# **Criando um serviço RESTful com Web API em C#**

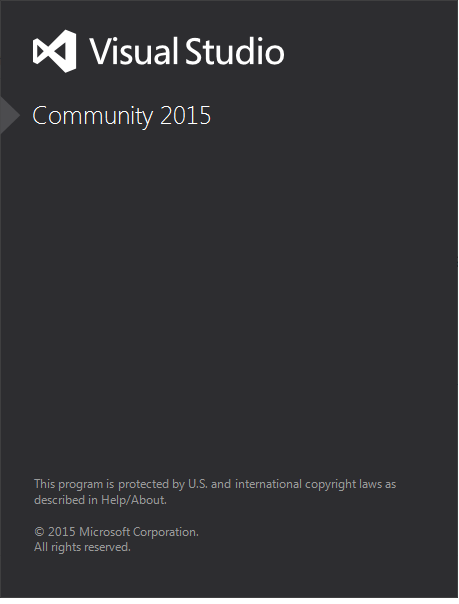
Nesse tutorial vamos aprender a criar um serviço RESTful usando Web API, esse é um recurso do .NET Framework que permiti criar de uma forma muito fácil serviços REST, serviços com essa arquitetura tem uma usabilidade ampla entre varias plataformas, quando criamos um serviço REST usando WEB API, ele não está restrito para ser usando apenas na plataforma .NET, podemos também consumir esse serviço em outras plataformas.

Você pode querer ler também:

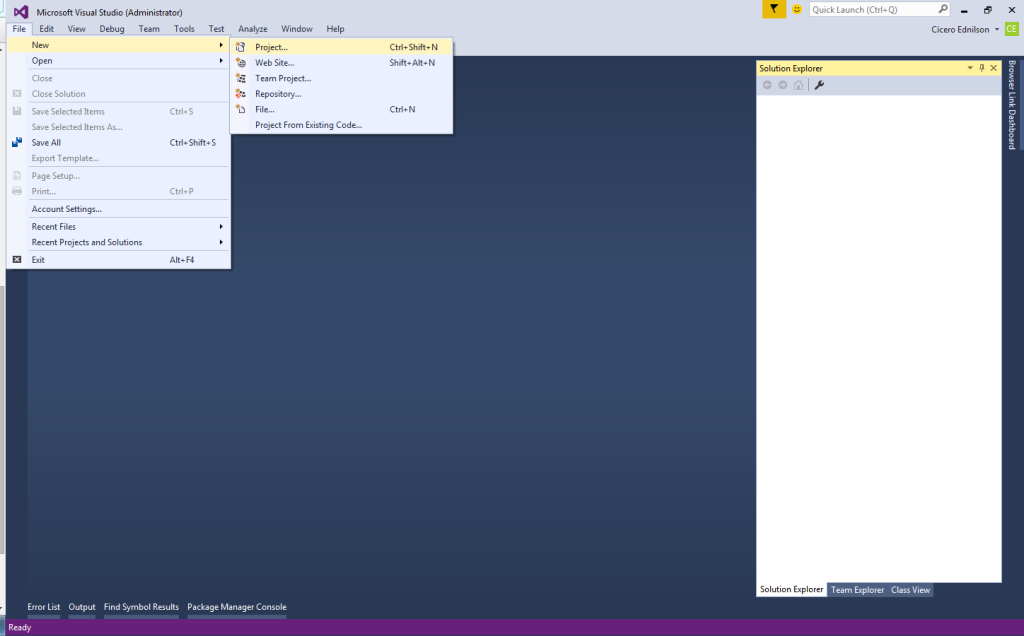
* [Criando um Client RESTful em C#](http://www.ciceroednilson.com.br/criando-um-client-restful-em-c/)
* [Criando um serviço Rest com WCF usando C# e Mysql.](http://www.ciceroednilson.com.br/criando-um-servico-rest-com-wcf-usando-c-e-mysql/)
* [Criando um Web Service RESTful com JAX-RS e Jboss 7.1 para calcular o preço de um veículo para o consumidor final.](http://www.ciceroednilson.com.br/criando-um-web-service-restful-com-jax-rs-e-jboss-7-1-para-calcular-o-preco-de-um-veiculo-para-o-consumidor-final/)

#### **Criando o Serviço Rest**

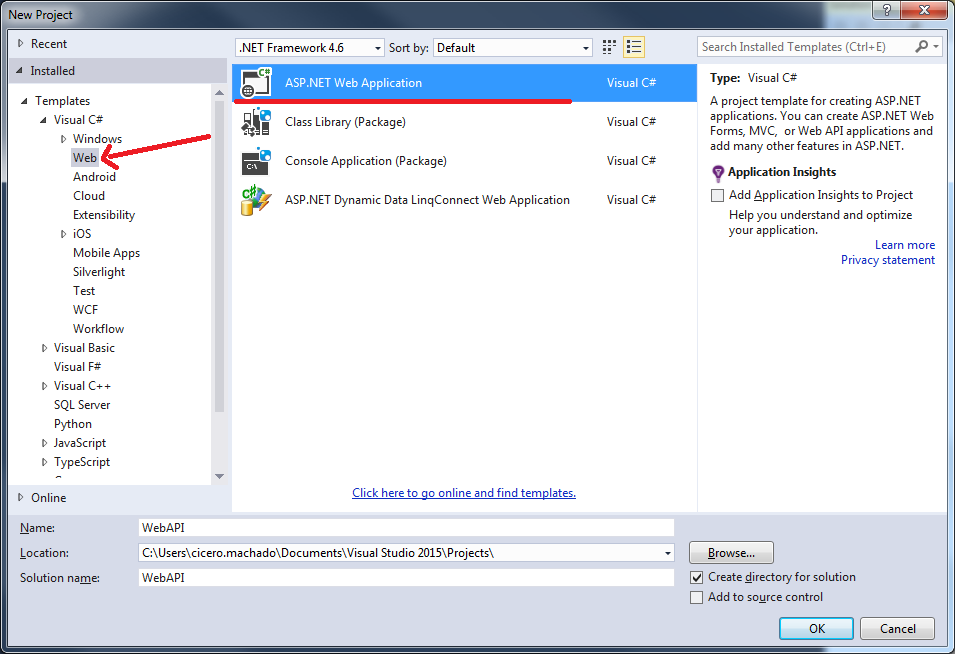
Vamos abrir o Visual Studio.



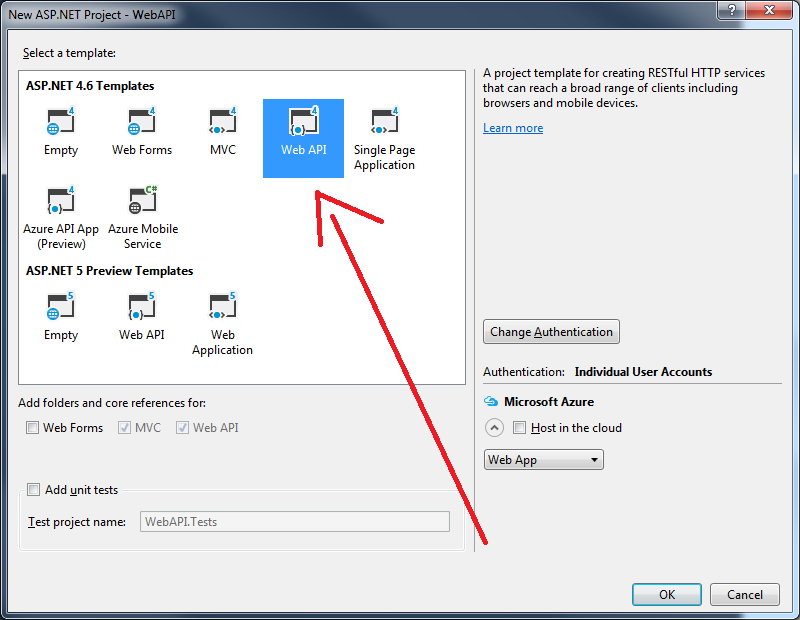
Agora vamos até o menu File -> New -> Project…



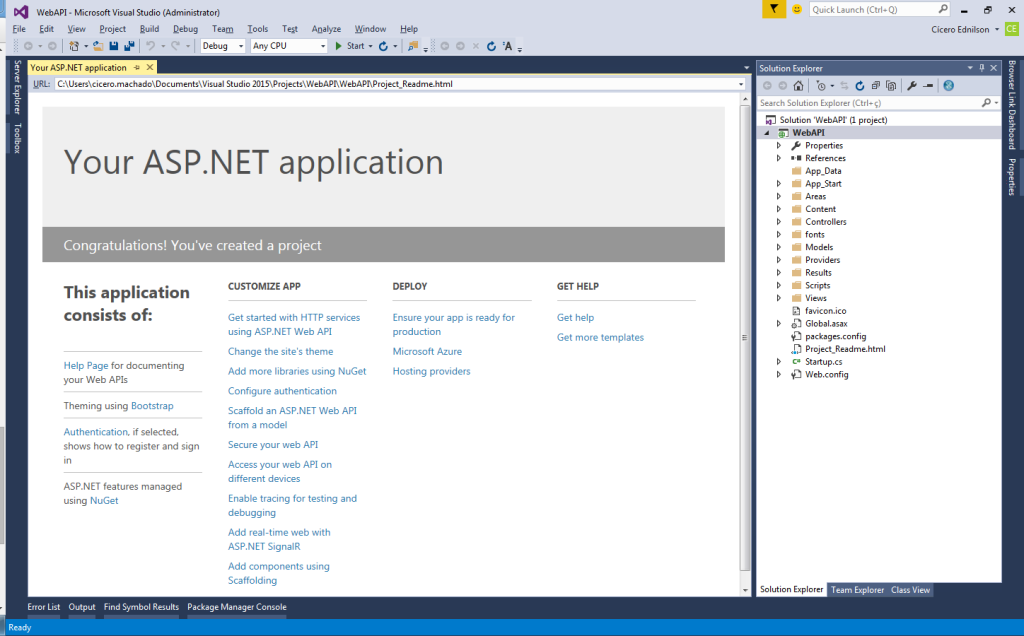
Agora em Web vamos selecionar o item ASP.NET Web Application, depois em Name vamos colocar o nome de WebAPI para o nosso projeto, depois basta clicar em OK.



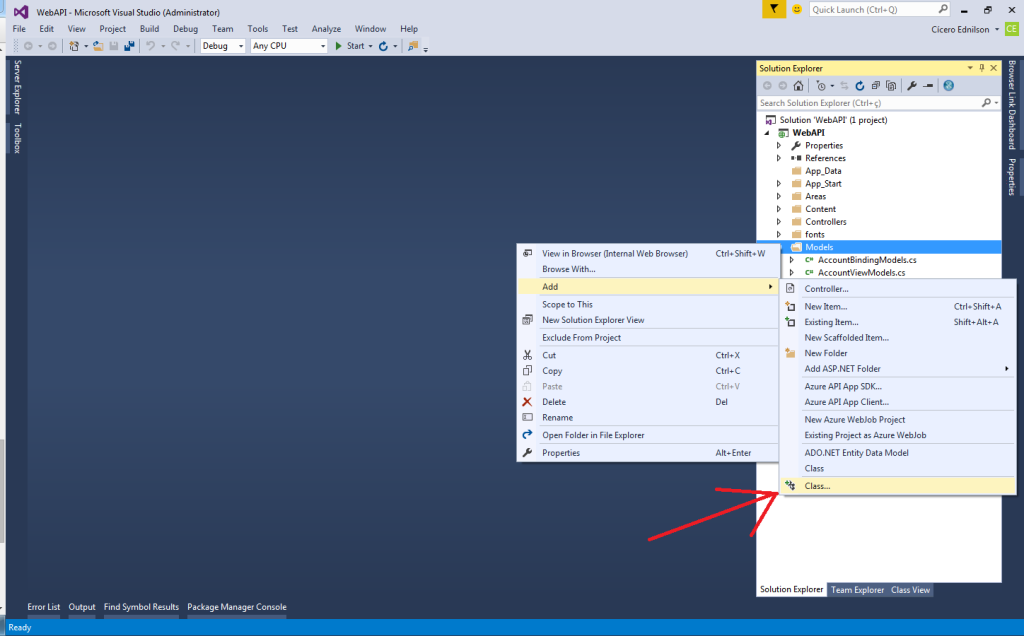
Agora vamos selecionar Web API e depois vamos clicar em OK.



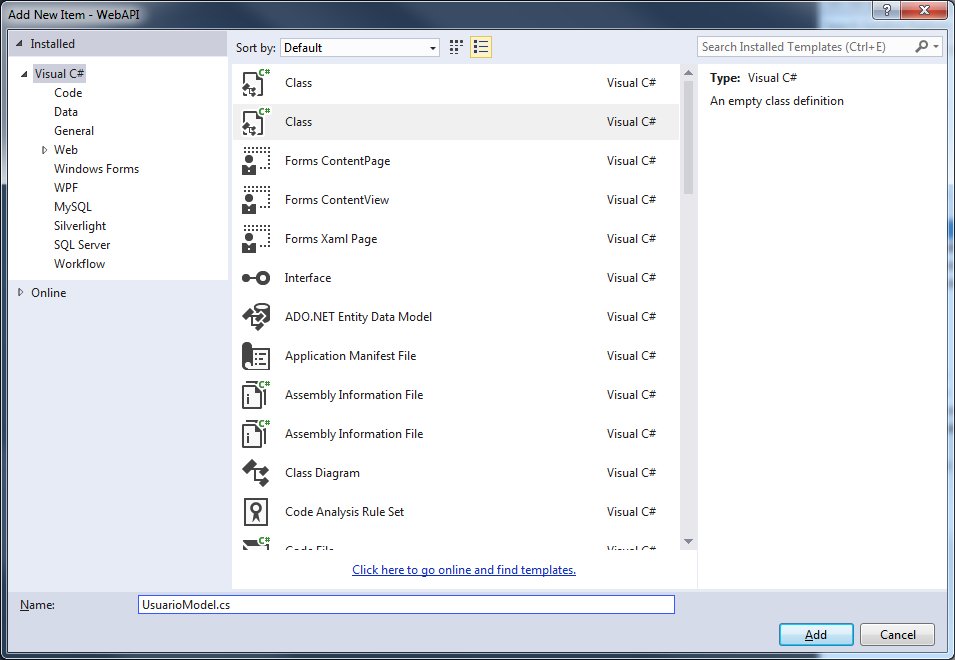
Podemos ver que foi criado um projeto com a arquitetura do ASP.NET MVC apesar de usarmos um template para a criação de um Web Service Rest, dessa forma se fôssemos criar uma aplicação ASP.NET MVC e precisássemos disponibilizar algumas informações do nosso sistema, poderíamos fazer em um único projeto.



Agora vamos clicar sobre a pasta Models e vamos até Add-> Class..



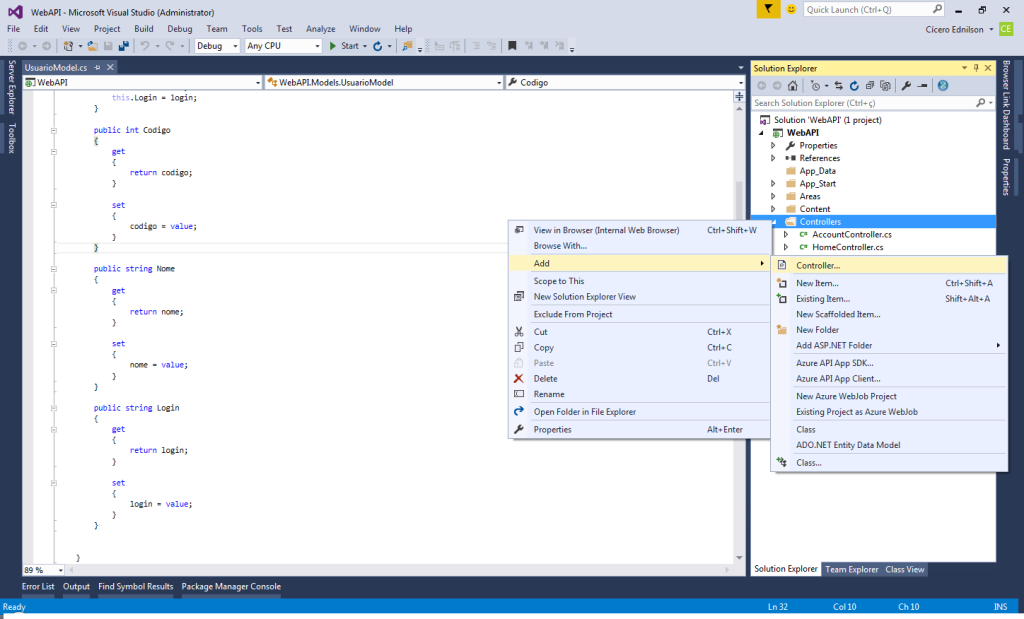
Agora vamos dar o nome de UsuarioModel para a nossa classe, depois basta clicar em OK.



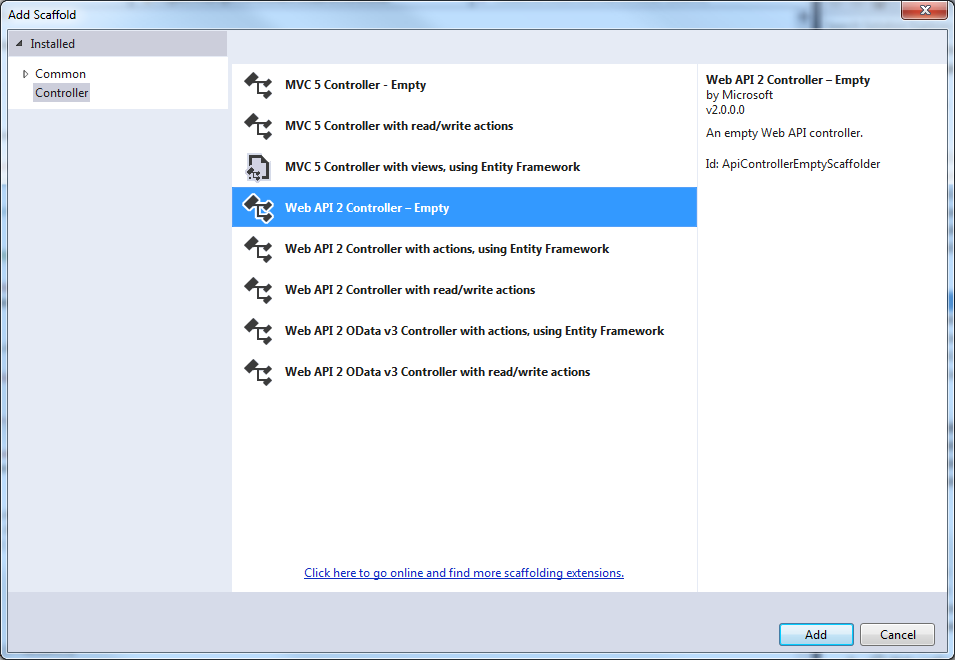
Agora vamos deixar nossa classe com os atributos e propriedades como mostra o código abaixo.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62 | **namespace** WebAPI.Models  {  **public** **class** UsuarioModel  {    **private** **int** codigo;  **private** **string** nome;  **private** **string** login;    **public** UsuarioModel()  {  }    **public** UsuarioModel(**int** codigo, **string** nome, **string** login)  {  **this**.Codigo = codigo;  **this**.Nome = nome;  **this**.Login = login;  }    **public** **int** Codigo  {  **get**  {  **return** codigo;  }    **set**  {  codigo = **value**;  }  }    **public** **string** Nome  {  **get**  {  **return** nome;  }    **set**  {  nome = **value**;  }  }    **public** **string** Login  {  **get**  {  **return** login;  }    **set**  {  login = **value**;  }  }      }  } |

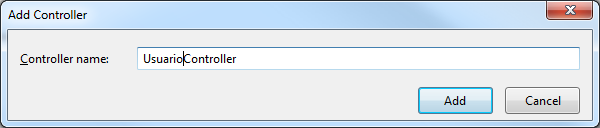
Agora vamos criar um Controller para implementarmos os nosso métodos que vão receber e retornar os dados em JSON, para isso vamos clicar com o botão direito sobre a pasta Controllers e vamos até Add->Controller…



Na tela que foi aberta vamos selecionar Web API 2 Controller – Empty e depois vamos clicar em Add.



Vamos dar o nome de UsuarioController e clicar em Add.



Agora vamos deixar o código do nosso Controller igual o código abaixo.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74  75  76  77  78  79 | **using** System.Collections.Generic;  ﻿**using** System.Collections.Generic;  **using** System.Linq;  **using** System.Web.Http;  **using** WebAPI.Models;    **namespace** WebAPI.Controllers  {    [RoutePrefix("api/usuario")]  **public** **class** UsuarioController : ApiController  {  **private** **static** List<UsuarioModel> listaUsuarios = new List<UsuarioModel>();    [AcceptVerbs("POST")]  [Route("CadastrarUsuario")]  **public** **string** CadastrarUsuario(UsuarioModel usuario)  {    listaUsuarios.**Add**(usuario);    **return** "Usuário cadastrado com sucesso!";  }    [AcceptVerbs("PUT")]  [Route("AlterarUsuario")]  **public** **string** AlterarUsuario(UsuarioModel usuario)  {    listaUsuarios.**Where**(n => n.Codigo == usuario.Codigo)  .**Select**(s =>  {  s.Codigo = usuario.Codigo;  s.Login = usuario.Login;  s.Nome = usuario.Nome;    **return** s;    }).ToList();        **return** "Usuário alterado com sucesso!";  }    [AcceptVerbs("DELETE")]  [Route("ExcluirUsuario/{codigo}")]  **public** **string** ExcluirUsuario(**int** codigo)  {    UsuarioModel usuario = listaUsuarios.**Where**(n => n.Codigo == codigo)  .**Select**(n => n)  .First();    listaUsuarios.**Remove**(usuario);    **return** "Registro excluido com sucesso!";  }    [AcceptVerbs("GET")]  [Route("ConsultarUsuarioPorCodigo/{codigo}")]  **public** UsuarioModel ConsultarUsuarioPorCodigo(**int** codigo)  {    UsuarioModel usuario = listaUsuarios.**Where**(n => n.Codigo == codigo)  .**Select**(n => n)  .FirstOrDefault();    **return** usuario;  }    [AcceptVerbs("GET")]  [Route("ConsultarUsuarios")]  **public** List<UsuarioModel> ConsultarUsuarios()  {  **return** listaUsuarios;  }  }  } |

Agora vamos entender os itens que usamos no nosso UsuarioController.

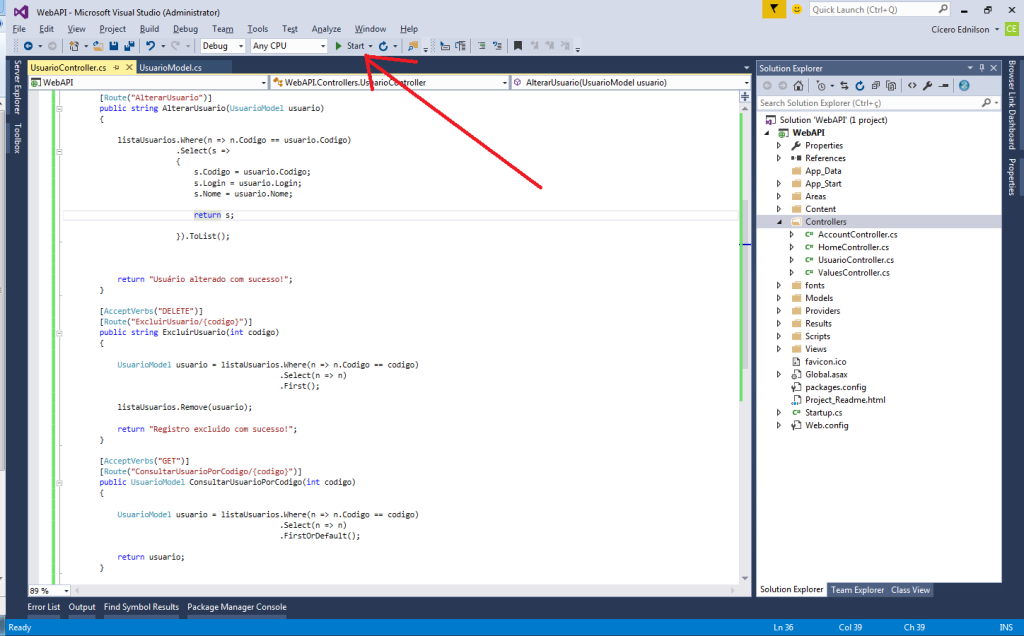
**RoutePrefix:** Determina a rota de acesso para o controller, exemplo: http://localhost:porta/api/usuario/metodo.

**AcceptVerbs:**Determina o tipo do método http que o método do Controller vai responder.

**Route:**Determina o nome para acesso ao método, no nosso exemplo coloquei o mesmo nome do método do Controller, mas você pode definir qualquer outro nome para acesso.

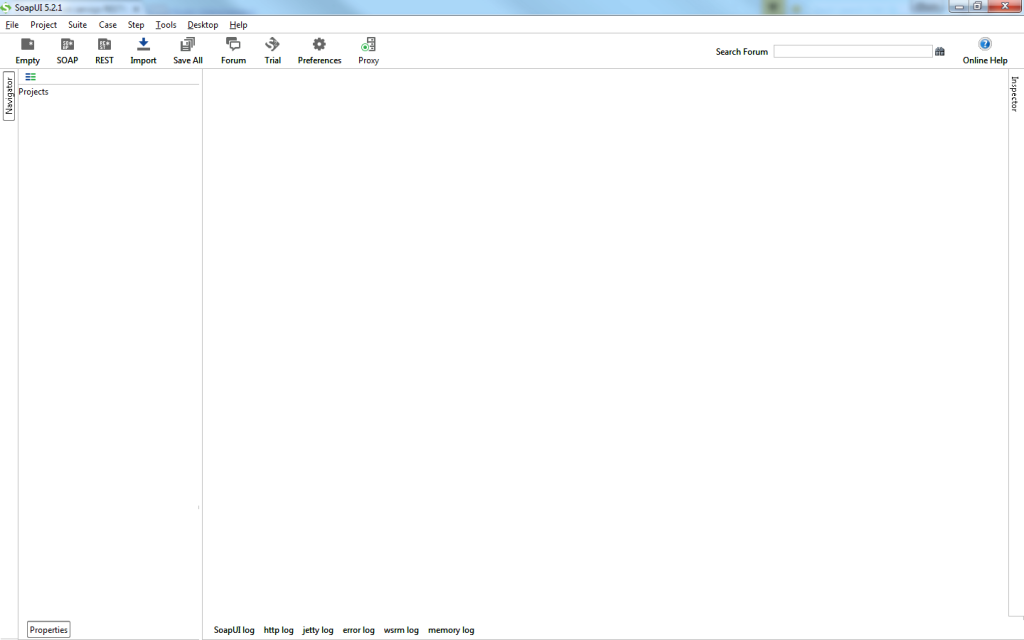
Outra coisa que também podemos fazer em vez de usar o AcceptVerbs é usar as opções HttpPost, HttpPut, HttpDelete e HttpGet, esses atributos também vão responder da mesma forma.

Agora vamos testar o nosso serviço Rest que criamos, vamos clicar em Start no Visual Studio para deixar a nossa aplicação rodando no IIS Express.

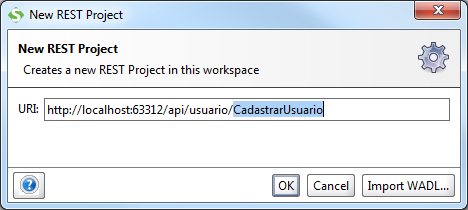


#### **Testando o serviço no SoapUI.**

Vamos clicar na opção REST do SoapUI.



Na tela que foi aberta vamos colocar a url no nosso serviço com o método **CadastrarUsuario** que criamos e depois vamos clicar em OK.

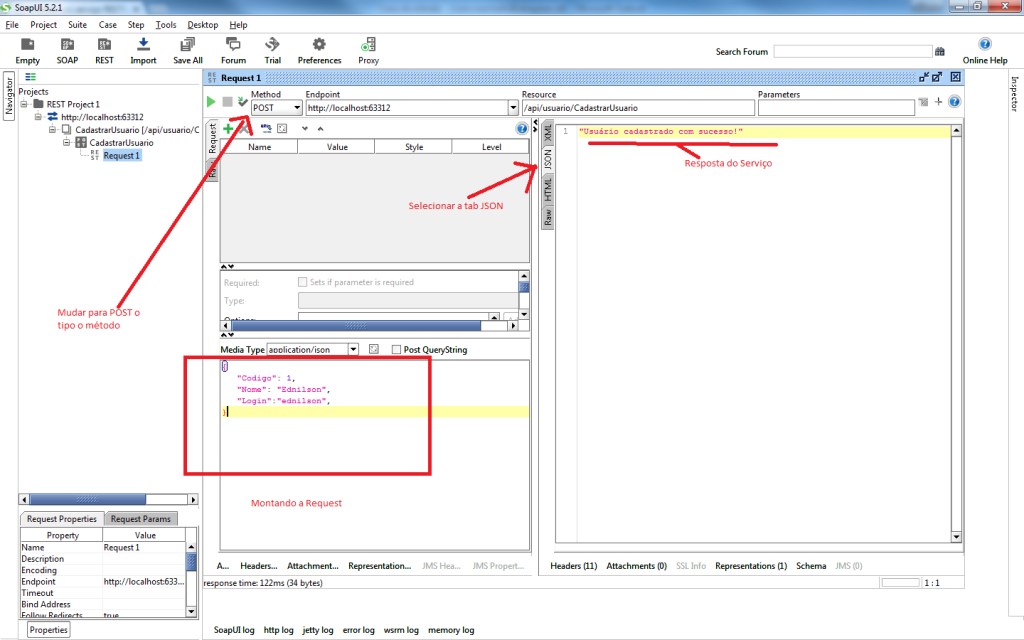


Agora vamos testar a nossa operação CadastrarUsuario, no nosso serviço vamos conseguir simular um crud devido a lista que criamos como static, então enquanto o servidor estiver rodando os dados vão ser mantidos em memória.

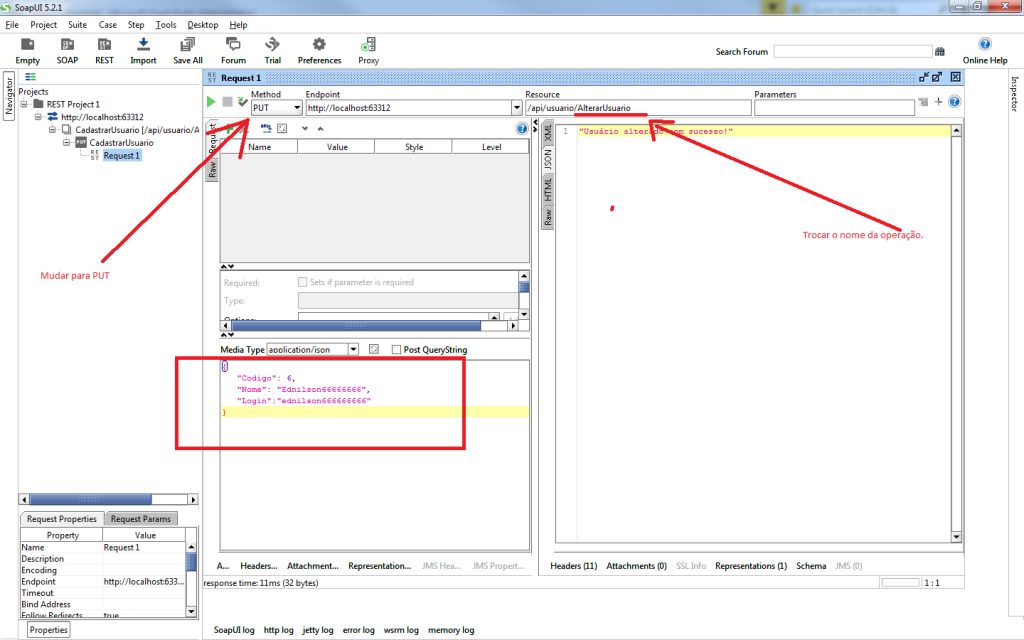
Cadastre uns seis usuários para realizarmos os testes das outras operações, troque pelo menos o valor do campo código.

Parâmetros de Request.

{  
“Codigo”: 1,  
“Nome”: “Ednilson”,  
“Login”:”ednilson”  
}

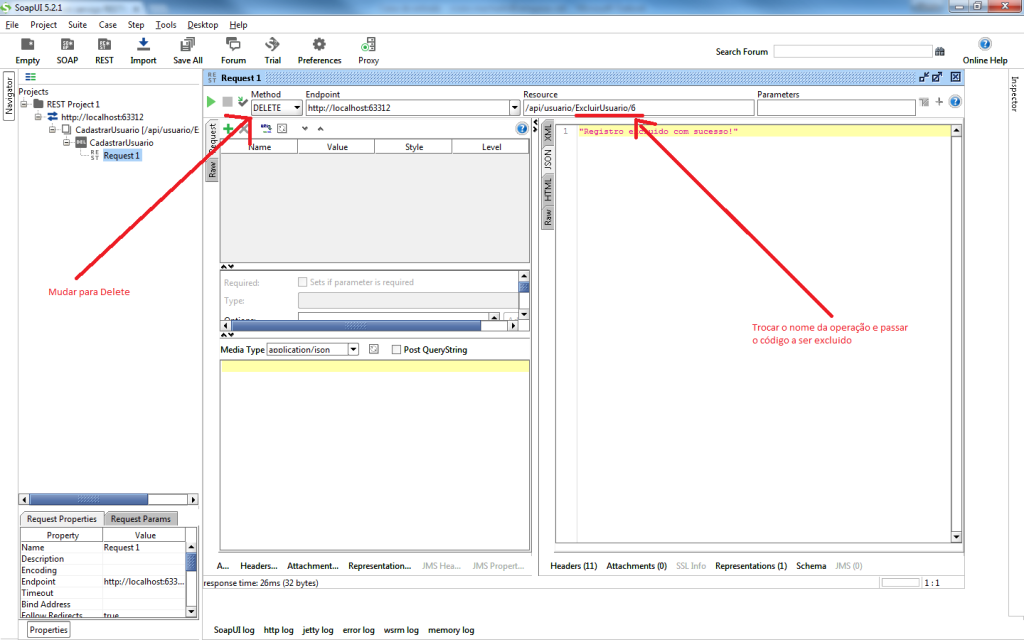


Agora vamos testar a operação AlterarUsuario como mostra a imagem abaixo.

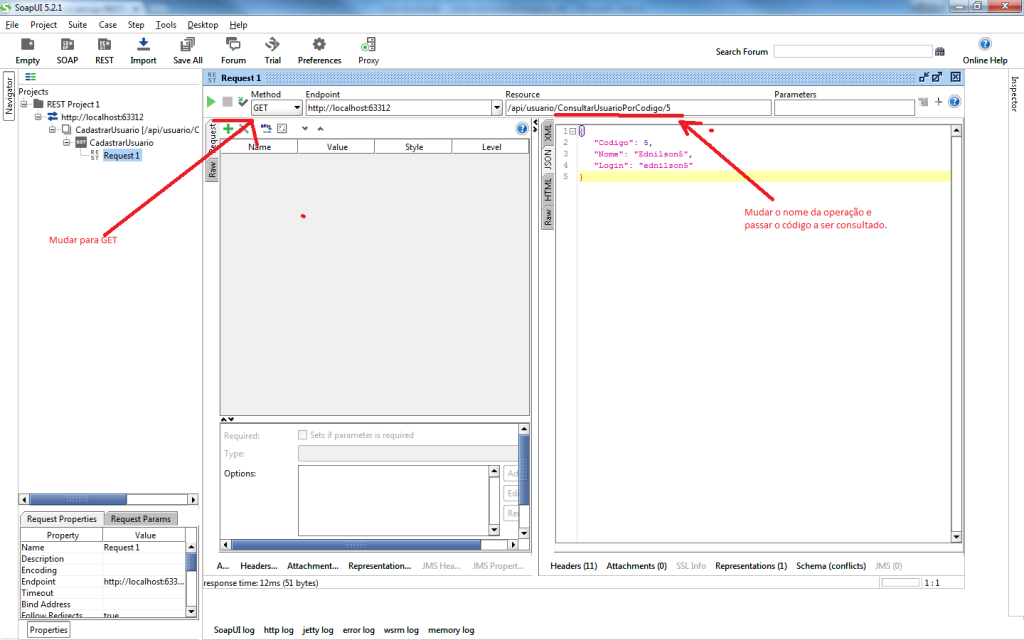


Agora vamos testar a operação**ExcluirUsuario**, basta mudar o nome do método na url e

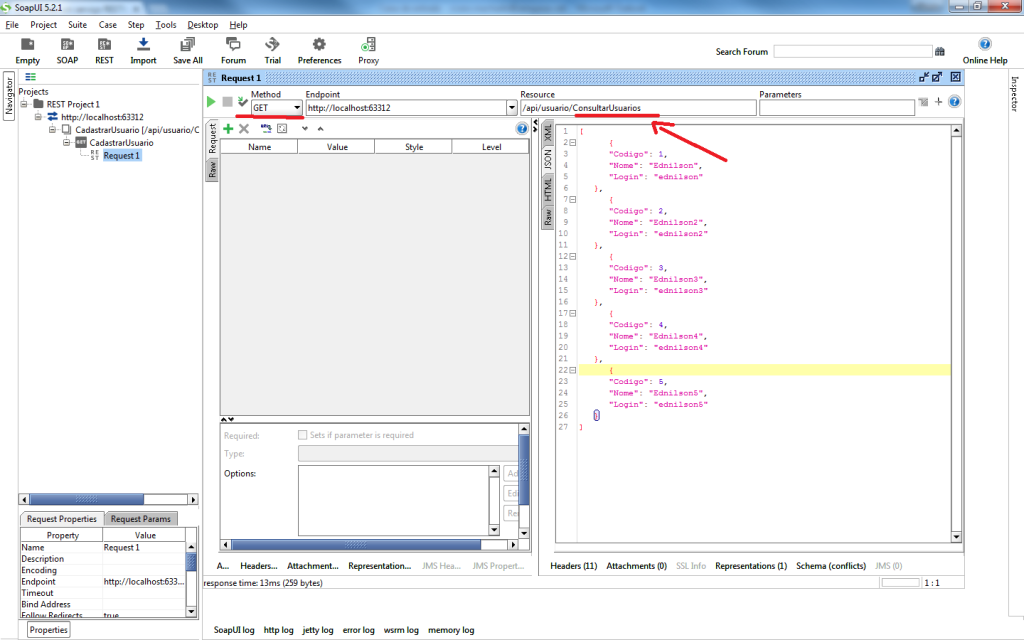
também passar o código a ser excluído.



Agora vamos testar a operação ***ConsultarUsuarioPorCodigo***, vamos mudar o nome do método na url e também vamos passar o código a ser consultado.



E por fim vamos testar a operação ConsultarUsuarios que vai retornar todos os usuários que estão na nossa lista.



Observação:

Esse tutorial teve um comentário bem interessante do Paulo Antônio Matos Barros, Ele informou que o padrão determinado para uri em operações crud seja uma por recurso e cada verbo [get,post,put,delete] seja usado para realizar a sua ação determinada, então a definição do nosso serviço deveria ficar como mostra o código abaixo.

|  |  |
| --- | --- |
| 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29  30  31  32  33  34  35  36  37  38  39  40 | [RoutePrefix("api")]  **public** **class** UsuarioController : ApiController  {    [AcceptVerbs("POST")]  [Route("usuario")]  **public** **string** CadastrarUsuario(UsuarioModel usuario)  {  *//AÇÃO*  }    [AcceptVerbs("PUT")]  [Route("usuario")]  **public** **string** AlterarUsuario(UsuarioModel usuario)  {  *//AÇÃO*  }    [AcceptVerbs("DELETE")]  [Route("usuario/{codigo}")]  **public** **string** ExcluirUsuario(**int** codigo)  {  *//AÇÃO*  }    [AcceptVerbs("GET")]  [Route("usuario/{codigo}")]  **public** UsuarioModel ConsultarUsuarioPorCodigo(**int** codigo)  {  *//AÇÃO*  }    [AcceptVerbs("GET")]  [Route("usuario")]  **public** List<UsuarioModel> ConsultarUsuarios()  {  *//AÇÃO*  }    } |

E abaixo a forma de execução da implementação realizada acima.

POST executa o método CadastrarUsuario através da url http://localhost:63312/api/usuario

PUT executa o método AlterarUsuario através da url http://localhost:63312/api/usuario

DELETE executa o método ExcluirUsuario através da url http://localhost:63312/api/usuario/3

GET executa o método ConsultarUsuarioPorCodigo através da url http://localhost:63312/api/usuario/4

GET executa o método ConsultarUsuarios através da url http://localhost:63312/api/usuario

Com essas alterações podemos deixar nossas URIs seguindo boas práticas para desenvolvimento de serviços Rest.

Você pode pegar o código fonte desse tutorial [aqui](http://www.ciceroednilson.com.br/wp-content/uploads/2016/01/WebAPI.zip).

Até a próxima.