



A <PROGRAMAÇÃO> PRECISA DE TI

REST e API's WEB

Laboratório Web

REST: AN ARCHITECTURAL STYLE FOR BUILDING APIs.

Stands for 'Representational State Transfer'. We decide that HTTP verbs and URLs mean something.

REST

- É um estilo arquitetônico para sistemas distribuídos de hipermídia (Internet), não é um protocolo ou um standard
- Define como é que um sistema distribuído como a Internet, deve comportar-se
- Aqui um componente é um recurso, e esse recurso é acessado por uma interface comum utilizando métodos HTTP
- O REST serviu como base para o desenvolvimento do HTTP 1.1

REST

- Uma API REST utiliza URIs (**Uniform Resource Identifiers**) para endereçar recursos
- Um URI RESTful deve referir-se a um recurso como um objeto (substantivo) em vez de referir-se a uma ação (verbo)

```
http://api.example.com/device-management/managed-devices/{device-id}  
http://api.example.com/user-management/users/{id}  
http://api.example.com/user-management/users/admin
```


Quando é que uma API é considerada RESTful?

1. Interface uniforme, simplificando a arquitetura do sistema, melhorando a visibilidade das interações
2. Arquitetura cliente-servidor composta de clientes, servidores e recursos, com solicitações geridas por HTTP
3. Comunicação cliente-servidor sem estado, o que significa que o estado do cliente não é armazenado entre as solicitações

Quando é que uma API é considerada RESTful?

3. Dados armazenáveis em cache que otimizam as interações cliente-servidor
4. Um sistema em camadas que organiza e define o comportamento dos componentes
5. Code-on-demand (opcional): a capacidade de enviar código executável do servidor para o cliente quando solicitado, estendendo a funcionalidade do cliente

API: A SET OF TOOLS FOR BUILDING A SOFTWARE APPLICATION.

Stands for 'Application Programming Interface'. On the web the tools are usually made available via a set of URLs which accept and send only data via HTTP and TCP/IP.

API & WEB APIs

- Serviços RESTful são também conhecidos como web APIs
- Uma web API é uma interface de programação de aplicações para a internet
- Esta web API normalmente é composta por vários **endpoints** públicos num sistema de mensagens pedido-resposta
- Estas respostas são expressas em JSON ou XML, através de um servidor web HTTP

Routing

- Refere-se à forma como a aplicação responde a um pedido do cliente a um *endpoint* específico
- É responsável por fazer o *match* dos pedidos HTTP e encaminhar esses pedidos para os *endpoints* da aplicação
- Esse encaminhamento é efetuado de acordo com o *route path* e o método HTTP
- O *route path* pode ser uma string, um padrão de strings ou expressões regulares

Route parameters

- São segmentos do URL que são usados para capturar os valores especificados pela sua posição no URL



< > ↻ ☰  localhost:8081/user/1

```
{"name":"mahesh","password":"password1","profession":"teacher","id":1}
```

Query parameters

- São segmentos do URL que são usados para atribuir valores a parâmetros
- Esses parâmetros tem que ser designados e atribuido um valor, **name=David**
- Inicia-se com ? e separa-se com &

```
http://localhost:3000/search?name=David&age=27
```

ENDPOINT: ONE URL IN A WEB API.

Sometimes that endpoint (URL) does multiple thing by making choices based on the HTTP request headers.

Endpoints

- É um URI ou caminho, com um método HTTP específico (GET, POST, PUT, DELETE, PATCH)
- É algo que pode ser selecionado, ao fazermos *match* do URL e do método HTTP
- É executado ao correremos o método associado a esse pedido para devolver uma resposta
- Normalmente os dados são enviados em formato JSON

JSON:

**“JAVASCRIPT OBJECT NOTATION”
– A STANDARD FOR STRUCTURING
DATA THAT IS INSPIRED BY
JAVASCRIPT OBJECT LITERALS**

Javascript engines are built to understand it.

```
{  
  "firstname": "John",  
  "lastname": "Doe",  
  "address": {  
    "street": "101 Main St.",  
    "city": "New York",  
    "state": "NY"  
  }  
}
```

Porquê utilizar JSON?

- Como o formato JSON é apenas texto, pode ser facilmente enviado de e para um servidor e usado como formato de dados por qualquer linguagem de programação
- .NET possui funcionalidades para serializar e desserializar JSON
 - Serializar: converter um objeto numa string em notação JSON
 - Desserializar: converter uma string em notação JSON num objeto
- Portanto, se recebermos dados no formato JSON, podemos usá-los como qualquer outro objeto C#

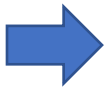
Exemplo JSON

- Um objeto JSON é representado por { }
- Uma propriedade JSON é representada por "NOME" : "VALOR"
- Este objeto contém uma propriedade **PersonList** que é um array
- Dentro desse array temos outros objetos JSON com outras propriedades

```
{
  "PersonList": [
    {
      "FirstName": "Pedro",
      "LastName": "Lopes",
      "Profession": "Teacher",
      "Age": 22,
      "Id": 1
    },
    {
      "FirstName": "Ana",
      "LastName": "Matos",
      "Profession": "Driver",
      "Age": 28,
      "Id": 2
    },
    {
      "FirstName": "Marco",
      "LastName": "Gouda",
      "Profession": "Student",
      "Age": 17,
      "Id": 3
    }
  ]
}
```

Desserializar JSON

```
{
  "PersonList": [
    {
      "FirstName": "Pedro",
      "LastName": "Lopes",
      "Profession": "Teacher",
      "Age": 22,
      "Id": 1
    },
    {
      "FirstName": "Ana",
      "LastName": "Matos",
      "Profession": "Driver",
      "Age": 28,
      "Id": 2
    },
    {
      "FirstName": "Marco",
      "LastName": "Gouda",
      "Profession": "Student",
      "Age": 17,
      "Id": 3
    }
  ]
}
```



```
public class People
{
  0 references
  public List<Person>? PersonList { get; set; }
}
```



```
public class Person
{
  0 references
  public string FirstName { get; set; }
  0 references
  public string LastName { get; set; }

  0 references
  public string Profession { get; set; }

  0 references
  public int Age { get; set; }

  0 references
  public int Id { get; set; }
}
```

Desserializar JSON

```
var jsonData = File.ReadAllText("data.json");  
People people = JsonSerializer.Deserialize<People>(jsonData);
```

- Neste caso em específico estamos a ler o JSON de um ficheiro
- Utilizamos a classe **JsonSerializer** e o método estático **Deserialize** para desserializar a string para um objeto
- **<People>** indica qual a classe/tipo que será desserializado

Serializar JSON

```
Person person = new Person()  
{  
    Id = 1,  
    Age = 20,  
    FirstName = "Pedro",  
    LastName = "Lopes",  
    Profession = "Teacher"  
};
```



```
People people = new People();  
people.PersonList.Add(person);
```



```
string json = JsonSerializer.Serialize<People>(people);
```

- Criamos uma nova instância da classe Person
- Criamos uma nova instância da classe People
- Serializamos a instância da classe People numa string JSON



```
{"PersonList":  
  [{"FirstName":"Pedro","LastName":"Lopes","Profession":"Teacher",  
    "Age":20,"Id":1}]}
```


Aplicação WEB em ASP.NET Core

```
1  var builder = WebApplication.CreateBuilder(args);
2
3  var app = builder.Build();
4
5  // Configure the HTTP request pipeline.
6  if (!app.Environment.IsDevelopment())
7  {
8      app.UseExceptionHandler("/Error");
9      app.UseHsts();
10 }
11
12 app.UseHttpsRedirection();
13
14 app.MapGet("/", () => "Hello World!");
15
16 app.Run();
17
```

1. Criação da aplicação WEB com configurações pré-definidas
2. Compilação da aplicação
3. Configuração do ambiente
4. Aplicação de *middleware* para redirecionamento de pedidos HTTP
5. Implementação do endpoint *root*
6. Execução da aplicação

Métodos HTTP

- **GET** - Providencia apenas acesso de leitura a um ou mais recursos
- **POST** - Utilizado para criar um novo recurso
- **DELETE** - Utilizado para apagar um recurso
- **PUT** - Utilizado para atualizar um recurso

GET

Método GET

Caminho/rota do endpoint

```
app.MapGet("/people", () =>
{
    return Results.Ok(people);
});
```

Callback que será invocado

GET { parameter }

Método GET

Caminho/rota do endpoint com o parâmetro

Callback que será invocado e o parâmetro passado como argumento

```
app.MapGet("/people/{id}", (int id) => {  
    :  
})
```


POST - BODY

Método POST

Caminho/rota do endpoint

```
app.MapPost("/people", (Person person) => {
```

Callback que será invocado com a nova pessoa a ser criada (enviada) pelo BODY

DELETE { parameter }

Método DELETE

Caminho/rota do endpoint com o parâmetro

Callback que será invocado e o parâmetro passado como argumento

```
app.MapDelete("/people/{id}", (int id) =>  
{
```

PUT {parameter} & BODY

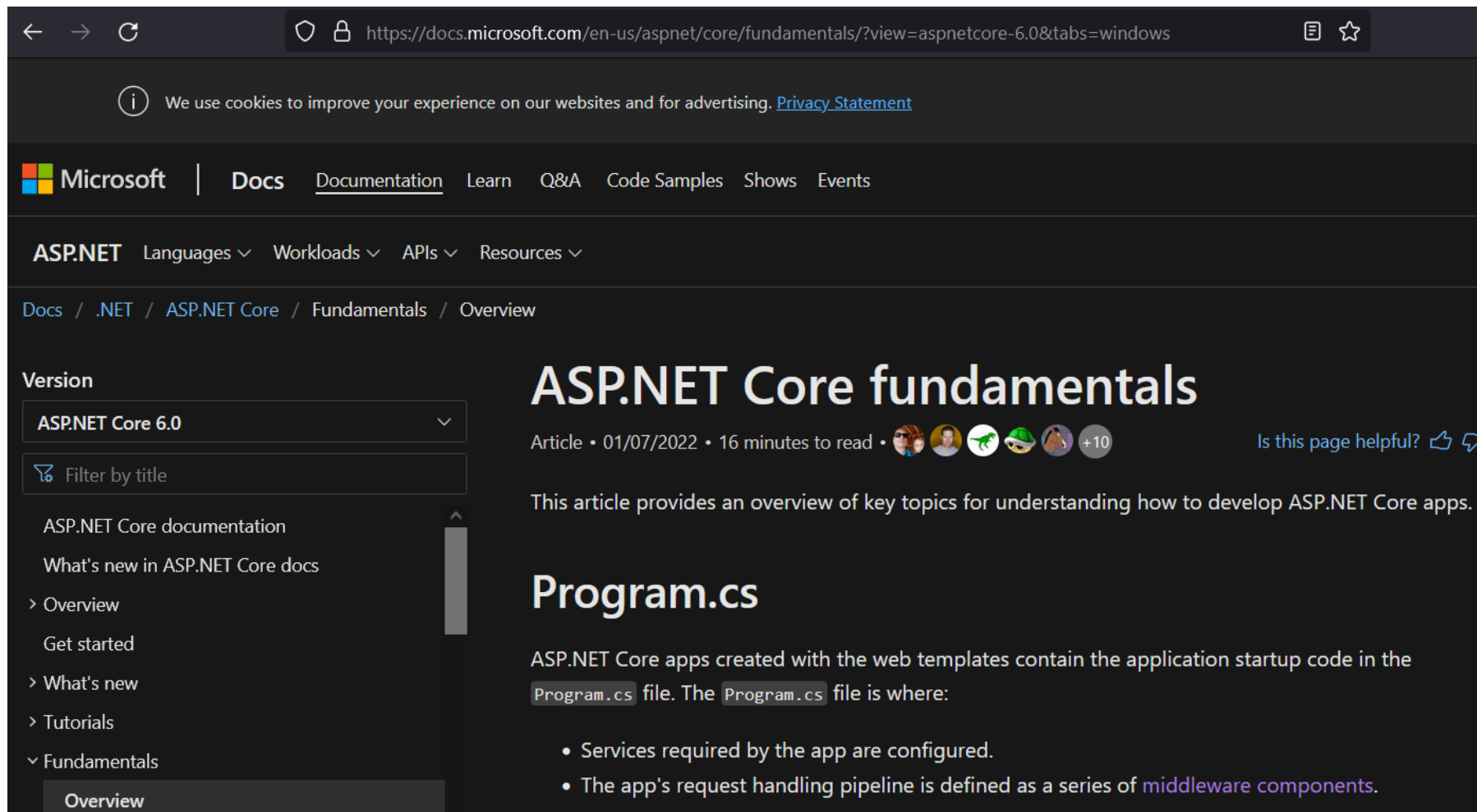
Método PUT

Caminho/rota do endpoint

Callback que será invocado
e o parâmetro passado
como argumento com os
novos dados (enviados) pelo BODY

```
app.MapPut("/people/{id}", (int id, Person inputPerson) => {  
    var person = people.PersonList.Find(p => p.Id == id);  
})
```

Referências



The screenshot shows the Microsoft Docs website for ASP.NET Core fundamentals. The browser address bar displays the URL: <https://docs.microsoft.com/en-us/aspnet/core/fundamentals/?view=aspnetcore-6.0&tabs=windows>. A cookie notice is visible at the top. The navigation bar includes the Microsoft logo and links to Docs, Documentation, Learn, Q&A, Code Samples, Shows, and Events. Below this, the ASP.NET section is expanded, showing links for Languages, Workloads, APIs, and Resources. The breadcrumb trail indicates the path: Docs / .NET / ASP.NET Core / Fundamentals / Overview. On the left, a sidebar for the 'Version' dropdown is set to 'ASP.NET Core 6.0', and a search filter 'Filter by title' is present. The main content area features the title 'ASP.NET Core fundamentals' with a sub-header 'Article • 01/07/2022 • 16 minutes to read'. A 'Is this page helpful?' feedback prompt is also visible. The introductory text states: 'This article provides an overview of key topics for understanding how to develop ASP.NET Core apps.' Below this, the section 'Program.cs' is introduced, explaining that ASP.NET Core apps created with web templates contain application startup code in the `Program.cs` file. A list of key points follows:

- Services required by the app are configured.
- The app's request handling pipeline is defined as a series of [middleware components](#).

qualificar

TAL

X

</>

A <PROGRAMAÇÃO