

# Fluxo de Caixa – Implementação

## Índice

### Sumário

Índice .....	1
1 – Breve descrição .....	2
2 – Repositórios de Código.....	2
3 – O que foi implementado?.....	3
4 – Camadas .....	3
5 -Pacotes Externos .....	4
6 – Setup Banco.....	5
7 – Instalação.....	5
Método um.....	5
Método dois .....	5

## 1 – Breve descrição

O sistema foi desenvolvido usando .Net Core, com implementações de design patterns:]

- DDD
- Alguns conceitos Solid
- Unit Of Work
- Repository
- ValidationResult
- MVVM
- O(ZERO) Procedure (apesar de que 2 métodos foram duplicados com o uso de proc. apenas para demonstrar que se conhece o uso de SQL e proc.)
- Injeção de dependência

O serviço de api está on line disponibilizado e publicados nos endereços:

<http://fluxodecaixa.dietcode.com.br/swagger/index.html>

<https://fluxodecaixa.french.com.br/swagger/index.html>

Tem implementado, fake a camada de autenticação. Se fosse totalmente implementada seria usado certificado em banco para criptografar e decriptografar os dados em um campo binário na tabela e usando JWT.

O serviço de Ping é apenas um Ping, se houver resposta positiva o serviço está on, se der erro o site todo está fora.

Os códigos algumas classes contêm observação do porquê de cada uso ou situação no método comentado.

## 2 – Repositórios de Código

No Repositório Git. Subi o projeto principal, e lá tem também a minha biblioteca de apoio para uso em sistemas.

O repositório deste projeto está em:

<https://github.com/Humberfrench/FluxoDeCaixa>

O repositório dos pacotes nugget está em:

<https://github.com/Humberfrench/DietTools> (Dietcode é o nome fantasia da minha empresa e marca)

Existe outros repositórios alguns 100% implementados e outros foram estudos que demonstram a referência e o conhecimento de DDD. O meu conhecimento foi obtido através do Eduardo Pires, um colega de trabalho disponibilizou ao entrarmos na empresa os vídeos dele falando disso para entrarmos no conhecimento e aplicar o uso da metodologia para isso

**Jack** – Para controle de assistência social e distribuição de sacolinhas de Natal, usando DDD, NHibernate, Net Framework, Ajax com Javascript.

**Reddington** – Para gerenciamento de escolinha de moral cristã a alunos da assistência social, usando DDD, NHibernate, Net Framework, Ajax com Javascript.

**Jim Halpert** – Incompleto. Api em Net Core para munir o sistema front de estudo usando Angular (não finalizado, era estudo)

### 3 – O que foi implementado?

Foi pedido um fluxo de caixa, com lançamentos que sejam especificados a débito ou crédito. E que fosse mostrado um relatório consolidado. Como foi pedido um serviço, foi disponibilizado um serviço REST com o que foi pedido e mais algumas informações

O modelo de implementação é que o cliente faz o lançamento se ele erra, pode ser um cancelamento, não excluimos o lançamento, apenas marcamos como estornado, para até um controle interno de estornos do dia/mês.

- Lançamento (débito ou crédito, valor e descrição) ~~(foi pedido)~~
- Estorno – para cancelar o lançamento

#### Relatórios/Consultas

- Todos os lançamentos do dia
- Todos os estornos do dia
- Todos os lançamentos de uma data específica
- Todos os lançamentos de um rangem de datas
- Todos os lançamentos de um mês (para isso entra mês e ano para evitar conflito)
- Todos os estornos de um mês (para isso entra mês e ano para evitar conflito)
- Relatório consolidado do dia ~~(foi pedido)~~
- Relatório consolidado de uma data específica
- Relatório consolidado do dia – duplicado, porém usando SQL Server procedure
- Relatório consolidado de uma data especifica – duplicado, porém usando SQL Server procedure

Foi colocado um serviço de autenticação, não implementado, apenas para constar, porém se isso fosse realmente um sistema etc., este seria um item obrigatório

Um item de Ping que apenas avalia a disponibilidade do serviço

Como os métodos estão abertos poderão ser chamados via Swagger por aqui:

<http://fluxodecaixa.dietcode.com.br/swagger/index.html>

<https://fluxodecaixa.french.com.br/swagger/index.html>

OU pelo endereço que for implantado no servidor corporativo:

[http://<endereco\\_corp\\_do\\_site>/swagger/index.html](http://<endereco_corp_do_site>/swagger/index.html)

Devido ao tempo não implementei telas, pois até nem foi pedido, e nem fiz um executável pra chamadas, pois como está sem auth e aberto não vi necessidade

### 4 – Camadas

O sistema divide em n camadas, cada uma com uma responsabilidade

Camada REST/Api – é onde estão declarados os end points

Camada App – é onde o retorno é convertido para uma classe anêmica de viewmodel para não expor os objetos do domínio que podem possuir regras dentro dele. Nessa camada é analisado o retorno do serviço via validation result e assim comita se as mudanças na Unit of work, via save changes no Entity framework. Na camada APP temos outra classe que contém as classes de ViewModel e de interfaces de integração do serviço de app

Camada de Domínio – Esta camada contém dois tipos de classes, no domínio temos as classes que representam as tabelas do banco e interfaces de implementação do serviço e repositório, enumeradores e object values que seriam classes que retornam dados específicos do banco, a classe de serviços forma com o Domínio o CORE do sistema, com regras, cálculos, implementações etc.

Camada Repositório é a camada que abstraímos os dados e consultas específicas e o CRUD das tabelas.

Há ainda a camada IOC, que é o gestor de dependência para o projeto inteiro. Essa classe existe para que na camada de REST não tenha a referência a TODOS os projetos, então assim realmente torna independente POIS se eu quiser inserir uma classe de serviço nova no lugar da existente, basta eu TER as mesmas implementações da interface.

## 5-Pacotes Externos

No repositório Git será encontrado a pasta Nuget Packages. Lá contém os pacotes publicados:

- Dietcode Api Core – Gerenciador de retorno da API
- Dietcode Api Core Results – Tipos de retorno e resultados da api e serviço
- Dietcode Domain Validator – É o pacote do Validation Result
- Dietcode Lib – Lib de Helper.

O Validation result é uma classe que contém uma propriedade de retorno que retorna o objeto de retorno desejado pelo método.

Em caso de erros, a propriedade Invalid virá como true, ou a Valid estará com valor false, e assim é preciso observar a mensagens retornadas na propriedade Erros.

Ele é generic. Se não declara como generic o retorno vem com tipo object, se não ele vem com o tipo T desejado.

A Core e Core results são apenas formatos de retorno do serviço, posso retornar um tipo de dado chamado MethodResult que é herdado os tipos comuns de status code, por exemplo um Method result do tipo OK, é um status 200, um NotFound é 404 e assim por diante.

Na lib de Helper temos um conversor muito importante que transforma via JSON uma classe do domain para uma de viewmodel. Se as classes de domain for diferente da View model, só virão as classes comuns. É um automapper personalizado.

Para instalar os pacotes basta referenciar a pasta dele no config de source do nugget no Visual Studio. Se não encontrar será preciso exportar o pacote com um comando nugget.

```
dotnet nuget push <caminho do arquivo >\nome.nupkg -s <caminho destino>
```

É provável que precise usar o command prompt como administrador e ir até a pasta que tem o nuget.exe que está dentro das pastas do visual Studio.

Sem esses pacotes não funcionará a solução. Dúvidas poderei resolver via telefone.

Outros pacotes são públicos pela Microsoft e parceiros.

## 6 – Setup Banco

No repositório existe uma pasta SQL. Dentro dela tem uma cópia de backup do banco. E em outra pasta os scripts de criação do banco, duas tabelas e duas procedures.

O projeto se rodar vai apontar para este banco: Server=dietcode.com.br\\Dev. Ele está em minha casa em um servidor meu interno. É um servidor genérico de dev. Não tem problemas o uso e abuso de inserir e mexer nos dados.

## 7 – Instalação

Existem N opções.

### Método um

Antes de tudo criar uma pasta e criar um site no iis, com DNS, ou na pasta criada adicionara aplicação/diretório virtual para que possa colocar o app publicado.

A mais simples é no repositório GIT clonar o projeto e na pasta publish tem a última versão publicada já, é só implementar no iis. E chamar. POIS está configurado para um banco online e vai rodar.

### Método dois

Criar o banco no servidor interno ai de vocês. Pode ser via restore do banco (.bak) que está disponibilizado junto do repositório Git. OU rodar os scripts na ordem:

1. Criar banco
2. Rodar scripts para criar tabela de tipo de lançamento
3. Rodar scripts para criar tabela de Lançamento
4. Rodar scripts para criar as procedures

Após isso alterar a string de conexão para conectar-se ao banco. É preciso dar permissões ao usuário que vai ser usado para conectar a essa base. Poderia usar o SA, porém, como é um ambiente corporativo é descartada esta possibilidade.

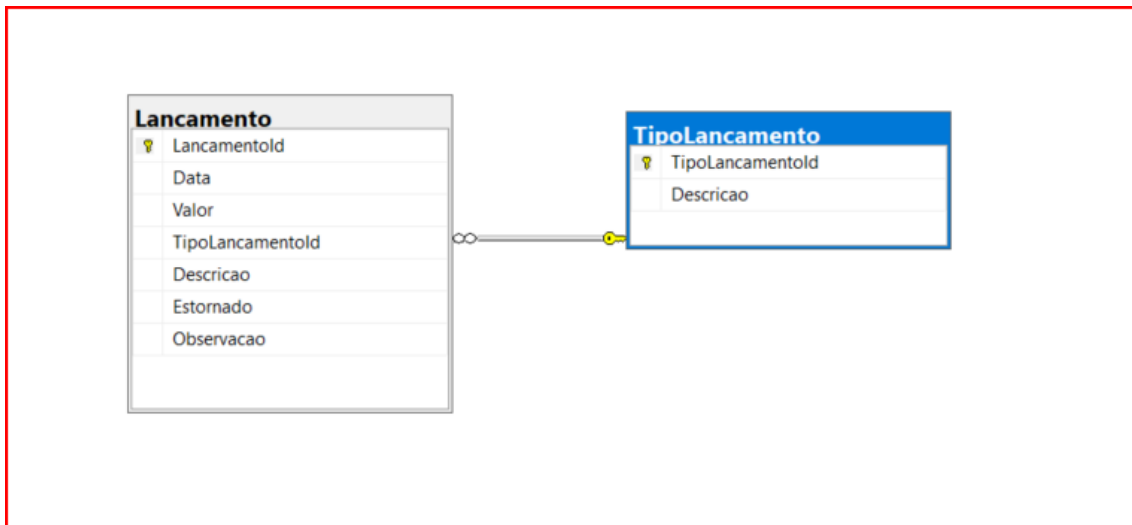
Peço desculpas por não disponibilizar via container que é algo que ainda está fora do meu conhecimento. Mas caso a parceria nossa vingue, me proponho a entender e conhecer melhor a fundo e trabalhar conforme os padrões e metodologias da empresa.

## 8- Documentação Tecnica

Colocando tudo junto no mesmo doc. para evitar o envio de múltiplos arquivos.

## 8.1- Diagrama de Banco (básico)

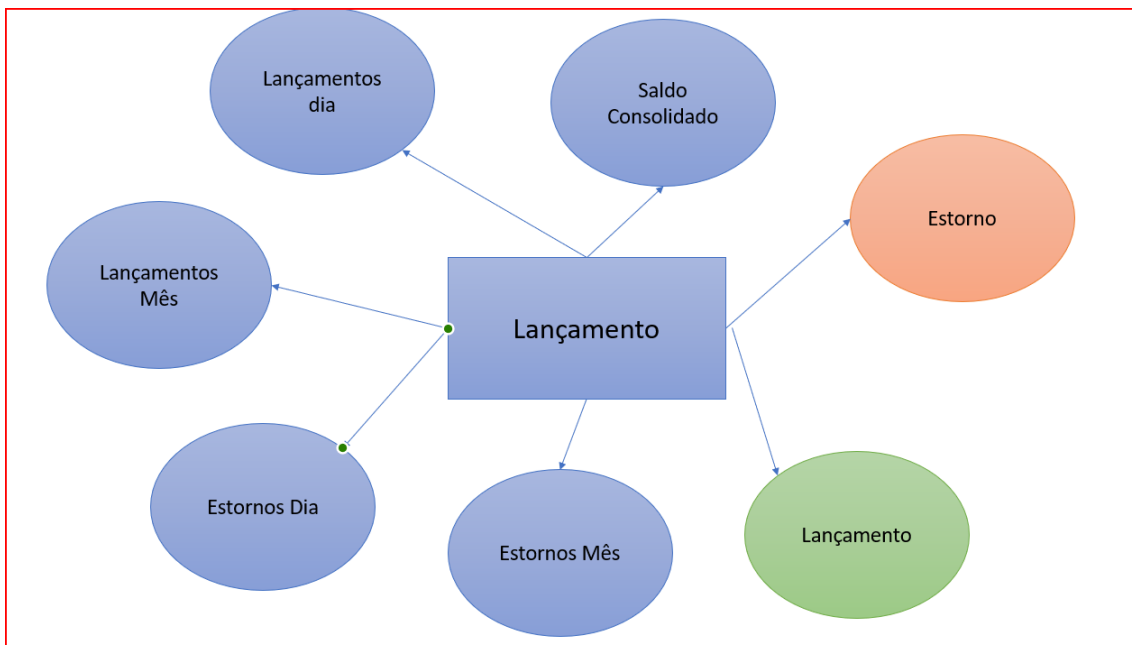
O banco contém duas tabelas:



A de lançamento que contém todos os lançamentos uma tabela auxiliar que tem os cadastros de tipos de lançamentos.

Infelizmente não tenho nenhuma ferramenta aqui disponível que modele alguns diagramas.

Em tese a visão global do conceito dos lançamento pode ser visualizada no diagrama abaixo:



As partes do projeto separadas de acordo com as responsabilidades em si

REST	APP	Service/Domain	Repository
<ul style="list-style-type: none"> <li>chamadas</li> </ul>	<ul style="list-style-type: none"> <li>De View model para domain</li> <li>De domain para View model</li> <li>Início de transação</li> <li><u>Commit</u></li> <li>Interface de serviços de App</li> <li>Serviços de App</li> <li>Organização do retorno de acordo com o resultado do serviço</li> </ul>	<ul style="list-style-type: none"> <li><u>Dominio</u></li> <li>Core</li> <li>Validações</li> <li>Organização e regras</li> <li>Interface do Serviço</li> <li>Interfaces de Repositório</li> <li>Interface de chamadas REST (caso haja)</li> </ul>	<ul style="list-style-type: none"> <li>Obtenção de dados</li> <li>CRUD</li> <li><u>Queries</u></li> <li>Gestão do Contexto do DB</li> </ul>

#### Observações:

O serviço de API só efetua a chamada, e alguma conversão de entrada de string para data, e retorna como “Completo” com o conteúdo ou não de acordo com o resultado do método do serviço.

O domínio é toda a regra, soma, tudo que se diz ao negócio. Tem a configuração das entidades de domínio e os comportamentos dos mesmos.