Banking App

Inspired by : Revolut mobile app.

The banking app is working with 2 types of users : Admin and Customer and has current account and savings.

When a customer creates an account he will be automatically be provided with 3 accounts (RON,EUR,USD) from which he can transfer, withdraw, deposit money.

Also he can create or add a 3 possible savings accounts with an initial annual interest rate of 2% which will be paid daily hence adding to the existing money and increasing it.

The project does not have any fees for specific monetary operations.

**The admin:**

The admin ca make C.R.U.D. operations on one searched by full name customer account.

He has the ability to add multiple admin accounts and also do some updates with customer accounts.

One customer account can be : blocked : the current account and also if existing saving accounts.

The big account , including the 3 current + savings can be deleted . Savings accounts can be one by one deleted , without affecting the big one but the 3 current cannot be deleted without deleting the whole main bank account.

The admin can set almost anything , besides IBANs and account numbers which are uniquely generated upon creation. He can modify balances , set/update interest rate.

The idea was for him to update the frequency rates : daily/weekly/monthly/yearly , as should had the customer when configuring the balance account , however I’ve decided to keep it simple and keep the accounts on a daily interest rate update schema. The project has a script that upon running the program ,launches a method that checks if there are accounts that have now been updated that particular day and if so , update them and set “last updated time” as current date. This way if the program is run again shortly after , it will see that that account has already been updated that day and there is no further need.

In the manage customer page there is a vertical list of text fields , in a similar manner as the register page has , where the customer can create the account and insert personal data , the admin can update the same data one by one .

In both scenarios the fields come with additional validation for checking inserted data for correctness ,errors, duplicability etc. In the register form , all of them will be verified as all must be non empty and correctly introduces .

The register form will be checked one by one , signaling the errors to the user , after full validation 3 accounts with unique ids and Ibans will be generated with 0 amount in each .

Admin must enter his password before any operation. Password as well as balances are visibly available to admin however password is hidden when customer whished to check his personal details.

**How is money represented in this project?**

The money is represented in Long integer type , without fraction format. Hence the arithmetic , representation , storage , precision, scalability (I’ve used a scale of 1.000.000 2.6 is represented as 2.600.000) .Long integers provide precise and accurate representation for large numbers without the rounding errors that can occur with floating-point arithmetic. This is particularly important in financial calculations where exact amounts are crucial.

Arithmetic operations on integers are generally faster than on floating-point numbers, because they do not involve processes like normalization and rounding. This can lead to better performance in applications that require intensive numeric computations.

Long integers can store very large numbers efficiently, using a fixed amount of memory. This makes them suitable for applications that require handling large numeric values without consuming excessive resources.

**The Customer:**

The customer after logging in , will be greeted by default in his RON account. The balance in RON will be displayed . This works in all 3 current accounts also , the customer can select which panel to showcase . There are buttons for showing personal data , current account data.

The main 3 operations are: Withdraw, Deposit, Transfer.

Depositing : the user enters the amount from 100 to 100.000 which it will be deposited in the account currently showing on the panel (selected one). There exists verification upon user data text imputed.

In real life, depositing money into an account typically involves interactions between different banks and financial institutions. If a customer has an account with a debit card in Revolut and wants to deposit money into that account. Lets take Revolut for example as this project is mainly inspired by it…

Bank Transfer:

* The customer can transfer money from another bank account to their Revolut account using bank transfer methods like SWIFT or SEPA. This involves providing the Revolut account's IBAN and BIC/SWIFT code.

Debit Card Top-Up:

* The customer can use their linked debit card from another bank to top up their Revolut account. This is often done through the Revolut app, where the customer enters their debit card details to add funds instantly.

Here , everything is simplified the user just enters a valid amount , meaning no fractional representation , a whole number. And then money will be added hence simulating the deposit operation effectively.

The same goes with withdraw , the customers enters an amount which will try , at first , to get from current selected account . If there is not enough money , the app will try to fetch it from the remaining 2 accounts 2 by one , this way the user can successfully withdraw amounts supported by all 3 accounts telling the user from where also money had to be taken from. If the amount cannot be withdraw from all 3 accounts put together , a warning will pop up. This the withdraw logic of the project , especially when the user is provided with exactly 3 accounts . However if there were more , probably would stop at current one and decline the operation. This way the user would have to use his other “pocket” accounts to exchange money between them and have sufficient funds in the one that matters or from external source.

Also an example from real life would be ATM withdrawals:

Using a Revolut debit card: The customer can use their Revolut debit card to withdraw cash from ATMs around the world. Revolut often offers fee-free withdrawals up to a certain limit each month.

Transfer goes like so:

The user will transfer money from his currently selected account . He can decide in which currency to send that balance and also , at least in this project , he can decide in which of the 3 recipient person`s accounts to send the money . Conversions between the money will be made , notifying how much money has been withdrawn and then sent. Here also applies the withdraw chain logic in case of insufficient balance. In real life one can chose from which account and with which currency to send.But receiving becomes more automated:If your base currency is GBP and you receive USD but don't have a USD account, it will be converted to GBP and if the recipient has a USD pocket account , it will be sent directly there based on currency match.

Also there is a Past Transfer button which upon clicking will show vertically , delimited by borders , the past transfers sent by the user from that selected account as well as received cash from someone into that current account . The name, username , currency with/to/from as well as exact time of the transfer are provided.

**Saving accounts:**

As explained earlier the customer can chose to open or not saving accounts in all 3 possible currencies. He will need to input a starting balance over 100 and it will be offered a starting 2% annual interest rate , paid daily . This interest rate can be modified by the admin.

Inside the saving account UI , the user can chose to withdraw/deposit money , exactly as in the current accounts. He can further add , in the remaining currencies , another saving account. If he has all possible 3 , will be notices . Here no transfer logic has been implemented.

The code was written using Java SWING. There are multiple classes , serving data base operations , logic operations or GUI operations or just holding GUI components.

The project uses in some parts inheritance and polymorphism hence reducing a part of DRY.

The database code has been written inside PostgreSQL.

There is a special class used to make the conversions between the 3 currencies , but the rate between them is static , not volatile and changing with the daily market.

**What I found challenging:**

Working with user typed data , there are so many cases and possibilities , especially when working with many text fields , must handle them separately on even in groups.

Almost everything regarding non “drag and drop” GUI , as writing code from scratch concerning gui , if there are many components and logic , it becomes crowded and code becomes heavily suffocating and long . I`ve tried my best of resolving this issues by using Component classes with inheritance and or polymorphism to reduce duplication.

**Further improvements ideas:**

The most important has to be : using the Currency Data Online API , working with multiple currencies would have been easier , not just the 3 . The user could choose from a considerable list of currencies and the conversions would be handled by the API , the application should only make requests/calls for different operations and data retrieval to the API. This way it would become more dynamic and up to date , especially generic and not bound to only 3 currencies.

Adding past view for deposit/withdraws.

Adding a friend/contact list from which the user can transfer money (again).

Exchange money between the “pocket” accounts.

Instead of warning the user that phone number or email are taken . A confirmation code should`ve sent and confirm the number or email and in the DB a column which will have the randomly generated code to be verified with.

A generally better looking GUI and more features for some parts: saving account , past transfer list.

Passwords store inside the DB should have been encrypted.