02

Acessando o serviço pelo browser

Agora queremos acessar o nosso serviço pelo navegador! Para conseguir este resultado, a primeira coisa que precisamos fazer é informar no contrato do nosso serviço que eles podem ser acessados pelo browser. Podemos fazer isso através da anotação `WebInvoke`:

```
[ServiceContract]
public interface IClienteService
{
    [OperationContract]
    [WebInvoke]
    void Add(Cliente c);

    [OperationContract]
    Cliente Buscar(string nome);
}
```

Veja que o Visual Studio irá pedir para gerar esta classe, porém ela já existe dentro de `System.ServiceModel.Web`.

Vamos adicionar a referência dele ao nosso projeto. No *Solution Explorer*, dentro de **Passagens**, vamos clicar com o botão direito do mouse em *References* e selecionar *Add -> Reference*. Na aba *Assemblies* selecionaremos **System.ServiceModel.Web**.

Depois, utilizando o atalho *CTRL + . (PONTO)** em cima da palavra `WebInvoke`, o Visual Studio irá sugerir o `using`, e assim poderemos usar a anotação no nosso projeto.

Para poder acessar o nosso serviço pelo browser, precisamos passar alguns parâmetros para o `WebInvoke`:

- O método.
- O formato da resposta.
- A url que devemos acessar.

Portanto, a nossa anotação ficará semelhante com a abaixo:

```
public interface IClienteService
{
    [OperationContract]
    [WebInvoke(Method = "GET" , ResponseFormat = WebMessageFormat.Xml , UriTemplate = "addClien
    void Add(Cliente c);

    [OperationContract]
    Cliente Buscar(string nome);
}
```

Mas como pegamos o **nome** e o **cpf** da *url*, e passamos para o nosso método?! Bem, o **WCF** segue a convenção de que, o nome do **parâmetro na url** é o nome do parâmetro no método a ser invocado. Sendo assim, precisamos mudar os parâmetros do nosso método `Add` para:

```
public interface IClienteService
{
    [OperationContract]
    [WebInvoke(Method = "GET" , ResponseFormat = WebMessageFormat.Xml , UriTemplate = "addClient
    void Add(string nome, string cpf);

    [OperationContract]
    Cliente Buscar(string nome);
}
```

Como mudamos a interface, também precisamos refletir esta alteração na implementação dela. Portanto, a classe `ClienteService` ficará com o seguinte método `Add` :

```
public void Add(string nome, string cpf)
{
    Cliente c = new Cliente();
    c.Nome = nome;
    c.Cpf = cpf;

    ClienteDao dao = new ClienteDao();
    dao.Add(c);
}
```

Além disso, também precisamos informar para o nosso *endpoint* que ele pode ser acessado via web. Para isso, dentro de **Passagens**, no *Solution Explorer*, vamos clicar com o botão direito no **App.config** e selecionar *Edit WCF Configurations*, e vamos acessar a pasta *Advanced -> Endpoint Behaviors*.

Vamos adicionar um *endpoint behavior*, clicando em `New Endpoint Behavior Configuration` e nomeá-lo como **web**. Logo após isso, vamos clicar em *Add...*, selecionar a opção **webHttp** e clicar em *Add*. Feito isso, vamos acessar o nosso *endpoint* na pasta *Services -> Passagens.ClienteService -> Endpoints* e na opção *BehaviorConfiguration*, vamos selecionar a opção **web** (o *endpoint* que acabamos de criar). Também precisamos alterar a opção *Binding* para **webHttpBinding**. Salvando as configurações, o nosso serviço deve estar disponível via web.

Defina o projeto **Passagens** como *startUp Project* e aperte *Start* para iniciar o serviço. Acesse pelo browser o endereço http://localhost:8080 (http://localhost:8080). Deverá ser exibido a home do nosso serviço.

Agora acesse o método que acabamos de configurar com a seguinte url:
http://localhost:8080/cliente/addCliente/NomeDoCliente;CpfDoCliente
(http://localhost:8080/cliente/addCliente/NomeDoCliente;CpfDoCliente).

Acessando ela, deverá aparecer uma página em branco. Porque!? O retorno do nosso serviço é void! Então vamos mudar o retorno para `true` , em `ClienteService` :

```
public bool Add(string nome,string cpf)
{
    Cliente c = new Cliente();
    c.Nome = nome;
    c.Cpf = cpf;

    ClienteDao dao = new ClienteDao();
    dao.Add(c);
```

```
        return true;
    }
```

E em `IClienteService`, precisamos dizer que o método `Add` retorna um booleano:

```
    public interface IClienteService
    {
        [OperationContract]
        [WebInvoke(Method = "GET" , ResponseFormat = WebMessageFormat.Xml , UriTemplate = "addClient
        bool Add(string nome, string cpf);

        [OperationContract]
        Cliente Buscar(string nome);
    }
```

Agora o retorno deverá ser um **xml**, quando a página for acessada.

Mas nós não conseguimos ver quais clientes estão cadastrados! Então vamos criar um método que lista todos os clientes para nós. Para isso, vamos na interface do nosso serviço (`IClienteService`), e criar um novo método:

```
    [OperationContract]
    [WebInvoke(Method = "GET" , ResponseFormat = WebMessageFormat.Xml , UriTemplate = "getClientes/'
    List<Cliente> getClientes();
```

A implementação desse método na classe `ClienteService` fica:

```
    public List<Cliente> getClientes() {
        return ClienteDao.clientes;
    }
```

Mas para podermos acessar o atributo `cliente` de `ClienteDao`, precisamos colocá-lo como público. Portanto, na classe `ClienteDao`:

```
    public static List<Cliente> clientes = new List<Cliente>();
```

Agora podemos ver todos os clientes cadastrados! Basta acessar a url http://localhost:8080/cliente/getClientes/ (http://localhost:8080/cliente/getClientes/).

Agora, para poder buscar um cliente especifico, basta adicionar a anotação `WebInvoke` no nosso método `Buscar`:

```
    [OperationContract]
    [WebInvoke(Method = "GET" , ResponseFormat = WebMessageFormat.Xml , UriTemplate = "SearchCliente
    Cliente Buscar(string nome)
```

Experimente retornar também no formato **Json**, tanto para o método `Add` quanto no `Buscar`.