DEPARTMENT OF INFORMATION TECHNOLOGY

CHAITANYA BHARATHI INSTITUTE OF TECHNOLOGY (A)

2024-2025

- **Title:** ATM SYSTEM
- **Name of students:** G. Krishna Manohar Reddy (160123737040)
  Kethavath Naveen (160123737045)
  Palle Sairam Goud (160123737056)
- **Institution and Department:** IT Department, CBIT
- **Guide/Supervisor's Name:** Ms. P. Kiranmaie (Assistant Professor, IT)
- **Submission Date:** 16-11-2024

TABLE OF CONTENTS

1. ABSTRACT

Problem Statement

In the modern world, Automated Teller Machines (ATMs) play a crucial role in providing essential banking services. However, designing a backend system that can efficiently handle multiple users, manage transactions, and maintain a robust data structure is a challenging task. This project addresses these challenges by simulating an ATM system capable of managing real-world scenarios like customer queuing, transaction processing, and account management.

Proposed Solution

This project aims to develop an **ATM System** using data structures in Python, incorporating features such as user authentication, queue management, transaction logging, and real-time balance updates. The system efficiently handles transactions like withdrawals, deposits, balance inquiries, and fund transfers while maintaining transaction history. By leveraging data structures such as queues and linked lists, the project demonstrates how ATMs manage backend processes effectively.

Data Structures Used: Queue, Linkedlist

Key Results or Observations:

The ATM System project successfully simulates core ATM functionalities, including secure user authentication, real-time transaction processing, and efficient customer queue management. The project enhances technical skills in data structure implementation and Python programming while providing an intuitive understanding of ATM backend operations. This simulation also highlights the importance of appropriate data structures for efficient system design.

2. INTRODUCTION

2.1 Understanding Sudoku:

ATMs (Automated Teller Machines) are indispensable in modern banking, enabling customers to perform various transactions like withdrawals, deposits, and balance inquiries. ATMs ensure a convenient and secure banking experience, handling multiple customers efficiently through systematic operations.

This project simulates an ATM backend, mimicking real-world ATM functionalities by incorporating customer queue management, secure authentication, and transaction logging. The simulation provides insights into backend processes and demonstrates how data structures can effectively manage customer interactions and transactions. The project is a valuable learning tool for understanding how ATMs work and the importance of efficient system design.

2.2 Problem Statement:

In a real-world scenario, ATMs manage numerous transactions daily, requiring robust backend systems to ensure security, accuracy, and efficiency. Handling concurrent customers, managing transaction histories, and providing seamless services are key challenges.

This project addresses these challenges by developing a functional ATM system using Python and data structures. The system focuses on providing essential banking operations like deposits, withdrawals, and transfers while ensuring smooth user interactions through customer queue management. Additionally, the system ensures data integrity and persistence across multiple sessions, simulating the core functionalities of an ATM.

2.3 Objective:

The main objective of the ATM System project is to develop a functional and efficient simulation of ATM operations. Specific goals include:

- Providing a secure user authentication mechanism.
- Managing customer queues to simulate real-world ATM usage.
- Facilitating core banking transactions such as deposits, withdrawals, balance inquiries, and fund transfers.
- Logging transaction history for each account and ensuring data persistence.

The project aims to demonstrate the practical application of data structures like queues and linked lists to enhance transaction processing efficiency and customer management.

2.4 Scope:

The scope of this ATM System project includes:

Core Functionality:

- **Transaction Processing:**
    - Supports withdrawals, deposits, balance inquiries, and fund transfers.
- **User Authentication:**
    - Ensures secure access to accounts using a PIN-based login system.
- **Queue Management:**
    - Simulates real-world customer queues, serving users in a first-come, first-served manner.
- **Transaction History Management:**
    - Logs each transaction using a linked list for efficient storage and retrieval.

Advanced Features:

- **Data Persistence:**

- Saves account data and transaction history to CSV files for continuity across sessions.

- **Real-Time Feedback:**
  - Displays user positions in the queue and provides real-time transaction updates.
- **Dynamic Queue Setup:**
  - Randomized queue generation to simulate real-world scenarios.
- **Transaction Receipts:**
  - Generates detailed receipts for every session's transactions.

METHODOLOGY

3.1 Data Structures Used:

1. **Queue:**

   A queue is a linear data structure that follows the First-In-First-Out (FIFO) principle.

- **Purpose in the project:**
  - Manages customer waiting lines in the ATM system.
  - Ensures customers are served in the order they arrive.
- **Functionality:**
  - Enqueue: Adds a customer to the queue.
  - Dequeue: Removes the next customer to be served.
  - Display: Shows the current queue status.

2. **LinkedList:**

   A linked list is a sequence of nodes where each node contains data and a reference to the next node.

- **Purpose in the project:**
  - Stores the transaction history of each account.
  - Maintains efficient addition of transactions in chronological order.

- **Functionality:**
  - Add Transaction: Appends a new transaction node to the list.
  - Display Transactions: Iterates through the list to show all stored transactions.
  - Save to CSV: Serializes the transaction history into a CSV file for persistence.

3.2.1 Queue Management Approach:

Efficient customer handling in ATMs is crucial for a smooth user experience. The ATM System uses a queue-based mechanism to simulate real-world ATM usage scenarios.

- **Steps:**
  1. **Customer Arrival:** When a user logs in, they are placed in the queue.
  2. **Queue Updates:** Customers in the queue are informed of their position until it's their turn.
  3. **Transaction Execution:** The system serves customers in a first-come, first-served manner.

3.2.2 Transaction Logging and Persistence:

- **Linked List:**
  - Each transaction (withdrawal, deposit, transfer) is stored as a node in the linked list.
  - This allows for efficient traversal and display of transaction history during a session.
- **CSV File Persistence:**
  - At the end of a session, the linked list is saved to a CSV file for future reference.
  - When the system restarts, transaction history is reloaded from the CSV files.

4. IMPLEMENTATION

4.1 Programming Environment

The ATM simulation project was primarily developed using the **Python** programming language. The main reasons for choosing Python include its readability, ease of use, and the large number of libraries and tools available for rapid development.

Core Libraries and Tools:

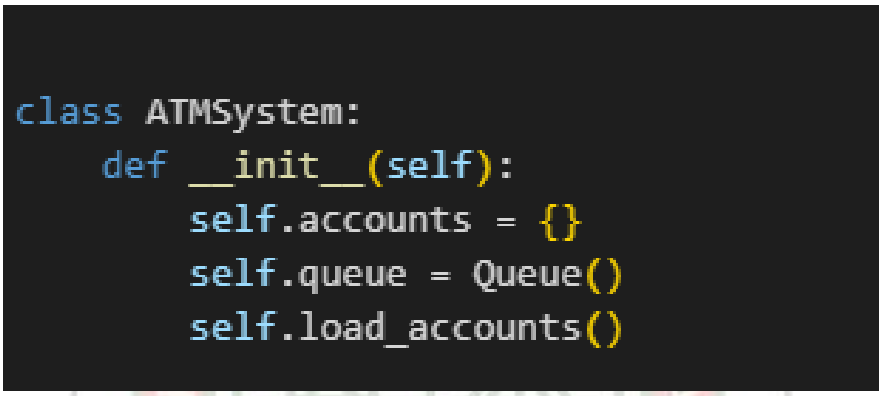
- **pandas:** This library was used for handling data related to accounts and transactions. It allows for easy loading, manipulation, and saving of account data and transaction histories using CSV files.
- **os:** This built-in library was used for file operations, such as checking if the account and transaction files already exist.
- **random:** This library was used to create randomness, particularly for simulating a queue of users in the ATM system. It was used to randomly shuffle the queue and generate the positions of users.
- **datetime:** This library was used to generate timestamps for transactions, recording the time when deposits, withdrawals, and transfers occurred.

A simple **text-based user interface (CLI)** has been implemented for the ATM simulation, allowing users to interact with the system using the command line. While this is functional, a more sophisticated graphical user interface (GUI) could be developed using libraries such as **Tkinter** or **Pygame** to improve the user experience and make it visually more appealing.

4.2 Code Overview:

ATMSystem Code:

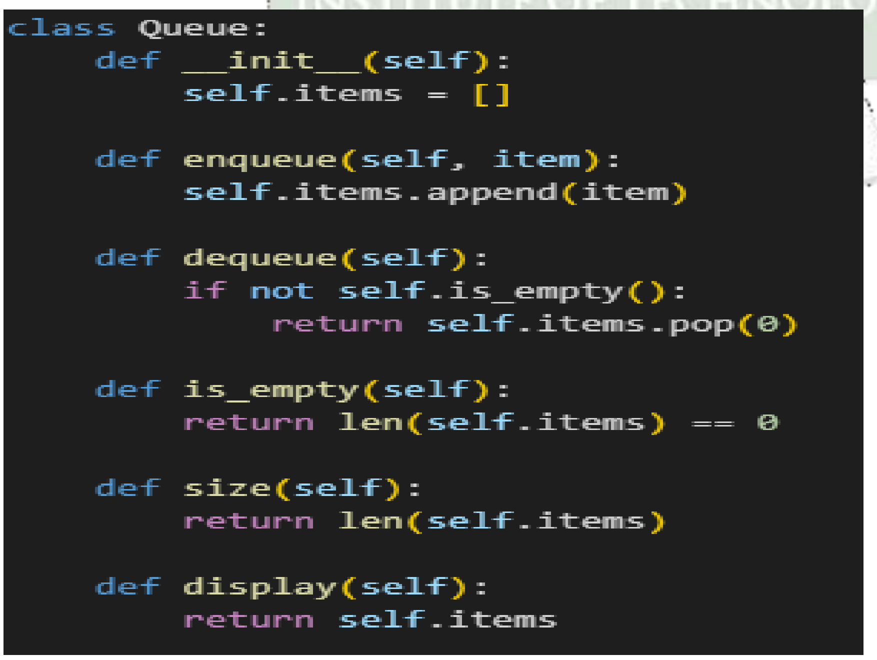
The **ATMSystem** class is the heart of the simulation. It manages the overall process of the ATM operation, including account management, queue handling, and transaction processing.



```python
class ATMSystem:
    def __init__(self):
        self.accounts = {}
        self.queue = Queue()
        self.load_accounts()
```

Queue Management:

The **Queue** class is used to simulate the line of customers at the ATM. Customers are added to the queue when they arrive, and they are processed in a first-come, first-served manner.



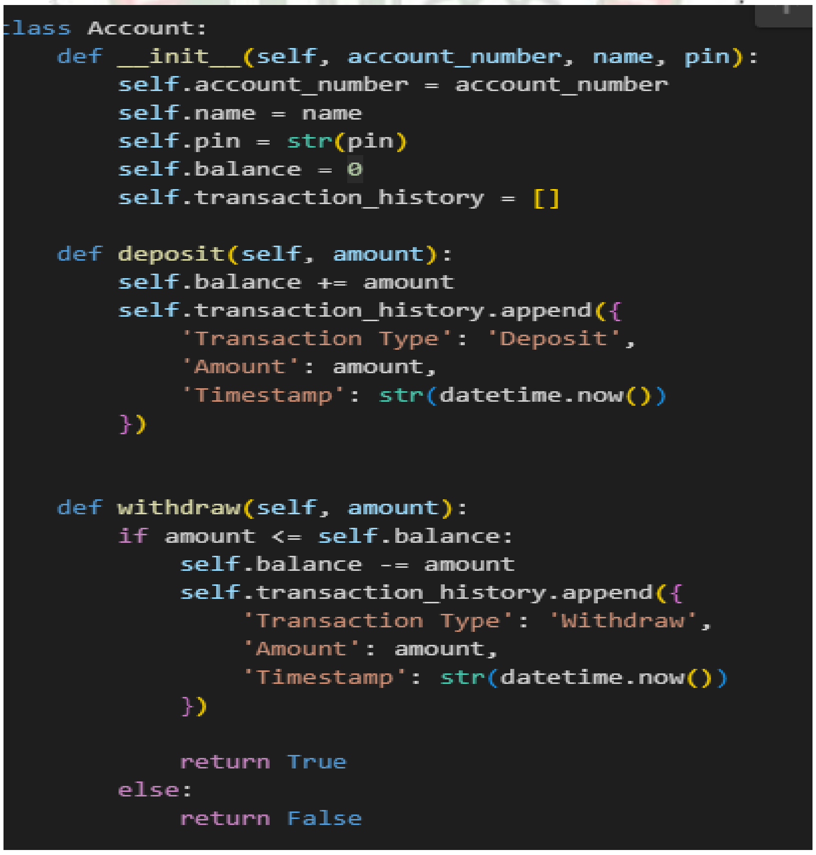
```python
class Queue:
    def __init__(self):
        self.items = []

    def enqueue(self, item):
        self.items.append(item)

    def dequeue(self):
        if not self.is_empty():
            return self.items.pop(0)

    def is_empty(self):
        return len(self.items) == 0

    def size(self):
        return len(self.items)

    def display(self):
        return self.items
```

Account Management:

The **Account** class represents an individual user's account and stores their information, such as the account number, PIN, balance, and transaction history

Key methods within the **Account** class include:

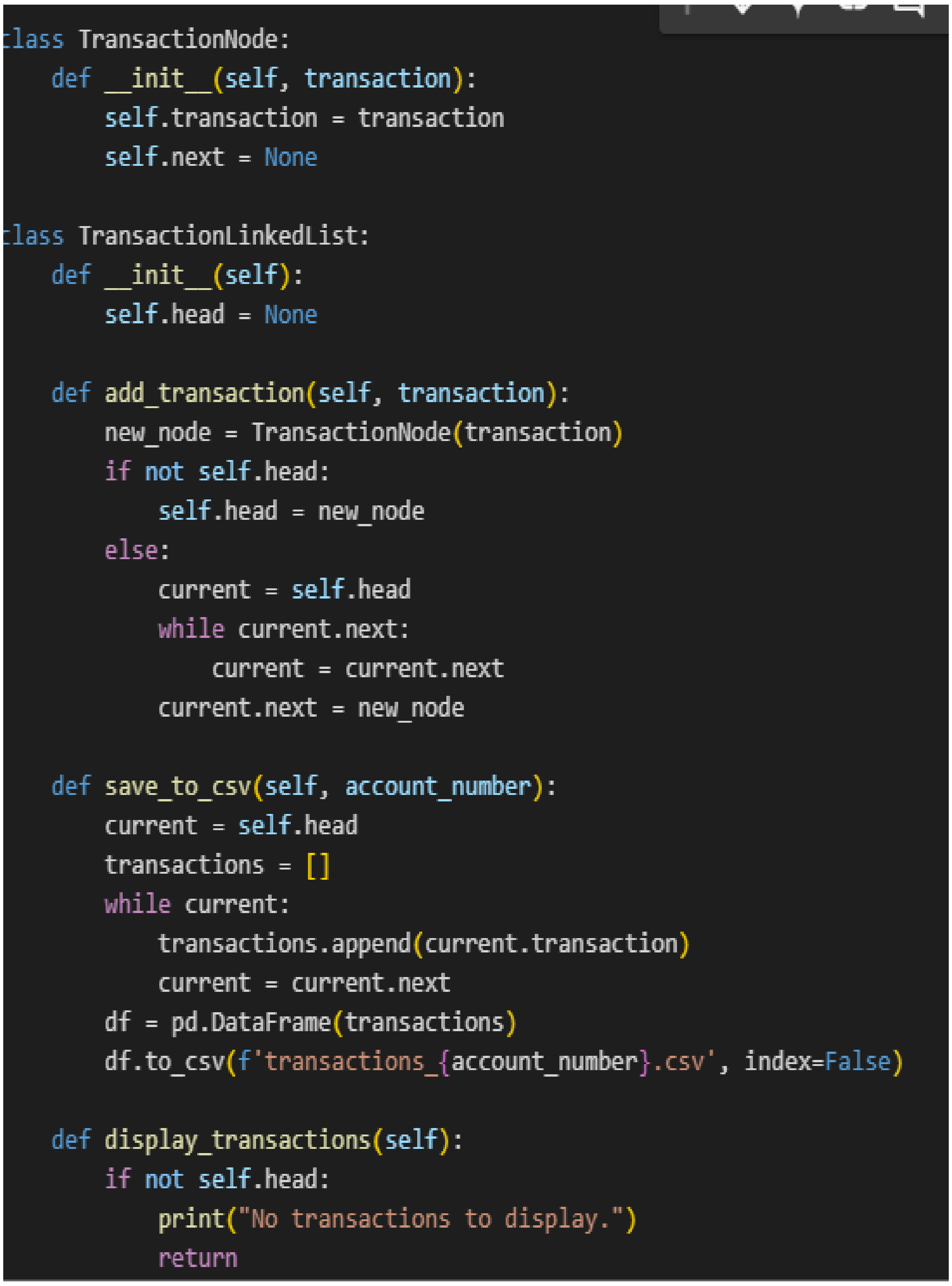
- deposit(amount): Deposits the specified amount into the account.
- withdraw(amount): Withdraws the specified amount if there are sufficient funds.
- transfer(amount, target_account): Transfers a specified amount to another account.
- get_balance(): Returns the current balance of the account.
- get_transaction_history(): Returns the transaction history for the account.



```
class Account:
    def __init__(self, account_number, name, pin):
        self.account_number = account_number
        self.name = name
        self.pin = str(pin)
        self.balance = 0
        self.transaction_history = []

    def deposit(self, amount):
        self.balance += amount
        self.transaction_history.append({
            'Transaction Type': 'Deposit',
            'Amount': amount,
            'Timestamp': str(datetime.now())
        })


    def withdraw(self, amount):
        if amount <= self.balance:
            self.balance -= amount
            self.transaction_history.append({
                'Transaction Type': 'Withdraw',
                'Amount': amount,
                'Timestamp': str(datetime.now())
            })

            return True
        else:
            return False
```

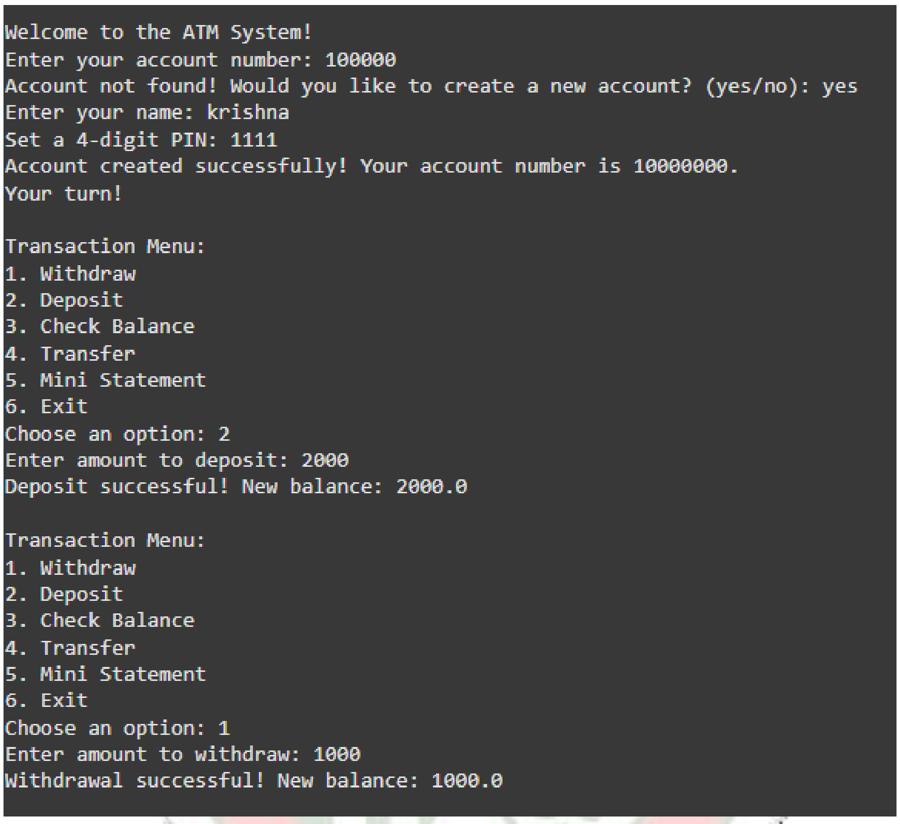
Transaction Mangement:

The **TransactionLinkedList** class is used to store each transaction made on the account. This linked list ensures that all transactions are preserved and can be accessed for future reference.

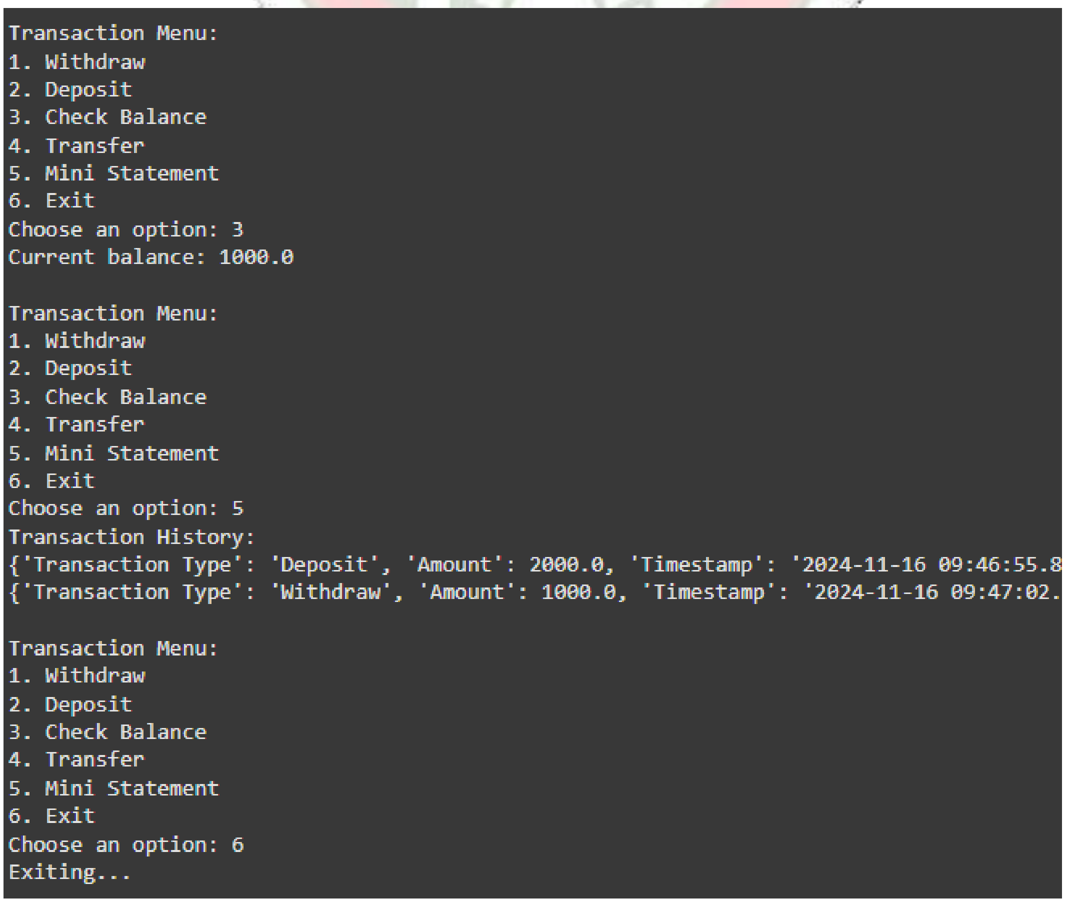


```
lass TransactionNode:
    def __init__(self, transaction):
        self.transaction = transaction
        self.next = None

lass TransactionLinkedList:
    def __init__(self):
        self.head = None

    def add_transaction(self, transaction):
        new_node = TransactionNode(transaction)
        if not self.head:
            self.head = new_node
        else:
            current = self.head
            while current.next:
                current = current.next
            current.next = new_node

    def save_to_csv(self, account_number):
        current = self.head
        transactions = []
        while current:
            transactions.append(current.transaction)
            current = current.next
        df = pd.DataFrame(transactions)
        df.to_csv(f'transactions_{account_number}.csv', index=False)

    def display_transactions(self):
        if not self.head:
            print("No transactions to display.")
            return
```

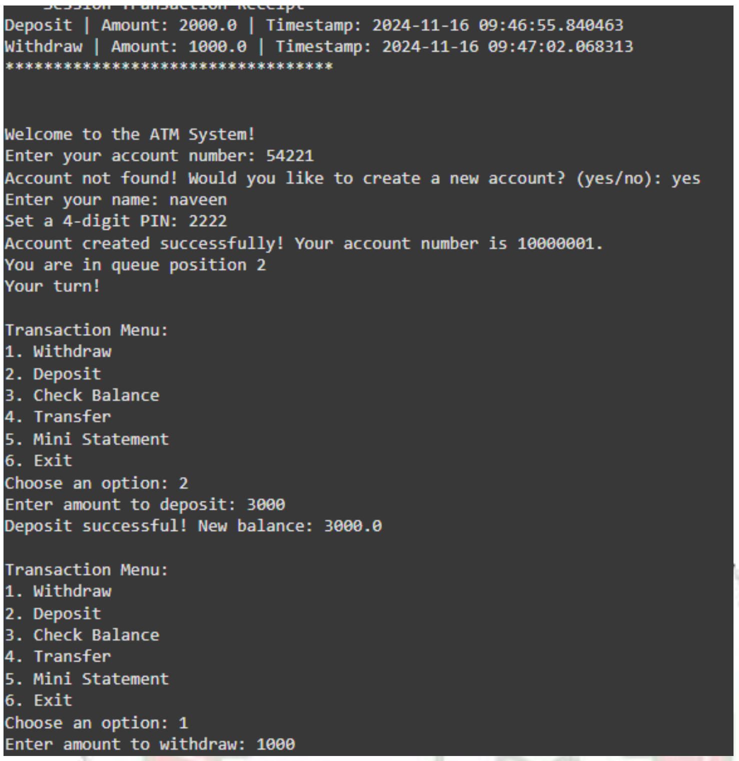
4.3 Output Screenshots:



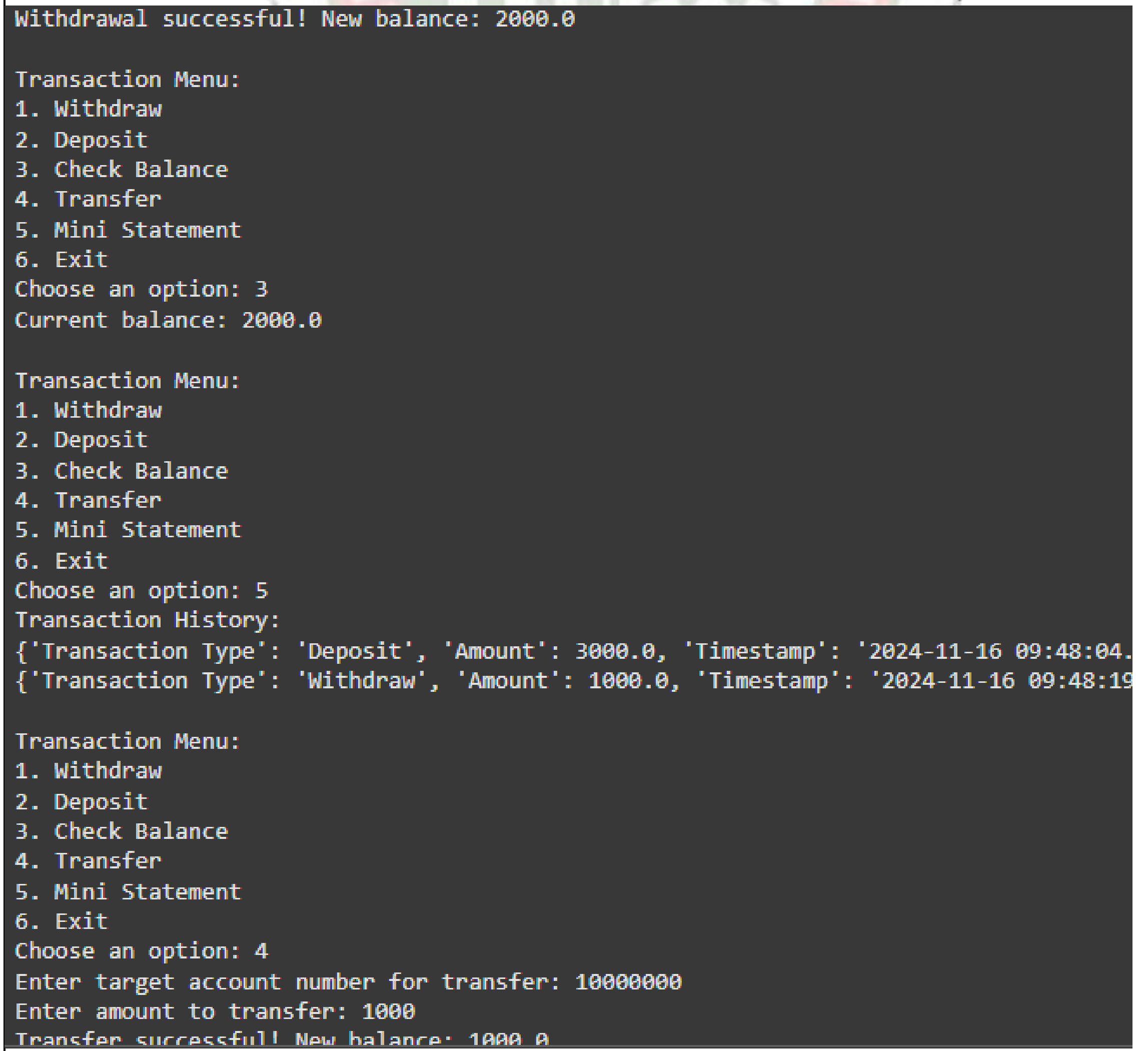
```
Welcome to the ATM System!
Enter your account number: 100000
Account not found! Would you like to create a new account? (yes/no): yes
Enter your name: krishna
Set a 4-digit PIN: 1111
Account created successfully! Your account number is 10000000.
Your turn!

Transaction Menu:
1. Withdraw
2. Deposit
3. Check Balance
4. Transfer
5. Mini Statement
6. Exit
Choose an option: 2
Enter amount to deposit: 2000
Deposit successful! New balance: 2000.0

Transaction Menu:
1. Withdraw
2. Deposit
3. Check Balance
4. Transfer
5. Mini Statement
6. Exit
Choose an option: 1
Enter amount to withdraw: 1000
Withdrawal successful! New balance: 1000.0
```



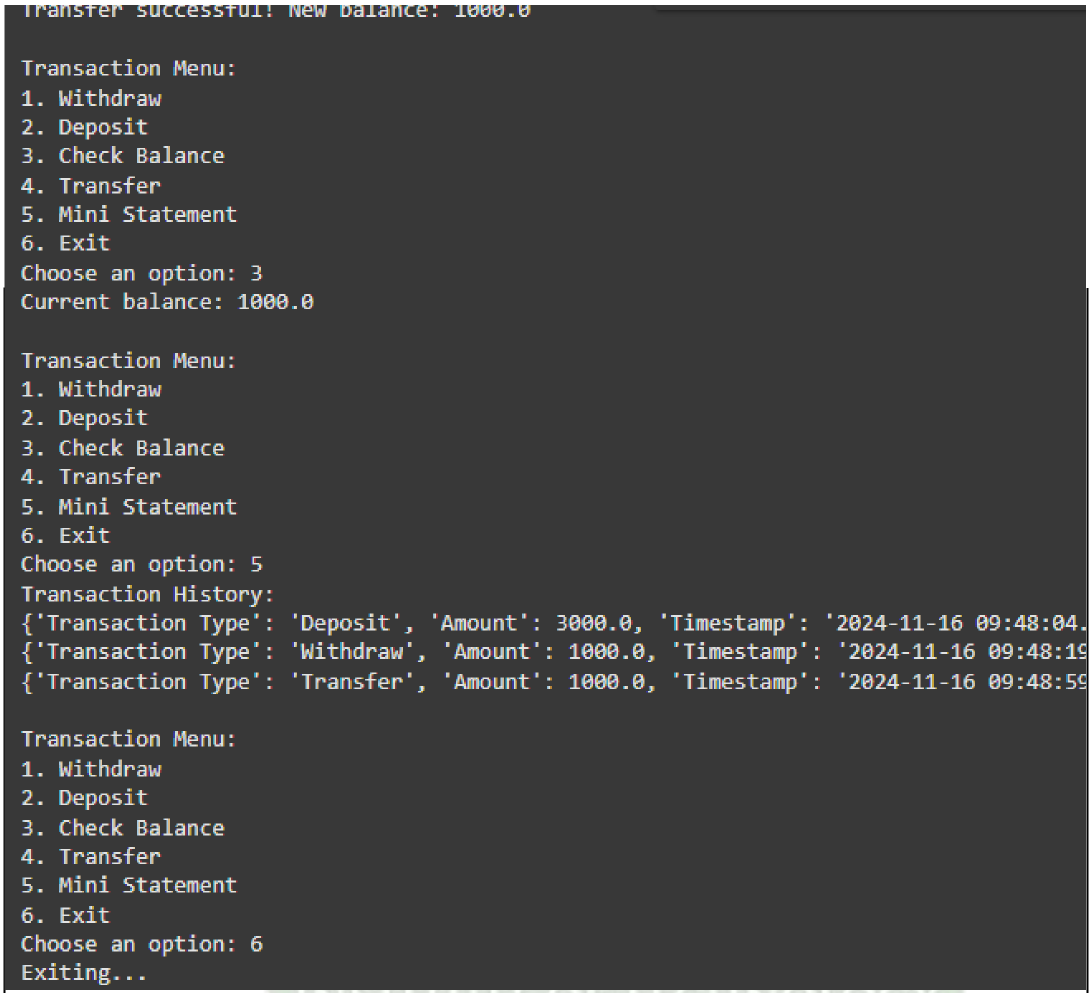
```
Transaction Menu:
1. Withdraw
2. Deposit
3. Check Balance
4. Transfer
5. Mini Statement
6. Exit
Choose an option: 3
Current balance: 1000.0

Transaction Menu:
1. Withdraw
2. Deposit
3. Check Balance
4. Transfer
5. Mini Statement
6. Exit
Choose an option: 5
Transaction History:
{'Transaction Type': 'Deposit', 'Amount': 2000.0, 'Timestamp': '2024-11-16 09:46:55.8
{'Transaction Type': 'Withdraw', 'Amount': 1000.0, 'Timestamp': '2024-11-16 09:47:02.

Transaction Menu:
1. Withdraw
2. Deposit
3. Check Balance
4. Transfer
5. Mini Statement
6. Exit
Choose an option: 6
Exiting...
```



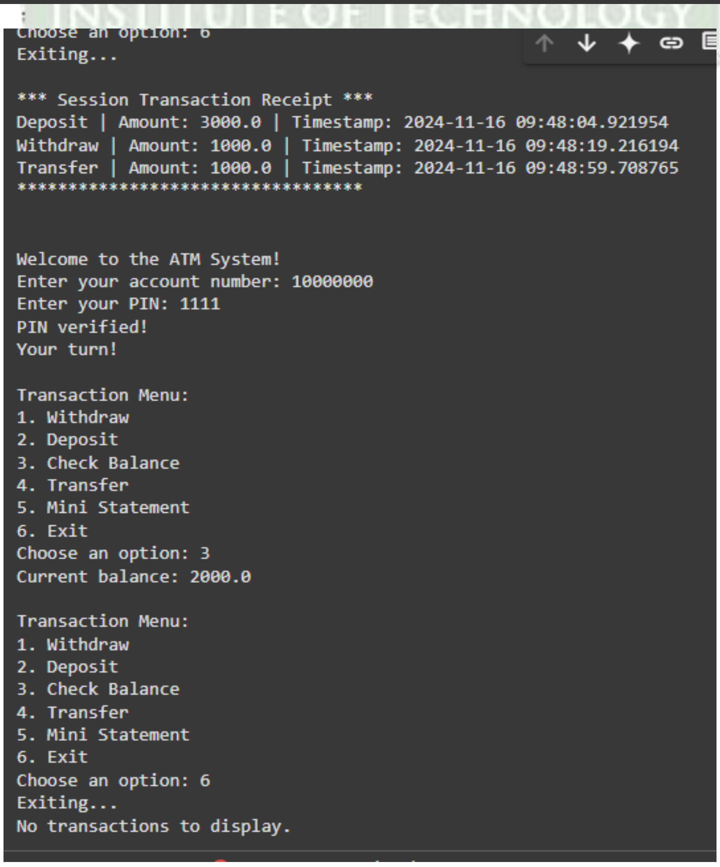
```
Session Transaction Receipt
Deposit | Amount: 2000.0 | Timestamp: 2024-11-16 09:46:55.840463
Withdraw | Amount: 1000.0 | Timestamp: 2024-11-16 09:47:02.068313
**********************************


Welcome to the ATM System!
Enter your account number: 54221
Account not found! Would you like to create a new account? (yes/no): yes
Enter your name: naveen
Set a 4-digit PIN: 2222
Account created successfully! Your account number is 10000001.
You are in queue position 2
Your turn!

Transaction Menu:
1. Withdraw
2. Deposit
3. Check Balance
4. Transfer
5. Mini Statement
6. Exit
Choose an option: 2
Enter amount to deposit: 3000
Deposit successful! New balance: 3000.0

Transaction Menu:
1. Withdraw
2. Deposit
3. Check Balance
4. Transfer
5. Mini Statement
6. Exit
Choose an option: 1
Enter amount to withdraw: 1000
```



```
Withdrawal successful! New balance: 2000.0

Transaction Menu:
1. Withdraw
2. Deposit
3. Check Balance
4. Transfer
5. Mini Statement
6. Exit
Choose an option: 3
Current balance: 2000.0

Transaction Menu:
1. Withdraw
2. Deposit
3. Check Balance
4. Transfer
5. Mini Statement
6. Exit
Choose an option: 5
Transaction History:
{'Transaction Type': 'Deposit', 'Amount': 3000.0, 'Timestamp': '2024-11-16 09:48:04.
{'Transaction Type': 'Withdraw', 'Amount': 1000.0, 'Timestamp': '2024-11-16 09:48:19

Transaction Menu:
1. Withdraw
2. Deposit
3. Check Balance
4. Transfer
5. Mini Statement
6. Exit
Choose an option: 4
Enter target account number for transfer: 10000000
Enter amount to transfer: 1000
Transfer successful! New balance: 1000.0
```



```
Transfer successful! New balance: 1000.0

Transaction Menu:
1. Withdraw
2. Deposit
3. Check Balance
4. Transfer
5. Mini Statement
6. Exit
Choose an option: 3
Current balance: 1000.0

Transaction Menu:
1. Withdraw
2. Deposit
3. Check Balance
4. Transfer
5. Mini Statement
6. Exit
Choose an option: 5
Transaction History:
{'Transaction Type': 'Deposit', 'Amount': 3000.0, 'Timestamp': '2024-11-16 09:48:04.
{'Transaction Type': 'Withdraw', 'Amount': 1000.0, 'Timestamp': '2024-11-16 09:48:19
{'Transaction Type': 'Transfer', 'Amount': 1000.0, 'Timestamp': '2024-11-16 09:48:59

Transaction Menu:
1. Withdraw
2. Deposit
3. Check Balance
4. Transfer
5. Mini Statement
6. Exit
Choose an option: 6
Exiting...
```



```
Choose an option: 6
Exiting...

*** Session Transaction Receipt ***
Deposit | Amount: 3000.0 | Timestamp: 2024-11-16 09:48:04.921954
Withdraw | Amount: 1000.0 | Timestamp: 2024-11-16 09:48:19.216194
Transfer | Amount: 1000.0 | Timestamp: 2024-11-16 09:48:59.708765
**********************************


Welcome to the ATM System!
Enter your account number: 10000000
Enter your PIN: 1111
PIN verified!
Your turn!

Transaction Menu:
1. Withdraw
2. Deposit
3. Check Balance
4. Transfer
5. Mini Statement
6. Exit
Choose an option: 3
Current balance: 2000.0

Transaction Menu:
1. Withdraw
2. Deposit
3. Check Balance
4. Transfer
5. Mini Statement
6. Exit
Choose an option: 6
Exiting...
No transactions to display.
```

5. RESULTS AND ANALYSIS

5.1 Performance Analysis

The performance of the ATM simulation is influenced by several factors, including queue management, account operations, and transaction processing. The primary performance concerns are the efficiency of the customer queue processing and the time taken to perform transaction operations such as withdrawals, deposits, and transfers.

Time Complexity:

Queue Management: The time complexity of enqueue and dequeue operations in the queue is O(1) since these operations only involve adding or removing an element from the front or rear of the queue.

Account Operations: The account operations such as deposit, withdrawal, balance check, and transfer all have time complexity O(1). Each of these operations simply involves modifying or accessing the account's balance or transaction history.

Transaction History Handling: The time complexity of adding a transaction to the transaction history is O(1) since it involves appending a transaction to the linked list.

Space Complexity:

- **Queue**: The space complexity for the queue is O(N), where N is the number of customers in the queue. Each customer in the queue is represented by an object, and the queue grows as more customers join.
- **Account Data**: The space complexity for storing account data is O(M), where M is the number of accounts. Each account holds information such as account number, name, PIN, balance, and transaction history.
- **Transaction History**: The space complexity for storing transaction history is O(T), where T is the number of transactions. Each transaction is stored as a node in a linked list, and the list grows as more transactions occur.

6. CONCLUSION

6.1 Summary:

The ATM simulation system provides a comprehensive and efficient platform for managing customer transactions. Key features include:

- **Queue Management**: A simple queue system ensures customers are processed in a first-come, first-served manner.
- **Account Operations**: Users can perform a range of operations such as deposits, withdrawals, transfers, and balance checks.
- **Transaction History**: All account transactions are recorded, and users can view their transaction history.

6.2 Future Work:

Advanced Queue Management: Implement priority queues for high-priority customers (e.g., VIP users) to optimize customer service.

Enhanced Security: Implement additional security features, such as multi-factor authentication, to ensure account safety.

Transaction Optimization: Integrate features like batch processing for transactions to handle a large volume of operations more efficiently.

Real-time Transaction Updates: Implement real-time transaction processing and notifications to provide instant feedback to users.

Mobile Integration: Expand the system to support mobile transactions, enabling users to interact with the ATM remotely.

REFERENCES

YouTube: -DSA COURSES

Abdul Bari

mycodeSchool

Coursera - Data Structures and Algorithms Specialization - A series of courses from the University of California, San Diego.

Udacity - Data Structures and Algorithms Nanodegree - A more in depth, project-based approach to learning DSA