ATM Management System

**IILM University**

**B.Tech C.S.E. - IBM**

**A Project Report**

**Abstract: ATM Management System Using Python**

**The ATM Management System is a Python-based project designed to simulate the operations of an Automated Teller Machine (ATM). This system provides a user-friendly interface for managing banking transactions such as account creation, balance inquiries, deposits, and withdrawals. By employing Python’s object-oriented programming (OOP) principles and file handling, the system ensures data integrity and simplicity of use.**

**The system includes the following key features:**

**• Account Creation: Enables users to create new accounts by providing a unique account number, name, PIN, and initial deposit.**

**• User Authentication: Ensures secure access to accounts using a valid account number and PIN combination.**

**• Balance Inquiry: Displays the current account balance in real-time.**

**• Deposits and Withdrawals: Allows users to deposit funds or withdraw cash, with proper validation to prevent overdrafts.**

**• Data Persistence: Utilizes JSON files to store account details, ensuring that data is saved between sessions.**

**This project demonstrates the integration of fundamental programming concepts with practical applications, making it an excellent example of how Python can be used to develop robust and efficient software solutions. The ATM Management System is scalable and can be expanded with additional features such as transaction history, interest calculations, or multi-user access.**

**Introduction: ATM Management System**

**An Automated Teller Machine (ATM) is an essential part of modern banking that provides customers with 24/7 access to financial transactions, such as cash withdrawals, deposits, and account inquiries. The ATM Management System using Python is a software application designed to replicate these functionalities in a virtual environment. This system aims to simplify banking operations, enhance accessibility, and ensure secure transactions for users.**

**The project provides a platform for managing user accounts, verifying identities, and performing basic banking operations efficiently. It employs Python programming language, known for its simplicity and versatility, to create a user-friendly interface and reliable backend operations. Data persistence is achieved using JSON, which ensures that account information is securely stored and retrieved for future use.**

**Key Features:**

**1. Account Creation: Users can create accounts with unique credentials and an initial deposit.**

**2. Authentication System: Ensures secure access by requiring a valid account number and PIN.**

**3. Balance Inquiry: Allows users to check their current account balance.**

**4. Deposit and Withdrawal: Facilitates seamless deposits and cash withdrawals with validation.**

**5. Data Management: Uses a JSON file to securely store and manage account details.**

**Purpose:**

**The primary goal of the ATM Management System is to simulate real-world ATM functionalities in a controlled and efficient way. This project not only serves as a learning tool for understanding Python programming and file handling but also lays the foundation for developing more complex financial software systems.**

**Scope:**

**This system can be used as a standalone tool for educational purposes or as a prototype for developing more advanced applications. With additional features such as transaction history, multi-user access, and integration with a database, the ATM Management System can be scaled to meet real-world banking requirements.**

**This project highlights the importance of secure and efficient software in financial systems while providing an engaging and practical way to learn programming concepts.**

**Problem Statement: ATM Management System**

**In today’s fast-paced world, banking systems play a critical role in managing financial transactions. Automated Teller Machines (ATMs) are a convenient medium for customers to access their accounts without visiting a bank branch. However, developing a simplified and secure virtual simulation of an**

**ATM system is necessary for educational purposes, as well as for testing and prototyping banking solutions.**

**The current challenge lies in creating a user-friendly and reliable system that provides essential ATM functionalities such as account creation, secure authentication, balance inquiry, deposits, and withdrawals. Additionally, the system should ensure data persistence, prevent unauthorized access, and handle errors gracefully to enhance the user experience.**

**This project seeks to address the following issues:**

**1. Simplified Banking Access: Providing a platform where users can perform basic banking operations efficiently.**

**2. Secure Authentication: Implementing a mechanism to ensure only authorized users can access their accounts.**

**3. Data Storage: Ensuring that user data is securely stored and retrieved for future use.**

**4. Error Handling: Designing a system that can handle invalid inputs, insufficient balances, and other edge cases effectively.**

**5. Scalability and Modularity: Creating a foundation that can be expanded with additional features such as transaction history, interest calculations, and database integration.**

**By addressing these challenges, the project aims to simulate a functional ATM system while providing a hands-on experience in software development and problem-solving using Python.**

**Literature Review: ATM Management System**

**The development of an ATM Management System stems from the need to simulate and understand the operations of an Automated Teller Machine, a critical component of modern banking systems. Over the years, researchers and developers have explored various aspects of ATM systems, including their design, security, and functionality. The following review highlights key concepts, existing solutions, and gaps addressed by this project:**

**1. Evolution of ATM Systems:**

**The first ATMs were introduced in the late 1960s, revolutionizing banking by offering 24/7 access to financial services. Modern ATMs now support a wide range of functions, including deposits, withdrawals, fund transfers, and account management. Simulating these features in a software-based system provides a foundation for understanding their technical and operational complexities.**

**2. Security and Authentication:**

**Research in ATM systems highlights the importance of secure user authentication to prevent unauthorized access. Traditional ATMs use PINs, biometrics, and card-based authentication. This project implements a simplified PIN-based security system, adhering to the principle of confidentiality in data handling.**

**3. Data Management and Persistence:**

**Efficient data storage and retrieval are vital for any banking system. Traditional ATM systems rely on relational databases for real-time processing. In this project, a lightweight JSON file is used to persist account data, providing a simpler yet effective mechanism for managing account information in smaller-scale systems.**

**4. User-Friendly Interfaces:**

**Studies emphasize that user interfaces in ATMs should be intuitive and straightforward. This project adopts a text-based menu-driven interface, making it accessible for educational purposes and for users with minimal technical expertise.**

**5. Simulated Systems for Education:**

**Simulation of banking systems in programming projects has been widely explored as a tool for teaching software design, algorithm development, and secure programming practices. This project draws inspiration from such educational initiatives to develop a scalable and modular ATM system prototype.**

**Significance and Contribution:**

**While several ATM management system simulations exist, many focus on isolated functionalities or lack comprehensive error handling and scalability. This project contributes by:**

**• Integrating core ATM functionalities (account creation, authentication, transactions).**

**• Providing a modular framework that can be extended with advanced features.**

**• Offering a practical, hands-on experience for learners exploring Python programming, file handling, and system design.**

**This literature foundation underscores the relevance of the project in bridging the gap between theoretical knowledge and practical application in financial software systems.**

**Code**

import csv

import os

# Define filenames

CUSTOMER\_FILE = "customers.csv"

# Function to initialize default customer data

def initialize\_customers():

    customers = [

        {'Name': 'Mayank Srivastava', 'Pin': '6394', 'Balance': 200000000.00},

        {'Name': 'Harsh Joshi', 'Pin': '9632', 'Balance': 19000000.00},

        {'Name': 'Aayan Saifi', 'Pin': '7316', 'Balance': 3000000.00},

        {'Name': 'Kartikey', 'Pin': '9874', 'Balance': 10000.00},

        {'Name': 'Rishabh', 'Pin': '8521', 'Balance': 150099.00},

    ]

    # Write default customer data to CSV

    with open(CUSTOMER\_FILE, mode='w', newline="") as file:

        writer = csv.DictWriter(file, fieldnames=['Name', 'Pin', 'Balance'])

        writer.writeheader()

        writer.writerows(customers)

# Function to read customer data from CSV

def read\_customers():

    customers = {}

    if os.path.exists(CUSTOMER\_FILE):

        with open(CUSTOMER\_FILE, mode='r', newline="") as file:

            reader = csv.DictReader(file)

            for row in reader:

                customers[row['Name']] = {

                    'Pin': row['Pin'],

                    'Balance': float(row['Balance'])

                }

    else:

        # Initialize with default customers if file does not exist

        initialize\_customers()

        return read\_customers()  # Recursively call to read data after initialization

    return customers

# Function to write customer data to CSV

def write\_customers(customers):

    with open(CUSTOMER\_FILE, mode='w', newline="") as file:

        writer = csv.DictWriter(file, fieldnames=['Name', 'Pin', 'Balance'])

        writer.writeheader()

        for name, data in customers.items():

            writer.writerow({'Name': name, 'Pin': data['Pin'], 'Balance': data['Balance']})

# Function to authenticate the user

def authenticate\_user(customers):

    while True:

        name = input("Enter your name: ")

        if name not in customers:

            print("Customer not found.")

            continue

        pin = input("Enter your PIN: ")  # Use input instead of getpass.getpass

        if pin == customers[name]['Pin']:

            return name

        else:

            print("Incorrect PIN. Please try again.")

# Function to check balance

def check\_balance(customers, user):

    print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

    print(f"Your current balance is: ${customers[user]['Balance']:.2f}")

    print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

# Function to withdraw amount

def withdraw\_amount(customers, user):

    while True:

        try:

            amount = float(input("Enter amount to withdraw: "))

            if amount <= 0:

                print("Amount must be greater than zero.")

            elif amount > customers[user]['Balance']:

                print("Insufficient balance.")

            else:

                pin = input("Enter your PIN to confirm withdrawal: ")  # Use input instead of getpass.getpass

                if pin == customers[user]['Pin']:

                    customers[user]['Balance'] -= amount

                    write\_customers(customers)

                    print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

                    print(f"Withdrawal successful. New Balance: ${customers[user]['Balance']:.2f}")

                    print("\t\tThanks for the transaction.\t\t")

                    print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

                    return

                else:

                    print("Incorrect PIN. Withdrawal failed.")

        except ValueError:

            print("Invalid amount. Please enter a number.")

# Function to handle user login

def login():

    print('------------------')

    print("1. Login")

    print("2. Exit")

    print('--------------------')

    choice = int(input("Enter Your Choice: "))

    if choice == 1:

        u1 = input("Enter Username: ")

        pwd1 = input("Enter User Password: ")

        if u1 == 'root' and pwd1 == '123':

            print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

            print("Login Successful!")

            print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

            return True

        else:

            print("Invalid credentials. Please try again.")

            return login()

    else:

        print("Exiting...")

        exit()

# Main function to manage ATM operations

def atm\_management\_system():

    print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

    print("---------------------------| IILM University |-------------------------")

    print("---------------------------| ATM Management System |-------------------")

    if login():

        customers = read\_customers()

        user = authenticate\_user(customers)

        while True:

            print()

            print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")

            print(" \n|Options: |")

            print("|\*\*\*\*\*\*\*\*\*\*\*1. Check Balance \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*| ")

            print("|\*\*\*\*\*\*\*\*\*\*\*2. Withdraw Money \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*| ")

            print("|\*\*\*\*\*\*\*\*\*\*\*3. Exit \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*| ")

            print("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_")

            print()

            choice = input("Enter your choice: ")

            if choice == '1':

                check\_balance(customers, user)

            elif choice == '2':

                withdraw\_amount(customers, user)

            elif choice == '3':

                print("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")

                print("Thank you! Have a nice day!")

                break

            else:

                print("Invalid choice. Please enter 1, 2 or 3.")

if \_\_name\_\_ == "\_\_main\_\_":

    atm\_management\_system()

**Methodology: ATM Management System**

**The methodology for developing the ATM Management System involves a structured approach, encompassing requirement analysis, design, development, testing, and deployment. Each phase ensures the delivery of a secure, user-friendly, and functional system capable of simulating ATM operations.**

**1. Requirement Analysis**

**• Functional Requirements:**

**• Users can create accounts with unique account numbers, PINs, and an initial balance.**

**• Secure authentication using account number and PIN.**

**• Basic banking functionalities:**

**• Balance inquiry**

**• Deposit**

**• Withdrawal**

**• Persistent storage of account data.**

**• Non-Functional Requirements:**

**• User-friendly interface.**

**• Secure and error-free transactions.**

**• Data reliability through efficient file handling.**

**2. System Design**

**• Architecture:**

**• A menu-driven application for interaction with users.**

**• Object-oriented programming (OOP) approach for modularity.**

**• JSON file for persistent storage of account data.**

**• Components:**

**• Account Management: Handles account creation and authentication.**

**• Transaction Management: Manages deposits, withdrawals, and balance inquiries.**

**• Data Handling: Uses JSON for storing and retrieving account information.**

**3. Development Process**

**• Programming Language: Python.**

**• Modules and Tools:**

**• json: For file-based data storage.**

**• Python standard libraries for input handling and basic validation.**

**• Implementation Steps:**

**1. Create a Python class ATM to encapsulate all functionalities.**

**2. Implement methods for:**

**• Account creation**

**• Authentication**

**• Balance inquiry**

**• Deposit and withdrawal**

**3. Use a JSON file for persistent data storage.**

**4. Develop a user-friendly menu system for navigation.**

**4. Testing**

**• Unit Testing:**

**• Test individual functionalities such as account creation, deposit, withdrawal, and authentication.**

**• Integration Testing:**

**• Verify the interaction between components like authentication and transaction handling.**

**• Validation Testing:**

**• Ensure the system handles edge cases such as:**

**• Invalid PIN entry.**

**• Withdrawal exceeding the balance.**

**• Duplicate account numbers.**

**• User Testing:**

**• Gather feedback from users to refine the interface and functionality.**

**5. Deployment**

**• Deploy the system as a standalone Python application.**

**• Distribute the program along with instructions for use.**

**• Ensure the application is portable and can run on any Python-enabled environment.**

**6. Maintenance and Future Enhancements**

**• Maintenance:**

**• Regularly update the system to fix bugs and improve performance.**

**• Future Enhancements:**

**• Add features like transaction history, multi-user support, and database integration.**

**• Implement advanced security measures such as OTP-based authentication.**

**This methodology ensures the ATM Management System is developed systematically, adhering to best practices in software engineering, and provides a scalable solution for learning and prototyping financial applications.**

**Conclusion: ATM Management System**

**The development of the ATM Management System using Python successfully demonstrates the simulation of essential banking operations in a secure and user-friendly environment. Through this project, key functionalities such as account creation, secure user authentication, balance inquiry, deposits, and withdrawals were implemented efficiently. The use of JSON for data persistence ensures reliability and allows data to be saved across sessions.**

**This project serves as a practical application of Python programming concepts, including object-oriented programming, file handling, and error management. It provides an educational platform for understanding the technical and operational aspects of ATM systems, making it a valuable tool for both learning and teaching.**

**The system has been designed to be modular and scalable, providing a foundation for future enhancements. Features such as transaction history, database integration, and advanced security mechanisms can be added to make the system more robust and closer to real-world applications.**

**In conclusion, the ATM Management System achieves its objectives by offering a functional, secure, and easy-to-use prototype that simulates real-world banking operations. It highlights the importance of structured problem-solving and the application of programming skills in creating practical solutions for everyday challenges.**

***Submitted by: Mayank Srivastava***

***14971***

***Under the Guidance of: Mrs. Surbhi Purwar***