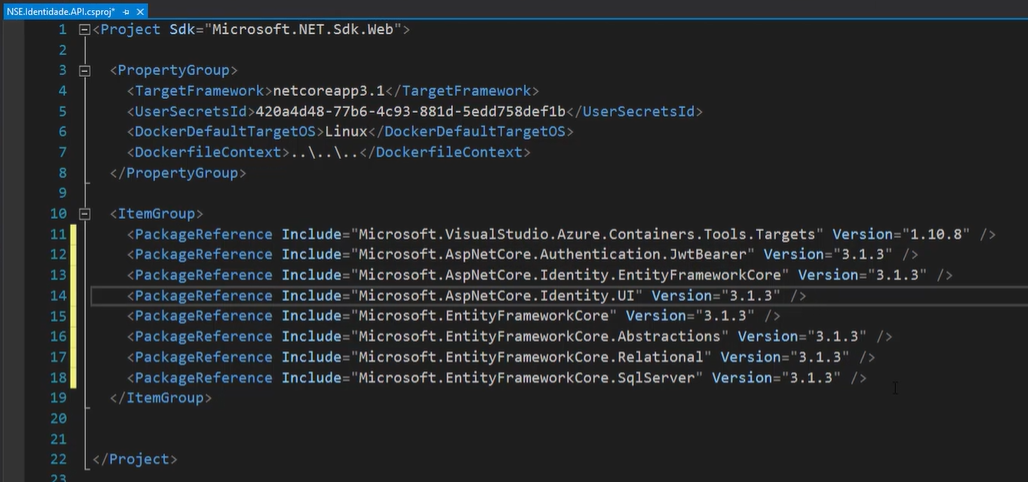
**Setup da arquitetura base**

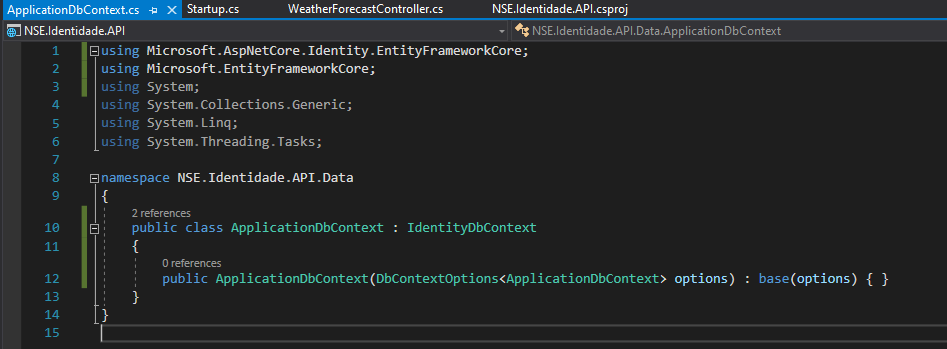
* Criar blank solution
* Criar as pastas necessárias para aplicação
* Criar a aplicação wev mvc na pasta web
* Criar APIs naa pasta servisse
* Crias class library na pasta building blocks/core

**Configuração do Identity**

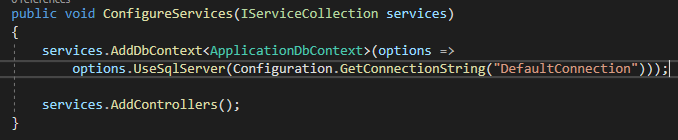
* Escrever as referências no csproj da API. Para abrir basta clicar duas vezes em cima do projeto da API
* Verificar se a versão dos pacotes são compatíveis com a versão do framework
* Uma vez salvo o arquivo, as dependências já serão instaladas.



* Para continuar com a configuração, deve-se criar uma pasta Data que é onde ficará o contexto que fará o acesso ao identity
* Criar a pasta Configuration, que fará a abstração de todas as configurações para que a classe startup fique mais limpa. Criar um extension métods para cada tipo de responsabilidade.
* Criar a pasta Extensions, onde serão escritas quaisquer classes que venham a estender o comportamento da API
* Criar a pasta Models, onde ficarão os modelos que representam os dados.
* Criar o ApplicationDbContext dentro de Data



* Mapear esse contexto no startup



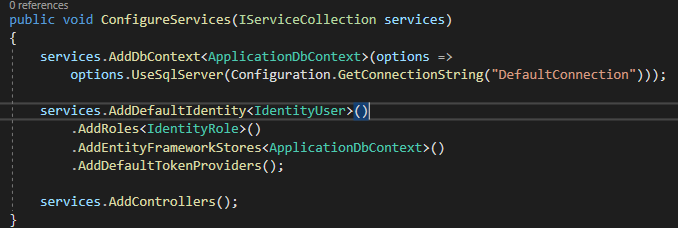
* Configurar a connection string no appsettings.json

"connectionStrings": {

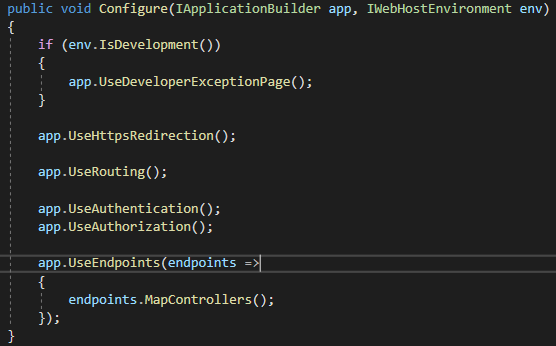
"DefaultConnection": "Server=(localdb)\\mssqllocaldb;Database=NerdStoreEnterpriseDB;Trusted\_Connection=True;MutipleActiveResultSets=true"

},

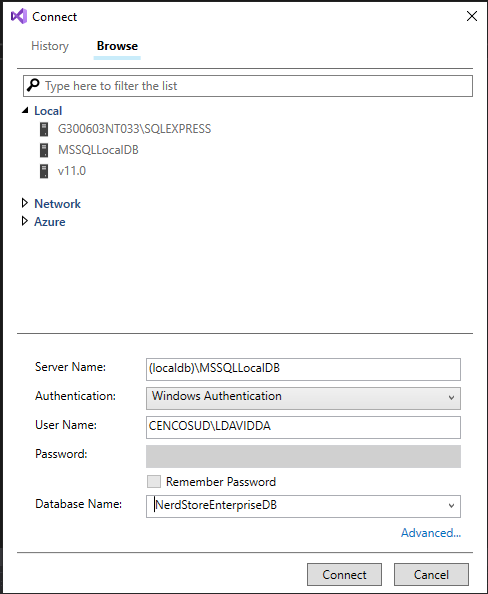
* Adicionar suporte ao identity no configurationsServices



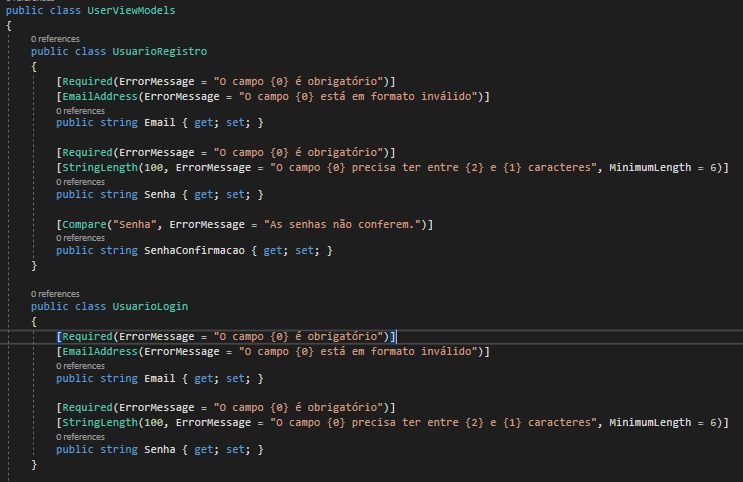
* Colocar o useAuthentication no Configure



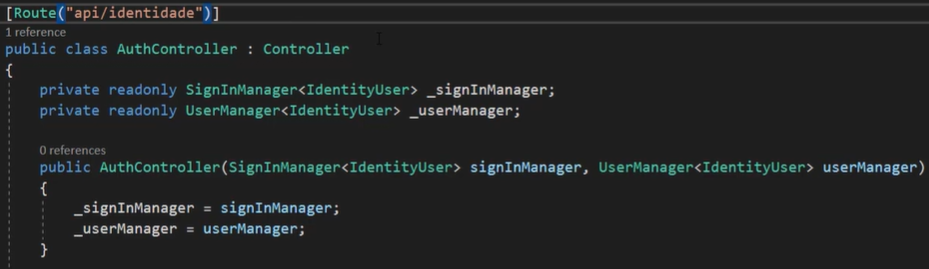
* Para que seja possível realizar a criação de uma migration, deve-se instalar o pacote Tools. Lembrar de Escolher o Default Project na hora de instalar.
  + Install-Package Microsoft.EntityFrameworkCore.Tools
* Realizar a migration
  + Add-migration Initial. Lembrar de referenciar o API como start up Project para que seja possível realizar a migration.
  + Update-datebase
* Adicionar ao SQL Server Object Explorer



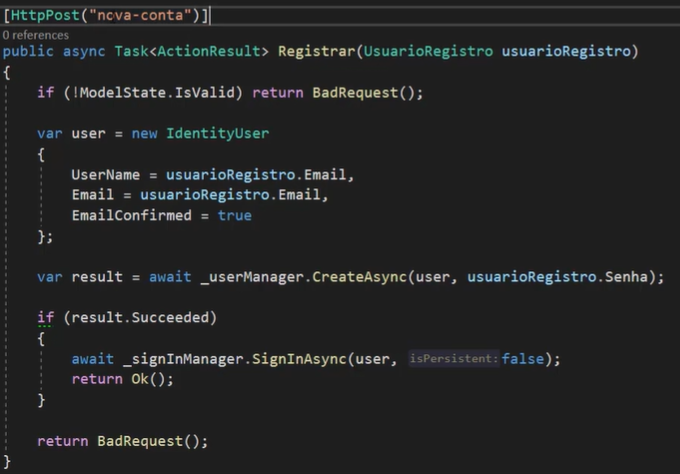
* **Registro e login de usuário**
* Na pasta Models, criar uma classe chamada UserViewModels, no plural, pois será utilizado mais de uma model dentro do arquivo.
  + Criar da UserViewModel deve-se criar as classes UsuarioRegistrar e UsuarioLogin



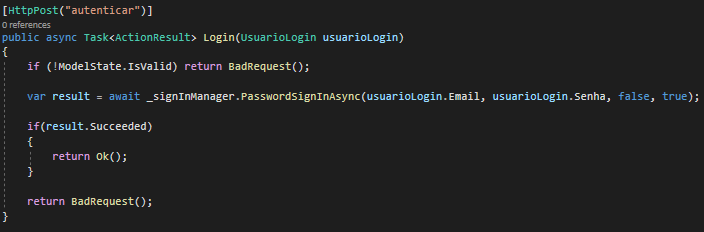
* Criar a controller principal AuthController
  + Herdar de controller
  + Injetar por dependências o SignInManager e o UserManager
  + Inicializar pelo construtor a injeção de dependências dos dois
  + Definir a rota



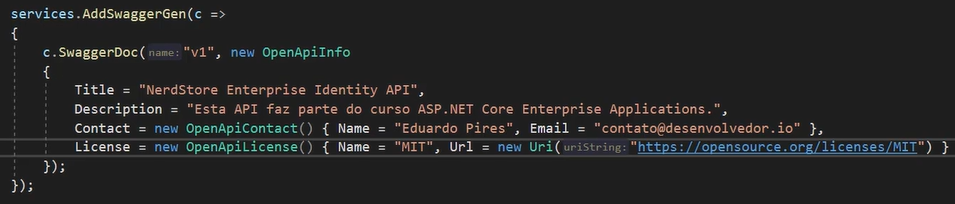
* Criar o método registrar e método login, cada um utilizando as devidas viewmodels
* No método registrar
  + Deve-se criar uma instância de identity user
  + Atribuir a variável result a criação do usuário
  + Realizar o login caso a criação proceda corretamente. A segunda variável do SignInAsync define se o usuário será lembrado ou não.
  + Definir o verbo e a rota



* Criar a mesma estrutura para o método login, porém utilizando a instância do sign in manager.



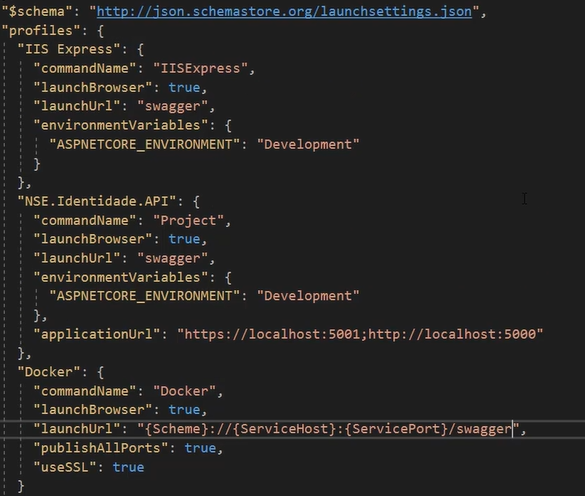
* **Documentação da API e testes**
* Clicar no projeto NSE.Identidade.API e instalar o pacote Swashbuckle.AspNetCore version=“5.3.3”. Pode fazer isso também através do Package Manager Console.
* Depois de instalar já possível começar a configurar as dependências do Swagger.
* Ir no StartUp.cs, no ConfigureServices e adicionar o MiddleWare de gereção de documentação ao pipeline
  + Adicionar linha services.AddSwaggerGen() .Essa configuração permite customização, o que faz com que a API fique um pouco melhor documentada.



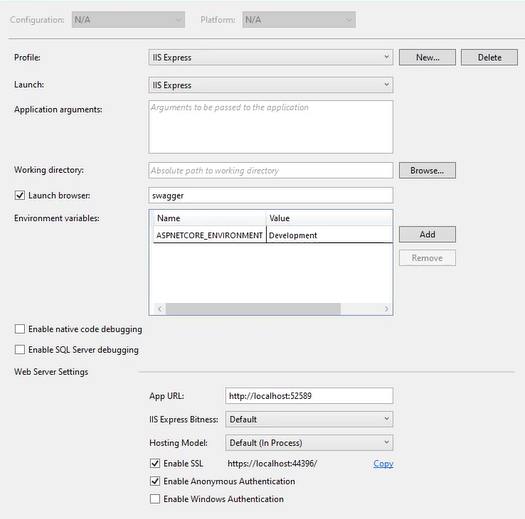
* No Configure será feito o uso do Swagger
  + Adicionar o app.UseSwagger() e o app.UseSwagger.UI(). Para definir o arquivo JSON que gerará a documentação, deve-se configurar o SwaggerUI



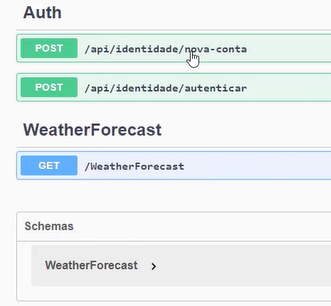
* Para rodar a aplicação, nesse primeiro momento não rodar com o Docker. Alterar o arquivo Identidade/Properties/lauchSetting.json para que seja possível abrir com o Swagger.



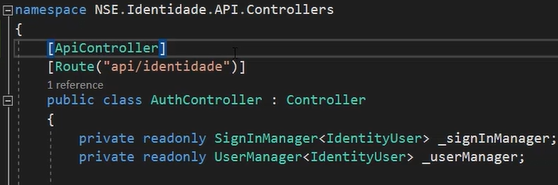
* Caso ainda continue abrindo com o Docker
  + Botão direito na aplicação NSE.Identidadee.API, propriedades, debug e altere de docker para IIS Express.



* Compilar e Rodar a aplicação para ver se estar tudo dentro do esperado. Caso ocorra tudo certo a documentação do swagger aparecerá .



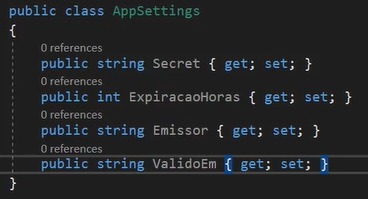
* Realizar o cadastro de uma pessoa e verificar se será gravado no banco de dados.
* Uma vez testado, deletar as classes e controllers que não são necessárias.
* Nas controllers, lembrar de definir que tipo de controller ela é.
  + Adcionar [ApiController]. Isso é importante pois o swagger já entenderá que é uma API e trafegará arquivos JSON



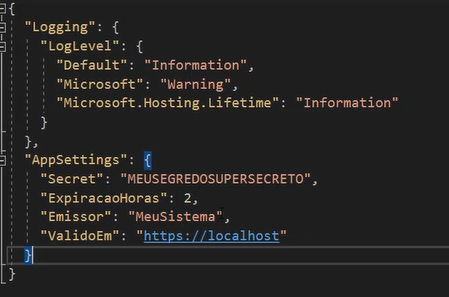
* **Configurando o JWT na API ­­**
* No ConfigureServices da StartUp da API, configurar o tipo de autenticação que será usado. Nesse caso será o Json Web Token.



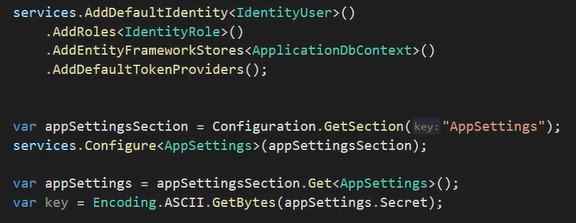
* Até essa etapa o JWT ainda não foi adicionado. A parte anterior apenas delimita que serão usados os padrões do JWT.
* Para continuar com a configuração, deve-se criar uma classe AppSettings.cs na pasta Extensions. Essas variáveis da classe serão usadas no preenchimento dos atributos do JWT.



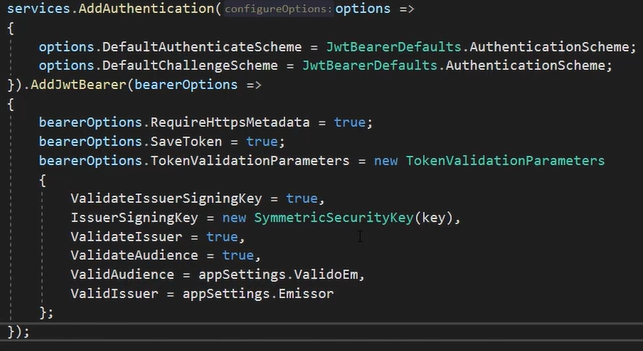
* No appsettings.Development.json, configurar o ambiente em que irá trabalhar. Os valores da classe AppSetings.cs serão designados nesse arquivo json.



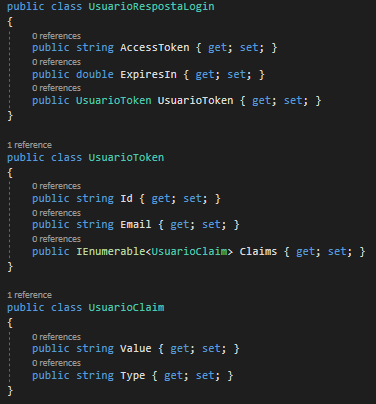
* No ConfigureServices da StartUp deve-se realizar a configuração para que seja possível importar os dados do arquivo AppSettings.json.



* Uma vez importado os dados do appSettings.json, deve prosseguir com a implementação do JWT no ConfigureServices.



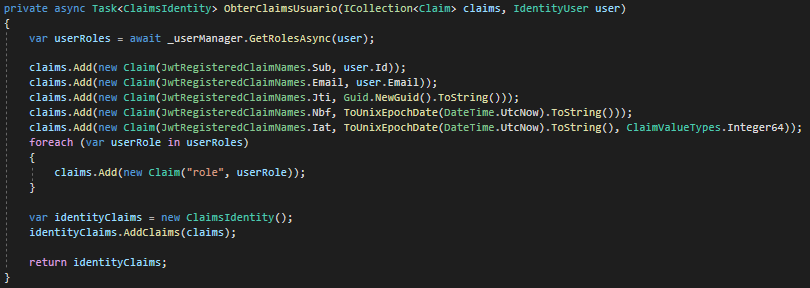
* **Emitindo JWT pela controller**
* Com a configuração de suporte ao JWT já concluída, agora a segunda parte da configuração será para que a controller possa gerar um tokken válido para quando o usuário fizer o login ou se registrar.
* Criar uma model UsuarioResposataLogin que será utilizado na construção do método seguinte. Criar também as model subsequentes. A criação da model pode ser feita na mesma classe, a UserViewModel .



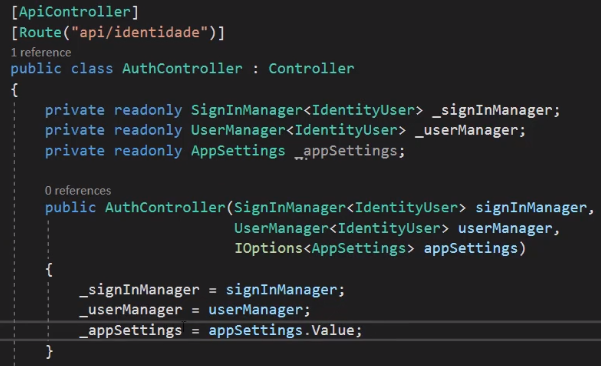
* Na AuthController:
  + Criar o método ToUnixEpochDtae que terá como função gerar a data de emissão e data de expiração do tokken. Esse método transforma a data no padrão offset, que é padrão do Unix.



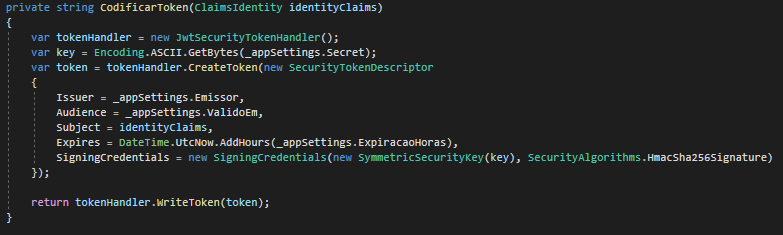
* + Criar o método que irá obter aa claims do usuário.



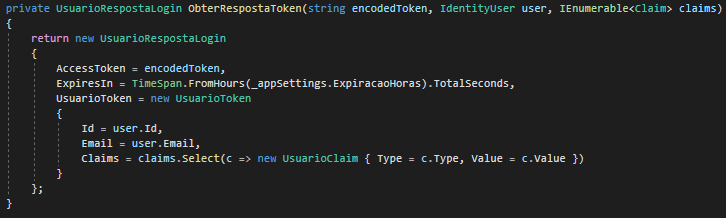
* + Injetar AppSettings para que seja possível utilizar o AppSecrets e assim usar a chave. Nesse caso o construtor não será resolvido como uma instância de AppSettings e sim como uma de IOptions<AppSettings>.



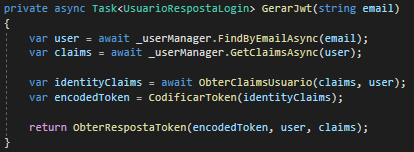
* + Criar método que codifica o tokken



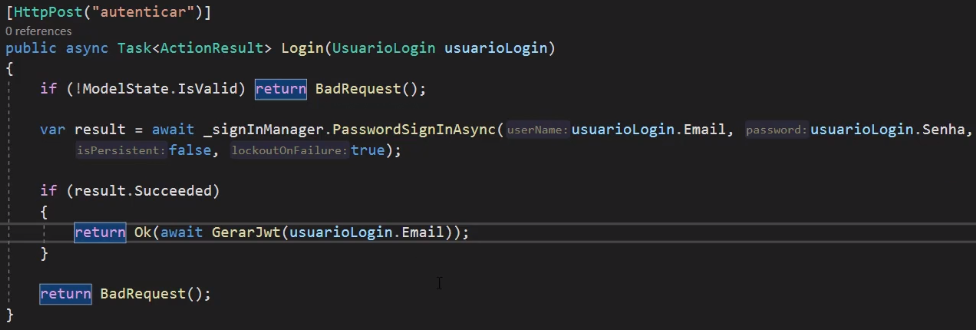
* + Criar método que popula a resposta.



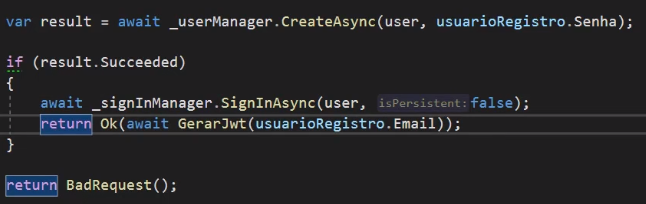
* + Como esses métodos já preparados, pode criar o método que englobará todos os processos.



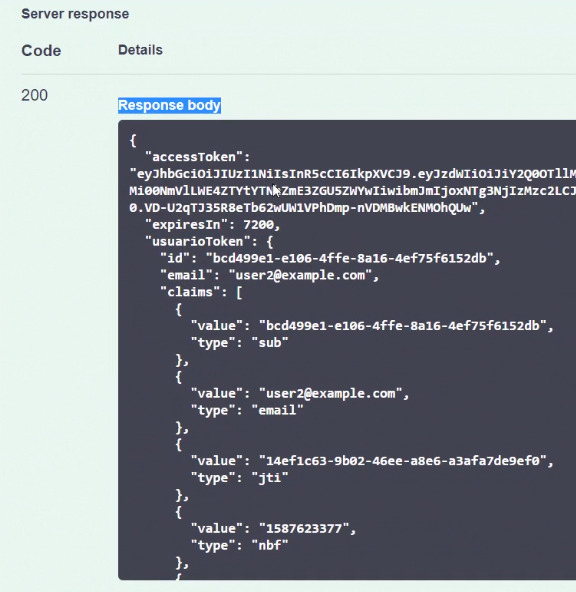
* No Método Login, colocar os dados do tokken na resposta.



* Realizar o mesmo procedimento como método de registrar usuário.



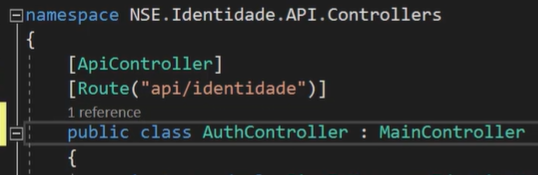
* Rodar a aplicação para verificar se estar tudo certo.



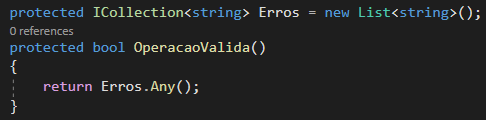
* **Response customizado**
* Como boa prática, deve-se criar uma controller base.
  + Essa controller deve herdar de Controller
  + Uma vez que herda de controller, pode-se implementar o atributo ApiController
  + Deve ser uma controller abstrata para que não possa trabalhar com ela diretamente. Ela tem de ser herdada



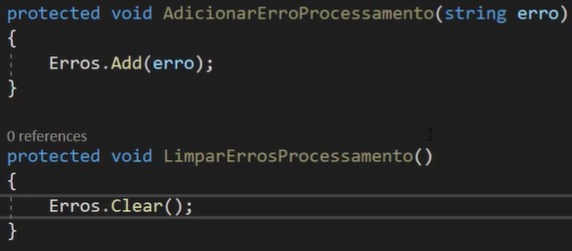
* AuthController deve agora herdar de maincontroller



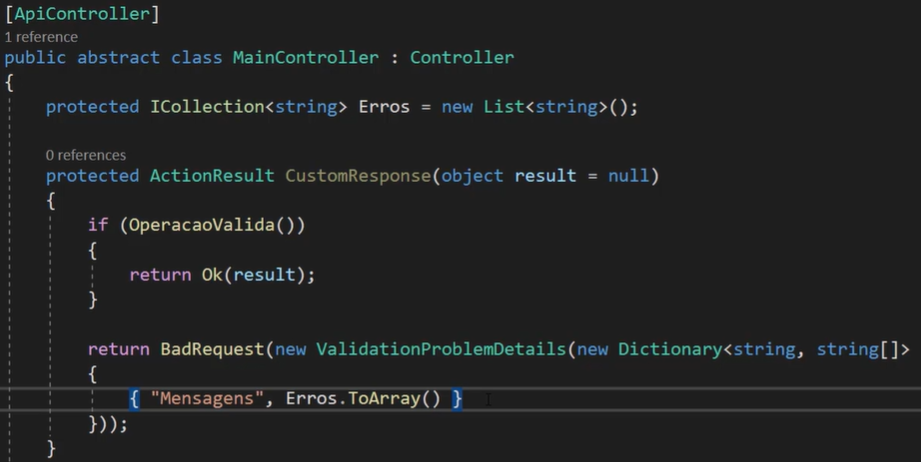
* Na MainController
  + Criar uma protected collection Erros, pois apenas quem herdar pode ter acesso a coleção de erros.
  + Criar o método OperacaoValida



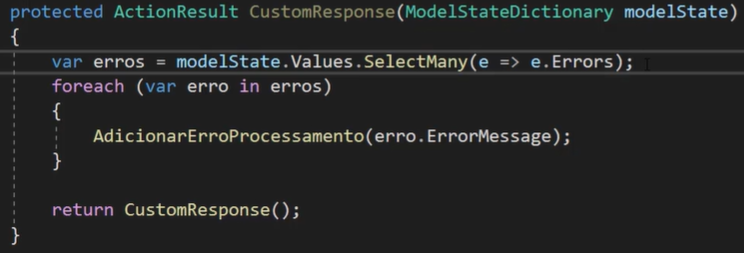
* + Criar o método AdicionarErroProcessamento e o método LimparErrosProcessamento



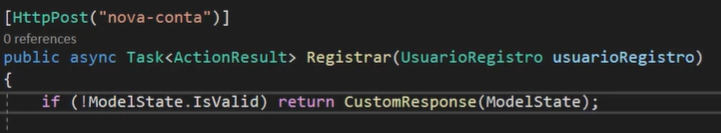
* + Criar uma CustomResponse
    - No BadRequest desse método, deve-se implementar o retorno do ValidationProblemDetails, pois é uma pratica recomendada na hora de se trabalhar com uma API.
    - Caso haja erros, eles serão passados em coleção dentro de um objeto chamado mensagem, que estará no corpo do response.



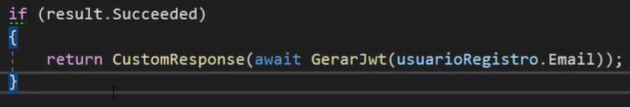
* + Criar uma sobrecarga do método CustomResponse para quando precisarmos tratar um erro de validação da ViewModel



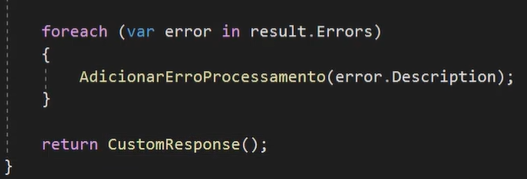
* Na AuthController
  + No método Registrar,
    - Na validação da ModelState implementar o retorno dessa CustomResponse



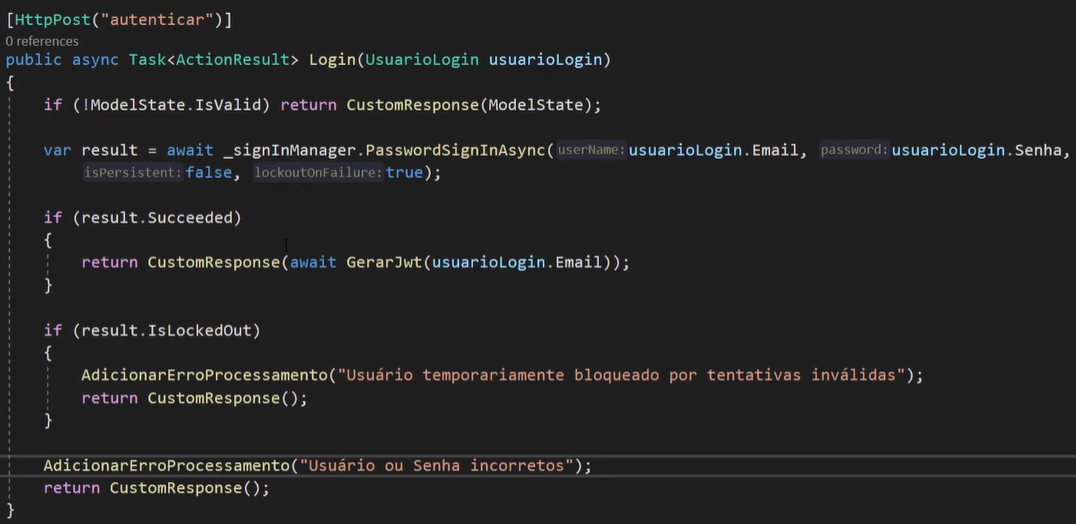
* + - Retirar o SignInAsync, pois não se está fazendo login, apenas gerando um token.
    - No Ok(), implementar o custom response.



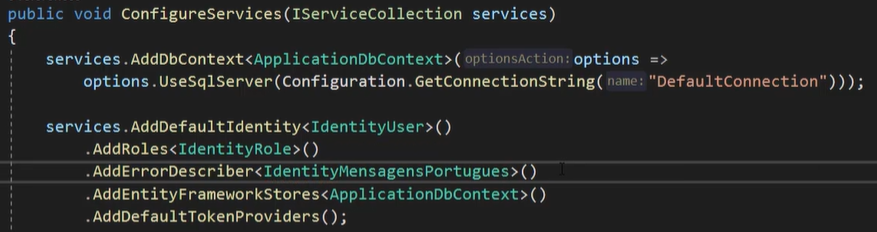
* + - Implementar a varredura de erros para que possam serem listados caso o procedimento venha a falhar.
    - Colocar também o CustomResponse



* + Realizar o mesmo tipo de modificação do método Login
    - Criar uma condição específica para quando o usuário estiver bloqueado.



* Testar e ver como está funcionando
* **Customizando as mensagens do Identity**
* Na pasta extensions da API Identidade, criar uma classe com o nome de IdentityMensagensPortugues
  + Nela estará uma coleção de mensagens de erro do Identity
* No startup, onde existe a configuração do identity, vai e adiciona a referência AddErrorDescriber



* Testar se as mensagens estão vindo em português
* **Refatoração e melhorias**
* Mudar a ConnectionString do do arquivo global para o de desenvolvimento, ou seja, do appsettings.json para o appsettings.Development.json.
* Na Startup configurar para que seja possível identificar qual dos dois ambientes a aplicação irá usar.

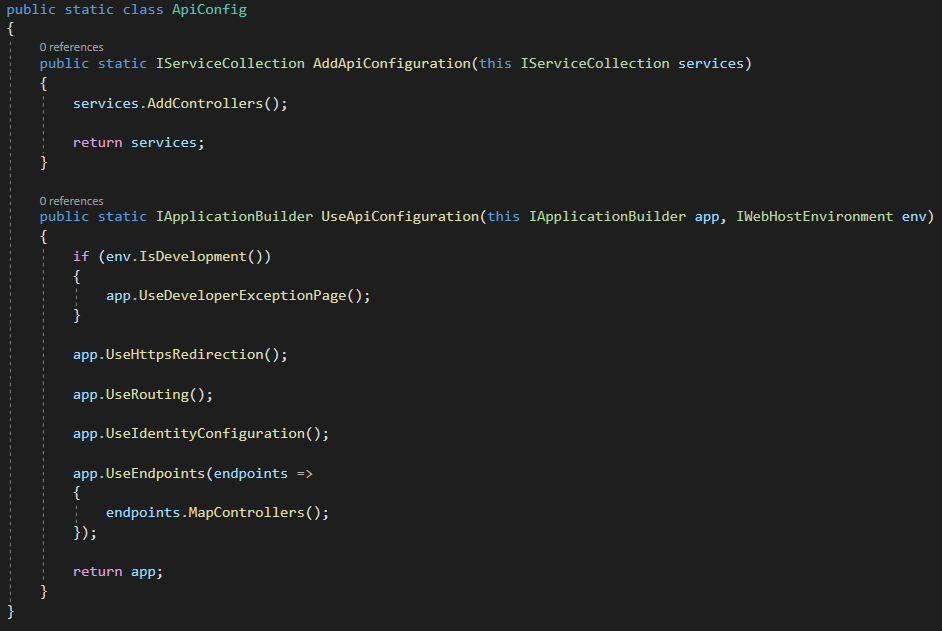


* Para criar o secrets basta clicar com botão direto no NSE.Identidade.API e depois clicar em Manager User Secrets.
  + Pode ter a mesma estrutura do appsettings, porém o arquivo ficará na sua máquina.
* Ajustar o arquivo launchSettings para que seja possível rodar a aplicação em algum modo específico.

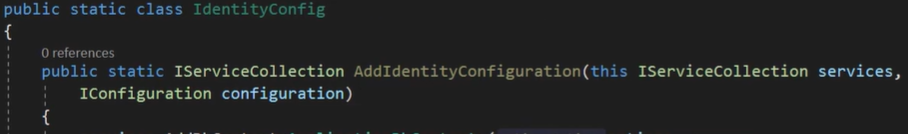


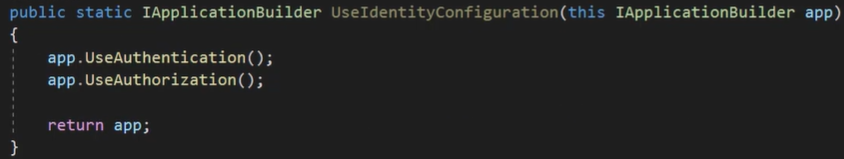


* Criar método de extensão para cada responsabilidade da startup.
* Para isso deve-se primeiramente criar na pasta configuration, classes chamadas ApiConfig, IdentityConfig e SwaggerConfig
* ApiConfig :
  + Lembrando que a configuração do Identity sempre tem de estar entre o UseRouting e o UseEndpoint

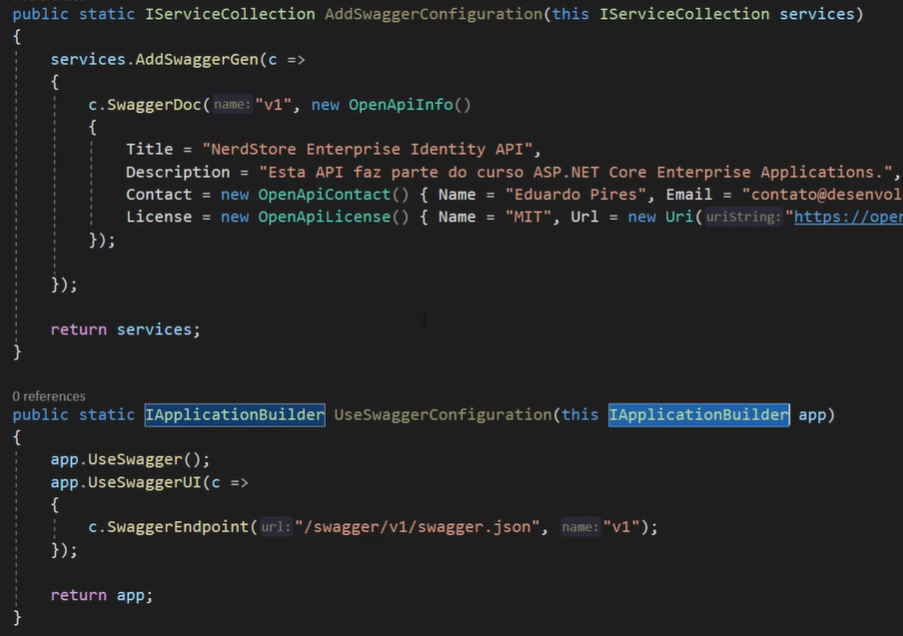


* Realizar o mesmo procedimento para IdentityConfig:

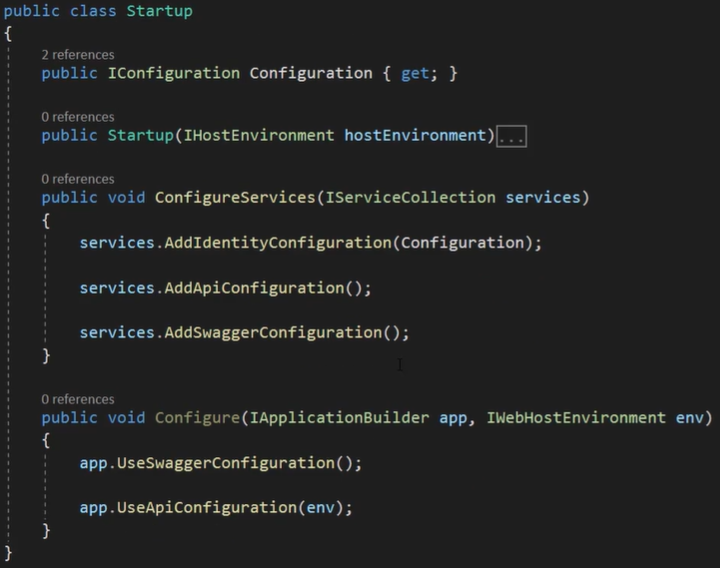




* Para o Swagger também



* No final, a classe startup estará muito mais limpa.



* Testar