Technical Documentation

Author: Alexandre Otávio Guedes Drummond

Software Name: Blockchain Mini Bank

Software Version: 2.0

General Information:

The "Konv Mini Bank" application aims to enable the user to interact with their bank account through ATM's console, for deposit, withdrawal, balance inquiry, and account statement operations. The application is connected to the bank database, so it is always updated after carrying out an operation.

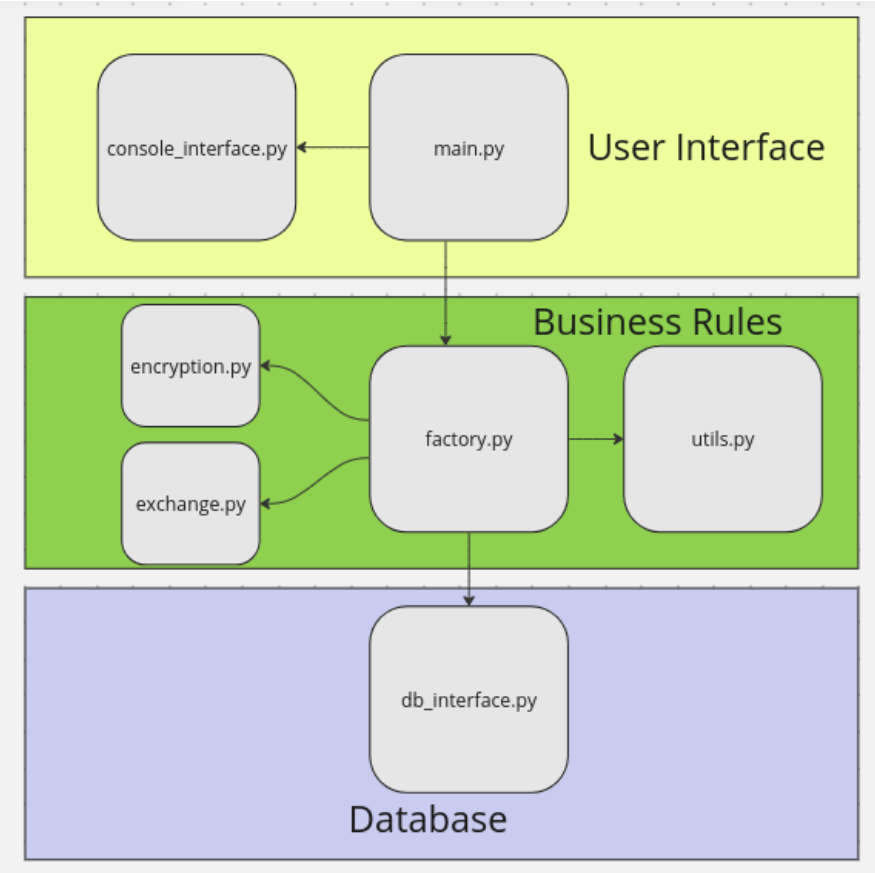
Additions to this version:

- The Flask API framework is applied to request information about money exchange rates from an external API.
- Blockchain-based logic is implemented to enable verification of data consistency, ensuring the system is secure against data overwriting in the database.
- Appropriate integration tests to validate the new functionalities.
- Chat-GPT was used to write docstrings, type hints, and pylint in scripts, thereby improving the readability of the code.

Technical Information

Structure:

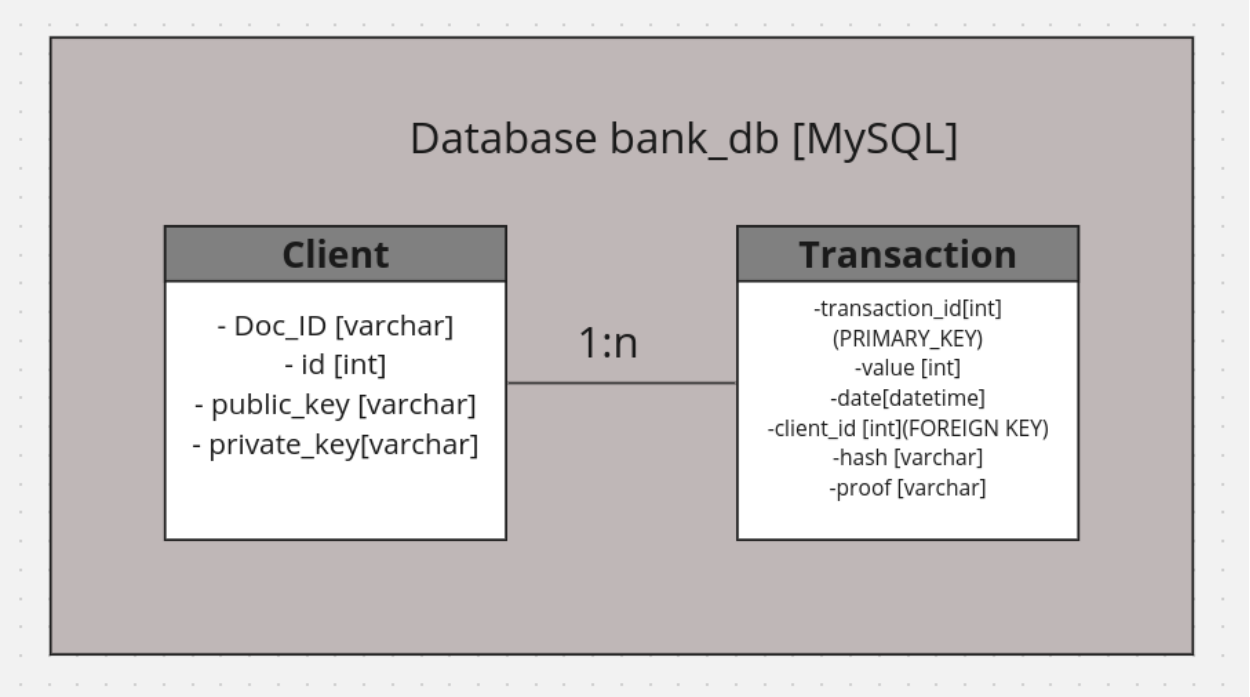
Although it is a small system, the application has been structured following desirable software architecture principles to allow its scalability and ease of maintenance. Thus, the following structure was elaborated:



● **main.py:** Component responsible for running the application. It orchestrates the various modules so that the user can interact with the database through the console.
● **console_interface.py:** Responsible for functionalities that allow the user to interact with the system through the command line, displaying the menu on the screen and receiving responses.
● **factory.py:** Contains the objects that encapsulate system's operation business rules, executed through class methods and using a few of SOLID principles for software design.
● **db_interface.py:** Contains all functions that allow interaction with the database. Built using MySQL database.
● **utils.py:** Responsible for storing various functions required to apply business rules.
● **encryption_factory.py:** Contains the objects that provide the blockchain functionality through encryption of the data.

● **exchange_tool.py:** Contains the object that refers to an external API to get information about money exchange rate.
● **.env:** Stores the environment variables, which improve the maintainability of the code through preventing that information to be hardcoded in the script.

Attention was given to achieve the maximum decoupling between components in order to obtain a modular system. Thus, although the MySQL database was chosen, it could be replaced by any variation (also known as SQL “flavors”), relational or not, by adapting the *"db_interface.py"* file. The same applies to the user interface, *"console_interface.py"*.



The *Event Sourcing* model was chosen for transmitting and storing bank transaction data, using small data blocks that allow monitoring each customer's account since its creation.

Operation:

When initialized, the system connects to the correct user through his/her document number, after checking that the provided value is in the Brazilian standard of 11 numeric characters. When initializing an instance of the *Client* class, it is checked in the database whether there is already a user with that document value, and if positive, the object is instantiated with the corresponding id in the database. In the case it is an unknown number, a new user is created and the id is generated automatically. In addition to the mentioned information, a class attribute

"is_online" is determined so that the system monitors the user's presence while it is being used, allowing the application to shut down when the operation is over.

Synchronization between information in the app and in the database is constantly established after each new operation. This characteristic allows the data to remain updated even in case of failures during operations.

Before confirming the desired function, the user's CPF is requested again as a way of double checking the user.

The storage of data by *Transaction* objects makes the withdrawal and deposit functions quite similar, differing only in the negative value of transaction in the case of withdrawals. In addition, before withdrawing, it is verified whether the available balance is sufficient, an unnecessary operation in deposit, and the combination of notes delivered is displayed, presenting always the smallest possible amount of bills according to the values.

The function to withdraw the bank statement and check account balance also starts with the user validation through their document number, and if validated, the statement is obtained by querying the database for all transactions with the client's id. The balance is calculated by a simple sum of all the client's transactions, starting from their profile creation in the database, and if there are no transactions at all (new customer), the value 0 (zero) is automatically delivered. Finally, the obtained information is formatted and displayed on the screen.

In order to make code reading more fluid, error handling functionalities were preferably placed at the lowest level, considering the need for a modular system. Therefore, most of this type of handling is concentrated in "factory.py". This step was of great importance to obtain a better user experience, since it is undesirable system's failures due to incorrect input to cause its failure. Instead, the application became "catastrophic failure-proof" by always returning to the initial menu if it encounters any unplanned situation, a characteristic achieved by introducing the "is_online" class attribute for the client. The application is only terminated when this attribute has its state changed to the boolean "False" through the user interface.

Encryption: In order to apply blockchain concepts and maintain a fast running system without the need to create a completely secure system, a simple acceptance criteria was implemented for obtaining a hash using a limited number of combinations such as "abc", "012", etc. This decision was made to ensure a seamless user experience at the ATM, as searching for valid values to assemble the blockchain would have otherwise taken a significant amount of time.

Tests: It was not explicitly requested in the test statement the creation of automated tests, however, since this is an important part of any scalable system, evaluations of the main application functionalities were implemented and are available in the tests\unit_tests.py and tests\int_tests.py components.

Dependencies: Since it was requested in the test statement no type of framework to be used, it was chosen to extend the challenge and create a system with the smallest possible

amount of external libraries. This option may be desirable in making the application more robust against conflicts due to dependency versioning. System's dependencies are presented in the following list:

Package	Version	Package	Version
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	flask	2.2.3
cryptography	40.0.2		

Installation Guide

All commands can be executed through the command prompt of the machine used. To do so, simply initialize the prompt and change the directory to the location where you want to install it with the command "*cd path\to\folder*". Then, execute the following commands:

	Task	Command
1	Verify if the tools are installed	*$ docker -v* *$ docker-compose -v* *$ git –version* *$ pip –version*
2	Clone the repository containing source code	*$ git clone https://github.com/aogdrummond/konv_mini_bank.git*
3	Create virtual environment	*$ python -m venv konv_bank_venv*
4	Install the dependencies in virtual environment	*$ pip install -r requirements.txt*
5	(Optional) Determine the path for data persistence. Add the directory path to the “docker-compose.yml” file	*Adicionar o caminho do diretório ao arquivo “docker-compose.yml”*
6	Download the image and create the container with the database	*$ docker-compose up*
7	Activate virtual environment	*$ konv_bank_venv\Scripts\activate [Windows]* *ou* *$ konv_bank_venv\bin\activate [Linux\Mac]*
8	Run the application	*$ python main.py*