N_PROG SIST II_A6 - Texto de apoio

Site: EAD Mackenzie
Tema: PROGRAMAÇÃO DE SISTEMAS II {TURMA 03B} 2023/1
Livro: N_PROG SIST II_A6 - Texto de apoio

Impresso por: FELIPE BALDIM GUERRA .
Data: sexta, 5 mai 2023, 01:01

Descrição

Índice

1. DESENVOLVENDO WEB SERVICES RESTFUL

Web Services REStful

A interoperabilidade entre sistemas tem sido uma tendência bem evidente do mercado de TI de modo que, praticamente, qualquer aplicação corporativa desenvolvida nos dias de hoje possui integrações com outros sistemas, seja para backup, e-mail, storage, conversão de arquivos, busca de informações, processamento de imagens ou qualquer outra função entre um incontável número de possibilidades.

Web Services são uma forma padronizada de integrar sistemas, na qual uma aplicação oferece um serviço HTTP (**servidor**) e a outra o consome (**cliente**), e foram criados com o objetivo de possibilitar a interoperabilidade entre os sistemas.

Há, basicamente, dois modelos de implementação de Web Service: os baseados em **SOAP** (**Simple Object Access Protocol)** e os baseados no padrão **REST** (**Representational State Transfer**). Atualmente, os Web Services RESTful (implementados usando o padrão REST) vêm ganhando espaço no mercado e têm sido utilizados por empresas gigantes como Google, Facebook, Yahoo!, Amazon, eBay, Microsoft, entre outras, devido, principalmente, a sua simplicidade em relação aos Web Services SOAP. Isso chega a ser uma ironia, pois quem leva **Simple** no nome são os Web Services SOAP e não os Web Services RESTful. A tabela abaixo dá uma visão comparativa entre os dois tipos de Web Services:

REST	SOAP
mais simples a implementação	**menos simples** a implementação
menos *overhead* de comunicação	**mais** *overhead* de comunicação
Stateless – sem estado ou histórico	*Stateful* – guarda o estado ou histórico
menos segurança	**mais segurança**

Até pouco tempo atrás, não existia uma maneira “oficial” de desenvolver Web Services RESTful na plataforma Java, ou seja, não tínhamos uma API para implementar o Web Service. Isso acontece porque o REST é um modelo relativamente novo, e também porque ele usa basicamente o protocolo HTTP.

Entretanto, com o intuito de padronizar e simplificar a criação desse tipo de serviço em Java, foi criada a especificação **JAX-RS 2.0**, integrante da **plataforma Java EE** e descrita em detalhes no **JSR 339** (**Java Specification Request**), o qual pode ser consultado no link a seguir: <https://jcp.org/en/egc/view?id=339>.

A especificação **JAX-RS 2.0** define um conjunto de **anotações**, **classes** e **interfaces** para a desenvolvimento de Web Services RESTful e, como implementação de referência, temos o **projeto Jersey**, o qual utilizaremos em nossos exemplos, mas existem diversas outras implementações no mercado, tais como: **Apache CFX**, **RESTEasy**, **Restlet**, entre outros. Isso demonstra, mais uma vez, que o padrão REST vem ganhando espaço no mercado.

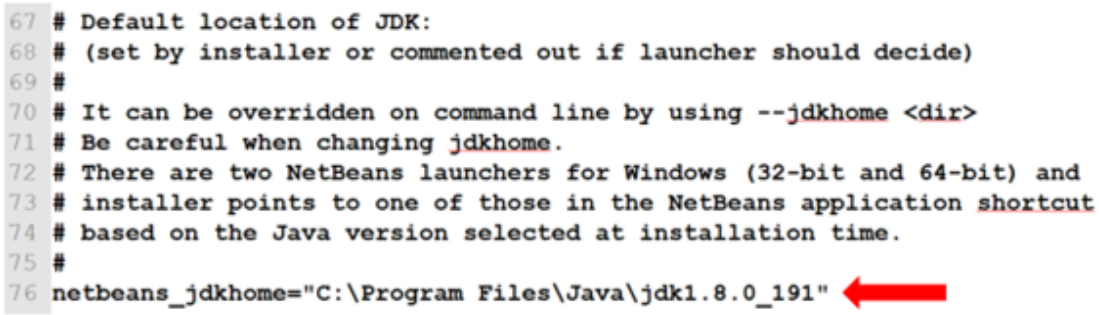
Implementação de Web Service RESTful

O desenvolvedor que conhece o padrão REST, com certeza estará um passo à frente dos demais. Por conta disso, implementaremos um Web Service seguindo esse padrão. O Web Service será desenvolvido utilizando a IDE **NetBeans 12.0 LTS** com o servidor de aplicação **GlassFish 5.1.0** já instalado e configurado.

Para os próximos passos de configuração, você precisará seguir rigorosamente as orientações, pois qualquer configuração realizada erroneamente não nos permitirá implementar e testar nosso Web Service. Isso já é uma preparação para o que a vida profissional lhe reserva, já que configurar e testar o ambiente de trabalho é uma responsabilidade do desenvolvedor.

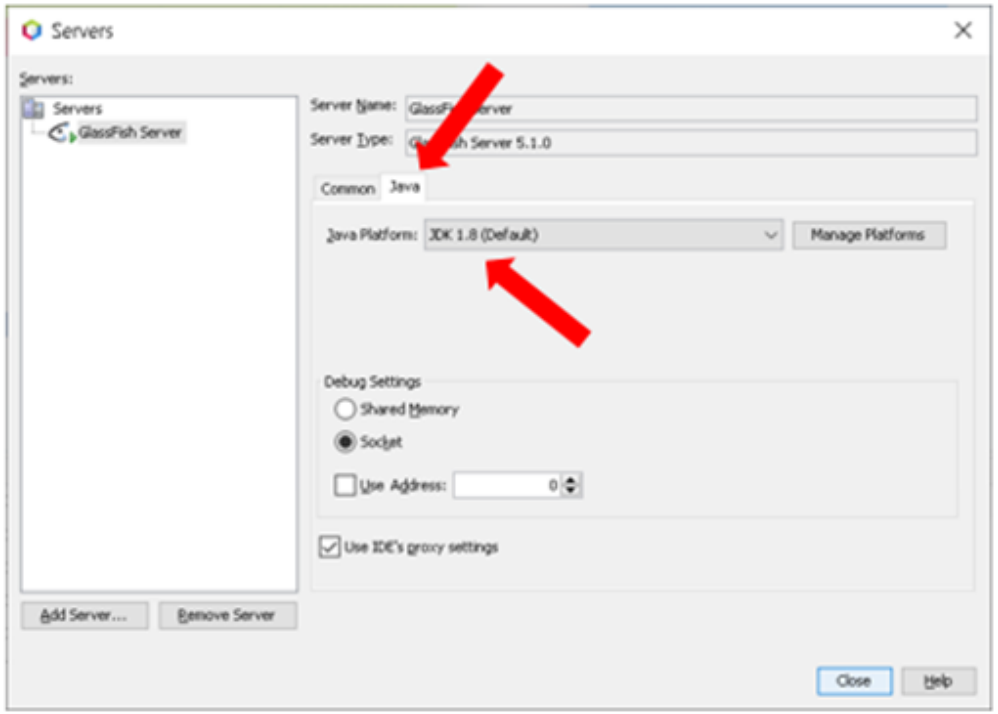
O primeiro ponto a ser observado são os requisitos de software para o funcionamento do servidor de GlassFish 5.1.0, conforme descrito no site a seguir: <https://glassfish.org/docs/5.1.0/release-notes/release-notes.html>. O GlassFish requer a versão 8 da JDK (Java Development Kit). Se você não a tiver instalada em seu computador, poderá baixar essa versão no link a seguir: <https://www.oracle.com/br/java/technologies/javase/javase-jdk8-downloads.html>.

Após instalar essa versão da JDK, é importante verificar se sua IDE NetBeans 12 também está configurada para trabalhar com essa versão da JDK. Para ter certeza de que ela está configurada corretamente, acesse o arquivo: **C:\Program Files\NetBeans-12.0\netbeans\etc\netbeans.conf** e verifique se a variável de ambiente do arquivo **netbeans_jdkhome** está com o valor correto, ou seja, o caminho no qual está instalado a JDK em seu computador. A variável **netbeans_jdkhome** deve estar configurada como no trecho abaixo:



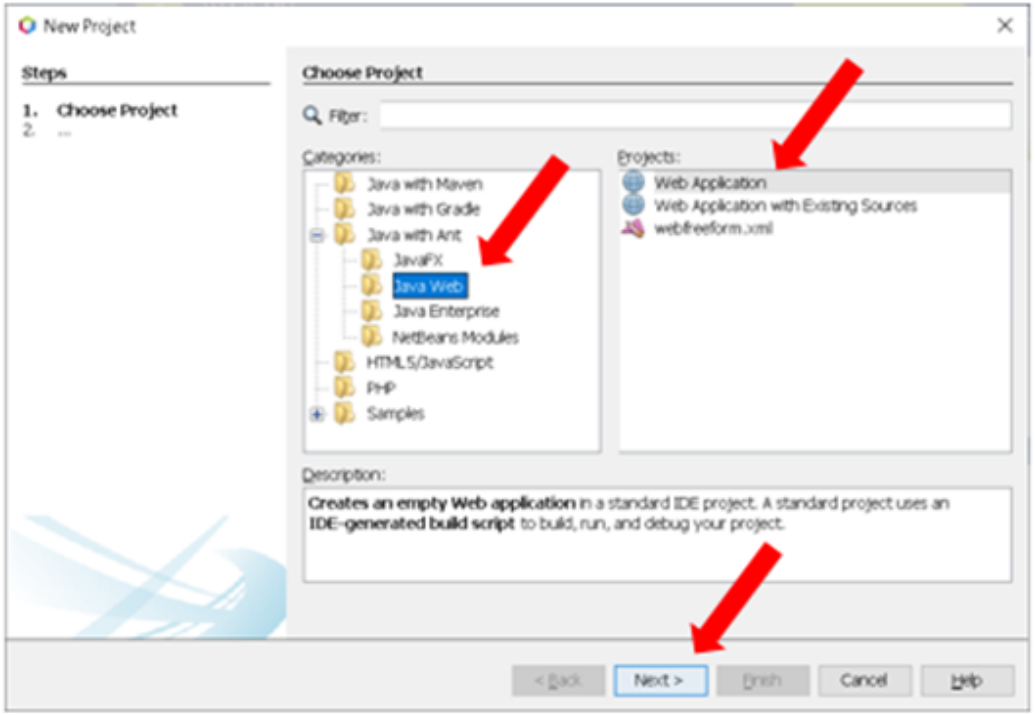
Fonte: Elaborada pelo autor.

Concluída essa configuração prévia no arquivo **netbeans.conf**, é possível checar no NetBeans se deu tudo certo. Basta acessar no menu principal **Tools -> Servers**, e teremos a janela abaixo. Selecione a aba **Java** e verifique se a **JDK 1.8** está selecionada como default.



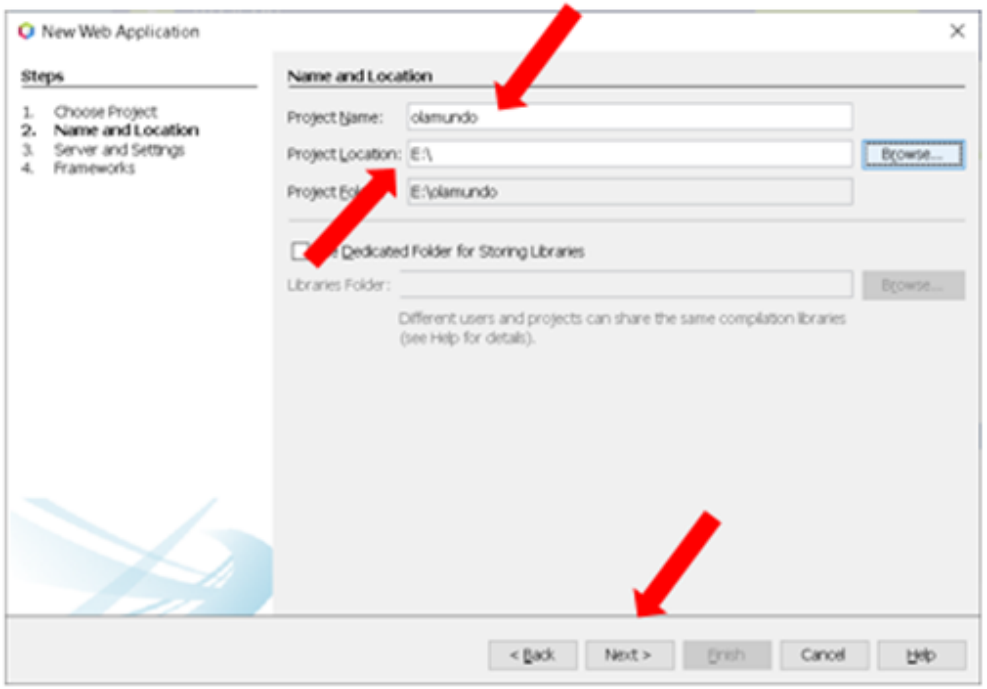
Fonte: Elaborada pelo autor.

Feita a verificação dos requisitos de software, podemos agora iniciar o desenvolvimento de um Web Service RESTful no **NetBeans**. Em primeiro lugar, é necessária a criação de um novo projeto **Web Application**. Para isso, no menu principal no NetBeans, clique em **File -> New Project**. Em seguida, será apresentada a janela abaixo. Selecione as opções **Java Web -> Web Application**, e depois clique em **Next**. A próxima tela ilustra esses passos.



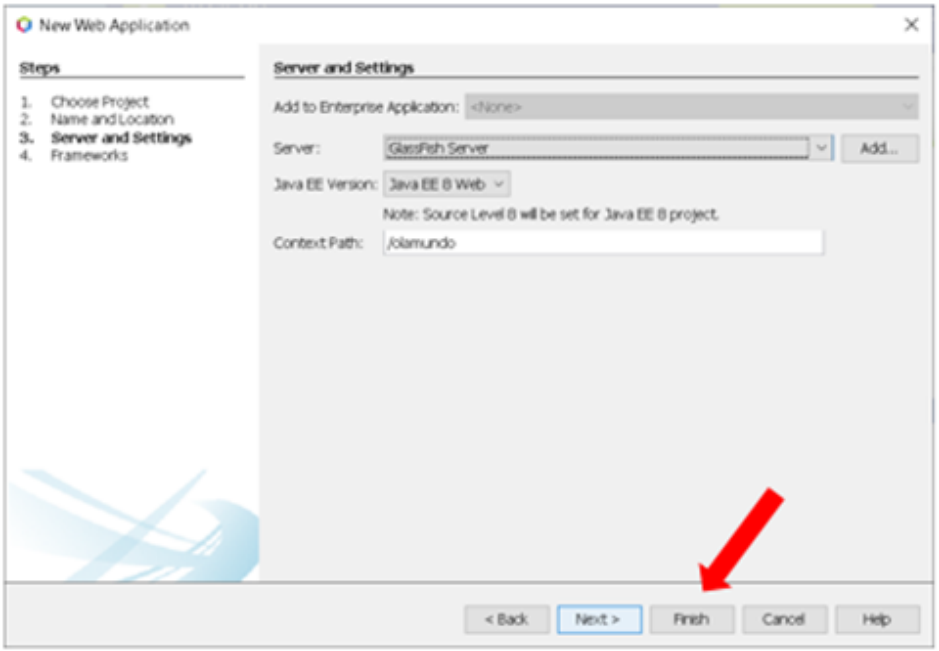
Fonte: Elaborada pelo autor.

Em seguida, aparecerá a janela abaixo. Dê um nome para o projeto (**Project Name**) e sua localização (**Project Location**). Em meu caso, coloquei o nome do projeto com **olamundo**, tudo minúsculo. Em seguida, clique em **Next**.



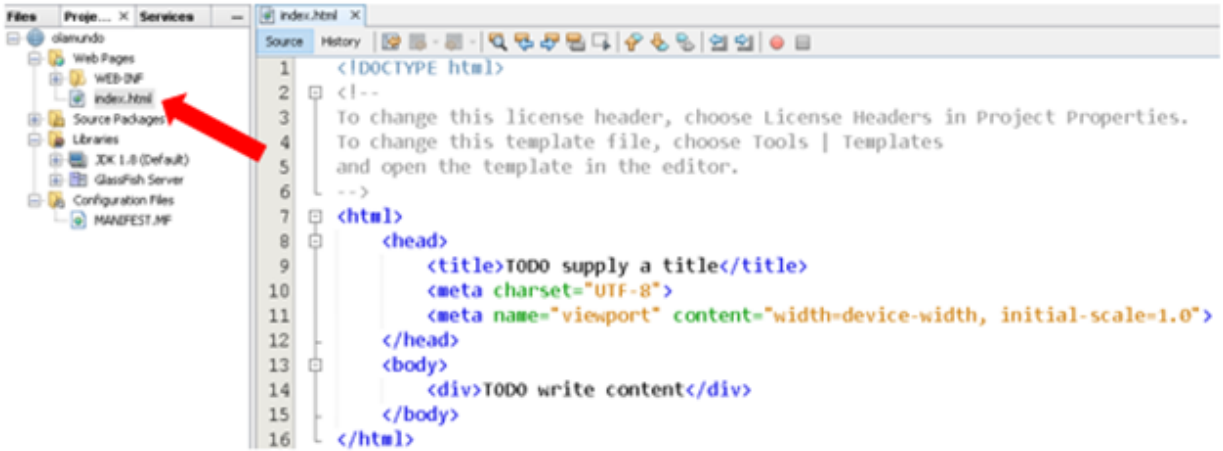
Fonte: Elaborada pelo autor.

Na próxima janela, configuramos o servidor de aplicação. Em nosso caso, utilizaremos o **GlassFish** que já está configurado, ou seja, não será necessário fazer nada, apenas clicar no botão **Finish**.



Fonte: Elaborada pelo autor.

Temos, assim, um projeto **Web Application**, ou seja, uma aplicação Web com a página inicial **index.html,** conforme ilustrado abaixo:

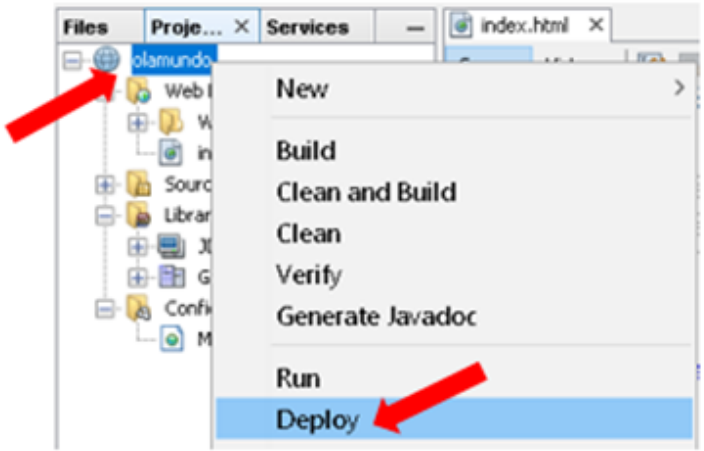


Fonte: Elaborada pelo autor.

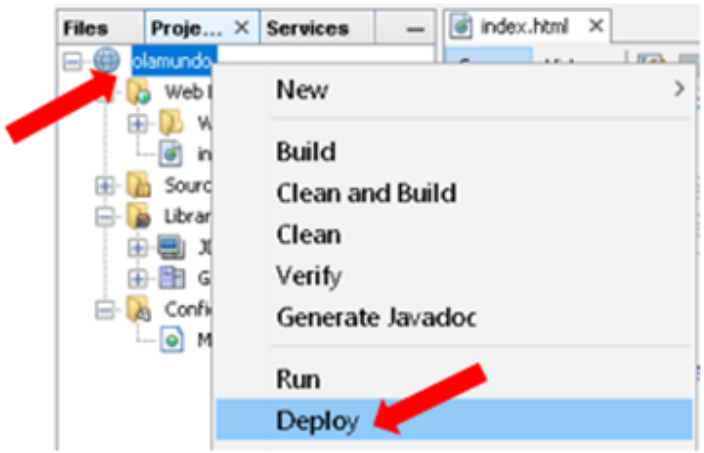
No arquivo **index.html**, podemos alterar o conteúdo e trocar a mensagem entre as tags , por exemplo:

```
<body>
    <div>Aplicação Web - Olá mundo !!</div>
</body>
```

Para testar a aplicação, clique com o botão do mouse no **projeto olamundo** e escolha a opção **Deploy** no menu pop-up.

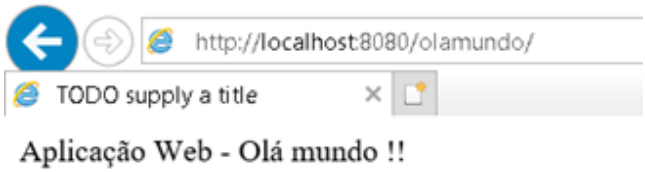


Em seguida, o **Servidor GlassFish** é carregado, conforme a janela de log abaixo:

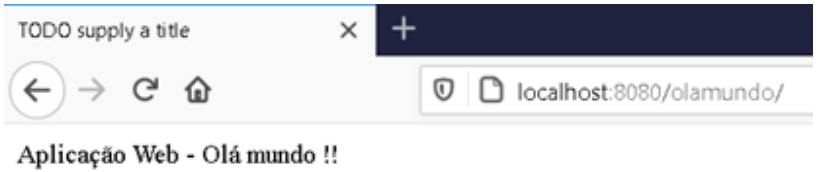


Fonte: Elaborada pelo autor.

Por fim, para verificar se deu tudo certo, abra um navegador para teste. Em meu caso, testei no **IExplorer** e **Firefox**, e obtive as janelas abaixo, respectivamente, e digite a seguinte URL: <http://localhost:8080/olamundo>.



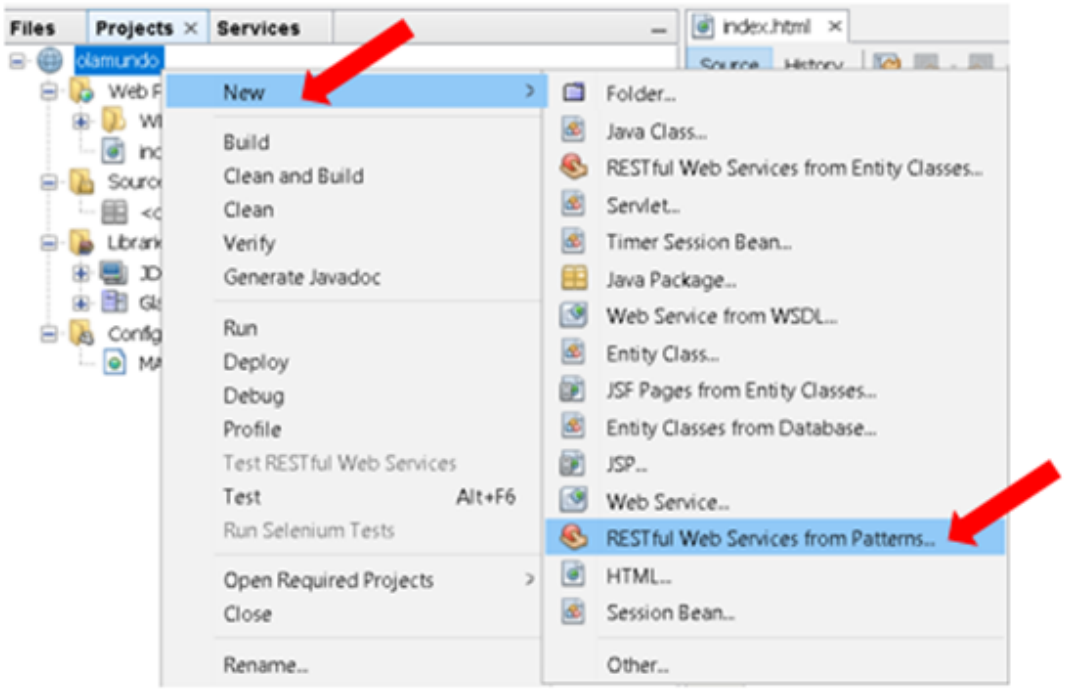
Fonte: Elaborada pelo autor.



Fonte: Elaborada pelo autor.

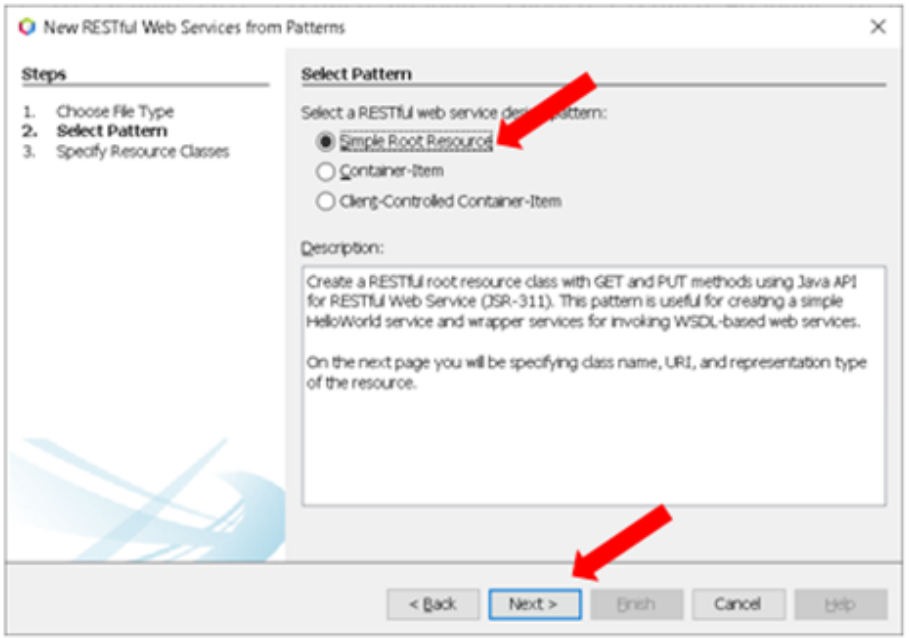
Esses testes são importantes para verificar se o **GlassFish** está instalado e configurado da forma correta, mas ainda não temos um Web Service RESTful.

Para criar o projeto **Web Service RESTful**, clique com o botão direito do mouse no projeto **Web Application**. Em nosso caso, o projeto tem o nome **olamundo**. Em seguida, aparecerá um menu pop-up, basta clicar em **New -> Web Services RESTful from Patterns...** e teremos a janela a seguir:



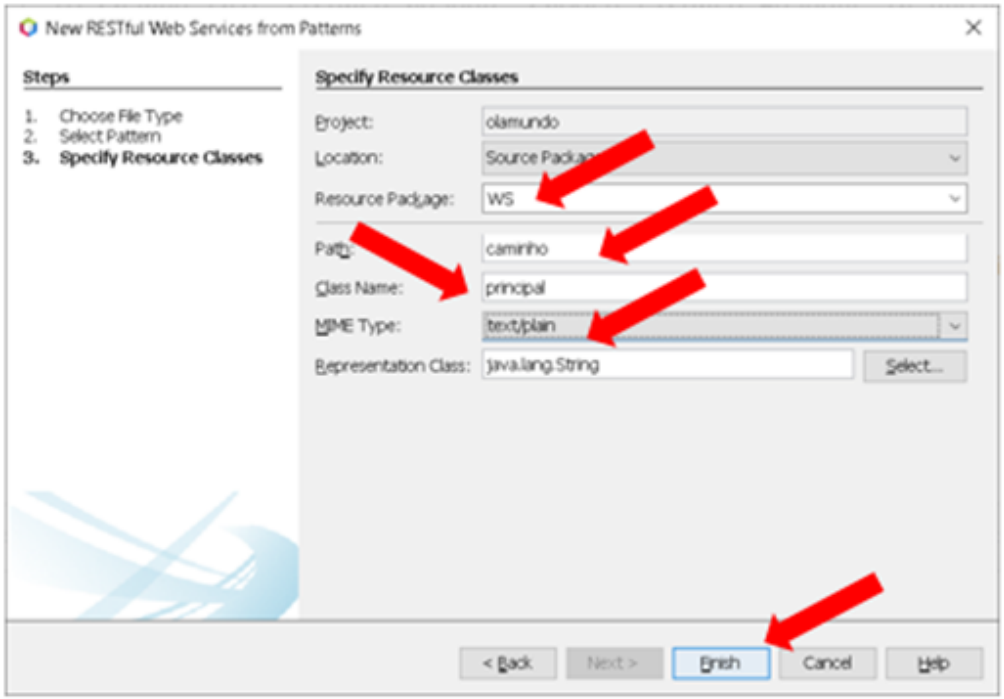
Fonte: Elaborada pelo autor.

Nessa janela, escolha a opção **Simple Root Resource** e clique em **Next**.



Fonte: Elaborada pelo autor.

Teremos, assim, a próxima janela abaixo. Nela, você deve preencher os seguintes campos: **Resource Package** (Pacote de Recurso), ou seja, como serão organizadas suas classes nas pastas do projeto; **Path** (Caminho), ou seja, o caminho para chegar a sua classe na URL; **Class Name** (Nome da Classe), ou seja, o nome da classe principal; e **MIME Type** (Tipo de Mídia de Internet), ou seja, o tipo de formato da comunicação. Nesse caso, escolha o formato texto **text/plain**, e depois clique em **Finish**. Abaixo, você pode ver os valores que preenchi nos campos. É importante que, para esse exemplo, você preencha os campos dessa janela com os mesmos valores.



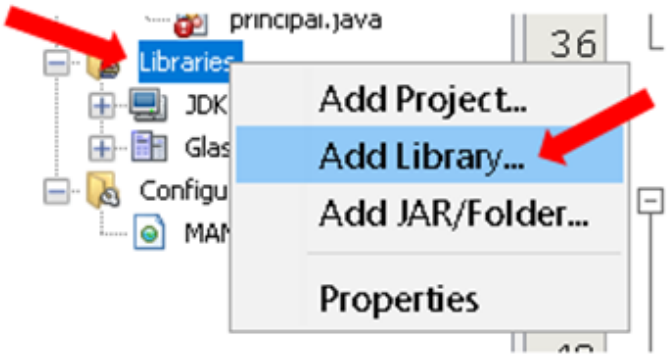
Fonte: Elaborada pelo autor.

No projeto, são criadas duas classes, **ApplicationConfig** e **principal**, dentro do **pacote WS**. A classe **ApplicationConfig** contém a configuração do Web Service e a classe **principal** tem a estrutura do WebService já com métodos para **GET** e **PUT**. Ficaria, então, algo mais ou menos assim:

```
33      /**
34       * Retrieves representation of an instance of WS.principal
35       * @return an instance of java.lang.String
36       */
37      @GET
        @Produces(javax.ws.rs.core.MediaType.TEXT_PLAIN)
39      public String getText() {
40          //TODO return proper representation object
41          throw new UnsupportedOperationException();
42      }
43      /**
44       * PUT method for updating or creating an instance of principal
45       * @param content representation for the resource
46       */
47      @PUT
        @Consumes(javax.ws.rs.core.MediaType.TEXT_PLAIN)
49      public void putText(String content) {
50      }
```

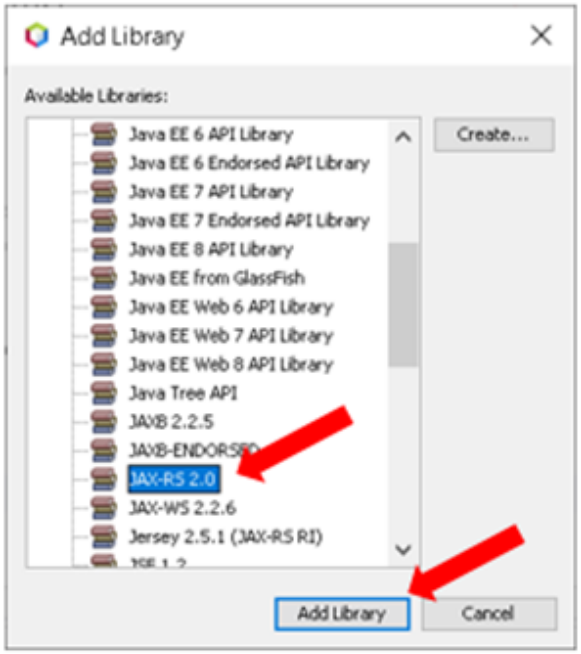
Fonte: Elaborada pelo autor.

Veja que o NetBeans acusa um erro. Ele relata que o pacote **javax.ws.rs.core** não foi encontrado no projeto. Esse pacote consta na implementação da especificação **JAX-RS 2.0**. Para solucionar esse problema, basta adicionar a biblioteca com a implementação. Para isso, clique com o botão direito do mouse em **Libraries,** no projeto, e, no menu pop-up, escolha a opção **Add Library...**. A tela a seguir apresenta o menu pop-up:



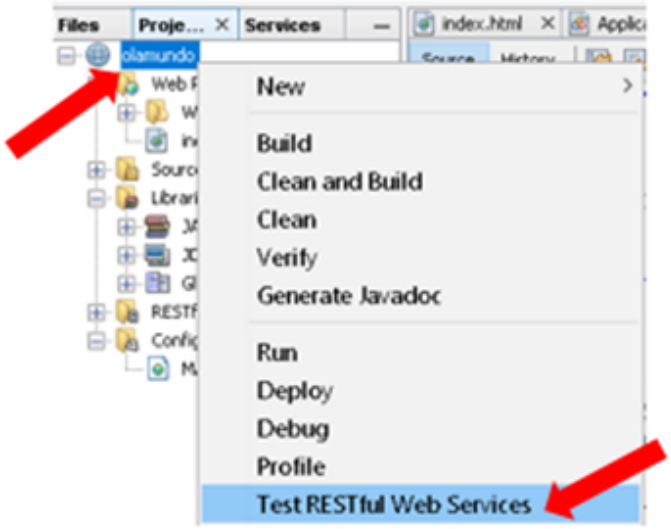
Fonte: Elaborada pelo autor.

A seguir, aparece a janela abaixo para adicionar a biblioteca **JAX-RS 2.0** e, por fim, clique no botão **Add Library**.



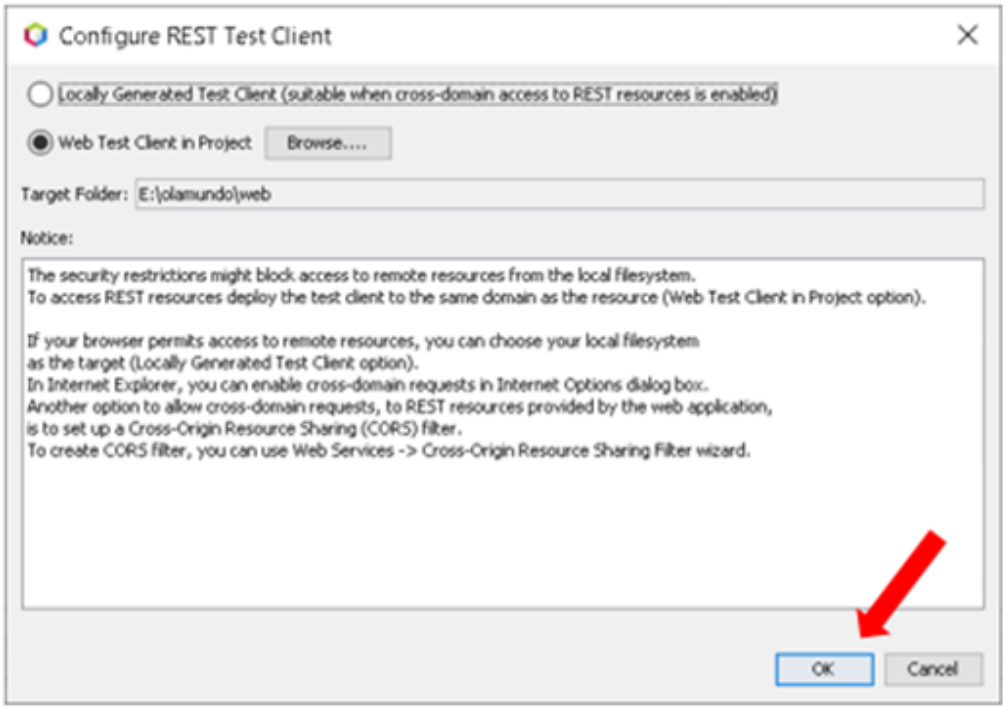
Fonte: Elaborada pelo autor.

Pronto. Finalmente, temos agora um Web Service RESTful sem erros de compilação. Agora, é só testar. Clique com o botão direito no **projeto olamundo** e escolha a opção **Test RESTful Web Services**.



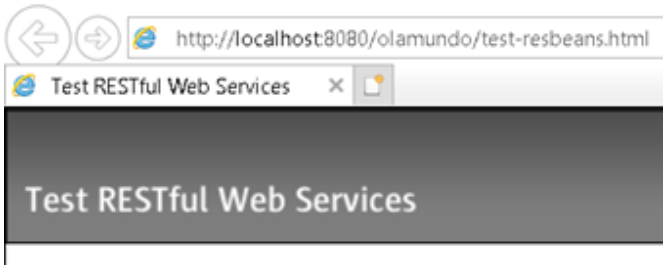
Fonte: Elaborada pelo autor.

Feito isso, a janela abaixo aparecerá. Basta clicar no botão OK para que um browser configurado como padrão seja carregado na página principal do Web Service, ou seja, o arquivo **test-resbeans.html**.



Fonte: Elaborada pelo autor.

Na barra de endereço do browser, temos a seguinte URL: <http://localhost:8080/olamundo/test-resbeans.html>.



Fonte: Elaborada pelo autor.

Para acessar o método GET de nosso Web Service, precisamos primeiro saber o caminho correto de nosso recurso. A biblioteca **JAX-RS 2.0** inclui algumas anotações importantes para a implementação do Web Service, como a anotação **@javax.ws.rs.ApplicationPath("webresources")** com a String **"webresources"** na classe **ApplicationConfig**. Se você quiser, pode modificar a String dentro da anotação, ou até mesmo deixá-la vazia, dessa forma: **@javax.ws.rs.ApplicationPath("")**.

```
@javax.ws.rs.ApplicationPath("webresources")
public class ApplicationConfig extends javax.ws.rs.core.Application {

    @Override
    public Set<Class<?>> getClasses() {
        Set<Class<?>> resources = new java.util.HashSet<>();
        addRestResourceClasses(resources);
        return resources;
    }
```

Fonte: Elaborada pelo autor.

Na **classe principal**, temos a anotação **@Path("caminho")** com a String **"caminho"**. Esse valor foi definido no momento em que criamos o projeto do Web Service no campo **Path**. Note que, na classe principal, também temos o método **getText()** que podemos reescrever para que ele retorne a nossa String com a mensagem de **"Web Service RESTful – Olá Mundo !!"**, como apresentado no trecho de código abaixo:

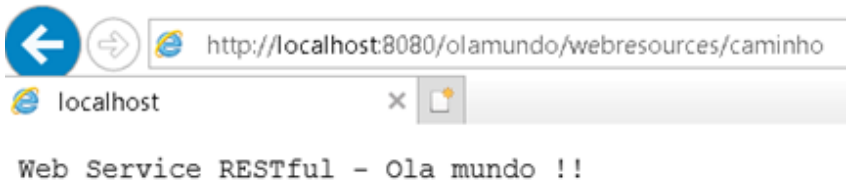
```java
@Path("caminho")
public class principal {
    @Context
    private UriInfo context;

    public principal() {
    }
    /**
     * Retrieves representation of an instance of WS.principal
     * @return an instance of java.lang.String
     */
    @GET
    @Produces(javax.ws.rs.core.MediaType.TEXT_PLAIN)
    public String getText() {
        return "Web Service RESTful - Ola Mundo !!";
    }
```

Fonte: Elaborada pelo autor.

A partir do nome do projeto (**olamundo**), a anotação da classe **ApplicationConfig** (**webresources**) e a anotação da classe principal (**caminho**), temos a URI completa para acessar nosso único recurso, o método **getText()**: <http://localhost:8080/olamundo/webresources/caminho>.

Como na classe principal só temos um método GET, ele é identificado automaticamente pela URI, e finalmente temos nosso tradicional **Olá Mundo** rodando com um Web Service RESTful. É lindo, né? Deu trabalho, mas conseguimos!

A imagem a seguir é um print da tela do browser **IExplorer**, acessando nosso Web Service com a URI sendo digitada diretamente na barra de endereço do browser.



Fonte: Elaborada pelo autor.

Imagine que, agora, implementaremos mais um recurso em nosso Web Service, ou seja, um outro método **GET** (**getPotencia()**) que calcula a potência entre dois números. No exemplo, o método receberá por parâmetro dois valores inteiros e calculará o primeiro parâmetro elevado ao segundo parâmetro. A implementação completa do novo método seria:

```
@GET
@Produces(javax.ws.rs.core.MediaType.TEXT_PLAIN)
@Path("potencia/{N1}/{N2}") // anotação pra representar o path do caminho e os parametros
public String getPontecia(@PathParam("N1") int N1, @PathParam("N2") int N2) {
    double resp= Math.pow(N1,N2);
    return "Potencia: "+N1+"^"+N2+" igual a "+ resp;
}
```

Fonte: Elaborada pelo autor.

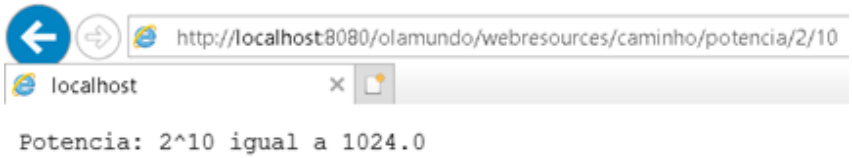
Na implementação do método **getPotencia()**, temos, logo no início, a anotação @Path("potencia/{N1}/{N2}") na qual estão descritos o **nome do recurso** e os dois **parâmetros** que são informados ao recurso na barra de endereço do browser. Um exemplo da URI completa seria: <http://localhost:8080/olamundo/webresources/caminho/potencia/2/10>.

Nesse caso, temos:

- **URL:** http://localhost:8080/olamundo/ :(URL + nome de nosso Web Service)
- **caminho:** /webresources/caminho/

- **nome do recurso e parâmetros:** potencia/2/10

Para testar o recurso, basta acessar o browser e inserir a URI completa do recurso. Veja abaixo o resultado da chamada do recurso:



Fonte: Elaborada pelo autor.

E, por fim, temos mais um método em nosso Web Service. Na próxima aula, veremos como integrar o acesso às tabelas do banco de dados do sistema acadêmico utilizando um Web Service RESTful.