

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.

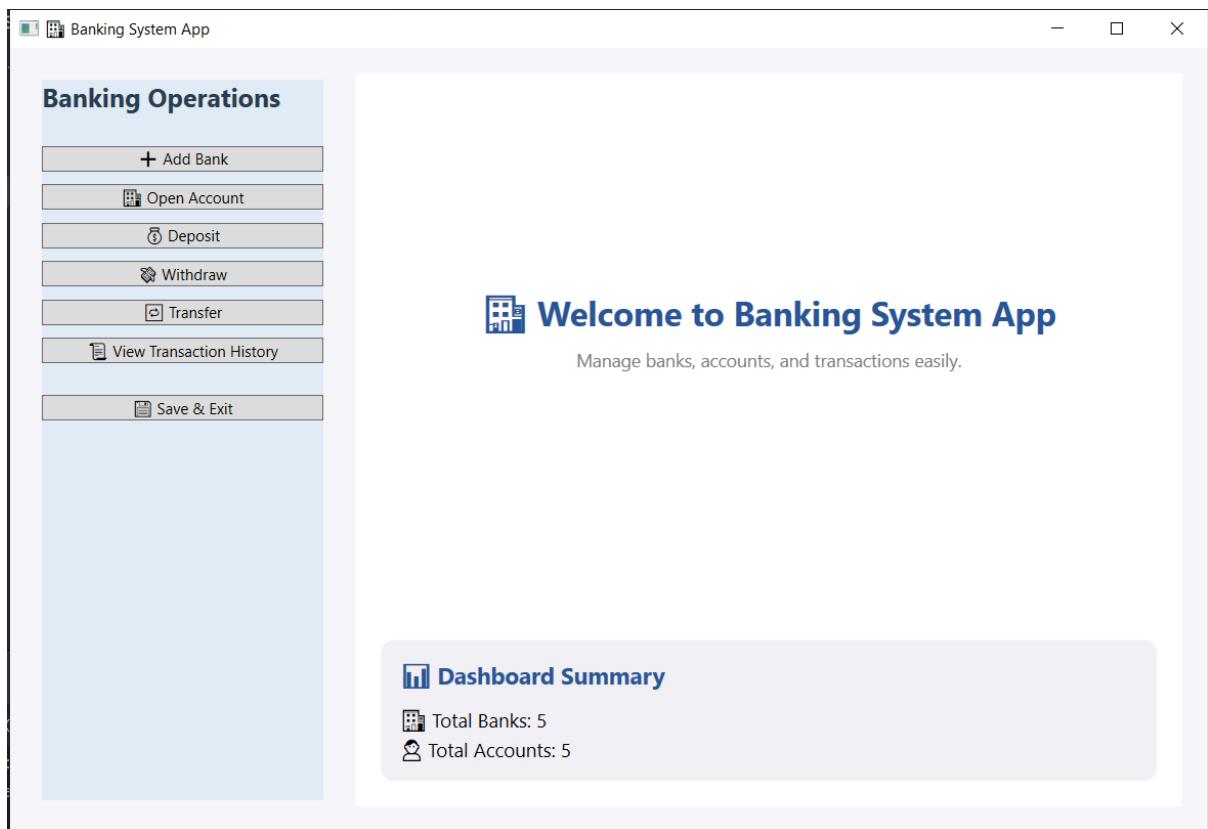
- **Architecture:**

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):

The screenshot shows the "Add New Bank" dialog box. It has fields for "Bank Name" (Banca Transilvania), "SWIFT Code" (RO213123), "Country" (RO), and "Location" (B). At the bottom is a blue button labeled "+ Add Bank".

The screenshot shows a code editor with Java code and a terminal window. The code defines a class with attributes: Name (Banca Transilvania), SwiftAccount (RO213123), Location (2), Country (0), Balance (0), and Accounts (empty array). The terminal window shows the application running and outputting "Data loaded successfully" and "Bank: Banca Transilvania added successfully in RO".

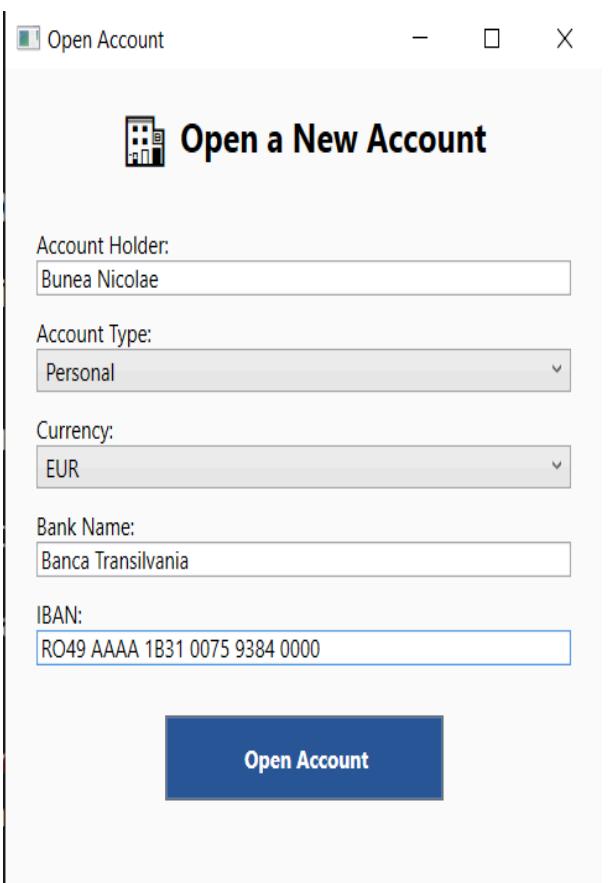
```
144 "Name": "Banca Transilvania",
145 "SwiftAccount": "RO213123",
146 "Location": 2,
147 "Country": 0,
148 "Balance": 0,
149 "Accounts": []
150 }
151 1
```

Run BankingSystemApp

Data loaded successfully

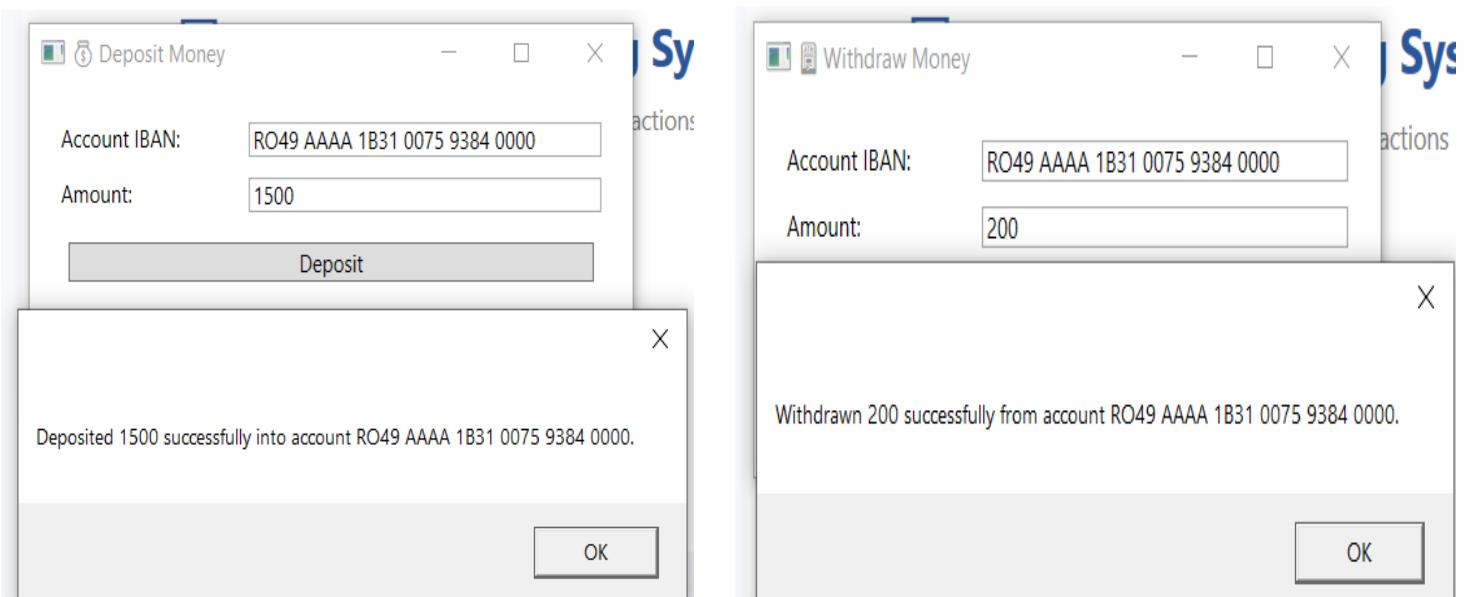
Bank: Banca Transilvania added successfully in RO

- Add an account into specified bank:



"Name": "Banca Transilvania",
 "SwiftAccount": "R0213123",
 "Location": 2,
 "Country": 0,
 "Balance": 0,
 "Accounts": [
 {
 "AccountHolder": "Bunea Nicolae",
 "Type": 0,
 "Currency": 1,
 "IBAN": "RO49 AAAA 1B31 0075 9384 0000",
 "OpenDate": "2025-11-14T20:23:03.1755555+02:00",
 "CloseDate": "2999-01-01T00:00:00",
 "Amount": 0,
 "Location": 2,
 "TransactionHistory": []
 }]
 Success
 Account for Bunea Nicolae created successfully in Banca Transilvania!

- Deposit / Withdraw



Deposit Money

Deposit

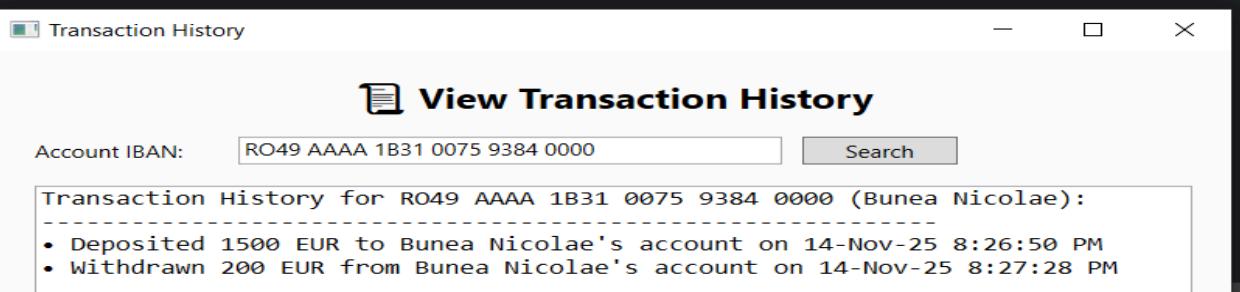
Deposited 1500 successfully into account RO49 AAAA 1B31 0075 9384 0000.

Withdraw Money

Withdraw

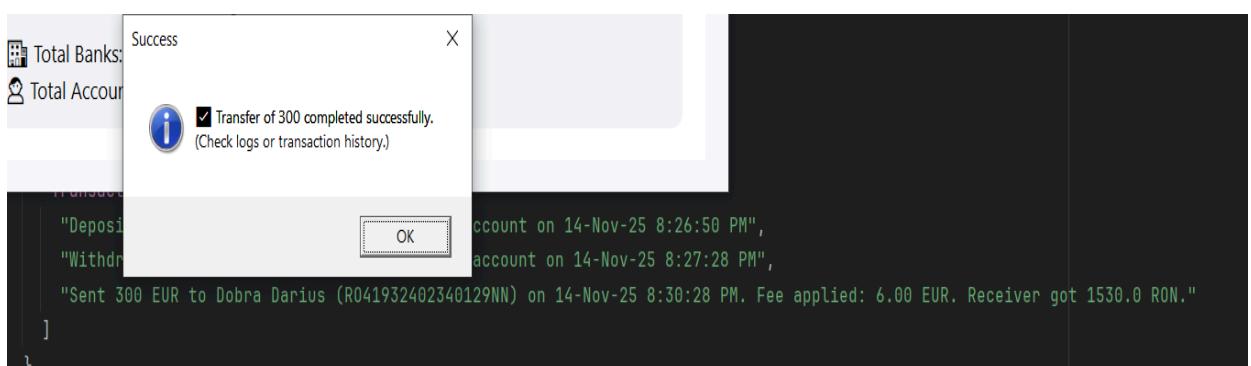
Withdrawn 200 successfully from account RO49 AAAA 1B31 0075 9384 0000.

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



The screenshot shows a Windows application window titled "View Transaction History". Inside, there is a search bar with the account IBAN "RO49 AAAA 1B31 0075 9384 0000" and a "Search" button. Below the search bar, a section titled "Transaction History for RO49 AAAA 1B31 0075 9384 0000 (Bunea Nicolae):" displays two transactions:

- Deposited 1500 EUR to Bunea Nicolae's account on 14-Nov-25 8:26:50 PM
- Withdrawn 200 EUR from Bunea Nicolae's account on 14-Nov-25 8:27:28 PM

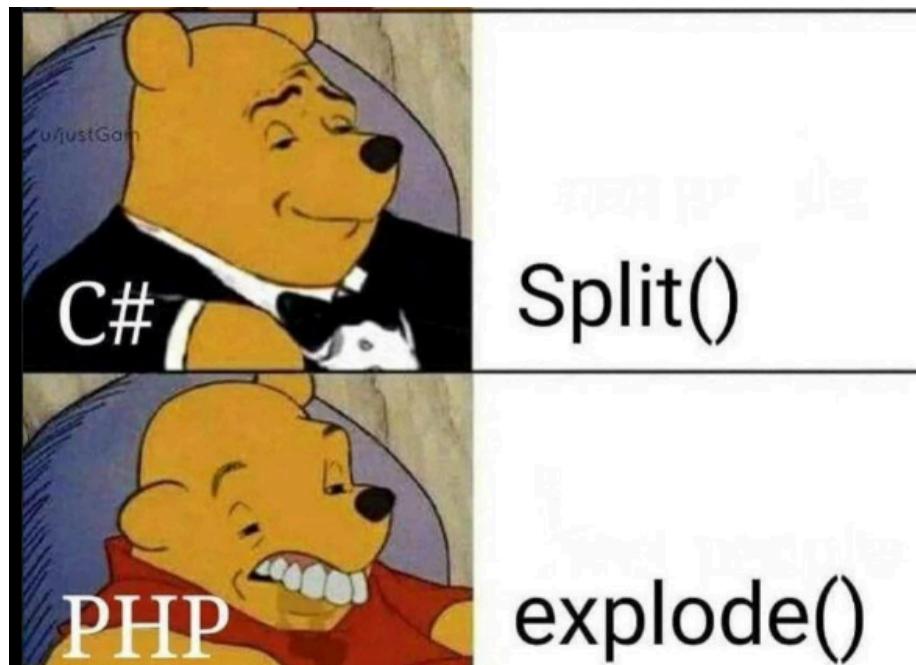


The application stores its data in a JSON file using the `JsonStorageService` class. Whenever an operation is performed (adding a bank, depositing money, creating an account), the service updates the in-memory data and then saves it to the JSON file.

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