

## TUTORIAL: USANDO O .NET CORE PARA CRIAR UMA API SIMPLES, PRIMEIROS PASSOS

Autor: Lucas Allan Almeida Oliveira

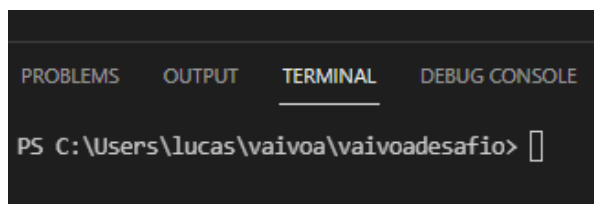
Olá, bem-vindo a esse tutorial que vai te auxiliar a criar sua primeira API utilizando *.NET Core* em C#, juntamente com o *Entity Framework Core*. Nós vamos abordar pontos básicos para utilizar essas tecnologias e no final teremos uma API com 3 funções básicas:

1. Enviar um e-mail de usuário pelo método POST e gerar um número de cartão aleatório e salvar no nosso BD
2. Retornar todos os objetos salvos no nosso BD
3. Utilizar um *endpoint* que recebe um e-mail específico e retorna todos os cartões que foram criados com este e-mail, ordenados do mais antigo para o mais recente.

Para seguir esse tutorial é necessário ter o dotnet e o VSCode instalados na máquina, assim como ter noções básicas de como utilizar o PostMan para testar a nossa API. Obs.: Alguns comandos apresentados só funcionam no Windows.

### 1. Criação do nosso projeto

Abra o terminal e utilize o comando `cd` para navegar entre os diretórios, até entrar na pasta onde você deseja criar o projeto.



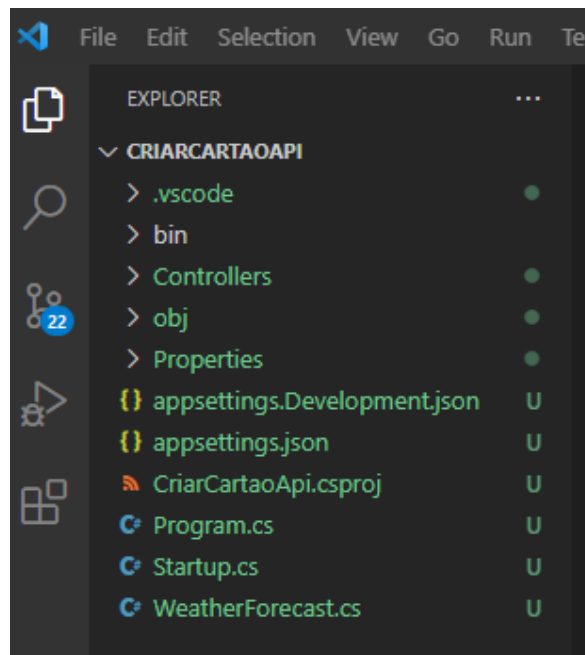
```
PROBLEMS  OUTPUT  TERMINAL  DEBUG CONSOLE
PS C:\Users\lucas\vaivao\vaivoadesafio>
```

No terminal, execute os comandos abaixo:

```
dotnet new webapi -o CriarCartaoApi
cd CriarCartaoApi
dotnet add package Microsoft.EntityFrameworkCore.InMemory
code -r ../CriarCartaoApi
```

Caso alguma caixa de diálogo perguntar se você deseja adicionar os ativos necessários, escolha sim.

Abra o projeto no Visual Studio Code



Confie no certificado, executando o código abaixo no terminal

```
dotnet dev certs https --trust
```

## 2. Ajustando o modelo que foi criado

Como o objetivo deste tutorial é mais simples, não vamos utilizar o Swagger (leia mais sobre o Swagger aqui: <https://docs.microsoft.com/pt-br/aspnet/core/tutorials/web-api-help-pages-using-swagger?view=aspnetcore-5.0>), assim sendo, acesse o arquivo **Properties\launchSettings.json** e atualize o campo **launchUrl** do *profile* **CriarCartaoApi** de **Swagger** para **api/cartões** e exclua o perfil **IIS Express**.

```
1 {
2   "$schema": "http://json.schemastore.org/launchsettings.json",
3   "iisSettings": {
4     "windowsAuthentication": false,
5     "anonymousAuthentication": true,
6     "iisExpress": {
7       "applicationUrl": "http://localhost:34535",
8       "sslPort": 44379
9     }
10  },
11  "profiles": {
12    "IIS Express": {
13      "commandName": "IISExpress",
14      "launchBrowser": true,
15      "launchUrl": "swagger",
16      "environmentVariables": {
17        "ASPNETCORE_ENVIRONMENT": "Development"
18      }
19    },
20    "CriarCartaoApi": {
21      "commandName": "Project",
22      "dotnetRunMessages": "true",
23      "launchBrowser": true,
24      "launchUrl": "swagger",
25      "applicationUrl": "https://localhost:5001;http://localhost:5000",
26      "environmentVariables": {
27        "ASPNETCORE_ENVIRONMENT": "Development"
28      }
29    }
30  }
31 }
```

### 3. Início da implementação

Agora vamos criar uma classe de modelo que vai representar os dados gerenciados pelo nosso aplicativo. Como este é um tutorial para início nos estudos em .Net Core, teremos apenas uma classe: **Cartao**.

Adicione ao projeto uma pasta chamada **Models**, e dentro dela crie um arquivo de classe chamado **Cartao**. Dentro deste arquivo, insira o código abaixo, que representa os campos que vamos utilizar no nosso app. Nós teremos apenas quatro campos na nossa classe:

- A primary key, **Id**, do tipo **long**
- Um campo para salvar o email, **Email**, do tipo **string**
- Um campo para salvar o número de cartão que será gerado, **CardNumber**, do tipo **string**
- E por último um campo que guarda a data/horário que a criação do cartão foi executada, **DataSolicita**, do tipo **string**

```
using System;

namespace CriarCartaoApi.Models
{
    public class Cartao
    {
        public long Id { get; set; }
        public string Email { get; set; }
        public string CardNumber { get; set; }
        public string DataSolicita { get; set; }
    }
}
```

Está na hora de adicionar um contexto ao nosso projeto, que nada mais é que a classe principal do nosso app que controla a funcionalidade do Entity Framework para um modelo de dados. Para isso crie a classe **CartaoContext** dentro da pasta **Models**. Copie dentro da classe o código abaixo.

```
using Microsoft.EntityFrameworkCore;

namespace CriarCartaoApi.Models
{
    public class CartaoContext : DbContext
    {
        public CartaoContext(DbContextOptions<CartaoContext> options)
            : base(options)
        {
        }

        public DbSet<Cartao> Cartoes { get; set; }
    }
}
```

Em seguida nós precisamos registrar o contexto do banco de dados no container de injeção de dependências. Esse container é quem vai fornecer o serviço aos controladores.

Acrescente algumas linhas de código no arquivo **Startup.cs** e se preferir remova as configurações referentes ao **Swagger**.

```
using Microsoft.EntityFrameworkCore;
using CriarCartaoApi.Models;
```

E dentro da função **ConfigureServices** deste mesmo arquivo, adicione o seguinte código

```
public void ConfigureServices(IServiceCollection services)
{
    services.AddDbContext<CartaoContext>(opt =>
        opt.UseInMemoryDatabase("CartoesLista"));
    services.AddControllers();
}
```

#### 4. Criação do scaffold para gerar o nosso controlador

Agora nós vamos fazer o *scaffold* de um controlador, que é uma ação que vai facilitar o nosso trabalho e criar algumas funções automaticamente. Para isso execute os seguintes códigos no terminal

```
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
```

```
dotnet aspnet-codegenerator controller -name CartoesController -async -api -m Cartao -dc CartaoContext -outDir Controllers
```

## 5. Configuração do controlador criado pelo Scaffold

Abra o arquivo **CartoesController** que está dentro da pasta **Controllers** (este arquivo foi criado durante a execução do *scaffold*) e atualize o nome da rota de acesso à api, substitua

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

por

```
[Route("api/Cartoes")]
```

Ainda neste mesmo arquivo, atualize a instrução **return** no **PostCartao** para usar o operador **nameof**:

```
return CreatedAtAction(nameof(GetCartao), new { id = cartao.Id }, cartao);
```

## 6. Construindo as funções para gerar Cartão e pesquisar por email

Agora vamos iniciar a construção da nossa função, o nosso primeiro objetivo é fazer com que, ao receber um email pelo método **POST** da nossa API, deve ser gerado um número de cartão de crédito aleatório, sendo este número retornado ao cliente que fez a requisição à API. Outro objetivo é inserir a data do sistema no campo **DataSolicita** da nossa classe **Cartao**. Para isso insira o código abaixo no **controler** na seção do **HttpPost**.

```
[HttpPost]
public async Task<ActionResult<Cartao>> PostCartao(Cartao cartao)
{
    //Código que gera o número do cartão com 16 dígitos
    string newCardNumber = "";
    Random randNum = new Random();
    for (int i = 0; i < 16; i++)
    {
        int digito = randNum.Next(0,9);
        newCardNumber += digito;
    }
    cartao.CardNumber = newCardNumber;

    //Buscando a data e hora do momento da criação do cartão
    string data = DateTime.Now.ToString();
    cartao.DataSolicita = data;

    _context.Cartoes.Add(cartao);
    await _context.SaveChangesAsync();
    return CreatedAtAction(nameof(GetCartao), new { id = cartao.Id
}, cartao.CardNumber);
}
```

Agora, vamos ao código no método **Get**, que nos permitirá recuperar todos os cartões cadastrados para um e-mail, usando o *endpoint* **api/cartoes/pesquisa/{email}**.

O que esse código faz, é basicamente usar o e-mail que foi inserido na chamada da API para fazer uma pesquisa e retornar todos os objetos (cartoes) criados com aquele e-mail. Além disso os dados retornados são ordenados de acordo com a data de criação, onde os cartões criados primeiro aparecem nas primeiras posições do *array* (retorno do método).

```
[HttpGet("pesquisa/{email}")]
public async Task<ActionResult<Cartao>> GetbyEmail(string email)
{
    var contas = from c in _context.Cartoes
                  select c;
    if (!String.IsNullOrEmpty(email))
    {
        contas = contas.Where(c => c.Email.Equals(email));
        contas = contas.OrderBy(c => Convert.ToDateTime(c.DataSoli
cita));
    }

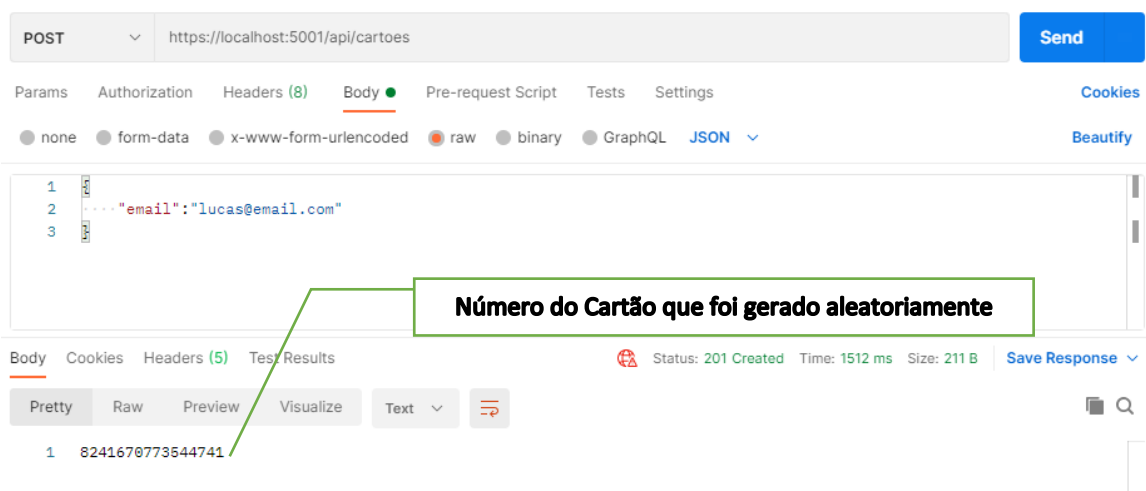
    if (contas == null)
    {
        return NotFound();
    }

    var cartoesUsuario = await contas.ToListAsync();
    return CreatedAtAction(nameof(GetbyEmail), cartoesUsuario);
}
```

## 7. Prontinho!!! Agora vamos testar o nosso app

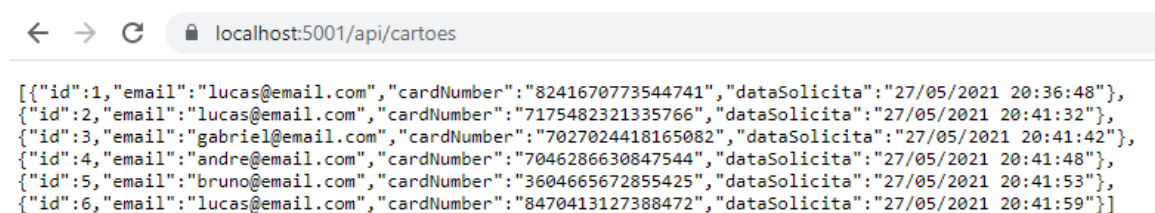
Se você fez tudo certinho é para o seu programa estar funcionando. Vamos testá-lo? Para isso vamos utilizar o PostMan para fazer algumas requisições do tipo GET na nossa API, dessa forma poderemos inserir alguns dados no nosso contexto.

Ao enviar um JSON com o email, no formato **{"email": "lucas@email"}** para o *endpoint* **api/cartoes**, o nosso app cria um objeto com id único, gera um número de cartão aleatório, salva o e-mail do usuário e guarda a data/hora da requisição. Após salvar essas informações, o método POST finaliza retornando o número do cartão que foi gerado.



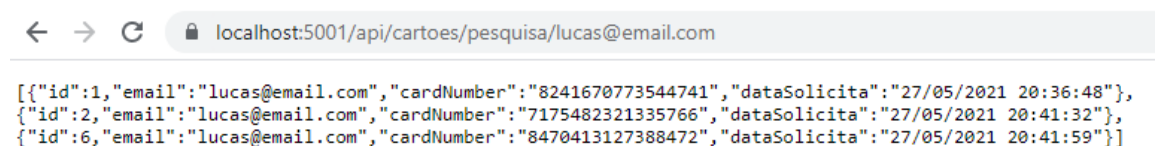
Crie alguns cartões com o auxílio do *software PostMan*, utilizando diferentes e-mails, mas ao mesmo tempo repetindo alguns dos e-mails, para que daqui a pouco nós possamos usar o método GET e testar a funcionalidade de filtragem por email.

Ao utilizar o *endpoint* `api/cartoes` no navegador, o método GET retornou todos os objetos que foram criados na forma de um *array* de objetos, mostrando todos os parâmetros que utilizamos na nossa Classe `Cartao`.



Para finalizar, vamos testar o *endpoint* de pesquisa, que permite que façamos uma filtragem e retornar apenas os objetos criados com um determinado e-mail. Para isso usaremos o *endpoint* `api/cartoes/pesquisa/lucas@email.com`

Agora, o retorno da nossa API contém apenas os objetos gerados utilizando o e-mail `lucas@email.com`. Repare que os objetos estão ordenados pela data de criação, onde os cartões mais antigos aparecem primeiro.



É isso aí, finalizamos o nosso Tutorial. Espero que tenham gostado e busquem se aprofundar cada vez mais nessa linguagem. Se você quiser pode acessar este projeto pronto no meu GitHub, basta clonar o repositório e executá-lo na sua máquina, o link é esse aqui:

<https://github.com/lucas11allan/vaivoadesafio>

Obrigado por terem acompanhado o Tutorial até o fim.

## REFERÊNCIAS

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

<https://docs.microsoft.com/pt-br/aspnet/core/mvc/controllers/routing?view=aspnetcore-5.0>

<https://docs.microsoft.com/pt-br/archive/msdn-magazine/2016/august/asp-net-core-real-world-asp-net-core-mvc-filters>

<https://pt.stackoverflow.com/questions/32715/como-filtrar-uma-lista-de-forma-ass%C3%ADncrona-usando-linq>