Banking System APP

Bunea Nicolae
October, 2025

The Banking System App is a WPF desktop application designed to simulate the operation of a basic banking system. The purpose of this project is to allow users to create banks, open accounts, and perform financial operations such as: deposits, withdrawals, and transfers.
The project also includes a complete transaction history log and persistent data storage using JSON serialization.

The application was developed in C# using Windows Presentation Foundation (WPF) and follows an organized architecture
with dedicated models, services, and UI components. Its goal is to demonstrate object-oriented programming,
event-driven UI development, and data management through serialization.

I developed this application as part of the Fundamentals concepts of Programming Languages laboratory requirements.

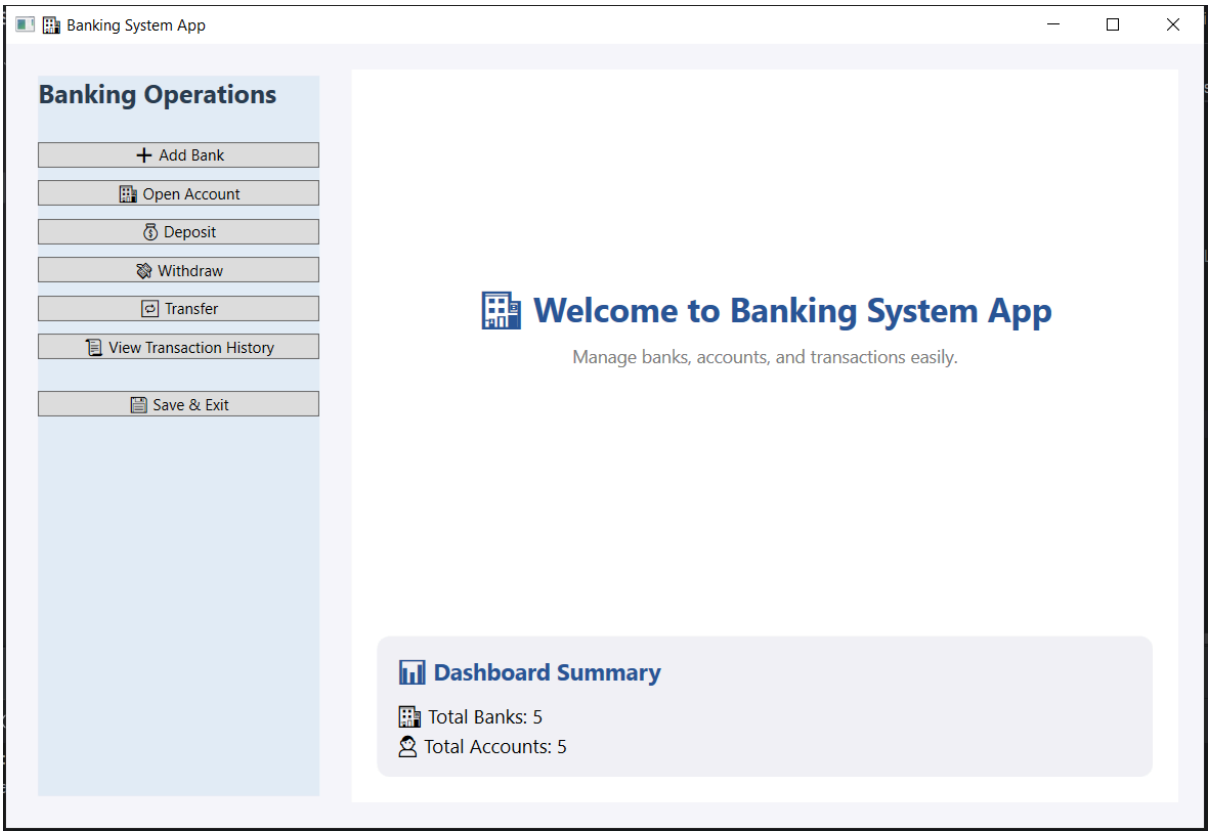
- Architecture:

The system is structured into several main components that ensure clean separation of responsibilities:

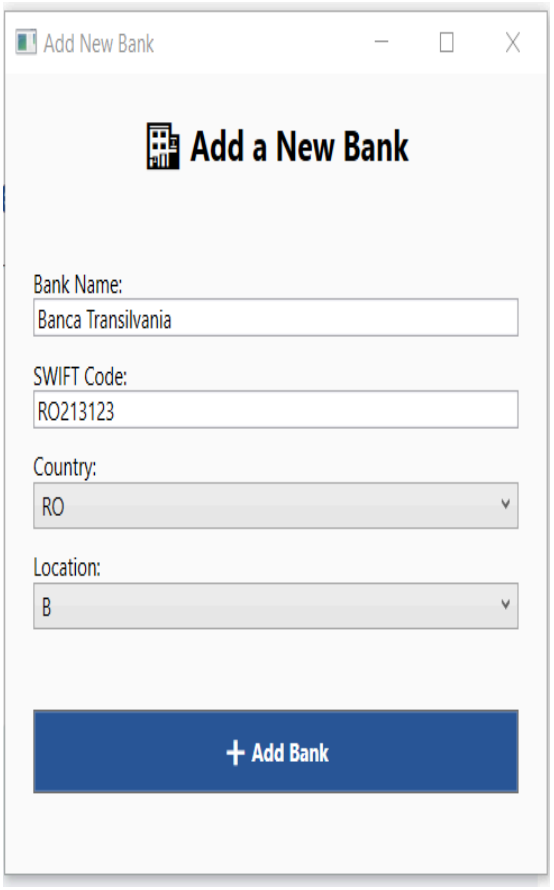
• Models – contain the data structures used in the system (Bank, Account).
• Services – implement logic for data handling, storage, and business operations.
• Enums – define constant types such as account currency, account type, bank country, etc.
• Windows – WPF XAML interfaces that users interact with (DepositWindow, TransferWindow, etc.).
• Data – contains the banks.json file which stores persistent data.

The application follows a service-based architecture, where UI components communicate with services that manage
data operations. JSON is used for persistent storage, allowing the application to save and reload all banks, accounts, and transactions between sessions.

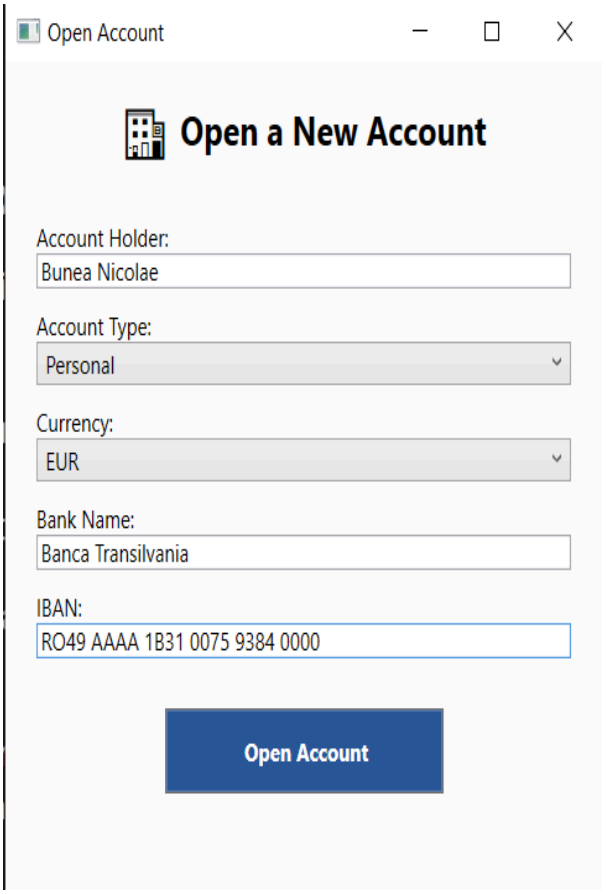
How it looks:

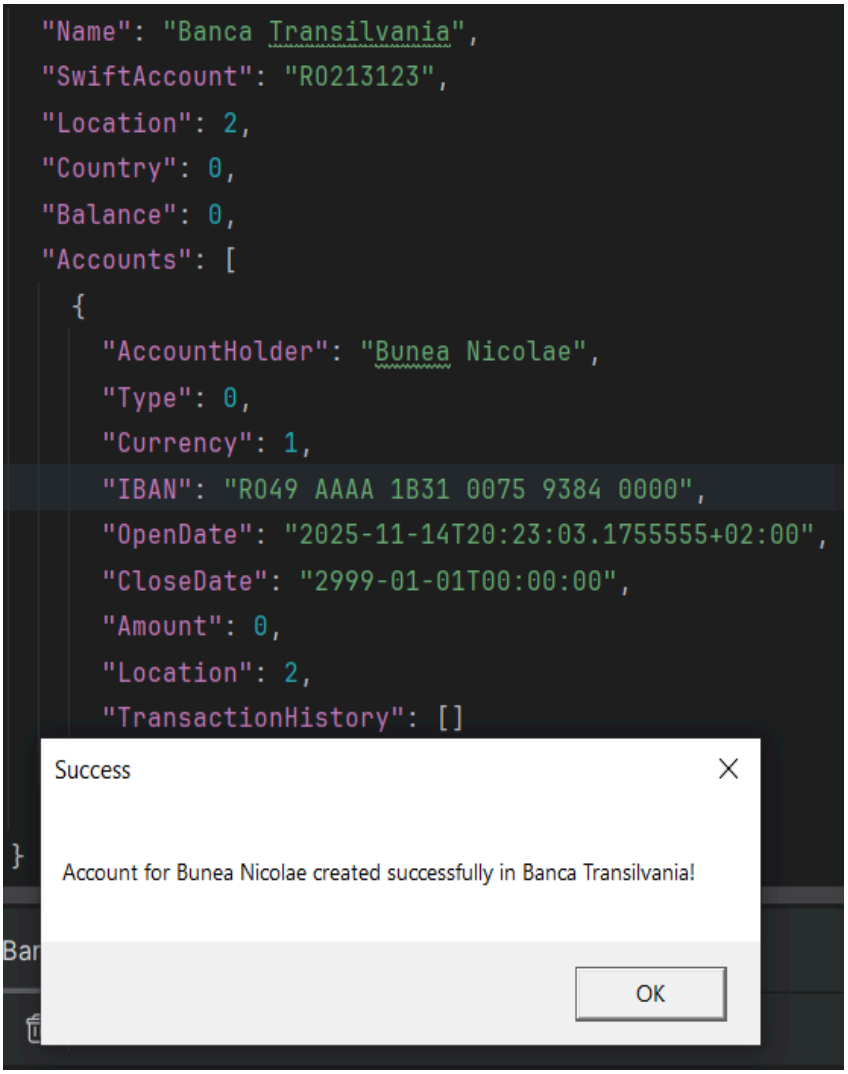


- Create a bank (Add bank):

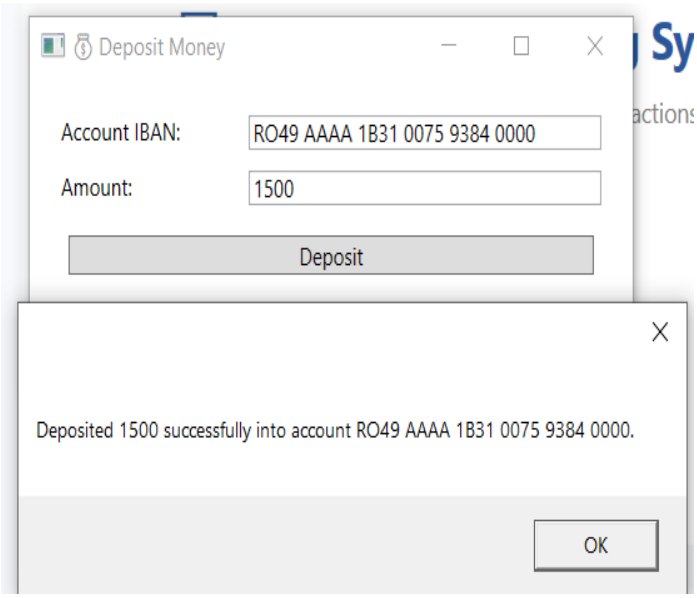


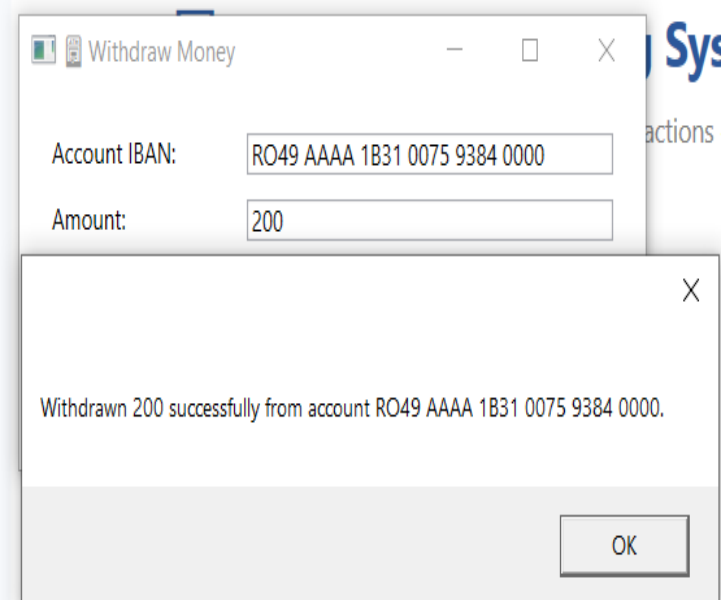
- Add an account into specified bank:



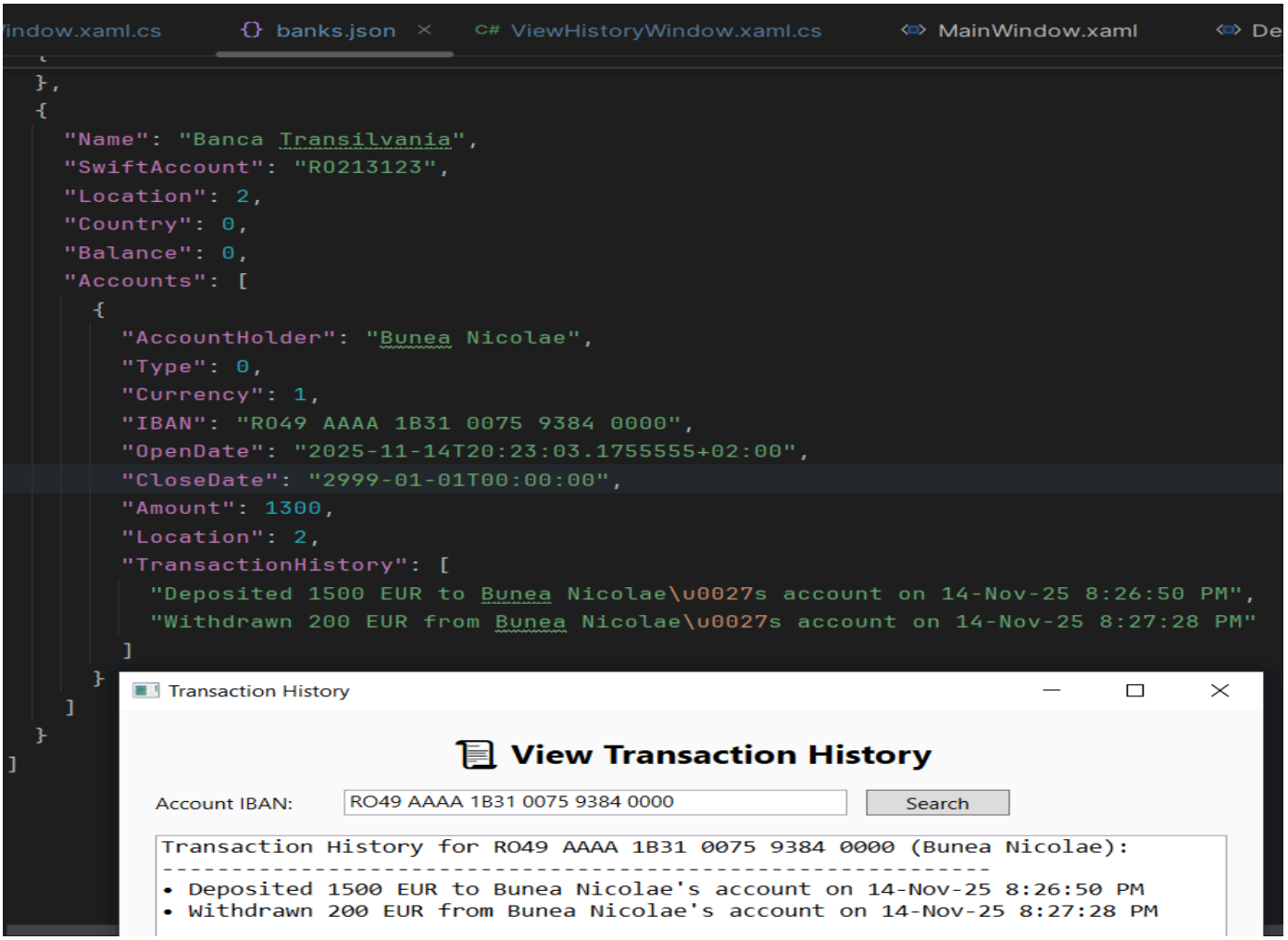


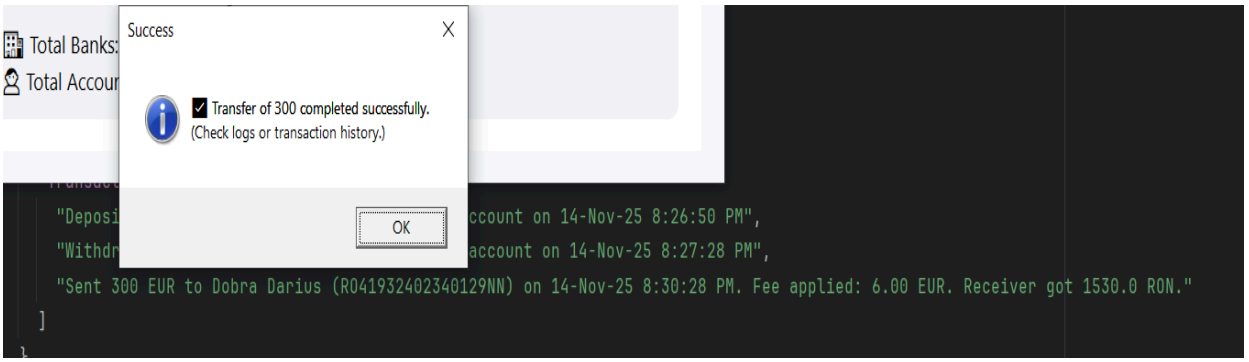
- Deposit / Withdraw





- Everything updated in JSON, and can be seen in Transaction History:

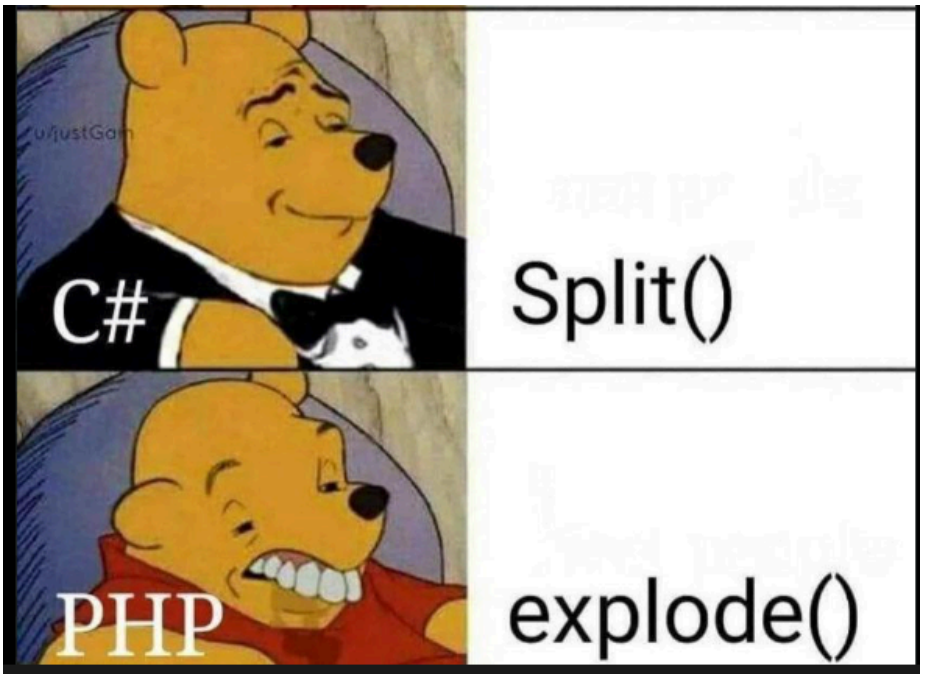




The application stores its data in a JSON file using the JsonStorageService class. Whenever an operation is performed (adding a bank, depositing money, creating an account), the service updates the in-memory data and then saves it to the JSON file.

The BankService class contains all operational logic:
• Adding banks and accounts
• Processing deposits and withdrawals
• Validating transfers
• Applying fees to transactions
• Updating transaction histories

Each account stores its history as a list of strings, and each operation automatically generates a timestamped message.



Enjoyed working with C#

THE END