

# REVISÃO ASP.NET 2025 D.WEB.AV

## Sumário

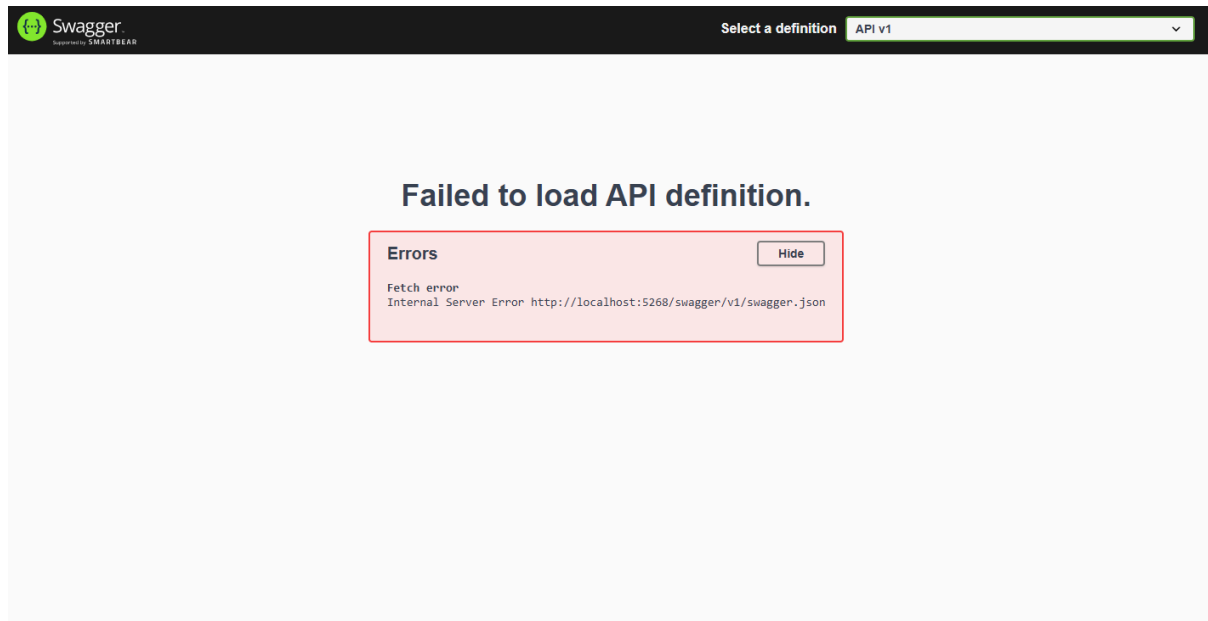
- aula 28/04/2025.
- CONECTADO NO WI.FI INSTALAÇÕES NECESSÁRIAS E SETUP DO PROJETO
- CRIANDO CONTROLLERS E ROTAS.
- CRIANDO MODELS E CONEXÃO COM BANCO DE DADOS COM ENTITY FRAMEWORK.
- CRIANDO INTERFACES E CONTINUAÇÃO DOS ENDPOINTS.
- REQUISIÇÕES VIA HTTP FILE E TESTANDO END POINTS.
- CONFIGURAÇÕES DE AUTENTICAÇÃO JWT
- FINALIZANDO AUTHENTICATION

INFOS DESSAS CORES SÃO:

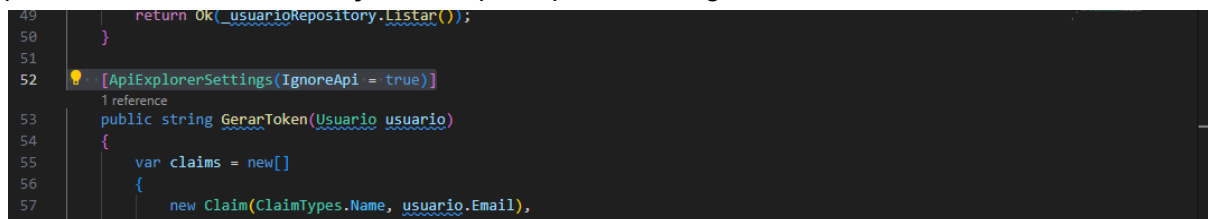
**Essenciais**

**Utilidades**

# Aula 28/04/2025



A causa deste erro é o nosso “GerarToken”, pois o Swagger pensa que ele é um endpoint porém ele não é, uma solução seria pedir para a API ignorar o “GerarToken”.



Anotações: Lembre-se, sempre que exportar o repositório é preciso reiniciar as migrations e o banco de dados

-Aprofundar práticas e conhecimentos com jwt Tokens

Dupla de avaliação Narcizo

## **PARTE 1, CONECTADO NO WI.FI INSTALAÇÕES NECESSÁRIAS E SETUP DO PROJETO**

Antes da criação do projeto, iremos criar um repositório chamado avaliação nele iremos colocar um READ ME com os nomes dos alunos e então iremos clonar o rep para finalmente criar o projeto dentro deste repositório Fora isso essas configs são necessárias

```
git config --global user.name "Katsumouley"
```

```
git config --global user.email "vitorluizsantosmougenot@gmail.com"
```

criação do projeto base

```
Cmd/: dotnet new webapi -n Revisao --no-https --use-controllers
```

```
Cmd/: dotnet new sln -n Revisao
```

```
Cmd/: dotnet sln Revisao.sln add Revisao/Revisao.csproj
```

Assim criamos o Solution e o projeto base, e os conectamos

```
PS E:\_MyDocuments\CSharpness\ASP.NET-entityframework\revisão bruta para prova> dotnet new webapi -n revisao --no-https --use-controllers
O modelo "API Web do ASP.NET Core" foi criado com êxito.

Processando ações pós-criação...
Restaurando E:\_MyDocuments\CSharpness\ASP.NET-entityframework\revisão bruta para prova\revisao\revisao.csproj:
A restauração foi bem-sucedida.

PS E:\_MyDocuments\CSharpness\ASP.NET-entityframework\revisão bruta para prova> dotnet new sln -n Revisao
O modelo "Arquivo de Solução" foi criado com êxito.

PS E:\_MyDocuments\CSharpness\ASP.NET-entityframework\revisão bruta para prova> dotnet sln Revisao.sln add .\revisao\revisao.csproj
O projeto 'revisao\revisao.csproj' foi adicionado à solução.
PS E:\_MyDocuments\CSharpness\ASP.NET-entityframework\revisão bruta para prova>
```

Não poderemos ter acesso a internet então esta etapa é de extrema importância primeiramente.

Agora podemos abrir o cmd dentro da pasta ./revisao e dar um

```
Cmd/: dotnet watch run
```

para rodar o projeto como um teste, apenas para verificar se o projeto está a funcionar dentro do cmd podemos fazer

```
Press in Cmd/: Ctrl + R para restartar o projeto
```

```
Press in Cmd/: Ctrl + C para encerrar o projeto
```

outros comandos de utilidade pública também são

```
Cmd/: dotnet build
```

Certo, ainda estamos conectados na internet, então agora precisamos instalar todas as bibliotecas que iremos utilizar.

Instalando os pacotes EF CORE e MYSQL

```
Cmd/: dotnet add package Pomelo.EntityFrameworkCore.MySql --version 8.*
```

```
Cmd/: dotnet add package Microsoft.EntityFrameworkCore.Design --version 8.*
```

Tenha em mente que ainda não estamos fazendo o setup do banco de dados estamos apenas instalando os pacotes que serão necessários para o mesmo

```
Cmd/: dotnet add package Microsoft.AspNetCore.Authentication.JwtBearer --version: 8.*
```

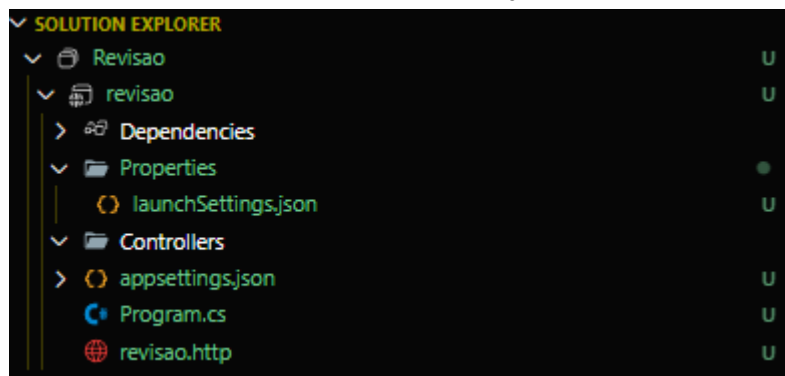
E por fim uma das últimas coisas necessárias de instalar antes do projeto é o pacote de autenticação Jwt

Fora isso outras coisas necessárias verificar seria a conexão com o github para realizar commit

e por fim podemos desconectar do WI.FI

## PARTE 2, CRIANDO CONTROLLERS E ROTAS.

Após fazer a limpa essencial no projeto criado  
Nosso sln explorer vai aparecer desse jeito



vamos criar duas pastas para separar a organização do arquivo



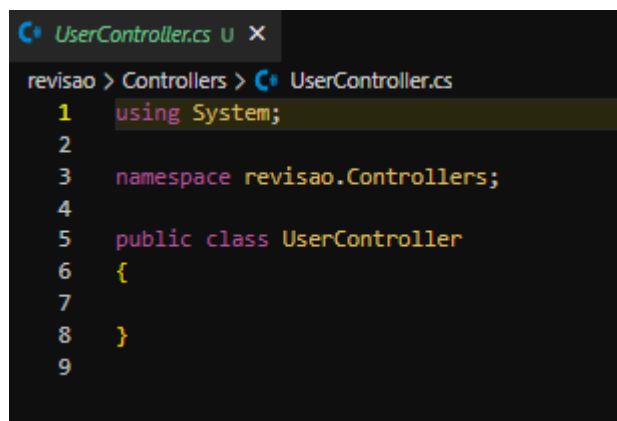
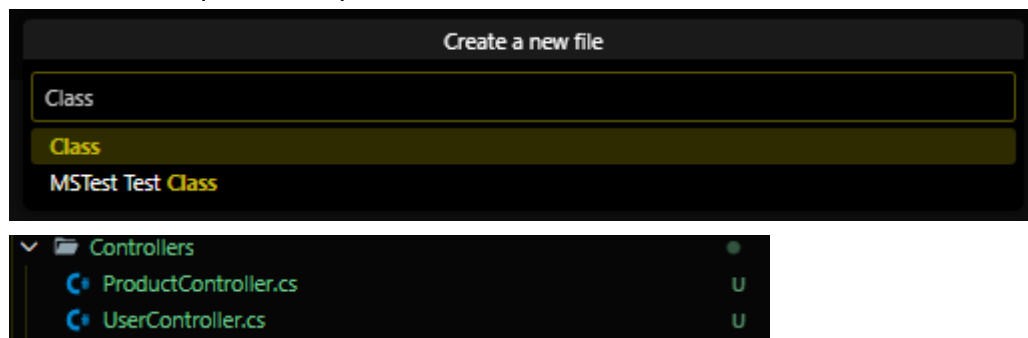
Sendo elas Models, onde iremos armazenar nossos modelos  
e TestRuns, onde iremos armazenar nossos testes

então vamos criar os controllers que a prova pede  
neste exemplo vamos criar 2 controllers

ProductController.cs

UserController.cs

vamos criar arquivos do tipo class



Agora dentro dos Controllers vamos criar nossa ROTA  
neste caso vamos criar todos os 4 tipos de endpoints essenciais para os controller  
inclusive 1 endpoint em especial(Listar item em específico)  
Durante a prova podemos criar os endpoints pedidos  
GET, POST, PUT e DELETE.  
para isso vamos começar especificando a rota antes da public class

```
revisao > Controllers > ProductController.cs
1  using System;
2  using Microsoft.AspNetCore.Mvc;
3
4  namespace revisao.Controllers;
5
6  [ApiController]
7  [Route("api/product")]
8  public class ProductController
9  {
10
11  }
12
```

Agora vamos importar métodos para controllers de API's

```
[ApiController]
[Route("api/product")]
public class ProductController : ControllerBase
{
}

```

o "ControllerBase" nos fornece métodos úteis como `Ok()`, `BadRequest()`, `NotFound()`, `Created()`, etc. Importante também prestar atenção nos Usings durante a sessão toda!  
Agora que temos a rota especificada e os métodos importados, podemos por fim criar os endpoints.

```
1  using System;
2
3  namespace revisao.Controllers;
4
5  [ApiController]
6  [Route("api/product")]
7  public class ProductController : ControllerBase
8  {
9      [HttpGet("list")]
10     public IActionResult Listar()
11     {}
12     [HttpGet("list/{id}")]
13     public IActionResult Listar()
14     {}
15     [HttpPost("create")]
16     public IActionResult Listar()
17     {}
18     [HttpPut("update/{id}")]
19     public IActionResult Listar()
20     {}
21     [HttpDelete("delete/{id}")]
22     public IActionResult Listar()
23     {}
24
25
26 }
27
```

## **PARTE 3, CRIANDO MODELS E CONEXÃO COM BANCO DE DADOS COM ENTITY FRAMEWORK.**

Agora iremos dar um saída dos Endpoints e iremos criar os modelos com qual iremos trabalhar e logo após iremos criar as interfaces para trabalharmos com os Endpoints novamente.

Vamos criar 3 modelos

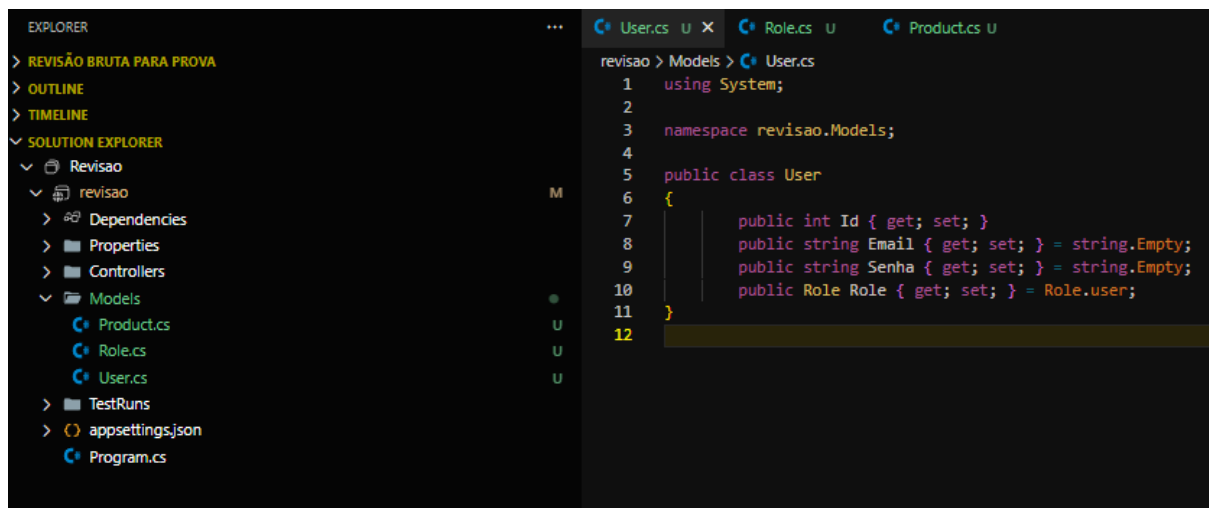
Product.cs

User.cs

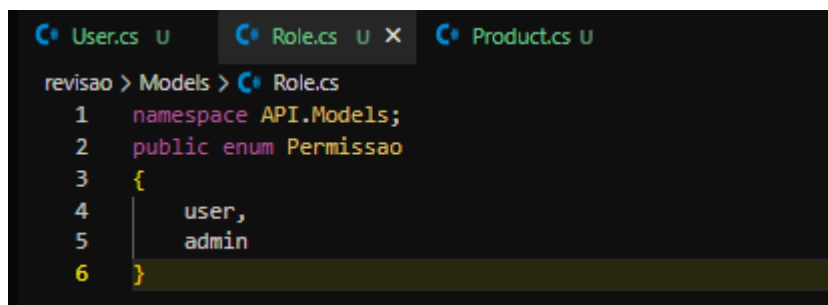
e Role.cs

Sendo o Role.cs representando o cargo do usuário(administrador ou usuário comum)

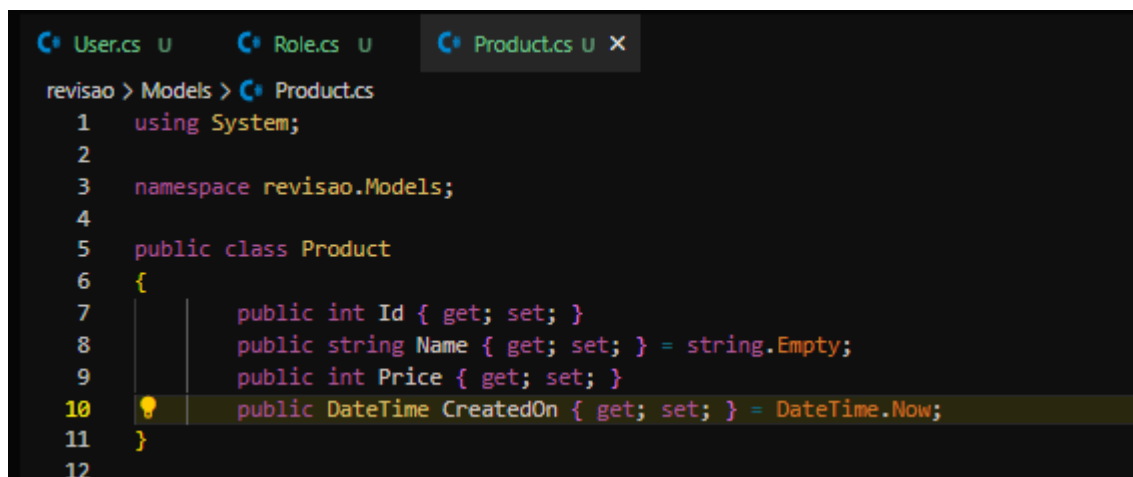
Ficando mais ao menos assim



```
1 using System;
2
3 namespace revisao.Models;
4
5 public class User
6 {
7     public int Id { get; set; }
8     public string Email { get; set; } = string.Empty;
9     public string Senha { get; set; } = string.Empty;
10    public Role Role { get; set; } = Role.user;
11 }
12
```



```
1 namespace API.Models;
2 public enum Permissao
3 {
4     user,
5     admin
6 }
```



```
1 using System;
2
3 namespace revisao.Models;
4
5 public class Product
6 {
7     public int Id { get; set; }
8     public string Name { get; set; } = string.Empty;
9     public int Price { get; set; }
10    public DateTime CreatedOn { get; set; } = DateTime.Now;
11 }
12
```

Agora que temos os modelos definidos, vamos dar um início a adição do banco de dados. Vamos iniciar isso criando uma pasta Data, onde iremos gerenciar interfaces e nosso banco de dados.



Agora dentro de Data vamos criar nosso DbContext com os DbSet necessários

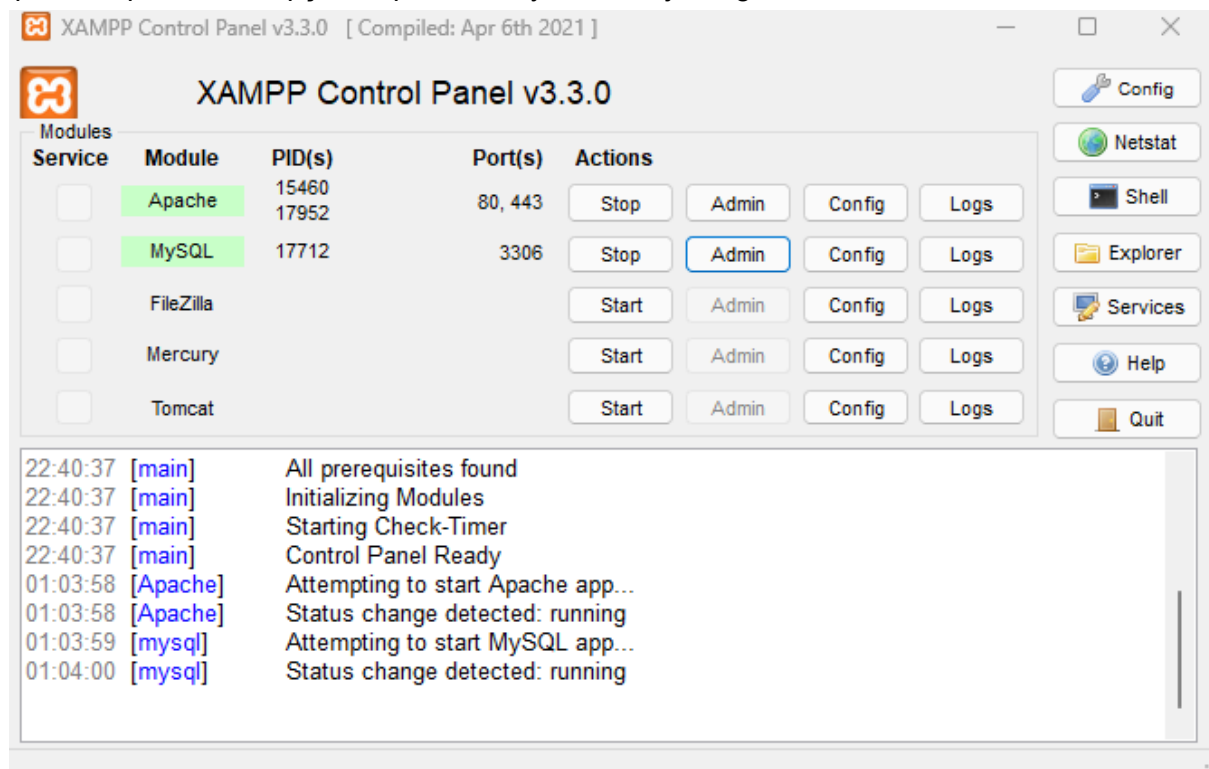
```
1 using System;
2 using Microsoft.EntityFrameworkCore;
3 using Models;
4
5 namespace revisao.Data;
6
7 public class AppDbContext : DbContext
8 {
9     public AppDbContext(DbContextOptions<AppDbContext> options) : base(options) { }
10
11     public DbSet<Product> Products { get; set; }
12     public DbSet<User> Users { get; set; }
13 }
14
15
```

Agora vamos para o nosso Program.cs configurar o nosso banco de dados

```
1 var builder = WebApplication.CreateBuilder(args);
2
3 builder.Services.AddControllers();
4 builder.Services.AddOpenApi();
5 //CONFIGURANDO BANCO DE DADOS**
6 var connectionString = builder.Configuration.GetConnectionString("DefaultConnection");
7
8 builder.Services.AddDbContext<AppDbContext>(options =>
9 |     options.UseMySQL(connectionString, ServerVersion.AutoDetect(connectionString)));
10 //
11 var app = builder.Build();
12
13 // Configure the HTTP request pipeline.
14 if (app.Environment.IsDevelopment())
15 {
16     app.MapOpenApi();
17 }
18
19 app.UseAuthorization();
20
21 app.MapControllers();
22
23 app.Run();
24
```



E por fim, vamos estabelecer uma conexão com um banco de dados, Acredito que o mais fácil no momento da prova será utilizando o Xampp, então abra o painel de controle e faça questão que ambas opções Apache e MySQL estejam ligadas.



Em seguida abra o painel de controle para pegar a porta para conectar o banco de dados

```

1  {
2    "Logging": {
3      "LogLevel": {
4        "Default": "Information",
5        "Microsoft.AspNetCore": "Warning"
6      }
7    },
8    "AllowedHosts": "*",
9    "ConnectionStrings": {
10     "Database": "server=localhost;port=3306;database=revisao;user=root;password="
11   }
12 }
13
  
```

Em seguida execute os seguintes comandos para criar os migrations e atualizar o banco de dados.

**Cmd/:** dotnet ef migrations add Inicial

**Cmd/:** dotnet ef database update

Caso o comando não esteja disponível, isso se deve por que você não instalou o pacote EF no começo do projeto, avise o professor e conecte se a internet novamente e execute este comando

**Cmd/:** dotnet tool install --global dotnet-ef

Caso tudo tenha dado certo, agora podemos prosseguir para as próximas etapas do projeto. Caso não tenha dado certo, use o comando abaixo para fazer o debug e verifique o material de consulta para ver se há algo que você está esquecendo, Caso esteja dando erro, verifique também de comentar os endpoints para realizar a conexão

Na etapa em que estamos ainda não construímos os endpoints completos então pode ser que não esteja executando por conta disso.

```
/*
    [HttpPost("signup")]
    public IActionResult Listar()
    {}

    [HttpGet("login")]
    public IActionResult Listar()
    {}

    [HttpGet("list")]
    public IActionResult Listar()
    {}

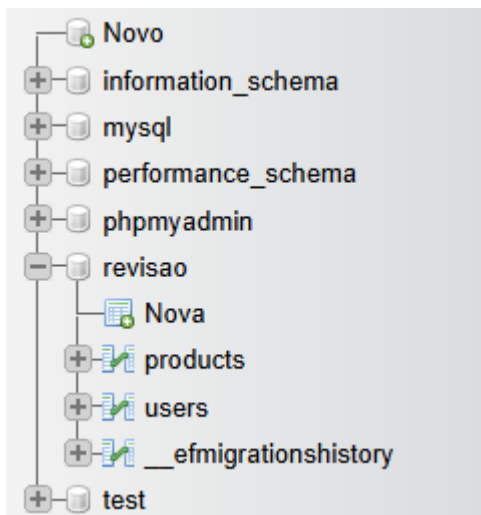
    [HttpGet("list/{id}")]
    public IActionResult Listar()
    {}

    [HttpPut("update/{id}")]
    public IActionResult Listar()
    {}

    [HttpDelete("delete/{id}")]
    public IActionResult Listar()
    {}
*/
```

Cmd/: dotnet build

Após isso verifique se o banco de dados está atualizado no canto do painel de administração.



## PARTE 4, CRIANDO INTERFACES E CONTINUAÇÃO DOS ENDPOINTS.

Agora vamos criar nossas Interfaces dentro de uma pasta chamada repositories para trabalhar com nossos repositórios, com o propósito de delegar o acesso a dados a uma camada própria.



Em nosso caso vamos criar as funções básicas que queremos realizar para cada interface.

```
1  using System;
2  using revisao.Models;
3
4  namespace revisao.Repositories;
5
6  public class IProductRepository
7  {
8      List<Product> ListarTodos();
9      Product? BuscarPorId(int id);
10     void Cadastrar(Product product);
11     void Atualizar(Product product);
12     void Deletar(Product product);
13     void Salvar();
14 }
15
```

```
1  using System;
2  using revisao.Models;
3
4  namespace revisao.Repositories;
5
6  public class IUserRepository
7  {
8      void SignUp(User user);
9      User? Login(string email, string password);
10     List<User> ListAll();
11     User? GetById(int id);
12     void Update(User user);
13     void Delete(User user);
14     void Save();
15 }
16
```

Agora que temos a interface vamos implementar nosso repositório

```
revisao > Repositories > ProductRepository.cs
1  using System;
2  using revisao.Models;
3  using revisao.Data;
4  using Microsoft.AspNetCore.Mvc;
5
6  namespace revisao.Repositories;
7
8  public class ProductRepository : IProductRepository
9  {
10     private readonly AppDbContext _context;
11     //Essa construção injeta a instância do AppDbContext
12     //no repositório para permitir o acesso ao banco de dados.
13     public ProductRepository( AppDbContext context)
14     {
15         _context = context;
16     }
17
18     public List<Product> List()
19     {
20         return _context.Products.ToList();
21     }
22
23     public Product? SearchId()
24     {
25         return _context.Products.FirstOrDefault(p => p.Id == id);
26     }
27
28     public void Create(Product product)
29     {
30         _context.Products.Add(product);
31     }
32
33     public void Update(Product product)
34     {
35         _context.Products.Update(product);
36     }
37
38     public void Delete(Product product)
39     {
40         _context.Products.Remove(product);
41     }
42
43     public void Save()
44     {
45         _context.SaveChanges();
46     }
47
48 }
49
```

revisao > Repositories > UserRepository.cs

```
1  using System;
2  using revisao.Models;
3  using revisao.Data;
4  using Microsoft.AspNetCore.Mvc;
5
6  namespace revisao.Repositories;
7
8  public class UserRepository : IUserRepository
9  {
10     private readonly AppDbContext _context;
11     //Essa construção injeta a instância do AppDbContext
12     //no repositório para permitir o acesso ao banco de dados.
13     public UserRepository( AppDbContext context)
14     {
15         _context = context;
16     }
17
18     public void SignUp(User user)
19     {
20         _context.Users.Add(user);
21     }
22     public User? Login(string email, string password)
23     {
24         return _context.Users.FirstOrDefault(x => x.Email == email && x.Password == password);
25     }
26     public List<User> List()
27     {
28         return _context.Users.ToList();
29     }
30
31     public User? SearchId()
32     {
33         return _context.Users.FirstOrDefault(p => p.Id == id);
34     }
35
36     public void Update(User user)
37     {
38         _context.Users.Update(user);
39     }
40
41     public void Delete(User user)
42     {
43         _context.Users.Remove(user);
44     }
45
46     public void Save()
47     {
48         _context.SaveChanges();
49     }
50 }
51
```

Agora precisamos registrar o Repositório no Container de Injeção de Dependência dentro do Program.cs

```
//Com isso, o repositório será automaticamente injetado em qualquer classe que precise dele.  
builder.Services.AddScoped<IProductRepository, ProductRepository>();  
builder.Services.AddScoped<IUserRepository, UserRepository>();  
//
```

E por fim podemos finalmente retornar aos controllers

Agora que temos os repositórios podemos deixar com que os controllers sejam responsáveis apenas por lidar com requisições HTTP.

vamos começar puxando nossas interfaces para nossos controllers com o mesmo método de injeção que utilizamos anteriormente

```
[ApiController]  
[Route("api/User")]  
public class IUserController : ControllerBase  
{  
    private readonly IUserRepository _repository;  
  
    public UserController(UserRepository repository)  
    {  
        _repository = repository;  
    }  
}
```

```
[ApiController]  
[Route("api/product")]  
public class ProductController : ControllerBase  
{  
    private readonly IProductRepository _repository;  
  
    public ProductController(IProductRepository repository)  
    {  
        _repository = repository;  
    }  
}
```

E então agora finalmente podemos ir construindo os endpoints

## GETS

```
[HttpGet("list")]
public IActionResult ListAll()
{
    var produtos = _repository.List();
    return Ok(produtos);
}
```

```
[HttpGet("list/{id}")]
public IActionResult ListById(int id)
{
    var produto = _repository.SearchId(id);
    if (produto == null)
    {
        return NotFound("Produto não encontrado.");
    }
    return Ok(produto);
}
```

```
[HttpGet("login")]
public IActionResult Login([FromQuery] string email, [FromQuery] string password)
{
    var user = _repository.Login(email, password);
    if (user == null)
    {
        return Unauthorized("Credenciais inválidas.");
    }
    return Ok(user);
}
```

```
[HttpGet("list")]
public IActionResult ListAll()
{
    var users = _repository.List();
    return Ok(users);
}

[HttpGet("list/{id}")]
public IActionResult ListById(int id)
{
    var user = _repository.SearchId(id);
    if (user == null)
    {
        return NotFound("Usuário não encontrado.");
    }
    return Ok(user);
}
```

## POSTS

```
[HttpPost("create")]
public IActionResult Create([FromBody] Product product)
{
    _repository.Create(product);
    _repository.Save();
    return CreatedAtAction(nameof(ListById), new { id = product.Id }, product);
}
```

```
[HttpPost("signup")]
public IActionResult Signup([FromBody] User user)
{
    _repository.SignUp(user);
    _repository.Save();
    return CreatedAtAction(nameof(ListById), new { id = user.Id }, user);
}
```

## PUT

```
[HttpPut("update/{id}")]
public IActionResult Update(int id, [FromBody] Product product)
{
    var existente = _repository.SearchId(id);
    if (existente == null)
    {
        return NotFound("Produto não encontrado.");
    }

    existente.Name = product.Name;

    _repository.Update(existente);
    _repository.Save();

    return Ok(existente);
}
```

```
[HttpPut("update/{id}")]
public IActionResult Update(int id, [FromBody] User updatedUser)
{
    var existing = _repository.SearchId(id);
    if (existing == null)
    {
        return NotFound("Usuário não encontrado.");
    }

    existing.Email = updatedUser.Email;
    existing.Password = updatedUser.Password;

    _repository.Update(existing);
    _repository.Save();

    return Ok(existing);
}
```



e Por fim mas não menos importante  
DELETE

```
[HttpDelete("delete/{id}")]
public IActionResult Delete(int id)
{
    var user = _repository.SearchId(id);
    if (user == null)
        return NotFound("Usuário não encontrado.");

    _repository.Delete(user);
    _repository.Save();

    return NoContent();
}
```

```
[HttpDelete("delete/{id}")]
public IActionResult Delete(int id)
{
    var produto = _repository.SearchId(id);
    if (produto == null)
    {
        return NotFound("Produto não encontrado.");
    }

    _repository.Delete(produto);
    _repository.Save();

    return NoContent();
}
```

Com isso temos todos os nossos Endpoints realizados, porém o projeto ainda não terminou

## PARTE 5, REQUISIÇÕES VIA HTTP FILE E TESTANDO ENDPOINTS

Agora vamos testar e ver se nosso Projeto está a rodar

Como estou muito cansado, vou largar aqui como deve ser cada projeto HTTP

```
1 @revisao_HostAddress = http://localhost:5189
2
3 ### Criar novo usuário (signup)
4 Send Request
5 POST {{revisao_HostAddress}}/api/User/signup
6 Content-Type: application/json
7 {
8   "email": "usuario@teste.com",
9   "password": "123456",
10  "role": 0
11 }
12
13 ###
14
15 ### Login de usuário
16 Send Request
17 GET {{revisao_HostAddress}}/api/User/login?email=usuario@teste.com&password=123456
18 Content-Type: application/json
19
20 ###
21
22 ### Listar todos os usuários
23 Send Request
24 GET {{revisao_HostAddress}}/api/User/list
25 Content-Type: application/json
26
27 ###
28
29 ### Listar usuário por ID
30 Send Request
31 GET {{revisao_HostAddress}}/api/User/list/1
32 Content-Type: application/json
33
34 ###
35
36 ### Atualizar usuário
37 Send Request
38 PUT {{revisao_HostAddress}}/api/User/update/1
39 Content-Type: application/json
40 {
41   "id": 1,
42   "email": "usuario@atualizado.com",
43   "password": "novaSenha",
44   "role": 0
45 }
46
47 ###
48
49 ### Deletar usuário
50 Send Request
51 DELETE {{revisao_HostAddress}}/api/User/delete/1
52 Content-Type: application/json
```

```

1  @revisao_HostAddress = http://localhost:5189
2
3  ### Listar todos os produtos
4  Send Request
5  GET {{revisao_HostAddress}}/api/product/list
6  Content-Type: application/json
7
8  ###
9
10 ### Listar produto por ID
11 Send Request
12 GET {{revisao_HostAddress}}/api/product/list/1
13 Content-Type: application/json
14
15 ###
16
17 ### Criar novo produto
18 Send Request
19 POST {{revisao_HostAddress}}/api/product/create
20 Content-Type: application/json
21 {
22   "name": "Produto Teste"
23 }
24
25 ###
26
27 ### Atualizar produto existente
28 Send Request
29 PUT {{revisao_HostAddress}}/api/product/update/1
30 Content-Type: application/json
31 {
32   "id": 1,
33   "name": "Produto Atualizado",
34   "createdOn": "2024-01-01T00:00:00"
35 }
36
37 ###
38
39 ### Deletar produto
40 Send Request
41 DELETE {{revisao_HostAddress}}/api/product/delete/8
42 Content-Type: application/json

```

Faça essas requisições e as teste

Com base nelas que iremos saber se o projeto está funcionando ou não

Caso elas tenham dado erro, nós precisaremos revisar cada parte e etapa do projeto

olhe o código de consulta e compare com o seu código

Por exemplo, eu esqueci de injetar corretamente os repositórios e fiquei 1 hora tentando resolver este problema. Então revise bem o código

também verifique o banco de dados para confirmar o funcionamento da API

Recente

Favoritos

Novo

information\_schema

mysql

performance\_schema

phpmyadmin

revisao

Nova

products

users

\_emigrationshistory

test

Visualizar

Estrutura

SQL

Procurar

Inserir

Exportar

Importar

Privilegios

Operacoes

Monitoramento

Acionadores

Mostrando registros 0 - 6 (7 no total. Consulta levou 0.0002 segundos.)

SELECT \* FROM `products`

Perfil [ Editar em linha ] [ Editar ] [ Demonstrar SQL ] [ Criar código PHP ] [ Atualizar ]

Mostrar tudo

Número de linhas: 25

Filtrar linhas: Procurar nesta tabela

Ordenar pela chave: Nenhum

Opções extras

Editar

Copiar

Remover

1

Produto Atualizado

2025-05-05 04:06:19.609482

Editar

Copiar

Remover

2

Produto Teste

2025-05-05 04:07:47.349909

Editar

Copiar

Remover

3

Produto Teste

2025-05-05 04:07:48.544369

Editar

Copiar

Remover

4

Produto Teste

2025-05-05 04:07:49.001219

Editar

Copiar

Remover

5

Produto Teste

2025-05-05 04:07:49.229715

Editar

Copiar

Remover

6

Produto Teste

2025-05-05 04:07:49.587510

Editar

Copiar

Remover

7

Produto Teste

2025-05-05 04:07:49.789121

Marcar todos

Com marcados: Editar Copiar Remover Exportar

## PARTE 6. CONFIGURAÇÕES DE AUTENTICAÇÃO JWT.

Certo, agora temos o projeto praticamente inteiro pronto

Porém falta uma coisa

Temos login e cadastro do usuário para nada?

Vamos aplicar agora o uso do token JWT (JSON Web Token), que se trata de um padrão aberto (RFC 7519) usado para **autenticação e troca segura de informações** entre partes como um objeto JSON.

Lá no começo do projeto nós instalamos seu pacote  
então agora iremos por em uso

primeiramente vá em seu appsettings.json novamente e coloque sua chave secreta  
Essa chave será usada para assinar e validar o token. Ela deve ter no mínimo 32 caracteres, com boa entropia.

```
{
  "Logging": {
    "LogLevel": {
      "Default": "Information",
      "Microsoft.AspNetCore": "Warning"
    }
  },
  "AllowedHosts": "*",
  "ConnectionStrings": {
    "Database": "server=localhost;port=3306;database=revisao;user=root;password="
  }
  "JwtSettings": {
    "SecretKey": "1722u0ecRh5i7L5yJC8SaxSvygbketej7VQTexUA0w="
  }
}
```

você pode gerar sua chave com os seguintes comandos

Cmd/: openssl rand -base64 32

Ou

Cmd/: [Convert]::ToBase64String((1..32 | ForEach-Object { Get-Random -Minimum 0 -Maximum 256 })))

Agora vamos adicionar ao Program.cs o seguinte

```
17 builder.Services.AddScoped<IUserRepository, UserRepository>();
18 //Autenticação JWT
19 var chaveJwt = builder.Configuration["JwtSettings:SecretKey"];
20
21 builder.Services.AddAuthentication(JwtBearerDefaults.AuthenticationScheme)
22     .AddJwtBearer(options =>
23     {
24         options.TokenValidationParameters = new TokenValidationParameters
25         {
26             ValidateIssuer = false,
27             ValidateAudience = false,
28             ValidateLifetime = true,
29             ClockSkew = TimeSpan.Zero,
30             ValidateIssuerSigningKey = true,
31             IssuerSigningKey = new SymmetricSecurityKey(
32                 Encoding.UTF8.GetBytes(chaveJwt!))
33         };
34     });
35
36 builder.Services.AddAuthorization();
37 var app = builder.Build();
38
```

E também peça para o app utilizar o Authentication

```
// Configure the HTTP request pipeline.  
if (app.Environment.IsDevelopment())  
{  
    app.MapOpenApi();  
}  
  
app.UseAuthentication();  
app.UseAuthorization();  
  
app.MapControllers();
```

Caso tenha dúvida de como funciona as configs do authentication aqui tem uma breve explicação

```
ValidateIssuer = false,           // Ignora o emissor do token  
ValidateAudience = false,        // Ignora o público do token  
ValidateLifetime = true,          // Verifica se o token está expirado  
ClockSkew = TimeSpan.Zero,        // opcional: remove tolerância de 5 min  
ValidateIssuerSigningKey = true,   // Garante que a assinatura do token é válida  
IssuerSigningKey = new SymmetricSecurityKey(  
    Encoding.UTF8.GetBytes(chaveJwt!)) // Chave secreta usada para validar o token
```

Tendo isto configurado podemos finalmente prosseguir para o conteúdo mais recente abordado, que é a autenticação em si.

A claro

e não se esqueça do using

```
using Microsoft.AspNetCore.Authentication.JwtBearer;  
using Microsoft.EntityFrameworkCore;  
using Microsoft.IdentityModel.Tokens;  
using System.Text;
```

## PARTE 7, FINALIZANDO AUTHENTICATION

Abra o seu UserController, e adicione o seguinte código

Primeiramente importe o seguinte

```
using Microsoft.IdentityModel.Tokens;  
using System.IdentityModel.Tokens.Jwt;  
using System.Security.Claims;  
using System.Text;  
primeiramente faça a injeção do IConfiguration
```

```
private readonly IUserRepository _repository;  
private readonly IConfiguration _configuration;  
  
public UserController(IUserRepository repository, IConfiguration configuration)  
{  
    _repository = repository;  
    _configuration = configuration;  
}
```

E por fim, Implementar isto no final do código

```
[ApiExplorerSettings(IgnoreApi = true)]//Esta linha é utilizada quando queremos que nosso código ignore um endpoint  
private string GenerateToken(User user)  
{  
    var claims = new[]  
    {  
        new Claim(ClaimTypes.Name, user.Email),  
        new Claim(ClaimTypes.Role, user.Role.ToString())  
    };  
  
    var chave = Encoding.UTF8.GetBytes(_configuration["JwtSettings:SecretKey"]);  
    var credenciais = new SigningCredentials(  
        new SymmetricSecurityKey(chave), SecurityAlgorithms.HmacSha256);  
  
    var token = new JwtSecurityToken(  
        claims: claims,  
        expires: DateTime.UtcNow.AddSeconds(30),  
        signingCredentials: credenciais);  
  
    return new JwtSecurityTokenHandler().WriteToken(token);  
}
```

A função **GerarToken** é responsável por gerar um **JWT (JSON Web Token)** que autentica o usuário na aplicação. Esse token contém **informações (claims)** como email e papel do usuário, é assinado com uma **chave secreta**, e tem um **tempo de expiração** definido.

🔒 O que ela faz:

1. **Cria os claims:**
  - **ClaimTypes.Name:** armazena o email do usuário.
  - **ClaimTypes.Role:** armazena o papel (ex: admin, user).
2. **Lê a chave secreta** do **appsettings.json** (**JwtSettings:SecretKey**).
3. **Cria credenciais de assinatura** usando HMAC-SHA256 com essa chave.
4. **Define a validade do token**, ex: **DateTime.UtcNow.AddSeconds(30)**.
5. **Gera e retorna o token JWT** como uma string.

E agora para finalizar, não se esqueça de ajustar o Login para utilizar o Authentication

```
[HttpPost("login")]
public IActionResult Login([FromBody] User user)
{
    var foundUser = _repository.Login(user.Email, user.Password);
    if (foundUser == null)
        return Unauthorized("Credenciais inválidas.");

    var token = GenerateToken(foundUser);
    return Ok(new { token });
}
```

E Proteja as rotas com [Authorize]

importando

```
using Microsoft.AspNetCore.Authorization;
```

```
[Authorize]
```

```
[HttpGet("listar")]
```

```
public IActionResult Listar()
```

```
{
```

```
    // Apenas usuários autenticados acessam aqui
```

```
}
```

```
[Authorize(Roles = "admin")]
```

```
[HttpDelete("deletar/{id}")]
```

```
public IActionResult Deletar(int id)
```

```
{
```

```
    // Apenas admins podem deletar
```

```
}
```



Agora para testarmos ajuste as requisições HTTP, novamente, em caso de erros revise tudo

[illegible]

```
@revisao_HostAddress = http://localhost:5189
```

```
### Criar novo usuário (signup)
```

```
Send Request
```

```
POST {{revisao_HostAddress}}/api/User/signup
```

```
Content-Type: application/json
```

```
{  
  "email": "Notadmin@example.com",  
  "password": "StrongPassword123",  
  "Role": 1  
}
```

```
###
```

```
### Login de usuário
```

```
Send Request
```

```
POST {{revisao_HostAddress}}/api/User/login
```

```
Content-Type: application/json
```

```
{  
  "email": "newuser@example.com",  
  "password": "StrongPassword123"  
}
```

```
###
```

```
### Listar todos os usuários
```

```
Send Request
```

```
GET {{revisao_HostAddress}}/api/User/list
```

```
Content-Type: application/json
```

```
###
```

```
### Listar usuário por ID
```

```
Send Request
```

```
GET {{revisao_HostAddress}}/api/User/list/1
```

```
Content-Type: application/json
```

```
###
```

```
### Atualizar usuário
```

```
Send Request
```

```
PUT {{revisao_HostAddress}}/api/User/update/1
```

```
Content-Type: application/json
```

```
{  
  "id": 1,  
  "email": "usuario@atualizado.com",  
  "password": "novaSenha",  
  "role": 0  
}
```

```
###
```

```
### Deletar usuário
```

```
Send Request
```

```
DELETE {{revisao_HostAddress}}/api/User/delete/1
```