Roteiro de Atividades – *Webservices*

1. Instalação das ferramentas

1. Instalar o Eclipse para desenvolvimento J2EE (disponível em https://www.eclipse.org/downloads/packages/release/2019-06/r/eclipse-ide-enterprise-java-developers)

2. Instalar o Apache Tomcat (versão recomendável: 8.5.x)
   1. Fazer o download do arquivo disponível em http://ftp.unicamp.br/pub/apache/tomcat/tomcat-8/v8.5.45/bin/apache-tomcat-8.5.45.zip
   2. Descompactar em alguma pasta
   3. Executar o arquivo *startup.sh* (Linux) ou *startup.bat* (Windows)
   4. Em um navegador, acessar http://localhost:8080

3. Instalar o cliente http POSTMAN.
   1. Em linux, no terminal, utilizar o comando `snap install postman`
   2. Em outros sistemas operacionais, acessar https://www.getpostman.com/downloads/

2. Atividades – Parte 1

1. No eclipse, criar um novo *Dynamic Web Project*

2. No projeto recém criado, verificar se o arquivo *web.xml* existe dentro da pasta *WebContent>WEB-INF.* Se não existir, clicar com o botão direito sobre o projeto e selecionar a opção *Java EE Tools > Generate Deployment Descriptor Stub.*

3. Ajustar o conteúdo do arquivo *web.xml* para que o conteúdo fique conforme o seguinte:

```
<?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://oracle.com/webfolder/technetwork/jsc/xml/ns/javaee/web-
app_3_0.xsd"
        version="3.0">
        <display-name>tutorial_jaxrs</display-name>
        <servlet>
                <servlet-name>blu3024</servlet-name>
                        <servlet-class>com.sun.jersey.spi.container.servlet.ServletContainer</servlet-
class>
                        <load-on-startup>1</load-on-startup>
        </servlet>
        <servlet-mapping>
                        <servlet-name>blu3024</servlet-name>
                        <url-pattern>/*</url-pattern>
        </servlet-mapping>

</web-app>
```

4. Transformar o projeto em um projeto *Maven*: clicar sobre o projeto com o botão direito do *mouse* e selecionar a opção *Configure > Convert to Maven Project*.

5. Adicionar as dependências necessárias ao arquivo *pom.xml*, que deve ter o seguinte conteúdo:

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
	xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
	xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
http://maven.apache.org/xsd/maven-4.0.0.xsd">
	<modelVersion>4.0.0</modelVersion>
	<groupId>Teste_REST_1</groupId>
	<artifactId>Teste_REST_1</artifactId>
	<version>0.0.1-SNAPSHOT</version>
	<packaging>war</packaging>
	<build>
		<sourceDirectory>src</sourceDirectory>
		<plugins>
			<plugin>
				<artifactId>maven-compiler-plugin</artifactId>
				<version>3.7.0</version>
				<configuration>
					<source>1.8</source>
					<target>1.8</target>
				</configuration>
			</plugin>
			<plugin>
				<artifactId>maven-war-plugin</artifactId>
				<version>3.2.1</version>
				<configuration>
					<warSourceDirectory>WebContent</warSourceDirectory>
				</configuration>
			</plugin>
		</plugins>
	</build>

	<dependencies>
		<dependency>
			<groupId>com.sun.jersey</groupId>
			<artifactId>jersey-bundle</artifactId>
			<version>1.19.4</version>
		</dependency>
	</dependencies>
</project>
```

6. Atualizar as dependências: clicar com o botão direito do *mouse* sobre o projeto e selecionar a opção *Maven>Update Project.*

7. Criar uma classe Java para implementar o webservice: clicar sobre o projeto com o botão direito do *mouse* e selecionar a opção *New>Class.*

8. Implementar a classe conforme a seguir:

```
@Path("/teste")
public class <nome da classe> {

	@GET
	public String sayHello(){
		return "Hello world";
	}

}
```

9. Exportar o projeto para um arquivo *.war*: clicar sobre o projeto com o botão direito do *mouse* e selecionar a opção *Export>WAR File*

10. Copiar o arquivo *.war* produzido para a pasta TOMCAT_HOME/webapps (onde TOMCAT_HOME é a pasta de instalação do Apache Tomcat).

11. Iniciar (ou reiniciar, se for o caso) o Tomcat

12. Em um navegador *web*, acessar o endereço *http://localhost:8080/<nome do projeto>/teste* e analisar o resultado.

3. Atividades – Parte 2

1. Adicionar o seguinte método à classe criada anteriormente :

```
@GET
@Path("/{name}")
public String sayHelloWithParameter(@PathParam("name") String msg) {
	return "Hello " + msg;
}
```

2. Acessar o endereço http://localhost:8080/<nome do projeto>/teste/nome e analisar o resultado.

4. Atividades – Parte 3

1. Implementar a classe *ValorClass* conforme abaixo:

```
public class ValorClass {

	private int valor = 0;
	private static final ValorClass instancia = new ValorClass();

	private ValorClass() {

	}

	public ValorClass getInstance() {
		return instancia;
	}


	public int getValor() {
		return valor;
	}

	public void setValor(int valor) {
		this.valor = valor;
	}

	public static ValorClass getInstancia() {
		return instancia;
	}


}
```

2. Na primeira classe criada, implementar os seguintes métodos:

```
 @PUT
@Path("/testeput")
public void doPost(@FormParam("value") int value) {
	ValorClass r = ValorClass.getInstancia();
	r.setValor(value);
}

@GET
@Path("/getvalor")
public String getValor() {
	return Integer.toString(ValorClass.getInstancia().getValor());
}
```

3. Enviar um *PUT* para o servidor, passando um número inteiro como parâmetro

4. Acessar http://localhost:8080/<nome do projeto>/teste/getvalor e observar o resultado

5. Atividades – Parte 5

Implementar um programa cliente que envie requisições GET e PUT para o recurso implementado nos passos anteriores.