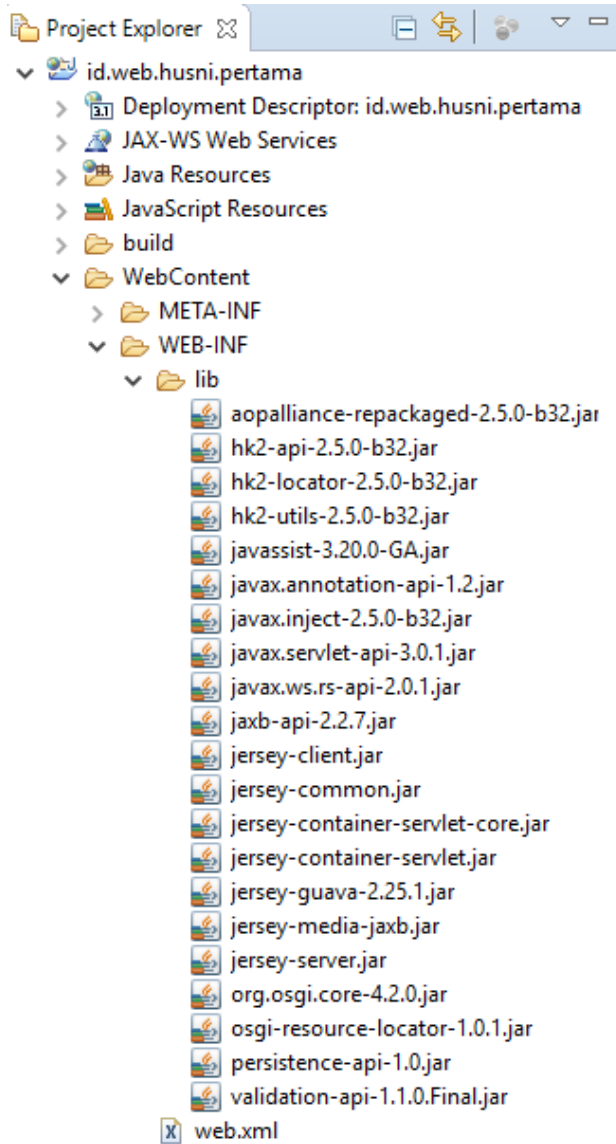
Pastikan bahwa kita membuat web.xml deployment descriptor.



6.2. Mengkonfigurasi Pemanfaatan Jersey

Sesuai dengan panduan setup Jersey sebelumnya, kita harus menyalin semua file .jar dari framework Jersey ke dalam direktori WebContent > Web-INF > Lib. Caranya cukup mudah, yaitu dengan blok semua file .jar dari Jersey, *drag* dan *drop* ke dalam folder Lib dari proyek yang sedang dikerjakan (di dalam Eclipse).

Berikut ini adalah tampilan dari Project Explorer di Eclipse untuk proyek id.web.husni.pertama setelah semua file .jar dari Jersey di salin ke dalam folder Lib.



6.3. Kelas Java REST API

Buat kelas Hello yang akan berfungsi sebagai penyedia REST API atau Web Service. Klik kanan nama proyek > New > Class. Berikan nama Hello.

New Java Class

Create a new Java class.

Source folder:

Package:

☐ Enclosing type:

Name:

Modifiers: ☒ public ☐ package ☐ private ☐ protected
☐ abstract ☐ final ☐ static

Superclass:

Interfaces:

Which method stubs would you like to create?

☐ public static void main(String[] args)
☐ Constructors from superclass
☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))

☐ Generate comments

Berikut adalah kode program dari file Hello.java:

```
package id.web.husni.pertama;

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

// Obyek Java sederhana. Tidak meng-extend kelas atau meng-implement suatu interface

// Kelas mendaftarkan metode-metodenya untuk request HTTP GET menggunakan anotasi @GET.
// Menggunakan anotasi @Produces untuk mendefinisikan bahwa tipe MIME yang digunakan
// mencakup: text, XML dan HTML.

// Request dari web browser secara default adalah tipe MIME HTML.

// Set path ke URL basis + /hello
@Path("/hello")
public class Hello {
    // Metode ini dipanggil jika request bertipe MIME TEXT_PLAIN
    @GET
    @Produces(MediaType.TEXT_PLAIN)
    public String sayPlainTextHello() {
        return "Hello Jersey";
    }
}
```

```

}

// Metode ini dipanggil jika request bertipe MIME XML
@GET
@Produces(MediaType.TEXT_XML)
public String sayXMLHello() {
    return "<?xml version='1.0'>" + "<hello> Hello Jersey" + "</hello>";
}

// Metode ini dipanggil jika request bertipe MIME HTML
@GET
@Produces(MediaType.TEXT_HTML)
public String sayHtmlHello() {
    return "<html> " + "<title>" + "Hello Jersey" + "</title>"
        + "<body><h1>" + "Hello Jersey" + "</body></h1>" + "</html> ";
}
} //akhir kelas Hello

```

Kelas ini mendaftarkan dirinya sendiri sebagai suatu sumber daya get melalui anotasi `@GET`. Kemudian memanfaatkan anotasi `@Produces` kelas Hello ini mendefinisikan bahwa ia menyampaikan tipe MIME *text* dan *HTML*. Ia juga mendefinisikan melalui anotasi `@Path` bahwa layanannya tersedia di bawah URL `hello`.

Web browser akan selalu me-request dengan tipe MIME HTML. Versi teks dapat dilihat menggunakan tool seperti `curl`, aplikasi client REST API berbasis Console atau langsung dari dalam Eclipse Neon.

6.4. Mendefinisikan Servlet Dispatcher dari Jersey

Kita perlu mendaftarkan Jersey sebagai servlet dispatcher bagi request REST. Buka file `web.xml` dan lakukan perubahan seperti di bawah ini:

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">
    <display-name>id.web.husni.pertama</display-name>
    <servlet>
        <servlet-name>Jersey REST Service</servlet-name>
        <servlet-class>org.glassfish.jersey.servlet.ServletContainer</servlet-class>
        <!--Register-kan sumber daya dan provider di bawah paket id.web.husni.pertama. -->
        <init-param>
            <param-name>jersey.config.server.provider.packages</param-name>
            <param-value>id.web.husni.pertama</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>
</web-app>

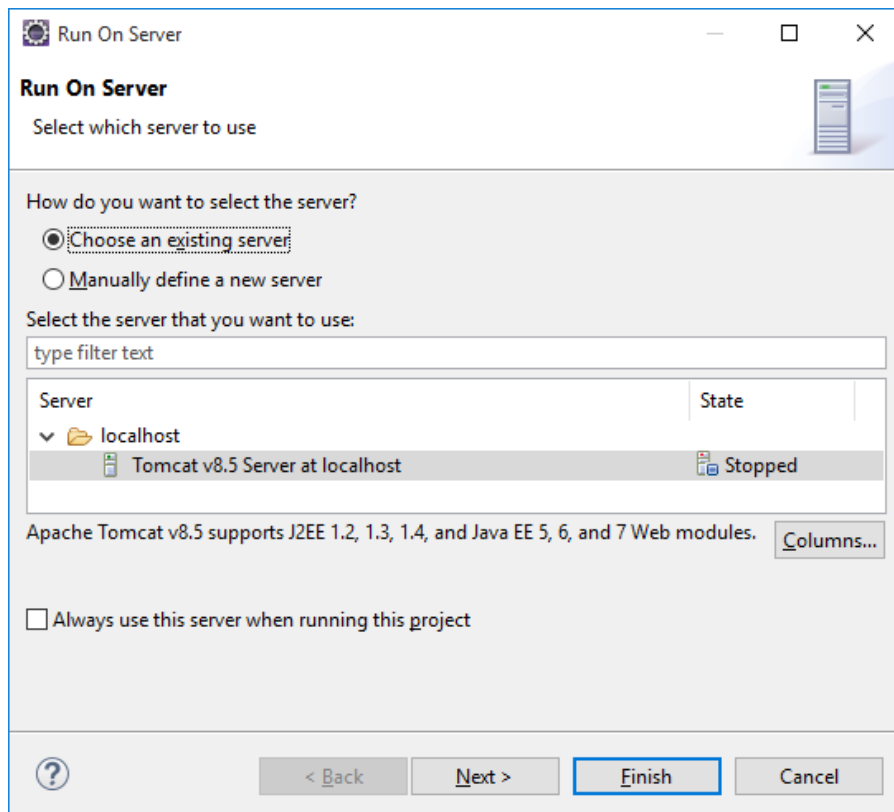
```

Parameter `jersey.config.server.provider.packages` mendefinisikan di dalam paket mana Jersey akan mencari kelas-kelas web service. Properti ini harus menunjuk ke kelas sumber daya kita. URL -Pattern mendefinisikan bagian dari URL basis yang akan ditempati aplikasi yang dibuat.

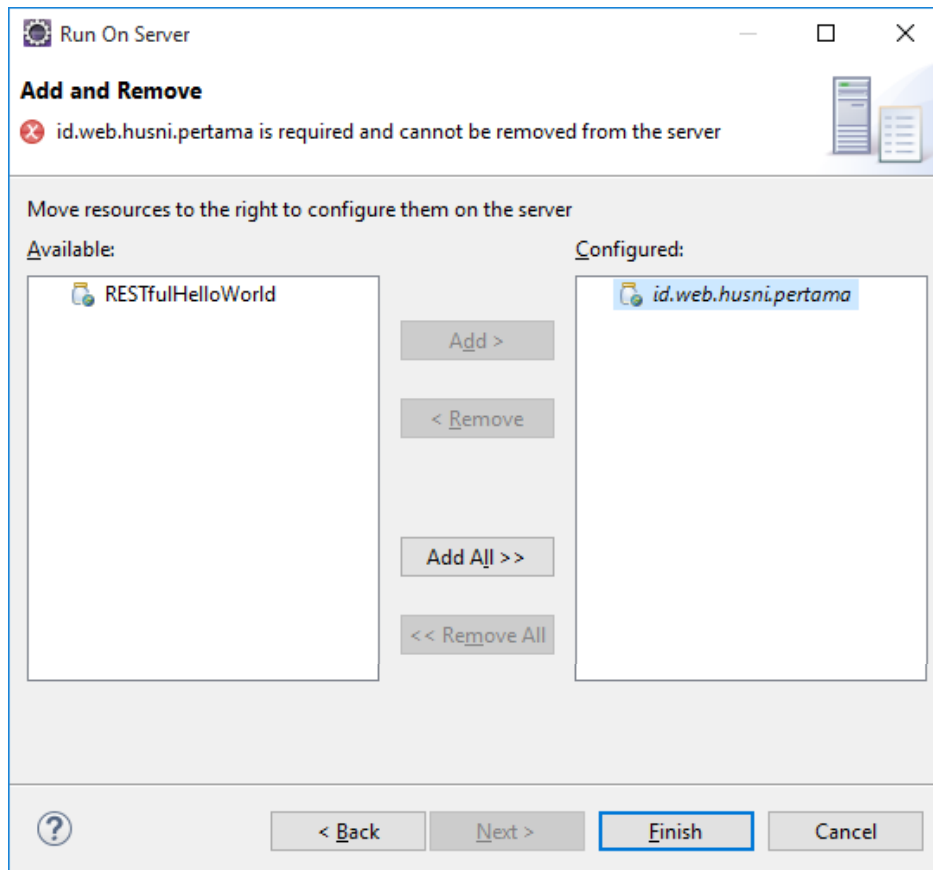
6.5. Menjalankan Layanan REST

Jalankan aplikasi web tersebut di dalam Eclipse. Caranya cukup mudah:

1. Klik kanan nama proyek (id.web.husni.pertama), pilih Run As > 1. Run On Server
2. Pada jendela Run On Server, pilih server tau Container yang akan digunakan. Dalam hal ini, kita pilih misalnya Tomcat v8.5 Server at Localhost. Klik Next.



3. Pilih atau masukkan proyek ke dalam kotak Configured:. Klik Finish.

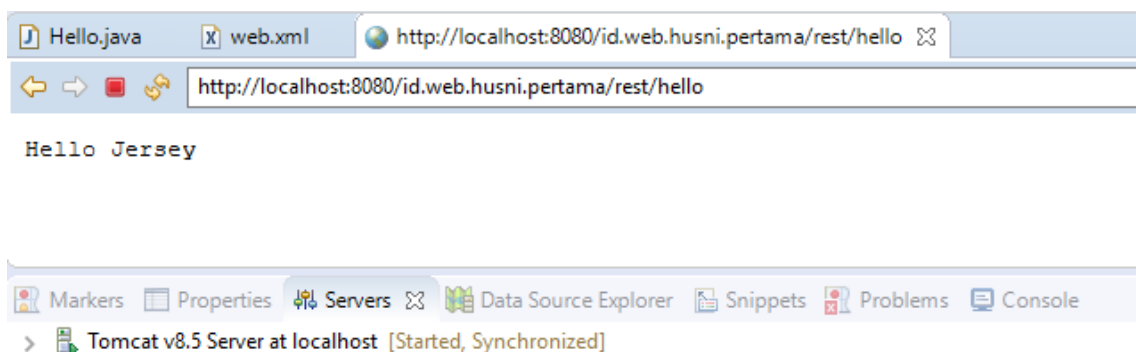


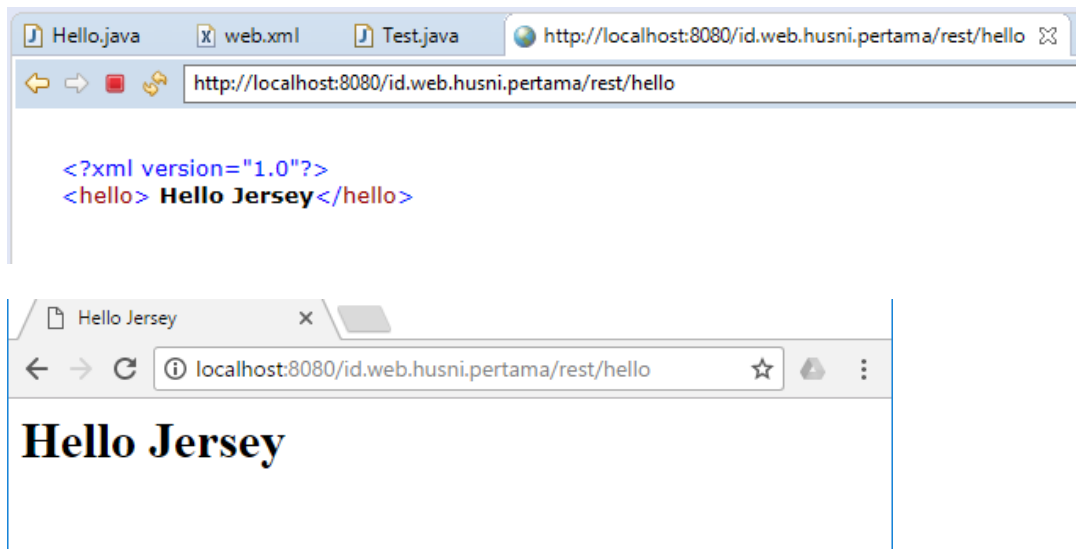
Kita dapat mengakses sumber daya yang telah didefinisikan mengikuti URL berikut:

<http://localhost:8080/id.web.husni.pertama/rest/hello>

Nama ini diturunkan dari "display-name" yang didefinisikan dalam file web.xml, ditambahkan dengan servlet-mapping URL-pattern dan anotasi @Path hello dari file kelas Hello. Kita akan memperoleh pesan "Hello Jersey".

Web browser me-requests representasi HTML dari sumber daya. Di bawah ini adalah hasil yang diperoleh saat sumber daya REST diakses melalui fasilitas browser *built-in* Eclipse dan Web browser Google Chrome.



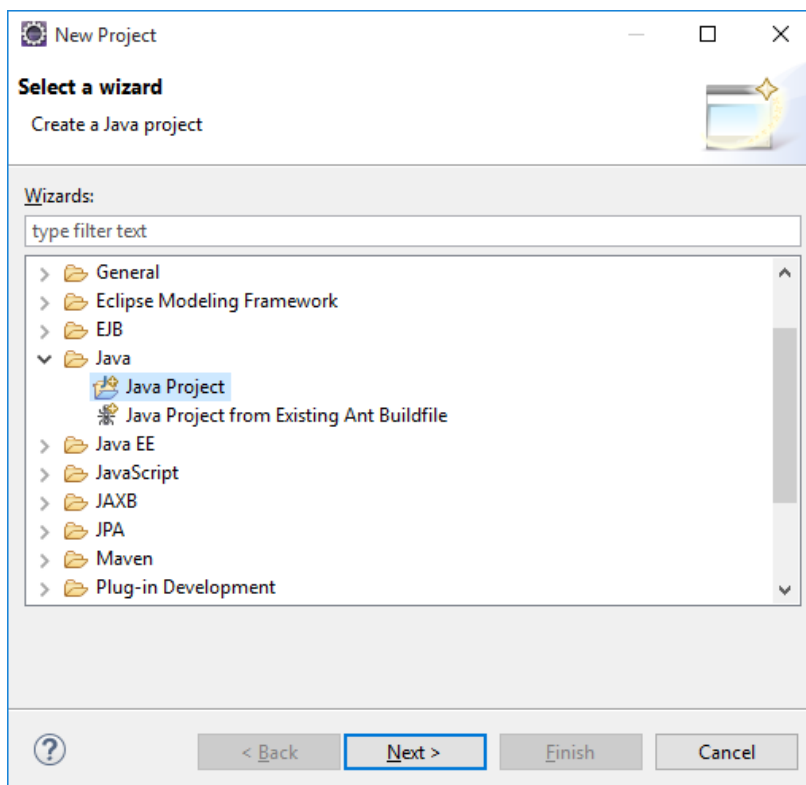


7. Membuat Client dari REST API

7.1 Menggunakan Pustaka Jersey

Framework Jersey mempunyai suatu pustaka client REST yang dapat digunakan untuk menguji atau membangun client REST API sungguhan dalam Java. Penggunaan pustaka tersebut dituntun oleh tutorial di bawah ini.

Buat suatu proyek Java baru dan berikan nama misalnya *id.web.husni.pertama.client*.



New Java Project

Create a Java Project

Create a Java project in the workspace or in an external location.

Project name: id.web.husni.pertama.client

☒ Use default location

Location: C:\Users\husni\workspace\id.web.husni.pertama.client

Browse...

JRE

☒ Use an execution environment JRE: JavaSE-1.8

☐ Use a project specific JRE: jre1.8.0_121

☐ Use default JRE (currently 'jre1.8.0_121')

[Configure JREs...](#)

Project layout

☐ Use project folder as root for sources and class files

☒ Create separate folders for sources and class files

[Configure default...](#)

Working sets

☐ Add project to working sets

New...

Working sets:

Select...

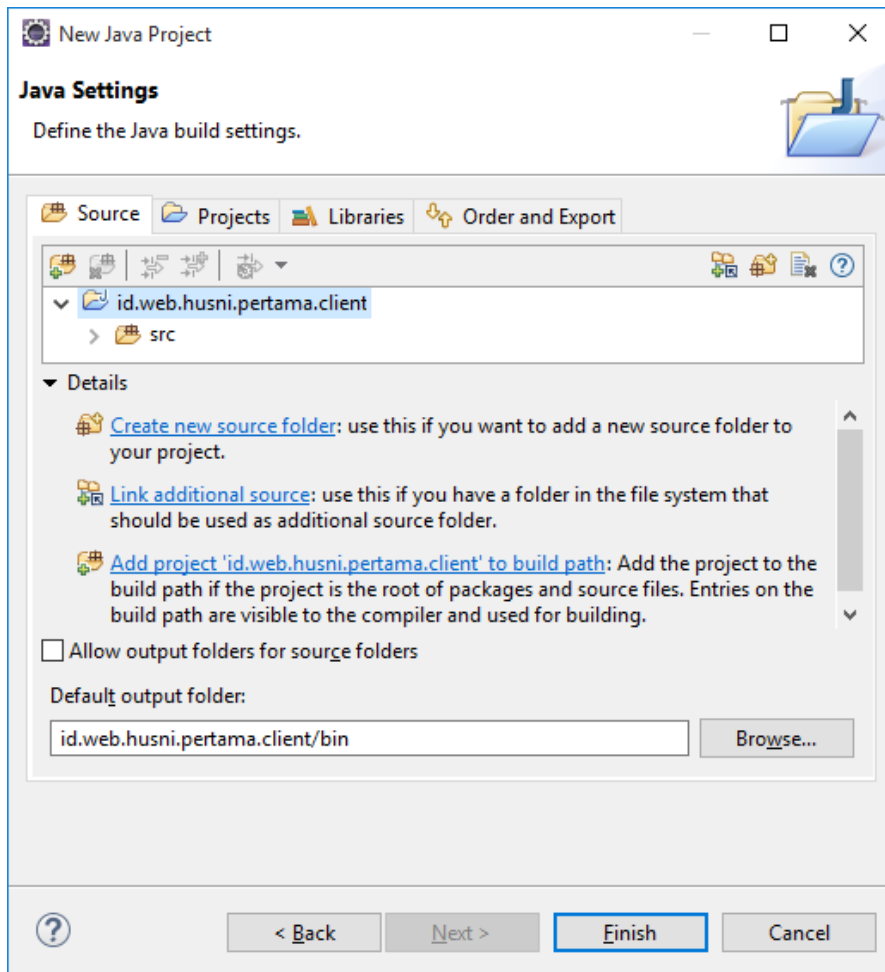
?

< Back

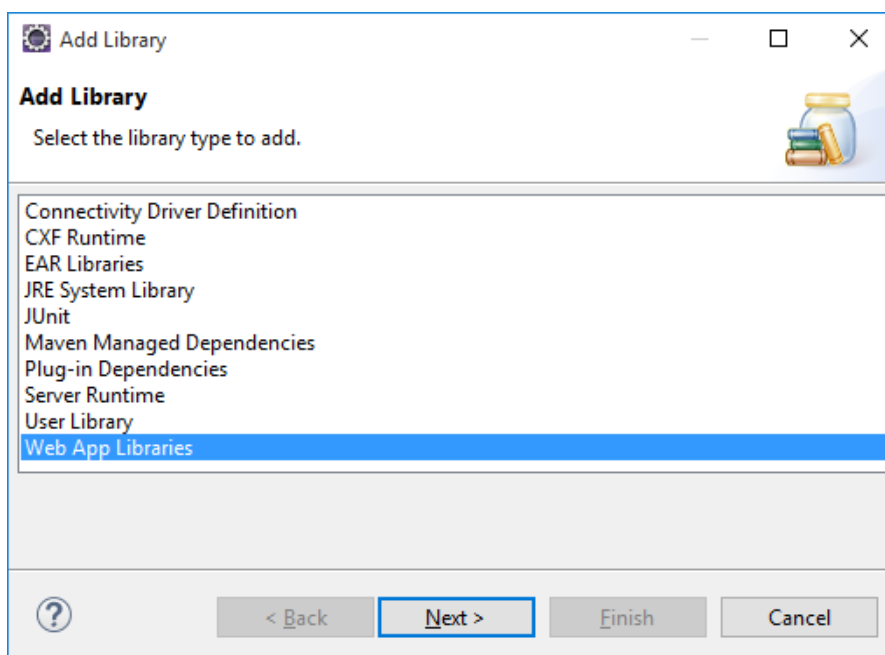
Next >

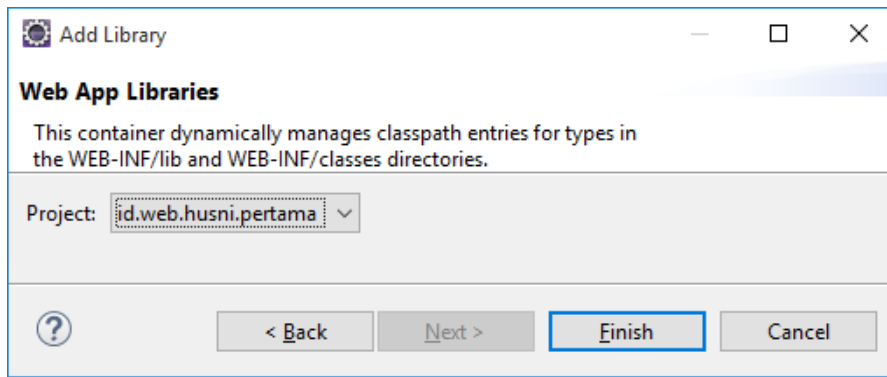
Finish

Cancel



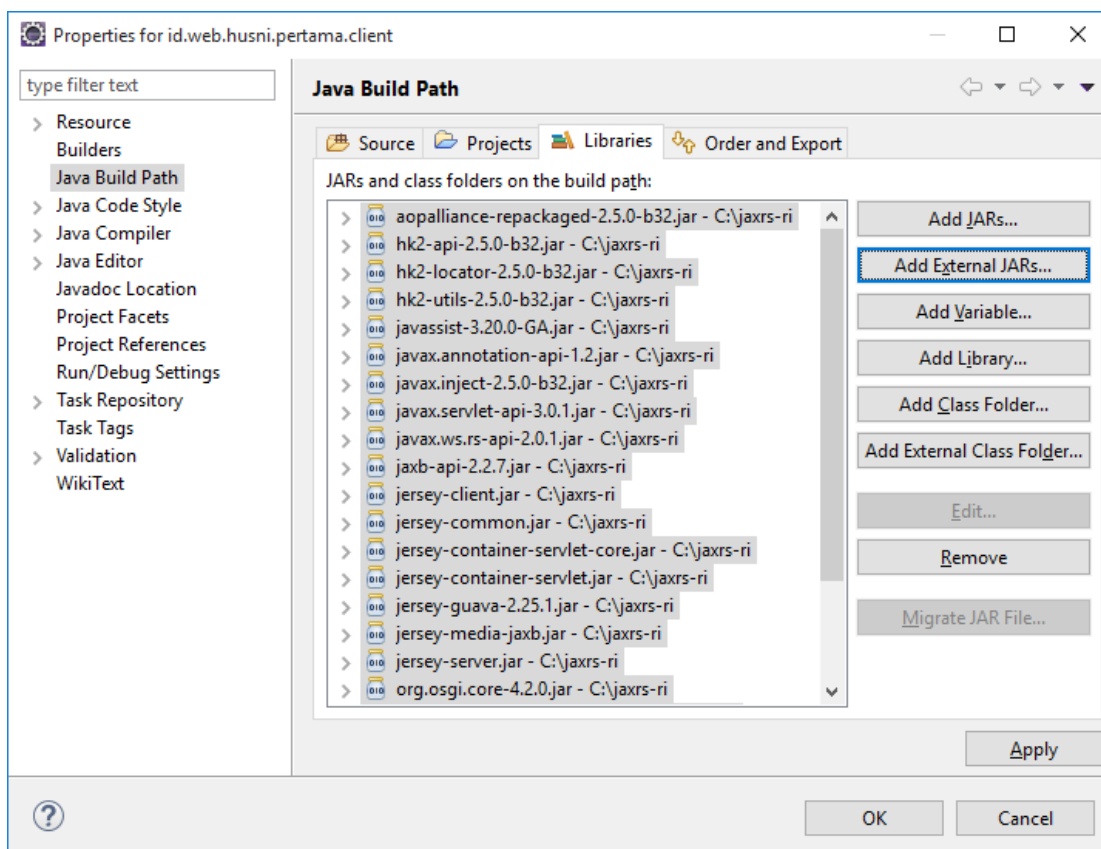
Tambahkan semua file .jar dari Jersey ke dalam proyek ini. Klik kanan nama proyek > Build Path > Add Libraries...



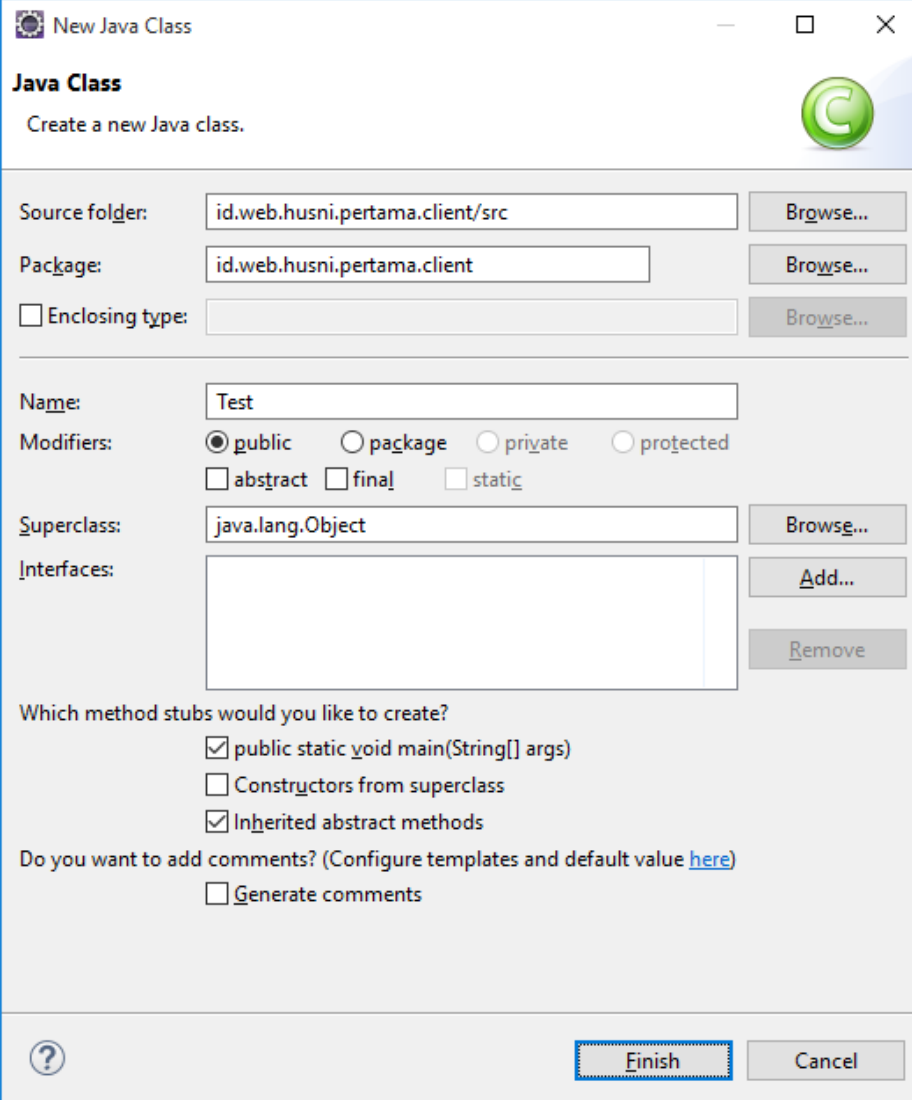


Umumnya proyek REST API Provider dan Customer dibangun pada mesin yang berbeda, sehingga cara di atas tidak dapat dilakukan. Cara yang banyak dilakukan adalah menyalin semua file JAR (misalnya awalnya ada di C:\jaxrs-ri) ke dalam proyek Java dengan langkah-langkah berikut:

- Klik kanan nama proyek > Build Path > Configure Build Path...
- Klik Add External JARs. Pada jendela Browse, pilih semua file JAR yang akan dimasukkan ke dalam proyek.
- Klik Apply atau OK.



Buat kelas Test (Test.java) seperti di bawah ini (Klik kanan nama proyek > New > Class)

The image shows a 'New Java Class' dialog box from an IDE. It has a title bar with a question mark icon, the text 'New Java Class', and standard window controls. The main area is titled 'Java Class' with a subtitle 'Create a new Java class.' and a green circular icon with a 'C'. The dialog is divided into several sections. The first section contains three text fields: 'Source folder:' with the value 'id.web.husni.pertama.client/src', 'Package:' with 'id.web.husni.pertama.client', and 'Enclosing type:' which is empty. Each field has a 'Browse...' button to its right. The second section contains a 'Name:' field with 'Test', a 'Modifiers:' section with radio buttons for 'public' (selected), 'package', 'private', and 'protected', and checkboxes for 'abstract', 'final', and 'static'. Below this is a 'Superclass:' field with 'java.lang.Object' and an 'Add...' button. The third section is titled 'Which method stubs would you like to create?' and contains three checkboxes: 'public static void main(String[] args)' (checked), 'Constructors from superclass' (unchecked), and 'Inherited abstract methods' (checked). The fourth section is titled 'Do you want to add comments? (Configure templates and default value [here](#))' and contains a 'Generate comments' checkbox (unchecked). At the bottom, there is a 'Finish' button and a 'Cancel' button.

New Java Class

Java Class
Create a new Java class.

Source folder: Browse...

Package: Browse...

☐ Enclosing type: Browse...

Name:

Modifiers: ☒ public ☐ package ☐ private ☐ protected
☐ abstract ☐ final ☐ static


Superclass: Browse...

Interfaces: Add... Remove

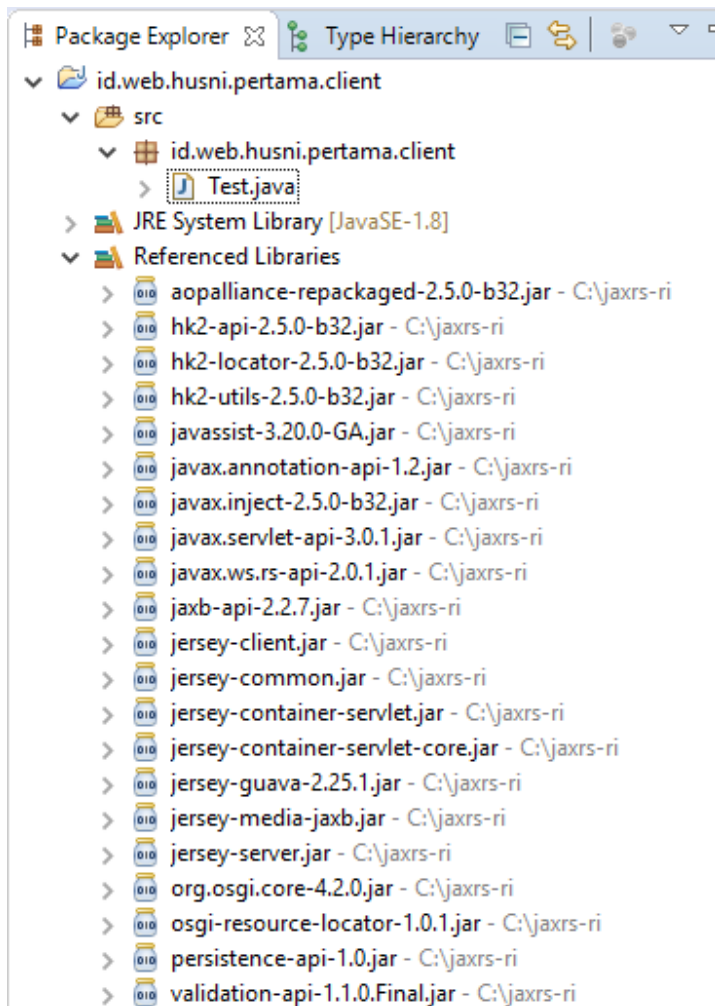
Which method stubs would you like to create?

☒ public static void main(String[] args)
☐ Constructors from superclass
☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))
☐ Generate comments

 **Finish** Cancel

Sekarang struktur direktori dari aplikasi client kita menjadi seperti pada Gambar di bawah ini.



Kode lengkap dari file Test.java adalah sebagai berikut:

```
package id.web.husni.pertama.client;

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.Response;
import javax.ws.rs.core.UriBuilder;

import org.glassfish.jersey.client.ClientConfig;

public class Test {

    public static void main(String[] args) {
        ClientConfig config = new ClientConfig();

        Client client = ClientBuilder.newClient(config);

        WebTarget target = client.target(getBaseURI());

        String response = target.path("rest").path("hello")
            .request().accept(MediaType.TEXT_PLAIN)
            .get(Response.class).toString();
    }
}
```

```

String plainAnswer = target.path("rest").path("hello")
    .request().accept(MediaType.TEXT_PLAIN)
    .get(String.class);

String xmlAnswer = target.path("rest").path("hello")
    .request().accept(MediaType.TEXT_XML)
    .get(String.class);

String htmlAnswer= target.path("rest").path("hello")
    .request().accept(MediaType.TEXT_HTML)
    .get(String.class);

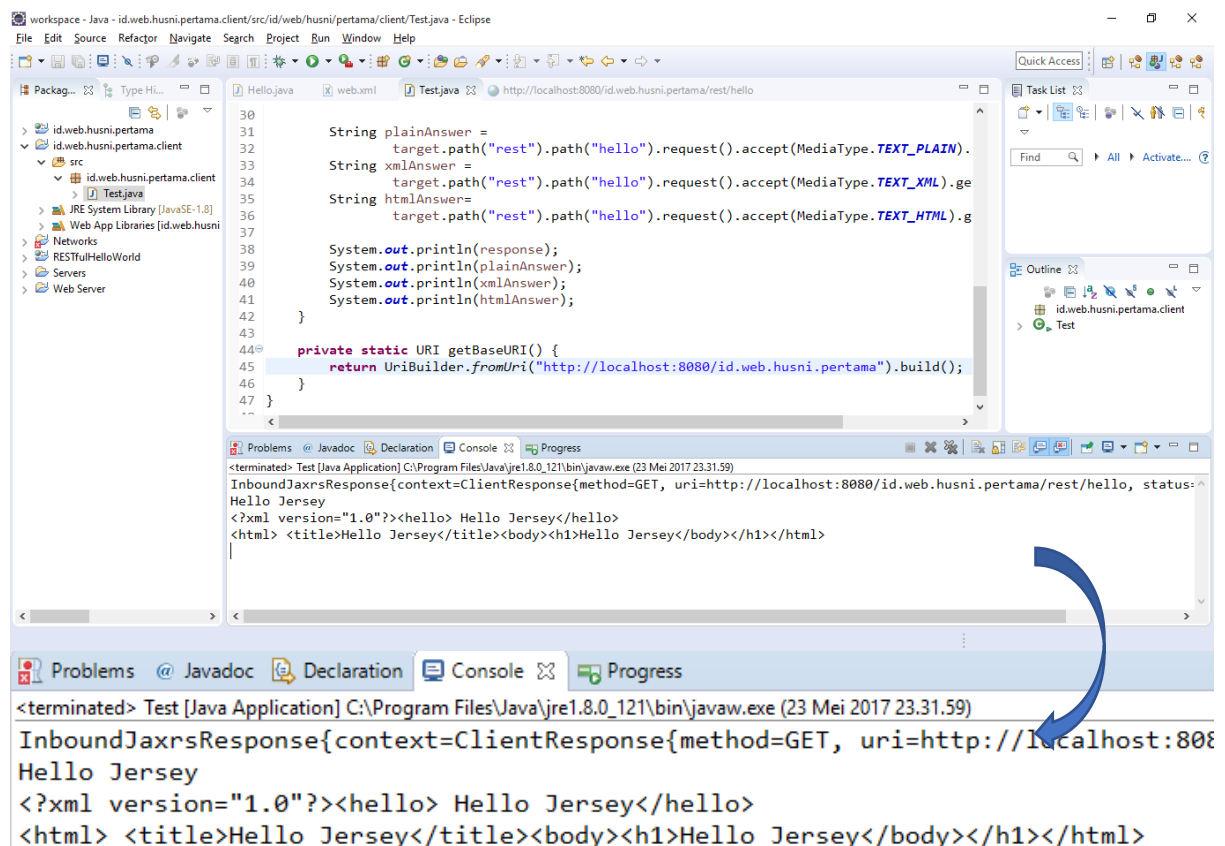
System.out.println(response);
System.out.println(plainAnswer);
System.out.println(xmlAnswer);
System.out.println(htmlAnswer);
}

private static URI getBaseURI() {
    return UriBuilder.fromUri("http://localhost:8080/id.web.husni.pertama").build();
}
}

```

Bagaimana mengujinya?

1. Pastikan REST API server dari proyek id.web.husni.pertama telah berjalan. Ikuti langkah-langkah sebelumnya.
2. Eksekusi file Test.java. Cara paling mudah adalah klik kanan file Test.java dan pilih Run As > Java Application.



7.2 Menggunakan Kelas URL

Apakah client REST API harus dibuat dengan Jersey? TIDAK. Kita dapat menggunakan pustaka lain atau bahkan tanpa pustaka tambahan sama sekali. Kita dapat menggunakan kelas jaringan built-in Java 8 yang bernama URL. Bagaimana cara membuat client REST API dengan pendekatan ini?

Berikut adalah kode program dari file TestURL.java yang berfungsi sebagai client dari REST API:

```
package id.web.husni.pertama.client;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.net.HttpURLConnection;
import java.net.MalformedURLException;
import java.net.URL;

public class TestURL {
    public static void main(String[] args) {
        try {
            URL url = new URL("http://localhost:8080/id.web.husni.pertama/rest/hello");
            HttpURLConnection conn = (HttpURLConnection) url.openConnection();
            conn.setRequestMethod("GET");

            if (conn.getResponseCode() != 200) {
                throw new RuntimeException("Failed : HTTP error code : "
                    + conn.getResponseCode());
            }

            BufferedReader br = new BufferedReader(new InputStreamReader(
                (conn.getInputStream())));

            String output;
            System.out.println("Output from Server .... \n");
            while ((output = br.readLine()) != null) {
                System.out.println(output);
            }

            conn.disconnect();

        } catch (MalformedURLException e) {
            e.printStackTrace();
        } catch (IOException e) {
            e.printStackTrace();
        }
    }
}
```

Saat file TestURL dieksekusi, diperoleh tampilan berikut di Console:

Output from Server

```
<html> <title>Hello Jersey</title><body><h1>Hello Jersey</body></h1></html>
```

Sekarang, kita punya 2 pilihan untuk mengakses Provider REST API, yaitu menggunakan pustaka Jersey atau kelas URL bawaan Java.

8. RESTful Web Service dan JAXB

JAX-RS mendukung pembuatan otomatis dari data XML dan JSON melalui JAXB. Sebagai pengantar mengenai XML dan penanganannya di Java silakan membaca tutorial singkatnya di <http://www.vogella.com/tutorials/JavaXML/article.html>. Sedangkan pengantar mengenai JAXB silakan melihatnya di <http://www.vogella.com/tutorials/JAXB/article.html>. Kita dapat melanjutkan tutorial ini tanpa harus membaca kedua tutorial tersebut (XML dan JAXB), tetapi pemahaman yang lebih baik terhadap keduanya akan lebih memudahkan memahami apa yang di bahas di sini.

8.1. Membuat Proyek REST API JAXB

Buatlah suatu *Dynamic Web Project* baru dan beri nama *id.web.husni.jersey.jaxb*. Pastikan pilihan web.xml deployment descriptor dicentang.

8.2. Mengkonfigurasi Pemanfaatan Jersey

Silakan membaca kembali bagaimana setup manual dari Jersey ke dalam proyek Web Dinamis di Eclipse (cukup *drag* dan *drop*).

8.3. Membuat File-file Web Service

Buat kelas dari domain yang ditangani (Todo.java), ini sering dinamakan sebagai model.

```
package id.web.husni.jersey.jaxb;

import javax.xml.bind.annotation.XmlRootElement;

@XmlRootElement
// JAX-RS mendukung pemetaan otomatis dari kelas terannotasi JAXB ke XML dan JSON
// Keren bukan?
public class Todo {
    private String summary;
    private String description;

    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; }
}
```

Buat kelas sumber daya berikut (TodoResource.java). Kelas ini seherhananya mengembalikan suatu instance dari kelas Todo.

```
package id.web.husni.jersey.jaxb;

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

@Path("/todo")
public class TodoResource {
    // Metode ini dipanggil jika requestnya XML
}
```

```

@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;
}

// Ini dapat digunakan untuk menguji integrasi dengan 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;
}
}

```

Lakukan perubahan terhadap web.xml sehingga menjadi:

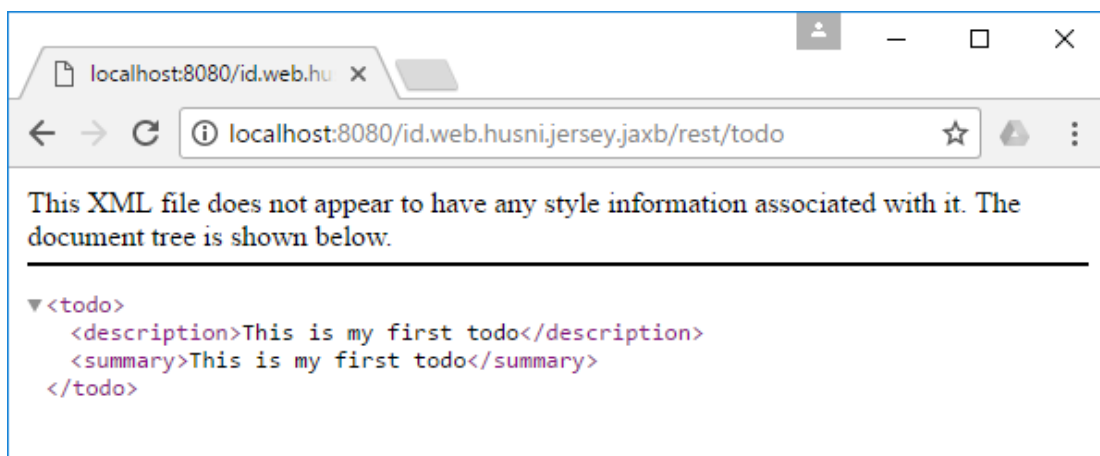
```

<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://java.sun.com/xml/ns/javaee"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd" id="WebApp_ID" version="3.0">
    <display-name>id.web.husni.jersey.jaxb</display-name>
    <servlet>
        <servlet-name>Jersey JAXB 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>id.web.husni.jersey.jaxb</param-value>
        </init-param>
        <load-on-startup>1</load-on-startup>
    </servlet>
    <servlet-mapping>
        <servlet-name>Jersey JAXB REST Service</servlet-name>
        <url-pattern>/rest/*</url-pattern>
    </servlet-mapping>
</web-app>

```

Jalankan aplikasi web ini di dalam Eclipse dan validasikan bahwa kita dapat mengakses layanan ini. Aplikasi kita harusnya tersedia di bawah URL berikut:

<http://localhost:8080/id.web.husni.jersey.jaxb/rest/todo>



8.4. Membuat Client

Buat suatu proyek Java baru bernama *id.web.husni.jersey.jaxb.client* dan tambahkan file-file .jar dari Jersey ke dalam proyek melalui jendela Build Path (seperti dijelaskan pada bagian 7 tutorial ini).

Buat kelas TodoTest (TodoTest.java) yang berfungsi sebagai client bagi REST API yang baru saja dibuat.

```

package id.web.husni.jersey.jaxb.client;

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;

import org.glassfish.jersey.client.ClientConfig;

public class TodoTest {

    public static void main(String[] args) {
        ClientConfig config = new ClientConfig();
        Client client = ClientBuilder.newClient(config);

        WebTarget target = client.target(getBaseURI());
        // Mendapatkan XML
    }
}

```

```

String xmlResponse = target.path("rest").path("todo").request()
    .accept(MediaType.TEXT_XML).get(String.class);
// Mendapatkan XML bagi aplikasi
String xmlAppResponse = target.path("rest").path("todo").request()
    .accept(MediaType.APPLICATION_XML).get(String.class);

// Untuk respon JSON juga tambahkan pustaka Jackson ke aplikasi web kita
// Dalam hal ini kita juga mengubah registrasi client menjadi
// ClientConfig config = new ClientConfig().register(JacksonFeature.class);
// Dapatkan JSON bagi Aplikasi
// System.out.println(target.path("rest").path("todo").request()
//     .accept(MediaType.APPLICATION_JSON).get(String.class));

System.out.println(xmlResponse);
System.out.println(xmlAppResponse);
}

private static URI getBaseURI() {
    return UriBuilder.fromUri(
        "http://localhost:8080/id.web.husni.jersey.jaxb").build();
}
}

```

Hasil yang diperoleh saat mengeksekusi kelas TodoTest di atas adalah

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?><todo><description>This is my first
todo</description><summary>This is my first todo</summary></todo>
<?xml version="1.0" encoding="UTF-8" standalone="yes"?><todo><description>This is my first
todo</description><summary>This is my first todo</summary></todo>

```

9. CRUD RESTful Web Service

Bagian ini memperlihatkan cara pembuatan suatu RESTful Web Service yang mempunyai fitur CRUD (Create, Read, Update, Delete) secara lengkap. Itu memungkinkan kita untuk memelihara daftar TODO di dalam aplikasi web kita via pemanggil HTTP. Kekurangan dari contoh ini adalah tidak adanya koneksi dari REST API ke database tertentu. Silakan anda menambahkan fitur koneksi via JDBC misalnya ke database MySQL atau SQLite.

9.1. Memulai Proyek

Pertama, mari kita buat suatu proyek web dinamis baru dan berikan nama *id.web.husni.jersey.todo*. Tambahkan pustaka Jersey ke dalam proyek tersebut.

Lakukan perubahan terhadap file web.xml sehingga menjadi sebagai berikut:

```

<?xml version="1.0" encoding="UTF-8"?>
<web-app xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xmlns="http://java.sun.com/xml/ns/javaee"
    xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://java.sun.com/xml/ns/javaee/web-app_3_0.xsd"
    id="WebApp_ID" version="3.0">
    <display-name>id.web.husni.jersey.todo</display-name>
    <servlet>
        <servlet-name>Jersey CRUD 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>id.web.husni.jersey.todo</param-value>
        </init-param>
    </servlet>

```



```

    <load-on-startup>1</load-on-startup>
</servlet>
<servlet-mapping>
    <servlet-name>Jersey CRUD REST Service</servlet-name>
    <url-pattern>/rest/*</url-pattern>
</servlet-mapping>
</web-app>

```

Sekarang buatlah model data (kelas Todo, file Todo.java) dan Singleton (kelas TodoDao, file TodoDao.java) berikut yang akan berfungsi sebagai data provider bagi model. Kita menggunakan implementasi berbasis pada enumerasi. Kelas Todo dianotasi dengan suatu anotasi JAXB.

Kode program dari Todo.java adalah

```

package id.web.husni.jersey.todo;

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;
    }

    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; }
}

```

Sedangkan kode dari file TodoDao.java adalah

```

package id.web.husni.jersey.todo;

import java.util.HashMap;
import java.util.Map;

import id.web.husni.jersey.todo.Todo;

public enum TodoDao {
    instance;

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

    private TodoDao() {
        Todo todo = new Todo("1", "Learn REST");
        todo.setDescription("Read http://www.learnweb.com/tutorials/REST/article.html");
        contentProvider.put("1", todo);
        todo = new Todo("2", "Do something");
    }
}

```

```

        todo.setDescription("Read complete http://www.amaron.com");
        contentProvider.put("2", todo);
    }

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

```

9.2. Membuat Form HTML Sederhana

Layana REST dapat diakses melalui form HTML. Form HTML berikut akan membolehkan kita mem-post data baru ke web service. Silakan buat halaman web bernama create_todo.html berikut di dalam folder WebContent dari proyek yang dibuat.

```

<!DOCTYPE html>
<html>
  <head>
    <title>Form to create a new resource</title>
  </head>
  <body>
    <form action=" ../id.web.husni.jersey.todo/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>

```

9.3. Rest API Provider

Sekarang adalah membuat kode yang bertugas melayani request dari customer REST API. Kelas TodoResource dan TodosResource ini berperan sebagai pengelola sumber daya REST (API Provider).

Berikut ini adalah kode lengkap dari file TodoResource.java:

```

package id.web.husni.jersey.todo;

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 id.web.husni.jersey.todo.TODODao;

```

```

import id.web.husni.jersey.todo.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;
    }

    //Integrasi Aplikasi
    @GET
    @Produces({MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON})
    public Todo getTodo() {
        Todo todo = TodoDao.instance.getModel().get(id);
        if(todo==null)
            throw new RuntimeException("Get: Todo with " + id + " not found");
        return todo;
    }

    // untuk browser
    @GET
    @Produces(MediaType.TEXT_XML)
    public Todo getTodoHTML() {
        Todo todo = TodoDao.instance.getModel().get(id);
        if(todo==null)
            throw new RuntimeException("Get: Todo with " + id + " not found");
        return todo;
    }

    @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.instance.getModel().remove(id);
        if(c==null)
            throw new RuntimeException("Delete: Todo with " + id + " not found");
    }

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

```

Sedangkan kode lengkap dari file TodosResource.java adalah seperti ini:

```

package id.web.husni.jersey.todo;

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 id.web.husni.jersey.todo.TODODao;
import id.web.husni.jersey.todo.TODO;

// Akan memetakan sumber daya ke todos URL
@Path("/todos")
public class TodosResource {

    // Membolehkan meng-insert obyek kontekstual ke dalam kelas,
    // misal ServletContext, Request, Response, UriInfo
    @Context
    UriInfo uriInfo;
    @Context
    Request request;

    // Mengembalikan daftar todo untuk pengguna dalam browser
    @GET
    @Produces(MediaType.TEXT_XML)
    public List<TODO> getTodosBrowser() {
        List<TODO> todos = new ArrayList<TODO>();
        todos.addAll(TODODao.INSTANCE.getModel().values());
        return todos;
    }

    // Mengembalikan daftar todo untuk aplikasi
    @GET
    @Produces({ MediaType.APPLICATION_XML, MediaType.APPLICATION_JSON })
    public List<TODO> getTodos() {
        List<TODO> todos = new ArrayList<TODO>();
        todos.addAll(TODODao.INSTANCE.getModel().values());
        return todos;
    }

    // Mengembalikan jumlah todo
    // Gunakan http://localhost:8080/id.web.husni.jersey.todo/rest/todos/jumlah
    // untuk mendapatkan jumlah total dari record.
    @GET
    @Path("count")
    @Produces(MediaType.TEXT_PLAIN)
    public String getCount() {
        int count = TODODao.INSTANCE.getModel().size();
        return String.valueOf(count);
    }

    @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.instance.getModel().put(id, todo);

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

// Mendefinisikan bahwa parameter path berikutnya setelah todos
// diperlakukan sebagai parameter dan dilewatkan ke TodoResources
// Memungkinkan untuk memasukkan
// http://localhost:8080/id.web.husni.jersey.todo/rest/todos/1
// 1 akan diprlakukan sebagai parameter todo dan dilewatkan ke TodoResource
@Path("/{todo}")
public TodoResource getTodo(@PathParam("todo") String id) {
    return new TodoResource(uriInfo, request, id);
}
}

```

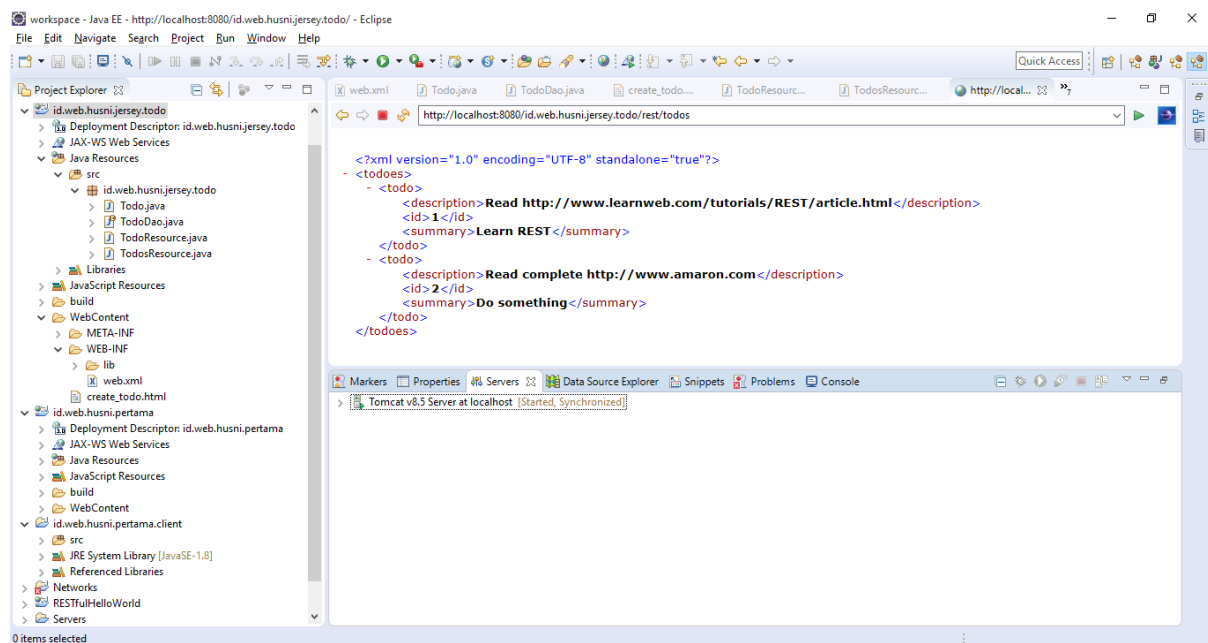
Kelas TodosResource ini menggunakan anotasi `@PathParam` untuk mendefinisikan bahwa id disisipkan sebagai parameter.

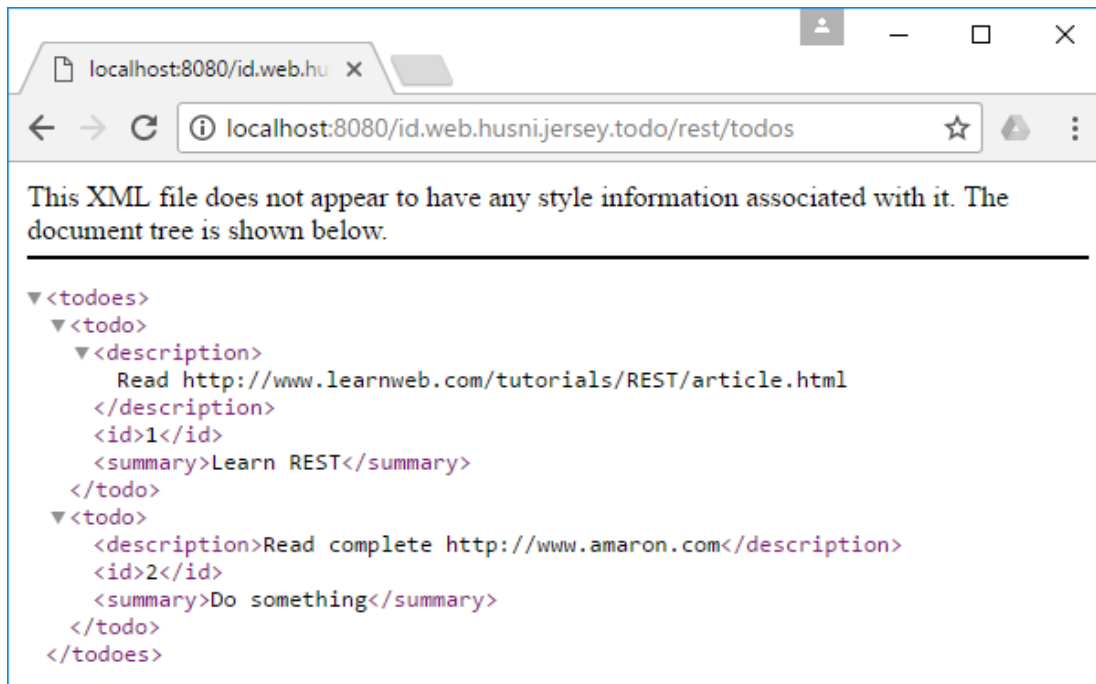
9.4. Menjalankan REST API

Run your web application in Eclipse and test the availability of your REST service under:

<http://localhost:8080/id.web.husni.jersey.todo/rest/todos>

You should see the XML representation of your TODO items.





Untuk melihat jumlah item TODO gunakan:

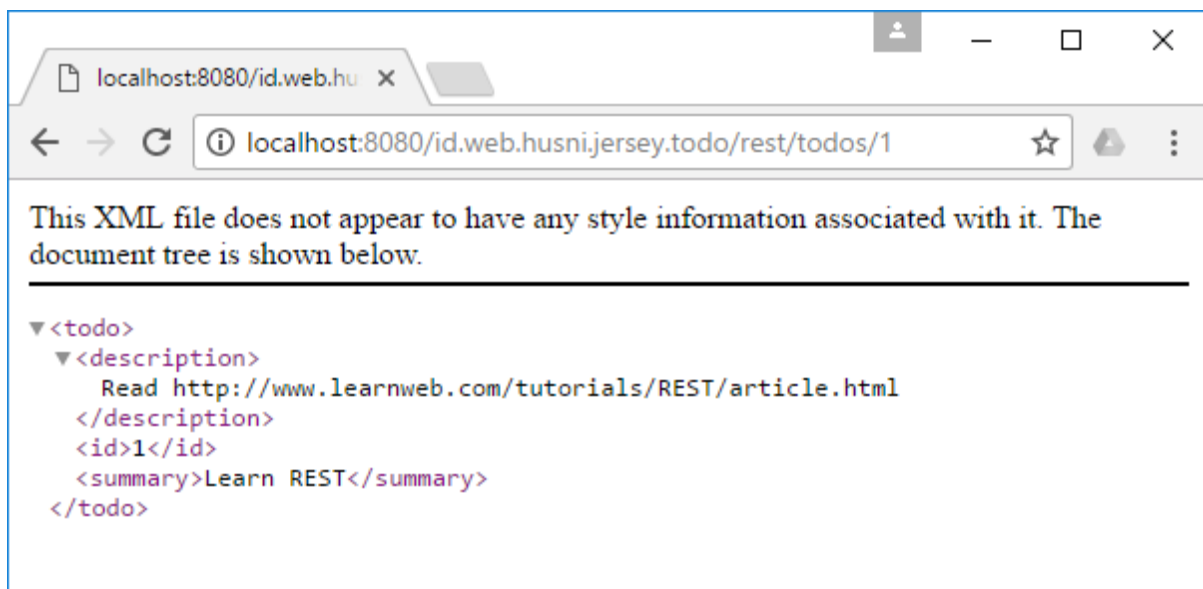
<http://localhost:8080/id.web.husni.jersey.todo/rest/todos/count>

Sedangkan untuk melihat suatu TODO yang eksis gunakan:

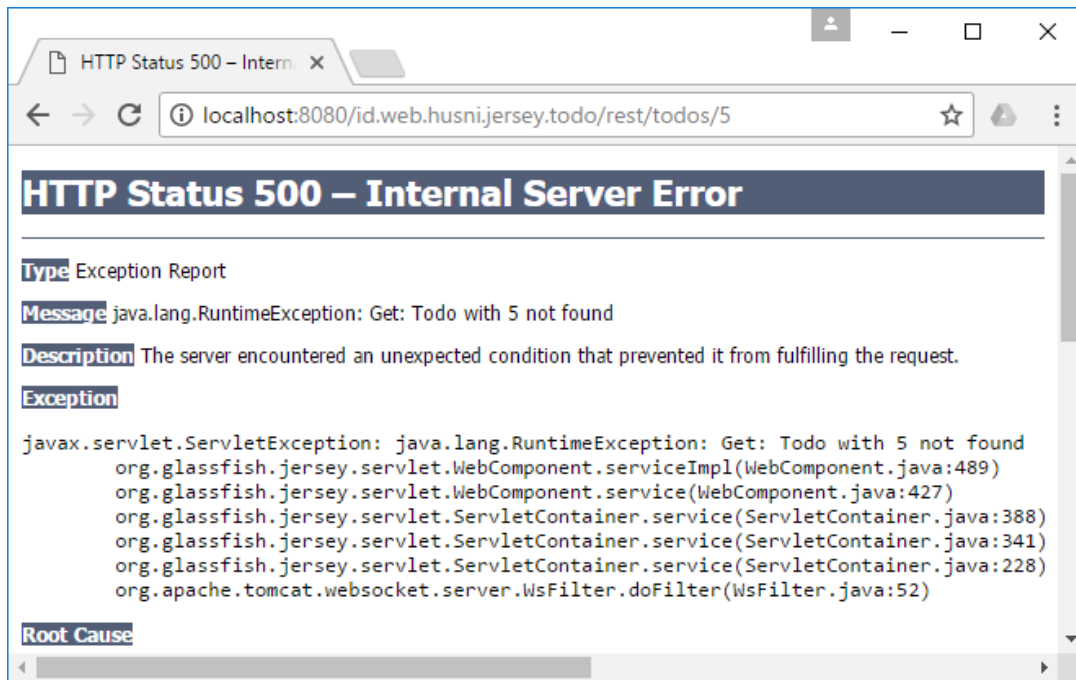
<http://localhost:8080/id.web.husni.jersey.todo/rest/todos/{id}>", misalnya:

<http://localhost:8080/id.web.husni.jersey.todo/rest/todos/1>

untuk melihat TODO dengan ID 1.



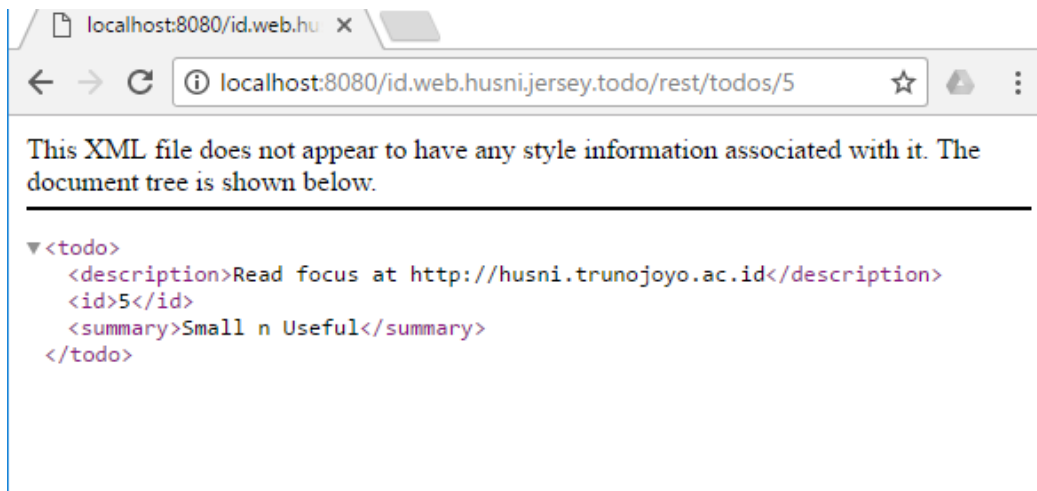
Saat ini kita hanya mempunyai TODOs dengan ID 1 dan 2. Semua request ke ID lain akan menghasilkan kode error HTTP seperti di bawah ini:



Sekarang kita lihat bagaimana metode POST digunakan untuk sumber daya (data) baru ke dalam sistem.

A screenshot of a web browser window showing a form titled 'Form to create a new res'. The form is located at 'localhost:8080/id.web.husni.jersey.todo/create_todo.html'. It contains the following fields:
ID: 5
Summary: Small n Useful
Description: Read focus at <http://husni.trunojoyo.ac.id>
A 'Submit' button is located at the bottom left of the form.

Dan pemanggilan URL <http://localhost:8080/id.web.husni.jersey.todo/rest/todos/5> telah memberikan hasil seperti gambar di bawah ini:



Perlu diingat bahwa dengan web browser kita hanya dapat mengirimkan request GET dan POST HTTP. Bagian berikutnya akan memaparkan bagaimana menggunakan pustaka client Jersey untuk mengirimkan request get, post dan delete.

9.5. Membuat Client

Untuk menguji web service yang dibuat kita dapat membuat suatu kelas baru di dalam proyek server kita (hanya untuk testing). Proyek ini sudah siap dengan semua pustaka yang disyaratkan sehingga pembuatan proyek baru nantinya akan lebih mudah dan tertata.

Buat kelas Tester berikut:

```
package id.web.husni.jersey.todo;

import java.net.URI;

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 javax.ws.rs.core.Form;
import javax.ws.rs.core.MediaType;
import javax.ws.rs.core.Response;
import javax.ws.rs.core.UriBuilder;

import org.glassfish.jersey.client.ClientConfig;

import id.web.husni.jersey.todo.TODO;

public class Tester {
    public static void main(String[] args) {

        ClientConfig config = new ClientConfig();
        Client client = ClientBuilder.newClient(config);
        WebTarget service = client.target(getBaseURI());

        // Membuat satu todo
        TODO todo = new TODO("3", "Blabla");
        Response response =
service.path("rest").path("todos").path(todo.getId()).request(MediaType.APPLICATION_XML).put(Entity.entity(todo, MediaType.APPLICATION_XML), Response.Status.CREATED);

        // Mengembalikan kode 201 == created resource
```



```

        System.out.println(response.getStatus());

        // Mendapatkan Todos

        System.out.println(service.path("rest").path("todos").request().accept(MediaType.TEXT_XML).
            get(String.class));

        // Mendapatkan JSON bagi aplikasi
        //
        System.out.println(service.path("rest").path("todos").request().accept(MediaType.APPLICATION_
            JSON).get(String.class));

        // Mendapatkan XML bagi aplikasi

        System.out.println(service.path("rest").path("todos").request().accept(MediaType.APPLICATION_
            XML).get(String.class));

        //Mendapatkan Todo dengan id 1
        Response checkDelete =
            service.path("rest").path("todos/1").request().accept(MediaType.APPLICATION_XML).get();

        //Menghapus Todo dengan id 1
        service.path("rest").path("todos/1").request().delete();

        //Mendapatkan semua Todos id 1 yang akan dihapus

        System.out.println(service.path("rest").path("todos").request().accept(MediaType.APPLICATION_
            XML).get(String.class));

        //Membuat suatu Todo
        Form form =new Form();
        form.param("id", "4");
        form.param("summary", "Demonstration of the client lib for forms");
        response =
            service.path("rest").path("todos").request().post(Entity.entity(form, MediaType.APPLICATION_
                FORM_URLENCODED), Response.class);
        System.out.println("Form response " + response.getStatus());

        //Ambil semua todo, id 4 baru saja dibuatkan

        System.out.println(service.path("rest").path("todos").request().accept(MediaType.APPLICATION_
            XML).get(String.class));
    }

    private static URI getBaseURI() {
        return UriBuilder.fromUri("http://localhost:8080/id.web.husni.jersey.todo").build();
    }
}

```

Hasil yang diperoleh saat eksekusi client ini adalah

```

201
<?xml version="1.0" encoding="UTF-8"
standalone="yes"?><todoes><todo><description>Read
http://www.learnweb.com/tutorials/REST/article.html</description><id>1</id><summary>Learn REST</summary></todo><todo><description>Read complete
http://www.amaron.com</description><id>2</id><summary>Do
something</summary></todo><todo><id>3</id><summary>Blabla</summary></todo><todo><d
escription>Read focus at
http://husni.trunojoyo.ac.id</description><id>5</id><summary>Small n
Useful</summary></todo></todoes>
<?xml version="1.0" encoding="UTF-8"
standalone="yes"?><todoes><todo><description>Read

```

```

http://www.learnweb.com/tutorials/REST/article.html</description><id>1</id><summary>Learn REST</summary></todo><todo><description>Read complete
http://www.amaron.com</description><id>2</id><summary>Do
something</summary></todo><todo><id>3</id><summary>Blabla</summary></todo><todo><d
escription>Read focus at
http://husni.trunojoyo.ac.id</description><id>5</id><summary>Small n
Useful</summary></todo></todoes>
<?xml version="1.0" encoding="UTF-8"
standalone="yes"?><todoes><todo><description>Read complete
http://www.amaron.com</description><id>2</id><summary>Do
something</summary></todo><todo><id>3</id><summary>Blabla</summary></todo><todo><d
escription>Read focus at
http://husni.trunojoyo.ac.id</description><id>5</id><summary>Small n
Useful</summary></todo></todoes>
Form response 200
<?xml version="1.0" encoding="UTF-8"
standalone="yes"?><todoes><todo><description>Read complete
http://www.amaron.com</description><id>2</id><summary>Do
something</summary></todo><todo><id>3</id><summary>Blabla</summary></todo><todo><i
d>4</id><summary>Demonstration of the client lib for
forms</summary></todo><todo><description>Read focus at
http://husni.trunojoyo.ac.id</description><id>5</id><summary>Small n
Useful</summary></todo></todoes>

```

9.6. Mengakses Layanan REST via Halaman HTML

Contoh di atas memperlihatkan bagaimana suatu form HTML mengirimkan request bermetode POST kepada layanan REST yang dibuat. Berikut ini adalah tampilan dari Form tersebut.

10. Referensi

- Jersey Homepage: <http://jersey.java.net/>
- JSR 311: <http://jcp.org/aboutJava/communityprocess/final/jsr311/index.html>
- JAXB Tutorial: <http://www.vogella.com/tutorials/JAXB/article.html>