Documentação Técnica

Autor: Alexandre Otávio Guedes Drummond

Nome do Software: Konv Mini Bank

Versão do Software: 1.0

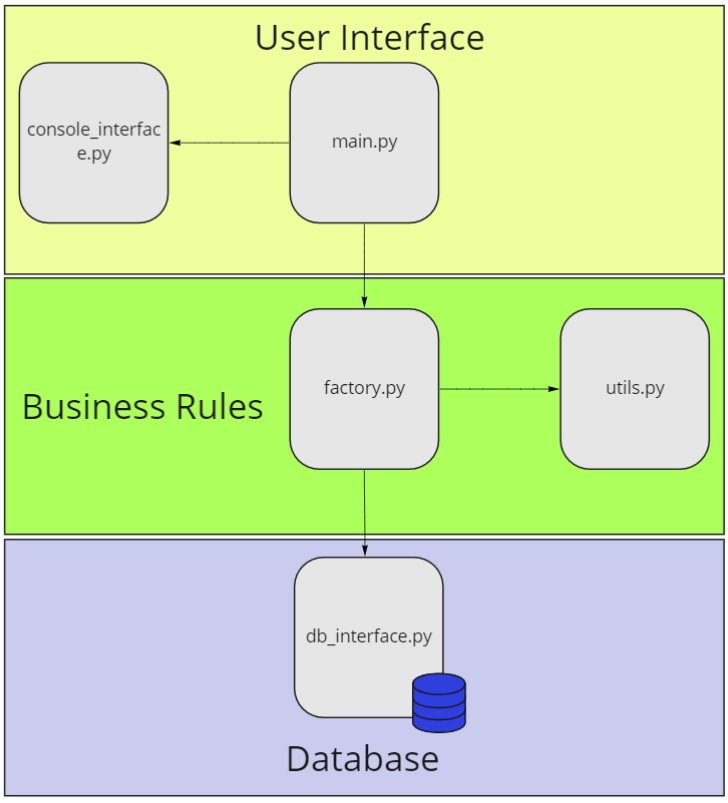
Informações Gerais:

O aplicativo “Konv Mini Bank” possui o objetivo de possibilitar a interação do usuário pelos caixas eletrônicos com suas contas bancárias por meio do console, pelas operações de depósito, saque e consulta de saldo e extrato em sua conta. A aplicação se encontra conectada com o banco de dados do banco Konv de forma a estar sempre atualizado ao receber uma operação.

Informações Técnicas

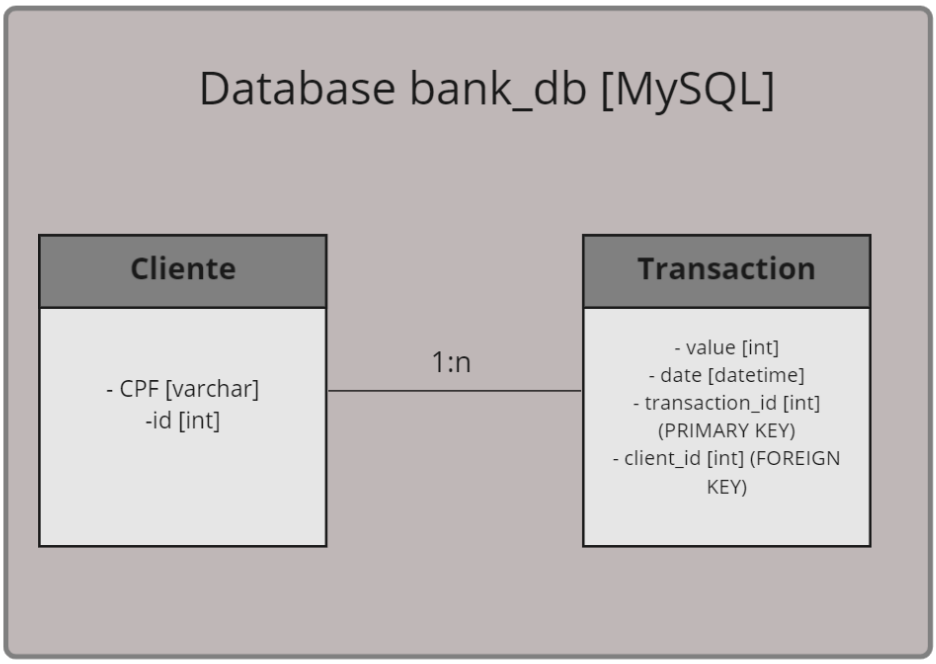
Estrutura:

Apesar de se tratar de um sistema de pequeno porte, a aplicação foi estruturada seguindo padrões desejáveis de arquitetura de software para favorecer sua escalabilidade e facilidade de manutenção. Assim, foi elaborada a seguinte estrutura:



- **main.py:** Componente de execução do aplicativo. Responsável por orquestrar os diversos módulos e de forma que o usuário interaja com o banco de dados por meio do console.
- **console_interface.py:** Responsável pelas funcionalidades que permitam a interação do usuário com o sistema por meio da linha comando, exibindo o menu na tela e recebendo respostas.
- **factory.py:** Contém os objetos que encapsulam as regras de negócio do funcionamento do sistema, executadas por meio de métodos de classe.
- **db_interface.py:** Contém todas as funções que permitam interação com o banco de dados. Construído utilizando o banco MySQL.
- **utils.py:** Responsável por armazenar funções diversas necessárias para a execução das regras de negócio.

Foi tomado o cuidado de alcançar o máximo desacoplamento entre componentes para que fosse obtido um sistema modular. Assim, apesar de ter sido optado pelo banco de dados MySQL, esse poderia ser substituído por qualquer variação, relacional ou não, apenas adaptando o arquivo "*db_interface.py*". O mesmo é válido para a interface com o usuário, "*console_interface.py*".



Foi optado pelo modelo de *Event Sourcing* para a transmissão e armazenamento dos dados bancários *Transaction*, utilizando blocos de dados pequenos mas que permitem o monitoramento das contas de cada cliente desde sua criação.

Funcionamento:

Ao ser inicializado, o sistema se conecta ao usuário correto por meio de sua informação de CPF, após conferimento de o valor fornecido estar no padrão de 11 caracteres numéricos. Ao inicializar uma instância da classe *Client*, é verificado no banco de dados se já existe um usuário com aquele valor de CPF e, caso positivo, o objeto é instanciado com o valor correspondente de *id* na base de dados. Caso negativo, é criado um usuário novo e o *id* é gerado automaticamente. Além das informações citadas, ainda é determinado um atributo de classe *"is_online"* para que o sistema acompanhe a presença do usuário enquanto estiver sendo usado.

A sincronização entre as informações no aplicativo e na base de dados é feita constantemente, a cada nova operação. Essa característica permite que os dados permaneçam atualizados mesmo no caso de falhas durante operações.

Antes de confirmar a função desejada, o CPF do usuário é solicitado novamente como forma de validação do usuário, em todas as três funções.

O armazenamento dos dados por objetos *Transaction* faz com que as funções de saque e depósito sejam bastante semelhantes, se diferenciando pelo valor negativo da transação no caso dos saques. Além disso, antes de sacar é verificado se o saldo disponível é suficiente, operação desnecessária no depósito, e é exibido a combinação entregue de notas, com a menor quantidade possível de cédulas de acordo com os valores disponíveis.

A função de retirar o extrato bancário e consulta do saldo se inicia também com a validação do usuário por meio de seu CPF e , caso validado, o extrato é obtido por meio de consulta ao banco de dados de todas as transações com o *id* do cliente. O saldo é calculado por uma simples soma de todas as transações do cliente, a partir a criação de seu perfil na base de dados, sendo que caso não haja transação nenhuma (cliente novo) é entregue automaticamente o valor 0 (zero). Por fim, a informação obtida é formatada e exibida na tela.

Com objetivo de tornar a leitura do código mais fluida, as funcionalidades de tratamento de erro foram preferencialmente colocadas no nível mais baixo, considerando a necessidade de um sistema modular. Por isso, a maior parte do desse tipo de tratamento está concentrado em "*factory.py*". Essa etapa foi de grande importância para obtenção de uma melhor experiência do usuário, uma vez que é indesejável que falhas no sistema devido entradas incorretas levem à sua quebra. Ao invés disso, a aplicação se tornou à prova de falhas catastróficas ao retornar sempre ao menu inicial caso encontre alguma situação não planejada, característica alcançada ao introduzir o atributo de classe *"is_online"* para o cliente. A aplicação é encerrada somente quando esse atributo tem seu estado alterado para o boolean *"False"* por meio da interface com o usuário.

Testes:

Não foi explicitamente solicitado no enunciado do teste a criação de testes automatizados, entretanto por essa ser uma parte importante de qualquer sistema foram implementadas avaliações das principais funcionalidades da aplicação, disponíveis no componente tests\unit_tests.py e tests\int_tests.py.

Dependências:

Uma vez que foi solicitado no enunciado do teste que não fosse usado qualquer tipo de framework, optou-se por estender o desafio e criar um sistema com a menor quantidade de bibliotecas externas quanto possível. Essa opção pode ser desejável ao tornar a aplicação mais robusta contra conflitos devido ao versionamento de dependências.

As dependências do sistema estão apresentadas na lista a seguir:

Pacote	**Versão**	**Pacote**	**Versão**
atomicwrites	1.4.0	pathspec	0.9.0
attrs	21.4.0	platformdirs	2.5.2
black	22.6.0	pluggy	1.0.0
click	8.1.3	protobuf	4.21.2
colorama	0.4.5	py	1.11.0
iniconfig	1.1.1	pyparsing	3.0.9
mypy-extensions	0.4.3	tomli	2.0.1
mysql-connector-python	8.0.29	typing-extensions	4.3.0
packaging	21.3		

Guia de instalação

Todos os comandos podem ser executados pelo prompt de comando da máquina utilizada. Para isso, basta inicializar o prompt e alterar o diretório para o local onde se deseja instalá-lo com o comando *"cd path\to\folder"*. Em seguida, execute os comandos a seguir:

	Tarefa	Comando
1	Verificar se as ferramentas estão instaladas	*$ docker -v* *$ docker-compose -v* *$ git –version* *$ pip –version*
2	Clonar o repositório contendo o código fonte	*$ git clone https://github.com/aogdrummond/konv_mini_bank.git*
3	Criação do ambiente virtual	*$ python -m venv konv_bank_venv*
4	Instalação das dependências no ambiente virtual	*$ pip install -r requirements.txt*
5	(Opcional) Determinar o caminho para persistência dos dados	*Adicionar o caminho do diretório ao arquivo "docker-compose.yml"*
6	Download da imagem e criação do container com o banco de dados	*$ docker-compose up*
7	Ativação do ambiente virtual	*$ konv_bank_venv\Scripts\activate [Windows]* *ou* *$ konv_bank_venv\bin\activate [Linux\Mac]*
8	Executar a aplicação	*$ python main.py*