

# DotNet Core + EntityFramework

## Sumário

Criar uma pasta “ProjDotNetCoreApi” .....	1
Nesta pasta criar o projeto via Terminal:.....	1
Incluir o EntityFramework no projeto:.....	1
A seguinte estrutura será criada automaticamente: .....	1
Executar o projeto:.....	1
Neste momento os seguintes arquivos serão criados automaticamente: .....	2
Após executar o projeto podemos utilizar o Swagger (documentação) para validar os endpoints .....	2
Swagger: .....	2
E podemos chamar API através do Postman: .....	3
Criar os Controllers:.....	3
Implementação do Projeto .....	4
Menu .....	4
Dog .....	5
Breed .....	5
PetContext.....	5
Startup.....	6
appsettings.json .....	6
Ajustes no DogController .....	7
Gerar banco.....	8
Teste API.....	8
Referência .....	9

Criar uma pasta “ProjDotNetCoreApi”

Nesta pasta criar o projeto via Terminal:

```

TERMINAL  PROBLEMS  OUTPUT  DEBUG CONSOLE

Windows PowerShell
Copyright (C) Microsoft Corporation. Todos os direitos reservados.

Experimente a nova plataforma cruzada PowerShell https://aka.ms/pscore6

PS C:\C#\DotnetCore\ProjDotNetCoreApi> dotnet new webapi
  
```

Incluir o EntityFramework no projeto:

```

PS C:\C#\DotnetCore\ProjDotNetCoreApi> dotnet add package Microsoft.EntityFrameworkCore.InMemory
  
```

dotnet add package Microsoft.EntityFrameworkCore.InMemory

A seguinte estrutura será criada automaticamente:

```

1 using System;
2 using System.Collections.Generic;
3 using System.Linq;
4 using System.Threading.Tasks;
5 using Microsoft.AspNetCore.Builder;
6 using Microsoft.AspNetCore.Hosting;
7 using Microsoft.AspNetCore.HttpsPolicy;
8 using Microsoft.AspNetCore.Mvc;
9 using Microsoft.Extensions.Configuration;
10 using Microsoft.Extensions.DependencyInjection;
11 using Microsoft.Extensions.Hosting;
12 using Microsoft.Extensions.Logging;
13 using Microsoft.OpenApi.Models;
14
15 namespace ProjDotNetCoreApi
16 {
17     1 reference
18     public class Startup
19     {
20         0 references
21     }
22 }
  
```

Executar o projeto:

```

Run  Terminal  Help  Startup.cs - Proj

Start Debugging  F5
Run Without Debugging  Ctrl+F5
Stop Debugging  Shift+F5
Restart Debugging  Ctrl+Shift+F5
Open Configurations
Add Configuration...
Step Over  F10
  
```

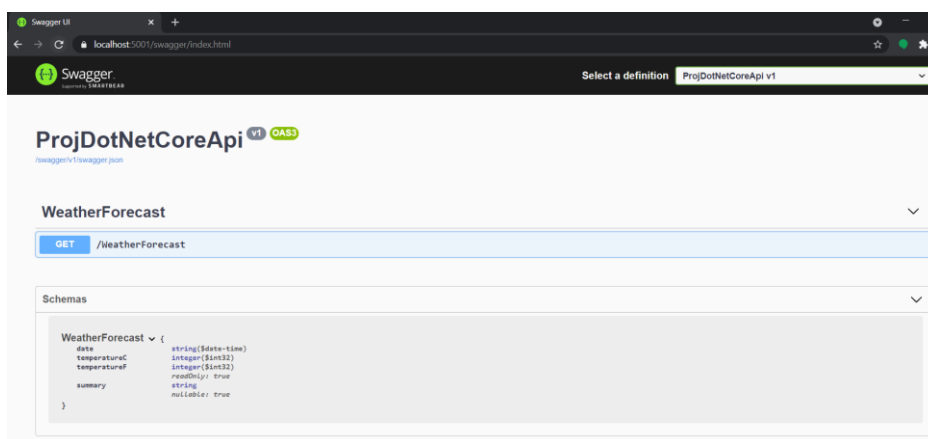
Neste momento os seguintes arquivos serão criados automaticamente:

- .vscode/launch.json
- .vscode/tasks.json

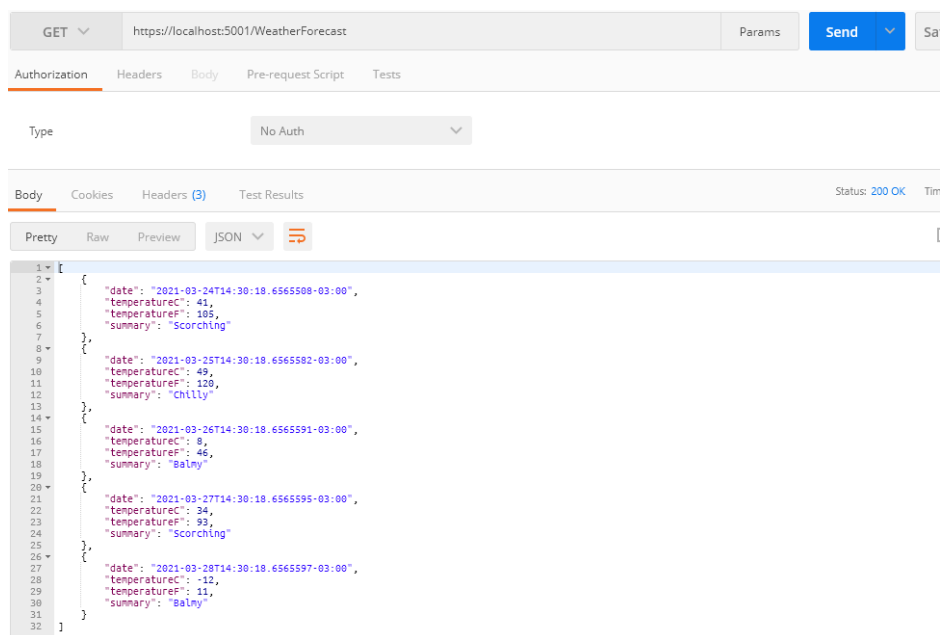
Após executar o projeto podemos utilizar o Swagger (documentação) para validar os endpoints existentes em nossa aplicação.

<https://localhost:5001/swagger/index.html>

Swagger:



E podemos chamar API através do Postman:



Após criar as classes: Pet, Dog e Breed

Criar a Classe (Dal) PetContext e fazer uma alteração na classe Startup.cs (Código nas próximas páginas)

Vamos instalar os seguintes pacotes com o objetivo de criar o Controller de forma automática:

- `dotnet add package Microsoft.VisualStudio.Web.CodeGeneration.Design`
- `dotnet add package Microsoft.EntityFrameworkCore.Design`
- `dotnet add package Microsoft.EntityFrameworkCore.SqlServer`
- `dotnet tool install -g dotnet-aspnet-codegenerator`

Criar os Controllers:

- `dotnet aspnet-codegenerator controller -name DogController -async -api -m Dog -dc PetContext -outDir Controllers`
- `dotnet aspnet-codegenerator controller -name BreedController -async -api -m Breed -dc PetContext -outDir Controllers`

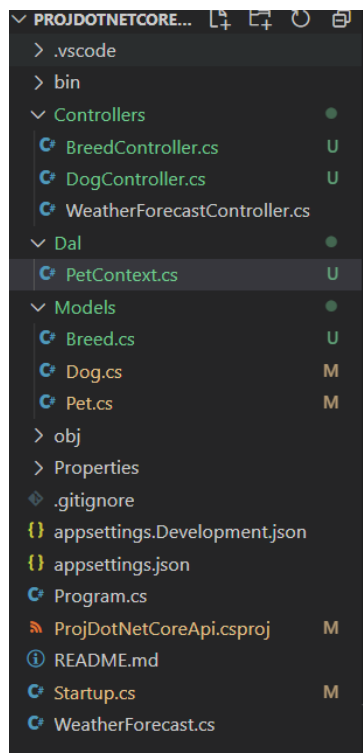
Após a Implementação dos controllers temos os seguintes endpoints criados:

<https://localhost:5001/swagger/index.html>



## Implementação do Projeto

### Menu



## Dog

```
public class Dog
{
    #region Propriedades
        public int Id { get; set; }
        public string Name { get; set; }
        public virtual Breed Breed { get; set; }
    #endregion
}
```

## Breed

```
public class Breed
{
    #region Propriedades
        public int Id { get; set; }
        public string Description { get; set; }
    #endregion
}
```

## PetContext

```
public class PetContext : DbContext
{
    public PetContext(DbContextOptions<PetContext> options) : base(options)
    { }
    public DbSet<Dog> Dogs {set; get;}
    public DbSet<Breed> Breeds {set; get;}

    protected override void OnModelCreating(ModelBuilder modelBuilder)
    {
        modelBuilder.Entity<Dog>()
            .HasOne(b => b.Breed);
        base.OnModelCreating(modelBuilder);
    }
}
```

## Startup

```

public IConfiguration Configuration { get; }

    // This method gets called by the runtime. Use this method to add services to
    the container.

    public void ConfigureServices(IServiceCollection services)
    {
        services.AddControllers();
        services.AddSwaggerGen(c =>
        {
            c.SwaggerDoc("v1", new OpenApiInfo { Title = "ProjDotNetCoreApi", Version = "v1" });
        });
        services.AddDbContext<PetContext>(options =>
            options.UseSqlServer(Configuration.GetConnectionString("SqlDbConnection")));
    }

```

## appsettings.json

```

{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft": "Warning",
      "Microsoft.Hosting.Lifetime": "Information"
    }
  },
  "ConnectionStrings": {
    "SqlDbConnection": "Data Source=(LocalDB)\\MSSQLLocalDB;Initial Catalog=dbdognew;
Integrated Security=True;"
  },
  "AllowedHosts": "*"
}

```

## Ajustes no DogController

```
// GET: api/Dog
[HttpGet]
public async Task<ActionResult<IEnumerable<Dog>>> GetDogs()
{
    //var dogs = await _context.Dogs.ToListAsync();
    return await _context.Dogs.Include(b => b.Breed).ToListAsync();
}
```

```
// GET: api/Dog/5
[HttpGet("{id}")]
public async Task<ActionResult<Dog>> GetDog(int id)
{
    var dog = await _context.Dogs.Include(b => b.Breed).FirstOrDefaultAsync(d
=> d.Id == id);
    if (dog == null)
    {
        return NotFound();
    }
    return dog;
}
```

```
[HttpPost]
public async Task<ActionResult<Dog>> PostDog(Dog dog)
{
    dog.Breed = _context.Breeds.First(b => b.Id == dog.Breed.Id);
    _context.Dogs.Add(dog);
    await _context.SaveChangesAsync();

    return CreatedAtAction("GetDog", new { id = dog.Id }, dog);
}
```



## Gerar banco

- dotnet tool install --global dotnet-ef
- dotnet ef migrations add v1
- dotnet ef database update

## Teste API

GET ▼ https://localhost:5001/api/Dog

Authorization Headers Body Pre-request Script Tests

Type No Auth ▼

Body Cookies Headers (3) Test Results

Pretty Raw Preview JSON ▼

```
1 {
2   {
3     "id": 1,
4     "name": "Mel",
5     "breed": {
6       "id": 1,
7       "description": "Labrador"
8     }
9   },
10  {
11    "id": 2,
12    "name": "Java",
13    "breed": {
14      "id": 2,
15      "description": "Poodle"
16    }
17  },
18  {
19    "id": 3,
20    "name": "Bili",
21    "breed": {
22      "id": 3,
23      "description": "Pastor Alemão"
24    }
25  }
26 }
```

GET ▼ https://localhost:5001/api/Dog/2

Authorization Headers Body Pre-request Script Tests

Type No Auth

Body Cookies Headers (3) Test Results

Pretty Raw Preview JSON ▼

```
1 {
2   {
3     "id": 2,
4     "name": "Java",
5     "breed": {
6       "id": 2,
7       "description": "Poodle"
8     }
9   }
10 }
```

## Referência

<https://docs.microsoft.com/pt-br/aspnet/core/tutorials/first-web-api?view=aspnetcore-5.0&tabs=visual-studio>