Sistema de movimentação bancária

O desafio

Fazer um sistema de controle financeiro que possua as seguintes funcionalidades:

1) Visualizar o movimento e o saldo bancário previamente importado de um arquivo OFX.
2) Armazenar os dados importados em um banco de dados para posteriormente recalcular com novas importações.
3) Utilizar uma API para visualizar a cotação do dia das 3 principais moedas do mercado (Dólar, Euro e Bitcoin).
4) A armazenar os dados das cotações para que no futuro possam ser gerados consultas sobre o histórico das variações destas moedas.

Fase de análise

Nesta etapa foram levantados dois dos principais pontos do projeto (O arquivo OFX e a API de cotação).

1) Ao analisar a estrutura do arquivo OFX foi observado que os dados estão assinalados com TAGs parecidas como as utilizadas em html facilitando a leitura e decomposição de seus dados.
2) A API do site https://hgbrasil.com/status/finance que realiza a cotação das 3 moedas traz outras informações, o importante é que o resultado da consulta está no formato JSON, facilitando a leitura dos mesmos.

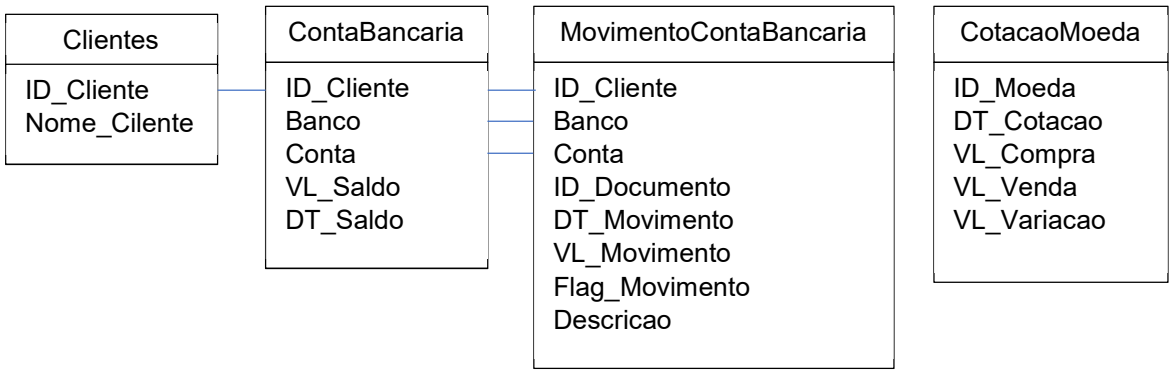
Fase de modelagem do banco de dados

Nesta etapa ao analisar qual o banco de dados seria adotado, decidi pelo SQLite por ser leve, não necessitar de instalação de software de terceiros, por ser fácil de se criar e apresentar o fator portabilidade, com isso o sistema pode ser utilizado em outros sistemas operacionais inclusive em dispositivos móveis. O passo seguinte foi realizar o levantamento dos dados fornecidos através do arquivo OFX e da API de cotação.

1) No arquivo OFX foram escolhidas 8 Tags para compor as tabelas ContaBancaria e MovimentoContaBancaria, segue abaixo descrição das Tags:
   - <BANKID> identifica o Banco
   - <ACCTID> Identifica a Agencia/Conta
   - <TRNTYPE> identifica o tipo de transação (Debito/Credito/Outro)
   - <DTPOSTED> identifica a data da transação
   - <TRNAMT> identifica o valor da transação
   - <CHKNUM> identifica o documento da transação
   - <MEMO> identifica a descrição da transação
   - <BALAMT> identifica o saldo final do período exportado.
2) Despois de verificar o funcionamento da API e constatar que a mesma me retorna os dados no formato JSON as coisas ficaram mais fáceis. Aqui apenas precisarei das informações:
   - Tipo de Moeda – USD, EUR e BTC
   - Valor de venda – para as 3 moedas
   - Valor de compra – para as 3 moedas
   - Variação – para as 3 moedas

Obs.: Como o arquivo OFX me retorna os dados sobre o banco e conta, decidi modelar as tabelas prevendo uma possível expansão no sistema, com isso será fácil implantar novas funcionalidades como, por exemplo, acrescentar um filtro de seleção banco/conta, transformando o programa em um sistema em multiconta, isso sem onerar em nada no funcionamento de hoje.

A representação da modelagem dos dados está apresentada abaixo:



Fase de desenvolvimento

Nesta etapa decidi por quais tecnologias seriam utilizadas no sistema.

1) Para o arquivo OFX decidi por fazer um componente para a leitura e sua interpretação. O mesmo encontra-se na unit ofxloader.pas e sua função é ler o arquivo OFX e decompor todas as principais TAGs do arquivo.
2) Para a API decidi fazer a chamada através do REST, para tanto utilizei os componentes TRESTClient, TRESTRequest, TRESTResponse e TJSONValue. Fazendo a chamada da API pela URL https://api.hgbrasil.com/finance?key=publica a mesma retorna através do componente TRESTResponse um conjunto JSON e através do componente TJSONValue os dados são interpretados.
3) Para acesso e criação do banco de dados, foi utilizado os componentes FireDAC por fazerem acesso a praticamente qualquer tipo de base de dados e facilitar a migração para outras bases.
4) Na primeira versão o programa foi feito utilizando a VCL, mas como a base de dados é em SQLite, não fazia muito sentido deixar o programa preso no Windows, então migrei para uma segunda versão, nesta utilizei o FMX (Firemonkey), dessa forma o sistema passa a ser multi plataforma. Na transcrição para o FMX, aproveitei a inclui algumas melhorias, como o verificador de scripts, que checa se os arquivos CriaTabelas.SQL e PopulaClientes.SQL estão presentes no diretório da aplicação e também inclui um diferencial na Grid, onde os débitos são apresentados em vermelho e os créditos em verde.
5) O layout da tela foi seguido fielmente como o sugerido pela equipe de UX.

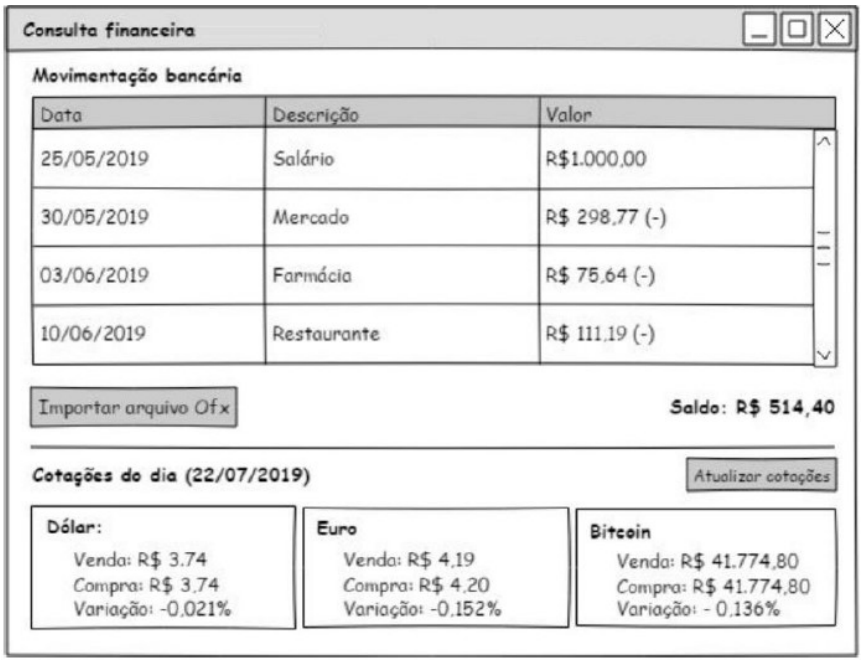


Diagrama simplificado do programa