**Set up da aplicação 1**

* Criar as pastas sql, src e tests.
* Criar o arquivo de configuração global.json para definir qual versão do sdk irá utilizar. Para saber qual versão utilizar, verificar no cmd através do comando “dotnet - -info”. O base path aparecerá com o caminho que contem todas as versões.
* Criar as três camadas necessárias
* Camada App:

- Template: ASP.NET Core Web Application. (MVC)

- Authentication: Individual User Accounts

* Camada Business:

- Template: Class Library (.NET Core)

- Pastas: Interfaces,Models, Validations, Notificacoes e Services

* Camada Data:

- Template: Class Library (.NET Core)

- Pastas: Context, Repository, Migrations e Mappings

* Na criação das models, lembrar que é preciso estabelecer um relacionamento visível com o Entity Framework. O Entity precisa entender, via declaração de propriedade, as relações entre as models.

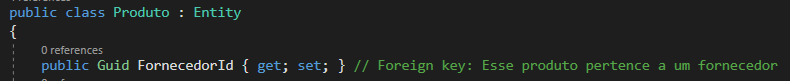
- Relação de um para muitos.



- Relação de um para um



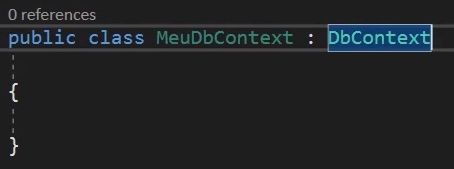
* É necessário delimitar a Foreign Key nas models que tem de pertencer a outra.



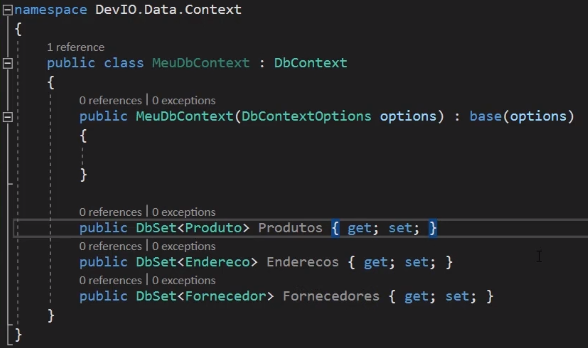
**Mapeando as entidades para o banco de dados**

RESUMO: Configurar as entidades do banco, mapear e criar o banco.

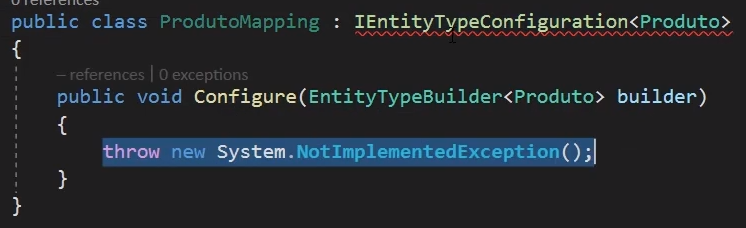
* Para a classe do MeuDbContext.cs sempre deve-se instalar o package do Entity Framework Core na camada correta. Usar o comando “Install-Package”.
* Sempre relacionar a classe com o DbContext



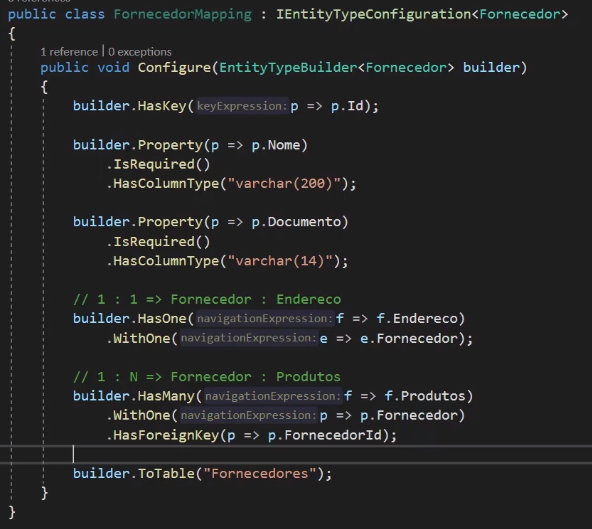
* No seu construtor deve-se receber o DbContextOptions e depois repassar para a classe base
* Mapear os DbSets
* Lembrar de adicionar a referência com a camada de negócios.



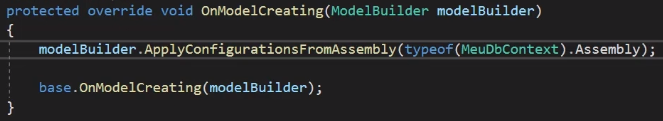
* Não configurar as entidades direto no banco sem informação alguma. Para isso deve-se primeiro mapear tamanho de coluna, tipo de campo e etc. Para realizar tal ação num modelo de três camadas, a utilização da ferramenta fluent api é recomenda. Um dos pontos positivos disso é não poluir suas entidades com data notations.
* Dentro de cada mapping (ProdutoMapping, FornecedorMapping, etc) as devidas configurações serão feitas.
* Cada mapping deve herdar de IEntityTypeConfiguration<> . Depois de herdado deve ser implementada a interface.



* É a classe que possui as outras que configura o mapeamento. Por isso configuramos dentro de FornecedorMapping. Nessa configuração, para que use a propriedade HasColumnType, é necessário que se instale o pacote Microsoft.EntityFrameworkCore.Relational.



* Depois desse processo de configuração ainda é preciso realizar o mapeamento no DbContext.
* O método OnModelCreating será chamado durante a criação do modelo no banco.



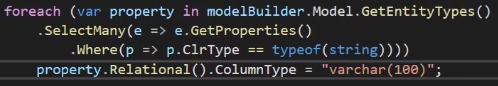
* Ao aplicar o método modelBuilder, ocorrerá uma busca de todas as entidades que estão mapeadas nesse DbContext. Encontradas, vai buscar classes que herdem de IEntityTypeConfiguration para as entidades que estejam relacionadas no DbContext e vai registrá-las de uma vez só.



* Não é recomendado deixar as classes no banco de dados com o delete cascade ativado. Para isso pode desabilitar tal ação com o seguinte código.



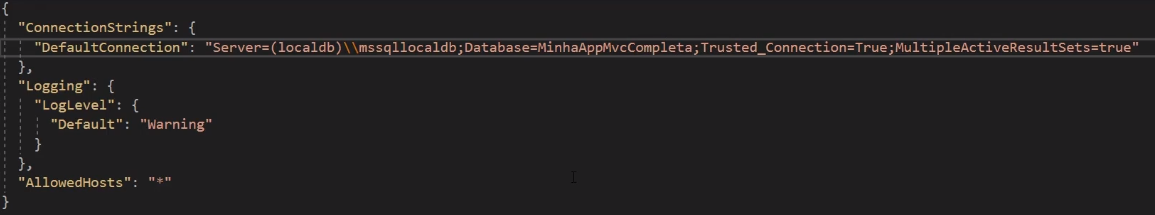
* Para evitar situações em que se esquece de mapear alguma coluna e ela seja criada como nvarchar (max) , recomenda-se usar o código abaixo.



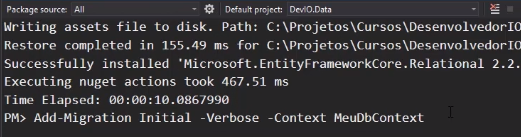
* Depois de configurado o DbContext, deve-se configurá-lo na aplicação. Primeiro passo, realizar configuração de dependência. Segundo passo, ir no ConfigureServices da StartUp e realização a configurações abaixo.



* Ir no arquivo appsentings.json para mudar o nome do banco de dados para um mais fácil



* Depois de tudo configurado deve-se criar uma migration. Lembrar que a aplicação possui dois contextos. A migration pode ser aplicada tanto para um quanto para outro. Nesse caso, foi aplicada para MeuDbContext.

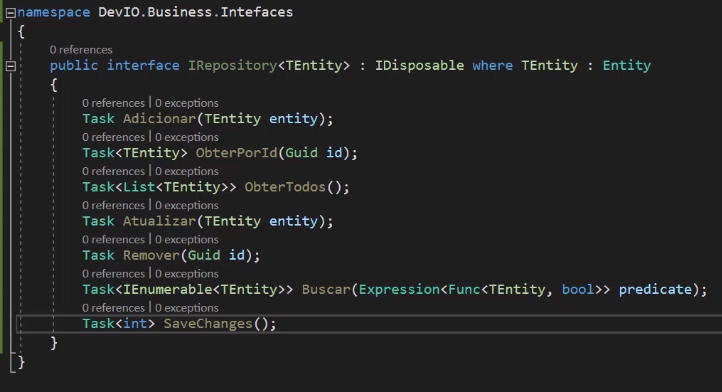


* Caso queira gerar um script migration, deve-se primeiro instalar o pacote Microsoft.EntityFrameworkCore.SqlServer. Depois basta gerar o script no package manager console através do código “Script-Migration –Context MeuDbContext”.
* Para atualizar o banco de dados, basta utilizar o comando Update-Database – Context ApplicationDbContext. Atualiza primeiro um contexto depois o outro.

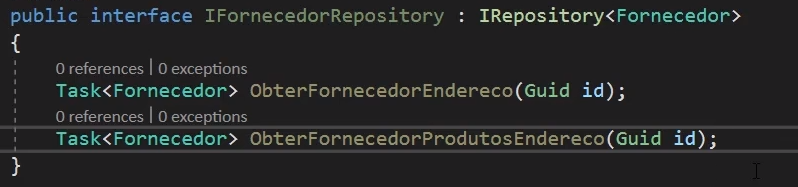
**Acessando o banco via repositórios**

Resumo: Facilitar para que a camada de negócios e a camada de apresentação possuam um meio de acesso para conversar com o banco sem que seja diretamente pelo DbContext.

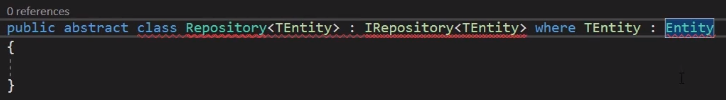
* Neste modelo não é adequado injetar o DbContext diretamente na controllers.
* Para que esse acesso seja viabilizado, se utiliza o pattern repository.
* Primeiro se cria uma interface genérica para que possa oferecer todos os métodos necessários para qualquer entidade.
* Nessa interface será criado os métodos que se deseja usar no banco. Adicionar, remover, obter e etc.
* A implementação da interface IDisposable obriga a interface a fazer o dispose, ou seja, liberar memória.
* A interface só poderá ser utilizada caso seja filha da Entity.
* A implementação do método buscar permite passar uma expressão lambda dentro do método para que se busque qualquer entidade por qualquer parâmetro.



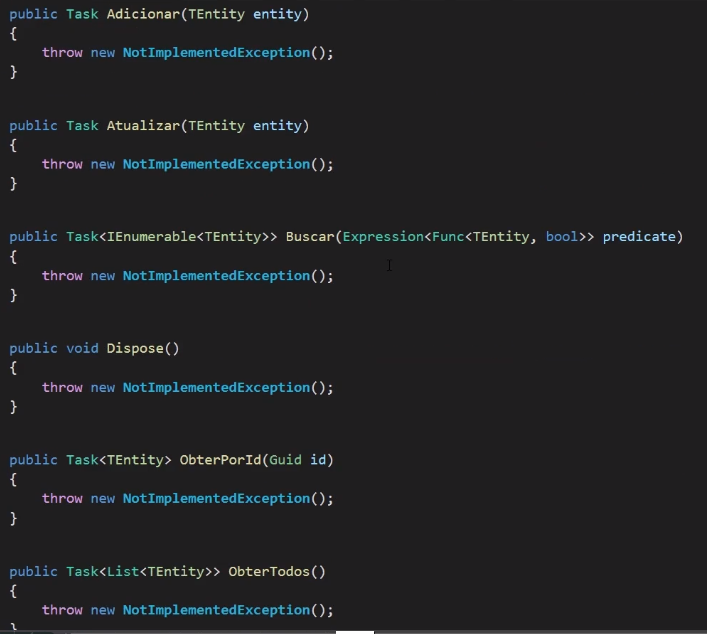
* Sem essa interface a camada de negócios não seria capaz de se comunicar com a camada de acesso a dados.
* Dentro de cada Interface (Produto,Fornecedor e Endereço) todos os métodos da interface genérica já estarão disponíveis devido a sua herança de IRepository. Caso queira, pode-se adicionar métodos específicos para ajudar.



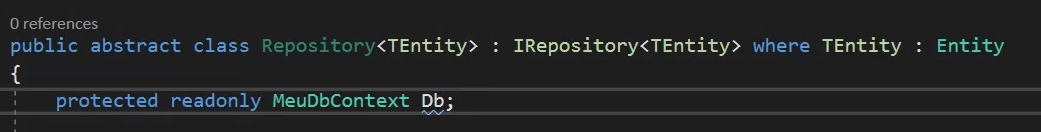
* Uma vez feitas todas as interfaces, pode dar início a criação das classes concretas na pasta repository localizado na camada de dados.
* A classe repository será uma classe genérica, ou seja, só pode ser herdada. Não pode ser instanciada. Nesse caso a palavra abstract deve ser escrita para descreve-la.
* Lembra de implementar o parâmetro genérico TEntity.
* Implementar a interface IRepository de TEntity. Sendo que esse Entity é diferente do Entity anterior. A TEntity atual deve ser filha de Entity e por isso deve ser descrito na criação da classe.



* Uma vez que um contrato é implementado, os seus métodos devem ser implementados.

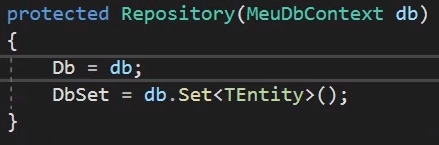


* Para ter acesso ao contexto deve-se declarar o context. Deve ser declaro com protected pois tanto o repository quanto quem herdar dele poderão ter acesso ao DbContext.



* Criando um atalho para o DbSet evita-se que o código fique muito verboso. Para que esse atalho seja criado deve-se também injetar o DbContext no construtor.



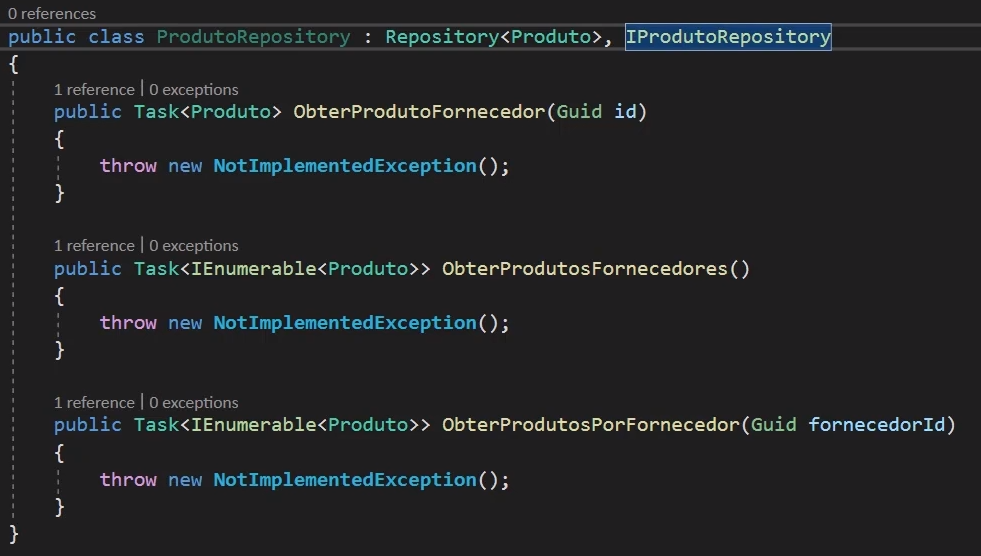


* Com esse atalho a seguindo substituição pode ser feita.





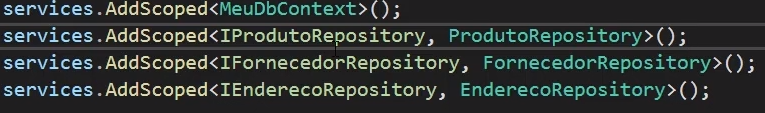
* Transformando os métodos em “virtual”, pode se fazer o override quando for necessário.
* Uma vez que o repository esteja devidamente configurado, deve ser herdado pelos demais. Além disso as interfaces também têm de ser implementadas.



* Lembrar que como a classe base é abstrata não se pode criar a instância dela. Como nesse caso recebe algo no construtor, alguma outra classe terá que passar para ela. Então no ProdutoRepository será feito um construtor e dentro dele será recebido o MeuDbContext que será passado para classe base.

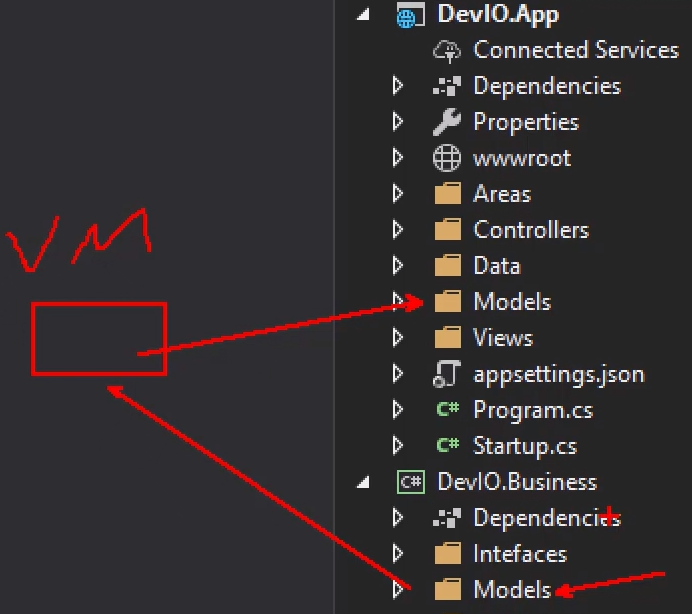


* Depois de feitas todas as consultas, deve-se fazer a resolução da injeção de dependência no configure services da classe start up.

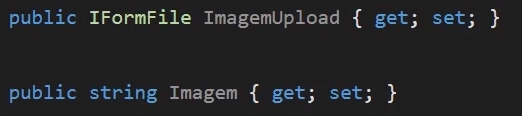


**Mapeando Entidades em View Models com Automapper**

* A exibição dos modelos de Negócios na view do MVC nunca deve ser feita de maneira direta. Para isso deve-se primeiro replicar os modelos para um modelo intermediário, chamado de view model, que será representado na pasta models.
* View models também podem ser chamadas de Data Transfer Object (DTO).



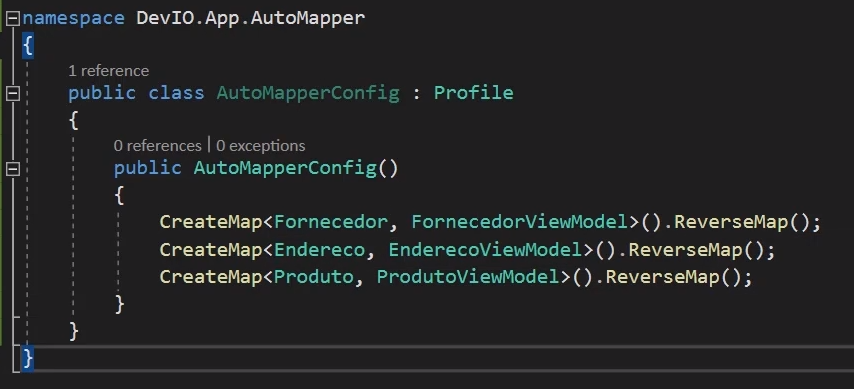
* Configurar as ViewModels. Delimitar se o campo é required, definir string máximo e mínimo, campo chave e etc.
* No caso específico da configuração do upload de arquivo, o campo imagem não pode ser uma string. Para isso duplica-se o campo Imagem. “Imagem” continua a mapear o banco de dados, enquanto “ImagemUpload” define as propriedades do arquivo.



* Para fazer a transformação de view model para entidade e de entidade pra view model usar uma ferramenta chamada de auto mapper. Deve ser instalado pelo package manager console. Lembrar que a instalação de ser feita adicionando a extensão ASP.Net “automapper.extensions.microsoft.dependencyinjection” na camada de apresentação do MVC.
* Configurar o automapper no configureservices da start up. Nessa configuração tem de escrever qual assembly irá usar.



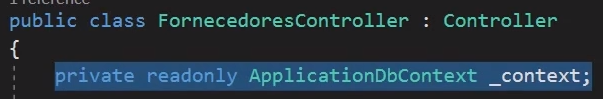
* Criar uma classe de configuração de um perfil de mapeamento na camada do MVC. Dentro dessa classe deve ser criado um construtor para realizar o mapeamento.
* Caso dentro de uma das classes que estão sendo mapeadas(Fornecedor, Produto etc) tenha um construtor ou parâmetro, é necessário fazer dois mapeamentos. Um em um sentido e outro no outro sentido.
* Através da configuração no start up o automapper entende que deve procurar qualquer classe que tenha profile como herança dentro do assembly (DevIO.App).
* A injeção de dependência deve também ser feito nas controllers.

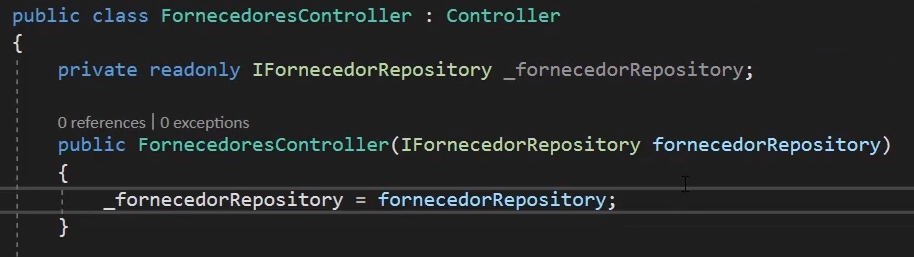


**Scaffolding das Controllers e Views**

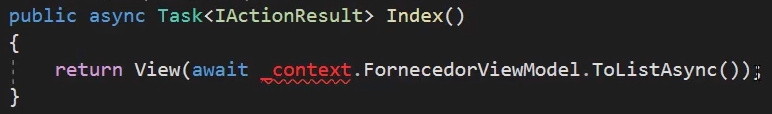
Resumo: Criar as view, que são as telas da aplicação, através do processo de scaffolding. As view serão baseadas nas view models, que são os mapeamentos das entidades de negócios.

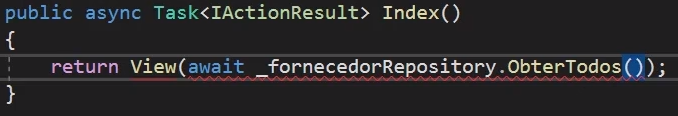
* Lembrar de comentar o ImagemUpload para que o processo de scaffolding consiga ser feito
* Retirar o DbSet do ApplicationDbContext(app/data) para melhorar a controller.
* Retirando o DbSet do ApplicationDbContext a controller irá quebrar. Para consertá-la tem de substituir o contexto do EntityFramework pelo repositório atual e injetá-lo via construtor.



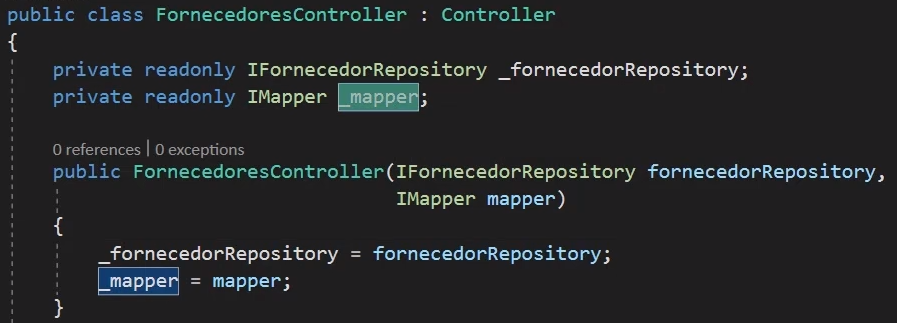


* Com esse fornecedorRepository um meio de acesso a dados é feito. Com isso deve-se então trocar os returns das Task que se utilizavam do \_context.

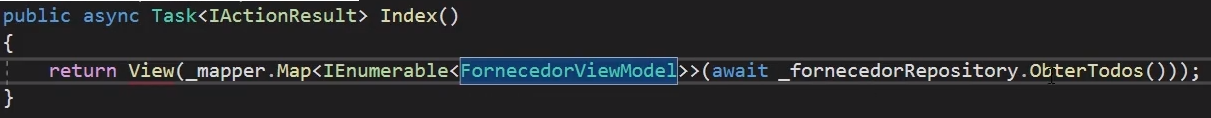




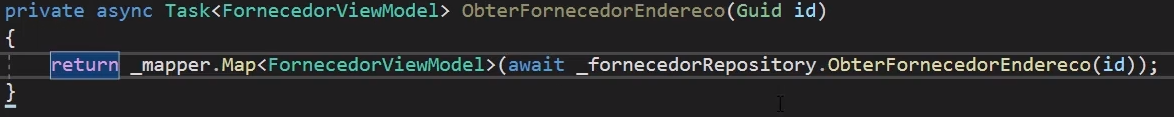
* Verificando a index de Fornecedores da View, percebe-se que ela tem uma lista de FornecedorViewModel que precisa ser convertida. Para isso, primeiro deve implementar o mapper e depois injetar no construtor.

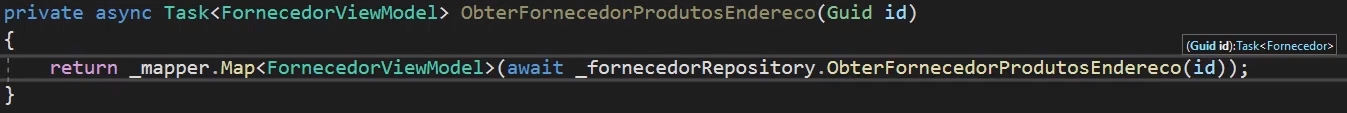


* Para converter basta passar o parâmetro do que você quer, que nesse caso é o IEnumerables<FornecedorViewModel>, e dentro dos parâmetros desse método você passa a fonte da informação.

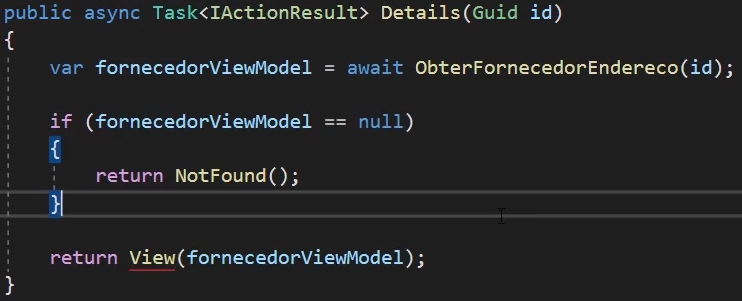


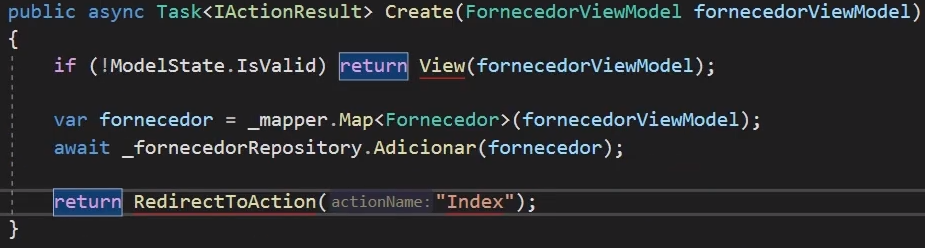
* Esses dois métodos foram criados devido a sua ampla repetição dentre os procedimentos de create, edit e delete.



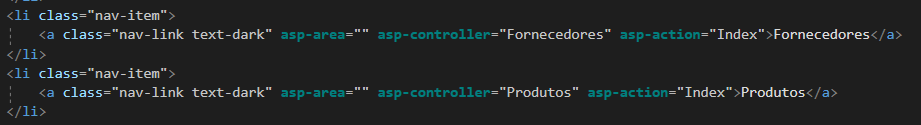


* Com os métodos já criados, pode começar a realizar a alterações nos procedimentos.



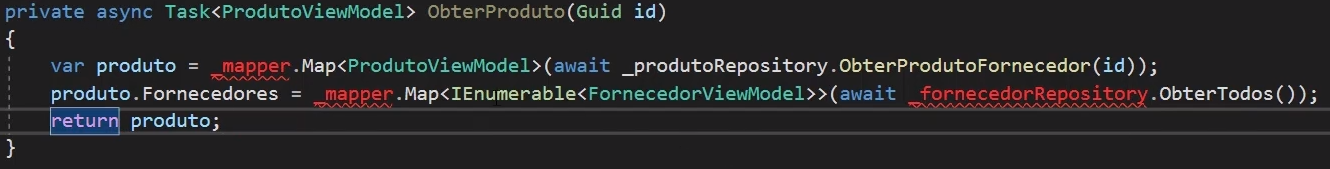


* Lembrar de criar a referência de layout page



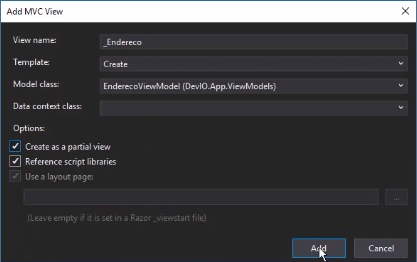
* Com esse controller devidamente criada e listada na layout page, deve-se agora criar as outras controllers com base na ProdutoViewModel.
* Com a controller criada, deve-se então realizar as alterações assim como na controller anterior.
* Para que seja possível ter uma lista de fornecedores que carregue a lista da controller para popular o drop down, deve-se primeiro criar um IEnumerable na ViewModel.





**Customização das Views**

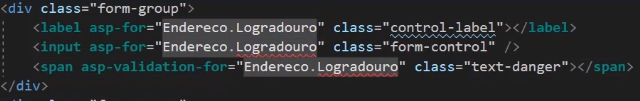
* Resumo: Customizar as views que foram criadas através do processo de scaffolding
* A primeira etapa será cuidar do cadastro de fornecedores. Para isso deve ser criada uma nova partial view na view de fornecedores.



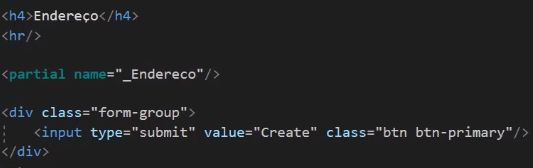
* As divs de formulários serão as partes necessárias
* Mudar na parte superior a viewmodel que será utilizada



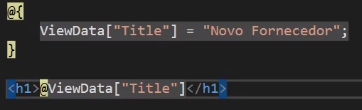
* Os dados de endereço precisam vir da classe fornecedor. Endereço é uma classe dentro de fornecedor.



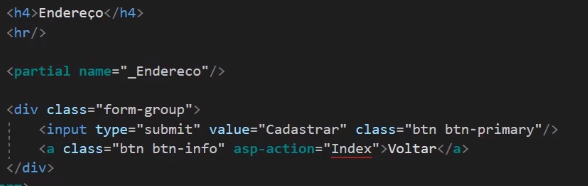
* Remover os dados que não são necessários na criação do endereço. Id, validação e id do fornecedor.
* Reorganizar a informações de modo que sejam disponibilizadas numa ordem coerente. Ex: Nesse caso o CEP virá primeiro para que seja o primeiro campo no formulário de endereço.
* Na view/fornecedores/create já pode colocar a chamada da partial view



* Lembrar de ajustar o formulário do create.



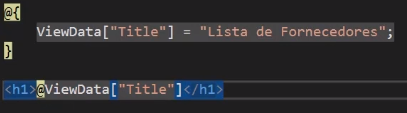
* Mudar de link para botão. Colocar o botão dentro da div. Traduzir o que tem de ser traduzido.



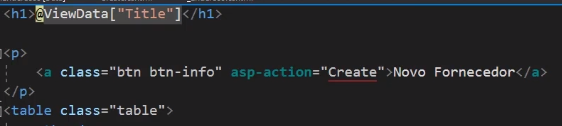
* Ajustar o tipo fornecedor para que exiba apenas as opções existentes.



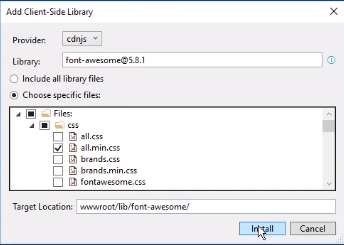
* Agora fazer algumas melhorias no index de fornecedores ( app/view/fornecedores/index)



* Colocar um botão de novo fornecedor



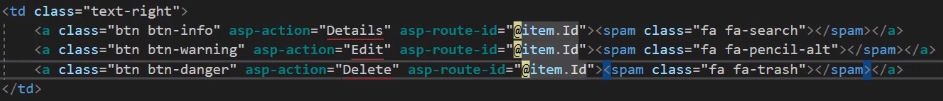
* Remover a exibição de TipoFornecedor
* Para melhorar os botões, algumas modificações devem ser feitas. Colorir, retirar os separadores “|”, alinhar a direita, e usar ícones ao invés de texto em sua identificação.
* Para que seja possível usar ícones, um tipo de fonte tem de ser instalado através do cliente side library (botão direito->add-> cliente side library). A instalação deve ser feita na pasta lib (app/wwwrot/lib). Nesse caso basta escolher o arquivo all.min.css e a pasta webfontes.



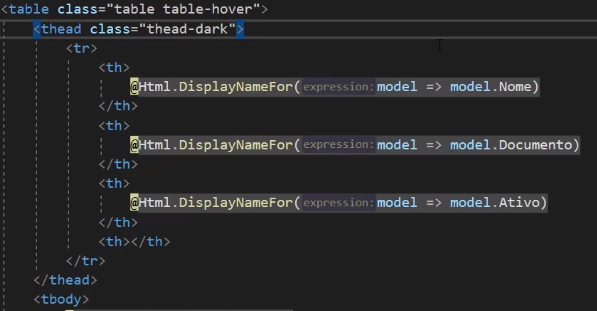
* Depois de instalado, o tipo de fonte deve ser referenciado no formulário \_Layout (app/views/shared/\_layout)



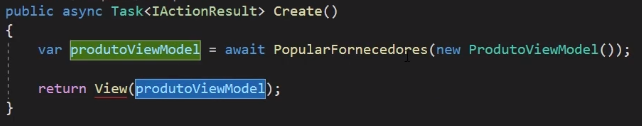
* Depois é só alterar todos os detalhes anteriormente mencionados



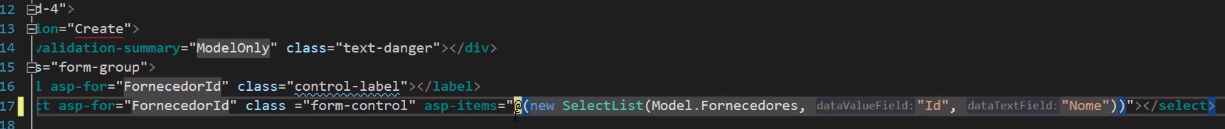
* Melhorar também alguns aspectos da tabela. Table-hover permite que a parte fique destacada quando o mouse passa por cima. Thead-dark deixa o cabeçalho na cor preta.



* Agora deve ajustar o cadastro de produtos. Para que as informações sejam exibidas corretamente, o formulário create deve ser ajustado (app/view/produtos/create).
* O “@” significa que um código c sharp está sendo escrito.
* Mudar a forma de obter os fornecedores. Antes viam através de um view bag para uma select list.

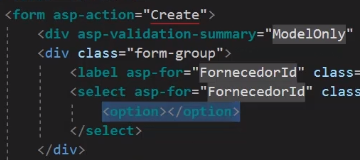


* Uma vez que na produtocontroller no create já está devolvendo uma model populada, não se faz necessário passar através e uma view bag.
* Esse método de obtenção será uma nova select list .





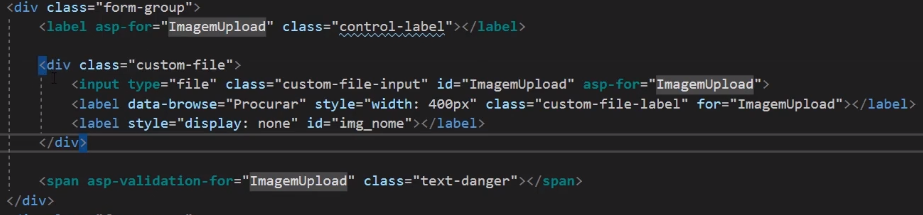
* Caso queira pode-se colocar a opção em branco para que inicialmente não venha nada selecionado.



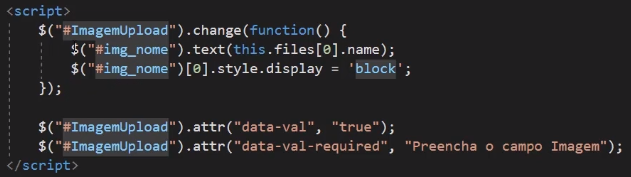
* Lembrar de colocar o spam de validação



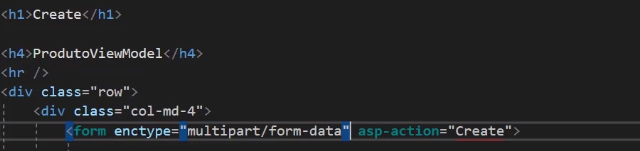
* Mudar o campo imagem para imagem upload e substituir o input. Custom-file é uma classe css do boostrap para upload de imagem.



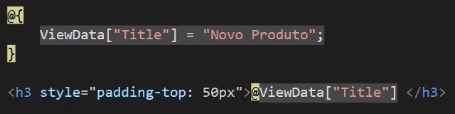
* Para mostrar o nome da imagem quando ela for selecionada, deve-se adicionar este código na parte de scripts do formulário.



* Neste momento, caso tente adicionar um produto, a aplicação irá quebrar pois o valor foi atribuído a imagemupload e não a imagem. Ou seja, o campo imagem não foi populado. Este problema será resolvido na controller.
* Ainda referente a view atual, caso se queira fazer um upload de imagem, tem de usar o enctype.



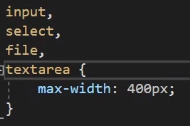
* Padding-top pode ser usado para baixar o texto da sua posição original



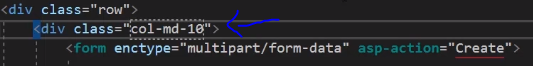
* Realizar o mesmo procedimento de ajuste que foi feito no formulário index de fornecedores, no formulário index de produto.

**Upload de imagem do produto**

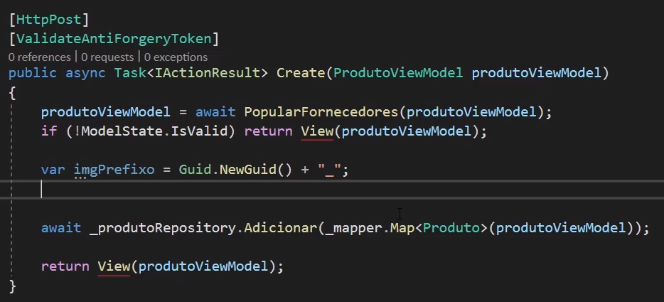
* Para que no formulário todas a caixas de informação tenham o mesmo tamanho, é necessário configurar um css global. (wwwrot/css/site.css). Abaixo de tudo, coloca-se esse pedaço de código.



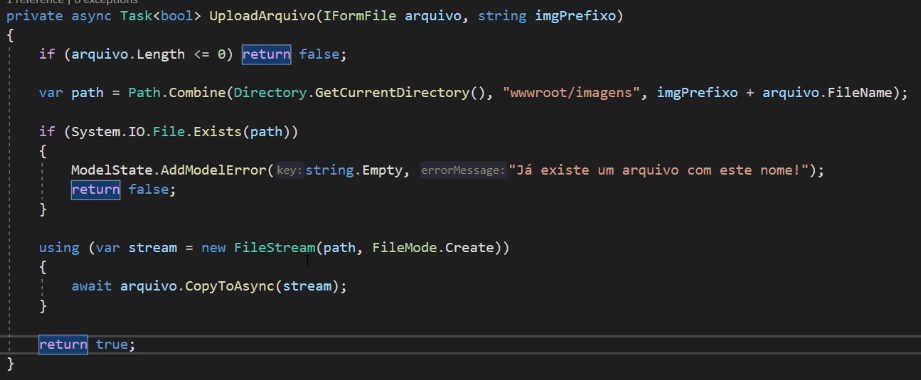
* Caso essa configuração seja feita e ainda assim o tamanho não esteja padronizado, é possível que no formulário create tenha algo limitando esse tamanho. Antes apenas 4 colunas estavam disponíveis para o formulário. Muda-se para 10 que o padrão ficará correto. O total são 12 colunas.



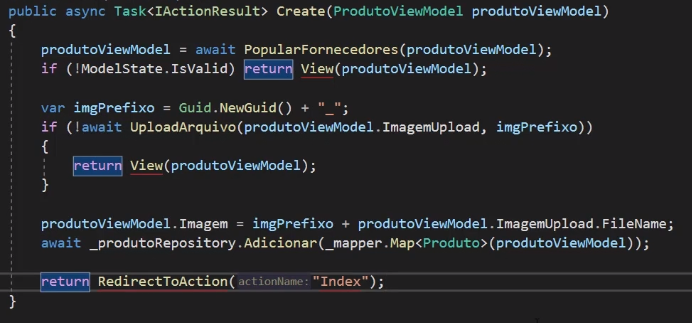
* O formulário já está pronto. Agora é trabalhar na controller para que o campo imagem seja populado corretamente.
* Primeiro cria-se um padrão para a imagem.



* Depois já pode fazer o upload. É recomendado fazer esse processo num método a parte.



* Dentro da task create, deve-se fazer uma verificação para caso não seja possível salvar o arquivo. Retornar para index caso tudo dê certo.

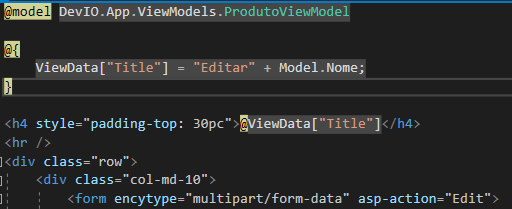


* Para que a imagem seja exibida corretamente, deve criar um campo dentro da index de produto e colocar uma tag.

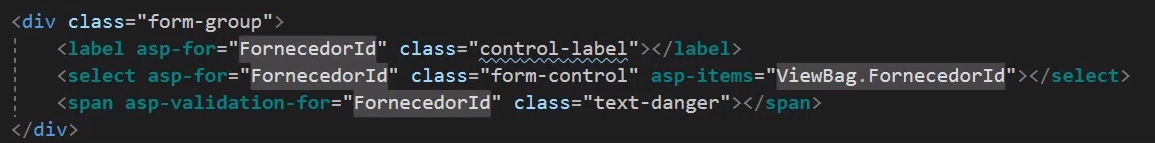


**Customizando a edição do produto**

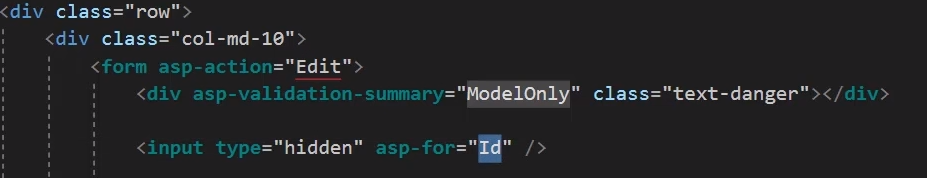
* Adequar a tela de edição de produto e a Controller para que o fornecedor seja sempre enviado e possibilitar a edição de imagem.
* Na view edit de Produtos já pode colocar o padrão com que está se trabalhando.



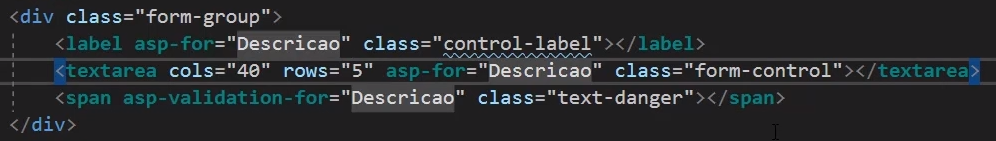
* O edit do Produto está permitindo que se troque de fornecedor através da drop down list. Retirar o select list para que essa opção não esteja disponível para o usuário.



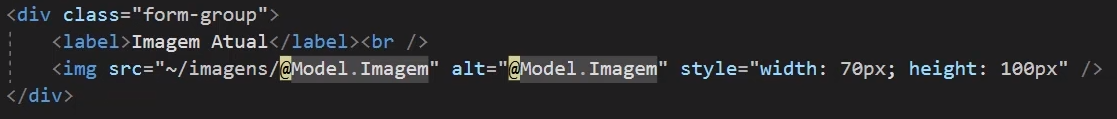
* Lembrar de realizar validação via regra de negócio para que não se possa alterar o fornecedor.(Escolha do cliente)
* O fornecedor desse produto já está embutido no produto view model através do Id. Devido a isso não há a necessidade de passar o Id no formulário.

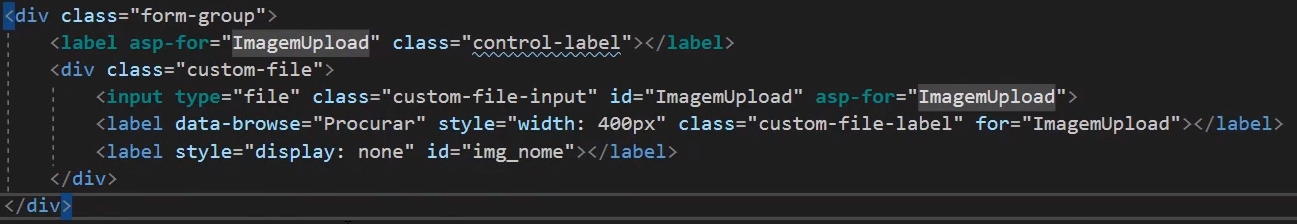


* O fornecedor desse produto já está embutido no produto view model através do Id
* Mudar o campo de descrição de input para text area.

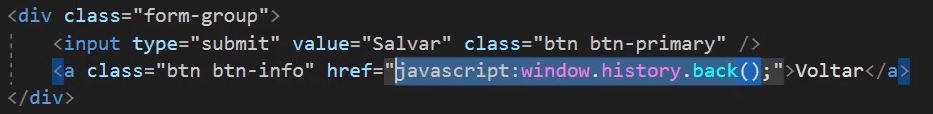


* Separar as divs da imagem. Uma com o lable para imagem. Outra com a possiblidade de realizar o upload dessa imagem, que é igual ao que foi feito com imagemupload no formulário de criação.

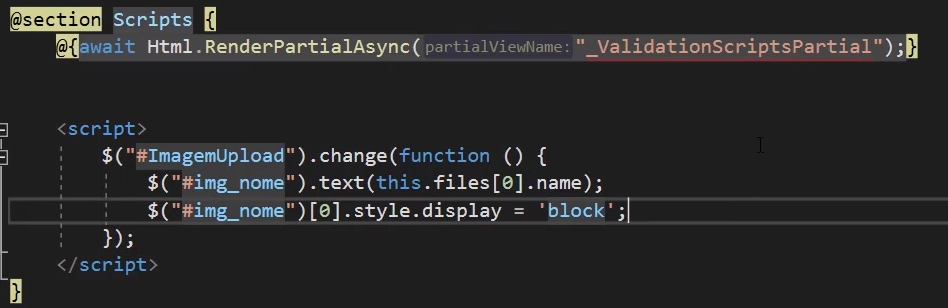




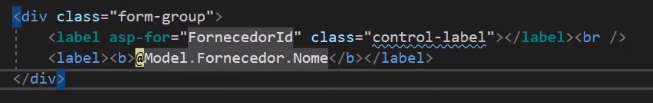
* Para o botão de voltar, nesse caso se usou javascript para que seja mais fácil voltar para qualquer tela que estivesse antes. Caso contrário teria de se fazer uma verificação de qual tela o usuário estava antes para só assim manda-lo para lá novamente. Com isso, o link back to list localizado logo abaixo pode ser retirado.



* Como não existe a exigência de preencher imagemupload não é necessário colocar os atributos de validação. Basta colocar o método que quando for mudar o conteúdo do imagem upload, o nome do arquivo será exibido logo abaixo.



* Caso queira apenas mostrar a informação do fornecedor usa-se a tag label. Na view edit de produto pode ser incluído esse trecho.



* Agora se faz necessário realizar a implementação na controller para que seja possível realizar o upload;