

Introdução a ASP .NET CORE 2

Bruno Oliveira

Tópicos

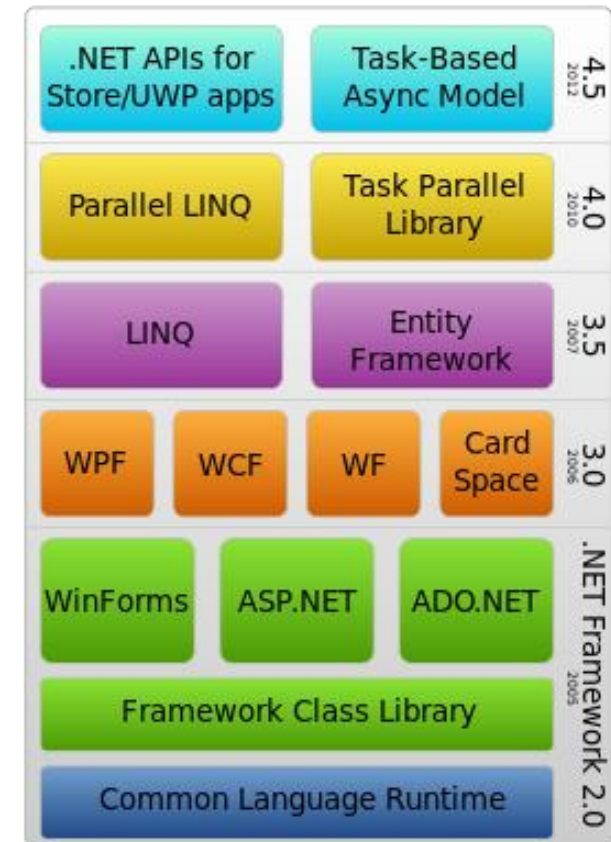
- Introdução à framework .NET;
- Desenvolvimento WEB utilizando ASP.NET Core 2.2;
 - Introdução a SQL Server;
 - Inserção, eliminação e leitura de dados;

Página GitHub do Workshop:

<https://github.com/brunobmo/ASP.NET-Training/tree/master/2019>

Framework .NET

- .NET Framework é um ambiente de execução que fornece uma [biblioteca](#) de classes abrangente;
- permite aos programadores desenvolver aplicações robustas com código confiável para todas as principais áreas de desenvolvimento;



Framework .NET Core

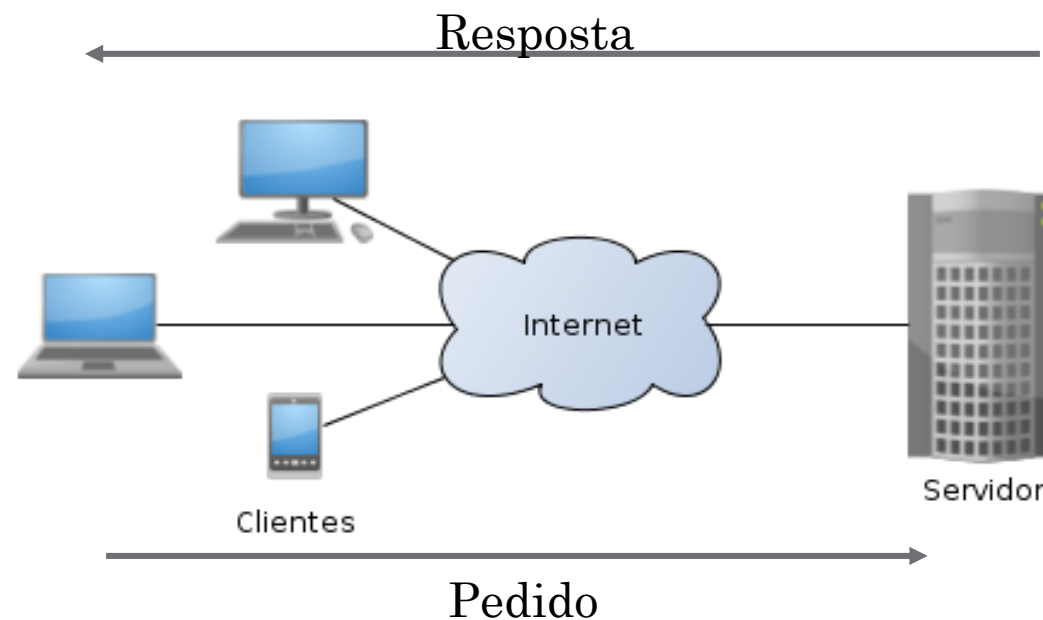
- A framework .NET Core 2.2 é versão **modular** da .NET Framework;
- Uma das características principais desta biblioteca passa pela capacidade de instalar os componentes que são **necessários** para a aplicação;
- Permite que diferentes **versões** de uma aplicação possam coexistir na mesma máquina sem problemas de compatibilidade da framework .NET;
- A ASP.NET Core foi completamente **reescrita** a partir da framework ASP.NET de forma a ser *cross-platform, open source* e **sem limitações** de **compatibilidade**;

Microsoft Web Stack

- Modelos de desenvolvimento:
 - **Web Forms (2002)**: Permite a construção de websites dinâmicos utilizando uma interface *drag-and-drop* baseado num modelo orientado a eventos a eventos; Permite a criação de aplicações de forma rápida e simples;
 - **ASP.NET MVC (2009)**: A framework **MVC** (MVC5) da Microsoft aplica o padrão MVC (Model-View-Controller) sobre o ASP.NET, permitindo o desenvolvimento de aplicações web que promovem a reutilização, organização e desempenho do código;
 - **ASP.NET Core** é diferente das versões anteriores.

Não esquecer o protocolo HTTP

- Na arquitetura **Cliente/Servidor** os **clientes** (por exemplo web browser) realizam **pedidos** de forma a requisitar recursos disponibilizados por **servidores web**;
- A comunicação cliente/servidor é realizada através do **protocolo** de aplicação: HTTP (*HyperText Transport Protocol*);
- Para dois computadores comunicarem é necessário que ambos conheçam o protocolo em termos de **sintaxe**, **semântica** e o **timing**.



Caso de estudo

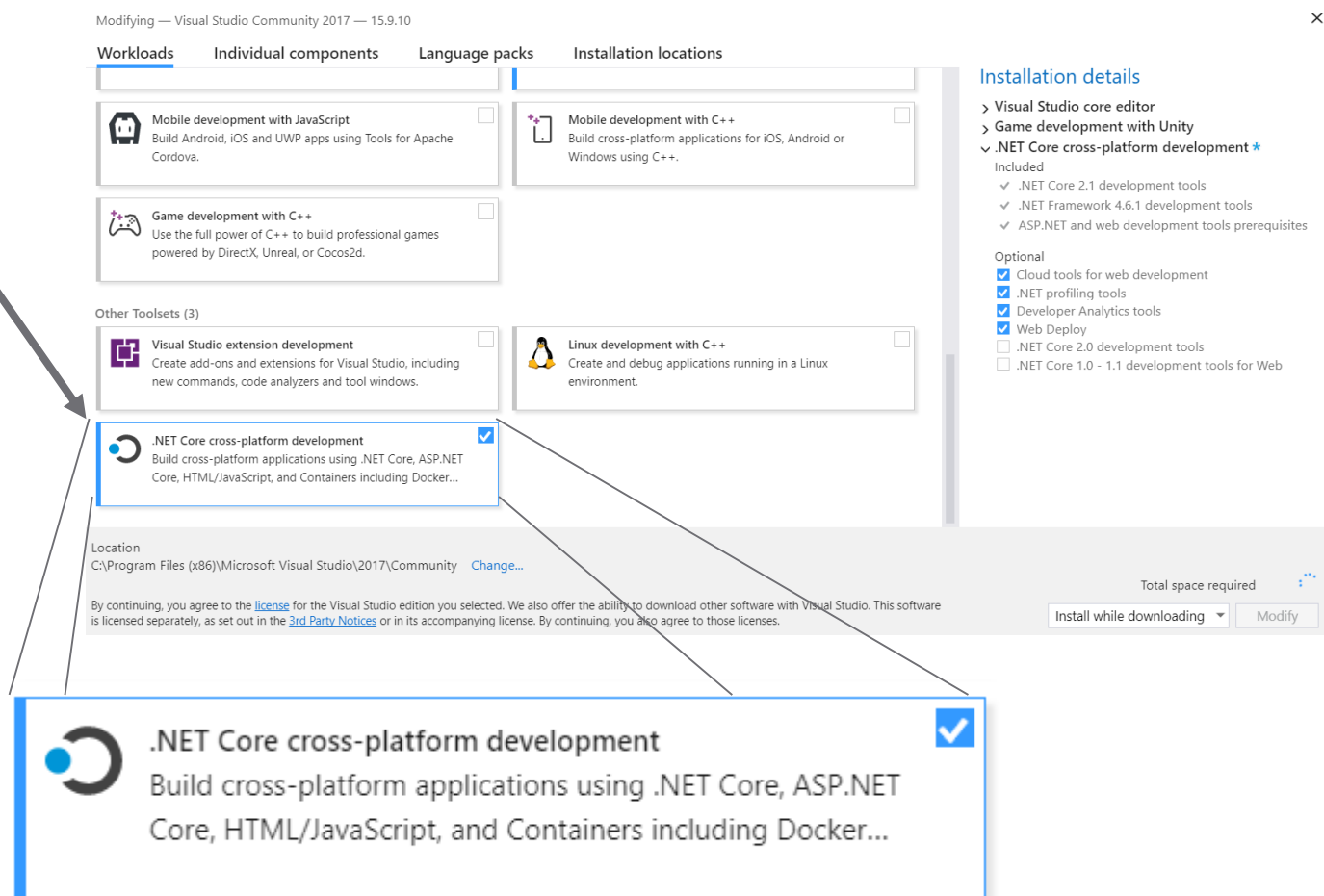
- Desenvolver uma pequena aplicação para gestão de tarefas;
- Para cada **utilizador** da aplicação é necessário armazenar:
 - **Código** de utilizador
 - **Nome** de utilizador
 - **Email**;
- O utilizador regista um conjunto de **tarefas**, sendo necessário armazenar:
 - **Identificador** da tarefa;
 - **Título** da tarefa
 - **Descrição** da tarefa;
 - **Data** da tarefa;

Ferramentas e tecnologias

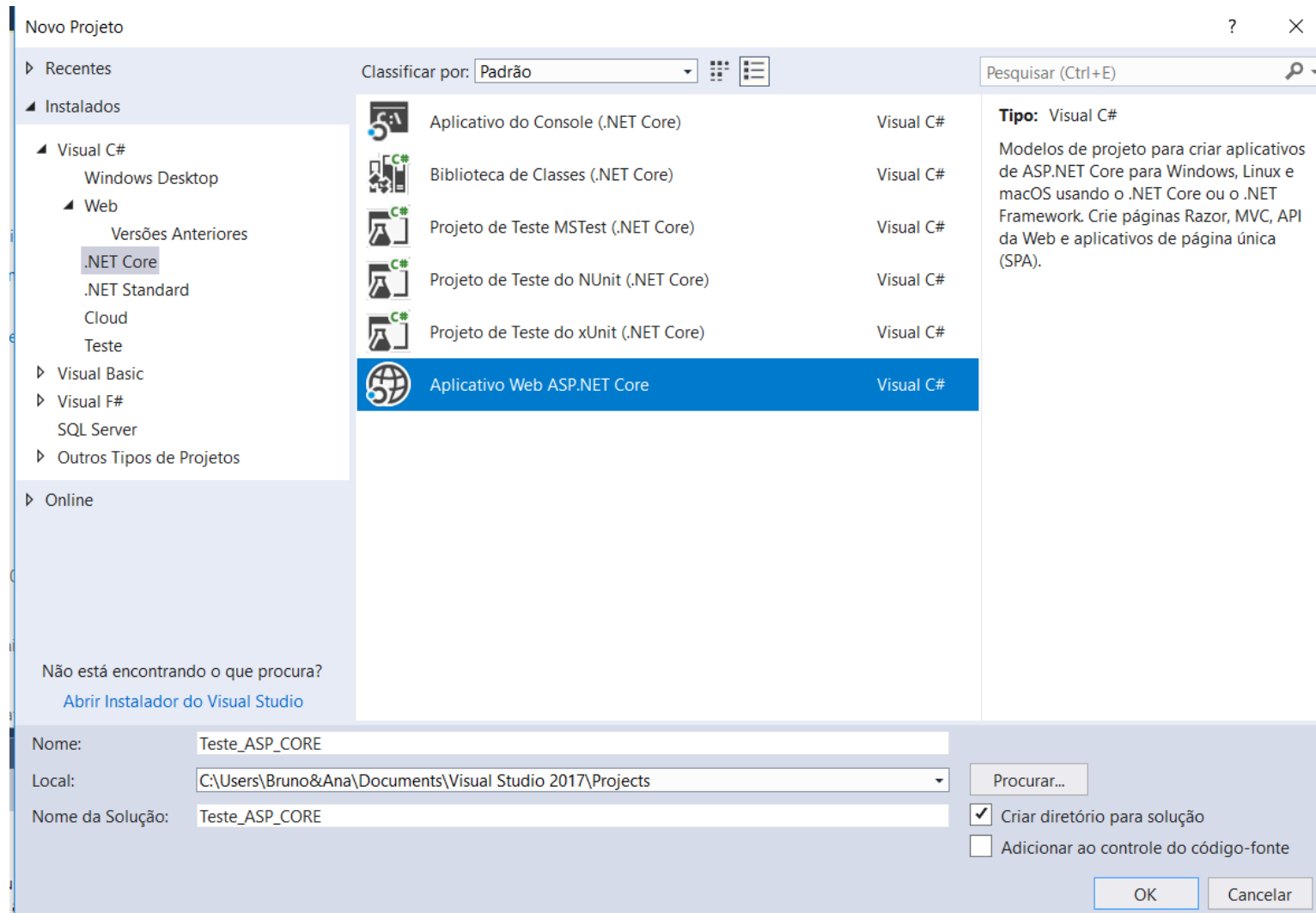
- O **SQL Server** é um SGBD da Microsoft, muito utilizado a nível empresarial que permite a gestão de dados de suporte a toda a organização;
- O **C#** ("C sharp") é uma linguagem de programação criada para o desenvolvimento de aplicações que executam sobre a Framework .NET.
- C# é uma linguagem poderosa, tipada e orientada a objetos;
- **ASP.NET Core** é uma framework para a construção de aplicações web e serviços;

Pré-requisitos

- Instalar SQL Server com Management Studio;
- Instalar o Visual Studio 2017 e certificar-se que o pacote .NET Core se encontra instalado;
- Link de download:
 - <https://visualstudio.microsoft.com/downloads/>
- É compatível com Mac e Linux!




Criar projeto




.NET Core

ASP.NET Core 2.2


[Saiba mais](#)




Vazio




API




Aplicativo Web




Aplicativo Web
(Modelo-
Exibição-
Controlador)




Biblioteca de
Classes Razor



Angular



React.js



React.js e
Redux

Um modelo de aplicação semelhante a um aplicativo web

[Saiba mais](#)

Autor:

Origem:

Autenticação:

Alterar

[Get additional project templates](#)

☐ **Habilitar Suporte ao Docker** (Requires [Docker para Windows](#))

Sistema Operacional:

Windows

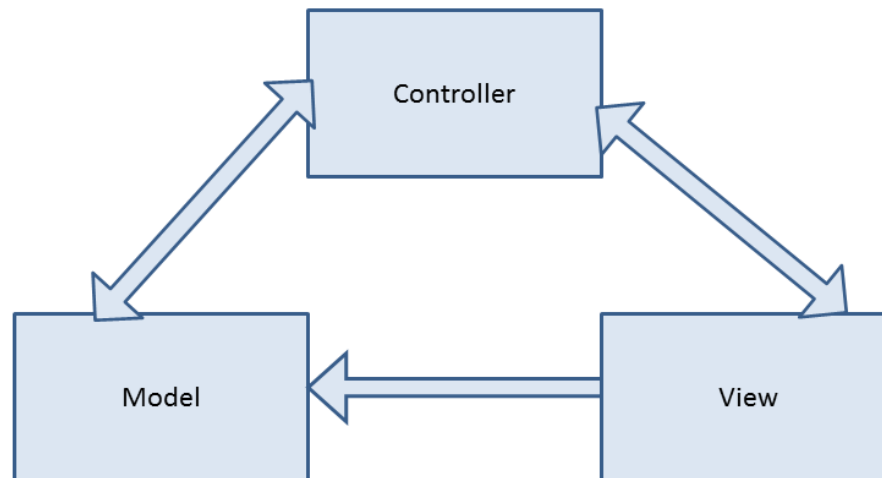
☒ **Configurar para HTTPS**

Criar um novo projeto com o visual studio:

- Caso não surja a opção ASP.NET Core 2.2, poderá instalar essa versão através do link:
- <https://dotnet.microsoft.com/download/dotnet-core/2.2>

MVC – Model View Controller

- É um padrão de arquitetura que separa a aplicação em três áreas distintas:
 - **Model**: Representa os dados que a aplicação utiliza;
 - **View**: Representa a interface gráfica com que o utilizador interage;
 - **Controller**: Representa a lógica aplicacional que relaciona as três áreas distintas;



MVC – Model View Controller

- Algumas *práticas* devem ser seguidas para tirar partido das *vantagens* da utilização do padrão MVC:
 - O *model* deverá ser um objeto com propriedades de *escrita* e *leitura* para suportar uma única *view*;
 - A *lógica* da *view* deverá ser *restringida* à interação do utilizador e *não* deve incluir lógica de negócio;
 - Os *controllers* devem ser *independentes* em relação à forma como os dados do *model* são manipulados;
 - Os *controllers* devem ser independentes em relação à forma como os dados são *armazenados* para além do *model*;

Web API

- O HTTP não é apenas utilizado para servir páginas web, podendo ser utilizado para **expor** serviços tendo por base o protocolo HTTP;
- Uma web API representa um conjunto de **sub-rotinas** com o âmbito de gerir dados entre cliente-servidor;
- Desta forma, aplicações *third-party* podem **interagir** com o servidor utilizando um protocolo aplicacional;
- A resposta poderá ser retornada utilizando **XML/JSON** ou outro qualquer tipo de formato de dados;
- O objetivo é que o resultado **não** contenha informação relacionada com o **layout** ou interação com o utilizador.

App startup: ASP.NET Core

- A classe startup:
 - Utiliza o método `ConfigureServices` para configurar os serviços da app;
 - Um serviço é um componente reutilizável que disponibiliza uma funcionalidade;
 - Os serviços são consumidos através do mecanismo de `dependency injection` ou `Application services`;
 - O método `Configure` cria um pipeline de processamento do pedido HTTP por parte da app;
 - No método `configure` podemos configurar, por exemplo, páginas específicas de desenvolvimento, gestão de exceções, redireccionamento, disponibilização de ficheiros estáticos ou páginas HTML/Razor;

Documentação adicional:
<https://docs.microsoft.com/en-us/aspnet/core/fundamentals/startup?view=aspnetcore-2.2>

Injeção de dependências

- O ASP.NET Core possui um mecanismo de gestão de dependências;
- Ao invés de instanciar classes específicas para gerir dependências de um projeto, as classes recebem as instâncias (tipicamente) como parâmetros do constructor;
- O ficheiro [startup.cs](#) permite configurar/registar as dependências do projeto;
- É necessário instalar o package (provavelmente já estará instalado: [Microsoft.AspNetCore.App](#))

Injeção de dependências

- No ficheiro: [startup.cs](#):
- 1) Remova o *middleware*: [app.Run](#) uma vez que não invoca *middlewares* subsequentes.
- 2) Coloque de acordo com o exemplo:

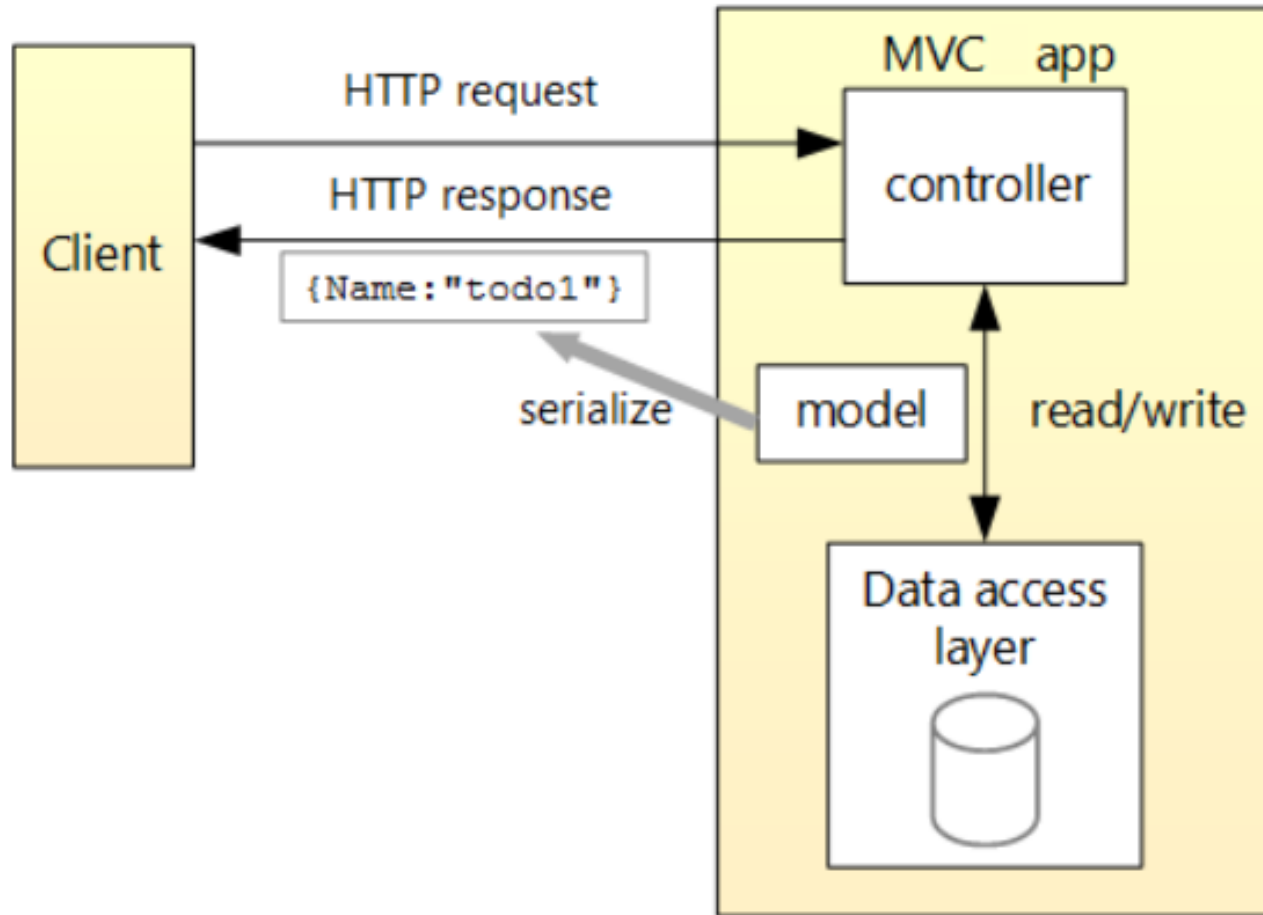
```
(...)  
public void ConfigureServices(IServiceCollection  
services)  
{  
    services.AddMvc();  
}  
  
public void Configure(IApplicationBuilder app,  
IHostingEnvironment env) {  
    if (env.IsDevelopment()) {  
        app.UseDeveloperExceptionPage();  
    }  
    app.UseMvcWithDefaultRoute();  
}  
(...)
```

URL e verbos HTTP

- Ao desenvolver uma API estamos preocupados em gerir dados e não páginas HTML.
- Os verbos HTTP são a chave para compreender o que o cliente necessita.
- Vamos considerar a existência de uma API: api/users.

Recurso	Leitura (GET)	Inserir (POST)	Atualizar (PUT)	Atualização parcial (PATCH)	Eliminar (DELETE)
Ação	Retorna uma lista de utilizadores	Cria um utilizador	Atualiza um utilizador	Atualiza utilizadores apenas com atributos específicos	Elimina um ou mais utilizadores
Resposta	Lista de utilizadores	Apresenta ou redireciona para o URL do novo utilizador	Código de estado HTTP	Código de estado HTTP	Código de estado HTTP

Arquitetura

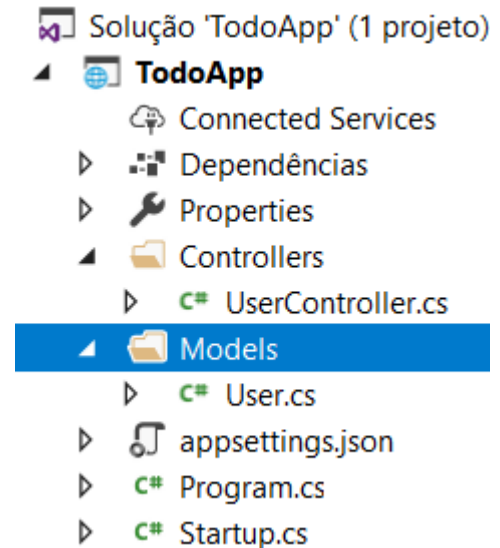


Modelos

- Crie uma pasta **Models** e uma classe com o nome **User**:

```
(...)  
public class User  
{  
    public int codigo { set; get; }  
    public string nome { set; get; }  
    public string email { set; get; }  
}  
(...)
```

Auto-Implemented Properties



Documentação adicional:

<https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/auto-implemented-properties>

Routing

- Crie uma pasta **Controllers** e uma classe com o nome **UserController**.
- Utilize o seguinte código:

```
[Route("api/[controller]")]
public class UserController : Controller
{
    [HttpGet]
    public User[] Get()
    {
        return new[] {(...) }
    }
    [HttpGet("{codigo}")]
    public User Get(int codigo)
    {
        var users = new[] {(...) };
        return users.FirstOrDefault(x => x.codigo ==
codigo);
    }
}
```

Indica que **controller** pode gerir todos os pedidos com o prefixo: **api** no URL

Especifica o verbo para a ação (GET). O GET está associado a leitura de dados. Podem ser utilizados vários verbos dependendo das necessidades: **HttpGet**, **HttpPost**, **HttpPut**, etc.

Neste caso, deverá ser indicado um parâmetro no URL: (api/user/1

Documentação adicional: Attribute routing

<https://docs.microsoft.com/en-us/aspnet/core/mvc/controllers/routing?view=aspnetcore-2.2>

Routing

- O Post também pode ser utilizado para a atualização de dados:

`[HttpPost]`

```
public IActionResult Add([FromBody] User user) {  
    var users = new List<User>();  
    users.Add(user);  
    return new CreatedResult($"/api/user/{user.Id}", user);  
}
```

Método POST

Este método retorna o objeto criado através do URL de forma a que o cliente possa aceder ao recurso. É emitido um código com o valor 201 (created).

Documentação adicional: Action return types

<https://docs.microsoft.com/en-us/aspnet/core/web-api/action-return-types?view=aspnetcore-2.2>

Eliminar dados através da API

`[HttpDelete]`

```
public IActionResult Delete([FromQuery] int id) {  
    var users = new List<User> {  
        new User() {Id = 1, Firstname = "Ugo", Lastname = "Lattanzi",  
                    Twitter = "@imperugo"}, new User() {Id = 2, Firstname = "Simone", Lastname =  
"Chiaretta", Twitter = "@simonech"} };  
    var user = users.SingleOrDefault(x => x.Id == id);  
  
    if (user != null)  
    {  
        users.Remove(user);  
        return new EmptyResult();  
    }  
  
    return new NotFoundResult();  
}
```

Status code 200 (OK). O corpo da resposta fica vazio

Representa a mensagem 404 que é utilizada quando o pedido do cliente não é encontrado

Excerto de código disponível em:

<https://github.com/brunobmo/ASP.NET-Training/tree/master/2019/Workshop1/ProjetoParte1>

Testar a API!

- Retornar todos os utilizadores
 - </api/user>
- Retornar um utilizador específico
 - </api/user/1>
- Eliminar um utilizador
 - </api/user?codigo=1>
- Adicionar um utilizador
 - </api/user/> (por POST)

Request
body:JSON

```
{  
  "id": 4,  
  "firstname": "bruno",  
  "lastname": "oliveira",  
  "twitter": "não tem"  
}
```

response

201 Created

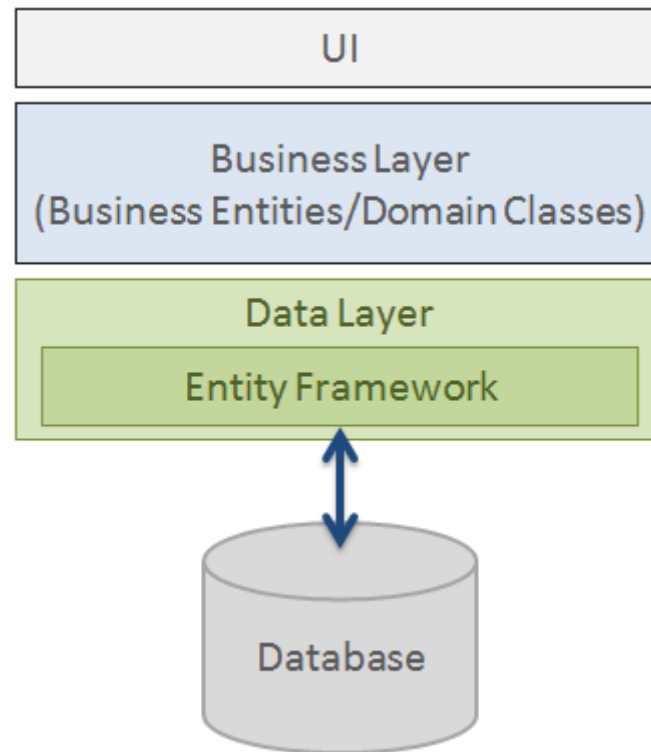
Headers▼

```
content-type: application/json; charset=utf-8  
location: /api/user/4  
server: Microsoft-IIS/10.0  
x-sourcefiles: =?UTF-8?B?QzpcVXNlcnNcQnJ1bm8mQW5hXE  
VNQX0NvcnVcYXBpXHVzZXJc?=  
x-powered-by: ASP.NET  
date: Mon, 01 Apr 2019 17:51:23 GMT  
content-length: 71  
X-Firefox-Spdy: h2
```

```
{  
  "id": 4,  
  "firstname": "bruno",  
  "lastname": "oliveira",  
  "twitter": "não tem"  
}
```


Entity Framework

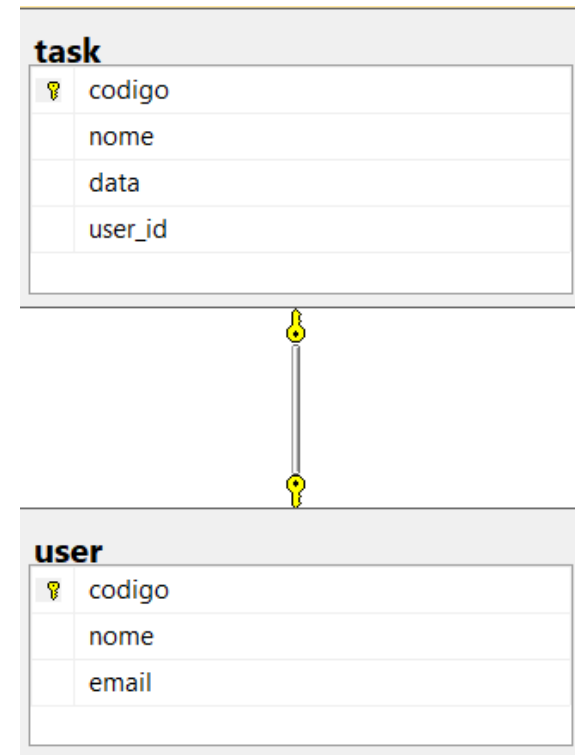
- A Entity Framework é um ORM suportado pela Microsoft que permite simplificar a interação dos programadores com a base de dados utilizando objetos .NET;



© EntityFrameworkTutorial.net

Base de dados

- Registrar **utilizadores (user)** e **tarefas (task)** de uma equipa:
 - User
 - Codigo
 - Nome
 - Email
 - Task
 - Codigo
 - Nome
 - Data
- Iniciar o SQL server Management Studio, criar a base de dados e criar a tabela **User** e a tabela **Task**;



Base de dados

- Tabela user:

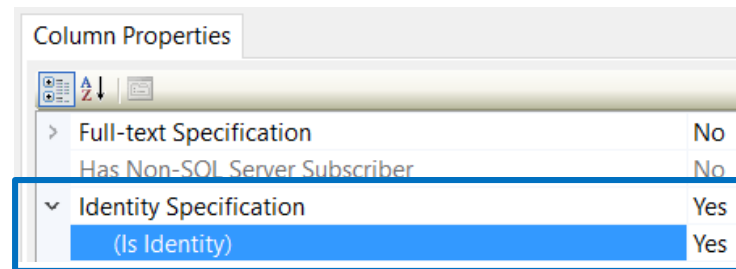
LAPTOP-7D49V42J.a...op2019 - dbo.task			
LAPTOP-7D49V42J.a...op2019 - dbo.user			
	Column Name	Data Type	Allow Nulls
🔑	codigo	int	<input type="checkbox"/>
	nome	varchar(50)	<input type="checkbox"/>
	email	varchar(75)	<input type="checkbox"/>

- Tabela task:

LAPTOP-7D49V42J.a...op2019 - dbo.task			
	Column Name	Data Type	Allow Nulls
🔑	codigo	int	<input type="checkbox"/>
	nome	varchar(40)	<input type="checkbox"/>
	data	datetime	<input type="checkbox"/>
	user_id	int	<input type="checkbox"/>

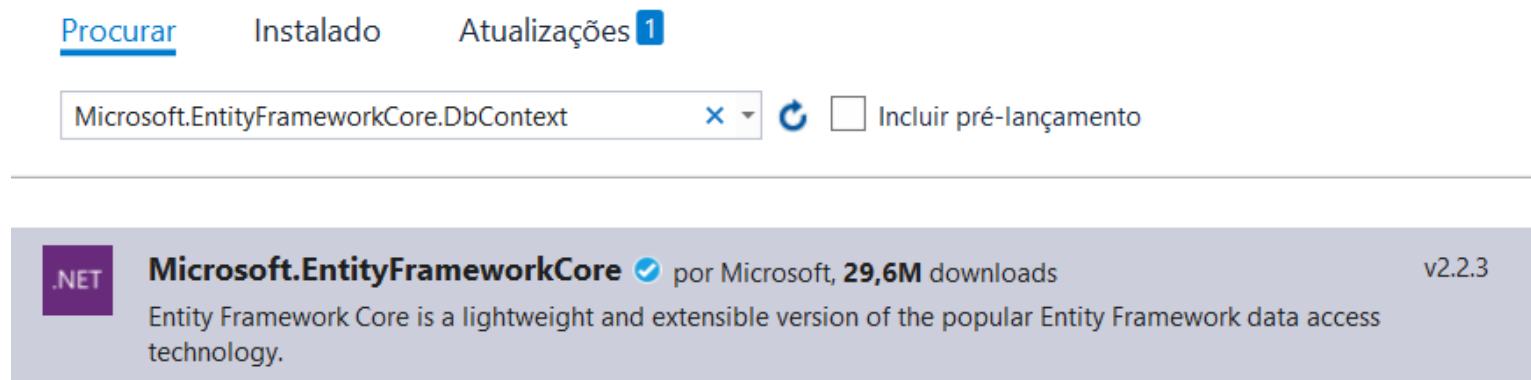
- Considerações:

- **codigo**: auto incremental (propriedade **identity**)



Adicionar o database context

- O database context é uma classe que coordena a Entity Framework para o modelo de dados;
- É necessário instalar o package: `Microsoft.EntityFrameworkCore.DbContext`;



Adicionar o database context

- Modificar o **Model: User**, adicionando a classe **UserContext**:

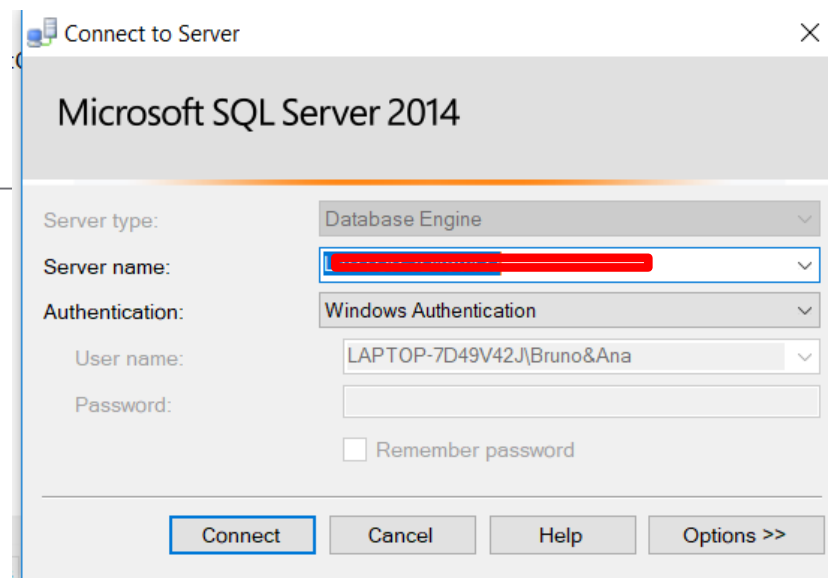
Conjunto de entidades utilizada para operações CRUD

```
(...)  
public class UserContext : DbContext  
{  
    public UserContext(DbContextOptions<UserContext> options)  
        : base(options)  
    {  
    }  
    public DbSet<User> TodoItems { get; set; }  
}  
(...)
```

Alterar a classe startup

- Registrar o *database context* com o sistema de *dependency injection* de forma a disponibilizar o serviço aos *controllers*:

```
(...)  
public void ConfigureServices(IServiceCollection services)  
{  
    /**SQL Server*/  
    var connection = @"Server=xpto;Database=aspNetCoreWorkshop2019;Trusted_Connection=True;ConnectRetryCount=0";  
    services.AddDbContext<UserContext>(options => options.UseSqlServer(connection));  
    /**SQL Server*/  
    services.AddMvc();  
}  
(...)
```



Registrar o database context

- Vamos modificar o UserController de forma a injetar o UserContext no controller:

```
public class UserController : Controller
{
    private readonly UserContext _context;

    public UserController(UserContext context)
    {
        _context = context;
    }
}
```

(...)

Documentação adicional

<https://www.entityframeworktutorial.net/entityframework6/dbset.aspx>

Alterar o user model

- A Entity Framework (EF) disponibiliza anotações específicas para o mapeamento entre as classes e a base de dados relacional:
 - **Key**: Por convenção a EF define o atributo Id ou um termo que contenha Id. Neste caso utilizamos a palavra código e como tal temos de anotar o modelo;
 - **Required**: Assegura que o atributo tem sempre um valor;
 - **StringLength**: controla o número de caracteres do modelo;
 - **[Table("User")]** ou **[Column("nome")]**: Não é necessário para o exemplo mas pode ser utilizado caso o nome da classe ou propriedades não possuem o mesmo nome das tabelas;

```
public class User
{
    [Key]
    public int codigo { set; get; }
    [Required]
    [StringLength(50)]
    public string nome { set; get; }
    [Required]
    [DataType(DataType.EmailAddress)]
    public string email { set; get; }
}
```

Documentação adicional

<https://docs.microsoft.com/en-us/ef/ef6/modeling/code-first/data-annotations>

Alterar os métodos get

```
[HttpGet]
public User[] Get(){
    return _context.user.ToArray();
}
```

```
[HttpGet("{codigo}")]
public ActionResult Get(int codigo)
{
    var user = _context.user.Find(codigo);
    if (user == null){
        return NotFound();
    }
    return Ok(user);
}
```

← HTTP 404

← HTTP 200 (ok), retornando o objeto

Documentação adicional
<https://docs.microsoft.com/en-us/aspnet/core/web-api/action-return-types?view=aspnetcore-2.2>

Alterar os métodos adicionar

```
[HttpPost]
public IActionResult Add([FromBody] User user)
{
    context.user.Add(user);
    context.SaveChanges();
    return new CreatedResult($"/api/user/{user.codigo}", user);
}
```

Deteta as alterações nas instâncias e procede às alterações na base de dados

Alterar os métodos eliminar

```
[HttpDelete]
public IActionResult Delete([FromQuery] int codigo)
{
    var user = _context.user.Find(codigo);
    if (user == null){
        return NotFound();
    }
    _context.user.Remove(user);
    _context.SaveChanges();
    return NoContent();
}
```

← HTTP Status Code 204

Excerto de código disponível em:

<https://github.com/brunobmo/ASP.NET-Training/tree/master/2019/Workshop1/ProjetoParte2>

Testar a API!

- Retornar todos os utilizadores
 - </api/user>
- Retornar um utilizador específico
 - </api/user/1>
- Eliminar um utilizador
 - </api/user?codigo=1>
- Adicionar um utilizador
 - </api/user/> (por POST)

Request
body:JSON

```
{
  "id": 4,
  "firstname": "bruno",
  "lastname": "oliveira",
  "twitter": "não tem"
}
```

response

201 Created

Headers▼

```
content-type: application/json; charset=utf-8
location: /api/user/4
server: Microsoft-IIS/10.0
x-sourcefiles: =?UTF-8?B?QzpcVXNlcj1bm8mQW5hXE
VNQX0NvcjVcYXBpXHVzZXJc?
x-powered-by: ASP.NET
date: Mon, 01 Apr 2019 17:51:23 GMT
content-length: 71
X-Firefox-Spdy: h2
```

```
{
  "id": 4,
  "firstname": "bruno",
  "lastname": "oliveira",
  "twitter": "não tem"
}
```

Adicionar o modelo para as tarefas

```
(...)  
public class Task  
{  
    [Key]  
    public int codigo { set; get; }  
    [Required]  
    [StringLength(20)]  
    public string nome { set; get; }  
    [Required]  
    [Column(TypeName = "datetime")]  
    public DateTime data { set; get; }  
    [Required]  
    public int user_id { set; get; }  
    [NotMapped]  
    [JsonIgnore]  
    public User user { set; get; }  
}  
(...)
```

Documentação adicional

<https://www.entityframeworktutorial.net/code-first/configure-one-to-many-relationship-in-code-first.aspx>

Adicionar o modelo para as tarefas

- Adicionar ao modelo User:

```
public virtual ICollection<Task> Tasks { get; set; }
```

Documentação adicional

<https://www.entityframeworktutorial.net/code-first/configure-one-to-many-relationship-in-code-first.aspx>

Criar o TaskController

```
(...)  
[Route("api/[controller]")]  
public class TaskController : Controller  
{  
    private readonly UserContext _context;  
    public TaskController(UserContext context){  
        _context = context;  
    }  
(...)
```

Criar o método de devolver tarefa

```
[HttpGet("{codigo}")]  
public ActionResult Get(int codigo){  
    var task = _context.task.Find(codigo);  
    if (task == null){  
        return NotFound();  
    }  
    return Ok(task);  
}
```


Criar o método de adicionar tarefa

```
[HttpPost]
public IActionResult Add([FromBody] Models.Task task){
    _context.task.Add(task);
    _context.SaveChanges();
    return new CreatedResult("/api/task/{task.codigo}", task);
}
```

Configurar o modelo

- Configurar o modelo utilizando a *Fluent API* para assegurar a restrição de chave estrangeira;
- Na classe `UserContext`, realizar o *override* do método `OnModelCreating`:

```
protected override void OnModelCreating(ModelBuilder modelBuilder)
{
    modelBuilder.Entity<Task>()
        .HasOne(t => t.user)
        .WithMany(u => u.Tasks)
        .HasForeignKey(t => t.user_id)
        .HasConstraintName("ForeignKey_User_Task");
}
```

- Adicionar o DbSet para Task:

```
public DbSet<Models.Task> task { get; set; }
```

Documentação adicional

<https://docs.microsoft.com/en-us/ef/core/modeling/relational/fk-constraints>

<https://www.entityframeworktutorial.net/code-first/fluent-api-in-code-first.aspx>

Tarefas do utilizador

```
[HttpGet("gettasks/{codigo}")]
public IActionResult getUserTasks(int codigo)
{
    var user = _context.user.Find(codigo);
    if (user == null){
        return NotFound();
    }
    var tasks = _context.task.Where(s=>s.user_id == codigo); ← LINQ Query
    foreach (Models.Task t in tasks){
        user.Tasks.Add(t);
    }
    return Ok(user);
}
```

Documentação adicional

<https://www.entityframeworktutorial.net/Querying-with-EDM.aspx>

Testar a API!

- Adicionar uma tarefa:
 - </api/user?id=2>
- Retornar todas as tarefas de um utilizador
 - </api/user/gettasks/1>
- Retornar uma tarefa específica
 - </api/task/4>

Excerto de código disponível em:

<https://github.com/brunobmo/ASP.NET-Training/tree/master/2019/Workshop1/ProjetoParte3>

Questões?

- Projeto completo na página:
 - <https://github.com/brunobmo/ASP.NET-Training/tree/master/2019/Workshop1>

Introdução à framework .NET com C# e ASP

Bruno Oliveira