

**Colégio Técnico de Campinas
Profª. : Simone Pierini Facini Rocha
Departamento de Processamento de Dados**

Apostila da Disciplina Aplicações Distribuídas e Orientada a Serviços

Campinas

2018

JSON (JavaScript Object Notation) é um formato leve para **armazenamento e transmissão** de informações no formato texto. É fácil para os humanos ler e escrever e para as máquinas analisar e gerar. O JSON é um formato de texto que é completamente independente de linguagem, mas usa convenções que são familiares aos programadores da família C, incluindo C, C ++, C #, Java, JavaScript, Perl, Python e muitos outros. Essas propriedades fazem da JSON uma linguagem de intercâmbio de dados ideal.

Isto explica o fato de o JSON ter sido adotado por empresas como Google e Yahoo, cujas aplicações precisam transmitir grandes volumes de dados.

JSON é construído em duas estruturas:

- Uma coleção de pares nome / valor.
- Uma lista ordenada de valores.

Sintaxe

Para cada valor representado, atribui-se um nome (ou rótulo) que descreve o seu significado. Esta sintaxe é derivada da forma utilizada pelo JavaScript para representar informações. Por exemplo, para representar o ano de 2017, utiliza-se a seguinte sintaxe:

Representação o ano de 2017

```
"ano": 2017
```

Um par nome/valor deve ser representado pelo nome entre aspas, seguido de dois pontos, seguido do valor.

Os valores podem possuir 3 **tipos básicos**: numérico (inteiro ou real), booleano e string.

Os valores do tipo string devem ser representados entre aspas.

- Número real
"altura": 1.60
- String

"site": "www.google.com"

- Número negativo

"temperatura": -2

- Booleano

"conectado": true

A partir dos tipos básicos, é possível construir **tipos complexos**: array e objeto.

Os arrays são delimitados por colchetes, com seus elementos separados entre vírgulas.

- Array de Strings

["RJ", "SP", "MG", "ES"]

- Matriz de Inteiros

```
[  
  [1,5],  
  [-1,9],  
  [1000,0]  
]
```

Os objetos são especificados entre chaves e podem ser compostos por múltiplos pares nome/valor, por arrays e também por outros objetos. Desta forma, um objeto JSON pode representar, virtualmente, qualquer tipo de informação.

- Objeto

```
{  
  "titulo": "Toy Story",  
  "resumo": "Woody, um cowboy de pano é o brinquedo favorito de Andy",  
  "ano": 1995,  
  "genero": ["aventura", "animação", "comédia", "fantasia"]  
}
```

É possível representar mais de um objeto ou registro de uma só vez. Por exemplo, dois filmes são representados em um array.

- Array de objetos

```
[
  {
    "titulo": "Toy Story",
    "resumo": "Woody, um cowboy de pano é o brinquedo favorito de Andy",
    "ano": 1995,
    "genero": ["aventura", "animação", "comédia", "fantasia"]
  },
  {
    "titulo": "Carros 3",
    "resumo": "Veterano das pistas, Relâmpago McQueen se vê em apuros após o surgimento de um novato bastante veloz",
    "ano": 2017,
    "genero": ["animação", "aventura"]
  }
]
```

- Palavra-chave “null” deve ser utilizada para a representação de valores nulos

```
"idade":null
```

Exemplos de JSON

Exemplo 1

```
[
  {"cidade": "Campinas", "estado": "São Paulo"},
  {"cidade": "Colatina", "estado": "Espírito Santo"},
  {"cidade": "Belo Horizonte", "estado": "Minas Gerais"},
  {"cidade": "Volta Redonda", "estado": "Rio de Janeiro"}
]
```

Exemplo 2

```
[
  {"montadora": "Fiat", "codigo": "1", "modelo": "Uno"},
  {"montadora": "Fiat", "codigo": "2", "modelo": "Punto"},
  {"montadora": "Fiat", "codigo": "3", "modelo": "Siena"},
  {"montadora": "Ford", "codigo": "4", "modelo": "Ecosport"},
  {"montadora": "Ford", "codigo": "5", "modelo": "Fiesta"},
  {"montadora": "Ford", "codigo": "6", "modelo": "Prisma"},
  {"montadora": "Volkswagen", "codigo": "7", "modelo": "Gol"},
  {"montadora": "Volkswagen", "codigo": "8", "modelo": "Fox"},
  {"montadora": "Volkswagen", "codigo": "9", "modelo": "Golf"}
]
```

Exemplo 3

```
[
  {"RA": "16101", "nome": "Maria", "curso": "PD"},
  {"RA": "16102", "nome": "José", "curso": "Mec"},
  {"RA": "16103", "nome": "João", "curso": "Eletr"},
  {"RA": "16104", "nome": "Teresa", "curso": "Enf"},
  {"RA": "16105", "nome": "Isabela", "curso": "Plast"},
  {"RA": "16106", "nome": "Ricardo", "curso": "Alim"}
]
```

```
{ "RA": "16107", "nome": "Eduardo", "curso": "EMH"},
{ "RA": "16108", "nome": "Carlos", "curso": "ST"},
{ "RA": "16109", "nome": "Marina", "curso": "MA" }
]
```

Exemplo 4

```
[
{ "nome": "Joao da Silva", "email": "joao@joao.com.br", "usuario": "joao", "senha": "joao"},
{ "nome": "Carlos", "email": "carlos@carlos.com.br", "usuario": "carlos", "senha": "carlos" }
]
```

Exemplo 5

```
{ "Alunos": [
  { "RA": "17101", "nome": "José", "notas": [9,7,8] },
  { "RA": "17102", "nome": "João", "notas": [10,8,9] },
  { "RA": "17103", "nome": "Maria", "notas": [8,7,10] }
]
}
```

Exemplo 6

```
{ "aluno": [
  { "nome": "João", "provas": [ { "nota": 8 }, { "nota": 6 }, { "nota": 10 }, { "nota": 2 } ] },
  { "nome": "Maria", "provas": [ { "nota": 3 }, { "nota": 5 }, { "nota": 8 }, { "nota": 1 } ] },
  { "nome": "Pedro", "provas": [ { "nota": 7 }, { "nota": 6 }, { "nota": 6 }, { "nota": 8 } ] },
]
}
```

Exercícios

1. Coloque as informações no format JSON

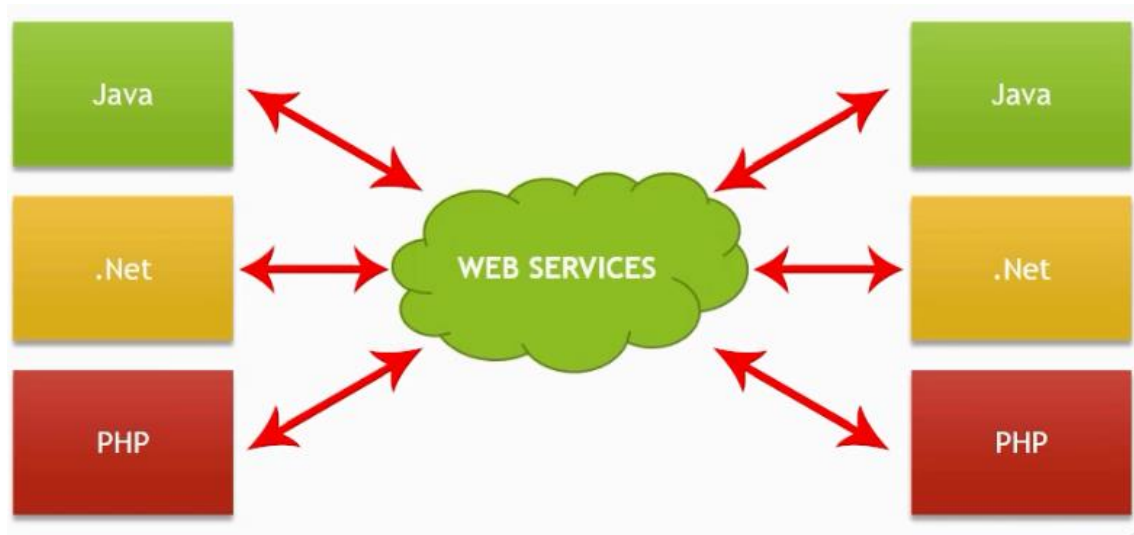
a) Endereço

Rua Armando Sales de Oliveira, cidade Campinas, estado SP

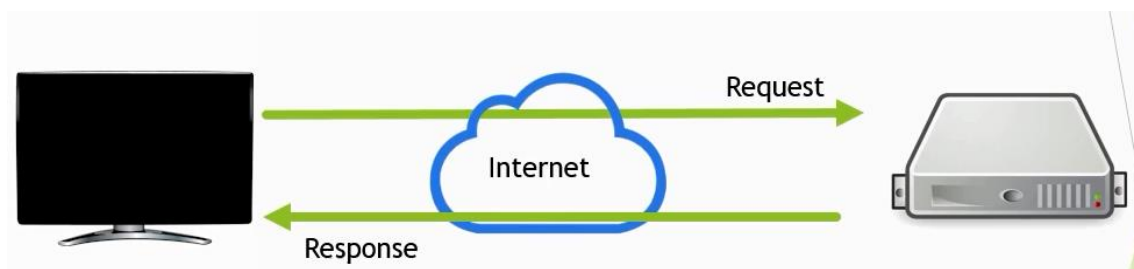
b) Empregados: Jason Jones, 38, Ada Pascalina, 35 e Delphino da Silva,

Web Services

O Web Service atua como uma camada de interface que intermedia todas comunicações entre diferentes linguagens de programação e sistemas operacionais



O protocolo HTTP, através da Internet, permite a comunicação entre os serviços do servidor e o cliente, que pode ser um browser, celular, ou televisão.



A comunicação se dá em uma requisição (Request) , que é enviada até o servidor, através da internet, baseada no protocolo HTTP e, é retornada até o cliente através de uma resposta (Response).

REST

REST – Representational State Transfer. Na tradução para o português, Tranferência de Estado Representacional.

É um formato de Web Service. Faz com que os recursos Web estejam disponíveis através de URL, links únicos que dão acesso aos recursos.

Todas comunicações do REST são baseadas nos verbos HTTP.

GET

Serviço de busca de informações. Utilizado para consulta.

POST

Serviço de criação de dados. Utilizado para inserir.

PUT

Serviço de atualização de informação..

DELETE

Serviço de deleção.

O REST é baseado em URI(Identificador Uniforme de Serviços) , que são usados para expor o que os serviços autodescritivos faz, dando significado a URL.

Exemplo:

<http://localhost:8080/ServidorRestFUL/webresources/postAluno>

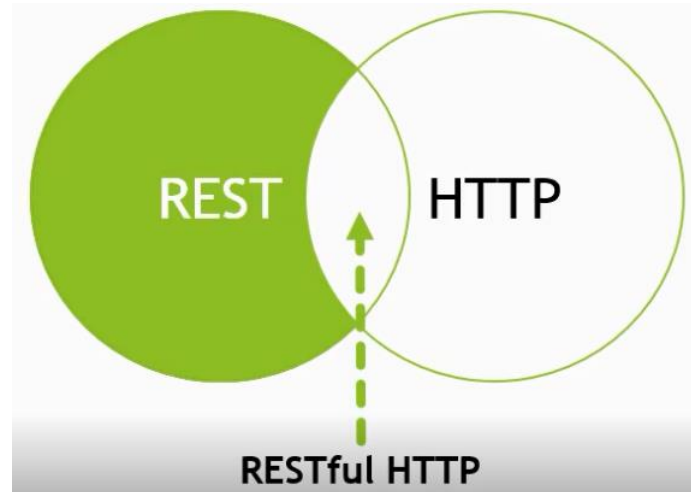
Nesse exemplo, o recurso postAluno é buscado através da URL.

Cada comunicação com o servidor é independente. Tudo o que é necessário ser enviado (Request) e recebido (Response) ao servidor deve ser feito de uma só vez. É uma comunicação integral.

REST e RESTFUL

REST é o conceito, o novo paradigma de encapsular o protocolo Http para desenvolver serviços.

O RESTFUL é uma representação do conceito REST, que aplica todas as exigências e regras do protocolo Http.

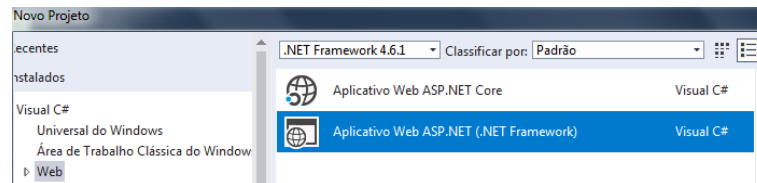


O **paradigma REST** realizar a integração entre as aplicações, através de dados nos formatos XML, JSON, HTML, PDF (binários), imagens e textos comuns.

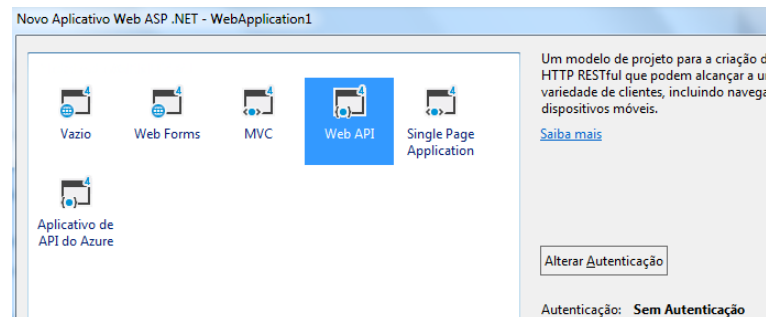
Webservice RestFul em C#

Servidor

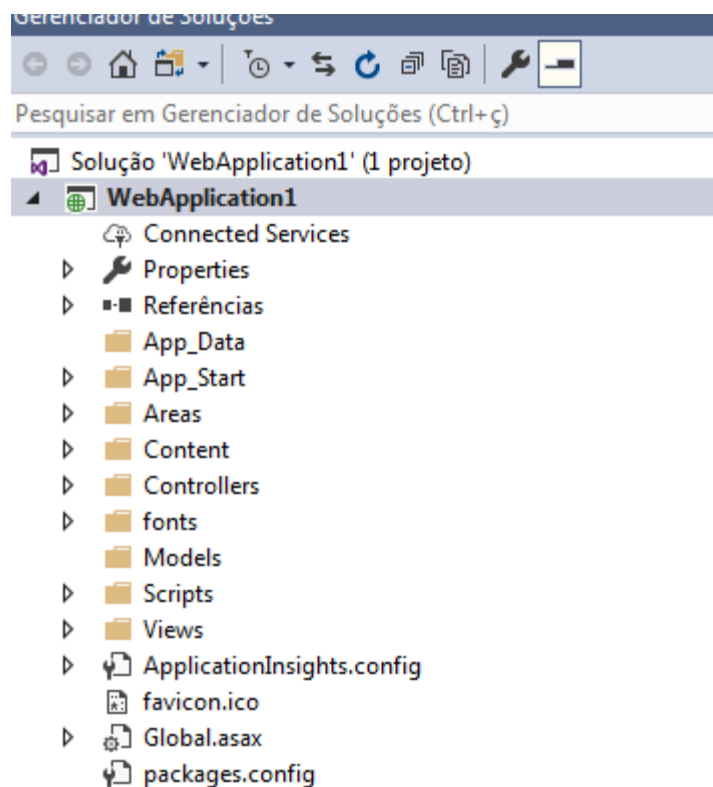
- 1- No framework do Visual Studio, criar um novo projeto Web
- 2- Selecionar a opção Aplicativo Web ASP.NET (.NET framework) Visual C#



- 3- Selecionar a opção Web API, Sem Autenticação.



- 4- Será aberto um projeto MVC



5- Na pasta Models, criar a classe Aluno.cs

```
using System;

using System.Collections.Generic;

using System.Linq;

using System.Web;

namespace Cotuca.WebApiAluno.Models

{

    public class Aluno

    {

        public int Id { get; set; }

        public string Nome { get; set; }

        public string Curso { get; set; }

    }

}
```

6- Na pasta Models, criar a interface IAlunoRepositorio.cs

```
using System;

using System.Collections.Generic;

using System.Linq;

using System.Text;

using System.Threading.Tasks;

namespace Cotuca.WebApiAluno.Models

{

    public interface IAlunoRepositorio

    {

        //Definição dos métodos a serem implementados

        IEnumerable<Aluno> GetAll();

        IEnumerable<Aluno> GetPorCurso(string curso);

        Aluno GetPorId(int id);

        Aluno Add(Aluno item);

        void Remove(int id);

    }

}
```

```

        void Update(Aluno item);
    }
}

```

7- Na pasta Models, criar a classe AlunoRepositorio.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Web;
using Cotuca.WebApiAluno.Models;
using System.Windows.Forms;
using System.Web.Http;

namespace Cotuca.WebApiAluno.Models{
    public class AlunoRepositorio : IAlunoRepositorio
    {
        private List<Aluno> alunos = new List<Aluno>();
        private int _nextId = 1;

        //Construtor
        //Adiciona itens à lista
        public AlunoRepositorio(){
            Add(new Aluno { Nome = "João Paulo", Curso = "Mecatrônica" });
            Add(new Aluno { Nome = "Caio", Curso = "Informática" });
            Add(new Aluno { Nome = "Maria Clara", Curso = "Eletro" });
            Add(new Aluno { Nome = "Eduardo", Curso = "Enfermagem" });
        }

        //Retorna tudo da lista de alunos
        public IEnumerable<Aluno> GetAll() {
            return alunos;
        }
    }
}

```

```

//Adiciona item à lista
public Aluno Add(Aluno item) {
    if (item == null) {
        throw new ArgumentNullException("item");
    }

    item.Id = _nextId++;
    alunos.Add(item);
    return item;
}

//Seleciona aluno por curso
public IEnumerable<Aluno> GetPorCurso(string curso) {
    return alunos;
}

//Seleciona aluno por id
public Aluno GetPorId(int id) {
    return alunos.Find(a => a.Id == id);
}

public void Update(Aluno item) {
    if(item == null) {
        throw new ArgumentNullException("item");
    }

    alunos.Where(a => a.Id == item.Id)
        .Select(a =>
            {
                a.Id = item.Id;
                a.Nome = item.Nome;
                a.Curso = item.Curso;
                return a;
            }).ToList();
}

//Remove um item da lista pelo id
public void Remove(int id) {

```

```

        alunos.RemoveAll(a => a.Id == id);
    }
}
}

```

8- Na pasta Controllers, criar a classe AlunosController.cs

```

using System;
using System.Collections.Generic;
using System.Linq;
using System.Net;
using System.Net.Http;
using System.Web.Http;
using Cotuca.WebApiAluno.Models;
using System.Windows.Forms;

/* Descritivo:
 *
 * Usa o repositório definido pela classe AlunoRepositorio para implementar a
 * Web API 2 expondo os seguintes serviços:
 *
 * GetAllAlunos - retorna todos os alunos;
 * GetAlunoPorId - retorna aluno por id;
 * GetAlunoPorCurso - retorna alunos por curso;
 * PostAluno - inclui um novo aluno;
 * PutAluno - altera um aluno;
 * DeleteAluno - exclui um aluno;
 *
 * Observe que os nomes dos métodos do controlador AlunosController
 * iniciam com o nome dos verbos HTTP (GET, POST, PUT e DELETE)
 * Essa metodologia ajuda o Visual Studio a mapear a requisição do
 * cliente para os métodos do controller (convenção).
 */

```

```

namespace Cotuca.WebApiAluno.Controllers
{
    [RoutePrefix("api/alunos")]
    public class AlunosController : ApiController
    {
        static readonly IAlunoRepositorio repositorio = new AlunoRepositorio();

        /* Método: GetAllAlunos(): IEnumerable<Aluno>
        *
        * Consulta todos os alunos, chamando o método GetAll(): IEnumerable<Aluno>
        * da classe AlunoRepositorio.cs
        *
        */
        public IEnumerable<Aluno> GetAllAlunos()
        {
            return repositorio.GetAll();
        }

        /* Método: GetAlunoPorCurso(string curso): IEnumerable<Aluno>
        *
        * Consulta todos os alunos, chamando o método GetPorCurso(string curso): Aluno
        * da classe AlunoRepositorio.cs
        *
        */
        public IEnumerable<Aluno> GetAlunoPorCurso(string curso)
        {
            // MessageBox.Show("Controller");

            // return repositorio.GetAll().Where(
            // a => string.Equals(a.Curso, curso, StringComparison.OrdinalIgnoreCase));

            return repositorio.GetPorCurso(curso).Where(
                a => string.Equals(a.Curso, curso, StringComparison.OrdinalIgnoreCase));
        }
    }
}

```

```

/* Método: GetAlunoPorId(int id): Aluno
*
* Consulta todos os alunos, chamando o método GetPorId(int id): Aluno
* da classe AlunoRepositorio.cs
*
*/

public Aluno GetAlunoPorId(int id){

    //O Objeto item recebe o retorno do método Get do repositório
    Aluno item = repositorio.GetPorId(id);
    if (item == null)
    {
        throw new HttpResponseMessage(HttpStatusCode.NotFound);
    }

    //retorna no formato json
    return item;
}

//Inclui um novo aluno

//HttpResponseMessage: mensagem de resposta que será trafegada dentro do
protocolo HTTP

public HttpResponseMessage PostAluno(Aluno item)
{
    item = repositorio.Add(item);

    // O objeto Request está dentro da classe ApiController
    // Coloque o cursor sobre Request e digite F12

    //Uma das assinaturas do CreateResponse pede para passar o status HTTP do
response

    var response =
        Request.CreateResponse<Aluno>(HttpStatusCode.Created, item);

    //A URL pede o Id do aluno
    string uri = Uri.Link("DefaultApi", new { id = item.Id });
    response.Headers.Location = new Uri(uri);
    return response;
}

```

```
}
```

```
//Altera
```

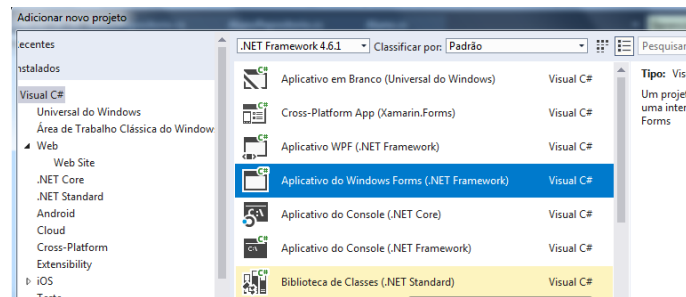
```
public HttpResponseMessage PutAluno(Aluno item) {  
    Aluno aluno = repositorio.GetPorId(item.Id);  
    if (aluno == null)  
        throw new HttpResponseMessage(HttpStatusCode.NotFound);  
  
    repositorio.Update(item);  
    var response = Request.CreateResponse<Aluno>(HttpStatusCode.Created, item);  
    return response;  
}
```

```
//Exclui
```

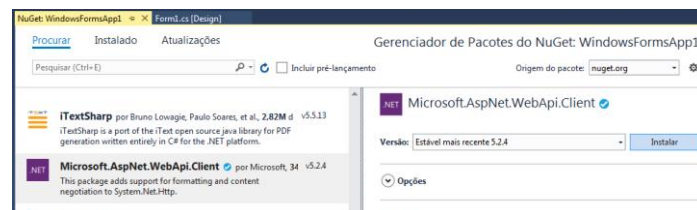
```
public void DeleteAluno(int id){  
    Aluno item = repositorio.GetPorId(id);  
    if (item == null)  
    {  
        throw new HttpResponseMessage(HttpStatusCode.NotFound);  
    }  
    repositorio.Remove(id);  
}  
}  
}
```


Cliente

- 1- No mesmo Projeto Solution, clicar com o botão do lado direito do mouse, Adicionar, Novo Projeto, Visual C#, Aplicativo do Windows Forms (.NET Framework)



- 2- Fazer referência ao pacote Microsoft.AspNet.WebApi.Client no projeto via Nuget. No menu Ferramentas, selecionar a opção Gerenciador de Pacotes do NuGet e, Gerenciar Pacotes do NuGet para a Solução...
- 3- Selecionar o pacote Microsoft.AspNet.WebApi.Client e botão Instalar



- 4- Criar um formulário com 5 Labels, 4 TextBoxs, 1 DataGridView e 6 botões, conforme o modelo abaixo:

- 5- Criar a classe Aluno.cs

```
using System;  
  
using System.Collections.Generic;
```

```

using System.Linq;
using System.Text;
using System.Threading.Tasks;

namespace Cotuca.AlunoCliente
{
    public class Aluno
    {
        public int Id { get; set; }
        public string Nome { get; set; }
        public string Curso { get; set; }
    }
}

```

6- Programar os métodos Click dos botões, conforme código abaixo:

```

using Newtonsoft.Json;
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Net.Http;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Cotuca.AlunoCliente
{
    public partial class Form1 : Form
    {
        private string URI = "";
    }
}

```

```
private int nextId = 1;

private string mensagemURI = "Preencha o campo porta da URI";
```

```
Aluno aluno = new Aluno();
```

```
HttpClient client = new HttpClient();

HttpResponseMessage response = new HttpResponseMessage();
```

```
public Form1()
```

```
{
    InitializeComponent();
}
```

```
private async void GetAll()
```

```
{
    URI = txtURI.Text;
    if (!preencherURI(URI))
        MessageBox.Show(mensagemURI);
```

```
else
```

```
{
```

```
    /*
```

```
    * Método HttpClient.GetAsync (Uri)
```

```
    *
```

```
    * Envia uma requisição GET para o URI especificado
```

```
    * com uma operação assíncrona.
```

```
    */
```

```
        //O objeto response(HttpResponseMessage) recebe a resposta do envio de
        requisição ao endereço URI
```

```
        response = await client.GetAsync(URI);
```

```
        //Se o envio de requisição for atendido
```

```
        if (response.IsSuccessStatusCode)
```

```
{
```

```

/*
 * Propriedade HttpResponseMessage.Content
 * Obtém ou define o conteúdo de uma mensagem de resposta HTTP.
 *
 * Para criar código assíncrono usamos as palavras chaves async
 * e await onde por padrão um método modificado por uma palavra-chave
 * async contém pelo menos uma expressão await.
 */

//A variável AlunoJsonString recebe o resultado da requisição
var AlunoJsonString = await response.Content.ReadAsStringAsync();

/*
 * Classe JsonConvert
 *
 * Fornece métodos para conversão entre tipos .NET e tipos JSON.
 *
 * DeserializeObject<T>(String)
 *
 */

gvAluno.DataSource =
JsonConvert.DeserializeObject<Aluno[]>(AlunoJsonString).ToList();
}
else
    MessageBox.Show("Não foi possível obter o produto : " +
response.StatusCode);
}
}

private async void GetAlunoByCurso(string nomeCurso)
{
    URI = txtURI.Text;
    if (!preencherURI(URI))

```

```

        MessageBox.Show(mensagemURI);
    else
    {
        //O BindSource é usado para vincular dados de um objeto a um um
        componente do windows form (GridView)

        BindingSource bsDados = new BindingSource();

        URI = txtURI.Text + "/?curso=" + nomeCurso.ToString();

        response = await client.GetAsync(URI);

        if (response.IsSuccessStatusCode)
        {
            var AlunoJsonString = await response.Content.ReadAsStringAsync();

            bsDados.DataSource = JsonConvert.DeserializeObject<Aluno[]>(AlunoJsonString).ToList();
            gvAluno.DataSource = bsDados;
        }
        else
        {
            MessageBox.Show("Falha ao obter o aluno pelo curso : " +
            response.StatusCode);
        }
    }
}

private async void GetAlunoById(int id)
{
    URI = txtURI.Text;

    if (!preencherURI(URI))

        MessageBox.Show(mensagemURI);
    else
    {
        BindingSource bsDados = new BindingSource();

        URI = txtURI.Text + "/" + id.ToString();

        response = await client.GetAsync(URI);
    }
}

```

```

        if (response.IsSuccessStatusCode)
        {
            var AlunoJsonString = await response.Content.ReadAsStringAsync();

            bsDados.DataSource =
            JsonConvert.DeserializeObject<Aluno>(AlunoJsonString);

            gvAluno.DataSource = bsDados;
        }
        else
        {
            MessageBox.Show("Falha ao obter o aluno : " + response.StatusCode);
        }
    }
}

```

```

private async void AddAluno()
{
    URI = txtURI.Text;
    aluno.Id = nextId++;
    aluno.Nome = txtNome.Text;
    alunoCurso = txtCurso.Text;

    if (!preencherURI(URI))
        MessageBox.Show(mensagemURI);
    else
    {
        var serializedAluno = JsonConvert.SerializeObject(aluno);

        //A classe StringContent adiciona o conteúdo json em um objeto HTTP
        var content = new StringContent(serializedAluno, Encoding.UTF8,
"application/json");

        var result = await client.PostAsync(URI, content);
    }

    GetAll();
}

```

```

private async void UpdateAluno(int id)
{
    URI = txtURI.Text;
    aluno.Id = id;
    aluno.Nome = txtNome.Text;
    alunoCurso = txtCurso.Text;

    if (!preencherURI(URI))
        MessageBox.Show(mensagemURI);
    else
    {
        response = await client.PutAsJsonAsync(URI, aluno);
        if (response.IsSuccessStatusCode)
            MessageBox.Show("Aluno atualizado");
        else
            MessageBox.Show("Falha ao atualizar o aluno : " + response.StatusCode);
    }
    GetAll();
}

```

```

private async void DeleteAluno(int id)
{
    URI = txtURI.Text;
    int alunoID = id;

    if (!preencherURI(URI))
        MessageBox.Show(mensagemURI);
    else
    {
        URI = txtURI.Text + "/" + alunoID;
        response = await client.DeleteAsync(URI);
        if (response.IsSuccessStatusCode)

```

```
        MessageBox.Show("Aluno excluído com sucesso");
    else
        MessageBox.Show("Falha ao excluir o aluno : " + response.StatusCode);
    }
    GetAll();
}
```

```
public bool preencherURI(string uri)
{
    if (uri == "http://localhost:porta/api/alunos")
        return false;
    else
        return true;
}
```

```
private void btnConsultar_Click(object sender, EventArgs e)
{
    GetAll();
}
```

```
private void btnConsultarCurso_Click(object sender, EventArgs e)
{
    aluno.Curso = txtCurso.Text;
    if (aluno.Curso != "")
        GetAlunoByCurso(aluno.Curso);
}
```

```
private void btnInserir_Click(object sender, EventArgs e)
{
    AddAluno();
}
```

```
private void btnAlterar_Click(object sender, EventArgs e)
{

```



```

        if (txtId.Text == "")
            MessageBox.Show("Preencha o campo Id");
        else
        {
            aluno.Id = Convert.ToInt32(txtId.Text);
            if (aluno.Id != -1)
                UpdateAluno(aluno.Id);
        }
    }

    private void btnExcluir_Click(object sender, EventArgs e)
    {
        if (txtId.Text == "")
            MessageBox.Show("Preencha o campo Id");
        else
        {
            aluno.Id = Convert.ToInt32(txtId.Text);
            if (aluno.Id != -1)
                DeleteAluno(aluno.Id);
        }
    }

    private void btnPorId_Click(object sender, EventArgs e)
    {
        if (txtId.Text == "")
            MessageBox.Show("Preencha o campo Id");
        else
        {
            aluno.Id = Convert.ToInt32(txtId.Text);
            GetAlunoById(aluno.Id);
        }
    }
}

```

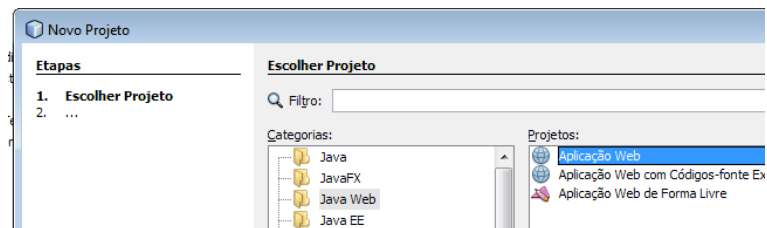
```
private void Form1_Load(object sender, EventArgs e)
{
    txtURI.Text = "http://localhost:porta/api/alunos";
}
}
}
```

Webservice RestFul em Java

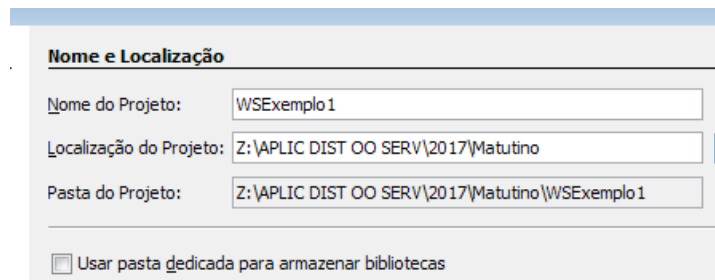
Exemplo 1

Etapas 1

Abrir o NetBeans e criar uma Aplicação Web:

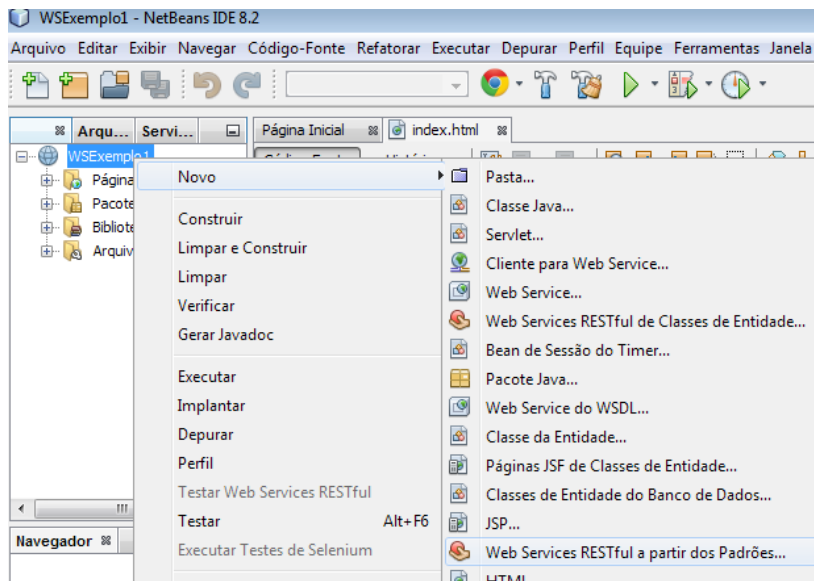


O nome do projeto será WSExemplo1



Usar o GlassFish como servidor.

Após a criação do projeto, clicar com o botão do lado direito do mouse sobre o projeto e selecionar a opção Web Service RESTful a partir dos Padrões.



Selecionar o Padrão Recurso Raiz Simples.

Em classes de recursos colocar as seguintes especificações:

Pacote: br.unicamp.cotuca

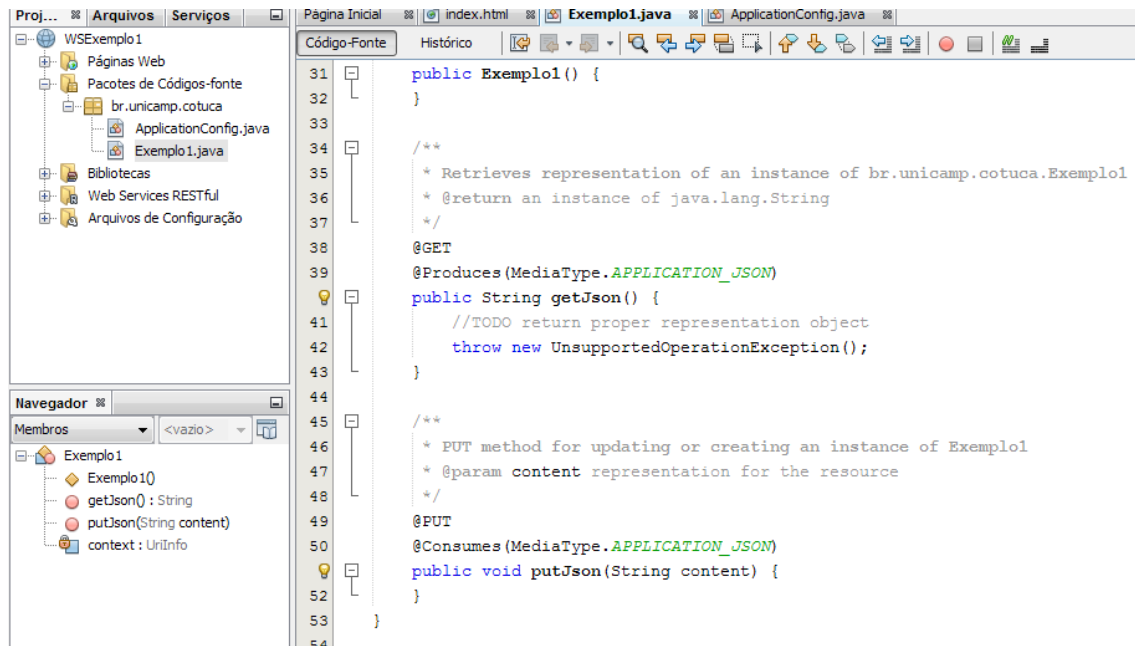
Nome da Classe: Exemplo1

Tipo MIME: application/JSON

Especificar Classes de Recurso

Projeto:	<input type="text" value="WSExemplo1"/>
Localização:	<input type="text" value="Pacotes de Códigos-fonte"/>
Pacote do Recurso\:	<input type="text" value="br.unicamp.cotuca"/>
Caminho\:	<input type="text" value="generic"/>
Nome da Classe:	<input type="text" value="Exemplo"/>
Tipo MIME:	<input type="text" value="application/json"/>
Classe de Representação:	<input type="text" value="java.lang.String"/> <input type="button" value="Selecionar..."/>

No projeto, serão criados os arquivos ApplicationConfig.java e a classe Exemplo1.java com as estruturas dos métodos GET e PUT do RESTFul



A estrutura @PUT é para receber informação e processar.

A estrutura @GET pode trabalhar enviando e recebendo dados

Nesse exemplo, será trabalhada a estrutura @GET.

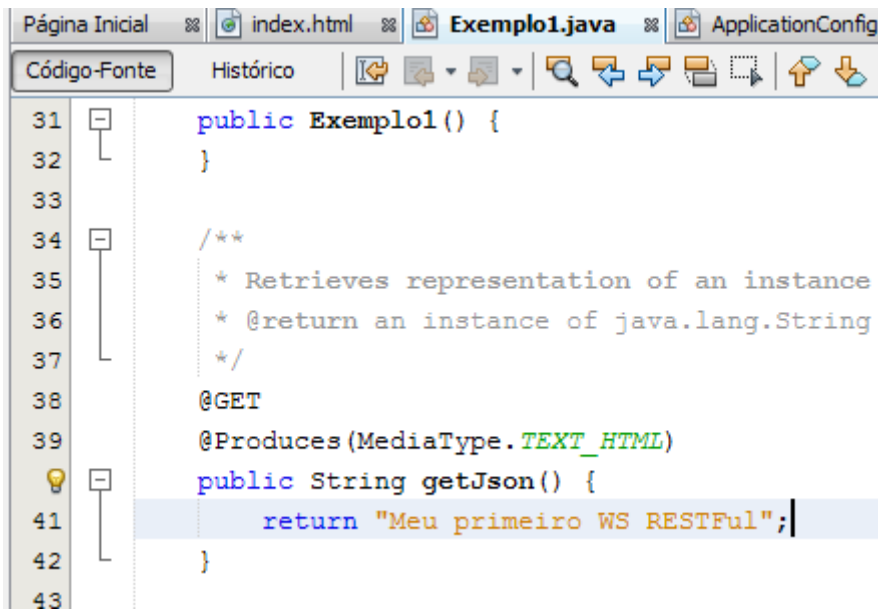
Em @Produces será produzida uma aplicação JSON. Também, poderia ser XML, ou texto.

Trocar o @Produces para (MediaType.TEXT_HTML).

Apagar o comentário e a linha throw...,

Acrescentar a linha:

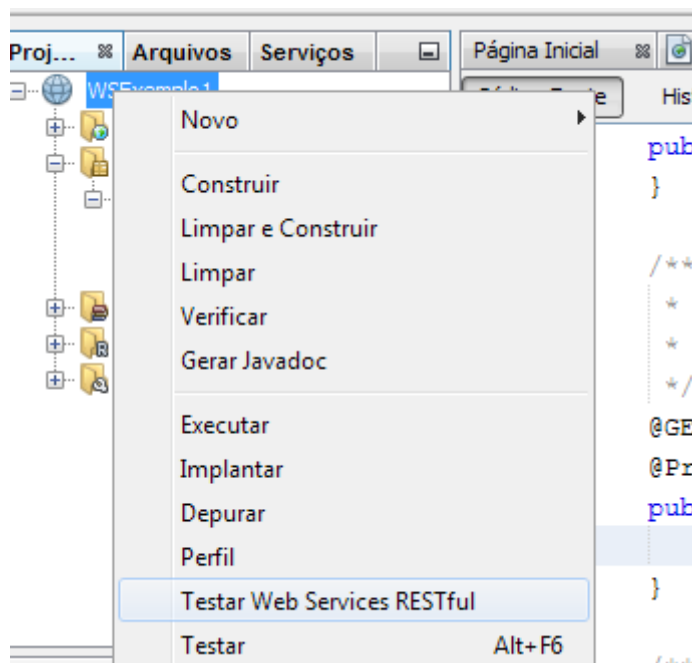
return "Meu primeiro WS RESTFul";



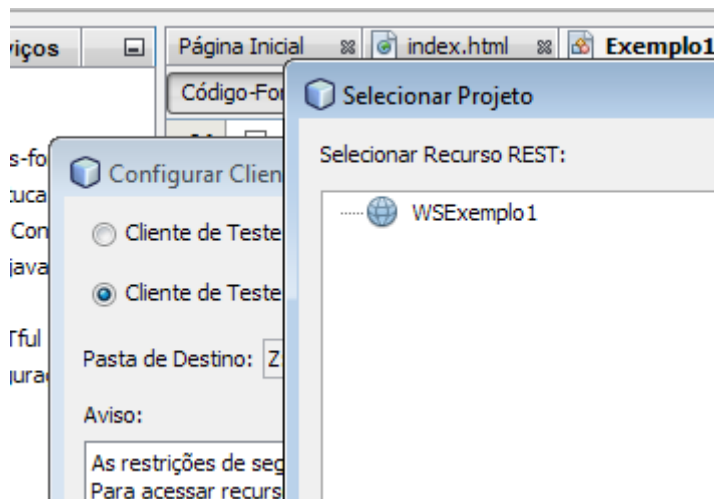
```
31 public Exemplo1() {
32 }
33
34 /**
35  * Retrieves representation of an instance
36  * @return an instance of java.lang.String
37  */
38 @GET
39 @Produces(MediaType.TEXT_HTML)
40 public String getJson() {
41     return "Meu primeiro WS RESTFul";
42 }
43
```

Para testar esse primeiro exemplo, será usada uma interface do próprio NetBeans, que usa o GlassFish como servidor.

Para isso, clicar com o botão do lado direito do mouse sobre o projeto e selecionar a opção Testar Web Service RESTFul.

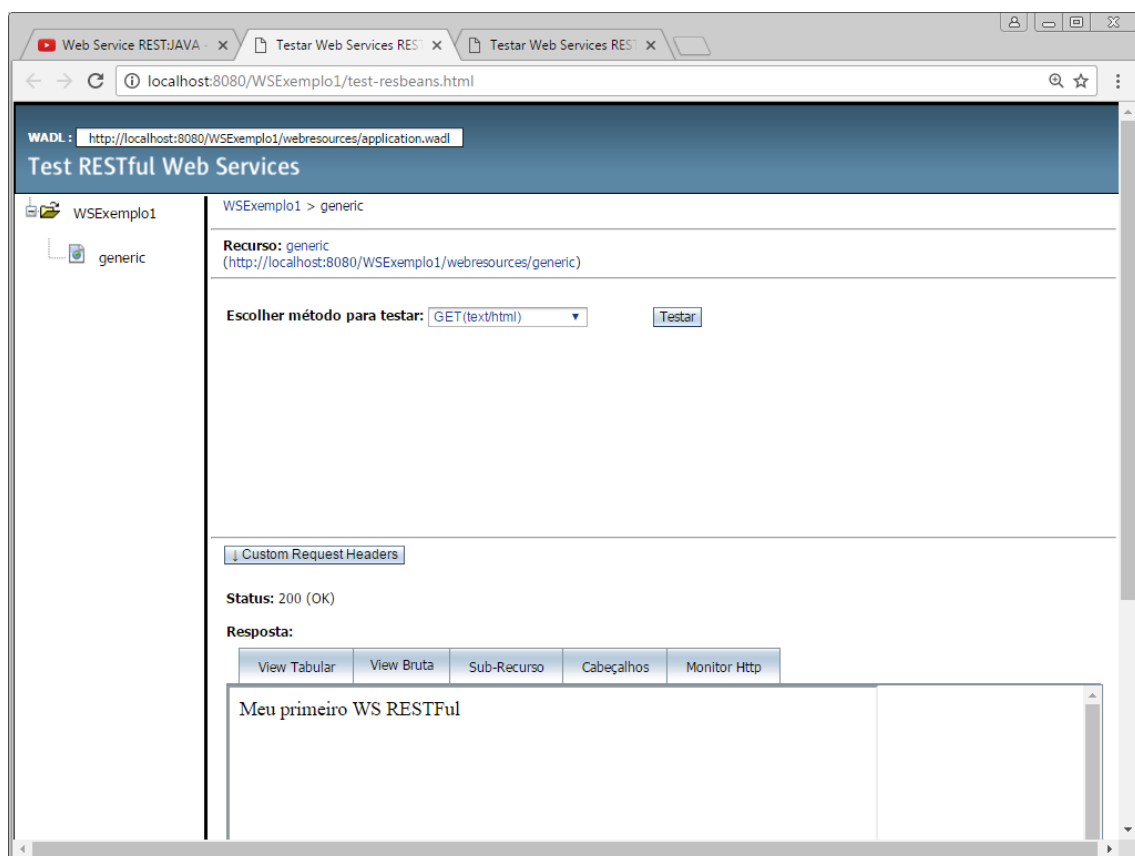


Escolher o projeto.



Selecionar o botão OK.

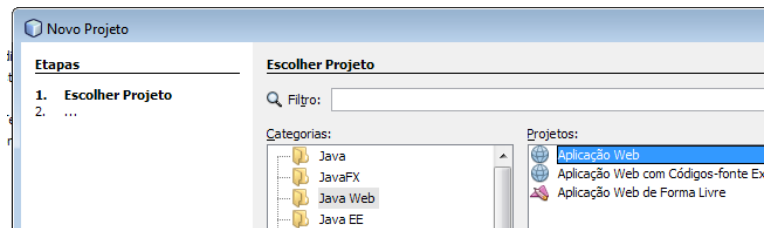
O Browser será aberto com a tela de Test RESTFul Web Services. O processo de abertura dessa tela pode ser demorado. Selecionar o botão Testar.



Em Resposta será mostrado o return do método @GET: “Meu primeiro WS RestFul”. Essa etapa significa que o Web Service RESTFul está funcionando corretamente.

Exemplo 2

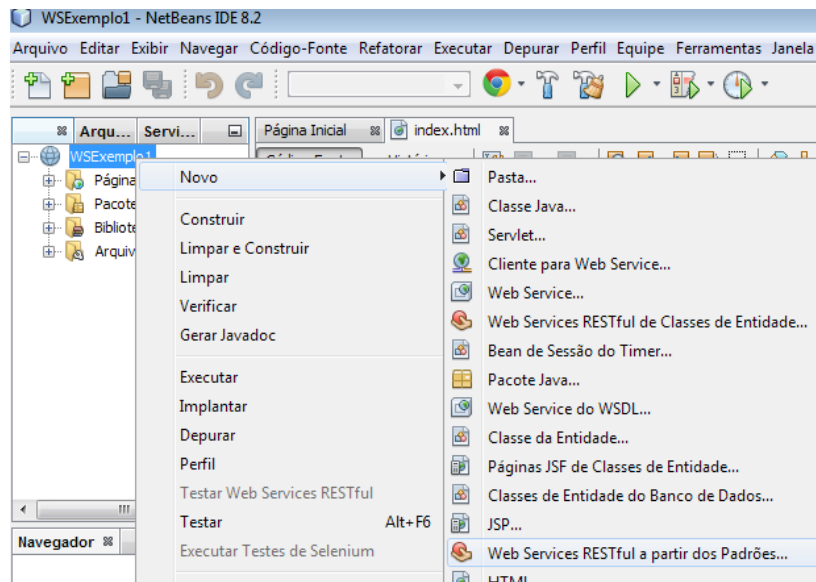
Abrir o NetBeans e criar uma Aplicação Web:



O nome do projeto será ServidorRestFul

Usar o GlassFish como servidor.

Após a criação do projeto, clicar com o botão do lado direito do mouse sobre o projeto e selecionar a opção Web Service RESTFul a partir dos Padrões.



Selecionar o Padrão Recurso Raiz Simples.

Em classes de recursos escrever as seguintes especificações:

Pacote: br.unicamp.cotuca

Nome da Classe: GenericResource

Tipo MIME: application/JSON

Criar a classe Aluno.java

```
package cotuca.unicamp.br;
```

```
/**
```

```
*
```

```
* @author simone
```

```
*/
```

```
public class Aluno {
```

```
    private int Id;
```

```
    private String Nome;
```

```
    private String Curso;
```

```
    public int getId() {
```

```
        return Id;
```

```
    }
```

```
    public void setId(int Id) {
```

```
        this.Id = Id;
```

```
    }
```

```

public String getNome() {

    return Nome;

}

public void setNome(String Nome) {

    this.Nome = Nome;

}

public String getCurso() {

    return Curso;

}

public void setCurso(String Curso) {

    this.Curso = Curso;

}

}

```

Na classe GenericResource criar os atributos contexto e listaAluno

```
@Path("generic")
```

```
public class GenericResource {
```

```
    @Context
```

```
    private UriInfo context;
```

```
    private static ArrayList<Aluno> listaAluno = new ArrayList<Aluno>();
```

criar o constructor da classe com os valores a serem inseridos no ArrayList listaAluno.

```
public GenericResource() {
```

```
    Aluno a1 = new Aluno();
```

```
a1.setId(1);

a1.setNome("Caio");

a1.setCurso("Informática");
```

```
Aluno a2 = new Aluno();

a2.setId(2);

a2.setNome("Maria");

a2.setCurso("Eletro");
```

```
Aluno a3 = new Aluno();

a3.setId(3);

a3.setNome("Eduardo");

a3.setCurso("Enfermagem");
```

```
listaAluno = new ArrayList<Aluno>();

listaAluno.add(a1);

listaAluno.add(a2);

listaAluno.add(a3);
```

```
}
```

Criar o método getAll() para listar todos os alunos da listaAluno.

```
@GET
```

```
@Path("/getAll")
```

```
@Produces(MediaType.APPLICATION_JSON)
```

```
public List<Aluno> getAll() {  
  
    //TODO return proper representation object  
  
    return listaAluno;  
  
}
```

Criar o método getId() para listar o aluno com o id passado como parâmetro.

@GET

@Path("/getId/{id}")

@Consumes(MediaType.APPLICATION_JSON)

@Produces(MediaType.APPLICATION_JSON)

```
public Aluno getId(@PathParam("Id") int id) {  
  
    List<Aluno> listaTudo = getAll();  
  
    Aluno result = null;  
  
    for (Aluno a : listaTudo) {  
  
        if (a.getId() == id) {  
  
            result = a;  
  
            break;  
  
        }  
  
    }  
  
    if (result == null) {  
  
        throw new WebApplicationException(404);  
  
    }  
  
    return result;  
  
}
```

Criar o método `getCurso()` para listar os alunos com o nome passado como parâmetro.

```
@GET

@Path("/getCurso/{nomeCurso}")

@Consumes(MediaType.APPLICATION_JSON)

@Produces(MediaType.APPLICATION_JSON)

public List<Aluno> getCurso(@PathParam("nomeCurso") String curso) {

    List<Aluno> listaTudo = getAll();

    List<Aluno> result = new ArrayList<Aluno>();

    for (Aluno a : listaTudo) {

        if ( (a.getCurso() != null) && ( a.getCurso().toLowerCase().contains(curso.toLowerCase()) )){

            result.add(a);

        }

    }

    return result;

}
```

Criar o método `postAluno()` para inserir itens na lista os alunos com o id passado como parâmetro.

```
@POST

@Path("/postAluno")

@Consumes(MediaType.APPLICATION_JSON)

@Produces(MediaType.APPLICATION_JSON)

public ArrayList<Aluno> postAluno(Aluno aluno) {

    if (aluno.getNome() == null || aluno.getNome().trim().equals("")) {

        throw new WebApplicationException(

            Response.status(Response.Status.BAD_REQUEST).entity(

                "O nome do aluno é obrigatório").build());

    }

}
```

```

    }

    else

    {

        listaAluno.add(aluno);

        return listaAluno;

    }

}

```

Criar o método putAluno() para alterar itens na lista os alunos com Aluno passado como parâmetro.

```

@PUT

@Path("/putAluno")

@Consumes(MediaType.APPLICATION_JSON)

@Produces(MediaType.APPLICATION_JSON)

public ArrayList<Aluno> putAluno(Aluno aluno) {

    if (aluno.getNome() == null ||

        aluno.getNome().trim().equals("") ||

        aluno.getCurso() == null ||

        aluno.getCurso().trim().equals("") ) {

        throw new WebApplicationException(

            Response.status(Response.Status.BAD_REQUEST).entity(

                "O nome do aluno e curso são obrigatórios").build());

    }

    else

    {

        List<Aluno> listaTudo = getAll();

        for (Aluno a : listaTudo) {

            if(a.getId()==aluno.getId()){

```

```

        a.setNome(aluno.getNome());

        a.setCurso(aluno.getCurso());

    }

}

}

return listaAluno;

}

```

Criar o método `excluirAluno()` para excluir itens na lista os alunos com `Aluno` passado como parâmetro.

```

@GET

@Path("/excluirAluno/{id}")

@Consumes(MediaType.APPLICATION_JSON)

public List<Aluno> excluirAluno(@PathParam("id") Integer id)

    throws Exception {

    Aluno result = null;

    List<Aluno> listaTudo = getAll();

    for (Aluno a : listaTudo) {

        if (a.getId() == id) {

            result = a;

            listaTudo.remove(result);

            break;

        }

    }

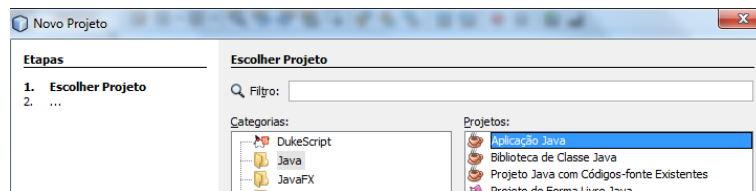
    return listaTudo;

}

```

Criando um cliente java para consumir o java RESTFUL

No NetBeans, criar um novo projeto aplicação Java e nomeá-lo `ClienteProjeto1`



Na classe ClienteProjeto1 codificar:

```
public class ClienteProjeto1 {

    public static void main(String[] args) throws MalformedURLException, ProtocolException, IOException {

        ClienteProjeto1 cp = new ClienteProjeto1();

        String urlGetAll = "http://localhost:8080/ServidorRestFUL/webresources/generic/getAll";

        String urlGetId = "http://localhost:8080/ServidorRestFUL/webresources/generic/getId/3";

        String urlGetCurso = "http://localhost:8080/ServidorRestFUL/webresources/generic/getCurso/eletro";

        String urlPostAluno = "http://localhost:8080/ServidorRestFUL/webresources/generic/postAluno";

        String urlPutAluno = "http://localhost:8080/ServidorRestFUL/webresources/generic/putAluno";

        //String getAll = cp.sendGetAll(urlGetAll);

        //String getId = cp.sendGetId(urlGetId);

        //String getCurso = cp.sendGetCurso(urlGetCurso);

        //String postAluno = cp.sendPostAluno(urlPostAluno);

        String putAluno = cp.sendPutAluno(urlPutAluno);

        //Mostra no formato json

        //System.out.println(getAll);

        //System.out.println(getId);

        // System.out.println(getCurso);

        //System.out.println(postAluno);

        System.out.println(putAluno);    }
```

```

private String sendGetAll(String url) throws MalformedURLException, ProtocolException, IOException
{
    URL objURL = new URL(url);

    HttpURLConnection con = (HttpURLConnection) objURL.openConnection();

    con.setRequestMethod("GET");

    int responseCode = con.getResponseCode();

    System.out.println("\nEnviando requisição 'GET' para URL: " + url);

    System.out.println("Response Code: " + responseCode);

    BufferedReader br =

        new BufferedReader(new InputStreamReader(con.getInputStream()));

    String inputLine;

    StringBuffer response = new StringBuffer();

    while ((inputLine = br.readLine()) != null){

        response.append(inputLine);

    }

    br.close();

    con.disconnect();

    return response.toString();

}

private String sendGetId(String url) throws MalformedURLException, ProtocolException, IOException
{
    URL objURL = new URL(url);

    HttpURLConnection con = (HttpURLConnection) objURL.openConnection();

    con.setRequestMethod("GET");

    int responseCode = con.getResponseCode();

    System.out.println("\nEnviando requisição 'GET' para URL: " + url);

    System.out.println("Response Code: " + responseCode);

    BufferedReader br =

        new BufferedReader(new InputStreamReader(con.getInputStream()));

    String inputLine;

```

```

StringBuffer response = new StringBuffer();

while ((inputLine = br.readLine()) != null){

    response.append(inputLine);

}

br.close();

con.disconnect();

return response.toString();

}

private String sendGetCurso(String url) throws MalformedURLException, ProtocolException, IOException
{

    URL objURL = new URL(url);

    HttpURLConnection con = (HttpURLConnection) objURL.openConnection();

    con.setRequestMethod("GET");

    int responseCode = con.getResponseCode();

    System.out.println("\nEnviando requisição 'GET' para URL: " + url);

    System.out.println("Response Code: " + responseCode);

    BufferedReader br =

        new BufferedReader(new InputStreamReader(con.getInputStream()));

    String inputLine;

    StringBuffer response = new StringBuffer();

    while ((inputLine = br.readLine()) != null){

        response.append(inputLine);

    }

    br.close();

    con.disconnect();

    return response.toString();

}

```

```
private String sendPostAluno(String url) throws MalformedURLException, IOException{
```

```
    URL objURL = new URL(url);
```

```
    HttpURLConnection con = (HttpURLConnection) objURL.openConnection();
```

```
    con.setDoOutput(true);
```

```
        con.setRequestMethod("POST");
```

```
        con.setRequestProperty("Content-Type", "application/json");
```

```
    //Formata em json o item da lista a ser inserido com POST
```

```
    String output = "{\"curso\":\"Mecatrônica\",\"id\":4,\"nome\":\"Alexandre\"}";
```

```
    /*
```

O POST significa inserir um novo item na lista.

Um programa que precisa escrever um dado em algum local (destino)

precisa de um OutputStream ou um Writer.

Stream significa fluxo de dados, seja para leitura ou para escrita

```
    */
```

```
    System.out.println("\nEnviando requisição 'POST' para URL: " + url);
```

//Pega a conexão aberta em con (getOutputStream()) e faz OutputStream, ou seja, faz o fluxo de dados do cliente para o servidor

```
    OutputStream os = con.getOutputStream();
```

```
    //Escreve o output transformado em Bytes
```

```
        os.write(output.getBytes());
```

```
    //Verifica o código de retorno do HttpURLConnection con
```

```
    int responseCode = con.getResponseCode();
```

```
    //Se retornar 200 significa que deu certo
```

```
    System.out.println("Response Code: " + responseCode);
```

```
    //Armazena o retorno do método POST do servidor
```

```
    BufferedReader br = new BufferedReader(new InputStreamReader((con.getInputStream())));
```

```
    String inputLine;
```

```

StringBuffer response = new StringBuffer();

//Percorre o que está armazenado no BufferedReader e guarda em inputLine

while ((inputLine = br.readLine()) != null){

    response.append(inputLine);

}

br.close();

con.disconnect();

//Devolve o resultado

return response.toString();

}

private String sendPutAluno(String url) throws MalformedURLException, IOException{

    URL objURL = new URL(url);

    HttpURLConnection con = (HttpURLConnection) objURL.openConnection();

    con.setDoOutput(true);

    con.setRequestMethod("PUT");

    con.setRequestProperty("Content-Type", "application/json");

    String output = "{\"curso\":\"Eletroeletrônica\",\"id\":2,\"nome\":\"Maria Clara\"}";

    System.out.println("\nEnviando requisição 'PUT' para URL: " + url);

    OutputStream os = con.getOutputStream();

    os.write(output.getBytes());

    int responseCode = con.getResponseCode();

    System.out.println("Response Code: " + responseCode);

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

    String inputLine;

    StringBuffer response = new StringBuffer();

    while ((inputLine = br.readLine()) != null){

```

```
        response.append(inputLine);  
    }  
  
    br.close();  
  
    con.disconnect();  
  
    return response.toString();  
}  
}
```

Antes de testar o cliente, execute o projeto WebService RESTFUL java clicando com o botão do lado do mouse sobre o nome do projeto e selecionar a opção Testar WebServices RESTFul.

REFERÊNCIAS:

Web Service Rest. Parte 1. Criando o primeiro método. Acesso em: 27/04/2017. <https://www.youtube.com/watch?v=m-8x87qN8KE>

<http://www.json.org/>

<http://www.devmedia.com.br/introducao-ao-formato-json/25275>