Assignment 5 - Python Banking System

2020310083 Hyungjun Shon

Dept. of System Management Engineering
Dept. of Software
Sungkyunkwan University

November 20, 2024

Contents

1 Objective

Make a Python Banking System that is a simple console-based application for managing user accounts and transactions. It includes features such as user registration, login, deposit, withdrawal, transfer, and viewing transaction history. The application utilizes JSON files for persistent data storage and operates within a structured directory framework.

2 Features

1. User Registration:
   - Allows new users to register with a username, password, and PIN.
   - Generates a unique account number for each user.
   - Initializes the account balance to $100.
   - Creates a transaction file for each user.
2. User Login:
   - Authenticates users based on their username and password.
   - Redirects authenticated users to the main menu.
3. Deposit, Withdraw, and Transfer:
   - Enables users to manage their balance by depositing, withdrawing, or transferring funds.
   - Validates user input, including PINs and sufficient balances.
4. Transaction History:
   - Displays a detailed log of all transactions performed by the user.
5. Data Persistence:
   - Stores user information in a central `user_data.json` file.
   - Logs transactions in individual JSON files for each user.
6. Error Handling:
   - Includes validation for inputs such as passwords, PINs, and amounts, . . .
   - Displays user-friendly error messages for invalid operations.

3 Architecture of `JSON` files

The application is designed with simplicity and modularity in mind, leveraging JSON files for persistent data storage. Below is a detailed breakdown of the architecture:

3.1 File Structure

3.2 `JSON` file format

```
./json/
|
+-- user_data.json            # Stores all user information in a dictionary.
+-- transactions/             # Directory containing individual transaction files.
    +-- user1.json            # Transaction history for user1.
    +-- user2.json            # Transaction history for user2.
    +-- user3.json            # Transaction history for user3.
    +-- ...                   # Additional transaction files for other users.
```

3.2.1 User Data:

- File: `user_data.json`
- Format: A dictionary where each key is a username and the value is another dictionary containing user details.
- Example:

```
{
  "user1": {
    "username": "user1",
    "password": "TEST!1234",
    "pin": "1234",
    "account_number": "82404",
    "balance": 157.0
  },
```

```
  "user2": {
    "username": "user2",
    "password": "TEST!5678",
    "pin": "5678",
    "account_number": "62056",
    "balance": 98.0
  },
  "user3": {
    "username": "user3",
    "password": "TEST!1234",
    "pin": "1234",
    "account_number": "71704",
    "balance": 100
  }
}
```

3.2.2 Transaction Logs:

- File: `transactions/<username>.json` (stored individually for each user to ensure privacy and maintain consistency)
- Format: A list of dictionaries, each representing a transaction of each user.
- Example: (`user1.json`)

```
[
  {
    "time": "2024-11-20 11:06:18",
    "type": "Transfer",
    "from": "user2",
    "to": "user1",
    "amount": 57.0
  },
  {
    "time": "2024-11-20 11:05:32",
    "type": "Transfer",
    "from": "user1",
    "to": "user2",
    "amount": 35.0
  },
  {
    "time": "2024-11-20 11:05:00",
    "type": "Deposit",
    "amount": 45.0
  },
  {
    "time": "2024-11-20 11:04:38",
    "type": "Withdraw",
    "amount": 10.0
  }
]
```

4 Function Explanation

4.1 Helper Functions

4.1.1 `clear_screen()`

```python
def clear_screen():
    """
    Clears the terminal screen.
    """

    os.system("cls" if os.name == "nt" else "clear")
```

- **Purpose**: Clears the terminal screen for a clean user interface.
- **Logic**:

  - Uses `os.system()` to determine the operating system and executes the appropriate command:
    * `cls` for Windows.
    * `clear` for Unix-based systems (Linux, macOS).

- **Usage**: Called before menus and user inputs to avoid clutter.

4.1.2 `load_data(file_name)`

```python
def load_data(file_name):
    """
    Load data from JSON file. If the file doesn't exist, return an empty dictionary.

    @param: file_name (str): name of the file to load data from
    @return: data (dict or list): data from the files
    """

    # if the file do not exist, create with empty data
    if not os.path.exists(file_name):
        if file_name == USER_DATA_FILE:
            save_data({}, file_name)  # dict for user_data.json
        else:
            save_data([], file_name)  # list for transaction files

    # if the file exists, load the data from the file
    with open(file_name, "r") as file:
        return json.load(file)
```

- **Purpose**: Reads data from a specified JSON file. If the file does not exist, it creates an empty file and returns an empty dictionary.
- **Parameters**:
  - `file_name` (str): The name of the file to read.
- **Logic**:
  - Checks if the file exists using `os.path.exists(file_name)`.
  - If the file does not exist:
    * Creates the file with empty data.
    * if `file_name` is `user_data.json`, it saves an empty dictionary.
    * if `file_name` is a transaction file, it saves an empty list.
  - If the file exists:
    * Opens the file in read mode (`'r'`).
    * Reads and parses the JSON data using `json.load()`.
- **Returns**:
  - A dictionary or list parsed from the file.
- **Usage**: Used to load `user_data.json` and individual transaction files.

4.1.3 `save_data(data, file_name)`

```python
def save_data(data_to_save, file_name):
    """
    Save data to a JSON file.

    @param: data (dict): data to save to the file
    @param: file_name (str): name of the file to save the data to
    """

    try:
        with open(file_name, "w") as file:
            json.dump(data_to_save, file)
    except IOError as e:
        print(f"Error saving data: {e}")
    return
```

- **Purpose**: Saves data to a specified JSON file.
- **Parameters**:
  - `data_to_save` (dict or list): The data to save.

  - `file_name` (str): The name of the file to save the data in.
- **Logic**:
  - Opens the file in write mode (`'w'`) and uses `json.dump()` to write the data.
  - Handles `IOError` exceptions to report save failures.
- **Usage**: Saves user data (`user_data.json`) and transaction logs.

4.1.4 generate_account_number(user_data)

```python
def generate_account_number(user_data):
    """
    Generate a unique account number for a new user (5 random digits).

    @param: user_data (dict): dictionary of existing users
    @return: account_number (str): unique account number for the new user
    """

    while True:
        account_number = str(random.randint(10000, 99999))

        # check if the account number is unique
        if not any(
            user["account_number"] == account_number for user in user_data.values()
        ):
            return account_number
```

- **Purpose**: Generates a unique 5-digit account number for new users.
- **Parameters**:
  - `user_data` (dict): A dictionary containing all existing users.
- **Logic**:
  - Uses `random.randint(10000, 99999)` to generate a random 5-digit number.
  - Checks if the generated number is already associated with an existing user.
  - Repeats the process until a unique account number is generated.
- **Returns**:
  - A unique 5-digit account number.
- **Usage**: Ensures account numbers do not collide during user registration.

4.1.5 is_password_valid(password)

```python
def is_password_valid(password):
    """
    Check password validity.

    @param: password (str): password to check
    @return: errors (list): list of errors if the password is invalid
    """

    errors = []
    if len(password) < 7:
        errors.append("Password must be at least 7 characters long.")
    if not any(char.isupper() for char in password):
        errors.append("Password must contain at least one uppercase letter.")
    if not any(char in SPECIAL_CHARACTERS for char in password):
        errors.append("Password must contain at least one special character.")
    return errors
```

- **Purpose**: Validates the strength of a password based on predefined rules.
- **Parameters**:
  - `password` (str): The password to validate.
- **Logic**:
  - Checks for the following:

        * Minimum length of 7 characters.
        * At least one uppercase letter.
        * At least one special character from `SPECIAL_CHARACTERS`.
    - Collects errors for any violated rules.
- **Returns**: errors list
- **Usage**: Ensures secure passwords during registration.

4.1.6 `is_pin_valid(pin)`

```python
def is_pin_valid(pin):
    """
    Check PIN validity (must be 4 digits).

    @param: pin (str): PIN to check
    @return: valid (bool): True if the PIN is valid, False otherwise
    """

    return len(pin) == 4 and pin.isdigit()
```

- **Purpose**: Validates the format of a PIN.
- **Parameters**:
    - `pin` (str): The PIN to validate.
- **Logic**:
    - Checks if the PIN is exactly 4 characters long and consists only of digits.
- **Returns**:
    - `True` if valid.
    - `False` otherwise.
- **Usage**: Used during registration and financial operations for authentication.

4.1.7 `log_transaction(username, transaction)`

```python
def log_transaction(username, transaction):
    """
    Log a transaction in the user's individual transaction file.

    @param: username (str): username of the user
    @param: transaction (dict): transaction to log
    """

    # load the transaction file
    transaction_file = f"{TRANSACTION_DIR}/{username}.json"
    transactions = load_data(transaction_file)

    # insert the transaction at the beginning of the list and save the file
    transactions.insert(0, transaction)
    save_data(transactions, transaction_file)


def validate_amount(amount_str):
    """
    Validate and convert amount input.
    """

    # check if the amount is a valid number and greater than 0
    try:
        amount = float(amount_str)
        if amount <= 0:
            return False, "Amount must be greater than 0."
        return True, amount
    except ValueError:
        return False, "Invalid amount format."
```

- **Purpose**: Logs a transaction in the user's individual transaction file.

- **Parameters**:
  - `username` (str): The user associated with the transaction.
  - `transaction` (dict): A dictionary containing transaction details (e.g., type, amount, timestamp).
- **Logic**:
  - Determines the file path for the user's transaction file.
  - Loads existing transactions using `load_data()`.
  - Adds the new transaction to the beginning of the list.
  - Saves the updated list using `save_data()`.
- **Usage**: Ensures all transactions are recorded persistently.

4.1.8 `validate_amount(amount_str)`

```python
def validate_amount(amount_str):
    """
    Validate and convert amount input.
    """

    # check if the amount is a valid number and greater than 0
    try:
        amount = float(amount_str)
        if amount <= 0:
            return False, "Amount must be greater than 0."
        return True, amount
    except ValueError:
        return False, "Invalid amount format."
```

- **Purpose**: Validates and converts an amount input to a float.
- **Parameters**:
  - `amount_str` (str): The amount to validate.
- **Logic**:
  - Checks if the amount is a valid number and greater than 0.
  - Converts the amount to a float.
  - Returns a tuple containing a boolean and the amount or an error message.
- **Usage**: Used for validating monetary inputs in financial operations.

4.2 User Management Functions

4.2.1 `register_user(user_data)`

```python
def register_user(user_data):
    """
    Register a new user and create their transaction file.

    @param: user_data (dict): dictionary of existing users
    """

    clear_screen()

    # get username and check if it is already taken
    print("\nRegister a new user:")
    username = input("Enter username: ")
    if username in user_data:
        print("Error: Username already exists.")
        input("\nPress Enter to return to the menu.")
        return

    # get password and validate it
    password = input("Enter login password: ")
    if len(errors := is_password_valid(password)):
        print("Invalid password:")
        for error in errors:
            print(f"- {error}")
        return input("\nPress Enter to return to the menu.")

```

```
    # get PIN and validate it
    pin = input("Enter a 4-digit PIN: ")
    if not is_pin_valid(pin):
        print("Error: PIN must be a 4-digit number.")
        return input("\nPress Enter to return to the menu.")

    # generate a unique account number for the new user
    account_number = generate_account_number(user_data)

    # create a new user in the user_data dictionary and save it to user_data file
    user_data[username] = {
        "username": username,
        "password": password,
        "pin": pin,
        "account_number": account_number,
        "balance": 100,
    }
    save_data(user_data, USER_DATA_FILE)

    # create an empty transaction file for the new user
    save_data([], f"{TRANSACTION_DIR}/{username}.json")

    print(f"Registration successful! Your account number is {account_number}.")
    input("\nPress Enter to return to the menu.")
```

- **Purpose**: Registers a new user with a unique username, password, PIN, and account number.
- **Parameters**:
  - `user_data` (dict): Dictionary of existing users.
- **Logic**:
  - Prompts the user for a unique username. If the username already exists, aborts registration.
  - Validates the password and PIN using `is_password_valid()` and `is_pin_valid()`.
  - Generates a unique account number using `generate_account_number()`.
  - Adds the user to `user_data` with an initial balance of $100.
  - Creates an empty transaction file for the new user.
  - Saves the updated `user_data` to `user_data.json`.
- **error handling**:
  - If the username already exists, displays an error message and returns.
  - If the password is invalid, displays error messages and returns.
  - If the PIN is invalid, displays an error message and returns.
- **Usage**: Ensures a streamlined user registration process.

4.2.2 login_user(user_data)

```
def login_user(user_data):
    """
    Log in an existing user.
    """

    clear_screen()

    print("\nLogin:")
    username = input("Enter username: ")
    password = input("Enter password: ")

    # check if the username is in the user_data dictionary and the password is corrects
    if username not in user_data or user_data[username]["password"] != password:
        print("Error: Invalid username or password.")
        input("\nPress Enter to return to the menu.")
        return None
    return username
```

- **Purpose**: Authenticates an existing user.
- **Parameters**:
  - `user_data` (dict): Dictionary of existing users.

- **Logic**:
  - Prompts the user for their username and password.
  - Checks if the username exists in `user_data` and if the password matches.
  - If valid, returns the username.
  - If invalid, displays an error and returns `None`.
- **Returns**:
  - The username if login is successful.
  - `None` otherwise.
- **Usage**: Enables access control for the banking system.

4.3 Financial Operation Functions

4.3.1 `show_history(username)`

```python
def show_history(username):
    """
    Show the transaction history for a user.
    """

    clear_screen()

    # load the transaction file
    transaction_file = f"{TRANSACTION_DIR}/{username}.json"
    transactions = load_data(transaction_file)

    # print the transaction history
    print("\n--- Transaction History ---\n")
    if not transactions:
        print("No transactions found.")
        return input("\nPress Enter to return to the menu.")

    for transaction in transactions:
        if transaction["type"] == "Transfer":
            print(f"[{transaction['time']}]\n")
            print(f"- Type: {transaction['type']}")
            print(f"- From: {transaction['from']}")
            print(f"- To: {transaction['to']}")
            print(f"- Amount: ${transaction['amount']}")
        else:
            print(f"[{transaction['time']}]\n")
            print(f"- Type: {transaction['type']}")
            print(f"- Amount: ${transaction['amount']}")
        print("-" * 50)
    return input("\nPress Enter to return to the menu.")
```

- **Purpose**: Displays the transaction history of a user.
- **Parameters**:
  - `username` (str): The username of the user.
- **Logic**:
  - Loads the user's transaction file using `load_data()`.
  - If the file is empty, displays a message indicating no transactions.
  - Iterates through the transactions and prints details with appropriate formatting.
- **Usage**: Provides users with a clear view of their financial activities.

4.3.2 `withdraw(user_data, username)`

```python
def withdraw(user_data, username):
    """
    Withdraw money for the logged-in user.
    """

    clear_screen()

    # get the user from the user_data dictionary
```

```
    user = user_data[username]
    print("\nWithdraw Money:")

    # get the amount and validate it
    amount_str = input("Enter amount to withdraw: ")
    valid, result = validate_amount(amount_str)
    if not valid:
        return input(f"Error: {result}. Press Enter to return to the menu.")
    amount = result

    # check if the amount is greater than the user's balance
    if amount > user["balance"]:
        return input("Error: Insufficient balance. Press Enter to return to the menu.")

    # get the PIN and check if it is correct
    pin = input("Enter PIN: ")
    if pin != user["pin"]:
        return input("Error: Incorrect PIN. Press Enter to return to the menu.")

    # deduct the amount from the user's balance
    user["balance"] -= amount

    # log the transaction in the transaction file
    log_transaction(
        username,
        {
            "time": datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
            "type": "Withdraw",
            "amount": amount,
        },
    )
    save_data(user_data, USER_DATA_FILE)
    print(f"Withdrawal successful. New balance: ${user['balance']}")
    input("\nPress Enter to return to the menu.")
    return
```

- **Purpose**: Withdraws money from the user's account.
- **Parameters**:
  - `user_data` (dict): Dictionary of existing users.
  - `username` (str): The username of the user performing the withdrawal.
- **Logic**:
  - Validates the withdrawal amount using `validate_amount()`.
  - Checks if the user has sufficient balance.
  - Authenticates the operation using the user's PIN.
  - Deducts the amount from the user's balance.
  - Logs the transaction using `log_transaction()`.
  - Saves the updated `user_data`.
- **error handling**:
  - if any error occur during the widthdrawal process, displays an error message and returns.
- **Usage**: Allows secure withdrawals.

4.3.3 deposit(user_data, username)

```
def deposit(user_data, username):
    """
    Deposit money for the logged-in user.
    """

    clear_screen()
    # get the user from the user_data dictionary
    user = user_data[username]
    print("\nDeposit Money:")

    # get the amount and validate it
    amount_str = input("Enter amount to deposit: ")
    valid, result = validate_amount(amount_str)
    if not valid:
```

```
        return input(f"Error: {result}. Press Enter to return to the menu.")
    amount = result

    # get the PIN and check if it is correct
    pin = input("Enter PIN: ")
    if pin != user["pin"]:
        return input("Error: Incorrect PIN. Press Enter to return to the menu.")

    # add the amount to the user's balance
    user["balance"] += amount

    # log the transaction in the transaction file
    log_transaction(
        username,
        {
            "time": datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
            "type": "Deposit",
            "amount": amount
        },
    )
    save_data(user_data, USER_DATA_FILE)
    print(f"Deposit successful. New balance: ${user['balance']}")
    input("\nPress Enter to return to the menu.")
```

- **Purpose**: Deposits money into the user's account.
- **Parameters**:
  - `user_data` (dict): Dictionary of existing users.
  - `username` (str): The username of the user performing the deposit.
- **Logic**:
  - Validates the deposit amount using `validate_amount()`.
  - Authenticates the operation using the user's PIN.
  - Adds the amount to the user's balance.
  - Logs the transaction using `log_transaction()`.
  - Saves the updated `user_data`.
- **error handling**:
  - if any error occur during the deposit process, displays an error message and returns.
- **Usage**: Supports seamless deposits.

4.3.4 `transfer(user_data, username)`

```
def transfer(user_data, username):
    """
    Transfer money to another user.
    """

    clear_screen()
    # get the user from the user_data dictionary
    user = user_data[username]
    print("\nTransfer Money:")

    # get the recipient's account number
    account_number = input("Enter recipient's account number: ")

    # find the recipient by account number
    recipient = next(
        (
            user
            for user in user_data.values()
            if user["account_number"] == account_number
        ),
        None,
    )

    # check if the recipient is found and is not the same as the sender
    if not recipient:
        return input(
            "Error: Recipient account not found. Press Enter to return to the menu."
```

```
        )

    if account_number == user["account_number"]:
        return input(
            "Error: You cannot transfer money to your own account. Press Enter to return to the menu."
        )

    # get the amount and validate it
    amount_str = input("Enter amount to transfer: ")
    valid, result = validate_amount(amount_str)
    if not valid:
        return input(f"Error: {result}. Press Enter to return to the menu.")
    amount = result

    # check if the amount is greater than the user's balance
    if amount > user["balance"]:
        return input("Error: Insufficient balance. Press Enter to return to the menu.")

    # get the PIN and check if it is correct
    pin = input("Enter PIN: ")
    if pin != user["pin"]:
        return input("Error: Incorrect PIN. Press Enter to return to the menu.")

    # Deduct from sender and add to recipient
    user["balance"] -= amount
    recipient["balance"] += amount

    # Log transactions for both sender and recipient to their transaction files
    log_transaction(
        username,
        {
            "time": datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
            "type": "Transfer",
            "from": username,
            "to": recipient["username"],
            "amount": amount,
        },
    )
    log_transaction(
        recipient["username"],
        {
            "time": datetime.now().strftime("%Y-%m-%d %H:%M:%S"),
            "type": "Transfer",
            "from": username,
            "to": recipient["username"],
            "amount": amount,
        },
    )
    save_data(user_data, USER_DATA_FILE)
    print(f"Transfer successful. New balance: ${user['balance']}")
    input("\nPress Enter to return to the menu.")
```

- **Purpose**: Transfers money from one user to another.
- **Parameters**:
  - `user_data` (dict): Dictionary of existing users.
  - `username` (str): The username of the sender.
- **Logic**:
  - Validates the recipient's account number.
  - Ensures the recipient is not the sender.
  - Validates the transfer amount using `validate_amount()`.
  - Checks if the sender has sufficient balance.
  - Authenticates the operation using the sender's PIN.
  - Deducts the amount from the sender's balance and adds it to the recipient's balance.
  - Logs transactions for both sender and recipient.
  - Saves the updated `user_data`.
- **error handling**:
  - if any error occur during the transfer process, displays an error message and returns.
- **Usage**: Facilitates secure and traceable transfers.

4.4 Main Program Functions

4.4.1 main_screen(user_data, username)

```python
def main_screen(data, username):
    """
    Main screen for a logged-in user.
    """

    # get the user from the user_data dictionary
    user = data[username]

    # show the main menu
    while True:
        try:
            clear_screen()
            print("=== Main Menu ===")
            print(f"Welcome, {username}")
            print(f"Account: {user['account_number']}")
            print(f"Balance: ${user['balance']:.2f}")
            print("\nOptions:")
            print("1. View Transaction History")
            print("2. Withdraw Money")
            print("3. Deposit Money")
            print("4. Transfer Money")
            print("5. Logout")

            # get the choice and validate it
            choice = input("\nChoose an option (1-5): ").strip()
            if choice == "1":
                show_history(username)
            elif choice == "2":
                withdraw(data, username)
            elif choice == "3":
                deposit(data, username)
            elif choice == "4":
                transfer(data, username)
            elif choice == "5":
                input(
                    "Successfully logged out. Press Enter to return to the initial menu."
                )
                return
            else:
                input("Invalid option. Please press Enter to try again.")
                continue

        except Exception as e:
            print(f"An error occurred: {e}")
            input("\nPress Enter to return to the menu.")
            continue
```

- **Purpose**: Displays the main menu for logged-in users and handles menu actions.
- **Parameters**:
  - `user_data` (dict): Dictionary of existing users.
  - `username` (str): The username of the logged-in user.
- **Logic**:
  - Continuously displays the menu until the user logs out.
  - Prompts the user for a choice and maps it to corresponding actions (e.g., viewing history, depositing money).
  - Executes the chosen action.
- **error handling**:
  - if choose invalid option, displays an error message and continue.
- **Usage**: Serves as the central hub for user operations.

4.4.2 init_screen()

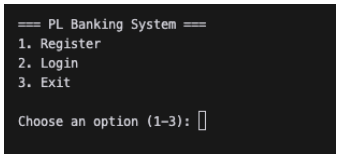
```python
def init_screen():
    """
```

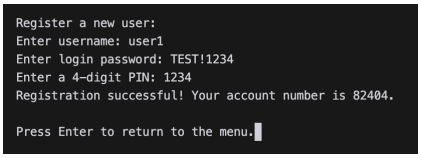
```
    Initial screen.
    """

    while True:
        clear_screen()
        user_data = load_data(USER_DATA_FILE)
        print("\n=== PL Banking System ===")
        print("1. Register")
        print("2. Login")
        print("3. Exit")
        choice = input("\nChoose an option (1-3): ").strip()

        # define the actions for each choice and execute them
        if choice == "1":
            register_user(user_data)
        elif choice == "2":
            username = login_user(user_data)
            if username:
                main_screen(user_data, username)
        elif choice == "3":
            print("Exiting the program. Goodbye!")
            break
        else:
            input("Invalid option. Please press Enter to try again.")
```

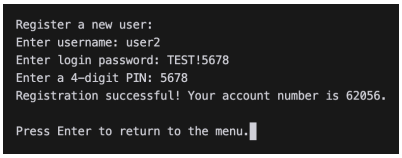
- **Logic**:
  - Displays the initial menu (Register, Login, Exit).
  - Handles user registration and login.
  - Calls `main_screen()` for authenticated users.
  - Terminates the program on exit.
- **error handling**:
  - if choose invalid option, displays an error message and returns.
- **Usage**: Ensures smooth navigation and user flow.

5 Screenshots

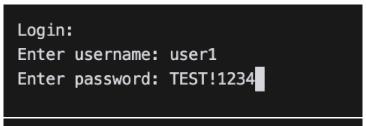
5.1 register 2 users

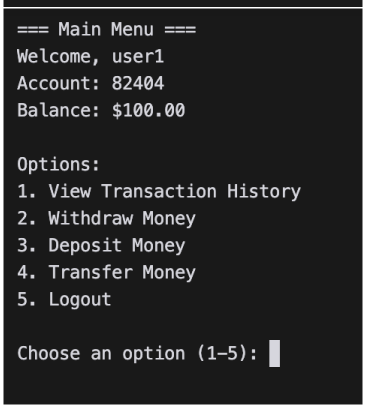




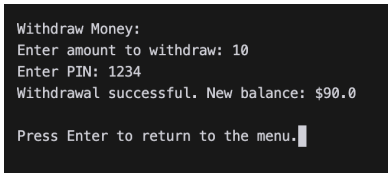


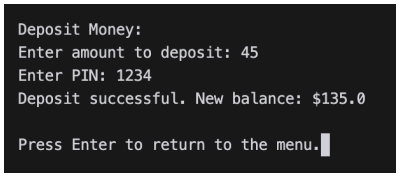
5.2 login to user1

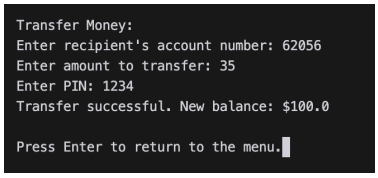




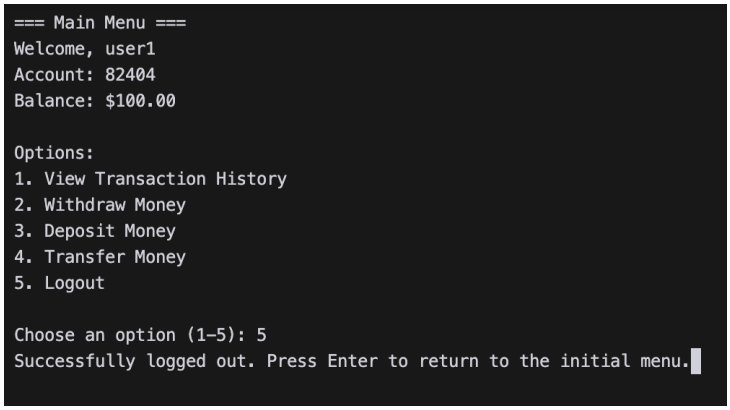
5.3 operation of user1



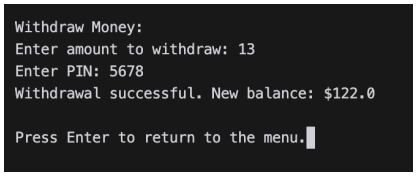


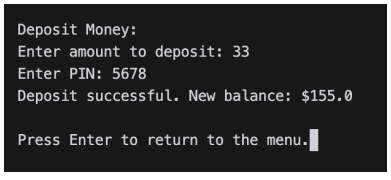


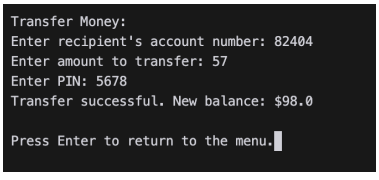
5.4 logout



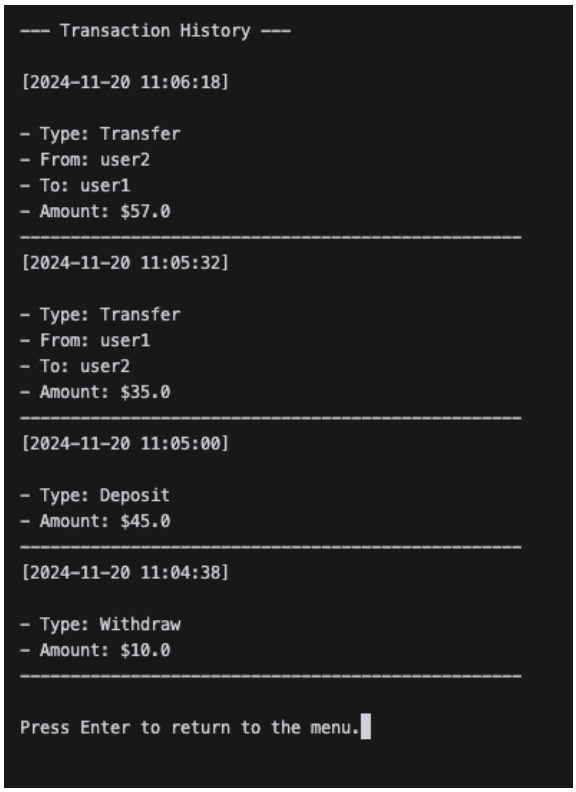
5.5 operation of user2

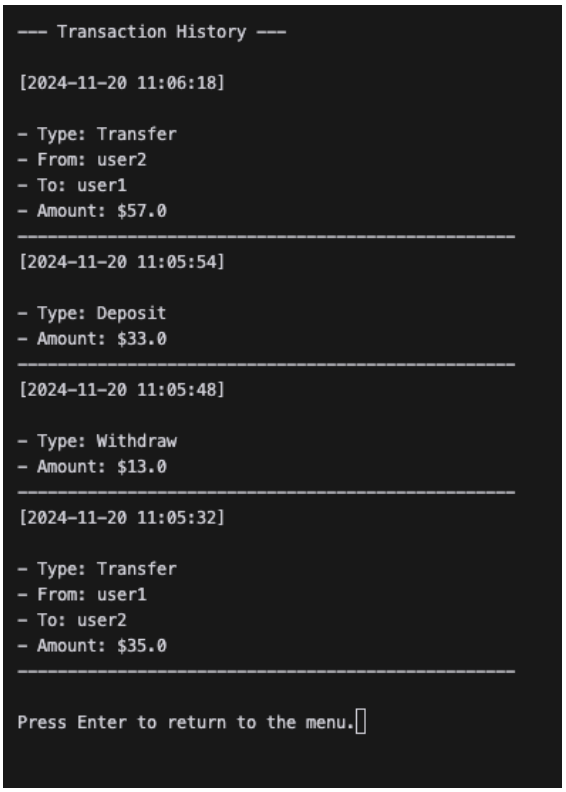




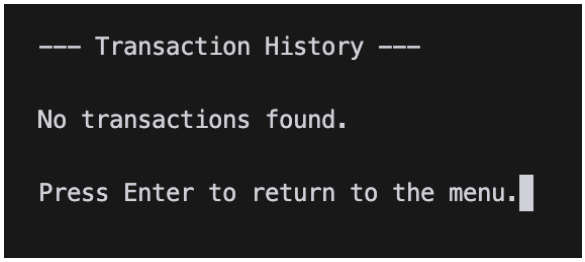


5.6 transaction history of 2 users





5.7 empty transaction history



6 error handling

6.1 invalid option

```
=== PL Banking System ===
1. Register
2. Login
3. Exit

Choose an option (1-3): wrong
Invalid option. Please press Enter to try again.
```

```
=== Main Menu ===
Welcome, user3
Account: 71704
Balance: $100.00

Options:
1. View Transaction History
2. Withdraw Money
3. Deposit Money
4. Transfer Money
5. Logout

Choose an option (1-5): wrong
Invalid option. Please press Enter to try again.
```

6.2 register

```
Register a new user:
Enter username: user1
Error: Username already exists.

Press Enter to return to the menu.
```

```
Register a new user:
Enter username: user3
Enter login password: wrong
Invalid password:
- Password must be at least 7 characters long.
- Password must contain at least one uppercase letter.
- Password must contain at least one special character.

Press Enter to return to the menu.
```

```
Register a new user:
Enter username: user3
Enter login password: TEST!1234
Enter a 4-digit PIN: 12345
Error: PIN must be a 4-digit number.

Press Enter to return to the menu.
```

6.3 login

```
Login:
Enter username: wrong
Enter password: TEST!1234
Error: Invalid username or password.

Press Enter to return to the menu.
```

```
Login:
Enter username: user3
Enter password: wrong
Error: Invalid username or password.

Press Enter to return to the menu.
```

6.4 withdraw

```
Withdraw Money:
Enter amount to withdraw: wrong
Error: Invalid amount format.. Press Enter to return to the menu.
```

```
Withdraw Money:
Enter amount to withdraw: 200000
Error: Insufficient balance. Press Enter to return to the menu.
```

```
Withdraw Money:
Enter amount to withdraw: -3
Error: Amount must be greater than 0.. Press Enter to return to the menu.
```

6.5 deposit

```
Deposit Money:
Enter amount to deposit: wrong
Error: Invalid amount format.. Press Enter to return to the menu.
```

```
Deposit Money:
Enter amount to deposit: 10
Enter PIN: 12345
Error: Incorrect PIN. Press Enter to return to the menu.
```

6.6 transfer

```
Transfer Money:
Enter recipient's account number: wrong
Error: Recipient account not found. Press Enter to return to the menu.
```

```
Transfer Money:
Enter recipient's account number: 71704
Error: You cannot transfer money to your own account. Press Enter to return to the menu.
```

```
Transfer Money:
Enter recipient's account number: 82404
Enter amount to transfer: wrong
Error: Invalid amount format.. Press Enter to return to the menu.
```

```
Transfer Money:
Enter recipient's account number: 82404
Enter amount to transfer: 200000
Error: Insufficient balance. Press Enter to return to the menu.
```

```
Transfer Money:
Enter recipient's account number: 82404
Enter amount to transfer: -3
Error: Amount must be greater than 0.. Press Enter to return to the menu.
```

```
Transfer Money:
Enter recipient's account number: 82404
Enter amount to transfer: 10
Enter PIN: 12345
Error: Incorrect PIN. Press Enter to return to the menu.
```