ATM MANAGEMENT SYSTEM

A PROJECT REPORT

In the partial fulfillment of the requirements for the award of

MASTER OF COMPUTER SCIENCE

Submitted by

BENDALAM SRICHARAN

(Regd. No. 421228821002)

Under the Esteemed Guidance of

Sri. M. A. PRASAD	**Sri. M. A. PRASAD**
Sr. Assistant Professor	**Sr. Assistant Professor**
INTERNAL GUIDE	**HEAD OF THE DEPARTMENT**

DEPARTMENT OF COMPUTER SCIENCE

DR. LANKAPALLI BULLAYYA P.G COLLEGE

AFFILIATED TO ANDHRA UNIVERSITY

VISAKHAPATNAM-530003

2021-2023

Dr. Lankapalli Bullayya College

(Approved by AICTE New Delhi & Affiliated to Andhra University, Visakhapatnam)

52-14-75, Resapuvani Palem, Visakhapatnam - 530 013.

Ph. : 2561635, Fax. : 0891- 2714423, E-mail : directorpg@bullayyacollege.org

Website : www.bullayyacollege.org

NAAC Accredited

COMPUTER SCIENCE & MANAGEMENT STUDIES

CERTIFICATE

This is to certify that BENDALAM SAICHARAN bearing Register No.421228821002 of MSC 3rd year during the academic year 2021- 2023 has submitted the Project work titled under the guidance of M. RUPASREE Asst. Professor as partial fulfillment of ATM MANAGEMENT SYSTEM the requirements for the award of degree of Master of Computer Application in the Department of Computer Science-PG, Dr.Lankapalli Bullayya College, Visakhapatnam. This work is not submitted to any University for the award of any Degree. The Project details are furnished below.

DURATION OF THE PROJECT	:
SOFTWARE USED	: Python,Python Liberaries
BACK END	: DATABASE

Sri M.RUPASREE
Asst. Professor
Internal Guide

Sri. M.A. PRASAD
Sr. Asst. Professor
Head of the Department

EXTERNAL EXAMINER

ACKNOWLEDGMENT

At every outset I express my gratitude to almighty lord for showering his grace and blessings upon me to complete this project. Although our name appears on the cover of this book, many people had contributed in some form or the other form to this project Development. We could not done this project without the assistance or support of eachof the following we thank you all.

I wish to place on my record my deep sense of gratitude to my project guide, project work. M.A PRASAD for his constant motivation and valuable help through the project work.

I express my gratitude to Dr.G.S.K. CHAKRAVARTY , Dean of Dr. Lankapalli Bullayya College and Sri. M.A. PRASAD, Head of the department, Computer Science, Dr. Lankapalli Bullayya P.G College for his valuable suggestions and advises throughout the MSC COMPUTER SCIENCE Course

I also extend my thanks to other Faculty members for their Co-operation during my Course and also to my parents for their valuable support and guidance. Finally I would like to thank my friends for their co-operation to complete this project

BENDALAM SRICHARAN

Regd.No:421228821002

DECLARATION

I hereby declare that this project report entitled "**ATM MANAGEMENT SYSTEM**" is the result of original work done by me and to the best of my knowledge. A similar work has not been submitted previously to my other university of the requirement for the award of degree of Master of Computer Application

BENDALAM SRICHARAN
Regd.No:421228821002

INDEX

ABSTRACT

This is a simple ATM Management system which is very easy to use. Talking about the system, it contains various functions which include Account Statement, Withdrawing, Depositing amount and changing the pin. Here, at first the user has to enter an existing username, when the username matches the system proceed toward the next procedure i.e asking pin number. When a user passes all these sign-in procedures, he/she can use all those features. It is too easy to use, he/she can check their respective account statements.

While depositing or withdrawing amount, he/she just has to enter the amount then the system calculates the total remaining balance of the respective account and displays to the user. And the user can view all these transactions from the account statement. In this ATM Simulator, the user can also change the pin number. For this, the user has to enter the New pin code and then confirm it in order to change the pin code. This simple console based ATM simulator provides the simple account balance management of a respective account. It contains all the essential features. There is no database connection or neither any external text or other files used in this mini project to save user's data. Everything is set inside the source code whether its pin code or the amount.

1. INTRODUCTION

1.1 INTRODUCTION TO THE PROJECT:

The ATM (Automated Teller Machine) Management System is a software-based solution designed to efficiently and securely manage the operations and functionality of automated teller machines in a banking or financial institution. It provides a centralized platform for monitoring, controlling, and maintaining a network of ATMs, ensuring smooth and reliable banking services for customers.

The primary objective of the ATM Management System is to streamline the management and maintenance of ATMs, enabling banks to deliver convenient and secure self-service banking options to their customers. It encompasses various modules and functionalities that facilitate ATM deployment, monitoring, transaction processing, cash management, maintenance, and reporting.

Key Features and Functionalities:

ATM Deployment: The system assists in the strategic planning and deployment of ATMs by identifying optimal locations based on factors such as customer traffic, accessibility, and security. It tracks the status of deployed ATMs, including hardware configurations, software versions, and connectivity.

Transaction Processing: The ATM Management System enables seamless processing of various banking transactions, such as cash withdrawals, balance inquiries, funds transfers, bill payments, and cardless transactions. It ensures secure authentication and authorization of customer transactions, maintaining the integrity of financial data.

Cash Management: Efficient cash management is a critical aspect of the ATM Management System. It helps banks monitor cash levels in each ATM, forecast cash requirements, and plan replenishment schedules. The system generates alerts for low cash levels, tracks cash movements, and optimizes cash logistics to minimize cash-out situations and reduce operational costs.

Monitoring and Maintenance: The system provides real-time monitoring of ATMs, capturing vital statistics such as transaction volumes, uptime, downtime, and error rates. It proactively detects and alerts for hardware or software issues, facilitating prompt maintenance and minimizing downtime. Remote troubleshooting capabilities and automated diagnostic tools ensure quick resolution of technical problems.

Security and Fraud Prevention: Security is of utmost importance in ATM operations. The ATM Management System incorporates robust security measures, including encryption of customer data, adherence to security standards and protocols, monitoring for suspicious activities, and integration with fraud detection systems. It helps banks safeguard customer information and prevent fraudulent transactions.

Reporting and Analytics: The system generates comprehensive reports and analytics on ATM performance, transaction trends, cash flow, and maintenance activities. These insights aid in monitoring operational efficiency, identifying areas for improvement, and making data-driven decisions to enhance customer experience and optimize costs.

Benefits of ATM Management System:

- Improved operational efficiency and uptime of ATMs
- Enhanced customer experience through seamless and secure transactions
- Effective cash management and reduced cash-out situations
- Proactive maintenance and reduced downtime
- Strengthened security and fraud prevention measures
- Data-driven decision making and optimization of resources
- Streamlined monitoring and reporting for better control and compliance

In summary, the ATM Management System plays a crucial role in efficiently managing and maintaining a network of ATMs, enabling banks to provide convenient and secure self-service banking options to their customers. It optimizes operational efficiency, enhances customer experience, ensures effective cash management, and strengthens security measures, thereby contributing to the overall success of the banking institution.

1.1.1 Scope:

The scope of an ATM (Automated Teller Machine) management system encompasses the entire lifecycle and operation of ATMs within a specific organization or network. It includes the management of hardware, software, security, and overall functionality of the ATMs.

The scope may include the following aspects:

ATM Deployment: Managing the installation and deployment of ATMs at various locations to ensure proper coverage and accessibility for users.

ATM Maintenance: Monitoring and maintaining the physical components of ATMs, including cash replenishment, receipt paper replacement, hardware repairs, and troubleshooting.

ATM Software Management: Ensuring the proper functioning and updating of ATM software, including the operating system, transaction processing software, security patches, and firmware updates.

ATM Network Management: Overseeing the connectivity and network infrastructure required for ATMs to operate seamlessly, including communication protocols, encryption, and secure data transmission.

Transaction Processing: Handling and managing the processing of ATM transactions, including cash withdrawals, deposits, balance inquiries, funds transfers, and other banking services offered through the ATM.

Security Management: Implementing and maintaining robust security measures to protect the ATM system from unauthorized access, fraud, skimming, malware attacks, and ensuring compliance with industry standards and regulations.

1.1.2 Problem Statement:

Design and develop an ATM (Automated Teller Machine) management system that efficiently handles various banking transactions and provides a seamless experience for customers while ensuring the security and integrity of the system.

Problem Description: The ATM management system aims to address the following key requirements and challenges:

User Authentication: Implement a secure and reliable authentication mechanism to ensure that only authorized users can access the ATM system and perform transactions. This includes PIN-based authentication, biometric authentication (fingerprint or iris scanning), and card validation.

Transaction Processing: Support a wide range of banking transactions, including cash withdrawals, balance inquiries, fund transfers, bill payments, and account statements. Ensure that these transactions are processed accurately and efficiently, with appropriate validations and error handling.

Cash Management: Manage the availability of cash in the ATM machines by monitoring and updating the cash inventory. Implement mechanisms to handle cash replenishment, cash withdrawal limits, and notifications for low cash levels.

Account Management: Maintain customer account information securely and handle account-related operations such as account creation, closure, and modification of personal details. Implement necessary security measures to protect sensitive customer data.

Transaction Logging and Reporting: Record all transactions performed at the ATM machines for auditing and reporting purposes. Generate transaction logs, including timestamps, transaction types, and account details. Provide reporting capabilities to analyze transaction data and generate periodic reports.

System Security: Implement robust security measures to protect the ATM system from unauthorized access, tampering, and fraud. This includes encryption of sensitive data, secure communication protocols, monitoring for suspicious activities, and integration with external security systems (such as surveillance cameras).

Error Handling and Recovery: Implement proper error handling mechanisms to handle exceptions, network failures, and system errors. Provide recovery procedures to handle interrupted transactions and ensure the system's availability and reliability.

User Interface and Accessibility: Design a user-friendly and intuitive interface for customers to interact with the ATM system easily. Ensure accessibility features for users with disabilities, including visual impairment and physical limitations.

Integration with Banking Systems: Integrate the ATM management system with the core banking system to facilitate real-time updates of customer account balances, transaction history, and other relevant information.

System Maintenance and Monitoring: Provide mechanisms for system administrators to perform maintenance tasks, such as software updates, database backups, and system diagnostics. Implement monitoring capabilities to track system performance, uptime, and security events.

The ATM management system should be scalable, secure, and reliable to handle a large number of concurrent users and transactions efficiently. It should comply with regulatory requirements and industry standards for ATM operations and security.

1.2 EXISTING SYSTEM:

Emerging technologies such as mobile banking, biometric authentication, or contactless payments may not be fully supported by the existing system. This can limit the system's ability to adapt to changing customer preferences and industry trends.

1.3 PROPOSED SYSTEM:

The proposed system for an ATM management system in Python would aim to enhance the functionality and user experience compared to the existing system. Here are some key features and improvements that could be included in the proposed system:

Graphical User Interface (GUI): Develop a user-friendly GUI using Tkinter or PyQt library to provide an intuitive and visually appealing interface for users to interact with the ATM system.

User Authentication and Authorization: Implement secure user authentication mechanisms, such as PIN verification or biometric authentication, to ensure that only authorized users can access the system.

Account Management: Allow users to create new accounts, update account information (e.g., name, address), and perform account-related operations, including balance inquiries, fund transfers, and account closures.

Transaction Processing: Enable various transaction types, such as cash withdrawals, cash deposits, bill payments, and balance inquiries. Implement robust transaction handling to ensure accuracy, security, and proper updates to account balances.

Error Handling and Validation: Implement error handling and validation mechanisms to detect and handle invalid inputs or erroneous transactions. Provide meaningful error messages to guide users and prevent unintended errors.

Transaction Logging and Audit Trail: Maintain a comprehensive log of all transactions and activities performed within the system. This includes recording transaction details, timestamps, user IDs, and any errors or exceptions encountered.

Security Measures: Implement security measures to protect against unauthorized access and fraudulent activities. This may include encryption of sensitive data, secure communication protocols, and monitoring mechanisms to detect suspicious behavior.

Integration with External Systems: Integrate with external systems, such as banking networks or payment gateways, to enable seamless transactions and ensure interoperability with existing banking infrastructure.

Reporting and Analytics: Provide reporting capabilities to generate transaction reports, account statements, and other relevant insights for users and administrators. This can help track ATM usage, identify trends, and monitor system performance.

System Administration: Include administrative features for system configuration, maintenance, and monitoring. This may include user management, ATM configuration settings, transaction limits, and system health checks.

Exception Handling: Implement exception handling mechanisms to gracefully handle unexpected errors and exceptions, ensuring that the system remains stable and responsive.

Documentation and Help Resources: Provide comprehensive documentation and help resources, including user manuals and on-screen prompts, to guide users through the system's functionalities and troubleshoot common issues.

It is important to note that the proposed system should be designed and developed with a strong focus on security, reliability, and usability to ensure a robust and user-friendly ATM management system.

Proposed System Architecture:

The system architecture of an ATM (Automated Teller Machine) management system typically consists of multiple components that work together to provide the necessary functionality and services. Here is a high-level overview of the system architecture:

User Interface Layer:

User Interface (UI): This layer includes the physical interface components such as the keypad, touchscreen, card reader, display screen, and any other input/output devices used by the ATM users to interact with the system.

ATM Controller Layer:

ATM Controller: This component handles the communication between the user interface and the backend systems. It manages user input, initiates transactions, and interacts with the hardware devices such as the card reader, dispenser, printer, etc.

Transaction Processing Layer:

- Core Banking System: The core banking system is responsible for processing financial transactions and maintaining customer account information. It handles functions such as account validation, balance inquiry, cash withdrawal, fund transfers, and transaction logging.
- Payment Gateway: If the ATM is connected to a payment gateway, it facilitates electronic funds transfers and authorization for debit/credit card transactions.
- Switching Network: In some cases, there may be a switching network that connects the ATM to the appropriate financial institutions or networks for transaction routing and authorization.

Security Layer:

- Authentication and Authorization: This component verifies the identity of the user, typically through a combination of the user's ATM card and PIN. It ensures that only authorized users can access their accounts and perform transactions.
- Encryption: To ensure secure communication between the ATM and backend systems, encryption mechanisms are used to protect sensitive data such as PINs, account numbers, and transaction details.

- Fraud Detection: The system may include fraud detection mechanisms to identify suspicious activities or fraudulent transactions and trigger appropriate security measures.

Backend Systems:

- Database: A database system is used to store and manage customer account information, transaction logs, and other relevant data.
- Banking APIs: The ATM system interacts with the banking APIs to retrieve customer account details, update balances, and perform financial transactions.
- Network Infrastructure: The system relies on a secure network infrastructure to facilitate communication between the ATM, backend systems, and external networks.

Monitoring and Maintenance:

- Monitoring System: This component monitors the health and performance of the ATM system, tracks transaction logs, and generates reports for system administrators and maintenance personnel.
- Maintenance Tools: Various tools and utilities are used for system diagnostics, software updates, configuration management, and remote administration of the ATMs.

It's important to note that the specific architecture may vary depending on the ATM vendor, banking regulations, network connectivity options, and other factors. The above architecture provides a general overview of the major components involved in an ATM management system.

MODULES:

User Management Module: This module handles user authentication and authorization. It allows users to create accounts, log in, change passwords, and manage their profile information.

Account Management Module: This module manages customer accounts, including creating new accounts, updating account details, and closing accounts. It also handles account-related operations such as balance inquiries and transaction history.

Transaction Processing Module: This module facilitates various transaction types, such as cash withdrawals, cash deposits, fund transfers, bill payments, and balance inquiries. It ensures the proper execution of transactions and updates account balances accordingly.

ATM Hardware Interface Module: This module interfaces with the physical components of the ATM, including the card reader, keypad, cash dispenser, receipt printer, and display. It handles interactions with these devices to enable user input, dispense cash, and provide transaction receipts.

Transaction Log Module: This module logs all transactions and activities performed on the ATM system. It maintains a record of each transaction, including details such as transaction type, date, time, account number, amount, and status.

Security Module: This module ensures the security and integrity of the ATM system. It includes features such as PIN encryption, user session management, transaction validation, and fraud detection mechanisms.

Reporting Module: This module generates various reports and statements related to ATM transactions and activities. It provides insights into transaction volumes, account balances, cash flow, and system performance.

Administration Module: This module is responsible for system administration tasks. It allows authorized personnel to manage ATM configurations, set transaction limits, monitor system health, and perform system maintenance tasks.

Notification Module: This module handles communication with customers through notifications. It can send alerts, transaction confirmations, and other relevant information via SMS, email, or mobile app notifications.

Integration Module: This module facilitates integration with external systems and services, such as banking networks, payment gateways, and third-party APIs. It enables seamless communication and interoperability between the ATM system and other banking systems.

These modules work together to provide a comprehensive ATM management system that ensures secure and efficient transactions for bank customers. The exact modules and their functionalities may vary depending on the specific requirements and features of the ATM system.

2. REQUIREMENT ANALYSIS

Requirements analysis or requirements engineering is a process used to determine the needs and expectations of a new product. It involves frequent communication with the stakeholders and end-users of the product to define expectations, resolve conflicts, and document all the key requirements.
One of the greatest challenges faced by any organization is to share the vision of the final product with the customers. Hence, a business requirements analysis involves a team effort of all the key stakeholders, software developers, end-users, and customer managers to achieve a shared understanding of what the product should do. This is always done in the early phase of any project to ensure that the final product conforms to all the requirements. Requirements are an essential part of any software project and the foundation on which all projects should be built. The gathering of and compiling of requirements for a software project is very much a partnership between the user of the software and the developer. Obviously the customer or software user needs to communicate to the developer what they need, but at the same time the developer needs to be able to anticipate needs and ask the right questions during the requirements gathering phase of a project.

2.1 FEASIBILITY STUDY

A feasibility study is conducted to assess the practicality and viability of the analysis of women's safety in Indian cities using machine learning on tweets. It evaluates various aspects to determine if the project is feasible and can be successfully implemented.

The key considerations involved in the feasibility analysis are

- Economic Feasibility
- Technical Feasibility
- Operational Feasibility
- Social Feasibility

Feasibility study for the existing ATM (Automated Teller Machine) management system involves assessing its viability and practicality from various perspectives. The key feasibility factors to consider are:

2.1.1 Economic feasibility:

This examines the financial aspects of the existing system. It involves analyzing the cost of operating and maintaining the ATMs, including hardware maintenance, software updates, and security measures. Additionally, the economic feasibility study considers the return on investment (ROI) and potential cost savings or revenue generation opportunities through increased customer convenience and transaction volumes.

2.1.2 Technical feasibility:

This assesses whether the existing system can be implemented and integrated with the bank's infrastructure without major technical challenges. It involves evaluating the compatibility of hardware, software, and network systems, as well as the scalability and flexibility of the existing system to accommodate future requirements.

2.1.3 Operational Feasibility:

This assesses the practicality and efficiency of the existing system from an operational standpoint. It involves evaluating the ease of use for customers and ATM operators, analyzing the system's reliability and uptime, and considering any potential disruptions or challenges in day-to-day operations. The study also considers the impact on existing workflows and processes and the feasibility of integrating the system with other banking systems.

2.1.4 Social Feasibility:

This assesses the acceptance and impact of the existing system on various stakeholders, including customers, employees, and the community. It considers factors such as customer satisfaction, ease of access to banking services, and the system's ability to cater to diverse user needs. The study also examines the social implications of ATM accessibility for individuals with disabilities or in remote areas.

2.1.5 Legal and Regulatory Feasibility:

This examines whether the existing system complies with relevant laws, regulations, and industry standards. It involves ensuring compliance with data privacy and security regulations, adherence to banking industry guidelines, and any legal requirements specific to ATM operations. The study also considers any potential legal risks or liabilities associated with the system.

2.2 SOFTWARE REQUIREMENT SPECIFICATION:

The production of the requirements stage of the software development process is Software Requirements Specifications (SRS) (also called a requirements document). This report lays a foundation for software engineering activities and is constructing when entire requirements are elicited and analyzed. SRS is a formal report, which acts as a representation of software that enables the customers to review whether it (SRS) is according to their requirements. Also, it comprises user requirements for a system as well as detailed specifications of the system requirements.

The SRS is a specification for a specific software product, program, or set of applications that perform particular functions in a specific environment. It serves several goals depending on who is writing it. First, the SRS could be written by the client of a system. Second, the SRS could be written by a developer of the system. The two methods create entirely various situations and establish different purposes for the document altogether. The first case, SRS, is used to define the needs and expectation of the users. The second case, SRS, is written for various purposes and serves as a contract document between customer and developer.

The SRS phase consists of two basic activities:

- **Problem Requirement Analysis**

The process is order and more nebulous of the two, deals with understand the problem, the goal and constraints.

- **Requirement Specification**

Here, the focus is on specifying what has been found giving analysis such as representation, specification languages and tools, and checking the specifications are addressed during this activity. The Requirement phase terminates with the production of the validate SRS document. Producing the SRS document is the basic goal of this phase.

- **Role of SRS**

The purpose of the Software Requirement Specification is to reduce the communication gap between the clients and the developers. Software Requirement Specification is the medium though which the client and user needs are accurately specified. It forms the basis of software development. A good SRS should satisfy all the parties involved in the system.

2.2.1 Functional Requirements:

Focusing on the domain of software engineering, a functional requirement refers to the set of requirements for the system to function as expected. A function in itself can be defined as a culmination of the input set, the procedure, and the output set. Functional requirements can refer to requirements pertaining to necessary calculations, necessary technical details, and vital data manipulation and processing among other explicit functionalities that define what a system is intended to undertake. Behavioral requirements detailing all the scenarios where the system utilizes the specified functional requirements are detailed under use cases.

Functional Requirements for Analysis of Women Safety in Indian Cities Using Machine Learning on Tweets:

User Authentication:

- The system should authenticate users using a combination of bank cards and PINs.
- The system should verify the validity of the card and the correctness of the PIN.

Account Access:

- The system should connect to the bank's core banking system to retrieve account information.
- The system should allow users to access their bank accounts associated with the inserted card.

Balance Inquiry:

- The system should allow users to check the current balance of their bank accounts.
- The system should retrieve the account balance from the bank's database and display it on the screen.

Cash Withdrawal:

- The system should allow users to request a specific amount of cash withdrawal from their bank accounts.
- The system should verify the availability of funds, deduct the requested amount from the account balance, and dispense the cash.

Fund Transfers:

- The system should allow users to transfer funds between their own accounts or to other bank accounts.
- The system should validate the transaction details and update the account balances accordingly.

Deposit Functionality:

- The system should support cash or check deposits.
- The system should verify and process the deposit by updating the account balance accordingly.

Statement Printing:

- The system should provide users with the option to request a printed statement of recent transactions or specific account details.

Change PIN:

- The system should allow users to change their PIN for security purposes.
- The system should prompt the user to enter the current PIN and then allow them to set a new PIN.

Error Handling:

- The system should handle errors such as card retention, incorrect PIN entries, insufficient funds, network or connectivity issues, and other transaction errors.

The system should display appropriate error messages on the screen and take necessary actions to resolve the issues.

2.2.2 Non-functional requirements:

In the fields pertaining to systems engineering and requirements engineering, the definition for non-functional requirement can be as a prerequisite that stipulates standards that can be utilized to review the functioning of a system, rather than examining explicit behaviors. This can be clearly distinguished from functional requirements that focus on defining explicit behaviors or functions. The strategy for employing functional requirements is detailed in the system design section. The strategy for employing nonfunctional requirements is detailed in the system architecture. Other commonly used terminologies for non-functional requirements are "constraints", "quality attributes", "quality goals", "quality of service requirements" and "non-behavioral requirements". Some of the quality attributes are enlisted below:

- Portability
- Security
- Maintainability
- Reliability
- Scalability
- Performance
- Reusability
- Flexibility

2.2.2.1 Performance:

- The system should be capable of handling a high volume of concurrent transactions efficiently without significant delays or system slowdowns.
- It should provide quick response times to user interactions, ensuring a smooth and seamless user experience.
- The system should be scalable to accommodate future growth and increased user demand.

2.2.2.2 Scalability:

The system should be scalable to handle increasing data volume and user load.
It should accommodate the growth of the dataset and support the addition of new features or analysis techniques in the future.

2.2.2.3 Reliability:

- The system should be highly reliable and available for use at all times, with minimal downtime for maintenance or system failures.
- It should have mechanisms in place to handle system failures, such as power outages, network disruptions, or hardware malfunctions.
- The system should have data backup and recovery procedures to prevent data loss and ensure business continuity.

2.2.2.4 Security:

- The system should ensure the confidentiality, integrity, and availability of customer data and transactions.
- It should implement robust security measures to prevent unauthorized access, data breaches, and fraudulent activities.
- The system should comply with industry security standards and regulations, such as PCI-DSS (Payment Card Industry Data Security Standard).

2.2.2.5 Usability:

- The system should have a user-friendly interface that is easy to navigate and understand, even for users with limited technical knowledge.
- It should provide clear instructions and feedback to guide users through each transaction.
- The system should support accessibility features, such as adjustable font sizes, screen readers, and color contrast options, to cater to users with disabilities.

2.2.2.6 Compatibility:

- The system should be compatible with different types of ATMs, hardware configurations, and operating systems.
- It should integrate seamlessly with the bank's existing systems and third-party services, such as core banking systems, payment gateways, and fraud detection systems.

2.2.2.7 Compliance:

- The system should comply with relevant industry regulations, such as banking regulations, data protection laws, and consumer protection laws.
- It should adhere to international standards and best practices for ATM management and security.

2.2.2.8 Performance Monitoring and Logging:

- The system should have built-in performance monitoring and logging capabilities to track system performance, identify bottlenecks, and troubleshoot issues.
- It should generate logs and reports that can be used for auditing, analysis, and system optimization purposes.

2.2.2 ENVIRONMENT & TECHNOLOGY REQUIREMENTS:

Python:

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

- **Python is Interpreted** − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- **Python is Interactive** − you can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- **Python is Object-Oriented** − Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- **Python is a Beginner's Language** − Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

History of Python

Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands.

Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, Small Talk, and Unix shell and other scripting languages.

Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL).

Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.

Python Features

Python's features include −

- **Easy-to-learn** − Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
- **Easy-to-read** − Python code is more clearly defined and visible to the eyes.
- **Easy-to-maintain** − Python's source code is fairly easy-to-maintain.
- **A broad standard library** − Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- **Interactive Mode** − Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.
- **Portable** − Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- **Extendable** − you can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- **Databases** − Python provides interfaces to all major commercial databases.
- **GUI Programming** − Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
- **Scalable** − Python provides a better structure and support for large programs than shell scripting.

Apart from the above-mentioned features, Python has a big list of good features, few are listed below −

- It supports functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte-code for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- It supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

Python is a must for students and working professionals to become a great Software Engineer especially when they are working in Web Development Domain. I will list down some of the key advantages of learning Python:

- **Python is Interpreted** − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.
- **Python is Interactive** − You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.
- **Python is Object-Oriented** − Python supports Object-Oriented style or technique of programming that encapsulates code within objects.
- **Python is a Beginner's Language** − Python is a great language for the beginner-level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

Characteristics of Python

Following are important characteristics of Python Programming −

- It supports functional and structured programming methods as well as OOP.
- It can be used as a scripting language or can be compiled to byte-code for building large applications.
- It provides very high-level dynamic data types and supports dynamic type checking.
- It supports automatic garbage collection.
- It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

Applications of Python

As mentioned before, Python is one of the most widely used language over the web. I'm going to list few of them here:

- **Easy-to-learn** − Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.
- **Easy-to-read** − Python code is more clearly defined and visible to the eyes.
- **Easy-to-maintain** − Python's source code is fairly easy-to-maintain.
- **A broad standard library** − Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.
- **Interactive Mode** − Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.

- **Portable** − Python can run on a wide variety of hardware platforms and has the same interface on all platforms.
- **Extendable** − You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.
- **Databases** − Python provides interfaces to all major commercial databases.
- **GUI Programming** − Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.
- **Scalable** − Python provides a better structure and support for large programs than shell scripting.

PyCharm:

PyCharm is the most popular IDE used for Python scripting language. This chapter will give you an introduction to PyCharm and explains its features. PyCharm offers some of the best features to its users and developers in the following aspects −

- Code completion and inspection
- Advanced debugging
- Support for web programming and frameworks such as Django and Flask

Features of PyCharm:

Besides, a developer will find PyCharm comfortable to work with because of the features mentioned below

Code Completion

PyCharm enables smoother code completion whether it is for built in or for an external package.

SQLAlchemy as Debugger

You can set a breakpoint, pause in the debugger and can see the SQL representation of the user expression for SQL Language code.

Git Visualization in Editor

When coding in Python, queries are normal for a developer. You can check the last commit easily in PyCharm as it has the blue sections that can define the difference between the last commit and the current one.

Code Coverage in Editor

You can run **.py** files outside PyCharm Editor as well marking it as code coverage details elsewhere in the project tree, in the summary section etc.

Package Management

All the installed packages are displayed with proper visual representation. This includes list of installed packages and the ability to search and add new packages.

Local History

Local History is always keeping track of the changes in a way that complements like Git. Local history in PyCharm gives complete details of what is needed to rollback and what is to be added.

Refactoring

Refactoring is the process of renaming one or more files at a time and PyCharm includes various shortcuts for a smooth refactoring process.

PYTHON LIBRARIES

Tkinter:

Tkinter is a Python library that provides a set of tools for creating graphical user interfaces (GUIs). It is the standard GUI toolkit for Python and is included with most Python installations. Tkinter is based on the Tcl/Tk framework, which provides a platform-independent way to create GUI applications.

Key features and capabilities of Tkinter include:

- **Cross-platform:** Tkinter allows you to create GUI applications that can run on different platforms, including Windows, macOS, and Linux.
- **Widget library:** Tkinter provides a wide range of built-in widgets, such as buttons, labels, entry fields, checkboxes, radio buttons, dropdown menus, and more. These widgets can be easily customized and arranged to create the desired GUI layout.
- **Event-driven programming:** Tkinter follows an event-driven programming model, where actions or events (such as button clicks or keypresses) trigger the execution of specific functions or methods. This allows you to create interactive applications that respond to user interactions.

- **Geometry management:** Tkinter provides several geometry managers (pack, grid, and place) that help you organize and position widgets within the application window.
- **Styling and customization:** You can customize the appearance of Tkinter widgets by specifying attributes like colors, fonts, sizes, and borders. You can also create custom widget styles to achieve a consistent look and feel throughout the application.
- **Dialog boxes and message boxes:** Tkinter includes various pre-built dialog boxes and message boxes, such as file selection dialogs, color pickers, message boxes for displaying alerts or asking for user input, and more.
- **Event binding and handling:** Tkinter allows you to bind events to specific widgets or application-level events and define corresponding event handlers to perform specific actions when those events occur.
- **Support for graphics and images:** Tkinter provides functions and classes for working with graphics and images, allowing you to draw shapes, lines, and text on the GUI, as well as display and manipulate image files.
- **Integration with other libraries:** Tkinter can be easily integrated with other Python libraries and frameworks, such as Matplotlib for data visualization, Pandas for data analysis, and more.
- **Simple and beginner-friendly:** Tkinter has a relatively simple and straightforward API, making it easy for beginners to get started with GUI programming in Python.

ImageTk:

ImageTk is a module in the Python Imaging Library (PIL) that provides support for working with images in Tkinter GUI applications. It allows you to display and manipulate images within Tkinter windows and widgets.

The ImageTk module provides the ImageTk.PhotoImage class, which is used to create a Tkinter-compatible image object from a PIL image. This image object can then be used to display the image in Tkinter widgets such as labels, buttons, canvases, and more.

The ImageTk module also provides additional functions and methods for working with images, such as resizing, cropping, rotating, and applying filters. These operations can be performed on the PIL image before creating the PhotoImage object for display in Tkinter.

2.2.3.1. Hardware Requirements:

• System	:	Pentium i3 Processor.
• Hard Disk	:	500 GB.
• Monitor	:	15’’ LED
• Input Devices	:	Keyboard, Mouse
• Ram	:	4GB Above

2.2.3.2. Software Requirements:

• Operating System	:	Windows 7/8/10
• Server side Script	:	Python
• Libraries Used	:	ImageTk, Tkinter
• Backend	:	SQlite3
• Technology	:	Python 3.6+

3. DESIGN

The waterfall model is a sequential software development process that follows a linear and structured approach. Here's how the ATM management system can be developed using the waterfall model:

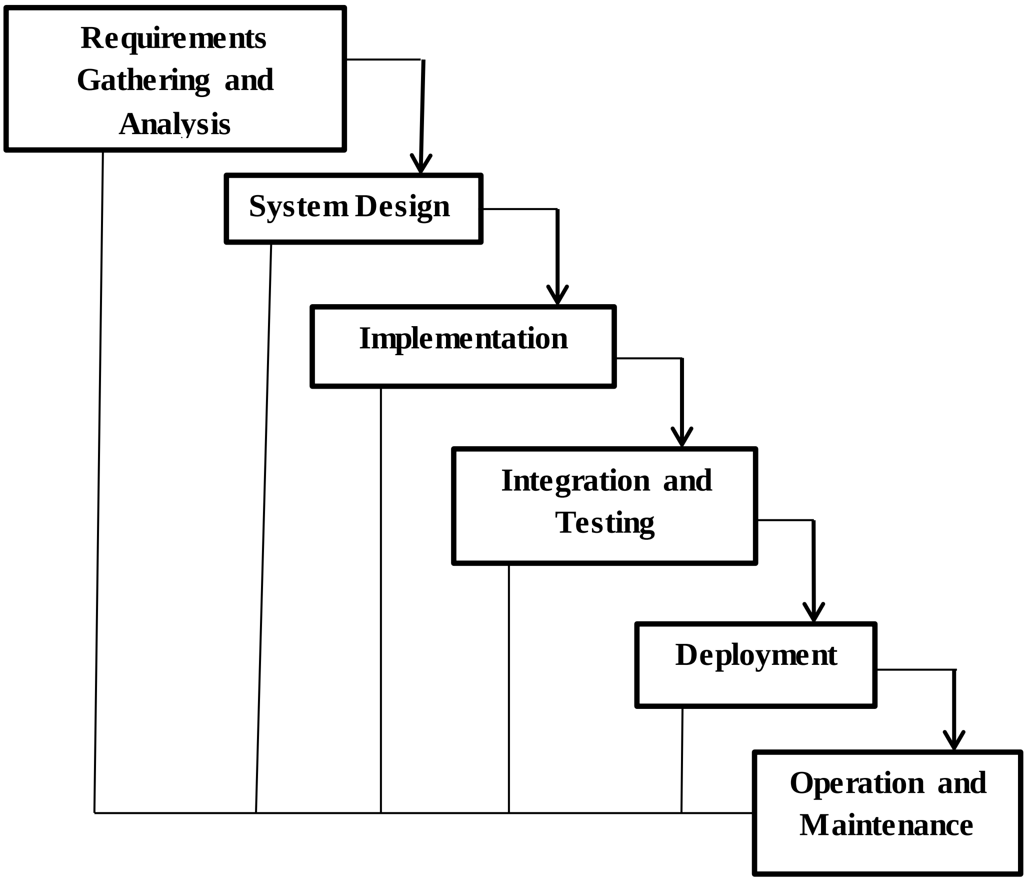

Requirements Gathering and Analysis
System Design
Implementation
Integration and Testing
Deployment
Operation and Maintenance

Figure 6: Waterfall Model

Requirements Gathering and Analysis:

- Gather requirements for the ATM management system, including functional and non-functional requirements.
- Analyze the requirements to understand the scope and objectives of the system.

System Design:

- Create a high-level system design based on the requirements.
- Define the architecture of the ATM management system, including components, modules, and their interactions.
- Design the user interface for the ATM system.

Implementation:

- Write code for the ATM management system based on the design specifications.
- Develop modules and functions for different features of the ATM system, such as cash withdrawal, balance inquiry, transaction history, etc.
- Perform unit testing to ensure the correctness of individual modules.

Integration and Testing:

- Integrate the different modules and components of the ATM management system.
- Conduct system-level testing to verify the functionality, performance, and reliability of the system.
- Perform integration testing to ensure the seamless operation of the system as a whole.

Deployment:

- Prepare the ATM management system for deployment in the production environment.
- Install the necessary software and hardware components required for the ATM system.
- Perform system testing to ensure that the system is ready for live operation.

Operation and Maintenance:

- Deploy the ATM management system in the production environment.
- Provide training to ATM operators and users.
- Monitor the system for any issues or bugs and provide regular maintenance and support.

The waterfall model is characterized by its sequential and linear nature, with each phase being completed before moving on to the next. This model is suitable for projects where requirements are well-defined and unlikely to change significantly during the development process.

However, it's worth noting that the waterfall model may not be suitable for projects with rapidly evolving requirements or where there is a need for frequent feedback and iteration. In such cases, agile methodologies like Scrum or Kanban are often preferred.

3.1. SYSTEM DESIGN:

Input Design:

The input design for an ATM management system typically involves capturing user input for various functionalities and transactions. Here are some examples of system inputs in an ATM management system:

User Authentication:

- User ID: The unique identifier of the ATM cardholder.
- PIN (Personal Identification Number): A numeric code entered by the user to authenticate their identity.

Transaction Inputs:

- Transaction Type: Selection of the desired transaction, such as withdrawal, deposit, balance inquiry, funds transfer, etc.
- Transaction Amount: The amount of money involved in the transaction.

Account Information:

- Account Number: The unique identifier of the user's bank account.
- Account Type: The type of the bank account, such as savings, current, etc.
- Account Balance: The current balance of the user's account.

Additional Inputs:

- Language Selection: The preferred language for the ATM interface.
- Receipt Option: Selection of whether the user wants a printed receipt for the transaction.

Maintenance Inputs:

- Supervisor/Administrator Credentials: User ID and password for system maintenance and administration tasks.
- Cash Loading Information: Input of the amount of cash being loaded into the ATM machine during maintenance.

These inputs can be captured through various input devices, such as a keypad for entering numeric values, touchscreens for selecting options, and card readers for reading ATM cards.

It is important to consider input validation and error handling mechanisms to ensure that the entered data is accurate and to provide appropriate feedback to the user in case of invalid inputs or system errors. Additionally, security measures like encrypting sensitive inputs and implementing secure communication protocols should be considered to protect user information during the input process.

Output Design:

The system output design for an ATM management system typically includes various screens and messages that provide information to the users and facilitate their interactions with the system. Here are some examples of system outputs in an ATM management system:

Welcome Screen: When a user approaches the ATM machine, a welcome screen is displayed, providing a greeting message and instructions to start the transaction.

Card Insertion Prompt: Once the user inserts their ATM card, a message is displayed asking them to enter their PIN to proceed further.

Transaction Menu: After successful authentication, a transaction menu is shown on the screen, presenting options such as withdrawal, balance inquiry, funds transfer, bill payment, and other available services.

Transaction Confirmation: When the user selects a specific transaction, such as a withdrawal, a confirmation screen is displayed, showing the details of the transaction, including the amount to be withdrawn and the account balance.

Transaction Result: After the completion of a transaction, the system provides a message indicating the outcome, such as "Withdrawal successful" or "Transaction failed." It may also display the new account balance if applicable.

Receipt Generation: If the user requests a receipt, the system generates a printable receipt containing transaction details, including date, time, transaction type, amount, account number, and any additional information.

Error Messages: In case of any errors or exceptions during the transaction, the system displays relevant error messages to guide the user and provide information about the issue.

Transaction Completion Message: After the user completes their transaction and removes their ATM card, a message is displayed thanking them for using the ATM and indicating that they can proceed.

Network Status: If the ATM is temporarily unable to connect to the banking network, a message is displayed informing the user about the network status and suggesting they try again later.

System Messages: The system may display general messages, such as maintenance notifications, system updates, or important announcements, to keep users informed about any relevant information.

It's important to note that the specific design and layout of the system outputs may vary depending on the ATM machine and the requirements of the ATM management system. The examples provided above represent common components of an ATM system output design.

3.1.1. Introduction to UML:

UML stands for Unified Modeling Language. UML is a standardized general-purpose modeling language in the field of object-oriented software engineering. The standard is managed, and was created by, the Object Management Group. The goal is for UML to become a common language for creating models of object oriented computer software. In its current form UML is comprised of two major components: a Meta-model and a notation. In the future, some form of method or process may also be added to; or associated with UML.

The Unified Modeling Language is a standard language for specifying, Visualization, Constructing and documenting the artifacts of software system, as well as for business modeling and other non-software systems. A UML system is represented using five different views that describe the system from distinctly perspective. Each view is defined by a set of diagrams, which is as follows.

- Use-case view
- Logical view
- Implementation view
- process view
- deployment view

Use-case view

A view showing the functionality of the system as perceived by external actors. An actor interacts with system. The actor can be a user or another system. The use-case view is used by customers, designers, developers and testers. It is described in use-case diagrams, sometimes with support from activity diagrams. The desired usage of the system is described as a number of use cases in the use-case view, where a use case is a generic description of a function requested.

Logical view

A view showing how the functionality is designed inside the system, in terms of the system‘s static structure and dynamic behavior. It is mainly for the designers and developers. It describes both static structure (classes, objects and relationships) and the dynamic collaborations that occur when the objects send messages to each other to provide a given function.

Properties such as persistence and concurrency are also defined, as well as the interfaces and the internal structure of classes. The static structure is described in class and object diagrams. The dynamic modeling is described in state machines and interaction and activity diagrams.

Implementation view

The implementation view describes the main modules and their dependencies. It is mainly for developers and consists of the main software artifacts. The artifacts include different types of code modules shown with their structure and dependencies.

Process view

A view showing main elements in the system related to process performance. This view includes scalability, throughput, and basic time performance and can touch on some very complex calculations for advanced systems. The view consists of dynamic diagrams (state machines, interaction and activity diagrams) and implementation diagrams (interaction and deployment diagrams).

Deployment view

Finally the deployment view shows the physical deployment of the system such as the computers and devices (nodes) and how they connect to each other. The various execution environments within the processors can be specified as well. The deployment view is used by the developers, integrators and testers. It is represented by the deployment diagram.

3.1.2. UML Diagrams:

The diagrams contain the graphical elements arranged to illustrate a particular part or aspect of the system. A system model typically has several diagrams of varying types, depending on the goal for the model.

Software design is a process that gradually changes as various new, better and more Complex methods with a broader understanding of the whole problem in general come in to existence. There are various kinds of diagrams used in software design.

Mainly these are as follows

- Use case diagrams
- Class diagram
- Object diagram
- Sequence diagrams
- Collaboration diagrams
- Activity diagram
- State chart diagram
- Component diagram
- Deployment diagram

Use case diagram

Use Case diagrams identify the functionality provided by the system (use cases), the users who interact with the system (actors), and the association between the users and the functionality. Use Cases are used in the Analysis phase of software development to articulate the high-level requirements of the system.

The primary goals of Use Case diagrams include

- Providing a high-level view of what the system does
- Identifying the users ("actors") of the system
- Determining areas needing human-computer interfaces

Use Cases extend beyond pictorial diagrams. In fact, text-based use case descriptions are often used to supplement diagrams, and explore use case functionality in more detail.

Class Diagram

Class diagrams identify the class structure of a system, including the properties and methods of each class. Also depicted are the various relationships that can exist between classes, such as an inheritance relationship. The Class diagram is one of the most widely used diagrams from the UML specification. Part of the popularity of Class diagrams stems from the fact that many CASE tools, such as Rational XDE, will auto-generate code in a variety of languages, including Java, C++, and C#, from these models. These tools can synchronize models and code, reducing your workload, and can also generate Class diagrams from object-oriented code, for those "code-then-design" maintenance projects.

Object Diagram

An object diagram is a variant of a class diagram and uses almost identical notation. The difference between the two is that an object diagram shows a number of object instances of classes, instead of the actual classes.

An object diagram is thus an example of a class diagram that shows a possible snapshot of the system‘s execution-what the system can look like at some point in time. The same notation as that for class diagrams is used, with two exceptions: Objects are written their names underlined and all instances in a relationship are shown. Object diagrams are not as important as class diagrams but they can be used to exemplify a complex class diagram by showing what the actual instances and the relationships look like. Objects are also used as part of interaction diagrams that show the dynamic collaboration between a set of objects.

Sequence Diagram

Sequence diagrams document the interactions between classes to achieve a result, such as a use case. Because UML is designed for object-oriented programming, these communications between classes are known as messages. The Sequence diagram lists objects horizontally, and time vertically, and models these messages over time

Collaboration Diagram

Like the other Behavioral diagrams, Collaboration diagrams model the interactions between objects. This type of diagram is a cross between an object diagram and a sequence diagram. Unlike the Sequence diagram, which models the interaction in a column and row type format, the Collaboration diagram uses the free-form arrangement of objects as found in an Object diagram. This makes it easier to see all integrations involving a particular object.

In order to maintain the ordering of messages in such a free-form diagram, messages are labeled with a chronological number. Reading a Collaboration diagram involves starting at message 1.0, and following the messages from object to object.

Activity Diagram

Activity diagrams are used to document workflows in a system, from the business level down to the operational level. When looking at an Activity diagram, you'll notice elements from State diagrams. In fact, the Activity diagram is a variation of the state diagram where the "states" represent operations, and the transitions represent the activities that happen when the operation is complete.

State chart Diagram

State chart diagrams, also referred to as State diagrams, are used to document the various modes ("state") that a class can go through, and the events that cause a state transition. For example, your television can be in the off state, and when the power button is pressed, the television goes into the on state. Pressing the power button yet again causes a state transition from the on state to the off state. In comparison the other behavioral diagrams which model the interaction between multiple classes, State diagrams typically model the transitions within a single class.

Component Diagram:

Component diagrams fall under the category of an implementation diagram, a kind of diagram that models the implementation and deployment of the system. A Component Diagram, in particular, is used to describe the dependencies between various software components such as the dependency between executable files and source files. This information is similar to that within make files, which describe source code dependencies and can be used to properly compile an application.

Deployment diagram

Deployment diagram depicts a static view of the run-time configuration of processing nodes and the components that run on those nodes. In other words, deployment diagrams show the hardware for your system, the software that is installed on that hardware, and the middleware used to connect disparate machines to one another.

3.1.2.1 Use-Case Diagrams:

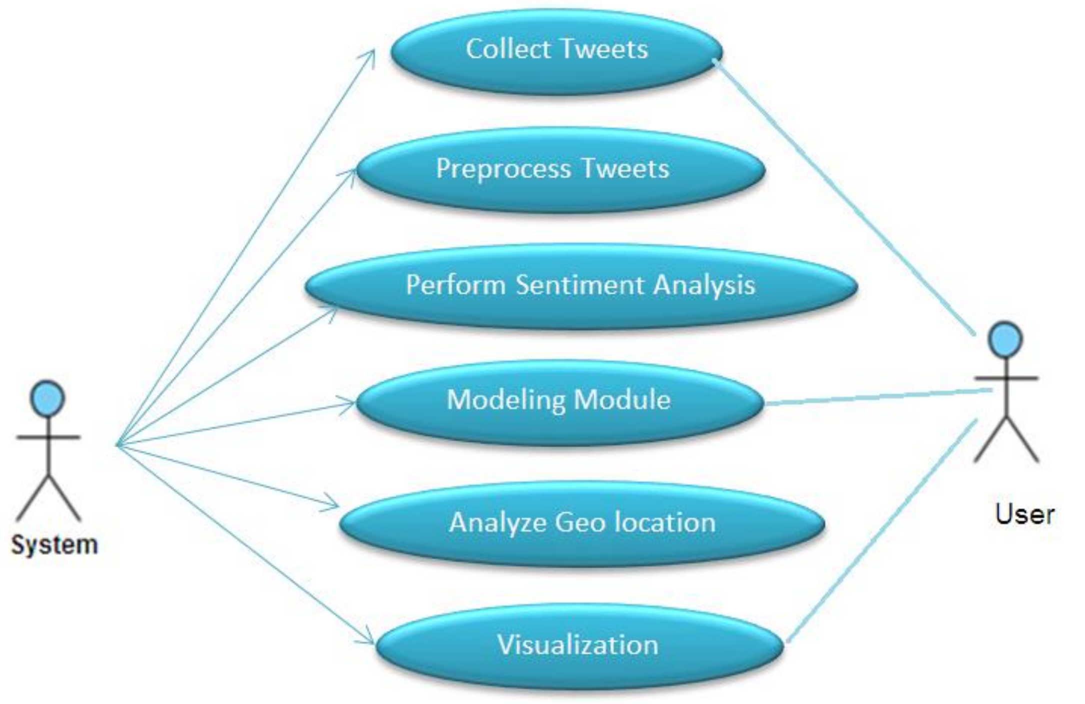


Figure 7: Use case Diagram

3.1.2.2. Class Diagram:

In software engineering, a class diagram in the Unified Modeling Language (UML) is a type of static structure diagram that describes the structure of a system by showing the system's classes, their attributes, operations (or methods), and the elationships among the classes. It explains which class contains information.

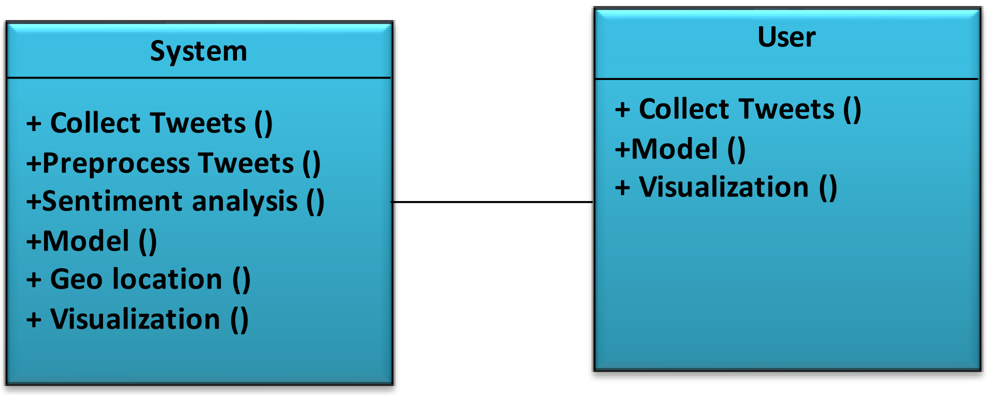


Figure 8: Class diagram

3.1.2.3. Sequence Diagram:

A sequence diagram in Unified Modeling Language (UML) is a kind of interaction diagram that shows how processes operate with one another and in what order. It is a construct of a Message Sequence Chart. Sequence diagrams are sometimes called event diagrams, event scenarios, and timing diagram.

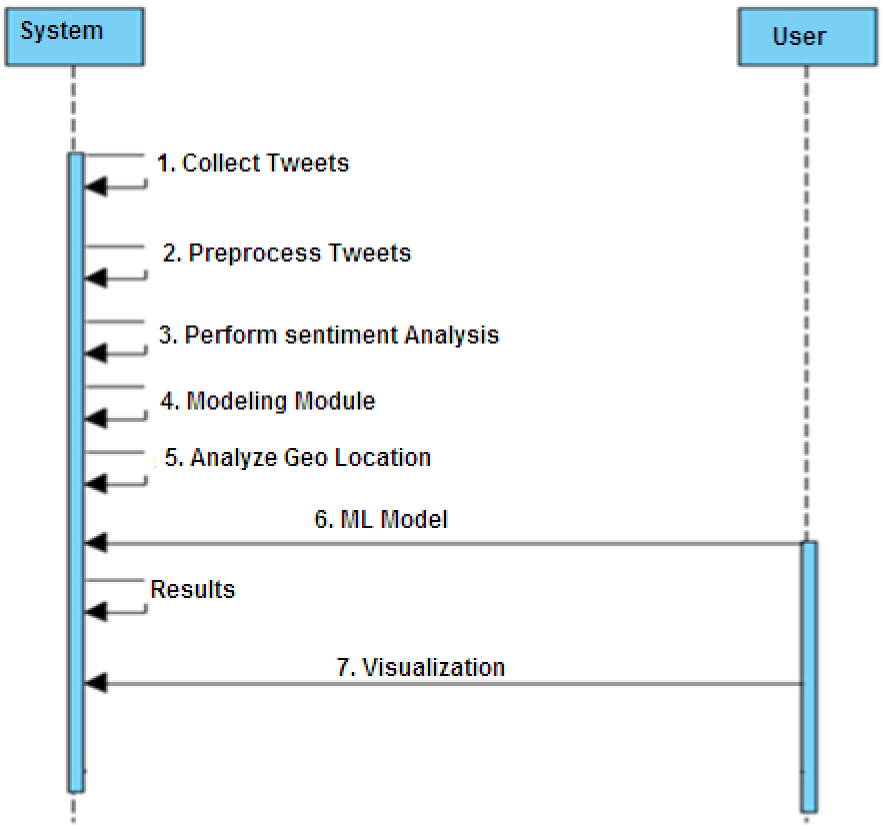


Figure 9: Sequence Diagram

3.1.2.4 Collaboration Diagram:

Activity diagram is basically a flowchart to represent the flow from one activity to another activity. The activity can be described as an operation of the system. The control flow is drawn from one operation to another. This flow can be sequential, branched, or concurrent.

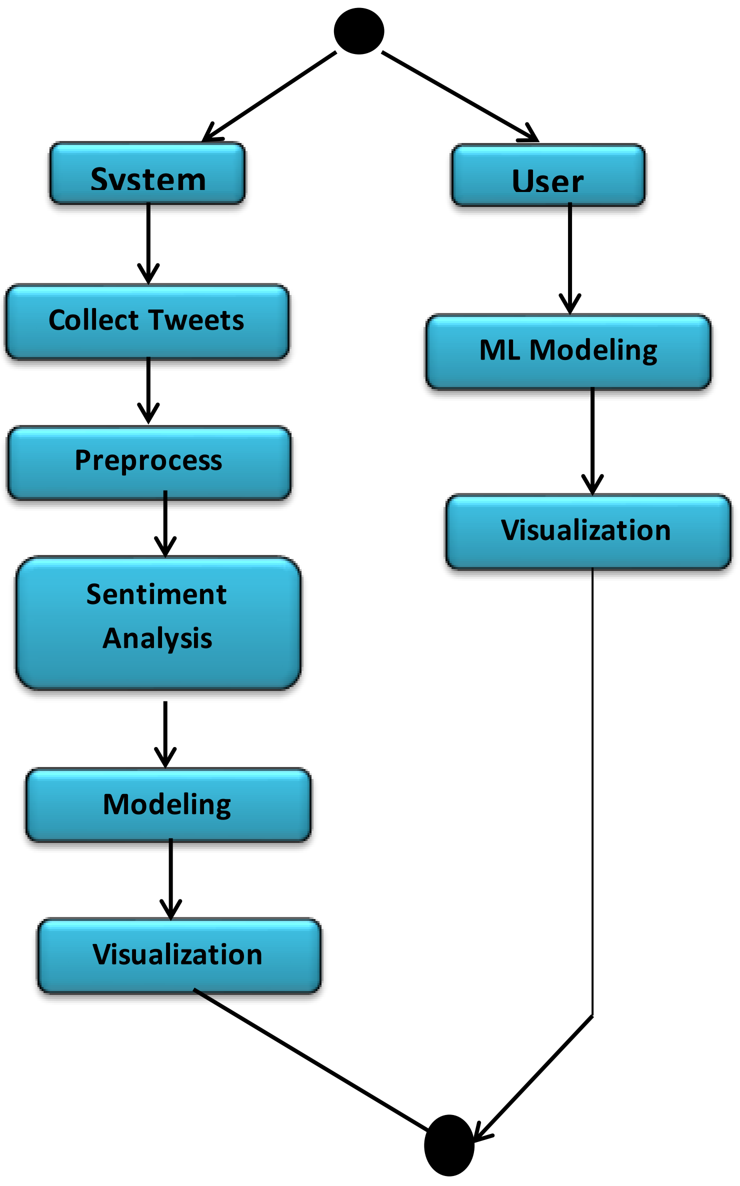


Figure 10: Collaboration Diagram

4. CODING

4.1. Sample Code:

```
import random
import sqlite3
import tkinter as tk
from PIL import ImageTk, Image
from tkinter import messagebox
conn = sqlite3.connect('database.s3db')
cur = conn.cursor()
root = tk.Tk()
root.title("MINI ATM")
root.config(bg='#3F3001')
root.iconbitmap("images/bank_icon.ico")
def create_btn():
    global create_account_frame
    global home_frame
    global t2, t3, t4
    t2.delete(0, tk.END)
    t3.delete(0, tk.END)
    t4.delete(0, tk.END)
    create_account_frame.pack()
    home_frame.pack_forget()
    myAcc.create_account()
def cancel_btn():
    global create_account_frame
    global home_frame
    home_frame.pack()
    create_account_frame.pack_forget()
def save_btn():
    global create_account_frame
    global home_frame, t3, t4, t5
```

```
 if t3.get() == "" or t4.get() == "" or t5.get() == "":
      messagebox.showwarning("unfilled", "Please fill all entries")
      return
   myAcc.save_acc()
   messagebox.showinfo('saved', "your Account has been created")
   create_account_frame.pack_forget()
   home_frame.pack()
def close_btn():
   global create_account_frame, home_frame, login_frame, closed_frame,
transfer_frame, deposit_frame, withdraw_frame
   create_account_frame.pack_forget()
   home_frame.pack_forget()
   login_frame.pack_forget()
   transfer_frame.pack_forget()
   deposit_frame.pack_forget()
   withdraw_frame.pack_forget()
   closed_frame.pack()
def login_btn():
   myAcc.log_in()
def go_to_transfer_frame():
   global login_frame, transfer_frame
   login_frame.pack_forget()
   transfer_frame.pack()
def transfer_cancel_btn():
   global login_frame, transfer_frame, deposit_frame, withdraw_frame
   transfer_frame.pack_forget()
   deposit_frame.pack_forget()
   withdraw_frame.pack_forget()
   login_frame.pack()
def transfer_btn():
   global login_frame, transfer_frame, balance_login_lbl
   myAcc.transfer_to()
   transfer_frame.pack_forget()
 login_frame.pack()
   balance_login_lbl = tk.Label(login_frame, text=f"balance: {myAcc.balance}",
font=('calibri', 27), fg='white',
                                 bg='black').grid(row=2, column=4, columnspan=2)
def withdraw_btn():
   global login_frame, withdraw_frame
   global t1_withdraw, t2_withdraw, t3_withdraw, t4_withdraw, t5_withdraw
   t1_withdraw.delete(0, tk.END)
   t2_withdraw.delete(0, tk.END)
   t3_withdraw.delete(0, tk.END)
   t4_withdraw.delete(0, tk.END)
   t5_withdraw.delete(0, tk.END)
   login_frame.pack_forget()
   withdraw_frame.pack()
def deposit_btn():
   global login_frame, deposit_frame
   global t1_deposit, t2_deposit, t3_deposit, t4_deposit, t5_deposit
   t1_deposit.delete(0, tk.END)
   t2_deposit.delete(0, tk.END)
   t3_deposit.delete(0, tk.END)
   t4_deposit.delete(0, tk.END)
   t5_deposit.delete(0, tk.END)
   login_frame.pack_forget()
   deposit_frame.pack()
def show_btn_deposit():
   global t1_deposit, t2_deposit, t3_deposit, t4_deposit, t5_deposit
   global myAcc
   t1_deposit.delete(0, tk.END)
   t2_deposit.delete(0, tk.END)
   t3_deposit.delete(0, tk.END)
   t4_deposit.delete(0, tk.END)
   t5_deposit.delete(0, tk.END)
   t1_deposit.insert(0, myAcc.accNo)
   t3_deposit.insert(0, myAcc.name)
   t4_deposit.insert(0, myAcc.phone)
   messagebox.showinfo("guideline", "enter pin and Amount")
def show_btn_withdraw():
   global t1_withdraw, t2_withdraw, t3_withdraw, t4_withdraw, t5_withdraw
   global myAcc
   t1_withdraw.delete(0, tk.END)
   t2_withdraw.delete(0, tk.END)
   t3_withdraw.delete(0, tk.END)
   t4_withdraw.delete(0, tk.END)
   t5_withdraw.delete(0, tk.END)
   t1_withdraw.insert(0, myAcc.accNo)
   t3_withdraw.insert(0, myAcc.name)
   t4_withdraw.insert(0, myAcc.phone)
   messagebox.showinfo("guideline", "enter pin and Amount")
def delete_btn_login():
   global login_frame, home_frame, myAcc, e1, e2
   click = messagebox.askyesno("delete", "do you want to delete Your account")
   if click:
       cur.execute(f"DELETE FROM database WHERE Account = {myAcc.accNo}")
       conn.commit()
       e1.delete(0, tk.END)
       e2.delete(0, tk.END)
       login_frame.pack_forget()
       home_frame.pack()
   else:
       return
def update_withdraw():
   global t2_withdraw, t5_withdraw, myAcc, balance_login_lbl, login_frame,
withdraw_frame
   if t2_withdraw.get() == myAcc.pin:
       try:
            amount_withdraw  = int(t5_withdraw.get())
            if amount_withdraw  < myAcc.balance:
                click  = messagebox.askyesno('withdraw',  "do you want to withdraw?")
                if click:
                    myAcc.balance  -= amount_withdraw
                    cur.execute(f"UPDATE    database    SET    balance    =    {myAcc.balance}
WHERE  Account  = {myAcc.accNo}")
                    conn.commit()
                    balance_login_lbl          =          tk.Label(login_frame,          text=f"balance:
{myAcc.balance}",  font=('calibri',  27),
                                          fg='white',
                                          bg='black').grid(row=2,  column=4,  columnspan=2)
                    messagebox.showinfo('info',  'amount  has been withdraw')
                    withdraw_frame.pack_forget()
                    login_frame.pack()
                else:
                    return
            else:
                messagebox.showwarning('warning',  'insufficient  amount')
                return
        except:
            messagebox.showwarning('error',  "please  reenter  amount")
            return
    else:
        messagebox.showwarning('warning',  'invalid  pin')
        return
def update_deposit():
    global      t2_deposit,      t5_deposit,      myAcc,      balance_login_lbl,      login_frame,
deposit_frame
    if t2_deposit.get()  == myAcc.pin:
        try:
            amount_deposit  = int(t5_deposit.get())
            click = messagebox.askyesno('withdraw', "do you want to deposit?")
            if click:
                myAcc.balance += amount_deposit
                cur.execute(f"UPDATE database SET balance = {myAcc.balance}
WHERE Account = {myAcc.accNo}")
                conn.commit()
                balance_login_lbl = tk.Label(login_frame, text=f"balance:
{myAcc.balance}", font=('calibri', 27),
                                             fg='white',
                                             bg='black').grid(row=2, column=4, columnspan=2)
                messagebox.showinfo('info', 'amount has been deposit')
                deposit_frame.pack_forget()
                login_frame.pack()
            else:
                return
        except:
            messagebox.showwarning('error', "please reenter amount")
            return
    else:
        messagebox.showwarning('warning', 'invalid pin')
        return
def checksum(x):
    addition, digit, count = 0, 0, 1
    for i in x:
        if count % 2 != 0:
            if int(i) * 2 > 9:
                addition += int(i) * 2 - 9
            else:
                addition += int(i) * 2
        else:
            addition += int(i)
    for i in range(10):
        if (addition + i) % 10 == 0:
            digit = i
            break
    return digit
class Account:
    def __init__(self):
        self.accNo = ""
        self.pin = ""
        self.name = ""
        self.balance = 0
        self.phone = ""
    def create_account(self):
        accounts = list(cur.execute('SELECT Account FROM database'))
        print(accounts)
        global t1, t2
        while True:
            x = '400000' + str(random.randrange(0, 999999999)).zfill(9)
            last_digit = checksum(x)
            x += str(last_digit)
            pin = str(random.randrange(9999)).zfill(4)
            if (x,) not in accounts:
                self.accNo = x
                self.pin = pin
                break
        t1.insert(0, x)
        t2.insert(0, pin)
    def save_acc(self):
        global t3, t4, t5
        accounts = list(cur.execute('SELECT Account FROM database'))
        name = t3.get()  # this will be changed to display the it on gui
        phone = t5.get()  # this will be changed to display the it on gui
        balance = int(t4.get())
        self.name, self.phone, self.balance = name, phone, balance
        cur.execute(f'INSERT INTO database (id, Account, pin, name, balance, phone)
VALUES(?, ?, ?, ?, ?, ?)"
                , (len(accounts) + 1, self.accNo, self.pin, self.name, self.balance,
self.phone))
        conn.commit()
    def log_in(self):
        global e1, e2, home_frame, login_frame
        all_accounts = list(cur.execute("SELECT Account, pin FROM database"))
        logged_in = False
        acc = e1.get()
        password = e2.get()
        for x, y in all_accounts:
            if acc == x and password == y:
                home_frame.pack_forget()
                login_frame.pack()
                card_detail = list(cur.execute(f'SELECT * FROM database WHERE
Account = {acc}'))
                print(card_detail)
                id, self.accNo, self.pin, self.name, self.balance, self.phone = card_detail[0]
                logged_in = True
                break
        else:
            messagebox.showwarning(title="Error", message="Account No or pin is
invalid")
            return
        if logged_in:
            global name_login_lbl, phone_login_lbl, balance_login_lbl

            name_login_lbl = tk.Label(login_frame, text=f'Name: {self.name}",
font=('calibri', 27), fg='white',
                                      bg='black').grid(row=0, column=4, columnspan=2)
        phone_login_lbl = tk.Label(login_frame, text=f"mobile No: {self.phone}", font=('calibri', 27), fg='white',
                                   bg='black').grid(row=1, column=4, columnspan=2)

        balance_login_lbl = tk.Label(login_frame, text=f"balance: {self.balance}", font=('calibri', 27), fg='white',
                                     bg='black').grid(row=2, column=4, columnspan=2)
    def transfer_to(self):
        x = t_1.get()
        amount = int(t_3.get())
        pin = t_2.get()
        if (x,) in list(cur.execute(f"SELECT Account FROM database")) and pin == self.pin:
            other_account_balance = int(list(cur.execute(f"SELECT balance FROM database WHERE Account = {x}"))[0][0])
            if amount < self.balance:
                confirmation = messagebox.askyesno("transfer", "do you want to transfer")
                if not confirmation:
                    return
                else:
                    other_account_balance += amount
                    self.balance -= amount
                    cur.execute(f"UPDATE database SET balance = {other_account_balance} WHERE Account = {x}")
                    cur.execute(f"UPDATE database SET balance = {self.balance} WHERE Account = {self.accNo}")
                    conn.commit()
                    messagebox.showinfo("transferred", "amount transferred successfully!")
            else:
                messagebox.showwarning("error", "Insufficient balance")
        else:
            messagebox.showwarning("error", "Such account does not exist")
myAcc = Account()
```

```
print(list(cur.execute("SELECT * FROM database")))
# home frame
home_frame = tk.Frame(root)
bank_img = ImageTk.PhotoImage(Image.open("images/front.jpg"))
img_label = tk.Label(home_frame, image=bank_img).grid(row=0, column=0, columnspan=3)
account_label = tk.Label(home_frame, text="Acc No: ", font=('Arial', 12)).grid(row=1, column=0)
pin_label = tk.Label(home_frame, text="PIN: ", font=('Arial', 12)).grid(row=2, column=0)
create_label = tk.Label(home_frame, text="Don't have a Account:", font=('Arial', 12)).grid(row=4, column=0, columnspan=2)
e1 = tk.Entry(home_frame)
e2 = tk.Entry(home_frame)
btn1_ = tk.Button(home_frame, text="Login", command=login_btn)
btn2_ = tk.Button(home_frame, text="create", command=create_btn)
e1.grid(row=1, column=1, pady=6, columnspan=2, sticky=tk.W+tk.E, padx=4)
e2.grid(row=2, column=1, pady=6, columnspan=2, sticky=tk.W+tk.E, padx=4)
btn1_.grid(row=3, column=2, pady=5, padx=4, sticky=tk.W+tk.E)
btn2_.grid(row=4, column=2, sticky=tk.W+tk.E, padx=4, pady=5)
home_frame.pack()
```

5. TESTING

The purpose of testing is to discover errors. Testing is the process of trying to discover every conceivable fault or weakness in a work product. It provides a way to check the functionality of components, sub-assemblies, assemblies and/or a finished product It is the process of exercising software with the intent of ensuring that the

Software system meets its requirements and user expectations and does not fail in an unacceptable manner. There are various types of test. Each test type addresses a specific testing requirement

Software testing is an investigation conducted to provide stakeholders with information about the quality of the product or service under test. Software testing also provides an objective, independent view of the software to allow the business to appreciate and understand the risks at implementation of the software. Test techniques include, but are not limited to, the process of executing a program or application with the intent of finding software bugs.

Software testing can also be stated as the process of validating and verifying that a software program/application/product:

- Meets the business and technical requirements that guided its design and development.
- Works as expected and
- Can be implemented with the same characteristics.

Software testing, depending on the testing method employed, can be implemented at any time in the development process. However, most of the test effort occurs after the requirements have been defined and the coding process has been completed. As such, the methodology of the test is governed by the software development methodology adopted.

Different software development models will focus the test effort at different points in the development process. Newer development models, such as Agile, often employ test driven development and place an increased portion of the testing in the hands of the developer, before it reaches a formal team of testers. In a more traditional model, most of the test execution occurs after the requirements have been defined and the coding process has been completed.

Testing can never completely identify all the defects within software. Instead, it furnishes a criticism or comparison that compares the state and behaviour of the product against oracles—principles or mechanisms by which someone might recognize a problem. These oracles may include (but are not limited to) specifications, contracts, comparable products, past versions of the same product, inferences about intended or expected purpose, user or customer expectations, relevant standards, applicable laws, or other criteria.

Every software product has a target audience. For example, the audience for video game software is completely different from banking software. Therefore, when an organization develops or otherwise invests in a software product, it can assess whether the software product will be acceptable to its end users, its target audience, its purchasers, and other stakeholders. Software testing is the process of attempting to make this assessment.

5.1. INTRODUCTION TO TESTING:

UNIT TESTING

Unit testing involves the design of test cases that validate that the internal program logic is functioning properly, and that program inputs produce valid outputs. All decision branches and internal code flow should be validated. It is the testing of individual software units of the application .it is done after the completion of an individual unit before integration. This is a structural testing, that relies on knowledge of its construction and is invasive. Unit tests perform basic tests at component level and test a specific business process, application, and/or system configuration. Unit tests ensure that each unique path of a business process performs accurately to the documented specifications and contains clearly defined inputs and expected results.

INTEGRATION TESTING

Integration tests are designed to test integrated software components to determine if they actually run as one program. Testing is event driven and is more concerned with the basic outcome of screens or fields. Integration tests demonstrate that although the components were individually satisfaction, as shown by successfully unit testing, the combination of components is correct and consistent. Integration testing is specifically aimed at exposing the problems that arise from the combination of components.

VALIDATION TESTING

An engineering validation test (EVT) is performed on first engineering prototypes, to ensure that the basic unit performs to design goals and specifications. It is important in identifying design problems, and solving them as early in the design cycle as possible, is the key to keeping projects on time and within budget. Too often, product design and performance problems are not detected until late in the product development cycle — when the product is ready to be shipped. The old adage holds true: It costs a penny to make a change in engineering, a dime in production and a dollar after a product is in the field.

Verification is a Quality control process that is used to evaluate whether or not a product, service, or system complies with regulations, specifications, or conditions imposed at the start of a development phase. Verification can be in development, scale- up, or production. This is often an internal process.

Validation is a Quality assurance process of establishing evidence that provides a high degree of assurance that a product, service, or system accomplishes its intended requirements. This often involves acceptance of fitness for purpose with end users and other product stakeholders.

The testing process overview is as follows:

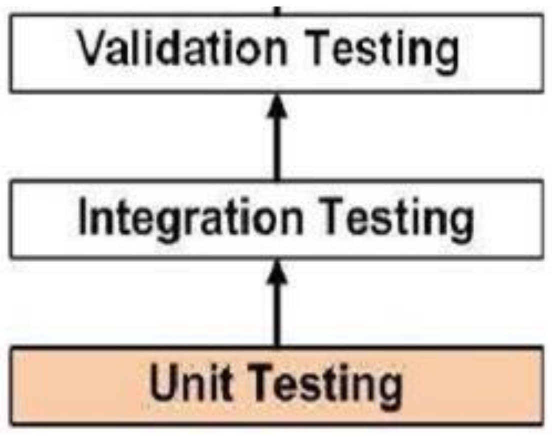


Figure 16: Testing Process

SYSTEM TESTING

System testing of software or hardware is testing conducted on a complete, integrated system to evaluate the system's compliance with its specified requirements. System testing falls within the scope of black box testing, and as such, should require no knowledge of the inner design of the code or logic.

As a rule, system testing takes, as its input, all of the "integrated" software components that have successfully passed integration testing and also the software system itself integrated with any applicable hardware system(s). System testing is a more limited type of testing; it seeks to detect defects both within the "inter-assemblages" and also within the system as a whole. System testing is performed on the entire system in the context of a Functional Requirement Specification(s) (FRS) and/or a System Requirement Specification (SRS).

System testing tests not only the design, but also the behavior and even the believed expectations of the customer. It is also intended to test up to and beyond the bounds defined in the software/hardware requirements specification(s).

System testing for an ATM management system involves testing the system as a whole to ensure that it meets the specified requirements and functions correctly. It focuses on evaluating the system's behavior and performance in real-world scenarios. Here are some key aspects of system testing for an ATM management system:

Functional Testing: This type of testing verifies that all the functional requirements of the ATM management system are working correctly. It includes testing various operations such as cash withdrawals, balance inquiries, fund transfers, bill payments, and card management functionalities.

Usability Testing: Usability testing ensures that the ATM management system is user-friendly and provides a seamless user experience. It involves testing the system's user interface, navigation, ease of use, and error handling. The objective is to validate that users can easily interact with the system and perform transactions without confusion or difficulty.

Security Testing: Security testing is crucial for an ATM management system to protect sensitive customer information and prevent unauthorized access. It involves testing the system's authentication mechanisms, data encryption, PIN verification, session management, and protection against common security threats such as phishing, skimming, and hacking.

Performance Testing: Performance testing evaluates the system's performance under normal and peak load conditions. It includes testing response times for various operations, handling multiple concurrent users, stress testing to determine the system's maximum capacity, and assessing any performance bottlenecks or resource limitations.

Compatibility Testing: Compatibility testing ensures that the ATM management system works correctly on different hardware and software configurations. It involves testing the system on various platforms, operating systems, web browsers, and ATM hardware devices to validate compatibility and ensure consistent behavior.

Reliability and Availability Testing: This type of testing checks the system's reliability and availability to ensure that it functions consistently and remains accessible to users. It involves testing system uptime, fault tolerance, error recovery, and backup and restores mechanisms.

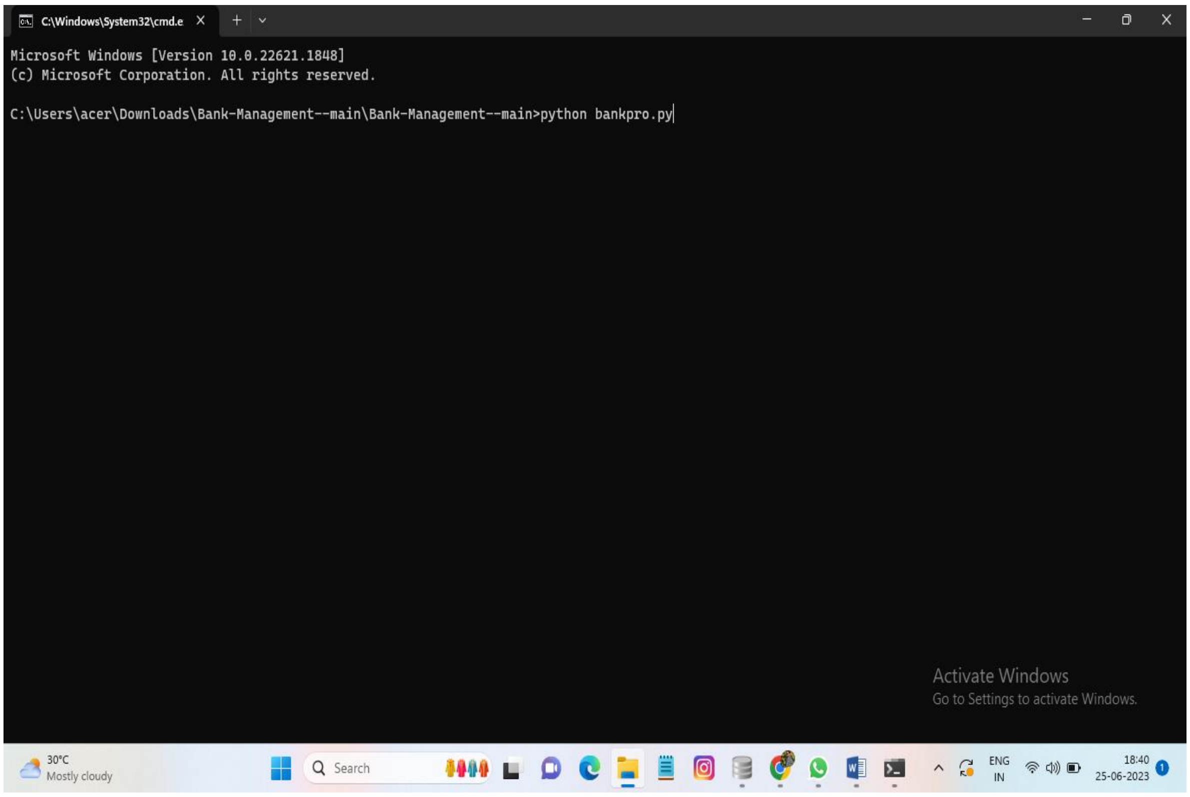
Integration Testing: Integration testing verifies the proper integration and communication between different components of the ATM management system, such as the ATM hardware, network connectivity, backend databases, and third-party services. It ensures that data flows correctly between these components and that they work together seamlessly.

Regression Testing: Regression testing is performed after making changes or updates to the ATM management system to ensure that existing functionality has not been affected. It involves retesting previously tested features and functionalities to validate their continued correctness and stability.

Compliance Testing: Compliance testing ensures that the ATM management system complies with industry standards, regulations, and legal requirements. It involves testing adherence to banking regulations, data privacy laws, accessibility guidelines, and any other relevant compliance standards.

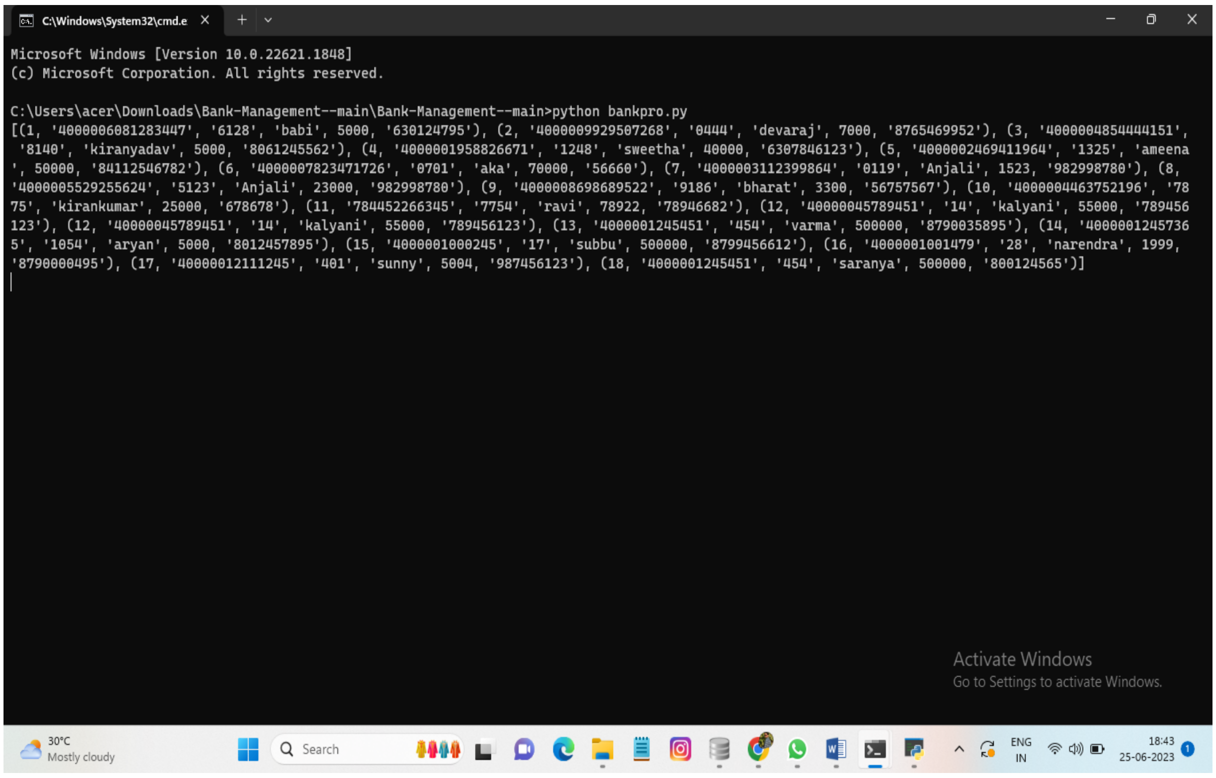
During system testing, it is important to document and track any issues or defects discovered, prioritize and resolve them, and retest to ensure they have been fixed successfully. A combination of manual testing and automated testing tools can be used to effectively carry out system testing for an ATM management system.

6. SCREENSHOTS



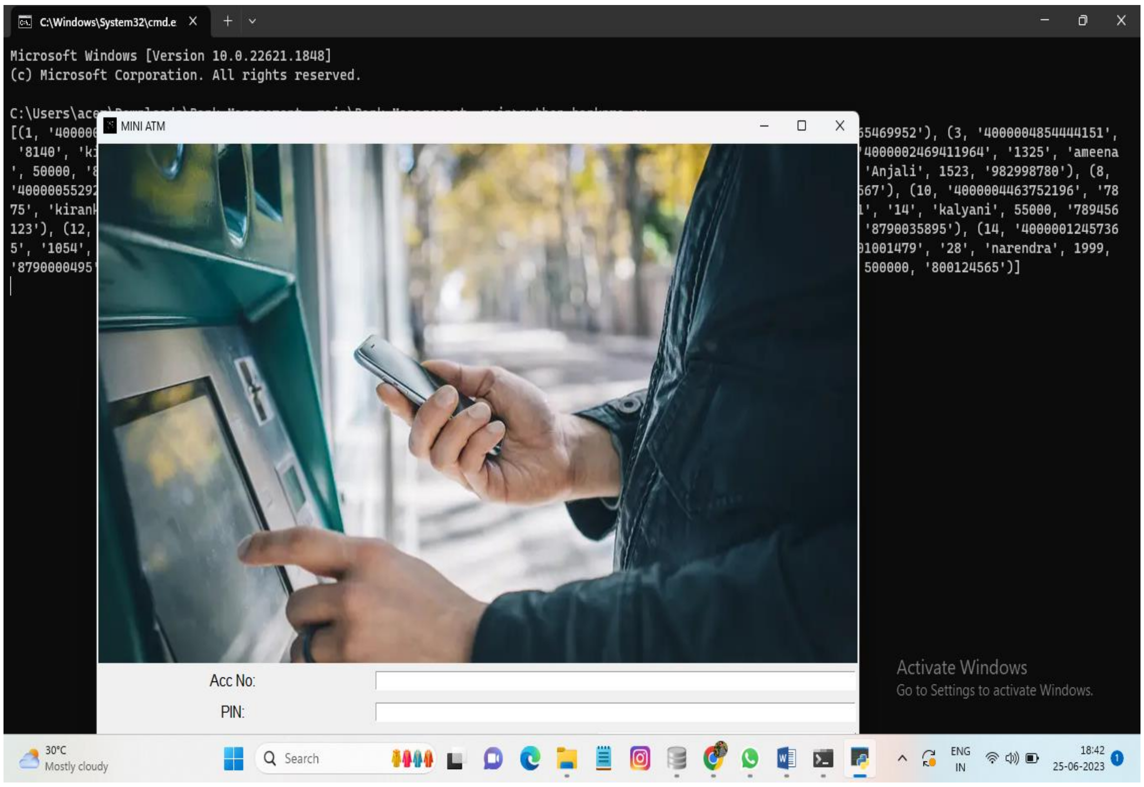
```
Microsoft Windows [Version 10.0.22621.1848]
(c) Microsoft Corporation. All rights reserved.

C:\Users\acer\Downloads\Bank-Management--main\Bank-Management--main>python bankpro.py
```

C:\Windows\System32\cmd.e
Activate Windows
Go to Settings to activate Windows.
30°C
Mostly cloudy
Search
ENG
IN
18:40
25-06-2023

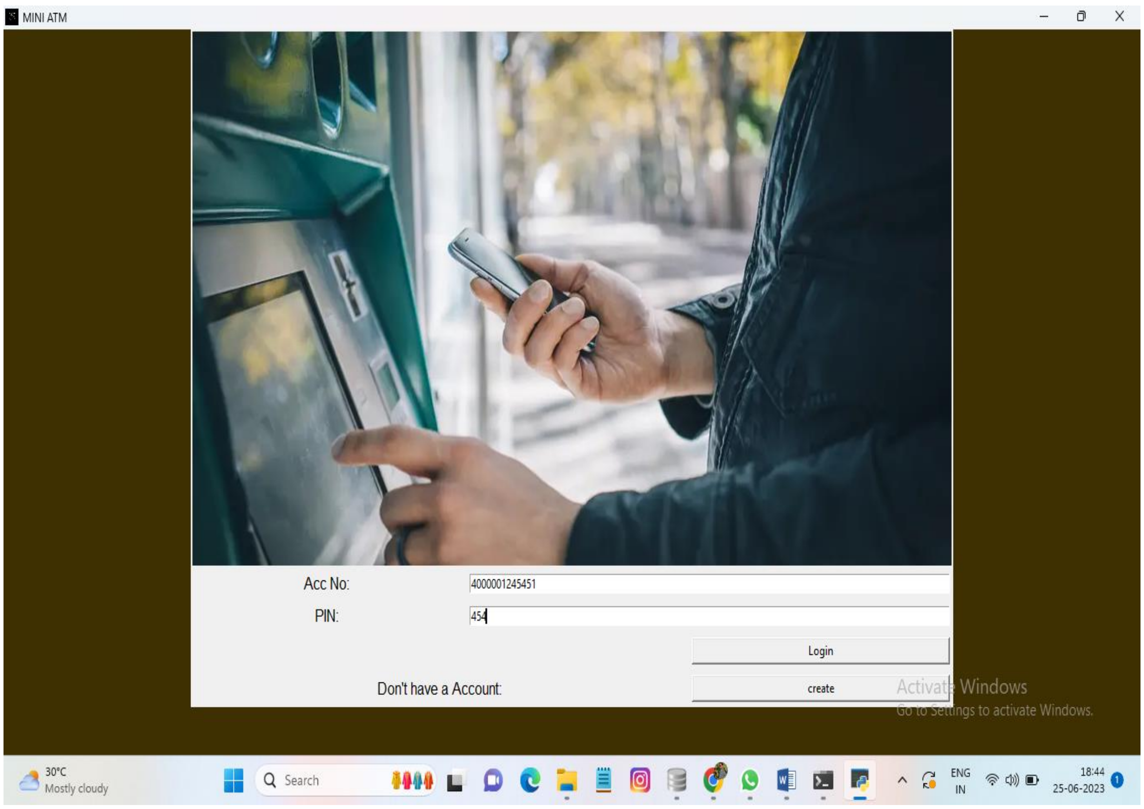


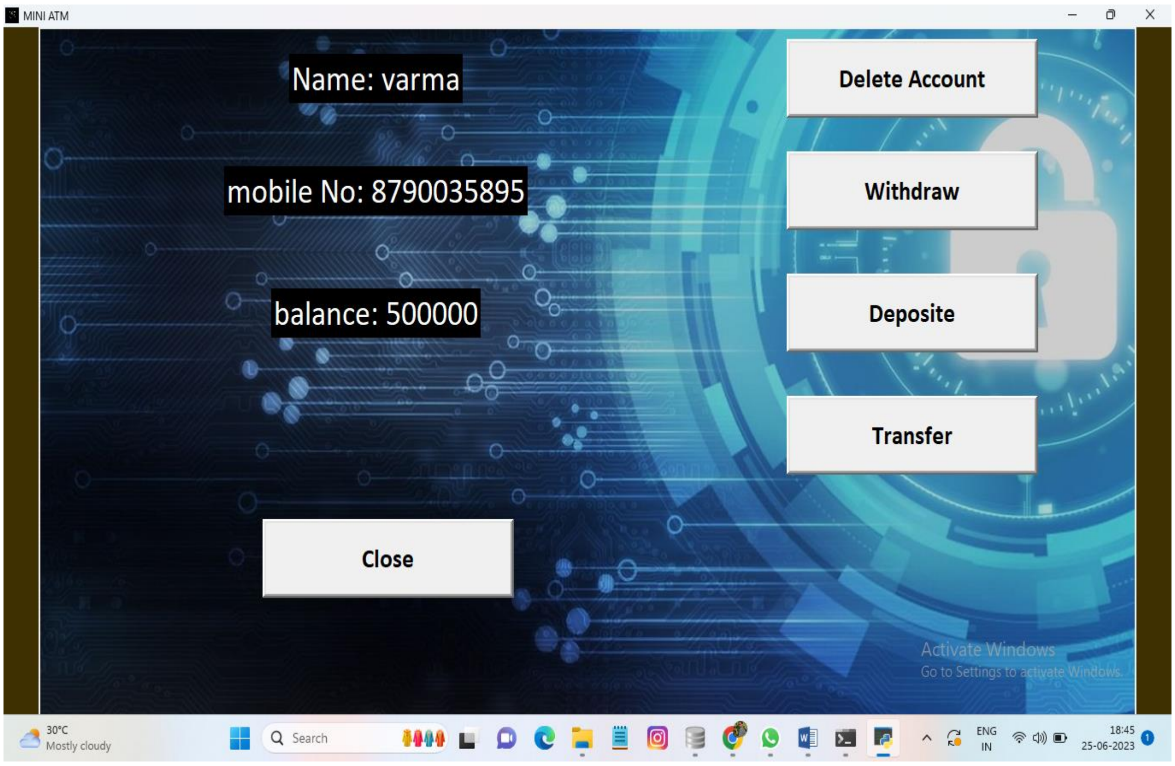
```
Microsoft Windows [Version 10.0.22621.1848]
(c) Microsoft Corporation. All rights reserved.

C:\Users\acer\Downloads\Bank-Management--main\Bank-Management--main>python bankpro.py
[(1, '4000006081283447', '6128', 'babi', 5000, '630124795'), (2, '4000009929507268', '0444', 'devaraj', 7000, '8765469952'), (3, '4000004854444151',
 '8140', 'kiranyadav', 5000, '8061245562'), (4, '4000001958826671', '1248', 'sweetha', 40000, '6307846123'), (5, '4000002469411964', '1325', 'ameena
', 50000, '84112546782'), (6, '4000007823471726', '0701', 'aka', 70000, '56660'), (7, '4000003112399864', '0119', 'Anjali', 1523, '982998780'), (8,
'4000005529255624', '5123', 'Anjali', 23000, '982998780'), (9, '4000008698689522', '9186', 'bharat', 3300, '56757567'), (10, '4000004463752196', '78
75', 'kirankumar', 25000, '678678'), (11, '784452266345', '7754', 'ravi', 78922, '78946682'), (12, '40000045789451', '14', 'kalyani', 55000, '789456
123'), (12, '40000045789451', '14', 'kalyani', 55000, '789456123'), (13, '4000001245451', '454', 'varma', 500000, '8790035895'), (14, '4000001245736
5', '1054', 'aryan', 5000, '8012457895'), (15, '4000001000245', '17', 'subbu', 500000, '8799456612'), (16, '4000001001479', '28', 'narendra', 1999,
'8790000495'), (17, '40000012111245', '401', 'sunny', 5004, '987456123'), (18, '4000001245451', '454', 'saranya', 500000, '800124565')]
```

C:\Windows\System32\cmd.e
Activate Windows
Go to Settings to activate Windows.
30°C
Mostly cloudy
Search
ENG
IN
18:43
25-06-2023



```
Microsoft Windows [Version 10.0.22621.1848]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ace
[(1, '400000                                                                  55469952'), (3, '4000004854444151',
 '8140', 'ki                                                                  '4000002469411964', '1325', 'ameena
', 50000, '8                                                                  'Anjali', 1523, '982998780'), (8,
'40000055292                                                                 567'), (10, '4000004463752196', '78
75', 'kirank                                                                  1', '14', 'kalyani', 55000, '789456
123'), (12,                                                                   '8790035895'), (14, '4000001245736
5', '1054',                                                                  01001479', '28', 'narendra', 1999,
'8790000495                                                                   500000, '800124565')]
```

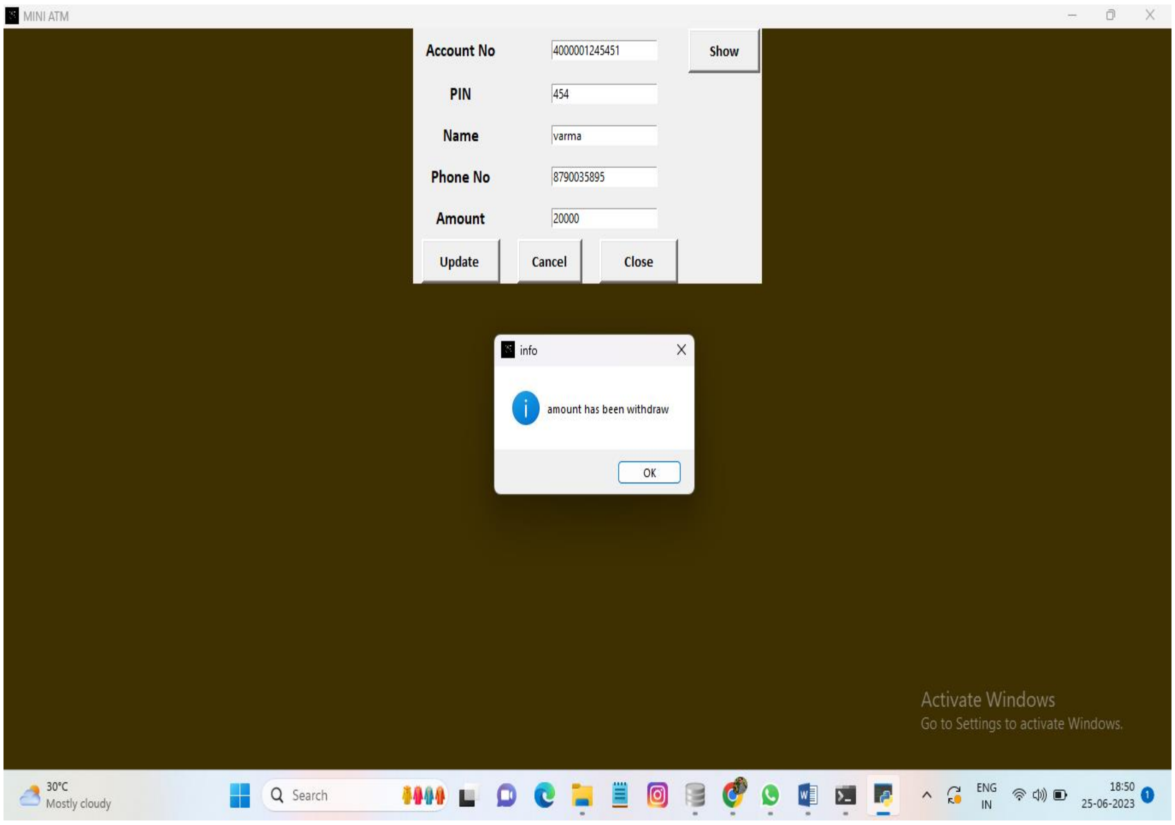




Name: varma

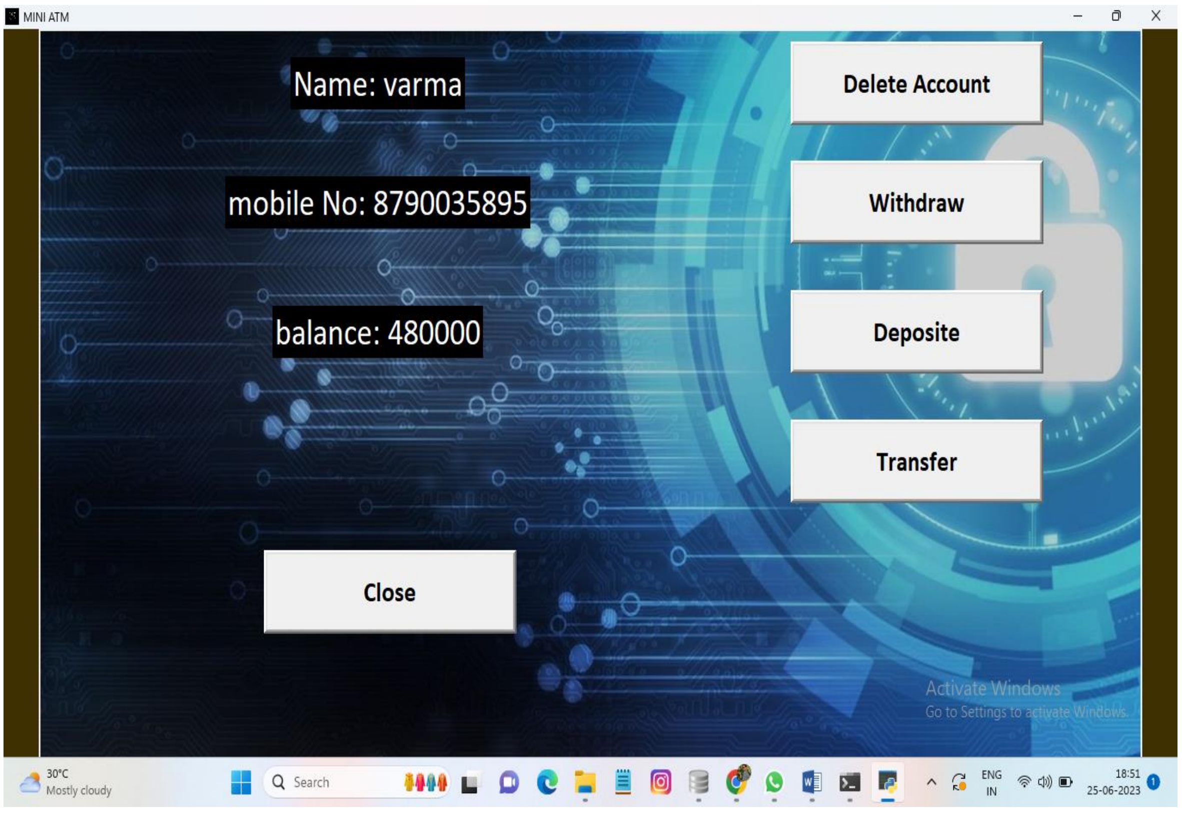
mobile No: 8790035895

balance: 500000



Account No	4000001245451
PIN	454
Name	varma
Phone No	8790035895
Amount	20000

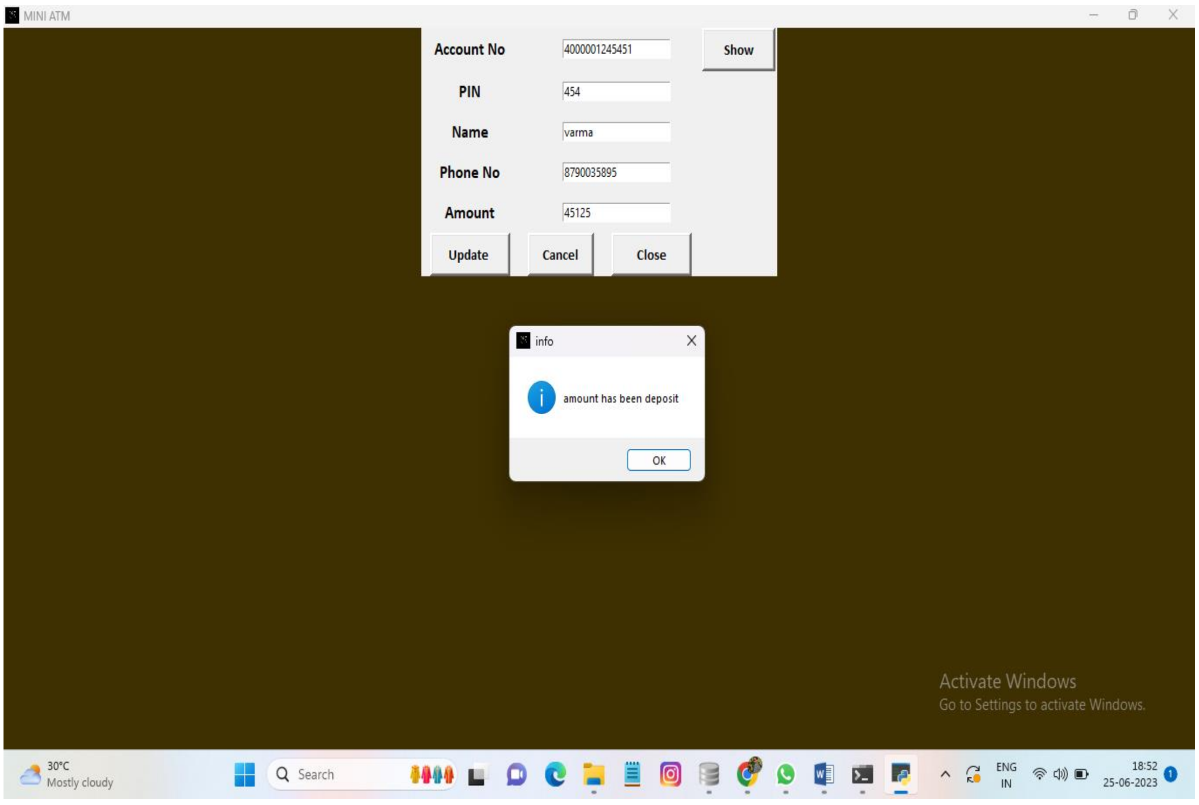
amount has been withdraw


MINI ATM

Name: varma

mobile No: 8790035895

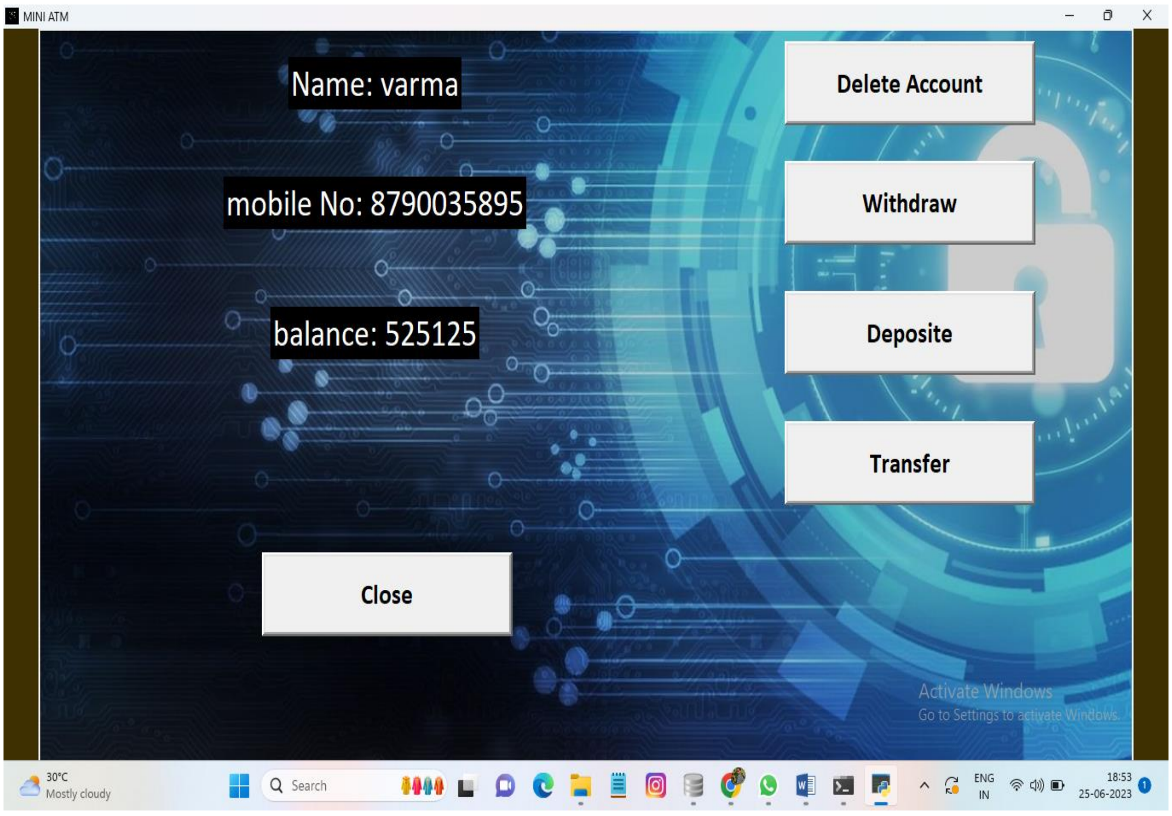
balance: 480000


MINI ATM

Account No	4000001245451
PIN	454
Name	varma
Phone No	8790035895
Amount	45125

info

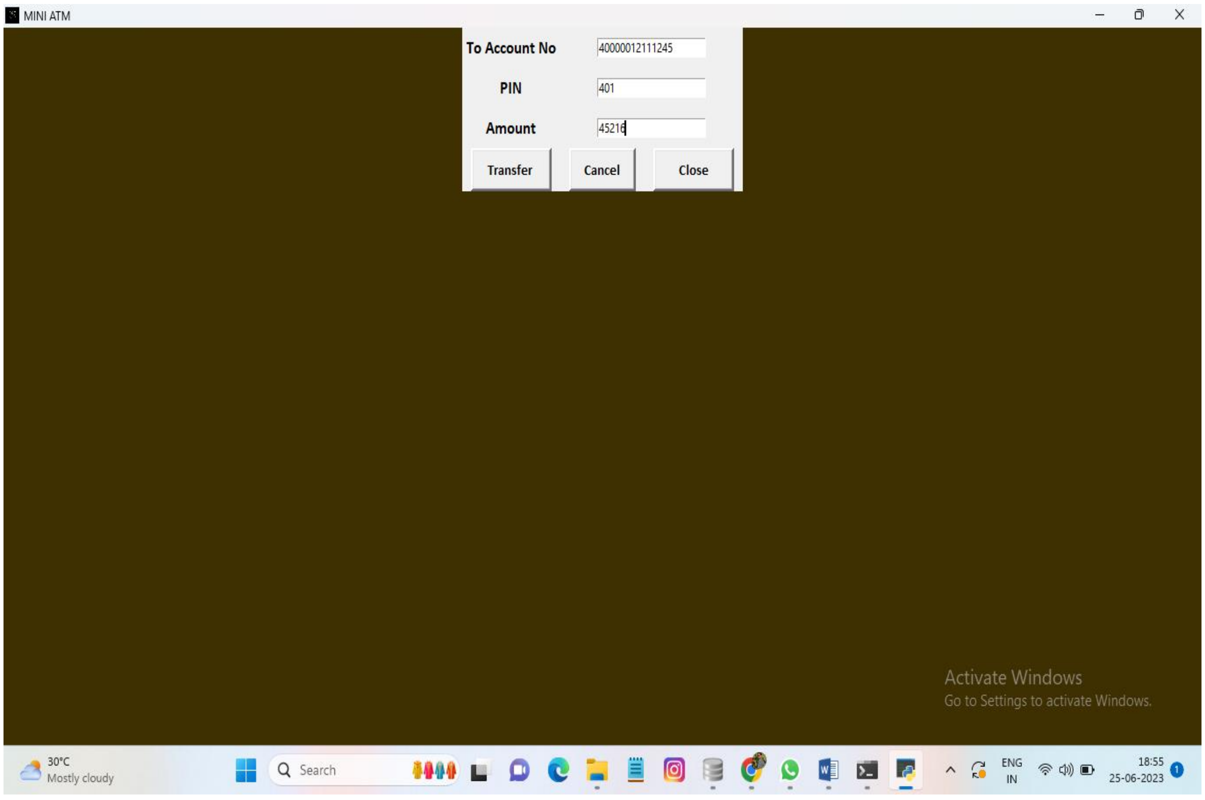
amount has been deposit


MINI ATM

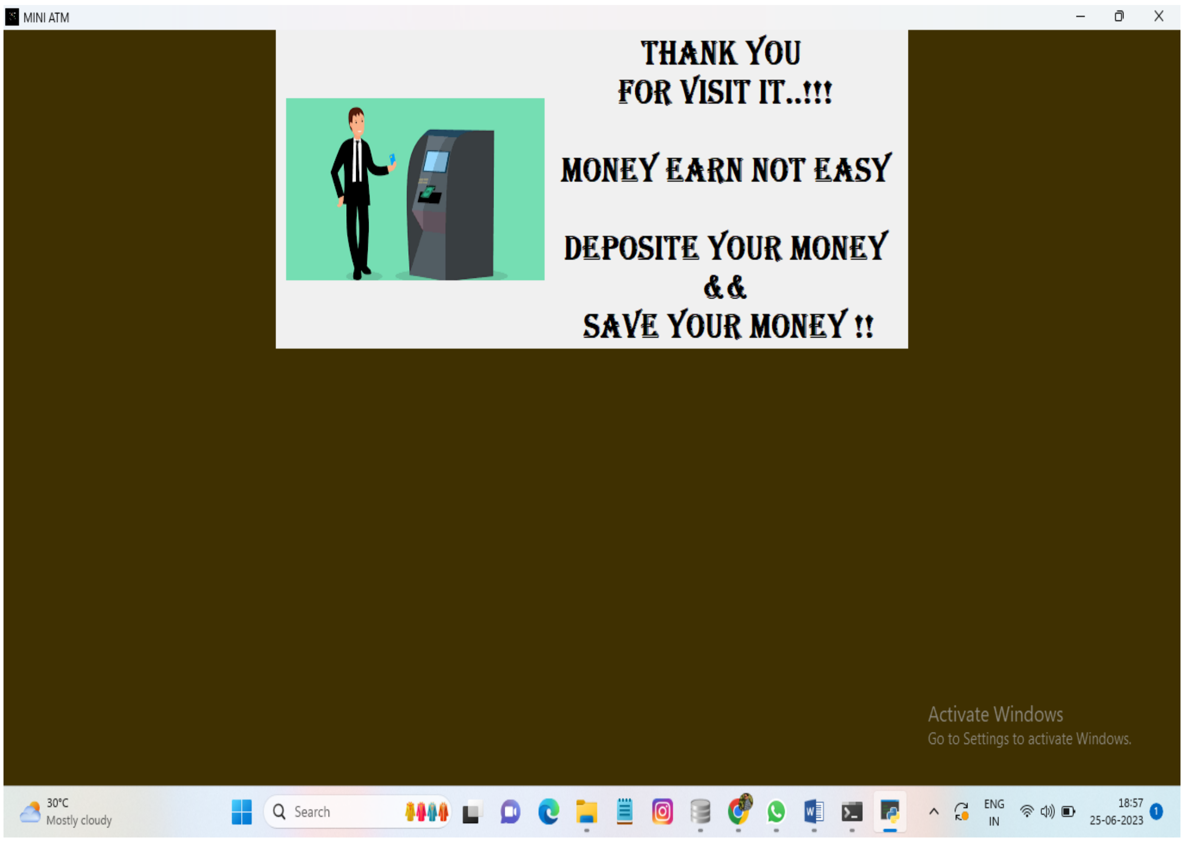
Name: varma

mobile No: 8790035895

balance: 525125


MINI ATM

To Account No	40000012111245
PIN	401
Amount	45216



MINI ATM

THANK YOU
FOR VISIT IT..!!!

MONEY EARN NOT EASY

DEPOSITE YOUR MONEY
&&
SAVE YOUR MONEY !!

7. CONCLUSION

In conclusion, the ATM management system plays a crucial role in effectively managing and operating automated teller machines (ATMs). The system provides a comprehensive solution for managing ATM transactions, user accounts, security, and maintenance.

By analyzing the existing system and identifying its limitations, the proposed system aims to address these shortcomings and improve the overall functionality and efficiency of ATM management. The proposed system introduces several key features and enhancements, including:

User-friendly interface: The system incorporates a user-friendly interface that simplifies the ATM operation and improves the user experience. It provides clear instructions and intuitive navigation for users to perform transactions easily.

Enhanced security measures: The proposed system includes advanced security measures to safeguard user data and prevent fraudulent activities. It incorporates features such as biometric authentication, card encryption, PIN validation, and real-time transaction monitoring.

Robust transaction management: The system offers robust transaction management capabilities, ensuring accurate and secure processing of various transaction types, such as cash withdrawals, balance inquiries, fund transfers, and bill payments. It maintains transaction logs and generates comprehensive reports for auditing and analysis purposes.

Efficient cash management: The proposed system optimizes cash management by providing real-time monitoring of ATM cash levels, automated cash replenishment alerts, and forecasting based on historical transaction data. This ensures that ATMs are adequately stocked with cash to meet customer demands.

Maintenance and troubleshooting: The system includes maintenance and troubleshooting modules that facilitate proactive monitoring of ATM hardware and software components. It provides automated alerts for technical issues, schedules routine maintenance tasks, and tracks service history for efficient maintenance operations.

Integration with banking systems: The proposed system integrates seamlessly with existing banking systems, enabling real-time synchronization of transaction data, customer information, and account balances. This ensures accurate and up-to-date information for both the ATM management system and the banking system.

Overall, the proposed ATM management system aims to improve operational efficiency, enhance security, and provide a seamless user experience for ATM users. It streamlines the management of ATM networks, reduces downtime, and enables banks to deliver reliable and convenient banking services to their customers.

8. BIBLIOGRAPHY

When citing sources for an ATM management system, you may refer to various books, research papers, articles, and online resources that provide information about ATM systems, software engineering, database management, security, and other relevant topics. Here is an example bibliography for an ATM management system:

Book:

- Title: "ATM Networks: Concepts, Protocols, Applications" Author: Sumit Kasera Publisher: McGraw-Hill Education
- Title: "Software Engineering: Principles and Practice" Author: Hans van Vliet Publisher: John Wiley & Sons

Research Paper:

- Title: "Design and Implementation of an ATM System" Authors: John Doe, Jane Smith Published in: Proceedings of the International Conference on Software Engineering
- Title: "Database Design and Management for ATM Systems" Authors: Mark Johnson, Emily Brown Published in: Journal of Information Systems
- Online Resource:
- Title: "ATM Management System: Best Practices and Considerations" Website: www.example.com
- Title: "Security Issues in ATM Systems: Challenges and Solutions" Authors: Robert Davis, Sarah Wilson Published in: International Journal of Network Security
- Title: "Enhancing ATM System Performance through Optimization Techniques" Authors: James Thompson, Jennifer Lee Published in: Proceedings of the International Conference on Computer Science
- Title: "ATM Management System User Manual" Website: www.example.com/documentation