* Definindo as entidades

-> Criar a pasta Models

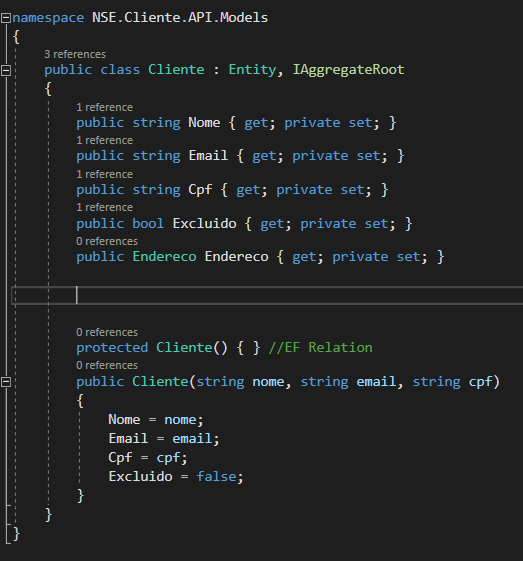
-> Criar e modelar a classe de clientes

->Herdar de Entity e de IAggregateRoot

-> Configurar a relação dela com o Entity e com a classe de

Endereços

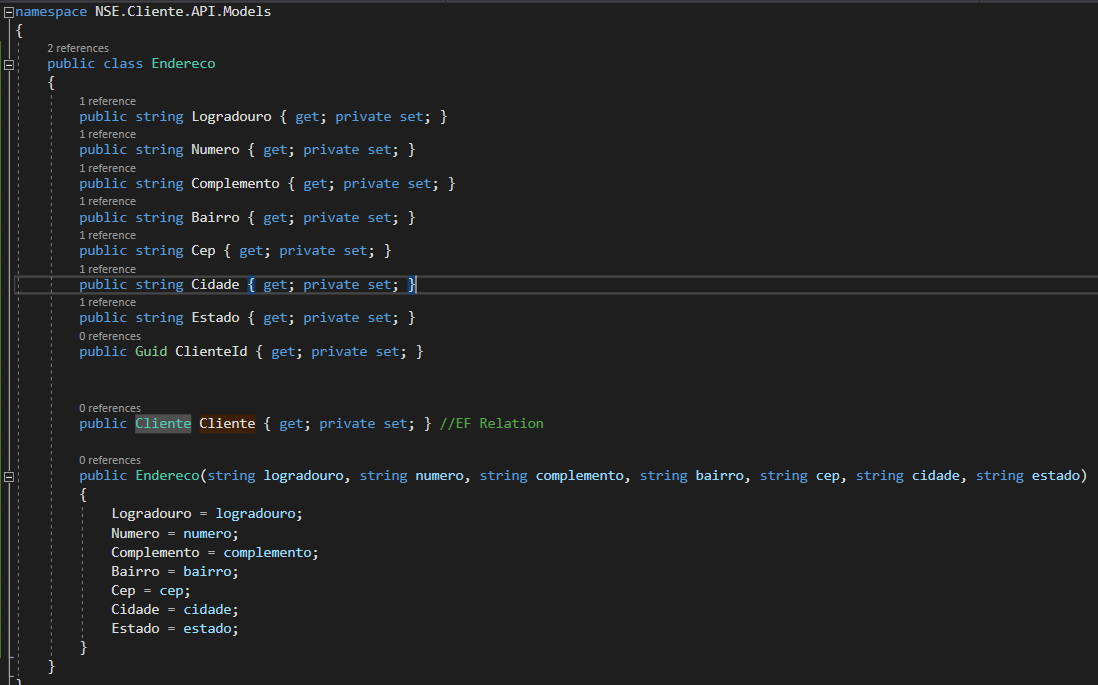
-> Configurar construtor da própria classe como também o que será usado apenas como relação com o Entity.



-> Criar e modelar a classe de Endereço

-> Configurar a relação dela com classe de clientes

-> Configurar construtor



* Objetos de valor e validações

-> No building blocks/core/NSE.Core/DomainsObjects, criar as classes cpf e email para que sempre seja possível verificar essas duas variáveis

-> No objeto de valor do CPF:

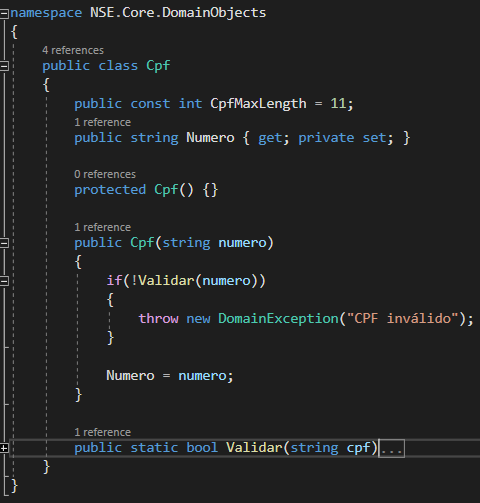
-> Verificar o seu tamanho máximo e se contêm apenas números.

-> Construtor da classe

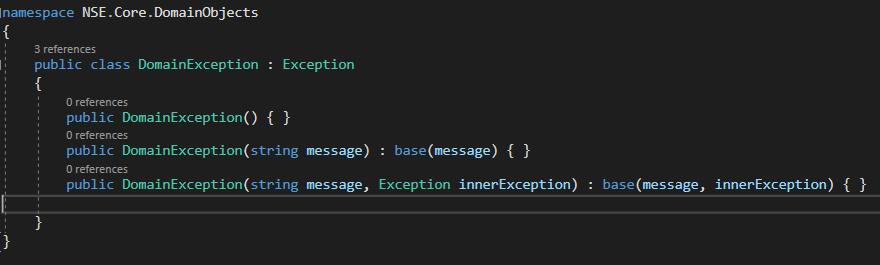
-> Construtor do Entity

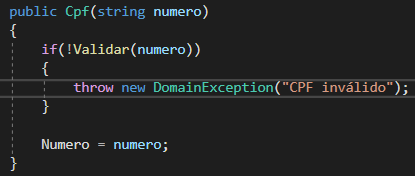
-> Caso esse construtor não seja feito dentro do objeto de valor, na hora que isso for mapeado para o banco e for materializado da consulta do banco para o objeto, o Entity Framework não saberá construir.

-> Criar um código de validação de CPF

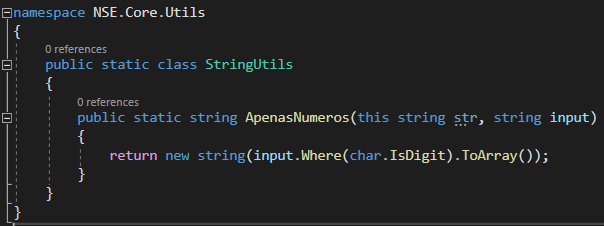


-> Não permitir que a entidade seja montada de forma inconsistente. Para isso deve-se criar uma classe DomainException e utilizá-la no construtor.





-> Dentro da NSE.Core, criar uma pasta Utils e nela criar uma classe estática que retorne apenas os números da string CPF.



-> No objeto de valor Email

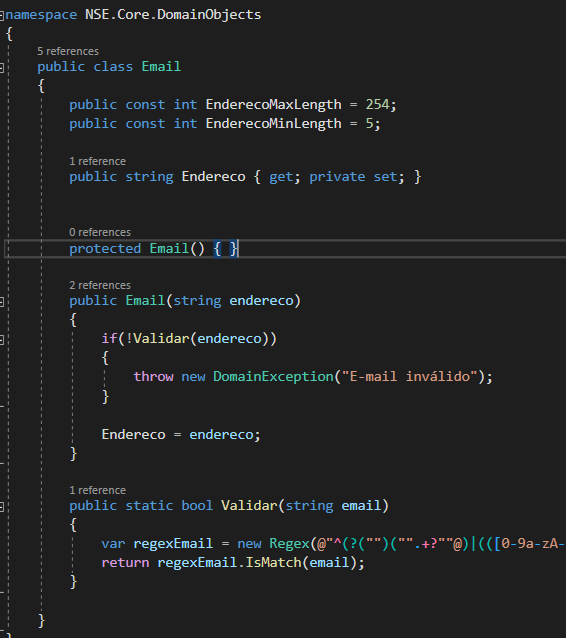
-> Verificar seu tamanho máximo e mínimo

-> Construtor da classe

-> Retornar uma exception caso o email não seja válido.

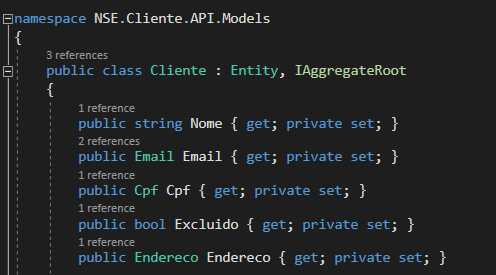
-> Construtor do Entity

-> Método de validação do email.

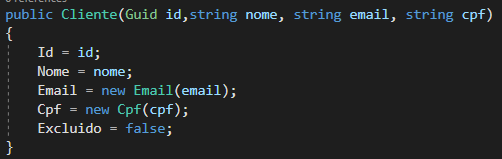


-> Na api de cliente, na model Cliente:

->Mudar as variáveis Cpf e Email do tipo string para o tipo Cpf e Email respectivamente.



-> No construtor, para o email e para o cpf, deve criar uma nova instância para ambos. Adicionar um guid id no construtor do Cliente. Alimentar o id dentro do construtor.



->Criar métodos de trocar e-mail e atribuir endereço

