



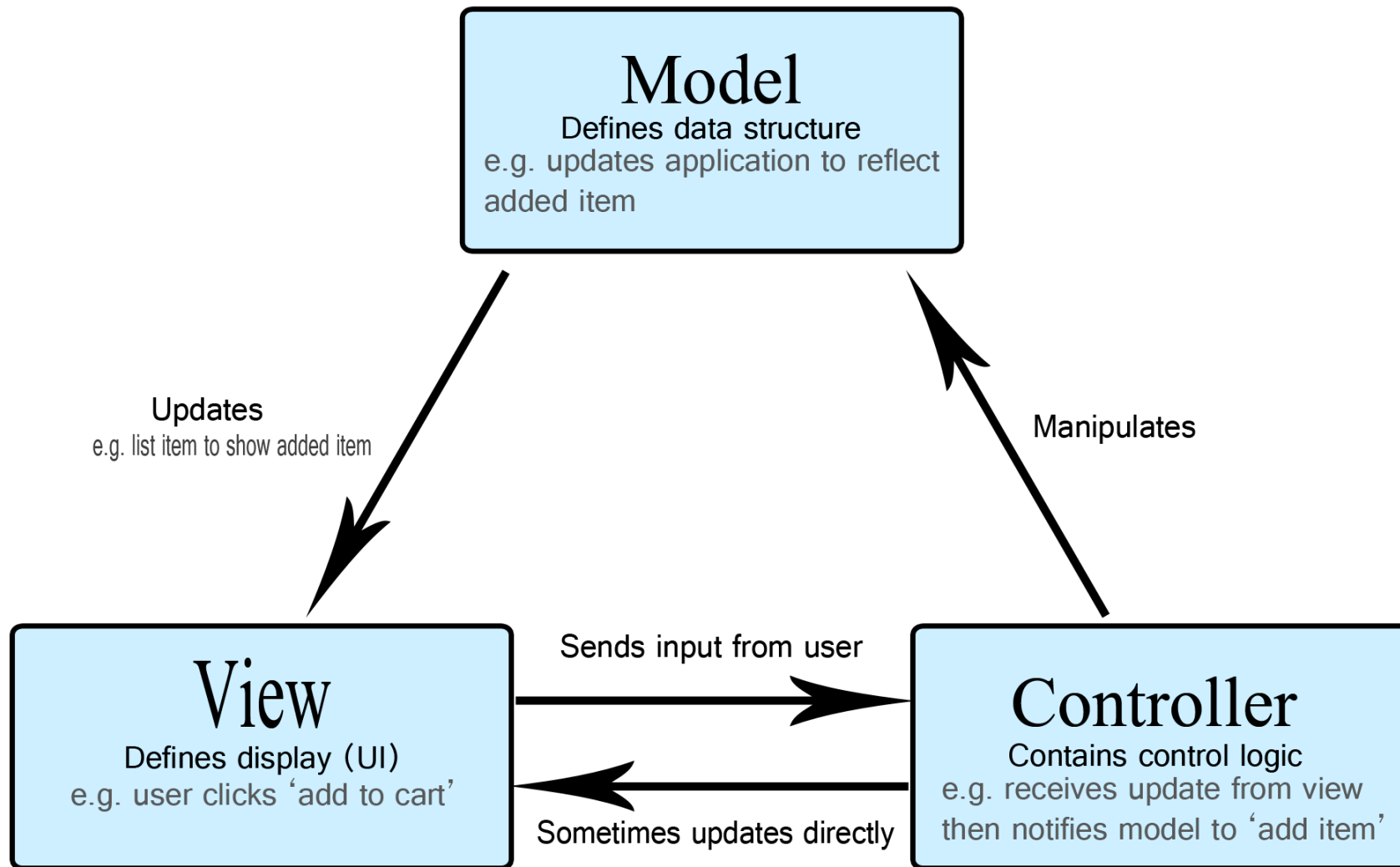
A <PROGRAMAÇÃO> PRECISA DE TI

# ASP.NET Core MVC

Laboratório Web

# MVC - Model View Controller

- É um padrão em desenho de software utilizado para implementar interfaces, representação de dados e lógica de control
- Enfatiza a separação entre a lógica de negócio e a visualização
- Model: Representa os dados e a lógica de negócio
- View: Responsável pela visualização dos dados
- Controller: Encaminha comandos entre o model e a view



# Model

- É uma classe que representa uma tabela na BD
- Permite criar instâncias manipuláveis através de dados e operações efetuadas na BD
- Podemos definir regras de validação

```
public class Person
{
    public int PersonId { get; set; }

    [Required]
    [MinLength(2)]
    public string Name { get; set; }

    [Phone]
    public string PhoneNumber { get; set; }

    [EmailAddress]
    public string Email { get; set; }
}
```

# View

- Em .NET utilizamos Razor para criar as nossas páginas HTML
- As Views serão responsáveis por devolver o conteúdo necessário para desenhar as páginas HTML
- Temos acesso aos modelos para preencher as views

```
<table class="table">
  <thead>
    <tr>
      <th>@Html.DisplayNameFor(model => model.Name)</th>
      <th>@Html.DisplayNameFor(model => model.PhoneNumber)</th>
      <th>@Html.DisplayNameFor(model => model.Email)</th>
    </tr>
  </thead>
  <tbody>
    @foreach (var item in Model) {
      <tr>
        <td>@Html.DisplayFor(modelItem => item.Name)</td>
        <td>@Html.DisplayFor(modelItem => item.PhoneNumber)</td>
        <td>@Html.DisplayFor(modelItem => item.Email)</td>
      </tr>
    }
  </tbody>
</table>
```

string Person.Email { get; set; }

- Email
- Equals
- GetHashCode
- GetType

# Controller

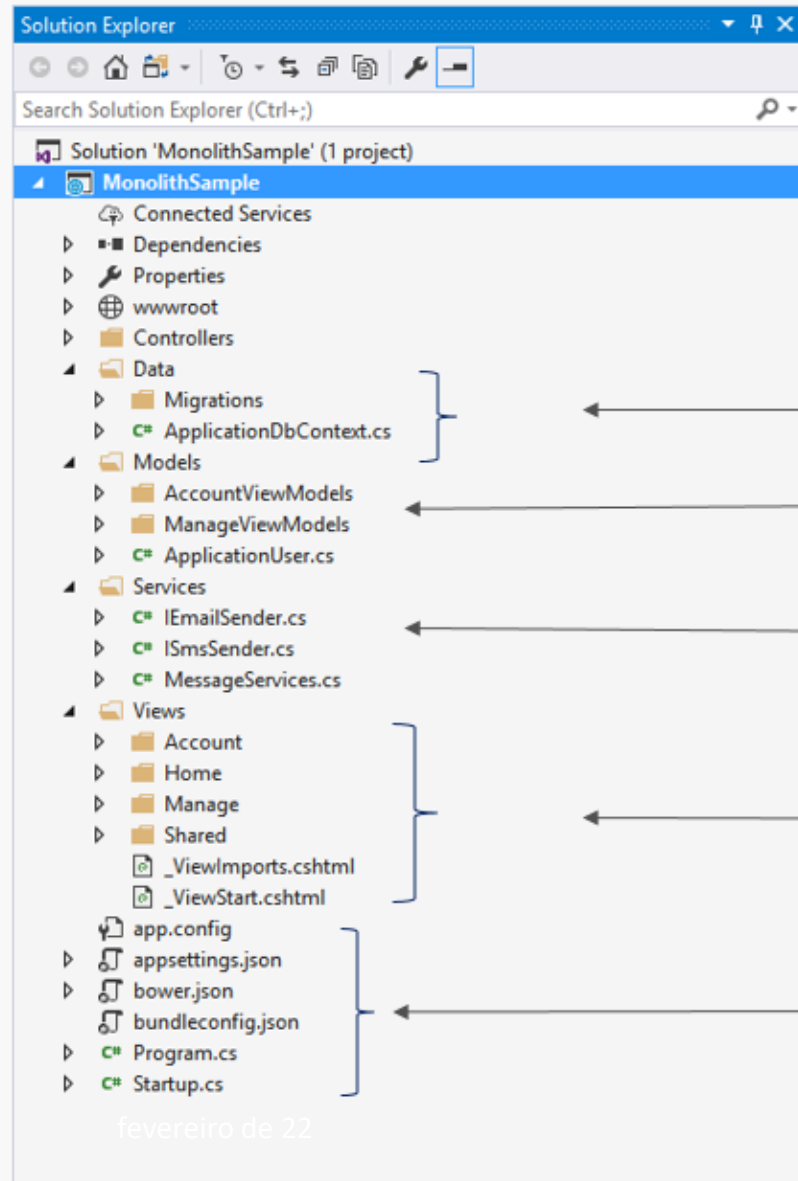
- Encaminha os pedidos para as ações dos controllers
- Podemos implementar 2 tipos de controllers:
  - APIs Controller
  - View Controller
- Os dados do pedido HTTP são associados ao método do controller

```
public class PeopleController : Controller
{
    private readonly AddressBookContext _context;

    public PeopleController(AddressBookContext context)
    {
        _context = context;
    }

    // GET: /people
    public async Task Index()
    {
        return View(await _context.People.ToListAsync());
    }
}
```

# VS Solution Structure



## Data Access Logic

- EF Migrations
- EF DbContext and model design

## UI Models

## Application Services (interfaces and implementations)

## Presentation Logic

## Application Entry Point and Configuration



# Web API com Controllers

- Cada controller disponibiliza ações (endpoints) aplicados a um recurso
- O nome da rota para esse controller é o nome do controller, por exemplo `EmployeesController` a rota será `"/employees"`
- Um controller responde a pedidos web API se for “marcado” com o atributo `[ApiController]`
- Um controller para uma **api** é uma classe que herda da classe `ControllerBase`, senão deve herdar da classe `Controller` para suportar Views



# Web API com Controllers

“Marcar” controller como ApiController

```
[ApiController]
```

```
[Route("[controller]")]
```

Definição da rota padrão  
para todos os endpoints

1 reference

```
public class EmployeesController : ControllerBase  
{  
    ...  
}
```

O nosso controller herda  
da classe ControllerBase

# Atributos dos endpoints

- Atributos são usados para decorar os endpoints e providenciar informação extra sobre:

- Método HTTP
- Rota do endpoint
- Nome do endpoint
- Tipo de dados que consome
- Tipo de dados que devolve
- Tipo de respostas que devolve

```
[HttpPost(Name = "PostEmployee")]  
[Consumes(MediaTypeNames.Application.Json)]  
[ProducesResponseType(StatusCodes.Status201Created)]  
[ProducesResponseType(StatusCodes.Status400BadRequest)]
```

# Tipos de retorno dos endpoints

- Tipo específico – apropriado quando apenas temos um resultado possível, por exemplo uma instância de uma classe criada por nós
- *ActionResult* – apropriado quando temos várias possibilidades de resultado, por exemplo **Ok** ou **NotFound** de forma síncrona
- *Task<ActionResult>* - similar ao caso anterior mas desta forma a resposta é assíncrona

# Métodos HTTP

- *[HttpDelete]*
- *[HttpGet]*
- *[HttpHead]*
- *[HttpOptions]*
- *[HttpPatch]*
- *[HttpPost]*
- *[HttpPut]*

[HttpGet]

0 references

```
public IEnumerable<Employee> Get()  
{  
    return employees.EmployeesList;  
}
```

# GET - Controller

Método GET

`[HttpGet]`

0 references

```
public IEnumerable<Employee> Get()  
{  
    return employees.EmployeesList;  
}
```

Callback que será invocado

# GET { parameter} - Controller

Método GET

Caminho/rota do endpoint com o parâmetro e o nome

```
[HttpGet("{id}", Name = "GetById")]
[ProducesResponseType(StatusCodes.Status200OK, Type = typeof(Employee))]
[ProducesResponseType(StatusCodes.Status404NotFound)]
0 references
public IActionResult Get(int id)
{
    Employee? emp = employees.EmployeesList.Find(e => e.UserId == id);

    if(emp == null)
    {
        return NotFound($"ID: {id} not found!");
    }
    else
    {
        return Ok(emp);
    }
}
```

Tipo de dados da resposta

Callback que será invocado e o parâmetro passado como argumento

# POST & BODY Request - Controller

Método POST

```
[HttpPost]
[Consumes(MediaTypeNames.Application.Json)]
[ProducesResponseType(StatusCodes.Status201Created)]
[ProducesResponseType(StatusCodes.Status400BadRequest)]
0 references
public IActionResult Post([FromBody] Employee employee)
{
```

- Tipo de dados que o endpoint recebe
- Diferentes tipos de resposta que produz
  - Sucesso
  - Erro

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



# DELETE { parameter }

Método DELETE

Tipos de resposta produzidas

```
[HttpDelete("{id}")]  
[ProducesResponseType(StatusCodes.Status200OK)]  
[ProducesResponseType(StatusCodes.Status404NotFound)]  
0 references  
public IActionResult Delete(int id)  
{  
    :  
}
```

Parâmetro passado  
como argumento

# PUT { parameter} & BODY Request - Controller

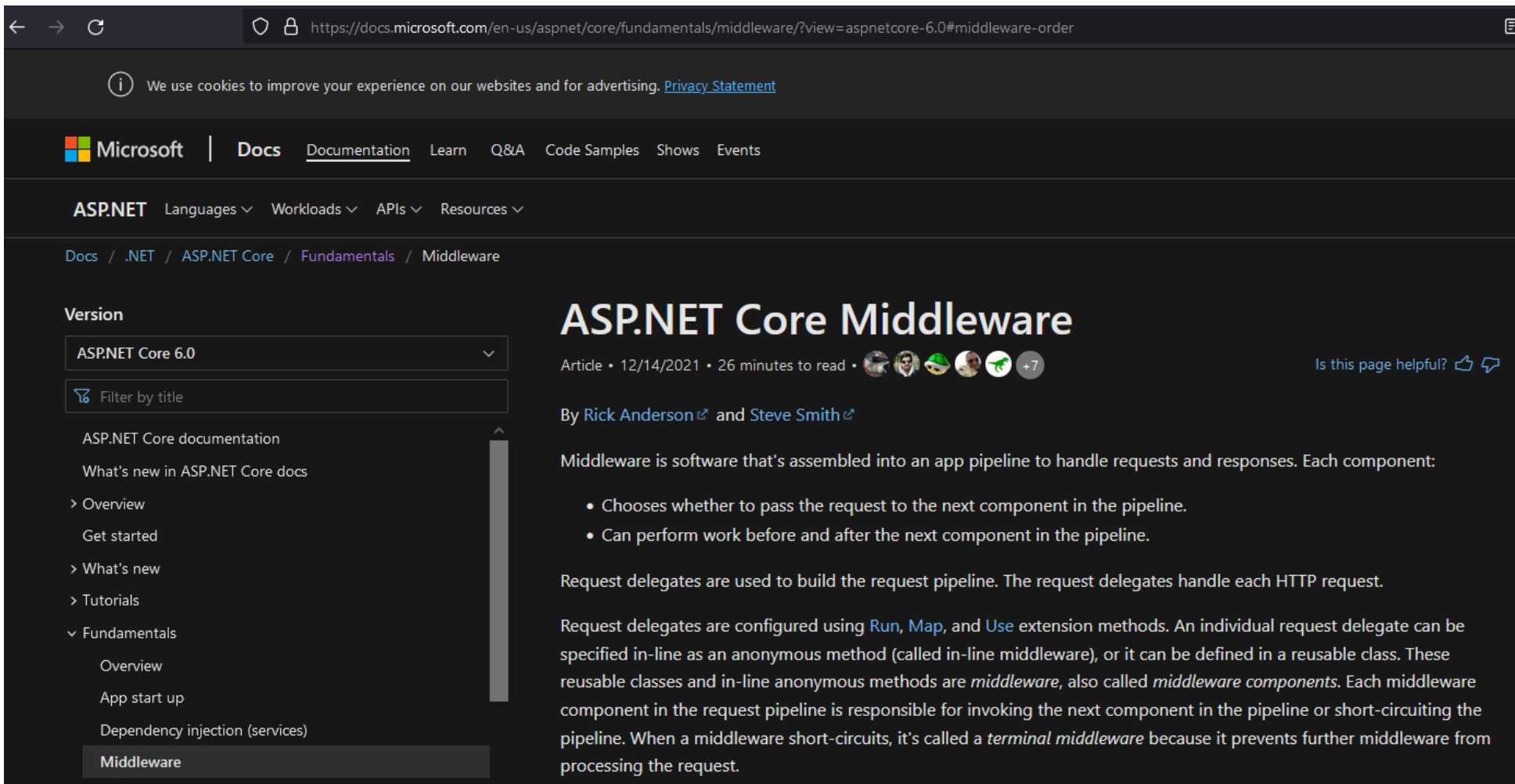
Método DELETE

Tipos de resposta produzidas

```
[HttpPut("{id}")]
[ProducesResponseType(StatusCodes.Status200OK, Type = typeof(Employee))]
[ProducesResponseType(StatusCodes.Status404NotFound)]
0 references
public IActionResult Put(int id, [FromBody] Employee employee)
{
    ...
}
```

Parâmetros passados  
na rota e no body

# Referências



The screenshot shows the Microsoft Docs website for ASP.NET Core Middleware. The browser address bar displays the URL: <https://docs.microsoft.com/en-us/aspnet/core/fundamentals/middleware/?view=aspnetcore-6.0#middleware-order>. A cookie notice is visible at the top. The navigation bar includes the Microsoft logo, 'Docs', and links to 'Documentation', 'Learn', 'Q&A', 'Code Samples', 'Shows', and 'Events'. Below this, the 'ASP.NET' section is expanded, showing 'Languages', 'Workloads', 'APIs', and 'Resources'. The breadcrumb trail reads: Docs / .NET / ASP.NET Core / Fundamentals / Middleware. On the left, a 'Version' dropdown is set to 'ASP.NET Core 6.0', and a search filter 'Filter by title' is present. The left sidebar lists the following items: 'ASP.NET Core documentation', 'What's new in ASP.NET Core docs', '> Overview', 'Get started', '> What's new', '> Tutorials', 'v Fundamentals', 'Overview', 'App start up', 'Dependency injection (services)', and 'Middleware' (which is highlighted). The main content area features the title 'ASP.NET Core Middleware', the article date '12/14/2021', a reading time of '26 minutes', and a list of authors including Rick Anderson and Steve Smith. The text explains that middleware is software assembled into an app pipeline to handle requests and responses, and lists two key capabilities: choosing whether to pass the request to the next component and performing work before and after the next component. It also mentions that request delegates are used to build the request pipeline and handle each HTTP request. The final paragraph states that request delegates are configured using `Run`, `Map`, and `Use` extension methods, and that individual request delegates can be specified in-line as anonymous methods (in-line middleware) or defined in reusable classes (middleware components). Each middleware component is responsible for invoking the next component or short-circuiting the pipeline, with terminal middleware preventing further processing.

← → ↻ <https://docs.microsoft.com/en-us/aspnet/core/fundamentals/middleware/?view=aspnetcore-6.0#middleware-order>

We use cookies to improve your experience on our websites and for advertising. [Privacy Statement](#)

Microsoft | Docs [Documentation](#) [Learn](#) [Q&A](#) [Code Samples](#) [Shows](#) [Events](#)

ASP.NET [Languages](#) ▾ [Workloads](#) ▾ [APIs](#) ▾ [Resources](#) ▾

[Docs](#) / [.NET](#) / [ASP.NET Core](#) / [Fundamentals](#) / [Middleware](#)

**Version**

ASP.NET Core 6.0 ▾

Filter by title

ASP.NET Core documentation

What's new in ASP.NET Core docs

> Overview

Get started

> What's new

> Tutorials

v Fundamentals

Overview

App start up

Dependency injection (services)

**Middleware**

## ASP.NET Core Middleware

Article • 12/14/2021 • 26 minutes to read • +7

Is this page helpful?

By [Rick Anderson](#) and [Steve Smith](#)

Middleware is software that's assembled into an app pipeline to handle requests and responses. Each component:

- Chooses whether to pass the request to the next component in the pipeline.
- Can perform work before and after the next component in the pipeline.

Request delegates are used to build the request pipeline. The request delegates handle each HTTP request.

Request delegates are configured using `Run`, `Map`, and `Use` extension methods. An individual request delegate can be specified in-line as an anonymous method (called in-line middleware), or it can be defined in a reusable class. These reusable classes and in-line anonymous methods are *middleware*, also called *middleware components*. Each middleware component in the request pipeline is responsible for invoking the next component in the pipeline or short-circuiting the pipeline. When a middleware short-circuits, it's called a *terminal middleware* because it prevents further middleware from processing the request.

# qualificar

TAL

X

</>

## A <PROGRAMAÇÃO