**BANK MANAGEMENT SYSTEM - PROJECT**

**Objective**

Our goals are key banking features and building a Python-based banking system with a Tkinter GUI that makes account management, transactions, and reporting easier for administrators and users. SQLite or a flat file will be used to manage persistence. We will state that we have taken our time to complete the project we intend to work on. We have chosen to move forward with the Desktop-Based Application (Tkinter) for the Bank Project. Class-based design with OOP principles is what we use, along with a file or database (SQLite/MySQL). To create a unique account number, we will utilize a random module. The modules we manage are Account, Transaction, Admin, and Reports.

**Menu Options**

1. Open/Create Account  
2. User Login  
3. Admin Login

**Explanation or Procedure of Menu Options**

**1. Open/Create Account**

### 📋 **Account Types**

- Savings (4% interest per annum)  
- Current (0% interest)  
- Deposit (fixed term)

**Required Account Details**

| **Field** | **Example** |
| --- | --- |
| Account Type  Account No | Savings/Fixed deposit/Current  2025617 (via random.randint()) |
| Name | John Doe |
| Address | 123 Main Street, NY |
| Mobile | 9876543210 |
| Email | [john@example.com](mailto:john@example.com) |
| KYC Details | PAN/Aadhar/Driving License ABCDE1234F(Alphanumeric value with Auto Capital letters) |

**Attributes:**

* + Account No (auto-generated using a random number)
  + Name, Address, KYC, Mobile, Email
  + Balance, Interest Rate (based on account type)

**Procedure:**

* Use a GUI form with input fields.
* On submit, validate and store details in SQLite or a file.
* Show confirmation with an account number.

**2. User Functions**

### 🔐 **Login:** User enters Account No & Password

### 🔄 **Change Password:** Prompt for old password and confirm new password entry twice

### 📄 **View Transactions/Reports**

Show table/grid of:

1. Date: 12/05/2025
2. Type: Deposit/Withdrawal/Transfer
3. Amount: 10,000.00
4. Balance: 24,000.00

### 💰 **Transactions**

#### a. **Add Amount**

balance += deposit\_amount

#### b. **Withdraw Amount** (with insufficient balance check)

if balance >= withdraw\_amount:

balance -= withdraw\_amount

else:

Raise Exception ("Insufficient Balance")

#### c. **Check Balance:** Display current balance

*d.* ***Transfer Funds:*** (with atomic validation: If A fails, B doesn’t update)

Transfer amountfrom A to B atomically

Example: A to B transfer of ₹1000:

If A has ₹500 → Transaction fails for both A & B

If A has ₹1500 → A = 500, B += 1000

Example 2, if A.balance >= 1000:

A.balance -= 1000

B.balance += 1000

else:

raise Exception ("Insufficient Balance")

➡️ Ensure **atomic** operation with try/except blocks

🔐 Logout

**3. Admin Functions**

### 🔐 **Login:** Admin login with secure credentials

### 📊 **View Total Accounts:** Count records from SQLite

SQL code: SELECT COUNT (\*) FROM accounts;

### 📆 **Generate Reports:**

### Filter transactions:

### Daily

### Monthly

### Yearly

SQL Code: SELECT \* FROM transactions WHERE date >= '2025-05-01';

### 🔍 **Monitor All Transactions:** View the log of all users' actions

## **📊 View Account Summary:** View the Records of all users in SQLite

🔐 Logout

## 🚨 **Exception Handling (Examples)**

| **Scenario** | **Example Code Snippet** |
| --- | --- |
| Insufficient Balance | raise Exception ("Insufficient funds for this transaction") |
| Invalid Input (e.g., str in amount) | amount = float(entry.get()) except: showerror(...) |
| Duplicate Email/Phone | Check existing records before saving. |
| Missing Mandatory Fields | Validate empty inputs in Tkinter before submission. |

**Persistence Layer**

Choose one:

* ✅ **SQLite** (Recommended for better structure)
* File-based (CSV/TXT using json or pickle modules)