**Objetivo da Demanda**

Foi desenvolvido um cadastro de Nota Fiscal.

A funcionalidade consiste em permitir que o usuário acesse via desktop a tela e informe o seu pedido assim como os itens do pedido. O sistema gera uma nota fiscal a partir dos dados desse pedido e armazena no banco de dados e em um arquivo xml.

**Arquitetura**

A conjunto arquitetural foi pensado para atender os cinco princípios SOLID:

* Uma classe deve ter uma única responsabilidade;
* As classes devem poder ter seu comportamento estendido sem que haja a necessidade de alterar o seu código;
* Instâncias de objetos devem poder ser substituídas por instâncias de seus subtipos, sem que o sistema deixe de funcionar corretamente.
* Interfaces específicas são melhores do que uma interface única;
* Dependa de uma abstração e não de uma implementação.

A arquitetura também foi pensada visando baixo acoplamento, permitindo que várias partes do sistema interajam sem que haja muita dependência entre os módulos ou as classes. Isso também favorece a reutilização dos códigos, assim como a escalabilidade em um cenário futuro. Toda a regra de negócio da aplicação foi implementada no domínio.

O projeto foi organizado em pastas a fim de facilitar a navegação e separar a responsabilidade dos diferentes contextos trabalhados dentro da aplicação. Foi adotada também a separação das responsabilidades em projetos. Evita que um projeto tenha mais bibliotecas(dll) referenciadas do que aquelas que ele realmente usa ou depende.

Foi utilizada a criação de projetos de testes unitários para cada projeto que tenha partes testáveis na aplicação. Por exemplo, o projeto de repositório terá seu projeto de testes unitário independente dos demais.

Foi adotada a separação das classes de domínio das classes que representam modelos da camada de apresentação. Adição de uma camada de tradução(Application) entre os modelos que representam a camada de exibição e os modelos de domínio.

**Relatório de implementação**

* Presentation
  + Os campos Estado de Origem e Estado de Destino estavam permitindo a inserção de qualquer caractere, como existiam opções fixas o campo foi apresentado com combobox. Com isso garantiu a inserção somente de valores válidos.
  + “É possível indicar estados inválidos tanto na origem quanto no destino” o documento trás essa afirmação, mas não deixa claro o critério para invalidez. Consegui inferir a implementação citada no item acima. Outra coisa que notei, foi que para alguns estados não existe uma regra que define CFOP, seria essa validação?
  + A coluna valor dentro do tabela estava permitindo a inserção de qualquer caracter, porém ao tirar o foco reportava uma exceção em uma caixa de dialogo. A validação foi implementada e a mensagem foi tratada para o usuário.
  + Alguns valores de entrada não estavam sendo tratados quando eram nulos. Foram tratadas essas entradas.
* Application
  + O método EmitirNotaFiscal() basicamente fazia a tradução do modelo que chegava da camada de apresentação para o modelo de domínio, essa parte foi transferida para a camada Application, que tem essa responsabilidade em especifico.
  + Todo o controle de transação foi realizado nessa camada, caso falhe qualquer operação nas camadas inferiores os dados não serão persistidos no banco.
  + Como não existiam regras de negócio complexas para validar durante as transações(maioria das regras envolvia cálculo de campos) com o banco, foi adotado um retorno mais genérico de erro.
* Service
  + A camada de serviço ficou com um papel menor devido a necessidade do projeto, mas optei por manter ela pensando em um cenário realístico, onde é preciso prever a escalabilidade da aplicação.
* Domínio
  + O método EmitirNotaFiscal() que ficava dentro na Nota Fiscal foi removido, isso porque ele estava sendo responsável por aplicar regras que na verdade eram responsabilidade do Item da Nota Fiscal.
  + Para o cálculo de CFOP foi identificado que somente o Estado de Destino já era suficiente para o cálculo. Com a implementação do Exercício 7 talvez seja válida a implementação do padrão Strategy.
  + Ainda com relação ao cálculo do CFOP existe um problema na regra, que atribui dois CFOPs diferentes para a mesma condição. Quando o estado de destino é “SE”, segundo a regra o mesmo pode assumir os valores "6.007" e "6.009". No projeto adotei o “6009” como padrão porque ele é usado em outra regra de negócio.
  + No exercício 7 é especificado somente a forma de calculo do valor do percentual do desconto e não como mesmo entra no restante do cálculo do valor do item.
* Repository
  + Para realizar alguns testes unitários necessitei usar um método de busca usando o Entity, no entanto o Entity reportou a necessidade de alterar o mapeamento das propriedades que estavam como decimal no banco e double na aplicação. Isso impactou em algumas mudanças no interior da aplicação, uma vez que c# trata double como tipo primitivo e decimal não.