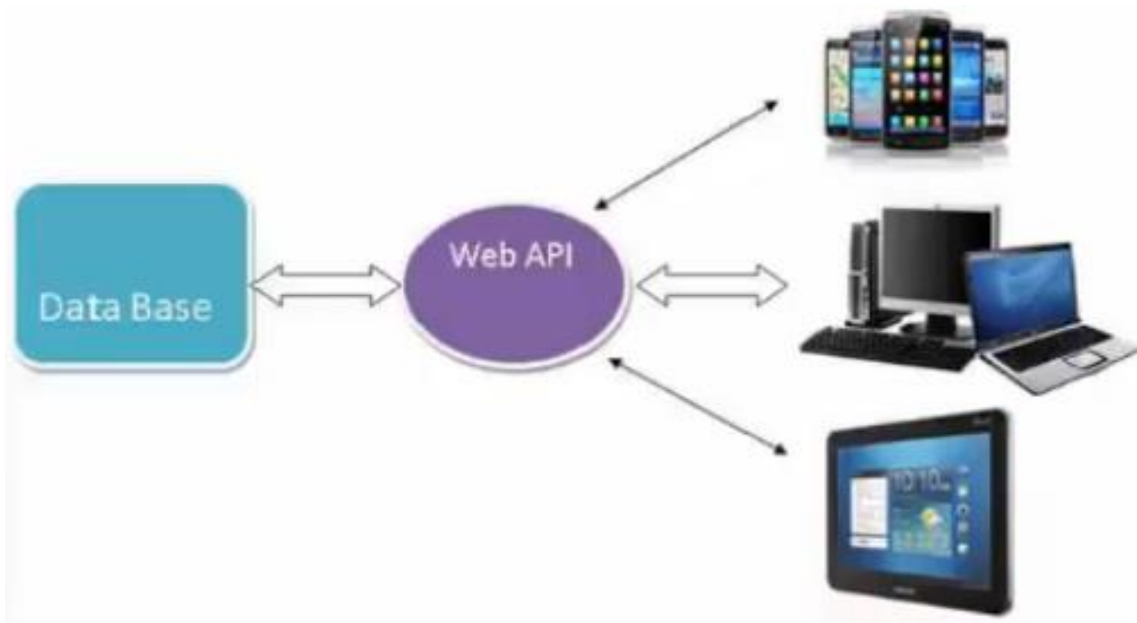


A ASP .NET Core Web API

A ASP .NET Core Web API é um framework para a construção de serviços baseados em HTTP que podem ser acessados em diferentes aplicativos e em diferentes plataformas, como WEB, mobile, nuvem, etc.

Dessa forma a WEB API oferece diversos serviços que podem ser consumidos por diversos tipos de clientes.

E esses serviços são criados para acessar dados de serviços em aplicação WEB, aplicativos móveis, e outros dispositivos externos.



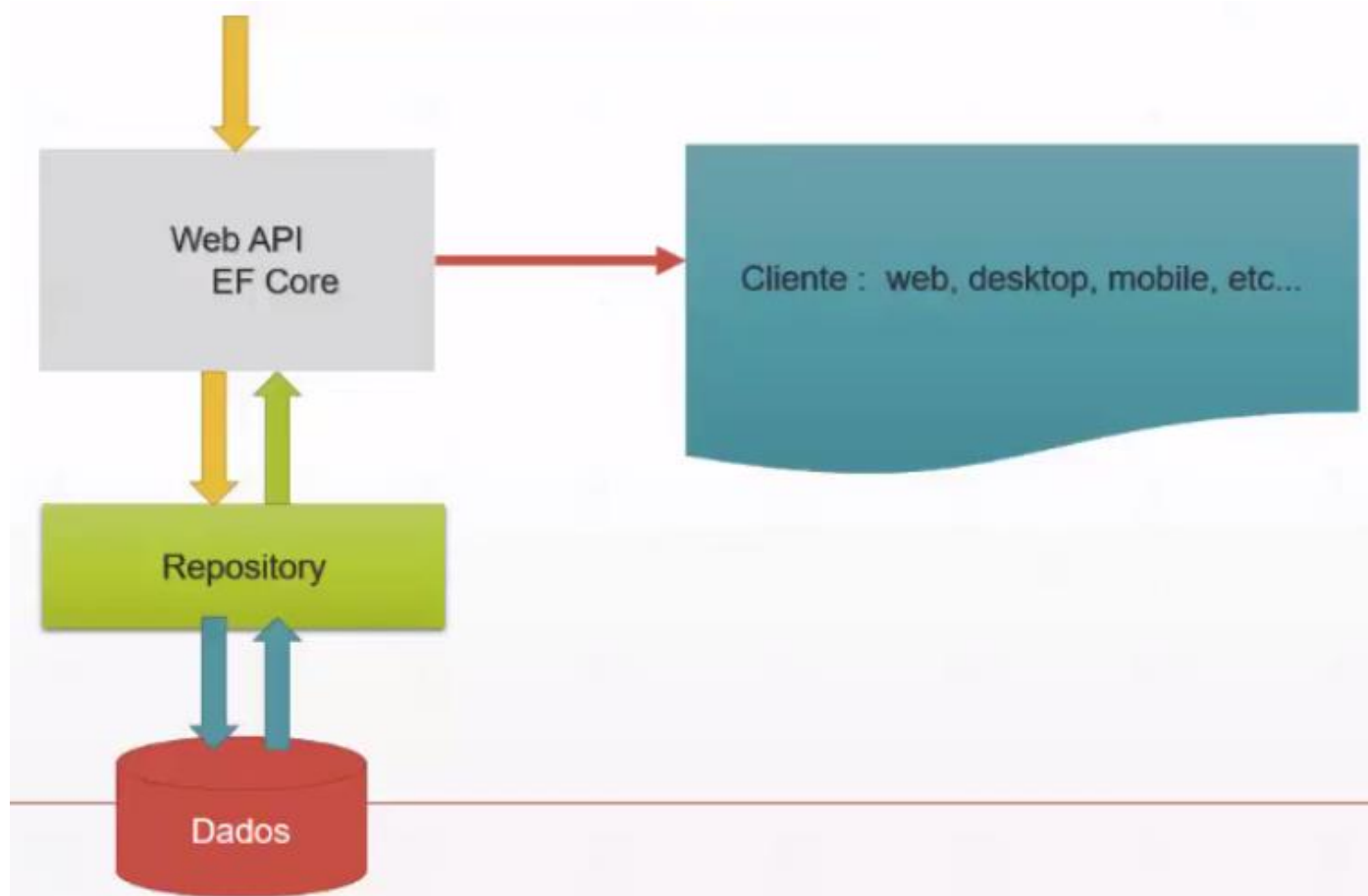
- Criar serviços **RESTful** que é um estilo arquitetônico para aplicações em rede de hipermídia.
- Criar serviços da WEB que são leves, fáceis de manter e escalar e oferecem suporte a largura de banda limitada.
- Criar um serviço HTTP simples que suporta XML, JSON e outros formatos de dados.

O estilo de arquitetura web api é um estilo híbrido que é derivado do estilo REST

Nessa aula criaremos uma WEB API para expor serviços para gerenciar informações de usuários.

Usaremos o Entity Framework Core e o SQL Server.

Definiremos uma **WEB API** usando o **Entity Framework**, que ao receber uma requisição vai acessar um **repositório**, que vai buscar informações em um **banco de dados** e retornar essas informações para um **cliente** que vai consumir essas informações, que pode ser uma aplicação **WEB, Desktop, mobile**, etc



Usamos MVC para criar aplicações WEB onde temos dados e suas visualizações exibidas em páginas HTML.

Já a WEB API ela é usada para criar serviços aderentes a arquitetura **REST** que retornam apenas dados que poderão ser consumidos por diversos tipos de clientes

*A utilização da arquitetura **REST**, portanto, permite a comunicação entre aplicações. Ao abrir o navegador, ele estabelece uma conexão TCP/IP com o servidor de destino e envia uma requisição GET HTTP, com o endereço buscado.*

O servidor, então, interpreta a requisição, retornando com uma resposta HTTP ao navegador. Essa resposta pode ser completa, com representações em formato HTML, ou apresentar erro, afirmando que o recurso solicitado não foi encontrado.

Esse processo é repetido diversas vezes em um período de navegação. Cada nova URL aberta ou formulário submetido refaz as etapas que descrevemos. Dessa forma, esses elementos permitem a criação de aplicações web, desenhando a forma como navegamos na internet.

Os Web Services que adotam REST são mais leves e perfeitos na busca da metodologia ágil. Outro diferencial é a flexibilidade, sendo possível escolher o formato que melhor se encaixa para as mensagens do sistema. Os mais utilizados, além do texto puro, são JSON e XML, dependendo da necessidade de cada momento.

Antes da ASP .NET Core, a WEB API e o MVC eles eram muito parecidos. Ambos seguiam o padrão MVC e usavam controlers e actions.

A diferença básica era que a WEB API não tinha um mecanismo de views Razor. Em vez disso, ela foi projetada para ser usada com APIs REST

Já o MVC foi projetado para aplicativos WEB com front-ends em HTML.

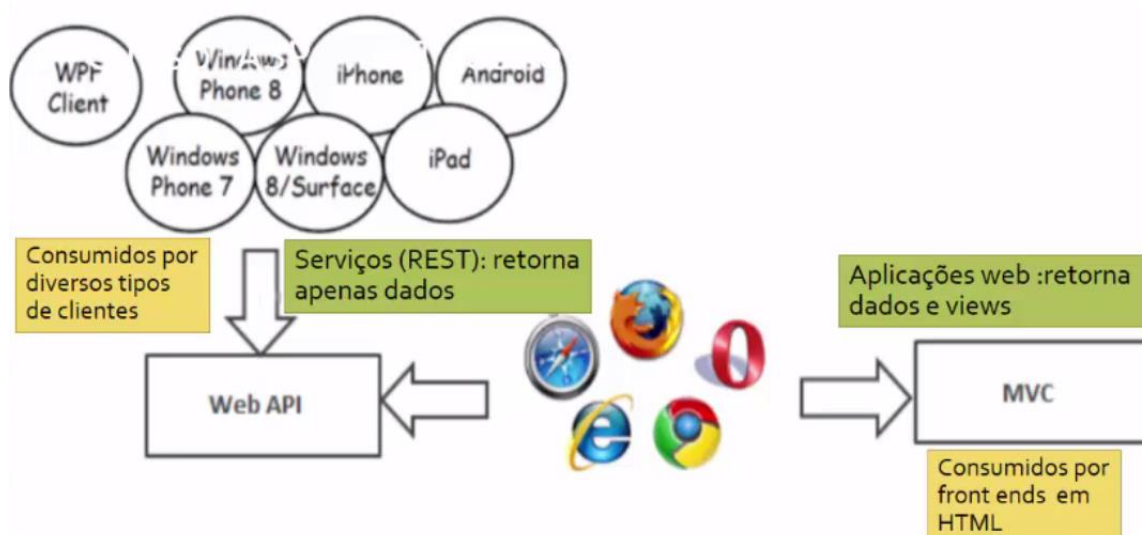
Na ASP .NET MVC5, nós tínhamos 2 tipos de controladores: um para atender a WEB API e outro para atender o MVC, e esses controladores estavam em namespaces diferentes.

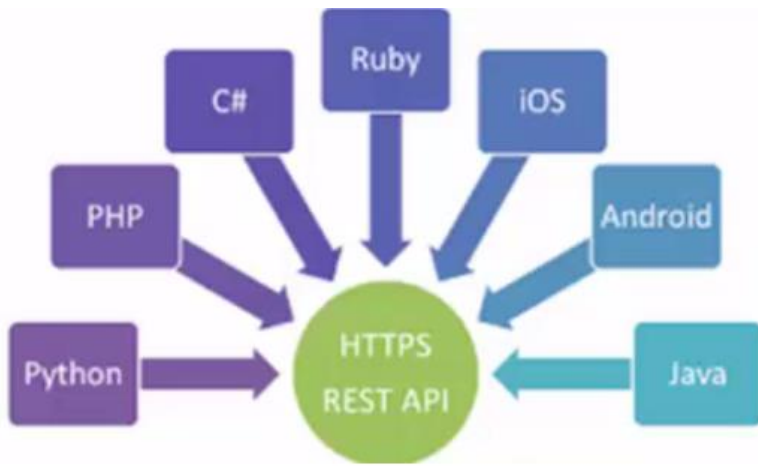
E tínhamos 2 configurações de rotas distintas, um para controles MVC, e outros para controles WEB API, isso na ASP .NET MVC5.

Na ASP .NET Core, a WEB API e o MVC agora eles estão integrados e agora temos apenas uma classe controller tanto para WEB API, como para MVC e eles usam o mesmo roteamento.

Se você quiser criar aplicações WEB com dados e visualizações HTML? Então você vai criar um controle MVC.

Você quer criar um serviço que retorne dados que podem ser consumidos por diversos clientes? Então você vai usar um controller





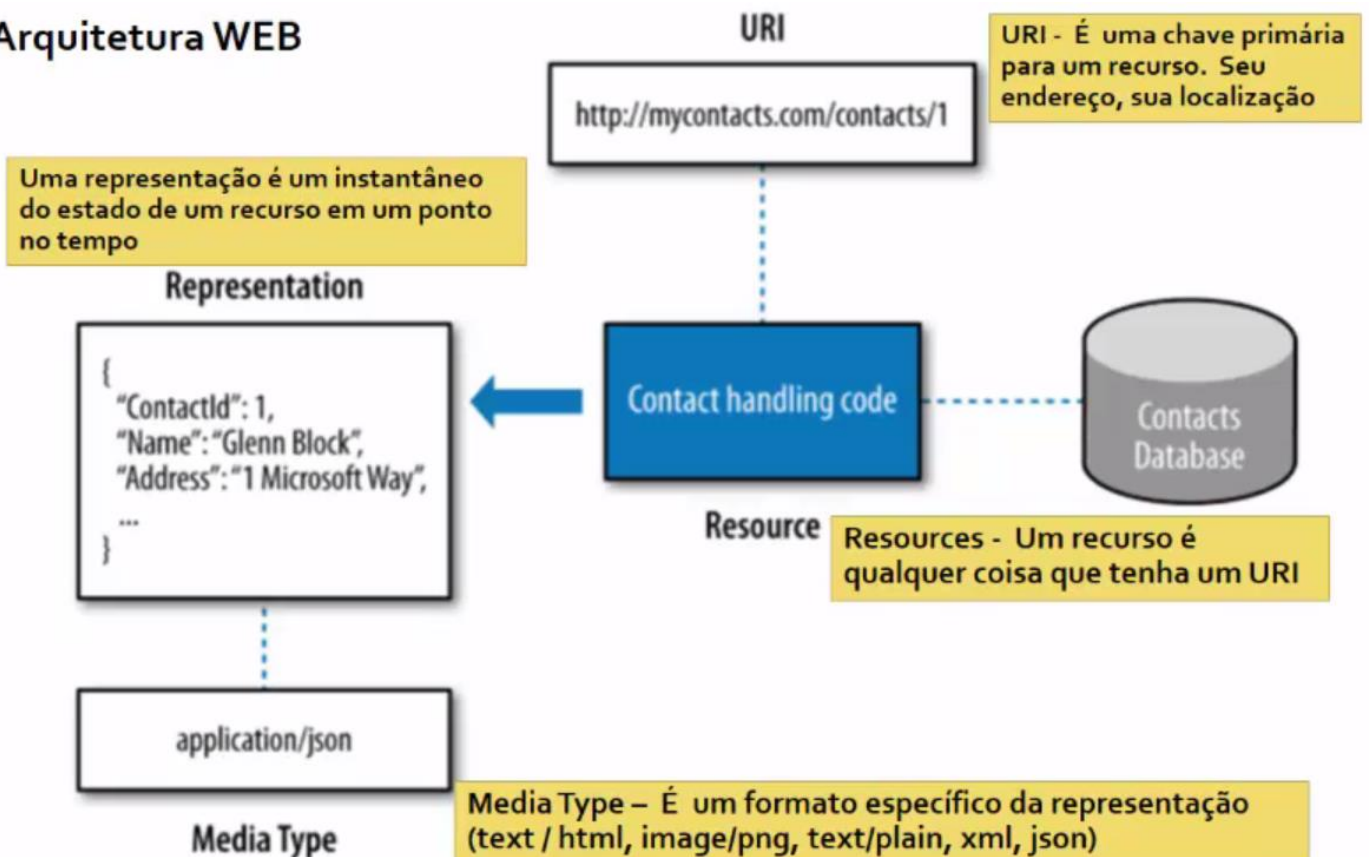
A ASP .NET WEB API é um framework para construir serviços HTTP (REST) sobre a plataforma .NET.

Representational State Transfer (REST) - Transferência de Estado Representacional - é um estilo de arquitetura para a concepção de sistemas distribuídos.

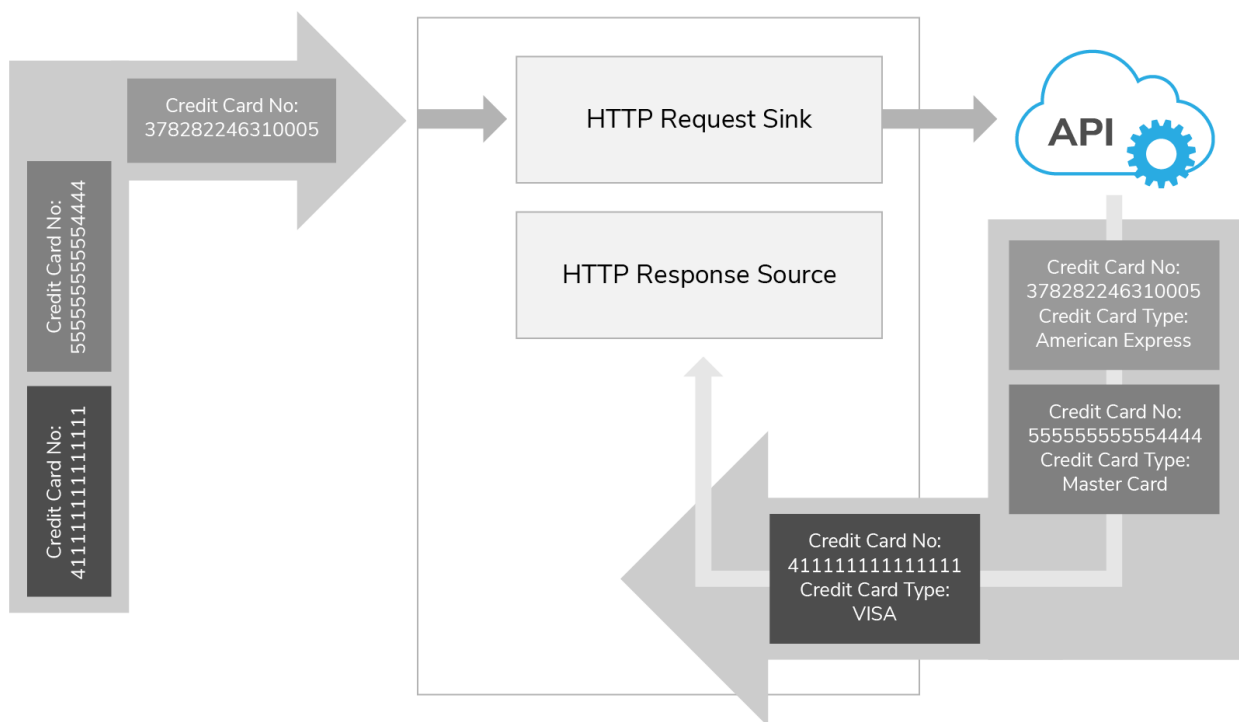
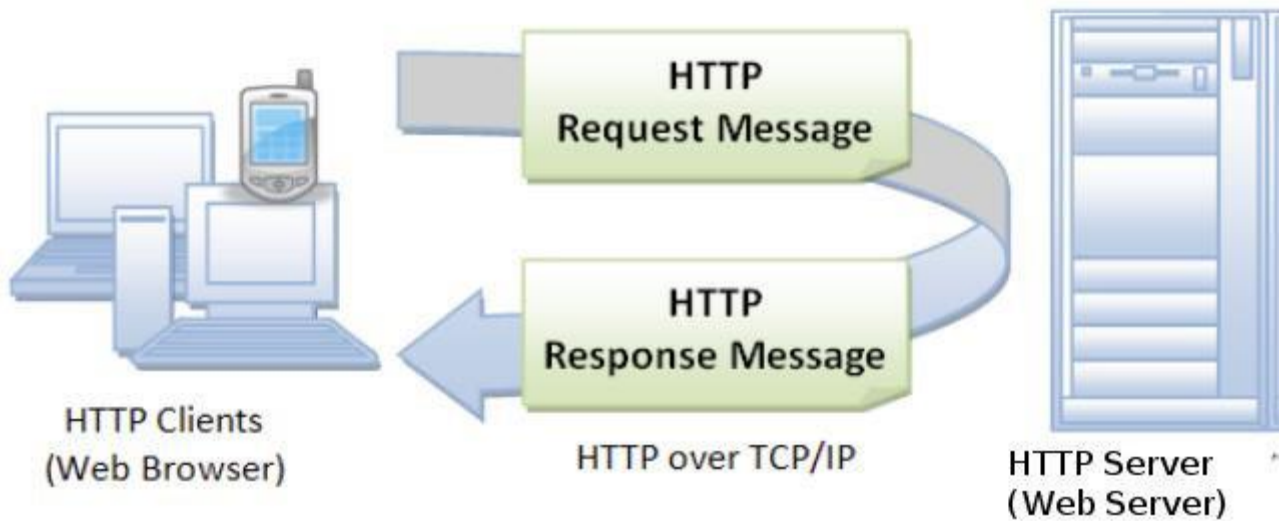
A arquitetura REST não está restrito ao HTTP, se adequa muito bem na arquitetura WEB.

A WEB é construída com base em 4 conceitos: Resource, URI, Representation e Media Type.

Arquitetura WEB



Essa arquitetura se encaixa no estilo REST.



Métodos HTTP:

GET – retorna uma informação (idempotente)

GET /produtos
GET /produtos/1

POST – Criar um novo Recurso

POST /produtos

PUT – Atualizar um recurso existente

PUT /produtos/1

DELETE – Deletar um recurso

DELETE /produtos/1

HTTP status codes

Indica o resultado da requisição HTTP

1XX – informational (Ex: 100, ...)

2XX – success (Ex: 200, 201,...)

3XX – redirection (Ex: 302, ...)

4XX - client error (Ex: 404, ..)

5XX - server error (Ex: 500, ...)

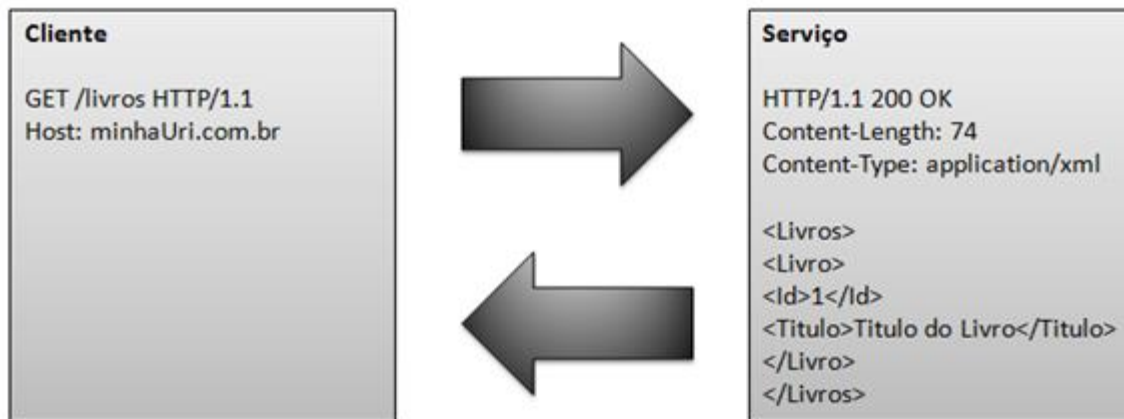


Figura 2 - Listagem de todos os livros da biblioteca

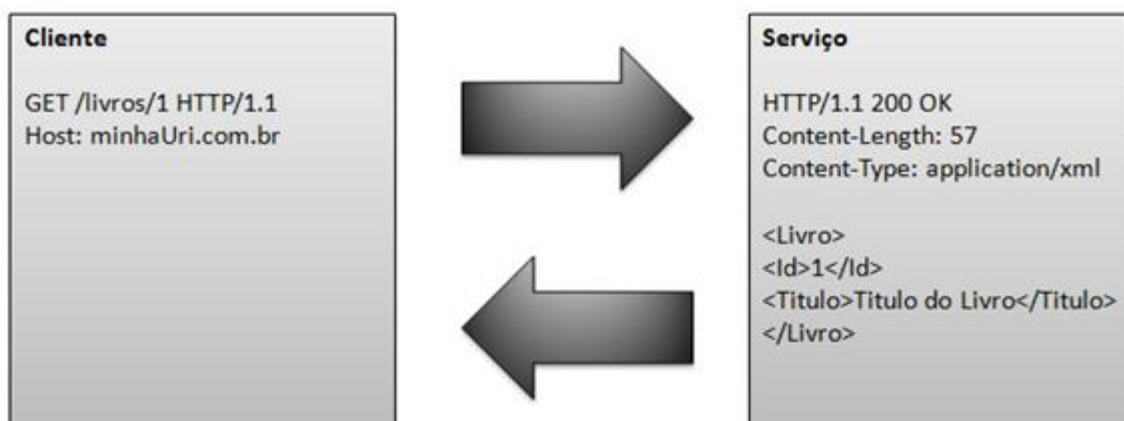


Figura 3 – Buscando um livro específico

HTTP GET – Obter informações de um recurso.

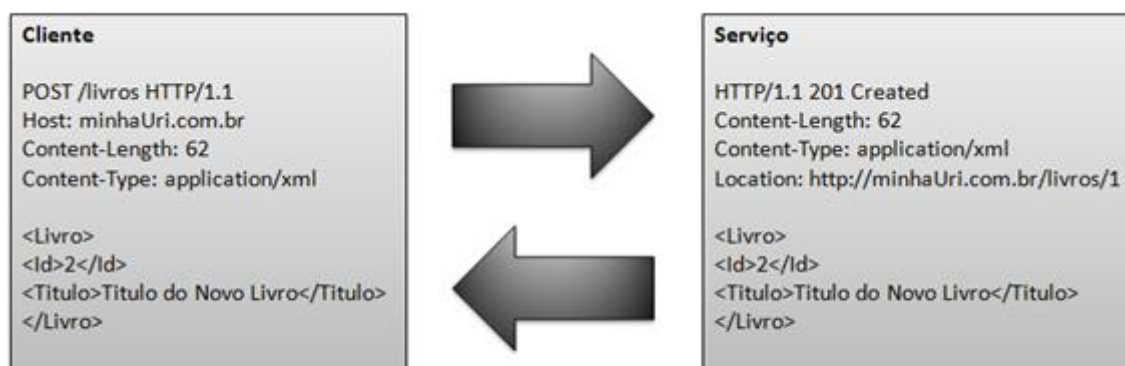


Figura 4- Criação de um livro através do verbo *POST*

HTTP POST – Criar um recurso.

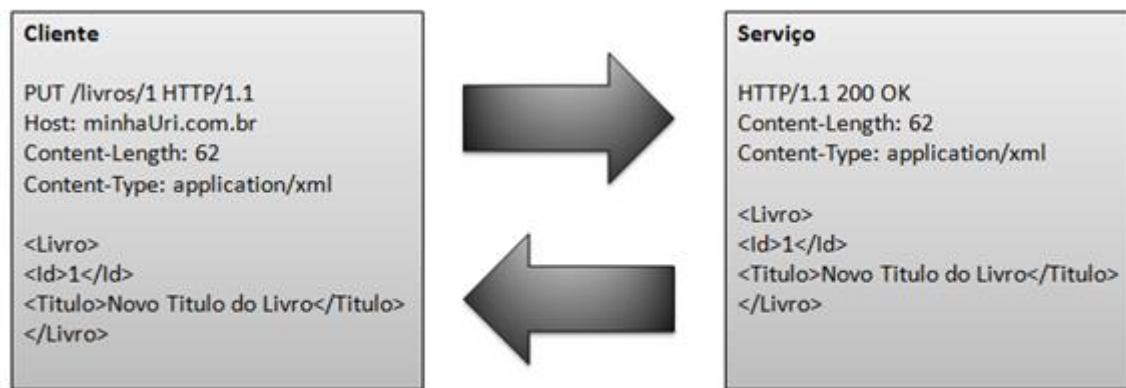


Figura 5 – Alteração de um livro existente

HTTP PUT – alterar um recurso.

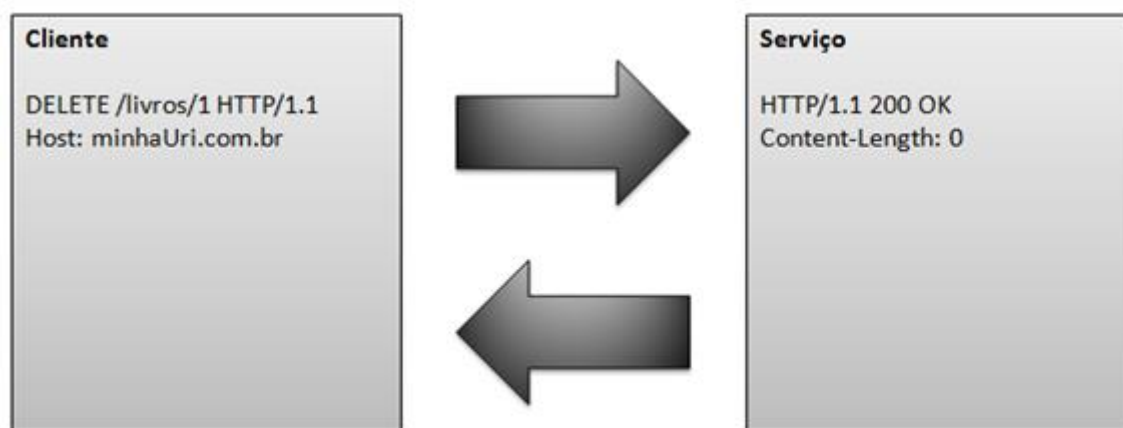


Figura 6 – Exclusão de um livro com o verbo DELETE

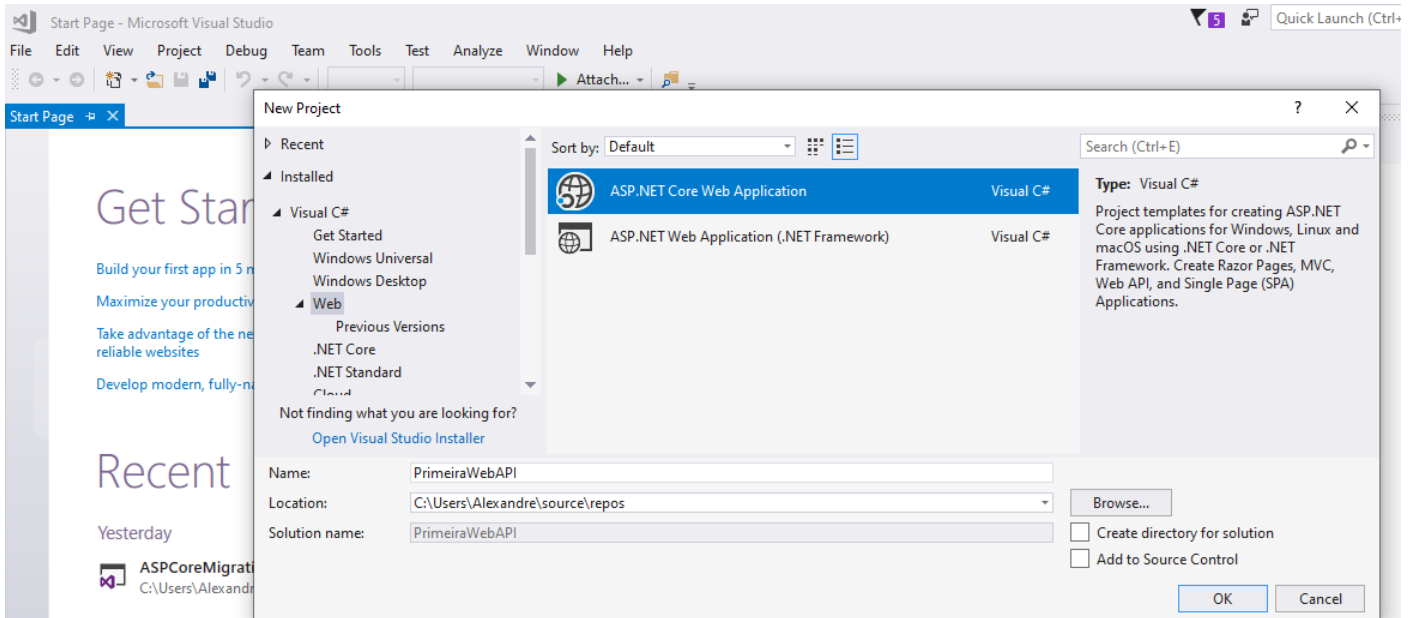
HTTP DELETE – Excluir um recurso

A WEB API pode ser vista como um conjunto de serviços expostos via HTTP, com o objetivo de integrar a aplicação que você está criando para diversos tipos de clientes, ou seja, ela pode ser consumida por navegadores, tablets, smartphones.

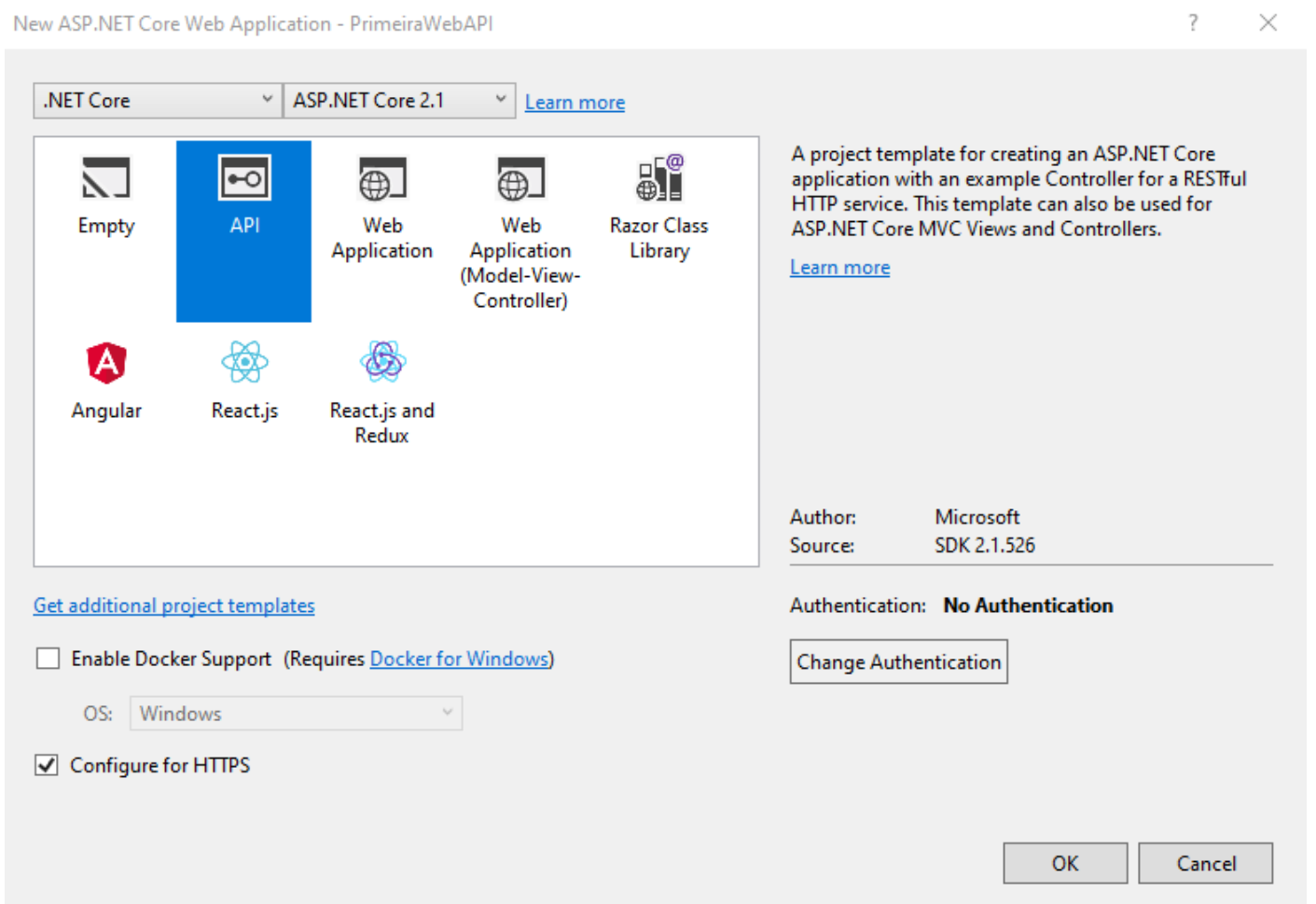
Esses serviços são listados como requisições HTTP e retornam a resposta em um formato específico, geralmente no formato JSON.

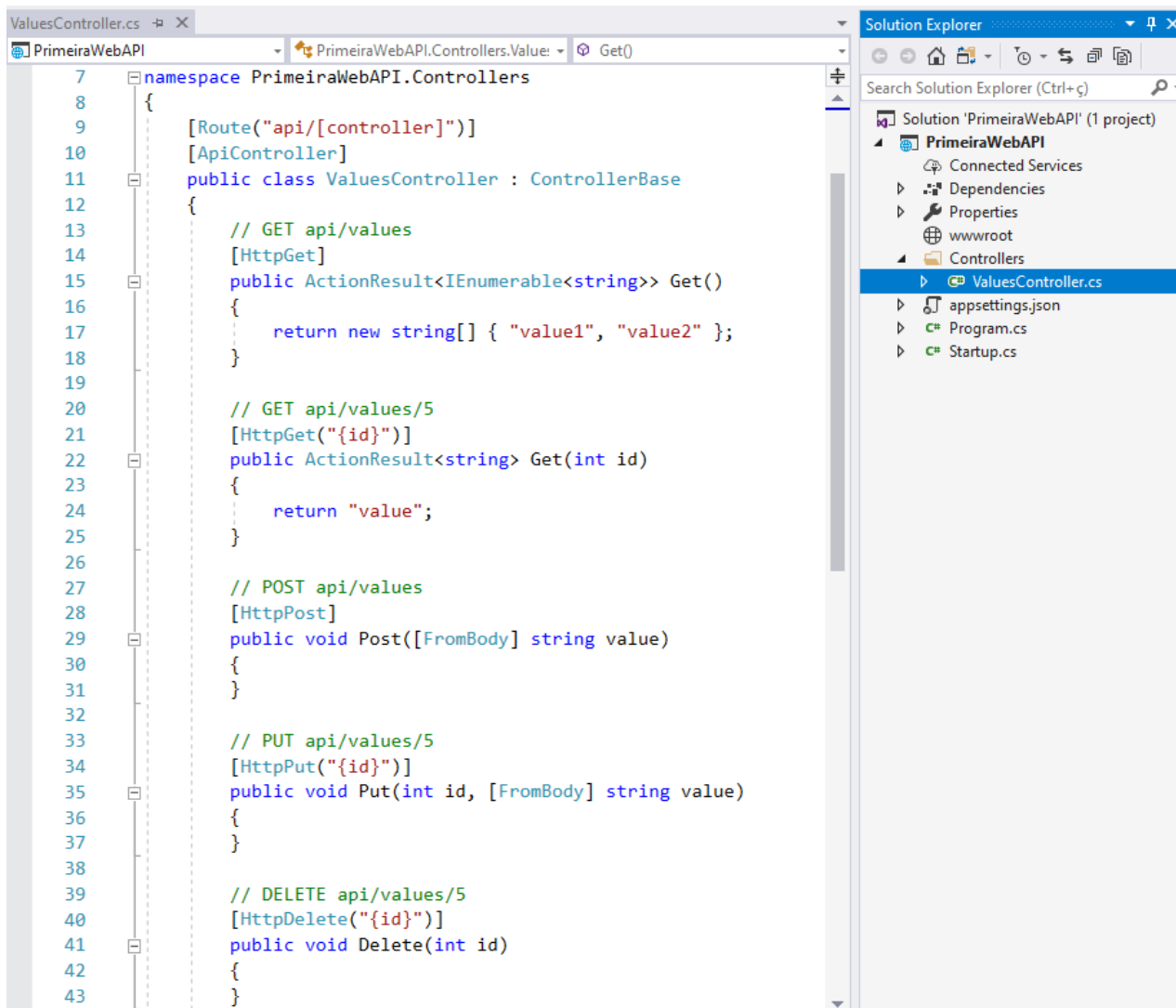
Essa resposta pode ser a informação sobre um livro, um arquivo, uma música, um texto. E o cliente que consumirá isso pode ser uma página WEB, um javascript, um CSS, uma aplicação desktop, mobile.

Vamos construir nossa aplicação WEB API



Na próxima tela escolha API





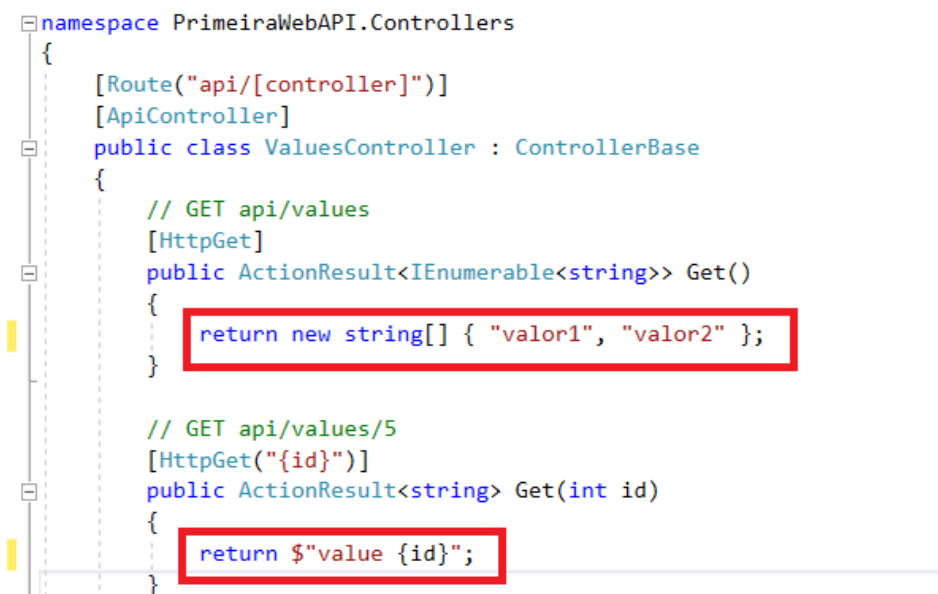
The screenshot shows the Visual Studio IDE. On the left, the 'ValuesController.cs' file is open, displaying the following code:

```
7 namespace PrimeiraWebAPI.Controllers
8 {
9     [Route("api/[controller]")]
10    [ApiController]
11    public class ValuesController : ControllerBase
12    {
13        // GET api/values
14        [HttpGet]
15        public ActionResult<IEnumerable<string>> Get()
16        {
17            return new string[] { "value1", "value2" };
18        }
19
20        // GET api/values/5
21        [HttpGet("{id}")]
22        public ActionResult<string> Get(int id)
23        {
24            return "value";
25        }
26
27        // POST api/values
28        [HttpPost]
29        public void Post([FromBody] string value)
30        {
31        }
32
33        // PUT api/values/5
34        [HttpPut("{id}")]
35        public void Put(int id, [FromBody] string value)
36        {
37        }
38
39        // DELETE api/values/5
40        [HttpDelete("{id}")]
41        public void Delete(int id)
42        {
43        }
44    }
```

On the right, the 'Solution Explorer' shows the project structure for 'PrimeiraWebAPI' (1 project). The 'Controllers' folder is expanded, and 'ValuesController.cs' is selected.

Abrindo o Controller **ValuesController.cs** notamos os métodos GET, POST, PUT e DELETE.

O primeiro GET retorna uma lista, o segundo retorna um valor específico, um POST que vai criar um recurso, um PUT que vai alterar um recurso e um DELETE que vai deletar um recurso. Faremos uma alteração para verificar a aplicação.

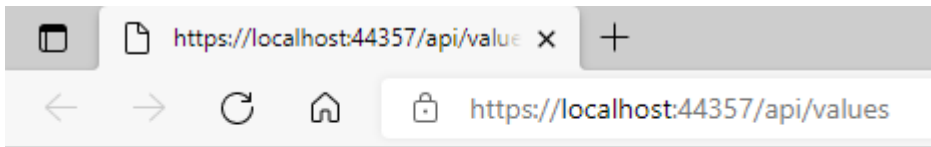


The screenshot shows the 'ValuesController.cs' file with two lines of code highlighted in red boxes:

```
namespace PrimeiraWebAPI.Controllers
{
    [Route("api/[controller]")]
    [ApiController]
    public class ValuesController : ControllerBase
    {
        // GET api/values
        [HttpGet]
        public ActionResult<IEnumerable<string>> Get()
        {
            return new string[] { "valor1", "valor2" };
        }

        // GET api/values/5
        [HttpGet("{id}")]
        public ActionResult<string> Get(int id)
        {
            return $"value {id}";
        }
    }
```

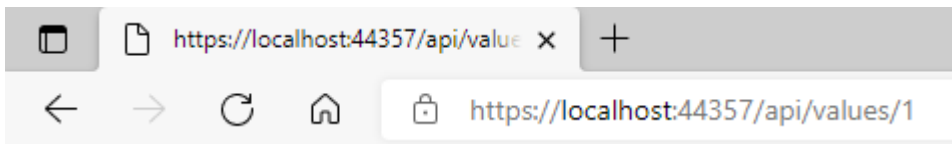
Ao rodar nossa aplicação ela já chamou a api values.



```
["valor1","valor2"]
```

E já deu um GET e retornou valor1 e valor2.

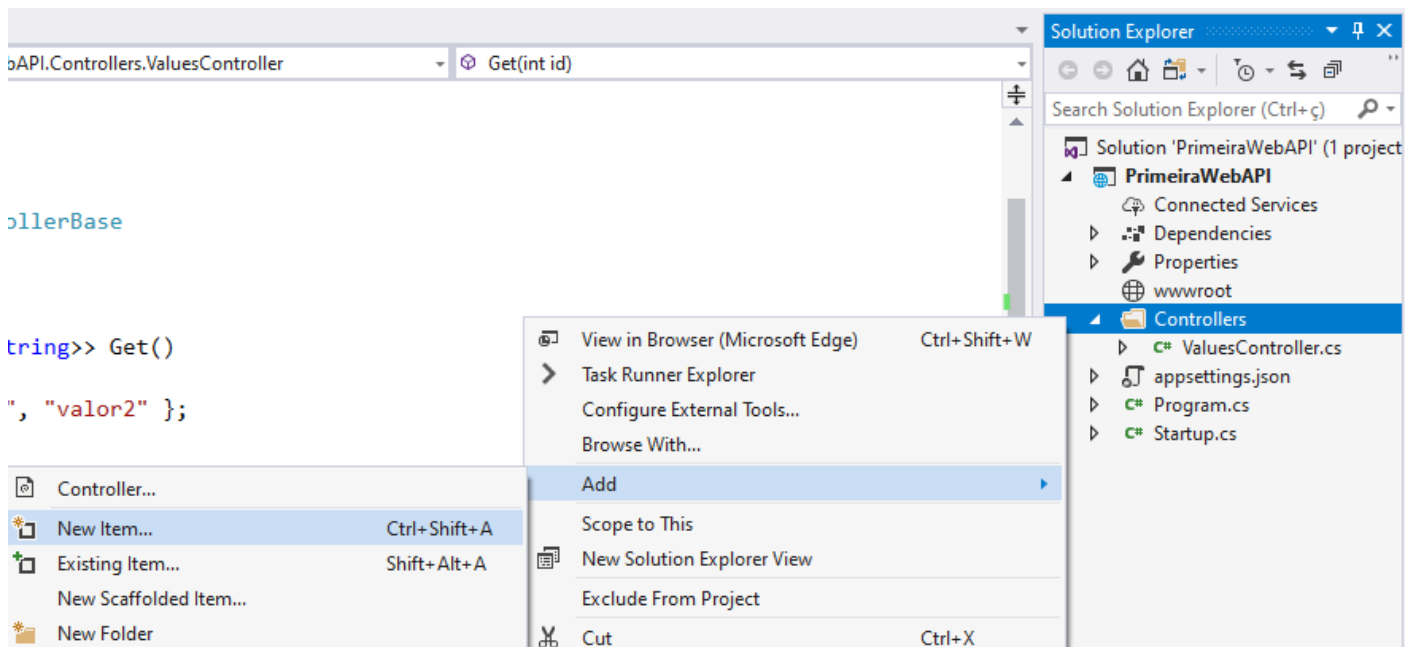
Posso passar um ID específico e me retorna apenas um valor.

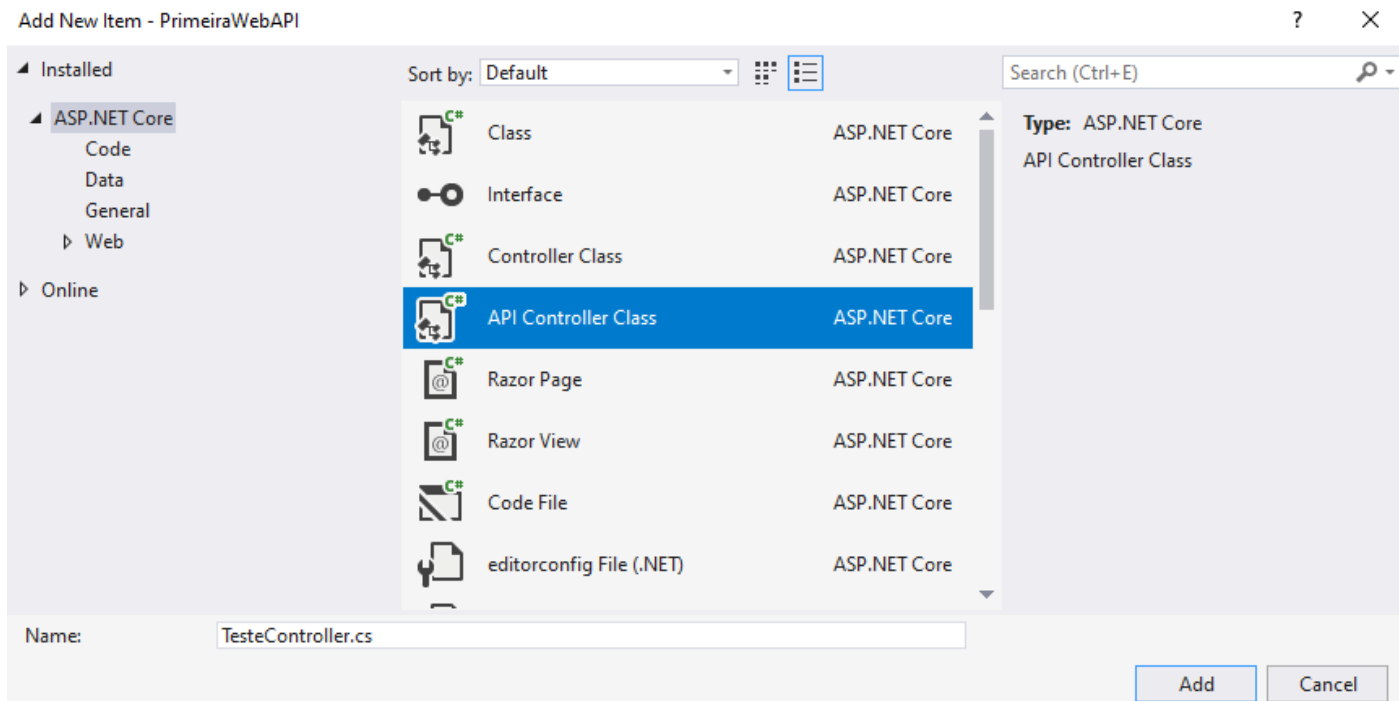


```
value 1
```

Nesse caso o mapeamento está funcionando.

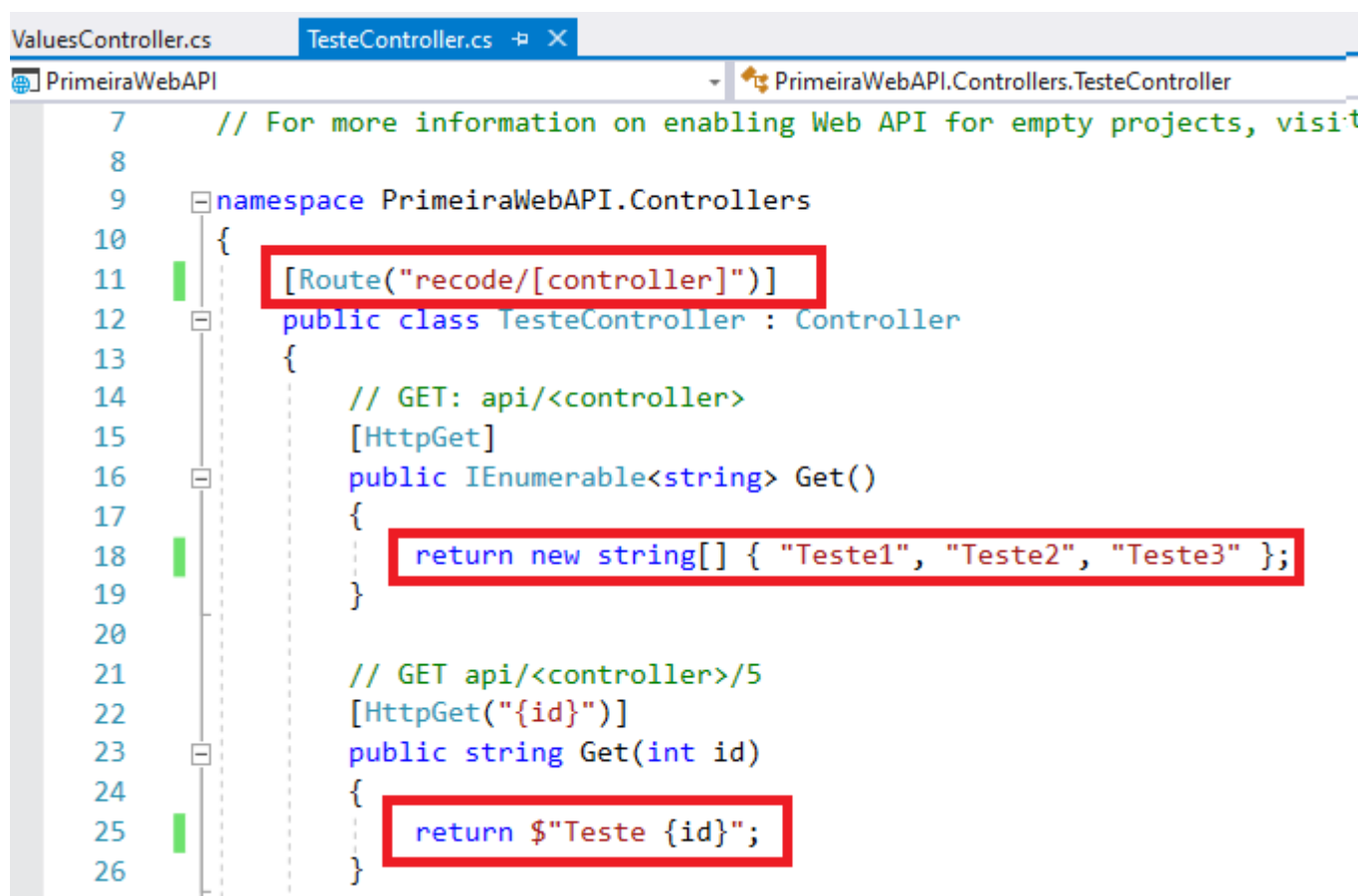
Vamos adicionar um novo controlador.





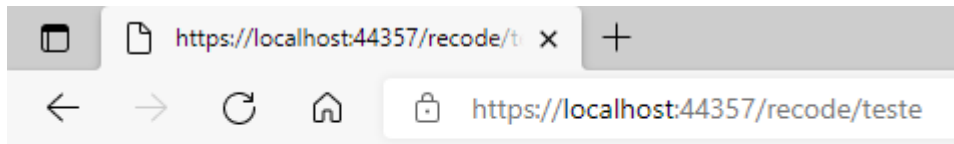
Teste é o nome da minha API e Controller é o sufixo de todo controlador.

Farei apenas uma pequena alteração para verificar se a nossa aplicação esta funcionando.



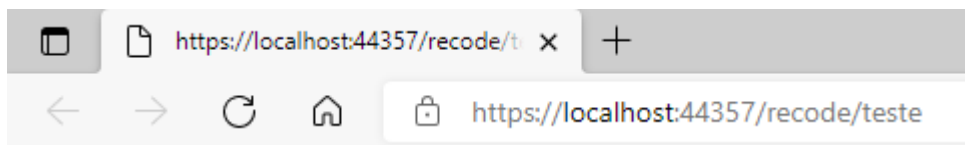
`return $"Teste {id}";` Esse recurso se chama interpolação de strings.

Vamos rodar a aplicação e mudar a url.



`["Teste1", "Teste2", "Teste3"]`

Para aparecer primeiro a outra api, faremos a seguinte alteração:



`["Teste1", "Teste2", "Teste3"]`