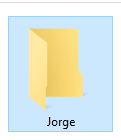
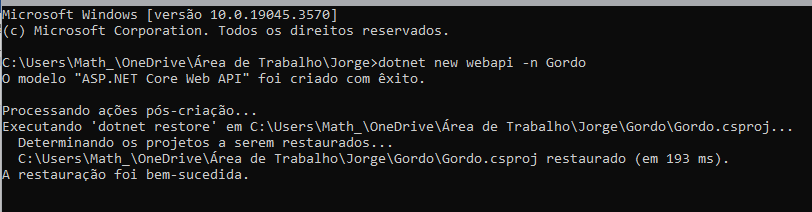
Passo a Passo C# - Angular

1. Crie uma pasta na área de trabalho e abra a pasta que você criou escreva CMD dentro do diretório de pesquisa na pasta



1. Dentro do cmd digite dotnet –version para verificar a versão do dotnet instalada na maquina caso possua mais de uma versão o comando para criar o projeto é (dotnet new webapi -n (O nome do seu projeto) -f net6.0, no exemplo abaixo o nome do meu projeto é “Gordo”).



1. Próximo passo é criar a solução ainda no cmd deve ser digitado um novo comando  
   que é dotnet new sln -n (Nome do arquivo)



1. Se for bem sucedido a temos que adicionar essa solução ao projeto -> aproveitando que já foi criado no mesmo diretório vamos utilizar o seguinte comando CD + nome da pasta projeto nesse caso CD + Gordo



E dentro dessa pasta deve ser adicionado o seguinte comando (dotnet sln ../(o nome da sua solução).sln add (O nome do seu projeto).csproj). ***Exemplo: dotnet sln ../SolutionProj.sln add Gordo.csproj***Caso seja bem sucedido a resposta vai ser essa -> ******

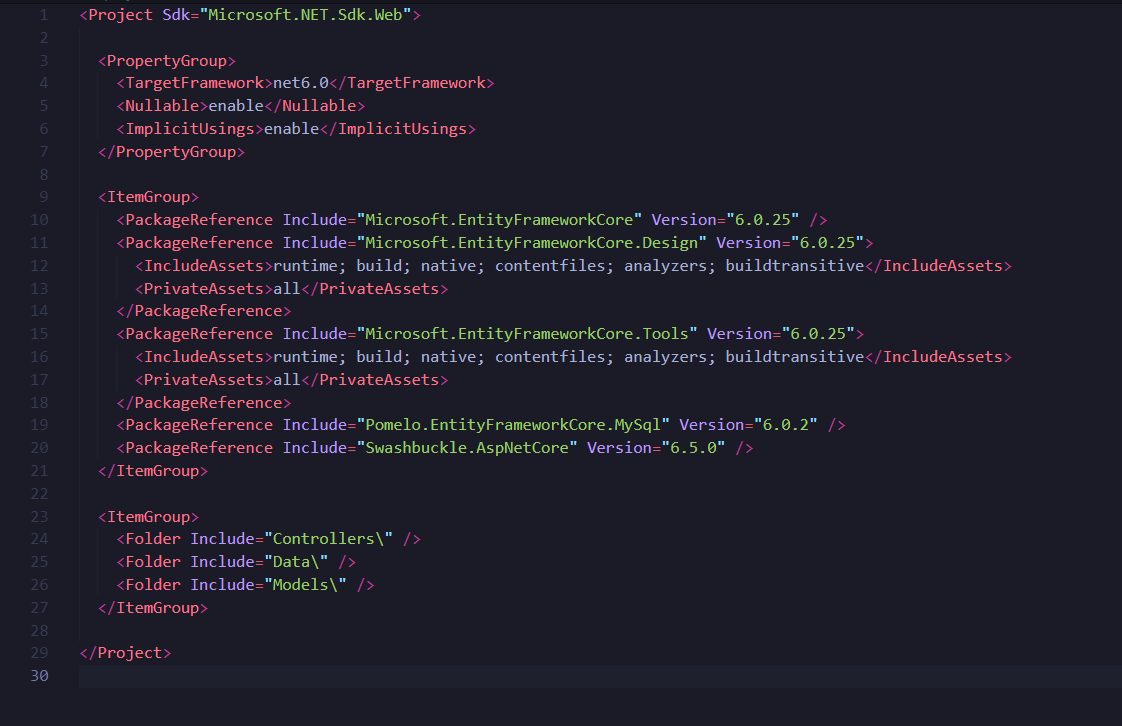
1. Caso tudo isso tenha dado certo E passado já podemos ir até a pasta e abrir com o vsCode, La você deve encontrar a aba de gerenciamento de solução e ela deve estar parecida com isso

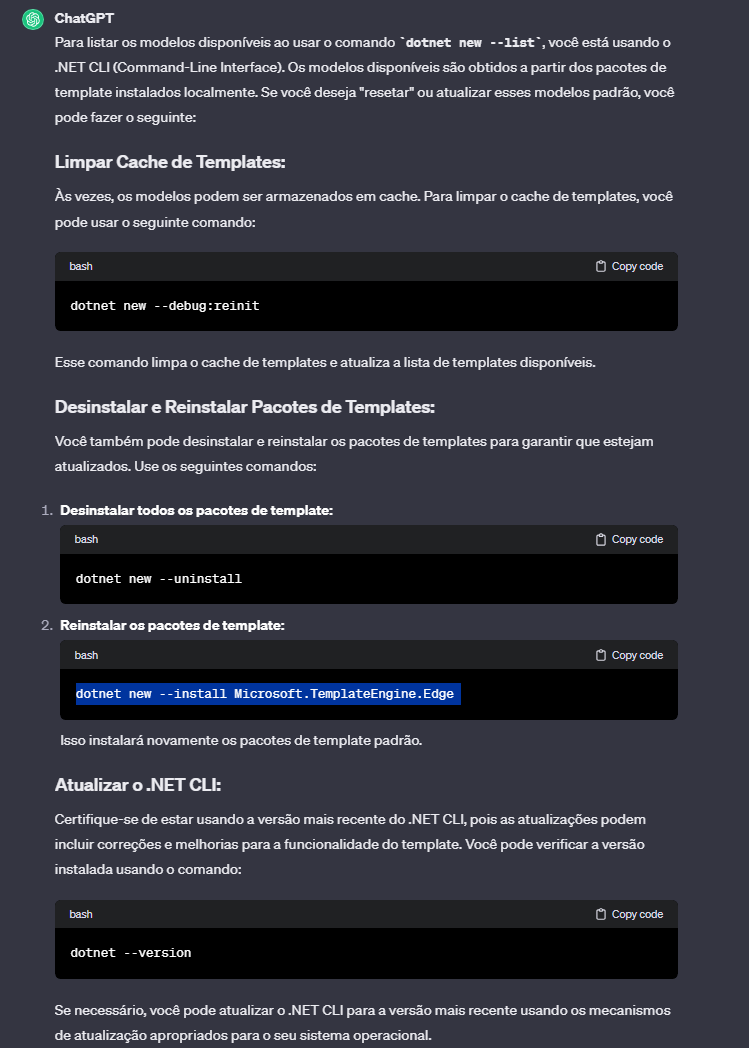


Nessa etapa eu gosto de criar as pastas padrões Models, Controllers, Data e remover os arquivos a mais nesse momento é Weatherforecast.cs e o Weather Controller nesse momento para agilizar o processo já vamos abrir o nuget e pegar os arquivos necessários para utilizar o

* ***dotnet add package Microsoft.EntityFrameworkCore --version 6.0.25***
* ***dotnet add package Microsoft.EntityFrameworkCore.Design --version 6.0.25***
* ***dotnet add package Microsoft.EntityFrameworkCore.Tools --version 6.0.25***
* ***dotnet add package Pomelo.EntityFrameworkCore.MySql --version 6.0.2***

Nesse exemplo o BD que vou utilizar é o MySql -> com Pomelo.Entity Como estamos colocando o Entity framework chegou o momento de começar a projetar as models e os relacionamentos -> existem alguns relacionamentos comuns

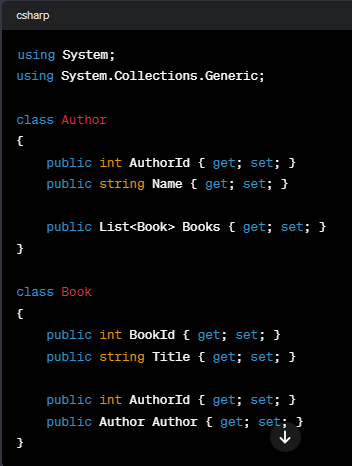
***O arquivo Gordo.csjproj deve estar assim***  
  


***Caso de erro na hora de criar os templates  
  
***

Tipos de Relacionamentos

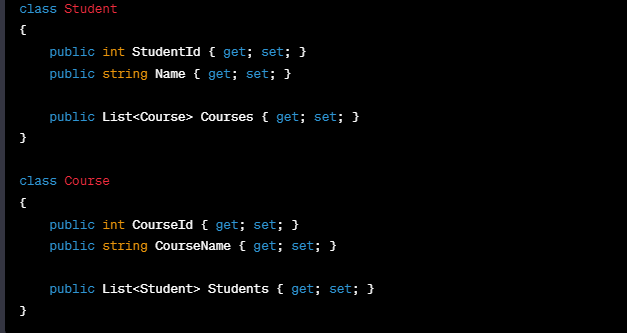
***Um-para-Muitos (One-to-Many)***

Um exemplo clássico seria um relacionamento entre um Author (Autor) e seus Books (Livros), onde um autor pode ter muitos livros, mas um livro tem apenas um autor.

******

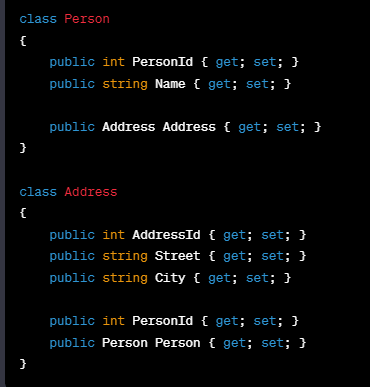
***Muitos-para-Muitos (Many-to-Many):***

Vamos usar um exemplo de um relacionamento entre Students (Alunos) e Courses (Cursos), onde um aluno pode se inscrever em vários cursos e um curso pode ter vários alunos.



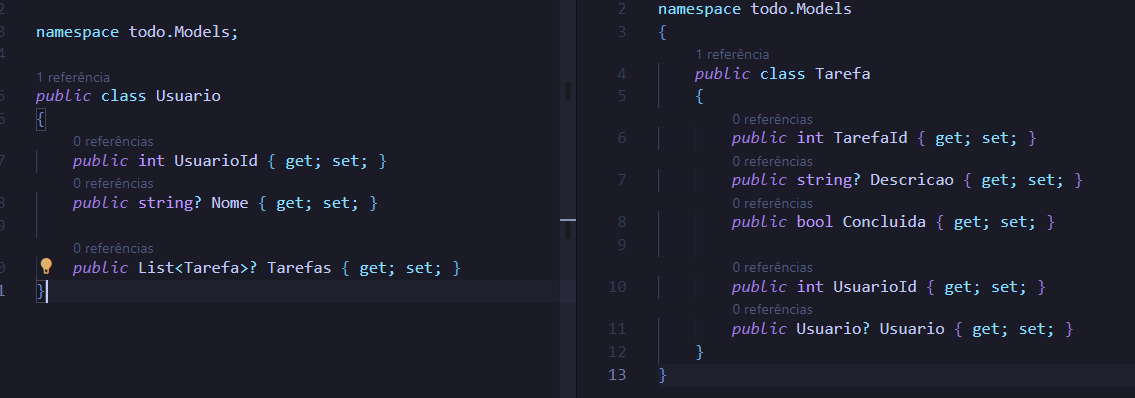
**Um-para-Um (One-to-One):**

Vamos considerar um relacionamento entre um Person (Pessoa) e seu Address (Endereço), onde uma pessoa tem um endereço e um endereço pertence a uma única pessoa.



Bem nesse exemplo eu vou fazer um To-do list começando pela MODEL vou criar uma Model para Tarefa e outra para Usuário.

O Relacionamento é Um para muitos: Um usuário pode ter várias tarefas.

***Após criar as models   
  
***

Explicação sobre as notações

**Atributos Key:**

**TarefaId** e **UsuarioId** são anotados com [Key]. Isso indica que essas propriedades são chaves primárias nas tabelas correspondentes no banco de dados. Cada entidade no Entity Framework Core precisa ter uma propriedade anotada com [Key] para representar a chave primária.

Atributos **Required**:

**Descricao** e Nome são anotados com **[Required].** Isso indica que essas propriedades não podem ser nulas no banco de dados. Essa anotação é usada para garantir que esses campos sejam sempre preenchidos.

Atributos **ForeignKey** e **Navigation Property:**

**UsuarioId** na classe Tarefa é anotado com **[ForeignKey("Usuario")].** Isso indica que **UsuarioId** é uma chave estrangeira que se refere à chave primária **UsuarioId** na tabela **Usuario**. Essa anotação estabelece um relacionamento entre as tabelas **Tarefa** e **Usuario**.

**Usuario** na classe Tarefa é uma **Navigation Property**, que representa a parte "não chave" do relacionamento. Isso permite navegar da Tarefa para o **Usuario**. A propriedade de navegação é anotada com ? indicando que é opcional (pode ser nula).

Tarefas na classe **Usuario** é uma **Navigation Property** que representa a coleção de tarefas associadas a um usuário. A propriedade de navegação é anotada com ? indicando que é opcional (pode ser nula).

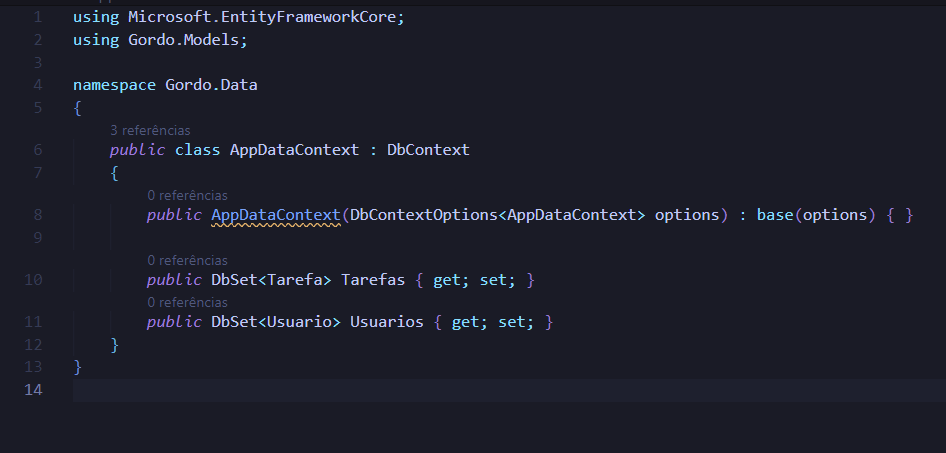
**Atributo Column**:

Você não está utilizando explicitamente, mas você poderia usar **[Column("NomeUsuario")]** para especificar o nome da coluna no banco de dados se você quiser que seja diferente do nome da propriedade em C#.

Config da Data e continuação do código

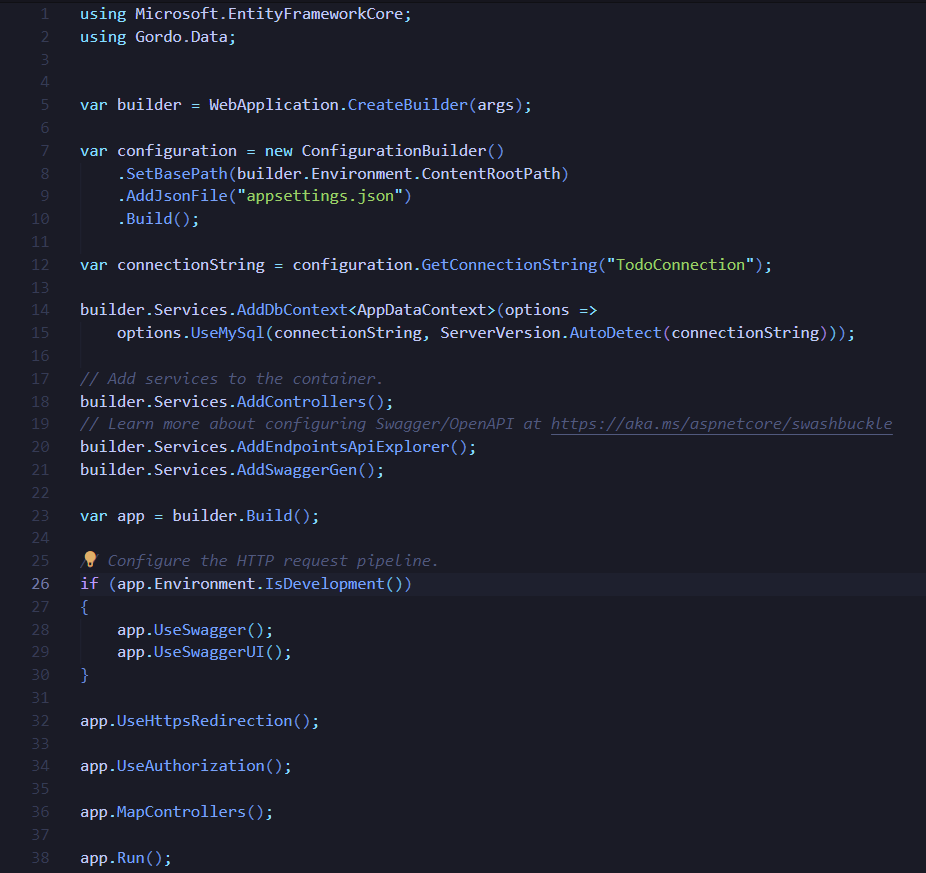
***Devemos ir até a pasta data criada e começar a configurar o banco e a sua conexão***

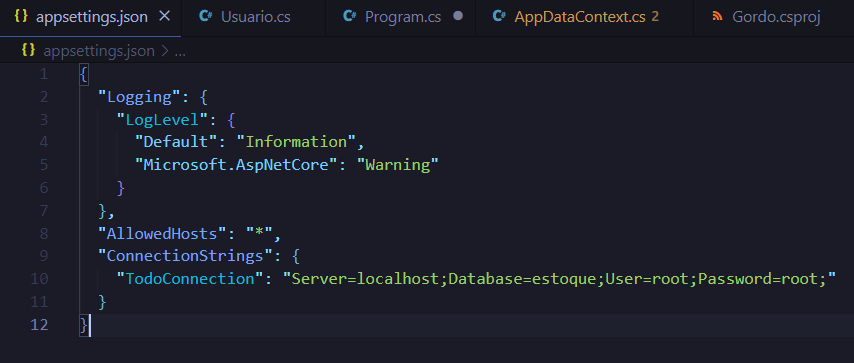
***Devemos criar na data a “classe” AppDataContext.cs***

******

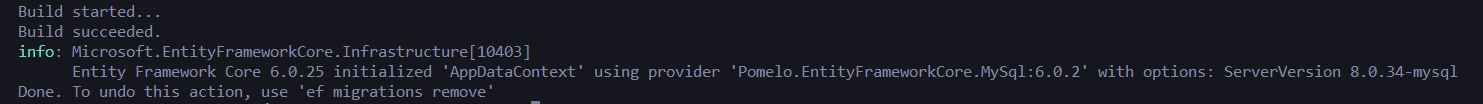
O DbSet é utilizado para salvar isso como nome da tabela, ou seja, o meu conjunto de Tarefa Vira Tarefas e o meu Conjunto de Usuário vira Usuários no bd.

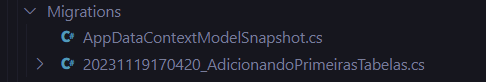
A program.cs deve estar com assim para que o MySql funcione



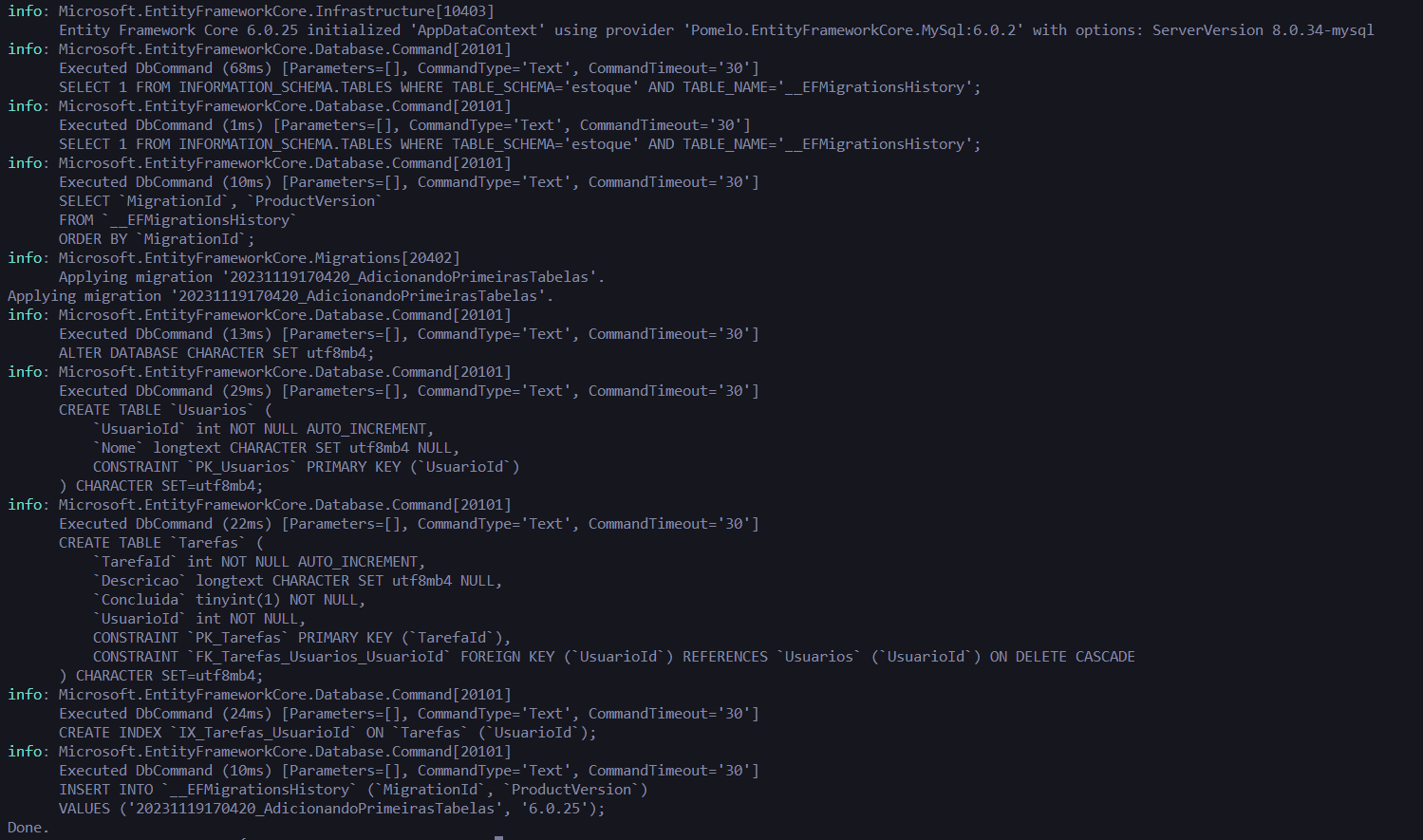
E Appsettings.json deve estar assim   
  


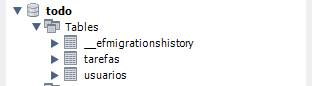
Após configurar a Data é necessário criar uma pasta migrations para que seja alocadas o “Designer” Do banco de dados   
  
Após criar a pasta deve se abrir o terminal dentro do Visual Studio e rodar o codigo com o nome da migração no meu caso vai ser   
  
AdicionandoPrimeirasTabelas então o meu comando vai ficar assim  
  
Exemplo: dotnet ef migrations add AdicionandoPrimeirasTabelas



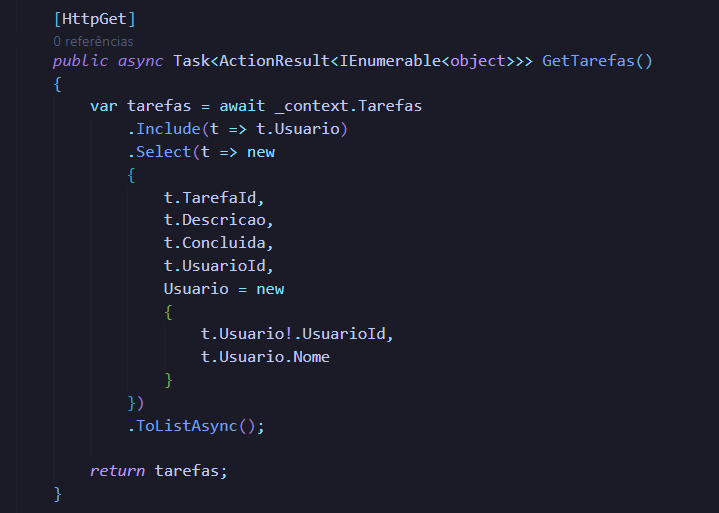
Se for bem sucedido deve voltar essa mensagem na sua tela e a migration recém criada vai estar assim  
  


O próximo comando vai ser o responsável por “UPAR O BANCO”   
  
O comando é: ***Dotnet ef database update***. Após utilizar ele vai aparecer as informações de criação e de referencias no seu banco seguindo a lógica da sua model.



Nesse momento você pode ir conferir o seu banco e ele deve estar similar a isso  
  


Agora que já possuímos as models e as tabelas respectivas no banco devemos povoar A Controller para que ela consiga fazer as validações necessárias no backend para que seja possível salvar as informações de entrada  
  
Vamos lá na Pasta Controller para criar a Controller Tarefa e Usuario e povoar ela com o CRUD padrão   
  
POST, GET, UPDATE and DETELE.

Como foi feito uma relação de 1 para muitos   
  
Onde um usuário pode ter varias tarefas pedi para o chat montar um CRUD que já possua o relacionamento   
  
Vou upar eles no git e deixar alguns prints com coisas importantes  
  


***GET (Listar todas as Tarefas):***

***Endpoint: GET /api/Tarefa***

***Descrição: Retorna uma lista de todas as tarefas registradas no sistema, incluindo informações básicas sobre cada tarefa e detalhes do usuário associado.***

***Ação: Realiza uma consulta ao banco de dados para recuperar todas as tarefas. Utiliza a biblioteca Entity Framework para incluir informações sobre o usuário associado a cada tarefa. Mapeia os dados para um formato específico (ID, Descrição, Concluída, ID do Usuário e informações do Usuário).***

***Retorna a lista formatada como um objeto JSON contendo todas as tarefas.***

***GET (Detalhes de uma Tarefa específica):***

***Endpoint: GET /api/Tarefa/{id}***

***Descrição: Retorna informações detalhadas sobre uma tarefa específica, incluindo detalhes do usuário associado.***

***Ação: Localiza a tarefa no banco de dados com base no ID fornecido. Utiliza o Entity Framework para incluir informações sobre o usuário associado à tarefa. Mapeia os dados para um formato específico (ID, Descrição, Concluída, ID do Usuário e informações do Usuário). Retorna os detalhes da tarefa como um objeto JSON.***

***POST (Criar uma nova Tarefa):***

***Endpoint: POST /api/Tarefa***

***Descrição: Cria uma nova tarefa no sistema com base nos dados fornecidos, associando-a a um usuário existente.***

***Ação: Valida se os dados fornecidos para a nova tarefa são válidos.Adiciona a nova tarefa ao banco de dados.Recupera o nome do usuário associado ao ID do usuário fornecido. Retorna os detalhes da tarefa recém-criada, incluindo o nome do usuário associado.***

***PUT (Atualizar uma Tarefa existente):***

***Endpoint: PUT /api/Tarefa/{id}***

***Descrição: Atualiza os detalhes de uma tarefa existente com base nos dados fornecidos.***

***Ação: Valida se a tarefa com o ID fornecido existe no banco de dados. Atualiza as propriedades da tarefa com base nos dados fornecidos. Salva as alterações no banco de dados. Recupera os detalhes da tarefa atualizada, incluindo informações do usuário associado. Retorna os detalhes da tarefa atualizada como um objeto JSON.***

***DELETE (Excluir uma Tarefa):***

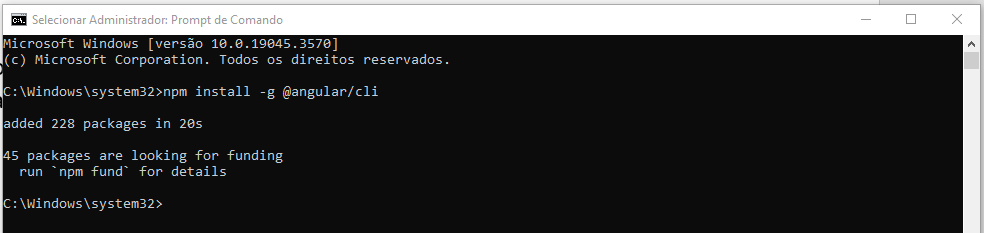
***Endpoint: DELETE /api/Tarefa/{id}***

***Descrição: Remove uma tarefa específica do sistema.***

***Ação: Valida se a tarefa com o ID fornecido existe no banco de dados. Remove a tarefa do banco de dados. Retorna uma resposta sem conteúdo (No Content) indicando que a tarefa foi removida com sucesso.***

Instalando o Angular -> Começando a aplicação front end

comando esse que instala o angular no seu computador (npm install -g @angular/cli) esse comando instalará o angular diretamente no seu pc, deve aparecer essa mensagem de instalação do angular.

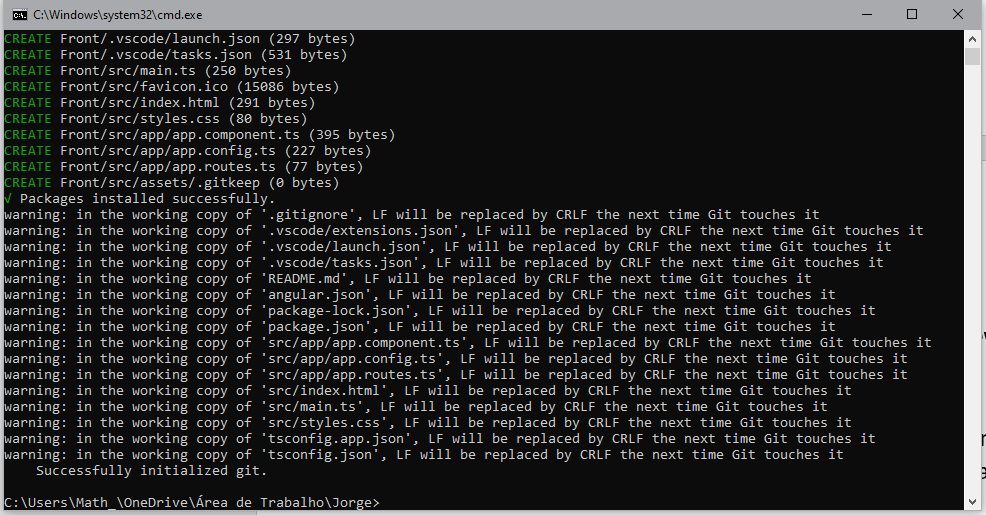


A imagem que vai aparecer é essa daqui -> Com isso podemos seguir até a pasta que foi criada lá no passo 1

Após rodar o cmd devemos digitar o comando: **ng new (nome – Front) –minimal.**

Após rodar esse comando vai aparecer um pequeno menu pedindo para especificar qual vai ser a folha de estilo utilizada. Escolha CSS e diga não para a próxima pergunta feita.

Ele deve dar a seguinte resposta



Agora iremos arrumar um grande fator e bem extenso sobre os documentos, então farei de uma forma um por um primeiramente começando pelo angular.json, abra o sua pasta front no VsCode, e abra o arquivo angular.json, vamos arrumar 2 variáveis deste documento a variavel “inlineTemplate” e a variável "inlineStyle" as duas estarão true troque para false isso ajudará no futuro .

