**A**

**MINOR PROJECT**

**ON**

**Basic Banking Management System**

*In partial fulfilment* *of the requirement for the award of the degree of*

**BACHELOR OF TECHNOLOGY**

**IN**

**COMPUTER SCIENCE AND ENGINEERING**

*Submitted by*

**Name: Akshat Gupta Name: Ansh Gupta**

**Roll No: 2302250100024 Roll No: 2302250100041**

**Section: A Section: A**

**Under the supervision of**

**Seema Yadav**

**Assistant Professor**

A blue and white logo

Description automatically generated

**ACCURATE COLLEGE, GREATER NOIDA - 201316**

AFFLIATED TO

**Dr. A.P.J. ABDUL KALAM TECHNICAL UNIVERSITY, LUCKNOW**

**Declaration**

I hereby declare that this project titled "Basic Banking Management System" is an original piece of work carried out by me under the guidance of my supervisor. This project has not been submitted for any other degree or diploma. All references and sources used have been duly acknowledged. I take full responsibility for any errors or omissions in this work.

**Name:**

Akshat Gupta

Ansh Gupta

**Date:**

//2024

**Abstract**

The Basic Banking Management System (BBMS) is a comprehensive software application designed to automate and enhance various banking operations. This project addresses the inefficiencies of traditional manual banking systems by providing an intuitive graphical user interface (GUI) that allows users to manage their accounts effectively. The BBMS includes functionalities such as account creation, fund deposits and withdrawals, balance inquiries, account updates, and closures.

Utilizing Python and the Tkinter library for GUI development, the system offers a seamless experience for users while ensuring data integrity and security. The project employs the Pickle module for persistent data storage, enabling users to retrieve their information even after closing the application. Through thorough testing and validation, the BBMS demonstrates a significant improvement in transaction efficiency and user satisfaction. Future enhancements may include user authentication, transaction history, and mobile application development to broaden accessibility.

Acknowledgment

I would like to express my sincere gratitude to all those who have contributed to the successful completion of this project. First and foremost, I would like to thank my supervisor, Seema Yadav for their invaluable guidance, support, and encouragement throughout the development process. Their insights have been instrumental in shaping the project.

I also wish to acknowledge my peers and classmates who provided feedback and collaborated with me during various stages of this project. Their support and constructive criticism were invaluable.

Additionally, I am grateful to my family for their unwavering support and encouragement, which motivated me to pursue this project with dedication and enthusiasm.

Lastly, I would like to thank the developers and contributors of Python and Tkinter for creating such powerful tools that made this project possible. Their work continues to inspire and facilitate learning in the field of programming and software development.

**Table of Contents**

**Chapter No. | TITLE**

**ABSTRACT**

**LIST OF FIGURES**

**1 | INTRODUCTION**

**1.1 | OVERVIEW OF THE PROJECT**

**2 | LITERATURE SURVEY**

**3 | AIM AND SCOPE OF PRESENT INVESTIGATION**

**3.1 | AIM OF THE PROJECT**

**3.2 | SCOPE AND OBJECTIVE**

**3.3 | SYSTEM REQUIREMENTS**

**3.3.1 | HARDWARE REQUIREMENTS**

**3.3.2 | SOFTWARE REQUIREMENTS**

**3.4 | SOFTWARE USED**

**3.4.1 | PYTHON LANGUAGE**

**3.4.2 | PYTHON CHARACTERISTICS**

**3.4.3 | APPLICATIONS OF PYTHON**

**3.5 | JUPYTER**

**3.6 | TKINTER**

**3.7 | PICKLE**

**4 | EXPERIMENTAL OR MATERIAL METHODS**

**4.1 | DESIGN METHODOLOGY**

**4.1.1 | EXISTING SYSTEM**

**4.1.2 | PROPOSED SYSTEM**

**4.2 | MODULE DESCRIPTION**

**4.3 | ARCHITECTURE DIAGRAM**

**4.3.1 | ER DIAGRAM**

**5 | RESULTS AND PERFORMANCE ANALYSIS**

**5.1 | HOME PAGE**

**5.2 | NEW ACCOUNT**

**5.3 | TRANSACTION**

**5.4 | UPDATION**

**5.5 | BALANCE ENQUIRY**

**5.6 | CLOSE ACCOUNT**

**5.7 | REPORTS**

**6 | CONCLUSION AND FUTURE ENHANCEMENTS**

**6.1 | CONCLUSION**

**6.2 | FUTURE ENHANCEMENT**

**REFERENCES**

**SOURCE CODE**

**Basic Banking Management System**

**1. Introduction**

The Basic Bank Management System (BBMS) is a powerful software tool designed to automate and simplify banking operations. By consolidating key functions into one platform, the BBMS enhances the user experience, boosts efficiency, and reduces errors associated with manual banking processes.

**1.1 Overview of the Project**

The goal of this project is to develop a functional BBMS that allows users to manage their bank accounts effectively. The system supports a variety of operations such as account creation, deposits, withdrawals, balance inquiries, account updates, and account closures. The user-friendly graphical interface (GUI) ensures that users can interact with the system easily, making it accessible for all.

**2. Literature Survy**

The transition from manual to automated banking systems has significantly improved the banking experience. Research shows that automation increases efficiency and customer satisfaction by providing real-time access to account information. This project builds on this progress by creating a desktop application using Python, which allows for better user interaction and streamlined operations.

**3. Aim and Scope of Present Investigation**

**3.1 Aim of the Project**

The project's primary objective is to develop a comprehensive Bank Management System that covers essential banking operations, enabling users to carry out transactions securely and efficiently.

**3.2 Scope and Objective**

The BBMS focuses on offering a variety of banking services through a simple and intuitive interface. The specific goals of the project include:

* **Creating and managing accounts (creation, updates, and closures).**
* **Handling transactions (deposits and withdrawals).**
* **Allowing users to check their account balances and transaction history.**

**3.3 System Requirements**

**3.3.1 Hardware Requirements** **Processor:**

A minimum of a 2 GHz dual-core processor or higher. **RAM:** At least 4 GB of RAM for optimal performance. **Storage:** A minimum of 500 MB of free space on the hard drive for the application and its data. **Display:** A monitor with a resolution of at least 1366x768 pixels.

**3.3.2 Software Requirements** **Operating System:**

Compatible with Windows 10/11, macOS, or Linux. **Python Version:** Python 3.x with the necessary libraries. **Tkinter Library:** A built-in library required for the GUI. **Text Editor or IDE:** It is recommended to use an IDE such as Jupyter or Visual Studio Code.

**3.4 Software Used**

**3.4.1 Python Language**

Python is an interpreted, high-level programming language known for its simplicity and versatility. It is used across various fields like web development, data science, and automation.

**3.4.2 Python Characteristics**

Key features that make Python ideal for this project include:

* **Ease of Learning:** Python’s syntax is simple and easy to understand.
* **Extensive Libraries:** A broad collection of libraries that simplify the development process.
* **Cross-Platform Compatibility:** Python code can run on different operating systems with minimal changes.

**3.4.3 Applications of Python**

Python is widely used in areas like web development and artificial intelligence. In this project, it serves to build an interactive GUI while managing backend processes.

**3.5 Jupyter**

Jupyter Notebook is an open-source application that allows users to create and share documents combining code, text, and data visualizations. It supports multiple languages, including Python, and helps in prototyping and presenting ideas clearly.

**3.6 Tkinter**

Tkinter is the built-in GUI toolkit for Python that allows developers to build desktop applications with ease. It provides many features, including:

* **A variety of widgets like buttons, labels, and text fields for designing interfaces.**
* **Event-driven programming to respond to user actions.**
* **Cross-platform compatibility.**

**3.7 Pickle**

Pickle is a Python module used to serialize and deserialize objects, which allows storing complex data types (like account details and transaction records) in a binary format, ensuring that user data persists across sessions.

**4. Experimental or Material Methods**

**4.1 Design Methodology**

The design methodology follows a systematic process to develop the Bank Management System, starting with requirement gathering, followed by analyzing existing systems, and then creating a digital solution.

**4.1.1 Existing System**

Traditional banking systems often rely on manual processes that can lead to inefficiencies, human errors, and long wait times for users.

**4.1.2 Proposed System**

The proposed BBMS automates critical banking functions, providing real-time updates and immediate feedback on user transactions, resulting in more efficient service and an improved user experience.

**4.2 Module Description**

The BBMS consists of several modules, each with specific functions:

* **Account Management Module:** Handles tasks like account creation, updates, and closure.
* **Transaction Module:** Manages deposits and withdrawals while ensuring accurate account balance updates.
* **Balance Enquiry Module:** Allows users to check their account balance.
* **Reporting Module:** Provides transaction summaries and account details.

**4.3 Architecture Diagram**

The architecture diagram visually represents the interaction between different components of the application, such as the user interface, business logic, and data storage. It helps clarify how the system operates.

**4.3.1 ER Diagram**

The Entity-Relationship (ER) diagram shows the database structure and the relationships between entities like accounts and transactions, acting as a blueprint for data management.

**5. Results and Performance Analysis**

The BBMS was thoroughly tested for both functionality and performance. Each feature was reviewed to ensure it met the project's goals and user expectations.

**5.1 Home Page**

The home page serves as the main navigation point, with easy-to-use buttons that give access to all system features.

**5.2 New Account**

This feature lets users input personal information to create new accounts, with built-in validation to ensure unique account numbers and correct data.

**5.3 Transaction Method**

Transactions can be carried out through a simple interface. The system checks the inputs, updates account balances in real-time, and confirms the transaction's success.

**5.4 Updation**

Users can update their account details through this feature. The system verifies the changes and ensures they are reflected instantly.

**5.5 Balance Enquiry**

This functionality lets users check their account balance by entering their account number, providing immediate access to their financial status.

**5.6 Close Account**

To close an account, users provide the account number, and the system processes the closure while maintaining data integrity.

**5.7 Reports**

This feature generates transaction summaries and account reports, helping users track their banking activities.

**6. Conclusion and Future Enhancements**

**6.1 Conclusion**

The Basic Banking Management System effectively demonstrates the advantages of automation in improving banking operations. Its easy-to-use interface and comprehensive features offer a robust solution for managing personal banking tasks.

**6.2 Future Enhancements**

Potential future upgrades could include:

* **User Authentication:** Introducing secure login systems to protect user data.
* **Transaction History:** Adding a feature to view past transactions for better tracking.
* **Mobile Application:** Developing a mobile version of the BBMS to enhance user accessibility.

**References**

A detailed list of resources, including books, scholarly articles, and online materials, that were referenced throughout the project to support its development and understanding of banking systems, software design, and user interface development.

**Source Code**

The complete source code for the Banking Management System is provided in this section. This includes all modules, classes, and functions, along with comments explaining the purpose of key sections of the code. The source code serves as a reference for developers interested in understanding the implementation details or building upon this project.

import tkinter as tk

from tkinter import ttk, messagebox

import os

class Account:

def \_\_init\_\_(self, acc\_no=0, name="", acc\_type="", deposit=0, pin=""):

self.accNo = acc\_no

self.name = name

self.type = acc\_type

self.deposit = deposit

self.pin = pin # Store PIN

class BankManagementSystem:

def \_\_init\_\_(self, master):

self.master = master

self.master.title('Banking Management System')

self.master.geometry('800x600')

self.accounts = []

self.current\_frame = None

self.load\_accounts() # Load accounts when the application starts

self.create\_widgets()

def load\_accounts(self):

filepath = r'C:\Users\aksha\OneDrive\Desktop\banking\data.txt'

if os.path.exists(filepath):

with open(filepath, 'r') as f:

for line in f:

if line.strip():

try:

acc\_no, name, acc\_type, deposit, pin = line.strip().split(',')

account = Account(int(acc\_no), name, acc\_type, int(deposit), pin)

self.accounts.append(account)

except ValueError:

print(f"Skipping invalid line: {line.strip()}")

def create\_widgets(self):

self.main\_frame = tk.Frame(self.master)

self.main\_frame.pack()

title\_label = tk.Label(self.main\_frame, text='Banking Management System', font=('Arial', 24, 'bold'))

title\_label.pack(pady=20)

links = ['New Account', 'Deposit', 'Withdrawal', 'Balance Enquiry',

'Update Details', 'Close Account', 'Account List', 'Exit']

for item in links:

tk.Button(self.main\_frame, text=item, command=lambda txt=item: self.show\_frame(txt)).pack(pady=4)

def show\_frame(self, option):

if self.current\_frame:

self.current\_frame.pack\_forget()

if option == 'main':

self.current\_frame = self.main\_frame

elif option == 'New Account':

self.current\_frame = self.create\_new\_account\_frame()

elif option == 'Deposit':

self.current\_frame = self.create\_deposit\_frame()

elif option == 'Withdrawal':

self.current\_frame = self.create\_withdrawal\_frame()

elif option == 'Balance Enquiry':

self.current\_frame = self.create\_balance\_enquiry\_frame()

elif option == 'Update Details':

self.current\_frame = self.create\_update\_details\_frame()

elif option == 'Close Account':

self.current\_frame = self.create\_close\_account\_frame()

elif option == 'Account List':

self.current\_frame = self.create\_account\_list\_frame()

elif option == 'Exit':

self.master.destroy()

return # Prevent further execution after destroying

self.current\_frame.pack()

def create\_new\_account\_frame(self):

frame = tk.Frame(self.master)

tk.Label(frame, text='Account No.').grid(row=0, column=0)

tk.Label(frame, text='Name').grid(row=1, column=0)

tk.Label(frame, text='Account Type').grid(row=2, column=0)

tk.Label(frame, text='Initial Deposit').grid(row=3, column=0)

tk.Label(frame, text='PIN').grid(row=4, column=0)

acc\_no = tk.StringVar()

name = tk.StringVar()

acc\_type = tk.StringVar(value="Savings")

deposit = tk.StringVar()

pin = tk.StringVar() # PIN input field

tk.Entry(frame, textvariable=acc\_no).grid(row=0, column=1)

tk.Entry(frame, textvariable=name).grid(row=1, column=1)

tk.Entry(frame, textvariable=acc\_type).grid(row=2, column=1)

tk.Entry(frame, textvariable=deposit).grid(row=3, column=1)

tk.Entry(frame, textvariable=pin, show="\*").grid(row=4, column=1) # Mask the PIN input

tk.Button(frame, text='Create Account', command=lambda: self.create\_account(acc\_no.get(), name.get(), acc\_type.get(), deposit.get(), pin.get())).grid(row=5, column=1)

tk.Button(frame, text='Back', command=lambda: self.show\_frame('main')).grid(row=5, column=0)

return frame

def create\_account(self, acc\_no\_str, name, acc\_type, deposit\_str, pin):

try:

acc\_no = int(acc\_no\_str)

deposit = int(deposit\_str)

if any(account.accNo == acc\_no for account in self.accounts):

messagebox.showerror('Error', 'Account number already exists.')

return

account = Account(acc\_no, name, acc\_type, deposit, pin)

self.accounts.append(account)

self.save\_accounts()

messagebox.showinfo('Success', 'Account created successfully!')

except ValueError:

messagebox.showerror('Error', 'Please enter valid numbers for Account No. and Deposit.')

def create\_account\_list\_frame(self):

frame = tk.Frame(self.master)

columns = ('#', 'Account No', 'Name', 'Type', 'Deposit')

table = ttk.Treeview(frame, columns=columns, show='headings')

for col in columns:

table.heading(col, text=col)

for idx, account in enumerate(self.accounts, start=1):

table.insert("", "end", values=(idx, account.accNo, account.name, account.type, account.deposit))

table.pack()

tk.Button(frame, text='Back', command=lambda: self.show\_frame('main')).pack()

return frame

def create\_deposit\_frame(self):

frame = tk.Frame(self.master)

tk.Label(frame, text='Account No.').grid(row=0, column=0)

tk.Label(frame, text='Deposit Amount').grid(row=1, column=0)

acc\_no = tk.StringVar()

amount = tk.StringVar()

tk.Entry(frame, textvariable=acc\_no).grid(row=0, column=1)

tk.Entry(frame, textvariable=amount).grid(row=1, column=1)

tk.Button(frame, text='Deposit', command=lambda: self.deposit(acc\_no.get(), amount.get())).grid(row=2, column=1)

tk.Button(frame, text='Back', command=lambda: self.show\_frame('main')).grid(row=2, column=0)

return frame

def deposit(self, acc\_no\_str, amount\_str):

try:

acc\_no = int(acc\_no\_str)

amount = int(amount\_str)

account = next((acc for acc in self.accounts if acc.accNo == acc\_no), None)

if account:

account.deposit += amount

self.save\_accounts()

messagebox.showinfo('Success', f'Deposited {amount} to account {acc\_no}.')

else:

messagebox.showerror('Error', 'Account not found.')

except ValueError:

messagebox.showerror('Error', 'Please enter valid numbers.')

def create\_withdrawal\_frame(self):

frame = tk.Frame(self.master)

tk.Label(frame, text='Account No.').grid(row=0, column=0)

tk.Label(frame, text='Withdrawal Amount').grid(row=1, column=0)

tk.Label(frame, text='PIN').grid(row=2, column=0)

acc\_no = tk.StringVar()

amount = tk.StringVar()

pin = tk.StringVar()

tk.Entry(frame, textvariable=acc\_no).grid(row=0, column=1)

tk.Entry(frame, textvariable=amount).grid(row=1, column=1)

tk.Entry(frame, textvariable=pin, show="\*").grid(row=2, column=1) # Mask the PIN input

tk.Button(frame, text='Withdraw', command=lambda: self.withdraw(acc\_no.get(), amount.get(), pin.get())).grid(row=3, column=1)

tk.Button(frame, text='Back', command=lambda: self.show\_frame('main')).grid(row=3, column=0)

return frame

def withdraw(self, acc\_no\_str, amount\_str, pin):

try:

acc\_no = int(acc\_no\_str)

amount = int(amount\_str)

account = next((acc for acc in self.accounts if acc.accNo == acc\_no), None)

if account:

if account.pin == pin: # Verify the PIN

if account.deposit >= amount:

account.deposit -= amount

self.save\_accounts()

messagebox.showinfo('Success', f'Withdrew {amount} from account {acc\_no}.')

else:

messagebox.showerror('Error', 'Insufficient funds.')

else:

messagebox.showerror('Error', 'Incorrect PIN.')

else:

messagebox.showerror('Error', 'Account not found.')

except ValueError:

messagebox.showerror('Error', 'Please enter valid numbers.')

def create\_balance\_enquiry\_frame(self):

frame = tk.Frame(self.master)

tk.Label(frame, text='Account No.').grid(row=0, column=0)

acc\_no = tk.StringVar()

tk.Entry(frame, textvariable=acc\_no).grid(row=0, column=1)

tk.Button(frame, text='Check Balance', command=lambda: self.check\_balance(acc\_no.get())).grid(row=1, column=1)

tk.Button(frame, text='Back', command=lambda: self.show\_frame('main')).grid(row=1, column=0)

return frame

def check\_balance(self, acc\_no\_str):

try:

acc\_no = int(acc\_no\_str)

account = next((acc for acc in self.accounts if acc.accNo == acc\_no), None)

if account:

messagebox.showinfo('Balance', f'Balance for account {acc\_no}: {account.deposit}')

else:

messagebox.showerror('Error', 'Account not found.')

except ValueError:

messagebox.showerror('Error', 'Please enter a valid account number.')

def create\_update\_details\_frame(self):

frame = tk.Frame(self.master)

tk.Label(frame, text='Account No.').grid(row=0, column=0)

acc\_no = tk.StringVar()

tk.Entry(frame, textvariable=acc\_no).grid(row=0, column=1)

tk.Button(frame, text='Fetch Details', command=lambda: self.fetch\_account\_details(acc\_no.get())).grid(row=1, column=1)

tk.Button(frame, text='Back', command=lambda: self.show\_frame('main')).grid(row=1, column=0)

return frame

def fetch\_account\_details(self, acc\_no\_str):

try:

acc\_no = int(acc\_no\_str)

account = next((acc for acc in self.accounts if acc.accNo == acc\_no), None)

if account:

self.update\_account\_details(account)

else:

messagebox.showerror('Error', 'Account not found.')

except ValueError:

messagebox.showerror('Error', 'Please enter a valid account number.')

def update\_account\_details(self, account):

frame = tk.Frame(self.master)

tk.Label(frame, text='Name').grid(row=0, column=0)

tk.Label(frame, text='Account Type').grid(row=1, column=0)

tk.Label(frame, text='Deposit').grid(row=2, column=0)

name\_var = tk.StringVar(value=account.name)

acc\_type\_var = tk.StringVar(value=account.type)

deposit\_var = tk.StringVar(value=str(account.deposit))

tk.Entry(frame, textvariable=name\_var).grid(row=0, column=1)

tk.Entry(frame, textvariable=acc\_type\_var).grid(row=1, column=1)

tk.Entry(frame, textvariable=deposit\_var).grid(row=2, column=1)

tk.Button(frame, text='Update', command=lambda: self.update\_account(account, name\_var.get(), acc\_type\_var.get(), deposit\_var.get())).grid(row=3, column=1)

tk.Button(frame, text='Back', command=lambda: self.show\_frame('main')).grid(row=3, column=0)

frame.pack()

def update\_account(self, account, name, acc\_type, deposit\_str):

try:

deposit = int(deposit\_str)

account.name = name

account.type = acc\_type

account.deposit = deposit

self.save\_accounts()

messagebox.showinfo('Success', 'Account details updated successfully!')

self.show\_frame('main')

except ValueError:

messagebox.showerror('Error', 'Please enter a valid deposit amount.')

def create\_close\_account\_frame(self):

frame = tk.Frame(self.master)

tk.Label(frame, text='Account No.').grid(row=0, column=0)

acc\_no = tk.StringVar()

tk.Entry(frame, textvariable=acc\_no).grid(row=0, column=1)

tk.Button(frame, text='Close Account', command=lambda: self.close\_account(acc\_no.get())).grid(row=1, column=1)

tk.Button(frame, text='Back', command=lambda: self.show\_frame('main')).grid(row=1, column=0)

return frame

def close\_account(self, acc\_no\_str):

try:

acc\_no = int(acc\_no\_str)

account = next((acc for acc in self.accounts if acc.accNo == acc\_no), None)

if account:

self.accounts.remove(account)

self.save\_accounts()

messagebox.showinfo('Success', 'Account closed successfully!')

else:

messagebox.showerror('Error', 'Account not found.')

except ValueError:

messagebox.showerror('Error', 'Please enter a valid account number.')

def save\_accounts(self):

filepath = r'C:\Users\aksha\OneDrive\Desktop\banking\data.txt'

with open(filepath, 'w') as f:

for account in self.accounts:

f.write(f"{account.accNo},{account.name},{account.type},{account.deposit},{account.pin}\n")

if \_\_name\_\_ == '\_\_main\_\_':

root = tk.Tk()

app = BankManagementSystem(root)

root.mainloop()